Chapitre 14

Accès aux périphériques

Nous avons vu, pour l'instant, comment le microprocesseur a accès aux registres et à la mémoire vive. Il faut aussi avoir accès aux périphériques (d'entrée-sortie) de façon à introduire des données (*via* la clavier, la souris, un lecteur de disquettes, un disque dur, un lecteur de CD-ROM, une tablette à digitaliser, un scanner, un capteur...) et à communiquer les résultats (*via* l'écran du moniteur, une imprimante...). Nous allons voir dans ce chapitre comment le microprocesseur accèdent à ceux-ci.

14.1 Principe de l'accès aux périphériques

Les périphériques peuvent être d'une très grande complexité mais, du point de vue de la programmation et de l'échange des données, ils reposent sur un principe simple.

Contrôleur de périphérique.- Un périphérique est complexe, comprenant à la fois des parties mécaniques et des parties électroniques. La communication microprocesseur/périphérique et la programmation du périphérique se fait grâce à une partie électronique appelée **contrôleur** (du périphérique).

Adresse de périphérique.- Comme pour le microprocesseur, la partie essentielle d'un contrôleur de périphérique est un circuit intégré, que l'on peut considérer comme une boîte noire possédant un certain nombre de broches. Un ensemble ou plusieurs ensembles de huit broches permettent d'accéder au contrôleur du périphérique. Un tel ensemble est relié au microprocesseur de façon à correspondre à une **adresse** donnée **de périphérique**. Celle-ci n'est déterminée ni par le concepteur du microprocesseur, ni par celui du circuit intégré du contrôleur ; c'est le concepteur de l'ordinateur qui la choisira de façon plus ou moins arbitraire.

Périphériques mappé en mémoire ou indépendant.- Il existe deux façons de relier un périphérique au microprocesseur, le concepteur du microprocesseur choisissant l'une d'elle. L'adresse de périphérique peut être une adresse de mémoire vive (qui sera donc dédiée à un périphérique) : on parle alors de **périphérique mappé en mémoire**. Cette adresse peut également être spécifique aux périphériques, avec un jeu d'adresse indépendant du jeu d'adresses mémoire : on parle alors de **périphérique indépendant**.

Là encore, le fait qu'un périphérique donné soit, sur un système donné, mappé en mémoire ou indépendant ne dépend ni du périphérique, ni réellement du microprocesseur, mais du concepteur du système.

Cependant, si tous les microprocesseurs permettent le mappage en mémoire des périphériques, seuls certains microprocesseurs ont un jeu d'adresses indépendant pour les périphériques. C'est le cas des microprocesseurs *Intel* mais pas des microprocesseurs Motorola, qui équipaient les premiers MacIntosh *Apple*.

Notion de port.- Les microprocesseurs à entrée-sortie indépendante accèdent aux périphériques *via* un certain nombre de **ports**, chacun étant repéré par une adresse. Un port de périphérique n'est rien d'autre qu'une liaison physique pour un octet.

Un tel microprocesseur indique, *via* une des broches, qu'il veut accéder à un port (et non à un élément mémoire). Il utilise le bus des adresses pour indiquer le numéro du port. Il utilise les broches des données pour envoyer ou recevoir une donnée.

Il appartient au concepteur de l'architecture de l'ordinateur de relier comme il le veut, et donc de déterminer que tel ou tel port correspond à tel ou tel périphérique (ou partie de périphériques). Bien entendu, pour un ordinateur donné, tous les ports ne sont pas utilisés (reliés).

Registres internes des périphériques.- Les données sont rarement transmises directement. Un périphérique contient en général un ou plusieurs **registres internes** (toujours en très petit nombre, comme pour un microprocesseur) et on communique à travers ceux-ci.

Un périphérique utilisera en général plusieurs ports, souvent un par registre interne bien qu'on puisse aussi indiquer, à l'aide de broches spécifiques, à quel registre on veut avoir affaire.

Programme interne.- Le contrôleur contient un programme (éventuellement câblé) qui détermine ce qu'il doit faire des données des registres internes. On parle de **firmware**. Celui-ci est indépendant du microprocesseur et ne nous concerne pas pour l'instant.

14.2 Cas du 8088

14.2.1 Aspect matériel

Le microprocesseur 8088 permet à la fois le mappage en mémoire et l'accès indépendant *via* un port.

Nombre de ports.- Dans le cas du 8088, seules huit broches peuvent être utilisées directement pour déterminer l'adresse d'un port : les broches `A0` à `A7` du bus des adresses. Ceci détermine le nombre maximum de ports à 256. On peut également utiliser, en plus, les broches `A8` à `A15` *via* le contenu du registre DX (pour les broches A0 à A15) ; on peut donc ainsi accéder à 65 536 ports.

Capacité d'un port.- Dans le cas du 8088, l'entrée ou la sortie *via* un port se fait soit octet par octet, soit mot (de deux octets) par mot, et ceci toujours *via* l'accumulateur (AL ou AX suivant la capacité désirée).

14.2.2 Entrée dans l'accumulateur

Langage symbolique.- L'instruction pour l'entrée est `IN` (pour *INput*) :

```
IN A, source
```

où A est l'un des des accumulateurs AL ou AX (déterminant ainsi la capacité du port) et `source` soit une constante sur huit bits, soit le registre DX.

Langage machine.- L'instruction IN est codée :

- sur deux octets dans le cas d'une **entrée immédiate**, par :

  ```
  | 1110 010w | port |
  ```

  avec $w = 0$ pour AL et $w = 1$ pour AX, soit E4h ou E5h ;
- sur un octet dans le cas d'une **entrée implicite**, c'est-à-dire *via* le registre `DX`, par :

  ```
  | 1110 110w |
  ```

  avec $w = 0$ pour AL et $w = 1$ pour AX, soit ECh ou EDh.

