Chapitre 6

Le support du système d'exploitation

6.1 Les interruptions et les exceptions

Une *interruption* pour origine une cause externe au déroulement normal et séquentiel du programme. Si la prise en compte d'une interruption par le processeur ressemble à un appel de routine (instruction CALL du 80X86), une telle instruction est inscrite dans le programme et donc arrive toujours lorsque le déroulement du programme (le *processus*) est dans un état déterminé, alors qu'une interruption peut se produire dans n'importe quel état du processus : on dit qu'une interruption est *asynchrone*.

Le plus souvent, une interruption est provoquée par un dispositif d'entrée-sortie. Typiquement, par ce moyen le dispositif d'entrée-sortie attire l'attention du processeur pour initier un dialogue entre lui-même et le système d'exploitation (via le processeur). Par exemple, après que le système d'exploitation ait demandé l'écriture d'un secteur sur le disque à partir d'un tampon en mémoire centrale, cette écriture est prise en charge par le contrôleur du disque sans intervention du processeur. Ainsi, le processeur peut consacrer du temps à exécuter les instructions d'autres processus. Cependant, il doit pouvoir être informé de la terminaison de sa demande d'écriture. Le contrôleur envoie donc un signal au processeur (la dite *interruption*). Celui-ci exécutera alors la *routine de service* de l'interruption et le dialogue fait par cette routine permettra au système d'exploitation d'avoir le compte-rendu de la demande d'écriture.

De nombreux périphériques sont connectés à l'unité centrale. Leurs demandes d'interruption doivent être hiérarchisées en fonction de leur urgence. Cette urgence est dépendante du périphérique lui-même.

Il est possible qu'un périphérique fasse une demande d'interruption alors que le processeur est en train de servir (d'exécuter la *routine de service*) d'un autre périphérique. Il peut y avoir deux politiques ici, soit le processeur interrompt la routine de service qu'il est en train d'exécuter, soit il traitera la nouvelle interruption à la fin de la routine qu'il est en train d'exécuter. Dans tous les cas, lorsqu'une routine de service d'interruption est terminée, le processeur doit reprendre ce qu'il était en train de faire précédemment. Pour cela, une routine de service doit sauvegarder l'état du processeur : le compteur ordinal, le registre d'etat, les registres généraux, etc. Ainsi le processus interrompu ne s'apercevra pas que son exécution a été interrompu.

Les exceptions sont des événements engendrés par l'exécution du programme lui-même. Ce sont soit des événements anormaux (division par 0, accès à une adresse interdite, etc.), soit des événements programmés par une instruction (INT pour un 80X86), soit encore une combinaison des deux avec une instruction comme BOUND sur un 80X86 qui provoque une exception si la valeur testée n'est pas entre les deux bornes. Les exceptions sont synchrones par rapport aux programmes, dans ce sens, elles ressemblent beaucoup à des appels de routines. Cependant, les exemples cités font apparaître la différence de nature qui vient du caractère d'erreur à l'exécution, laquelle erreur est *piégée*.

Enfin, il peut exister un mode trace (pour le 80X86, ce mode est actif si le bit 8 (*TF : Trap Flag*) du registre d'état a la valeur 1) qui provoque une exception à la fin de chaque instruction. Un

Numéro de vecteur	Adresse mémoire	Affectation
0	000	Erreur de division
1	004	Aide à la mise au point
2	008	NMI
3	00C	Point d'arrêt
4	010	Instruction INTO
5	014	Instruction BOUND
6	018	Code opération invalide
7	01C	Coprocesseur non disponible
8	020	Double faute
9	024	Débordement de segment par le coprocesseur
10	028	TSS non valide
11	02C	Segment non présent
12	030	Faute sur la pile
13	034	Protection générale
14	038	Faute de page
16	040	Erreur coprocesseur

TAB. 6.1 – Vecteurs d'interruptions du 80386

tel mode permet de donner la main à un programme de supervision (un *dévermineur* (en anglais : debugger)) entre chaque instruction, ce qui permet de *tracer* un programme, c'est-à-dire de l'exécuter instruction par instruction, ou encore en *pas à pas*.

