Microprocesseurs (MIC)

Chapitre 3 : Mode réel et mode protégé

Avant le processeur Intel 80286

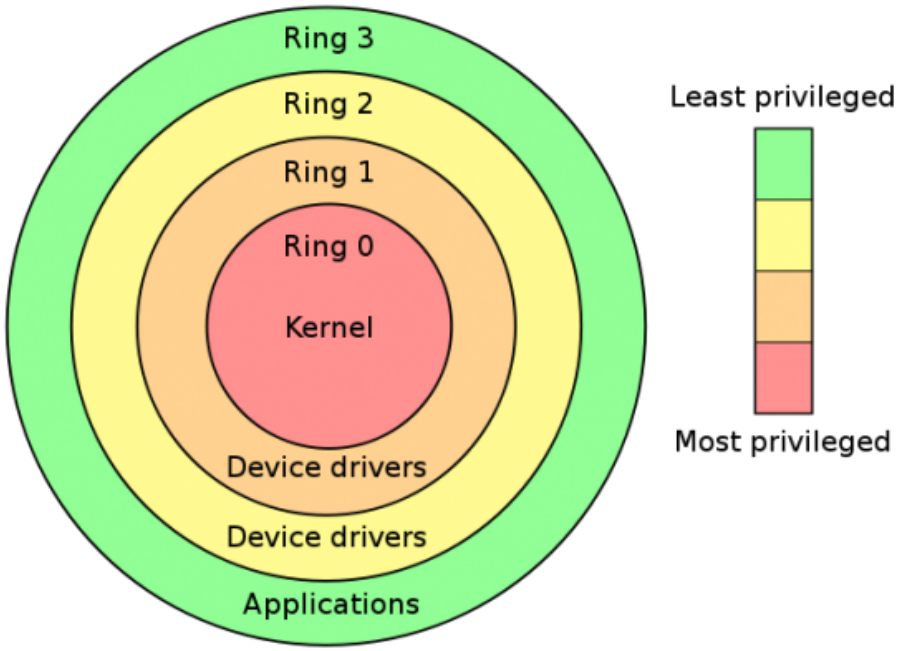
- Chaque programme avait accès à toute la mémoire centrale
- Le système MS-DOS travaillait dans ce mode
- Risques de sécurité :
  - Un programme peut modifier les données d'un autre programme
  - Un programme peut même modifier les données du système d'exploitation !
- Solution : introduction du « mode de fonctionnement protégé » à partir du processeur 80286

Le mode de fonctionnement protégé

- Disponible depuis le processeur 80286
- Souvent abrégé en « mode protégé »
- Ce mode offre des protections à deux niveaux :
  - Un programme ne peut pas accéder à toute la mémoire de l'ordinateur
  - Un programme ne peut pas toujours utiliser toutes les instructions du processeur
- La quasi totalité des systèmes informatiques actuels fonctionnent en mode protégé

Les quatre anneaux de protection du mode protégé

- Le mode protégé offre quatre niveaux de protection, appelés des « anneaux de protection » (*protection rings* en anglais)
- A tout moment, le processeur travaille dans un des quatre niveaux de protection
- A certains moments-clé, le processeur change de niveau de protection



Les quatre anneaux de protection du mode protégé

- Les quatre niveaux de protection sont numérotés de 0 à 3, du plus permissif au moins permissif :

Niveau	Description
0	Les programmes peuvent employer toutes les instructions du processeur et accéder à toute la mémoire centrale (employé pour le code de l'OS sous Windows et Linux)
1	Les programmes ne peuvent employer qu'un nombre réduit d'instructions, et n'ont pas accès à toute la mémoire (non employé sous Windows et Linux)
2	Les programmes ne peuvent employer qu'un nombre réduit d'instructions, et n'ont pas accès à toute la mémoire (non employé sous Windows et Linux)
3	Les programmes ne peuvent employer qu'un nombre réduit d'instructions, et n'ont pas accès à toute la mémoire (employé par les programmes utilisateurs sous Windows et Linux)

Exemple d'emploi d'une instruction interdite

- L'instruction CLI sert à suspendre les interruptions en mettant le flag IF à 0
- Elle n'est permise qu'en ring 0, donc seul l'OS peut l'employer
- L'exemple suivant montre un essai d'utilisation de CLI dans un programme utilisateur

```
; cli.asm

global principal
section .text

principal:
   cli

; fin
mov eax, 1
mov ebx, 0
int 0x80
```

Le mode réel

- Malgré l'introduction du mode protégé, les processeurs 80286 et suivants peuvent encore travailler dans le mode précédent, non-protégé : il s'agit du *mode réel*
- Ce mode permet notamment d'exécuter des vieux OS comme MS-DOS, qui n'étaient pas prévus pour le mode protégé
- En réalité, lors de l'allumage de l'ordinateur, le processeur commence toujours à fonctionner en mode réel, puis l'OS le fait passer en mode protégé
- En mode réel, la taille totale de la mémoire est limitée à 1 Mb (2^{20} bytes), et tous les programmes peuvent y écrire partout sans protection !

