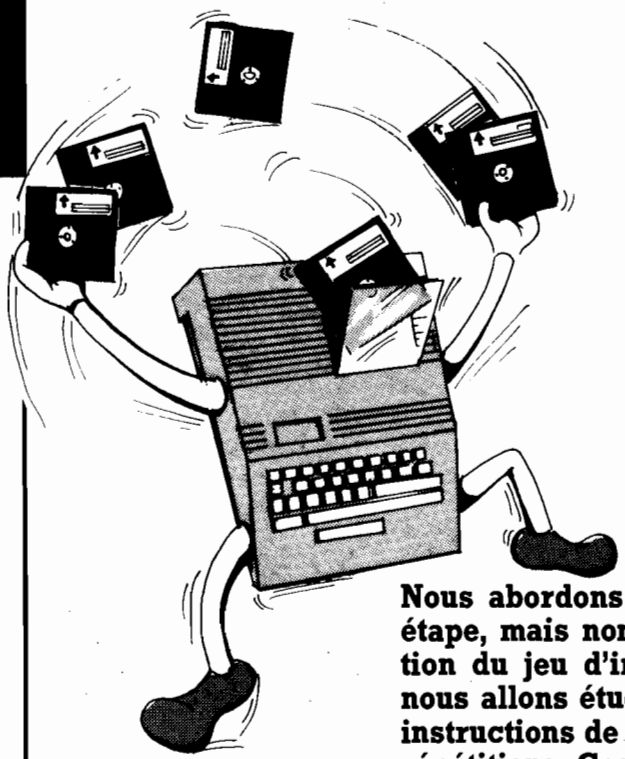
INITIATION

L'ABC DE LA MICRO-INFORMATIQUE



LE JEU D'INSTRUCTIONS DU 8088

Nous abordons aujourd'hui l'avant-dernière étape, mais non la moindre, de la présentation du jeu d'instructions du 8088, puisque nous allons étudier, entre autres choses, les instructions de sauts, de branchements et de répétitions. Ces dernières sont extrêmement importantes puisque c'est grâce à elles qu'un programme n'est plus limité à une suite d'instructions toujours exécutées dans le même ordre. En effet, nous allons voir qu'il va être possible de sauter des morceaux de programmes, d'en répéter d'autres et d'utiliser des sous-programmes ou procédures, tout comme en n'importe quel langage évolué.

LES SOUS-PROGRAMMES

Supposons que vous souhaitiez écrire un programme qui, sur votre PC, utilise un certain nombre d'informations que vous serez amené à frapper au clavier. Pour lire le clavier de votre PC et récupérer le code des touches qui sont frappées, vous allez devoir écrire un certain nombre d'instructions propres à la structure matérielle de l'interface clavier de cet appareil. Vous concevez bien que si votre programme, que nous appellerons le programme principal, doit exploiter dix touches frappées au clavier, vous n'allez pas réécrire dix fois le même bloc d'instructions pour lire le clavier. Vous allez faire un sous-programme qui contiendra juste ce qu'il faut pour lire une touche. Ce sous-programme sera appelé aux moments opportuns par le programme principal et fournira à ce dernier le code de la touche frappée dans un registre déterminé.

Un sous-programme, en langage machine ou assembleur, n'a que très peu de contraintes à respecter. Tout d'abord, il faut savoir qu'aucune instruction particulière n'en signale le début. Il faut donc le placer judicieusement dans votre programme principal pour qu'on ne puisse pas y entrer et, donc, l'exécuter « par erreur ». Il doit, en revanche, se terminer par une instruction de fin particulière qui est RET (que nous verrons dans un instant) qui assure le retour correct au programme principal. Enfin, mais cela tombe sous le sens, il faut faire attention, surtout si vous avez l'habitude des langages évolués, au fait que le microprocesseur n'a qu'un jeu de registres internes. Ne confondez donc pas ces derniers avec des noms de variables locales ou globales telles qu'on les rencontre dans les langages évolués. Si votre programme principal utilise le registre AX et que son contenu ne doit pas être affecté, il est évidemment interdit d'utiliser ce même AX dans les sous-programmes appelés par le programme principal (sauf bien sûr si vous le sauvegardez sur la pile grâce à l'instruction PUSH par exemple).

