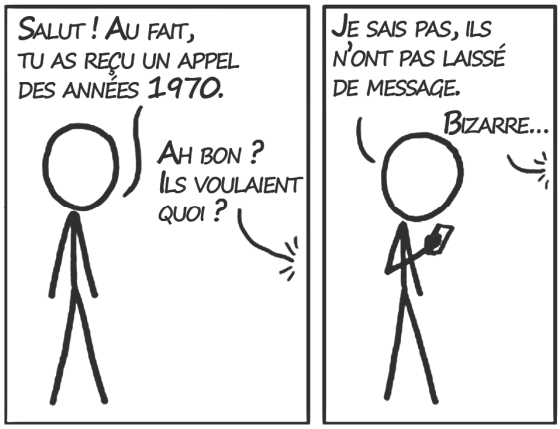
TROISIÈME PARTIE

Machines

Dans cette troisième partie, nous voyons que derrière les informations, il y a toujours des objets matériels : ordinateurs, réseaux, robots, etc. Les premiers ingrédients de ces machines sont des portes booléennes (chapitre 13) qui réalisent les fonctions booléennes vues au chapitre 10. Ces portes demandent à être complétées par d'autres circuits, comme les mémoires et les horloges, qui introduisent une dimension temporelle (chapitre 14). Nous découvrons comment fonctionnent ces machines que nous utilisons tous les jours (chapitre 15). Nous verrons que les réseaux, comme les oignons, s'organisent en couches (chapitre 16*). Et nous découvrons enfin les entrailles des robots, que nous apprenons à commander (chapitre 17*).



XKCD

Les portes booléennes

13

Au commencement était le transistor,
puis nous créâmes les portes booléennes
et, à la fin de la journée, les ordinateurs.

Dans ce chapitre, nous voyons de quoi sont faits les ordinateurs à l'échelle microscopique. Nous partons du transistor et construisons successivement des circuits *non* et *ou* qui vont nous permettre ensuite de construire les circuits de toutes les fonctions booléennes, comme nous l'avons vu au chapitre 10.

Frances Allen (1932-) est une pionnière de la parallélisation automatique des programmes, c'est-à-dire de la transformation de programmes destinés à être exécutés sur un ordinateur séquentiel – contenant un unique processeur – en des programmes destinés à être utilisés sur un ordinateur parallèle – contenant plusieurs processeurs. Elle est aussi à l'origine de nouvelles méthodes, fondées sur la théorie des graphes, pour optimiser les programmes. Elle a reçu le prix Turing en 2006 pour ces travaux.

Nous connaissons des algorithmes depuis plus de quatre mille ans, pourtant nous n'avons pas cherché à les exprimer dans des langages de programmation avant le milieu du XX^{e} siècle. C'est en effet seulement à ce moment que les progrès de l'électronique nous ont permis de construire les premiers ordinateurs. La construction de ces machines a donc eu un effet important sur la manière dont nous concevons aujourd'hui les notions d'algorithme, de langage et d'information.

Le circuit NON

ALLER PLUS LOIN

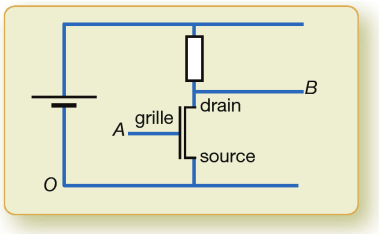
Les circuits CMOS

Dans ce livre nous utilisons un seul type de transistors appelés N-Mos. On construit aujourd'hui plus souvent des circuits qui utilisent deux types de transistors N-Mos et P-Mos, afin de minimiser la consommation d'électricité et la production de chaleur.

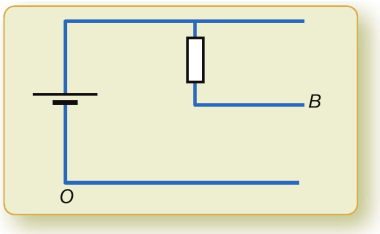
Comme beaucoup de systèmes complexes, un ordinateur peut se décrire à de nombreuses échelles. À l'échelle la plus petite, un ordinateur est un assemblage de transistors. Un transistor est un circuit électronique à trois fils appelés *le drain*, *la source* et *la grille*. La résistance entre le drain et la source est ou bien très petite ou bien très grande en fonction de la tension appliquée entre la grille et la source. Quand cette tension est inférieure à un certain seuil, cette résistance est très grande, on dit que le transistor est *bloqué* ; quand la tension est supérieure à ce seuil, la résistance est très petite, on dit que le transistor est *passant*. Avec un transistor, une résistance et un générateur dont la tension est supérieure au seuil de basculement du transistor, on peut construire le circuit 1.

Si on applique entre le point *A* et le point *O* une tension inférieure au seuil de basculement du transistor, celui-ci est bloqué et le circuit est équivalent au circuit 2, si bien que la tension entre les points *B* et *O* est égale à la tension d'alimentation. Elle est donc supérieure au seuil de basculement. Si, en revanche, on applique entre les points *A* et *O* une tension supérieure au seuil de basculement du transistor, celui-ci est passant et le circuit est équivalent au circuit 3, si bien que la tension entre les points *B* et *O* est nulle. Elle est donc inférieure au seuil de basculement.

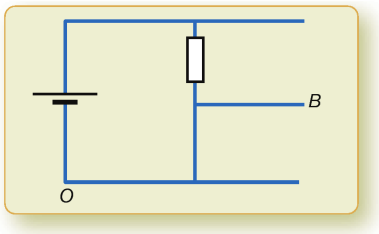
1



2



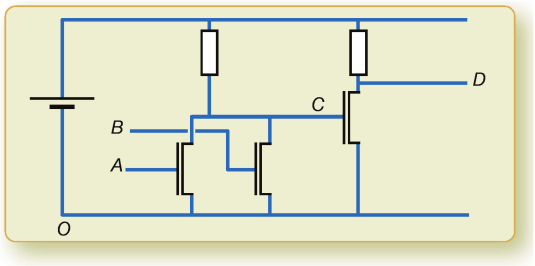
3



Si on décide qu'une tension inférieure au seuil de basculement représente le bit 0 et qu'une tension supérieure à ce seuil représente le bit 1, les deux remarques précédentes se reformulent ainsi : si on donne au circuit le bit 0 en *A*, il donne le bit 1 en *B* ; si on lui donne le bit 1 en *A*, il donne le bit 0 en *B*. Autrement dit, ce circuit calcule une fonction booléenne : la fonction *non*.

Le circuit OU

Le circuit 4 est construit selon les mêmes principes, mais il a deux entrées *A* et *B*.



4

Si on donne aux deux entrées *A* et *B* le bit 0, les deux transistors dans la partie gauche du circuit sont bloqués, si bien que la tension entre les points *C* et *O* est égale à la tension d'alimentation, supérieure au seuil de basculement. Le transistor de droite est donc passant et la tension entre les points *D* et *O* est nulle ; autrement dit le point *D* est dans l'état 0.

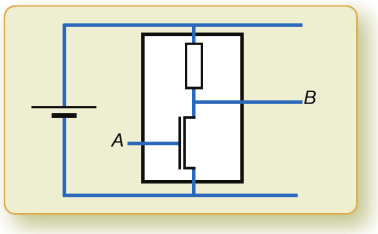
Si on donne à l'une ou l'autre des entrées *A* et *B* le bit 1, au moins l'un des deux transistors dans la partie gauche du circuit est passant, si bien que la tension entre les points *C* et *O* est nulle. Le transistor de droite est donc bloqué et la tension entre *D* et *O* est égale à la tension d'alimentation. Le point *D* est par conséquent dans l'état 1.

La table de ce circuit est donc la suivante (voir ci-contre) où l'on reconnaît la table de la fonction *ou*.

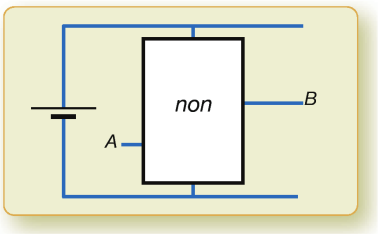
A	B	D
0	0	0
0	1	1
1	0	1
1	1	1

On peut schématiser ces circuits de manière plus succincte en remplaçant le morceau de dessin représentant le transistor et la résistance encadrés dans la figure 5 par un simple rectangle (6) et en remplaçant de même le morceau de dessin représentant les trois transistors et les deux résistances encadrés dans la figure 7 par un rectangle (8).

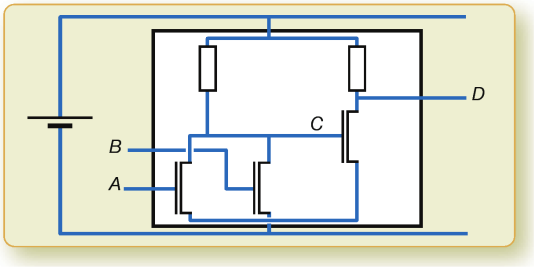
5



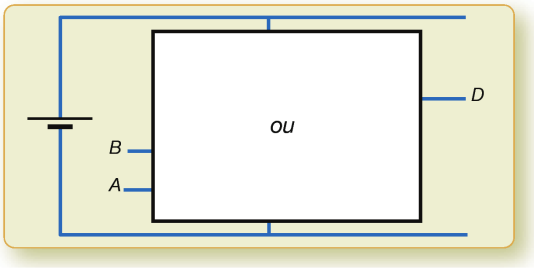
6



7



8



On arrive ainsi à une représentation, à un autre niveau, du même circuit.

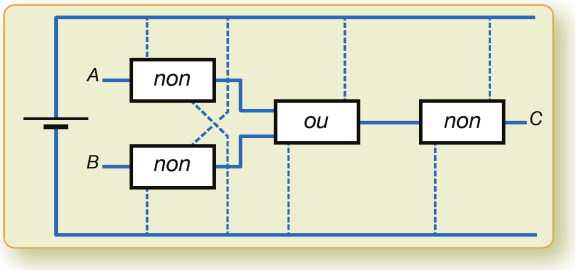
Quelques autres portes booléennes

Les circuits *non* et *ou* s'appellent des *portes booléennes* ou parfois des *portes logiques*.

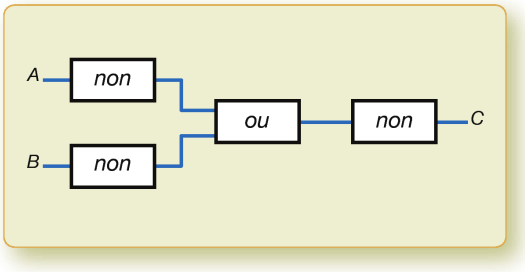
Dans ce chapitre et le suivant, on constitue petit à petit une boîte à outils de circuits réutilisables pour concevoir des circuits plus sophistiqués. Les portes *non* et *ou* sont les deux premiers éléments de cette boîte à outils.

Bien souvent, quand on représente un circuit, on ne dessine pas le générateur : il est implicite que chaque porte est alimentée. On obtient alors une troisième manière de représenter les circuits où le circuit 9 est représenté comme sur le schéma 10.

9

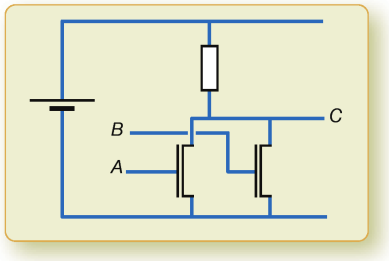


10



Exercice 13.1 (avec corrigé)

Quelle est la table du circuit suivant ?



Si on donne à l'une ou l'autre des entrées A *et* B *le bit 1, au moins l'un des deux transistors dans la partie gauche du circuit est passant, si bien que le point* C *est dans l'état 0. Sinon le point* C *est dans l'état 1.*

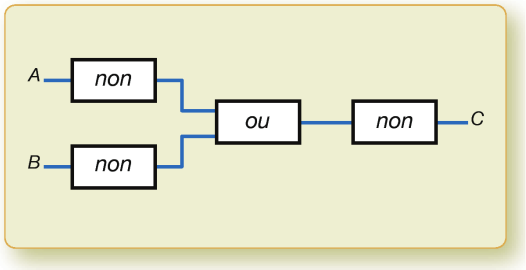
La table de ce circuit est donc la suivante (voir ci-contre).

Il s'agit de la table de la fonction booléenne qui à A *et* B *associe* non (A ou B)*.*

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Exercice 13.2 (avec corrigé)

Quelle est la table du circuit suivant ? Est-ce la table d'une fonction booléenne connue ?



La table de ce circuit est :

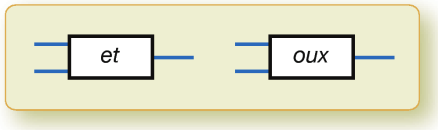
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

C'est celle de la fonction booléenne et.

Exercice 13.3

Construire un circuit réalisant la fonction booléenne *ou exclusif*, vue au chapitre 10.

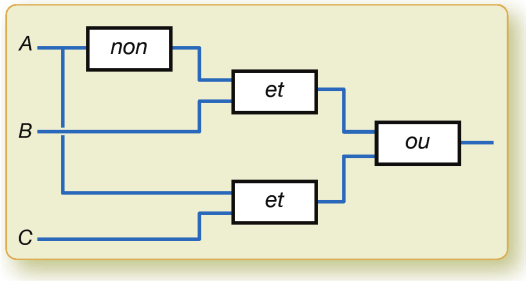
En plus des portes *ou* et *non*, on a désormais dans sa boîte à outils les portes *et* et *oux* :



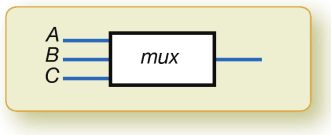
Exercice 13.4 (avec corrigé)

Construire un circuit réalisant la fonction multiplexeur vue au chapitre 10.

La fonction multiplexeur peut se définir par mux (A, B, C) = (non (A) et B) ou (A et C)*. Un circuit, parmi d'autres, réalisant cette fonction est donc :*



On peut désormais utiliser directement le circuit suivant, dont l'unique sortie transmet la valeur de B *ou de* C *selon la valeur de* A *:*



Exercice 13.5 (avec corrigé)

Construire le circuit réalisant le calcul de la fonction *Cout* définie par la table :

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Quelle est cette fonction ?

On utilise la méthode de décomposition par multiplexage (voir le chapitre 10). La fonction Cout *(*Cin, A, B*) s'écrit* mux *(*Cin, g *(*A, B*), g' (*A, B*)), avec* g *(*A, B*) et* g' *(*A, B*) définies par les tables ci-contre :*

g (A, B)

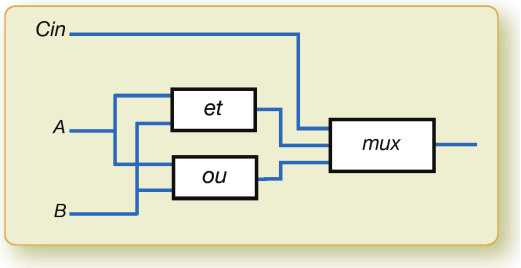
A	B	g
0	0	0
0	1	0
1	0	0
1	1	1

g' (A, B)

A	B	g'
0	0	0
0	1	1
1	0	1
1	1	1

On reconnaît les tables des fonctions et *et* ou, *respectivement. Si bien que* Cout *(*A, B , Cin*) =* mux *(*Cin, A et B, A ou B*).*

Un circuit calculant cette fonction est donc :



Cette fonction est la fonction chiffre des deuzaines de A + B + Cin, *qui sert au calcul de la retenue dans l'algorithme de l'addition (voir le chapitre 18). Ce circuit porte le nom de* Carry out *(retenue sortante).*

Exercice 13.6

Construire un circuit réalisant les opérations calculant la fonction *chiffre des unités de A + B + Cin* (voir le chapitre 18). Construire un circuit *additionneur un bit* qui prend en entrée deux nombres de un bit et donne en sortie leur somme, sur deux bits.

Construire un circuit à quatre entrées et trois sorties, qui ajoute deux nombres exprimés sur deux bits.

Construire un circuit à seize entrées et neuf sorties qui ajoute deux nombres binaires de huit bits.

Exercice 13.7

Construire un circuit à 9 entrées A_0 … A_7 et D et 8 sorties B_0 … B_7 telles que :

1. lorsque $D = 0$, $B_i = A_i$ pour i compris entre 0 et 7,
2. lorsque $D = 1$, $B_i = A_{i-1}$ pour i compris entre 1 et 7 et $B_0 = 0$.

Quand $D = 1$, ce circuit réalise un *décalage à gauche* d'un nombre exprimé en binaire sur huit bits, ce qui correspond à la multiplication par 2 de ce nombre. Dans cette multiplication, le chiffre le plus à gauche, A_7, est oublié afin de faire tenir le résultat sur huit bits.

Exercice 13.8

Construire un circuit à 11 entrées A_0 … A_7 et D_0 … D_2 et 8 sorties B_0 … B_7 telles que $B_i = A_{i-d}$ pour i entre d et 7 et $B_i = 0$ pour i entre 0 et $(d - 1)$, où d est le nombre entre 0 et 7 représenté en binaire par D_0 … D_2. Ce circuit réalise un *décalage à gauche* de d bits d'un nombre binaire de huit bits, ce qui correspond à la multiplication par 2^d. Ce circuit est un composant important d'un circuit réalisant une multiplication binaire.

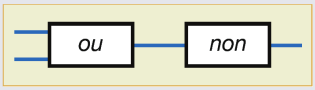
À l'issue de ce chapitre, on dispose donc, parmi d'autres, des circuits booléens suivants dans sa boîte à outils :

- les portes booléennes *non*, *ou*, *et* et *oux*,
- le multiplexeur *mux*,
- des additionneurs de nombres de différentes tailles,
- des multiplieurs par 2 et par 2^d.

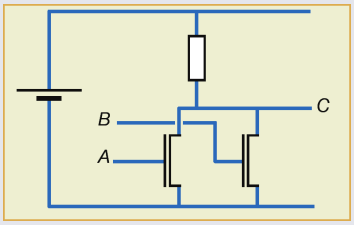
ALLER PLUS LOIN **Penser les systèmes complexes**

Au cours de ce chapitre, nous sommes partis d'une manière de décrire des circuits formés de transistors et de résistances pour, peu à peu, faire émerger une autre manière de décrire certains de ces circuits, sous la forme d'assemblages de portes booléennes. Comme chaque porte est constituée de plusieurs composants électroniques, cette description sous la forme d'un assemblage de portes booléennes est une description à *plus grande échelle* que celle sous forme de transistors et de résistances. Une question se pose alors : peut-on concevoir des circuits en assemblant des portes booléennes et en ignorant complètement la manière dont ces portes sont réalisées avec des transistors ?