6.1.1 Le mode superviseur

Il existe deux modes d'exécution : le mode utilisateur et le mode superviseur. L'idée est que certaines actions ne doivent pouvoir être effectuées que lorsque le processeur est en mode superviseur, en train d'exécuter des instructions qui ont été soigneusement contrôlées, celles du système d'exploitation. Parmi ces instructions sensibles, on peut citer :

- passer en mode superviseur, ou en mode pas à pas,
- modifier le masque d'interruption (dans le registre d'état),
- accéder aux périphériques ou aux données du système d'exploitation.

Ainsi toutes les actions non autorisées vont provoquer l'exécution en mode superviseur de la routine associée à l'exception levée. Ces routines faisant partie du système d'exploitation, c'est lui qui reprend le contrôle. Cela permet d'assurer le protection du système d'exploitation et la protection inter-tâches pour un système multi-tâche.

6.1.2 Le mécanisme des interruptions et des exceptions

Nous examinons plus en détail ici le mécanisme des interruptions en étudiant l'implémentation du 8086 qui est représentatif des traitements réalisés par l'ensemble des processeurs.

A chaque source d'interruption ou d'exception est associé un numéro qui est utilisé par le processeur comme un index dans une table d'adresses des routines de services de ces différentes interruptions ou exceptions (chaque adresse de routine étant sur quatre octets, le numéro de vecteur est multiplié par quatre pour trouver l'adresse dans la table des adresses). L'adresse de la routine de service est parfois appelée un *vecteur*.

Le 8086 possède 256 vecteurs d'interruptions qui utilisent la mémoire entre les adresses 000000 et 0003FF, soit un kilo-octet. La table 6.1 donne l'affectation de ces différents vecteurs aux sources d'interruption et d'exception.

Une interruption arrive sur le processeur par l'intermédiaire d'une ou plusieurs connexions électriques, et donc autant de pattes affectées à cet usage sur le boîtier du processeur. En ce qui concerne

le 8086, deux pattes NMI, et INTR (cf. page ??). sont utilisées, la première pour les interruptions non masquables, et la deuxième pour les autres interruptions.

Lorsqu'une interruption arrive, celle-ci est asynchrone par rapport aux déroulement des instructions du programme en cours d'exécution. Le processeur termine toujours l'instruction en cours avant de traiter l'interruption. En fait, pour cela, il ne teste l'état de ces fils qu'entre l'exécution de deux instructions. De plus, une interruption sur la patte INTR n'est prise en compte que si le bit 9 (*IF : Interrupt Flag*) du registre d'état est à 1. Deux instructions permettent de positionner ce bit, `CLI` (*CLear Interrupt flag*) le met à zéro, et `STI` (*SeT Interrupt flag*) le met à un.

Lorsqu'une interruption est prise en compte, les actions suivantes sont effectuées :

1. le compteur ordinal et le registre d'état sont empilés,
2. le bit 8 du registre d'état reçoit la valeur 0 (TF $\leftarrow$ 0) : on supprime le mode *trace*,
3. le bit 9 du registre d'état reçoit la valeur 0 (IF $\leftarrow$ 0) : on masque les interruptions,
4. le processeur récupère le numéro de vecteur n (voir la section suivante pour le numéro de vecteur des interruptions),
5. le compteur ordinal reçoit la valeur du vecteur associé au numéro n (PC $\leftarrow$ mem$[n \times 4]$).

6.1.3 Le numéro de vecteur pour les interruptions

Lorsqu'une exception interne au processeur se produit (une division par zéro par exemple), sa qualité est connue par le processeur et donc aussi son numéro de vecteur, ainsi l'étape 4 de la section précédente ne pose pas de problème au processeur. Pour les interruptions, le processeur établit un dialogue préliminaire avec un circuit périphérique : le contrôleur d'interruptions. Dans ce dialogue, le processeur demande au contrôleur d'interruptions le numéro du vecteur d'interruption à considérer. Ce dernier lui répond sur le bus d'adresse ce numéro de vecteur.

