Chapitre 21

Les interruptions

Comme leur nom l'indique, les *interruptions* (*interrupt* en anglais) viennent interrompre le déroulement normal du microprocesseur. Elles sont générées par appel d'un périphérique.

Elles donnent lieu à l'exécution d'un programme, appelé **routine de service** ou **routine de traitement de l'interruption** (en anglais *interrupt service routine*, abrégé en ISR, ou *interrupt handler*), en se souvenant que **routine** est un synonyme de sous-programme.

21.1 Types d'interruptions

On distingue trois types d'interruptions d'après la façon dont elles sont appelées : les interruptions appelées par la microprocesseur lui-même, celles appelées par le logiciel et celles appelées par les périphériques. On parle quelquefois d'**exceptions** pour les deux premiers types.

<u>Interruptions appelées par le microprocesseur</u>.- Les concepteurs du 8086 ont réservé les interruptions de numéro 00H à 12h pour être appelées par le microprocesseur. On parle quelquefois d'**interruptions internes**. Ces interruptions sont appelées par le microprocesseur en général en réponse à une erreur, par exemple l'interruption 00h lorsqu'on essaie de diviser par 0.

<u>Interruptions logicielles</u>.- Ces interruptions sont appelées par le programme.

<u>Interruptions externes</u>.- Les périphériques peuvent déclencher une interruption grâce à deux broches du microprocesseur 8086 :

– La broche appelée NMI (pour *NonMaskable Interrupt*) en position haute déclenche une interruption de type 02h. Cette interruption n'est pas masquable, c'est-à-dire qu'elle peut intervenir à tout moment (on ne peut pas ne pas en tenir compte).

– La broche appelée INTR (pour l'anglais *INTeRrupt*) en position haute permet d'activer n'importe laquelle des 256 interruptions. Ceci se fait en plaçant le numéro de l'interruption désirée sur le bus des données.

Lorsque l'une de ces deux broches passe en position haute, le microprocesseur termine l'instruction qu'il est en train d'exécuter et passe à la routine de service de l'interruption correspondante.

21.2 Les interruptions internes

Les cinq premières interruptions, sur les douze réservées par Intel pour le microprocesseur, et donc la signification des 20 premiers octets, ont effectivement reçu une implémentation par Intel :

Interruption	Adresses	Fonction
0	00 à 03h	Division par 0
1	04 à 08h	Pas à pas (TRAP ou *Single-Step*)
2	09 à 0Bh	Interruption non masquable (NMI)
3	0Ch à 0Fh	Interruption logicielle sur 1 octet INT 3
4	10h à 13h	Interruption sur *overflow* INTO

mais c'est au concepteur du système d'y placer les adresses qu'il désire et de concevoir les routines de service adéquates. Ces dernières se conçoivent comme les routines pour les interruptions logicielles.

Exemple.- Les adresses ne sont pas imposées par les concepteurs du 8086 mais par les concepteurs des routines de service associées. Elles dépendent donc de la version du BIOS, de la version du système d'exploitation et des programmes utilisateur suivant celui qui l'a implémentée.

On peut se servir de `debug` pour obtenir les adresses de notre système. Par exemple pour l'interruption INT 3, on obtient sur mon système :

```
C:>debug
-d 0000:000C L4
0000:0000 65 04 70 00                  e.p.
-q
```

ce qui donne l'adresse `0070 :0465`.

21.3 Les interruptions logicielles

Nous avons rencontré la notion de sous-programme et nous avons vu comment mettre en place un tel sous-programme. Un sous-programme peut être conçu par l'utilisateur. Certains sous-programmes importants peuvent être conçus (avec beaucoup de soins) pour être réutilisés par un grand nombre d'utilisateurs. En général ils sont placés dans des *bibliothèques* ; nous verrons plus tard comment. Il est également prévu qu'un certain nombre d'entre eux, les plus importants, puissent être appelés par une méthode quelque peu spécifique : ce sont les **routines de service** associées à des *interruptions logicielles*.

21.3.1 Mise en place

21.3.1.1 Appel des interruptions

Syntaxe.- La syntaxe de l'appel d'une interruption logicielle en langage symbolique pour le microprocesseur 8086/8088 est tout simplement :

```
INT constante
```

où `constante` est un entier compris entre `00h` et `FFh`. Il y a donc 256 interruptions possibles pour le microprocesseur 8086.

Sémantique.- Reprenant l'instruction `RST x` du microproceseur Intel précédent, le 8085, cette instruction est équivalente à un appel long, mais les nouvelles valeurs de CS et de IP sont stockées à des adresses définies par construction.

