Chapitre 1

Processeur, mémoire et bus

1.1 Les niveaux d'un système informatique

1.1.1 Vision logicielle

- langage de très haut niveau
- langage de haut niveau
- langage d'assemblage
- niveau du système d'exploitation
- langage machine (binaire)

Notions de *machine virtuelle*, de *compilation* et d'*interprétation*, de *système d'exploitation*.

1.1.2 Vision matérielle

- langage machine (binaire)
- processeur, mémoire
- unité de contrôle, unité arithmétique et logique
- portes élémentaires
- transistors
- ...

1.1.3 Vision informationelle

- base de données réparties
- base de données
- fichiers, piles, files
- enregistrements, tableaux
- entiers, flottants, caractères
- mots
- bits

1.2 Architecture de Von Neumann

Von Neumann en 1946 a tracé les principales voies sur lesquelles baser un calculateur électronique. La caractéristique essentielle qui définit le modèle de la machine de Von Neumann est la *procéduralité* — ce qui signifie que tout problème doit être décrit au calculateur sous la forme d'une séquence d'opérations à exécuter : le *programme*.

On notera tout de suite la différence entre *programme* et *processus* : un processus est la combinaison d'un *processeur* et d'un *programme*, le processeur exécutant le programme instruction par

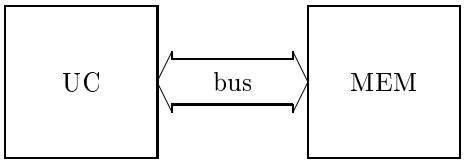

FIG. 1.1 – L'architecture de Von Neumann

instruction. Lors de l'exécution, le texte du programme et l'état des données du processus associé sont conservés dans la *mémoire*. La mémoire contient donc programmes et données.

Les deux composants de l'architecture de Von Neumann sont le *processeur* et la *mémoire* ; ils sont reliés par un *bus* — un bus est essentiellement un ensemble de fils parallèles (voir figure 1.1). La mémoire est un tableau de cellules ; une cellule est un paquet de bits et toutes les cellules ont le même nombre de bits. Ce tableau est caractérisé par deux données :

- le nombre de cellules,
- la taille d'une cellule.

Une cellule est la plus petite quantité de mémoire directement adressable. Chaque cellule a sa propre adresse, deux cellules adjacentes ont des adresses qui diffèrent de 1.

1.2.1 Le processeur

Le processeur est un *interprète* d'un certain langage défini par un ensemble d'instructions. On lui donne aussi le nom d'*unité centrale* (UC) (en anglais : Central Processing Unit, CPU). Pour interpréter une instruction, le processeur doit successivement :

- charger et décoder l'instruction (phase *fetch*),
- l'exécuter (phase *execute*),
- identifier la prochaine instruction à charger et exécuter.

1.2.1.1 Les registres

Le compteur ordinal Entre l'exécution de deux instructions, un processus est entièrement décrit par son état. Cet état comprend en particulier la valeur du *compteur ordinal* (en anglais : program counter), qui indique quelle est la prochaine instruction à exécuter. L'exécution d'une instruction modifie cet état (entre autre le compteur ordinal). Ce compteur ordinal est dans le processeur : c'est un *registre*.

Un *registre* n'est rien d'autre qu'un morceau de mémoire qui est à l'intérieur du processeur. Pour des raisons d'efficacité et de facilité de programmation, en plus du compteur ordinal, il existe d'autres registres. Certains sont accessibles au programmeur, ce sont les registres généraux, le registre d'état, et quelques autres.

Le registre d'état Ce registre contient des *indicateurs* (en anglais : flags) sur le résultat de la dernière opération exécutée. Par exemple, l'indicateur ZF (comme *Zero Flag*) est positionné à un si le résultat de la dernière opération était nul, un autre indiquera un résultat positif... (cf. fig. 1.4).

L'accumulateur et les registres généraux Il y a toujours au moins un registre général dont le contenu peut être utilisé comme l'un des opérandes d'une opération dyadique, ou comme l'unique opérande d'une opération monadique. Dans les processeurs actuels, les registres généraux sont en assez grand nombre (entre 10 et 100). Dans les premiers processeurs, il n'y en avait qu'un seul et on l'appelait alors l'*accumulateur*.

L'unité arithmétique et logique L'*unité arithmétique et logique* (UAL) (en anglais : Arithmetical and Logical Unit, ALU) est celle qui est effectivement chargée de calculer le résultat des

opérations comme les additions, multiplications, décalages, opérations booléennes,... par contre elle ne s'occupe pas d'aller chercher les opérandes.

1.2.1.2 La notion de *mot*

Au niveau du processeur, on parle de *mot* (en anglais : word). Selon le contexte, ce *mot* peut désigner l'unité d'information transférée entre les différents composants de l'ordinateur (essentiellement le processeur et la mémoire centrale), mais le plus souvent le *mot* est l'objet que sait basiquement traiter le processeur (pour une addition par exemple). Les choix faits à ce niveau ont été très variables : 1, 8, 12, 16, 18, 24, 27, 32, 36, 48, 60 ou 64 bits par mots. De plus, maintenant, les processeurs savent travailler sur plusieurs longueurs de mots. Ainsi la signification du mot "mot" n'est pas très claire maintenant.

Pour le 80386 et ses sucesseurs, le *mot* est un mot de 32 bits. Cependant ce processeur sait aussi travailler sur des mots de 8 bits ou de 16 bits.

