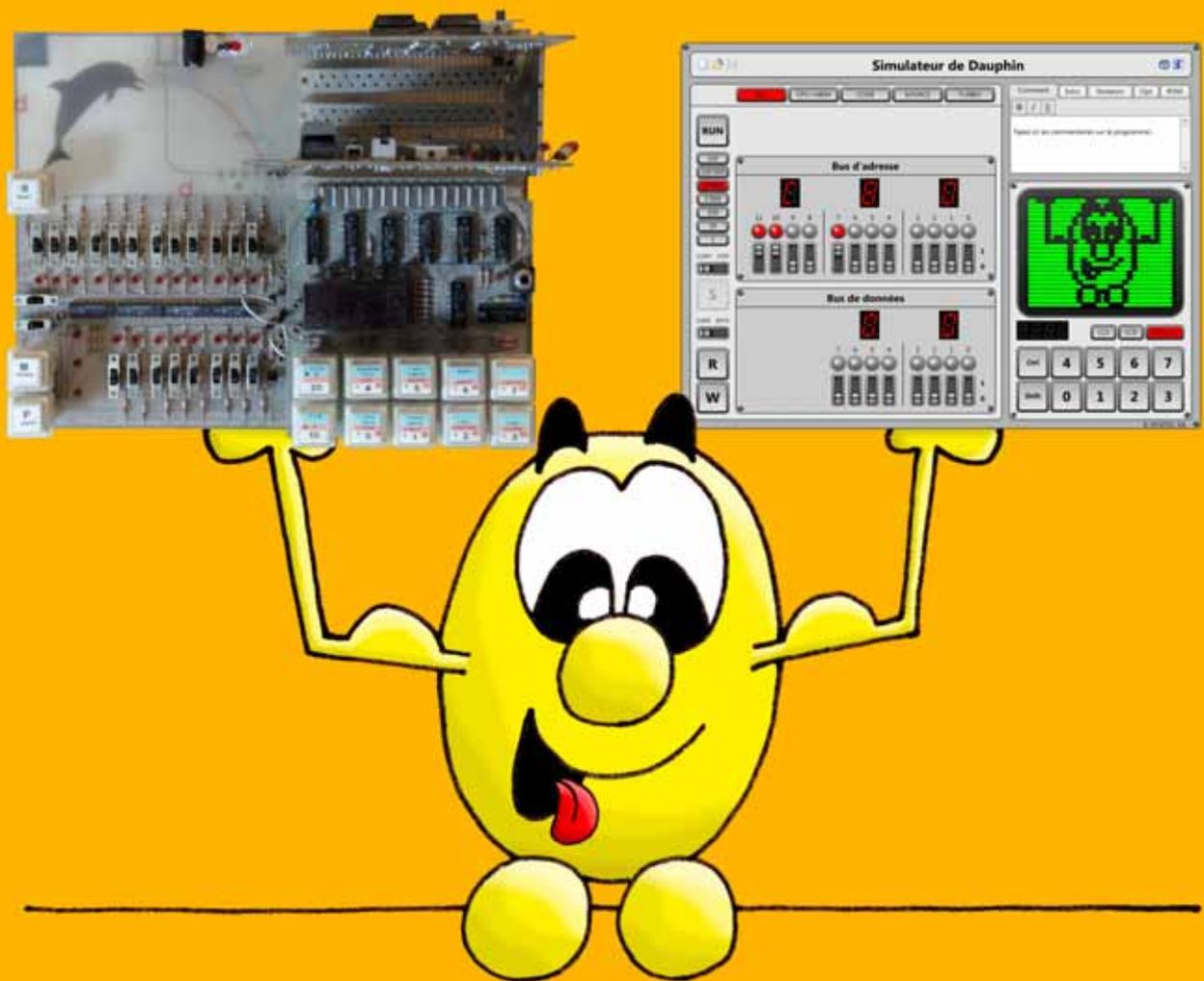


Comprendre les microprocesseurs



Le Dauphin de 1977 et son simulateur de 2007

Octobre 2007

Logiciel, texte et dessins de Daniel Roux



Edition spéciale, offerte à l'occasion du 30^{ème} anniversaire d'EPSITEC SA.

Ce manuel est prévu pour être utilisé avec le logiciel « Simulateur de Dauphin », que vous pouvez télécharger gratuitement à cette adresse :

www.epsitec.ch

EPSITEC SA est l'éditeur des logiciels de gestion Crésus.

Edition octobre 2007 / DR

© 2007 EPSITEC SA. Tous droits réservés.

Imprimé en Suisse.

Sans la permission expresse et écrite de la part de EPSITEC SA, Ch. de la Mouette 5, CH-1092 Belmont, aucune partie de ce manuel ne peut être reproduite ou transmise par quelque moyen que ce soit.

Intel® est une marque déposée de Intel Corporation aux Etats-Unis et dans d'autres pays.