Cette question demande une réponse nuancée : comprendre à la fois l'échelle des portes booléennes et celle des transistors permet parfois de réaliser des circuits plus petits, donc moins chers et plus rapides. Par exemple, si on veut construire un circuit qui réalise l'opération booléenne qui à A et B associe *non* (A *ou* B) et si l'on sait uniquement associer des portes booléennes, on construira le circuit suivant



qui, *in fine*, utilise quatre transistors, trois pour la porte *ou* et un pour la porte *non*. Si, en revanche, on sait aussi comment ces portes sont construites avec des transistors, on peut remarquer que le circuit



qui ne comporte que deux transistors, convient. Raisonner à une petite échelle permet donc d'économiser deux transistors. Il est souvent utile, quand on raisonne à une échelle donnée, de faire une incursion à l'échelle inférieure ou à l'échelle supérieure.

Cela dit, il y a aussi des avantages à construire des circuits avec des portes booléennes en ignorant, ou en feignant d'ignorer, les échelles inférieures. Par exemple, les portes booléennes sont aujourd'hui fabriquées avec des transistors, mais par le passé, elles ont été fabriquées avec d'autres composants : des relais, des tubes à vide, etc. et il est possible que, dans le futur, elles soient réalisées avec d'autres composants, aujourd'hui non encore inventés. Raisonner à l'échelle des portes booléennes, sans prendre en compte la manière dont elles sont fabriquées, permet de conserver la même organisation des circuits à l'échelle des portes, même quand la manière dont ces portes sont fabriquées change.

De plus, il est illusoire d'espérer penser simultanément un système aussi complexe qu'un ordinateur à toutes les échelles. S'il est donc, bien entendu, utile d'avoir une culture générale qui donne une idée de la manière dont un ordinateur se décrit à toutes les échelles, il est aussi souvent nécessaire de savoir penser à une échelle unique, en ignorant, ou en feignant d'ignorer, la manière dont les composants que l'on assemble sont fabriqués. Ainsi, aux chapitres 14 et 15, on construira des circuits de plus en plus élaborés en réutilisant les circuits précédents comme de nouveaux composants.

ALLER PLUS LOIN **Qui a inventé l'ordinateur ?**

Contrairement à la pénicilline, il est très difficile de dire qui a inventé l'ordinateur.
Sans remonter aux machines à calculer du XVII^{e} siècle de Wilhelm Schickard, Blaise Pascal, Gottfried Wilhelm Leibniz, etc., ou à la machine analytique imaginée au XIX^{e} siècle par Charles Babbage, l'apparition de l'ordinateur a été préparée par une grande créativité dans le domaine de la construction de machines à la fin du XIX^{e} siècle et au début du XX^{e} siècle, avec, par exemple, la machine à recensement de Herman Hollerith construite en 1889 ou l'analyseur différentiel de Harold Locke Hazen et Vannevar Bush construit entre 1928 et 1931, qui étaient déjà des machines polyvalentes.
La notion d'universalité a été définie mathématiquement en 1936 par Alonzo Church et Alan Turing et il semble que la première machine universelle ait été le Z3, construite en 1941 par Konrad Zuse, même si on ne s'en est rendu compte qu'*a posteriori*. La première machine électronique à utiliser le binaire semble avoir été la machine Colossus, construite par Thomas Flowers au cours de la seconde guerre mondiale, mais cette machine n'était pas universelle. La première machine conçue pour être universelle fut sans doute l'ENIAC, construite en 1946 par John Mauchly, Presper Eckert et John Von Neumann. Pour certains néanmoins, ce n'était pas encore un ordinateur, son programme n'étant pas enregistré dans la mémoire. La première machine à programme enregistré a sans doute été la machine *Baby*, construite à Manchester en 1948 par Frederic Williams, Tom Kilburn et Geoff Tootill.
Il semble donc difficile d'attribuer l'invention de l'ordinateur à un inventeur unique. Il y a plutôt eu un foisonnement d'innovations de la fin des années trente au début des années cinquante qui, chacune à sa manière, ont contribué à l'invention de l'ordinateur.

Ai-je bien compris ?

- Comment réaliser une porte *non* avec des transistors ?
- Comment réaliser une porte *ou* avec des transistors ?
- Est-il nécessaire de savoir réaliser une porte *non* et une porte *ou* avec des transistors pour les assembler en des circuits plus complexes ?

Le temps et la mémoire

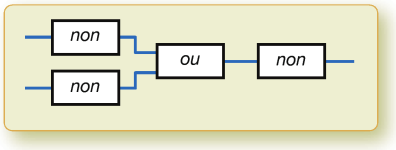
14

Le temps est ce qui permet d'éviter de tout faire en même temps.

Dans ce chapitre, nous voyons comment les circuits électroniques prennent le temps en compte. Nous voyons d'abord comment fabriquer un circuit mémoire. Puis, comment un circuit particulier, l'horloge, permet de synchroniser tous les autres.

Otto Schmitt (1913-1998) est un pionnier du *génie biomédical*. En 1934, en étudiant la propagation de l'influx nerveux dans les nerfs des calmars, il a compris qu'un circuit en boucle fermée positive – c'est-à-dire dans lequel la sortie est connectée à l'entrée, sans inversion de valeur – avait deux états stables et pouvait donc être utilisé pour mémoriser une grandeur. En électronique, une *bascule de Schmitt* est une forme de circuit bistable, qui utilise cette idée de boucle fermée positive.

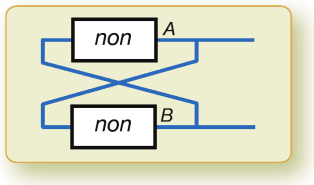
1



Les circuits que nous avons vus au chapitre 13, par exemple le circuit 1 illustré ci-contre, ont des entrées (deux à gauche sur la figure) et des sorties (une à droite sur la figure) et l'état des sorties est déterminé par celui des entrées. Dans cet exemple, la sortie est dans l'état 1 quand les deux entrées sont dans l'état 1 et elle est dans l'état 0 quand au moins l'une des entrées est dans l'état 0. L'état des sorties à un instant donné dépend de l'état des entrées à ce même instant, mais pas de l'état des entrées une seconde ou une minute plus tôt. Un tel circuit, qui ignore le temps, s'appelle un circuit *combinatoire*. Il y a, autour de nous, beaucoup de circuits combinatoires. Par exemple, une lampe est allumée quand son interrupteur est fermé et elle est éteinte quand cet interrupteur est ouvert ; l'état de la lampe dépend de la position de l'interrupteur, mais pas de la position de l'interrupteur une seconde ou une minute plus tôt.

Cependant, il y a aussi autour de nous des circuits moins amnésiques, dont l'état à un instant donné dépend non seulement de l'état de ses entrées à cet instant, mais aussi de leur état passé. Par exemple, quand nous appuyons sur la touche 1 d'une calculatrice, le chiffre 1 apparaît sur l'écran, mais quand nous relâchons cette touche, le chiffre 1 ne disparaît pas : l'état de l'écran à un instant donné dépend donc non seulement de l'état du clavier à ce même instant, mais aussi de toute l'histoire du clavier. Un tel circuit s'appelle un *circuit séquentiel*. Les ordinateurs sont, bien entendu, des circuits séquentiels car, comme nous l'avons vu au chapitre 1, l'exécution d'un programme modifie un état, qui est une description abstraite de l'état de l'ordinateur et dépend donc de toutes les instructions exécutées dans le passé.

La mémoire



Le circuit séquentiel le plus simple est *le circuit mémoire un bit* qui permet de mémoriser un 0 ou un 1. Construire un tel circuit n'est pas difficile, mais il faut procéder en plusieurs étapes. La première est de construire un circuit qui a deux états stables, par exemple celui de la figure ci-contre.

Ce circuit a deux états stables car :

- Si la sortie *A* de la première porte *non* est dans l'état 0, alors l'entrée de la seconde porte *non*, qui est *A* également, est aussi dans l'état 0 ; par conséquent, sa sortie *B* est dans l'état 1, donc l'entrée de la première porte, qui est *B* également, est dans l'état 1, ce qui participe à perpétuer le fait que sa sortie *A* soit dans l'état 0.

ALLER PLUS LOIN **La rupture de symétrie**

Il est difficile de prédire l'état dans lequel se retrouve un circuit mémoire un bit quand on le met sous tension. Il se trouve d'abord, pendant une courte durée, dans un état instable dans lequel les deux sorties *A* et *B* sont dans l'état 0. L'entrée des deux portes *non* est alors dans l'état 0. Rapidement, l'une d'elles produit un état 1 en sortie, un peu avant l'autre, si bien que l'autre, ayant désormais son entrée dans l'état 1, garde sa sortie dans l'état 0. C'est donc une différence de vitesse entre les portes *non* qui détermine l'état du circuit quand on le met sous tension. Cette différence de vitesse est elle-même due à une infime différence de température, de longueur de fil, de pureté des matériaux utilisés pour construire les transistors, etc. Ce phénomène est un exemple de *rupture de symétrie*. Dans le circuit, les points *A* et *B* sont parfaitement symétriques, mais pour arriver à un état ou à un autre, il faut que cette symétrie soit rompue. Les ruptures de symétrie sont fréquentes en physique. Par exemple, si on pose une balle de ping-pong sur le sommet du filet, de manière parfaitement symétrique, elle ne peut pas tomber d'un coté ou de l'autre sans briser cette symétrie. Pourtant il est très rare qu'elle reste en équilibre au sommet du filet : elle finit en général par tomber d'un côté ou de l'autre. Ici encore, un souffle de vent, une petite secousse, ou une imperfection dans la construction de la balle suffit à décider de quel côté elle tombera.

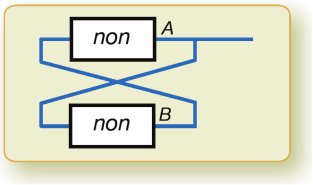
- Si, en revanche, la sortie *A* de la première porte *non* est dans l'état 1, alors l'entrée de la seconde porte *non*, qui est *A* également, est aussi dans l'état 1 ; par conséquent, sa sortie *B* est dans l'état 0, donc l'entrée de la première porte, qui est *B* également, est dans l'état 0, ce qui participe à perpétuer le fait que sa sortie *A* soit dans l'état 1.

Autrement dit, les deux états stables de ce circuit sont :

- $A = 0$ et $B = 1$,
- $A = 1$ et $B = 0$.

En supprimant la sortie *B* et en ne gardant que la sortie *A* (1) on obtient un circuit qui a deux états stables. La sortie *A* vaut 0 dans le premier et 1 dans le second. On peut donc dire que ce circuit *mémorise* la valeur 0 dans le premier cas et la valeur 1 dans le second. Ce circuit est donc un *circuit mémoire*.

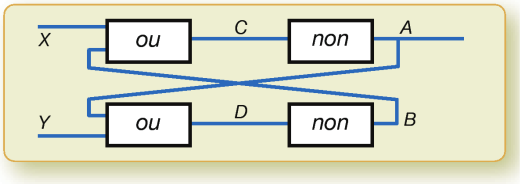
1



Toutefois, ce circuit ayant une sortie, mais pas d'entrée, il n'est pas possible de changer son état et donc la valeur mémorisée.

Pour ce faire, il faut construire un circuit (2), un peu plus complexe, en ajoutant deux portes *ou*.

2



Tant que les entrées *X* et *Y* sont dans l'état 0, tout se passe comme dans le circuit précédent. En effet, si la sortie *A* de la première porte *non* est

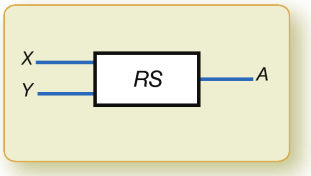
dans l'état 0, alors le point *D* à l'entrée de la seconde est dans l'état 0 également, car 0 *ou* 0 = 0. Et si la sortie *A* est dans l'état 1, le point *D* est dans l'état 1 également, car 1 *ou* 0 = 1. Le point *D* est donc dans le même état que la sortie *A* dans les deux cas. De même, le point *C* à l'entrée de la première porte *non* est dans le même état que *B*. Tout se passe donc comme si les deux portes *ou* n'étaient pas là.

En revanche, si pendant une courte durée on met l'entrée *X* dans l'état 1 tout en laissant l'entrée *Y* dans l'état 0, alors le point *C* passe dans l'état 1 quelle que soit la valeur de *B* car 1 *ou* 0 = 1 et 1 *ou* 1 = 1 ; la sortie *A* passe donc dans l'état 0, le point *D* également et le point *B* passe dans l'état 1. On force donc le circuit à se mettre dans l'état *A* = 0 et *B* = 1, c'est-à-dire à mémoriser la valeur 0. Et quand l'entrée *X* sera revenue à la valeur 0, le circuit restera dans cet état.

Au cours d'étapes qui s'enchaînent très vite, l'état des points *C*, *D*, *A* et *B* devient :

X	Y	C	D	A	B
1	0				
1	0	1			
1	0	1		0	
1	0	1	0	0	
1	0	1	0	0	1

De même, si pendant une courte durée, on met l'entrée *Y* dans l'état 1 tout en laissant l'entrée *X* dans l'état 0, on force le circuit à mémoriser la valeur 1. Ce circuit mémorise donc une valeur 0 ou 1 et, en stimulant l'entrée *X* ou l'entrée *Y*, on peut changer la valeur mémorisée. Ce circuit s'appelle une *bascule RS* (*Reset-Set*) et on peut le représenter comme ci-contre.



À la boîte à outils de circuits commencée au chapitre 13, on peut donc ajouter un premier circuit séquentiel : la bascule *RS*.

On peut aller un peu plus loin et construire un troisième circuit qui mémorise une valeur *V* qu'on lui fournit lorsqu'on stimule l'entrée *S*.

Exercice 14.1 (avec corrigé)

1. Construire un circuit combinatoire à deux entrées *V* et *S* et à deux sorties *X* et *Y* tel que, si l'entrée *S* est dans l'état 0, alors *X* et *Y* sont dans l'état 0 et si *S* est dans l'état 1, alors *X* est dans l'état *non V* et *Y* est dans l'état *V*.
2. Connecter ce circuit combinatoire avec la bascule *RS* ci-avant pour obtenir un circuit qui :
   - quand *S* est dans l'état 0, ne change pas d'état et produit la valeur mémorisée sur la sortie *A*,
   - quand *S* est dans l'état 1 et *V* dans l'état 0, mémorise la valeur 0 et met la sortie *A* dans l'état 0,

- quand *S* est dans l'état 1 et *V* dans l'état 1, mémorise la valeur 1 et met la sortie *A* dans l'état 1.

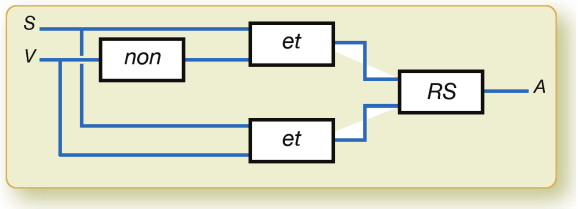
1 *Les tables des fonctions booléennes* X *et* Y *sont les suivantes :*

S	V	X
0	0	0
0	1	0
1	0	1
1	1	0

S	V	Y
0	0	0
0	1	0
1	0	0
1	1	1

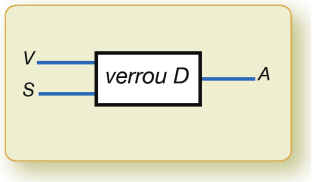
où on reconnaît les fonctions S et non (V) *et* S et V. *Il suffit donc de construire les circuits correspondants.*

2 *On relie la sortie du circuit* S et (non V) *à l'entrée* X *de la bascule* RS *et la sortie du circuit* S et V *à l'entrée* Y *de la bascule* RS.



Ainsi, quand S *est dans l'état 0, le circuit ne change pas d'état et quand* S *est dans l'état 1, il mémorise la valeur* V. *On a donc enfin un véritable circuit mémoire contrôlable : mettre l'entrée* S *dans l'état 1 provoque la mémorisation de la valeur de l'entrée* V. *Cette valeur peut être lue par la suite sur la sortie* A, *jusqu'à ce que l'on provoque la mémorisation d'une nouvelle valeur. Ce circuit s'appelle un* verrou D (D *pour* Donnée *ou* Data).

Verrou D



Exercice 14.2

Quand on utilise plusieurs composants identiques dans un même circuit, on peut différencier leurs entrées et sorties en leur donnant un numéro : ainsi, si l'on a plusieurs verrous *D*, on appellera V_1 l'entrée *V* du premier, A_2 la sortie du deuxième, etc.

On définit un circuit séquentiel appelé *bascule D* en reliant la sortie A_1 d'un premier verrou *D* avec l'entrée V_2 d'un second verrou *D*, et en reliant l'entrée S_1 du premier à une porte *non* dont la sortie est raccordée à l'entrée S_2 du second.

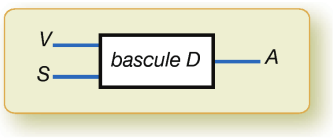
Si l'on considère ce circuit dans son intégralité, ses entrées sont donc l'entrée V_1 du premier verrou *D* et une entrée *S* unique qui alimente à la fois S_1 directement et S_2 via la porte *non*. Sa sortie est la sortie A_2 du second verrou *D*.

Dessiner ce circuit.

Montrer que quand *S* est dans l'état 0, le second verrou met sa valeur à jour, mais pas le premier, et quand *S* est dans l'état 1, le premier verrou met sa valeur à jour, mais pas le second.

Ce circuit s'appelle une *bascule D*.

Bascule D



Son intérêt est que la valeur mémorisée n'est mise à jour qu'à l'instant précis où l'entrée S passe de l'état 1 à l'état 0, alors qu'un simple verrou D laisse passer les valeurs de V vers A tant que S reste à 1. Il est ainsi plus facile de commander l'évolution de ce circuit dans le temps.

Exercice 14.3

On utilise un additionneur un bit (voir le chapitre 13) avec deux entrées A et B et deux sorties S et C, qui sont respectivement le chiffre des unités et le chiffre des deuzaines de $A + B$. On relie la sortie S de l'additionneur à l'entrée V_1 d'une première bascule D, la sortie A_1 de cette bascule étant elle-même reliée à l'entrée A de l'additionneur. On relie également la sortie C de l'additionneur à l'entrée V_2 d'une deuxième bascule D. Enfin, on relie ensemble les entrées S_1 et S_2 des deux bascules et on les alimente avec une entrée commune S.

Dessiner ce circuit.

À chaque fois que l'entrée S des bascules passe dans l'état 1 puis revient à l'état 0, les valeurs mémorisées par les bascules D sont mises à jour. Montrer que le nouvel état de la première bascule est le chiffre des unités de la somme de son ancien état et de l'entrée B. Montrer que le nouvel état de la seconde bascule est le chiffre des deuzaines de cette somme.

