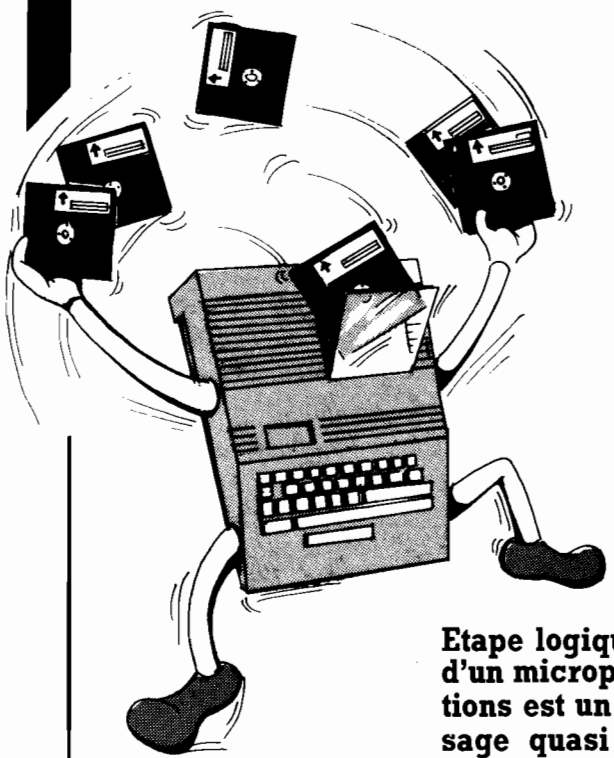
L'ABC DE LA MICRO-INFORMATIQUE



LE JEU D'INSTRUCTIONS DU 8088

Etape logique suivante dans la présentation d'un microprocesseur, celle du jeu d'instructions est un peu fastidieuse mais est un passage quasi obligé si l'on veut réellement s'initier à la micro-informatique autrement que pour pouvoir tenir des conversations « de salon ».

Nous allons donc aujourd'hui entreprendre cette présentation que nous allons tout de même essayer de rendre aussi agréable à lire que le permet le sujet ! Une telle présentation peut être faite par ordre alphabétique, et c'est sous cette forme qu'on la trouve d'ailleurs dans les manuels de programmation des fabricants de microprocesseurs, mais c'est alors assez peu pédagogique. Nous avons décidé de vous présenter les diverses instructions groupées par fonctions comme cela se fait généralement dans les (bons) stages d'initiation à la programmation en langage machine.

Pour ce faire, nous avons retenu les groupes que voici :
– instructions de mouvement ou transfert de données ;
– instructions arithmétiquees et logiques ;
– instructions de branchement ;
– instructions de contrôle du processeur ;
– instructions de chaînes ;
– interruptions.
Les quatre premières catégories d'instructions se retrouvent sur tous les microprocesseurs du marché, car elles constituent la base des possibilités d'un tel circuit ; les deux dernières catégories, par contre, sont plus spécialement destinées au 8088.

Instructions de mouvement des données

La première instruction, et la plus utilisée, est l'instruction MOV que nous avions d'ailleurs choisie comme exemple pour la présentation des modes d'adressage réalisée le mois dernier.
Elle s'utilise sous la forme MOV destination, source, et déplace bien évidemment le contenu de « source » vers « destination ». Attention à l'ordre d'écriture de ces deux éléments qui est un peu contre nature.
La majorité des modes d'adressage peut être utilisée tant pour source que pour destination, aux réserves suivantes près :
– Il est impossible de faire un transfert de mémoire à mémoire. Il faut obligatoirement passer par un registre du 8088 pour cela. Cette contrainte est d'ailleurs commune à tous les microprocesseurs actuels.
– Il est interdit de charger une valeur immédiate dans un registre de segment. Il faut obligatoirement passer par un registre général pour cela.
– Il est interdit d'utiliser le registre IP comme source ou destination.
Dans la pratique, ces restrictions sont assez peu contraignantes comme nous le verrons lors des exemples de programmation.
Dans le même ordre d'idées, viennent ensuite les instructions PUSH et POP qui permettent, respectivement, de mettre des données sur la pile et de récupérer des données à partir de la pile. La pile n'est rien d'autre qu'une zone mémoire particulière, référencée par le pointeur de pile (ou *stack pointer* en américain). Ce dernier est constitué par l'association du registre SP et