Remarque La taille des mots sur lesquels savent travailler les processeurs actuels est un multiple de 8 : voir la colonne "Opérations" dans le tableau 1.1. De même la taille de la *cellule* mémoire des processeurs actuels (et aussi de tous ceux qui sont cités dans le tableau 1.1) est de 8 bits.

Constructeur	Année	Microprocesseur	Opérations	Bus de données	Bus d'adresse
Intel	1971	4004	4		
Intel	1972	8008	8	8	16
Intel	1974	8080	8	8	16
Zilog	1976	Z80	16-8	8	16
Motorola	1979	68000	32-16-8	16	24
Motorola	1984	68020	32-16-8	32	32
Motorola	1987	68030	32-16-8	32	32
Motorola	1990	68040	32-16-8	32	32
Motorola		68060	32-16-8	32	32
Intel	1978	8088	16-8	8	20
Intel	1978	8086	16-8	16	20
Intel	1982	80186	16-8	16	20
Intel	1982	80286	16-8	16	24
Intel	1985	80386	32-16-8	32	32
Intel	1989	80486	32-16-8	32	32
Intel	1993	Pentium	32-16-8	32	32
Intel	1995	Pentium Pro	32-16-8	32	32

TAB. 1.1 – Quelques microprocesseurs

1.2.2 Un exemple de processeur : le 80386

1.2.2.1 Le brochage du processeur 80386

La figure 1.2 montre le brochage du processeur Intel 80386. Sur les 132 broches du boîtier de type *pin grid array*, 124 sont utilisés pour les différents signaux. Le processeur est implémenté en technologie CMOS, il contient 275 000 transistors sur une surface de $1cm^2$.

La figure 1.3 montre l'organisation logique de ces signaux. Les éléments essentiels y sont le *bus d'adresse* (A2–A31), le *bus de données* (D0–D31) et le *bus de contrôle* qui est composé des autres connexions à l'exception des alimentations électriques et de l'entrée de l'horloge.