La gestion de la mémoire en mode protégé

- Le mode protégé emploie 3 types d'adresses :
  1. Les adresses *logiques* : les adresses employées dans vos programmes assembleur, et vues par le processeur
  2. Les adresses *linéaires* : issues d'une traduction des adresses logiques au travers du processus de *segmentation*
  3. Les adresses *physiques* : véritables adresses de la mémoire centrale

Les adresses logiques

- Les **adresses logiques** sont celles employées dans vos programmes assembleur, et visibles notamment dans Kdbg
- Expérience : exécutez deux fois le même programme en même temps sur linux1 et comparez les adresses grâce à Kdbg
  - Les deux programmes emploient les mêmes adresses !
  - Or deux programmes ne peuvent occupper simultanément le même espace en mémoire !
  - **En réalité**, les adresses employées par votre programme (adresses *logiques*) ne sont pas les véritables adresses *physiques* employées par la mémoire centrale
  - Un mécanisme de **traduction** sera nécessaire pour traduire ces adresses logiques en adresses physiques

Adresses logique et MOV

- Introduisons une nouvelle variante du MOV, pas encore vue au labo : `MOV [adresse], valeur`
- Cette instruction place une valeur à l'adresse-mémoire spécifiée entre crochets
- Exemple : l'instruction `MOV [10], AX` place la valeur de AX en mémoire centrale à l'adresse logique 10
- Attention, ici 10 est bien une adresse logique, et non la véritable adresse physique à laquelle sera placé le contenu de AX !
- AInsi, deux programmes différents peuvent effectuer un `MOV [10], AX` sans se marcher sur les pieds, car l'OS traduira leur deux adresses « 10 » vers des adresses physiques différentes.

Segmentation

- En mode protégé, la mémoire d'un programme est divisée en morceaux appelés des *segments*
- Chaque segment d'un programme peut être situé à un endroit différent de la mémoire centrale
- La localisation précise de chaque segment en mémoire est stockée dans une table appelée *table des segments*
- Cette table contient l'adresse et la taille de chaque segment, ainsi qu'une information disant à quel programme appartient le segment
- Les informations de la table des segments permettent d'empêcher à un programme d'accéder aux segments d'un autre programme (protection de la mémoire)

Segments typiques d'un programme

- **Segment de code** : contient les instructions du programme
- **Segment de données** : contient les variables globales du programme (voir labo)
- **Segment de pile** : contient les variables locales du programme

Les registres de segment

- Le registre **CS** (*code selector*) permet de retrouver l'adresse du segment de code dans la table des segments
- Le registre **DS** (*data selector*) permet de retrouver l'adresse du segment de données dans la table des segments
- Le registre **SS** (*stack selector*) permet de retrouver l'adresse du segment de pile dans la table des segments

- Ces registres indiquent en réalité **où dans la table des segments** on doit aller chercher l'adresse du segment correspondant
- D'autres registres de segment existent (ES,FS,GS)

Traduction de l'adresse logique en adresse linéaire

- Les adresses logiques sont en réalité des décalages (anglais *offset*) par rapport au début de leur segment
- Il faut donc rajouter à chaque adresse logique l'adresse du début du segment correspondant
- **Adresse linéaire = adresse du début du segment + adresse logique**
- Exemple :
  - `MOV [10], AX` signifie « mettre le contenu de AX à l'adresse 10 *du segment de données* »
  - On obtient l'adresse de début du segment de données grâce au registre DS et à la table des segments
  - On rajoute cette adresse à 10 pour obtenir l'adresse linéaire