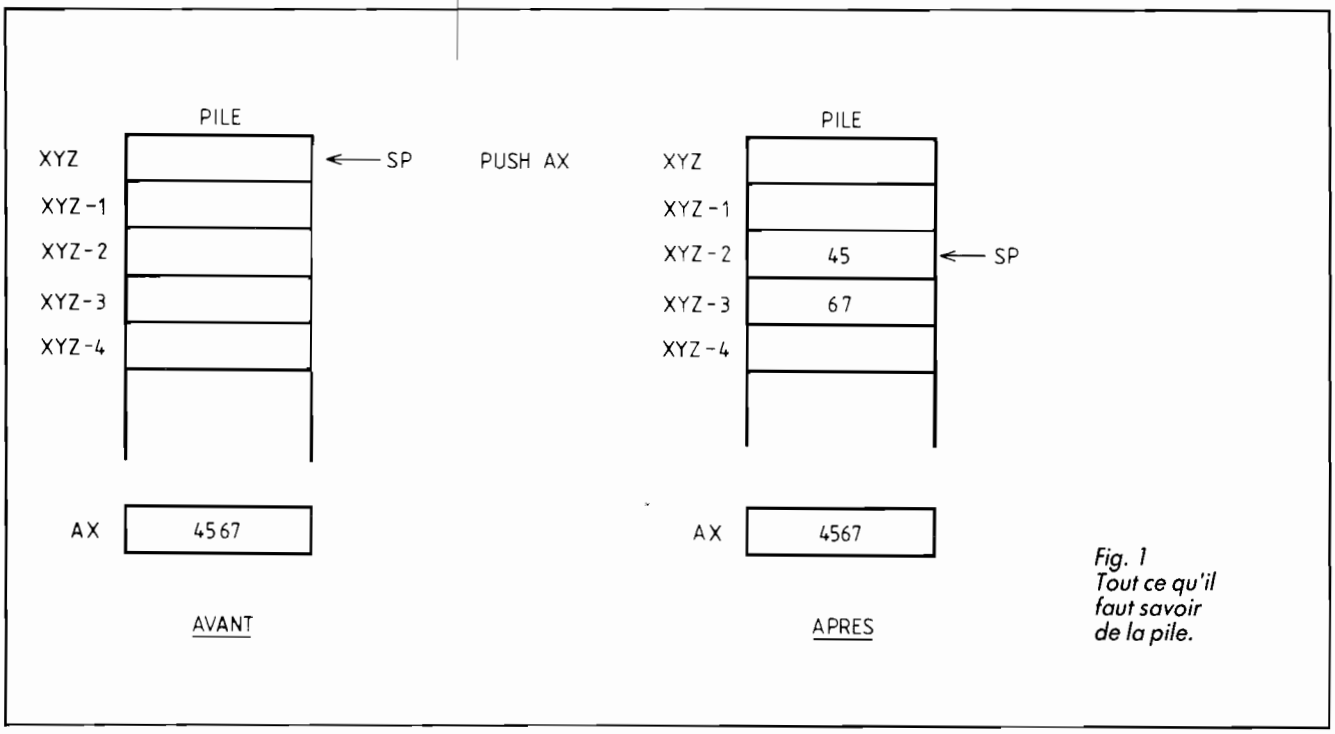


Fig. 1
Tout ce qu'il faut savoir de la pile.

du registre de segment SS comme leurs noms respectifs le laissent prévoir.
Cette pile est une pile LIFO, c'est-à-dire Last In First Out ou, en bon français, dernier entré premier sorti. En d'autres termes, la dernière donnée stockée sur la pile sera la première à pouvoir être récupérée et *vice versa*. De plus, le pointeur de pile « descend » au fur et à mesure que l'on stocke des données sur cette dernière comme schématisé figure 1. Enfin, la pile ne travaille qu'avec des mots de 16 bits. Il est donc impossible d'y stocker ou de récupérer seulement un octet. Cette notion de pile se retrouve, elle aussi, sur tous les microprocesseurs du marché. La pile fonctionne toujours comme expliqué ci-avant, seule la taille des mots stockés sur la pile diffère d'un circuit à un autre.
Les instructions PUSH et POP s'utilisent de la façon suivante :

PUSH source,
POP destination.

Dans le premier cas, le pointeur de pile est diminué de 2 unités (puisque nous avons dit qu'il descendait et que la pile ne contenait que des mots de 16 bits), puis le contenu de source est « poussé » sur la pile (ce qui justifie le nom du mnémonique). Dans le second cas, la valeur pointée par le pointeur de pile est placée dans destination puis le pointeur de pile est incrémenté de 2 unités (il remonte donc puisque la pile se libère de deux octets).