Alimentation

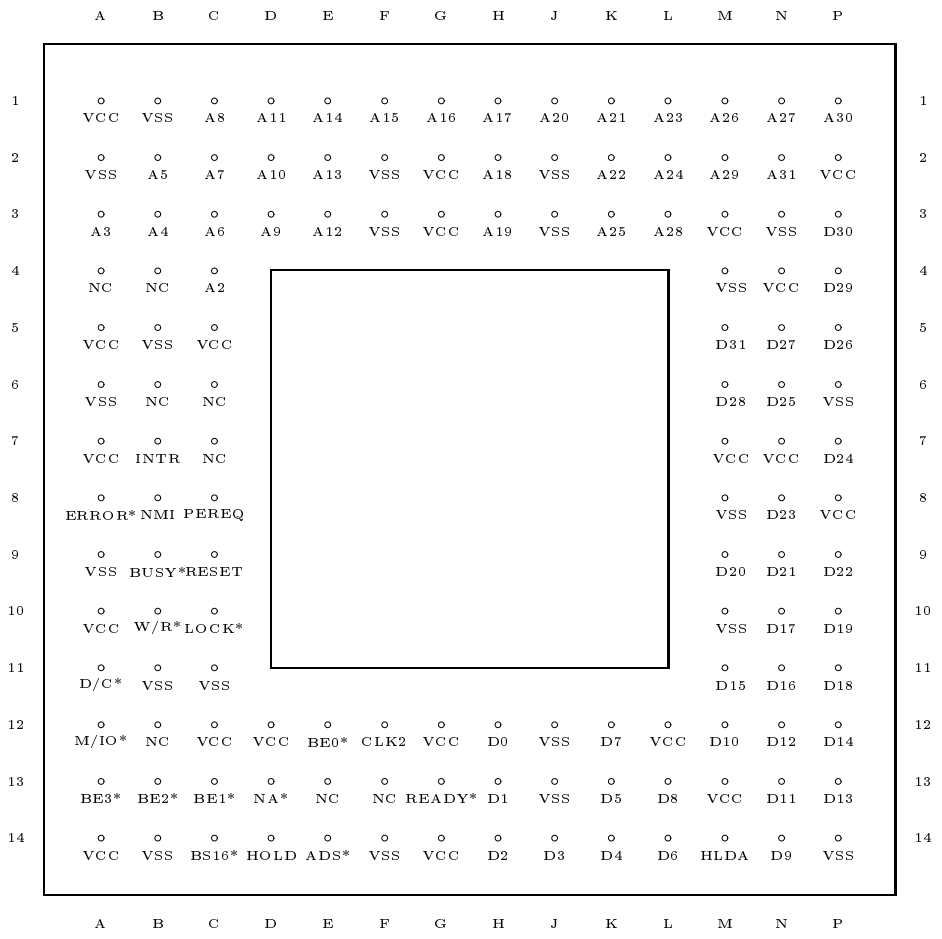

FIG. 1.2 – Brochage du 80386

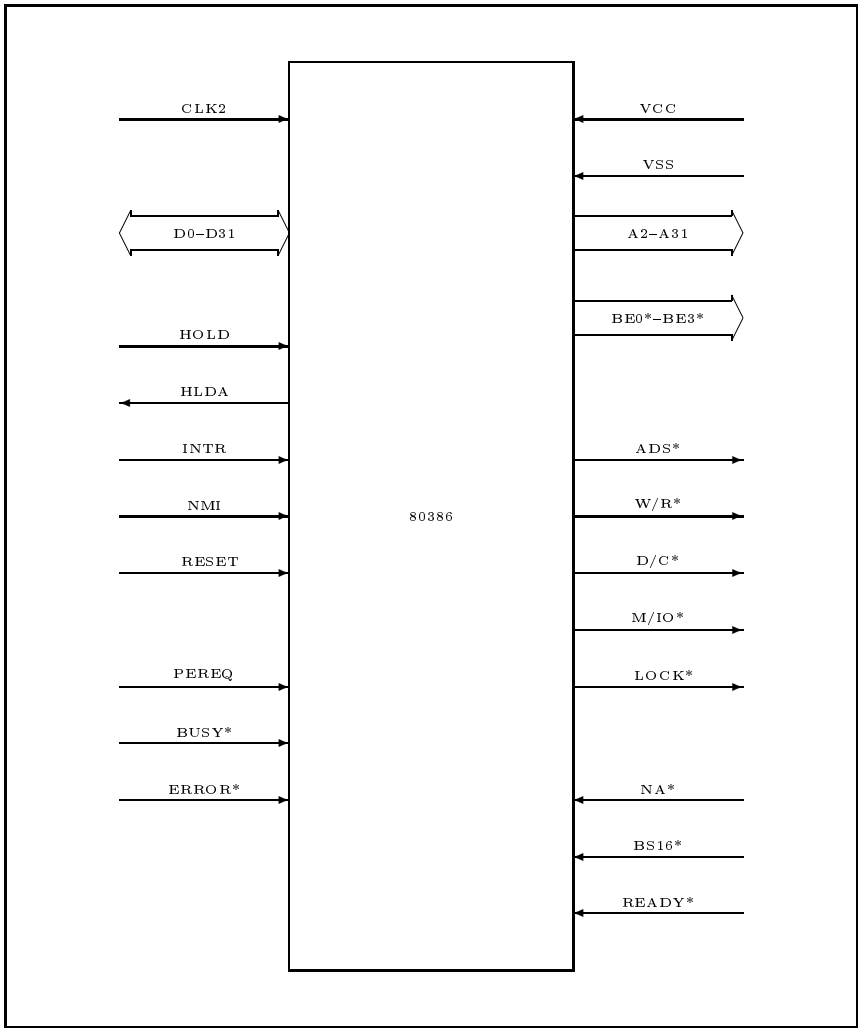

FIG. 1.3 – Signaux du 80386

	Introduction Date	Clock Speeds	Bus Width	Number of Transistors	Addressable Memory	Virtual Memory	Brief Description
4004	11/15/71	108 KHz	4 bits	2,300 (10 microns)	640 bytes		First microcomputer chip, Arithmetic manipulation
8008	4/1/72	108 KHz	8 bits	3,500	16 KBytes		Data/character manipulation
8080	4/1/74	2 MHz	8 bits	6,000 (6 microns)	64 KBytes		10× the performance of the 8008
8086	6/8/78	5 MHz 8 MHz 10 MHz	16 bits	29,000 (3 microns)	1 Megabyte		10× the performance of the 8080
8088	6/1/79	5 MHz 8 MHz	8 bits	29,000 (3 microns)			Identical to 8086 except for its 8-bit external bus
80286	2/1/82	8 MHz 10 MHz 12 MHz	16 bits	134,000 (1.5 microns)	16 Megabytes	1 gigabyte	3–6× the performance of the 8086
Intel386 DX	10/17/85	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	275,000 (1 micron)	4 gigabytes	64 terabytes	First X86 chip to handle 32-bit data sets
Intel386 SX	6/16/88	16 MHz 20 MHz	16 bits	275,000 (1 micron)	4 gigabytes	64 terabytes	16-bit address bus enabled low-cost 32-bit processing
Intel486 DX	4/10/89	25 MHz 33 MHz 50 MHz	32 bits	1,200,000 (1 micron, .8 micron with 50 MHz)	4 gigabytes	64 terabytes	Level 1 cache on chip
Intel486 SX	4/22/91	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	1,185,000 (.8 micron)	4 gigabytes	64 terabytes	identical in design to Intel486 DX but without math coprocessor
Pentium	3/22/93	60MHz 66MHz 75MHz 90MHz 100MHz 120MHz 133MHz 150MHz 166MHz	32 bits	3.1 million (.8 micron)	4 gigabytes	64 terabytes	superscaler architecture brought 5× the performance of the 33-MHz Intel486 DX processor
Pentium Pro	3/27/95	150MHz 180MHz 200MHz	32 bits	5.5 million (.32 micron)	4 gigabytes	64 terabytes	dynamic execution architecture drives high-performing processor

TAB. 1.2 – L'histoire de la famille X86 de Intel

VCC 5 volts.

VSS 0 volts.

Horloge

CLK2 Entrée du signal d'horloge.

Signal d'initialisation

RESET (entrée)

Bus de données

D0-D31 (Data) Bus de données.

Bus d'adresses La capacité d'adressage est de 4 giga octets (2^{32}) de 0x0000 0000 à 0xFFFF FFFF. L'espace d'adressage des entrées/sorties est de 64 kilo octets de 0x0000 0000 à 0x0000 FFFF. L'accès au coprocesseur se fait aux adresses 0x8000 00F8 à 0x8000 00FF.

