## Première partie

# Programmation avec les macro-instructions du BIOS

# Chapitre 1

# Programmer avec le BIOS

Nous avons vu comment programmer en langage machine en passant en revue les instructions implémentées dans le microprocesseur, à savoir le 8088 dans notre cas. Nous avons évidemment été gêné par le fait que nous ne puissions pas utiliser les périphériques, c'est-à-dire réaliser des entrées grâce au clavier, un affichage sur le moniteur et, dans une moindre mesure jusqu'ici, accéder au disque dur pour sauvegarder les fichiers.

Pour pouvoir programmer les périphériques, il faut connaître l'architecture de l'ordinateur construit autour du microprocesseur, qui comprend de nombreux autres circuits intégrés, comprenant eux-mêmes des *registres* permettant de les programmer.

On peut évidemment en théorie programmer ainsi directement mais ceci est assez lourd. On se sert donc du principe d'encapsulation pour ajouter des pseudo-instructions au microprocesseur.

## 1.1 Étude générale

### 1.1.1 Pseudo-instructions d'un microprocesseur

La plupart des microprocesseurs prévoient la mise en place de **pseudo-instructions**, ou **appels système**, sous la forme d'interruptions logicielles.

Ceci permet de construire une couche supérieure, standardisée et indépendante du matériel. C'est lors de l'écriture de la routine de service de l'interruption que l'on tient compte du matériel effectivement utilisé.

### 1.1.2 Notion de BIOS

Nous avons déjà dit que les premiers micro-ordinateurs contenaient le système d'exploitation et un interpréteur BASIC en mémoire ROM, ce qui fait que l'on pouvait y accéder dès le démarrage. L'évolution rapide du matériel, d'une part, des fonctionnalités ajoutées au système d'exploitation, d'autre part, exigeaient de changer souvent la ROM contenant ces éléments, c'est-à-dire, en pratique, de changer d'ordinateur.

Cette façon de faire n'était donc pas très pratique puisque le système était figé. Ceci a conduit à diviser le système d'exploitation en deux parties : une partie résidente en mémoire ROM, inamovible, ne contenant que l'essentiel, appelée BIOS (pour*Basic Input Output System*), et une partie, nettement plus importante en taille, chargée à partir de la première, pouvant changer souvent.

Le nom BIOS lui-même provient du fait que la première partie doit implémenter la façon d'accéder au clavier, au moniteur et à la mémoire de masse (disquette, disque dur) pour pouvoir charger la seconde partie.

Le concept de BIOS est devenu si répandu qu'il a fini par permettre de charger plusieurs systèmes d'exploitation. Il est donc devenu une partie du logiciel système à part entière.

### 1.1.3 Liste des interruptions du BIOS d'IBM

Les macro-instructions définies par IBM pour le BIOS du premier PC sont devenues un standard de fait.

Rappelons que les concepteurs du 8086 ont réservé les interruptions 00h à 12h comme interruptions internes. Le BIOS implémente les routines de service de certaines de ces interruptions (en particulier l'interruption 00h), utilise les interruptions 13h à 1Bh et en fait également les interruptions 05h, 08h, 09h et 0Bh qui ne sont pas utilisées sur le 8086 bien que réservées.

On a :

— `INT 00h` : **Division par zéro**. Le gestionnaire est appelé lorsqu'un essai de division par zéro est tenté. Le BIOS de l'IBM PC se contente de faire apparaître un message et suspend le système.

— `INT 01h` : **Étape par étape**. Cette interruption est utilisée par *debug* et d'autres débogueurs pour exécuter un programme ligne par ligne.

— `INT 03h` : **Point d'arrêt**. Cette interruption est utilisée par *debug* et d'autres débogueurs pour arrêter l'exécution d'un programme.

— `INT 04h` : **Dépassement de capacité**.

Les interruptions dont la fonctionnalité est définie par le BIOS sont :

— `INT 05h` : **Impression de l'écran**.

— `INT 08h` : **Temporisateur du système**.

— `INT 09h` : Interruption du **clavier** avec de nombreuses fonctions.

— `INT 0Bh` : contrôle du **premier port série** COM1.

— `INT 0Ch` : contrôle du **deuxième port série** COM2.

— `INT 0Dh` : contrôle du **deuxième port parallèle** LPT2.

— `INT 0Eh` : contrôle des **lecteurs de disquette**.

— `INT 0Fh` : contrôle du **premier port parallèle** LPT1.

— `INT 10h` : interruption avec de nombreuses fonctions concernant le **moniteur**.

— `INT 11h` : identification de la **configuration du PC**, utilisée en particulier lors du démarrage de l'ordinateur.

— `INT 12h` : détermination de la **taille mémoire** (jusqu'à 640 KiO, taille maximale gérée par MS-DOS), en particulier lors du démarrage de l'ordinateur.

— `INT 13h` : fonctions concernant les entrées-sorties sur les **disquettes** et les **disques durs**.

— `INT 14h` : contrôle des **communications série**.

— `INT 15h` : contrôle du **lecteur de cassette**.

— `INT 16h` : entrée au **clavier**.

— `INT 17h` : sortie sur l'**imprimante**.

— `INT 18h` : entrée en **BASIC** situé en mémoire ROM. Cette interruption était en particulier appelée lorsqu'aucun disque système n'était trouvé. Cette interruption est inactive en général de nos jours.

— `INT 19h` : démarrage de l'**amorce du système**. Si un disque système est présent, le système est chargé, sinon il est fait appel à l'interruption `int 18h`.

— `INT 1Ah` : lit le **temps** (date et heure) ou le met à jour.

— `INT 1Bh` : prend le contrôle lors d'une interruption (**CTRL-break**) effectuée au clavier.

## 1.2 Le clavier

### 1.2.1 Principe d'implémentation du clavier

Sur l'IBM-PC, lorsqu'on presse une touche du clavier, le micro-contrôleur du clavier détermine le *code clavier* (*scan code* en anglais) de la touche qui a été pressée (un octet) et fait appel à l'interruption matérielle 09h. La rotine de celle-ci rappatrie le code clavier et place celui-ci ainsi que le code ASCII du caractère correspondant dans le *tampon du clavier* :

- Le **code clavier** est un numéro unique assigné à chaque touche du clavier, que ce soit une touche de caractère standard ou une touche de caractère spécial. De ce point de vue la touche majuscule gauche, par exemple, est différente de la touche majuscule droite.
- Le **tampon du clavier** est une partie de la mémoire vive, de 15 caractères pour l'IBM-PC, permettant d'entreposer les caractères avant l'utilisation par le programme.
- On utilise de plus des **octets de statut** pour déterminer le caractère à placer dans le tampon. Si on a appuyé auparavant sur la touche majuscule et qu'on appuie maintenant sur la touche 'b' alors on doit placer 'B' dans le tampon.

Il y a deux interruptions liées au clavier, jouant des rôles différents :

- L'interruption 09h répond à une *demande d'interruption matérielle* ; elle ne sert pas à la programmation.
- L'interruption 16h permet de lire le *caractère situé dans le tampon du clavier* et de le transmettre au programme qui y a fait appel.

## 1.2.2 L'interruption 16h

### 1.2.2.1 Lire un caractère : fonctions 00h et 10h

Introduction.- La fonction 00h permet de lire un caractère du clavier initial à 83 touches situé dans le tampon du clavier. N'acceptant pas les touches supplémentaires telles que <F11> et <F12>, il a été créé la fonction 10h qui effectue la même chose, mais pour le clavier dit *étendu*.

Mise en place.- On place le numéro de fonction, ici 10h donc, dans le registre AH et on fait appel à l'interruption 16h. La routine de l'interruption vérifie s'il existe un caractère présent dans le tampon. S'il n'y en a pas, elle attend qu'il y en ait un (on dit que c'est une instruction **bloquante**). S'il y en a un, elle place le caractère ASCII (ou un code pour les touches non associées à un caractère ASCII) dans le registre AL et le code de recherche dans AH, suivant le tableau (non entièrement explicite) suivant :

| Touche appuyée | AH | AL |
|---|---|---|
| Caractère ASCII | Code de recherche | Code ASCII |
| Touche de fonction étendue | Code de recherche | 00h |
| Touche de controle dupliquée | Code de recherche | E0h |

Il n'y a pas écho du caractère à l'écran.