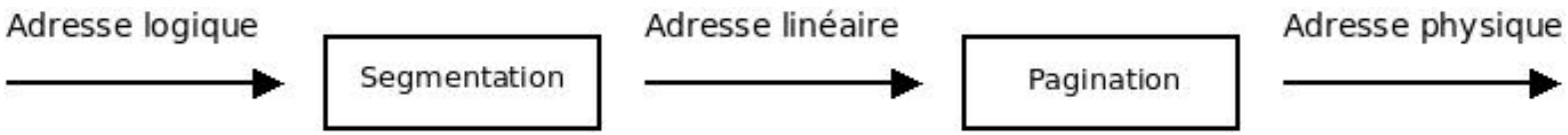
La pagination

- La pagination est un deuxième mécanisme de gestion de la mémoire offerts par le processeur Intel 80386
- Il consiste à découper les programmes en morceaux de taille égale appelés des *pages*
- Chaque page du programme sera ensuite placée en mémoire à un endroit différent
- Ainsi, un même segment peut être divisé en plusieurs pages, situées à des endroits différents de la mémoire
- Les adresses linéaires doivent donc elles-mêmes être traduites pour retrouver les **adresses physiques** !

La pagination (II)

- La pagination (contrairement à la segmentation) peut être désactivée dans le processeur 80386
- Si on n'emploie pas de pagination, les adresses linéaires et les adresses physiques sont égales
- Nous ne verrons pas en détail cette année le fonctionnement de la pagination
- La pagination est fortement utilisée par les systèmes d'exploitation modernes, dont Linux

Récapitulatif des étapes de traduction d'adresses



Le processeur et la traduction d'adresses

- Le processeur n'utilise que des adresses logiques dans ces registres (EIP, ...)
- Ces adresses doivent donc être traduites en adresses physiques (segmentation, pagination) avant d'accéder véritablement à la mémoire
- Cette traduction d'adresse se fait grâce à un puce particulière appelée **MMU** (*Memory Management Unit*, i.e. « Unité de Gestion de la Mémoire »)

Le processeur et la traduction d'adresses (II)

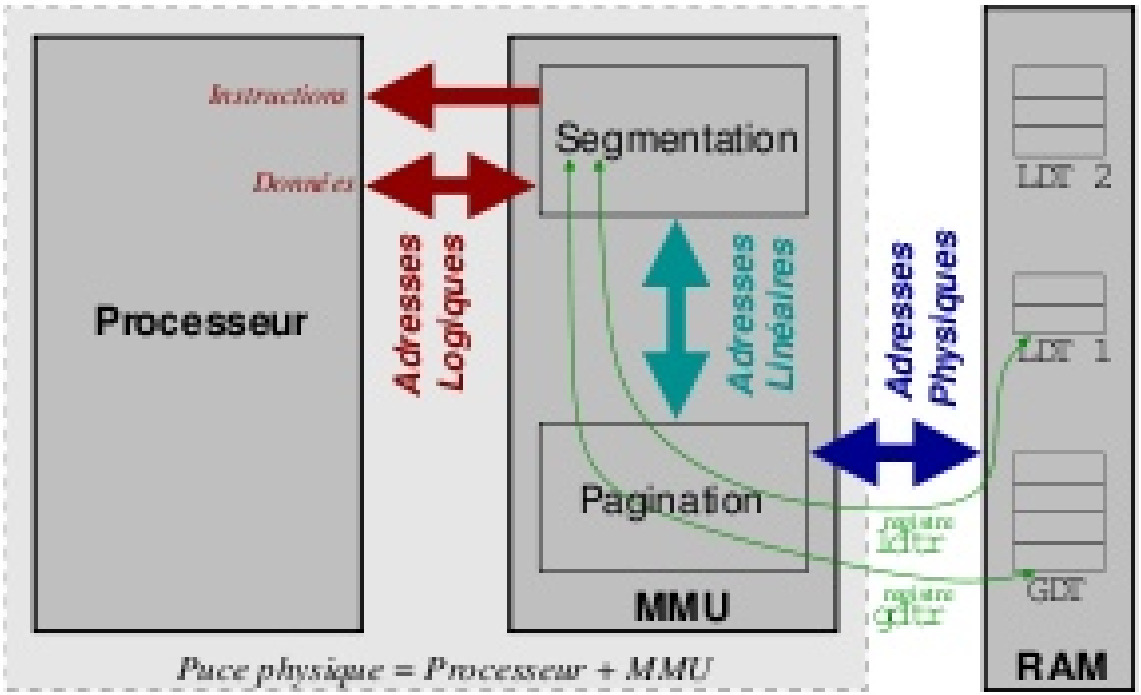
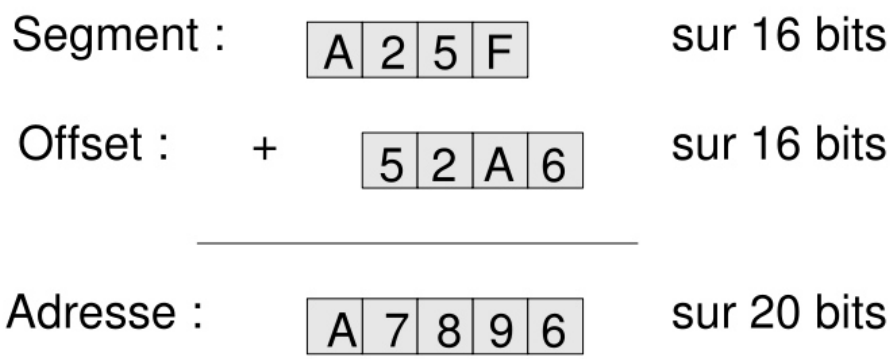


