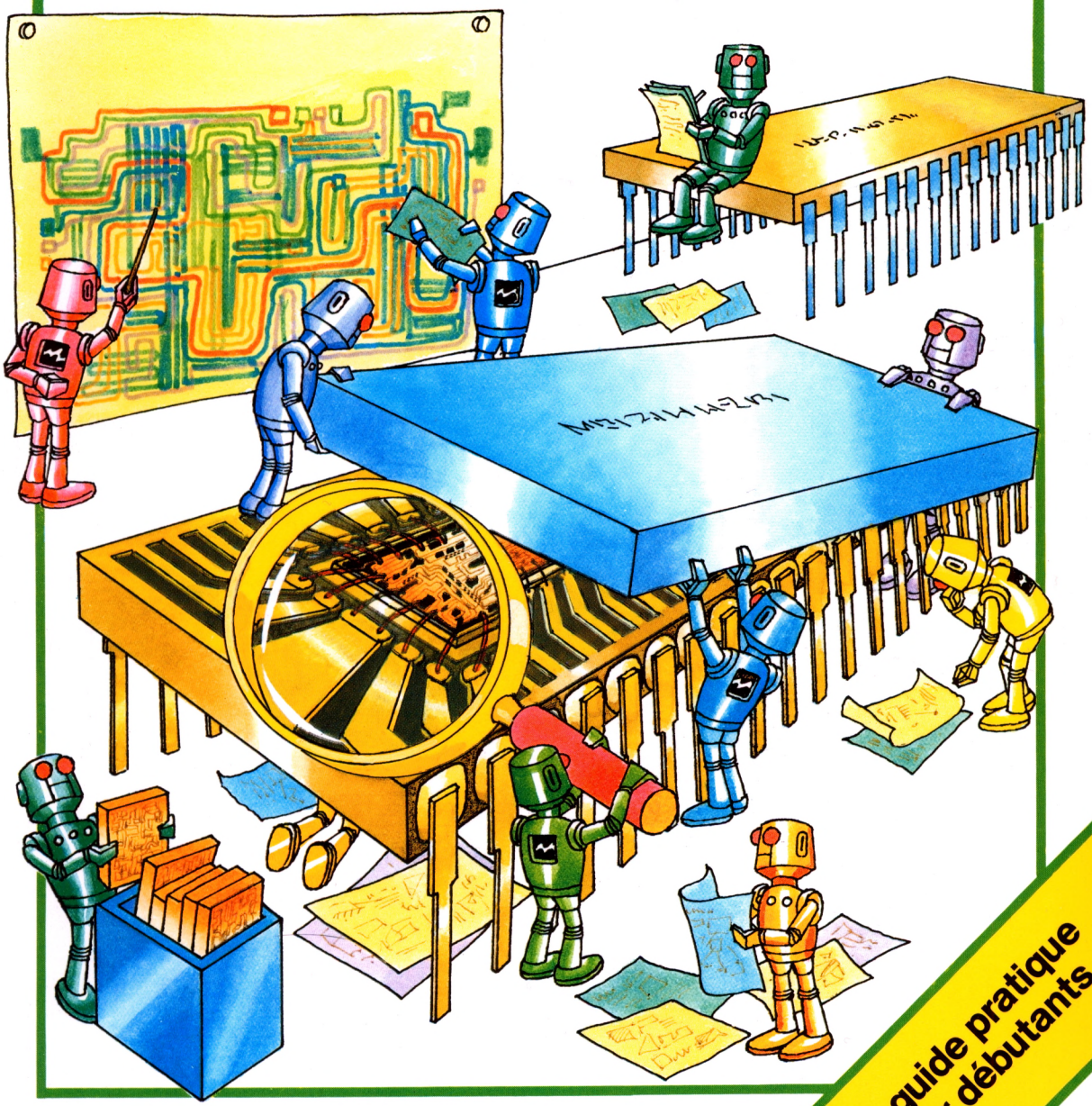
GUIDE DE L'ASSEMBLEUR ET DU MICROPROCESSEUR

POUR Z80 ET 6502



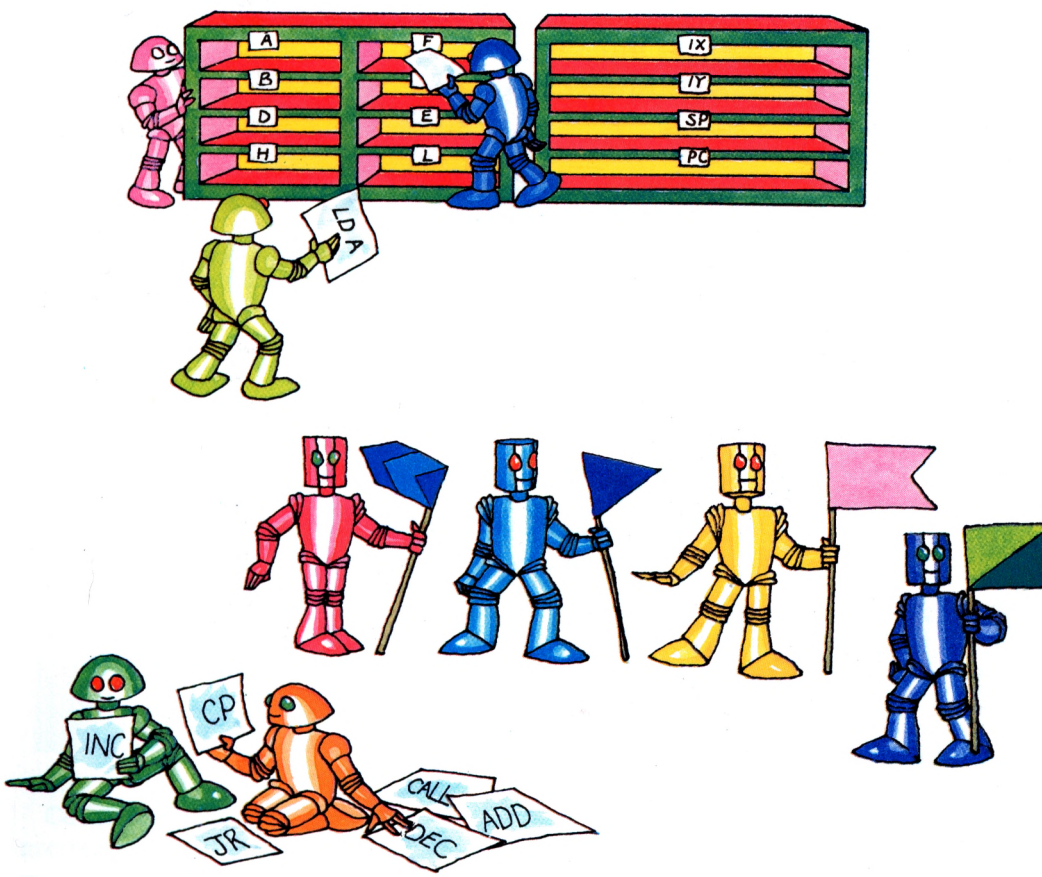
Un guide pratique pour débutants

Hachette Jeunesse

GUIDE DE L'ASSEMBLEUR ET DU MICROPROCESSEUR

PREMIÈRE PARTIE

GUIDE DE L'ASSEMBLEUR



par Lisa Watts et Mike Wharton

Conception graphique de Graham Round et Lynne Norman. Illustrations de Naomi Reed et Graham Round. 6 502 consultants : A.P. Stephenson et Chris Oxlade. Édition française : Patrick Baradeau. Adaptation, assistance technique et conseil : European Media Business, 9, Place des Ternes, 75017 Paris.

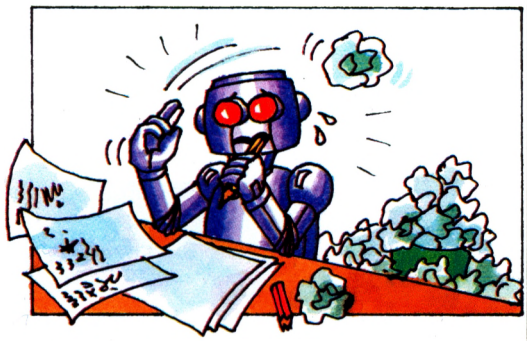
ÉCHOS-ÉLECTRONIQUE

HACHETTE

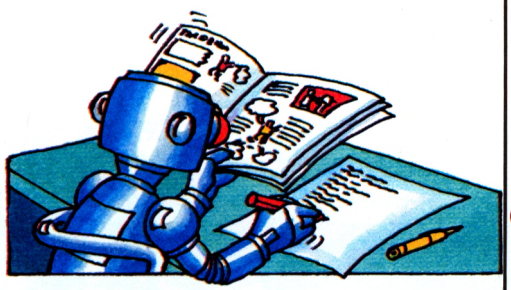
Guide de l'assembleur

Avant-propos

Ce livre est un guide d'initiation simple et progressif au langage machine. Ce dernier permet d'écrire des programmes occupant peu de place en mémoire et exécutés plus vite que ceux écrits en BASIC.

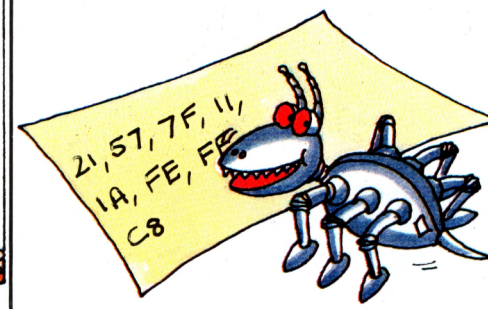


Vous serez peu à peu familiarisé avec les principes de base du langage machine. Vous apprendrez à écrire des programmes simples (additionner deux nombres, afficher un message sur l'écran), à les charger dans votre micro-ordinateur et à les exécuter.

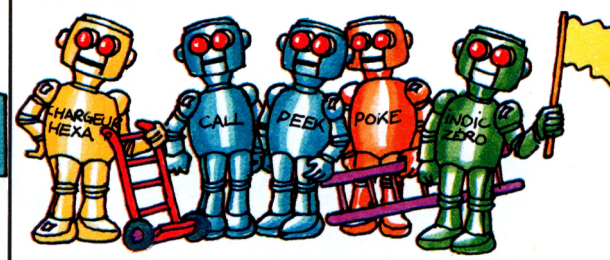


Ce livre est consacré aux microprocesseurs Z80 ou 6502 *. Chaque microprocesseur (« puce » qui renferme l'Unité Centrale de Traitement) a son propre langage. Les micro-ordinateurs qui utilisent le même microprocesseur ont donc le même langage machine.

Le langage machine paraît complexe et difficile car il impose des règles très strictes. Ne vous découragez pas au début si les programmes vous semblent incompréhensibles. On s'habitue très vite à ces suites de lettres et de chiffres où les erreurs sont difficiles à repérer. Quand vous écrivez un progamme en langage machine, soyez méthodique et très attentif, et vérifiez tout deux ou trois fois.



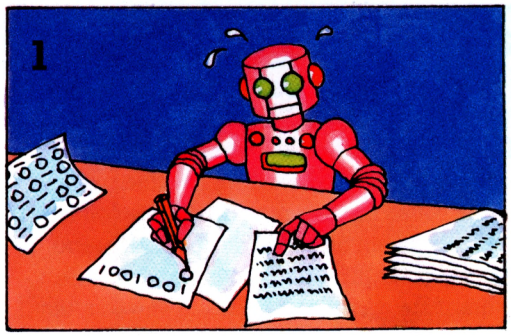
Sauf si vous êtes vraiment « mordu », vous n'avez pas intérêt à écrire de très longs programmes dans ce « langage » : dans bien des cas, le BASIC sera plus facile et aussi efficace. Mais rien ne vaut le langage machine pour faire un jeu d'action rapide ou pour créer de beaux effets graphiques ! Pour rendre vos programmes BASIC encore plus amusants, glissez-y donc des sous-programmes écrits en langage machine.

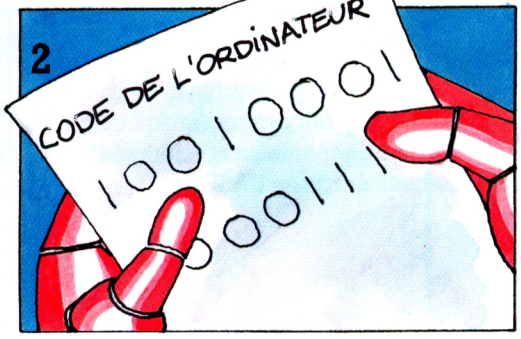


A la fin du livre, des tableaux de conversion vous aident à écrire vos programmes, et un lexique vous en explique tous les termes. Tout au long de ce guide, vous trouverez des tas de devinettes et d'idées de programmes, dont les solutions figurent page 44.

* Voici les principaux micro-ordinateurs familiaux utilisant le Z80 : ZX81, Spectrum, Amstrad, Laser 310, VG 5000 et tous les ordinateurs au standard MSX. Le microprocesseur 6502 équipe : Apple, Atari, Oric, Atmos ; le Commodore 64, qui utilise le 6510, comprend le langage machine du 6502.

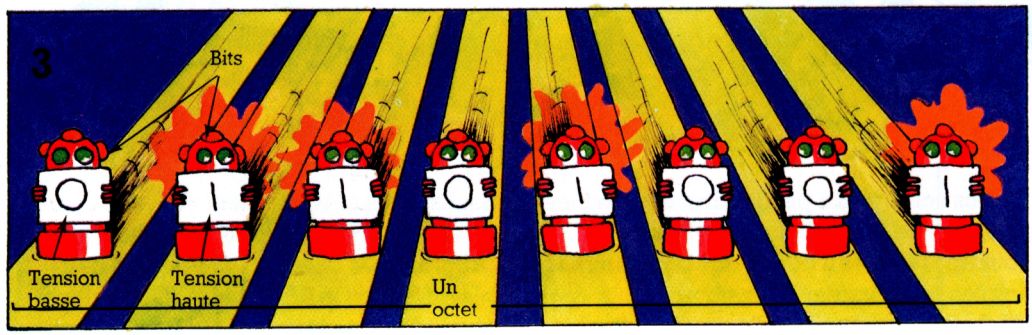
Qu'est-ce que le langage machine ?



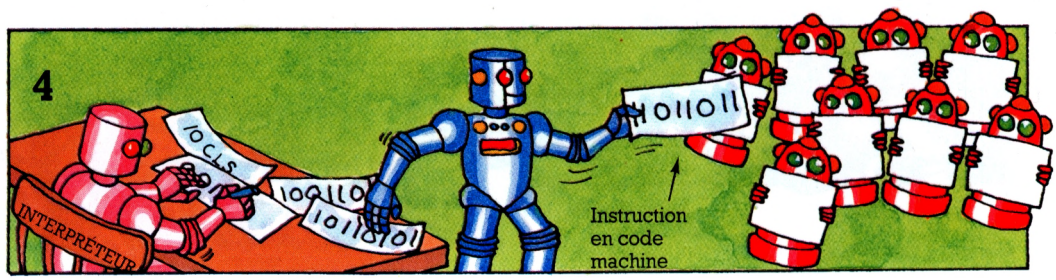


L'ordinateur ne comprend que ce qui est écrit dans son propre langage : le code machine. Quand vous lui donnez un programme en BASIC, il commence par traduire toutes les instructions et les données dans son code machine.

En code machine, chaque instruction ou élément d'information est représenté par un nombre binaire. Bien que le système binaire n'utilise que deux chiffres (1 ou 0), tout nombre peut être traduit en binaire et représenté par une succession de 1 et de 0 *.



Dans l'ordinateur, les nombres binaires se traduisent par des tensions électriques : 0 correspond à une tension basse, 1 à une tension haute. Chaque « 1 » ou « 0 » est appelé « bit », abréviation de l'anglais « binary digit » (chiffre binaire). Les bits circulent dans l'ordinateur par groupes de huit, appelés « octets ». Chaque octet correspond à une instruction ou à un élément d'information traduit en code machine.



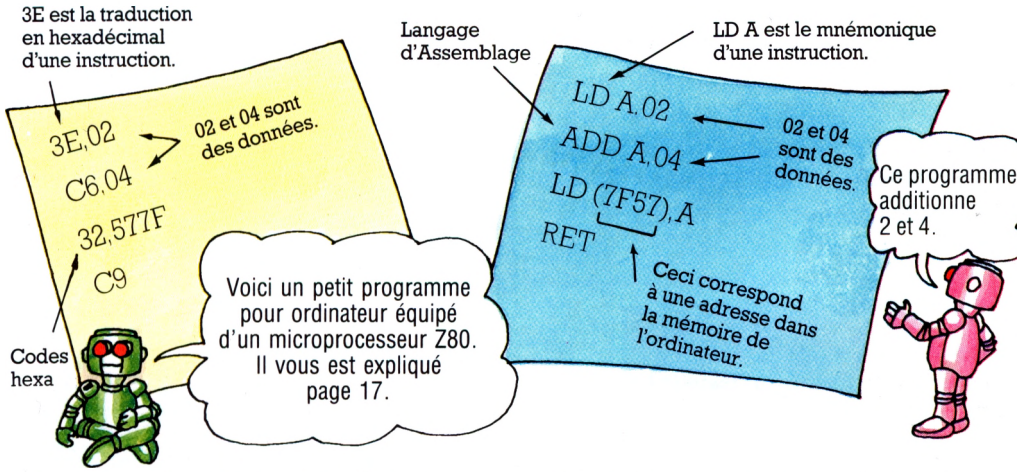
Chaque tâche effectuée par l'ordinateur (additionner deux nombres, effacer l'écran...) nécessite une série d'instructions en code machine. Quand vous tapez une commande BASIC, un programme spécial, l'« interpréteur », traduit votre commande en code machine pour que votre micro-ordinateur comprenne.

Pour désigner des programmes écrits dans un code beaucoup plus proche de celui du microprocesseur que le BASIC, on emploie l'expression « langage machine ». Quand vous l'utilisez, vous devez spécifier les instructions correspondant à chacune des différentes opérations que devra effectuer l'ordinateur pour exécuter une tâche.

* Pour plus amples renseignements concernant le système binaire, voir page 28.

Programmer en langage machine

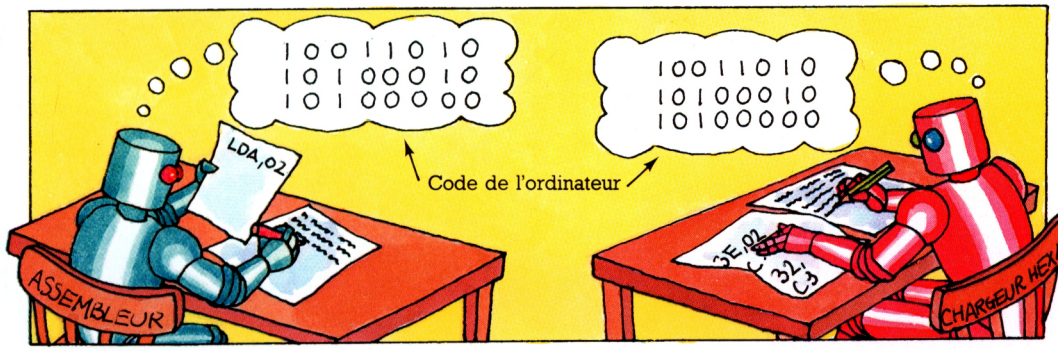
Il y a plusieurs façons d'écrire des programmes en langage machine. Vous pourriez écrire toutes les instructions en nombres binaires, mais ce serait très fastidieux ! C'est pourquoi on a inventé un autre système numérique, l'hexadécimal, beaucoup plus pratique que le binaire. Les programmes peuvent aussi être écrits en « langage d'assemblage ». Dans ce cas, chaque instruction est représentée par un groupe de lettres, appelé « mnémonique », qui évoque le terme anglais correspondant à cette instruction.



Ce programme est écrit en hexadécimal, système qui comporte 16 chiffres et utilise les symboles 0 à 9 et A à F pour représenter les nombres de 0 à 15. (Pour apprendre à compter en hexadécimal, reportez-vous à la page 11.) Le nombre hexadécimal du début de chaque ligne est une instruction (exemple : 3E).

C'est l'équivalent hexadécimal du nombre binaire correspondant à cette instruction.

Voici le même programme en langage d'assemblage. Chaque ligne commence par le mnémonique d'une instruction qui correspond au nombre hexadécimal du programme de gauche. Par exemple, LD A (qui vient de l'anglais « LOAD A ») a la même signification que 3E. Dans ces deux programmes, chaque ligne comporte une instruction qui a son équivalent dans le code de l'ordinateur.



Pour utiliser un langage d'assemblage, il vous faut un programme spécial appelé « assembleur » qui traduit les mnémoniques dans le code binaire de l'ordinateur. Ce programme est intégré sur certains micro-ordinateurs ; mais le plus souvent vous devez acheter la cassette correspondante et charger l'assembleur en mémoire. Vous pouvez aussi écrire votre programme en utilisant les mnémoniques du langage d'assemblage, puis les traduire en hexadécimal. Certains ordinateurs acceptent directement les nombres hexadécimaux ; si ce n'est pas le cas du vôtre, utilisez le petit programme de la page 24, appelé « chargeur hexa », qui convertira les nombres hexadécimaux de vos programmes en codes compréhensibles par votre micro-ordinateur.

Découvrez votre ordinateur

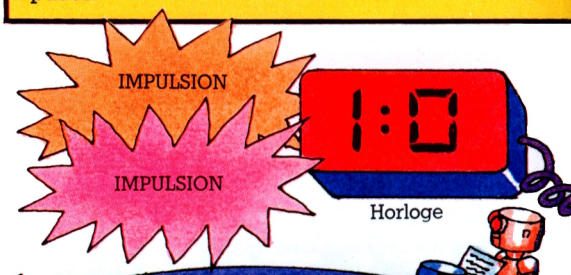
Quand vous faites un programme en langage machine, vous devez tout dire à votre micro-ordinateur : où trouver et stocker les données, comment écrire sur l'écran, etc., alors qu'en BASIC des programmes spéciaux et intégrés le font pour vous. Pour donner les bonnes instructions, vous devez donc bien comprendre ce qui se passe dans votre micro-ordinateur, dont ces deux pages vous présentent les éléments essentiels. Leur fonctionnement sera expliqué dans les pages suivantes.

Dans l'ordinateur

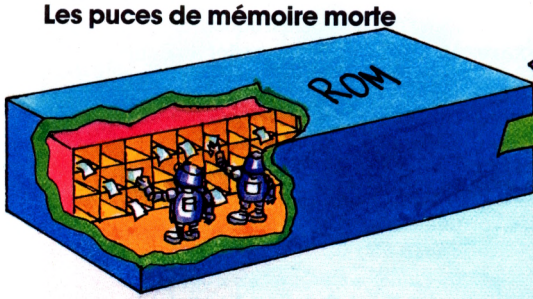
Sous le clavier d'un micro-ordinateur familial, il y a un circuit imprimé : celui-ci est couvert de pistes métalliques qui permettent la circulation de courants électriques. Sur ce circuit imprimé sont fixées de nombreuses puces.

A quoi servent les puces ?

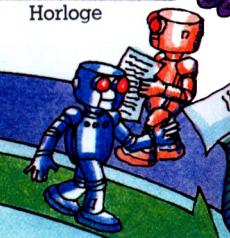
Ce dessin montre le rôle des différentes puces contenues dans un ordinateur. Des messages circulent d'une puce à l'autre sous forme d'octets (groupes de huit bits correspondant à des tensions hautes et basses).



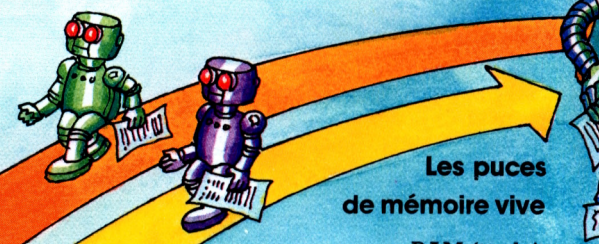
Les puces de mémoire morte



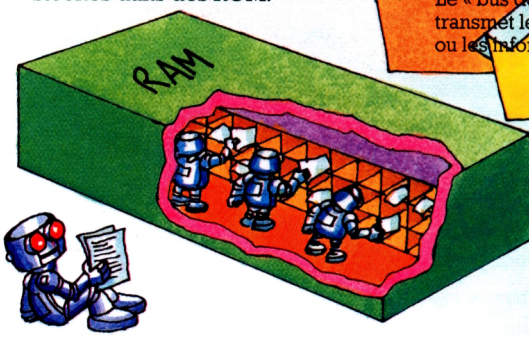
ROM (en français MEM : MEmoire Morte), est l'abréviation de Read Only Memory. Cela signifie que l'ordinateur peut lire les informations contenues dans les ROM, mais qu'il ne peut rien y écrire. Les instructions en code machine qui commandent le fonctionnement de l'ordinateur, l'interpréteur (le programme qui traduit le BASIC en code machine) sont généralement stockés dans des ROM.



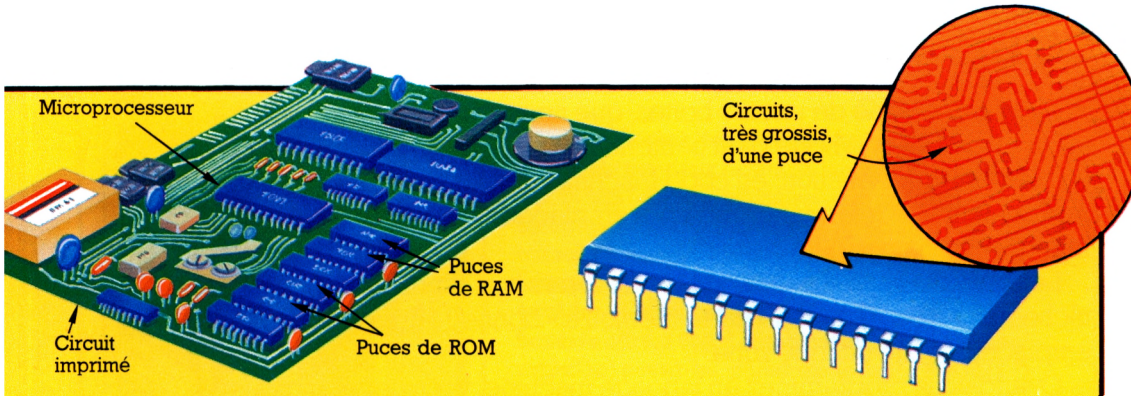
Les octets du code machine circulent entre les puces en suivant les pistes du circuit imprimé. On distingue trois sortes de pistes (que l'on appelle des « bus »), correspondant à des tâches différentes.



Les puces de mémoire vive

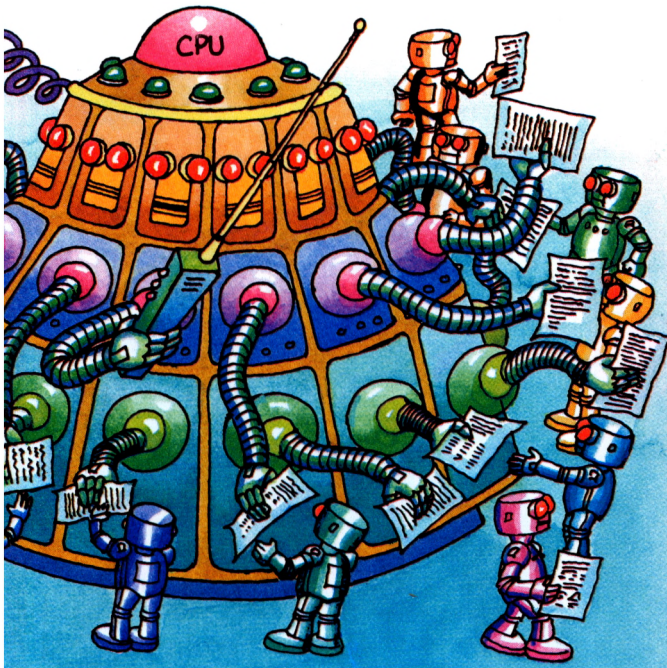


RAM (traduit en français par mémoire vive), est l'abréviation de Random Access Memory. Vous pouvez y accéder librement pour lire ou écrire des informations. C'est là que le micro-ordinateur stocke vos programmes ; mais quand vous l'éteignez, toutes les informations stockées en RAM sont effacées.



Le véritable nom de la puce est « circuit intégré ». A l'intérieur, des impulsions électriques, codées suivant le système binaire, circulent le long de circuits électriques microcospiques : c'est ainsi que s'effectue tout le travail de l'ordinateur. A chaque tâche correspondent des puces particulières, comme le montre le dessin ci-dessous.

L'horloge
C'est un morceau de quartz qui bat plusieurs millions de fois par seconde pour contrôler le rythme des impulsions électriques.



Le microprocesseur
Cette puce renferme l'Unité Centrale de Traitement (en anglais CPU) qui exécute calculs et comparaisons, et coordonne toutes les autres tâches que doit effectuer l'ordinateur. Le programme qui commande l'UCT est stocké en ROM.

Les différents microprocesseurs

Il y a plusieurs modèles de microprocesseurs. Pour les micro-ordinateurs familiaux, les plus utilisés sont le Z80, que l'on trouve sur les ZX81, Spectrum, Amstrad, MSX..., et le 6502 qui équipe les Apple, VIC 20, ORIC... Les M05 et T07-70 utilisent le 6809, ALICE le 6803, etc.

Chaque type de microprocesseur a un langage machine spécifique : tous les ordinateurs dotés du même microprocesseur utilisent donc le même langage machine.

Il existe souvent plusieurs versions du même microprocesseur. Par exemple, le Z80A est plus rapide que le Z80 ; le 6502A et le 6510 sont des versions du 6502 ; mais elles comprennent le langage machine de la première version du microprocesseur. Ce livre est destiné aux ordinateurs équipés d'un Z80 ou d'un 6502.

La mémoire de l'ordinateur

Imaginez une série de petites boîtes, que nous appellerons « emplacements mémoire ». Chaque boîte peut contenir un octet, c'est-à-dire une instruction ou un élément d'information, en code machine. Pour que l'ordinateur retrouve rapidement une information, chaque emplacement mémoire est repéré par un nombre qu'on appelle l'« adresse ».

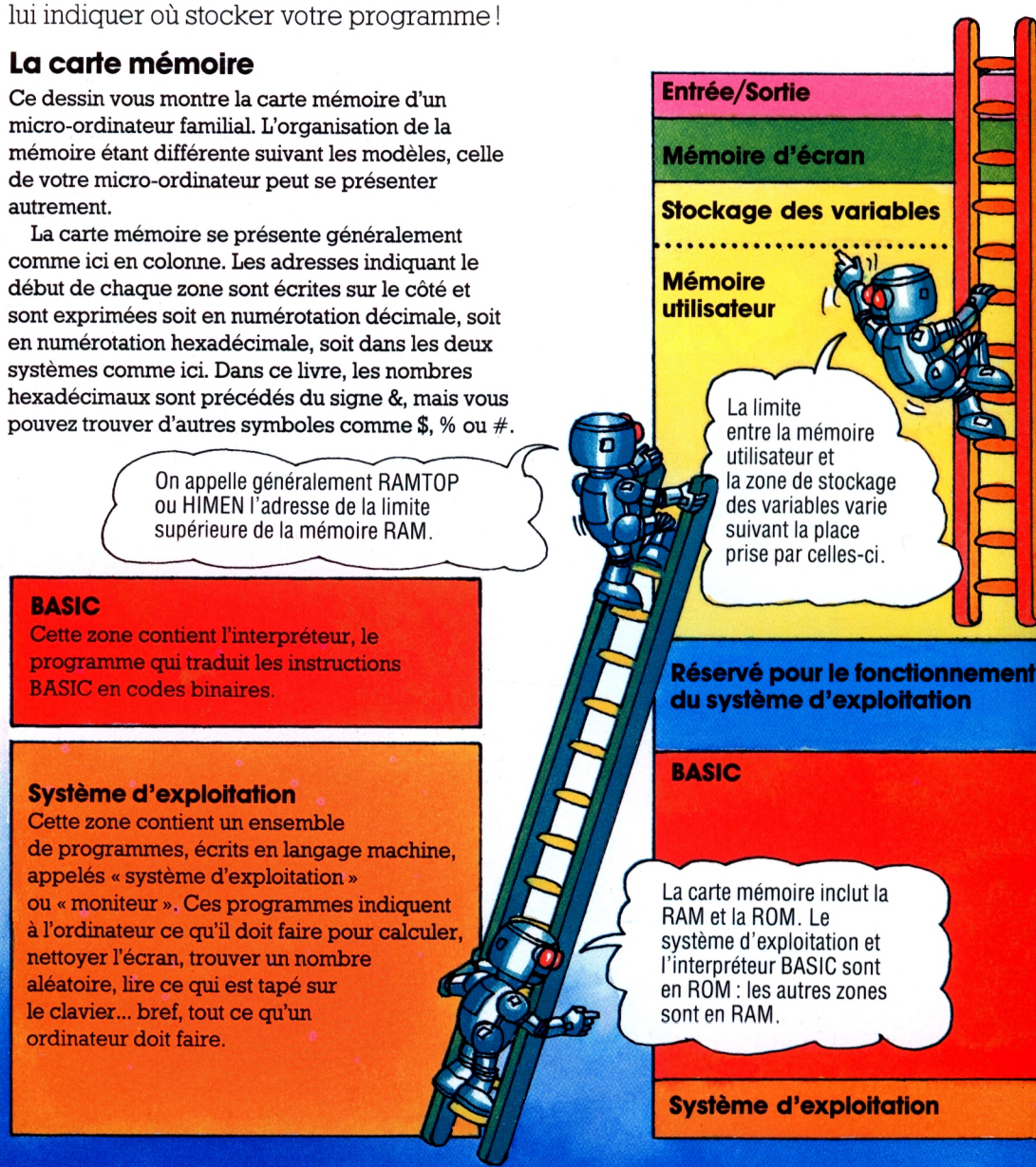
La mémoire est divisée en plusieurs zones correspondant aux différentes tâches de l'ordinateur. La « carte mémoire » donne l'adresse du premier emplacement de chaque zone.

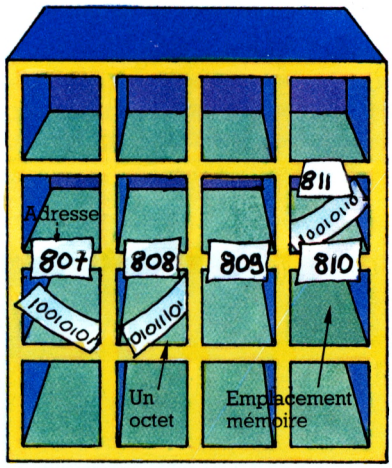
Quand vous programmez en langage machine, vous devez indiquer à l'ordinateur où il doit aller chercher les instructions et les informations, et où il doit les stocker : vous devez donc lui préciser les adresses-mémoire et, même, lui indiquer où stocker votre programme !

La carte mémoire

Ce dessin vous montre la carte mémoire d'un micro-ordinateur familial. L'organisation de la mémoire étant différente suivant les modèles, celle de votre micro-ordinateur peut se présenter autrement.

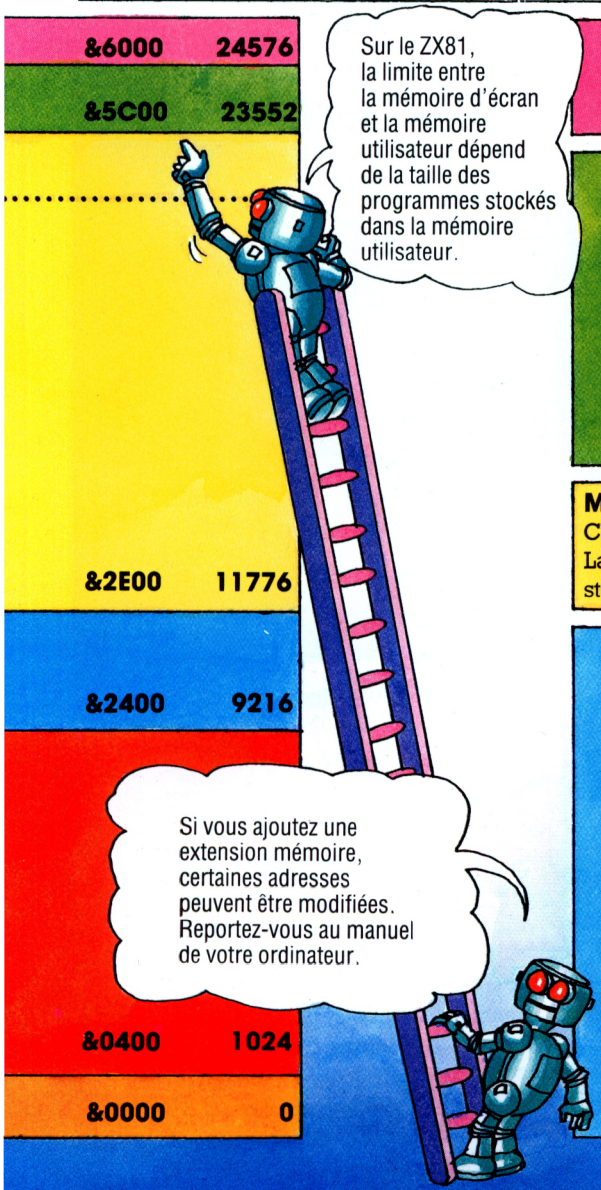
La carte mémoire se présente généralement comme ici en colonne. Les adresses indiquant le début de chaque zone sont écrites sur le côté et sont exprimées soit en numérotation décimale, soit en numérotation hexadécimale, soit dans les deux systèmes comme ici. Dans ce livre, les nombres hexadécimaux sont précédés du signe &, mais vous pouvez trouver d'autres symboles comme $, % ou #.





Les adresses-mémoire

Dans l'ordinateur, les adresses-mémoire se traduisent par deux octets de langage machine, soit 16 bits (ou 16 combinaisons de tensions électriques, hautes ou basses). Avec un microprocesseur Z80 ou 6502, la taille maximale de la mémoire (ROM et RAM ensemble) ne peut excéder 64 kilos-octets. En effet, 16 chiffres binaires offrent 65536 combinaisons (ou adresses-mémoire) possibles, numérotées de 0 à 65535 (2^{16}). Un kilo-octet (1 K) comportant 1024 octets (2^{10}), 65535 ÷ 1024 correspondent bien à 64 K. Chacun de ces emplacements mémoire peut contenir un octet.



Entrée/Sortie

Ces adresses-mémoire sont reliées au port d'entrée/sortie de votre ordinateur.

Mémoire d'écran

C'est là que sont stockées les informations affichées sur l'écran. Toute information placée dans cette zone est automatiquement affichée. Sur la plupart des micro-ordinateurs, on trouve une carte d'affichage qui précise les emplacements correspondant aux différents points de l'écran.

Mémoire-utilisateur

C'est là que sont stockés vos programmes. La valeur des variables et les tableaux sont stockés dans la partie supérieure de cette zone.

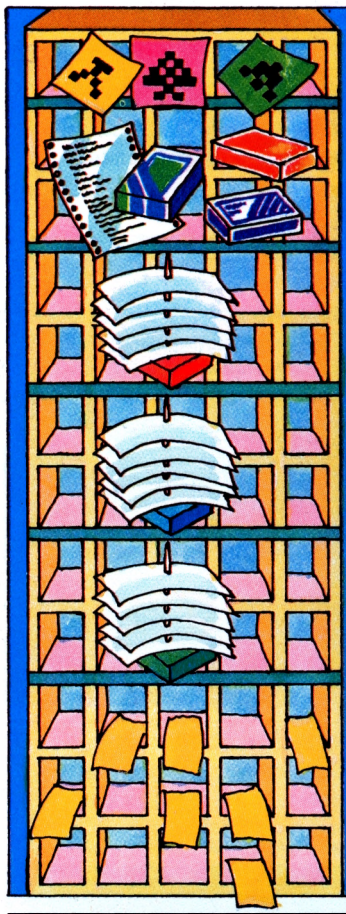
Réservé pour le système d'exploitation

L'ordinateur utilise cette zone pour garder une trace de tout ce qui se passe pendant l'exécution d'un programme. C'est là qu'il stocke, par exemple, la position du curseur, la couleur de l'écran, le numéro de la ligne de programme en cours...

Cette zone est divisée en plusieurs parties correspondant aux différentes tâches : les manuels d'utilisation de certains ordinateurs en donnent une carte détaillée. La page suivante vous explique l'utilisation de cette zone.

Dans la zone de travail de l'ordinateur

Voici une vue plus détaillée de l'espace-mémoire réservé au fonctionnement du système d'exploitation. Dans le manuel de votre ordinateur, vous trouverez pour cette zone, soit une carte spéciale, soit la liste des différentes adresses et leur utilisation. Sur certains ordinateurs (Sinclair par exemple) ces adresses sont dispersées sur tout l'espace-mémoire au lieu d'être regroupées.



Caractères graphiques utilisateur

Si vous créez vos propres caractères graphiques, c'est ici qu'ils sont stockés.

Mémoires-tampons

Ce sont des emplacements destinés à stocker temporairement des données en provenance du clavier ou envoyées vers l'imprimante ou le lecteur de cassettes.

Pile machine

L'UCT utilise ces emplacements pour stocker des adresses pendant qu'elle exécute un programme en langage machine.

Pile BASIC

Appelée aussi pile GOSUB, elle sert à stocker les numéros de lignes associés aux instructions BASIC GOSUB et GOTO.

Pile du calculateur

C'est là que l'UCT place temporairement les nombres pendant les calculs.

Variables système

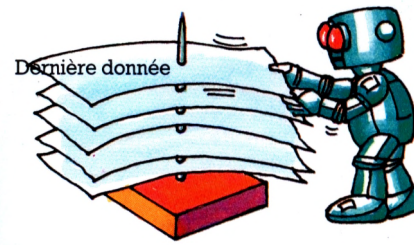
L'UCT y stocke les informations concernant le fonctionnement du micro-ordinateur : quelle est la position du curseur sur l'écran, sur quelle touche appuie l'utilisateur, où sont stockées les variables, etc.

Les pages mémoire

Pour faciliter la tâche de l'ordinateur, la mémoire est divisée en « pages ». Sur un micro-ordinateur, une page contient 256 emplacements mémoire ; un kilo-octet correspond donc à quatre pages ($4 \times 256 = 1024$).

La première page de la mémoire (celle qui correspond aux emplacements 0 à 255) est souvent appelée page 0, et chaque zone de la mémoire commence sur une nouvelle page. Par exemple, sur la carte mémoire de la page 8, la mémoire utilisateur commence à la page 45 (si la première est la page 0).

L'art d'empiler



Pour stocker temporairement des données, l'ordinateur se contente de les empiler : la première donnée entrée est en dessous, la dernière est au-dessus. On appelle cette technique LIFO (*Last In, First Out* : dernier entré, premier sorti).

Les nombres hexadécimaux

Dans les programmes en langage machine, les nombres et les adresses sont écrits en hexadécimal. Voyez ci-dessous comment faire les conversions.

Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Ce tableau donne les chiffres en hexadécimal (0 à 9 et A à F) et leur valeur en décimal. Pour les nombres supérieurs à 15, on emploie deux chiffres (ou plus), comme pour les nombres supérieurs à 9 en décimal. La valeur d'un chiffre hexa dépend de sa position.

Décimal			
1000	100	10	1
1	2	2	6

Hexadécimal		
256	16	1
4	C	A

Regardez un nombre décimal : le chiffre le plus à droite donne le nombre de 1(10^0), le deuxième le nombre de 10(10^1), le troisième le nombre de 100 (10^2), etc.

Dans un nombre hexadécimal, le premier chiffre donne aussi le nombre de 1(16^0), mais le deuxième donne le nombre de 16(16^1), le troisième le nombre de 256 (16^2), etc.

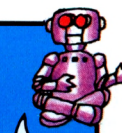
Convertir d'hexa en décimal

Pour convertir un nombre hexadécimal en décimal, traduisez d'abord chaque chiffre en décimal. Multipliez-le ensuite par la valeur que lui confère sa place dans le nombre, et additionnez les réponses.

256	16	1
4	C	A

4	12	10	Valeur décimale
× 256	× 16	× 1	
1024 +	192 +	10 = 1226	

&4CA est égal à 1226



Convertissez &A7 en décimal et 513 en hexadécimal. (Réponses page 44.)

Convertir de décimal en hexa

Pour convertir un nombre comme 1 226 en hexadécimal, divisez-le d'abord par 256 ; divisez ensuite le reste par 16 : vous obtenez ainsi le nombre de 256, de 16 et de 1 qu'il contient. Remplacez alors chaque résultat par le chiffre hexadécimal correspondant*.

1226 ÷ 256 = ... 4 = 4 en hexa
reste 202

202 ÷ 16 = 12 ..... 12 = C en hexa
reste 10 ..... 10 = A en hexa

1226 est égal à 4CA en hexadécimal

Conversion des adresses en hexa

Les deux chiffres de gauche d'une adresse exprimée en hexa (comme 5C64) indiquent la page de l'emplacement mémoire (voir page 10), ceux de droite la position dans la page.

D'hexa en décimal

Adresse &5C64

Page n° : &5C = 92 décimal
Position : &64 = 100 décimal

92 × 256 = 23552 + 100
= 23652

5C64 est égal à 23652.

De décimal en hexa

Adresse en décimal 23652

23652 ÷ 256 = 92 (n° de page)
reste 100 (position dans la page)

92 ÷ 16 = 5 reste 12 = &5C

100 ÷ 16 = 6 reste 4 = &64

23652 est égal à 5C64 en hexa.

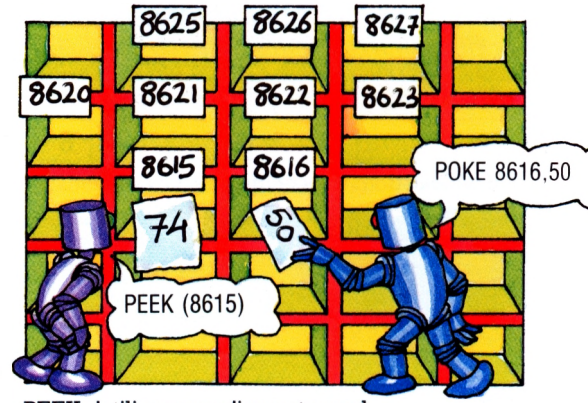
Pour convertir une adresse hexa en décimal, convertissez d'abord chaque paire de chiffres en décimal. Multipliez ensuite le numéro de la page par 256 (il y a 256 emplacements par page) et ajoutez-y le nombre qui donne la position.

