## Chapitre 10

# Mémoire virtuelle et pagination

Les concepts de *mémoire virtuelle* et de *pagination* ont été introduits pour pallier au manque de mémoire vive des premiers ordinateurs. Ils sont implémentés au niveau du système d'exploitation mais les microprocesseurs en facilitent la mise en place.

## 10.1 Principes

### 10.1.1 Mémoire virtuelle

Dans certains cas les besoins en mémoire s'accroissent et ne peuvent pas toujours être satisfaits par la mémoire vive physiquement présente. Le système d'exploitation met alors (fictivement, évidemment) à la disposition des programmes plus de mémoire qu'il n'en existe réellement. La méthode utilisée repose sur la constation suivante : l'intégralité du besoin en mémoire vive n'est pas constamment utilisée ; il est donc possible de stocker sur disque les parties inactives jusqu'à ce qu'elles soient de nouveau réclamées par un programme.

Le principe de la **mémoire virtuelle** est le suivant : les adresses manipulées par l'utilisateur ne sont pas les *adresses physiques* mais des **adresses** dites **virtuelles**. Un mécanisme, appelé **unité de gestion de la mémoire** (**MMU** pour l'anglais *Memory Management Unit*), traduit les adresses virtuelles en adresses physiques.

L'intérêt est le suivant : on peut s'arranger pour que la plage des adresses virtuelles soit (bien) plus grande que la plage des adresses physiques. Bien entendu la taille de la mémoire virtuelle effectivement utilisée à un moment donné doit être inférieure à la taille de la mémoire physique. Les adresses virtuelles correspondant à de la mémoire physique ne sont pas nécessairement contigues. D'un moment à l'autre, les adresses virtuelles ayant un correspondant physique ne sont pas nécessairement les mêmes.

Une façon de gérer la mémoire virtuelle est la *pagination*, que nous allons étudier maintenant.

### 10.1.2 Pagination

#### 10.1.2.1 Principe

Le principe de la **pagination** (*memory paging* en anglais) repose sur les *cadres*, les *pages* et les *tables de pages* :

**Cadre.** La mémoire vive disponible est partitionnée en **cadres** (*page frame* en anglais) de taille fixe.

**Page.** Une **page** est un bloc de données dont la taille est celle d'un cadre. Elle peut être stockée dans n'importe quel cadre de la mémoire vive ou sur disque.

**Table de pages.** Un ensemble de **tables de pages** spécifie la correspondance entre les adresses virtuelles et les adresses physiques.

On parle d'**adresse linéaire** (ou **virtuelle**) lorsque la pagination est activée. Des adresses linéaires contiguës à l'intérieur d'une même page correspondent à des adresses physiques contiguës. Par contre, ceci n'est pas le cas si elles se trouvent sur des pages distinctes.

Les microprocesseurs actuels contiennent des circuits, constituant l'**unité de pagination**, chargés de traduire automatiquement les adresses virtuelles en adresses physiques. Le mécanisme de conversion (figure 10.1) repose sur le principe suivant : une adresse virtuelle est décomposée en deux parties, un **numéro de page** et un **décalage** (*offset* en anglais) dans la page. Le numéro de page est utilisé comme indice dans un tableau, appelé **table des pages**, fournissant l'adresse physique (de début de page) en mémoire centrale de cette page. À cette adresse est ajouté le décalage pour obtenir l'adresse physique de l'élément mémoire concerné.

Remarque.- La segmentation de la mémoire des microprocesseurs *Intel* et la pagination relèvent du même principe de base : la différence entre les deux tient à ce que l'on utilise deux variables pour la segmentation et une seule pour la pagination.