14.2.3 Sortie du contenu de l'accumulateur

Langage symbolique.- L'instruction pour la sortie est `OUT` (pour *OUTput*) :

```
OUT destination, A
```

où A est l'un des des accumulateurs AL ou AX (déterminant ainsi la capacité du port) et `destination` soit une constante sur huit bits, soit le registre DX.

Langage machine.- L'instruction OUT est codée :

- sur deux octets dans le cas d'une sortie immédiate, par :

  ```
  | 1110 011w | port |
  ```

  avec $w = 0$ pour AL et $w = 1$ pour AX, soit E6h ou E7h ;

– sur un octet dans le cas d'une **sortie implicite**, par :

```
| 1110 111w |
```

avec $w = 0$ pour AL et $w = 1$ pour AX, soit EEh ou EFh.

14.3 Un exemple : voyants lumineux du clavier MF II

On aimerait bien commencer par un exemple permettant d'entrer un caractère via le clavier. Cependant, nous réservons un tel cas à plus tard, au moment de l'étude du BIOS, car nous verrons que ce n'est pas très simple. Nous allons donc considérer un exemple beaucoup plus simple, à savoir la manipulation des voyants lumineux du clavier.

Les voyants lumineux.- Sur le clavier dit MF Il (et ultérieurs) de l'IBM PC, il y a trois voyants lumineux, situés en haut à droite du clavier, dénommés « NumLock », « CapsLock » et « ScrollLock ». Les deux premiers permettent de rappeler respectivement à l'utilisateur que le pavé numérique de droite sert à entrer des chiffres et non des déplacements et qu'on entre des majuscules (et non des minuscules).

Les voyants lumineux ne sont pas pilotés par le contrôleur du clavier, lorsqu'on appuie sur la touche de verrouillage du pavé numérique par exemple, mais *via* le microprocesseur de l'unité centrale et, en fait, par le système d'exploitation.

Les registres du clavier.- Nous verrons plus tard en détail le fonctionnement du clavier de l'IBM PC et son interfaçage avec le microprocesseur. Voyons pour l'instant uniquement que ce qui est intéressant pour notre propos.

Le contrôleur d'un clavier pour compatible PC possède quatre registres internes, appelés **tampon d'entrée**, **tampon de sortie**, **registre de contrôle** et **registre de statut**. Sur un compatible PC, on utilise les deux ports d'adresses 60h et 64h pour accéder à ces registres. Il suffit de deux ports car chacun de ces registres est seulement accessible soit en écriture, soit en lecture. Pour le tampon d'entrée, le seul qui nous intéressera ici, on écrit sur le port 60h.

Les commandes du clavier.- Le comportement des claviers AT et MF II peut être programmé grâce à un jeu de commandes que l'on transmet au contrôleur de clavier *via* le tampon d'entrée. Nous verrons plus tard, lorsque nous étudierons le clavier en détail, ce jeu de commandes. Contentons-nous ici de la commande qui permet d'allumer et d'éteindre les voyants lumineux du clavier MF II.

On passe la commande :

```
EDh
```

au tampon d'entrée. Le contrôleur du clavier répond par un accusé de réception `ACK`, ne s'occupe plus des touches et attend un **octet d'indication**, qui est également passé *via* le tampon d'entrée.

La structure de l'octet d'indication est la suivante :

```
0 0 0 0 0 CPSL NUML SCRL
```

avec `CPSL` égal à 1 pour que le voyant lumineux de « CapsLock » soit allumé, `NUML` égal à 1 pour que celui de « NumLock » soit allumé et `SCRL` égal à 1 pour que celui de « ScrollLock » soit allumé.

Exemple.- Écrivons un programme permettant d'allumer le voyant lumineux de « NumLock » (sans que le pavé numérique ne soit pour autant verrouillé). En langage symbolique, nous avons :

```
MOV AL, ED
```

```
OUT 60, AL
MOV AL, 02
OUT 60, AL
RET
```

Traduisons ce programme, écrit en langage symbolique, en langage machine :

```
B0 ED
E6 60
B0 02
E6 60
C3
```

Utilisons alors `debug` pour écrire le programme `mf2.com` en langage machine :

```
C:\COM\>debug
-E CS:100 B0 ED E6 60 B0 02 E6 60 C
-R BX
BX 0000
:
-R CX
CX 0000
:09
-N mf2.COM
-W
Ecriture de 00009 octets
-Q
```

Lorsqu'on appelle ce programme en ligne de commande, on s'aperçoît bien que le voyant lumineux est allumé.

Remarque.- Il faut utiliser un vrai système d'exploitation MS-DOS pour lancer ce programme et non une fenêtre MS-DOS de Windows ou une émulation sous Linux. En effet ces deux derniers systèmes d'exploitation fonctionnent en deux modes : le *mode noyau* et le *mode utilisateur*. Le mode noyau permet de lancer ces systèmes d'exploitation et n'est plus accessible après (y compris par le superutilisateur `root`). Seul le mode noyau a accès aux entrées-sorties (c'est-à-dire aux instructions `IN` et `OUT`) ; en mode utilisateur on accède aux entrées-sorties à travers des *appels système*, uniquement pour ce qui est prévu par le système d'exploitation. C'est la raison pour laquelle il faut recompiler le noyau de Linux lorsqu'on change de périphérique et qu'il faut redémarrer l'ordinateur pour Windows.

14.4 Historique

Le premier ordinateur (EDSAC, 1949) disposait d'un seul périphérique : un télescripteur. L'utilisation de celui-ci était câblée et son langage symbolique disposait de deux instructions pour y accéder (voir chapitre ? ?).