A2-A31 (Address) Il n'y a pas de broche **A0** ni **A1**, les accès à un octet ou à un mot de 16 bits utilisent les signaux **BE0***, **BE1***, **BE2***, et **BE3***.

Signaux de définition des cycles bus

W/R* En écriture ce signal est positionné à la valeur haute, et en lecture à la valeur basse.

M/IO* Ce signal permet de distinguer les accès mémoire des accès entrées/sorties.

D/C* Permet de distinguer les cycles bus de contrôle des cycles bus de données.

LOCK* Permet au processeur de se réserver l'accès aux bus dans le cycle suivant. Ce signal est utile dans un cadre multiprocesseur.

Signaux de contrôle des bus

READY* (entrée) Indique la fin d'un cycle bus. Lors d'une lecture, la mémoire indique par ce signal qu'elle a fourni la donnée demandée. Le processeur verrouille alors les données du bus de données, et termine le cycle de lecture.

Lors d'une écriture, la mémoire indique par ce signal qu'elle a verrouillé les données. Le processeur relâche alors le bus de données et termine le cycle d'écriture.

ADS* (sortie) (ADress Status) Par cette sortie le processeur indique que le bus d'adresses est valide.

NA* (entrée) (Next Adress)

BS16* (entrée) (Bus Size)

Signaux de demande d'interruptions

INTR (entrée) (INTerrupt Request)

NMI (entrée) (Non Masquable Interrupt request)

Signaux de demande et de reconnaissance de prise de bus

HOLD (entrée) Signal de demande de mise en haute impédance des lignes D0–D31, A2–A31, BE0*–BE3*, W/R*, D/C*, M/IO*, LOCK*, ADS*.

HLDA (sortie) Signal de reconnaissance de **HOLD**.

Signaux de contrôle du coprocesseur

PEREQ (entrée) (Processor Extension REQuest) Par ce signal le coprocesseur indique qu'il désire faire une transaction mémoire.

BUSY* (entrée) Par ce signal le coprocesseur indique qu'il est en train d'effectuer une instruction et ne peut en accepter une autre.

ERROR* (entrée) Par ce signal le coprocesseur indique qu'il a découvert une condition exceptionnelle (division par zéro par exemple) non masquée.

1.2.2.2 L'architecture interne du processeur 80386

La figure 1.5 montre une partie des registres internes du processeur 80386.

Registres généraux Le processeur 80386 possède huit registres généraux de 32 bits.

EAX, EBX, ECX, EDX Ces quatre registres peuvent être considérés de quatre façons différentes. Ainsi :

- AL désigne les bits 0 à 7 du registre EAX.
- AH désigne les bits 8 à 15 du registre EAX.
- AX désigne les bits 0 à 15 du registre EAX.
- EAX désigne naturellement les bits 0 à 31 du registre EAX.

De même pour EBX, ECX, et EDX.

EAX (Accumulator), EDX (Data) interviennent pour les instructions de multiplication, de division ou d'entrées-sorties.

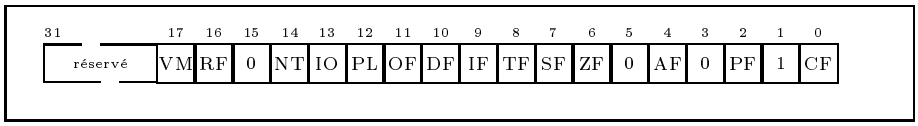

FIG. 1.4 – Registre des indicateurs du 80386

EBX, EBP (Base) sont souvent utilisés pour contenir l'adresse de base d'une structure de données en mémoire.

ECX (Count) est utilisé pour les instructions de boucle, de décalage ou de répétition.

ESI (Source Index), EDI (Destination Index) contiennent souvent un index qui s'incrémente au fur et à mesure de l'examen d'une structure de données.

ESP (Stack Pointer) est le pointeur de pile.

Registres de segments

CS (Code Segment) désigne le segment de code.

SS (Stack Segment) désigne le segment de la pile.

DS (Data Segment) désigne le segment de données.

ES, FS, GS (Extra Segment)

Registre des indicateurs EFLAGS Dans ce registre, chacun des bits possède sa propre signification, la figure 1.4 montre les noms des différents bits de ce registre. Nous indiquons ici la signification de quelques uns de ces indicateurs :

CF (Carry Flag) Bit de retenue.

PF (Parity Flag) Bit de parité, indique si les 8 bits de poids faible du résultat contiennent un nombre pair de 1.

ZF (Zero Flag) est positionné à 1 lors d'un résultat nul.

SF (Sign Flag) Copie du bit de poids fort du résultat.

OF (Overflow Flag) Bit de débordement en arithmétique signée.

Pointeur d'instruction EIP Le pointeur d'instruction contient l'adresse de la prochaine instruction à exécuter en séquence.

1.2.3 La mémoire

Tout support de conservation de l'information porte le nom de *mémoire*. On peut grossièrement classer la mémoire en trois catégories :

- la mémoire interne au processeur : les registres,
- la mémoire directement accessible par les bus d'adresses et de données du processeur : la mémoire centrale,
- les autres supports de mémoire accessibles par des opérations d'entrées/sorties : la *mémoire de masse* (disques, disquettes, bandes, cartouches, etc.).

Ces différentes sortes de mémoire sont très disparates sous plusieurs aspects :

- du niveau du logiciel qui les gère,
- de la rapidité (du débit),
- de la capacité.