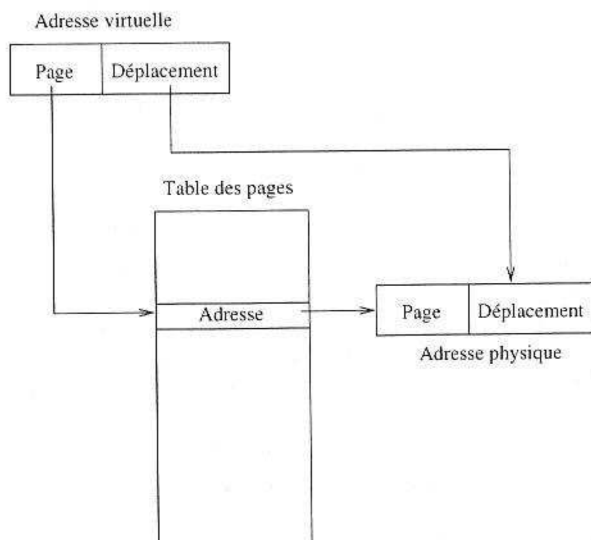


FIGURE 10.1 – Pagination

#### 10.1.2.2 Pagination à plusieurs niveaux

Notion.- Lorsqu'il n'y a qu'une seule table de pages, elle doit nécessairement se trouver en permance en mémoire vive ; cette façon de faire nécessite d'utiliser beaucoup de mémoire uniquement consacrée à la table complète. Elle n'est donc que rarement implémentée sous la forme d'une seule table contiguë en mémoire :

— Pour un microprocesseur 16 bits, on peut utiliser un seul niveau de pages. On a, par exemple, des pages de 4 kiO, soit douze bits pour le décalage. Il reste quatre bits pour les pages, soit 16 pages. On a donc une table de pages de taille raisonnable.

— Pour un microprocesseur 32 bits, il est moins raisonnable de n'utiliser qu'une seule page et donc un seul niveau de pagination. Toujours pour des pages de 4 kiO, il reste 20 bits pour les pages, soit une table de 1 Mi pages. Comme la description de chaque page exige plusieurs octets, cela représente une table de plusieurs MiO, ce qui occupe trop de place en mémoire vive. On utilise donc deux niveaux de tables de pages, dix octets étant affectés à chaque type de tables.

  Par exemple, le microprocesseurs *Intel* `80386` peut adresser 4 GiO, la taille des pages mémoire est de 4 kiO, et chaque entrée de la table occupe quatre octets ; sur de tels processeurs une table de page complète utiliserait 1 048 576 entrées, pour une occupation mémoire de 4 MiO.

— Pour un microprocesseur 64 bits, on utilise une pagination à trois niveaux.

Répertoire de tables de pages.- La table de pages est donc souvent décomposée en plusieurs **niveaux**, deux au minimum :

- un **catalogue** (ou **répertoire**) **de tables de pages** (*page directory* en anglais) représente le niveau deux, il contient les adresses des pages qui contiennent, quant à elles, des parties de la table des pages ;
- ces parties sont les tables de pages de niveau un.

La figure 10.2 représente la conversion d'adresse dans le cas d'une architecture utilisant une

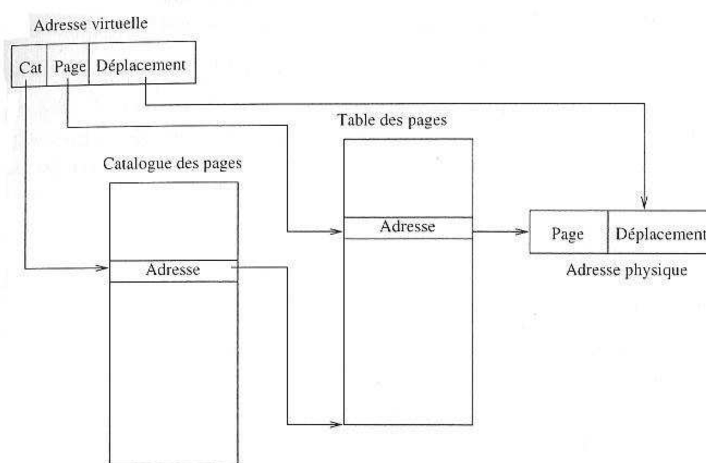


FIGURE 10.2 – Tables de pages

table de pages à deux niveaux.

L'intérêt de cette table de pages à deux niveaux repose sur le fait que la table de pages n'a pas besoin d'être chargée entièrement en mémoire vive. Si on utilise 6 MiO (contigus), seules trois pages sont utilisées pour la table des pages :

- la page contenant le répertoire, toujours présente en mémoire vive ;
- la page contenant la partie de la table des pages correspondant aux 4 premiers méga-octets de mémoire ;
- la page contenant la partie de la table des pages correspondant aux quatre méga-octets de mémoire suivants (dont seule la moitié des entrées est utilisée).

## 10.2 La pagination de l'architecture *Intel* IA-32

La pagination et la MMU sont mis en place de façon matérielle sur les microprocesseurs *Intel* depuis le `80386`. Pour les microprocesseurs 32 bits, elle utilise deux niveaux. Il faut initialiser le répertoire de table de pages, les tables de pages puis activer la pagination pour pouvoir utiliser les adresses virtuelles.