Exemple 1.- Exécuter le programme suivant dans `debug` :

```
MOV AH,0
INT 16H
INT 3
```

Interpréter le résultat après avoir appuer sur (a) Z (b) F1 (c) ALT.

(a) AX = 2C7Ah donc le code de recherche est 2Ch et le code ASCII est 7Ah.

(b) AX = 3B00h donc le code de recherche est 3B et le code « ASCII » est 0, car F1 n'est pas une touche ASCII.

(c) Rien ne se passe parce qu'il n'y a pas de code de recherche pour la touche ALT. Le statut des touches telles que ALT se trouve dans les octets de statut situés aux emplacements mémoire `40:17` et `40:18`.

Exemple 2.- Écrire un programme permettant de lire deux caractères au clavier et de les afficher dans l'ordre inverse.

#### 1.2.2.2 Déterminer si un caractère est présent : fonctions 01h et 11h

Introduction.- Ces fonctions vérifient si un caractère est présent dans le tampon du clavier. Le caractère n'est pas retiré du tampon s'il est présent, contrairement à ce qui se passe (évidemment) avec l'une des deux fonctions précédentes. Là encore, la différence entre les fonctions 01h et 11h est que la première ne concerne que les caractères du clavier originel à 83 touches.

Mise en place.- On place le numéro de fonction, ici 11h, dans le registre AH et on fait appel à l'interruption 16h. La routine de l'interruption vérifie s'il existe un caractère présent dans le tampon. S'il en est ainsi, la routine met à zéro l'indicateur de zéro et place le caractère dans le registre AL et son code de recherche dans AH. Sinon l'indicateur de zéro est mis à un et la routine n'attend pas.

L'intérêt de cette interruption est qu'elle n'est pas bloquante, contrairement à la précédente.

Exemple.-

#### 1.2.2.3 Lire les octets de statut : fonctions 02h et 12h

Mise en place.- La fonction 02h renvoie dans le registre AL l'octet de statut du clavier situé à l'emplacement 417h, la fonction 12h renvoie dans AL le même octet ainsi que l'octet situé à l'emplacement 418h dans le registre AH.

Exemple 1.- Exécuter le programme suivant avec `debug` avec la touche majuscule droite enfoncée et la LED de la touche CapsLock allumée :

```
MOV AH,02
INT 16H
INT 3
```

L'exécution donne AH = 41h = 0100 0001b, ce que l'on peut aussi vérifier avec `debug` en faisant :

```
-d 0:417 418
```

Exemple 2.- Testons si <Left Shift> (bit 1 de l'octet 417h) ou <Right Shift> (bit 0 du même octet) est pressé :

```
MOV   AH,02h          ; Requete de l'octet de statut
INT   16h             ; appel de l'interruption
OR    AL,00000011b    ; Majuscule gauche ou droite pressee ?
JE    xx              ; oui
```

#### 1.2.2.4 Écrire dans le tampon : fonction 05h

Introduction.- Cette opération permet à un programme d'insérer des caractères dans le tampon du clavier comme si l'utilisateur avait frappé quelque chose au clavier. On peut le faire tant que le tampon n'est pas plein.

Mise en place.- On place le caractère ASCII dans le registre CH et son code de recherche dans le registre CL.

## 1.3 L'affichage : l'interruption 10h

### 1.3.1 Notions sur l'affichage

L'affichage sur moniteur.- On a évidemment besoin d'un périphérique pour communiquer les résultats des opérations de l'ordinateur. Sur les premiers ordinateurs, il s'agissait de petites ampoules donnant le résultat en binaire, qu'il fallait ensuite décoder à la main. On a ensuite utilisé des imprimantes (dites rapides). Puis sont apparus les *terminaux* avec un affichage texte uniquement, pratiquement jusqu'à la fin des années 1970. Enfin est apparu l'*affichage graphique*, d'abord sous forme élémentaire, pour afficher, par exemple, le graphe d'une fonction. Puis sont apparus des affichages graphiques de plus en plus élaborés pour des dessins, disons genre bande dessinée, et enfin pour des photographies. L'utilisateur actuel connaît tous ces affichages, même l'écran d'invite est de nos jours graphique, à l'aide d'interface graphique comme Windows ou X-Window. Reste à programmer cet aspect graphique.

Mode texte.- Un écran en **mode texte** a d'abord permis, sur les premiers PC, d'afficher 25 lignes de 40 ou 80 caractères, c'est-à-dire 1 000 ou 2 000 caractères. Le comportement pour l'utilisateur est le suivant :

- on commence au coin supérieur gauche ;
- un **curseur** indique où sera affiché le prochain caractère ;
- lorsqu'on envoie un caractère **affichable**, celui-ci est affiché et le curseur se déplace d'une colonne ;
- lorsqu'on envoie un passage à la ligne, le curseur se positionne à la première colonne de ligne suivante ;
- lorsqu'on arrive à la fin de la ligne, soit le curseur se positionne à la première colonne de la ligne suivante, soit on perd les caractères (en envoyant un bip) ; le comportement dépend du système d'exploitation et non du BIOS ;
- lorsqu'on est arrivé à la dernière ligne et qu'on passe à la ligne suivante on a un effet de **défilement** (*scrolling* en anglais) : la première ligne n'est plus affichée, la seconde devient la première, la troisième la seconde, ... , la dernière la vingt-quatrième et le curseur se place au début de la dernière ligne.

Nécessité d'une carte graphique.- On peut donc penser que, du point de vue programmation, il suffit d'envoyer un caractère au port de l'écran. Ceci n'est cependant pas suffisant à cause d'une contrainte physique des écrans cathodiques d'alors.

Rappelons, sans entrer dans les détails, le principe de ceux-ci : un moniteur envoie, grâce à un canon à électrons, des électrons sur une couche de phosphore ; on peut diriger le faisceau de façon à ce qu'il décrive toutes les parties de l'écran ; on peut moduler son intensité de façon à ce qu'un point de l'écran s'éclaire ou non. Le phénomène de phosphorescence a une durée de vie très courte : ceci est à la fois un inconvénient, car il faut **rafraîchir** sans arrêt l'écran, et un avantage, car sinon comment effacer l'écran ou remplacer un caractère par un autre.

Nécessité d'un emplacement mémoire vidéo.- Une conséquence du besoin de rafraîchissement est qu'il faut un emplacement mémoire indiquant ce qui se trouve à l'écran. Cette mémoire peut se trouver où l'on veut, partie de la mémoire vive, partie du moniteur, mais elle est indispensable.

En supposant que l'on ait besoin d'un emplacement mémoire, un octet, par caractère, on voit que pour le cas du mode texte le plus simple il faut 1 kiO.

Mode graphique.- Dans le cas du **mode graphique**, l'unité n'est plus le caractère mais le **point écran** (*pixel* en anglais pour *PICTure ELement*, avec 'ct' devenant 'x' comme souvent en américain), qui est un point de l'écran, possédant des **attributs** (couleur, clignotement ou non...). On imagine bien que la mémoire nécessaire est plus importante.

D'autre part le mode graphique demande des interruptions supplémentaires pour afficher un point écran, choisir la couleur, déterminer où se trouve le curseur...

Pour ce dernier point il vaut mieux utiliser un *circuit intégré spécialisée*, que l'on peut changer lorsqu'on change de matériel.

Au début le circuit intégré et la mémoire graphique étaient placés sur la carte mère. Le défaut est qu'il faut alors changer la carte mère dès qu'on veut du matériel supérieur. On utilise donc plutôt une **carte graphique**, qui est une carte d'extension.

Les cartes graphiques d'origine.- La carte graphique livrée avec les différents modèles de PC a évoluée très rapidement avec les types successifs suivants :

`MDA` pour *Monochrome Display Adapter*,
`CGA` pour *Color Graphic Adapter*,
`EGA` pour *Enhanced Graphic Adapter*,
`MCGA` pour *MultiColor Graphics Array*, pour les IBM PS/2 modèles 25 et 30 uniquement,
`VGA` pour *Video Graphics Array*.

Ensuite les cartes graphiques n'ont plus été le fait d'IBM mais de diverses sociétés, connues sous le nom générique de cartes `SVGA`, pour *super VGA*, plus ou moins compatibles entre elles. On a cependant une compatibilité croissante MDA, CGA, EGA, VGA et SVGA. Vous avez toutes les chances de posséder une carte au moins SVGA, ce qui vous permettra d'émuler les modes que nous allons commenter.