Exercice 14.4

En utilisant huit copies du circuit construit à l'exercice précédent, construire un compteur 8 bits comportant une entrée I, tel que l'entier 8 bits mémorisé par compteur soit augmenté d'une unité à chaque fois que l'entrée I passe dans l'état 1 puis revient à l'état 0.

Exercice 14.5

Modifier le circuit de l'exercice précédent de manière à ajouter une entrée R telle que le nombre mémorisé par ce compteur soit initialisé à 0 quand l'entrée R est mise dans l'état 1.

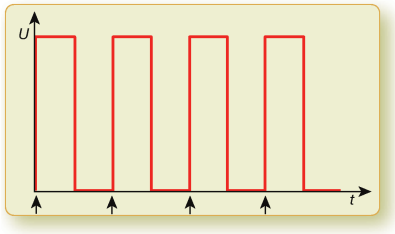
L'horloge

Quand on assemble des circuits séquentiels qui interagissent les uns avec les autres, on obtient un circuit *asynchrone* : l'état de chacun des circuits évolue dans le temps, en fonction de l'état des circuits auxquels il est connecté, mais de manière relativement désordonnée.

Beaucoup d'interactions que nous avons avec les autres sont asynchrones : dans un grand magasin, par exemple, chaque client fait ses courses relativement indépendamment des autres. Et quand plusieurs clients veulent payer leurs courses en même temps, il se forme une file d'attente. De ce fait, chaque client est à peu près sûr qu'il finira par payer et sortir, mais il n'a pas de garantie *a priori* sur le temps que cela prendra. À l'inverse, un petit nombre de nos interactions doivent être synchrones : jouer dans un orchestre demande non seulement de jouer toutes les notes de la partition

dans un certain ordre, mais également de les jouer à un moment donné. Les clients d'un magasin peuvent faire leurs courses chacun à son rythme, mais les musiciens d'un orchestre, les danseurs d'une chorégraphie ou les soldats qui marchent au pas, doivent agir dans une même temporalité. Une manière d'obtenir cette synchronie est de confier à l'un des musiciens, le chef d'orchestre, le rôle de battre la mesure.

En informatique, les machines de grande taille, par exemple les réseaux, sont des machines asynchrones. Chaque utilisateur va à son rythme et le réseau finit par répondre à tout le monde, mais sans garantie du temps que cela prendra. En revanche, les machines de petite taille, telles que les processeurs, sont des machines synchrones. C'est pour cela qu'il y a dans les ordinateurs un circuit, *l'horloge*, dont le rôle est de battre la mesure pour les autres circuits. Une horloge est simplement un circuit qui émet sur sa sortie un signal périodique, par exemple le signal suivant.



Chaque flèche sur la figure marque le début d'un cycle. Avec ce type d'horloge, la sortie est à 1 pendant la première moitié du cycle et à 0 pendant la seconde.

Chacun des circuits, en particulier les circuits mémoires, se synchronise sur un signal d'horloge. Par exemple, si on connecte la sortie de l'horloge sur l'entrée *S* d'une bascule *D*, on obtient un circuit qui enregistre la valeur de l'entrée *V* à chaque cycle.

Exercice 14.6

Construire un circuit comportant un additionneur 8 bits, à 16 entrées et 9 sorties, dont chaque entrée est reliée à la sortie *A* d'un verrou *D* différent, chaque sortie est reliée à l'entrée *V* d'un verrou *D* différent, et dont toutes les entrées *S* de ces 25 verrous sont reliées à une horloge unique.

Établir un lien entre la fréquence de l'horloge et le temps de propagation des signaux électriques portant les retenues des 8 additionneurs un bit composant l'additionneur 8 bits.

Estimer le temps de propagation maximal par porte booléenne, supposé constant et uniforme, compatible avec une fréquence de 1 GHz.

Exercice 14.7

Lorsque l'on relie l'entrée *S* d'une bascule *D* à une horloge, montrer que la sortie *A* se comporte comme l'entrée *V*, mais avec un cycle d'horloge de retard.

Fréquence d'horloge

La *fréquence d'horloge*, qui est souvent indiquée dans les caractéristiques techniques des ordinateurs, est le nombre de cycles par seconde. Par exemple, quand la fréquence d'horloge est 1 GHz, la période de l'horloge, c'est-à-dire la longueur d'un cycle, est de 1 nanoseconde, un milliardième de seconde.

ALLER PLUS LOIN **La réversibilité**

Le fonctionnement d'un circuit séquentiel est dynamique : il se décrit dans le temps. Comme à chaque fois que l'on observe un phénomène dynamique, on peut s'interroger sur sa réversibilité.
Par exemple, si on connaît la position x et la vitesse v, à un instant t, d'un point en mouvement rectiligne uniforme, on peut prédire sa position et sa vitesse à un instant t' du futur : sa position sera $x' = x + v\,(t' - t)$ et sa vitesse sera $v' = v$. Réciproquement, si on connaît sa position x' et la vitesse v' à l'instant t', on peut aussi « rétro-prédire » sa position et sa vitesse à un instant t du passé : sa position était $x = x' - v'\,(t' - t)$ et sa vitesse était $v = v'$. La variation de position et de vitesse d'un point en mouvement rectiligne uniforme est un phénomène dit réversible.
En revanche, quand on mélange un litre de gaz à la température T_1 avec un litre de gaz à la température T_2, on obtient deux litres de gaz à la température $T' = 2\,T_1\,T_2 / (T_1 + T_2)$. Si on connaît les températures T_1 et T_2, on peut donc prédire la température T'. En revanche, si on connaît la température T', il est impossible de rétro-prédire les températures T_1 et T_2 : deux litres de gaz à 300 K peuvent avoir été obtenus en mélangeant un litre à 290 K et un litre à 310.7 K, mais il peuvent aussi avoir été obtenus en mélangeant un litre à 280 K et un litre à 323 K. Mélanger des gaz de températures différentes est donc un phénomène dit irréversible. Non seulement il est impossible de rejouer le film en arrière et d'obtenir les deux litres de gaz aux températures T_1 et T_2, mais il est de plus impossible de définir ce que ces températures T_1 et T_2 doivent être, puisqu'il y a plusieurs solutions à l'équation $2\,T_1\,T_2 / (T_1 + T_2) = T'$.
Les évolutions des états des circuits vues dans ce chapitre sont des évolutions irréversibles, comme le mélange de deux volumes de gaz de températures différentes. Si l'entrée V d'un circuit mémoire est à 0, au moment où l'on met son entrée S à 1, ce circuit enregistre la valeur 0. La valeur précédemment mémorisée est irrémédiablement détruite. Il est donc impossible de rejouer le film à l'envers pour retrouver l'état initial du circuit, car les deux états initiaux possibles mènent au même état final.
La physique statistique et la théorie de l'information permettent de mesurer le degré d'irréversibilité de ces deux phénomènes : la croissance de l'entropie, ou la perte d'information, lors de la destruction d'un bit d'information est de $9{,}5\ 10^{-24}$ J/K et celle lors du mélange d'un litre d'un gaz parfait monoatomique à 10^5 Pa et 290 K avec un litre de ce même gaz à cette même pression et 310.7 K est de $9{,}9\ 10^{-4}$ J/K, soit en comptant cette perte d'information, non en J/K, mais en bits, 10^{20} bits.

ALLER PLUS LOIN **Transformation d'énergie et production de chaleur**

Un processeur consomme de l'électricité et la transforme essentiellement en chaleur. L'électricité consommée et la chaleur produite sont des facteurs physiques limitant les performances des processeurs.
Le coût énergétique d'un calcul a beaucoup diminué avec le temps puisqu'on est passé d'une production de calculs d'environ 400 calculs par kWh pour l'ENIAC, en 1946, à environ 1 million de milliards de calculs par kWh pour un processeur actuel. Dans cette estimation, « un calcul » est par exemple l'addition de deux nombres entiers. Cependant, comme on fait beaucoup plus de calculs qu'en 1946, la quantité d'électricité consommée pour ce faire a beaucoup augmenté pour arriver, en 2007, à environ 1 % de la production mondiale d'électricité. Cette estimation ne tient compte que de l'énergie transformée pour effectuer les calculs et non de celle transformée pour fabriquer, transporter et recycler les machines utilisées.
Les gros centres de serveurs de calcul ou de données ont des besoins énormes en électricité et cherchent souvent à se rapprocher géographiquement des centrales électriques. Ils requièrent de même des systèmes de climatisation de plus en plus sophistiqués, par exemple des systèmes de refroidissement qui utilisent de l'eau de mer.

Ai-je bien compris ?

- Quelle est la différence entre un circuit combinatoire et un circuit séquentiel ?
- Qu'est-ce qu'une bascule RS ?
- Quelle est la différence entre un circuit synchrone et un circuit asynchrone ?

L'organisation d'un ordinateur

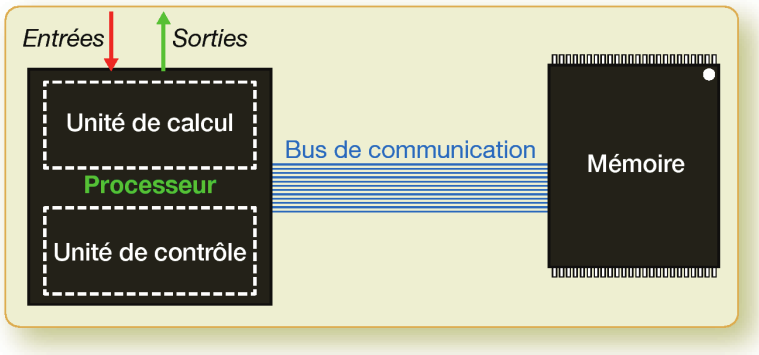
15

Pour fabriquer un ordinateur,
il suffit d'un fer à souder (ou presque…).

Dans ce chapitre, nous voyons de quoi sont faits les ordinateurs à une échelle plus proche de la nôtre et comment architecture et langages sont liés. Nous décrivons la manière dont sont assemblés le processeur de calcul, l'organisation de la mémoire et les bus permettant la circulation des données. Nous voyons comment programmer le processeur au moyen d'un langage machine simple et expliquons comment dérouler une séquence d'instructions. Nous adjoignons enfin les périphériques pour obtenir un ordinateur.

Dans les années 1940, à l'Université de Pennsylvanie, **John von Neumann** (1903-1957) a conçu, avec Presper Eckert et John Mauchly, deux des premiers ordinateurs : l'ENIAC, puis l'EDVAC. Ces ordinateurs étaient organisés selon l'architecture de von Neumann, utilisée dans la quasi-totalité des ordinateurs conçus depuis : séparation du processeur et de la mémoire, reliés par un bus de communication. L'ENIAC pesait vingt-sept tonnes.

Après avoir décrit le fonctionnement d'un ordinateur à l'échelle du transistor puis de la porte booléenne, nous abordons, dans ce chapitre, une troisième échelle de description, qui est celle qui va nous permettre de véritablement en comprendre les principes d'organisation. Un ordinateur est principalement composé de deux grands circuits : le *processeur* et la *mémoire*. Ces deux circuits sont reliés entre eux par des fils qui constituent un ou plusieurs *bus de communication*, parmi lesquels un *bus de données* et un *bus d'adresses*. Le processeur est composé de deux unités. L'*unité de contrôle* lit en mémoire un programme et donne à l'*unité de calcul* la séquence des instructions à effectuer. Le processeur dispose par ailleurs de *bus d'entrées et de sorties* permettant d'accéder aux autres parties de l'ordinateur, que l'on nomme les *périphériques*. Cette organisation générale, l'*architecture de von Neumann*, est étonnamment stable depuis les années quarante.



La mémoire est composée de plusieurs milliards de circuits mémoires un bit. Ces circuits sont organisés en agrégats de huit, seize, trente-deux, soixante-quatre bits, et parfois davantage, que l'on appelle des *cases mémoires* et qui peuvent donc mémoriser des mots de huit, seize, trente-deux, soixante-quatre bits, etc. Le nombre de ces cases définit la taille de la mémoire de l'ordinateur. Comme il faut distinguer ces cases les unes des autres, on donne à chacune un numéro : son *adresse*. La mémoire contient les *données* sur lesquelles on calcule et le *programme* qui décrit le calcul effectué, donné sous la forme d'une séquence d'*instructions*.

Le processeur, de son côté, n'a qu'un très petit nombre de cases mémoires, que l'on appelle des *registres*. On peut imaginer, par exemple, qu'il ne contient que deux registres, appelés *A* et *B*. Les registres peuvent contenir des données, mais aussi des adresses de cases mémoires.

ALLER PLUS LOIN

Taille de la mémoire

En général, on indique la taille de la mémoire en précisant le nombre d'octets, c'est-à-dire de mots de huit bits, qui peuvent être mémorisés. Ainsi, une mémoire de 4 gigaoctets (binaires), contient $4 \times 2^{30} \times 8 = 34\,359\,738\,368$ circuits mémoires un bit. Si la mémoire est organisée en mots de soixante-quatre bits, ces circuits sont répartis en 536 870 912 cases permettant de mémoriser un mot chacune.

ALLER PLUS LOIN

Les processeurs 32 et 64 bits

Lorsque l'on parle de *processeurs 32 bits* ou *64 bits*, on fait référence à la taille de ces registres.

Trois instructions

Pour échanger des données avec la mémoire, le processeur utilise deux procédés qui permettent l'un de transférer l'état d'un registre dans une case mémoire et l'autre de transférer l'état d'une case mémoire dans un registre.

Pour transférer le contenu du registre *A* dans la case mémoire d'adresse *n*, le processeur met les différents fils qui composent le bus d'adresses dans un état qui correspond à l'expression en base deux du nombre *n* et il met les différents fils qui composent le bus de données dans un état qui correspond au contenu du registre. Au signal d'horloge, chaque case de la mémoire compare son propre numéro au numéro arrivé sur le bus d'adresse ; seule la case numéro *n* se reconnaît et elle met alors ses différentes entrées *S* (voir le chapitre 14) dans l'état 1, de manière à enregistrer le mot arrivant sur le bus de données. Le procédé symétrique permet au processeur de récupérer une valeur précédemment enregistrée : les informations circulent toujours du processeur vers la mémoire sur le bus d'adresses, mais elles circulent dans l'autre sens sur le bus de données : c'est la case *n* qui connecte sa sortie au bus de données et c'est le registre qui met ses entrées *S* à 1 de manière à enregistrer le mot qui arrive sur le bus de données.

Ces deux opérations s'appellent le stockage (`STA`) et le chargement (`LDA`) du contenu d'une case mémoire dans le registre *A* (ST pour *STore*, LD pour *LoaD*). Il y a bien entendu des opérations similaires pour le registre *B* (`STB` et `LDB`).

Une autre opération que peut exécuter le processeur est l'addition des contenus des registres *A* et *B*. Le résultat de l'opération peut être stocké aussi bien dans le registre *A* (`ADD A`) que dans le registre *B* (`ADD B`). De même, `DEC A` *décrémente* la valeur contenue dans le registre *A*, c'est-à-dire soustrait 1 à la valeur contenue dans le registre *A* et stocke la valeur ainsi obtenue dans le registre *A* et `DEC B` réalise le même calcul sur la valeur contenue dans le registre *B*.

Si, par exemple, le processeur effectue successivement les opérations ci-contre et si, dans l'état initial, la case 7 de la mémoire contient le nombre 42, la case 8 le nombre 68, la case 9 le nombre 47 et la case 10 le nombre 33, l'exécution des huit opérations a comme effet de :

```
LDA 7
LDB 8
ADD A
LDB 9
ADD A
LDB 10
ADD A
STA 11
```

- `LDA 7` charger le contenu de la case 7, soit 42, dans le registre *A*,
- `LDB 8` charger le contenu de la case 8, soit 68, dans le registre *B*,
- `ADD A` additionner les contenus des registres *A* et *B* et mettre le résultat, 110, dans le registre *A*,
- `LDB 9` charger le contenu de la case 9, soit 47, dans le registre *B*,

- `ADD A` additionner les contenus des registres *A* et *B* et mettre le résultat, 157, dans le registre *A*,
- `LDB 10` charger le contenu de la case 9, soit 33, dans le registre *B*,
- `ADD A` additionner les contenus des registres *A* et *B* et mettre le résultat, 190, dans le registre *A*,
- `STA 11` stocker le contenu du registre *A*, soit 190, dans la case 11.

Au bout du compte, cette séquence d'opérations additionne les quatre nombres stockés dans les cases 7, 8, 9 et 10 de la mémoire et stocke le résultat dans la case 11.

Le langage machine

Un ordinateur doit être capable d'exécuter un programme. Il faut donc un moyen d'indiquer au processeur la séquence des instructions qu'il doit exécuter ; par exemple, la séquence `LDA 7`, `LDB 8`, `ADD A`, `LDB 9`, `ADD A`, `LDB 10`, `ADD A`, `STA 11`. Dans les premières machines, des cartes perforées ou un ruban perforé situé à l'extérieur de la machine indiquaient les opérations à effectuer, comme les cartes d'un orgue de Barbarie indiquent les notes à jouer l'une après l'autre. Puis cette idée a été abandonnée au profit d'une autre : celle d'enregistrer le programme dans la mémoire avec les données. Ainsi, on peut exprimer le programme précédent en binaire en décidant par exemple que *A* s'écrit 0, *B* s'écrit 1 et les instructions `LDA`, `LDB`, `STA`, `STB`, `ADD` et `DEC` s'écrivent respectivement 0, 1, 2, 3, 4 et 5. Le programme de notre exemple s'écrit alors 0, 7, 1, 8, 4, 0, 1, 9, 4, 0, 1, 10, 4, 0, 2, 11, ce qui commence à devenir assez difficile à lire, même s'il est facile de passer d'une représentation à l'autre. On peut ensuite stocker ce programme dans la mémoire, en commençant, par exemple, à la case 100 :

adresse	..	..	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	..	..
valeur	..	..	0	7	1	8	4	0	1	9	4	0	1	10	4	0	2	11	..	..

Il suffit maintenant d'ajouter au processeur un nouveau registre qui débute à 100, le *compteur de programme* ou *PC (program counter)*, et à chaque étape, le processeur :

- charge le contenu des cases mémoires d'adresses *PC* et *PC* + 1,
- décode le premier de ces nombres en une instruction (0 devient `LDA`, 1 `LDB`, etc.),

- exécute l'instruction en question,
- et ajoute 2 au registre *PC*.