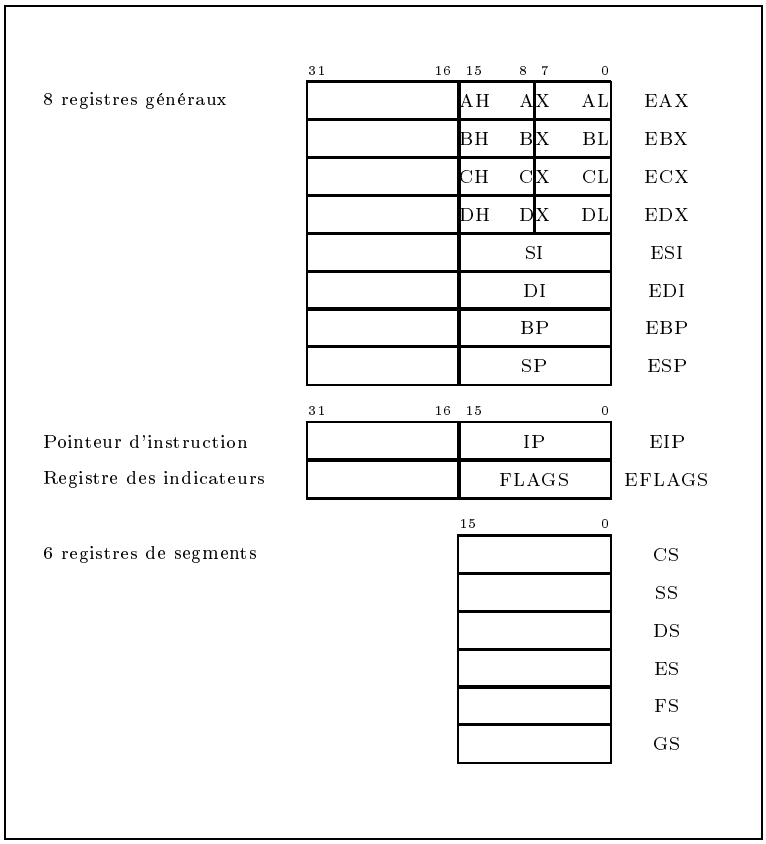

FIG. 1.5 – Architecture simplifiée du 80386

Lorsqu'on parle de *la mémoire* d'un ordinateur sans plus de précision, il s'agit de sa mémoire centrale. C'est de cette mémoire dont nous parlons dans cette section.

L'unité élémentaire de stockage et d'information est le *bit*. Mais les bits sont toujours regroupés en unités de plus haut niveau. Pour la mémoire centrale, le regroupement se fait sous forme de cellules. Nous avons déjà indiqué que la cellule est le plus petit morceau de mémoire adressable, dans le sens où si on veut écrire un programme qui ne modifie que trois bits dans une cellule, ce programme devra comporter une instruction qui charge dans un registre le contenu de la cellule en question, modifie les bits comme voulu, puis écrive toute la cellule en mémoire centrale.

Remarque Les cellules peuvent être elles-mêmes regroupées de différentes façons par différentes couches du logiciel ou du matériel. A chaque niveau de l'organisation, il doit y avoir une relation entre les tailles des informations traitées et les tailles des informations traitées par les niveaux voisins.

Certain regroupements de bits ont des noms :

- *Quartet* (en anglais : Nibble, Half-Byte), formé de quatre bits,
- *Octet* (en anglais : Byte), formé de huit bits.
- *Mot de n bits* formé de n bits.

1.2.3.1 Aspect logique : l'organisation de la mémoire centrale

D'un point de vue logique, par rapport à un aspect physique, la mémoire centrale est un tableau de cellules. Pour un processeur qui utilise des cellules de 8 bits, et a un bus d'adresses de 32 bits — donc peut adresser 2^{32} cellules — ce tableau pourrait être déclaré en Pascal sous la forme :

```
const WORD_LENGTH = 8;
      MEM_LENGTH  = 4294967296;
type bit = boolean;
     word = array [0..WORD_LENGTH-1] of bit;
var  mem  = array [0..MEM_LENGTH-1] of word;
```

ce qui s'écrirait en langage C sous la forme :

```
#define WORD_LENGTH  8
#define MEM_LENGTH   4294967296
typedef int bit;
typedef bit word[WORD_LENGTH];
word mem[MEM_LENGTH];
```

Dans la version en langage C, chaque `bit` ne peut prendre que deux valeurs *vraie* ou *fausse*, 0 ou 1.

> **Remarque** Cette définition insiste sur le fait qu'en aucune façon il n'y a une interprétation du mot mémoire comme un nombre.

Pour les microprocesseurs actuels, pour lesquels la notion de mot n'est pas très claire, et qui travaillent aussi bien sur des données de 1, 2, ou 4 (voire 8) octets, il faut voir la mémoire centrale à la fois :

- comme un tableau de N octets numérotés 0, 1, 2, 3, ..., $N-2$, $N-1$;
- comme un tableau de $N/2$ mots de 2 octets numérotés 0, 2, 4, 6, ..., $N-4$, $N-2$;
- comme un tableau de $N/4$ mots de 4 octets numérotés 0, 4, 8, 12, ..., $N-8$, $N-4$;

Lorqu'on parle d'un mot de plusieurs octets qui réside en mémoire, en fait il s'agit du contenu de plusieurs cellules consécutives. L'adresse du mot est l'adresse du premier octet de ce mot.