Ne cherchez plus ces cartes, elles n'existent plus mais les cartes modernes les émulent, grâce à des **modes**. Ceci est très important pour démarrer, car ce sont les seules cartes dont on soit sûr et qui sont traitées par le BIOS.

## 1.3.2 Mode texte

### 1.3.2.1 Les différents modes texte

Comme nous l'avons déjà dit, les cartes graphiques, à part la carte MDA, permettent à la fois des modes texte et des modes graphiques. Nous n'allons nous intéresser qu'aux modes texte ici.

Numéro de mode.- Chaque mode texte porte un numéro, dont nous verrons l'intérêt après. Voici ces différents modes avec leurs caractéristiques :

| Mode | Taille | Type | Carte | Résolution | Couleurs |
|---|---|---|---|---|---|
| 00 | 25 lignes<br>40 colonnes | Mono | CGA<br>EGA<br>MCGA<br>VGA | 320 × 200<br>320 × 350<br>320 × 400<br>360 × 400 | |
| 01 | 25 lignes<br>40 colonnes | Couleur | CGA<br>EGA<br>MCGA<br>VGA | 320 × 200<br>320 × 350<br>320 × 400<br>360 × 400 | 16<br>16 parmi 64<br>16 parmi 262 144<br>16 parmi 262 144 |
| 02 | 25 lignes<br>80 colonnes | Mono | CGA<br>EGA<br>MCGA<br>VGA | 640 × 200<br>640 × 350<br>640 × 400<br>720 × 400 | |
| 03 | 25 lignes<br>80 colonnes | Couleur | CGA<br>EGA<br>MCGA<br>VGA | 640 × 200<br>640 × 350<br>640 × 400<br>640 × 400 | 16<br>16 parmi 64<br>16 parmi 262 144<br>16 parmi 262 144 |
| 07 | 25 lignes<br>80 colonnes | Mono | MDA<br>EGA<br>VGA | 720 × 350<br>720 × 350<br>720 × 400 | |

De nos jours on utilise évidemment de préférence le mode 3 pour la couleur et le mode 7 en monochrome.

Notion de page.- Dès les premiers modèles d'IBM PC il y avait suffisamment de mémoire pour conserver les données de plusieurs pages, ce qui permettait de passer d'une page à l'autre. Ceci est peu utilisé de nos jours, puisque le microprocesseur est suffisamment rapide.

Il y a quatre pages (numérotées de 0 à 3) en mode 80 colonnes et huit pages (numérotées de 0 à 7) en mode 40 colonnes.

#### 1.3.2.2 L'écran en mode texte

En mode texte, l'écran correspond en général à 25 lignes (en anglais *row*), numérotées de 0 à 14, et à 80 colonnes (en anglais *column*), numérotées de 0 à 79. Ceci donne une grille de 1000 (= 25 × 80) emplacements adressables, l'emplacement en cours étant visualisé par un **curseur**.

Voici quelques exemples d'emplacements du curseur :

| | Format décimal | | Format hexadécimal | |
|---|---|---|---|---|
| Emplacement écran | Ligne | Colonne | Ligne | Colonne |
| Coin supérieur gauche | 00 | 00 | 00h | 00h |
| Coin supérieur droit | 00 | 79 | 00h | 4Fh |
| Centre de l'écran | 12 | 39/40 | 0Ch | 27h/28h |
| Coin inférieur gauche | 24 | 00 | 18h | 00h |
| Coin inférieur droit | 24 | 79 | 18h | 4Fh |

#### 1.3.2.3 Choix du mode : fonction 00h

Mise en place.- Pour se placer dans un mode donné, tout au moins jusqu'au mode VGA, on se sert de la fonction 00h, dite fonction *Set Video Mode*, de l'interruption vidéo du BIOS, soit 10h. Le registre AH doit donc contenir 0 et le registre AL le numéro du mode.

Par exemple pour se placer dans le mode 3 on écrira :

```
mov ah, 0
mov al, 3
int 10h
```

Effacement ou non de l'écran.- Par défaut, lorsqu'on change de mode on efface l'écran. Si on veut que le contenu demeure, il faut que le huitième bit de AL soit positionné, donc il faut placer 83h et non 3h dans le registre AL.

Exemples.- Écrivons les programmes `mod3.asm`, `mod83.asm` suivants permettant de se placer dans le mode texte 3, en effaçant ou non (mode 83) l'écran, et en bloquant ce mode tant qu'on n'a pas appuyé sur une touche.

```
TITLE mod3.asm
      .model small
      .stack 256
      .code
EXTRN getche:proc
debut:
      mov ah, 0
      mov al, 3h
      int 10h
      call getche ; permet de bloquer le mode
                  ; jusqu'a l'appui d'une touche
      mov ax, 4c00h
      int 21h
      end debut
-----------------------------------------------
TITLE mod83.asm
      .model small
      .stack 256
      .code
EXTRN getche:proc
debut:
      mov ah, 0
      mov al, 83h
      int 10h
      call getche ; permet de bloquer le mode
                  ; jusqu'a l'appui d'une touche
      mov ax, 4c00h
      int 21h
      end debut
```

Remarque.- Lorsqu'on exécute ces programmes on remarque que l'on ne revient pas nécessairement dans le mode duquel on était parti.

#### 1.3.2.4 Conservation du mode : fonction 0Fh

Introduction.- La remarque après l'exécution des programmes ci-dessus nous montre l'intérêt de conserver quelque part l'ancien mode, pour pouvoir y revenir après coup.

Mise en place.- La fonction 0Fh de l'interruption 10h, dite *Get Video Mode*, permet de récupérer le mode en cours. On place 0Fh dans le registre AH, on fait appel à l'interruption 10h et on obtient le numéro du mode dans le registre AL, le nombre de caractères par ligne dans le registre AH (20, 40 ou 80, soit 14h, 28h ou 50h) et le numéro de page active dans le registre BH.

Application.- Cette fonction nous permet de revenir au mode antérieur. On peut réécrire notre programme `mod11.asm` de la façon suivante :

```
TITLE retour.asm
      .model small
      .stack 256
      .code
EXTRN getche:proc
debut:
      mov ah, 0Fh
      int 10h
      push ax
      push bx     ; conservation du mode
      mov ah, 0
      mov al, 11h
      int 10h
      call getche ; permet de bloquer le mode
                  ; jusqu'a l'appui d'une touche
      pop bx
      pop ax
      int 10h     ; revient a l'ancien mode
      mov ax, 4c00h
      int 21h
      end debut
```

#### 1.3.2.5 Affichage d'un caractère : fonction 0Ah

Description.- On utilise la fonction 0Ah de l'interruption 10h pour afficher un caractère, en mode texte ou en mode graphique. Le registre AL doit contenir le numéro du caractère à afficher, le registre BH le numéro de page 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 **semi-graphiques** suivant un code déterminé par IBM. Une conséquence en 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 de se placer dans le mode 7 et d'afficher 120 'a' à partir du haut de l'écran.

```
TITLE printa.asm
      .model small
      .stack 256
      .code
debut:
      mov ah, 0
      mov al, 7h
      int 10h         ; mode 7
      mov ah, 0Ah
      mov al, 'a'
      mov bh, 0
      mov cx, 120
      int 10h         ; affiche 120 'a'
      mov ax, 4c00h
      int 21h
      end debut
```

#### 1.3.2.6 Positionnement du curseur : fonction 02h

Introduction.- Positionner le curseur est important en mode texte, puisque la position du curseur détermine l'endroit où le prochain caractère sera affiché.

Mise en place.- On utilise la fonction 02h de l'interruption 10h en plaçant le numéro de page, normalement 0, dans le registre BH, le numéro de ligne dans le registre DH et le numéro de colonne dans le registre DL.

Exemple.- La portion de programme suivant place le curseur à la huitième ligne et quinzième colonne :

```
MOV   AH,02h    ; positionnement du curseur
MOV   BH,00h    ; page numero 0
MOV   DH,08     ; ligne 8
MOV   DL,15     ; colonne 15
INT   10h       ; appel de l'interruption
```

Bien entendu on peut remplacer les deux lignes concernant le registre `DX` par une seule :

```
MOV   AH,02h    ; positionnement du curseur
MOV   BH,00h    ; page numero 0
MOV   DX,080Fh  ; ligne 8 et colonne 15
INT   10h       ; appel de l'interruption
```

#### 1.3.2.7 Définir la taille du curseur : fonction 01h

Introduction.- Le curseur n'est pas un caractère ASCII. Il existe seulement en mode texte. Le curseur par défaut ressemble à un blanc souligné, mais on peut adapter sa forme en utilisant la fonction 01h de l'interruption 10h.