6.1.4 Les exceptions programmées

Les exceptions programmées sont celles qui sont explicitement mises dans le programme en cours d'exécution, ce sont les instructions **INT** (exception inconditionnelle) ou **BOUND** (exception conditionnelle). Ces instructions font partie du jeu d'instructions du processeur parce que c'est le seul moyen pour un programme en mode utilisateur de passer en mode superviseur. De ce fait, ce passage se fait d'une façon contrôlée par le système d'exploitation, puisque c'est encore la routine de service de cette exception (qui fait partie du système d'exploitation) qui va être exécutée.

Les appels systèmes sont implantés grâce à un numéro de **INT** défini par l'implémentation du système d'exploitation. Ainsi sur un PC avec MS-DOS, l'appel du système se fait avec l'exception numéro 0x21, soit avec l'instruction `INT 33`. Les différentes fonctions du système d'exploitation sont différenciés par l'unique routine de service de l'exception grâce à un numéro de fonction passé sur la pile.

6.2 La gestion de la mémoire

Nous avons supposé jusqu'ici que les adresses émises par le processeur correspondaient exactement aux adresses dans la mémoire physique. Sous cette hypothèse, nous avons également vu que ceci impose des translations d'adresses au chargement des programmes pour que les sauts et les appels de routines soient faits aux adresses effectives en mémoire physique. De plus, pour un système multi-tâche, le problème de l'allocation mémoire est difficile et coûteux à cause de la *fragmentation*.

Pour éviter les coûteuses opérations de translation d'adresses et de compactage, et fournir d'autres services comme la protection des zones de mémoire allouées au système d'exploitation et des contrôleurs de périphériques, un schéma de conversion et de contrôle des adresses a été mis au point. Celui-ci est implanté sous la forme d'une *unité de gestion de mémoire* (UGM) (en anglais : Memory Management Unit, MMU). On parlera de :

- l'*espace d'adresses logique*, c'est l'espace adressé par le processeur et qui est à l'entrée de l'unité de gestion mémoire,
- l'*espace d'adresses physique*, c'est l'espace adressé sur la mémoire physique à la sortie de l'unité de gestion mémoire.

Une unité de gestion de mémoire se charge de convertir dynamiquement les adresses depuis l'espace d'adresses logique vers l'espace d'adresses physique et de contrôler que les adresses logiques sont dans des limites allouées.

6.2.1 La segmentation

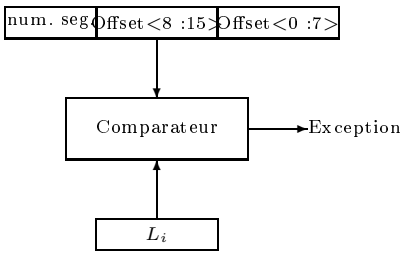
Nous prenons l'exemple de la table des descripteurs de segments de la MMU Z8010 pour la famille de processeur Z8000 de Zilog. Elle est composée de 64 registres de 32 bits. Chaque descripteur est lui-même composé de trois champs qui représentent l'adresse de base dans la mémoire physique du segment, sa longueur et les attributs du segment. L'adresse de base et la longueur sont mesurées dans une unité de 256 octets. Un segment est donc aligné sur une frontière divisible par 256.

31 16	15 8	7 0
Adresse de base 0	L0	A0
Adresse de base 1	L1	A1
⋮	⋮	⋮
Adresse de base 63	L63	A63

Un descripteur de segment est accédé dans cette table par un numéro de segment. Connaissant une adresse logique composée d'un numéro de segment et d'un décalage sur 16 bits, l'unité de gestion mémoire vérifie et traduit l'adresse logique en adresse physique. La vérification est faite sur deux points :

- comparaison de la partie haute du décalage avec la longueur du segment ;
- vérification des droits d'accès pour ce segment grâce au champ d'attributs.