Pour convertir une adresse décimale en hexa, divisez le nombre par 256 : vous obtenez le numéro de la page, le reste vous donnant la position sur la page. Vous n'avez plus qu'à convertir ces nombres en hexa comme indiqué ci-dessus.

* Regardez page 41 comment procéder avec une calculatrice.

Jouer avec PEEK et POKE

Deux mots BASIC, PEEK et POKE *, vous permettent de partir à la découverte des emplacements mémoire. Vous pouvez ainsi afficher la valeur des octets stockés en mémoire, et la modifier. PEEK et POKE sont suivis de l'adresse d'un emplacement mémoire (généralement exprimée en décimal). Si vous utilisez le système hexadécimal, n'oubliez pas de faire précéder le nombre du signe &, # ou $ (reportez-vous à votre manuel). Certains micro-ordinateurs n'acceptent que les nombres décimaux.



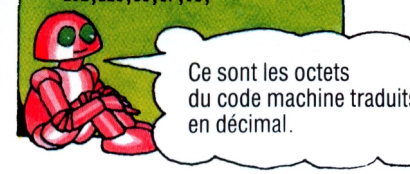
PEEK s'utilise pour n'importe quel emplacement mémoire, mais vous ne pouvez employer POKE que pour la mémoire RAM, puisque vous ne pouvez pas changer les octets stockés en ROM.

Utiliser PEEK

```
PRINT PEEK (12345)
46
PRINT PEEK (720)
240
PRINT PEEK (8643)
0
LET A=PEEK (1024)
PRINT A
176
```

```
10 FOR J=700 TO 725
20 PRINT PEEK (J);",";
30 NEXT J
```

```
RUN
36,27,234,56,21,0,0,
0,0,45,32,67,121,45,
47,89,63,21,0,87,241,
202,225,63,87,16,
```



PEEK (ou l'instruction équivalente) suivi de l'adresse d'un emplacement-mémoire ordonne au micro-ordinateur de regarder ce qui s'y trouve. Pour que le résultat soit affiché, utilisez PRINT PEEK, ou LET et une variable que vous afficherez ensuite. Écrivez un petit programme utilisant une boucle FOR...NEXT pour afficher les octets stockés dans des séries d'emplacements mémoire correspondant à des zones de mémoire différentes.
(Aidez-vous de la carte mémoire de votre ordinateur.)

Utiliser POKE

```
POKE 16763,60
PRINT PEEK (16763)
60
```

Cette ligne dit à l'ordinateur de mettre 60 dans l'emplacement 16763.

Utilisez PRINT PEEK pour voir le résultat.

```
10 FOR A=16763
TO 16768
20 INPUT N
30 POKE A,N
40 NEXT A
```

Cette ligne met le nombre N dans A.

Ces dessins vous montrent comment utiliser POKE. Vous pouvez agir sur tous les emplacements mémoire de la RAM, mais si vous modifiez la valeur des octets de la zone mémoire réservée au système d'exploitation, il risque de se passer des choses bizarres... Pour rétablir la situation, appuyez sur RESET ou éteignez puis rallumez votre micro-ordinateur.

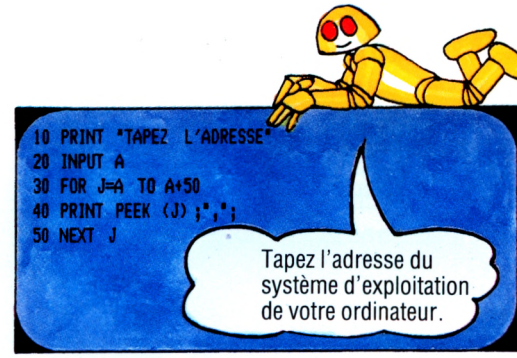
Essayez ce programme en faisant varier les adresses. Donnez toujours des nombres inférieurs à 256, nombre maximal pour un octet.

* Certains ordinateurs utilisent des commandes différentes. Reportez-vous à votre manuel.

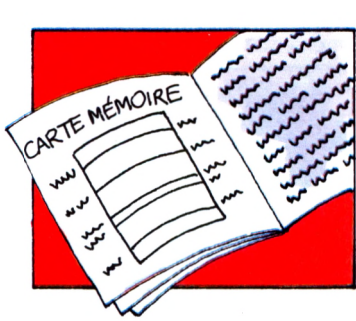
Que signifient ces nombres

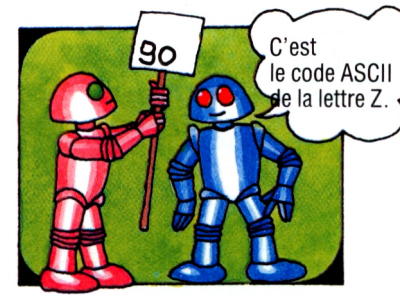
Quand l'ordinateur affiche sur l'écran le contenu d'un emplacement mémoire, c'est toujours un nombre qui peut varier de 0 à 255. En effet, chaque emplacement mémoire peut contenir un octet ; or les huit bits d'un octet permettent $2^8 = 256$ combinaisons. Chacune de ces combinaisons peut avoir plusieurs significations en langage machine.

Par exemple, le nombre binaire 00110000 (48 en décimal) peut être le code d'une instruction, d'une touche du clavier ou de la moitié de l'adresse d'un emplacement mémoire (chaque adresse est composée de deux octets).

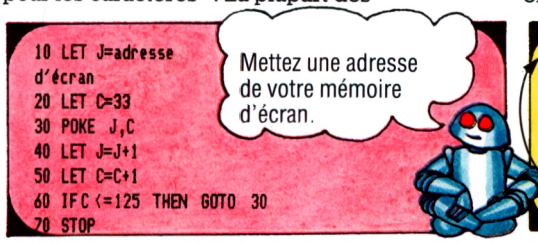


Cherchez dans le manuel de votre ordinateur l'adresse du système d'exploitation et essayez ce programme. Les nombres que vous verrez s'afficher seront la traduction décimale des octets de l'un des programmes en langage machine du système d'exploitation.





Dans la carte mémoire de votre micro-ordinateur, cherchez la mémoire d'écran et faites des POKE dans cette zone. Vous n'avez pas besoin d'utiliser PRINT PEEK, les octets stockés dans la mémoire d'écran étant automatiquement affichés. L'ordinateur interprète les nombres comme des codes pour les caractères *. La plupart des ordinateurs utilisent le code ASCII (prononcer « aski ») qui fixe un nombre pour chaque caractère. Certains, comme le ZX81, ont un code particulier. Le VIC 20 a des codes d'écran spécifiques pour l'affichage des caractères. Vous trouverez dans votre manuel la liste complète des codes de votre ordinateur.



Affichez les caractères de votre ordinateur avec ce programme qui commence au nombre 33 (code ASCII du signe « ! ») et se termine au nombre 125. Les autres nombres compris entre 0 et 255 correspondent à des touches spéciales (barre d'espace, DELETE...), à l'inversion vidéo ou aux caractères graphiques.

Vous pouvez généralement afficher un caractère à n'importe quel endroit de l'écran en précisant l'adresse de l'emplacement mémoire correspondant à ce point.
Ainsi, si votre mémoire d'écran commence en 1024 et si votre ordinateur affiche 32 caractères par ligne, l'adresse de la première position de la ligne sera $1024 + (32 \times 3) = 1120$.
(L'adresse 1024 est comptée 0.)

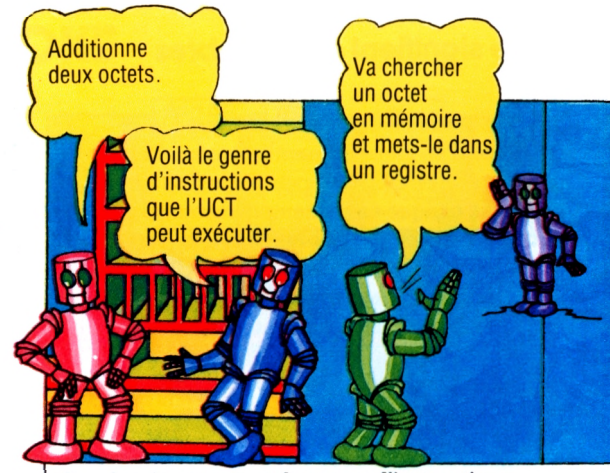
* Sur le Spectrum, les informations concernant l'affichage sont stockées dans des emplacements mémoire différents : vous ne pouvez donc par utiliser POKE de cette façon.

L'Unité Centrale de Traitement

Tout le travail de l'ordinateur consiste à aller chercher les instructions et les données dans la mémoire et à les faire traiter par l'UCT.

L'UCT comporte trois zones : les registres, où sont placés les octets de données pendant leur traitement ; l'UAL (Unité Arithmétique et Logique), où l'on additionne, soustrait ou compare les octets ; et l'unité de contrôle qui organise le tout.

L'agencement des registres du Z80 et du 6502 est différent, comme vous l'expliquent ces deux pages.



Ces dessins montrent le genre d'instructions que peut traiter l'UCT. Elles sont toutes très simples : aller chercher des données en mémoire et les mettre dans des registres, changer des octets de registre, les traiter dans l'UAL et stocker les résultats en mémoire. Il faut une centaine d'étapes comme

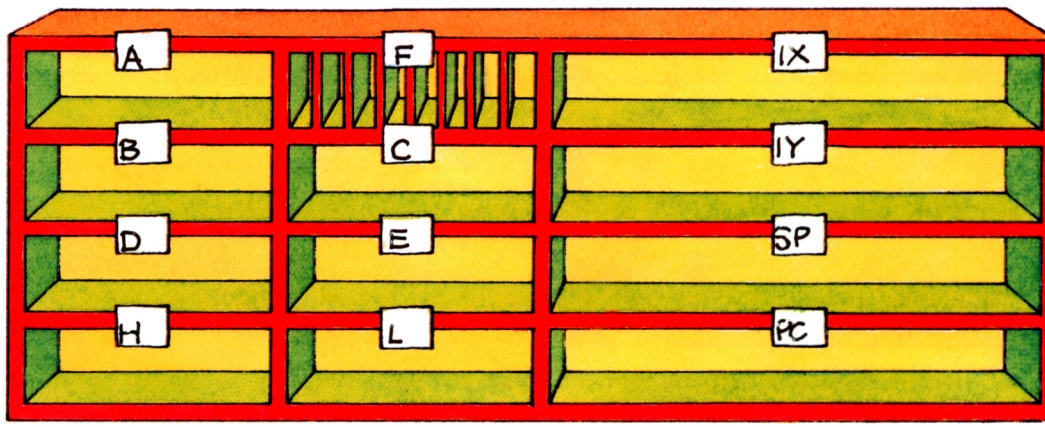
Les registres du Z80

La principale différence entre le Z80 et le 6502 est le nombre de registres : le Z80 en a plus, ce qui lui permet de stocker temporairement des octets dans l'UCT, alors que le 6502 doit les renvoyer en mémoire.

A « l'accumulateur » est le registre le plus important. C'est lui qui stocke les octets qui vont vers l'UAL ou qui en reviennent. Il ne peut contenir qu'un octet.

F le « registre d'état » (en anglais flags register) n'utilise que six de ses huits bits. Chacun est une sorte de drapeau qui indique un fait précis : par exemple, le bit de signe indique si un nombre est positif ou négatif ; le bit de retenue qu'un nombre est trop grand pour être stocké sur un seul octet, etc.

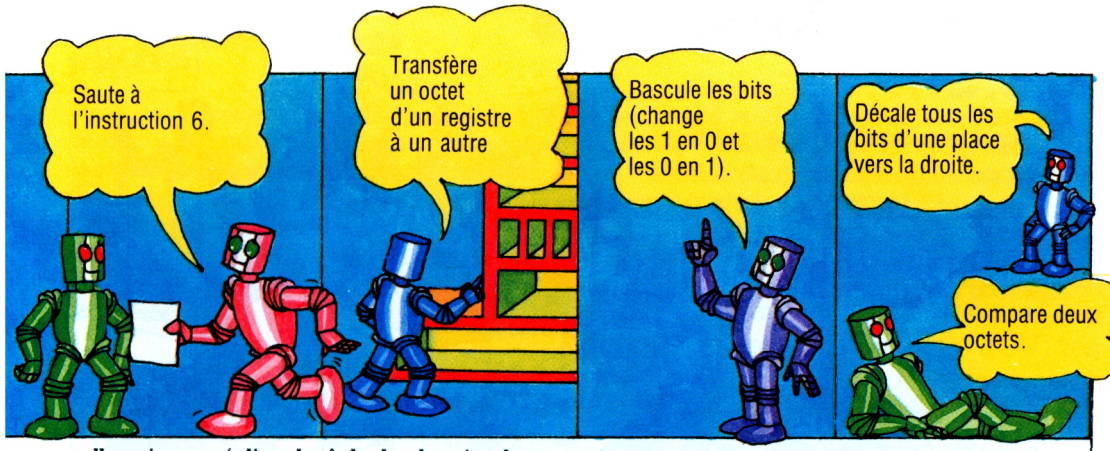
IX et **IY** ces « registres d'index » peuvent contenir seize bits. Ils sont utilisés avec certaines instructions pour calculer l'adresse d'un octet stocké en mémoire.



B, C, D, E, H et **L** les registres à tout faire, où sont stockés les octets en transit entre la mémoire et l'UAL. Chacun ne comporte que huit bits, mais ils peuvent former des paires (BC, DE, HL) pour stocker deux octets.

SP le « pointeur de pile » (stack pointer), qui comporte seize bits, stocke l'adresse du dernier élément mis sur la pile machine (c'est-à-dire l'endroit où l'UCT stocke temporairement les données).

PC le « compteur ordinal » (program counter) est un registre à 16 bits où est placée l'adresse du prochain octet à aller chercher en mémoire. Le nombre dans le compteur ordinal augmente d'une unité à chaque instruction.



celles-ci pour réaliser la tâche la plus simple (additionner deux nombres et afficher le résultat par exemple) ; mais l'UCT peut en effectuer 500 000 par seconde...
A chaque opération, l'unité de contrôle va chercher un octet d'instruction en mémoire ROM ou RAM, charge un octet de donnée dans les registres et effectue l'opération correspondant à l'instruction. Vous pouvez utiliser le langage machine pour dire à l'UCT ce qu'elle doit faire des octets situés dans les registres, mais vous ne pouvez intervenir ni sur l'UAL, ni sur l'unité de contrôle.

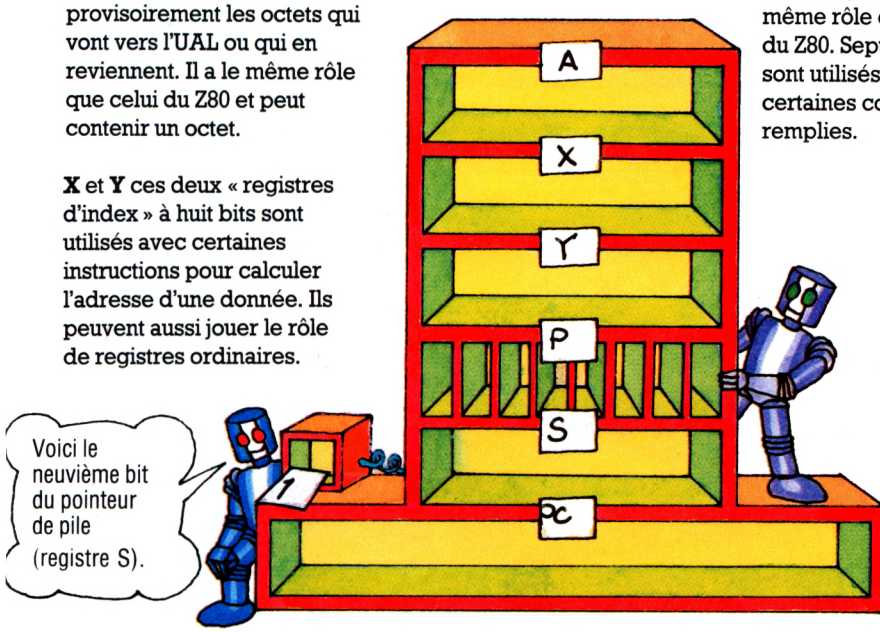
Les registres du 6502

Les principaux registres du 6502 sont les mêmes que ceux du Z80, mais certains portent des noms différents.

A l'« accumulateur » stocke provisoirement les octets qui vont vers l'UAL ou qui en reviennent. Il a le même rôle que celui du Z80 et peut contenir un octet.

X et **Y** ces deux « registres d'index » à huit bits sont utilisés avec certaines instructions pour calculer l'adresse d'une donnée. Ils peuvent aussi jouer le rôle de registres ordinaires.

P le « registre d'état » a le même rôle que le registre F du Z80. Sept de ses huit bits sont utilisés pour indiquer si certaines conditions sont remplies.

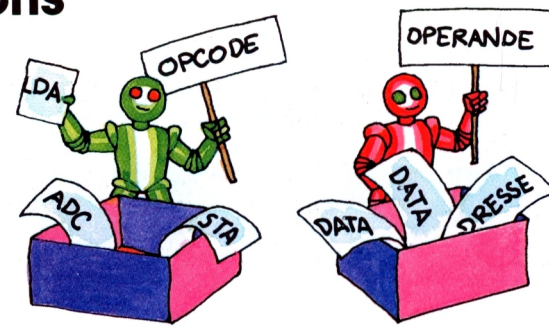


PC le « compteur ordinal » joue exactement le même rôle que celui du Z80.

S le « pointeur de pile » stocke l'adresse du dernier élément de la pile — la zone de la RAM où l'UCT stocke les données. Ce registre ne comportant que huit bits, on lui en ajoute un neuvième, dont la valeur est toujours 1 : sur le 6502, la pile se trouve toujours en page 1 de la mémoire. Le nombre figurant dans le registre indique la position sur la page.

Donner des instructions à l'U.C.T.

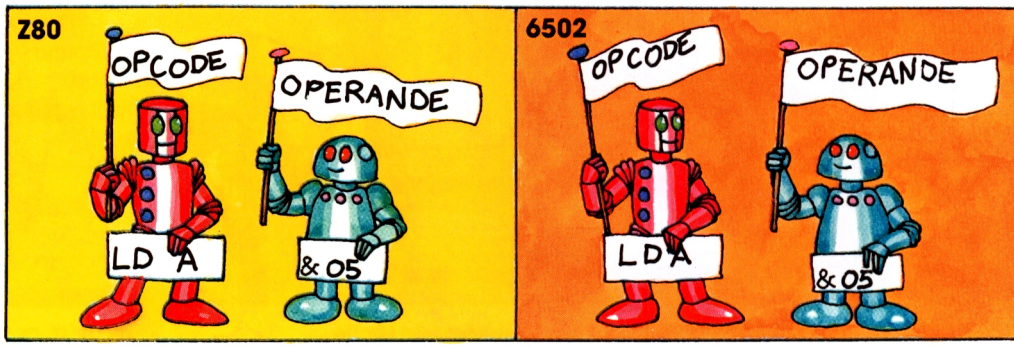
Un programme en langage machine se compose d'une liste d'instructions qui indiquent à l'UCT ce qu'elle doit faire des octets qui se trouvent dans les registres. Vous ne pouvez employer que les instructions comprises par l'UCT : pour les micro-ordinateurs dotés d'un Z80 ou d'un Z80A, il faut avoir recours au jeu d'instructions du Z80 ; pour ceux dotés d'un 6502, d'un 6502A ou d'un 6510, il faut utiliser le jeu d'instructions du 6502. (Vous en trouverez la liste à la fin du livre.)



Généralement, les instructions en langage machine comportent deux parties : le « code opératoire », ou « opcode » est l'instruction qui dit à l'UCT ce qu'elle doit faire ; l'« opérande » lui indique où elle trouvera la donnée à traiter. Chaque code opératoire correspond à une instruction du jeu d'instructions.

On écrit les opératoires en mnémoniques ou en codes hexa. Les mnémoniques sont des abréviations des mots anglais qui désignent l'opération effectuée : ainsi, LD A (ou LDA pour le 6502) signifie « charger (load) un octet dans l'accumulateur ». Les nombres hexadécimaux (à droite sur le dessin) sont la tranposition en hexa des codes binaires des instructions.

Pour utiliser les mnémoniques, beaucoup plus simples à manipuler que les codes hexa, il vous faut un assembleur, c'est-à-dire un programme qui traduit les mnémoniques en code machine *. Sinon, écrivez d'abord votre programme en mnémoniques, puis traduisez-le en codes hexa.



Voici un exemple d'instruction en langage machine, utilisant un mnémonique. Les deux versions, une pour le Z80, l'autre pour le 6502, ordonnent à l'ordinateur de charger le nombre hexadécimal 05 dans l'accumulateur.

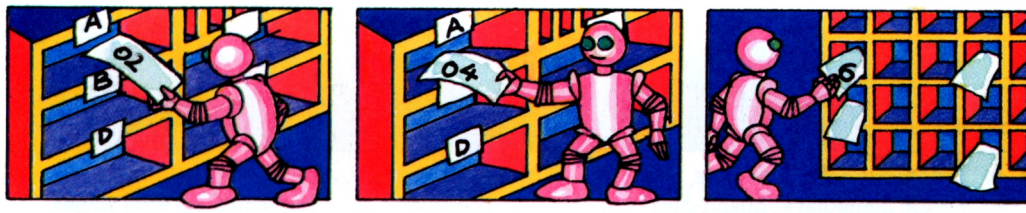
Le signe & indique que l'on est en système hexadécimal (en langage machine, les nombres sont toujours écrits en hexa). Sur le 6502, le signe # précédant un nombre indique qu'il s'agit d'une donnée.

* Pour en savoir plus sur les assembleurs, reportez-vous à la page 40.

Un programme simple

Voici deux programmes d'addition, l'un pour le Z80, l'autre pour le 6502. Tous deux sont écrits en mnémoniques ou, pour être précis, en langage d'assemblage (quand on utilise les codes hexa, on parle de code machine). Dans les pages suivantes, on vous explique comment traduire ces programmes en code machine et comment les essayer sur votre micro-ordinateur.

Les deux programmes suivent les mêmes étapes, bien que les instructions soient différentes *. En outre, sur le 6502, les données à traiter doivent toujours être placées dans l'accumulateur, alors que sur le Z80 elles sont placées dans la paire de registres HL si elles sont trop grandes pour tenir dans l'accumulateur.



Pour additionner deux nombres, il faut charger le premier dans l'accumulateur puis y ajouter le second nombre. Ensuite, on va mettre le résultat en mémoire. Voici les codes opératoires de ces instructions.

Z80 Mnémoniques	Commentaires
LD A,nombre	Charge un nombre dans l'accumulateur A. (LD est l'abréviatioin de l'anglais « load »).
ADD A,nombre	Ajoute le second nombre au contenu de l'accumulateur.
LD (adresse),A	Charge le contenu de l'accumulateur à l'adresse indiquée. Les adresses sont toujours mises entre parenthèses.

Les codes opératoires et les opérandes du Z80 sont séparés par des virgules.

6502 Mnémoniques	Commentaires
LDA nombre	Charge un nombre dans l'accumulateur A. (LD est l'abréviation de l'anglais « load ».)
ADC nombre	Ajoute le second nombre au contenu de l'accumulateur et met à 1 le bit de retenue si le résultat est trop important pour être stocké sur un octet. C est l'abréviation du mot « carry » : retenue (voir page 29).
STA adresse	Va mettre le contenu de l'accumulateur à l'adresse indiquée. ST vient de l'anglais « store » : stocker.

Z80 Programme d'addition

```
LD A,&02
ADD A,&04
LD (&7F57), A
```

Données

Adresse

Ce programme utilise trois opcodes : LD A, ADD A et LD (adresse),A.

6502 Programme d'addition

```
LDA #&02
ADC #&04
STA &7F57
```

Données

Adresse

Le signe # indique que l'opérande est une donnée.

Maintenant, il n'y a plus qu'à remplir les données et les adresses. Dans l'exemple ci-dessus, le programme additionne deux hexa à quatre hexa (soit 2 et 4 en décimal...) et stocke le résultat à l'emplacement mémoire 7F57 (toujours exprimé en hexa).

* A partir de maintenant, vous pouvez lire uniquement ce qui concerne le microprocesseur de votre micro-ordinateur.

Traduire un programme en codes hexa

Pour traduire les mnémoniques en codes hexa, il n'y a qu'une solution : consulter le tableau de correspondance de votre microprocesseur que vous trouverez à la fin du livre (voir page 42 et suivantes). Attention : le code hexa change suivant que l'opérande est une donnée, une adresse ou le nom d'un registre. En voici quelques exemples.

Z80		6502	
Mnémoniques	**Codes hexa**	**Mnémoniques**	**Codes hexa**
LD A,donnée	3E,donnée	LDA donnée	A9 donnée
LD A,(adresse)	3A,adresse	LDA adresse	AD adresse

Quand l'opérande est une donnée, on parle d'« adressage immédiat » ; quand c'est l'adresse où est stockée la donnée, on parle d'« adressage absolu ». La liste des mnémoniques et des codes hexa donnée à la fin du livre couvre toutes les instructions présentées dans ce manuel. (Si vous voulez faire des programmes plus complexes, vous trouverez la liste complète des instructions dans les livres conseillés page 40.)

Z80 Programme d'addition		6502 Programme d'addition	
Mnémoniques	**Codes hexa**	**Mnémoniques**	**Codes hexa**
LD A,donnée	3E,donnée	LDA donnée	A9 donnée
ADD A,donnée	C6,donnée	ADC donnée	69 donnée
LD (adresse),A	32,adresse	STA adresse	8D adresse

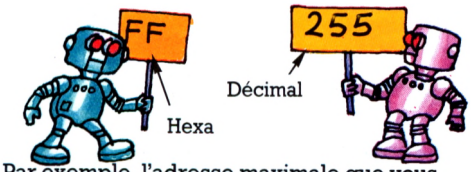
Voici les codes hexa du programme d'addition pour le Z80 et le 6502. On appelle parfois les instructions en mnémoniques « programme source », et celles en codes hexa « programme objet ».

Z80 Programme exact		6502 Programme exact	
Mnémoniques	**Codes hexa**	**Mnémoniques**	**Codes hexa**
LD A,&02	3E,02	LDA #&02	A9 02
ADD A,&04	C6,04	ADC #&04	69 04
LD (&7F57),A	32,577F	STA &7F57	8D 577F

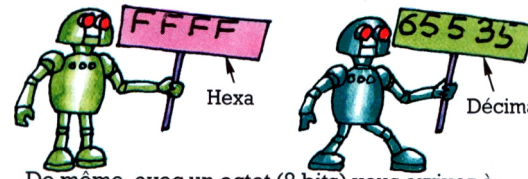
Maintenant, il n'y a plus qu'à mettre les chiffres correspondant aux adresses et aux données. Noubliez pas d'inverser les deux paires de chiffres qui constituent l'adresse (voir les explications de la page suivante).

Pour mieux comprendre les codes hexa

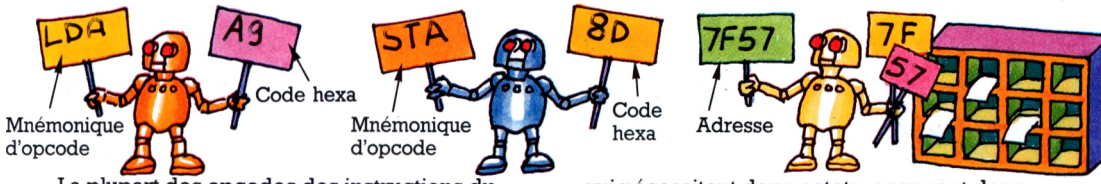
On utilise le système hexadécimal pour écrire les programmes en langage machine, parce qu'il est plus facile de traduire les chiffres binaires de l'ordinateur en hexadécimal qu'en décimal *.



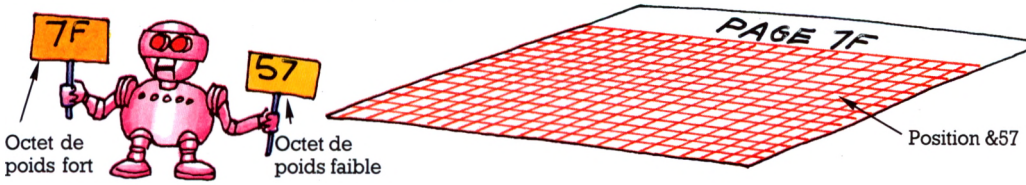
Par exemple, l'adresse maximale que vous pouvez obtenir avec seize chiffres binaires se dit 65535 en décimal, et FFFF en hexadécimal.



De même, avec un octet (8 bits) vous arrivez à 255 en décimal et à FF en hexa.



La plupart des opcodes des instructions du microprocesseur tiennent sur un octet, c'est-à-dire sur deux chiffres hexa. Les adresses, qui nécessitent deux octets, occupent deux paires de chiffres hexa.



La première paire de chiffres hexa, appelée octet de poids fort, donne le numéro de la page où est situé l'emplacement mémoire (voir page 10). La seconde paire, appelée octet de poids faible, indique la position de l'emplacement mémoire dans la page (1 page = 256 emplacements mémoire). En fait, pour des raisons d'ordre technique, il faut donner l'octet de poids faible d'abord, puis l'octet de poids fort.

A quoi ressemblent les programmes en langage machine ?

Les programmes en langage machine publiés dans les revues paraissent très confus. En voici deux exemples. (Ces programmes sont incomplets et ne peuvent tourner ainsi.)

Listage hexa

Chaque paire de chiffres correspond à une instruction, une donnée ou une adresse.

Adresses hexa

```
3A30  00 00 00 00 DB 01 CB 1F
3A38  CB 1F 30 07 3E 10 D3 00
3A40  CF 37 3F 21 2F 39 11 00
3A48  D0 01 C0 03 C5 0E 27 71
3A50  1A FE 58 28 28 CD C5 3A
3A58  CD A0 3A D3 01 4E 0D 79
3A60  FE 00 FA 88 3A F5 CD BD
3A68  08 FE 64 CA 06 3B F1 71
```

Voici ce qu'on appelle un listage hexa. Les quatre premiers chiffres de chaque ligne indiquent une adresse ; les autres paires correspondent à une instruction, à une donnée, ou à une adresse. Le premier code est stocké à l'adresse du début de ligne, les autres aux adresses suivantes.

Listage en langage d'assemblage

Adresses — Codes hexa — Mnémoniques

```
0340  A2 00        LDX #&00
0342  BD 4E 03     LDA &034E,X
0345  9D C0 83     STA &83C0,X
0348  E8           INX
0349  E0 0B        CPX #&0B
034B  D0 F5        BNE &F5
034D  00           BRK
```

Ce listing mêle codes hexa et mnémoniques. Le premier nombre de chaque ligne indique l'adresse où est stocké le premier octet de la ligne, le second donne les codes hexa du programme suivis des mnémoniques.

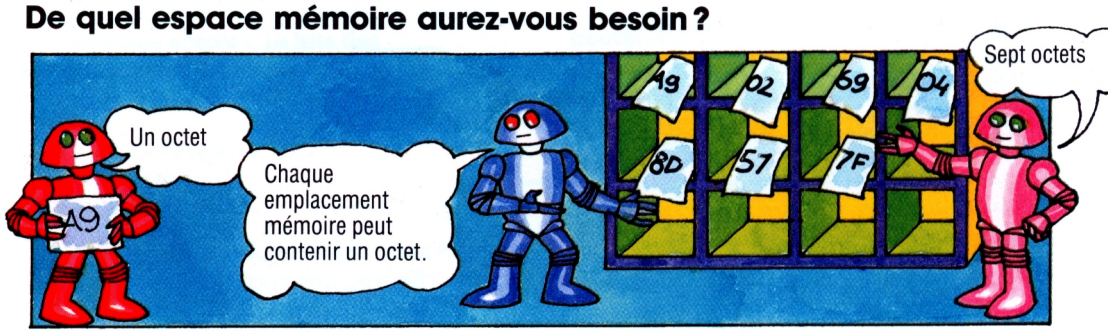
* Pour les conversions binaire/décimal, voir page 28.

Où trouver de la place en mémoire vive (RAM) ?

Avant de charger et de lancer le programme d'addition de la page 18, vous devez préciser dans quelle partie de la mémoire vous voulez qu'il soit stocké. Quand vous écrivez un programme en BASIC, l'interpréteur le stocke automatiquement dans la mémoire RAM. Mais en langage machine, vous court-circuitez l'interpréteur : vous devez donc tout faire...

Vous devez évidemment choisir une zone de la RAM où vous ne courez aucun risque d'interférence avec d'autres informations stockées en mémoire. Évitez donc les zones réservées aux variables système ou aux piles, sinon gare au « plantage » provoqué par la suppression d'informations indispensables... Veillez aussi à bien séparer votre programme en langage machine de tout programme BASIC que vous pourriez lui donner en même temps. En cas de « plantage », une seule solution : éteindre l'ordinateur, puis le rallumer !

De quel espace mémoire aurez-vous besoin ?



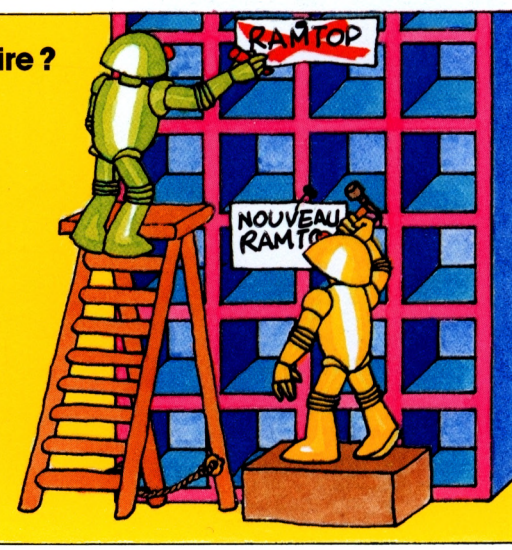
La longueur d'un programme en langage machine se calcule très facilement : il suffit de compter le nombre de paires de chiffres hexa (chaque paire correspond à un octet). Ainsi, le programme d'addition comportant sept octets, occupe sept emplacements mémoire.

La plupart des programmes en langage machine sont très courts et, pour commencer, une centaine d'octets devraient suffire amplement.

Où trouver de la place en mémoire ?

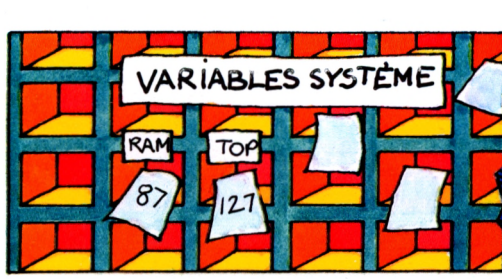
La place normale pour stocker les programmes en langage machine, c'est le haut de la zone utilisateur de la RAM. Mais c'est là aussi que sont placés les programmes BASIC ! Pour éviter tout mélange fâcheux, vous pouvez descendre le haut de la zone utilisateur : vous créerez ainsi une zone neutre que l'ordinateur n'utilisera que lorsque vous lui direz d'aller y mettre votre programme en langage machine.

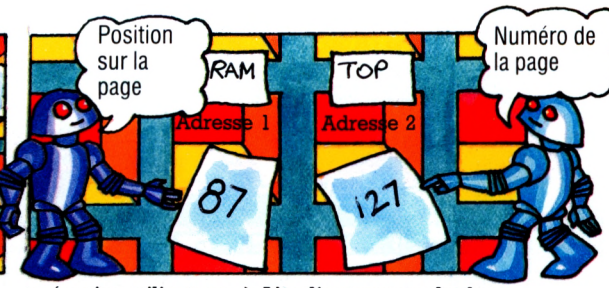
Le haut de la mémoire utilisateur est généralement appelé RAMTOP ou HIMEM. Voyez page suivante comment le modifier.



Comment abaisser la limite de la mémoire utilisateur ?

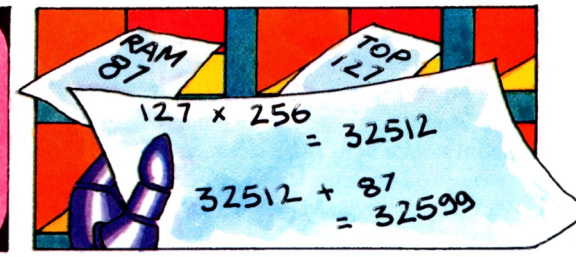
L'ordinateur garde en mémoire l'adresse de la RAMTOP dans les variables système. Il suffit donc de changer cette adresse pour descendre la limite supérieure de la mémoire utilisateur. Quoique le détail des instructions puisse varier d'un ordinateur à l'autre, la plupart suivent le processus décrit ci-dessous. (Vérifiez les instructions dans le manuel d'utilisation de votre ordinateur *.)





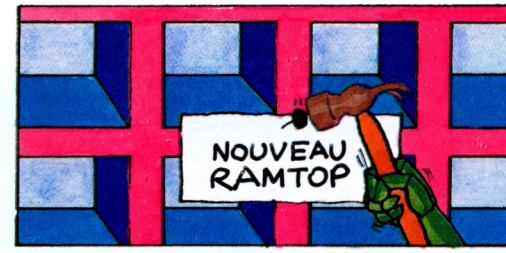
L'adresse de RAMTOP occupe deux emplacements consécutifs dans les variables système, un pour le numéro de la page, l'autre pour la position sur la page. Cherchez cette adresse dans votre manuel (vous la trouverez à RAMTOP ou à HIMEM ou à « sommet de la mémoire utilisateur »). L'ordinateur stocke les deux octets dans l'ordre inverse : le premier emplacement mémoire correspond donc à la position sur la page, le second emplacement au numéro de la page.

```
PRINT PEEK (adresse 1) + PEEK
(adresse 2)*256
```



Vous pouvez utiliser PRINT PEEK (ou la commande correspondante) comme ci-dessus pour afficher l'adresse de RAMTOP. Remplacez les mots entre parenthèses par les nombres qui conviennent. Cette commande convertit automatiquement les deux octets de l'adresse en décimal, en multipliant le numéro de page par 256 et en y ajoutant le nombre correspondant à la position sur la page.

```
CLEAR  adresse de RAMTOP-  100
                              Spectrum
HIMEM  adresse de RAMTOP-  100
                              Oric
```



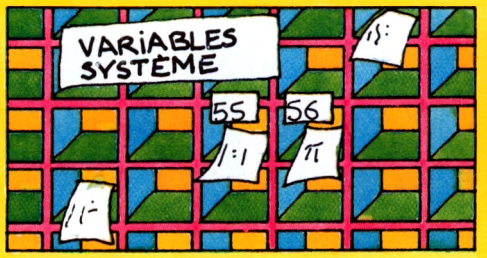
La plupart des micro-ordinateurs ont une commande particulière pour modifier cette adresse. Par exemple, il faut faire CLEAR sur le Spectrum, HIMEM sur l'Oric Atmos... En général, il faut faire suivre cette commande de l'adresse du haut de la mémoire utilisateur et en retrancher le nombre d'octets souhaité. (Consultez votre manuel.)

Dans l'exemple ci-dessus, le haut de la mémoire est abaissé de 100 emplacements, ce qui réserve 99 octets pour le langage machine, à partir de l'adresse qui suit le nouveau RAMTOP. Vous pouvez évidemment modifier ce nombre à votre guise.

* Pour le VIC 20 et le ZX81, voir page suivante.

Un truc pour le VIC 20

Le VIC 20 n'a pas de commande spécifique pour modifier les adresses des variables système. Voici les instructions pour modifier l'adresse du sommet de la mémoire utilisateur.



Cette adresse est stockée dans les variables système, aux emplacements 55 et 56 (ce dernier indique la page).

```
POKE 56,PEEK(56)-1
```

Pour abaisser la mémoire utilisateur de 256 emplacements, c'est-à-dire d'une page, tapez cette commande. L'ordinateur lit le nombre (qui correspond au numéro de la page) stocké dans l'emplacement 56, retranche 1 et remet le résultat dans l'emplacement 56. Autrement dit, cette commande abaisse d'une page le haut de la mémoire. Pour trouver la nouvelle adresse, il suffit de faire : PRINT PEEK(55) + PEEK(56) *256.

Un truc pour le ZX81

Sur le ZX81, la meilleure place pour stocker un programme en langage machine, c'est le début de la mémoire utilisateur. Pour ce faire, tapez un REM avant le programme de conversion hexa de la page 24, et tapez autant de chiffres qu'il y a d'octets dans votre programme en langage machine.

```
5 REM 1234567
```

Sept octets

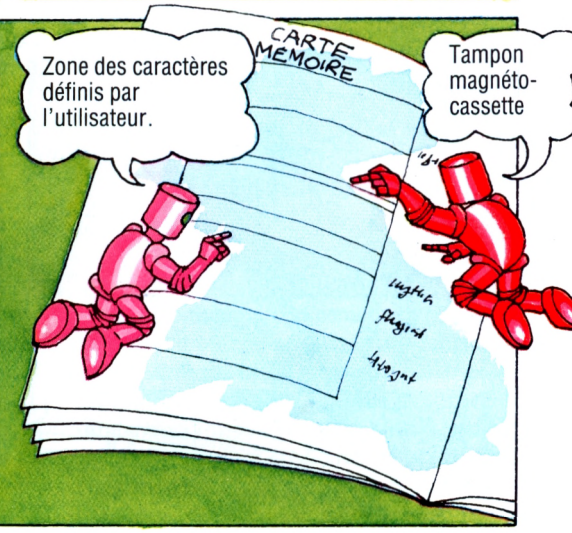
Chacun des chiffres placés après REM occupe un emplacement mémoire. Il n'y a donc plus qu'à utiliser POKE pour aller mettre les octets de votre programme dans les emplacements ainsi réservés.

Mais il vous faut l'adresse du premier chiffre. La mémoire utilisateur commence en 16509. Or la ligne REM occupe cinq emplacements : deux pour le numéro de ligne, un pour le contenu, un pour NEWLINE, un pour mémoriser la longueur de la ligne. Le premier chiffre est donc stocké en 16514.

D'autres emplacements pour le langage machine

Vous pouvez aussi mettre vos programmes en langage machine dans d'autres zones de la mémoire, à condition d'être sûr de ne pas les utiliser : la mémoire tampon pour le magnétocassette, la mémoire réservée aux caractères graphiques définis par l'utilisateur... Cherchez les adresses de ces zones dans votre manuel.

Celui-ci vous conseillera peut-être d'autres emplacements. Vous pourrez aussi trouver des conseils dans les revues et les livres.

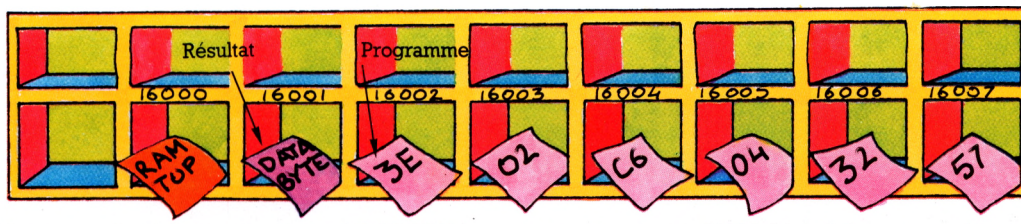


Charger et lancer un programme

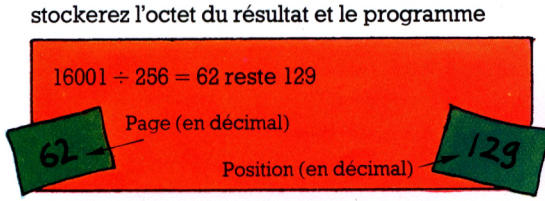
Voyons maintenant comment charger et lancer le programme d'addition de la page 18. Vous devez aller mettre chaque octet de votre programme dans l'un des emplacements mémoire de la zone que vous avez réservée. Si votre ordinateur n'accepte que les nombres décimaux, vous ferez appel au « chargeur hexa », petit programme BASIC de la page 24, qui convertira les codes hexa du langage machine en nombres décimaux qu'il ira mettre dans les emplacements mémoire que vous avez choisis. Mais n'oubliez pas de préciser d'abord l'adresse où sera stocké le résultat et d'ajouter l'instruction de retour que nous vous présentons ci-dessous.

Comment choisir une adresse pour le résultat

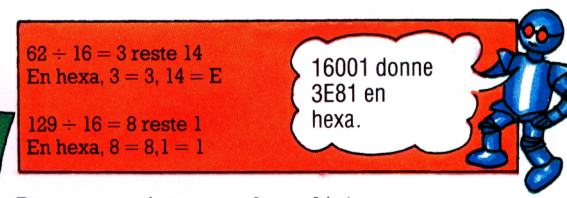
Il faut éviter de mélanger les octets du programme en langage machine avec les octets du résultat. Pour ces derniers, la meilleure place, c'est le début de la zone que vous avez réservée pour le langage machine.



Par exemple, si vous avez ramené RAMTOP à 16000, l'adresse du premier emplacement de la zone réservée sera 16001. C'est là que vous stockerez l'octet du résultat et le programme commencera en 16002. Mais il faut convertir cette adresse en hexa avant de l'intégrer dans le programme.

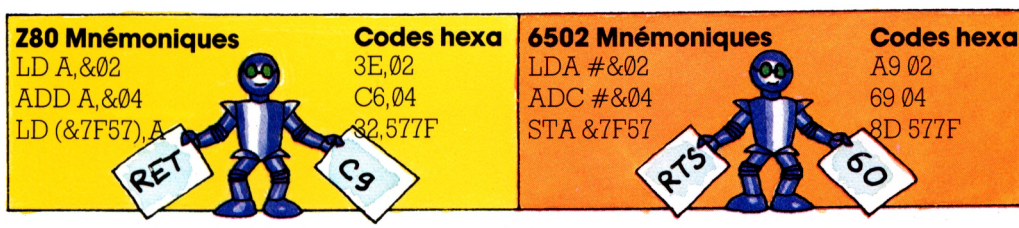


Pour faire la conversion, divisez l'adresse par 256 : le résultat vous donne la page, le reste la position (voir page 11).



Pour convertir ces nombres décimaux en hexa, divisez-les par 16 et convertissez les résultats et les restes en chiffres hexa, comme ci-dessus.

L'instruction de retour



A la fin de tout programme en langage machine, vous devez mettre l'instruction RET (pour le Z80) ou RTS (pour le 6502). Cette instruction dit à l'ordinateur d'arrêter l'exécution du programme en langage machine et de revenir à son point de départ (par exemple un programme en BASIC). Si vous omettez cette instruction, l'ordinateur va essayer de continuer et « se plantera » inévitablement *.