Mise en place.- On peut indiquer la ligne inférieure et la ligne supérieure du curseur dans l'amplitude `0:14` en VGA, `0:13` en monochrome et en EGA et `0:7` en CGA. Pour cela on utilise les registres suivants :

- CH (bits 4 à 0) pour la ligne supérieure ;
- CL (bits 4 à 0) pour la ligne inférieure.

Exemple.- Le programme suivant fait apparaître le curseur comme un rectangle de taille maximale, clignotant, en mode VGA :

```
MOV  AH,01h  ; fonction taille du curseur
MOV  CH,00   ; ligne superieure
MOV  CL,14   ; ligne inferieure
INT  10h     ; appel de l'interruption
```

Valeurs par défaut.- Les valeurs par défaut pour le curseur sont `0:14` en VGA, `12:13` en monochrome et en EGA et `6:7` en CGA.

#### 1.3.2.8 Lire la position du curseur : fonction 03h

Introduction.- Il est important de récupérer la position du curseur, en particulier lorsqu'on change de page pour un retour ultérieur.

Mise en place.- On utilise la fonction 03h de l'interruption 10h, en plaçant le numéro de page dans le registre BH. La fonction renvoie les valeurs suivantes :

- AX non changé ;
- BX non changé ;
- CH ligne de début du curseur ;
- CL ligne de fin du curseur ;
- DH ligne ;
- DL colonne.

Exemple.- Déterminons la position du curseur et plaçons-le à la position suivante à l'écran :

```
MOV  AH,03h       ; position du curseur ?
MOV  BH,00        ; de la page 0
INT  10h          ; appel de l'interruption
MOV  AH,02h       ; placer le curseur
INC  DL           ; colonne suivante
INT  10h          ; appel de l'interruption
```

#### 1.3.2.9 Sélectionner la page active : fonction 05h

Introduction.- Sélectionner la page active n'a évidemment de sens que pour les modes texte, à savoir 0 à 3 et 13 à 16. On peut choisir entre l'une des quatre pages 0 à 3.

Mise en place.- On utilise la fonction 05h de l'interruption 10h en placant le numéro de page dans le registre AL. Il n'y a pas de valeur de retour.

Exemple.- Pour sélectionner la page numéro 2, on utilise :

```
MOV   AH,05h      ; page active
MOV   AL,2        ; numero 2
INT   10h         ; appel de l'interruption
```

#### 1.3.2.10 Effacer et faire défiler l'écran : fonction 06h

Introduction.- La fonction 06h de l'interruption 10h permet d'effacer (en anglais *to clear*) ou de faire défiler l'écran. On peut tout effacer ou effacer en partie.

Mise en place.- Il faut d'abord charger les registres de la façon suivante :

- AH avec le numéro de fonction, soit 06h ;
- AL avec le nombre de lignes à faire défiler, ou 00h pour l'écran en entier (et donc l'effacer) ;
- BH avec l'attribut (couleur, vidéo inverse, clignotement) ;
- CX avec les numéros de ligne et de colonne de début ;
- DX avec les numéros de ligne et de colonne de fin.

Pour qu'il s'agisse de tout l'écran, il faut que CX soit égal à `00:00h` et DX à `18:4Fh`.

Exemple 1.- Pour que tout l'écran apparaisse avec un fond blanc (attribut 7h) et un avant-plan bleu (attribut 1h), on utilise le programme suivant :

```
MOV  AX,0600h   ; AH = 06h (defilement), AL = 00h (tout l'ecran)
MOV  BH,71h     ; fond blanc (7), avant-plan bleu (1)
MOV  CX,0000h   ; coin haut a gauche
MOV  DX,184Fh   ; coin en bas a droite
INT  10h        ; appel de l'interruption
```

Exemple 2.- Le programme suivant crée une fenêtre (possédant ses attributs propres) de 7 lignes et 30 colonnes, le coin haut à gauche étant `12:25` (et donc les autres coins étant `12:54`, `18:25` et `18:54`) :

```
MOV  AX,0607h   ; AH = 06h (defilement), AL = 07h (7 lignes)
MOV  BH,30h     ; fond cyan (3), avant-plan noir (0)
MOV  CX,0C19h   ; coin haut a gauche
MOV  DX,1236h   ; coin en bas a droite
INT  10h        ; appel de l'interruption
```

#### 1.3.2.11 Défilement vers le bas : fonction 07h

C'est la même chose que le défilement habituel, c'est-à-dire vers le haut (chaque ligne prenant la place de la ligne au-dessus). Ici chaque ligne (sauf la dernière se place à la ligne en-dessous) et il apparaît une ligne blanche en haut de l'écran.

Les paramètres sont les mêmes que dans le cas précédent (à part le numéro de fonction).

### 1.3.3 Utilisation des attributs

Jusqu'à maintenant nous avons peu parlé des *attributs* des caractères. Voyons comment les utiliser.

#### 1.3.3.1 Octet d'attribut

Définition.- Nous avons déjà dit que dans la mémoire graphique, chaque caractère (en mode texte) occupe deux octets : le numéro du caractère et un *octet d'attribut*.

Format.- Le format de l'octet d'attribut est le suivant :

```
             | Fond    |   |   Avant
            ---------------------------
Attribut : BL | R  G  B | I | R  G  B
Bit      : 7  | 6  5  4 | 3 | 2  1  0
            ---------------------------
```

Les lettres R, G et B indiquent le positionnement du bit pour les couleurs rouge, vert et bleu respectivement. Le bit 7 concerne le clignotement (`BL` pour *BLinking*). Les bits 6 à 4 concernent le fond, ou arrière-plan (*background* en anglais). Le bit 3 concerne une intensité renforcée. Les bits 2 à 0 concernent l'avant-plan (*foreground* en anglais), c'est-à-dire les caractères eux-mêmes.

Valeur par défaut.- La valeur par défaut, que ce soit en monochrome ou en couleur, est `0000 0111`, c'est-à-dire un fonc noir avec des caractères blancs.

#### 1.3.3.2 Moniteur couleur

Le fond a le choix entre 8 couleurs et l'avant-plan entre 16 couleurs. Le clignotement et l'intensenté renforcée ne s'appliquent qu'à l'avant-plan. On peut aussi choisir entre 16 couleurs pour le bord.

On peut combiner les trois couleurs primaires, ainsi que l'intensité renforcée pour l'avant-plan, pour obtenir une des couleurs suivantes :

| Couleur | I | R | G | B |
|---|---|---|---|---|
| Noir | 0 | 0 | 0 | 0 |
| Bleu | 0 | 0 | 0 | 1 |
| Vert | 0 | 0 | 1 | 0 |
| Cyan | 0 | 0 | 1 | 1 |
| Rouge | 0 | 1 | 0 | 0 |
| Magenta | 0 | 1 | 0 | 1 |
| Brun | 0 | 1 | 1 | 0 |
| Blanc | 0 | 1 | 1 | 1 |
| Gris | 1 | 0 | 0 | 0 |
| Bleu lumineux | 1 | 0 | 0 | 1 |
| Vert lumineux | 1 | 0 | 1 | 0 |
| Cyan lumineux | 1 | 0 | 1 | 1 |
| Rouge lumineux | 1 | 1 | 0 | 0 |
| Magenta lumineux | 1 | 1 | 0 | 1 |
| Jaune | 1 | 1 | 1 | 0 |
| Blanc intense | 1 | 1 | 1 | 1 |

Si le fond et l'avant-plan sont de la même couleur, les caractères ne seront évidemment pas visibles.