Alignement Un mot de deux octets ne peut être qu'à une adresse paire, et un mot de quatre octets ne peut être qu'à une adresse divisible par quatre. Cette contrainte porte le nom d'*alignement* des données.

Ordre des octets Les différents octets qui composent un mot peuvent être rangés dans la mémoire de différentes façons à partir de l'adresse de départ (l'adresse du mot). Les deux arrangements les plus naturels sont ceux dénommés *grand boutiste* (en anglais : big endian)et *petit boutiste* (en anglais : little endian)

Big endian poids forts aux adresses faibles ;

Little endian poids faibles aux adresses faibles.

Prenons deux exemples de processeurs 16 bits. Avec un 68000 qui utilise le mode *grand boutiste*, le mot `0x0102`[1], stocké en mémoire à l'adresse `a` occupe les octets d'adresses `a` et `a+1` comme suit :

adresse	contenu
a	0x01
a+1	0x02

alors qu'avec un processeur 8086 qui utilise le mode *petit boutiste*, ce même mot à la même adresse occupe les mêmes octets, mais de la façon suivante :

adresse	contenu
a	0x02
a+1	0x01

De même pour le mot de 32 bits `0x01020304` :

	68000	8086
a	0x01	0x04
a+1	0x02	0x03
a+2	0x03	0x02
a+3	0x04	0x01

1.2.3.2 Aspect physique : les boîtiers de mémoire

Physiquement, différentes caractéristiques peuvent être attribuées à la mémoire :

volatilité Une mémoire volatile perd son contenu (sa mémoire !) lorsqu'elle n'a plus d'alimentation électrique.

lecture/écriture Il y a des mémoire pour lequel l'accès peut se faire en lecture ou en écriture : les *mémoires vives* (en anglais : Random Access Memory, RAM)[2] et les mémoires à lecture seule : les *mémoires mortes* (en anglais : Read Only Memory, ROM).

temps d'accès

capacité

organisation La même capacité, disons 16 kilobits, peut être organisée de différentes façons par exemple : 16384×1 ou 4096×4 ou 2048×8 ou ...

statique/dynamique Il s'agit de technologies différentes, les mémoires statiques sont réalisées avec une bascule bistable (donc deux transistors) pour chaque bit, les mémoires dynamiques sont réalisées avec un condensateur et un transistor pour chaque bit. Les deuxièmes sont plus compactes, on peut avoir une plus grande capacité sur la même surface de silicium, par contre, elles nécessitent d'être *rafraichies* très fréquemment : le condensateur se décharge et perd donc la mémoire !

Quelques noms :

- RAM : (en anglais : *Random Access Memory*) la mémoire vive,
- ROM : (en anglais : *Read Only Memory*) la mémoire morte programmée une fois pour toute en usine,
- PROM : (en anglais : *Programmable Read Only Memory*) la mémoire morte programmable une fois et une seule avec un appareil spécial (remplace la ROM, dans le cas de petites quantités),

[1] Nous utilisons ici, informellement, la notation `0x....` qui dénote un nombre hexadécimal. Cette notation est explicitée au chapitre 2.

[2] Le nom anglais RAM n'évoque pas du tout la caractéristique principale de la RAM par rapport à la ROM. Il évoque plutôt la différence entre de la mémoire pour laquelle on peut accéder à chaque cellule indépendamment d'autres accès précédemment effectués sur cette RAM (donc accès *aléatoire*), et de la mémoire à accès séquentiel.

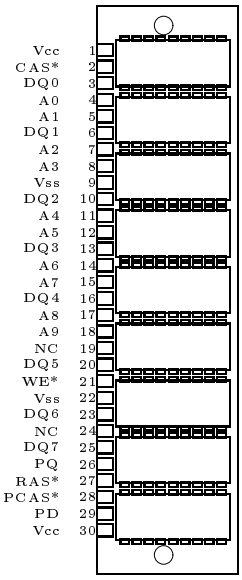

FIG. 1.6 – Brochage d'une barette SIM

- EPROM : (en anglais : *Erasable Programmable Read Only Memory*) on peut effacer le contenu en exposant le boîtier à des rayons ultraviolets pendant quelques minutes puis reprogrammer le contenu (phases de développement et de tests),
- EEPROM (en anglais : *Electrically Erasable Programmable Read Only Memory*) ça ressemble de plus en plus à de la RAM non volatile, mais les tensions et les courants nécessaires à l'écriture ne sont pas ceux habituellement utilisés pour les autres boîtiers.
- NOVRAM : (en anglais : *Non Volatile Random Access Memory*) c'est de la mémoire vive non volatile, ça peut se réaliser par exemple en combinant de la RAM statique avec une source d'énergie électrique.

Le plus souvent les boîtiers de mémoire sont montés sur des cartes normalisés (modules SIM) (voir figure 1.6) qui regroupent sur un petit circuit imprimé 8 (ou 9) boîtiers de mémoire dynamique. Chacun des boîtiers étant d'une capacité de $4M \times 1$, $16M \times 1$, $32M \times 1$, $64M \times 1$, ou $128M \times 1$. Une barette SIM a donc selon le cas une capacité de 4 à 128 méga-octets.