Cela étant vu, détaillons un peu la procédure d'utilisation d'un sous-programme. Ce dernier est appelé autant de fois que nécessaire grâce à des instructions adéquates placées dans le programme principal. Lorsque le 8088 rencontre une telle instruction, il commence par sauvegarder sur la pile la valeur du pointeur d'instructions (ou compteur ordinal) IP. En effet, ce pointeur contient l'adresse de l'instruction qui suit, celle d'appel du sous-programme, c'est-à-dire l'adresse de l'instruction qu'il faudra venir exécuter après la fin de ce dernier ; cette adresse s'appelle encore l'adresse de retour du sous-programme. Le pointeur d'instructions est ensuite chargé avec l'adresse qui suit l'instruction d'appel, adresse qui n'est autre que celle de début du sous-programme. Le 8088 peut donc aller exécuter la première instruction de ce dernier, puis celles qui suivent

jusqu'à rencontrer le RET de fin. Cette instruction a pour effet de récupérer sur la pile la valeur de IP qui avait été sauvegardée lors de l'appel du sous-programme et de la charger dans IP. L'exécution du programme principal peut alors reprendre au niveau de l'instruction qui suit celle d'appel du sous-programme.
Ce fonctionnement est parfaitement logique et n'est pas propre au 8088 puisqu'on le retrouve sur tous les microprocesseurs existants. Il appelle deux remarques très importantes. La première est qu'il est possible d'imbriquer des sous-programmes les uns dans les autres à l'infini ou presque. En effet, la seule limite est due à la taille de la pile qui ne doit pas déborder sous peine de perdre les adresses de retour. Compte tenu des tailles mémoires adressables par les microprocesseurs actuels, cette limite n'est, en pratique, jamais atteinte. La deuxième remarque, encore plus importante, est qu'il ne faut pas faire n'importe quoi avec la pile dans le sous-programme. En effet, l'instruction de fin du sous-programme suppose que le premier mot se trouvant sur la pile est la valeur à charger dans IP. Si, pour une raison ou pour une autre, vous avez modifié la valeur du pointeur de pile, vous risquez de récupérer n'importe quoi en fin de sous-programme et, donc, de retourner n'importe où.
Ceci ne signifie pas qu'il est interdit d'utiliser la pile dans les sous-programmes, mais que, si vous l'utilisez, il faut le faire « proprement » c'est-à-dire faire autant de sauvegardes que de récupérations afin de laisser la pile, en fin de sous-programme, dans l'état où elle se trouvait au début.