Exemples.- Les valeurs suivantes sont souvent utilisées :

| Fond | Avant | Fond | | | | Avant | | | | Hex |
|---|---|---|---|---|---|---|---|---|---|---|
| | | BL | R | G | B | I | R | G | B | |
| Noir | Noir | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| Noir | Bleu | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| Bleu | Rouge | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 14 |
| Vert | Cyan | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 23 |
| Blanc | Magenta lumineux | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 7D |
| Vert | Gris clignotant | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | A8 |

#### 1.3.3.3 Moniteur monochrome

Pour un moniteur monochrome, l'octet d'attribut est utilisé de la même façon sauf que le bit 0 concerne le soulignement. Les valeurs les plus utilisées sont les suivantes :

| Fond | Avant | Effet | Fond | | | | Avant | | | | Hex |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | BL | R | G | B | I | R | G | B | |
| Noir | Noir | Rien | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| Noir | Blanc | Normal | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 07 |
| Noir | Blanc | Clignotant | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 87 |
| Noir | Blanc | Intense | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0F |
| Blanc | Noir | Inverse | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 70 |
| Blanc | Noir | Inverse clignotant | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | F0 |
| | | souligné | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |

#### 1.3.3.4 Positionner l'attribut : fonction 09h

Mise en place.- Une fois l'attribut positionné, il le reste jusqu'à ce qu'une autre opération le change. On utilise la fonction 09h de l'interruption 10h pour afficher un certain nombre de caractères, à la fois en mode texte et en mode graphique, en indiquant l'octet d'attribut. Pour cela on place :

- dans le registre AH le numéro de fonction (09h) ;
- dans le registre AL le caractère à afficher ;
- dans le registre BL l'octet d'attribut ;
- dans le registre BH le numéro de la page ;
- dans le registre CX le nombre de fois que le caractère doit être affiché.

Exemple.- Faisons apparaître 12 astérisques bruns, clignotant (1110h) sur un fond bleu (0001) :

```
MOV   AH,09h     ; affichage
MOV   AL,'*'     ; asterisque
MOV   BH,00h     ; page 0
MOV   BL,1Eh     ; attribut
MOV   CX,12      ; 12 caracteres
INT   10h        ; appel de l'interruption
```

Remarques.- 1$^o$) On n'utilise pas le code ASCII mais un code créé par IBM, qui en diffère pour les 32 premiers caractères. On ne peut donc pas aller à la ligne ou émettre un bip, par exemple.

- 2$^o$) Le curseur n'est pas avancé automatiquement. Il faut donc utiliser la fonction 02h pour avancer celui-ci.

- 3$^o$) En mode texte, mais pas en mode graphique, lorsqu'on arrive en fin de ligne on passe à la ligne suivante.

#### 1.3.3.5 Lire un attribut : fonction 08h

Mise en place.- On peut lire le caractère et son octet d'attribut situé à la position du curseur, en mode texte ou graphique, grâce à la fonction 08h. Le numéro de page doit se trouver dans le registre BH. Le caractère est envoyé dans le registre AL et son attribut dans le registre AH.

En mode graphique, AL contiendra 00h s'il ne s'agit pas d'un caractère ASCII.

Exemple.-

```
MOV  AH,08h   ; lire caractere/attribut
MOV  BH,00h   ; page 0
INT  10h      ; appel de l'interruption
```

#### 1.3.3.6 Affichage sans attribut : fonction 0Ah

La seule différence entre les fonctions 09h et 0Ah, en mode texte, est que la fonction 0Ah utilise l'attribut en cours. C'est donc la fonction d'affichage que l'on utilise en général. Il faut toujours déplacer le curseur soi-même.

#### 1.3.3.7 Écriture télétype : fonction 0Eh

Introduction.- L'*écriture télétype* permet d'utiliser le moniteur comme un terminal, c'est-à-dire que le curseur est déplacé, il y a défilement automatique en fin d'écran et on peut utiliser les 32 premiers caractères ASCII pour le formattage (mais, bien entendu, on ne peut pas alors utiliser les caractères graphiques d'IBM).

Mise en place.- Il faut placer le numéro de fonction, 0Eh, dans le registre AH, le code ASCII du caractère à afficher dans dans le registre AL, le numéro de page dans le registre BH et la couleur d'avant-plan dans le registre BL, ce dernier dans le cas d'un mode graphique, puis faire appel à l'interruption 10h.

Exemple.-

```
MOV  AH,0Eh   ; ecriture teletype
MOV  AL, 'a'  ; affichage de 'a'
MOV  BH,00h   ; page 0
INT  10h      ; appel de l'interruption
```

## 1.3.4 Mode graphique

### 1.3.4.1 Les modes graphiques

Les divers modes graphiques.- Les modes graphiques jusqu'à la carte VGA sont les suivants :

| Mode | Type | Carte | Résolution | Couleurs |
|---|---|---|---|---|
| 04h | Couleur | CGA, EGA, MCGA, VGA | 320 × 200 | 4 |
| 05h | Mono | CGA, EGA, MCGA, VGA | 320 × 200 | |
| 06h | Mono | CGA, EGA, MCGA, VGA | 640 × 200 | |
| 0Dh | Couleur | EGA, VGA | 320 × 200 | 16 |
| 0Eh | Couleur | EGA, VGA | 640 × 200 | 16 |
| 0Fh | Mono | EGA, VGA | 640 × 350 | |
| 10h | Couleur | EGA, VGA | 640 × 350 | 16 |
| 11h | Couleur | MCGA, VGA | 640 × 480 | 2 parmi 262 144 |
| 12h | Couleur | VGA | 640 × 480 | 16 parmi 262 144 |
| 13h | Couleur | MCGA, VGA | 320 × 200 | 256 parmi 262 144 |

Les modes 04h, 05h et 06h sont les modes originels de la carte CGA. Ils sont également présents sur les cartes EGA et VGA pour des raisons de compatibilité ascendante. L'adresse de la mémoire graphique pour ces modes est B800[0].

Les modes 0Dh, 0Eh, 0Fh et 10h sont les modes originels de la carte EGA. Ils sont également présents sur les cartes VGA pour des raisons de compatibilité ascendante. L'adresse de la mémoire graphique pour ces modes est A000[0]. Ces modes supportent respectivement 8, 4, 2 et 2 pages graphiques, la page par défaut étant la page 0.

Les modes 11h, 12h et 13h sont les modes originels de la carte VGA (et de la carte MCGA). L'adresse de la mémoire graphique pour ces modes est A000[0].

Pas de curseur.- Il n'y a pas de curseur en mode graphique.

Caractères en mode graphique.- En mode graphique, il existe des patrons de points en mémoire ROM pour les 128 premiers caractères (les 32 premiers caractères étant les caractères semi-graphiques d'IBM).

L'**interruption 1Fh**, dont nous n'avons pas parlée auparavant, permet d'accéder à 1 KiO de mémoire vive pour définir les 128 caractères suivants, à raison de 8 octets par caractère.

Points écran.- Les modes graphiques utilisent les *points écran* pour dessiner. Par exemple, en mode 04h, on a 200 lignes de 320 points écran. Chaque octet de la zone mémoire graphique représente 4 points écran (c'est-à-dire 2 bits par point écran), numérotés de 0 à 3 de la façon suivante :

```
Octet : C1   C0   C1   C0   C1   C0   C1   C0
Pixel :    0         1         2         3
```

Il y a à tout moment quatre couleurs disponibles, numérotées de 0 à 3. La limitation à quatre couleurs est évidemment dû au fait qu'il y a deux bits par pixel. On peut choisir la couleur de fond 00b parmi les 16 couleurs suivantes :

| Couleur | Description |
|---|---|
| Noir | 0000 |
| Bleu | 0001 |
| Vert | 0010 |
| Cyan | 0011 |
| Rouge | 0100 |
| Magenta | 0101 |
| Brun | 0110 |
| Gris lumineux | 0111 |
| Gris | 1000 |
| Bleu lumineux | 1001 |
| Vert lumineux | 1010 |
| Cyan lumineux | 1011 |
| Rouge lumineux | 1100 |
| Jaune | 1110 |
| Blanc | 1111 |

On peut choisir les couleurs de premier plan 01b, 10b et 11b parmi les deux *palettes* suivantes :

| C1 | C0 | Palette 0 | Palette 1 |
|---|---|---|---|
| 0 | 0 | Fond | Fond |
| 0 | 1 | Vert | Cyan |
| 1 | 0 | Rouge | Magenta |
| 1 | 1 | Brun | Blanc |

Nous verrons que l'on utilise la **fonction 0Bh** pour choisir la couleur de fond et la palette.

#### 1.3.4.2 Choisir le mode : fonction 00h

Mise en place.- Nous avons déjà vu, à propos des modes texte, que pour se placer dans un mode donné, tout au moins jusqu'au mode VGA, on se sert de la fonction 00h, dite fonction *Set Video Mode*, de l'interruption vidéo du BIOS, à savoir 10h. Le registre AH doit donc contenir 0 et le registre AL le numéro du mode.