* Pour en savoir plus sur l'instruction de retour, voir page 35.

Le chargeur hexa

Voici le programme qui va vous permettre de charger le langage machine dans la mémoire de votre ordinateur. Complétez la ligne 160 avec les codes hexa de votre programme en langage machine, suivis du mot END pour prévenir l'ordinateur qu'il n'y a plus de données. L'ordinateur lit une paire de chiffres hexa à la ligne 40, les convertit en nombres décimaux aux lignes 70 à 110 et place ceux-ci en mémoire à la ligne 130. (Pour le Spectrum voir en bas de page ; pour le ZX 81 et les modèles Atari, voir page 48.)

```
10 PRINT "ADRESSE  OU SERA  STOCKE
LE LANGAGE  MACHINE  ?"
20 INPUT A
30 LET C=0
40 READ H$
50 IF H$="END"  THEN  GOTO  180
60 IF LEN(H$)<>2  THEN  GOTO  170
70 LET X=(ASC(H$)-48)*16
80 IF ASC(H$)  > 57 THEN LET X=
(ASC(H$)-55)*16
90 LET Y=ASC(RIGHT$(H$,1))
100  IF Y< 58 THEN LET X=X+Y-48
110  IF Y> 57 THEN LET X=X+Y-55
120  IF X< 0 OR X>255  THEN  GOTO  170
130  POKE  A+C,X
140  LET  C=C+1
150  GOTO  40
155  REM  EXEMPLES  DE DONNEES
160  DATA  EF,F6,E2,A9,END
170  PRINT  "DONNEE  INCORRECTE"
180  STOP
```

A est l'adresse du premier emplacement où vous avez choisi de stocker votre programme.

C est un compteur.

Met la première paire de chiffres hexa de la ligne 160 en H$.

Teste H$. Si H$ est le mot END, qui indique qu'il n'y a plus de donnée, va en 180.

Vérifie que H$ contient bien deux chiffres ; sinon va en 170.

Convertit le premier chiffre hexa en nombre décimal et le stocke en X.

Convertit le second chiffre hexa en un nombre décimal appelé Y qui est ajouté à X.

Vérifie que X est bien compris entre 0 et 255 ; sinon va en 170.

Place X dans l'emplacement mémoire choisi (au premier tour, C = 0, donc X est mis en A).

Ajoute 1 à C pour que l'équivalent décimal du code hexa suivant soit mis dans l'emplacement mémoire A+1.

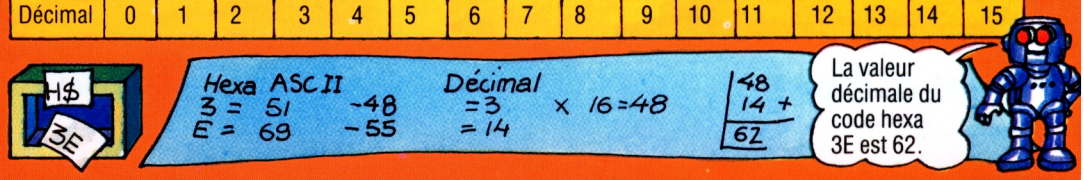
Revient à la ligne 40 pour lire le code hexa suivant.

Mettez ici vos codes hexa, suivis du mot END.

Affiche ce message s'il trouve une donnée erronée aux lignes 60 ou 120, puis s'arrête.

Comment travaille ce programme

Hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII	48	49	50	51	52	53	54	55	56	57	65	66	67	68	69	70
	moins 48										moins 55					
Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



A la ligne 70, l'ordinateur convertit le premier chiffre de H$ en code ASCII, grâce à l'instruction BASIC ASC. Ensuite, pour le transcrire en décimal, il soustrait 48 ou 55, comme le montre le tableau ci-dessus, et le multiplie par 16 parce que le premier chiffre hexa représente le nombre de 16 contenu dans le nombre. Le résultat est placé en X.

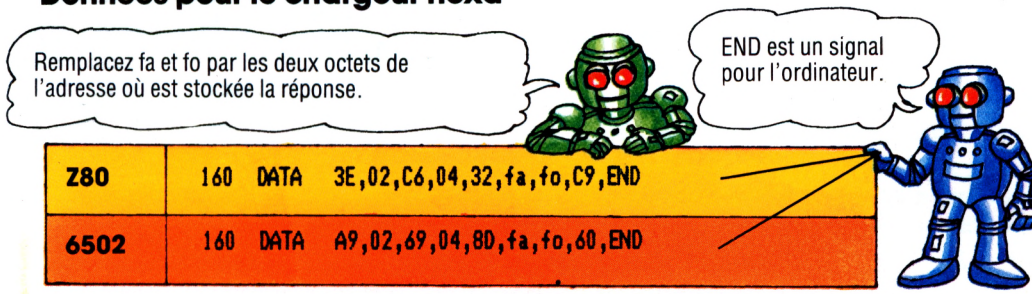
A la ligne 90, le second chiffre hexa est converti en code ASCII. Les lignes 100 et 110 correspondent à sa transcription en décimal ; le résultat est ajouté à X. (Cette fois, on ne multiplie pas le résultat par 16 : ce chiffre hexa indique le nombre de 1 contenu dans le code hexa.) La valeur stockée en X est l'équivalent décimal de la paire de chiffres hexa.

* Pour le Spectrum, remplacez ASC par CODE et mettez chaque paire de codes hexa entre guillemets. Modifiez ainsi la ligne 90 : LET Y=CODE(H$(2 TO)).

Utilisation du chargeur hexa

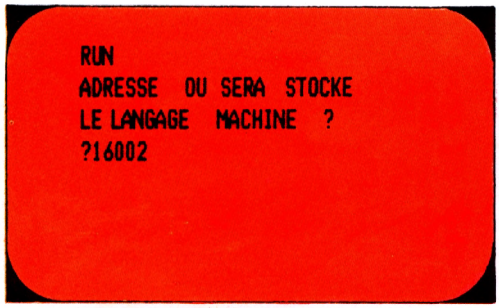
Maintenant, vous allez enfin pouvoir essayer le programme d'addition en langage machine. Tapez le programme de la page 24 en remplaçant les valeurs données en exemple à la ligne 160 par les vrais codes hexa du programme d'addition donnés ci-dessous.

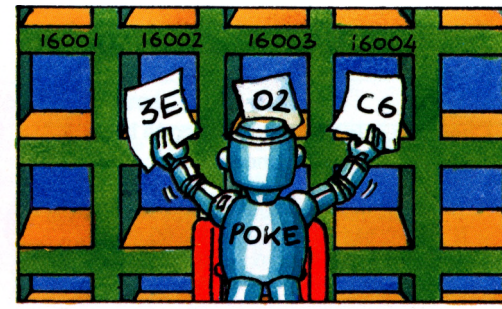
Données pour le chargeur hexa



Voici les codes hexa du programme d'addition. Remplacez les lettres fa (octet de poids faible) et fo (octet de poids fort) par les deux octets de l'adresse où doit être stockée la réponse. N'oubliez pas que l'octet de poids faible (position) doit être placé avant l'octet de poids fort (page).

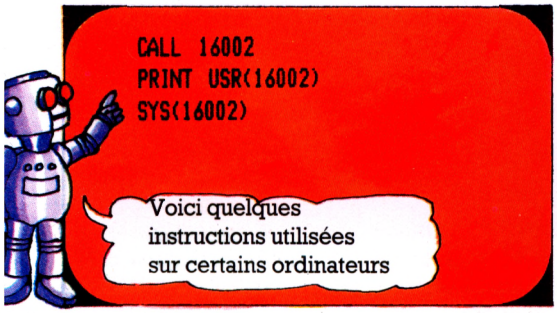
Lançons le programme

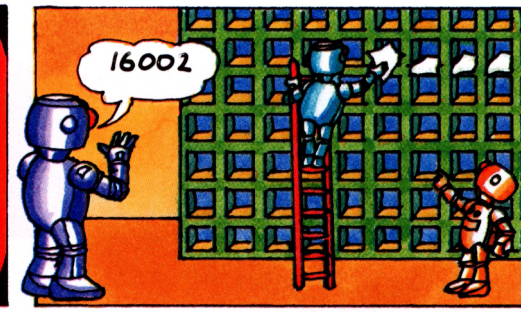




Lancez le programme. Quand il vous demande l'adresse, tapez celle du premier emplacement placé après celui où vous voulez stocker le résultat. Tapez cette adresse en décimal puisque le programme utilise l'instruction POKE.

Lançons le programme en langage machine





L'instruction qui commande l'exécution d'un programme en langage machine varie suivant les ordinateurs. Certains utilisent CALL, d'autres PRINT USR ou SYS ; ces instructions sont toujours suivies de l'adresse — en décimal — de l'emplacement du premier octet du programme. Dès qu'il trouve cette instruction, l'ordinateur va à l'adresse indiquée et exécute le programme. Cherchez dans votre manuel l'instruction reconnue par votre ordinateur.

Pour voir le résultat

```
PRINT PEEK(16001)
```

```
PRINT PEEK(16001)
6
```

L'ordinateur exécute les instructions du langage machine et stocke le résultat dans l'emplacement que vous lui avez assigné. Si vous voulez que le résultat soit affiché, utilisez PRINT PEEK suivi de l'adresse du résultat. Celui-ci sera affiché en décimal.

Des idées de programmes

Vous en savez maintenant assez sur le langage machine pour écrire des programmes simples. Aidez-vous du pense-bête ci-dessous pour vous rappeler les étapes à suivre. Vous trouverez les réponses page 44.

1. Écrivez un programme qui additionne 25 et 73 (en décimal) et stocke le résultat en mémoire.
2. Essayez maintenant d'écrire un programme qui additionne 64, 12 et 14 (en décimal) et stocke le résultat en mémoire.

Le programme d'addition ne marchera que si le résultat est inférieur à 255. Pour les nombres plus importants, voyez page 28.

Pense-bête pour le langage machine

1. Écrivez le programme en langage d'assemblage et convertissez les données en hexa.

2. Cherchez, à la fin du livre, le code hexa correspondant aux mnémoniques.

N'oubliez pas de taper END à la fin de votre liste de codes hexa, dans le chargeur hexa.

3. Ajoutez l'instruction de retour à la fin du programme (voir page 23).

4. Comptez le nombre d'octets et réservez la zone correspondante en mémoire RAM (voir pages 20-22).

Relevez l'adresse des emplacements réservés pour le résultat et le programme.

5. Cherchez quels emplacements mémoire vous utiliserez pour les résultats et convertissez les adresses en hexa (voir page 23).

6. Complétez le programme avec les adresses. N'oubliez pas d'inverser l'ordre des deux octets (voir pages 18-19).

Avant de lancer le chargeur hexa, vérifiez très soigneusement les codes hexa de la ligne de DATA.

7. Tapez le chargeur hexa (ou chargez-le si vous l'avez sauvegardé sur cassette) et complétez la ligne 160 avec les codes hexa suivis du mot END (voir page 24).

Si votre programme se « plante », vérifiez que vous avez utilisé les bons codes hexa.

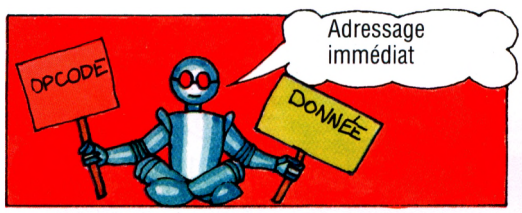
8. Lancez le chargeur hexa et rentrez l'adresse décimale du premier emplacement où doit être stocké votre programme (voir page 25).

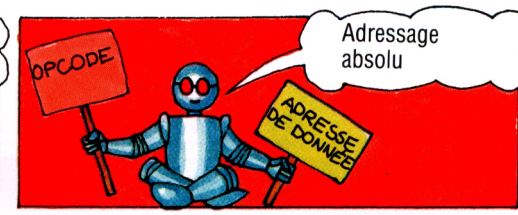
9. Lancez le programme en langage machine. Utilisez l'instruction reconnue par votre ordinateur, suivie de l'adresse en décimal du premier emplacement de mémoire (voir page 25).

Si vous changez les codes hexa du chargeur hexa, il faut relancer le programme pour stocker les nouveaux octets en mémoire.

Additionner des octets stockés en mémoire

Dans le programme précédent, les données étaient intégrées au programme (l'adressage immédiat). Mais parfois on souhaite que l'ordinateur utilise une donnée stockée en mémoire. L'opérande sera alors constituée de l'adresse où l'ordinateur trouvera la donnée (adressage absolu, ou direct, ou étendu).





Il y a encore bien d'autres façons de dire à l'ordinateur où il trouvera les données à utiliser. Toutes ces possibilités s'appellent « modes d'adressage ». Le code hexa de chaque instruction dépend du mode d'adressage que vous utilisez.

Programme pour additionner des nombres stockés en mémoire

Comparez les codes hexa des instructions de ce programme, qui utilise l'adressage absolu, avec ceux du programme précédent, qui utilisait l'adressage immédiat.

Z80 programme		
Mnémoniques	**Codes hexa**	**Commentaires**
LD A,(adresse 1)	3A,adresse 1	Met le nombre de l'adresse 1 dans l'accumulateur.
LD B,A	47	Met le nombre de l'accumulateur dans le registre B.
LD A,(adresse 2)	3A,adresse 2	Met le nombre de l'adresse 2 dans l'accumulateur.
ADD A,B	80	Ajoute le nombre du registre B à l'accumulateur.
LD (adresse 3),A	32,adresse 3	Stocke le contenu de l'accumulateur à l'adresse 3.
RET	C9	Retour.

Pour additionner 2 nombres placés en mémoire, il faut d'abord les charger dans des registres, par exemple l'accumulateur (A) et le registre B. Mais vous ne pouvez pas faire passer directement un nombre de la mémoire au registre B : vous devez donc placer le premier nombre en A et le transférer ensuite en B.

6502 programme		
Mnémoniques	**Codes hexa**	**Commentaires**
LDA adresse 1	AD adresse 1	Place le nombre de l'adresse 1 dans l'accumulateur.
ADC adresse 2	6D adresse 2	Ajoute le nombre de l'adresse 2 à l'accumulateur.
STA adresse 3	8D adresse 3	Stocke le contenu de l'accumulateur à l'adresse 3.
RTS	60	Retour.

Lançons le programme

Suivez soigneusement les étapes de la page 26. Vous devez d'abord aller mettre en mémoire les deux nombres à additionner. Placez-les par exemple dans les emplacements situés au début de la zone que vous avez réservée pour le langage machine ; ainsi ces données seront bien séparées des instructions. Convertissez ensuite les adresses en hexa et intégrez-les au programme. N'oubliez pas qu'il vous faut une troisième adresse pour le résultat et que vous devrez faire PRINT PEEK (adresse 3) pour voir le résultat affiché sur l'écran.

Manipuler les grands nombres

Les programmes des pages précédentes ne permettaient pas d'obtenir des résultats supérieurs à 255, nombre maximal que peuvent représenter les 8 bits d'un emplacement mémoire ou d'un registre. Pour manipuler des nombres plus grands, il faut en connaître un peu plus sur le système binaire et savoir utiliser l'indicateur de retenue. C'est ce que nous vous proposons maintenant.

Les nombres binaires

Le système binaire fonctionne exactement comme les systèmes décimal et hexadécimal. Mais comme il n'y a que deux chiffres, 0 et 1, on est amené à utiliser un grand nombre de chiffres dont la valeur dépend de leur place dans le nombre binaire.

1	1	1	1	1	1	1	1
×128	×64	×32	×16	×8	×4	×2	×1
128	+ 64	+ 32	+ 16	+ 8	+ 4	+ 2	+ 1 = **255**

Chacun des chiffres d'un nombre binaire a une valeur double de celui placé à sa droite. Le chiffre de droite indique si la valeur 1 est contenue ou non dans le nombre, le deuxième chiffre joue le même rôle pour la valeur 2, le troisième pour la valeur 4, et ainsi de suite. Pour convertir un nombre binaire en décimal, il faut multiplier chaque 1 par la valeur que lui confère sa place et additionner les résultats.

0	0	1	0	1	1	1	0
× 128	× 64	× 32	× 16	× 8	× 4	× 2	× 1
0 +	0 +	32 +	0 +	8 +	4 +	2 +	0 = **46**

1	0	0	0	0	1	1	1
× 128	× 64	× 32	× 16	× 8	× 4	× 2	× 1
128 +	0 +	0 +	0 +	0 +	4 +	2 +	1 = **135**

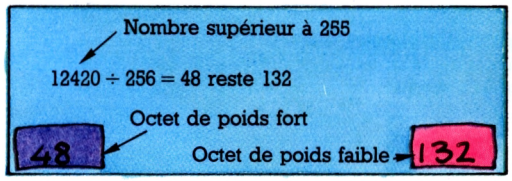
Quelle est la valeur décimale de ces nombres ? (Réponse page 44.)

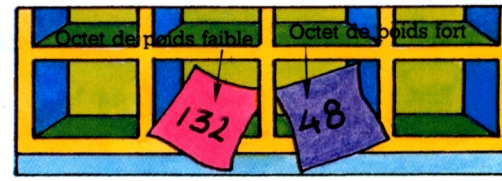
0	0	0	1	1	0	1	0
1	1	1	1	1	0	1	1
1	0	1	0	1	0	1	0

Voici quelques exemples pour bien vous familiariser avec les conversions binaire/décimal.

Pour donner de grands nombres à l'ordinateur

Dans l'ordinateur, les nombres supérieurs à 255 sont stockés sur 2 octets, appelés « octet de poids fort » et « octet de poids faible », comme les adresses. L'octet de poids fort indique combien de fois 256 est contenu dans le nombre, l'octet de poids faible, le reste. Comme pour les adresses, l'ordinateur traite d'abord l'octet de poids faible, que vous devez donc placer en mémoire avant l'octet de poids fort.





Quand vous donnez à l'ordinateur un nombre supérieur à 255, vous devez préciser la valeur de chaque octet. Pour cela, vous divisez le nombre par 256 : la réponse vous donne la valeur de l'octet de poids fort, le reste étant l'octet de poids faible. Pour utiliser le nombre comme donnée dans un programme en langage machine, il faut le convertir en hexa : divisez chaque octet par 16, et transcrivez le résultat et le reste en chiffres hexa, comme indiqué à la page 11.

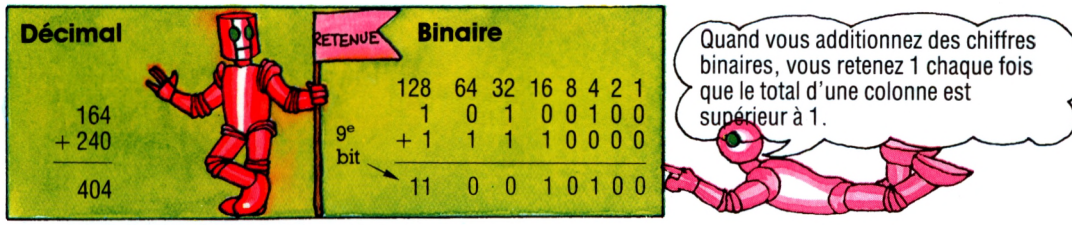
307; 21214; 759; 1023

Trouvez la valeur de l'octet de poids fort et de l'octet de poids faible de ces nombres, en décimal puis en hexa (réponses page 44).

L'indicateur de retenue

L'indicateur de retenue est un simple bit du registre d'état utilisé pour indiquer qu'un résultat est supérieur à 255, et ne peut être stocké sur un seul octet. Dans ce cas, l'indicateur de retenue doit prendre la valeur 1. Cet indicateur joue en quelque sorte le rôle d'un drapeau servant à signaler un événement (d'où l'anglais « carry flag »).

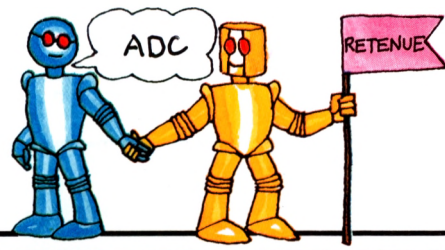
Représentez-vous l'indicateur de retenue comme un neuvième bit indiquant qu'un chiffre binaire 1 a été mis en retenue après le huitième bit d'un octet, comme dans l'exemple ci-dessous.

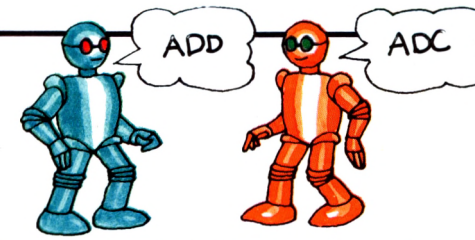


Le résultat de cette addition, 404, occupe 9 bits. Le neuvième indique combien de fois 256 est contenu dans ce nombre. Dans l'ordinateur, ce neuvième bit est le bit de retenue.

L'indicateur de retenue du Z80

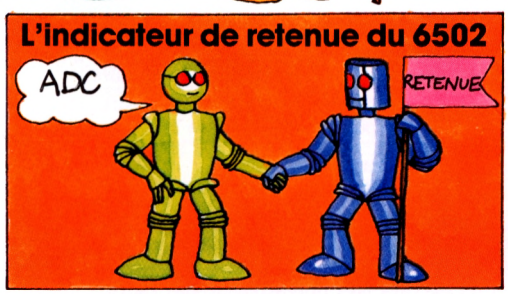
Le Z80 comporte deux instructions d'addition : ADD et ADC. L'instruction ADD dit à l'ordinateur d'additionner deux nombres et d'ignorer les éventuelles retenues des calculs précédents. Si le résultat du calcul impose une retenue, l'ordinateur donnera la valeur 1 à l'indicateur de retenue.



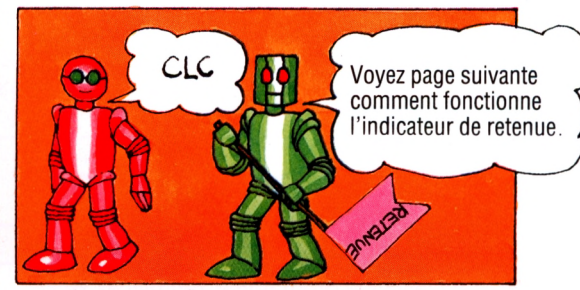


L'instruction ADC (de l'anglais « ADd with Carry » : addition avec retenue) commande à l'ordinateur d'additionner deux nombres plus l'indicateur de retenue et, suivant le résultat, de mettre ou non à 1 l'indicateur de retenue. Quand vous effectuez une série de calculs, il vaut mieux utiliser ADD pour la première somme (vous serez ainsi certain de ne pas inclure la retenue d'un calcul précédent) et ADC pour les opérations suivantes.

L'indicateur de retenue du 6502



Le 6502 ne possède qu'une instruction d'addition, ADC, qui inclut toujours dans les calculs le contenu de l'indicateur de retenue.



C'est pourquoi il est absolument indispensable de remettre celui-ci à zéro en utilisant l'instruction CLC (Clear Carry Flag) avant toute addition.

Programmes pour les grands nombres

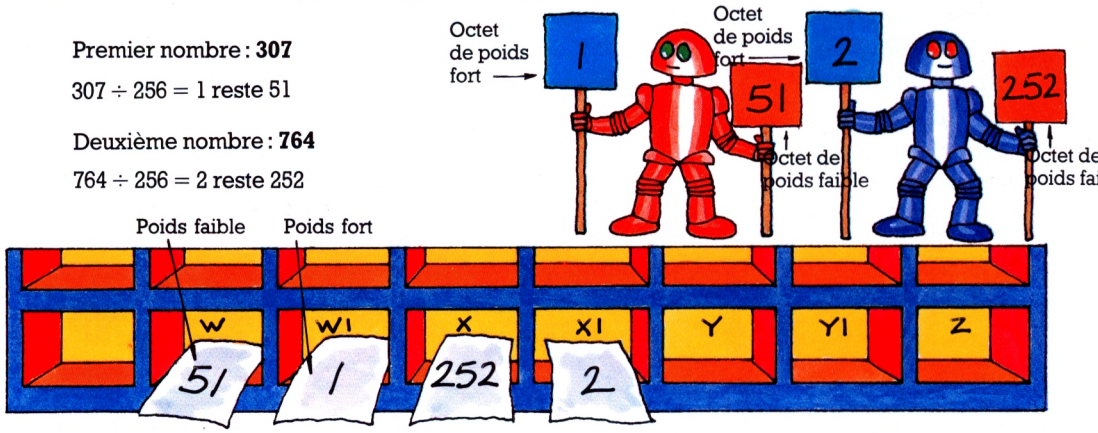
Avant d'exécuter ces programmes, vous devez calculer la valeur des octets de poids fort et de poids faible des nombres que vous allez additionner. Il faudra les mettre en mémoire. Supposons, par exemple, que vous additionnez 307 et 764.

Premier nombre : **307**

307 ÷ 256 = 1 reste 51

Deuxième nombre : **764**

764 ÷ 256 = 2 reste 252



Ensuite, vous mettez ces octets dans les emplacements mémoire situés au début de la zone que vous avez réservée pour le langage machine (n'oubliez pas de mettre l'octet de poids faible d'abord). Ici, les deux octets du premier nombre sont stockés en W et W1, ceux du second nombre en X et X1. Pour le résultat, il vous faut trois emplacements : Y pour l'octet de poids faible, Y1 pour l'octet de poids fort et Z pour une éventuelle retenue.

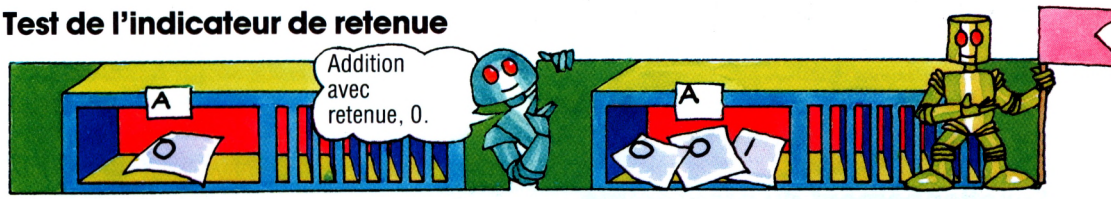
Programme pour le Z80

Ce programme est très simple car le Z80 permet de grouper les registres par paires, chaque paire contenant les deux octets d'un nombre. Vous pouvez ainsi utiliser les registres H et L (dits HL) d'une part, B et C (dits BC) de l'autre. Si vous n'utilisez pas l'accumulateur, remplacez-le par les registres HL. Voici les mnémoniques et les codes hexa du programme. (Pour le comprendre plus facilement, aidez-vous des dessins ci-dessus.)

Mnémoniques	Codes hexa	Commentaires
LD HL,(adresse W)	2A,adresse W	Met le contenu de l'adresse W (octet de poids faible du premier nombre) dans le registre L, et le contenu de l'adresse W1 (octet de poids fort du premier nombre) dans le registre H.
LD BC,(adresse X)	ED 4B,adresse X Cet opcode occupe 2 octets.	Met le contenu de l'adresse X (octet de poids faible du deuxième nombre) dans le registre C, et le contenu de l'adresse X1 (octet de poids fort du deuxième nombre) dans le registre B.
ADD HL,BC	09	Additionne les contenus de HL et BC en laissant les résultats en HL. N'ajoute pas le contenu de l'indicateur de retenue, mais met celui-ci à 1 si nécessaire.
LD (adresse Y),HL	22,adresse Y	Met l'octet de poids faible du résultat à l'adresse Y et l'octet de poids fort à l'adresse Y1.
LD A,&0 ADC A,&0 LD (adresse Z),A	3E,0 CE,0 32,adresse Z	Voir page suivante comment l'ordinateur teste l'indicateur de retenue.
RET	C9	Retour.

Avant de lancer le programme, remplacez W, X, Y et Z par leurs codes hexa (n'oubliez pas d'inverser les deux paires de chiffres hexa). Quand vous utilisez une paire de registres, il suffit de préciser une adresse par paire, l'ordinateur plaçant automatiquement l'octet de l'adresse suivante dans le deuxième registre.

Test de l'indicateur de retenue

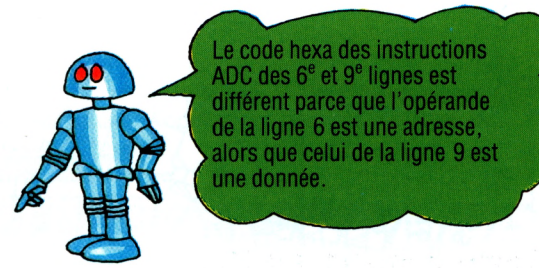


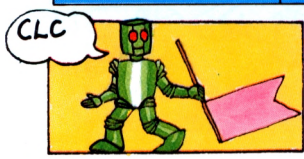
Les lignes 5 à 7 du programme servent à tester l'indicateur de retenue, dont le contenu ne peut être transféré directement dans un registre ou en mémoire. Pour connaître sa valeur, on fait une addition supplémentaire : on met donc la valeur 0 dans l'accumulateur (5e ligne), et on y ajoute 0 en utilisant l'instruction d'addition avec retenue (ADC). Le résultat est donc 0, sauf si l'indicateur de retenue a été utilisé lors du calcul précédent : l'accumulateur prend alors la valeur 1, qui est ensuite stockée à l'adresse Z (7e ligne).

Programme pour le 6502

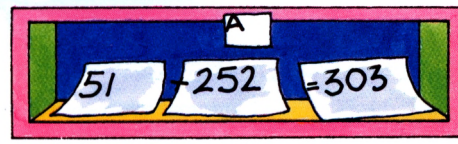
Voici le programme d'addition pour les nombres supérieurs à 255. Avant de le lancer, calculez l'octet de poids fort et l'octet de poids faible des deux nombres, puis mettez-les en mémoire comme indiqué page précédente.

Mnémoniques	Codes hexa
CLC	18
LDA adresse W	AD adresse W
ADC adresse X	6D adresse X
STA adresse Y	8D adresse Y
LDA adresse W1	AD adresse W1
ADC adresse X1	6D adresse X1
STA adresse Y1	8D adresse Y1
LDA #&0	A9 00
ADC #&0	69 00
STA adresse Z	8D adresse Z
RTS	60

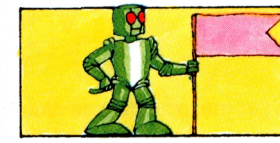




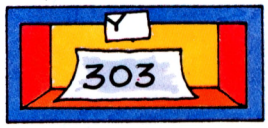
Le programme remet d'abord à zéro l'indicateur de retenue.

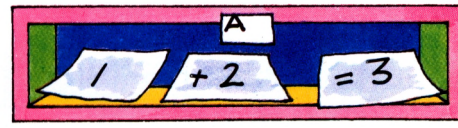


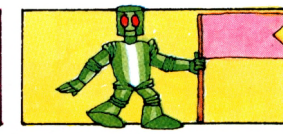
Ensuite, il place dans l'accumulateur l'octet de poids faible du premier nombre et y ajoute (avec la retenue éventuelle) l'octet de poids faible du deuxième nombre.



Si le résultat est supérieur à 255, l'indicateur de retenue prend la valeur 1.





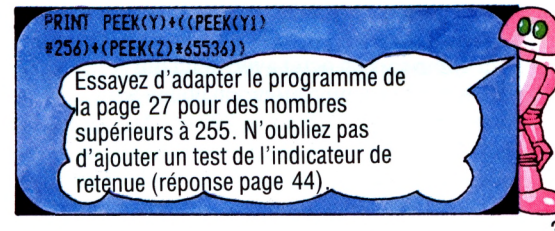


Le résultat est mis dans l'emplacement Y (4e ligne). Puis les deux octets de poids fort et la retenue éventuelle de l'addition précédente sont additionnés. Le résultat est placé en Y1 7e ligne).

Les lignes 8 à 10 testent l'indicateur de retenue selon la méthode indiquée plus haut.

Pour afficher le résultat

Le résultat est stocké sur trois octets. L'octet de poids faible (stocké en Y) indique le nombre d'unités, l'octet de poids fort (Y1) et la retenue (Z) indiquent respectivement combien de fois 256 et 65536 sont contenus dans le nombre. Pour afficher le résultat, utilisez la formule donnée dans l'écran de droite.

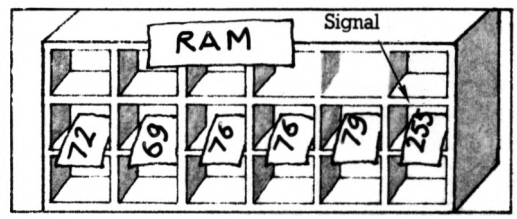


Afficher un message sur l'écran

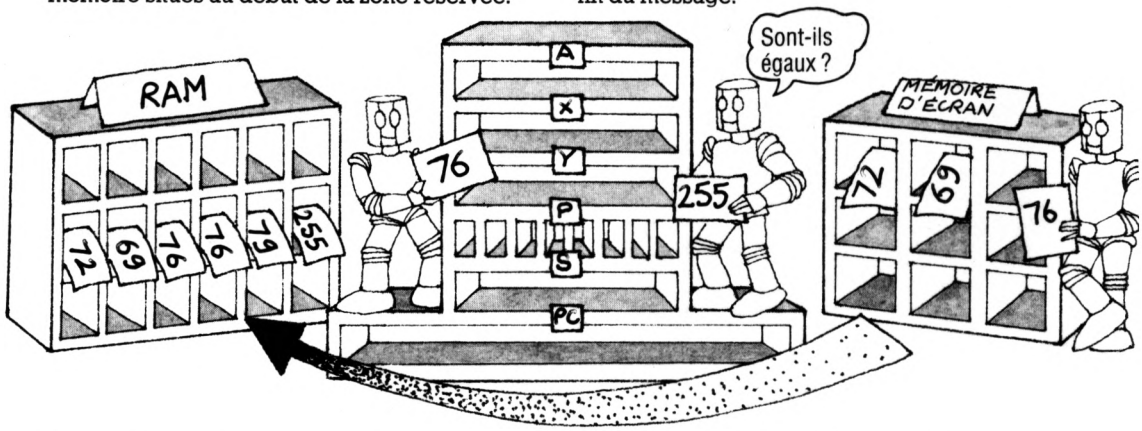
Voici maintenant deux programmes permettant d'afficher un message à l'écran : l'un pour le Z80, l'autre pour le 6502. Bien que différents dans le détail, ils suivent les mêmes principes de base.

Fonctionnement du programme

On va d'abord placer le code ASCII de chaque lettre dans les emplacements mémoire situés au début de la zone réservée.

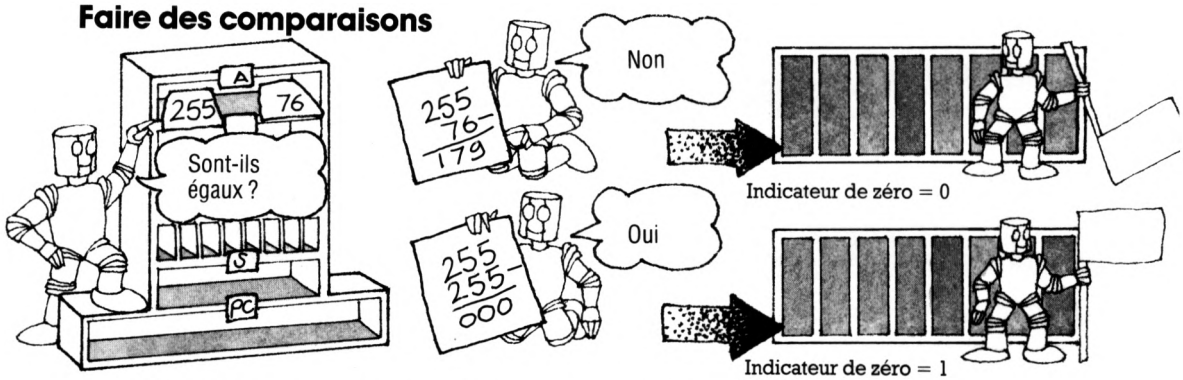


Chaque lettre occupe un octet. On termine par le code 255 qui indique à l'ordinateur la fin du message.



Le programme charge chaque octet du message dans l'accumulateur et le compare avec 255. S'il n'y a pas égalité, l'octet est placé dans la mémoire d'écran et est automatiquement affiché. Puis, l'ordinateur revient au début du programme pour traiter l'octet suivant.

Faire des comparaisons



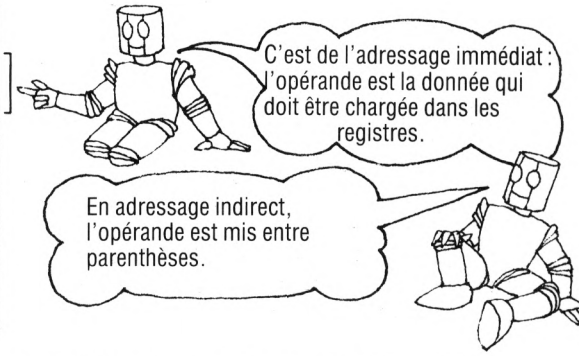
Pour comparer un octet avec celui qui se trouve dans l'accumulateur, on utilise les codes opératoires CP sur le Z80 et CMP sur le 6502. L'ordinateur fait alors un simulacre de soustraction (en fait la valeur des octets n'est pas modifiée) entre les octets : si le résultat est 0, l'indicateur de zéro du registre d'état prend la valeur 1. Sinon, il prend la valeur 0. Suivant le résultat, on commande à l'ordinateur de sauter à une autre partie du programme ou de traiter l'instruction suivante.

* Ce programme ne fonctionne pas correctement sur le Spectrum du fait de l'organisation de sa mémoire d'écran.

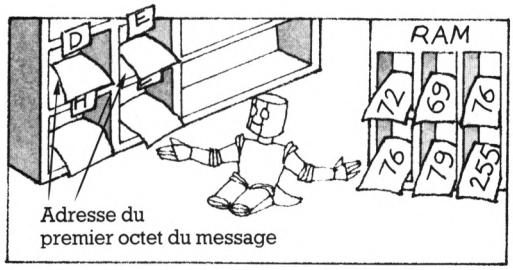
Programme d'affichage pour le Z80

Voici les mnémoniques et les codes hexa du programme. Avant de le lancer, utilisez POKE pour implanter le message dans la zone mémoire que vous avez libérée. Mettez les adresses des lignes 1 et 2. La dernière instruction commande à l'ordinateur de revenir à la troisième instruction (l'adresse de celle-ci doit figurer dans la dernière ligne du programme).

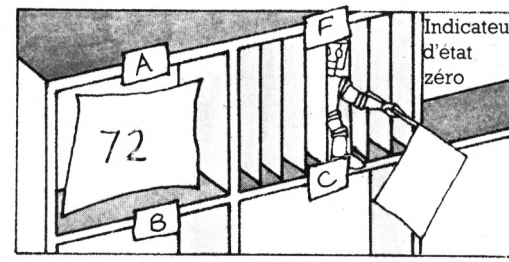
Mnémoniques	Codes hexa
LD HL,adresse d'écran	21,adresse d'écran
LD DE,adresse du message	11,adresse du message
LD A,(DE)	1A
CP &FF	FE,FF
RET Z	C8
LD (HL),A	77
INC DE	13
INC HL	23
JP,adresse de la 3e instruction	C3,adresse de la 3e instruction



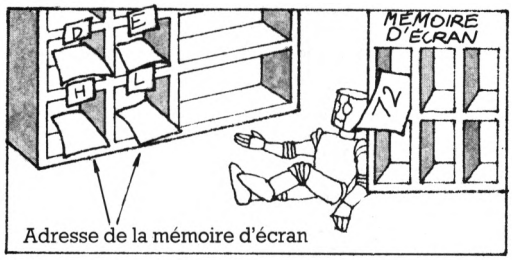
Dans ce programme, les paires de registres HL et DE sont utilisées comme pointeurs des adresses où l'ordinateur doit stocker ou rechercher les données, et les instructions des lignes 3 et 6 utilisent le procédé appelé « adressage indirect ». Dans les deux premières lignes, l'ordinateur met dans les registres HL l'adresse d'écran (celle où seront stockées les données), et dans les registres DE l'adresse du message (celle où l'on ira chercher les données).



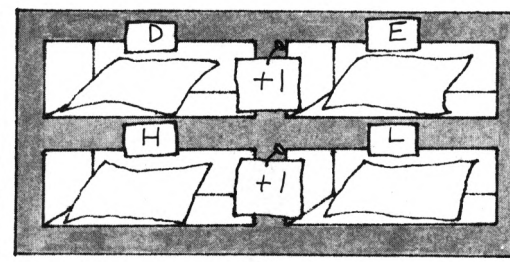
La troisième ligne commande à l'ordinateur de lire l'adresse stockée en DE, d'aller chercher l'octet stocké à cette adresse et de la mettre dans l'accumulateur. C'est ce qu'on appelle l'« adressage indirect ». La quatrième ligne compare l'octet de l'accumulateur au



nombre 255 (en hexa : &FF). La cinquième ligne commande de revenir au BASIC si l'indicateur de zéro est égal à 1 (c'est-à-dire si l'octet est égal à 255). Sinon, l'ordinateur passe à l'instruction suivante.



La sixième ligne utilise aussi l'adressage indirect. Elle commande à l'ordinateur de lire l'adresse qui se trouve en HL, puis de stocker le contenu de l'accumulateur à l'emplacement correspondant à cette adresse. INC signifie



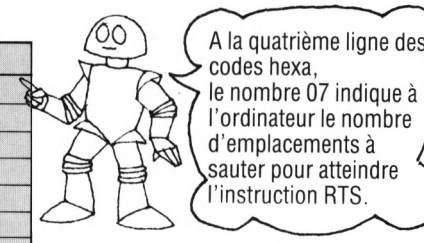
« incrémenter », c'est-à-dire augmenter d'une unité. Les lignes 7 et 8 incrémentent ainsi les adresses stockées en DE et HL pour que l'ordinateur trouve l'octet suivant quand il retourne à la troisième ligne du programme.

Programme d'affichage pour le 6502

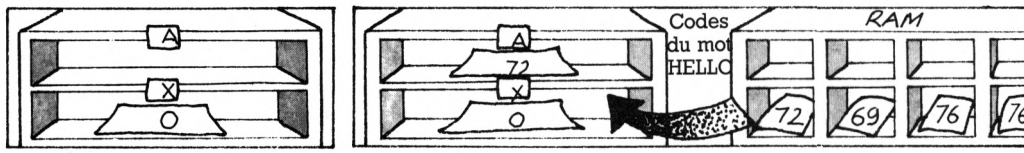
Voici les mnémoniques et les codes hexa correspondant au 6502. Avant de lancer le programme, utilisez POKE pour mettre les codes de votre message dans la zone de mémoire réservée. N'oubliez pas d'ajouter 255 pour signaler la fin du message. A la seconde ligne du programme, mettez l'adresse du premier emplacement mémoire occupé par votre message. A la cinquième ligne, mettez une adresse de la mémoire d'écran de votre ordinateur.

Vous devez aussi compléter la septième ligne avec l'adresse où est stockée la seconde instruction, pour que l'ordinateur puisse y revenir pour répéter le programme.

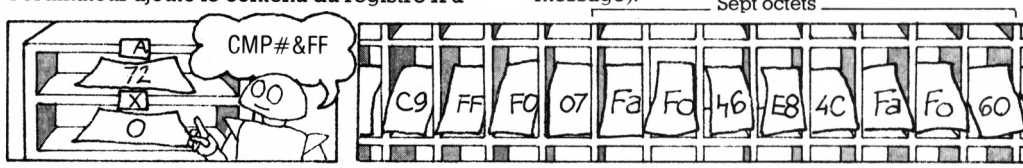
Mnémoniques	Codes hexa
LDX #&00	A2 00
LDA adresse du message,X	BD adresse du message
CMP #18FF	C9 FF
BEQ à instruction RTS	F0 07
STA adresse d'écran,X	9D adresse d'écran
INX	E8
JMP adresse de 2e instruction	4C adresse de 2e instruction
RTS	60



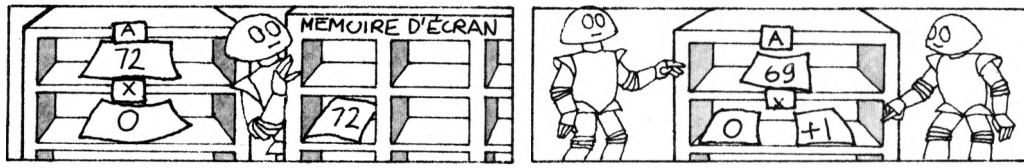
Ce programme utilise un autre type d'adressage, appelé « adressage indexé ». C'est-à-dire que le contenu des registres X ou Y est ajouté à l'opérande pour indiquer l'adresse de la donnée. La deuxième et la cinquième ligne utilisent ce type d'adressage.



A la première ligne, l'ordinateur met la valeur 0 dans le registre X. La seconde instruction utilise l'adressage indexé : l'ordinateur ajoute le contenu du registre X à l'adresse qui suit cette instruction. On obtient ainsi l'adresse de la donnée qui doit être chargée dans l'accumulateur (ici, un octet du message).



A la troisième ligne, CMP dit à l'ordinateur de comparer l'octet qui se trouve dans l'accumulateur, avec le nombre 255 (en hexa &FF), signal de la fin du message. Si les deux nombres sont égaux, l'indicateur de zéro prend la valeur 1. En cas d'égalité, l'instruction BEQ (Branch if EQual) envoie l'ordinateur à la fin du programme. Pour les codes hexa, il faut préciser le nombre d'emplacements que l'ordinateur doit sauter.



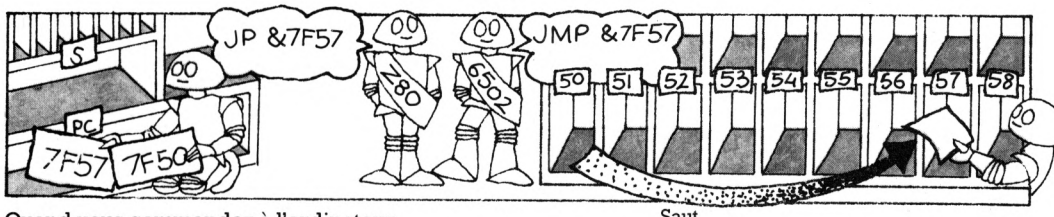
La cinquième ligne utilise l'adressage indexé pour mettre l'octet qui se trouve dans l'accumulateur à l'adresse qui suit l'instruction, augmentée de la valeur X. A la sixième ligne, on ajoute 1 au contenu du registre X grâce à l'instruction INX qui signifie « INcrémenter X ». Ensuite, l'ordinateur retourne à la deuxième instruction pour charger l'octet suivant dans l'accumulateur et le stocker à l'emplacement suivant de la mémoire d'écran.