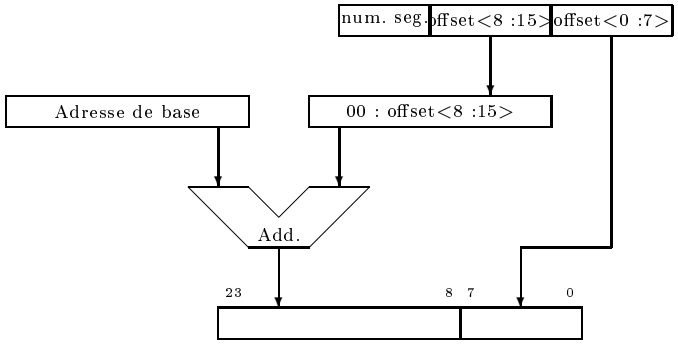
Une violation des droits d'accès ou un débordement par rapport à la longueur du segment provoque une exception.



Les huit attributs sont les suivants, certains sont positionnés par le processeur pour définir le type de segment, d'autres sont positionnés par l'unité de gestion de mémoire et pourront être lus par le système d'exploitation pour prendre certaines décisions — de remplacement par exemple :

REF bit 7, le segment a été référencé,
CHG bit 6, le segment a été modifié,
DIRW bit 5, autorisation d'interruption si le segment est accédé dans ses 256 dernières positions (utile pour gérer une pile),
DMA bit 4, accès interdit aux contrôleurs de DMA,
EXC bit 3, accès autorisé en exécution seulement, (le segment contient des instructions),
CPU bit 2, accès interdit à l'unité centrale,
SYS bit 1, accès réservé au mode superviseur,
RD bit 0, accès en lecture seulement.

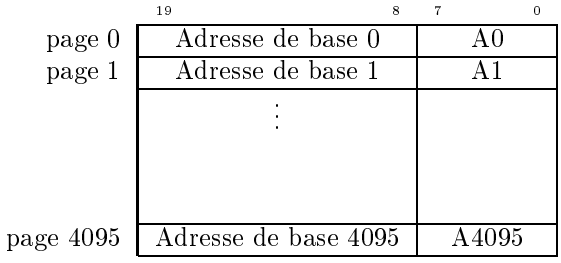
La traduction de l'adresse s'effectue selon le schéma suivant. L'additionneur est sur 16 bits. L'adresse de base est trouvée dans la table des descripteurs d'après le numéro de segment extrait de l'adresse logique.

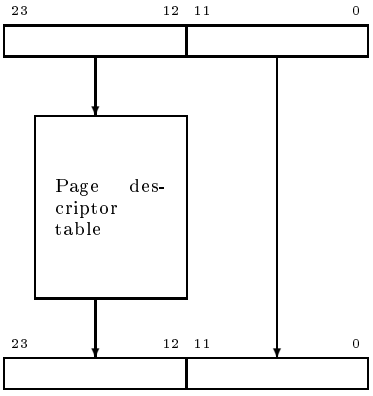


6.2.2 La pagination

Dans la méthode de pagination, toutes les pages ont la même longueur, il n'y a donc pas besoin de conserver une longueur dans un descripteur de page. En fait, une pagination simple (c'est-à-dire sans mémoire virtuelle) ne permet que de déplacer les pages logiques sur des pages physiques. Ce système ne résoud donc que le problème de la fragmentation externe, en créant éventuellement un peu de fragmentation interne.

Prenons l'exemple d'un processeur avec 24 lignes d'adresses et des pages de 4 kilo-octets. Il y a donc 4096 pages adressables et la table des pages doit avoir 4096 entrées. Une entrée dans cette table décrit la position physique en mémoire physique de la page, c'est donc un nombre sur 12 bits aussi. Tout comme pour la pagination, des attributs peuvent compléter ce champ, supposons que ces attributs soient codés sur 8 bits, la table des descripteurs a ainsi l'aspect suivant :