Par exemple pour se placer dans le mode 12h, le mode VGA standard, on fera :

```
MOV  AH,00h  ; Set Video Mode
MOV  AL,12h  ; VGA resolution 640x480
INT  10h     ; Appel de l'interruption
```

Remarque.- Attention ! Il n'y a pas d'écho en écriture par défaut en mode graphique. Il faut donc toujours revenir à un mode texte en fin d'utilisation d'un mode graphique, car « écrire en aveugle » n'est pas très pratique, et c'est pire si on n'est pas sûr du clavier émulé (*qwerty* ou *azerty*?).

Exemple.- Écrivons le programme `mod12.asm` permettant de se placer dans le mode graphique 12 et en bloquant ce mode tant qu'on n'a pas appuyé sur une touche, sans oublier de revenir à un mode texte en quittant le mode graphique.

```
TITLE mod12.asm
      .model small
      .stack 256
      .code
EXTRN getche:proc
debut:
      mov ah, 0
      mov al, 12h
      int 10h
      call getche ; permet de bloquer le mode
                  ; jusqu'a l'appui d'une touche
      mov al, 3h  ; on revient a un mode texte
      int 10h
      mov ax, 4c00h
      int 21h
      end debut
```

#### 1.3.4.3 Afficher un point écran : fonction 0Ch

Description de la fonction.- Pour afficher un point écran dans un mode graphique, on utilise la fonction 0Ch de l'interruption 10h. Par référence au texte, s'écrivant à partir du haut à gauche, l'origine des coordonnées est le coin nord-ouest ou, ce qui revient au même, supérieur gauche) de l'écran ; l'abscisse est décrite de gauche à droite, en points écran, en partant de 0 (et allant jusqu'au nombre de points écran par ligne moins une) ; l'ordonnée, c'est-à-dire le numéro de ligne, est décrite de haut en bas (contrairement à l'habitude mathématique), en partant de 0 (et allant jusqu'au nombre de lignes par écran moins une).

La valeur de l'abscisse se place dans le registre CX, la valeur de l'ordonnée dans le registre DX. Le registre AL doit contenir la couleur ; dans le cas monochrome il s'agit des valeurs 0 et 1, pour noir et blanc respectivement. Le registre BH contient éventuellement le numéro de page.

Exemple.- Écrivons un programme nous plaçant dans le mode graphique 11h, dessinant un segment de droite puis revenant à un mode texte lorsqu'on appuie sur une touche.

```
TITLE segment.asm
; se place en mode graphique et dessine un segment de droite
         .model small
         .stack 256
         .code
debut:
          mov ah, 00h
          mov al, 11h
          int 10h                ; mode 640x480 2 couleurs

          mov al, 1h             ; les points ecran dessines en blanc
          mov bh, 00h            ; premiere page graphique
          mov dx, 100h           ; segment a la 256ieme ligne
          mov cx, 50h            ; debut a la 80ieme colonne
          mov ah, 0Ch            ; ecrire un point ecran
 suite :
          int 10h
          inc cx
          cmp cx, 150h           ; jusqu'a la 240ieme colonne
          jne suite

          mov ah,1h              ; attend un caractere
          int 21h                ; le temps de voir

          mov al, 03h            ; revient en mode texte
          mov ah, 00h
          int 10h

          mov ax,4c00h
          int 21h

          end debut
```

#### 1.3.4.4 Choisir la palette : fonction 0Bh

Mise en place.- Dans le cas d'un mode graphique couleur, il faut choisir la palette. On utilise pour cela la fonction 0Bh de l'interruption 10h. La valeur du registre BH détermine la nature du registre BL :

- Lorsque BH = 00h, on choisit la couleur du fond. Le registre BL doit alors contenir l'une des seize couleurs possibles (dans les bits 0 à 3).
- Lorsque BH = 01h, on choisit la palette. Le registre BL doit alors contenir le numéro de palette, 0 ou 1.

Une fois la palette choisie, elle reste valable jusqu'à ce qu'on la change à nouveau. La palette est valable pour l'écran en son entier, y compris pour les points déjà dessinés.

Dans le cas d'un mode texte, la valeur pour BH = 00h concerne le bord.

Exemple.- Écrivons un programme nous plaçant dans le mode graphique 12h, affichant un point ecran puis revenant à un mode texte lorsqu'on appuie sur une touche.

```
TITLE couleur.asm
; se place en mode graphique et dessine un point ecran en couleur
         .model small
         .stack 256
         .code
debut:
          mov ah, 00h
          mov al, 12h
          int 10h                ; mode 640x480 16 couleurs

          mov ah,0Bh             ; choix de la couleur
          mov bh,00h             ; de fond
          mov bl,04h             ; rouge
          int 10h                ; appel de l'interruption

          mov ah,0Bh             ; choix de la palette
          mov bh,01h             ; de couleur
          mov bl,01h             ; vert, rouge, brun
          int 10h                ; appel de l'interruption

          mov ah,0Ch             ; affichage d'un point ecran
          mov al,03h             ; en brun
          mov bh, 00h            ; premiere page graphique
          mov dx, 100h           ; segment a la 256ieme ligne
          mov cx, 50h            ; a la 80ieme colonne
          int 10h                ; appel de l'interruption

          mov ah,1h              ; attend un caractere
          int 21h                ; le temps de voir

          mov al, 03h            ; revient en mode texte
          mov ah, 00h
          int 10h

          mov ax,4c00h
          int 21h

          end debut
```

### 1.3.5 Court-circuiter le BIOS

Introduction.- Nous avons vu que l'écran est rafraîchi en permanence, les données étant placées en *mémoire graphique*. Pour afficher, on peut donc court-circuiter les fonctions du BIOS en allant placer directement ce qu'il faut afficher à l'emplacement mémoire adéquat.

Ceci n'est pas nécessairement une bonne méthode de programmation, en particulier parce que l'adresse de la mémoire graphique dépend de la carte graphique. On conseille donc, en général, d'utiliser l'interruption 10h du BIOS ou l'interruption 21h du DOS.

Mémoire graphique.- Pour un moniteur monochrome, la mémoire graphique commence à l'emplacement B000[0]h, avec une capacité de 4 KiO, plus précisément 2 KiO pour les caractères et 2 KiO pour les attributs (clignotement, soulignement...). Pour un moniteur couleur de base (pour les cartes graphiques jusqu'à la carte VGA), la mémoire graphique commence à l'emplacement B800[0]h, avec une capacité de 16 KiO.

#### 1.3.5.1 Affichage de texte monochrome

Principe.- On peut utiliser les interruptions du DOS ou du BIOS pour afficher du texte, comme nous l'avons déjà vu. Mais ceci n'est pas très intéressant, car lent et avec très peu de fonctionnalités.

La meilleure façon est donc d'accéder directement à la mémoire graphique. C'est suffisant, comme nous le verrons, une fois le coupleur 6845 initialisé (ce que l'on peut faire grâce à une interruption du BIOS).

Problème.- Écrivons un programme demandant un chiffre, compris entre 1 et 8 (représentant un ensemble d'attributs), et affichant alors les 256 caractères avec ces attributs sur les premières lignes de l'écran en mode 3. Le programme s'arrête lorsqu'on entre autre chose que l'un de ces chiffres.

Les attributs.- En monochrome, les trois bits du fond et de l'avant-plan ne peuvent prendre que deux valeurs :

`000` pour le noir
`111` pour le blanc (ou la couleur verte ou ambre du moniteur).

On a donc les huit valeurs suivantes possibles d'attributs pour que quelque chose soit visible (noir sur noir ne donne rien) :

`0 000 0 111b = 07h` : normal, blanc sur noir.
`0 000 1 111b = 0Fh` : intense.
`1 000 0 111b = 87h` : clignotant.
`1 000 1 111b = 8Fh` : intense et clignotant.
`0 111 0 000b = 70h` : inverse, noir sur blanc.
`0 111 1 000b = 78h` : inverse intense.
`1 111 0 000b = 0F0h` : inverse clignotant.
`1 111 1 000b = 0F8h` : inverse, intense et clignotant.

Effacer l'écran.- Pour effacer l'écran, il suffit d'afficher l'espace (en normal) pour les 2000 emplacements.