Sauts et branchements

Envoyer l'ordinateur à une instruction située dans une autre partie du programme s'appelle « faire un branchement ». On distingue trois types de branchements : les sauts, les routines et les branchements conditionnels. Dans ce dernier cas, l'ordinateur effectue un test et, selon le résultat, exécute l'instruction suivante ou se dirige vers une autre partie du programme. Les sauts commandent à l'ordinateur d'aller à l'adresse indiquée.

Le compteur ordinal

Le compteur ordinal est un registre particulier à 16 bits qui contient l'adresse de l'instruction que l'ordinateur doit traiter ensuite. L'ordinateur lit le nombre contenu dans ce compteur et va à l'emplacement correspondant à cette adresse pour chercher la nouvelle instruction. Le compteur est alors augmenté d'une unité pour indiquer l'emplacement mémoire suivant.

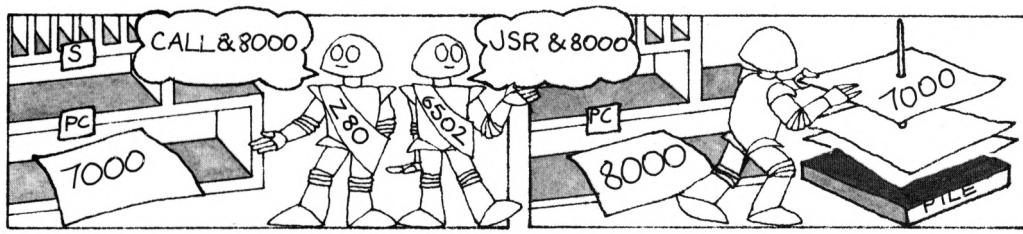


Saut

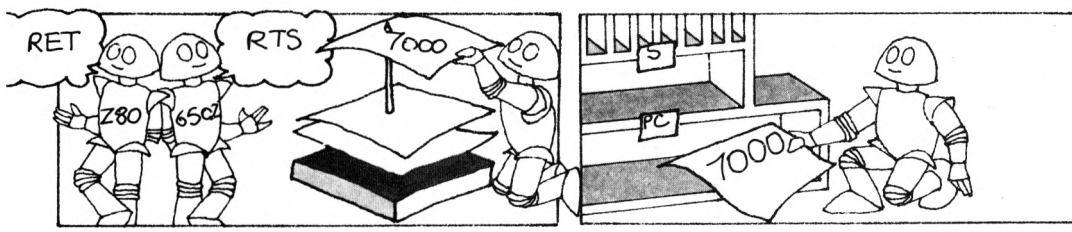
Quand vous commandez à l'ordinateur d'effectuer un saut ou un branchement à une certaine adresse, celle-ci est placée dans le compteur ordinal : l'ordinateur traite alors cette instruction et les instructions qui suivent. Le dessin montre les opcodes de saut (en anglais jump) du Z80 et du 6502.

Les routines

L'instruction « CALL adresse » (Z80) ou « JSR adresse » (6502) commande à l'ordinateur d'aller à la routine indiquée. Comme en BASIC, vous devez mettre une instruction de retour à la fin de la routine (RET pour le Z80 et RTS pour le 6502).



Quand vous commandez à l'ordinateur d'aller à une routine, l'adresse de celle-ci est placée dans le compteur ordinal. Son contenu (l'adresse de l'instruction qui suit CALL ou JSR) est mis de côté et stocké (en anglais « push ») dans la pile, partie de la mémoire réservée pour le microprocesseur (voir page 10).

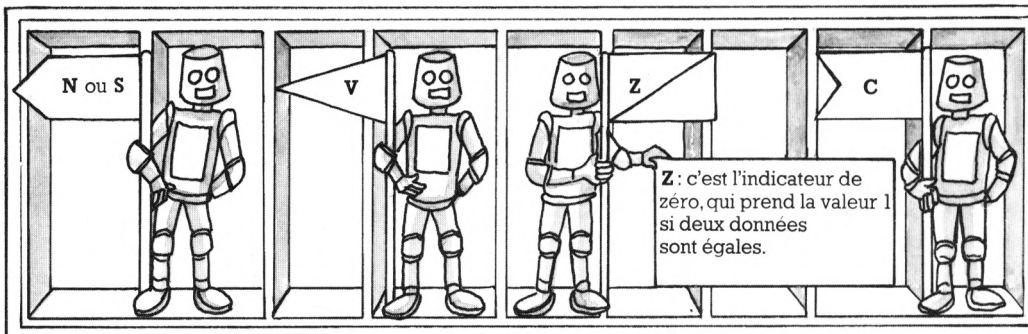


Quand l'ordinateur arrive à l'instruction RET ou RTS à la fin de la routine, il récupère le dernier élément de la pile et le place dans le compteur ordinal.

Cet élément, c'est l'adresse de l'instruction qui suivait celle qui a ordonné d'aller à la routine. Il en va de même si vous exécutez un programme en langage machine.

Branchements conditionnels

Avec le branchement conditionnel, l'ordinateur teste l'un des bits du registre d'état et, selon le résultat, effectue le branchement ou exécute l'instruction suivante. Voici les bits du registre d'état que vous pouvez tester.



N ou **S** : le bit de signe (N pour le 6502 et S pour le Z80) prend la valeur 1 quand le résultat d'un calcul est négatif, 0 quand il est positif.

V ou **P/V** : sur le 6502, on l'appelle bit de dépassement de capacité. Sur le Z80, il remplit deux rôles et s'appelle bit de parité/dépassement de capacité. Bit de dépassement de capacité, il prend la valeur 1 quand le résultat d'un calcul en complémentation à 2 donne une retenue (voir page 37). Bit de parité, il prend la valeur 1 s'il y a un nombre impair de 1 dans un octet et sert pour les tests.

C : C'est l'indicateur de retenue, qui prend la valeur 1 quand le résultat d'une addition ne peut tenir sur un octet.

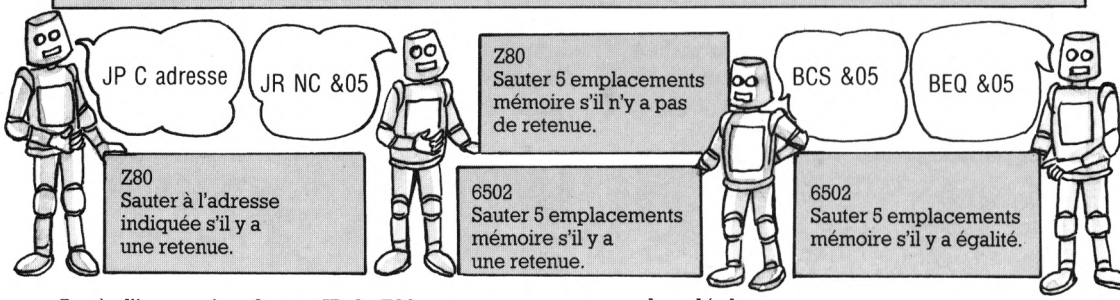
Outre l'instruction de comparaison, différentes instructions agissent sur ces indicateurs. Par exemple, l'instruction DEC (décrémenter) du 6502 affecte les indicateurs de signe et de zéro *.

Opcodes des branchements conditionnels

Voici les instructions de branchements conditionnels qui testent les bits.

Z80	
Sauter si...	
JP C	retenue (C = 1)
JP NC	pas de retenue (C = 0)
JP Z	égalité (Z = 1)
JP NZ	pas d'égalité (Z = 0)
JP M	négatif (S = 1)
JP P	positif (S = 0)
JP PO	parité impaire (P/V = 1)
JP PE	parité paire (P/V = 0)

6502	
Branchement si...	
BCS	retenue (C = 1)
BCC	pas de retenue (C = 0)
BEQ	égalité (Z = 1)
BNE	pas d'égalité (Z = 0)
BMI	négatif (N = 1)
BPL	positif (N = 0)
BVS	dépassement de capacité (V = 1)
BVC	pas de dépassement (V = 0)



Après l'instruction de test JP du Z80, vous précisez à l'ordinateur à quelle adresse il doit sauter. Avec le 6502, vous lui donnez le nombre d'emplacements mémoire qu'il doit sauter (dans un sens ou dans l'autre) pour trouver l'instruction suivante. C'est ce qu'on appelle l'adressage relatif ; le nombre porte le nom de « déplacement ».

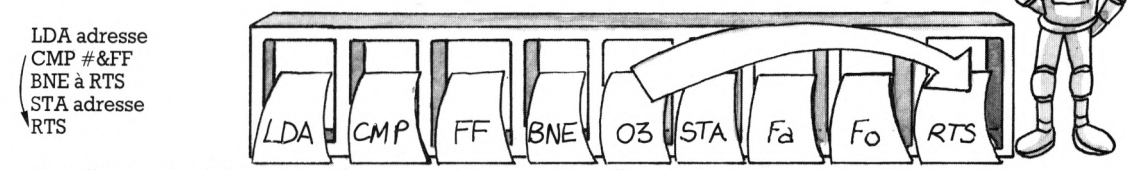
Le Z80 a une autre instruction de branchement conditionnel, « JR test », utilisée avec un déplacement plutôt qu'avec une adresse. Cette instruction, qui signifie « saut relatif », ne permet de tester que les indicateurs de zéro et de retenue.

* La liste complète du jeu d'instructions de votre microprocesseur vous indiquera quels sont les bits affectés par les différentes instructions.

Comment utiliser le déplacement

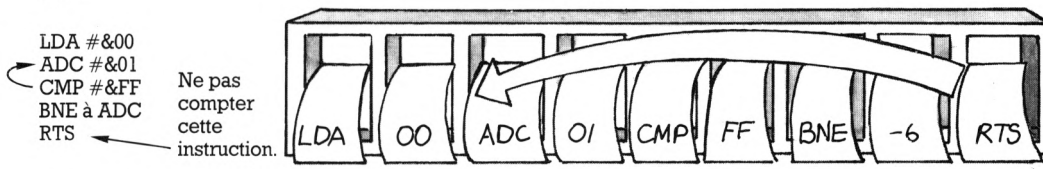
Rappelez-vous qu'une adresse occupe deux octets.

Quand vous précisez un branchement conditionnel par la valeur du déplacement, l'ordinateur calcule l'adresse de l'instruction à laquelle il doit sauter en ajoutant ou en soustrayant la valeur du déplacement au compteur ordinal. Pour calculer le déplacement, comptez le nombre d'octets en commençant après l'instruction de branchement (la première instruction ne compte pas parce que le compteur ordinal est toujours en avance d'une unité) et en incluant l'instruction d'arrivée. Voici deux programmes, conçus pour le 6502, qui illustrent ce processus. (La méthode est la même pour le Z80.)



Dans l'exemple ci-dessus, le déplacement, 3, indique à l'ordinateur de sauter à l'instruction RTS.

Dans l'exemple ci-dessous, le déplacement, – 6, fait revenir l'ordinateur à l'instruction ADC.



Les sauts avant et arrière

Pour effectuer un saut en avant, il suffit de convertir le déplacement en hexa et de l'insérer dans le programme. Pour sauter en arrière, par contre, il faudrait utiliser un nombre négatif, ce qu'on ne peut traduire avec un octet. On utilise donc un système de notation appelé « complémentation à deux ». Le bit de gauche est alors utilisé comme bit de signe : s'il est égal à 1, le nombre est négatif, s'il est égal à 0, le nombre est positif.

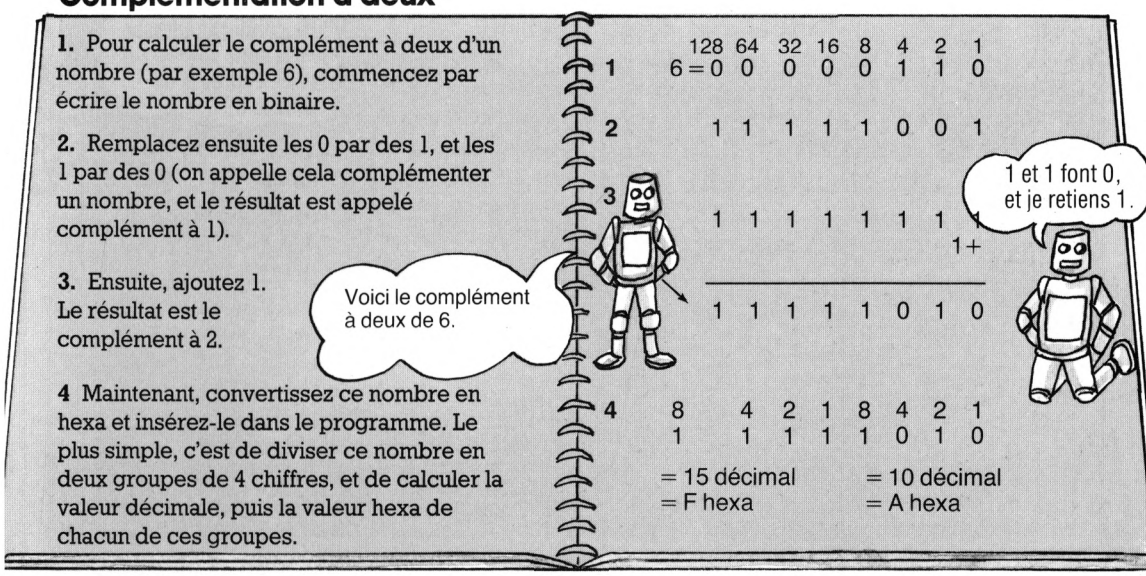
Complémentation à deux

1. Pour calculer le complément à deux d'un nombre (par exemple 6), commencez par écrire le nombre en binaire.

2. Remplacez ensuite les 0 par des 1, et les 1 par des 0 (on appelle cela complémenter un nombre, et le résultat est appelé complément à 1).

3. Ensuite, ajoutez 1. Le résultat est le complément à 2.

4 Maintenant, convertissez ce nombre en hexa et insérez-le dans le programme. Le plus simple, c'est de diviser ce nombre en deux groupes de 4 chiffres, et de calculer la valeur décimale, puis la valeur hexa de chacun de ces groupes.



Ainsi, l'équivalent hexa du complément à deux de 6 est FA, nombre que vous insérerez dans un programme pour un retour en arrière. Le plus grand nombre que vous puissiez exprimer en complémentation à deux est 128 : le déplacement vers l'arrière ne doit donc pas excéder cette valeur. En avant, le déplacement ne doit pas dépasser 127, le huitième bit indiquant que le nombre est positif.

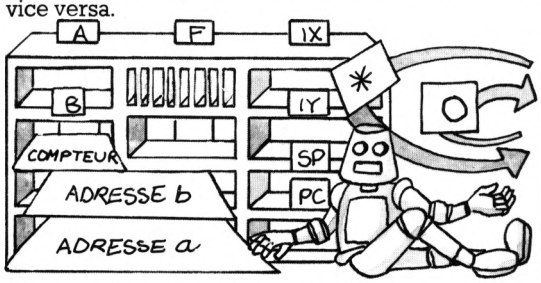
Calculez le complément à deux de 12. 18 et 9 (réponses page 48).

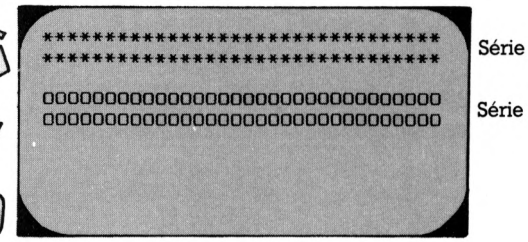
Programme d'affichage clignotant

Voici un programme qui affiche alternativement deux séries de signes sur l'écran. Vous verrez ainsi comment fonctionne un programme d'animation simple. Vous trouverez au bas de la page 39 des conseils pour lancer l'exécution de ce programme.

Z80 Effet de flash

Pour afficher alternativement deux séries de signes, on commence par charger un octet de chaque série dans les registres ; puis on met l'octet de la série b dans la mémoire d'écran de la série a, et vice versa.



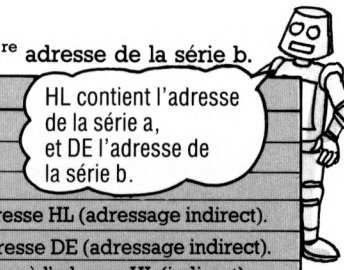


Le programme utilise l'adressage indirect. L'adresse d'écran du premier octet de chaque série est stockée dans les registres HL et DE. Chaque fois que l'ordinateur doit charger ou lire ces octets, il lit les adresses qui se trouvent dans ces registres. Après avoir interverti deux octets, il exécute l'instruction INC qui lui fait ajouter 1 à HL et DE, de sorte que, à chaque répétition du programme, l'ordinateur trouve les adresses des deux octets suivants.

Le registre B contient le nombre d'octets qu'il faut intervertir. A chaque répétition, B, utilisé comme un compteur, est décrémenté de 1. Quand B = 0, tous les octets ont été permutés.

Programme pour le Z80

n = nombre d'octets d'une série ; a = 1re adresse de la série a ; b = 1re adresse de la série b.



Mnémoniques	Codes hexa	Commentaires
LD B,n	06,n	Compteur
LD HL,(adresse a)	21,adresse a	Met l'adresse de la série a en HL.
LD DE,(adresse b)	11,adresse b	Met l'adresse de la série b en DE.
LD C,(HL)	4E	Charge C avec le contenu de l'adresse HL (adressage indirect).
LD A,(DE)	1A	Charge A avec le contenu de l'adresse DE (adressage indirect).
LD (HL),A	77	Stocke le contenu de l'accumulateur à l'adresse HL (indirect).
LD A,C	79	Met C (premier octet de la série a) dans l'accumulateur.
LD (DE),A	12	Stocke le contenu de l'accumulateur à l'adresse DE.
INC HL INC DE	23 13	Ajoute 1 à HL et DE.
DEC B	05	Décrémente le compteur, B.
LD A,&00	3E,00	Met 0 dans l'accumulateur.
CP B	B8	Compare B au contenu de l'accumulateur (0).
JR NZ à instruction 4	20,F3	Si B différent de 0, revient en arrière en sautant &F3 emplacements pour charger les octets suivants dans les registres. F3 est le complément hexa à deux du nombre 13 (voir page 37).
RET	C9	Retour.

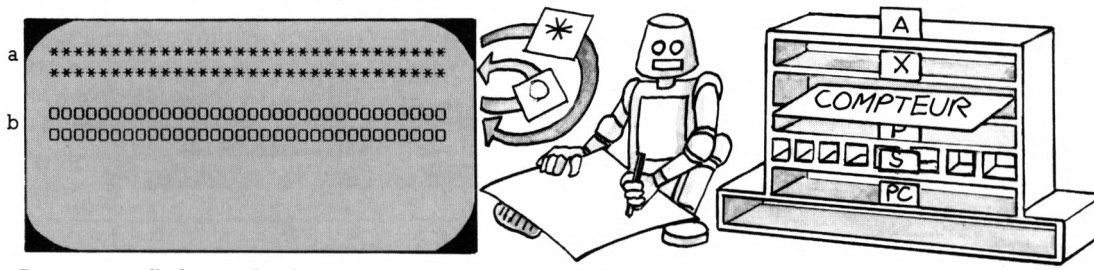
Pour compléter les données et les adresses

n : pour trouver n (nombre de caractères d'une série), multipliez le nombre de caractères d'une ligne par le nombre de lignes d'une série. Convertissez en hexa.

adresses a et b : si vous voulez permuter les deux premières lignes de l'écran avec les deux lignes suivantes, donnez à a la valeur de la première adresse de la mémoire d'écran de votre ordinateur. Donnez à b la valeur de l'adresse a augmentée du nombre d'octets que vous souhaitez permuter. Convertissez a et b en hexa.

6502 Effet de flash

Ce programme permute les deux séries, octet par octet (c'est-à-dire caractère par caractère), en commençant par le dernier octet de chaque série. Il charge ces octets dans les registres, puis stocke l'octet de la série a à l'adresse d'écran de la série b, et vice versa. Le programme est ensuite répété pour échanger la paire d'octets suivants.



Pour trouver l'adresse de chaque octet, on utilise l'adressage indexé : on charge dans le registre X le nombre total d'octets, et ensuite, pour charger ou stocker un octet, on ajoute le nombre placé dans le registre X à l'adresse du début de chaque série. L'instruction DEX (décrémenter X) enlève 1 à X, de sorte que le programme va chercher, à chaque répétition, l'octet suivant.

Programme pour le 6502

Pour déterminer la valeur de n, a et b, reportez-vous au bas de la page 38. Retranchez 1 à la valeur hexa de a et b pour que l'ordinateur trouve la dernière adresse de chaque série plutôt que la première adresse de la ligne suivante.

Mnémoniques	Codes hexa	Commentaires
LDX #n	A2 n	Charge X avec le nombre d'octets d'une série.
LDA adresse a,X	BD adresse a	Contenu de l'emplacement d'adresse a + X dans l'accumulateur.
TAY	A8	Transfère le contenu de l'accumulateur dans le registre Y.
LDA adresse b,X	BD adresse b	Contenu de l'emplacement d'adresse b + X dans l'accumulateur.
STA adresse a,X	9D adresse a	Stocke le contenu de l'accumulateur à l'adresse a + X.
TYA	98	Transfère le contenu du registre Y dans l'accumulateur.
STA adresse b,X	9D adresse b	Stocke le contenu de l'accumulateur à l'adresse B + X.
DEX	CA	Décrémente X. L'indicateur d'état zéro est mis à 1 quand X = 0.
BNE à instruction 2	D0 EF	Revient en arrière et saute &EF emplace. si X n'est pas égal à 0. EF est l'équivalent hexa de la complémentation à deux de 17 (p. 37).
RTS	60	Retour.

Charger et lancer le programme

La meilleure façon de lancer ce programme est de l'inclure sous forme de sous-programme en langage machine dans le programme de conversion hexa.

1. Chargez le programme de conversion hexa et mettez les codes hexa de votre ordinateur à la ligne 160.

2. A la ligne 180, ajoutez deux boucles pour aller mettre les caractères à afficher dans la mémoire d'écran. Voici ce que cela donne pour un écran de 40 colonnes sur lequel on affiche deux rangée de * (code 42) puis deux rangées de 0 (code 48).

```
180  FOR J=0  TO  79
190  POKE première adresse d'écran + J,42
200  NEXT J
210  FOR J=80  TO 159
220  POKE première adresse d'écran + J,48
230  NEXT J
```

3. Ajoutez ensuite à la fin du programme les lignes suivantes :

```
240  CALL adresse où est stocké
le langage machine
250  FOR K=1  TO 500
260  NEXT K
270  GOTO 240
```

Adapter le nombre 500 de la boucle de temporisation à votre ordinateur

4. Tapez RUN. Le programme va mettre les codes hexa en mémoire, puis les codes d'affichage dans la mémoire d'écran. La ligne 240 envoie à l'emplacement où est stocké le programme en langage machine. Celui-ci ne permutant qu'une fois les lignes affichées, la ligne 270 rappelle le programme en langage machine pour créer un effet de flash. Comme le langage machine est très rapide, il faut une boucle de temporisation.

Pour aller plus loin

Pour en savoir plus sur le langage machine, il ne vous reste plus qu'à écrire de petits programmes, puis à essayer et à comprendre les programmes des autres. Le langage machine est très utile pour des sous-programmes inclus dans un programme BASIC et chargés d'une tâche bien spécifique, comme le tri de données ou l'affichage d'effets graphiques (le langage machine étant plus rapide et occupant moins de mémoire que le BASIC). Vous trouverez des sous-programmes de ce genre dans les revues. S'ils sont écrits spécialement pour votre ordinateur, vous les utiliserez tels quels. S'ils sont écrits pour un micro-ordinateur qui a le même microprocesseur que le vôtre, vous devrez modifier certaines adresses.

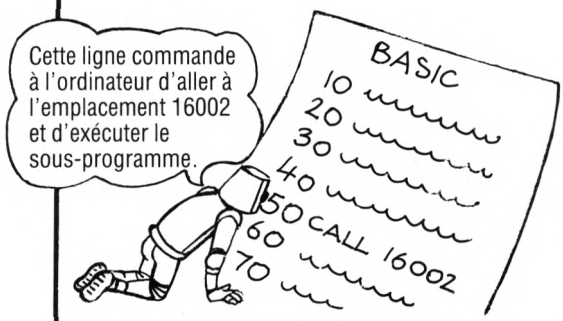
Sous-programmes en langage machine

Voici les étapes à respecter pour intégrer un sous-programme en langage machine à un programme BASIC.

1. Faire de la place en mémoire pour le langage machine en abaissant la limite de la mémoire utilisateur (voir pages 20-22).

2. Mettre les codes hexa du sous-programme à la ligne 160 du programme de conversion de la page 24. (N'oubliez pas l'instruction de retour à la fin du sous-programme en langage machine.) Ajoutez éventuellement des lignes pour stocker les résultats en mémoire.

3. Numérotez votre programme BASIC comme s'il continuait le programme de conversion. A l'endroit où vous désirez que votre ordinateur exécute le sous-programme en langage machine, mettez l'instruction



utilisée par votre ordinateur pour passer au langage machine, comme une simple ligne BASIC.

4. Rentrez votre programme BASIC et lancez-le. L'ordinateur va exécuter les instructions BASIC. Quand il arrivera à la ligne qui lui commande d'appeler le sous-programme en langage machine, il se rendra à l'adresse indiquée et exécutera les instructions. L'instruction de Retour renverra l'ordinateur à la ligne suivante du programme BASIC.

Utiliser un assembleur

Un assembleur est un programme (généralement disponible sur cassette ou disquette, mais parfois stocké en ROM) qui vous permet de rentrer un programme en langage machine en utilisant les mnémoniques, ce qui est plus facile.

Vous pourrez aussi écrire des commentaires pour retrouver plus facilement la signification de chaque ligne. L'assembleur affichera le programme à l'écran sous forme de mnémoniques et de nombres hexa, avec les adresses et les commentaires.

En outre, il permutera automatiquement les deux paires de chiffres hexa des adresses, et calculera les adresses et les déplacements. Certains assembleurs vous permettront aussi d'utiliser des noms pour les données (comme les variables en BASIC). Un bon assembleur doit comporter un « débuggeur » qui repère les fautes, et un éditeur qui permet de corriger facilement le programme.

Livres conseillés

Il existe de très nombreux livres sur le langage machine, mais la plupart sont écrits pour un micro-ordinateur particulier. Si ce type de livre vous intéresse, consultez donc les revues spécialisées.

En ce qui concerne simplement les microprocesseurs Z80 et 6502, vous pouvez vous reporter aux ouvrages suivants :

L'assembleur facile du Z80, par O. Lepape, Éditions Eyrolles.

L'assembleur facile du 6502, par F. Monteil, Éditions Eyrolles.

Programmation du Z80 et *Programmation du 6502*, par R. Zaks, Éditions Sybex.

Programmation en langage assembleur, par L.A. Leventhal, Éditions Radio : un volume pour le Z80, un volume pour le 6502. (Ces quatre volumes ne sont pas faciles d'accès pour les débutants, mais sont très complets.)

Tables de conversion décimal/hexadécimal

Ce tableau permet de convertir les nombres hexa de 0 à FF en décimal et vice versa.

D'hexa en décimal

Cherchez la ligne correspondant au premier chiffre hexa de votre nombre et la colonne correspondant au second chiffre hexa.
A l'intersection : le décimal du nombre hexa.

De décimal en hexa

Cherchez le nombre décimal dans le tableau. Lisez d'abord le chiffre hexa correspondant à cette ligne, puis celui qui correspond à la colonne. Par exemple, l'équivalent hexa de 154 est 9A.

Premier chiffre hexa \ Second chiffre hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Pour convertir les adresses

Cherchez d'abord l'équivalent décimal de la première paire de chiffres hexa de l'adresse : vous obtiendrez ainsi le numéro de la page. Cherchez ensuite l'équivalent de la seconde paire de chiffres hexa pour obtenir la position dans la page. Multipliez le numéro de la page par 256 et ajoutez-y la position dans la page.

Tableau pour la complémentation à deux

Ce tableau vous donne le complément à deux, en hexa, des nombres décimaux allant de − 1 à − 128. Pour trouver le complément à deux d'un nombre, cherchez celui-ci dans le tableau : le premier chiffre hexa est le chiffre correspondant à la ligne, le second celui qui correspond à la colonne.

1er chiffre hexa \ Second chiffre hexa	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
E	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
D	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
B	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
A	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
9	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128

Faire des conversions avec une calculatrice

Par exemple, pour convertir 134 en hexa, vous divisez d'abord par 16, puis vous cherchez l'équivalent hexa du résultat et du reste. Une calculatrice vous donnera le résultat suivant : 8.375. Le nombre entier 8 correspond au premier chiffre hexa ; vous obtiendrez le second chiffre hexa en multipliant les décimales par 16 :
$0.375 \times 16 = 6$.
Le résultat est donc : $134 \div 16 = 8$ reste 6, soit 86 en hexa.

Mnémoniques et codes hexa du microprocesseur Z80

Ces trois pages présentent toutes les instructions données dans ce livre. « Adressage implicite » signifie que, pour les codes hexa, il n'est pas nécessaire de spécifier l'opérande. Toutes les instructions ne figurent pas dans cette liste, et si vous voulez exploiter toutes les possibilités du Z80, vous devez vous procurer la liste complète du jeu d'instructions (voir p. 40). Ci-dessous les abréviations :

n = nombre **r** = registre **x** = adresse **d** = deplacement
nn = nombre sur 2 octets **rr** = paire de registres **c** = condition

ADC A,n Additionne la retenue et un nombre, n, à l'accumulateur. (Adressage immédiat.)

ADC A,n	CE,n

ADC A,r Additionne la retenue et le registre r à l'accumulateur. (Adressage implicite.)

ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D

ADC HL,rr Additionne la retenue et le contenu de la paire de registres rr à HL. (Adressage implicite.)

ADC HL,BC	ED4A
ADC HL,DE	ED5A
ADC HL,HL	ED6A

ADD A,n Additionne un nombre, n, à l'accumulateur. (Adressage immédiat.)

ADD,n	C6,n

ADD A,r Additionne un nombre, n, à l'accumulateur. (Adressage implicite.)

ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85

ADD HL,rr Additionne le contenu de la paire de registres rr à HL. (Adressage implicite.)

ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29

CALL x Va à la routine qui commence à l'adresse x. (Adressage immédiat.)

CALL x	CD x

CALL c,x Va à la routine d'adresse x en fonction de c. c peut être Z (égalité), NZ (pas d'égalité), C (retenue), NC (pas de retenue), PE (parité paire), PO (parité impaire), M (négatif), P (positif). (Adressage immédiat.)

CALL Z,x	CC,x
CALL NZ,x	C4,x
CALL C,x	DC,x
CALL NC,x	D4,x
CALL PE,x	EC,x
CALL PO,x	E4,x
CALL M,x	FC,x
CALL P,x	F4,x

CCF Complémentation de l'indicateur de retenue. (Adressage implicite.)

CCF	3F

CP n Compare le contenu de l'accumulateur avec la donnée n. (Adressage immédiat.)

CP n	FE n

CP r Compare le contenu du registre r avec l'accumulateur. (Adressage implicite.)

CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD

CP (HL) Compare le contenu de l'accumulateur avec celui de l'adresse stockée en HL. (Adressage indirect.)

CP (HL)	BE

DEC r Décrémente le registre r. (Adressage implicite.)

DEC A	3D
DEC B	05
DEC C	0D
DEC D	15
DEC E	1D
DEC H	25
DEC L	2D

DEC rr Décrémente la paire de registres rr. (Adressage implicite.)

DEC BC	0B
DEC DE	1B
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B

DEC (HL) Décrémente le contenu de l'adresse stockée en HL. (Adressage indirect.)

DEC (HL)	35

INC r Incrémente le registre r. (Adressage implicite.)

INC A	3C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC H	24
INC L	2C

INC rr Incrémente la paire de registres rr. (Adressage implicite.)

INC BC	03
INC DE	13
INC HL	23

INC (HL) Incrémente le contenu de l'adresse stockée en HL. (Adressage indirect.)

INC (HL)	34

JP x Saute à l'adresse x. (Adressage immédiat.)

JP x	C3 x

JP (rr) Saute à l'adresse stockée dans la paire de registres rr. (Adressage implicite.)

JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9

JP c,x Saute à l'adresse x en fonction de c.
c peut être Z (égalité), NZ (pas d'égalité), C (retenue), NC (pas de retenue), PE (parité paire), PO (parité impaire), M (négatif), P (positif).
(Adressage immédiat.)

JP Z,x	CA,x
JP NZ,x	C2,x
JP C,x	DA,x
JP NC.x	D2,x
JP PE,x	EA,x
JP PO,x	E2,x
JP M,x	FA,x
JP P,x	F2,x

JR d Saut relatif. Saute d'octets (déplacement).
(Adressage relatif.)

JR d	18 d

JR c,d Saut relatif. Saute d'octets en fonction de c. c peut être NZ (pas d'égalité), Z (égalité), NC (pas de retenue), C (retenue).
(Adressage relatif.)

JR NZ,d	20,d
JR Z,d	28,d
JR NC,d	30,d
JR C,d	38,d

LD r,n Charge le registre r avec la donnée n. (Adressage immédiat.)

LD A,n	3E,n
LD B,n	06,n
LD C,n	0E,n
LD D,n	16,n
LD E,n	1E,n
LD H,n	26,n
LD L,n	2E,n

LD rr,nn Charge la paire de registres rr avec le nombre nn codé sur 2 octets. (Adressage immédiat.)

LD BC,nn	01,nn
LD DE,nn	11,nn
LD HL,nn	21,nn

LDA, (x) Charge l'accumulateur avec le contenu de l'adresse x. (Adressage absolu.)

LD A, (x)	3A, (x)

LD rr, (x) Charge la paire de registres rr avec le contenu des adresses x et x + 1. (Adressage absolu.)

LD BC, (x)	ED4B, (x)
LD DE, (x)	ED5B, (x)
LD HL, (x)	2A, (x)

LD A,r Charge l'accumulateur avec le contenu du registre r. (Adressage implicite.)

LD A,A	7F
LD A,B	78
LD A,C	79
LD A,D	7A
LD A,E	7B
LD A,H	7C
LD A,L	7D

LD B,r Charge le registre B avec le contenu du registre r. (Adressage implicite.)

LD B,A	47
LD B,B	40
LD B,C	41
LD B,D	42
LD B,E	43
LD B,H	44
LD B,L	45

LD C,r Charge le registre C avec le contenu du registre r. (Adressage implicite.)

LD C,A	4F
LD C,B	48
LD C,C	49
LD C,D	4A
LD C,E	4B
LD C,H	4C
LD C,L	4D

LD D,r Charge le registre D avec le contenu du registre r. (Adressage implicite.)

LD D,A	57
LD D,B	50
LD D,C	51
LD D,D	52
LD D,E	53
LD D,H	54
LD D,L	55

LD E,r Charge le registre E avec le contenu du registre r. (Adressage implicite.)

LD E,A	5F
LD E,B	58
LD E,C	59
LD E,D	5A
LD E,E	5B
LD E,H	5C
LD E,L	5D

LD H,r Charge le registre H avec le contenu du registre r. (Adressage implicite.)

LD H,A	67
LD H,B	60
LD H,C	61
LD H,D	62
LD H,E	63
LD H,H	64
LD H,L	65

LD L,r Charge le registre L avec le contenu du registre r. (Adressage implicite.)

LD L,A	6F
LD L,B	68
LD L,C	69
LD L,D	6A
LD L,E	6B
LD L,H	6C
LD L,L	6D

LD r,(rr) Charge le registre r avec le contenu de l'adresse stockée dans la paire de registre rr. (Adressage indirect.)

LD A, (BC)	0A
LD A,(DE)	1A
LD A,(HL)	7E
LD B,(HL)	46
LD C,(HL)	4E
LD D,(HL)	56
LD E,(HL)	5E
LD H,(HL)	66
LD L,(HL)	6E

LD (x),A Stocke le contenu de l'accumulateur à l'adresse x. (Adressage absolu.)

LD (x),A	32,x

LD (x),rr Stocke le contenu des registres rr aux adresses x et x + 1.
(Adressage absolu.)

LD (x),BC	ED43,x
LD (x),DE	ED53,x
LD (x),HL	22,x

LD (rr),r Stocke le contenu du registre r à l'adresse contenue dans la paire de registres rr. (Adressage indirect.)

LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75

LD (rr),n Stocke la donnée n à l'adresse contenue dans la paire de registres rr. (Adressage immédiat/indirect.)

LD (HL),n	36

RET Retour de routine. (Adressage indirect.)

RET	C9

RET c Retour de routine en fonction de c. c peut être Z (égalité), NZ (pas d'égalité), C (retenue), NC (pas de retenue), PE (parité paire PO (parité impaire), P (positif, M (négatif). (Adressage indirect.)

RET Z	C8
RET NZ	C0
RET C	D8
RET NC	D0
RET PE	E8
RET PO	E0
RET M	F8
RET P	F0

SBC A,n Soustrait la retenue et la donnée n de l'accumulateur. (Adressage immédiat.)

SBC A,n	DE,n

SBC A,r Soustrait la retenue et le contenu du registre r de l'accumulateur. (Adressage implicite.)

SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D

SBC HL,rr Soustrait la retenue et le contenu des registres rr des registres HL (Adressage implicite.)

SBC HL,BC	ED42
SBC HL,DE	ED52
SBC HL,HL	ED62

SBC A, (HL) Soustrait la retenue et le contenu de l'adresse stockée dans les registres HL, de l'accumulateur. (Adressage indirect.)

SBC A,(HL)	9E

SCF Met à 1 l'indicateur de retenue. (Adressage implicite.)

SCF	37

SUB n Soustrait la donnée n de l'accumulateur. (Adressage immédiat.)

SUB, n	D6, n

SUB r Soustrait le registre r de l'accumulateur. (Adressage immédiat.)

SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95

SUB (HL) Soustrait le contenu de l'adresse stockée en HL de l'accumulateur. (Adressage indirect.)

SUB (HL)	96

Solutions

Page 11

& A7 est 167 en décimal. 513 en hexa est & 210

Pour calculer facilement le complément à 2 d'un nombre : soustrayez-le de 256, puis convertissez la réponse en hexa. Ex : 256 − 6 = 250, soit FA en hexa.

Page 26

1.25 + 73 (25 est & 19 et 73 est & 49)

Z80		6502		Commentaires
Mnémoniques	Codes hexa	Mnémoniques	Codes hexa	
LD A, &19	3E,19	LDA #&19	A9 19	Met &19 dans l'accumulat.
ADD A, &49	C6,49	ADC #&49	69 49	Ajoute &49 à l'accumulat.
LD (adresse), A	32, adresse	STA adresse	8D adresse	Stocke le contenu de l'accumulateur à l'adresse indiquée.
RET	C9	RTS	60	Retour

2.64 + 12 + 14 (64 est &40, 12 est &0C et 14 est &0E

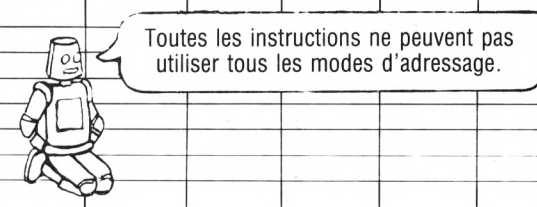
Z80		6502		Commentaires
Mnémoniques	Codes hexa	Mnémoniques	Codes hexa	
LD A, &40	3E,40	LDA #&40	A9 40	Mets &40 dans l'accumulat.
ADD A, &0C	C6,0C	ADC #&0C	69 0C	Ajoute &OC à l'accumulat.
ADD A, &0E	C6,0E	ADC #&0E	69 0E	Ajoute &OE à l'accumulat.
LD(adresse), A	32, adresse	STA adresse	8D adresse	Stocke le contenu de l'accumulateur de l'adresse indiquée.
RET	C9	RTS	60	Retour

Suite des solutions page 48.

Mnémoniques et codes hexa du microprocesseur 6502

Voici la liste des instructions et codes hexa présentés dans ce livre. Les mnémoniques sont expliqués dans la partie gauche ; les codes hexa correspondant aux différents modes d'adressage figurent dans la partie droite. L'adressage en page 0 est identique à l'adressage absolu (l'opérande indique l'adresse où est stockée la donnée), mais l'adresse doit se trouver en page 0 (emplacements 0-255) de la mémoire. En adressage implicite, il n'est pas nécessaire de spécifier l'opérande (ex : CLC). Si vous voulez exploiter toutes les possibilités du 6502, procurez-vous la liste complète de son jeu d'instructions (p. 40).

Mode d'adressage	*Immédiat*	*Absolu*	*Page zéro*	*Indexé X*	*Indexé Y*	*Implicite*	*Relatif*
Opérande	Donnée	Adresse	Adresse en page zéro	Adresse + registre X	Adresse + registre Y	Rien	Déplacement
ADC Addition avec retenue (additionne un octet, plus le bit de retenue, à l'accumulateur).	69	6D	65	7D	79		
BCC Branchement si bit de retenue = 0.							90
BCS Branchement si bit de retenue = 1.							B0
BEQ Branchement si égalité.							F0
BMI Branchement si le nomb. est négatif.							30
BNE Branchement si pas d'égalité.							D0
BPL Branchement si le nombre est positif.							10
BVC Branchement si pas de dépassement de capacité.							50
BVS Branchement si dépassement de capacite.							70
CLC Met à zéro le bit de retenue.						18	
CMP Compare avec l'accumulateur.	C9	CD	C5	DD	D9		
CPX Compare avec le registre X.	E0	EC	EH				
CPY Compare avec le registre Y.	C0	CC	C4				
DEC Décrémente (retranche 1 à) l'emplacement mémoire.		CE	C6	DE			
DEX Décrémente le registre X.						CA	
DEY Décrémente le registre Y.						88	
INC Incrémente (ajoute 1) l'emplacement memoire.		EE	E6	FE			
INX Incrémente le registre X.						E8	
INY Incrémente le registre Y.						C8	
JMP Saute à l'adresse spécifiée par l'opérande.		4C					
JSR Saute à la routine qui se trouve à l'adresse spécifiée par l'opérande.		20					
LDA Charge l'accumulateur.	A9	AD	A5	BD	B9		
LDX Charge le registre X.	A2	AE	A6		BE		
LDY Charge le registre Y.	A0	AC	A4	BC			
RTS Retour de routine.						60	
SBC Soustraction avec retenue. Soustrait de l'accumulateur et prend en compte le bit de retenue.	E9	ED	E5	FD	F9		
SEC Met à 1 l'indicateur de retenue.						38	
STA Stocke l'accumulateur a l'adresse indiquee.		8D	85	9D	99		
STX Stocke le registre X à l'adresse indiquee.		8E	86				
STY Stocke le registre Y à l'adresse indiquee.		8C	84				
TAX Transfère l'accumulateur au registre X.						AA	
TAY Transfère l'accumulateur au registre Y.						A8	
TXA Transfère le registre X dans l'accumulateur.						8A	
TYA Transfère le registre Y dans l'accumulateur.						98	



Lexique

Ce symbole est souvent utilisé pour signaler les nombres hexadécimaux. Sur le 6502, il sert à signaler une donnée.
& Autre symbole fréquemment utilisé pour signaler les nombres hexadécimaux.
Accumulateur : Registre où l'on place les octets sur lesquels seront effectuées des opérations arithmétiques ou logiques.
Adressage absolu : Mode d'adressage dans lequel l'instruction est suivie de l'adresse de la donnée. Appelé également adressage étendu ou direct.
Adressage implicite : Avec ce mode d'adressage, il n'est pas nécessaire de spécifier l'opérande si le programme est écrit en codes hexa.
Adressage indexé : Mode d'adressage dans lequel on ajoute le contenu d'un registre d'index à l'adresse précisée dans l'instruction pour calculer l'adresse réelle de la donnée.
Adressage indirect : Mode d'adressage dans lequel l'opérande sert de pointeur de données. L'opérande, qui peut être une adresse ou, sur le Z80, une paire de registres, contient l'adresse de la donnée.
Adressage page zéro : Mode d'adressage utilisé seulement sur le 6502 : l'opérande correspond toujours à une adresse de la page zéro de la mémoire.
Adressage relatif : Mode d'adressage dans lequel l'ordinateur calcule l'adresse de l'instruction suivante en additionnant le nombre appelé « déplacement » à l'adresse qui figure dans le compteur ordinal.
Adresse : Nombre servant à identifier un emplacement de la mémoire de l'ordinateur.
Adresse absolue : C'est l'adresse réelle d'une donnée.
Assembleur : Programme qui convertit en code machine les instructions écrites en mnémoniques.
Binaire : Système numérique utilisant deux chiffres, 0 et 1. Dans un nombre binaire, chaque chiffre vaut le double du chiffre placé à sa droite.
Bit : Unité du code machine, traduite par un 1 ou un 0, qui correspond à une impulsion (tension haute) ou à une absence d'impulsion (tension basse).
Bit de retenue : Bit du registre d'état, mis à 1 quand le résultat d'une addition ne peut tenir sur un octet.
Bit de signe : Bit du registre d'état qui indique si un nombre est positif ou négatif.
Bit de zéro : Bit du registre d'état, utilisé pour indiquer que le résultat d'une opération est zéro, ou que deux octets sont égaux.
Branchement : Instruction qui ordonne à l'ordinateur d'aller à une autre ligne du programme.
Branchement conditionnel : Instruction qui ordonne à l'ordinateur de tester un bit du registre d'état, et, suivant le résultat, d'exécuter la ligne suivante ou de sauter à une autre ligne.
Complémentation : Traitement qui consiste à changer tous les 0 d'un octet en 1, et tous les 1 en 0.
Complémentation à deux : Système de notation utilisé pour représenter les nombres négatifs. Pour trouver le complément à deux d'un nombre, on commence par complémenter ce nombre (voir plus haut), puis on ajoute 1.
Compteur ordinal : Registre (appelé aussi compteur d'instructions) qui contient l'adresse de la prochaine instruction à aller chercher en mémoire.
Déplacement : Nombre utilisé avec les instructions de saut ou de branchement pour indiquer à l'ordinateur combien d'emplacements mémoire il doit sauter.
Désassembleur : Programme qui effectue l'opération inverse de l'assembleur : à partir du code machine, il recrée un programme source en mnémoniques. Il permet, entre autres, d'examiner les programmes stockés dans la ROM de votre ordinateur.
Hexadécimal : Système numérique qui utilise seize chiffres (nombre 0 à 9 et lettres A à F). Chaque chiffre vaut seize fois le chiffre placé à sa droite.
HIMEM : Adresse la plus grande de la mémoire utilisateur.
Indicateur : Bit du registre d'état qui sert de drapeau (« flag ») pour indiquer qu'une condition (retenue, zéro...) est remplie.
Instruction : Opération qui doit être exécutée par l'Unité Centrale de Traitement.
Interpréteur : Programme qui traduit en code

machine les instructions écrites en BASIC (ou dans un autre langage évolué).