Enregistrer les programmes en mémoire permet de faire très simplement des boucles et des tests. On ajoute aux instructions précédentes une instruction `JMP` (*jump*) telle que `JMP` *n* charge simplement le nombre *n*, ou plutôt le nombre *n* - 2 qui sera augmenté de 2 immédiatement après l'exécution du `JMP`, dans le registre *PC* pour détourner le programme de sa route et le forcer à continuer son exécution à l'adresse *n*. De même, l'instruction `JMPZ` (*jump if zero*), qui effectue un saut si le contenu du registre *A* est 0, permet de faire des tests. On ajoute enfin l'instruction `END`, qui termine le programme. En langage machine, on suppose que `JMP`, `JMPZ` et `END` s'écrivent respectivement 6, 7 et 8 et que `END` prend 0 comme argument puisqu'il en faut un. Mais cet argument n'est pas utilisé.

Pour construire une boucle ou un test avec ces nouvelles instructions, il faut tout d'abord trouver une façon de traduire la condition du test ou la condition d'arrêt de la boucle par un test d'égalité à zéro. Par exemple, pour effectuer un test comme `x == 2`, on peut placer la valeur de `x` dans le registre *A*, exécuter deux fois l'instruction `DEC A` et enfin tester si le registre *A* contient 0. Ensuite, on écrit les séquences d'instructions qui correspondent aux différentes branches du test ou au corps de la boucle, et on utilise `JMPZ` et `JMP` pour diriger l'exécution du programme dans l'une ou l'autre de ces séquences. Par exemple, un programme qui lit une valeur `x` dans la case mémoire d'adresse 11, puis recopie la case mémoire d'adresse 12 dans la case mémoire d'adresse 20 si `x` vaut 2, ou la case mémoire d'adresse 13 dans la case mémoire d'adresse 30 dans le cas contraire, peut s'écrire :

100	0	`LDA 11`
101	11	
102	5	`DEC A`
103	0	
104	5	`DEC A`
105	0	
106	7	`JMPZ 114`
107	114	
108	1	`LDB 13`
109	13	

110	3	`STB 30`
111	30	
112	6	`JMP 118`
113	118	
114	1	`LDB 12`
115	12	
116	3	`STB 20`
117	20	
118	8	`END`
119	0	

ALLER PLUS LOIN

Pourquoi séparer A et B pour ADD et DEC mais pas pour LD et ST ?

Dans le langage machine que nous venons d'inventer, pour que le compteur de programme fonctionne correctement, chaque instruction utilise exactement deux cases mémoire : une pour son nom et une pour son argument. L'argument des instructions ADD et DEC est le nom d'un registre. En revanche, comme les instructions de chargement ont déjà un argument (l'adresse mémoire où aller chercher la donnée à charger), elle ne peuvent pas en avoir un second pour indiquer le registre utilisé. C'est pour cela que l'on a deux instructions LDA et LDB et de même deux instructions STA et STB.

ALLER PLUS LOIN

Calcul sur des structures de données plus complexes

Les instructions que nous venons de présenter ne permettent d'accéder qu'à un nombre limité de cases mémoire, dont les adresses sont les constantes entières qui servent d'arguments aux instructions LDA, STA, LDB et STB. Le calcul sur des structures de données plus complexes, comme des tableaux, nécessite d'autres instructions pour accéder à une case mémoire dont l'adresse est elle-même une donnée, en particulier le résultat d'un calcul. Les processeurs ont bien d'autres instructions encore : des opérations booléennes, des opérations sur les nombres entiers, des opérations sur les nombres flottants, etc.

SAVOIR-FAIRE Savoir dérouler l'exécution d'une séquence d'instructions

Le principe est de suivre, instruction par instruction, l'évolution du programme en observant les effets sur les valeurs contenues dans les registres, y compris le compteur de programme *PC*, et les valeurs contenues dans la mémoire, un peu comme on le ferait pour l'état de l'exécution d'un programme écrit en Java.

Exercice 15.1 (avec corrigé)

Écrire une séquence d'instructions qui multiplie par 5 le nombre contenu dans la case mémoire d'adresse 10 et stocke le résultat dans la case mémoire d'adresse 11.

Pour multiplier par 5, on fait 4 additions, en accumulant le résultat dans le registre A. *On note x le nombre rangé à l'adresse 10.*

	A	*B*	
LDA 10	*x*		*Charger dans* A *le nombre rangé à l'adresse mémoire 10*
LDB 10	*x*	*x*	*Charger dans* B *le nombre rangé à l'adresse mémoire 10*
ADD A	*x+x*	*x*	*Additionner* A *et* B*, résultat dans* A
ADD A	*x+x+x*	*x*	*Additionner* A *et* B*, résultat dans* A
ADD A	*x+x+x+x*	*x*	*Additionner* A *et* B*, résultat dans* A
ADD A	*x+x+x+x+x*	*x*	*Additionner* A *et* B*, résultat dans* A
STA 11	*x+x+x+x+x*	*x*	*Stocker le nombre contenu dans* A *à l'adresse mémoire 11*

La séquence LDA 10, LDB 10, ADD A*, ADD* A*, ADD* A*, ADD* A*, STA 11, s'écrit en langage machine 0, 10, 1, 10, 4, 0, 4, 0, 4, 0, 4, 0, 2, 11. On la stocke par exemple à partir de l'adresse 100 de la mémoire.*

État de la mémoire avant l'exécution du programme :

adresse	..	*10*	*11*	..	..	*100*	*101*	*102*	*103*	*104*	*105*	*106*	*107*	*108*	*109*	*110*	*111*	*112*	*113*	..
valeur	..	*x*		..	..	*0*	*10*	*1*	*10*	*4*	*0*	*4*	*0*	*4*	*0*	*4*	*0*	*2*	*11*	..

État de la mémoire après l'exécution du programme :

adresse	..	*10*	*11*	..	..	*100*	*101*	*102*	*103*	*104*	*105*	*106*	*107*	*108*	*109*	*110*	*111*	*112*	*113*	..
valeur	..	***x***	***5x***	..	..	*0*	*10*	*1*	*10*	*4*	*0*	*4*	*0*	*4*	*0*	*4*	*0*	*2*	*11*	..

Exercice 15.2

Expliquer ce que fait le programme suivant, écrit en langage machine, en supposant que le nombre *x* contenu dans la case mémoire d'adresse 10 est strictement positif.

adresse	..	10	11	..	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	..
valeur	..	x	y		0	10	1	10	7	112	5	0	4	1	6	104	3	11	8	0	..

Exercice 15.3

Écrire un programme qui lit deux valeurs x et y contenues respectivement dans les cases mémoires 11 et 12, calcule la différence $y - x$ et stocke le résultat à l'adresse 13. On suppose que ces deux valeurs sont des nombres entiers positifs.

Compléter ce programme pour qu'il stocke la valeur 0 à l'adresse 15 si x est égal à y, ou la valeur x sinon.

Exercice 15.4

Écrire un programme qui multiplie la valeur contenue à la case mémoire 12 par celle contenue dans la case mémoire 13 et stocke le résultat à l'adresse 14. On suppose que ces valeurs sont des nombres entiers positifs.

Quel problème l'écriture de ce programme pose-t-elle ? Quelle modification du processeur permettrait de contourner ce problème et donc de simplifier le programme ?

La compilation

Les premiers programmeurs écrivaient des programmes en langage machine qui ressemblaient à `LDA 7`, `LDB 8`, `ADD A`, `LDB 9`, `ADD A`, `LDB 10`, `ADD A`, `STA 11`, ou plus exactement à 0, 7, 1, 8, 4, 0, 1, 9, 4, 0, 1, 10, 4, 0, 2, 11, ce qui était très fastidieux et offrait de nombreuses possibilités de se tromper. On a alors cherché à concevoir des langages dans lesquels ce programme pouvait s'écrire :

```
e = a + b + c + d;
```

ce qui a permis de développer des programmes de manière plus rapide et plus sûre. Néanmoins, aujourd'hui encore, les ordinateurs ne comprennent que les programmes de la forme 0, 7, 1, 8, 4, 0, 1, 9, 4, 0, 1, 10, 4, 0, 2, 11. C'est pourquoi quand on écrit :

```
e = a + b + c + d;
```

on doit ensuite utiliser un programme qui transforme les programmes écrits en langage évolué en des programmes écrits en langage machine : un *compilateur*. Un programme existe donc toujours sous deux formes : le *code source* écrit en Java, C, etc. et le *code compilé* écrit en langage machine.

Le compilateur est un traducteur d'un langage évolué vers un langage machine. Une manière simple de compiler un programme est de traduire les instructions une à une. Cependant, comme pour les circuits vus au chapitre 13, le compilateur peut construire des programmes en langage machine plus efficaces en se plaçant aussi à l'échelle du langage machine. Par exemple, le temps d'accès à une donnée en mémoire peut être une centaine de fois supérieur au temps d'accès à un registre. Conserver une

valeur dans un registre entre deux instructions au lieu de la stocker pour la recharger ensuite peut donc faire gagner beaucoup de temps de calcul.

Les périphériques

Outre un processeur, une mémoire, une horloge et des bus reliant le processeur à la mémoire, un ordinateur est également constitué de *périphériques* : écrans, claviers, souris, disques, haut-parleurs, imprimantes, scanners, cartes réseau, clés de mémoire flash, etc. qui permettent au processeur d'échanger des informations avec l'extérieur : des êtres humains, à travers par exemple l'écran et le clavier, d'autres ordinateurs, à travers la carte réseau, et des outils de stockage, par exemple des disques. On peut grossièrement classer les périphériques en *périphériques d'entrée*, qui permettent au processeur de recevoir des informations de l'extérieur, et *périphériques de sortie*, qui lui permettent d'émettre des informations vers l'extérieur. Toutefois, beaucoup de périphériques sont à la fois des périphériques d'entrée et de sortie. Ainsi, un écran est a priori un périphérique de sortie, mais un écran tactile est aussi un périphérique d'entrée.

Pour échanger des informations avec les périphériques, le processeur procède d'une manière très similaire à celle qu'il utilise pour échanger des informations avec la mémoire. Par exemple, on peut donner une adresse à chaque pixel d'un écran noir et blanc, exactement comme on donne une adresse à chaque case de la mémoire ; stocker une valeur comprise entre 0 et 255 à cette adresse, avec l'instruction `STA` par exemple, aura pour effet d'allumer ce pixel à l'écran avec le niveau de gris correspondant. De même l'instruction `LDA` permet de recevoir des informations d'un périphérique de sortie, par exemple la position de la souris. Selon les processeurs, ce sont les instructions `STA` et `LDA` elles-mêmes qui sont utilisées, ou des instructions voisines, spécialisées pour la communication avec les périphériques.

Le système d'exploitation

La description des ordinateurs que l'on a donnée dans ce chapitre diffère quelque peu de l'expérience pratique qu'ont tous ceux qui ont déjà utilisé un ordinateur. Tout d'abord, on s'aperçoit que si on bouge la souris, le curseur de souris bouge sur l'écran, il semble donc y avoir un programme qui interroge en permanence la souris pour connaître sa position et dessine un cur-

seur de souris à l'endroit correspondant de l'écran. De même, il est possible d'utiliser simultanément plusieurs programmes, alors que dans la description que nous avons donnée, le processeur n'en exécute qu'un seul à la fois.

Cela est dû au fait que dès qu'on allume un ordinateur, un programme spécial est lancé : *le système d'exploitation*. Ce programme, souvent gigantesque, a plusieurs fonctions :

- Il permet l'exécution simultanée de plusieurs programmes, selon le système du *temps partagé* : le système d'exploitation exécute chacun des programmes à tour de rôle et pendant une courte durée, garantissant à chacun que ses données ne seront pas modifiées par les autres – ainsi, même si un programme croit utiliser les cases mémoire 1, 2 et 3, il se peut qu'il utilise en fait les cases 4097, 4098 et 4099, car le système d'exploitation a réservé les véritables cases 1, 2 et 3 pour un autre programme.
- Il gère les périphériques ; ainsi, pour imprimer un caractère sur l'écran, il n'est pas nécessaire d'allumer chaque pixel l'un après l'autre, mais on peut demander au système d'exploitation d'afficher un « A » et celui-ci traduira cette instruction en une suite d'instructions qui afficheront un « A » pixel par pixel. La partie du système d'exploitation qui gère un périphérique s'appelle un *pilote*.
- En particulier, il gère le disque, son découpage en fichiers, l'attribution d'un nom à chaque fichier et leur organisation arborescente (voir le chapitre 11).
- Il gère aussi l'écran, c'est-à-dire son découpage en fenêtres, l'ouverture et la fermeture des fenêtres.
- Dans certains systèmes utilisés par plusieurs personnes, il gère l'authentification de chaque utilisateur et les droits, en particulier de lecture et d'écriture des fichiers, associés à chacun d'eux.

ALLER PLUS LOIN
Plusieurs systèmes d'exploitation

Il existe plusieurs systèmes d'exploitation : Unix, Linux ou GNU/Linux, Windows, Mac OS, etc. Toutefois, le développement d'un système d'exploitation est une tâche si coûteuse en temps, qu'il n'existe guère plus d'une dizaine de systèmes d'exploitation réellement utilisés.

ALLER PLUS LOIN **Les ordinateurs parallèles**

Une manière de fabriquer des ordinateurs plus rapides est de mettre dans un ordinateur plusieurs processeurs qui calculent *en parallèle*, c'est-à-dire en même temps. Les ordinateurs qui ont plusieurs processeurs parallèles sont appelés *ordinateurs parallèles* ou *multicœurs*. Certains algorithmes sont très faciles à paralléliser : par exemple pour augmenter la luminosité d'une image en niveaux de gris, il faut ajouter une constante à chaque pixel. Chaque pixel peut être traité indépendamment des autres et utiliser deux processeurs au lieu d'un divise le temps de calcul par deux. Toutefois, d'autres algorithmes sont plus difficiles à paralléliser.

La programmation des processeurs parallèles est plus difficile car, en plus d'écrire des instructions, il faut prévoir sur quel processeur elles vont s'exécuter.

Les processeurs peuvent se partager une mémoire ou plusieurs. Si deux processeurs partagent la même mémoire et doivent se communiquer des données, il faut s'assurer que le premier ait bien fini de les calculer et de les stocker en mémoire avant que le second ne les lise. On dit que les deux processeurs doivent se *synchroniser*. Ils peuvent aussi avoir des mémoires différentes et communiquer par un bus. Les processeurs peuvent fonctionner sur la même horloge ; on dit alors qu'ils sont *synchrones*. Ou bien chaque processeur peut avoir sa propre horloge ; on dit qu'ils sont *asynchrones*. Le parallélisme existe aussi à l'intérieur des processeurs, entre les instructions du langage machine. Cette forme de parallélisme s'appelle le *parallélisme d'instructions*.

ALLER PLUS LOIN **La hiérarchie mémoire**

Il y a une évidente ressemblance entre les notions de case de la mémoire d'un ordinateur et de boîte associée à une variable dans un état de l'exécution d'un programme. Toutefois, ces deux notions ne sont pas absolument identiques. Un programmeur peut faire comme si une valeur stockée dans une boîte de nom x était stockée au même endroit de la mémoire d'un bout à l'autre de l'exécution du programme. Cependant, cette valeur peut en réalité changer de place. Elle peut être en mémoire, mais comme le système d'exploitation gère plusieurs programmes en temps partagé, il peut très bien attribuer à la boîte de nom x d'abord une case de la mémoire, puis une autre plus tard. Ce que le système d'exploitation garantit est que, quoi qu'il se passe en réalité, tout se passera de manière transparente pour le programmeur, qui peut donc faire comme si le contenu de cette boîte était toujours stocké dans la même case. Si on utilise des processeurs parallèles pour faire un calcul, la valeur de la boîte x peut être dans la mémoire d'un processeur ou d'un autre, voire dans plusieurs mémoires en même temps. On a vu aussi que la valeur peut être dans un registre au moment où elle est utilisée pour un calcul ou bien elle même calculée si elle est le résultat d'un autre calcul.

La réalité est encore un peu plus complexe car les processeurs disposent de *mémoires caches* ou *antémémoires*. Ce sont les mémoires *L1* et *L2* indiquées dans les caractéristiques techniques des processeurs. Ces mémoires sont d'accès plus rapides que la mémoire ordinaire, mais beaucoup moins grandes. Généralement, l'accès à une mémoire cache ne prend que quelques cycles d'horloge, mais sa capacité est limitée à quelques kilooctets ou mégaoctets. Pendant le calcul, la valeur de la boîte x peut aussi se trouver dans une de ces mémoires caches. Un mécanisme matériel gère automatiquement ces deux mémoires et les instructions du langage machine ne permettent même pas de choisir d'utiliser l'une ou l'autre. Toutefois, le principe à retenir est que toute donnée récemment utilisée a de fortes chances d'être encore dans un des caches. Donc éviter de trop éloigner les utilisations d'une même variable dans un programme peut faire gagner beaucoup de temps de calcul.

On a ici un nouvel exemple de la description d'une même réalité à différents niveaux d'abstraction.

ALLER PLUS LOIN **La distribution du code source d'un logiciel**

Certains auteurs et éditeurs de programmes ne donnent à leurs utilisateurs que le code compilé de leurs programmes. Ils peuvent ainsi garder leurs secrets de fabrication. Les usagers peuvent utiliser ces programmes, mais ne peuvent pas comprendre comment ils fonctionnent.

D'autres, à l'inverse, donnent à leurs utilisateurs à la fois le code compilé et le code source. Cela permet aux utilisateurs qui le souhaitent de comprendre comment le programme est conçu, de l'adapter à leurs propres besoins si le programme n'y répond qu'imparfaitement, ou de vérifier que le programme ne fait rien d'indésirable, par exemple qu'il ne contient pas d'espion qui communique à son insu des informations sur l'utilisateur à l'auteur du programme. Cela permet aussi aux utilisateurs de contribuer à l'amélioration du programme, en signalant des erreurs, en les corrigeant ou en ajoutant des composants aux programmes.

Les partisans de la distribution du code source des logiciels articulent leurs arguments sur deux plans : un plan éthique et philosophique et un plan scientifique et technique. Certains insistent sur l'impératif moral de distribuer le code source de ses programmes, avec l'objectif de diffuser à tous des connaissances et de leur permettre de se les approprier. D'autres insistent sur le fait que permettre aux utilisateurs de contribuer aux logiciels développés en améliore la qualité. De fait, certains logiciels tels les systèmes d'exploitation GNU/Linux sont aujourd'hui développés par des milliers de contributeurs de par le monde, ce qui serait impossible si le code source n'était pas publié.

ALLER PLUS LOIN **Des programmes auto-modifiants aux virus**

Stocker le programme à exécuter dans la mémoire, avec le reste des données, a permis de prendre conscience de la possibilité pour un programme de se modifier lui-même, par une simple instruction STA, dans la partie de la mémoire consacrée au stockage du programme. Si cette possibilité disparaît dans les langages de programmation évolués, elle est bien présente dans les programmes directement écrits en langage machine.