Cette instruction provoque le stockage du registre des indicateurs dans la pile, la mise à zéro des indicateurs TF et IF et la sauvgarde des contenus de CS et IP. Les nouveaux contenus de CS et de IP sont chargés à partir des cases mémoire, le contenu de IP étant chargé en dernier. Le pointeur de pile est décrémenté de 6.

Puisqu'il y a 256 types, le premier KiO de la mémoire vive, les cases d'adresse 000h à 3FFh, est réservé aux interruptions, à raison de quatre octets par type, emplacement mémoire appelé **tableau des vecteurs d'interruption** (en anglais *interrupt vector table*). L'adresse de la première instruction de la routine de service de l'interruption se trouve à l'adresse $4 \times n$ pour le type n : les deux octets de IP se trouvent à l'adresse $4 \times n$ et les deux octets de CS à l'adresse $4 \times n + 2$.

Il n'existe pas en général 256 interruptions implémentées ; leur nombre dépend du système d'exploitation et de ce qui a été prévu par les grands logiciels utilisateur.

Langage machine.- L'instruction d'interruption est codée sur un ou deux octets par :

`| 1100 110v | type si` $v = 1$ `|`

avec $v = 0$ pout INT 3.

21.3.1.2 Retour d'une interruption

Langage symbolique.- La routine de service d'une interruption doit se terminer par l'instruction :

```
IRET
```

pour *Interrupt RETurn*.

Sémantique.- Les contenus de CS, de CS et du registre des indicateurs sont rechargés depuis la pile. Le pointeur de pile est incrémenté de 6.

Langage machine.- Cette instruction est codée sur un octets par `1100 11011`, soit &HCF.

21.3.2 Un exemple d'appel d'interruption logicielle

Introduction.- Nous avons vu le principe de mise en place des entrées-sorties en utilisant les instructions IN et OUT. Nous ne sommes pas allés très loin pour l'instant, vu la délicatesse de la mise en œuvre.

Certaines fonctionnalités sont réalisées par le BIOS et implémentées comme interruptions logicielles. Nous n'allons pas écrire d'interruption pour l'instant, mais en utiliser une du BIOS.

Les interruptions du BIOS.- Puisqu'il n'y a que 256 interruptions possibles et que les concepteurs du BIOS et du système d'exploitation de l'IBM-PC avaient envie d'en implémenter plus, ils ont associés plusieurs actions à une même interruption logicielle. Chaque action s'appelle une **fonction** de l'interruption et se distingue des autres actions par la valeur du registre AH, appelée **numéro de la fonction**.

Par exemple, avec le BIOS de l'IBM-PC, on utilise la fonction 0Ah de l'interruption 10h pour afficher un caractère : le registre AL doit contenir le numéro du caractère à afficher, le registre BH le numéro de page (qui ne nous intéressera pas pour l'instant) et le registre CX le nombre de fois qu'il faut l'afficher.

Le **numéro du caractère** est son code ASCII sauf pour les 32 premiers caractères, qui sont des caractères dits **semi-graphiques** suivant un code déterminé par IBM. Une conséquence de ce choix est que les caractères de contrôle (bip, retour chariot...) ne sont pas actifs.

Le premier caractère est affiché à la position du curseur puis en continuant en allant à la ligne si besoin est.

Exemple.- Écrivons un programme permettant d'afficher 120 'a' à partir de la position du curseur. Rappelons que le code ASCII de 'a' est 61h et que 120 = 78h. Écrivons un sous-programme, d'abord en langage symbolique :

```
MOV AH, 0A
MOV AL, 61
MOV BH, 0
MOV CX, 78
INT 10
RETF
```

puis en langage machine :

```
&HB4 &H0A
&HB0 &H61
&HB7 &H00
&HB9 &H78 &H00
&HCD &H10
&HCB
```

Utilisons alors le programme QBasic suivant, sachant que le code occupe 12 octets :

```
CLS
DIM PM%(5)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 11
  READ Octet$
  POKE OffPM% + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
```

```
'Execution du programme machine
CALL ABSOLUTE(OffPM%)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA B4,0A
DATA B0,61
DATA B7,00
DATA B9,78,00
DATA CD,10
DATA CB
```

L'exécution :

```
B4    A     B0     61    B7
0     B9    78     0     CD
10    CB    0      0     0

B4    A     B0     61    B7aaaaaaaaaaaaaaaaaaaaa
0     B9    78     0     CD
10    CB    0      0     0
```

peut peut-être sembler bizarre mais correspond bien à ce qui a été demandé : 120 'a' sont affichés, mais sans déplacer le curseur ; une partie des ces 'a' est donc détruite par l'affichage suivant.