6.2.3 La mémoire virtuelle

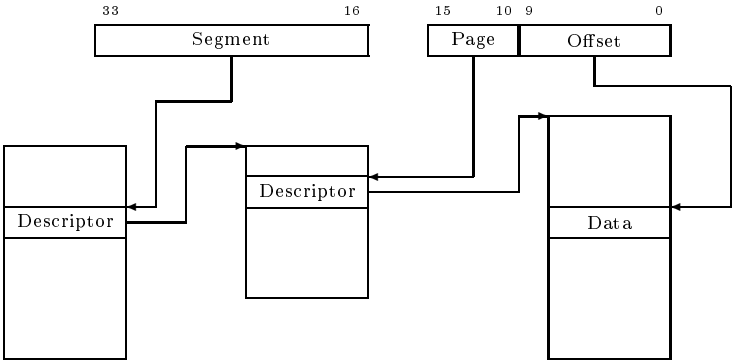
Si il y a moins de mémoire physique que ce qui est adressable par le processeur, on est conduit à implémenter une technique de mémoire virtuelle. La pagination se prête naturellement à cette

technique. En effet, il suffit d'un bit dans les attributs indiquant si la page est en mémoire centrale ou non. Si la page n'est pas en mémoire centrale (on parle de *défaut de page*), l'unité de gestion mémoire génère une exception qui est traitée par le système d'exploitation en chargeant la page manquante depuis la mémoire secondaire en mémoire centrale.

Le problème qui peut se poser cependant est dans le codage de la table des pages de la mémoire logique. En effet, pour un processeur récent, il n'est pas rare d'avoir 32 fils sur le bus d'adresses, ce qui fait un million (un peu plus 2^{20}) de pages de 4 kilo-octets. Si un descripteur occupe 4 octets, il faut 4 méga-octets pour conserver les descripteurs de pages, ...On ne peut donc plus conserver ces informations dans l'unité de gestion mémoire sous la forme d'un tableau. De plus, pour la plupart des processus, ce tableau serait quasiment vide. En fait, ce tableau est représenté d'une façon compacte en mémoire centrale, et seules quelques unes des entrées (celles qui ont été le plus récemment utilisées) sont dans l'unité de gestion mémoire : l'unité de gestion mémoire contient un cache sur l'ensemble de descripteurs. Lorsque le processeur présente une adresse logique à l'unité de gestion mémoire, cette dernière doit faire une recherche *associative* dans toutes ses entrées selon le numéro de page logique.

6.2.4 Pagination et segmentation multiniveaux

En fait, les deux techniques de pagination et de segmentation peuvent être combinées. Une adresse est alors composée de trois parties, le numéro de segment, le numéro de page et le déplacement dans la page. Cette technique permet d'avoir autant d'espaces d'adressage virtuel qu'il y a de segments. Nous donnons l'exemple de la conversion d'adresse dans le cas de MULTICS. Le numéro de segment est codé sur 18 bits, le numéro de page sur 6 bits, et les pages ont une taille de un kilo-octet.



Ce schéma peut être encore généralisé à n niveaux, et l'adresse doit être découpée en $n+1$ composantes. n de ces composantes servent d'index dans des tables de descripteurs des différents niveaux. En fait, seul le dernier niveau indexe un descripteur, les autres niveaux sont des répertoires hiérarchiques de descripteurs. La dernière composante est toujours un déplacement dans le bloc sélectionné au dernier niveau.

6.3 Exercices

Exercice 6.1 (Pagination) *On considère un programme qui lors de son exécution génère des accès à des pages virtuelles selon la séquence suivante (les pages sont regroupées quatre par quatre pour la commodité de lecture) :*

```
A B A C , A B A A , D B A E , A B A C , A A B D ,
B A A E , A B A C , A A D B , A B A E , A B A C
```

La gestion de la mémoire par pagination permet pour ce programme d'avoir simultanément 4 pages en mémoire physique, notées P_0, P_1, P_2, et P_3. Au démarrage aucune page de ce programme n'est présente en mémoire physique.