Cette possibilité de se dupliquer et de se transformer est en particulier utilisée par les *virus informatiques*. Un *virus* est un programme qui est conçu pour se dupliquer et se propager d'ordinateur en ordinateur à l'insu de leurs utilisateurs, par le réseau, ou tout autre support permettant de transmettre de l'information : clé de mémoire flash, CD-ROM, etc. Une fois installé sur un ordinateur, un virus peut espionner ses utilisateurs et communiquer les informations collectées par le réseau. Il peut aussi simplement utiliser la capacité de calcul de l'ordinateur hôte, par exemple pour envoyer des courriers en très grand nombre.

Ai-je bien compris ?

- Quels sont les principaux circuits d'un ordinateur ?
- Qu'est-ce que le langage machine ?
- Qu'est-ce que compiler un programme ?

XKCD

Les réseaux 16

CHAPITRE AVANCÉ

Les ordinateurs parlent aux ordinateurs.

Dans ce chapitre, nous voyons comment les ordinateurs communiquent entre eux, et comment ces communications se composent pour faire fonctionner le réseau Internet. Ces mécanismes de communication de machine à machine s'appellent des protocoles.

Les protocoles de la couche physique connectent les bus des ordinateurs. Les protocoles de la couche lien organisent un réseau local autour d'un serveur et repèrent les ordinateurs par l'adresse MAC de leur carte réseau. Les protocoles de la couche réseau organisent les réseaux locaux de proche en proche et repèrent les ordinateurs par leur adresse IP.

Nous expliquons comment les informations sont acheminées au travers du réseau à l'aide de routeurs.

Vinton Cerf (1943-) et **Robert Kahn** (1938-) ont inventé, au début des années 1970, le protocole de transmission de paquets de données IP (*Internet Protocol*) et le protocole de contrôle de flux de données TCP (*Transmission Control Protocol*). Il s'agit des deux principaux protocoles du réseau Internet. Ils donnent à ce réseau sa fiabilité, sa robustesse en cas de pannes ou de modifications et sa capacité à évoluer. Cela a valu à leurs auteurs le surnom de « pères d'Internet. »

Nous avons vu qu'un ordinateur pouvait se décrire à différentes échelles : les transistors s'assemblent en portes booléennes, qui s'assemblent à leur tour en composants – processeurs, mémoires, etc. – qui s'assemblent à leur tour en ordinateurs. Et nous pouvons continuer, car les ordinateurs s'assemblent à leur tour en réseaux de différentes tailles : des réseaux les plus simples, formés de deux ordinateurs reliés par un câble ou par radio, aux réseaux locaux connectant quelques ordinateurs entre eux – les ordinateurs d'un lycée par exemple –, qui s'assemblent à leur tour pour former le plus grand des réseaux : *Internet*, lequel relie presque tous les ordinateurs du monde.

Réseau

Un *réseau* est un ensemble d'ordinateurs et de connexions qui permettent à chaque ordinateur de communiquer avec tous les autres, éventuellement en passant par des intermédiaires.

Les protocoles

Si un programme P_A, par exemple un logiciel de courrier électronique, exécuté sur un ordinateur A, veut communiquer des informations à un autre programme P_B, exécuté sur un ordinateur B, il sous-traite cette tâche à un programme spécialisé Q_A, exécuté sur l'ordinateur A, qui met en œuvre un *protocole*. Ce programme Q_A dialogue, suivant les spécifications de ce protocole, avec un programme homologue Q_B exécuté sur l'ordinateur B, ce qui permet ainsi la communication entre les programmes P_A et P_B.

Protocole

Un *protocole* est un ensemble de règles qui régissent la transmission d'informations sur un réseau. Il existe de nombreux protocoles, chacun spécialisé dans une tâche bien précise.

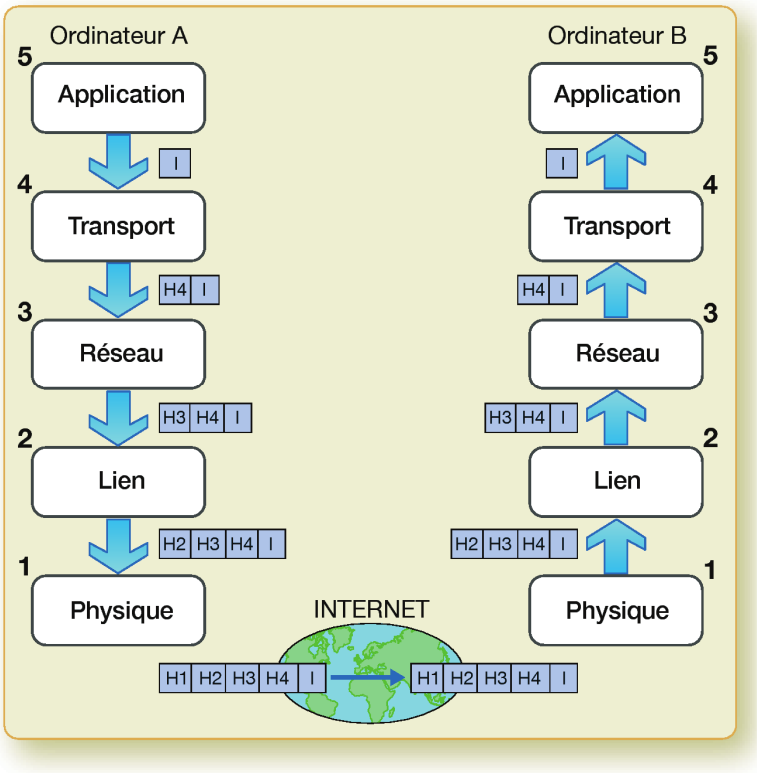
En fait, le programme Q_A sous-traite, à son tour, certaines tâches moins sophistiquées à d'autres programmes mettant en œuvre d'autres protocoles, qui sous-traitent, de même, certaines tâches encore plus élémentaires à d'autres protocoles, etc. On peut ainsi classer les protocoles en *couches* hiérarchiques, par le niveau de sophistication des tâches qu'ils exécutent.

Couche

Une *couche* est un ensemble de protocoles qui effectuent des tâches de même niveau. On distingue cinq couches appelées *couche application*, *couche transport*, *couche réseau*, *couche lien* et *couche physique*.

Ainsi, les informations envoyées par le programme de courrier électronique sont d'abord confiées à un protocole de la couche application, qui les confie à un protocole de la couche transport, qui les confie à un protocole de la couche réseau, qui les confie à un protocole de la couche lien, qui les confie à un protocole de la couche physique, qui les transmet effectivement vers l'ordinateur B.

Quand on confie une lettre à un facteur, on doit la mettre dans une enveloppe et ajouter sur l'enveloppe des informations supplémentaires : l'adresse du destinataire, sa propre adresse, une preuve de paiement, etc. De même, quand un protocole de la couche k+1 confie des informations à un protocole de la couche k, celui-ci ajoute à ces informations un *entête* H_k qui contient des informations, comme l'adresse de l'ordinateur destinataire, utilisées par le protocole de la couche k. On appelle cela l'*encapsulation* des informations. Quand les informations I confiées par la



Système en couches, piles de protocoles. Encapsulation et décapsulation de l'information.

couche application à la couche transport arrivent à un protocole de la couche physique, plusieurs en-têtes H_4, H_3, H_2, H_1 leur ont été ajoutés. Ces en-têtes sont supprimés à la réception : la couche k analyse puis supprime H_k avant de passer l'information à la couche $k+1$. On appelle cela la *décapsulation* des informations.

Aller plus loin **Les normes**

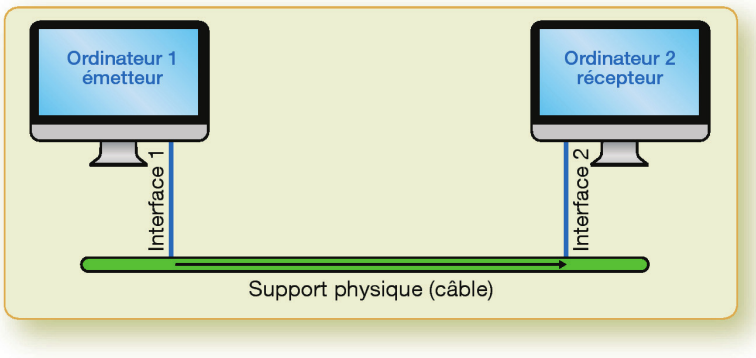
Des *normes* régissent les rôles de chaque couche, leurs interactions et les spécifications de chaque protocole. Ces normes permettent au réseau Internet de fonctionner à l'échelle mondiale et assurent la modularité du système : il est possible de modifier les protocoles à l'œuvre au sein d'une couche, sans modifier les protocoles des autres couches, et le système continue à fonctionner dans son ensemble. Cette modularité est analogue au fait de pouvoir changer un composant matériel d'un ordinateur, par exemple sa carte graphique, sans devoir rien changer d'autre. Cette modularité est essentielle pour permettre au système d'évoluer.

La communication bit par bit : les protocoles de la couche physique

Commençons la présentation de ces protocoles par ceux de la couche physique. Un protocole de la couche physique doit réaliser une tâche extrêmement simple : communiquer des bits entre deux ordinateurs reliés par un câble ou par radio.

Pour relier deux ordinateurs par un câble, et donc réaliser le réseau le plus simple qui soit, on pourrait prolonger le bus de l'un des ordinateurs, afin de le connecter au bus de l'autre. Ainsi, le processeur du premier ordinateur pourrait écrire, avec l'instruction STA, non seulement dans sa propre mémoire, mais aussi dans celle du second. Le processeur du second ordinateur pourrait alors charger, avec l'instruction LDA, la valeur écrite par celui du premier.

La technique utilisée en réalité n'est pas beaucoup plus compliquée : le processeur du premier ordinateur envoie des informations par le bus vers l'un de ses périphériques, *la carte réseau*, qui les transmet par un câble – ou par un autre support physique, par exemple par radio – à la carte réseau du second ordinateur qui les transmet, par le bus, à son processeur.



Transmission point à point

La transmission d'une carte réseau à l'autre met en œuvre un protocole, appelé *protocole physique*. Un exemple de protocole physique est la communication par codes-barres à deux dimensions, comme les pictogrammes *Flashcode* utilisés par les téléphones, où chaque bit est exprimé par un carré noir ou blanc.

Flashcode exprimant le nombre 5412082001000261

Les protocoles physiques qui permettent à deux ordinateurs de communiquer par câble ou par radio ne sont pas très différents. Cependant, au lieu d'utiliser des carrés noirs et blancs, ils représentent les informations par des signaux électromagnétiques, par exemple des variations de longueur d'onde, de phase ou d'intensité d'une onde.

Exercice 16.1

On utilise un lien physique peu fiable : à chaque fois que l'on transmet un 0 ou un 1, la probabilité que ce bit ne soit pas reconnaissable à l'arrivée est 3/10. Pour pallier ce manque de fiabilité, on utilise une forme de redondance (voir le chapitre 12) : quand l'ordinateur émetteur demande à sa carte réseau d'envoyer un 0, cette dernière envoie la suite de bits 0, 1, 0, 0, 1, qui est interprétée par la carte réseau de l'ordinateur récepteur comme un 0. De même, quand l'ordinateur émetteur demande à sa carte réseau d'envoyer un 1, elle envoie la suite de bits 1, 0, 1, 1, 0 qui est interprétée par la carte réseau de l'ordinateur récepteur comme un 1.

1. À partir de combien de bits erronés la suite de cinq bits envoyée n'est-elle plus discernable de l'autre suite ?
2. En déduire la probabilité qu'une suite de cinq bits envoyée ne soit pas reconnaissable à l'arrivée.
3. Quels sont les avantages et inconvénients de cette méthode ?

Sujet d'exposé **Protocoles et codages**

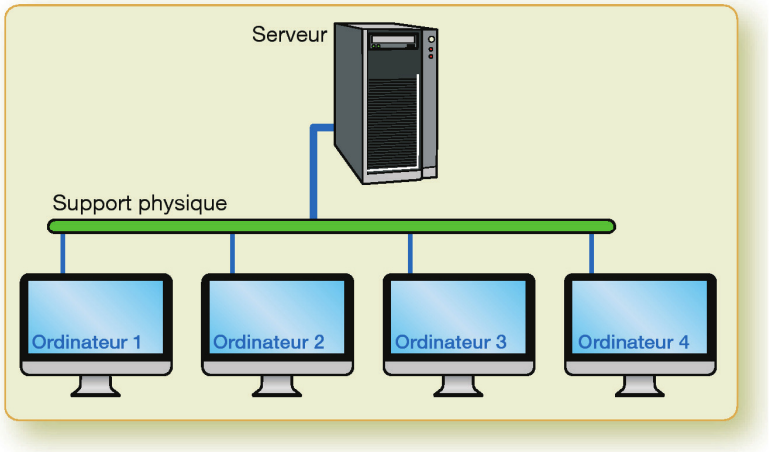
Chercher sur le Web quels sont les protocoles utilisés dans les liaisons RS-232 d'une part, et dans les liaisons USB d'autre part. Quels sont les avantages d'USB par rapport à RS-232 ? Chercher de même ce que sont les codages Manchester d'une part et NRZI d'autre part. Quels sont les avantages du codage Manchester par rapport au codage NRZI ?

Les réseaux locaux : les protocoles de la couche lien

L'étape suivante consiste à construire un réseau local, c'est-à-dire formé de quelques machines connectées, par un protocole physique, à un ordinateur central : un *serveur*. Pour envoyer des informations à un autre ordinateur, chaque ordinateur passe par le serveur.

De même qu'il était nécessaire de distinguer les différentes cases de la mémoire d'un ordinateur en donnant à chacune un nom, son *adresse*, il est nécessaire de distinguer les différents ordinateurs d'un réseau local en donnant à chacun un nom : son *adresse MAC* (*Medium Access Control*). Une adresse MAC est un mot de 48 bits que l'on écrit comme un sextuplet de nombres de deux chiffres en base seize, les seize chiffres s'écrivant 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f : par exemple 10:93:e9:0a:42:ac. Une adresse MAC unique est attribuée à chaque carte réseau au moment de sa fabrication. L'adresse MAC d'une carte réseau, périphérique d'un ordinateur, identifie ce dernier sur le réseau local.

Réseau local



Les protocoles grâce auxquels différents ordinateurs, connectés au même serveur, échangent des informations entre eux sont appelés des *protocoles de la couche lien*. Un protocole simple consiste, pour le serveur, à échanger des informations successivement avec chaque ordinateur du réseau. Ainsi, chaque ordinateur a un créneau de temps pré-établi qui lui est dédié périodiquement pour communiquer avec le serveur, durant lequel le protocole physique épelle bit à bit l'information à transmettre.

Toutefois, les protocoles lien les plus utilisés, par exemple Ethernet et WiFi, utilisent un autre mécanisme qui autorise chaque ordinateur à envoyer des informations au serveur à n'importe quel moment – toujours via un protocole physique. Ce mécanisme évite de réserver un créneau pour un ordinateur qui n'a peut-être rien à communiquer au serveur. Cependant, quand plusieurs ordinateurs envoient des informations en même temps, les messages se brouillent – on dit qu'il y a une *collision entre les messages* – et le serveur n'en reçoit aucun. Pour résoudre ce problème, on utilise une forme de redondance : quand le serveur reçoit les informations envoyées par un ordinateur, il en accuse réception et tant qu'un ordinateur n'a pas reçu d'accusé de réception, il renvoie son message périodiquement. Pour minimiser les risques de nouvelles collisions, ce message est en général renvoyé après un délai de longueur aléatoire. Ce protocole est un exemple d'algorithme qui ne peut fonctionner sans aléa.

Paquet

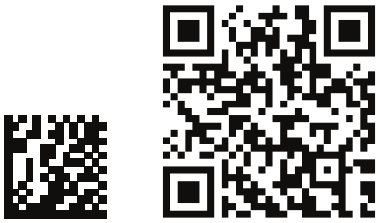
Un *paquet* est une suite de bits structurés selon un certain format, destinée à être échangée sur le réseau.

Le rôle des protocoles de la couche lien est également de définir le format des messages échangés : de même qu'un fichier PGM (voir le chapitre 9) n'est pas simplement une suite de pixels, mais contient aussi d'autres informations – la largeur et la hauteur de l'image, le nombre de niveaux de gris, etc. – et est structuré selon un certain format standard,

les messages qui circulent sur les réseaux, les *paquets*, ne sont pas simplement formés des bits transmis, mais contiennent des informations additionnelles et sont structurés selon un format standard.

Exercice 16.2

Un pictogramme n'est pas simplement une suite de carrés blancs et noirs, mais il contient aussi des informations qui permettent au téléphone d'en trouver les bords et l'orientation. Quelle est la partie du format de chacun de ces pictogrammes qui joue ce rôle ?



En omettant ces délimiteurs et en admettant qu'un carré blanc code un 0 et qu'un carré noir code un 1, combien de pictogrammes différents existe-t-il dans chacun de ces formats ? Comment ce nombre se compare-t-il au nombre de pages existant sur le Web ?

SAVOIR-FAIRE Trouver les adresses MAC des cartes réseau d'un ordinateur

Ouvrir une fenêtre *terminal*. Utiliser la commande `ifconfig` sous Linux, ou `ipconfig /all` sous Windows.

Exercice 16.3 (avec corrigé)

Trouver les adresses MAC des cartes réseau de son ordinateur.

Lorsqu'on tape la commande `ifconfig` ou `ipconfig /all`, différentes informations s'affichent, relatives à la connexion de son ordinateur au réseau. Ces informations sont généralement organisées en paragraphes qui correspondent à différentes cartes réseaux : par exemple un paragraphe qui correspond à la carte Ethernet qui gère la connexion par câble et un autre qui correspond à la carte WiFi qui gère la connexion par radio. On reconnaît dans chacun de ces paragraphes une adresse de la forme 10:93:e9:0a:42:ac : un sextuplet de nombres de deux chiffres en base seize. Il s'agit de l'adresse MAC de chacune de ces cartes.

Exercice 16.4

Noter l'adresse MAC d'une carte réseau de son ordinateur. De quelle connexion réseau s'agit-il précisément ? WiFi ? Ethernet ? Autre ? Éteindre, puis rallumer l'ordinateur. L'adresse a-t-elle changé ?