Le programme.- On peut utiliser le programme suivant en langage d'assemblage :

```
; mono.asm

DATA        SEGMENT
ATTRIBUTS           EQU THIS BYTE
NORMAL               DB  07h
INVERSE              DB  70h
INTENSE              DB  0Fh
CLIGNOTE             DB  87h
INTENSE_ET_CLI       DB  8Fh
INVERSE_ET_INT       DB  78h
INVERSE_ET_CLI       DB  0F0h
INV_ET_INT_ET_CLI    DB  0F8h
DATA        ENDS

PILE        SEGMENT STACK
            DW 100h DUP(?)
PILE        ENDS

CODE        SEGMENT
ASSUME CS:CODE, DS:DATA, SS:PILE
;
; adresse du segment de donnees
;
START:
            MOV  AX,DATA
            MOV  DS,AX
;
; ES pointe sur la memoire graphique
;
            MOV  AX,0B800h
            MOV  ES,AX
;
; passage au mode 3
;
            MOV AH,0
            MOV AL,3
            INT 10h
;
; efface l'ecran
;
            STD
            MOV  DI,0
            MOV  AL,' '
            MOV  AH,NORMAL
            MOV  CX,2000d
            REP  STOSW
;
; lit (ch) une touche sans echo
```

```
;
            MOV  AH,0
            INT  16h
;
; tant que ch dans l'intervalle '1' .. '8'
;   (soit 31h .. 39h) faire
;
IN_RANGE:
            CMP  AL,31h
            JC   DONE
            CMP  AL,39h
            JNC  DONE
;
; debut
; placer l'attribut dans AH
;
            SUB  AL,31h
            MOV  BL,AL
            MOV  BH,0h
            MOV  AH,ATTRIBUTES[BX]
;
; DI := debut de la ligne 1
; affiche le code caractere (AL) := 0
;
            MOV  DI,(1*160d)+(0*2)
            MOV  AL,0
;
; colonne count(DH) := 0
;
            MOV  DH,0
;
; repete le caractere a afficher avec code dans AL,
;             attribut dans AH
;
NEXT_CHARACTER:
            MOV  ES:[DI]+1,AH
            MOV  ES:[DI],AL
;
; avance d'une colonne avant le caractere suivant
;
            ADD  DI,4
            ADD  DH,2
;
; sommes-nous au-dela de la colonne 79 ?
;
            CMP  DH,79d
            JB   NEXT_LINE
;
; alors debut
; aller a la ligne
```

```
;
            ADD  DI,160
;
; reinitialiser le compteur de colonne
;
            MOV  DH,0
;
; fin
; code caractere suivant
;
NEXT_LINE:
            INC  AL
;
; jusqu'a ce que le code dans AL soit 256h
; (egal a 0 pour un octet)
;
            CMP  AL,0h
            JZ   NEXT_ATTRIBUTE
            JMP  NEXT_CHARACTER
;
; lit (ch) sans echo a l'ecran
;
NEXT_ATTRIBUTE:
            MOV  AH,0
            INT  16h
            JMP  IN_RANGE
;
; efface l'ecran
;
DONE:
            MOV  DI,0
            MOV  AL,' '
            MOV  AH,NORMAL
            MOV  CX,2000d
            REP  STOSW
;
; retour au DOS
;
            MOV  AX,4C00h
            INT 21h
;
CODE    ENDS
END     START
```

## 1.4 Les disques : les interruptions 0Eh et 13h

Il est important d'avoir accès aux disques, c'est-à-dire au disque dur et aux disquettes, grâce au BIOS puisque le système d'exploitation sera recherché sur ceux-ci.

### 1.4.1 Organisation des disques

Commençons par donner un modèle de l'organisation d'un disque.

#### 1.4.1.1 Description physique

Modèle.- La partie importante d'une **disquette** (en anglais *diskette* ou *floppy disk*) est un disque, en fait une couronne circulaire, dont les deux **faces** (en anglais *side* ou *surface*) sont utilisées. Un **disque dur** (en anglais *hard disk*) est un ensemble de disques de même axe, dont les deux faces sont également utilisées.

Piste.- Chaque face contient un certain nombre de **pistes** (en anglais *track*), qui sont des cercles de même axe que le disque. Ces pistes sont d'une certaine façon imaginaires, elles ne sont pas matérialisées. Les pistes sont numérotées à partir de 0, qui correspond à la piste la plus externe.

Secteur.- Chaque piste est divisée en un certain nombre de **secteurs** (en anglais *sector*) correspondant à un angle donné. Cet angle est $2\pi/n$, où $n$ est un entier naturel non nul ; une piste a une longueur d'autant plus courte qu'elle est plus proche du centre. Là encore un secteur n'est pas matérialisé. Un secteur contient un nombre fixé d'octets, le plus souvent 512. C'est la plus petite unité d'information accessible physiquement sur un disque.

Cylindre.- Un **cylindre** (en anglais *cylinder*) d'un disque est l'ensemble des pistes de même numéro. Un cylindre d'une disquette contient deux pistes et sur un disque dur $2 \times p$ pistes, où $p$ est le nombre de disques.

#### 1.4.1.2 Contrôleur de disque

Le **contrôleur de disque** (en anglais *disk drive controller*) est l'interface entre le microprocesseur et le disque. Le contrôleur reçoit, par exemple, de l'information pour lire des données sur un secteur donné ; son rôle est de placer la tête de lecture-écriture au bon endroit, de lire l'information et de la renvoyer au microprocesseur.

Le contrôleur est piloté par des interruptions BIOS spécifiée par les concepteurs de l'IBM-PC et implémentées par les concepteurs du disque ou du lecteur de disquette.

#### 1.4.1.3 Lecture du contenu d'un disque avec debug

On peut lire le contenu d'un disque en le reportant dans la mémoire centrale.

Syntaxe.- Par exemple pour charger les 32 premiers secteurs de la disquette A à l'emplacement habituel 100h, on fait :

```
-L 100 0 0 20
```

La syntaxe se comprend bien sur cet exemple :

- `L` est la commande de `debug` pour charger un fichier, comme nous l'avons déjà vu, ou des secteurs d'un disque, comme nous le voyons, en mémoire centrale ;
- le premier paramètre correspond à l'adresse en hexadécimal, il s'agit ici de CS :100, ce qui est de toute façon l'adresse par défaut ;
- le deuxième paramètre est le numéro du disque, avec 0 pour A, 1 pour B, 2 pour C, ...
- le troisième paramètre est le numéro du premier secteur (relatif) à charger, en hexadécimal ; ici on veut lire à partir du secteur 0, c'est-à-dire au tout début du disque ;
- le dernier paramètre est le nombre de secteurs consécutifs à charger, en hexadécimal.

Bien entendu on affiche ensuite le contenu chargé grâce à une commande `dump`.

Exemple.-

### 1.4.2 Lecture et écriture logique des secteurs

Les instructions de plus bas niveau concernant un disque sont la lecture et l'écriture d'un secteur. Il faut, pour cela, avec le BIOS, indiquer trois paramètres : la piste, la face et le secteur.

Remarque.- L'utilisation de cette interruption doit être réservée aux programmeurs expérimentés, puisqu'une mauvaise manœuvre risque de ne plus permettre au disque d'être manipulé avec MS-DOS : il faut alors, éventuellement, le reformater. On n'utilisera donc que des disquettes d'exemple, sans rien d'important dessus, pour l'écriture et uniquement la lecture pour le disque dur.

#### 1.4.2.1 Lecture de secteurs : fonction 02h

Description.- Pour lire un secteur, ou plusieurs secteurs situés sur la *même* piste, on utilise la fonction 02h de l'interruption 13h. Le registre AL doit contenir le nombre de secteurs à lire. Le registre CH doit contenir le numéro de cylindre (le premier cylindre étant numéroté 0), le registre CL contient éventuellement les deux bits forts (bits 6 et 7) du numéro de cylindre et surtout (bits 0 à 5) le numéro du premier secteur à lire (le numéro du premier secteur étant 1). Le registre DH doit contenir le numéro de la tête de lecture-écriture (0 ou 1 pour une disquette). Le registre DL doit contenir le numéro du disque (0 = A, 1 = B, 2 = C... mais aussi 80h pour le premier disque dur, 81h pour le second...). Les registres `ES:BX` doivent contenir l'adresse d'un tampon suffisant pour contenir la valeur des secteurs à lire.