Jeu d'instructions : Ensemble des instructions qui peuvent être exécutées par un microprocesseur donné.

Langage d'assemblage : Méthode de programmation qui utilise des lettres codées, appelées mnémoniques, pour représenter les instructions du code machine.

LIFO : Méthode utilisée par l'ordinateur pour stocker les informations dans une pile : la dernière information stockée est la première à ressortir de la pile (« last in/first out »).

Mémoire d'écran (ou mémoire d'affichage) : Emplacements de la RAM utilisés pour stocker les informations qui doivent être affichées.

Mémoire tampon : Mémoire qui sert à stocker provisoirement des informations en transit, par exemple entre l'ordinateur et l'imprimante.

Mémoire utilisateur : Partie de la RAM où sont stockés les programmes BASIC de l'utilisateur.

Microprocesseur : Cette puce renferme l'UCT, qui exécute les instructions du programme et contrôle toutes les opérations effectuées par l'ordinateur.

Mnémoniques : Lettres codées utilisées en langage d'assemblage pour représenter une instruction dans le code de l'ordinateur. Ces lettres évoquent généralement le mot anglais correspondant à l'instruction.

Modes d'adressage : Procédés utilisés pour indiquer à l'ordinateur où il trouvera les données à traiter.

Octet : Ensemble de 8 bits, qui représente un élément d'information en code machine.

Octet de poids faible : Dans une adresse, ce sont les deux chiffres hexa qui donnent la position d'un emplacement mémoire dans une page. Ce terme désigne aussi les deux chiffres hexa de droite d'un nombre hexa dont la valeur est supérieure à 255 en décimal.

Octet de poids fort : Dans une adresse, ce sont les deux chiffres hexa qui donnent le numéro de la page où est stocké un emplacement mémoire. Ce terme désigne aussi les deux chiffres hexa de gauche d'un nombre hexa dont la valeur est supérieure à 255 en décimal.

Opcode : Partie d'une instruction en langage d'assemblage, qui indique à l'ordinateur ce qu'il doit faire.

Opérande : Partie d'une instruction qui indique à l'ordinateur où il trouvera la donnée à traiter.

Page : Subdivision de la mémoire. Sur la plupart des micro-ordinateurs familiaux, une page comporte 256 emplacements mémoire.

Page zéro : Première page de la mémoire.

Pile : Zone de la mémoire utilisée par l'UCT pour stocker provisoirement des informations ; la dernière information stockée doit être rappelée la première (voir LIFO).

Pointeur : Emplacement mémoire (ou paire de registres) où est stockée l'adresse d'une donnée.

Pointeur de pile : Registre de l'UCT qui contient l'adresse du dernier élément stocké sur la pile.

Programme objet : Programme qui a été traduit en code machine et dont la source était écrite en langage évolué ou en langage d'assemblage.

Programme source : Programme écrit en langage évolué (BASIC) ou en langage d'assemblage, qui devra être interprété avant d'être exécuté.

RAMTOP : Équivalent de HIMEM.

Registres : Emplacements mémoire intégrés à l'UCT où sont stockées les instructions, les données et les adresses que traite l'ordinateur.

Registre d'état : Registre dans lequel chaque bit est utilisé séparément pour indiquer si une condition est remplie (voir indicateur).

Registres d'index : Registres utilisés pour l'adressage indexé ; sur le 6502, ils peuvent aussi être utilisés comme des registres ordinaires.

Saut : Instruction qui ordonne à l'ordinateur de sauter à une autre ligne du programme au lieu de passer à la suivante.

Système d'exploitation : Ensemble de programmes écrits en langage machine et stockés en ROM, qui indiquent à l'ordinateur ce qu'il doit faire pour exécuter les ordres donnés par l'utilisateur.

UAL : L'Unité Arithmétique et Logique est la partie de l'UCT qui traite les opérations arithmétiques et logiques.

Variables système : Emplacements mémoire de la RAM qui enregistrent les informations concernant l'utilisation du micro-ordinateur.

Solutions (suite)

page 28

00011010 = 26 en décimal.
11111011 = 251 en décimal.
10101010 = 170 en décimal.

	Décimal		Hexa	
	Poids fort	Poids faible	Poids fort	Poids faible
307	1	51	&01	&33
21214	82	222	&82	&DE
759	2	247	&02	&F7
1023	3	255	&03	&FF

page 31

Pour adapter le programme de la page 27 à des résultats supérieurs à 255, il faut supprimer l'instruction RET et ajouter les lignes ci-dessous. Pour afficher le résultat, utilisez la commande suivante : PRINT PEEK (adresse 3) + PEEK (adresse 4) *256.

Z80		6502		Commentaires
Mnémoniques	**Codes hexa**	**Mnémoniques**	**Codes hexa**	
LD A,&00	3E,00	LDA #&00	A9 00	Met 0 dans l'accumulateur.
ADC A,&00	CE,00	ADC # &00	69 00	Ajoute la retenue et 0 à l'accumulateur.
LD (adresse 4),A	32,adresse 4	STA adresse 4	8D adresse 4	Stocke le contenu de l'accumulateur à l'adresse 4.
RET	C9	RTS	60	Retour.

Page 37

En hexa, le complément à deux de 12 est &F4, celui de 18 est &EE et celui de 9 est &F7.

Index du guide de l'assembleur

Modifications pour le chargeur hexa

Modifications pour le ZX81 :

```
40 INPUT H$
70 LET X=
   (CODE(H$)-28)*16
80 Delete
90 LET Y=CODE
   (H$(2 TO ))-28
100 LET X=X+Y
110 Delete
155 Delete
160 Delete
```

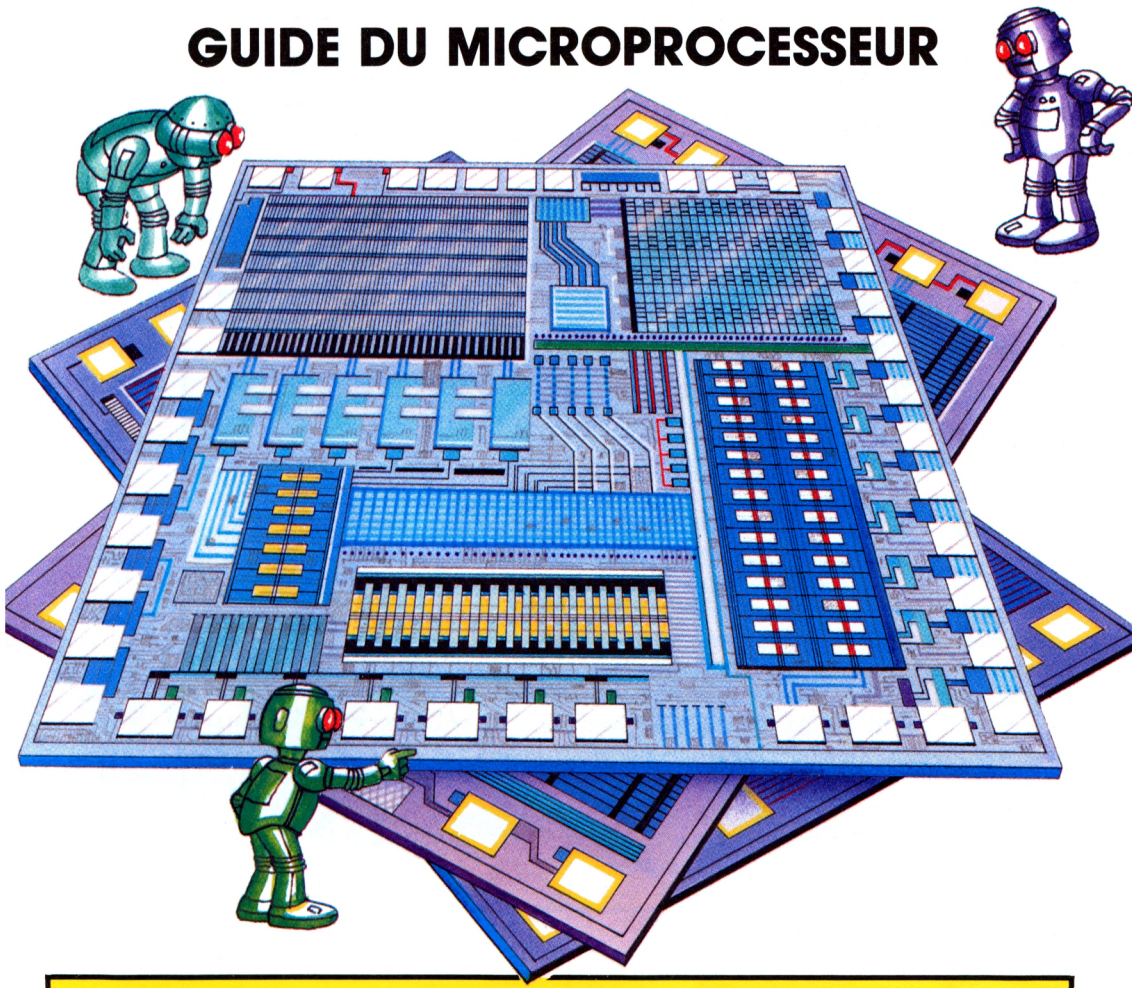
Modification pour Atari :

```
90 LET Y=ASC(A$(2))
```

GUIDE DE L'ASSEMBLEUR ET DU MICROPROCESSEUR

DEUXIÈME PARTIE

GUIDE DU MICROPROCESSEUR



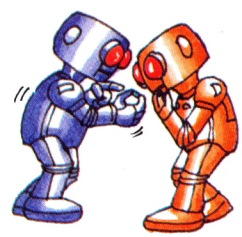
par Helen Davies et Mike Wharton

Avec la participation de Peter Dartnell de STC Semiconductors. Conception graphique : Graham Round, Iain Ashman et Roger Priddy. Illustrations de Graham Round, Mark Longworth, Graham Smith, Martin Newton, Jeremy Banks, Jeremy Gower, Chris Lyon et Jim Bamber. Édition française : Patrick Baradeau. Adaptation, assistance technique et conseil : European Media Business, 9, Place des Ternes, 75017 Paris.

ÉCHOS-ÉLECTRONIQUE

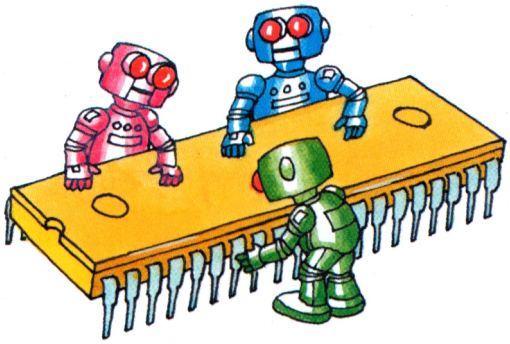
HACHETTE

Guide du microprocesseur

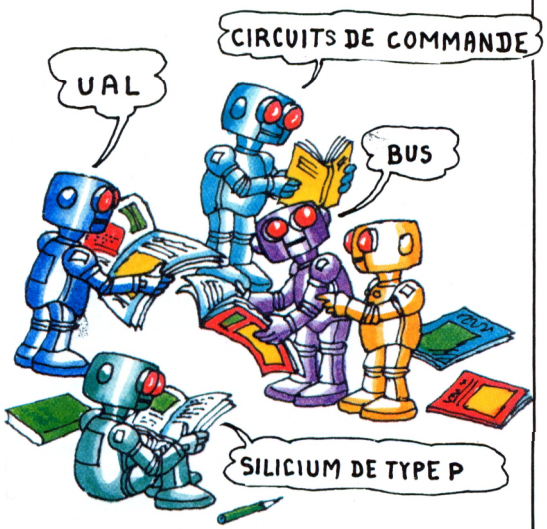


Avant-propos

Le circuit intégré, plus communément appelé « puce » de silicium, est probablement la plus grande invention de notre siècle. Mais cette petite merveille, qui semble capable de tout faire, vous paraît peut-être encore bien mystérieuse...



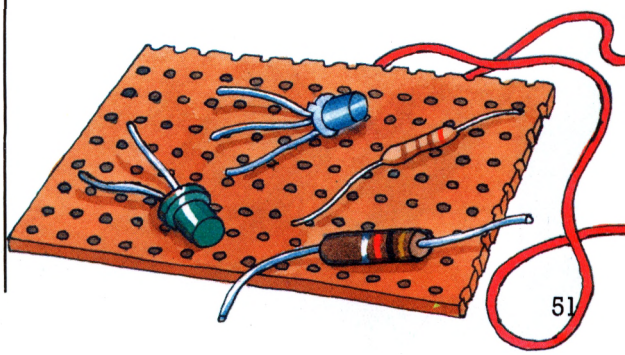
Alors, partez à la découverte de la puce ! Ce petit livre va vous révéler son fonctionnement et vous dévoiler quelques-unes de ces fabuleuses applications. Vous découvrirez les différents types de puces et leurs applications, et vous vous familiariserez avec tous les termes techniques.



A l'abri dans son petit boîtier, la puce a envahi tous les domaines de la vie quotidienne : micro-ordinateurs, téléviseurs, calculatrices, voitures, montres, téléphones, magnétoscopes, instruments de musique, équipement hospitalier, feux de signalisation, robots, usines, etc. Nous verrons comment une puce commande tous ces appareils, et vous pourrez même construire un circuit électronique pour concrétiser vos connaissances.

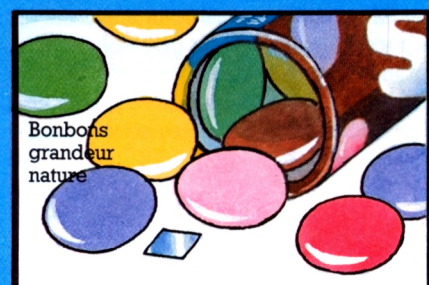
Intérieur d'un clavier d'ordinateur

Depuis l'invention de la puce en 1958, on n'a cessé d'accroître les capacités et les champs d'applications de la puce. Tout nouveau robot, tout nouvel ordinateur plus puissant est le fruit de puces plus performantes. Actuellement, on met au point des puces qui pourront reconnaître et reproduire les sons infiniment complexes de la langue humaine. Pourtant, toutes ces inventions ne sont que l'application des quelques principes de base présentés dans ce livre. Vous apprendrez également quelles sont les découvertes scientifiques qui ont été à l'origine de la puce.



Présentation de la puce

La puce est une minuscule pastille de silicium, plus petite que l'ongle du petit doigt. Elles est bourrée de microscopiques circuits électriques, parcourus par d'infimes courants, qui lui permettent de faire fonctionner des calculatrices, des ordinateurs, des robots ou des engins spatiaux. L'utilisation des courants électriques pour faire fonctionner des appareils s'appelle l'électronique ; en raison de la petitesse de ses circuits et de ses courants, la puce appartient au domaine de la micro-électronique.

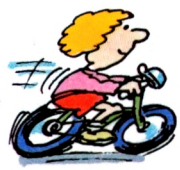


On voit ici la taille véritable d'une puce : un carré de 5 mm^2. Les bonbons donnent l'échelle.

Comment la puce commande

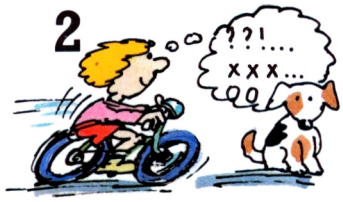
A l'aide de signaux très nombreux, transmis par des courants électriques, la puce envoie et reçoit des messages, calcule, compare des informations et prend des décisions simples : c'est un véritable cerveau électronique qui traite les informations (en anglais, « process », d'où le terme de microprocesseur). Notre cerveau aussi traite des informations en permanence :

1



Pour faire marcher une moto, le cerveau envoie des signaux à différentes parties du corps.

2

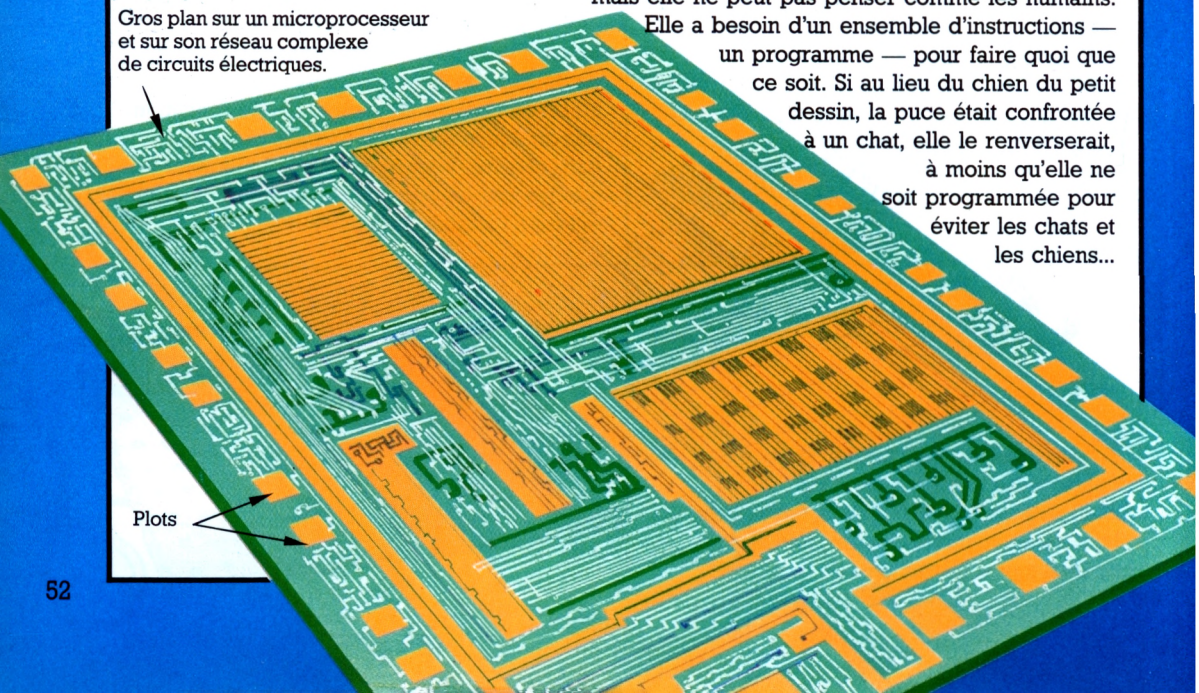


Face à un obstacle, les yeux envoient un message au cerveau, qui doit immédiatement traiter cette information.

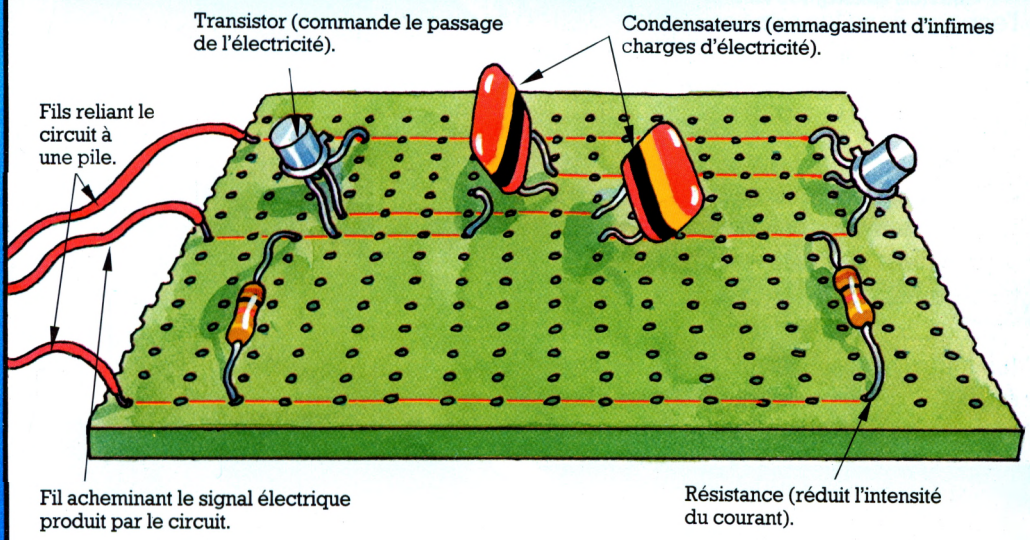
3

Après une étape d'analyse, le cerveau donne d'autres ordres au corps pour faire face à la nouvelle situation.

Comme notre cerveau, une puce peut prendre des décisions ; mais elle ne peut pas penser comme les humains. Elle a besoin d'un ensemble d'instructions — un programme — pour faire quoi que ce soit. Si au lieu du chien du petit dessin, la puce était confrontée à un chat, elle le renverserait, à moins qu'elle ne soit programmée pour éviter les chats et les chiens...



Qu'est-ce que l'électronique ?



L'électronique, c'est la technique qui consiste à commander et à manipuler des petits courants électriques. Pour faire varier l'intensité du courant, on utilise différents petits éléments, les composants, qui sont reliés par des fils pour constituer un circuit.

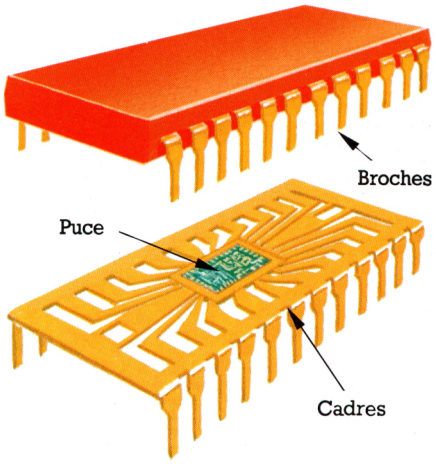
Le schéma ci-dessus représente un circuit comportant trois types de composants : des transistors, des résistances et des condensateurs. Ces éléments sont reliés par des pistes de cuivre fixées sur la face inférieure du circuit électrique. Les traits rouges montrent le chemin suivi par le courant électrique le long des pistes.

Une puce micro-électronique contient plusieurs centaines de circuits rassemblés sur une minuscule pastille de silicium. Leurs composants, également en silicium, sont reliés par d'invisibles pistes en aluminium gravées sur la surface de la puce. La puce, dont le nom scientifique est « circuit intégré », comprend un nombre croissant de circuits, grâce aux progrès techniques qui permettent d'intégrer de plus en plus de composants dans un espace de plus en plus réduit.

Conditionnement d'une puce

Les puces sont logées dans des boîtiers en plastique dotés de petites épingles, les broches. Ces broches en cuivre sont recouvertes d'or ou d'étain, pour mieux conduire l'électricité. Elles permettent le va-et-vient des signaux électriques de la puce et acheminent le courant qui la fait fonctionner.

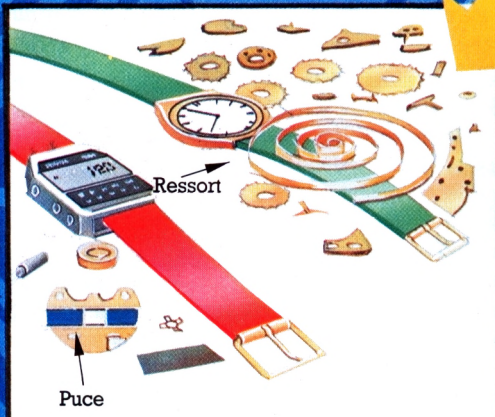
Si l'on ôtait le couvercle du boîtier, on verrait la puce au milieu d'une structure métallique la reliant aux broches. On appelle cette grille de métal le cadre, ou l'araignée. Des fils d'or plus fins que des cheveux transportent les signaux électriques de l'araignée à la puce. Ils sont soudés sur le pourtour de la puce, en différents points appelés plots. (Voir page 52.)



L'avènement de la puce

La puce a été mise au point pour la conquête de l'espace dans les années 1950 et 1960. Très vite, les producteurs ont exploité la petite taille de la puce et sa faible consommation pour fabriquer des calculatrices et des micro-ordinateurs. Parallèlement, les puces remplacèrent les anciens mécanismes qui actionnaient montres et appareils photos, et les circuits électriques volumineux des téléviseurs, radios et téléphones.

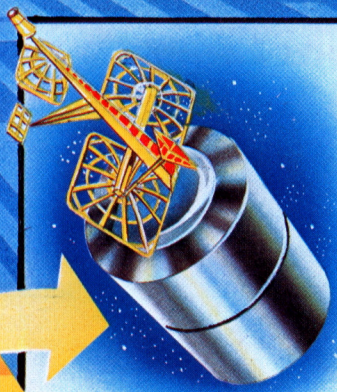
Une puce peut commander presque toutes les machines, à condition que les signaux électroniques puissent être traduits de façon compréhensible par la machine et vice versa.



Ces deux montres illustrent la différence entre les sytèmes mécanique et micro-électronique. La montre à ressort, constituée d'éléments plus nombreux, est moins performante. Pour la mettre en marche, il faut remonter un ressort ; en se détendant, celui-ci fait tourner les roues dentées qui, à leur tour, font tourner les aiguilles. Dans une montre électronique, la puce « bat la mesure » et donne l'heure en allumant des chiffres. Elle peut indiquer la date et le jour de la semaine, actionner une sonnerie, un minuteur et même une calculatrice incorporée à la montre.

La montre électronique est actionnée par de petites piles. Ce système, qui ne comporte aucune partie mobile pouvant se casser ou s'user, est beaucoup plus fiable que les rouages d'une montre à ressort.

Dans les ordinateurs, les calculatrices et les jeux vidéo, un clavier ou une manette de jeux traduit les ordres de l'utilisateur en signaux électriques, traités par une ou plusieurs puces. Pour afficher leurs réponses, les puces envoient des signaux électriques qui commandent l'affichage de nombres, mots ou images sur l'écran.



Les puces sont indispensables dans un engin spatial où des calculs complexes doivent être effectués en quelques secondes pour déclencher de nouvelles manœuvres et maintenir la trajectoire du satellite. Un homme serait incapable d'effectuer de pareils calculs avec une telle rapidité et une telle précision.

Robot de peinture au pistolet

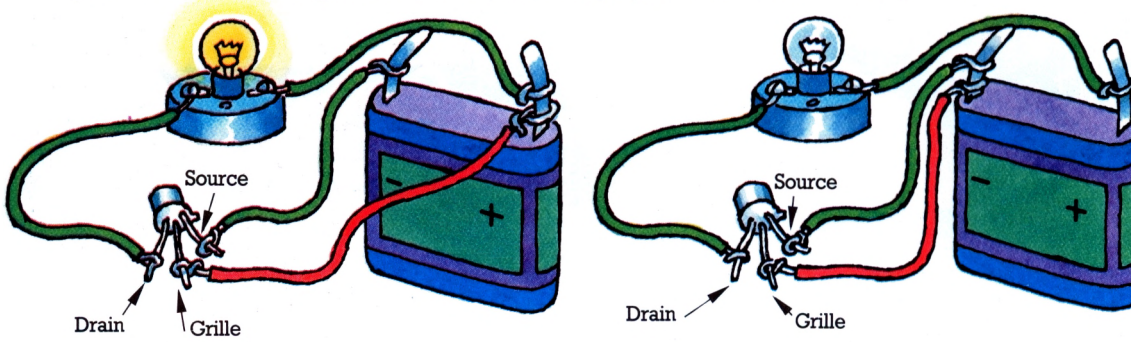
Un robot industriel est actionné par un grand nombre de moteurs, chacun actionnant une partie du robot. En commandant leur fonctionnement, la puce peut faire exécuter au robot des tâches très complexes.

Les puces commandent toutes sortes d'appareils ménagers, de jeux, de jouets. Ainsi, dans une machine à laver, la puce stocke les programmes de lavage sous forme de chaînes de signaux électriques. Dès que l'on met la machine en route, la puce verrouille la porte et sélectionne le programme demandé. Ensuite, grâce à ses signaux électriques, elle ouvre les conduites d'eau, actionne le chauffe-eau, commande la rotation du tambour...

Dans ce taxiphone, une puce enregistre électroniquement la somme introduite, la durée de la communication et en calcule le coût. Le cas échéant, elle fait rendre la monnaie dans le distributeur. Elle avertira automatiquement les dépanneurs en cas de panne... et la police en cas d'attaque !

Pour comprendre l'électronique

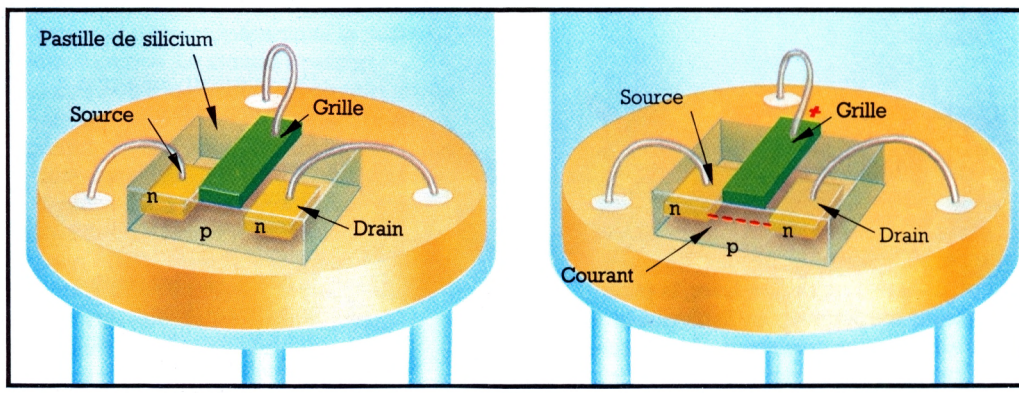
Dans les puces, les composants essentiels, ceux qui font le travail, sont les transistors qui commandent le passage et l'arrêt du courant. Ils ont le même rôle qu'un banal interrupteur, mais ils sont actionnés par une force électrique, la tension. On trouve des transistors partout, notamment dans les téléviseurs et les postes de radio. Le dessin ci-dessous montre comment on peut utiliser un transistor pour allumer et éteindre une lampe. Le transistor de ce circuit très simple joue le même rôle que ceux des microscopiques transistors d'une puce.



La lampe ne s'allume que si le courant de la pile peut circuler dans le circuit à travers le transistor. Les trois pattes de ce transistor s'appellent la source, la grille et le drain. Pour que le courant électrique traverse le transistor, la grille doit être sous tension : il faut donc la relier à la borne +, dont la tension est haute, comme sur le croquis de gauche. Si, comme sur le croquis de droite, elle est reliée à la borne négative, dont la tension est basse, la lampe ne s'allumera pas. Pour allumer ou éteindre la lampe, il suffit de faire varier la tension appliquée à la grille.

Fonctionnement des transistors

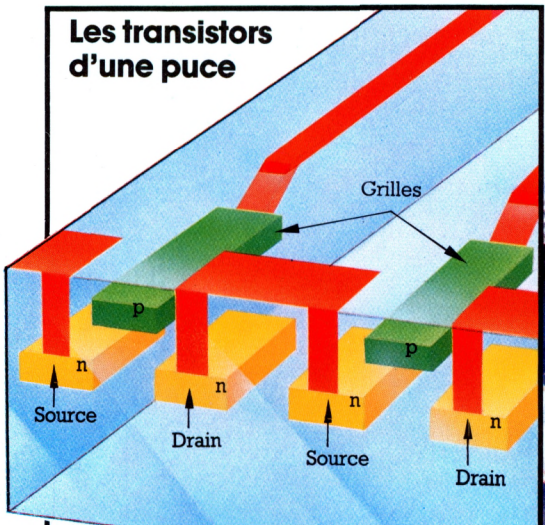
Ces dessins montrent le cœur du transistor représenté plus haut. Le transistor lui-même est une pastille de silicium, matériau chimique de la catégorie des semi-conducteurs qui ne conduisent l'électricité que dans certaines conditions. Pour fabriquer un transistor, on injecte dans le silicium des impuretés pour obtenir du silicium de type p et du silicium du type n, aux qualités différentes.



Les transistors ont généralement deux îlots de silicium du type n noyés dans un substrat du type p (ou inversement). Le silicium du type p empêche le courant de passer d'un îlot du type n à l'autre.

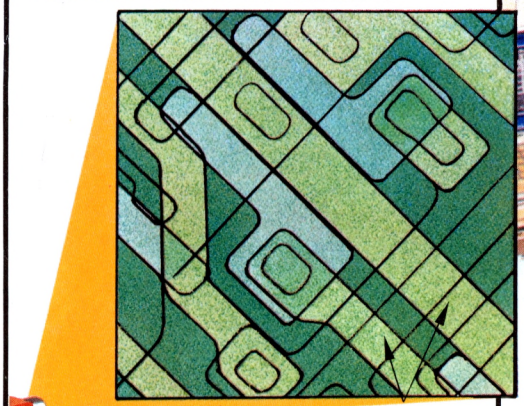
Quand on envoie une faible tension électrique à la grille métallique située au-dessus du silicium de type p, celui-ci se transforme en silicium de type n : le courant peut ainsi traverser le transistor.

Les transistors d'une puce



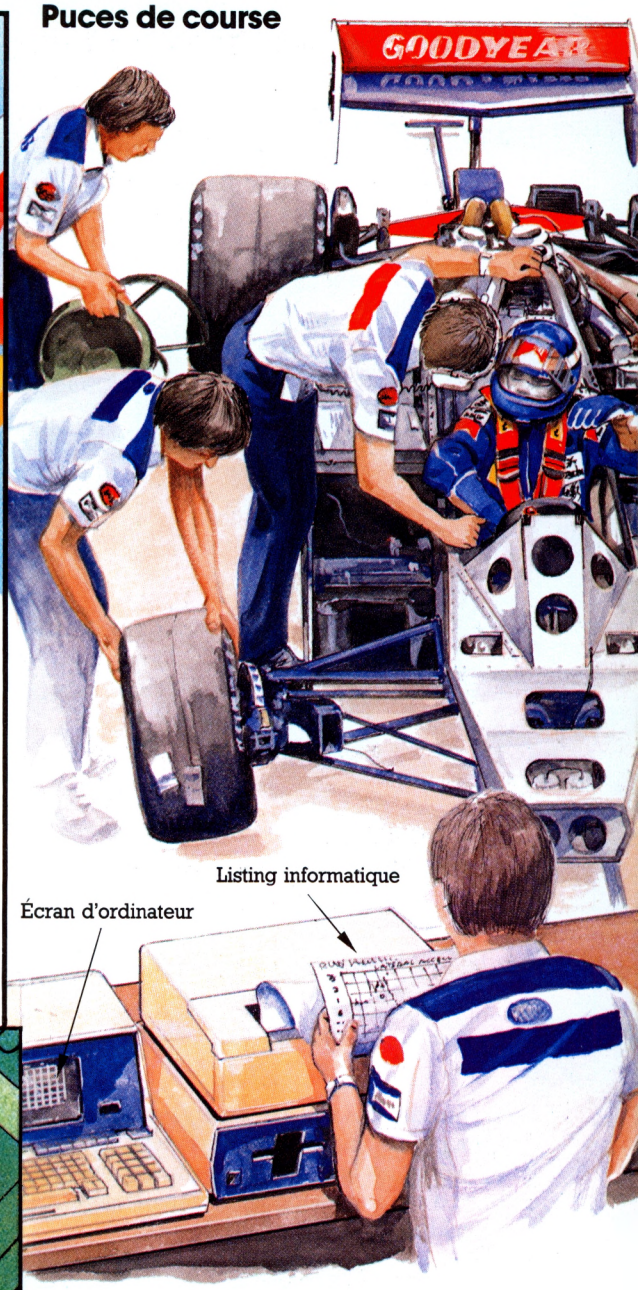
Les transistors d'une puce sont infiniment plus petits que ceux de la page de gauche, mais ils fonctionnent de la même façon. L'image ci-dessus représente deux transistors placés côte à côte à la surface d'une puce. Les connexions électriques des sources, drains et grilles, sont des pistes en aluminium, disposées au-dessus des transistors. Une couche de dioxyde de silicium sépare les pistes et les transistors pour assurer l'isolation électrique.

Ces transistors sont « fermés » (c'est-à-dire qu'ils laissent passer le courant) quand ils reçoivent une tension haute, et « ouverts » quand ils reçoivent une tension basse. Ces tensions émanent elles-mêmes d'autres transistors « fermés » ou « ouverts ».



Voici le détail d'une puce vue au microscope. Les parties vert pâle sont les pistes d'aluminium menant à la source, au drain et à la grille d'un même transistor. Elles sont grossies environ 2 000 fois.

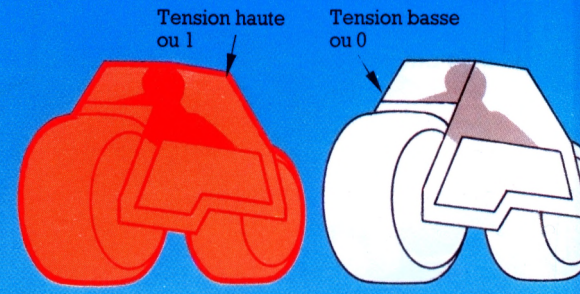
Puces de course



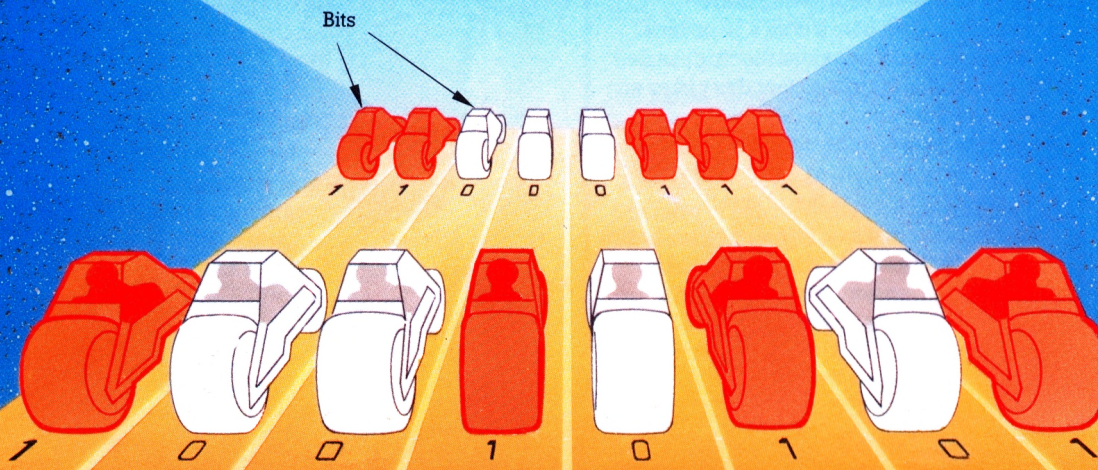
La suspension de cette voiture de formule 1 est commandée par des puces. Placées dans la voiture, celles-ci ajustent constamment la suspension en fonction du terrain et du déplacement des poids. L'ordinateur du stand de ravitaillement, relié à une petite antenne placée derrière la tête du conducteur, suit constamment la voiture. On peut ainsi étudier les réactions du système de suspension aux variations de vitesse et aux conditions de la route, par exemple dans les tournants.

Comment fonctionnent les puces

En utilisant les transistors d'une puce comme des interrupteurs, on crée des flux de courants électriques, qui servent de codes pour représenter des nombres, des lettres et toute autre information. Nous aussi, nous utilisons constamment des codes pour traduire des informations et résoudre des problèmes : langues, systèmes numériques, solfège sont autant de codes différents. La puce n'a qu'un seul code qu'elle utilise pour tout représenter : nombres, mots, images, mouvements.



Le code de la puce ne comporte que deux signaux : tension haute ou tension basse. Un tel code à deux éléments s'appelle un code binaire et s'écrit avec deux chiffres : le 1 correspond à la tension haute, le 0 à la tension basse. Ces 0 et ces 1 s'appellent des bits (abréviation de l'anglais « Binary digITs » : nombres binaires).

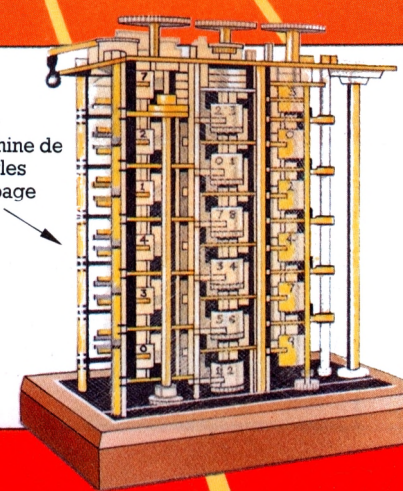


Dans une puce, les bits parcourent les circuits par groupes de huit bits, appelés octets. Chaque octet est le code d'un nombre, d'une lettre ou d'une autre information (température d'un liquide, position d'un bras de robot...)

Pour se déplacer dans la puce sans se mélanger, les bits suivent des pistes distinctes en aluminium. Un ensemble de huit pistes parallèles s'appelle un bus.

Pourquoi utiliser le système binaire ?

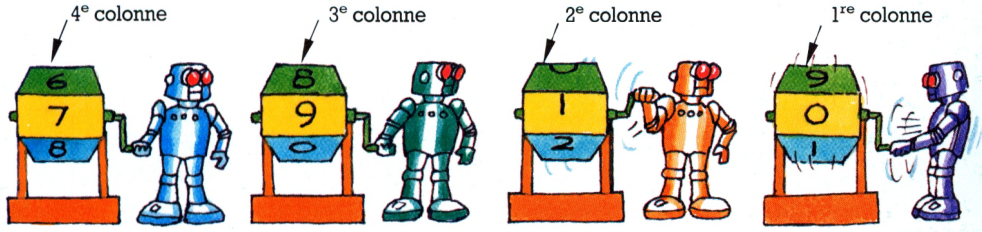
Quand les chercheurs ont essayé d'inventer des machines à calculer, ils ont d'abord utilisé le système décimal ; mais les machines étaient trop complexes pour bien fonctionner. En 1821, le mathématicien anglais Charles Babbage, a utilisé des roues à dix dents (une dent pour chaque chiffre de 0 à 9) pour représenter les colonnes d'unités, de dizaines, de centaines et de milliers. Les liaisons entre toutes ces roues étaient si compliquées que la machine n'a jamais fonctionné correctement.



Fonctionnement du code binaire

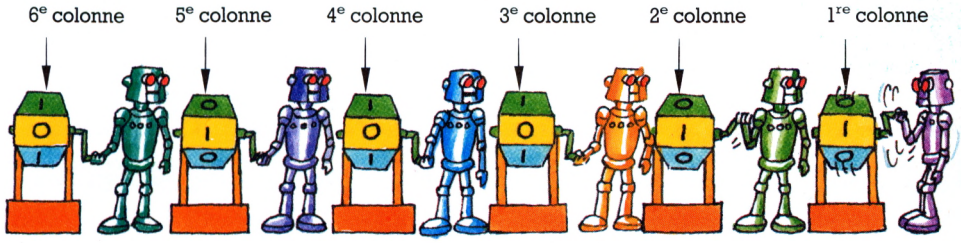
Le système binaire fonctionne comme les autres codes numériques, le système décimal par exemple. Ce dernier ne comportant que 10 chiffres (0123456789), pour écrire les nombres supérieurs à 9, on associe plusieurs chiffres selon des règles précises. Le système binaire utilise exactement les mêmes règles pour associer les 1 et les 0, comme vous l'explique cette page.

Décimal



Si l'on compte en décimal, à partir du nombre 9 il faut combiner deux chiffres. Le principe consiste à ajouter un chiffre à gauche, 1, et à recommencer à partir de 0 pour le chiffre de droite. Chaque fois que le chiffre de droite doit être supérieur à 9, on rajoute 1 au chiffre de gauche et l'on remet celui de droite à 0. Après 99, on pose un troisième chiffre, un nouveau 1, on remet à 0 les deux chiffres de droite et ainsi de suite...

Binaire



Le code binaire obéit aux mêmes règles. Mais, comme on ne dispose que de deux chiffres, on ne peut dépasser 1. On compte d'abord 0, 1, comme en décimal ; 2 devient « 10 » et 3 « 11 ». pour 4, on place un troisième chiffre à gauche : 4 s'écrit « 100 », 5 « 101 », 6 « 110 » et ainsi de suite. Avec un peu d'habitude, on compte et on calcule très facilement en binaire *.

Depuis plusieurs siècles, les mathématiciens utilisaient le système binaire. Mais ce n'est qu'en 1936 qu'un Allemand du nom de Zuse eut l'idée de l'exploiter pour une machine à calculer. Les nombres binaires sont trop longs et trop fastidieux à manipuler pour les hommes, mais ils sont parfaits pour les machines, car ils ne comportent que deux chiffres différents au lieu de dix.

Zuse était encore étudiant à l'université de Berlin lorsqu'il construisit sa première machine à calculer, la Z1, qu'il installa dans un coin du salon de ses parents. Il augmenta tellement la capacité de la machine que celle-ci occupa bientôt toute la pièce !

La Z1 utilisait des interrupteurs ordinaires pour traduire les 0 et les 1 binaires, et des ampoules électriques pour afficher les résultats des calculs. Ces interrupteurs étaient les ancêtres des transistors de la puce.

* Pour en savoir plus, voir page 88.

Concevoir une puce

La puce est constituée de plusieurs centaines de circuits microscopiques reliés les uns aux autres. Si ces circuits étaient réalisés à l'aide des composants électroniques traditionnels, ils rempliraient un gymnase...
Le travail du concepteur consiste à élaborer chaque circuit séparément, puis à relier tous ces circuits pour les faire tenir en un minimum d'espace. A l'origine, ce travail était fait à la main. De nos jours, les ordinateurs ont en mémoire de nombreux circuits, testés et éprouvés, qui peuvent servir de point de départ aux concepteurs.