Question 6.1.1 *Indiquer dans le tableau 6.2 quelles sont les pages effectivement présentes en mémoire avec un algorithme de remplacement FIFO (* First In First Out*) — la colonne PF (*Page Faults*) indique les défauts de page.*

Question 6.1.2 *Que penser de la séquence d'accès suivante :*

A B C D E A B C D E A B C D E

Question 6.1.3 *Quelle variable logicielle permet d'implanter un FIFO dans une gestion de mémoire virtuelle paginée ? Quel dispositif matériel permet d'implanter un FIFO dans un cache mémoire ?*

Un algorithme LRU peut être implanté en maintenant une liste d'accès aux pages :

- *lors de chaque accès à une page, celle-ci est mise en tête de liste,*
- *lorsqu'un défaut de page se produit, c'est la page en fin de liste qui est copiée sur le disque, la page physique est alors libre pour le chargement de la nouvelle page, et cette page est mise en tête de liste.*

Question 6.1.4 *Dans le tableau 6.3, complétez chaque ligne, en indiquant quelles sont les pages virtuelles présentes dans les pages physiques P_0, P_1, P_2, et P_3, et quel est l'état de la liste d'accès aux pages physiques.*

Pour stocker cette liste, six bits sont associés à chaque page. Trois bits (le champ `suiv`*) vont permettre de noter le numéro de la page suivante dans la liste, et trois bits (le champ* `prec`*) le numéro de la page précédente dans la liste. Si le numéro est négatif, cela indique respectivement le dernier élément de la liste, et le premier élément de la liste. De plus, deux registres de deux bits* `deb` *et* `fin` *conservent le numéro de la page en début de liste et de la page en fin de liste. Ainsi la liste $P_0 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$ est conservée avec les descripteurs de pages sous la forme suivante :*

Numéro de page	*prec*	*suiv*	*descripteur*
00	*100*	*011*	...
01	*011*	*010*	
10	*001*	*100*	
11	*000*	*001*	

deb	*fin*
00	*10*

Question 6.1.5 *Imaginons que dans cet état un défaut de page se produise, où est chargée la nouvelle page ? Quelle est la nouvelle liste ? Quelle est sa représentation ?*

Numéro de page	*prec*	*suiv*	*descripteur*
00			...
01			
10			
11			

deb	*fin*

Question 6.1.6 *Donner l'algorithme qui doit être implanté lors du défaut de page pour maintenir la liste (On aura besoin d'un autre registre de deux bits* `tmp` *pour conserver le numéro de l'avant dernière page dans la liste initiale, ce registre sera donc initialisé à* `fin->prec`*.).*

Exercice 6.2 (Mémoire virtuelle) ***Question 6.2.1*** *S'il faut 1 microseconde pour exécuter une instruction et n autres microsecondes pour traiter un défaut de page, donnez la formule du temps d'exécution réel d'une instruction s'il se produit un défaut de page toutes les k instructions.*

Question 6.2.2 *Une machine a un espace d'adressage de 32 bits et des pages de 8Ko. La table des pages est entièrement gérée par le matériel qui possède un mot de 32 bits par entrée. Lorsqu'un processus est exécuté, la table des pages est recopiée dans le matériel à partir de la mémoire, un mot toutes les 100 nanosecondes. Si chaque processus s'exécute pendant 100 millisecondes (y compris le temps de chargement de la table des pages), quelle fraction du temps de la CPU est-elle consacrée au chargement de la table des pages ?*

		FIFO				
		P_0	P_1	P_2	P_3	PF
1	A	A				1
2	B	A	B			1
3	A	A	B			0
4	C	A	B	C		1
5	A	A	B	C		0
6	B	A	B	C		0
7	A	A	B	C		0
8	A	A	B	C		0
9	D	A	B	C	D	1
10	B	A	B	C	D	0
11	A	A	B	C	D	0
12	E	E	B	C	D	1
13	A	E	A	C	D	1
14	B	E	A	B	D	1
15	A	E	A	B	D	0
16	C	E	A	B	C	1
17	A	E	A	B	C	0
18	A	E	A	B	C	0
19	B	E	A	B	C	0
20	D	D	A	B	C	1
21	B	D	A	B	C	0
22	A	D	A	B	C	0
23	A	D	A	B	C	0
24	E	D	E	B	C	1
25	A	D	E	A	C	1
26	B					
27	A					
28	C					
29	A					
30	A					
31	D					
32	B					
33	A					
34	B					
35	A					
36	E					
37	A					
38	B					
39	A					
40	C					
Nombre total de PF						