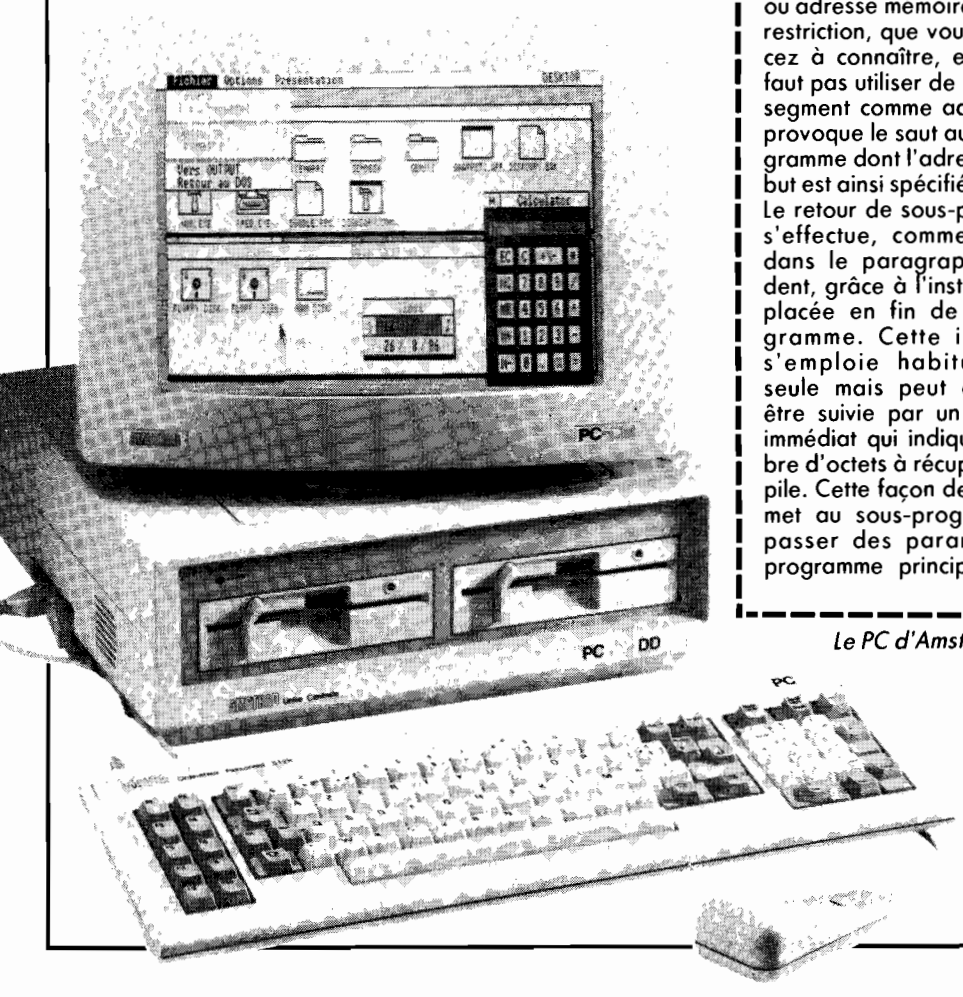
LES APPELS DE SOUS-PROGRAMMES

Maintenant que tous ces mécanismes ont été bien précisés, la présentation des instructions relatives aux sous-programmes va devenir un jeu d'enfant.
La première instruction d'appel est CALL. La syntaxe est fort simple puisque CALL est suivie d'une adresse exprimée dans le mode d'adressage de votre choix (immédiat, registre ou adresse mémoire). La seule restriction, que vous commencez à connaître, est qu'il ne faut pas utiliser de registre de segment comme adresse. Elle provoque le saut au sous-programme dont l'adresse de début est ainsi spécifiée.
Le retour de sous-programme s'effectue, comme expliqué dans le paragraphe précédent, grâce à l'instruction RET placée en fin de sous-programme. Cette instruction s'emploie habituellement seule mais peut également être suivie par un opérande immédiat qui indique le nombre d'octets à récupérer sur la pile. Cette façon de faire permet au sous-programme de passer des paramètres au programme principal. Il faut cependant redoubler de prudence lorsque l'on utilise cette possibilité, compte tenu des remarques faites ci-avant sur l'utilisation « propre » de la pile.

Sur la majorité des assembleurs 8088, CALL utilisée seule suppose que le sous-programme appelé se trouve dans le même segment que le programme principal. Sur certains assembleurs, il faut cependant écrire CALL NEAR (mot à mot : « appel près ») pour cela. Si le sous-programme se trouve dans un autre segment mémoire que le programme principal, il faut écrire CALL FAR qui est suivie d'un opérande qui peut revêtir divers aspects (comme pour CALL) mais qui est toujours un double mot. Le mot d'adresse la plus basse est chargé dans le pointeur d'instruction IP tandis que l'autre mot est chargé dans le registre de segment CS. Il faut noter que, dans ce cas, la valeur contenue dans le registre CS est sauvegardée sur la pile au même titre que le pointeur d'instructions IP ; en effet, si tel n'était pas le cas, le retour serait impossible puisque le contenu initial de CS est modifié par l'exécution du CALL FAR.

De ce fait, l'instruction de retour d'un sous-programme appelé de la sorte ne peut plus être RET (puisqu'elle ne fait récupérer que le registre IP sur la pile) mais devient RETF. Elle fait récupérer sur la pile le contenu de IP et celui de CS.

Ces distinctions étant assez contraignantes lorsque l'on programme en assembleur, les assembleurs évolués savent contrôler si les appels de sous-programmes se font dans le même segment ou entre segments différents, et le programmeur n'a plus à se soucier de quoi que ce soit ; il lui suffit d'écrire des CALL et des RET pour que ceux-ci soient automatiquement codés comme il faut.



Le PC d'Amstrad.

LES INSTRUCTIONS DE SAUTS INCONDITIONNELS

Il est parfois nécessaire de rompre l'ordre d'exécution séquentiel d'un programme sans nécessiter de retour au point de rupture. Il faut alors faire appel pour cela à une instruction de saut ou branchement inconditionnel. Comme sur tous les microprocesseurs actuels, cette instruction s'appelle, en 8088, JMP (abréviation de *jump* qui signifie sauter). Elle s'utilise suivie d'une adresse exprimée dans le mode d'adressage de votre choix et fait poursuivre le déroulement du programme à l'adresse ainsi spécifiée. Aucune sauvegarde n'est effectuée sur la pile puisqu'un tel saut n'implique aucune idée de retour.

Comme pour l'instruction CALL, il existe deux versions de JMP selon que l'on saute dans le même segment ou non (JMP et JMP FAR), mais les assembleurs évolués savent, là aussi, faire automatiquement le bon choix.