Affichage d'un circuit

Clavier d'ordinateur

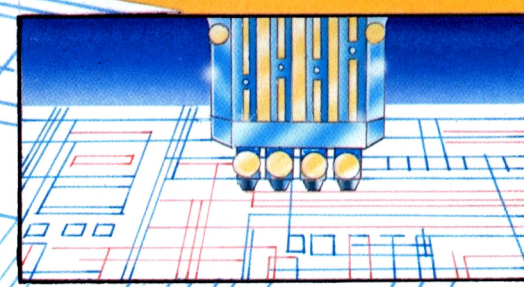
Tablette et crayon

▲ On voit ici un plan de circuit représenté sur un écran d'ordinateur et grossi environ 1 000 fois. Le concepteur donne des instructions à l'ordinateur en tapant sur le clavier, ou en dessinant à l'aide d'une tablette et d'un crayon spéciaux, reliés à l'ordinateur. Il peut ainsi étendre ou modifier le circuit, sauvegarder ce nouveau projet dans la mémoire de l'ordinateur, faire apparaître un autre plan sur l'écran...

Lorsque tous les circuits sont au point, le concepteur fait appel à l'ordinateur pour trouver le meilleur moyen de les relier. En effet, la rapidité d'exécution de la puce dépend de la distance que doivent parcourir les signaux électriques, dont la vitesse se mesure en nanosecondes (1 nanoseconde = 1 milliardième de seconde). Le concepteur vise donc à rapprocher au maximum les circuits pour accroître la vitesse d'exécution de la puce. Ce gain de place permet aussi d'augmenter le nombre des composants.

Carte de contrôle

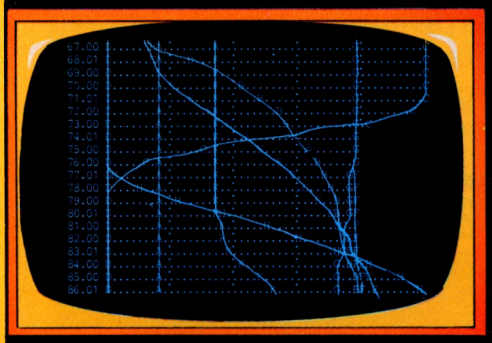
Appareil de dessin commandé par ordinateur, et appelé *table traçante.*

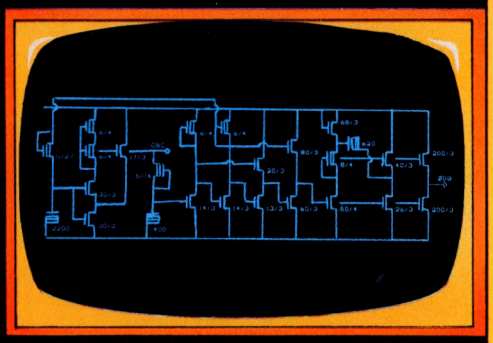


Une fois le projet terminé, l'ordinateur détermine la position précise de chaque composant et de chaque connexion électrique. On fabrique les composants en superposant plusieurs couches de matériaux chimiques différents. Le plan de chaque couche est dessiné par une table traçante très précise reliée à l'ordinateur (voir ci-dessus). Ces plans, qui sont environ 400 fois plus grands que la puce, permettent de vérifier que le dessin de chaque couche correspond exactement au modèle.

Tester le projet

Produire une puce unique coûte si cher que l'on ne peut pas fabriquer de prototype. On teste donc le dessin final en effectuant des simulations informatiques. Ce système permet de reproduire une situation dans les moindres détails : on peut ainsi mesurer les effets de telle ou telle action simulée par l'ordinateur.



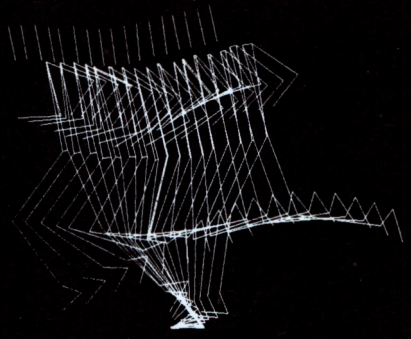


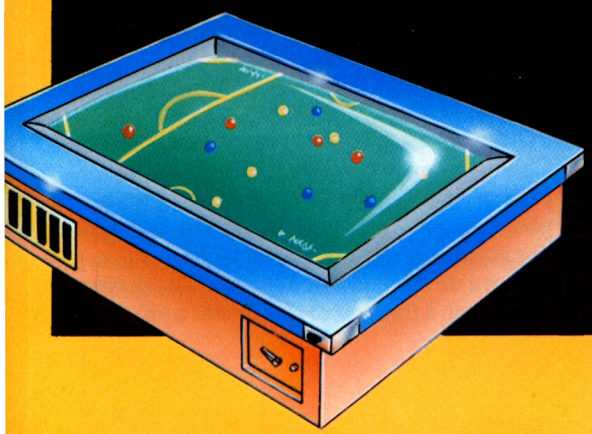
Un ordinateur peut simuler le passage du courant dans les circuits d'une puce : on voit ainsi comment fonctionnent les transistors, et quels chemins empruntent les signaux électriques. En fait, lorsque l'ordinateur teste le dessin, il ne montre pas le parcours suivi par les signaux électriques, mais il représente par des courbes les variations de tension des circuits testés.

Il est très difficile de tester la vitesse d'exécution de la nouvelle puce, car elle est presque toujours plus rapide que les puces qui font fonctionner l'ordinateur de simulation... L'ordinateur ne pouvant mesurer la vitesse de travail de la nouvelle puce, on doit faire de savants calculs pour l'estimer.

D'autres utilisations de la simulation

▼ Beaucoup de jeux informatiques simulent des situations réelles. Ainsi, dans un jeu de billard, l'ordinateur représente à l'écran des boules de billard. En appuyant sur des boutons, les joueurs peuvent « frapper » une boule à l'endroit et avec la force voulus. L'ordinateur reproduit alors le déplacement (direction, distance) que suivrait une vraie boule.

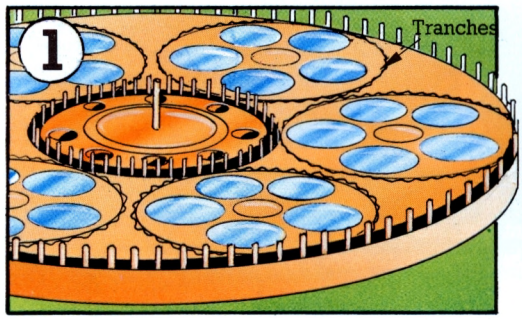




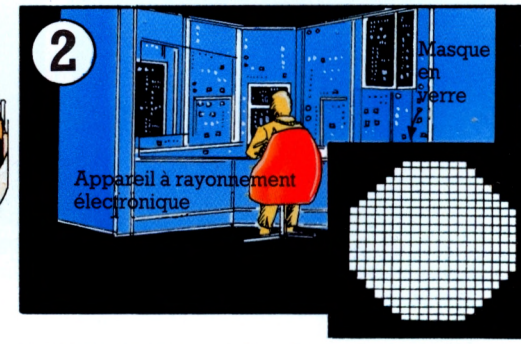
▲ La simulation par ordinateur peut servir à analyser et à augmenter les performances d'un athlète. On filme les mouvements de l'athlète, que l'ordinateur convertit en données visualisées à l'écran, par une série de droites recréant les mouvements précis du corps. On peut modifier les gestes et en mesurer les conséquences, qui sont visualisées à l'écran.

Fabriquer une puce

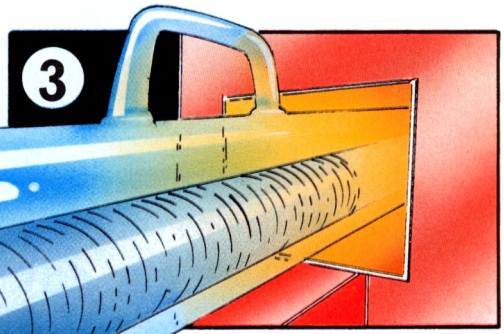
Seules des techniques de production d'une précision absolue permettent de faire tenir des centaines de circuits sur une puce de silicium de 5 mm^2. Les composants des circuits se mesurent en microns (1 micron = 1 millième de millimètre), et ne doivent pas être placés à plus d'un ou deux microns de l'endroit prévu. Les puces sont fabriquées dans des usines où l'on ne trouve pas de poussière, par des machines commandées par ordinateur. La fabrication est contrôlée par de puissants microscopes. Les composants et les connexions électriques des puces sont construits par couches successives de silicium (il peut y en avoir jusqu'à dix). Voyons les diverses étapes de la fabrication.



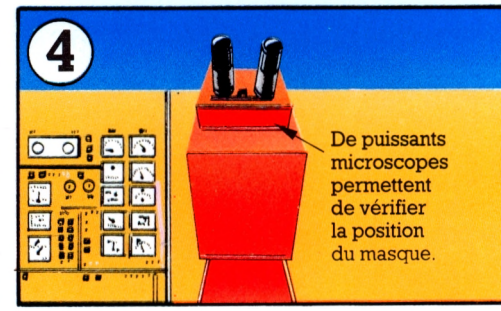
Le silicium est l'un des composants du sable. Pour fabriquer des puces, on fabrique sous vide des cristaux cylindriques de silicium pur que l'on découpe en tranches d'un demi-millimètre d'épaisseur. Ces tranches sont ensuite placées dans une machine qui polit parfaitement leur surface (voir croquis). Avec chaque tranche, on peut faire plusieurs centaines de puces.



A partir des dessins stockés dans la mémoire de l'ordinateur, on réalise des « photomasques », un pour chaque couche de la puce. Ce sont des carrés de verre sur lesquels est imprimé le dessin du circuit correspondant à une couche. L'impression se fait selon un procédé photographique ou selon une technique plus précise, appelée lithographie par rayonnement électronique. Sur chaque masque, dont la surface fait environ 10 cm^2, on peut juxtaposer les dessins de plusieurs centaines de puces.

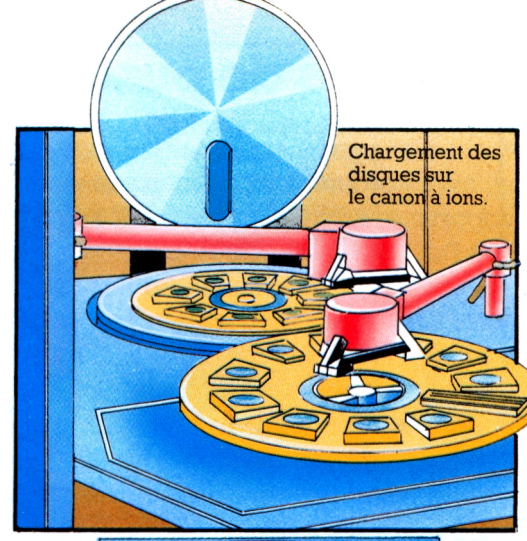


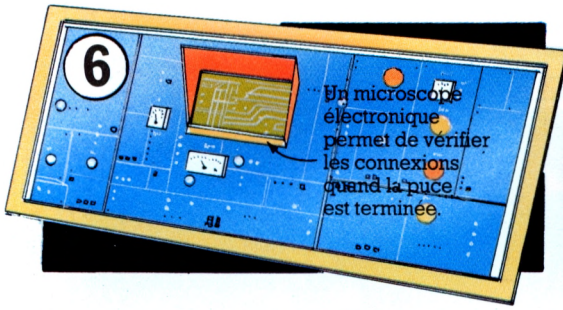
Les tranches sont soigneusement nettoyées, puis elles sont placées dans un fourneau d'oxydation chauffé au rouge pour être revêtues d'une fine couche d'isolant chimique, le dioxyde de silicium. Ensuite, on recouvre cet isolant d'un plastique mou et sensible à la lumière, le « photorésist ». (Cette étape et l'étape suivante sont répétées pour chaque couche.)



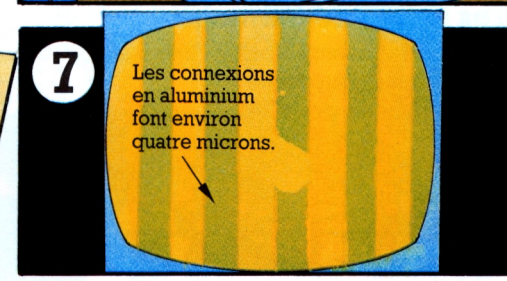
Pour transférer le dessin du circuit sur le silicium, on place le photomasque sur la tranche qui est ensuite exposée à une lumière ultra-violette. Ce procédé fait durcir le photorésist aux endroits non protégés par le photomasque. Ensuite, on utilise des acides et des solvants pour ôter le photorésist et le dioxyde de silicium placé dessous ; les parties de silicium à traiter sont ainsi mises à nu.

5 Les premières couches noyées dans le silicium sont des impuretés chimiques appelées dopants qui donnent deux éléments : le silicium de type n et le silicium de type p, utilisés pour les composants. Les dopants sont implantés par ionisation (les ions sont des particules chargées d'électricité). Les tranches sont introduites dans une machine qui projette des ions du dopant chimique : ces ions se déplacent à une vitesse fulgurante et frappent les tranches avec une force telle qu'ils s'incrustent dans les parties de silicium non protégées.

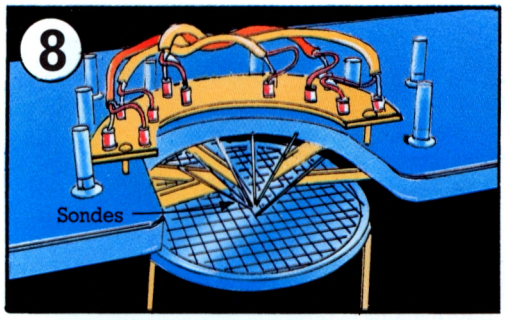




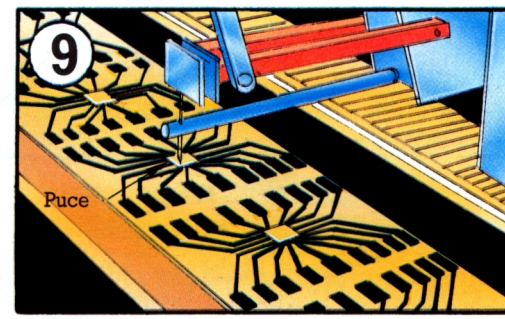
Les composants intégrés au silicium, on met en place des connexions électriques en aluminium. On peut superposer deux couches de connexions, à condition de les séparer par une couche isolante de dioxyde de silicium. Les connexions sont apposées selon un procédé d'évaporation, des masques permettant de définir les pistes.



Sur cet agrandissement d'un fragment de puce, on voit bien qu'une simple particule de poussière peut briser une connexion d'aluminium et détruire tout un circuit électronique. Il faut donc continuellement filtrer l'air des locaux de fabrication et le renouveler en permanence. De même, tous les employés doivent porter des combinaisons.



Une fois la fabrication terminée, des sondes électriques testent chaque puce. A cette étape, on en écarte déjà 70 pour cent qui sont défectueuses. Une scie à diamant ou à laser coupe alors la tranche pour isoler toutes les puces, et les puces défectueuses sont mises au rebut. Les puces sélectionnées sont ensuite conditionnées (étape n° 9).



Pour le conditionnement, des machines soudent des fils d'or sur les plots disposés sur les côtés, et les fixent sur les connecteurs métalliques du cadre. Ensuite, on recouvre la puce d'un boîtier en plastique et on met en place les broches. Les puces sont alors soumises à d'autres tests pour vérifier qu'elles fonctionneront parfaitement, même en plein air ou par un froid polaire...

Les types de puces

Voyons les principaux types de puces qui équipent un micro-ordinateur (voir ci-dessous). La plus importante est le microprocesseur (c'est souvent lui que l'on appelle « puce »). Il contient tous les circuits nécessaires pour commander un ordinateur ou tout autre appareil. Mais il ne peut fonctionner tout seul : il a besoin que d'autres circuits lui transmettent les instructions, que des mémoires stockent les informations pendant qu'il travaille, que des codeurs et décodeurs traduisent les signaux électriques en code binaire. Toutes ces fonctions sont assurées par d'autres puces.

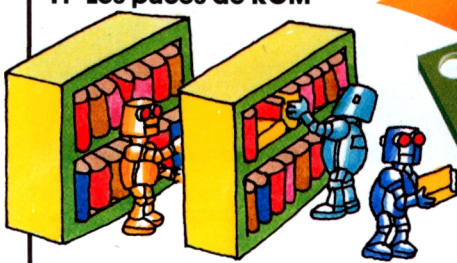
Le microprocesseur

Le microprocesseur est la puce qui effectue les calculs et les raisonnements logiques qui s'imposent pour le fonctionnement de la machine. Le microprocesseur d'un ordinateur s'appelle l'Unité Centrale de Traitement (UCT ou, en anglais, CPU).

Les puces de mémoire

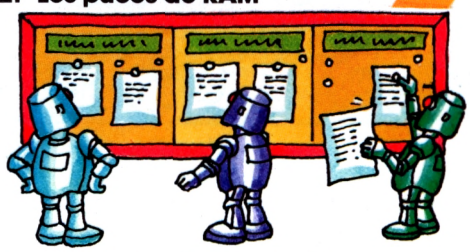
Ces puces servent à stocker l'information. On distingue deux grandes catégories :

1. Les puces de ROM



La mémoire morte (MEM), ou ROM (de l'anglais Read Only Memory) stocke en permanence des informations codées : au moment de la fabrication, les composants sont placés de façon à produire les mêmes signaux à chaque passage de l'électricité *. Les puces de ROM constituent en somme la bibliothèque des programmes qui indiquent au microprocesseur comment il doit travailler. Le microprocesseur peut seulement lire (« read ») des informations stockées dans une puce ROM, il ne peut rien y écrire.

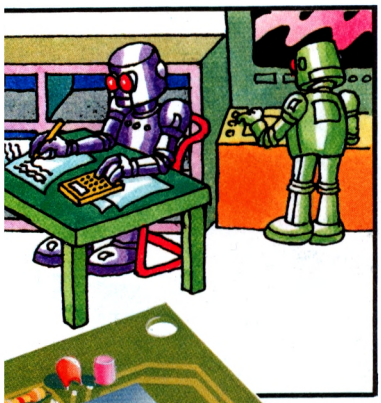
2. Les puces de RAM



La mémoire vive (MEV), appelée communément RAM (de l'anglais Random Access Memory), est une mémoire où l'on peut aussi bien « lire » qu'« écrire ». Ces puces RAM stockent temporairement des informations codées qui sont effacées quand elles ne sont plus utiles. A chaque fois qu'une information est stockée dans une RAM, les transistors de cette puce créent la chaîne de signaux correspondant à cette information.

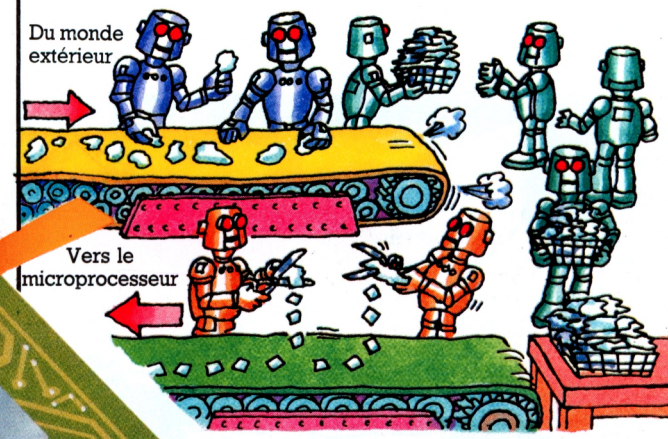
Les puces RAM ressemblent à un tableau d'affichage où le microprocesseur garde momentanément des informations qu'il utilise pour une opération spécifique.

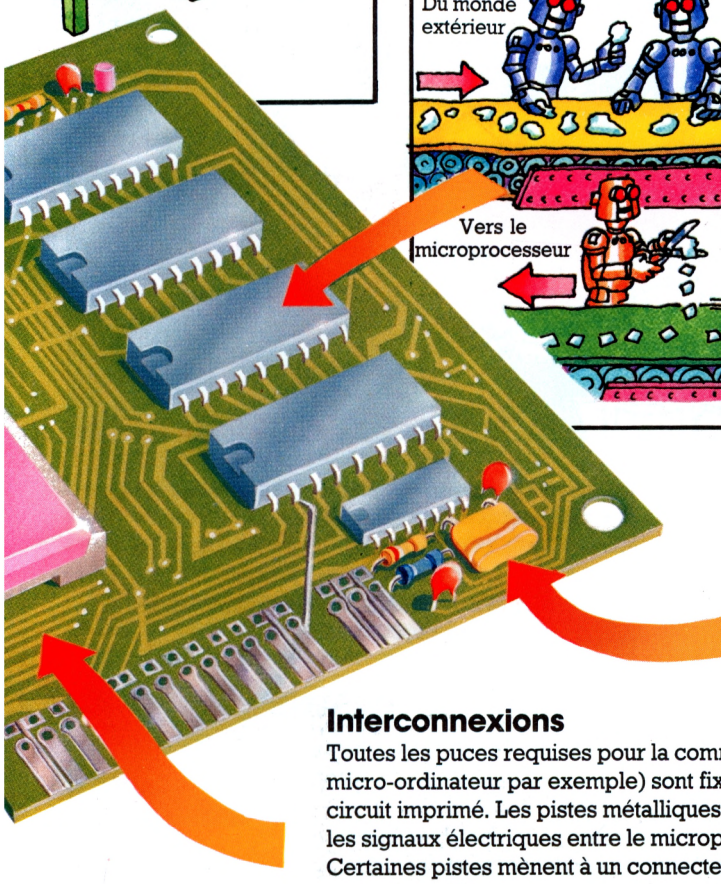
* Ces types de puce sont appelées *ROM préprogrammées.* Pour en savoir plus sur les ROM, reportez-vous à la page 95.



Les puces d'interface

Ces puces transforment les divers signaux électriques provenant du monde extérieur en 0 et 1, seuls signaux que sachent traiter le microprocesseur et les autres puces. Inversement, elles traduisent les codes binaires du microprocesseur en signaux compréhensibles par un téléviseur ou un appareil quelconque.





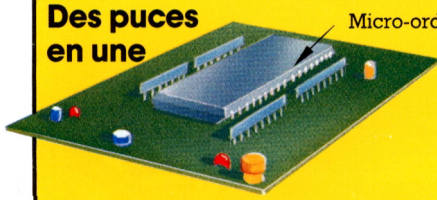
L'horloge

Bien qu'un microprocesseur effectue des milliers d'opérations par seconde, celles-ci sont faites l'une après l'autre. Pour que tout s'enchaîne correctement, une « horloge » en cristal de quartz bat la mesure...

Interconnexions

Toutes les puces requises pour la commande d'une machine (un micro-ordinateur par exemple) sont fixées sur une plaque spéciale, le circuit imprimé. Les pistes métalliques imprimées à sa surface véhiculent les signaux électriques entre le microprocesseur et les autres puces. Certaines pistes mènent à un connecteur, où l'on peut fixer des extensions diverses.

Des puces en une



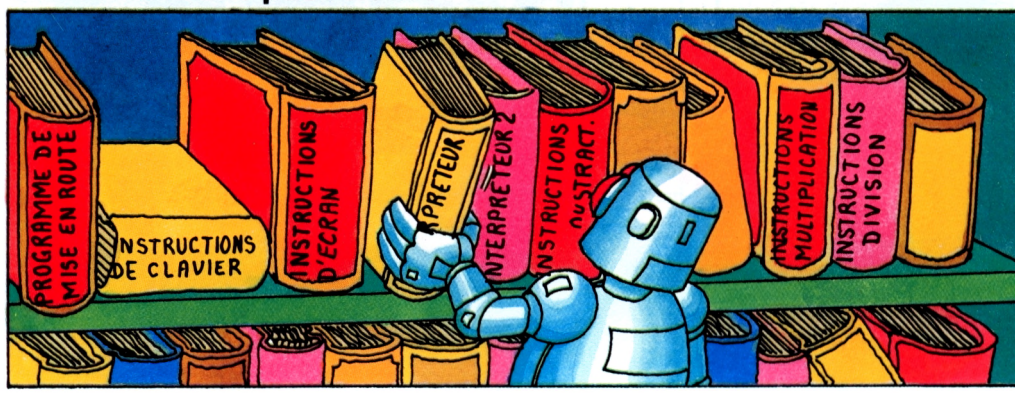
Les calculatrices, les jouets et les appareils ménagers (comme les machines à laver) sont généralement commandés par une puce unique qui comporte tous les circuits : commande, mémoire et interface. Ce type de puce est un micro-ordinateur ou microprocesseur spécialisé. Les circuits ROM, donc les instructions, sont intégrés à la puce ; aussi, le microprocesseur est-il seulement utilisable pour les tâches écrites dans les ROM.

Un microprocesseur ordinaire, comme celui du dessin au centre, n'est pas spécialisé. Ses instructions se trouvent dans une puce ROM séparée que l'on peut changer pour faire exécuter d'autres opérations au microprocesseur. Ainsi, le même microprocesseur pourrait servir à contrôler un micro-ordinateur, un jeu vidéo ou un robot si l'on changeait sa puce ROM.

Les puces de mémoire

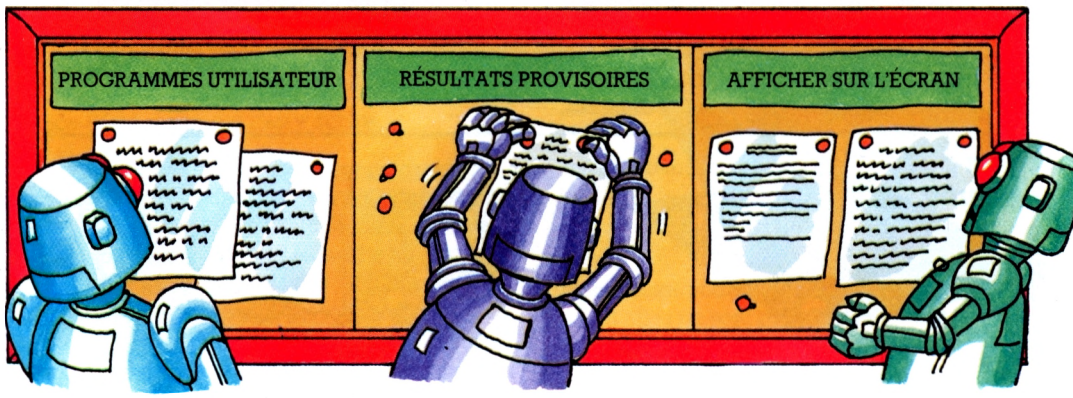
Elles servent à stocker des instructions et des informations codées électriquement. Malgré sa rapidité d'exécution, un microprocesseur ne sait effectuer qu'un nombre limité de tâches simples : par exemple, ajouter deux nombres ou comparer deux informations. Chaque tâche se présente sous la forme d'un code binaire à huit bits, appelé instruction de code machine. Les programmes, qui disent à l'ordinateur ce qu'il doit faire, sont des listes d'instructions stockées dans les puces de mémoire.

Que contient une puce ROM ?



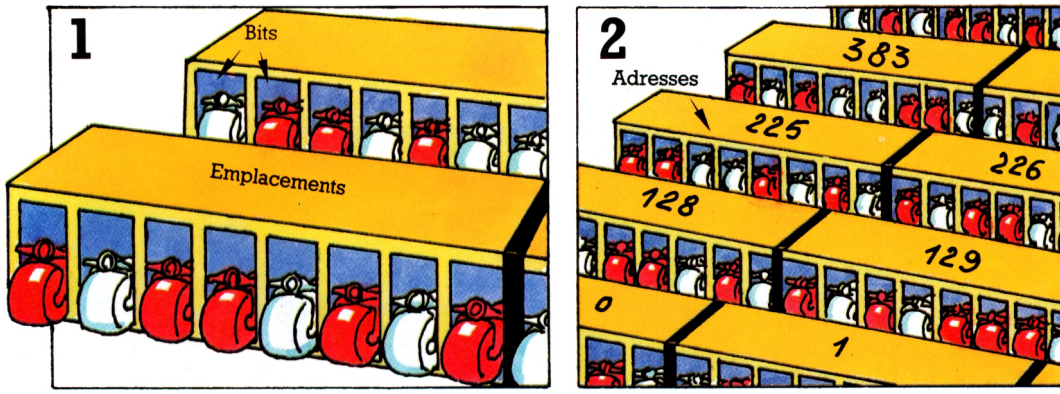
Certains programmes stockés en ROM dirigent le fonctionnement du micro-ordinateur. Ils constituent le Système d'Exploitation, ou Moniteur : ils indiquent au microprocesseur ce qu'il doit faire quand l'ordinateur est allumé, comment repérer qu'une touche du clavier est utilisée, où stocker les signaux électriques, comment faire apparaître des mots et des images, etc. La puce ROM contient aussi un programme, l'Interpréteur, qui traduit les instructions données par l'utilisateur en code machine, seul langage compris par le microprocesseur.

Que contient une puce RAM ?



Les puces RAM stockent temporairement les informations dont le microprocesseur a besoin pour une opération spécifique. Par exemple, il peut y placer le résultat d'un calcul qu'il utilisera ultérieurement dans un programme. C'est aussi dans les puces RAM que sont stockés les programmes donnés par l'utilisateur, et les résultats avant leur affichage à l'écran. Souvent, le microprocesseur va chercher un programme stocké en ROM et le copie dans les puces RAM avant de l'exécuter.

A l'intérieur d'une puce de mémoire

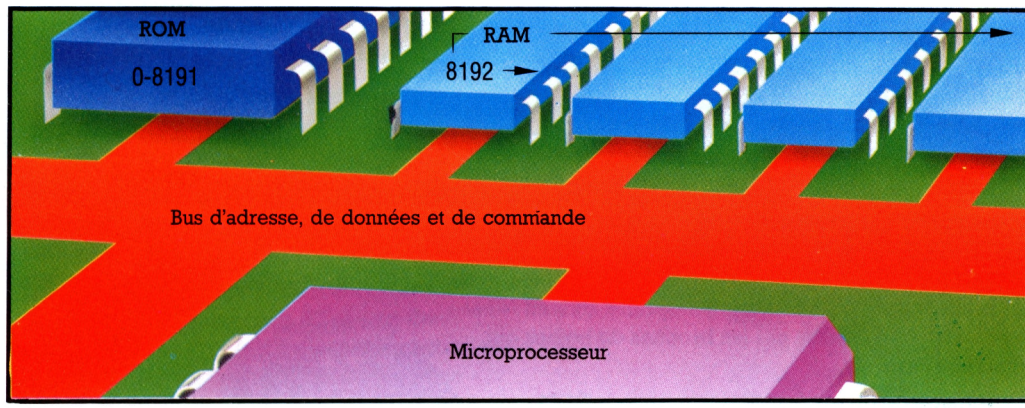


L'intérieur d'une puce ROM ou RAM ressemble à des rangées de boîtes. Chaque boîte peut contenir une partie d'une donnée. (Une donnée est une instruction ou une information en code machine.) Un groupe de huit boîtes, qui peut contenir un octet, s'appelle un emplacement mémoire. Chaque emplacement porte un numéro, l'adresse, qui permet au microprocesseur de retrouver rapidement l'information.

Coder les adresses

Un microprocesseur ne peut pas distinguer entre des puces ROM et des puces RAM : les adresses des différents emplacements ne doivent donc pas se recouper. (C'est comme si chaque maison d'une ville avait un numéro particulier au lieu de réutiliser les mêmes numéros avec des noms de rues différents...) La numérotation commence à 0 dans la ROM, et va en ordre croissant jusqu'aux puces de RAM.

On numérote les adresses en codes binaires à 16 bits, car on a besoin de beaucoup de places. Un code à huit bits ne donnerait que 256 adresses, alors que seize bits peuvent être combinés de 2^{16} façons différentes, ce qui donne 65 536 adresses.

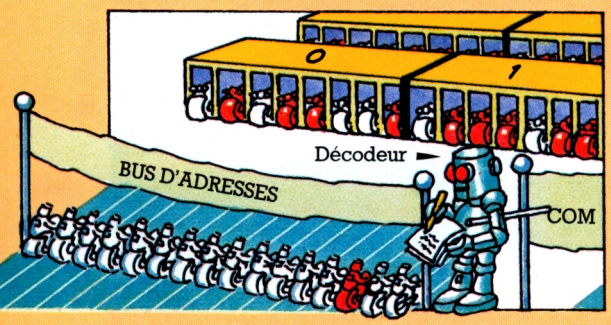


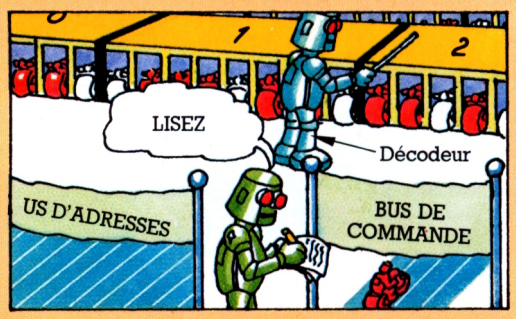
Relier les mémoires au microprocesseur

Les puces de mémoire sont reliées au microprocesseur par des réseaux de pistes conductrices, les bus. On compte trois bus : un pour les adresses, un pour les données, et un troisième pour les signaux de commande. Le bus d'adresse a seize pistes pour véhiculer les seize bits d'une adresse. Le bus de données a huit pistes, car les données, c'est-à-dire les instructions et les informations, sont codées en octets, c'est-à-dire sur huit bits. Sur le bus de commande circulent divers signaux, comme celui qui indique si c'est pour lire ou pour écrire une donnée que l'on va à un emplacement mémoire.

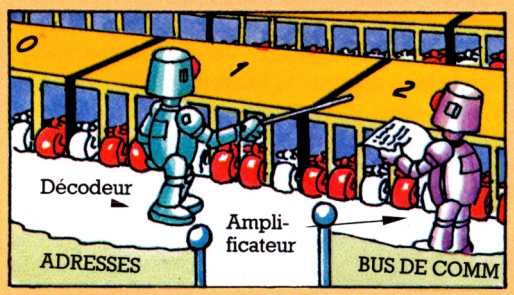
Fonctionnement des puces de mémoire

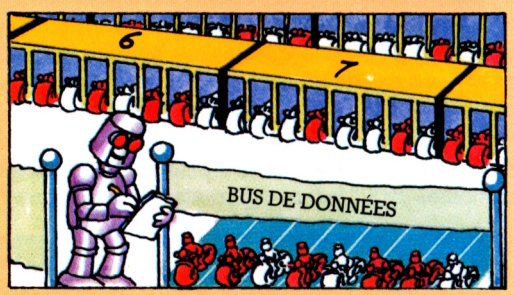
Ces dessins expliquent comment les informations et les instructions circulent entre les puces de mémoire et le microprocesseur.





Lorsque le microprocesseur a besoin d'une information stockée en mémoire, il envoie le code de l'adresse correspondante à la puce de mémoire où se trouve l'information, par l'intermédiaire du bus d'adresses. A l'arrivée du code, les circuits du décodeur de la puce lisent la chaîne de signaux et indiquent l'emplacement mémoire correspondant. Sur le bus de commande, un signal précise s'il faut, à cet emplacement, lire ou écrire une information. Dans les puces de ROM, seule la lecture est possible, alors qu'en RAM on peut lire ou écrire.



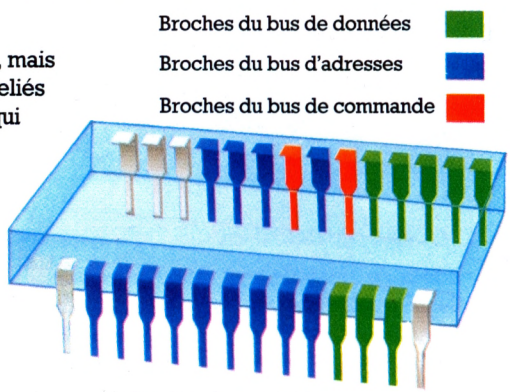


Si le signal de contrôle dit « Lire », les circuits appelés amplificateurs lisent le code stocké dans l'emplacement mémoire et en déposent une copie sur le bus de données. (Le code est simplement dupliqué : il n'est pas effacé de l'emplacement mémoire.)

Description des bus

Les bus ne sont pas des groupes de fils parallèles, mais des pistes tracées sur le circuit imprimé. Ils sont reliés directement à la puce par les broches du boîtier qui protège celle-ci.

L'image de droite représente une puce ROM de vingt-huit broches. Huit broches correspondent au bus de données à huit bits. Il n'y en a que treize pour le bus d'adresses car, une seule puce ne possédant pas 65 536 emplacements de mémoire, elle n'a pas besoin d'un bus d'adresses de seize bits. Cette puce a 8 192 emplacements mémoire : treize pistes lui suffisent pour les désigner ($2^{13} = 8\,192$).

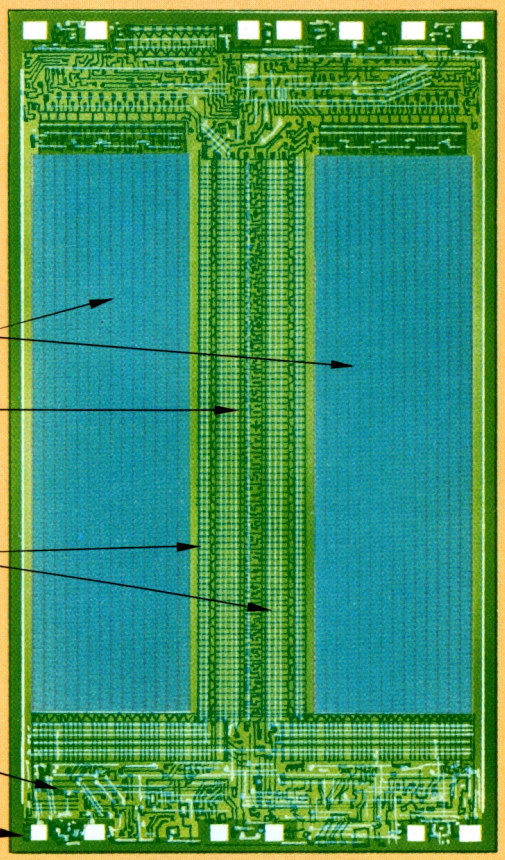


Les broches restantes servent à l'alimentation.

Comment se présente une puce de mémoire

Ce dessin représente une puce de mémoire considérablement agrandie. Les circuits sont répartis en deux blocs composés de milliers de circuits identiques, les « cellules », disposées en rangées et colonnes. Chaque cellule, souvent constituée de six composants électroniques, peut stocker un bit d'information.

Le bus de données est placé entre les deux blocs de mémoire pour que le trajet suivi par les signaux entre le bus et les cellules soit aussi bref que possible. Ceci conditionne en effet le « temps d'accès » de la puce, c'est-à-dire le temps que met une information pour aller de la mémoire au microprocesseur. Ce temps d'accès est généralement de 200 nanosecondes.



Blocs de circuits de mémoire

Amplificateurs : circuits qui lisent les données stockées dans les emplacements mémoire et les placent sur le bus de données.

Décodeurs de colonnes : circuits qui localisent la colonne (codifiée dans le second octet d'une adresse) où se trouve un emplacement mémoire.

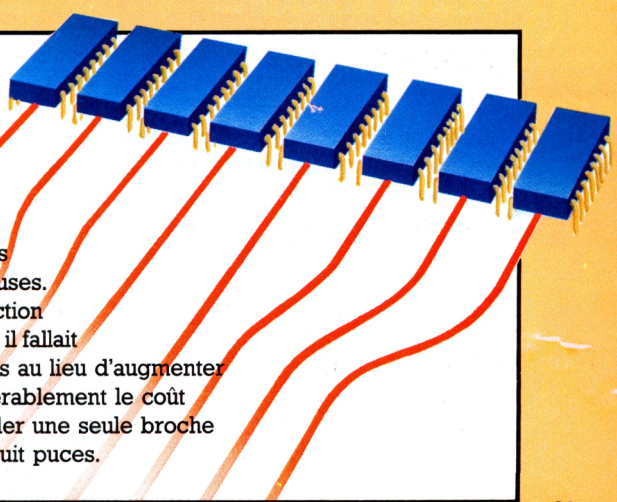
Décodeurs de rangées : circuits permettant de localiser la rangée (codifiée dans le premier octet d'une adresse à seize bits) où se trouve un emplacement mémoire.

Plots qui acheminent les signaux d'adresse, de données et de commande.

La taille des mémoires est souvent indiquée par le nombre de bits qu'elle peut contenir. Celle-ci accepte 16 384 bits (en deux blocs de 128 colonnes et 64 rangées). On dit qu'elle contient 16K (K * est l'abréviation de kilo qui correspond en informatique à 1 024).

Les puces RAM

La plupart des puces RAM n'ont qu'une broche pour les données. Elles sont donc disposées par groupes de huit et les octets de données se scindent en bits, chaque puce recevant ou envoyant un seul bit. En effet, à l'origine, les puces comportant beaucoup moins de cellules, les adresses étaient moins nombreuses. Toutefois, le progrès des techniques de production ayant accru le nombre de cellules de mémoire, il fallait davantage de broches pour les adresses. Mais au lieu d'augmenter le nombre de broches, ce qui accroît considérablement le coût de production, les fabricants ont préféré garder une seule broche pour les données et utiliser des groupes de huit puces.



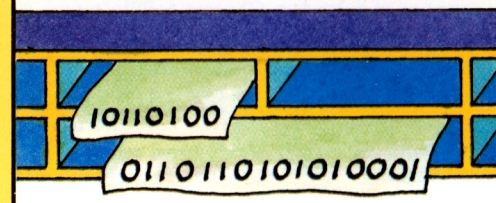
* La lettre K majuscule signifie kilo-octet. La taille des puces de mémoire est souvent exprimée ainsi. Par exemple, la puce de ROM de la page 68 fait 8K puisqu'elle stocke 8 192 (8 × 1 024) octets.

Le microprocesseur

Ces deux pages vous montrent les circuits d'un microprocesseur, c'est-à-dire d'une puce complexe capable de commander un micro-ordinateur, un robot, etc. Le microprocesseur « traite » les informations transmises par le micro-ordinateur ou le robot, et lui indique ce qu'il doit faire, grâce à des listes d'instructions (stockées dans les puces de mémoire) qui lui indiquent comment traiter les informations, que faire des résultats, quels signaux émettre...

Dans un microprocesseur, il y a trois types de circuits : les circuits de commande, les registres et les circuits de l'UAL (Unité Arithmétique et Logique), où sont effectuées les principales opérations.

Les registres



Les registres sont des emplacements mémoire incorporés au microprocesseur. Les registres qui comportent huit cases peuvent contenir un octet d'information ; d'autres en ont seize pour stocker deux octets. Les registres permettent de stocker des informations et des instructions dans le microprocesseur lorsque celui-ci les utilise, et de disposer des adresses qui doivent être envoyées vers les puces de mémoire. Certains registres, comme le compteur ordinal, ont des tâches très spécifiques, que nous verrons dans les pages suivantes.

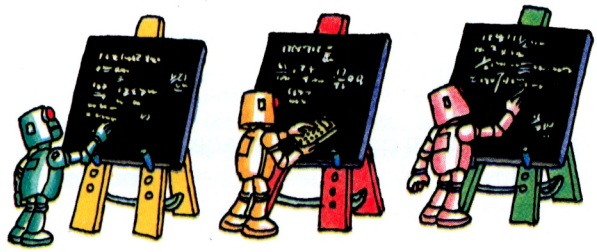
Les programmes

Toute opération effectuée par le microprocesseur peut se décomposer en une succession de tâches simples, chaque tâche correspondant à une instruction codée. A droite, on a transcrit en français, avec des chiffres décimaux, un exemple d'instructions destinées au microprocesseur.

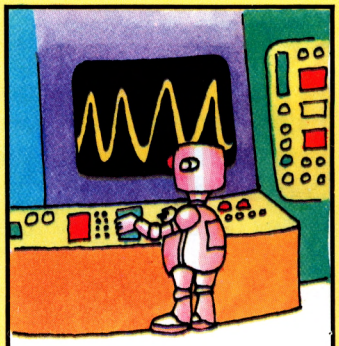
Tout programme est constitué d'une série d'instructions codées en système binaire. Les instructions sont généralement suivies d'une information ou de l'adresse où est stockée cette information. Ces renseignements, eux aussi codés en binaire, font également partie du programme. Les adresses occupent deux octets, car elles sont codées sur seize bits.

1er octet	CHARGER L'OCTET SUIVANT DANS LE REGISTRE A
2e octet	231 ← Ces deux octets sont des données
3e octet	CHARGER L'OCTET SUIVANT DANS LE REGISTRE B
4e octet	422 ← Ces octets sont des instructions
5e octet	AJOUTER LE REGISTRE B AU REGISTRE A
6e octet	STOCKER LE NOUVEAU CONTENU DU REGISTRE A A L'ADRESSE DES DEUX OCTETS SUIVANTS
7e octet	59 ← Ces deux octets forment une adresse
8e octet	75

L'unité Arithmétique et Logique

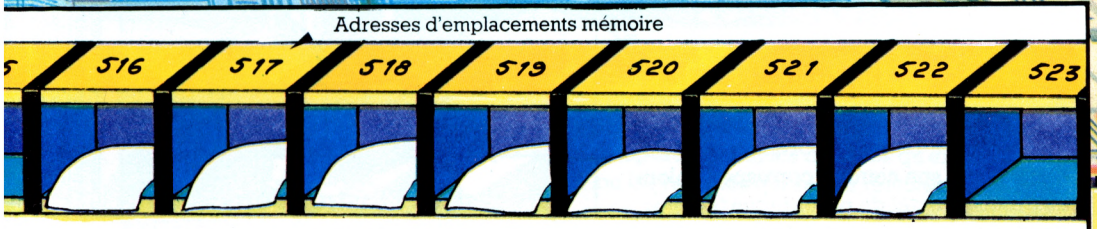


Cette partie du microprocesseur exécute les calculs et les raisonnements logiques nécessaires au fonctionnement de la machine. Pour ce faire, les codes binaires sont acheminés dans des circuits qui les modifient. En réalité, calculs et décisions ne résultent que d'opérations logiques très simples que nous étudierons pages 82 à 89.



Circuits de commande

Ils couvrent presque les deux tiers de la surface d'un microprocesseur. Ils organisent tout le travail du microprocesseur en émettant certains signaux pour ouvrir et fermer les registres, déclencher les opérations dans l'UAL, etc. Ces circuits de commande, qui sont expliqués dans les pages 26 à 29, sont reliés à l'horloge du microprocesseur.



Charger l'octet suivant dans le registre A. — 231

Charger l'octet suivant dans le registre B. — 422

Ajouter le registre B au registre A.