Si l'opération réussit, l'indicateur CF est mis à 0 et le registre AL contient le nombre effectif de secteurs lus. Sinon CF est mis à 1.

Exemple.- Écrivons un programme permettant de lire le premier secteur de la disquette située dans le disque A :

```
TITLE litsect.asm
; lit le premier secteur du disque A
      .model small
      .stack 256
      .data
tampon db 512 dup (?)
message db 'erreur de lecture', '$'
      .code
debut:
      mov ax, @data
      mov ds, ax
      mov es, ax            ; pour le tampon

      mov ah, 2             ; lire secteur
      mov al, 1             ; un secteur
      mov bx, offset tampon ; pour entreposer
      mov ch, 0             ; cylindre 0
      mov cl, 1             ; secteur 1
      mov dh, 0             ; tete 0
      mov dl, 0             ; disque A
      int 13h               ; lecture du secteur
      jc erreurs            ; en cas d'erreur
      jmp affiche
```

```
erreurs:
      mov dx, offset message
      mov ah, 9h
      int 21h               ; affiche erreur de lecture
      jmp fin
affiche:
      mov ah, 2h      ; affichage d'un caractere
      mov cl, 16      ; 16 lignes
nouvelle:
      mov dl, 13
      int 21h
      mov dl, 10
      int 21h
      mov ch, 32      ; 32 caracteres
ligne:
      mov dl, [bx]
      int 21h
      inc bx          ; caractere suivant
      dec ch
      cmp ch, 0
      jne ligne
      dec cl
      cmp cl, 0
      jne nouvelle
fin:
      mov ax, 4c00h
      int 21h
      end debut
```

#### 1.4.2.2 Écriture de secteurs : fonction 03h

Description.- Pour écrire un secteur, ou plusieurs secteurs situés sur la *même piste*, d'un disque préalablement formaté, on utilise la fonction 03h de l'interruption 13h. Les paramètres sont les mêmes que pour lire, le tampon correspondant bien sûr ici à ce qu'il faut écrire.

Si l'opération réussit, l'indicateur CF est mis à 0 et le registre AL contient le nombre effectif de secteurs écrits. Sinon CF est mis à 1 et le registre `ah` contient un code d'erreur.

Exemple.- Comme nous l'avons déjà dit, il faut être très prudent avec cette opération, puisqu'une fausse manœuvre risque de rendre un disque indisponible. Il vaut donc mieux s'essayer sur une disquette vierge (formatée). Commençons par enregistrer un texte quelconque sur cette disquette. Vérifions ensuite sur quel secteur il a été écrit, soit en lisant la FAT, soit en utilisant le programme précédent et en tâtonnant : il y a évidemment, s'il s'agit d'une diquette 3.5" haute densité, toutes les chances qu'il s'agisse du secteur de numéro relatif 33 (le premier de la zone des données), c'est-à-dire le secteur 16 de la piste 0 de la tête 1.

Écrivons un programme permettant d'écrire un autre texte sur ce secteur de la disquette située dans le disque A :

```
TITLE ecritsec.asm
; ecrit sur un secteur du disque A
      .model small
      .stack 256
      .data
tampon db 'essai d ecriture sur un secteur'
message db 'erreur en ecriture', '$'
      .code
debut:
      mov ax, @data
      mov ds, ax
      mov es, ax            ; pour le tampon

      mov ah, 3             ; ecriture secteur
      mov al, 1             ; un secteur
      mov bx, offset tampon ; pour entreposer
      mov ch, 0             ; cylindre 0
      mov cl, 16            ; secteur 16
      mov dh, 1             ; tete 1
      mov dl, 0             ; disque A
      int 13h               ; ecriture du secteur
      jc erreurs            ; en cas d'erreur
      jmp fin
erreurs:
      mov dx, offset message
      mov ah, 9h
      int 21h               ; affiche erreur en ecriture
      jmp fin
fin:
      mov ax, 4c00h
      int 21h
      end debut
```

Remarquons ce qui se passe lorsque le texte mis dans le tampon n'est pas de la même longueur que le texte précédemment enregistré sur la disquette : tout le contenu de 512 Ko de la mémoire vive situé à l'adresse du tampon est écrit sur la disquette ; le nombre de caractères enregistré dans le répertoire est affiché avec la commande `type`, par exemple ; ainsi le texte est soit tronqué, soit la fin du texte qui se trouvait précédemment à l'emplacement de la mémoire vive est affiché.

## 1.5 Taille de la mémoire : l'interruption 12h

Description de l'interruption.- L'interruption 12h permet de déterminer la quantité de mémoire vive, limitée à 640 KiO, quantité qu'elle place dans le registre AX.

Exemple.- Le programme suivant permet de déterminer cette quantité de mémoire :

```
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
-
```

## 1.6 Le temps : les interruptions 08h et 1Ah

## 1.7 Les ports séries et parallèles

## 1.8 L'impression : interruption 17h

Introduction.- De nos jours nous sommes habitués aux imprimantes laser ou couleur à jet d'encre à haute résolution, et donc graphique. Au début des micro-ordinateurs, une imprimante qui permettait d'imprimer du texte caractère par caractère était déjà beaucoup plus efficace qu'une machine à écrire : on pouvait commencer par visualiser le texte et le corriger si nécessaire. Ceci est un premier intérêt pour voir comment faire, même si cela est émulé sur les imprimantes modernes ; mais n'est-ce pas tout l'intérêt des interruptions. De toutes façons, et c'est le deuxième intérêt, les instructions sont transmises à l'imprimante de la même façon, caractère par caractère, la seule différence étant qu'il ne s'agit plus du texte proprement dit mais d'un programme source (dans des **langages de description de page** tels que `Postcript` ou `PCL`).

## 1.9 Le système

L'heure, la date

## 1.10 Bibliographie

[IBM-81] **Technical Reference Manual**, IBM (P/N 6025008), Boca Raton, Florida, 1981.

[Le BIOS conçu à l'origine par IBM est décrit dans le chapitre 3 du manuel de références techniques. L'annexe A du même manuel contient le fichier source complet du BIOS.]

[IBM-84] **Technical Reference – Options and Adapters**, IBM Corporation (P/N 6322509), Boca Raton, Florida, 1984, two volumes.

[Détaille les cartes d'extension, principalement pour le PC/XT. Comprend des schémas et le listage du BIOS associé.]

[IBM-85] **Technical Reference – Personal Computer AT**, IBM Corporation (P/N 6280099), Boca Raton, Florida, 1985.

[Le manuel d'IBM le plus détaillé, le dernier avec des schémas et le listage du BIOS.]

[Bra-86] Bradley, David J., **Assembly Language Programming for the IBM Personal Computer**, Prentice-Hall. Traduction française **Assembleur sur IBM PC**, Masson, 1986, XII + 372 p.

[David Bradley, qui a fait partie de l'équipe ayant conçu l'ordinateur individuel d'IBM, commente le BIOS, sans reprendre les éléments ci-dessus (qu'il suggère d'utiliser en parallèle), dans le chapitre 9 de son livre.]

[Abe-98] Abel, Peter, **IBM PC Assembly Language and Programming**, Fourth Edition, 1998, Prentice-Hall, XVI+606 p.

[Bonne introduction à la programmation en langage d'assemblage de l'IBM-PC. Il s'agit de la programmation en langage d'assemblage de l'IBM-PC et non de l'intel 8086, c'est-à-dire qu'il y a présentation des outils `debug` et MASM, la programmation avec les instructions du 8086 mais aussi avec les interruptions du BIOS et celles de MS-DOS.]

[BK-94] Brown, Ralf & Kyle, Jim, **PC Interrupts : A Programmer's Reference to BIOS, DOS, and Third-Party Calls**, Addison-Wesley, second edition 1994, VI + 1210 p.

[Les interruptions sont décrites officiellement dans les livres d'IBM ci-dessus. Cependant toutes les interruptions du BIOS n'y sont pas décrites, d'où la nécessité de recourir aux deux classiques que sont ce livre et le suivant. Celui-ci décrit les interruptions (et les conflits éventuels), sans donner aucune indication sur la façon dont sont conçues les routines de service associées. ]

[VGI-97] Van Gilluwe, Frank, **The undocumented PC**, Addison-Wesley, second edition 1997, XII + 1123 p. + diskette. Traduction française **PC Programmation système**, CampusPress France, 1999, X + 1242 p. (sans disquette et sans bibliographie).