1.2.4 Les bus d'adresses et de données

Le choix de la "forme" du tableau mémoire (nombre de cellules et taille d'une cellule) se retrouve dans le *bus* de communication entre le processeur et la mémoire centrale au niveau :

- du bus d'adresses : sa largeur fixe le nombre maximum d'adresses possibles, donc le nombre de cellules adressables, et donc le nombre de cellules du tableau ;
- du bus de données : pour les premiers processeurs, sa largeur est le nombre de bits d'une cellule mémoire, ainsi ces processeurs travaillent exactement sur une cellule lors d'un accès mémoire.

Remarque Aujourd'hui la (quasi ?) totalité des processeurs utilisent des cellules mémoire de 8 bits. Leurs bus de données ont une largeur de 16 ou 32 (voire 64 bits), pour pouvoir travailler sur 2 ou 4 (voire 8) cellules lors de chaque accès mémoire.

1.2.4.1 Le protocole de dialogue avec la mémoire

Prenons l'exemple du protocole entre un 80386 qui a 32 bits d'adresses et 32 bits de données avec sa mémoire. Les protocoles de lecture et d'écriture entre le processeur et sa mémoire sont illustrés par la figure 1.3.

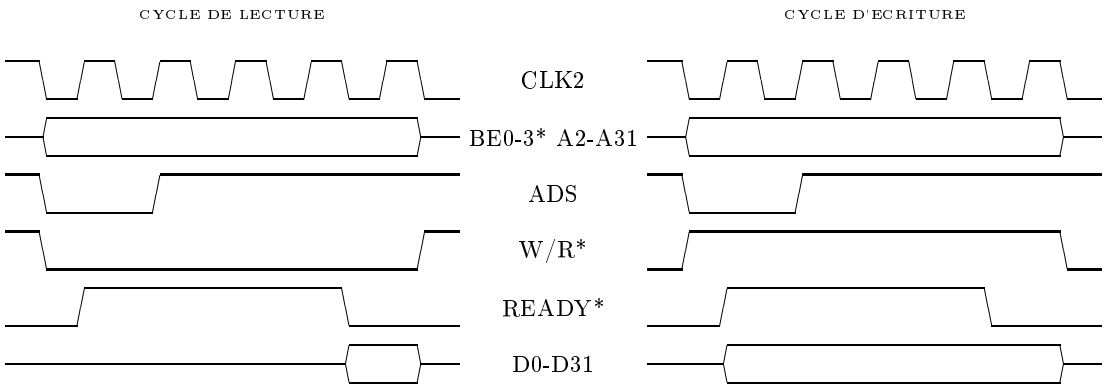

FIG. 1.7 – Protocole des cycles de lecture et d'écriture avec un 80386

Protocole de lecture

1. Le processeur dépose l'adresse sur le bus d'adresses **A1–A31** et **BE0*–BE3***, et met le signal **W/R*** à l'état bas pour demander une lecture.
2. La sortie **ADS*** est validée, ainsi la mémoire sait que l'état électrique présent sur le bus d'adresses désigne effectivement une adresse valide et que c'est une lecture qui est demandée.
3. Le processeur relâche le signal **ADS*** et attend la validation du signal **READY***, c'est la mémoire qui positionne ce signal après avoir positionné le bus de données **D0–D31** avec les valeurs lues dans la cellule accédée.
4. Le processeur échantillonne le bus de données **D0–D31** et relâche le bus d'adresses.

Protocole d'écriture

1. Le processeur dépose l'adresse sur le bus d'adresses **A1–A31** et **BE0*–BE3***.
2. La sortie **ADS*** est validée et le signal **W/R*** est positionné à un pour informer la mémoire d'une demande d'écriture.
3. Le processeur dépose la donnée sur le bus de données **D0–D31**.
4. Le processeur relâche les signaux **ADS***, et attend la validation du signal **READY***, c'est la mémoire qui positionne ce signal lorqu'elle a effectivement enregistré la modification mémoire.
5. Le processeur relâche les bus d'adresses et de données.

1.3 Exercices

Exercice 1.1 (Exécuter un programme) *Vous êtes le processeur et votre compteur ordinal contient 0...Le contenu de la mémoire se trouve dans le tableau 1.3 en page 16.*

Exercice 1.2 (Organisation de la mémoire) *Le tableau 1.4 en page 17 donne une représentation d'une partie du contenu d'une mémoire de 64 kilo-octets :*

Cette même mémoire est vue comme un tableau de 32 kilo-mots de 16 bits. Dessiner une représentation de la mémoire avec cette vision :

- *dans le cas* grand boutiste,
- *dans le cas* petit boutiste.

En fournissant une seule adresse mémoire, un processeur 80386 peut accéder soit à un octet, soit à un mot (de 16 bits), soit à un double mot (de 32 bits). On suppose pour les trois questions suivantes que EAX contient `0x88 0x99 0xAA 0xEE`.