Figure : D'après « Croisière au cœur d'un OS, Etape 2 : segmentation et interruptions » (GNU Linux Magazine France numéro 63, Juillet/Aout 2004), disponible online à l'adresse `http://sos.enix.org/wiki-fr/upload/SOSDownload/sos-texte-art2.pdf`

Mode réel et segmentation

- En mode réel, il y a une segmentation, mais très simplifiée
- Il n'y a pas de pagination, donc la segmentation donne directement des adresses physiques
- Adresse physique = adresse de début du segment*16 + offset
- L'adresse de début du segment et l'offset sont sur 16 bits, ce qui donne une adresse totale sur 20 bits :



Mode réel et segmentation (II)

- En mode réel, la segmentation ne contient aucun mécanisme de protection de la mémoire
- Tout programme peut accéder à n'importe quelle adresse physique en choisissant un segment et un offset qui mène à cette adresse
- Il n'y a pas de table des segments
- Les registres CS, DS, SS sont associés au code, aux données, et à la pile, comme en mode protégé
- Les autres registres de segment (ES,FS,GS) sont utilisables librement (cf. page suivante)

Mode réel et segmentation (III)

- Exemple d'un programme en mode réel qui définit son propre segment grâce au registre ES, la segmentation ne contient aucun mécanisme de protection de la mémoire
- Ce programme ne fonctionne pas (« segmentation fault ») en mode protégé, car il faut être en ring 0 pour accéder aux adresses 0xB8000

```
mov AX, 0xB800          ; On donne la valeur B800
mov ES, AX              ; au registre de segment ES
mov AL , 'h'            ; on donne une valeur
mov AH,10010111b        ; arbitraire a AX
mov [ES:0xA0] , AX      ; que l'on place aux positions
                        ; B80A0 et B80A1 en memoire.
```

Mode interruptions

- Rappel : la **table des interruptions** est la table disant à quelle adresse se trouve la routine de gestion de l'interruption n° i
- Cette table se trouve dans la mémoire centrale, dans une zone réservée appartenant au système d'exploitation
- Selon qu'on est en mode réel ou protégé, cette table est gérée de façon différente

Mode réel et interruptions

- En mode réel, les adresses sont données par une paire `(segment : offset)`, où `segment` et `offset` font 2 bytes chacun (donc 4 bytes en tout)
- En mode réel, la table des interruptions est à l'adresse fixe 0000 : 0000 (i.e. l'adresse 0 dans le segment 0).
- L'entrée i de la table contient l'adresse de la routine de gestion de l'interruption n° i
- L'adresse de la routine de gestion de l'interruption i se trouve donc à l'adresse $4 * i$

Mode protégé et interruptions

- En mode protégé, la table des interruptions est à l'adresse donnée par le registre `IDTR`
- L'entrée i de la table contient l'adresse de la routine de gestion de l'interruption n° i
- En mode protégé, les adresses (`segment : offset`) ont 2 bytes pour le `segment` et 4 bytes pout l'`offset` (6 bytes en tout)
- Chaque entrée de la table des interruptions contient les 6 bytes de l'adresse de la routine de gestion, ainsi que 2 bytes supplémentaires contenant d'autres informations (donc 8 bytes en tout par entrée de la table)
- L'adresse de la routine de gestion de l'interruption i se trouve donc à l'adresse `[IDTR]+8*i`.

Référence

- Un article intéressant qui reprend en grande partie la matière de ce cours (pages 1 à 4) :
  - « Croisière au cœur d'un OS, Etape 2 : segmentation et interruptions » (GNU Linux Magazine France numéro 63, Juillet/Aout 2004)
  - Disponible online à l'adresse `http://sos.enix.org/wiki-fr/upload/SOSDownload/sos-texte-art2.pdf`