Stocker le nouveau contenu du registre A à l'adresse des deux octets suivants. — 59 — 75

Ce croquis montre comment le programme de gauche serait stocké dans une puce de mémoire. Chaque octet du programme est un code de huit signaux électriques placés dans un emplacement mémoire. Le microprocesseur traite chaque octet un par un en suivant l'ordre numérique. Il sait si tel octet est une instruction, une adresse ou une donnée, grâce à sa place dans le programme. Ainsi, le premier octet d'un programme ne peut être qu'une instruction. Si un programmeur se trompait et commençait par un octet de données, le microprocesseur prendrait celui-ci pour une instruction, et tout le programme se « planterait ».

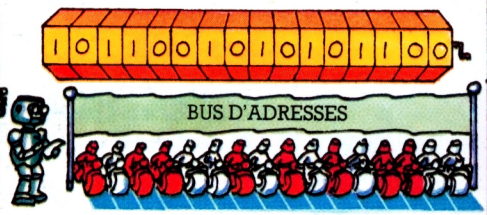
Les registres

Ils occupent environ le dixième de la surface d'une puce et servent à stocker momentanément les informations, les instructions et les adresses utilisées par le microprocesseur. Les registres qui ont une fonction spécifique sont décrits ci-dessous.

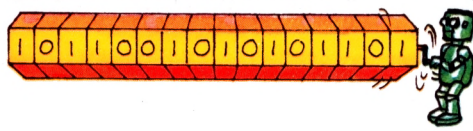
Les registres fonctionnent de la même façon que les emplacements mémoire d'une puce RAM : on peut y lire ou y écrire des données. Pour lire les données, on se contente de les copier ; elles ne sont effacées que lorsque l'on réécrit d'autres données par-dessus.

Le Compteur ordinal

C'est un registre à seize bits où sont stockées des adresses codées en binaire. Il sert à indiquer à quelle adresse il faudra aller chercher la donnée ou l'instruction suivante afin que le programme soit exécuté dans le bon ordre.



Au début du programme, on met dans le Compteur Ordinal l'adresse de la première instruction (c'est-à-dire du premier octet) qu'il place sur le bus d'adresses.



Tandis que l'on cherche et exécute cette instruction, le Compteur Ordinal augmente d'une unité : son nombre correspond alors au prochain octet du programme (comme pour un compteur de magnétophone). Le nombre contenu dans le compteur augmente ainsi automatiquement : on dit qu'il est « incrémenté ».

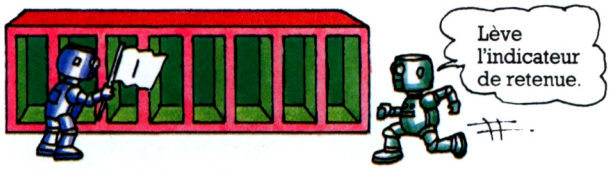
Le Compteur Ordinal désigne les adresses par ordre numérique croissant, sauf si on lui dit de sauter à une autre partie du programme. La nouvelle adresse est alors placée dans le Compteur Ordinal qui recommence à compter à partir de ce chiffre.

Registre d'instructions

Chaque instruction transmise au microprocesseur par les puces de mémoire parvient à ce registre de huit bits pour être exécutée par les circuits de commande. Une instruction reste dans ce registre jusqu'à ce que l'instruction suivante la remplace.

Le Registre d'État

L'UAL utilise séparément chacun des huit bits de ce registre pour indiquer si une condition est remplie ou non. Chaque bit, qu'on appelle « indicateur » (l'anglais, plus imagé, dit flag, « drapeau »), renseigne sur une condition particulière. Si le bit est 1 (tension haute), le drapeau est « levé » pour indiquer que la condition est remplie ; sinon, le bit est 0.



Par exemple, l'un des bits s'appelle « indicateur de retenue ». L'UAL l'utilise quand il fait des additions : si le résultat de l'addition est trop grand pour être exprimé avec huit bits, il ne peut tenir dans l'accumulateur. L'UAL indique alors au registre d'état de mettre l'indicateur de retenue à 1.

L'Accumulateur

Ce registre à huit bits est relié à l'UAL pour lui permettre d'effectuer les calculs. Ainsi, pour additionner deux nombres, il faut charger l'un d'eux dans l'accumulateur. Le résultat de l'addition est ensuite transféré dans l'accumulateur où il remplace le nombre de départ. C'est pourquoi l'on dit que ce registre « accumule » les résultats.

Registres de travail

Dans ces registres à huit bits, l'UAL stocke temporairement les données qu'elle utilise, ou les résultats qui seront réutilisés dans le programme. Un microprocesseur comporte généralement six à trente registres de travail, ce qui veut dire qu'il ne peut entreposer qu'un petit nombre de données, et que, le plus souvent, il doit aller les chercher dans les puces RAM chaque fois qu'il en a besoin.

Sur certains microprocesseurs, ces registres peuvent être couplés pour stocker les adresses des emplacements mémoire.

Les circuits de commande

On peut comparer les circuits de commande à un capitaine de vaisseau qui indique à chacun ce qu'il doit faire. Chaque instruction en langage machine qui arrive au Registre d'Instructions implique une série d'opérations : ouvrir ou fermer les registres, aller chercher une donnée en mémoire, etc. Les circuits de commande envoient les signaux adéquats, pour que chaque opération soit effectuée dans le bon ordre et au bon moment. Ces signaux sont eux-mêmes régulés par l'horloge du microprocesseur (voir pages 76 et 77).

Comment sont décodées les instructions

A l'intérieur des circuits de commande se trouve une liste de toutes les instructions comprises par le microprocesseur. Cette liste, appelée « Jeu d'Instructions », contient soixante-dix à cent cinquante instructions suivant le type du microprocesseur.

Dans les circuits de commande, un circuit de mémoire ROM, dit microprogrammé, contient l'ensemble des signaux correspondant aux opérations que doit effectuer le microprocesseur pour chaque instruction.

Quand une instruction en langage machine arrive dans le Registre d'Instructions, sa combinaison de 0 et de 1 correspond à l'adresse où sont stockés dans la mémoire ROM microprogrammée les signaux nécessaires à l'exécution de cette instruction.

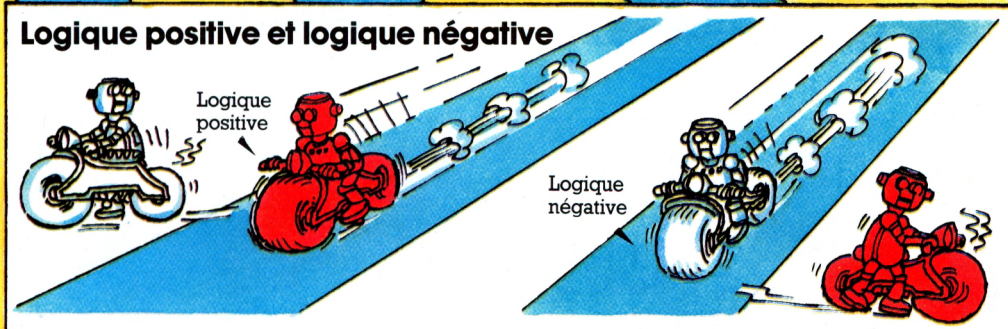
L'horloge

A intervalles réguliers, l'horloge du microprocesseur envoie des impulsions électriques aux circuits de commande pour coordonner toutes les tâches. Chaque fois qu'ils reçoivent une impulsion, les circuits émettent un signal.

Les signaux de commande

Les circuits de commande sont reliés aux autres circuits de la puce par soixante à cent pistes, les « lignes de commande ». Chacune d'entre elles correspond à une tâche spécifique. Un signal de commande est un simple signe binaire, 0 ou 1, qui provoque le lancement d'une opération *.

Logique positive et logique négative



Électroniquement parlant, un 1 correspond à une tension haute, et un 0 à une tension basse. Au repos, la tension d'une ligne de commande est en général basse ; quand on élève sa tension (signal 1), on envoie au circuit correspondant l'ordre de faire quelque chose. On appelle ce système la logique positive.

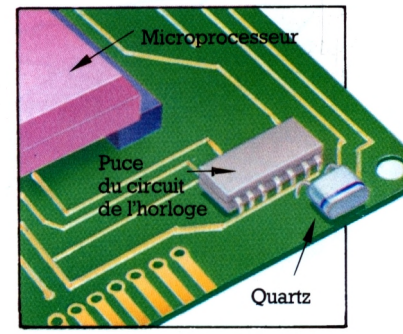
On utilise parfois le système inverse. Dans ce cas, la ligne a une tension haute au repos et, pour déclencher une opération, on fait passer la tension au niveau bas (signal 0). On est alors en logique négative. En fait, tous les signaux 0 et 1 du code machine correspondent à des tensions basses ou élevées.

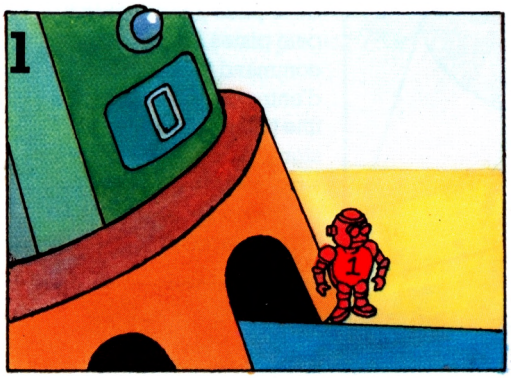
* Quelques-unes des lignes de commande constituent le bus de commande qui transmet des signaux aux puces autres que le microprocesseur.

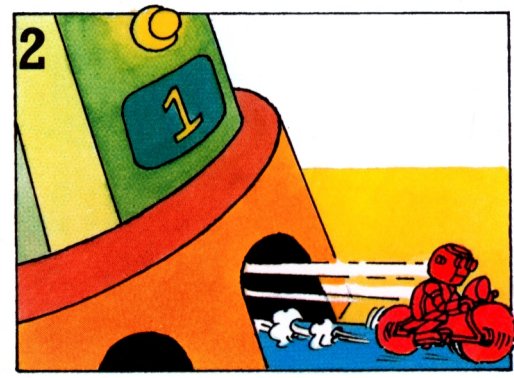
L'horloge du microprocesseur

Chaque opération effectuée par un circuit de la puce est provoquée par un signal de commande, lui-même déclenché par l'horloge du microprocesseur.

L'horloge est constituée d'un groupe de circuits (intégrés ou non au microprocesseur) reliés à un morceau de quartz, qui est toujours distinct du microprocesseur en raison de sa taille.



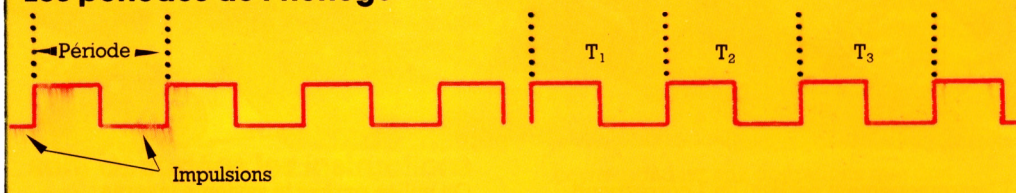




Un courant électrique qui traverse un cristal de quartz acquiert une parfaite régularité. Le circuit de l'horloge utilise ces vibrations électriques pour envoyer un flot régulier d'impulsions au circuit de commande. A chaque impulsion, celui-ci envoie un signal qui déclenche quelque chose dans une partie du microprocesseur. Aucune opération ne peut se faire sans ce signal.

La cadence de ces impulsions, mesurée en mégahertz, commande la vitesse de travail du microprocesseur. Un mégahertz correspond à 1 million d'impulsions par seconde. Les horloges des microprocesseurs tournent généralement à une vitesse de 2 à 4 mégahertz, ce qui signifie qu'un micro-ordinateur familial peut effectuer 4 millions d'opérations par seconde.

Les périodes de l'horloge

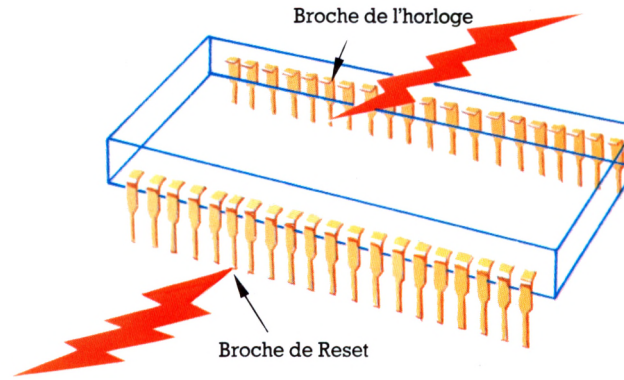


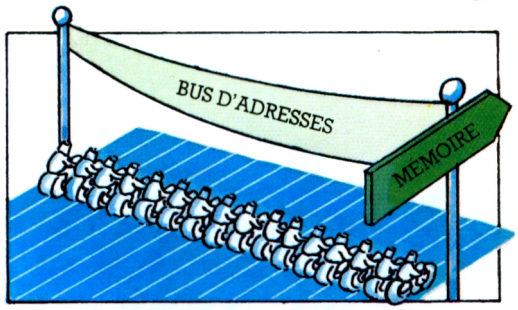
Voici une courbe représentant les impulsions d'une horloge avec ses deux niveaux de tension, bas et haut. L'impulsion est indiquée par le trait vertical correspondant à l'élévation de la tension. On appelle période l'intervalle de temps qui sépare deux impulsions.

Les ingénieurs et les programmeurs mesurent en périodes le temps mis par le microprocesseur pour exécuter une instruction. Le nombre de périodes d'une instruction correspond au nombre d'opérations électroniques élémentaires requises. Il varie de quatre à vingt suivant la complexité de l'instruction.

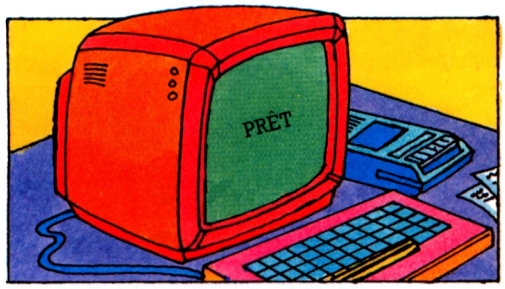
Mise en marche de la puce

Quand on met en marche un appareil équipé d'un microprocesseur, la première opération consiste à envoyer un signal électrique pour mettre le Compteur Ordinal à 0 : on appelle cela faire « Reset ». Ce signal lance aussi l'horloge qui commence à générer les signaux de commande.





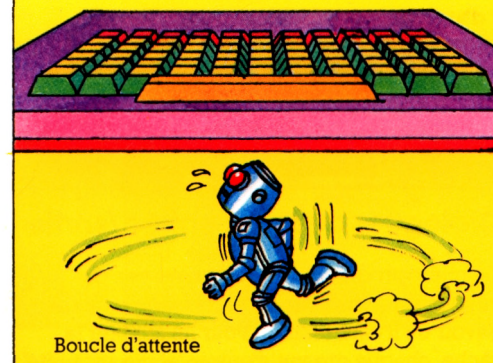
Les premiers signaux de commandes envoient sur le bus d'adresses le nombre contenu dans le Compteur Ordinal (0, soit en binaire 0000 0000 0000 0000) pour aller chercher le premier octet stocké à cette adresse dans le Registre d'Instructions. Celle-ci correspond toujours au premier emplacement mémoire de la puce de ROM.



Mettre en ordre de marche un ordinateur, c'est effacer les emplacements mémoire RAM, vérifier qu'ils enregistrent correctement les signaux électriques, contrôler les adresses d'entrée/sortie, enfin afficher sur l'écran le message indiquant à l'utilisateur qu'il peut commencer.

Pour tuer le temps

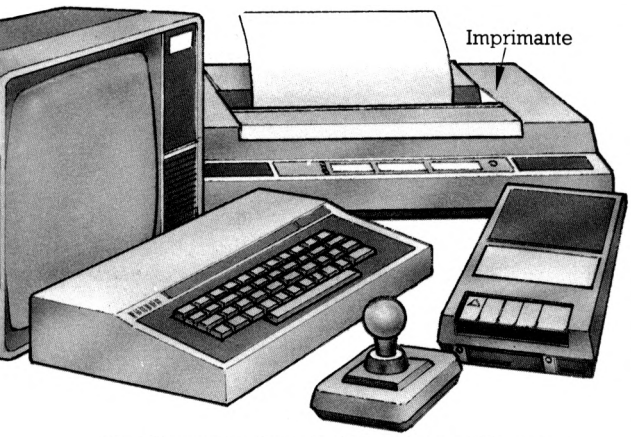
Tant que l'appareil est en marche, l'horloge envoie ses impulsions, si bien que le microprocesseur fait toujours quelque chose, même quand l'utilisateur ne lui demande rien. Des sous-programmes spéciaux (appelés « boucles d'attente ») intégrés aux programmes de la ROM font effectuer au microprocesseur des séries d'instructions jusqu'à ce qu'on lui demande de faire quelque chose d'utile.



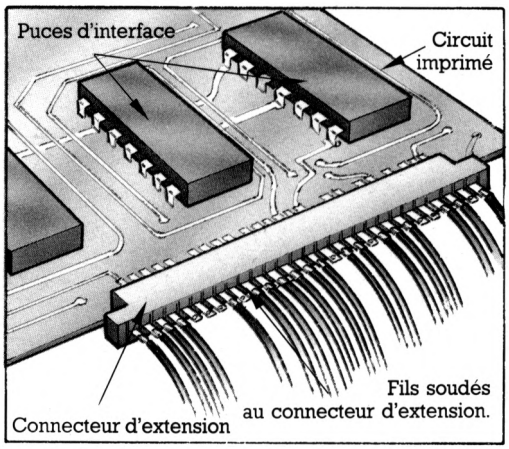
Sur un micro-ordinateur familial, l'une de ces boucles consiste à s'assurer qu'aucune touche du clavier n'a été enfoncée. En fait, le microprocesseur utilise énormément ce programme puisque l'on a calculé qu'il passe 98 pour cent de son temps à scruter le clavier !

Communiquer avec l'extérieur

Le microprocesseur n'est utile que s'il peut communiquer avec l'extérieur. Par exemple, celui qui commande un micro-ordinateur ou un jeu d'arcade doit recevoir des instructions de son utilisateur, les traiter et afficher des résultats. Le microprocesseur d'un bras de robot doit commander ses déplacements et recevoir des informations (appelées « feedback ») sur les manœuvres effectuées par le robot. Les éléments qui permettent ces échanges sont appelés entrée/sortie.



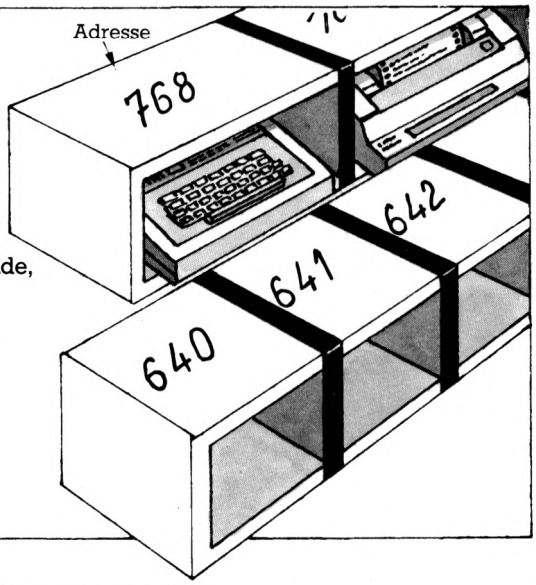
Les éléments d'entrée d'un micro-ordinateur sont le clavier, le lecteur de cassettes, les manettes de jeu..., et les éléments de sortie, l'écran de télévision ou l'imprimante *... Sur d'autres appareils, les organes d'entrée peuvent être des capteurs mesurant la température, la vitesse ou le débit d'un liquide. Les organes de sortie sont le plus souvent des moteurs électriques.



Les organes d'entrée/sortie sont reliés au microprocesseur par des fils généralement soudés à une sorte de longue prise appelée connecteur d'extension, qui vient se fixer sur le bord du circuit imprimé. Les signaux électriques transmis par la plupart des appareils ne fonctionnant pas en mode binaire, on emploie des puces spéciales, les interfaces, pour les convertir en signaux binaires et vice-versa.

Dialoguer avec les entrée/sortie

Le microprocesseur n'a aucun moyen de reconnaître un organe d'entrée/sortie, ni de savoir où il est branché. Pour lui faciliter la tâche, chaque organe d'entrée/sortie est traité exactement comme un emplacement mémoire. Il a sa propre adresse, il est relié aux bus d'adresses, de données et de commande, et l'information qu'il transmet ou qu'il reçoit se présente sous la forme d'un octet de données. Certains organes de sortie comportent deux adresses : le microprocesseur envoie les données à la première, et utilise la seconde (l'« adresse d'état ») pour obtenir des informations sur le fonctionnement de l'appareil.



* Les organes d'entrée/sortie d'un ordinateur s'appellent des « périphériques ».

Convertir les signaux

Voici comment on convertit les signaux électriques des organes d'entrée pour qu'ils soient utilisables par un microprocesseur.

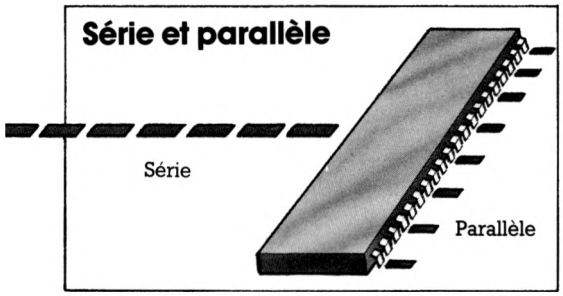
De l'analogique au numérique

Le signal électrique produit par un clavier d'ordinateur est totalement différent de celui d'un capteur de température. Chaque touche du clavier émet un signal lorsqu'elle est enfoncée, mais elle n'émet rien au repos ; ce type de signal est appelé numérique. Le capteur de température, lui, émet en permanence un signal, dont l'amplitude varie selon la température ; on parle alors d'un signal analogique.

Une montre traditionnelle à aiguilles est analogique, car c'est le déplacement continu de ses aiguilles qui indique l'heure. La montre électronique de droite est numérique, car elle affiche les chiffres par saccades.

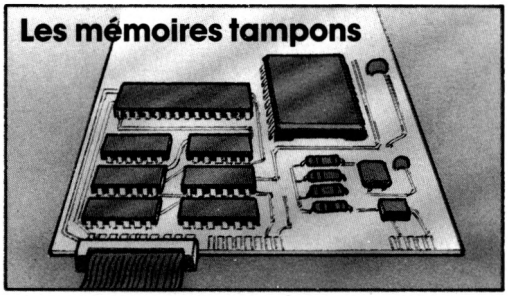
Le microprocesseur ne peut traiter que des signaux numériques : soit 1, soit 0, mais pas les valeurs intermédiaires. Toute information analogique (température, vitesse...) doit donc être convertie en signaux numériques avant de parvenir au microprocesseur : elle traverse donc une puce d'interface appelée convertisseur analogique/numérique. Ce système mesure les signaux analogiques plusieurs milliers de fois par seconde et transforme les résultats en nombres binaires envoyés au microprocesseur.

Série et parallèle



Les signaux électriques représentant une donnée parcourent le microprocesseur par groupes de huit, chacun circulant sur une piste distincte. On dit qu'ils circulent « en parallèle ». Mais dans certains appareils, les signaux doivent emprunter l'un après l'autre la même piste : on parle alors de « série ». Les programmes doivent alors traverser une interface qui convertit les signaux « série » en « parallèle ». Cette puce s'appelle convertisseur série/parallèle.

Les mémoires tampons

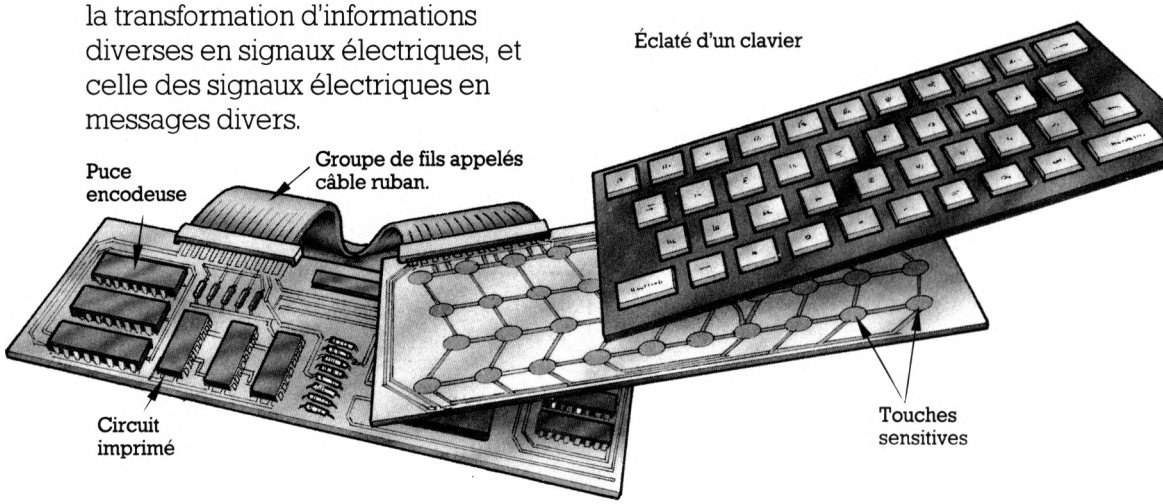


Les organes de sortie reliés au microprocesseur travaillent toujours beaucoup plus lentement que lui. Alors qu'il peut envoyer plus de mille caractères * par seconde vers l'imprimante, celle-ci mettra une minute ou deux à les taper. Pour compenser cette différence, on stocke dans certaines parties de la mémoire, appelées tampons, les caractères en attente.

* Un caractère peut être un nombre, une lettre ou un symbole.

Les entrée/sortie

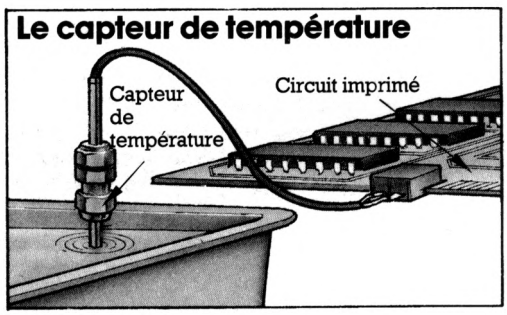
Voici quelques exemples illustrant la transformation d'informations diverses en signaux électriques, et celle des signaux électriques en messages divers.



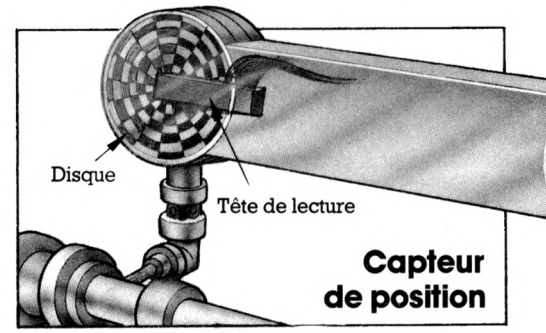
Le clavier de l'ordinateur

Certains ordinateurs ont un clavier sensitif : sous chaque touche, une sorte de pastille, imprimée sur une feuille de plastique, est reliée par des fils électriques à une grille figurant le clavier. Quand on enfonce une touche, les signaux correspondant à sa rangée et à sa colonne sont envoyés à une puce d'interface, l'encodeuse. Celle-ci identifie la touche et émet le signal binaire correspondant.

En fait, le clavier et son interface ont la même adresse : quand le microprocesseur attend une donnée du clavier, celle-ci provient en réalité de l'interface. Lorsque l'utilisateur tape des instructions, le microprocesseur va chercher chaque octet l'un après l'autre à l'adresse commune, et les met dans une puce RAM.



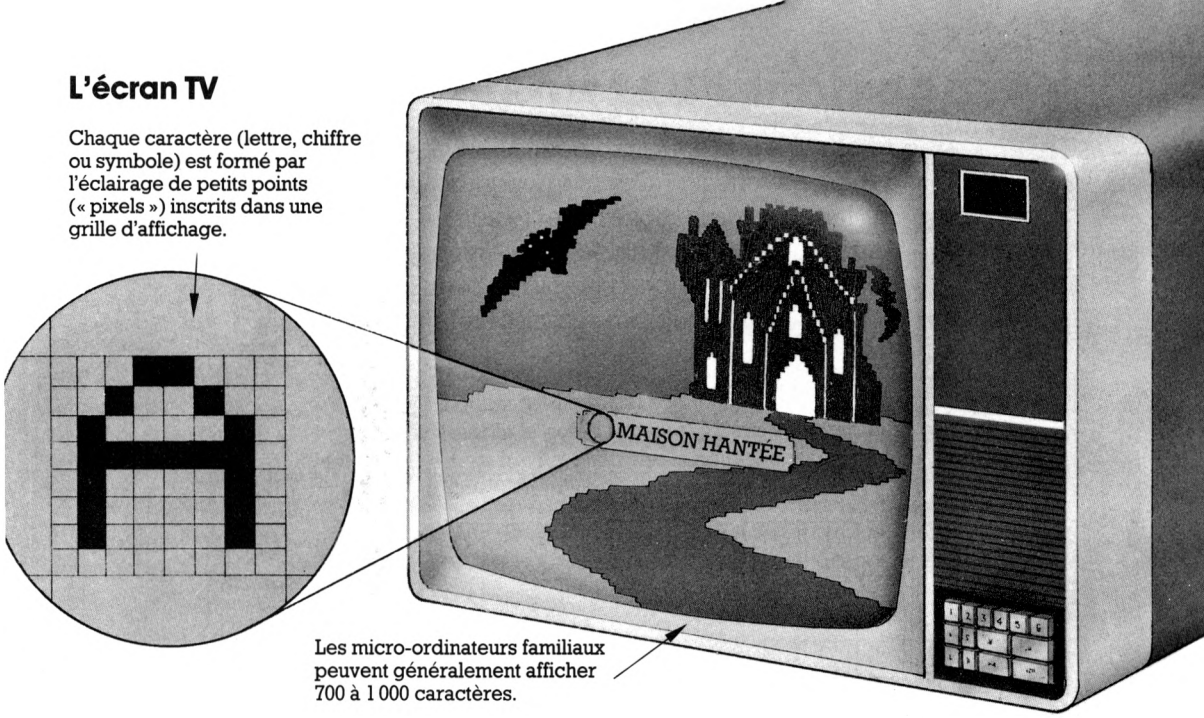
Ce capteur convertit la température d'un liquide (par exemple l'eau d'une machine à laver) en signaux électriques. Il renferme un petit composant électronique, appelé thermistor, qui produit un courant électrique variable selon la température de l'eau. Ce courant analogique doit être converti en signaux numériques avant son utilisation par le microprocesseur (cf. page 79).



Chaque articulation d'un bras de robot est équipée d'un capteur, appelé encodeur optique de position, qui permet au microprocesseur de connaître la position exacte de chaque partie du bras. Ce système comporte deux éléments : un disque placé sur la partie mobile du bras et une tête de lecture fixe. Le disque est divisé en segments, chacun étant recouvert d'une grille noire et blanche différente. La tête de lecture traduit en signaux binaires les grilles qui défilent devant elle, permettant ainsi de suivre les mouvements du bras.

L'écran TV

Chaque caractère (lettre, chiffre ou symbole) est formé par l'éclairage de petits points (« pixels ») inscrits dans une grille d'affichage.

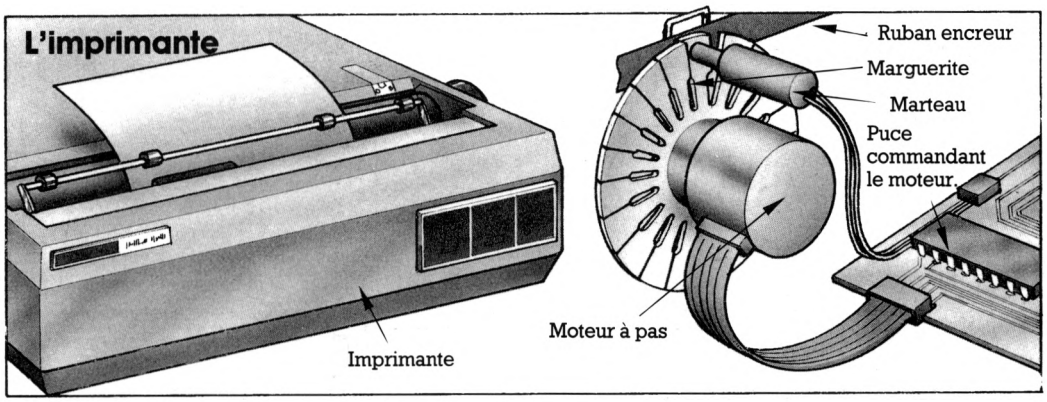


Les micro-ordinateurs familiaux peuvent généralement afficher 700 à 1 000 caractères.

L'écran du téléviseur, ou le moniteur, convertit les signaux électriques du micropocesseur en signaux lumineux pour former des mots, des nombres ou des dessins.

Chaque parcelle de l'écran pouvant contenir un caractère a une adresse distincte. Quand le microprocesseur doit afficher une information, il envoie, pour chaque caractère à afficher, un signal codé en binaire à une interface appelée générateur de caractères*. Celui-ci traduit ce code en signaux qui vont allumer des points particuliers de l'écran pour former le caractère voulu.

L'imprimante



Sur certaines imprimantes, les signaux électriques commandent un petit moteur qui actionne une sorte de roue en plastique appelée « marguerite », dont les pétales portent les mêmes caractères qu'une machine à écrire.

Quand le microprocesseur veut imprimer des données, il envoie, à l'adresse de l'imprimante, le code binaire des caractères souhaités. Les puces de l'imprimante convertissent ce code en signaux électriques pour amener la lettre voulue devant le marteau.

De nombreuses imprimantes ont deux adresses. La seconde transmet au microprocesseur divers renseignements : imprimante prête, manque de papier, ruban encreur à changer, etc.

* Le plus souvent, le générateur de caractères ne fait pas partie de la ROM, sauf sur quelques modèles de petits micro-ordinateurs.

Voyage dans l'UAL

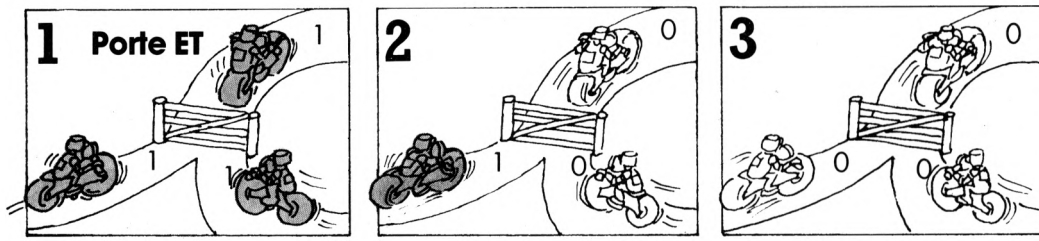
L'UAL est la partie du microprocesseur qui traite calculs et raisonnements logiques. Elle effectue ce travail en envoyant les octets de code binaire dans des circuits appelés « portes logiques », qui sont le cœur du microprocesseur.

On distingue plusieurs types de portes logiques suivant la manière dont sont agencés les transistors. Ces portes sont conçues pour produire des signaux différents selon les messages qu'elles reçoivent.

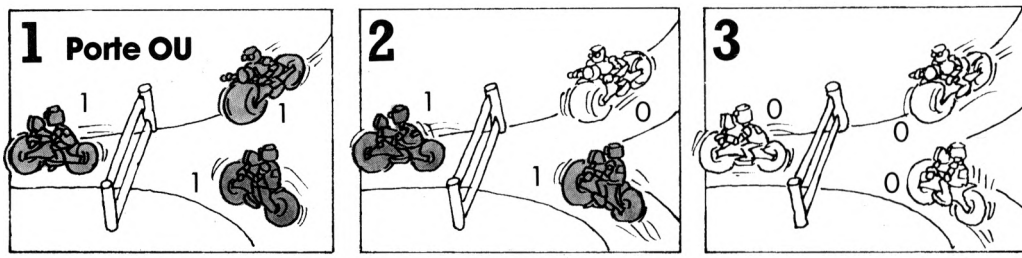
Comment fonctionnent les portes logiques

Les pistes d'entrée acheminent les signaux aux portes logiques, la piste de sortie transmet le résultat. La plupart des portes logiques comportent deux entrées et une sortie. On combine plusieurs portes pour exploiter les huit bits des codes de données.

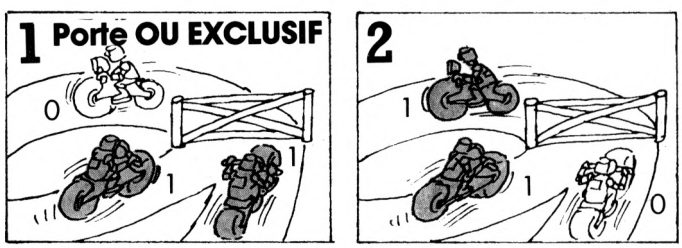
Voici comment fonctionnent les quatre principaux types de porte logique.



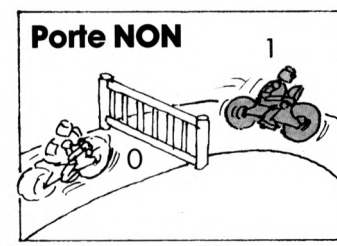
La porte ET comporte deux entrées et une sortie. Le signal de sortie est 1 si les deux signaux d'entrée sont 1 ; si la porte reçoit un ou deux signaux 0, le signal de sortie est 0 (dessins 2 et 3).



Si la porte OU reçoit un ou deux signaux d'entrée 1, son signal de sortie est 1.



Pour que le signal de sortie de cette porte, qui ressemble à la précédente, soit 1, il faut qu'un seul signal d'entrée soit égal à 1.

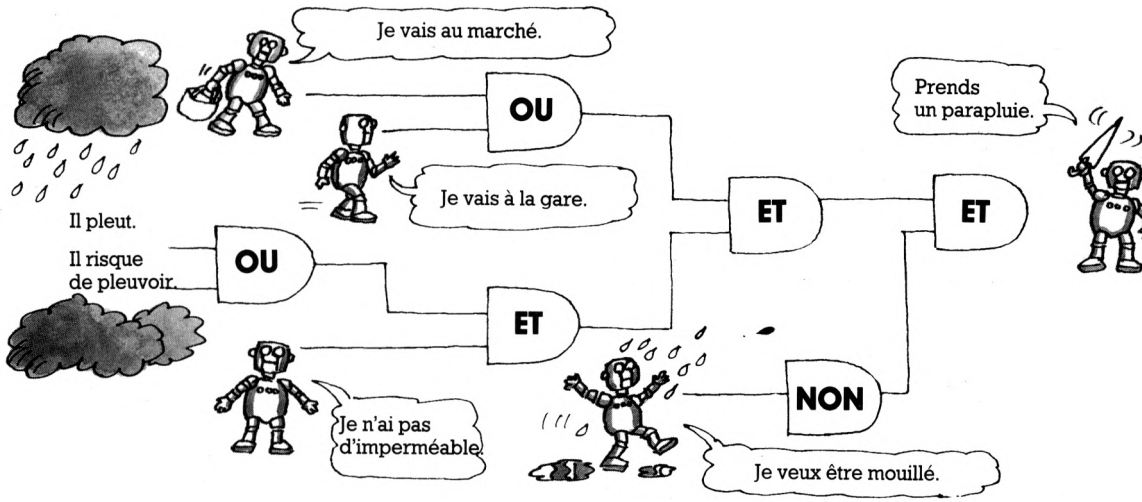


La porte NON n'a qu'une entrée. Elle émet toujours le signal inverse de celui qu'elle reçoit. (Si l'entrée est 1, la sortie est 0, et vice-versa*.)

* Cette porte est souvent appelée inverseur.

Les décisions complexes

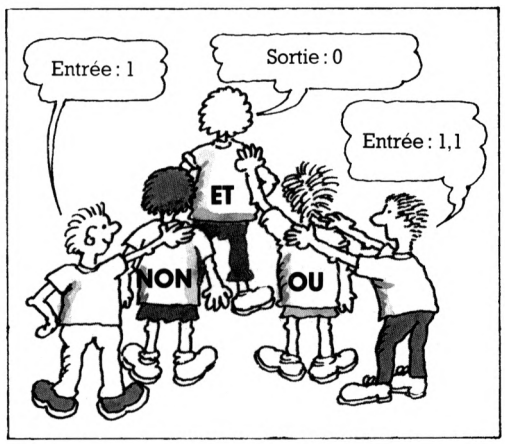
Chaque type de porte logique effectue une seule opération sur les signaux d'entrée. Pour que l'UAL puisse traiter des problèmes complexes dépendant de nombreux facteurs, on combine les portes comme dans l'exemple ci-dessous.



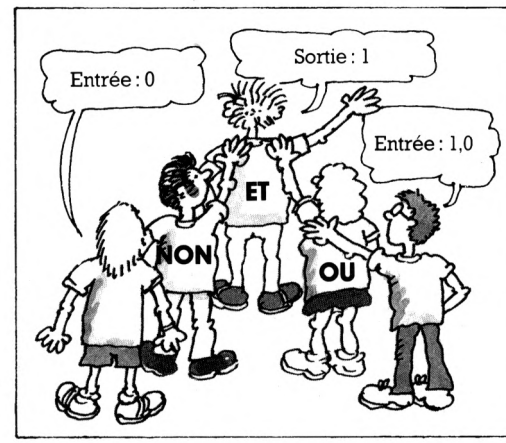
Dans ce circuit, le signal de sortie d'une porte devient l'un des signaux d'entrée de la porte suivante et ce, jusqu'à la porte finale. Tout problème qui peut être décomposé en une succession d'étapes logiques, peut être traité par les circuits logiques d'un microprocesseur.

Une chaîne humaine logique

Imaginons un circuit constitué par des êtres humains. Chaque personne représente une porte. Ses épaules sont les deux entrées, l'un de ses bras la sortie. Vous donnez au premier joueur les signaux d'entrée, et chacun doit donner le signal de sortie correspondant à la porte qu'il représente (voir page 82).



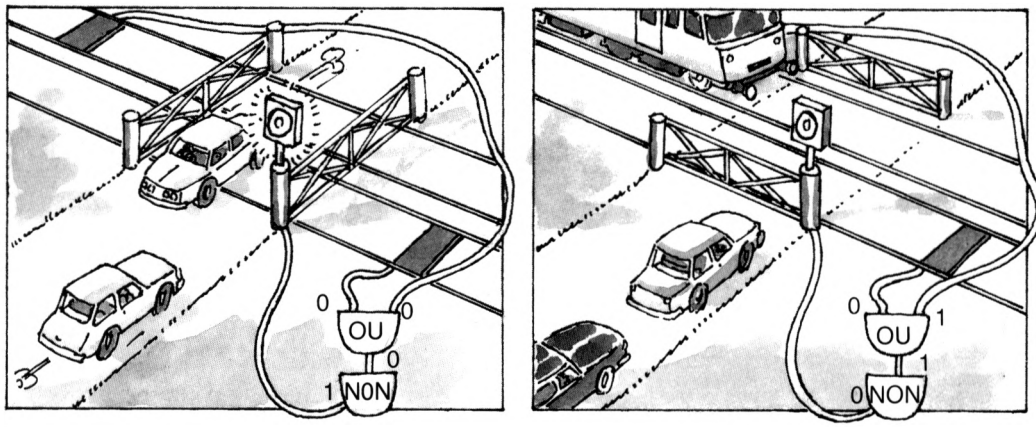
Si le joueur placé derrière vous touche votre épaule, cela équivaut à un signal d'entrée 1.



Lever le bras correspond à un signal de sortie 1. Pour le transmettre à la porte suivante, il faut toucher l'épaule du joueur suivant.

Construisez un circuit logique

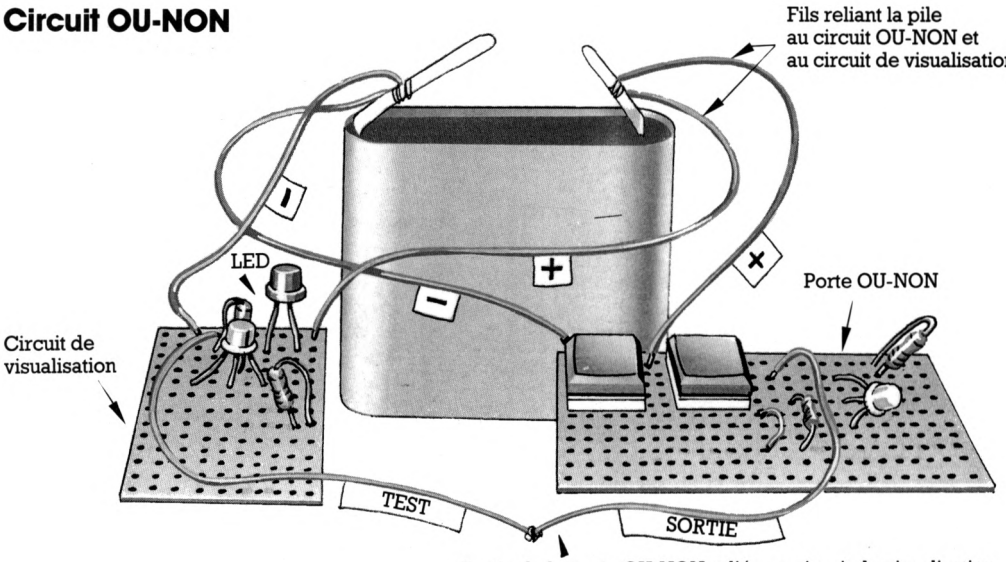
Ces dessins montrent comment une porte logique peut commander un appareil comme le feu d'un passage à niveau, et les pages suivantes vous expliqueront comment réaliser le circuit correspondant.



Le feu vert doit être allumé le plus souvent possible pour laisser passer les voitures. Mais dès que se présente un (ou deux) train(s), le feu vert doit s'éteindre. Le circuit logique comprend une porte OU combinée à une porte NON. Les entrées de la porte OU sont des interrupteurs placés sur les deux voies ferrées : ils envoient un signal 1 (une tension élevée) à la porte OU quand passe un train. Aussi, dès qu'un train se présente, la porte OU envoie un signal de sortie 1 à la porte NON. Celle-ci transforme le 1 en 0 (tension basse), ce qui éteint le feu vert.

La combinaison d'une porte OU et d'une porte NON s'appelle une porte OU-NON. Vous pouvez en construire une avec un seul transistor et deux résistances.