Il est possible de placer ainsi sur la pile une suite de valeurs grâce à plusieurs PUSH successifs ; il faut seulement faire attention ensuite à l'ordre des POP pour bien récupérer ce que l'on veut puisque la pile est une LIFO. Ainsi, la sauvegarde des registres généraux du 8088 pourrait être faite par :

PUSH AX
PUSH BX
PUSH CX
PUSH DX

mais il faudrait ensuite récupérer ces registres par :

POP DX
POP CX
POP BX
POP AX.

Une inversion dans l'ordre relatif des PUSH et des POP conduit, dans cet exemple, à échanger les contenus de certains registres. Si c'est involontaire, c'est une catastrophe ; par contre, cela peut être voulu pour une application particulière.

Dernière précision à propos de ces instructions ou, plus exactement, de la pile. Le fonctionnement du pointeur de pile est automatique puisqu'il est incrémenté ou décrémenté de 2 unités lors de chaque utilisation de PUSH ou POP ; il faut cependant penser à l'initialiser, avant la première utilisation, de façon à ce qu'il pointe sur une zone de mémoire vive du système. De plus, comme le pointeur descend au fur et à mesure que l'on place des informations sur la pile, on choisit en général comme valeur initiale l'adresse la plus haute d'une zone de mémoire vive.

Vient ensuite une instruction un peu plus performante qui permet d'échanger les contenus de deux registres ou les contenus d'une mémoire et d'un registre. Elle s'utilise de la façon suivante :

XCHG X, Y où X et Y peuvent être un nom de registre interne du 8088 ou une adresse mémoire. Les contenus de X et Y sont échangés par cette instruction, c'est-à-dire que X se retrouve dans Y et Y dans X. Seules restrictions d'emploi, logiques si l'on s'en réfère à l'instruction MOV vue ci-avant, il est impossible de faire de l'échange direct entre deux adresses mémoires et les registres ne doivent pas être des registres de segments.

La dernière instruction de mouvement de données au sens strict du terme est XLAT. Elle est assez peu intéressante car trop restrictive et particu-

lière. Avant de l'utiliser, le registre AL (c'est-à-dire les poids faibles de AX, rappelons-le) doit être chargé avec une autre valeur qui, combinée par défaut avec le contenu du segment DS, va servir à générer une adresse sur 20 bits. L'utilisation de l'instruction XLAT aura pour effet de charger dans AL la valeur contenue à l'adresse obtenue en ajoutant l'adresse sur 20 bits élaborée avec BX et DS avec le contenu de AL. Présenté autrement, on peut considérer que BX contient l'adresse de début d'un tableau et que le contenu initial de AL est la position d'une donnée dans ce tableau ; donnée que l'on récupère donc grâce à XLAT.

Voici un exemple concret fort simple :

MOV AL,06 charge 06 dans AL,
MOV BX,1000 charge 1000 dans BX,
XLAT charge AL avec le sixième élément du tableau commençant en 1000.

Les instructions que nous allons voir maintenant, bien qu'étant classées dans la rubrique mouvement de données, concernent plus particulièrement des adresses avec, en premier lieu, LEA qui signifie Load Effective Address. Cette instruction a pour fonction de calculer une adresse réelle compte tenu du mode d'adressage utilisé et de charger la valeur ainsi obtenue dans un registre 16 bits. Elle s'utilise sous la forme :

LEA registre, adressage où registre est n'importe quel registre 16 bits autre qu'un registre de segment et où adressage est un des modes d'adressage valide du 8088. Attention, l'adresse chargée dans le registre 16 bits ne tient pas compte du procédé d'adressage réel du 8088 qui, rappelons-le, génère des adresses sur 20 bits en utilisant les registres de segments. C'est uniquement la valeur « hors segment » qui est chargée dans le registre. On voit d'ailleurs mal comment il pourrait en être autrement puisqu'une adresse 20 bits ne peut en aucun cas tenir dans un registre 16 bits.
Voyons maintenant deux instructions particulières qui sont LDS et LES. Jusqu'à maintenant, nous avons vu qu'il était impossible d'accéder directement à des registres de segments avec des instructions de mouvement des données classiques. LDS et LES permettent de faire cela pour les seuls registres DS (avec LDS) et ES (avec LES). Ces deux instructions s'utilisent de la façon suivante :

LDS ou LES registre pointeur, adresse mémoire.