### 10.2.1 Mise en place

#### 10.2.1.1 Registres de contrôle

La pagination est contrôlée par les **registres de contrôle**, d'une longueur de 32 bits, dont nous avons déjà rencontré le premier, et en particulier son bit 0, à propos du passage au mode protégé.

La figure ci-dessous donne la structure de ces quatre registres `CR0` à `CR3` :

| | | MSW | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PG | 0000000000000000 | 00000000000 | ET | TS | EM | MP | PE | CR0 |
| Not used | | | | | | | | CR1 |
| Page fault linear address | | | | | | | | CR2 |
| Page directory base | | | 000000000000 | | | | | CR3 |

— Le registre de contrôle `CR1` n'est pas utilisé par IA-32 (ni non plus par x86-64 d'ailleurs). Il est réservé aux produits futurs.

— Le registre de contrôle `CR2` contient l'adresse linéaire de la dernière page à laquelle on a eu accès lors d'une interruption pour faute de pagination.

— Le registre de contrôle `CR3` contient l'adresse physique de base du répertoire de page sur 20 bits (ceux de poids supérieurs). Les 12 bits de poids inférieur contiennent 0.
L'adresse physique du répertoire de page doit donc être aligné sur une page.

— Le registre de contrôle `CR0` contient un certain nombre de bits de contrôle, dont nous avons déjà décrits le rôle du premier, PE (à propos du passage au mode protégé). Le seul bit nous intéressant ici est le bit 31, noté **PG** (pour *PaGination*), activant la pagination lorsqu'il est égal à 1. Dans ce cas, les tables de pages sont utilisées pour traduire les adresses virtuelles en adresses physiques.

#### 10.2.1.2 Taille des pages

Depuis le `80386`, l'unité de pagination des microprocesseurs *Intel* gère des pages de 4 kiO. Il faut donc 12 bits de décalage pour situer un octet dans une page puisque $2^{12} = 4\ 096$.

Depuis le *Pentium*, *Intel* gère également des pages de 4 MiO (*Intel* parle de *pagination étendue*). On utilise alors un cinquième registre de contrôle, noté CR4, pour spécifier la taille des pages. Nous étudierons ce cas plus tard.

#### 10.2.1.3 Pagination à deux niveaux

L'architecture IA-32 utilise la pagination (de façon optionnelle, rappelons-le) à deux niveaux :

— L'adresse physique du répertoire de pages, devant être un multiple de `0x1000`, rappelons-le également, est stockée dans le registre de contrôle `CR3`.

— Le répertoire de tables de pages contient jusqu'à 1 024 entrées de 32 bits, permettant de localiser jusqu'à 1 024 tables de pages. Le répertoire de tables de pages a donc une taille maximale de 4 kiO, taille d'une page.

— Chaque table de pages contient également jusqu'à 1 024 entrées de 32 bits, permettant de localiser jusqu'à 1024 pages. Une table de pages a donc également une taille maximum de 4 kiO, taille d'une page.

— Si les 4 GiO de mémoire sont paginés, le système doit allouer 4 kiO de mémoire pour le répertoire de tables de pages et jusqu'à 4 Ki fois 1024 octets, soit 4 MiO, pour les 1 024 tables de pages, représentant une ressource mémoire considérable.

#### 10.2.1.4 Structure d'une adresse virtuelle

Les 32 bits d'une adresse virtuelle sont répartis en trois champs :

| 31 — 22 | 21 — 12 | 11 — 0 |
|---|---|---|
| Directory | Page table | Offset |

— Puisque 1 024 = $2^{10}$, dix bits, les bits 22 à 31, forment un premier index, permettant de sélectionner la table de pages dans le répertoire des tables de page.

— Dix autres bits, les bits 12 à 21, constituent un second index, dit de *page*, pointant dans la table de pages sélectionnée par le premier index. On y trouve l'adresse physique du début de la page.

— Les 12 bits de poids faible, les bits 0 à 11, constituent le *décalage*.