21.3.3 Un exemple de routine d'interruption logicielle

21.3.3.1 Trace d'un programme avec interruption

Introduction.- Nous avons vu comment effectuer la trace d'un programme avec le logiciel `debug`. Il y a un problème lorsqu'un programme contient des interruptions. En effet si on utilise la commande `T`, lorsqu'on arrive à l'interruption, on va décrire toutes les instructions de la routine de celle-ci. Ce n'est pas ce qui est voulu en général : on suppose que le concepteur de l'interruption a mis en place la routine de service associée de façon correcte. On préférerait donc une commande qui exécute la routine de service en son entier et nous montre le résultat après celle-ci.

La commande P.- Une telle commande existe pour `debug`, il s'agit de la commande `P` (pour l'anglais *to Proceed*).

Exemple.- Déterminons la capacité de la mémoire vive de notre ordinateur.

En ce qui concerne les compatibles PC, celle-ci était déterminée au moment du démarrage de l'ordinateur et le résultat placé à l'emplacement de la mémoire vive (réservé par le BIOS) `0040 :0013h`.

Il existe une interruption du BIOS, l'interruption logicielle `int 12h`, qui permet de récupérer cette valeur et de la placer dans le registre `ax`.

Concevons donc un programme permettant de déterminer la taille de la mémoire :

```
C:\>debug
-a
17A4:0100 int 12
17A4:0102 nop
17A4:0103
```

Exécutons maintenant la trace de ce programme :

```
C:\>debug
-a
17A4:0100 int 12
17A4:0102 nop
17A4:0103
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0100   NV UP EI PL NZ NA PO NC
17A4:0100 CD12          INT     12
-P

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0102   NV UP EI PL NZ NA PO NC
17A4:0102 90            NOP
-
```

qui nous donne la valeur. Celle-ci doit de nos jours être interprétée car nous dépassons la capacité mémoire prévue lors de la conceptions des premières versions du BIOS.

21.3.3.2 La routine de service

Si on effectue la trace de ce programme en utilisant la commande `T` au lieu de la commande `P`, on obtient :

```
C:\>debug
-a
17A4:0100 int 12
17A4:0102 nop
17A4:0103
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0100   NV UP EI PL NZ NA PO NC
17A4:0100 CD12          INT     12
-t

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=F000  IP=F841   NV UP DI PL NZ NA PO NC
F000:F841 1E            PUSH    DS
-t

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFE6  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=F000  IP=F842   NV UP DI PL NZ NA PO NC
F000:F842 B84000        MOV     AX,0040
-t

AX=0040  BX=0000  CX=0000  DX=0000  SP=FFE6  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=F000  IP=F845   NV UP DI PL NZ NA PO NC
F000:F845 8ED8          MOV     DS,AX
-t

AX=0040  BX=0000  CX=0000  DX=0000  SP=FFE6  BP=0000  SI=0000  DI=0000
DS=0040  ES=17A4  SS=17A4  CS=F000  IP=F847   NV UP DI PL NZ NA PO NC
```

```
F000:F847 A11300        MOV     AX,[0013]              DS:0013=0280
-t

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFE6  BP=0000  SI=0000  DI=0000
DS=0040  ES=17A4  SS=17A4  CS=F000  IP=F84A   NV UP DI PL NZ NA PO NC
F000:F84A 1F            POP     DS
-t

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=F000  IP=F84B   NV UP DI PL NZ NA PO NC
F000:F84B CF            IRET
-t

AX=0280  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0102   NV UP EI PL NZ NA PO NC
17A4:0102 90            NOP
-
```

qui entre dans le détail de la routine de service de l'interruption.

Ici celle-ci est simple : elle consiste à aller chercher le renseignement là où il est placé.

21.4 Masquage des interruptions externes masquables

Il peut arriver que l'on ait besoin, lors de l'exécution d'une routine, qu'aucune interruption (nécessairement matérielle) ne soit prise en compte, pour ne pas gêner le bon déroulement du programme. Ceci est possible pour les interruptions externes masquables, c'est-à-dire celles déclenchées par la broche INTR.

Langage symbolique. - On se sert pour cela de l'instruction :

`CLI`

(pour l'anglais *Clear Interrupt Flag*), qui a pour effet de positionner à 0 l'indicateur IF. Ceci n'a aucun effet sur les interruptions externes non masquables (déclenchées par la broche NMI) ou les interruptions logicielles.

Pour permettre à nouveau l'exécution des interruptions masquables, on utilise l'instruction :

`STI`

(pour l'anglais *SeT Interrupt flag*), qui a pour effet de positionner à 1 l'indicateur IF.

Langage machine.- 1^o) L'instruction CLI est codé sur un octet par `1111 1010`, soit FAh.
- 2^o) L'instruction STI est codé sur un octet par `1111 1011`, soit FBh.