Elles ont pour effet de charger, en une seule fois, le registre pointeur *et* le registre segment considéré (ES ou DS) avec la valeur trouvée en mémoire à l'adresse spécifiée. Comme les registres pointeurs et segments sont des registres 16 bits, l'adresse mémoire spécifiée pointe en fait sur un double mot, c'est-à-dire sur 32 bits.
Les registres pointeurs les plus utilisés en assembleur 8088 sont BK, SI et DI. Dernières instructions de mouvement des données, les instructions d'entrées/sorties ne sont pas les moins importantes. En effet, contrairement à la majorité des microprocesseurs du marché qui ne font aucune distinction entre leur espace mémoire pur et les circuits d'interfaces, le 8088 dispose d'instructions spécifiques pour dialoguer avec les boîtiers d'interface. Ces instructions ont pour noms IN ou OUT et s'utilisent de la façon suivante :
IN accumulateur, port,
OUT port, accumulateur
où accumulateur est le registre AL ou AX selon que l'on transfère un octet ou un mot et où port est le numéro du port d'entrée/sortie utilisé. Si ce numéro est inférieur à 255 (cas le plus courant), il peut être écrit directement dans l'instruction ; dans le cas contraire, il faut charger ce numéro dans le registre DX et utiliser ce registre à la place de port dans les deux exemples ci-avant.
IN lit la donnée sur le port concerné et place cette valeur dans l'accumulateur spécifié tandis que OUT place le contenu de l'accumulateur spécifié sur le port concerné.
Cette façon de faire étant propre aux microprocesseurs de la famille 8088 de chez Intel, nous estimons utile de dire quelques mots du cas le plus général qui est celui où les circuits périphériques (ou circuits d'entrées/sorties si vous préférez) sont placés dans le même espace d'adressage que la mémoire.
Dans un tel cas, chaque circuit périphérique comporte un certain nombre de registres internes, vus du microprocesseur comme des adresses mémoire tout à fait classiques. De ce fait, il n'existe aucune instruction spécifique d'entrée/sortie puisqu'il suffit au microprocesseur de faire des lectures ou des écritures « en mémoire » (en fait, dans les registres des circuits périphériques) pour effectuer ces dernières. Une écriture dans un registre conduira ainsi à une sortie de donnée et une lecture à une entrée.

OPERANDE 1	OPERANDE 2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Fig. 2. – Table de vérité des opérateurs logiques du 8088.

Les instructions arithmétiques et logiques

Nous allons voir tout d'abord les instructions logiques qui sont, il faut bien le reconnaître, d'un abord plus facile que certaines instructions arithmétiques.
Le 8088 dispose des trois opérateurs logiques classiques AND, OR et XOR, qui sont respectivement le ET logique, le OU inclusif logique et le OU exclusif logique. Les tables de vérité de ces trois fonctions sont rappelées sur la figure 2 pour ceux d'entre vous qui n'ont pas l'habitude de ces appellations. Ces trois instructions s'utilisent de la même façon, à savoir :

INSTRUCTION destination, source

Elles font l'opération logique entre source et destination et placent le résultat dans destination. Comme pour l'instruction MOV, tous les modes d'adressage peuvent être utilisés mais il est interdit de faire ces opérations directement entre deux mémoires ou en utilisant les registres de segments.
L'intérêt essentiel de ces instructions est de masquer ou de forcer certains bits des opérandes concernés. L'instruction AND sert, par exemple, à mettre à 0 sélectivement des bits particuliers. L'instruction OR par contre sert à forcer à 1 des bits particuliers tandis que XOR permet de mettre en évidence des bits différents entre deux opérandes.
Indépendamment de cela, ces instructions positionnent les bits du registre d'état en fonction du résultat de l'opération qu'elles effectuent ce qui permet, par exemple, de les faire suivre par des instructions de branchement conditionnel (que nous verrons plus avant dans cette série);
Dans certaines applications, ce seul positionnement des

bits du registre d'état est recherché afin de pouvoir faire un branchement conditionnel. Les instructions ci-avant sont donc mal adaptées car elles modifient l'opérande destination. Pour éviter cela, le 8088 propose TEST qui s'utilise exactement comme AND, qui fonctionne comme AND (c'est-à-dire qu'il faut bien un ET logique entre les opérandes concernés) mais qui se contente de pasitionner les bits du registre d'état. Le résultat du ET logique est perdu et ne vient en aucun cas modifier le contenu de l'opérande de destination.