#### 10.2.1.5 Structure d'une entrée de table de pages

Les entrées du répertoire des tables de pages et des tables de pages ont la même structure, à un bit près. Chaque entrée a une taille de 32 bits, soit 4 octets, dont les champs sont les suivants :

| 31 … 12 | 11 … 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Adresse de base | réservé | D | A | P<br>C<br>D | P<br>W<br>T | U<br>/<br>S | R<br>/<br>W | P |

— Les vingt bits 12 à 31 permettent de spécifier l'adresse du début de la table de pages (pour le répertoire) ou de la page (pour une table de pages). Ils sont interprétés comme les 20 bits de poids fort de l'adresse physique.

  En effet chaque table de page et chaque page a non seulement une taille de 4 kiO mais on considère en plus qu'elle est *alignée*, c'est-à-dire que son adresse est un multiple de 4 096, donc que les 12 bits de poids faible de cette adresse sont égaux à 0.

— Les bits 7 à 11 sont réservés pour une utilisation qui pourra être définie ultérieurement par *Intel* et devant être nuls en attendant.

— Le bit 6 (D pour *Dirty*, c'est-à-dire *modifié*) n'a de sens que pour les entrées d'une table de pages. Il doit être nul pour une entrée du répertoire de tables de pages.

  Ce bit est positionné par le microprocesseur chaque fois qu'une opération d'écriture est réalisée sur la page : ceci permet au système d'exploitation de savoir qu'il doit en sauvegarder le contenu sur le disque. L'unité de pagination ne remet jamais ce drapeau à 0 ; c'est au système d'exploitation de le faire éventuellement (après sauvegarde sur disque).

— Le bit 5 (A pour *Accessed*) indique si on a eu accès à cette page (en lecture ou en écriture, cette fois-ci) : cela facilite la gestion de la mémoire virtuelle par le système d'exploitation.

  Comme pour le bit D, l'unité de pagination ne remet jamais ce bit à zéro ; c'est au système d'exploitation de le faire éventuellement.

— Les bit 4 (PCD pour *Page-level Cache Disable*) et 3 (PWT pour *Page-level Write-Through*) contrôlent la mise en cache et ne nous intéressent pas pour l'instant.

— Le bit 2 (U/S pour *User/System*) spécifie les privilèges : lorsque ce drapeau est égal à 0, le répertoire de tables de pages ou la table de pages n'est accessible qu'aux niveaux de privilège 0 à 2.

— Le bit 1 (R/W pour *Read/Write*) spécifie les droits de lecture et d'écriture lorsque que le bit 2 vaut 1 : lorsque le drapeau est égal à 0, la table de pages ou la page peuvent être lus et écrits au niveau de privilège 3 ; lorsqu'il est égal à 1, elle ne peut qu'être lue à ce niveau de privilège 3.

— le bit 0 (P pour *Present*) indique si la table de pages ou la page est présent en mémoire vive. C'est évidemment le système d'exploitation qui renseigne ce bit.

  Si ce drapeau est positionné, la page (ou la table de pages) référencée est présente en mémoire vive ; s'il est baissé, elle n'est pas en mémoire vive et les autres bits peuvent être utilisés par le système d'exploitation pour un usage qui lui est propre.

  Si ce drapeau est égal à 0 et que l'on essaie de faire appel à cette entrée, l'unité de pagination stocke l'adresse virtuelle dans le registre de contrôle `CR2` et lève l'exception 14, c'est-à-dire l'**exception de défaut de page** (*Page Fault* en anglais). Il appartient au concepteur du système d'exploitation d'écrire un gestionnaire d'interruption plaçant alors la page en mémoire vive.

#### 10.2.1.6 Activation de la pagination

Au démarrage des microprocesseurs *Intel*, la pagination n'est pas activée : les adresses virtuelles sont interprétées comme des adresses physiques. Pour activer la pagination, il faut positionner le drapeau `PG` (bit 31) du registre de contrôle `CR0`, après avoir initialisé le registre `CR3`, le répertoire et les tables de pages.

#### 10.2.1.7 Mécanisme de protection matérielle

L'unité de pagination utilise un mécanisme de protection différent de celui utilisé par l'unité de segmentation. Alors que les microprocesseurs *Intel* permettent d'utiliser quatre niveaux de privilège différents pour un segment, seulement deux niveaux de privilège peuvent être utilisés pour les pages et les tables de pages. Lorsque le drapeau `User/System` vaut 0, la page ne peut être adressée que lorsque le CPL est strictement inférieur à 3. Lorsqu'il est égal à 1, la page peut toujours être adressée.