TAB. 6.2 – Le cas de l'algorithme FIFO

		LRU					Liste
		P_0	P_1	P_2	P_3	PF	$P_3 \rightarrow P_2 \rightarrow P_1 \rightarrow P_0$
1	A	A				1	$P_0 \rightarrow P_3 \rightarrow P_2 \rightarrow P_1$
2	B	A	B			1	$P_1 \rightarrow P_0 \rightarrow P_3 \rightarrow P_2$
3	A	A	B			0	$P_0 \rightarrow P_1 \rightarrow P_3 \rightarrow P_2$
4	C	A	B	C		1	$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$
5	A	A	B	C		0	$P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$
6	B	A	B	C		0	$P_1 \rightarrow P_0 \rightarrow P_2 \rightarrow P_3$
7	A	A	B	C		0	$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$
8	A	A	B	C		0	$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$
9	D	A	B	C	D	1	$P_3 \rightarrow P_0 \rightarrow P_1 \rightarrow P_2$
10	B	A	B	C	D	0	$P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2$
11	A	A	B	C	D	0	$P_0 \rightarrow P_1 \rightarrow P_3 \rightarrow P_2$
12	E	A	B	E	D	1	$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$
13	A	A	B	E	D	0	$P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$
14	B	A	B	E	D	0	$P_1 \rightarrow P_0 \rightarrow P_2 \rightarrow P_3$
15	A	A	B	E	D	0	$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$
16	C	A	B	E	C	1	$P_3 \rightarrow P_0 \rightarrow P_1 \rightarrow P_2$
17	A	A	B	E	C	0	$P_0 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$
18	A	A	B	E	C	0	$P_0 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$
19	B	A	B	E	C	0	$P_1 \rightarrow P_0 \rightarrow P_3 \rightarrow P_2$
20	D	A	B	D	C	1	$P_2 \rightarrow P_1 \rightarrow P_0 \rightarrow P_3$
21	B	A	B	D	C	0	$P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_3$
22	A	A	B	D	C	0	$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$
23	A	A	B	D	C	0	$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$
24	E	A	B	D	E	1	$P_3 \rightarrow P_0 \rightarrow P_1 \rightarrow P_2$
25	A	A	B	D	E	0	$P_0 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$
26	B						
27	A						
28	C						
29	A						
30	A						
31	D						
32	B						
33	A						
34	B						
35	A						
36	E						
37	A						
38	B						
39	A						
40	C						
	Nombre total de PF						

TAB. 6.3 – Le cas de l'algorithme LRU

Real World, The n. :
1. In programming, those institutions at which programming may be used in the same sentence as FORTRAN, COBOL, RPG, IBM, etc.
2. To programmers, the location of non-programmers and activities not related to programming.
3. A universe in which the standard dress is shirt and tie and in which a person's working hours are defined as 9 to 5.
4. The location of the status quo.
5. Anywhere outside a university. "Poor fellow, he's left MIT and gone into the real world." Used pejoratively by those not in residence there. In conversation, talking of someone who has entered the real world is not unlike talking about a deceased person.

Imagine that Cray computer decides to make a personal computer. It has a 150 MHz processor, 200 megabytes of RAM, 1500 megabytes of disk storage, a screen resolution of 4096 x 4096 pixels, relies entirely on voice recognition for input, fits in your shirt pocket and costs $300. What's the first question that the computer community asks? "Is it PC compatible?"

Proposed Additions to the PDP-11 Instruction Set :

DC	Divide and Conquer
DMPK	Destroy Memory Protect Key
DO	Divide and Overflow
EMPC	Emulate Pocket Calculator
EPI	Execute Programmer Immediately
EROS	Erase Read Only Storage
EXCE	Execute Customer Engineer
HCF	Halt and Catch Fire
IBP	Insert Bug and Proceed
INSQSW	Insert into queue somewhere (for FINO queues [First in never out])
PBC	Print and Break Chain
PDSK	Punch Disk