Circuit OU-NON



Ce dessin vous montre à quoi ressemblera votre porte OU-NON. Pour la tester, vous aurez besoin d'un second circuit dit de visualisation. Celui-ci utilise un composant appelé LED qui s'allume quand il reçoit un signal 1*.

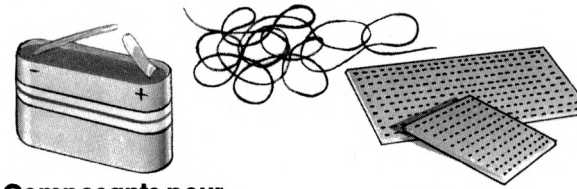
* Pour de plus amples renseignements sur les composants, reportez-vous à la page 92.

Le matériel nécessaire

Pour construire cette porte OU-NON, il vous faut une pile électrique, quelques composants électroniques et de quoi souder (reportez-vous aux pages 92 et 93 : vous y apprendrez à reconnaître les composants et à souder). Pour vous procurer ce matériel, adressez-vous à une boutique ou à une revue spécialisée.

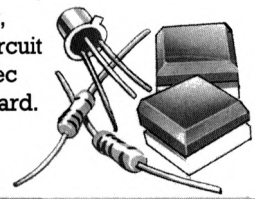
Ce que vous devez acheter

Une pile 4,5 volts ; 150 cm de fil de cuivre toronné ; deux plaques de « Veroboard » comprenant chacune 10 pistes de 24 trous.



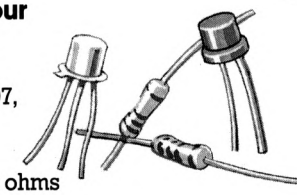
Composants pour la porte OU-NON

Un transistor BC 107, deux résistances de 1 Kohm (Ω), deux interrupteurs à poussoir, à deux broches, pour circuit imprimé compatible avec l'écartement du Veroboard.

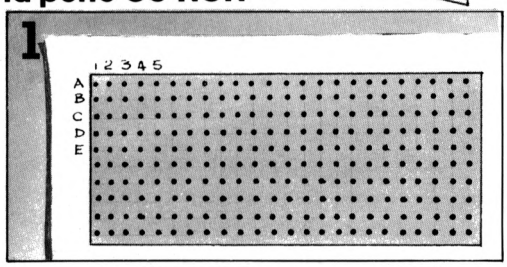


Composants pour la visualisation

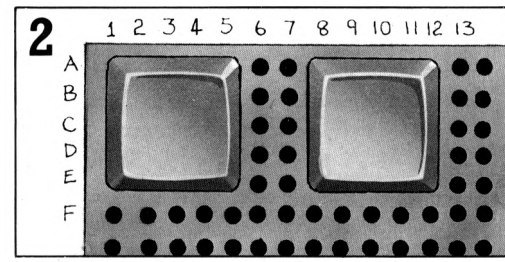
Une LED verte,
Un transistor BC 107,
Une résistance 1 Kohm,
Une résistance 270 ohms



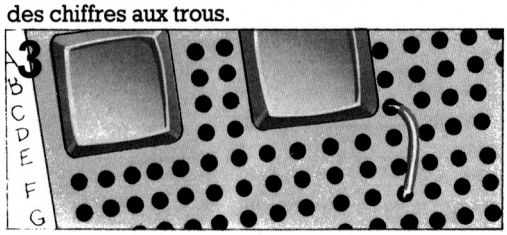
Construction de la porte OU-NON



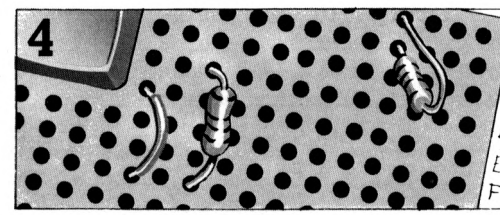
Disposez l'un des Veroboards sur une feuille de papier, les pistes étant placées horizontalement côté papier. Dessinez une grille en attribuant des lettres aux pistes, et des chiffres aux trous.



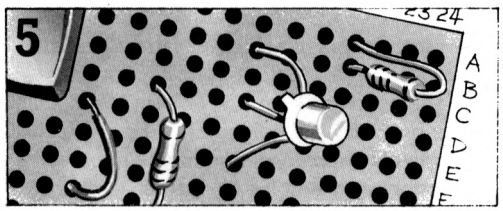
Soudez d'abord les broches de l'un des interrupteurs dans les trous A4 et E2, puis celles de l'autre dans les trous A11 et E9.



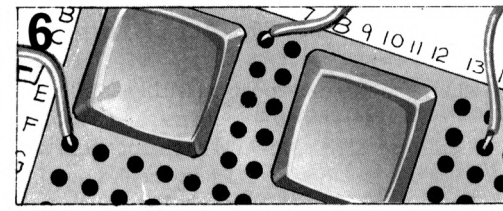
Coupez 15 mm de fil et dénudez chaque extrémité sur quelques millimètres. Soudez le fil dans les trous E13 et H13.



Soudez une résistance 1 Kohm (bandes marron, noire, rouge) dans les trous D15 et H15 ; soudez l'autre en A21 et B21.

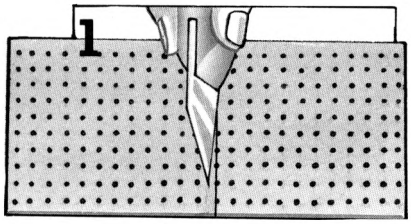


Soudez le transistor (collecteur en B18, base en D18, émetteur en F18). Pour identifier les pattes, reportez-vous à la page 92.

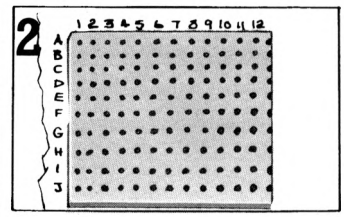


Dénudez deux fils de 30 cm et un de 6. Soudez un fil de 30 cm en A6 et collez-lui l'étiquette « + ». Soudez l'autre fil de 30 cm en F1 et appelez-le « − ». Soudez celui de 6 cm en B14 et appelez-le « SORTIE ».

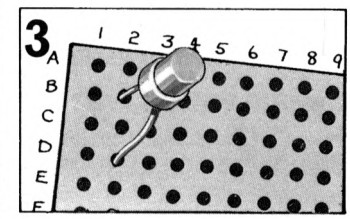
Construction du circuit de visualisation



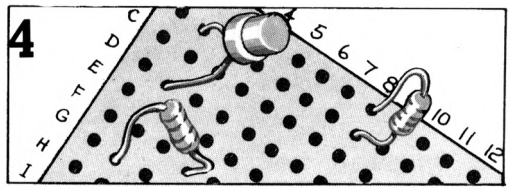
Coupez l'autre plaque de Veroboard pour qu'elle comporte dix pistes de douze trous. Amorcez la découpe avec un cutter.



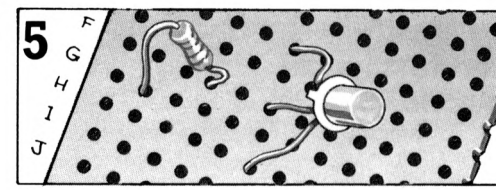
Mettez la plaque sur une feuille de papier, pistes côté papier à l'horizontale, et faites une grille comme ci-dessus.



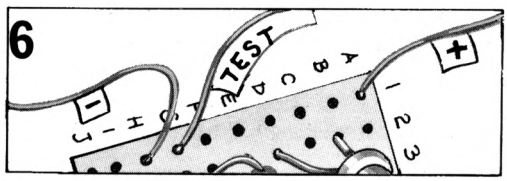
Soudez la LED dans les trous B2 et D2. La patte négative (voir page 92) doit être fixée en D2.



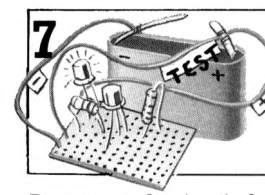
Soudez la résistance 1 Kohm (marron, noir, rouge) en G2 et F4. Soudez la résistance 270 ohms (rouge, violet, marron) en A8 et B8.



Soudez le transistor (collecteur en D6, base en F6, émetteur en H6).



Soudez un fil de 30 cm en A1 et mettez-lui l'étiquette « + ». Soudez un autre fil de 30 cm en H1 et appelez-le « − ». Soudez enfin un fil de 6 cm en G1 que vous appellerez « TEST ».

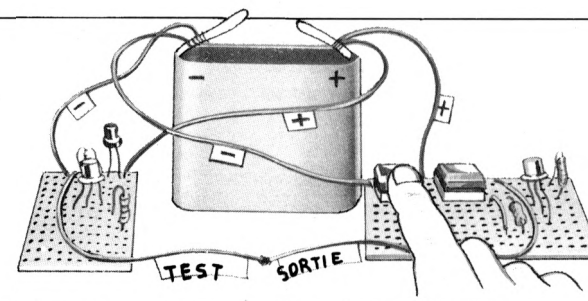


Si vous touchez la languette « − » avec le fil « TEST », la LED ne s'allume pas.

Pour tester le circuit de visualisation, raccordez-le à la pile électrique et mettez le fil « TEST » en contact avec le pôle « + » de la pile : la LED s'allume, montrant qu'il y bien une tension haute (1 en binaire*).

Essai de la porte OU-NON

Reliez le fil « SORTIE » de la porte OU-NON au fil « TEST » du circuit de visualisation (vous pouvez même les souder). Ensuite, raccordez les fils « + » des deux circuits au pôle « + » de la pile. Raccordez de même les fils « − » au pôle « − » de la pile.



Dès que les circuits sont branchés, la LED s'allume, indiquant qu'aucun interrupteur n'a été actionné (une porte OU-NON donne une tension de sortie haute quand les deux tensions d'entrée sont basses). Si vous appuyez sur l'un des interrupteurs (ou sur les deux), la lumière s'éteindra, indiquant que l'entrée est devenue 1, et la sortie 0.

* Si le circuit ne marche pas, vérifiez que tous les composants sont soudés à la bonne place et que les soudures sont correctes.

Pour en savoir plus sur les portes logiques

Les circuits logiques présentent l'intérêt de donner toujours les mêmes résultats. Une porte à deux entrées, comme la porte OU-NON, accepte en entrée quatre combinaisons binaires différentes : 0,0 ; 0,1 ; 1,0 ; 1,1. Les concepteurs de microprocesseurs écrivent des tables d'entrée/sortie qui montrent toutes les combinaisons d'entrées possibles et leurs résultats.

ENTRÉE A	ENTRÉE B	SORTIE
0	0	1
1	0	0
0	1	0
1	1	0

Table de vérité d'une porte OU-NON.

Si vous avez construit la porte OU-NON des pages précédentes, vous pouvez tester sa table de vérité. Appelez les interrupteurs, A et B, et essayez la table ligne par ligne en appuyant sur l'interrupteur A chaque fois que l'entrée A est égale à 1, et sur B quand l'entrée B est égale à 1. La LED verte donne la valeur de la sortie.

Une table d'entrée/sortie s'appelle en fait table de vérité. Amusez-vous à dresser celle des portes ET, OU et OU EXCLUSIF. (Aidez-vous des explications de la page 82 et vérifiez vos réponses page 95.)

Histoire des portes logiques

Les principes utilisés par les portes logiques électroniques et les tables de vérité ont été conçus vers 1850 par le mathématicien irlandais George Boole. Il étudiait le pourquoi des décisions humaines et n'avait probablement pas imaginé les développements futurs de ses théories.

Il conclut de ses travaux que tout décision humaine pouvait se ramener à trois processus intellectuels, qu'il appela ET, OU et NON. Ce sont ces processus que reproduisent les portes logiques.

Boole inventa les tables de vérité pour montrer que l'on pouvait prévoir toutes les conséquences d'un raisonnement, à condition que l'on sache si les éléments en sont vrais ou faux. Voici un exemple de table de vérité du raisonnement ET.

PROPOSITION A	PROPOSITION B	DÉDUCTION
Le soleil brille 	Je porte un manteau de fourrure 	J'ai trop chaud 
FAUX	FAUX	FAUX
VRAI	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	VRAI	VRAI

Si l'on remplace le mot vrai par le chiffre binaire 1, et faux par 0, cette table de vérité est absolument identique à celle d'une porte ET. Vérifiez-le en comparant avec la table de la page 95.

Quand on s'aperçut que l'on pouvait construire des circuits électroniques capables de reproduire les déductions de la logique humaine, le microprocesseur était né.

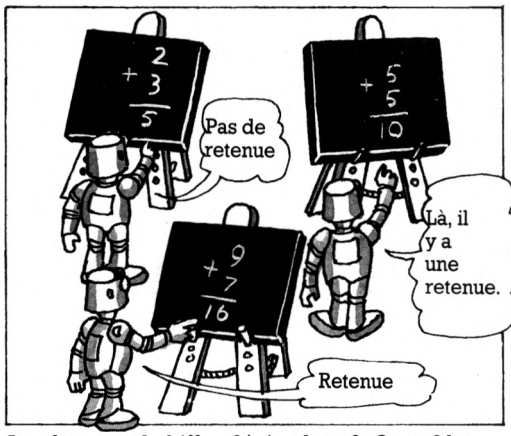
Calculez avec l'UAL

On peut combiner les portes logiques de façon à fabriquer un circuit capable de faire des additions. Celles-ci seront effectuées selon les règles du système binaire, expliquées ci-dessous.

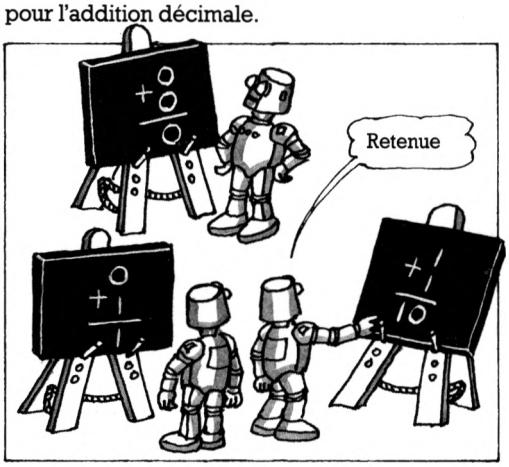
Ce circuit ne peut faire que des additions à un chiffre. Pour additionner des codes binaires à huit chiffres, on relie huit circuits les uns aux autres pour former un ensemble appelé additionneur.

Additionner en binaire

Les règles de l'addition binaire sont les mêmes que pour l'addition décimale.

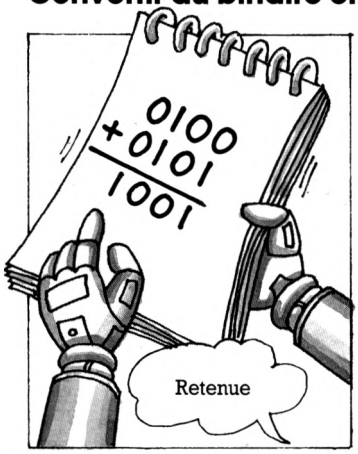


Le plus grand chiffre décimal est 9. Quand le résultat d'une addition est supérieur à 9 (5 + 5 ou 9 + 7), on note le résultat en retenant 1 et en remettant la colonne des unités à 0.



En binaire, les règles sont les mêmes. Mais, comme le plus grand chiffre est 1, vous devez faire une retenue dès que vous additionnez 1 + 1, comme sur l'exemple ci-dessus.

Convertir du binaire en décimal



Voilà comment on additionne deux nombres binaires à quatre bits. Vérifiez le résultat en convertissant les nombres en décimal.

BINAIRE	1	1	1	1
DÉCIMAL	8	4	2	1

$0100 = 4$

$0101 = 4+1 = 5$

$1001 = 8+1 = 9$

En binaire, le chiffre de droite équivaut à 2^0, le deuxième à 2^1, le troisième à 2^2, le quatrième à 2^3, etc. Chaque chiffre est donc égal au double du chiffre placé à sa droite. Pour convertir un nombre binaire en décimal, il suffit donc de remplacer chaque 1 par son équivalent décimal que vous trouvez dans le tableau ci-dessus. 0100 équivaut à 4 en décimal, 0101 à 5 et 1001 à 9.

Conception d'un circuit d'addition

Pour concevoir un circuit électronique capable d'additionner deux chiffres binaires, il faut dresser une table de vérité qui montre toutes les combinaisons offertes par ces chiffres et qui donne tous les résultats*.

Le circuit électronique devra être conforme à cette table, ses signaux d'entrée correspondant aux deux chiffres binaires. Mais il faut prévoir deux sorties, une pour le résultat, l'autre pour la retenue éventuelle.

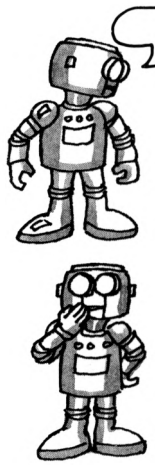


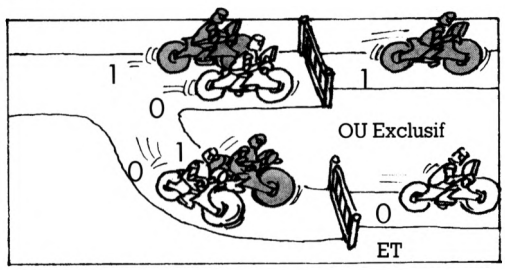
Table de vérité

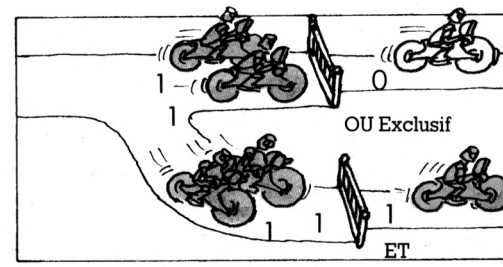
1. **0 + 0 = 0**
2. **1 + 0 = 1**
3. **0 + 1 = 1**
4. **1 + 1 = 10**

ENTRÉE DU PREMIER CHIFFRE	ENTRÉE DU DEUXIÈME CHIFFRE	SORTIE DU RÉSULTAT	SORTIE DE LA RETENUE
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Choisir les portes logiques

Deux portes logiques suffisent pour construire le circuit correspondant à cette table de vérité. Une porte OU Exclusif donnera le résultat, et une porte ET donnera l'éventuelle retenue. Les entrées, c'est-à-dire les chiffres à additionner, devront atteindre simultanément les deux portes.





Ces dessins montrent comment un circuit additionne 1 + 0 et 1 + 1. L'UAL comporte huit circuits semblables, chacun opérant sur l'un des huit bits des octets du code binaire. Un autre ensemble de circuits sert à prendre en compte les éventuelles retenues. Si le dernier bit d'un octet déclenche une retenue, l'indicateur de retenue du Registre d'État prend la valeur 1 (voir page 73).

Comment sont effectuées les autres opérations

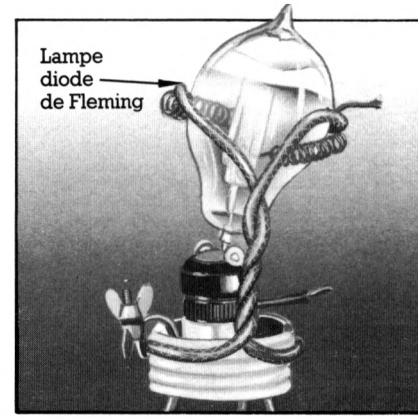
L'UAL résoud toutes les opérations en faisant de simples additions. Pour faire une soustraction, il donne une valeur négative à l'un des nombres et les additionne. Pour faire une multiplication ou une division, il se contente d'additionner ou de soustraire autant de fois que nécessaire. Les calculs plus complexes, comme les racines carrées, sont divisés en étapes élémentaires qui se ramènent à des additions. La démarche utilisée pour résoudre ces calculs s'appelle un algorithme. La puce d'une calculatrice renferme en mémoire ROM un algorithme par opération.

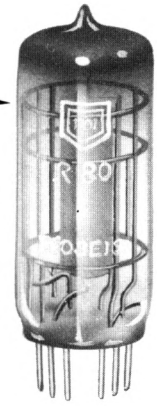
* Le système décimal comportant dix chiffres, l'addition de deux chiffres donne 10^2, soit 100 combinaisons.

Histoire de la puce

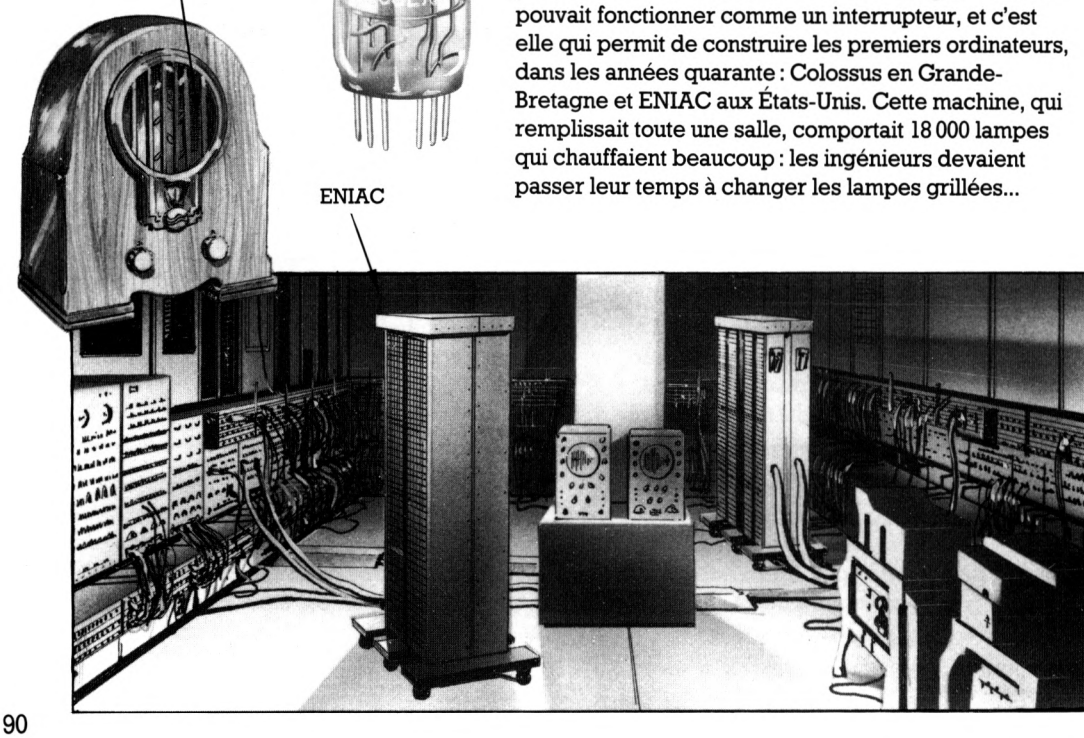
Les débuts de la puce remontent aux recherches qui ont été à l'origine de l'essor foudroyant qu'a connu l'électronique au cours de ce siècle. Les composants électroniques ont d'abord été utilisés pour reproduire des sons et pour réguler la vitesse des moteurs électriques. Puis leur miniaturisation a permis de concevoir et de fabriquer des microprocesseurs.

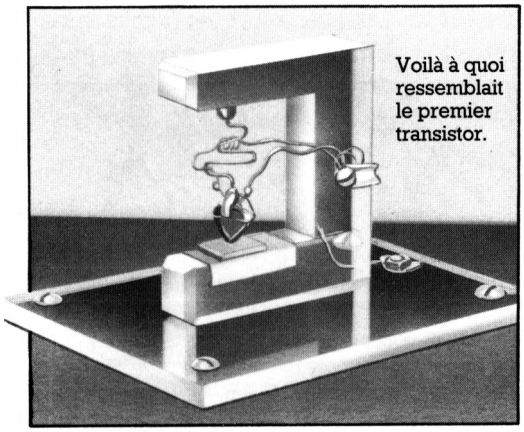
Ce n'est qu'au début de ce siècle que les savants découvrirent qu'ils pouvaient contrôler le courant électrique. L'électronique naît en 1905 avec l'invention de la « lampe à deux électrodes », ou diode, par Ambrose Fleming. Dans un tube sous vide, un fil porté à une certaine température émet des électrons (des particules chargées d'électricité négative) qui sont attirés par une plaque de métal placée à l'autre extrémité du tube : ainsi, le courant ne peut passer que dans un sens.





Deux ans plus tard, Lee de Forest s'aperçut qu'en mettant une grille métallique au milieu de la diode, on pouvait faire varier l'intensité du courant. Cette découverte, baptisée triode, permit l'essor de la radio, puis de la télévision. On découvrit aussi que la triode pouvait fonctionner comme un interrupteur, et c'est elle qui permit de construire les premiers ordinateurs, dans les années quarante : Colossus en Grande-Bretagne et ENIAC aux États-Unis. Cette machine, qui remplissait toute une salle, comportait 18 000 lampes qui chauffaient beaucoup : les ingénieurs devaient passer leur temps à changer les lampes grillées...



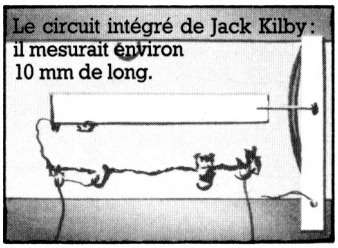


Voilà à quoi ressemblait le premier transistor.

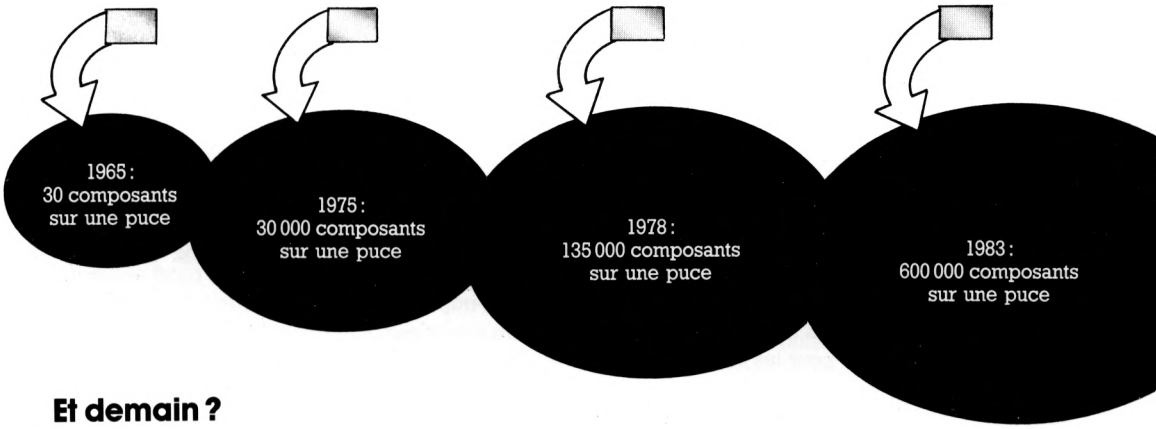
En 1947, trois Américains, Bardeen, Brattain et Shockley, inventèrent le transistor, composant capable de remplacer une lampe mais petit, solide et qui n'a pas besoin de chauffer. Les premiers transistors utilisaient comme semi-conducteur le germanium, qui fut plus tard remplacé par le silicium.

Les transistors servirent d'abord pour les appareils auditifs, puis ils remplacèrent les tubes dans les téléviseurs, magnétophones et radios, et équipèrent des matériels plus complexes comme les centraux téléphoniques. Leur utilisation dans les ordinateurs remonte à 1953.

En 1958, Jack Kilby, un Américain qui travaillait à Texas Instruments, eut l'idée de placer deux transistors sur un cristal de silicium : le premier circuit intégré était né.
Les programmes américains de défense et de conquête de l'espace exigeant une très grande miniaturisation, les chercheurs réussirent à faire tenir de plus en plus de composants sur une simple puce de 5 mm^2.



Le circuit intégré de Jack Kilby : il mesurait environ 10 mm de long.



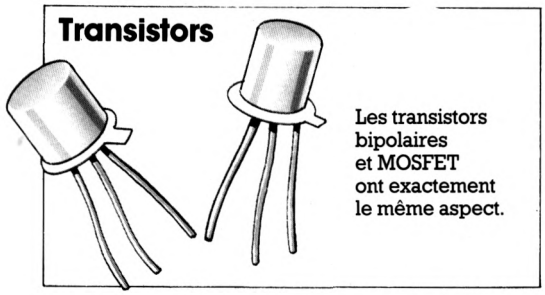
Et demain ?

Les fabricants essaient sans cesse d'accroître le nombre de composants et la rapidité des puces. On étudie actuellement un interrupteur à basse tension, très rapide, qui remplacerait les transistors. Ce super-interrupteur, imaginé en 1962 par Brian Josephson de l'Université de Cambridge, et mis au point par les laboratoires Bell aux États-Unis, se compose de deux fines couches de métal séparées par une couche isolante encore plus fine. A des températures très basses, les métaux deviennent hyper-conducteurs et constituent un interrupteur dix fois plus rapide que le meilleur transistor actuel. On estime qu'un ordinateur qui, actuellement, occupe une salle entière, pourrait être réduit au volume d'un ballon et fonctionner beaucoup plus vite grâce à ces jonctions de Josephson. Mais cet ordinateur devra travailler à des températures « polaires ».

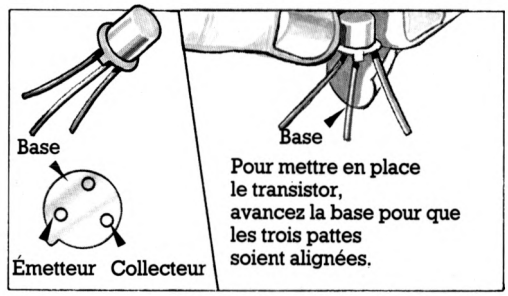
On étudie même la possibilité de faire pousser... des puces biologiques !

Conseils pour fabriquer le circuit

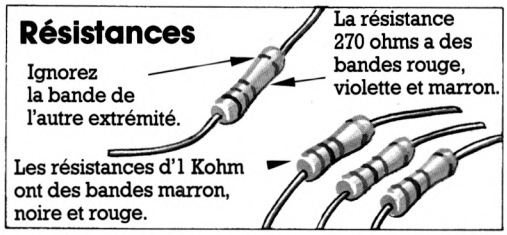
Ces deux pages vous donnent des « tuyaux » sur les composants électroniques et sur l'art de souder, pour vous aider à fabriquer les circuits présentés pages 84-86.



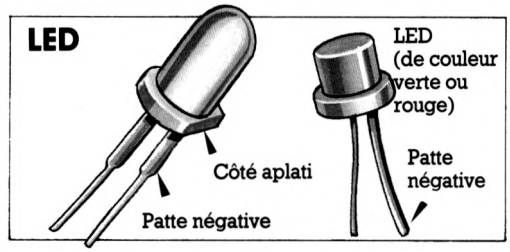
Pour les circuits de ce livre, on utilise des transistors bipolaires (ou de jonction) légèrement différents des transistors à effet de champ (appelés MOSFET ou FET, de l'anglais Field Effect Transistor) décrits page 8. Mais tous deux jouent le rôle d'interrupteurs.



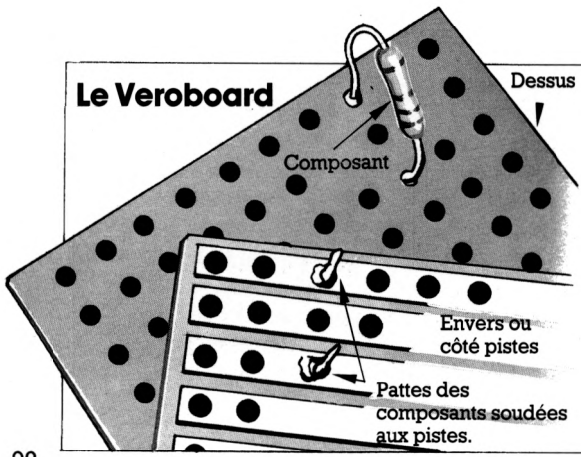
Les trois pattes d'un transistor bipolaire sont : l'émetteur, la base et le collecteur. Pour que le courant passe de l'émetteur au collecteur, il faut envoyer une tension électrique sur la base. Le dessin ci-dessus vous montre comment identifier les pattes du transitor BC107.



Les résistances réduisent l'intensité du courant de la pile pour qu'elle soit adaptée aux composants. Leur puissance, qui se mesure en ohms, est indiquée par le groupe de trois bandes colorées qui figure à l'une de leurs extrémités. Dans le cas des résistances, le symbole K a son sens habituel de 1 000, et non de 1 024 comme pour les octets.



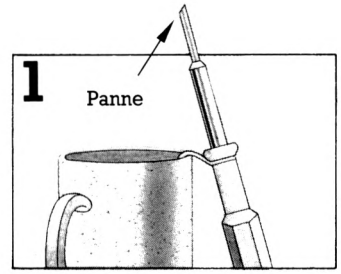
LED est l'abréviation de l'anglais Light Emitting Diode (diode électroluminescente). Une diode ne laisse passer le courant que dans un sens, la LED devient alors lumineuse. Elle a une patte positive et une patte négative, et il importe de ne pas se tromper au moment du montage. On reconnaît la patte négative soit parce qu'elle est plus grosse, soit parce qu'elle est placée près du côté aplati.



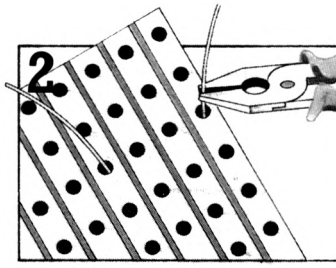
Le Veroboard, spécialement conçu pour fabriquer des circuits électroniques, comporte des pistes en cuivre qui relient des rangées de trous. On glisse les pattes des composants et les fils électriques dans les trous, et on les soude aux pistes en cuivre. Le courant électrique circule le long de ces pistes pour relier les différents composants. Pour déterminer la taille d'un morceau de Veroboard, on indique le nombre de pistes et le nombre de trous par piste que l'on désire (ici, dix pistes × vingt-quatre trous).

Comment souder

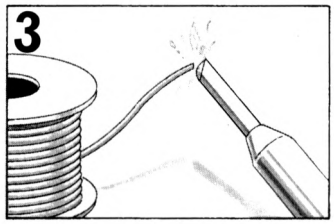
Il vous faut :



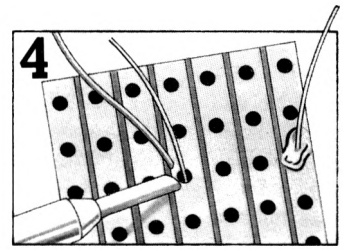
Branchez le fer à souder. Assurez-vous que la panne ne touche rien pendant que le fer chauffe.



Avant de souder un composant, vérifier son emplacement, glissez les pattes dans les trous, et courbez-les légèrement avec la pince.

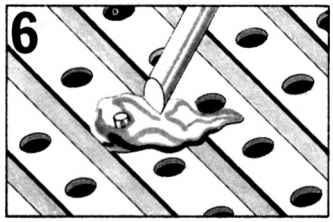


Quand la panne est chaude, approchez la soudure : une goutte doit fondre et adhérer à la panne.

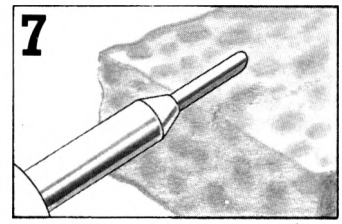


Puis placez simultanément le fer et la soudure de chaque côté de la patte ; une goutte de soudure doit fondre, entourer la patte et la fixer à la piste de cuivre.

Laissez refroidir quelques secondes. Coupez les pattes au ras de la soudure avec la pince coupante.



S'il y a de la soudure entre les pistes, enlevez-la soigneusement en passant le fer chaud dans les rainures.



Après chaque soudure, nettoyez la panne sur l'éponge humide. N'oubliez pas de débrancher le fer quand vous aurez fini.

Dessouder

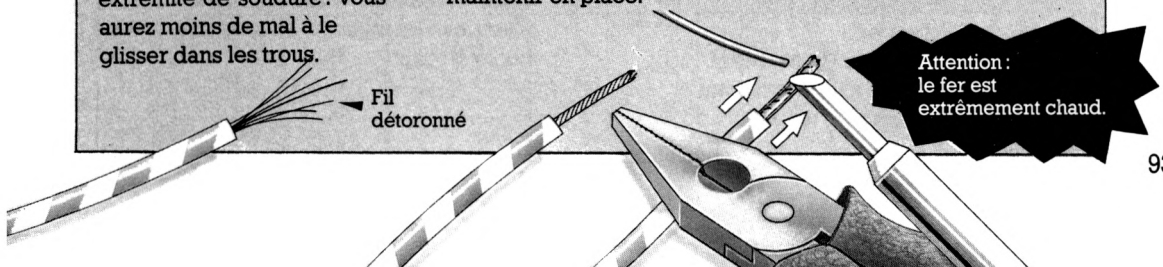
Pour dessouder un composant, demandez à quelqu'un de tenir le circuit et de faire levier avec un crayon glissé entre les pattes du composant pendant que vous faites fondre la soudure avec le fer.

Comment étamer

Si vous utilisez du fil électrique toronné, vous avez intérêt à l'étamer, c'est-à-dire à recouvrir son extrémité de soudure : vous aurez moins de mal à le glisser dans les trous.

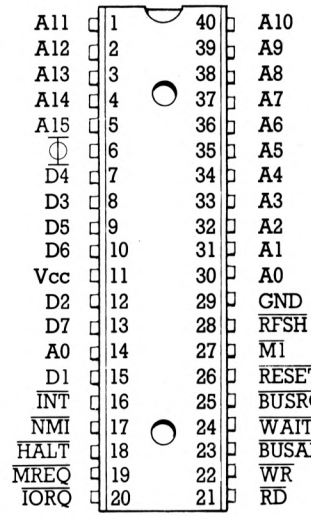
Dénudez le fil sur 1 cm environ. Resserrez bien les brins. Posez quelque chose de lourd sur le fil pour le maintenir en place.

Effleurez-le quelques instants avec la soudure et la panne : une fine couche de soudure doit l'enrober.



Carte des broches du microprocesseur

Par les broches du boîtier du microprocesseur circulent l'alimentation électrique, les signaux d'adresses, de données et de commande. Ce plan vous indique la disposition des broches de l'un des microprocesseurs les plus utilisés, le Z80, fabriqué par Zilog. On donne aux broches, numérotées de 1 à 40, des abréviations qui précisent leur rôle. Vous en trouverez l'explication ci-dessous. Les abréviations surmontées d'une barre horizontale indiquent que la broche est active quand la tension est basse (voir page 75).



A0-A15 Ces broches transmettent les seize bits des codes d'adresses. A signifie Adresse, et le nombre indique la place du bit dans les deux octets du code.

D0-D7 Ces broches transmettent les huit bits des codes de données. D signifie Data (« données »).

Φ Cette broche transmet le signal d'horloge (symbolisé par la lettre grecque Phi majuscule) au microprocesseur.

Vcc Entrée de l'alimentation électrique. Reliée à la borne + 5 volts.

INT (Interruption). Entrée de demande d'interruption : quand ce signal est activé, le microprocesseur termine le cycle en cours et s'arrête. Est utilisé en cas d'urgence (surchauffe, etc.).

NMI (Interruption non masquable). Le microprocesseur ne peut ignorer le signal transmis à cette entrée, contrairement à celui de la broche précédente, que l'on peut masquer par programmation.

HALT Le microprocesseur envoie ce signal pour indiquer aux autres puces qu'il s'est momentanément interrompu.

MREQ (Memory Request : appel mémoire). Signal de commande envoyé par le microprocesseur pour prévenir les puces de mémoire que le bus d'adresses est occupé.

IORQ (Input/Output Request : appel entrée/sortie). Prévient les organes d'entrée/sortie que le bus d'adresses est occupé.

RD (Read : lecture). Indique que la donnée doit être lue à un emplacement mémoire ou dans un organe d'entrée.

WR (Write : écriture). Indique que la donnée doit être écrite à un emplacement mémoire ou transmise vers un organe de sortie.

BUSAK (Bus Acknowledge : validation du bus de données). Utilisé lorsque le microprocesseur partage les bus d'adresses et de données avec d'autres microprocesseurs. Sert à signaler que le microprocesseur a libéré les bus d'adresses et de données.

WAIT (Attente). Entrée du signal envoyé au microprocesseur pour lui demander de laisser passer quelques cycles d'horloge. Est utilisé par exemple lorsqu'il faut aller chercher des données dans des mémoires plus lentes que les autres. Le microprocesseur s'interrompt dès que la tension de cette broche est basse, et ne reprend son travail que lorsqu'elle retrouve son niveau initial.

BUSRQ (Bus Request : demande de bus). Entrée du signal envoyé par un autre microprocesseur pour indiquer qu'il réclame les bus d'adresses ou de données.

RESET (Remise à zéro). Entrée du signal qui ramène à zéro le Compteur Ordinal lorsque l'ordinateur est mis en route ou lorsque l'utilisateur appuie sur la touche RESET.

M1 Signal envoyé par le microprocesseur pour prévenir les autres puces qu'il cherche une instruction en mémoire.

RFSH (Memory Refresh : rafraîchissement de la mémoire). Sortie du signal envoyé par le microprocesseur pour permettre aux mémoires RAM de garder intactes les données qu'elles stockent.

GND (Ground : masse). Sortie du courant électrique du microprocesseur. Est reliée à la borne 0 volt.

Lexique

ACIA (Asynchronous Communication Interface Adaptor). Interface qui permet d'effectuer des conversions du mode série au mode parallèle et vice-versa.
Bascule (Flip-Flop). Circuit électrique, constitué de plusieurs transistors, pouvant prendre deux états qui représentent les valeurs 1 et 0. Est utilisé pour former les cellules des puces de mémoire et les registres du microprocesseur.
CMOS (Complementary Metal Oxide Semiconductor). Circuit intégré à faible consommation, utilisant simultanément des circuits MOS à canal p et à canal n.
EAPROM ou **EAROM** [Electrically Alterable (Programmable) ROM]. Mémoire semblable à l'EPROM, mais dont on efface les programmes en appliquant de très fortes tensions à certaines broches (également appelées EEPROM ou EEROM).
EPROM Puce semblable à une PROM, mais dont les programmes peuvent être effacés par exposition aux rayons ultra-violets. C'est pourquoi cette puce a toujours un trou au sommet du boîtier.
LSI (Large Scale Integration). Expression utilisée pour quantifier les composants regroupés sur une puce : une puce LSI comporte 100 à 10 000 composants.
MOS (Metal Oxide Semiconductor). Technologie utilisée pour fabriquer la plupart des puces et qui utilise des métaux comme conducteurs et le dioxyde de silicium comme isolant.
MOFSET (Metal Oxide Semiconductor Field Effect Transistor). Il y a deux types de MOSFET : ceux à canal n (décrits pages 56-57) et ceux à canal p dans un substrat de type n.
MSI (Medium Scale Integration). Circuit intégré comprenant dix à cent composants.
NMOS Type de puce très rapide, qui ne contient que des MOSFET à canal n.
PMOS Type de puce qui ne contient que des MOSFET à canal p, grande consommatrice d'énergie.
Porte NON-ET Porte logique donnant des résultats opposés à ceux de la porte ET.
PROM (Programmable ROM). Type de puce dont les éléments binaires sont des fusibles ou des diodes qui permettent de programmer la puce après sa fabrication.
RAM dynamique Puce de mémoire qui a besoin d'être rafraîchie constamment pour que les données qu'elle contient ne soient pas détruites.
RAM statique Puce de mémoire qui n'a pas besoin d'être régénérée pour conserver ses données.
SSI (Small Scale Integration). Circuit intégré comprenant moins de dix composants.
TTL (Transistor Transistor Logic). Circuit logique utilisant des transistors bipolaires et non MOSFET.
UART (Universal Asynchronous Receiver/ Transmitter). Interface qui assure les conversions série/parallèle.
ULA (Uncommitted Logic Array). Puce comportant des circuits logiques que le fabricant peut associer au gré de l'utilisateur.
VIA (Versatile Interface Adaptor). Type d'interface qui peut effectuer toutes sortes de conversions (Analogique/Numérique, Série/ Parallèle).
VLSI (Very Large Scale Integration). Circuit intégré comportant plus de dix mille composants.

Table de vérité ET

ENTRÉE 1	ENTRÉE 2	SORTIE
0	0	0
1	0	0
0	1	0
1	1	1

Table de vérité OU

ENTRÉE 1	ENTRÉE 2	SORTIE
0	0	0
1	0	1
0	1	1
1	1	1

Table de vérité OU EXCLUSIF

ENTRÉE 1	ENTRÉE2	SORTIE
0	0	0
1	0	1
0	1	1
1	1	0

Index du guide du microprocesseur

Loi n° 49.956 du 16 juillet 1949
sur les publications destinées à la jeunesse - dépôt 11-85
Dépôt légal n° 1503-11-1985
ISBN : 2.01.011416.7 29.62.0546.01

Imprimé en Espagne

Dans la même collection :

Que ne peut-on pas faire avec un ordinateur ! Calculer, bien sûr, mais aussi lui poser des questions, écrire des poèmes, jouer à quantité de jeux plus palpitants les uns que les autres, composer même de la musique...
Petits guides pratiques d'introduction à la micro-informatique, les ouvrages de cette nouvelle collection font découvrir toutes les possibilités qu'offrent les micro-ordinateurs. Ils initient au langage et au fonctionnement de l'ordinateur, apprennent à programmer et — pourquoi pas ? — à créer des programmes originaux ! La clarté du texte, la gaieté des couleurs, la drôlerie des dessins, tout est conçu dans ces livres pour faire de cette initiation un plaisir.

MICRO-INFORMATIQUE

- Guide Hachette du micro-ordinateur.
- Introduction à la micro-informatique.
- Guide pratique du BASIC.
- Pratique et maîtrise du BASIC (pour écrire facilement vos programmes).
- Micro-Pratique (des programmes, des jeux, des montages à réaliser avec votre micro).
- Écrivez vos jeux d'aventures pour votre micro-ordinateur.
- Jeux électroniques : Battlegames.
- Jeux électroniques : Spacegames.
- Jeux électroniques : fantômes et lieux hantés.
- Comment jouer avec son ordinateur et sa vidéo.

MATHS

- La calculatrice de poche.
- Jouer et apprendre avec sa calculatrice.

NOUVELLES TECHNOLOGIES

- Passeport pour les médias.
- Tout savoir sur les robots.

9 782010 114168

29/0546/1
85-XI

Prix : 55,00 F TTC

ÉCHOS ÉLECTRONIQUE