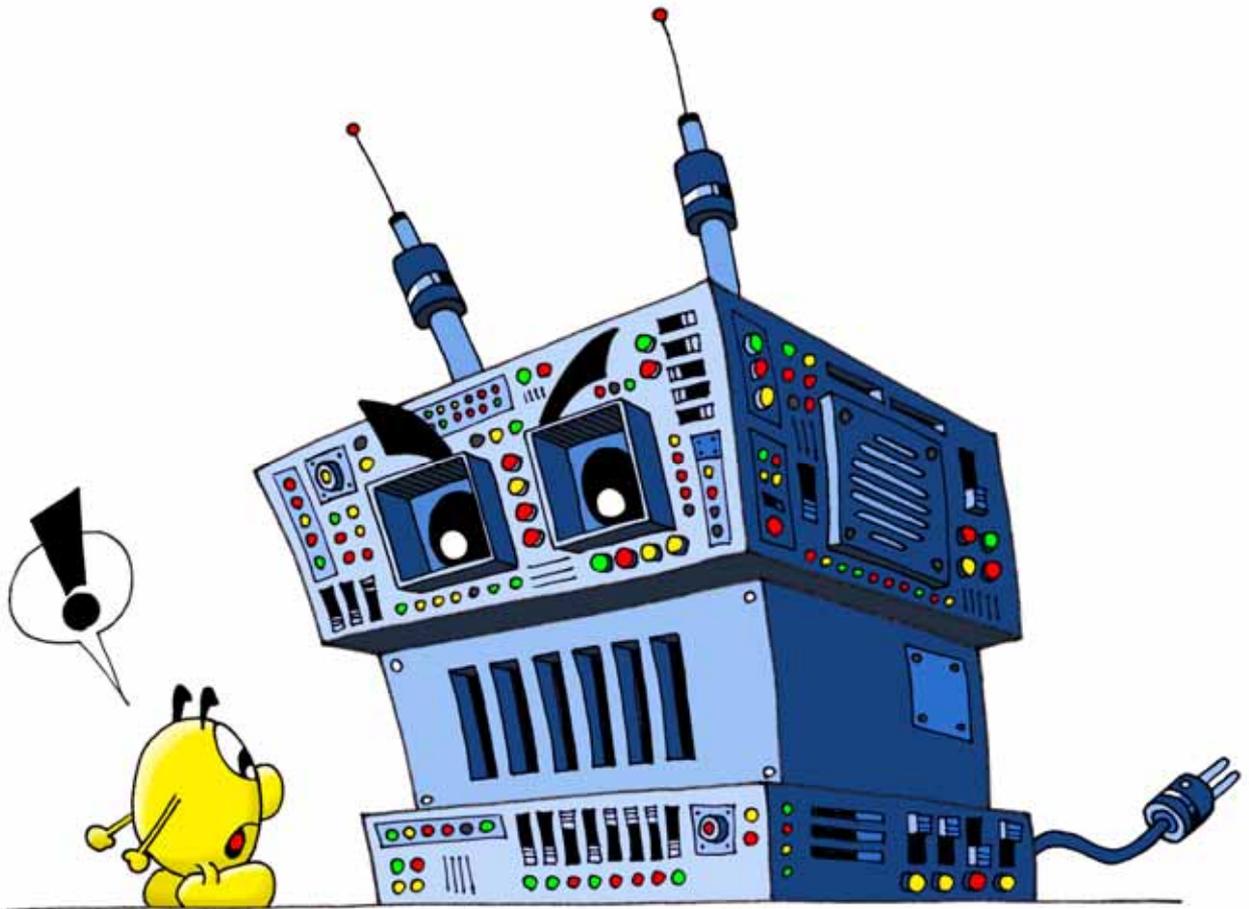
Windows® est une marque déposée de Microsoft Corporation aux Etats-Unis et dans d'autres pays.

Sommaire

1	Introduction	3
1.1	Le véritable Dauphin.....	3
1.2	Le simulateur de Dauphin.....	4
2	Binaire et hexadécimal	6
3	Pour ne pas perdre la mémoire.....	8
4	Une mémoire de Dauphin	10
4.1	Les périphériques	12
4.2	L'affichage.....	13
4.3	L'écran bitmap	15
4.4	Le clavier	16
4.5	La mémoire morte.....	17
5	Le processeur.....	18
5.1	Tout de suite ?.....	20
5.2	Dans les entrailles	21
5.3	Le saut	22
5.4	Premier saut dans l'inconnu.....	22
5.5	Mouvement perpétuel	24
5.6	L'addition s'il vous plait.....	25
5.7	Silence, on tourne	26
5.8	Sauter ou ne pas sauter, telle est la question	27
5.9	Lecture du clavier.....	28
5.10	Routines en mémoire morte	31
5.11	Codage manuel ou automatique	32
5.12	Bonjour.....	35
5.13	L'assembleur d'instructions.....	37
5.14	Encore des rebonds	39
5.15	La pile	42
6	A l'aide.....	44
7	C'est méga.....	44