Le réseau global : les protocoles de la couche réseau

Utiliser un ordinateur central est possible pour un réseau de petite taille, mais cette méthode ne peut pas s'appliquer à un réseau formé de plusieurs milliards d'ordinateurs, comme Internet : d'une part, l'ordinateur central serait vite surchargé, d'autre part s'il tombait en panne, ou s'il était détruit, les communications sur la Terre entière seraient interrompues. C'est pour cela que l'on a inventé d'autres protocoles pour les réseaux de grande taille, appelés *les protocoles réseau*, qui fédèrent les réseaux locaux de proche en proche et utilisent les protocoles lien pour assurer les communications locales. Le plus utilisé des protocoles réseau est le protocole IP (*Internet protocol*).

Avec le protocole IP, chaque machine a une adresse, appelée son *adresse IP*. Contrairement à l'adresse MAC, celle-ci n'est pas associée de manière durable à un ordinateur : quand un ordinateur est remplacé par un autre, le nouveau peut hériter de l'adresse IP de l'ancien. À l'inverse, si un ordinateur est déplacé d'un lieu à un autre, il change d'adresse IP. Les adresses IP classiques (IPv4) sont des mots de 32 bits écrits sous forme d'un quadruplet de nombres compris entre 0 et 255, par exemple 216.239.59.104.

POUR ALLER PLUS LOIN D'IPv4 à IPv6

Il n'y a donc que 2^{32} adresses IPv4 possibles, soit un peu plus de quatre milliards : c'est peu quand on sait que le réseau Internet contient déjà trois milliards d'ordinateurs. Cette pénurie en adresses IP, due à l'imprévoyance des pionniers d'Internet, qui cherchaient seulement à connecter quelques dizaines d'ordinateurs sans anticiper le succès de leur réseau, explique que l'on cherche aujourd'hui à les remplacer par des adresses de 128 bits (IPv6).

SAVOIR-FAIRE Trouver l'adresse IP attribuée à un ordinateur

Ouvrir une fenêtre *terminal*. Utiliser la commande `ifconfig` sous Linux ou `ipconfig /all` sous Windows.

Exercice 16.5 (avec corrigé)

Trouver les adresses IPv4 attribuées à son ordinateur.

Lorsqu'on tape la commande `ifconfig` ou `ipconfig /all`, différentes informations s'affichent, relatives à la connexion de son ordinateur au réseau. On reconnaît parmi ces informations des adresses de la forme 216.239.59.104 : un quadruplet de nombres compris entre 0 et 255. Il s'agit des adresses IPv4 attribuées à chacune des cartes réseaux de l'ordinateur.

Exercice 16.6

Noter les adresses IP attribuées aux cartes réseau de son ordinateur. Éteindre, puis rallumer cet ordinateur. Ces adresses ont-elle changé ?

Exercice 16.7

Soit un réseau connectant cinq ordinateurs : *A*, *B*, *C*, *D* et *E*. Est-il possible d'identifier chaque ordinateur de manière unique avec des adresses de 3 bits ? Décrire un tel plan d'adressage : attribuer une adresse à chaque ordinateur. Est-il également possible d'identifier chaque ordinateur de manière unique avec des adresses de 2 bits ?

Exercice 16.8

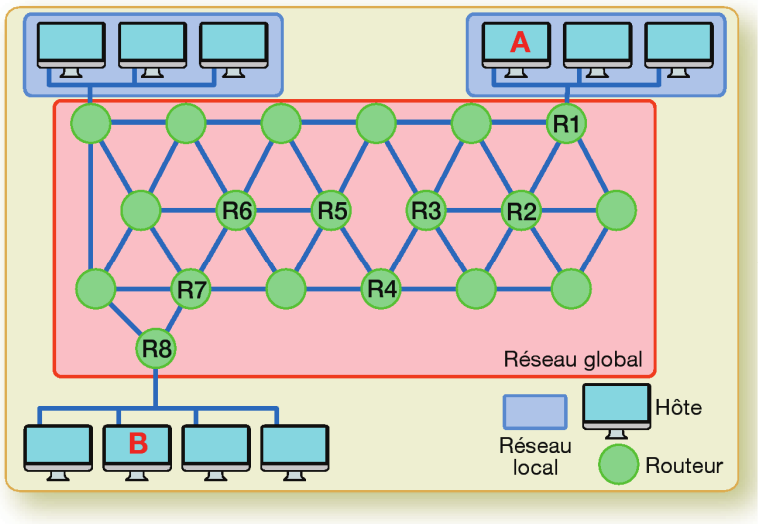
Combien y a-t-il d'adresses IPv4 ? Combien y a-t-il d'adresses IPv6 ? Combien y a-t-il de numéros de téléphone en France ?

Avec les protocoles réseau, la notion de serveur est remplacée par celle de *routeur*. Pour envoyer des informations à une machine dont l'adresse IP est *X*, un ordinateur commence par les envoyer au routeur auquel il est connecté. En fonction de l'adresse *X*, celui-ci l'envoie à un autre routeur, puis à un autre, etc. jusqu'à ce que les informations arrivent à destination. On appelle ce procédé le *routage*.

Routeur

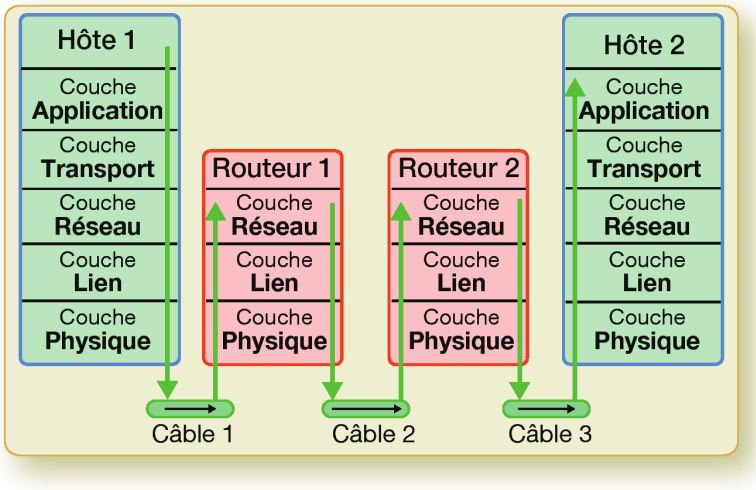
Un *routeur* est un ordinateur dont la seule fonction est d'acheminer des informations sur le réseau. Pour les distinguer des routeurs, on appelle les autres ordinateurs du réseau des *hôtes*.

Les routeurs n'ayant pour fonction que de relayer les paquets en direction de leur destination, ils ne mettent en œuvre que des protocoles des couches physique, lien et réseau, contrairement aux hôtes qui, eux, mettent en œuvre des protocoles de toutes les couches.



Réseau global. Routage de A à B en passant par les routeurs R1, R2, R3, R4, R5, R6, R7 et R8.

La figure suivante illustre le routage d'informations d'un hôte à un autre, en passant par deux routeurs et trois câbles : en chemin, sur chaque machine les traitant, les informations transmises sont successivement encapsulées quand elles passent d'une couche à la couche inférieure, puis décapsulées quand elles passent d'une couche à une couche supérieure.



Les protocoles à l'œuvre lors du routage. Chemin suivi par l'information à travers les couches sur chaque machine.

On peut comparer le routage au système du courrier postal. Si, à Sydney, un Australien expédie une lettre à l'adresse « 2004, route des Lucioles, Valbonne, France », le facteur australien n'apporte pas cette lettre directement à son destinataire, contrairement à ce que faisaient les « facteurs » au XVII^{e} siècle : il la met dans un avion en direction de Hong-Kong ou de Los Angeles, où un autre facteur la met dans un avion en direction de Francfort ou de Paris, où un autre facteur la met dans un avion en direction de Nice, où un facteur l'apporte finalement en camionnette, au 2004 de la route des Lucioles, à Valbonne.

Un routeur, qui reçoit des informations à envoyer à une adresse IP *X*, doit choisir le routeur suivant, en direction de *X*. De même que le facteur australien sait qu'une lettre pour Valbonne, ou plus généralement pour la France, peut passer par Hong-Kong, le routeur a un répertoire, appelé *table de routage*, qui indique que la première étape d'un chemin vers l'adresse *X* est le routeur dont l'adresse IP est *B*, auquel il est directement connecté. Il envoie donc ces informations vers le routeur *B*, qui consulte à son tour sa propre table de routage, et ainsi de suite jusqu'à arriver à l'ordinateur dont l'adresse est *X*. Comme dans un jeu de piste, on ne connaît pas le chemin à l'avance, mais, à chaque étape, on découvre la suivante.

Pour construire et maintenir à jour leurs tables de routage, les routeurs utilisent un *algorithme de routage*, par exemple *l'algorithme de Bellman-Ford* : outre acheminer des informations d'un point à un autre, chaque routeur envoie périodiquement à chaque routeur auquel il est connecté directement par un protocole lien, la liste des adresses IP vers lesquelles il connaît un chemin, dont bien entendu sa propre adresse, et la longueur

de ce chemin. Grâce à cet algorithme et au fur et à mesure qu'il reçoit des listes, chaque routeur découvre peu à peu l'état du réseau et, ce faisant, les chemins les plus courts pour atteindre les destinations présentes dans le réseau. Ainsi, un routeur qui reçoit du routeur *B* l'information que *B* a reçu du routeur *C* l'information que *C* est directement connecté à l'ordinateur *X*, met à jour sa table de routage en mémorisant qu'un chemin pour accéder à l'ordinateur *X*, en passant par deux intermédiaires, commence par le routeur *B*.

Cette mise à jour dynamique des tables de routage est ce qui donne sa souplesse au réseau Internet. Si par exemple le câble entre les routeurs *B* et *C* est détruit, les tables de routage se mettent graduellement à jour en découvrant d'autres chemins pour acheminer les informations vers la machine *X*. À l'origine, cette idée servait à donner une robustesse au réseau en temps de guerre : tant que deux points restaient connectés par un chemin dans le graphe des routeurs, il leur était possible de communiquer. Par la suite, on s'est rendu compte que cette souplesse était de toutes façons nécessaire dans un réseau de trois milliards d'ordinateurs, où il est inévitable que des ordinateurs soient en permanence ajoutés ou supprimés du réseau.

Si les protocoles réseau sont très souples, ils sont en revanche peu fiables : quand un routeur est saturé par trop d'informations à transmettre, il en détruit tout simplement une partie. En outre, quand des informations ont été envoyées d'un routeur à un autre trop longtemps sans arriver à destination, elles sont également détruites.

SAVOIR-FAIRE Déterminer le chemin suivi par l'information

Dans une fenêtre *terminal*, la commande `ping` suivie d'un nom de domaine affiche l'adresse IP associée à ce nom de domaine. La commande `traceroute` sous Linux ou `tracert` sous Windows, suivie d'une adresse IP, affiche les routeurs d'un chemin menant de son ordinateur à celui dont l'adresse IP est indiquée.

Exercice 16.9 (avec corrigé)

Trouver un chemin entre son ordinateur et l'ordinateur associé au nom de domaine www.google.fr.

Dans une fenêtre terminal, *on tape la commande* `ping www.google.fr` *et une adresse IP s'affiche, par exemple 173.194.34.55. C'est l'adresse d'un ordinateur de Google. On tape ensuite la commande* `traceroute 173.194.34.55` *ou* `tracert 173.194.34.55` *et une liste s'affiche des routeurs qui forment un chemin entre sa machine et celle de Google.*

SAVOIR-FAIRE Déterminer l'adresse IP du serveur par lequel un ordinateur est connecté à Internet

Dans une fenêtre *terminal*, la commande `netstat -r` sous Linux ou `ipconfig /all` sous Windows affiche différentes informations relatives à la connexion de son ordinateur au réseau, en particulier la *passerelle par défaut* (*default gateway*) qui est le routeur par lequel cet ordinateur est connecté à Internet.

Exercice 16.10 (avec corrigé)

Déterminer l'adresse IP du routeur par lequel son ordinateur est connecté à Internet.

On tape la commande `netstat -r` ou `ipconfig /all`. Dans les informations affichées, on recherche la passerelle par défaut (default gateway) *indiquée, qui est le routeur. Ce dernier est identifié soit directement par son adresse IP, soit par un nom de la forme `gw.adalovelace.fr` ; dans ce second cas, on peut trouver l'adresse IP associée à ce nom avec la commande `ping`.*

Exercice 16.11

Un algorithme de routage. On attribue à chaque élève une nouvelle adresse mail, par exemple `anonyme1@adalovelace.fr`, `anonyme2@adalovelace.fr`, etc. Chaque élève garde cette adresse secrète et il la marque sur une feuille de papier. Ces feuilles sont mélangées dans un chapeau et chacun tire une adresse en s'assurant qu'il ne tire pas la sienne propre. Chaque élève a uniquement le droit d'envoyer des courriers à l'adresse qu'il a tirée dans le chapeau et de répondre à l'envoyeur de tout courrier qu'il reçoit à son adresse .

Chaque élève envoie un courrier à l'adresse qu'il a tirée, dans lequel il indique son propre nom. Par exemple, Alice enverrait :

```
sujet : Hello
corps du message :
Alice (par -)
```

Chaque élève construit une table de routage qui indique, pour chaque élève dont il a entendu parler, l'adresse de la personne par qui il en a entendu parler pour la première fois. Par exemple, la table de routage d'Alice peut avoir cette forme :

```
Alice (par -)
Djamel (par anonyme2@adalovelace.fr)
Frédérique (par anonyme2@adalovelace.fr)
Hector (par anonyme7@adalovelace.fr)
```

À chaque fois qu'un élève reçoit un message, il le lit, met à jour sa table de routage et y répond en copiant, dans le corps du message l'état de sa table de routage mise à jour. Alice enverrait par exemple :

```
sujet : Hello
corps du message :
Alice (par -)
```

```
Djamel (par anonyme2@adalovelace.fr)
Frédérique (par anonyme2@adalovelace.fr)
Hector (par anonyme7@adalovelace.fr)
```

Au bout d'une dizaine de minutes, on arrête d'envoyer et de répondre aux messages.

On passe alors à la seconde phase de l'exercice : un élève essaie d'envoyer un véritable message à un autre élève. Pour cela, il envoie son message à l'adresse correspondant au nom de son destinataire dans sa table de routage.

Par exemple :

Alice envoie à l'adresse `anonyme2@adalovelace.fr` le message suivant :

```
Sujet : message pour Frédérique
Corps du message : N'oublie pas de me rapporter l'exemplaire de
Madame Bovary que je t'ai prêté il y a six mois. Merci. Alice.
```

Si un élève autre que son destinataire reçoit ce message, il le renvoie à l'adresse correspondant au nom de son destinataire dans sa propre table de routage, et ainsi de suite jusqu'à ce que le message arrive à destination.

À quelle condition un message arrive-t-il bien à son destinataire ?

La régulation du réseau global : les protocoles de la couche transport

Les programmes que l'on utilise tous les jours, par exemple les navigateurs web ou les programmes de gestion du courrier électronique, ne peuvent pas directement utiliser le protocole IP, principalement pour deux raisons : d'une part, le protocole IP ne permet de transférer d'un ordinateur à un autre que des informations de taille fixe : des paquets contenant au maximum 1 500 octets. D'autre part, comme on l'a vu, IP est un protocole peu fiable : dès qu'un serveur est surchargé, il détruit des informations qui n'arrivent donc pas à leur destinataire. On utilise donc un type supplémentaire de protocole : *les protocoles de transport*, dont le plus utilisé est le protocole TCP (*Transmission Control Protocol*). Les protocoles de transport utilisent les protocoles réseau pour acheminer des informations contenues dans des paquets IP, d'un bout à l'autre du réseau, et assurent que tout paquet IP envoyé est arrivé à bon port.

Pour poursuivre la comparaison avec le service postal, si on attache une importance particulière à une lettre, on l'envoie en recommandé et on attend un accusé de réception, qui permet de savoir que sa lettre a bien été reçue. La couche transport s'assure que la communication a bien lieu de bout en bout, comme prévu.

Comme les protocoles de la couche lien, TCP utilise une forme de redondance pour fiabiliser les communications. Il utilise les protocoles réseaux pour envoyer des paquets IP d'un ordinateur à un autre et pour envoyer un accusé de réception du destinataire à l'expéditeur. Tant que l'accusé de réception n'arrive pas, le même paquet est renvoyé périodiquement. Si trop d'accusés de réception n'arrivent pas, TCP ralentit la cadence d'envoi des paquets pour s'adapter à une congestion éventuelle du réseau global, puis ré-accélère la cadence quand les accusés de réception arrivent.

Une autre fonction de TCP est de découper les informations à transmettre en paquets de 1 500 octets. Par exemple, pour envoyer une page web de 10 000 octets, TCP découpe ces 10 000 octets en paquets plus petits, chacun de taille standard, et les envoie l'un après l'autre. À l'autre bout du réseau, quand tous ces paquets standards sont arrivés, TCP les remet dans l'ordre et ré-assemble leurs contenus pour reconstruire la page web.

ALLER PLUS LOIN Le port

Comme plusieurs programmes peuvent utiliser TCP en même temps et sur la même machine, TCP attribue à chacun d'eux un numéro : *un port*. Un numéro de port permet à TCP de distinguer les paquets correspondant aux différents programmes qui communiquent en même temps.

Exercice 16.12

Taper la commande `ping www.google.fr` dans une fenêtre *terminal* et observer ce qui se passe. En déduire la valeur d'un temps d'attente adéquat après lequel TCP devrait considérer que l'accusé de réception d'un paquet envoyé à `www.google.fr` est perdu. Quelle serait la conséquence d'utiliser un temps d'attente plus court ? Quelle serait la conséquence d'utiliser un temps d'attente plus long ?

Exercice 16.13

On suppose que la durée à attendre entre l'envoi d'un paquet et la réception d'un accusé de réception pour ce paquet est 1 seconde en moyenne, et que les paquets peuvent chacun contenir jusqu'à 1 500 octets de données. On considère les deux configurations suivantes pour TCP.

- Configuration A. TCP est configuré pour n'envoyer un nouveau paquet qu'après avoir reçu l'accusé de réception du paquet précédent.
- Configuration B. TCP est autorisé à envoyer des grappes de 20 paquets à la suite et à accuser réception avec un seul accusé de réception pour toute la grappe, au lieu d'un accusé de réception pour chaque paquet. En conséquence, un paquet perdu entraîne l'absence d'accusé de réception pour la grappe à laquelle il appartient, et donc demande de renvoyer la grappe entière au lieu du seul paquet perdu.