COMPARER POUR MIEUX SAUTER

Jusqu'à présent, nous avons choisi de dérouter un programme volontairement, soit par appel de sous-programme, soit par saut à une adresse de notre choix. Il est possible de faire une telle opération en fonction d'événements à priori inconnus grâce aux sauts ou branchements conditionnels. De telles instructions permettent donc à un programme de « prendre des décisions » en fonction de données non définies à l'avance puisque, selon la valeur de ces dernières, telle ou telle partie de programme sera exécutée.

Les instructions de sauts conditionnels exploitent les valeurs des bits du registre d'état du 8088 pour effectuer ou non les sauts. Ces bits, rappelons-le, sont positionnés par l'exécution de certaines instructions (logiques et arithmétiques en particulier), mais il est aussi possible de les positionner grâce à une instruction de comparaison. C'est d'ailleurs cette dernière que l'on rencontre habituellement avant un saut conditionnel.

La syntaxe d'utilisation de cette instruction est tout simplement :

CMP opérande 1, opérande 2

où opérande 2 est un registre, une adresse mémoire ou une constante.

Dans la pratique, cette instruction « compare » les deux opérandes en faisant opérande 1 – opérande 2 et en positionnant les bits du registre d'état en fonction du résultat obtenu. Notez bien que cette soustraction est purement fictive et que son résultat n'est disponible nulle part. De ce fait, aucun des opérandes n'est modifié par CMP.

LES BRANCHEMENTS CONDITIONNELS

Ces instructions fonctionnent en partie comme JMP en ce sens que le saut qu'elles déclenchent n'est suivi d'aucun retour. En revanche, le saut n'a lieu que si une condition, spécifiée par l'instruction elle-même, est réalisée. Dans le cas contraire, le programme continue en séquence, comme si l'instruction de saut n'existait pas.

Ces instructions sont au nombre de 17 et sont présentées figure 1. Comme vous pouvez le constater, tous les mnémoniques commencent par J (pour Jump) et sont suivis d'une, deux ou trois lettres qui rappellent quelle est la condition effectivement testée. Pratiquement, cette condition correspond au test d'un ou plusieurs bits du registre d'état du 8088. Les bits mis en cause sont indiqués dans le tableau de la figure 1 mais, lorsqu'on programme, il est beaucoup plus facile de ne tenir compte que de la signification du mnémonique de l'instruction. Un exemple permet de mieux comprendre ce que nous voulons dire par là. Si, après une comparaison, vous voulez faire un branchement si les deux valeurs comparées sont égales, vous écrirez tout naturellement JE (qui signifie *Jump on Equal* ou « saut si égal ») alors qu'autrement il vous faut vérifier que CMP positionne le bit ZF à 1 si les deux valeurs comparées sont égales et que JE fait un saut si, justement, ZF est à 1.

Pour vous aider dans cette tâche, vous trouverez en figure 2 les branchements à utiliser en fonction des valeurs relatives de deux opérandes

INSTRUCTION	CONDITION DE BRANCHEMENT
JA ou JNBE	CF = 0 et ZF = 0
JAE ou JNC ou JNB	CF = 0
JB ou JC ou JNAE	CF = 1
JBE ou JNA	CF = 1 ou ZF = 1
JCXZ	Registre CX = 0
JE ou JZ	ZF = 1
JNE ou JNZ	ZF = 0
JNP ou JPO	PF = 0
JP ou JPE	PF = 1
JG ou JNLE	ZF = 0 et OF = SF
JGE ou JNL	OF = SF
JL ou JNGE	OF = SF
JLE ou JNG	ZF = 1 ou OF = SF
JNO	OF = 0
JNS	SF = 0
JO	OF = 1
JS	SF = 1
De JG inclus à JS inclus, instructions pour entiers signés.	

Fig. 1. – Les instructions de branchements conditionnels du 8088.

NON SIGNES	SIGNES	SAUT POUR
JE	JE	Opérande 2 = Opérande 1
JNE	JNE	Opérande 2 = Opérande 1
JA	JG	Opérande 2 > Opérande 1
JAE	JGE	Opérande 2 >= Opérande 1
JB	JL	Opérande 2 < Opérande 1
JBE	JLE	Opérande 2 <= Opérande 1
CMP Opérande 1, Opérande 2		

Fig. 2. – Quelle instruction utiliser en fonction des valeurs relatives des opérandes comparées ?

mis en présence dans une instruction CMP.
Le tableau de la figure 3 indique quant à lui la signification exacte des mnémoniques car il faut bien avouer qu'il n'est pas toujours évident que JLE est l'abréviation de *Jump on Less than or Equal.*
A ce propos, remarquez qu'une distinction a été faite dans le tableau de la figure 1 selon que l'on utilisait des entiers signés ou non. En effet, il n'est pas possible d'employer les mêmes critères de tests pour les entiers non signés et signés puisque, chez ces derniers, le bit de poids fort est le bit de signe.
Cela étant vu, sachez que ces instructions s'utilisent exclusivement sous la forme :
JXXX adresse courte
où adresse courte est n'importe quelle adresse située à moins de – 128 ou + 127 octets de distance de l'adresse où se trouve le JXXX. Une telle limitation est évidemment contraignante et impose parfois de mettre dans les programmes des « relais de saut » qui, en utilisant des branchements inconditionnels après des conditionnels, permettent d'aller plus loin.