De plus, à la place des trois types de droits d'accès (lecture, écriture, exécution) associés à un segment, seulement deux types de droits (lecture et écriture) sont associés à une page. Si le drapeau `Read/Write` d'une entrée du répertoire de tables de pages ou d'une table de pages vaut 0, la table de pages ou la page correspondante ne peut être que lue ; sinon elle peut être lue et modifiée.

Le registre de contrôle `CR2` contient l'adresse linéaire de la dernière page à laquelle on a eu accès lors d'une interruption pour faute de pagination.

#### 10.2.1.8 Le cache de pagination

Le microprocesseur n'a pas besoin de consulter les tables de pagination lors de chaque accès mémoire, fort heureusement, comme il n'est pas nécessaire de repasser par les tables de descripteurs de segments à chaque accès mémoire. En effet, le microprocesseur dipose d'un tampon interne, appelé **cache de pagination** (**TLB** pour *Translation Lookaside Buffer* en anglais), capable de mémoriser 32 entrées de pages de 4 kiO, et donc d'une taille de 128 kiO. Ce tampon est « caché ».

À chaque accès mémoire, le processeur recherche d'abord si la page voulue possède son entrée dans ce tampon. Si c'est le cas, aucune autre recherche n'est utile. Les tests de sécurité sont effectués et l'adresse physique est calculée, le cas échéant : cela consiste à additionner l'adresse physique du début de la page, figurant dans le tampon, au décalage présenté par l'adresse virtuelle.

Si la page n'est pas présente dans le cache de pagination, il y a échec, ce qui oblige le processeur à passer par les tables de pagination. Il en profite pour charger l'entrée de la nouvelle page dans le tampon, abandonnant, par exemple, la plus ancienne entrée si le cache est plein.

### 10.2.2 Passage au mode protégé et retour

Lors du retour au mode réel, la première chose à effectuer, si la pagination est activée, est de la désactiver. Ceci doit être effectué à partir d'une page pour laquelle les adresses virtuelles sont égales aux adresses physiques (« *identity mapped* » en anglais). Il est prudent de mettre à zéro le registre `CR3` de façon à vider le cache de pagination.

## 10.3 Exemple

Utilisons, comme d'habitude, notre clé USB démarrant sur MS-DOS. Créons-y un répertoire PAGE dans lequel nous placerons les programmes concernant la pagination.

Écrivons un programme qui, comme d'habitude, place le microprocesseur en mode protégé et qui reviendra en mode réel à la fin.

Une fois en mode protégé, préparons la pagination. Rappelons que nous avons déjà remarqué que les emplacements mémoire `6000:0` et la suite, ainsi que `7000:0` et la suite, ne sont pas utilisés par le cœur du système d'exploitation MS-DOS : tout est à zéro au démarrage.

Plaçons notre répertoire de tables de pages à l'adresse `0x60000`, qui est bien un multiple de `0x1000`, avec une seule entrée, celle d'index 0, ayant la valeur `0x61007`. Ceci signifie que la table de pages se trouve à l'adresse `0x61000`, nécessairement un multiple de `0x1000`, qu'elle est présente et qu'on a les droits d'écriture et de lecture (ce qui n'a d'ailleurs pas beaucoup d'importance pour ces derniers pour l'instant, puisqu'on reste toujours au niveau de privilège 0).

Puisque nous ne savons pas trop comment le BIOS et MS-DOS utilisent le premier MiO de la mémoire, ne déplaçons surtout rien dans cette partie. Nous allons donc faire en sorte que les adresses virtuelles et physiques se confondent pour la plage `0x00000 - 0xFFFFF`. Ceci revient à placer `0x000007` pour l'index 0 de la table de pages, `0x001007` pour l'index 1 et ainsi de suite, en incrémentant de `0x1000` à chaque fois, pour les 256 premières entrées.

Nous nous sentons un peu plus libre au-delà du premier MiO (nous supposons bien sûr que la mémoire vive implantée est supérieure à 1 MiO, ce qui ne doit pas être un problème avec un ordinateur actuel). Nous allons donc, non pas placer `0x100007` pour l'entrée suivante, mais `0x102007`. Cela signifie que la plage d'adresses virtuelles `0x102000 - 0x102FFF` correspondra à la plage d'adresses physiques `0x100000 - 0x100FFF`.

Une fois la pagination mise en place, écrivons `0x12345678` à l'adresse virtuelle `0x100000`. Si tout s'est bien passé, on a écrit cette valeur à l'adresse physique `0x102000`.