1. Combien de temps faut-il au minimum pour envoyer 1 Mo à destination avec la configuration A ? Avec la configuration B ? Pour simplifier, on suppose que le temps passé à envoyer 20 paquets à la suite est négligeable par rapport à 1 seconde.
2. On suppose maintenant qu'en moyenne, 1% des paquets envoyés vers une destination se perdent en chemin. En moyenne, combien de paquets faudra-t-il faire transiter sur le réseau pour transférer le fichier de 1 Mo à destination avec la configuration A ? Avec la configuration B ? Pour simplifier, on considère qu'à la deuxième fois qu'on envoie un paquet, il arrive à destination à coup sûr, et qu'un accusé de réception envoyé arrive également à coup sûr.
3. Quels sont les avantages et inconvénients de la configuration B par rapport à la configuration A ?

Exercice 16.14

Sachant que chaque paquet envoyé par TCP est identifié avec un numéro de séquence exprimé sur 4 octets, et en admettant que chaque paquet envoyé peut contenir 1 500 octets de données à transmettre, quelle est la taille maximum d'un fichier à transmettre à partir de laquelle TCP devrait réutiliser un numéro de séquence déjà utilisé au début de l'envoi de ce même fichier ? Même question si on décompte des 1 500 octets les informations de contrôle nécessaires au fonctionnement de chaque couche, par exemple 20 octets d'en-tête TCP, 24 octets d'en-tête IP et 34 octets d'en-têtes WiFi et couche physique.

Exercice 16.15

Chercher sur le Web ce qu'est le protocole UDP. Quelles sont les différences entre UDP et TCP ? Quels sont les principaux avantages et inconvénients de chacun de ces protocoles ? Citer des exemples de programmes qui utilisent UDP, d'autres qui utilisent TCP, et expliquer ces choix.

Programmes utilisant le réseau : la couche application

Les protocoles des couches physique, lien, réseau et transport fournissent le socle d'Internet : ils permettent de transmettre de manière fiable des fichiers de toutes tailles, d'une machine à n'importe quelle autre machine connectée à Internet. En plus de ce socle, on distingue néanmoins un dernier type de protocoles, qui utilisent les services de la couche transport pour le compte de certains programmes que l'on utilise tous les jours, comme les navigateurs web ou les logiciels de courrier électronique. Il s'agit des *protocoles d'application*.

Les logiciels de courrier électronique utilisent par exemple le protocole d'application SMTP (S*imple Mail Transfer Protocol*), les navigateurs web utilisent le protocole d'application HTTP (*HyperText Transfer Protocol*) etc. Un autre protocole d'application important est DNS (*Domain Name System*) qui, aux adresses IP, associe des *noms de domaines* comme www.moi.fr.

Quand un navigateur cherche à accéder à une page web située sur un autre ordinateur, il utilise DNS pour trouver l'adresse IP de l'ordinateur hôte sur lequel cette page web se trouve, puis le protocole HTTP pour demander cette page à l'ordinateur hôte. Si cette page est par exemple la page d'accueil d'un annuaire électronique, elle contiendra des champs à remplir, et c'est à nouveau le protocole HTTP qui acheminera les infor-

Sujet d'exposé DNS

Présenter les principes de base de DNS.

mations renseignées vers l'ordinateur hôte, qui, en fonction de ces informations, renverra en général une autre page avec la réponse à la requête.

Au bout du compte, pour transférer une page web d'un ordinateur à un autre, HTTP confie cette page web au protocole TCP, qui la découpe en paquets et confie chaque paquet au protocole IP, qui choisit un lien à utiliser en direction de la destination, puis confie chaque paquet au protocole de lien en vigueur sur ce dernier, par exemple WiFi, qui le confie enfin au protocole physique, qui gère l'acheminement des bits codant ces paquets à travers ce lien. Chaque protocole, à son niveau, contribue à la communication. Chaque protocole est simple ; c'est de leur interaction qui naît la complexité.

Quelles lois s'appliquent sur Internet ?

Jusqu'au milieu du XX^{e} siècle, quand un livre ou un journal était publié, il l'était dans un pays particulier et sa publication était régie par les lois de ce pays. Quand un objet était vendu, il l'était dans un pays particulier et cette vente était régie par les lois de ce pays. Ainsi, la publication de certains textes ou la vente de certains objets était autorisée dans certains pays, mais interdite dans d'autres.

Parce que c'est un réseau mondial, Internet permet de publier, dans un pays, des textes qui peuvent être lus dans le monde entier et, de même, de vendre des objets qui peuvent être achetés dans le monde entier. Dès lors, quelle loi appliquer ? À cette question, qui est nouvelle, plusieurs réponses ont été imaginées, sans que personne ne sache encore laquelle s'imposera sur le long terme :

- l'application des lois du pays dans lequel le texte est publié ou l'objet mis en vente,
- l'application des lois du pays dans lequel le texte peut être lu ou l'objet acheté – ce qui obligerait, par exemple, un hébergeur de site web à bloquer l'accès à certains sites depuis certains pays,
- ou l'émergence d'un minimum de règles universelles.

SUJET D'EXPOSÉ

L'affaire LICRA contre Yahoo!

Qu'est-ce que l'article R.645-1 du Code Pénal français ? Qu'est-ce que le premier amendement de la Constitution des États-Unis ? En quoi sont-ils contradictoires ? Chercher des documents sur l'*affaire de la LICRA contre Yahoo!*, relative à la vente aux enchères d'objets nazis sur Internet (de l'ordonnance du Tribunal de Grande Instance de Paris du 20 novembre 2000, jusqu'à la décision de la Cour Suprême des États-Unis du 30 mai 2006). Montrer en quoi cette affaire illustre le problème de l'application de législations différentes selon les pays pour la vente sur Internet.

Qui gouverne Internet ?

Quelques règles d'organisation d'Internet doivent être universellement acceptées. Par exemple, pour que les ordinateurs du monde entier puissent communiquer, il est nécessaire qu'ils utilisent les mêmes protocoles et pour qu'il n'y ait pas de confusion entre les adresses IP, il faut que deux ordinateurs distincts n'aient jamais la même adresse. Internet n'est donc pas entièrement décentralisé : un petit nombre de décisions doivent être prises en commun.

Ici, plusieurs modes d'organisation sont en concurrence, à nouveau sans que personne ne sache lequel s'imposera sur le long terme :

- l'émergence d'organisations internationales régies par des traités entre États,
- l'émergence d'organisations internationales informelles, dont la légitimité vient uniquement de la confiance qui leur est accordée,
- l'émergence d'organisations propres aux pays où Internet est le plus développé (cette dernière solution ayant l'inconvénient d'augmenter les différences de développement d'Internet entre les pays).

Sujet d'exposé **Que sont…**

- l'*Internet Engineering Task Force* (IETF),
- l'*Internet Corporation for Assigned Names and Numbers* (ICANN),
- l'*Internet Society* (ISOC),
- le *World Wide Web Consortium* (W3C),
- le *WhatWG*,
- et l'*Union Internationale des Télécoms* (UIT) ?

Quel est le rôle et quel est le statut de chacune de ces organisations ?

Aller plus loin **Calculer dans les nuages (*cloud computing*)**

Au cours de l'histoire de l'informatique, des modes centralisatrices et décentralisatrices se sont succédées. Ainsi, jusqu'aux années 1970, les entreprises n'avaient qu'un seul ordinateur, auquel étaient connectés de nombreux terminaux, par exemple formés d'un clavier et d'un écran, qui permettaient à différentes personnes d'effectuer des calculs sur cet ordinateur. À partir des années 1980, ces terminaux ont été remplacés par des micro-ordinateurs, connectés par un réseau à un serveur. Les calculs n'étaient alors plus effectués par un ordinateur central, mais par chacun de ces micro-ordinateurs.

Depuis le milieu des années 2000, on voit apparaître un retour de la centralisation. Des entreprises, qui vendaient naguère des programmes à des clients qui les utilisaient pour effectuer des calculs sur leurs ordinateurs, proposent désormais à ces mêmes clients d'effectuer elles-mêmes ces calculs à leur place. Les clients ont juste besoin de communiquer leurs données à ces entreprises, qui font les calculs sur leurs propres ordinateurs et envoient le résultat de ces calculs à leurs clients. C'est ce qu'on appelle le *calcul dans les nuages (cloud computing)*.

Par exemple, au lieu d'installer un logiciel de courrier électronique sur son ordinateur et de l'utiliser pour envoyer des courriers, on peut, à chaque fois que l'on souhaite envoyer un courrier, se connecter à un ordinateur distant, en général au moyen d'une page web, et communiquer le texte de son courrier à cet ordinateur, qui se chargera de l'envoyer ; c'est l'idée du *Webmail*. De même, au lieu d'acheter un logiciel de comptabilité et de l'utiliser, une entreprise peut se connecter, à chaque fois qu'elle souhaite effectuer une opération comptable, à une machine distante qui effectue cette opération pour l'entreprise. L'ordinateur local ne sert plus qu'à communiquer des informations à cette machine distante, comme jadis les terminaux.

Utiliser des programmes sur une machine distante simplifie beaucoup de choses. Il n'est plus nécessaire d'installer des logiciels sur son ordinateur, de les mettre à jour de temps en temps, etc. De plus, il devient possible d'envoyer un courrier depuis n'importe quel ordinateur : d'un web café de Bogota ou de Caracas, comme de son bureau. Pour une entreprise, cela permet de diminuer la taille du service informatique, puisqu'il lui suffit désormais d'avoir quelques ordinateurs reliés au réseau.

Cependant, cette évolution présente aussi un risque de dépossession des utilisateurs de leur pouvoir : au lieu d'avoir ses programmes, ses courriers, ses photos, sa comptabilité, etc. sur son ordinateur, on préfère les confier à des entreprises et des ordinateurs distants. En outre, on a parfois une garantie assez faible de leur conservation sur le long terme, de son pouvoir de les effacer ou de son contrôle sur les usages que ces entreprises peuvent faire de ces données.

Ai-je bien compris ?

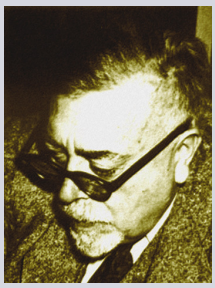
- Qu'est-ce qu'un protocole ? Qu'est-ce qu'une couche ?
- Quelles sont les cinq couches de protocoles dont sont composés les réseaux ?
- Comment fabriquer un protocole fiable en utilisant un protocole peu fiable ?

Les robots 17

CHAPITRE AVANCÉ

Un robot ? C'est un ordinateur à deux roues.

Dans ce chapitre, nous introduisons de nouveaux objets : les robots, qui sont essentiellement des ordinateurs munis de capteurs et d'actionneurs. Nous voyons comment les grandeurs captées sont numérisées et comment le principe de la boucle fermée permet de contrôler une action. Enfin, nous voyons comment programmer un robot à l'aide d'une boucle infinie dans laquelle les capteurs sont interrogés et les actionneurs activés.



Norbert Wiener (1894-1964) est le fondateur, à la fin des années 1940, de la science du pilotage ou *cybernétique*. Entouré d'un groupe interdisciplinaire de mathématiciens, logiciens, anthropologues, psychologues, économistes…, il a cherché à comprendre les processus de commande et de communication chez les êtres vivants, dans les machines et dans les sociétés. Le concept central de la cybernétique est celui de causalité circulaire ou contrôle en boucle fermée (*feedback*).

Les composants d'un robot

Comme un ordinateur ou un téléphone, un robot est formé d'un processeur, d'une mémoire et de périphériques. Ces derniers se divisent en périphériques de sortie, ou *actionneurs*, qui permettent au robot de se mouvoir et d'agir sur son environnement, et ses périphériques d'entrée, ou *capteurs*, qui lui permettent d'analyser cet environnement.

Dans ce chapitre, nous utilisons le robot mOway, mais les connaissances que nous présentons ne sont pas propres à ce robot et peuvent facilement se transposer à d'autres.

Les actionneurs du robot mOway sont deux moteurs qui font tourner ses roues et diverses diodes électroluminescentes que l'on peut allumer ou éteindre. Quand on fait tourner les deux roues du robot à la même vitesse, il avance en ligne droite. Quand on ralentit ou accélère une roue par rapport à l'autre, il tourne.

Les capteurs du robot mOway sont les suivants :

1 

2 

3 

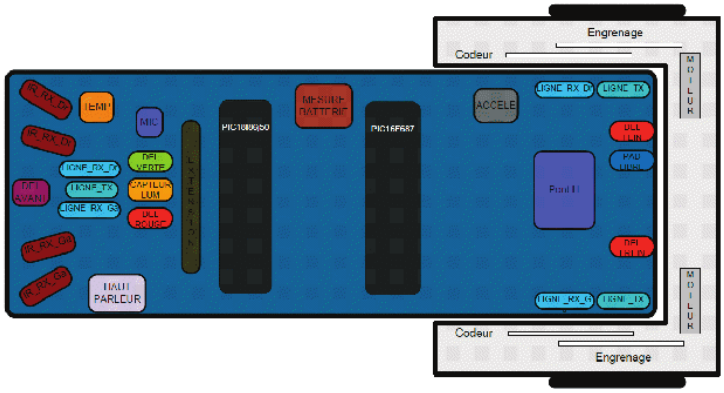
4 

- Quatre détecteurs d'obstacles 1 qui émettent régulièrement de brèves impulsions de lumière infrarouge. Quand le robot est proche d'un obstacle, cette lumière est réfléchie par l'obstacle et détectée par le robot.
- Un capteur de luminosité 2, qui identifie la direction et l'intensité d'une source lumineuse.
- Deux capteurs de couleur de sol 3, qui analysent la réflexion d'une lumière infrarouge sur le sol et permettent, par exemple, d'y repérer une ligne.
- Un capteur dont la résistance électrique varie avec la température.
- Un microphone qui détecte la présence ou l'absence d'un son dont la fréquence est comprise entre 100 Hz et 20 kHz, et aussi l'intensité de ce son.
- Un accéléromètre qui mesure l'accélération linéaire du robot et la gravité. C'est le même composant que sur les manettes de jeux et certains téléphones : une accélération déforme deux plaques souples d'un condensateur, ce qui fait varier sa capacité. Cet accéléromètre indique la direction verticale, ce qui permet, par exemple, de savoir si le robot s'est retourné.

Il est possible d'ajouter d'autres périphériques 4, grâce à des bus similaires à ceux que nous avons décrits au chapitre 15.

Le robot contient une batterie rechargeable qui le rend autonome, mais il peut dialoguer avec un ordinateur par radio ou par un réseau WiFi (voir le chapitre 16).

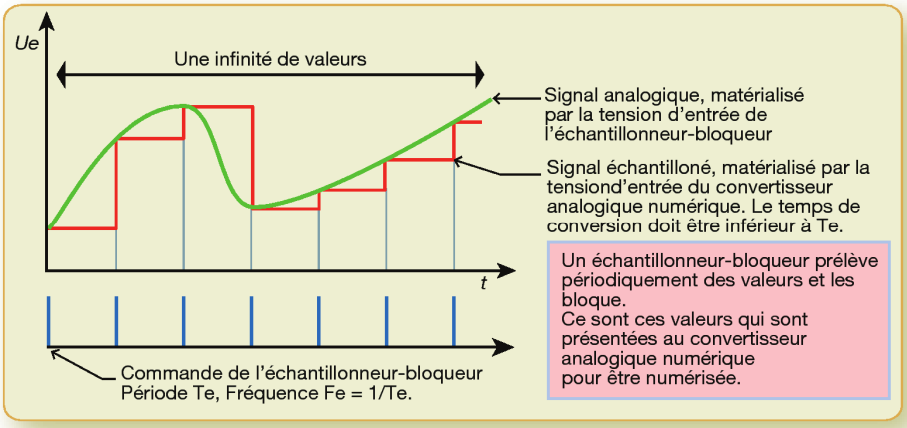
Tous les capteurs et actionneurs sont raccordés à deux *micro-contrôleurs*. Un micro-contrôleur est un circuit qui contient plusieurs composants, en particulier un processeur et de la mémoire (voir le chapitre 15). Le micro-contrôleur principal, appelé PIC18F86J50 sur la figure ci-après, exécute un programme chargé dans sa mémoire. Nous reviendrons plus tard sur le rôle du micro-contrôleur secondaire.



La numérisation des grandeurs captées

Les capteurs mesurent des grandeurs physiques : intensité lumineuse, intensité sonore, etc. et expriment en général ces grandeurs sous la forme d'une tension électrique. Pour que ces grandeurs puissent être utilisées par un processeur, cette tension doit, à son tour, être exprimée par un nombre représenté en binaire, comme le sont le niveau de gris d'un pixel quand on numérise une image ou la pression quand on numérise un son (voir le chapitre 9). Pour cela, deux composants sont utilisés : un *échantillonneur-bloqueur* et un *convertisseur analogique-numérique*. Le premier bloque la tension à une valeur stable, pendant que le second mesure cette valeur et produit un nombre, représenté en binaire, qui est la valeur de cette tension.

Par exemple, quand la valeur analogique est comprise entre 0 et 5 V et la valeur numérique comprise entre 0 et 1 023 (c'est-à-dire représentée sur 10 bits), la tension est mesurée par pas de 5 / 1 023 = 4,88 mV. Cette



valeur est appelée la *résolution* du convertisseur. Pour trouver la valeur numérique d'une tension, un convertisseur analogique-numérique n'utilise que des comparaisons entre cette tension et des multiples entiers de la résolution en procédant par dichotomie (voir le chapitre 20).

Exercice 17.1

Si un convertisseur analogique-numérique transforme une tension comprise entre 0 et 5 V en un nombre compris entre 0 et 1 023, à quelles valeurs analogiques correspondent les valeurs numériques 512, 256 et 768 ?

Exercice 17.2

Si un convertisseur analogique-numérique transforme une tension comprise entre 0 et 5 V en un nombre compris entre 0 et 1 023, en procédant par dichotomie, combien de comparaisons sont nécessaires pour déterminer la valeur numérique correspondante ?

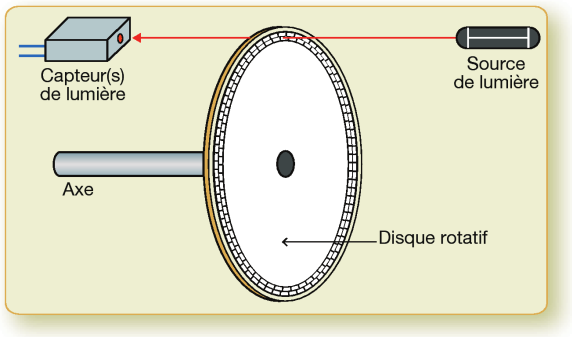
Exercice 17.3

Si on numérise une tension comprise entre 0 et 5 V, définie avec une précision de 20 mV, par un nombre compris entre 0 et 1 023, tous les bits de la valeur numérique ont-ils une signification ? Sur combien de bits suffirait-il de numériser cette valeur ?