LES INSTRUCTIONS DE BOUCLE

Si les instructions précédentes étaient communes à de très nombreux microprocesseurs, celles que nous allons voir maintenant sont plus originales et sont propres au 8088. Elles permettent en effet de réaliser une boucle, c'est-à-dire de faire répéter, autant de fois qu'on le désire, tout un bloc d'instructions. Pour cela il faut utiliser l'instruction LOOP sous la forme :
LOOP adresse courte, où adresse courte correspond à une adresse mémoire située de – 128 à + 127 octets de distance de LOOP. Toutes les instructions comprises entre cette adresse et LOOP seront répétées N fois, N ayant été placé au préalable dans le registre compteur CX du 8088. Ce registre est en effet décrémenté automatiquement de 1 à chaque tour de boucle et LOOP fait terminer la boucle lorsque CX devient nul.
Si une boucle doit être exécutée un nombre de fois inconnu à l'avance, l'instruction LOOP ne peut être utilisée mais il est alors possible de faire appel à LOOPE ou LOOPZ (ce sont deux mnémoniques différents pour la même instruction). Cette instruction s'utilise comme LOOP mais fait terminer la boucle lorsque CX devient nul ou lorsque le bit ZF du registre d'état du 8088 devient nul. Il est ainsi possible d'arrêter une boucle en fonction d'une condition (événement extérieur, résultat de calcul) dès lors que cette dernière peut être reflétée par l'état du bit ZF.
L'instruction complémentaire est aussi disponible sous la forme LOOPNE ou LOOPNZ qui fait terminer la boucle lorsque CX est nul ou lorsque ZF passe à 1. Elle s'utilise bien évidemment comme LOOP et LOOPNE.

CONCLUSION

Rassurez-vous, votre calvaire touche à sa fin et, dès le mois prochain, nous pourrons écrire quelques exemples de programmes et aborder d'autres sujets émanant directement de ce que nous avons appris sur la programmation en langage machine.

C. TAVERNIER

MNEMONIQUE	SIGNIFICATION	
JA	Jump on Above	Saut si supérieur
JNBE	Jump on Not Below or Equal	Saut si pas inférieur ou égal
JAE	Jump on Above or Equal	Saut si supérieur ou égal
JNB	Jump on Not Below	Saut si pas inférieur
JB	Jump on Below	Saut si inférieur
JNAE	Jump on Not Above qr Equal	Saut si non supérieur ou égal
JBE	Jump on Below or Equal	Saut si inférieur ou égal
JNA	Jump on Not Above	Saut si pas supérieur
JC	Jump on Carry	Saut si retenue
JCXZ	ump on CX register Zero	Saut si registre CX à zéro
JE	Jump on Equal	Saut si égal
JZ	Jump on Zero	Saut si zéro
JG	Jump on Greater than	Saut si plus grand que
JNLE	Jump on Not Less or Equal	Saut si non inférieur ou égal
JGE	Jump on Greater or Equal	Saut si supérieur ou égal
JNL	Jump on Not Less	Saut si non inférieur
JL	Jump on Less	Saut si inférieur
JNGE	Jump on Not Greater or Equal	Saut si non inférieur ou égal
JLE	Jump on Less than or Equal	Saut si inférieur ou égal
JNG	Jump on Not Greater than	Saut si pas supérieur à
JNC	Jump on Not Carry	Saut si pas de retenue
JNE	Jump on Not Equal to	Saut si pas égal à
JNZ	Jump on Not Zero	Saut si pas zéro
JNO	Jump on Not Overflow	Saut si pas d'overflow
JNS	Jump on Not Sign	Saut si signe à zéro
JNP	Jump on Not Parity	Saut si pas de parité
JPO	Jump on Parity Odd	Saut si parité impaire
JO	Jump on Overflow	Saut si overflow
JP	Jump on Parity even	Saut si parité paire
JPE	Jump on Parity equal	Saut si parité égale
JS	Jump on Sign	Saut si signe à un

Fig. 3. – Signification des mnémoniques des instructions de branchements conditionnels. Overflow = dépassement de capacité.