Désactivons alors la pagination et plaçons ce qui se trouve à l'adresse physique `0x102000` à l'adresse (physique) `0x70000`, de façon à pouvoir vérifier, une fois revenu à MS-DOS, ce qui se trouve à l'adresse `7000:0` et donc aussi à l'adresse `0x102000`.

```
#---------------------------------------------;
# page.s                                      ;
# Example program pagination in Protected Mode ;
#---------------------------------------------;
.section .text
.globl   _start
.code16
.org     0x100
_start:
#
# Save flags and DS for return to real mode
#
        movw    %cs,%dx
        movw    %dx,%ds
        movw    %dx,%ss
        pushf
        push    %ds
        movw    %ds,%dx
        movw    %dx,seg
#
#Setting base for code segments
#
```

```
        movw    %cs,%ax
        movzx   %ax,%eax
        shll    $4,%eax                 # eax=base for code segment
        movl    %eax,DESC1B
        movb    $0x9a,DESC1C            # set segment attribute
#
# Setting of GDTA
#
        movl    $0,%eax                 # deja fait mais prudence
        movl    $0,%ebx
        movw    %cs,%ax                 # deja fait mais prudence
        #movw   DESC0,%bx
        .byte   0x0bb
        .word   DESC0
        shll    $4,%eax                # deja fait mais prudence
        addl    %ebx,%eax
        movl    %eax,GDTA
#
# Make sure no ISR will interfere now
#
        cli
#
# LGDT is necessary before switch to PM
#
        lgdtl   GDTL
#
# Switch to PM
#
        movl    %cr0,%eax
        orb     $1,%al
        movl    %eax,%cr0
#
# Far jump to set CS & clear prefetch queue
#
        .byte   0x0ea
        .word   pm_in
        .word   8
#
# Load long segment descriptor from GDT into DS
#
pm_in:  movw    $0x10,%dx
        movw    %dx,%ds
        movw    %dx,%ss
        movw    %dx,%es
#----------------------------;
# A sample in protected mode: ;
#----------------------------;
#
# Load CR3
#
        movl    $0x60000,%eax
        movl    %eax,%cr3
#
# Set address page directory
```

```
#
        movl    $0x61007,%eax
addr32  movl    %eax,%ds:(0x60000)
#
# Remap 00000h-FFFFFh to 00000h-FFFFFh
#
        movl    $256,%ecx
        movl    $0x61000,%edi
        movl    $7,%eax
REPETER:
        movl    %eax,%es:(%edi)
        incl    %edi
        incl    %edi
        incl    %edi
        incl    %edi
        addl    $0x1000,%eax
        decl    %ecx
        jecxz   FIN
        jmp     REPETER
FIN:
#
# Remap 100000h-AFFFFh to 102000h-102FFFFh
#
        movl    $0x102007,%eax
        movl    $16,%ecx
REPEAT2:
        movl    %eax,%es:(%edi)
        incl    %edi
        incl    %edi
        incl    %edi
        incl    %edi
        addl    $0x1000,%eax
        decl    %ecx
        jecxz   FIN2
        jmp     REPEAT2
FIN2:
#
# Enable pagination
#
        movl    %cr0,%eax
        btsl    $31,%eax
        movl    %eax,%cr0
        jmp     FOO
FOO:    nop
#
# Sample using of address
#
        movl    $0x12345678,%eax
addr32  movl    %eax,%ds:(0x100000)
#
# Deactive pagination
#
        movl    %cr0,%eax
        btrl    $31,%eax
```

```
        movl    %eax,%cr0
        jmp     F002
F002:
        xorl    %eax,%eax
        movl    %eax,%cr3
#
# Read 102000h
#
addr32  movl    %ds:(0x102000),%eax
addr32  movl    %eax,%ds:(0x70000)
#------------------------------------;
# End of the sample in protected mode ;
#------------------------------------;
#
# Load 64kB segment descriptor from GDT into DS for return
#
        movb    $0x18,%dl
        movw    %dx,%ds
#
# Return from PM ro real mode
#
        andb    $0x0fe,%al
        movl    %eax,%cr0
#
# Far jump to restore CS & clear prefetch queue
#
        .byte   0x0ea
        .word   pm_out
seg:    .word   0
pm_out:
#
# Restore DS and flags
#
        movw    %cs,%dx
        movw    %dx,%ss
        pop     %ds
        popf
#
# Exit to MS-DOS
#
        ret
#
#---------------------------------------;
#       GDT to be used in Protected Mode ;
#---------------------------------------;
#
# null descriptor
#
DESC0:  .long   0                     # null descriptor
        .long   0
#
# descriptor for code segment in PM
#
DESC1:  .word   0x0FFFF               # limit 64kB
```

```
DESC1B: .word   0                     # base = 0 to set
        .byte   0                     # base
DESC1C: .byte   0x9A                  # code segment
        .byte   0                     # G = 0
        .byte   0
#
# descriptor for data segment in PM
#
DESC2:  .word   0x0FFFF               # limit 4GB
        .word   0                     # base = 0
        .byte   0                     # base
        .byte   0x92                  # R/W segment
        .byte   0x8F                  # G = 1, limit
        .byte   0
#
# descriptor for data segment return to real mode
#
DESC3:  .word   0x0FFFF               # limit 64kB
        .word   0                     # base = 0 to set
        .byte   0                     # base
        .byte   0x92                  # R/W segment
        .byte   0                     # G = 0, limit
        .byte   0                     # base
#
# GDT table data
#
GDTL:   .word   0x1F                  # limit
GDTA:   .long   0                     # base to set
```