Le contrôle de la vitesse : la méthode du contrôle en boucle fermée

Pour faire tourner un moteur à une vitesse déterminée, il ne suffit pas de fixer la tension d'alimentation du moteur, car la vitesse dépend aussi de la masse du robot, de la nature du terrain sur lequel le robot se déplace, des conditions climatiques si le robot est à l'extérieur, etc. La méthode appelée *contrôle en boucle fermée* utilise un capteur pour mesurer la vitesse du moteur, compare cette dernière à la vitesse souhaitée et réajuste la commande du moteur en fonction de l'écart constaté : si cette vitesse est inférieure à la vitesse souhaitée, on augmente la tension d'alimentation du moteur, si elle est supérieure, on la diminue. Mesurer en permanence la vitesse des moteurs et adapter leur tension d'alimentation en fonction de l'écart, par rapport à la consigne, est le rôle du micro-contrôleur secondaire, appelé PIC16F687 sur la figure.

Pour mesurer la vitesse de la roue du robot, on utilise un capteur de vitesse formé d'un disque qui alterne des zones opaques et transparentes, fixé sur l'axe du moteur et éclairé par une source de lumière. On calcule la vitesse du moteur en comptant le nombre de fois que la lumière est occultée par unité de temps. La fréquence de ce clignotement est proportionnelle à la vitesse. On utilise donc un circuit qui convertit cette fréquence en valeur de vitesse, de façon à pouvoir la contrôler.



Exercice 17.4

Pour le contrôleur de vitesse que l'on vient de décrire, quelle est la vitesse de la roue en fonction de la fréquence de clignotement, du nombre d'encoches sur la roue et du rayon de la roue ? Pour une roue de 2 cm de rayon et contenant 64 encoches, quelle vitesse correspond à la fréquence 128 Hz ? Quelle fréquence correspond à la vitesse 0,4 m/s ? Expliquer comment on peut exprimer la vitesse sous forme numérique, en utilisant un compteur numérique qui augmente d'une unité à chaque impulsion lumineuse.

Programmer un robot : les actionneurs

On programme le robot mOway en chargeant dans sa mémoire un programme, depuis un ordinateur ordinaire. Comme on l'a vu, ce programme est ensuite exécuté par le microcontrôleur principal. Ce programme doit être écrit, non en Java, mais en C. Cependant, le fragment de C, utilisé dans ce chapitre, est très similaire à Java.

On commence donc par écrire un programme et le compiler. On utilise pour cela l'environnement de développement *MPLAB*. On le transmet ensuite au robot à l'aide d'un câble USB et du programme *mOwayGUI* (*mOway Graphic User Interface*).

Pour écrire un programme, on utilise des fonctions qui interrogent les capteurs et commandent les actionneurs. Ces fonctions ne font pas partie du langage C lui-même, mais d'une extension de C fournie par le fabricant du robot. On indique que l'on utilise cette extension en ajoutant au début de son programme les commandes :

```
#include "lib_mot_moway.h"
#include "lib_sen_moway.h"
```

Pour faire avancer le robot, on utilise la fonction `MOT_STR`. Par exemple, l'instruction `MOT_STR(50,FWD,TIME,100);` fait avancer le robot à la vitesse 50, en marche avant, pendant 10 secondes (100 dixièmes de seconde).

Le premier argument de cette fonction est la vitesse à laquelle on fait avancer le robot. Il est compris entre 0 et 100 ; 0 correspond à 0 cm/s, 25 à 11 cm/s, 50 à 13 cm/s, 75 à 15 cm/s et 100 à 17 cm/s. Le deuxième, `FWD` ou `BACK`, définit le sens de la marche : avant ou arrière. Le troisième argument indique si l'on souhaite spécifier une durée ou une distance. Dans ce chapitre, nous spécifierons toujours une durée et cet argument sera `TIME`. Le quatrième argument, compris entre 0 et 255, est la durée du mouvement exprimée en dixièmes de secondes. On peut aussi faire avancer le robot pendant un temps infini en donnant, conventionnellement, la valeur 0 comme durée.

De même, pour faire tourner le robot, on utilise la fonction `MOT_ROT`. Par exemple, l'instruction `MOT_ROT(25,FWD,CENTER,LEFT,ANGLE,50);` fait tourner le robot à gauche, à la vitesse angulaire 25, d'un angle de 180 degrés (50 centièmes de tour).

Le premier argument de cette fonction est la vitesse angulaire à laquelle on fait tourner le robot. Il est compris entre 0 et 100 ; 0 correspond à

ALLER PLUS LOIN **Un programme sans fin**

Contrairement à beaucoup de programmes qui calculent un résultat et se terminent, un programme commandant un robot, un téléphone, un réseau et, plus généralement, un objet qui interagit avec son environnement, doit ne jamais se terminer, car le robot ou le téléphone ne cessent jamais d'interagir avec leur environnement. Quand un tel programme se termine, on dit que le robot ou le téléphone est *en panne*, car il ne répond plus aux sollicitations de son environnement et, bien souvent, on relance ce programme.

0 tour/s, 25 à 0,52 tour/s, 50 à 0,62 tour/s, 75 à 0,73 tour/s et 100 à 0,80 tour/s. Les deux arguments suivants permettent de choisir si l'on fait tourner le robot autour de son centre ou autour de l'une de ses roues. Dans ce chapitre, nous le ferons toujours tourner autour de son centre et ces arguments seront toujours `FWD` et `CENTER`. Le quatrième argument, `LEFT` ou `RIGHT`, définit le sens de rotation du robot : vers la gauche ou vers la droite. Le cinquième argument peut prendre deux valeurs, `ANGLE` ou `TIME` ; il indique si l'on souhaite spécifier l'angle de rotation ou la durée de la rotation. Le sixième argument est, en fonction de la valeur du cinquième, l'angle de la rotation exprimé en centièmes de tour ou sa durée exprimée en dixièmes de secondes. Il est compris entre 0 et 100 dans le premier cas et entre 0 et 255 dans le second. À nouveau, on peut aussi faire tourner le robot pendant un temps infini, en donnant, conventionnellement, la valeur 0 comme durée.

Quand on exécute une telle instruction `MOT_STR` ou `MOT_ROT`, on initie un mouvement, puis on passe à l'instruction suivante. Le robot continue alors son mouvement jusqu'à la fin, à moins que ce mouvement ne soit interrompu par l'initiation d'un autre mouvement.

En effet, si la poursuite de l'exécution du programme initie un second mouvement, alors le premier mouvement est interrompu. Par exemple, quand on exécute l'instruction :

```
MOT_STR(50,FWD,TIME,100);
MOT_ROT(25,FWD,CENTER,LEFT,ANGLE,50);
```

on commence par exécuter l'instruction `MOT_STR(50,FWD,TIME,100);` qui initie un mouvement rectiligne de 10 s, puis on exécute tout de suite la seconde instruction `MOT_ROT(25,FWD,CENTER,LEFT,ANGLE,50);` qui interrompt le mouvement en cours et initie un mouvement de rotation d'un angle de 180 degrés. Le mouvement rectiligne est interrompu quelques fractions de secondes seulement après avoir été initié. Il n'est donc pas effectué.

Si on souhaite effectuer le mouvement rectiligne en entier, avant de passer à la rotation, on doit utiliser la variable `MOT_END` qui prend la valeur 0 (équivalent de `false` en C) quand le robot est en mouvement et la valeur 1 (équivalent de `true` en C) quand le robot est immobile. Ainsi, quand on exécute l'instruction :

```
MOT_STR(50,FWD,TIME,100);
while(!MOT_END){}
MOT_ROT(25,FWD,CENTER,LEFT,ANGLE,50);
```

on initie un mouvement rectiligne de 10 s, on attend que le robot redevienne immobile, c'est-à-dire que le mouvement rectiligne soit achevé, puis on initie le mouvement de rotation.

Pour allumer et éteindre la diode électroluminescente, située à l'avant du robot, on utilise les instructions `LED_FRONT_ON();` et `LED_FRONT_OFF();`. Pour faire clignoter la diode électroluminescente verte, située sur le robot, on utilise l'instruction `LED_TOP_GREEN_ON_OFF();` Ici, le fait d'allumer une diode n'interrompt pas le mouvement du robot, les deux actions sont effectuées en même temps.

Pour attendre quelques secondes entre deux instructions, on utilise la fonction `Delay10KTCYx`, par exemple l'instruction `Delay10KTCYx(200);` interrompt l'exécution du programme pendant 2 s. L'argument de cette fonction est le temps du délai exprimé en centièmes de seconde.

Enfin, il faut exécuter au début de chaque programme les instructions `SEN_CONFIG();` et `MOT_CONFIG();` pour configurer les capteurs et les moteurs.

Par exemple, le programme :

```
main () {
  SEN_CONFIG();
  MOT_CONFIG();
  Delay10KTCYx(200);
  LED_TOP_GREEN_ON_OFF();
  MOT_STR(50,FWD,TIME,100);
  while(!MOT_END) {}
  MOT_ROT(25,FWD,CENTER,LEFT,ANGLE,50);
  while(1) {}}
```

attend 2 s avant de commencer, fait clignoter la diode verte, fait avancer le robot pendant 10 s, le fait tourner de 180 degrés et boucle à l'infini, afin que le programme ne se termine pas. Dans ce cas, la non-terminaison est un peu artificielle et sert surtout à éviter, comme on l'a expliqué, que le programme soit relancé.

Exercice 17.5

Écrire un programme qui fait avancer le robot en marche avant à la vitesse 100 pendant 5 s, puis le fait reculer à la vitesse 15 pendant 2 s.

Programmer un robot : les capteurs

D'autres fonctions interrogent les capteurs. Par exemple, l'expression `SEN_OBS_DIG(OBS_CENTER_L)` prend la valeur 1 (`true`) quand le capteur avant gauche détecte un obstacle, et la valeur 0 sinon. De même, les expressions

`SEN_OBS_DIG(OBS_CENTER_R)`, `SEN_OBS_DIG(OBS_SIDE_L)` et `SEN_OBS_DIG(OBS_SIDE_R)` renvoient des valeurs similaires pour les capteurs avant droit, côté gauche et côté droit respectivement.

De même, l'expression `SEN_LINE_DIG(LINE_L)` prend la valeur 0 si le capteur de couleur de sol situé sur la gauche du robot capte une couleur claire et 1 s'il capte une couleur sombre. Le fonctionnement est le même pour le capteur de couleur de sol, situé sur la droite du robot, avec l'expression `SEN_LINE_DIG(LINE_R)`.

À la différence des actionneurs que l'on commande quand on le souhaite, il faut interroger les capteurs de manière régulière, afin d'être prévenu de tous les événements qu'ils détectent. Une méthode pour ce faire est d'organiser le programme sous la forme d'une grande boucle qui, à chaque tour, interroge les capteurs et commande les actionneurs.

Le programme suivant par exemple fait clignoter la diode verte, puis initie un mouvement du robot en marche avant pour un temps infini. Si jamais le robot détecte un obstacle, on allume la diode située à l'avant du robot, on initie un mouvement de rotation de 180 degrés, ce qui interrompt le mouvement rectiligne, puis quand le mouvement de rotation est achevé, on relance le robot en marche avant pour un temps infini. Sinon, on éteint la diode située à l'avant du robot, si jamais elle est allumée.

```
void main() {
  SEN_CONFIG();
  MOT_CONFIG();
  LED_TOP_GREEN_ON_OFF();
  MOT_STR(50,FWD,TIME,0);
  while (1) {
    if (SEN_OBS_DIG(OBS_CENTER_L)){
      LED_FRONT_ON();
      MOT_ROT(25,FWD,CENTER,LEFT,ANGLE,50) ;
      while(!MOT_END){}
      MOT_STR(50,FWD,TIME,0);}
    else{
      LED_FRONT_OFF();}}
```

SAVOIR-FAIRE Écrire un programme pour commander un robot

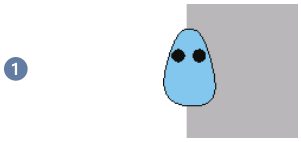
- Identifier les actionneurs et les capteurs à utiliser.
- Écrire les tests sur les valeurs des capteurs et les instructions initiant les actions dans une grande boucle infinie.
- Insérer les temporisations permettant aux actions de s'effectuer complètement si cela est nécessaire.
- Initialiser les capteurs et les actionneurs avant le début de la boucle.

Exercice 17.6 (avec corrigé)

Écrire un programme qui pilote un robot le long d'une ligne sombre dessinée sur un sol clair.

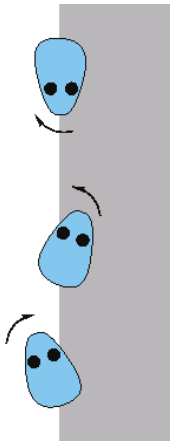
Les actionneurs utilisés sont, bien entendu, les moteurs que l'on commande avec instructions `MOT_STR` *et* `MOT_ROT`*. Les capteurs utilisés sont les capteurs de couleur de sol que l'on interroge avec les instructions* `SEN_LINE_DIG(LINE_L)` *et* `SEN_LINE_DIG(LINE_R)`*.*

On cherche à ce que le robot suive le côté gauche de la ligne, c'est-à-dire que son capteur gauche soit à l'extérieur de la ligne et son capteur droit à l'intérieur. 1



Tant que cela est le cas, le robot avance tout droit. En revanche, si les deux capteurs sont hors de la ligne, cela signifie que le robot est trop à gauche : il doit tourner à droite. Si les deux capteurs sont sur la ligne, le robot est trop à droite : il doit tourner à gauche. Si le capteur gauche est à l'intérieur de la ligne et le capteur droit à l'extérieur, cela signifie que le robot est dans le mauvais sens, il doit faire demi-tour. On utilise ici la même méthode que pour le contrôle de la vitesse du moteur : la méthode de contrôle en boucle fermée.

2



Il est important de placer le robot sur le bord de la ligne au moment où on le lance, sinon il ne fera que tourner sur lui-même.

```
void main () {
  Delay10KTCYx(200);
  SEN_CONFIG();
  MOT_CONFIG();
  while(1) {
    if (SEN_LINE_DIG(LINE_L)==0 && SEN_LINE_DIG(LINE_R)==1) {
      MOT_STR(80,FWD,TIME,0);}
    else {
      if (SEN_LINE_DIG(LINE_L)==0 && SEN_LINE_DIG(LINE_R)==0) {
        MOT_ROT(50,FWD,CENTER,RIGHT,TIME,0);}
      else {
        if (SEN_LINE_DIG(LINE_L)==1 && SEN_LINE_DIG(LINE_R)==1) {
          MOT_ROT(50,FWD,CENTER,LEFT,TIME,0);}
        else {
          MOT_ROT(50,FWD,CENTER,RIGHT,TIME,0);}}}}}
```

Exercice 17.7

Mettre le robot dans une enceinte avec un mur carré de 50 cm de côté, où il n'y a pas d'autres obstacles que les murs. Utiliser le détecteur d'obstacles pour le faire aller vers l'un des murs, puis tourner sans fin dans le sens des aiguilles d'une montre.

Exercice 17.8

Dans un espace sans limite avec deux robots face à face, écrire un programme qui fait danser ensemble les robots en les faisant tourner l'un autour de l'autre.

Exercice 17.9

On imagine un robot aspirateur dans une enceinte avec un mur carré de 50 cm de côté, où il n'y a pas d'autres obstacles que les murs. Écrire un programme pour que le robot passe l'aspirateur sur tout le sol. Essayer différentes méthodes : par des allers-retours, en spirale, etc. Essayer ces programmes.

ALLER PLUS LOIN **Les interruptions**

Organiser un programme en une grande boucle qui teste les capteurs en permanence est possible, mais souvent malcommode. On a donc introduit dans les langages de programmation des outils qui permettent d'exprimer la même chose de manière plus simple. Le programme décrit d'une part ce qu'il faut faire quand tout se passe normalement, par exemple avancer tout droit, et d'autre part des conditions qui définissent des *interruptions*, par exemple le fait qu'un détecteur signale un obstacle, et des instructions à exécuter en cas d'interruption, par exemple faire tourner le robot. Ces différentes instructions sont ensuite traduites automatiquement en un programme qui, de manière répétée, interroge les capteurs et, selon qu'une interruption est déclenchée ou non, exécute une instruction ou une autre.
Cette manière *réactive* de programmer, permet de mieux prendre en compte les aléas de l'environnement : d'une exécution d'un programme à une autre, un robot rencontre rarement deux fois la même situation et il doit s'adapter s'il rencontre une tache d'huile sur le sol, des obstacles nouveaux, etc.

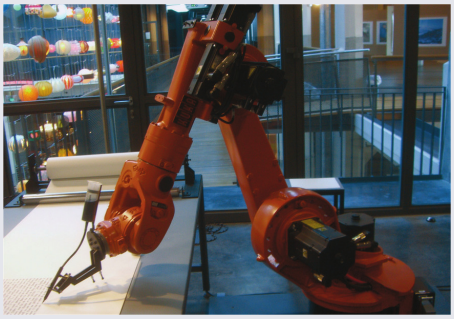
ALLER PLUS LOIN **Les mots « robot » et « robotique »**

Le mot « robot », dérivé d'un mot qui signifie « esclave », a été créé par l'écrivain Tchèque Karel Capek en 1920 et le mot « robotique » par un autre écrivain, Isaac Asimov, en 1942. L'origine littéraire de ces deux mots n'est pas due au hasard. Le fait que les ordinateurs imitent certaines facultés humaines, comme effectuer des multiplications ou jouer aux échec, a suscité le rêve de machines intelligentes. Mais la conception de robots, c'est-à-dire d'ordinateurs mobiles et autonomes, rejoint, de plus, une figure littéraire ancienne, qui du Golem à Pinocchio et du monstre de Frankenstein à WALL-E, pose la question de la frontière entre l'animé et l'inanimé.

ALLER PLUS LOIN **Les robots sont partout**

Dans l'industrie, les robots sont utilisés dans les chaînes de montage de nombreuses usines. En médecine, ils sont utilisés pour effectuer des opérations chirurgicales micro-invasives, pour effectuer des analyses, pour remplacer des membres paralysés, pour assister des personnes dépendantes, etc. Ils sont aussi utilisés dans l'exploration spatiale et sous-marine ou pour intervenir dans les zones inaccessibles, par exemple de centrales nucléaires. Des robots sont aussi utilisés dans des taches plus quotidiennes comme passer l'aspirateur, nettoyer une piscine ou garer une voiture.

Thyroïdectomie assistée par un robot - CHU de Nîmes



Le robot industriel *Robolab* recopiant la Bible

Ai-je bien compris ?

- Quels sont les composants d'un robot ?
- Qu'est-ce que le principe de la boucle fermée ?
- Comment organise-t-on un programme pour que les capteurs soient interrogés périodiquement ?