Question 1.2.1 *Quel est le contenu du registre EAX aprés l'exécution de l'instruction* `MOVB 12, %AL` *?*

Question 1.2.2 *Quel est le contenu du registre EAX aprés l'exécution de l'instruction* `MOV 12, %AX` *?*

Question 1.2.3 *Quel est le contenu du registre EAX aprés l'exécution de l'instruction* `MOVL 12, %EAX` *?*

Exercice 1.3 (Conversion de base) ***Question 1.3.1*** *Ecrire en base 2 les nombres suivants :*

- ◇ 0, 1, 8, 1024
- ◇ -1, -8, -1024
- ◇ 0.5, 0.25, 0.75

Question 1.3.2 *Ecrire en base 16 les nombres suivants :*

- ◇ 0, 1, -16, 1024
- ◇ 10.5
- ◇ 110100100110.11_2

Question 1.3.3 *Ecrire en base 10 le nombre suivant :*

- ◇ -110100100110.1100_2

Exercice 1.4 (Une question de taille...) ***Question 1.4.1*** *Estimer le nombre de caractères contenus dans un livre de 300 pages.*

Question 1.4.2 *Un caractère étant stocké sur un octet, combien de bits sont nécessaires pour représenter ce livre ?*

Question 1.4.3 *Combien faut-il de bandes magnétiques standard (800 BPI, 700m) pour stocker une bibliothèque de 1000 livres de 300 pages*[3] *?*

Exercice 1.5 (...et encore de taille...) *On considère une fonte de caractère où chaque caractère est défini dans une matrice de* 8×10 *pixels. La figure 1.8 montre le dessin d'un caractère* `g` *dans cette fonte.*

Question 1.5.1 *Combien d'octets sont nécessaires pour conserver le dessin d'un caractère ?*

Question 1.5.2 *Combien de caractères de cette police peut-on afficher sur un écran de* 640×400 *pixels ?*

Question 1.5.3 *Combien de mémoire faut-il pour un écran monochrome de* 640×400 *pixels ?*

Question 1.5.4 *Combien de mémoire faut-il pour conserver les pages d'un livre sous forme d'image ? Et pour la bibliothèque de l'exercice précédent ?*

Exercice 1.6 (...et de son) ***Question 1.6.1*** *Quelle est la quantité de données contenue sur un Compact Disc, pour lequel deux sources monophoniques sont échantillonnées à 44.1 kilo-hertz sur 16 bits chacune ?*

Question 1.6.2 *Combien faudrait-il de Compact Discs pour conserver la bibliothèque de l'exercice précédent sous forme textuelle ? Et sous forme d'images ?*

```
100 buckets of bits on the bus
100 buckets of bits
Take one down, short it to ground
FF buckets of bits on the bus

FF buckets of bits on the bus
FF buckets of bits
Take one down, short it to ground
FE buckets of bits on the bus

ad infinitum...
```

[3] BPI signifie *Bit per Inch*, un inch vaut 2.54 centimètres.

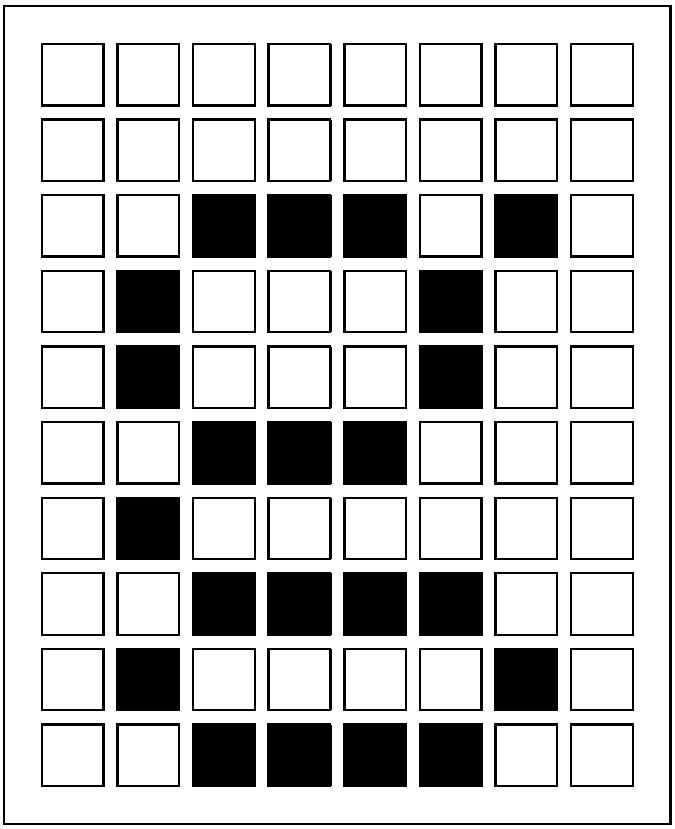

FIG. 1.8 – Le dessin d'un g

0	mem[21] reçoit 22
1	mem[20] reçoit 53
2	mem[mem[20]] reçoit 0
3	décrémenter mem[20]
4	si mem[21] est non nul aller en 7
5	mem[mem[20]] reçoit '0'
6	aller en 18
7	si mem[21] est non nul aller en 10
8	incr mem[20]
9	aller en 18
10	si mem[21] est pair aller en 14
11	mem[mem[20]] reçoit '1'
12	décrémenter mem[20]
13	aller en 16
14	mem[mem[20]] reçoit '0'
15	décrémenter mem[20]
16	mem[21] reçoit mem[21] div 2
17	aller en 7
18	imprimer la chaîne de caractères pointée par mem[20]
19	passer à l'exercice suivant
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	

TAB. 1.3 – Un programme et des données en mémoire

0	0x22
1	0x7C
2	0x02
3	0xC0
4	0xFA
5	0x00
6	0xD0
7	0x30
8	0x3C
9	0x00
10	0x12
11	0xD0
12	0x51
13	0x20
14	0x6E
15	0x00
16	0x56
17	0x10
...	.
	.
	.
65531	0xE0
65532	0x07
65533	0x4E
65534	0x5E
65535	0x4E

TAB. 1.4 – Un contenu de 64 kilo-octets de mémoire