Assemblons ce fichier `page1.s` sous Linux :

```
$ as page1.s --32 -o ./page1.o
$ objcopy -O binary ./page1.o ./page1.com
$ ls -l page1.com
-rwxr-xr-x 1 patrick patrick 569 sept. 28 10:57 page1.com
$
```

Il ne faut garder que les 313 derniers octets du binaire :

```
$ objcopy -O binary -i 800 --interleave-width 313 -b 256 ./page1.o ./page1.com
$
```

Après avoir exécuté ce programme sous MS-DOS, on retrouve bien la valeur voulue en `0x70000` :

```
C:\PROTECT>debug
-D 7000:0 L 10
7000:0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
C:\PROTECT>page1
C:\PROTECT>debug
-D 7000:0 L 10
7000:0 78 56 34 12 00 00 00 00-00 00 00 00 00 00 00 00 xV4.............
-q
```

ce qui montre que la pagination a bien été activée.

## 10.4 Historique

Mémoire virtuelle.- L'Allemand Fritz Rudolf GÜNTSCH (1925–2012) invente le principe de la mémoire virtuelle en 1956. Elle est alors consituée d'une mémoire rapide, de petite taille, et d'une mémoire plus lente, offrant la capacité qu'il désirait : il utilise comme mémoire de grande capacité 10 tambours tournant de façon asynchrone à côté d'une mémoire rapide de 600 mots.

Il écrit dans sa thèse [Gun-57], soutenue en 1957 : « *Le programmeur n'a pas besoin de prendre en considération le stockage à accès rapide* (*il n'a même pas besoin de savoir qu'il existe*). *Puisqu'il y a un seul type d'adresses, intervenant dans la programmation, comme s'il s'agissait d'une seule mémoire présente.* »

Pagination.- L'article [KELS-62] de référence de James KILBURN, paru en 1962, décrit le premier ordinateur doté d'un système de gestion de mémoire virtuelle paginée, utilisant un tambour comme extension de la mémoire centrale à tores de ferrite : l'Atlas.

## 10.5 Bibliographie

[GL-59] GÜNTSCH, F.-R. and R. LUKAS, *Magnetbandrechner der Technischen Universität Berlin*, **Elektronische Datenverarbeitung**, 2/1959, pp. 33–46. (Le calculateur à ruban magnétique de l'université de Berlin).

[Gun-57] GÜNTSCH, F.-R., **Logischer Entwurf eines digitalen Rechengeräts mit mehreren asynchron laufenden Trommeln und automatischem Schnellspeicherbetrieb**, TU Berlin, 1957 (Dissertation) (Esquisse logique d'une machine à calculer numérique ayant plusieurs tambours fonctionnant de façon asynchrone et une mémoire à accès rapide automatique).

[Jes-04] E. JESSEN, *Origin of the Virtual Memory Concept*, **IEEE Annals OF the History of Computing**, Vol. 26, 4/2004, page 71.

[KELS-62] KILBURN, EDWARDS, LANIGAN, and SUMMER, *One level storage system*, **IRE Transactions on elecronic computers**, EC-11, vol. 2, avril 1962, pp. 223–235.