Complément aux instructions précédentes, l'instruction NOT, qui s'utilise avec un seul opérande saus la forme :

NOT destination

effectue le complément bit à bit du contenu de destination c'est-à-dire change les 0 en 1 et *vice versa.* Contrairement aux instructions précédentes, NOT ne positionne aucun des indicateurs du registre d'état du 8088.

Outre ces opérations logiques « classiques », le 8088 sait aussi faire tourner et décaler le contenu de ses registres internes ou de ses mémoires grâce à 7 instructions particulières qui ont pour noms : SAR, SHL, SHR, ROL, ROR, RCL et RCR.

Elles s'utilisent toutes de la même façon :

Instruction registre ou mémoire, nambre de décalages.

Registre ou mémoire est le nom du registre ou l'adresse mémoire qui doit subir le décalage ou la rotation, tandis que nombre indique le nombre de bits à décaler. S'il y a un seul bit, la constante 1 peut être écrite dans nombre. Dans le cas contraire, le nombre de bits à décaler doit avoir été placé au préalable dans le registre CL qui prend alors la place de nombre dans la formulation ci-avant.

La figure 3, mieux qu'un long discours, montre quels sont les effets de ces diverses instructions. Nous vous ferons remarquer que les instructions ayant un A dans le mnémonique (SAR, SAL) sont des instructions de décalage arithmétique et conservent donc le signe du nombre décalé (revoir nos précédents articles où il était question de numération et de binaire si nécessaire pour le codage des signes). Remarquez aussi que deux formes de rotations vous sont proposées, en passant par le bit CF ou sans l'utiliser, ce qui permet de faire des rotations 8 et16 bits ou 9 et 17 bits.

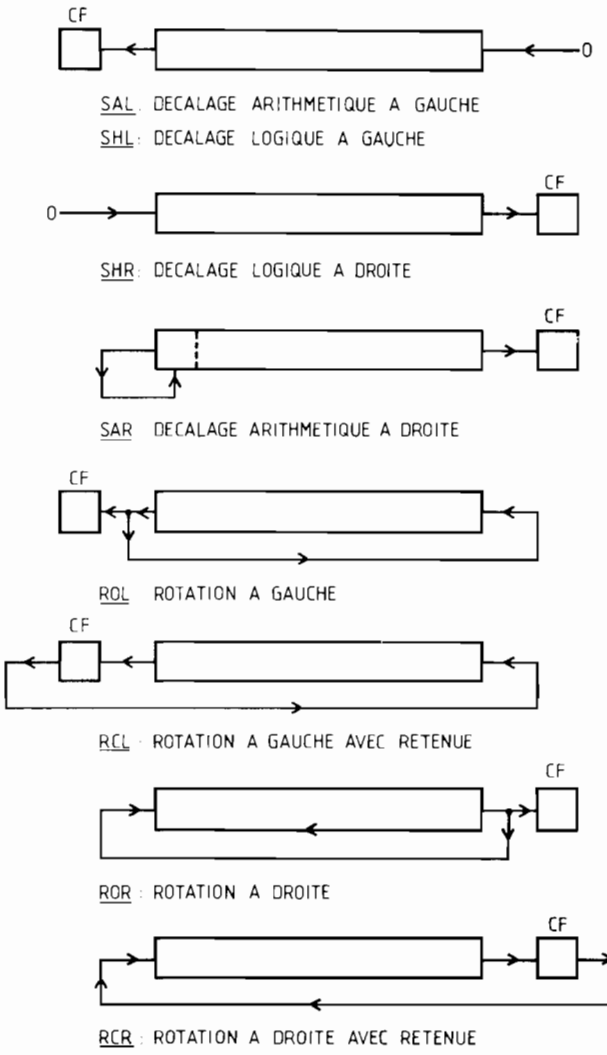


Fig. 3
Les diverses instructions de décalage et de rotation du 8088.

Conclusion

Nous en resterons là pour aujourd'hui afin que vous ne fassiez point d'indigestion de langage machine. Nous poursuivrons cette présentation du jeu d'instructions le mois prochain et nous serons alors fin prêts pour commencer à programmer notre 8088 en langage machine. Cela nous conduira tout naturellement à découvrir des « nouveautés » qui constitueront la suite logique de cette initiation.

C. TAVERNIER