1 Introduction

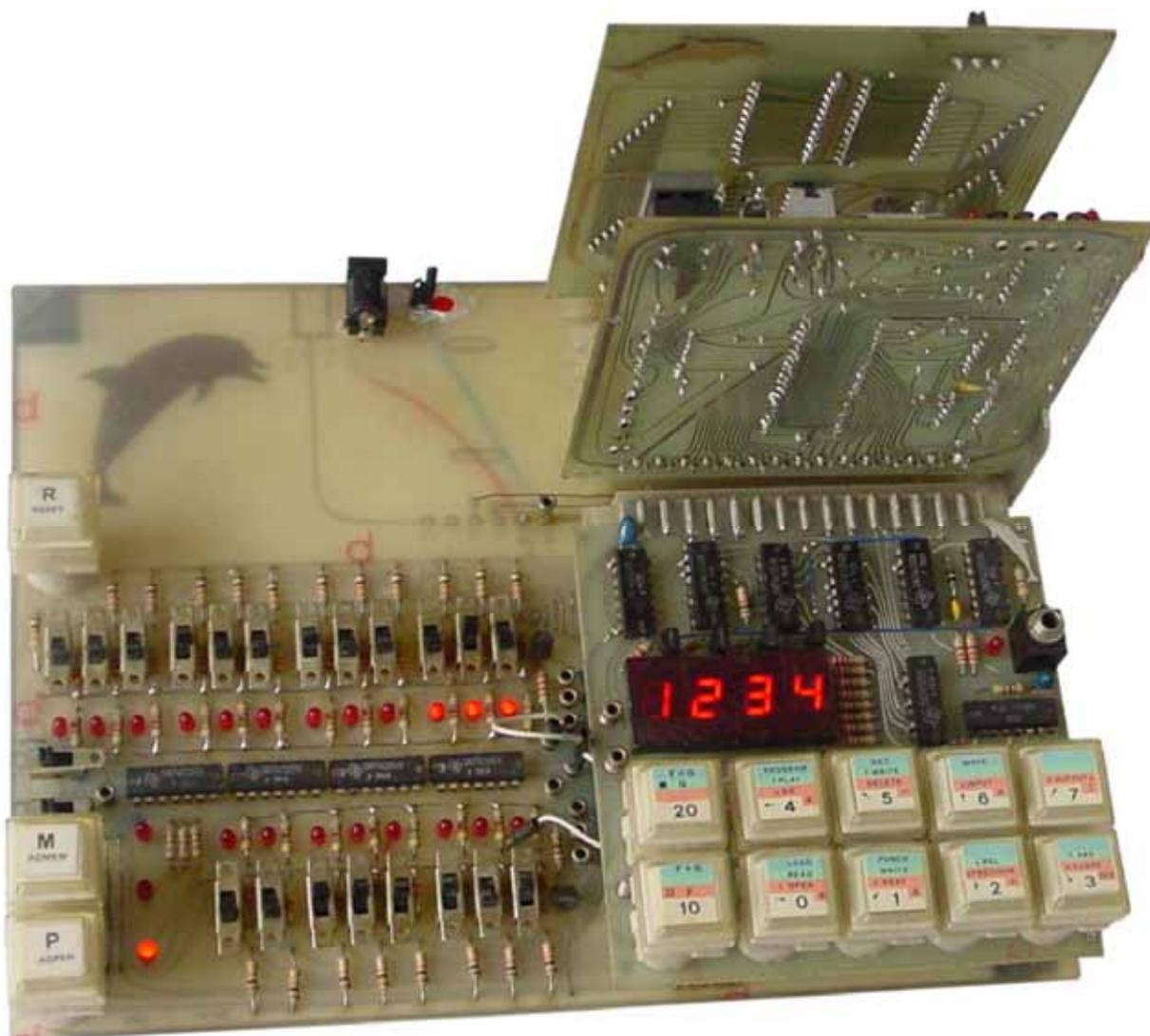
A la préhistoire de l'informatique, il y a un peu plus de 30 ans, les ordinateurs étaient de grosses machines boutonnières. L'opérateur réglait le monstre à l'aide d'une ribambelle de boutons, et l'ordinateur communiquait avec d'impressionnantes rangées de lumières clignotantes.



1.1 Le véritable Dauphin

En 1971, une invention révolutionne l'industrie des gros ordinateurs; Intel® réussit à graver le premier microprocesseur sur une petite plaquette de silicium. C'est le début de l'ère des microprocesseurs, qui ne vont cesser de se perfectionner.

En 1977, le professeur Jean-Daniel Nicoud crée le Dauphin, un génial petit ordinateur livré en kit, qu'il fallait construire soi-même. Cette sympathique machine a rencontré un grand succès en Suisse romande, permettant à de nombreux passionnés de s'initier à cette nouvelle discipline.



1.2 Le simulateur de Dauphin

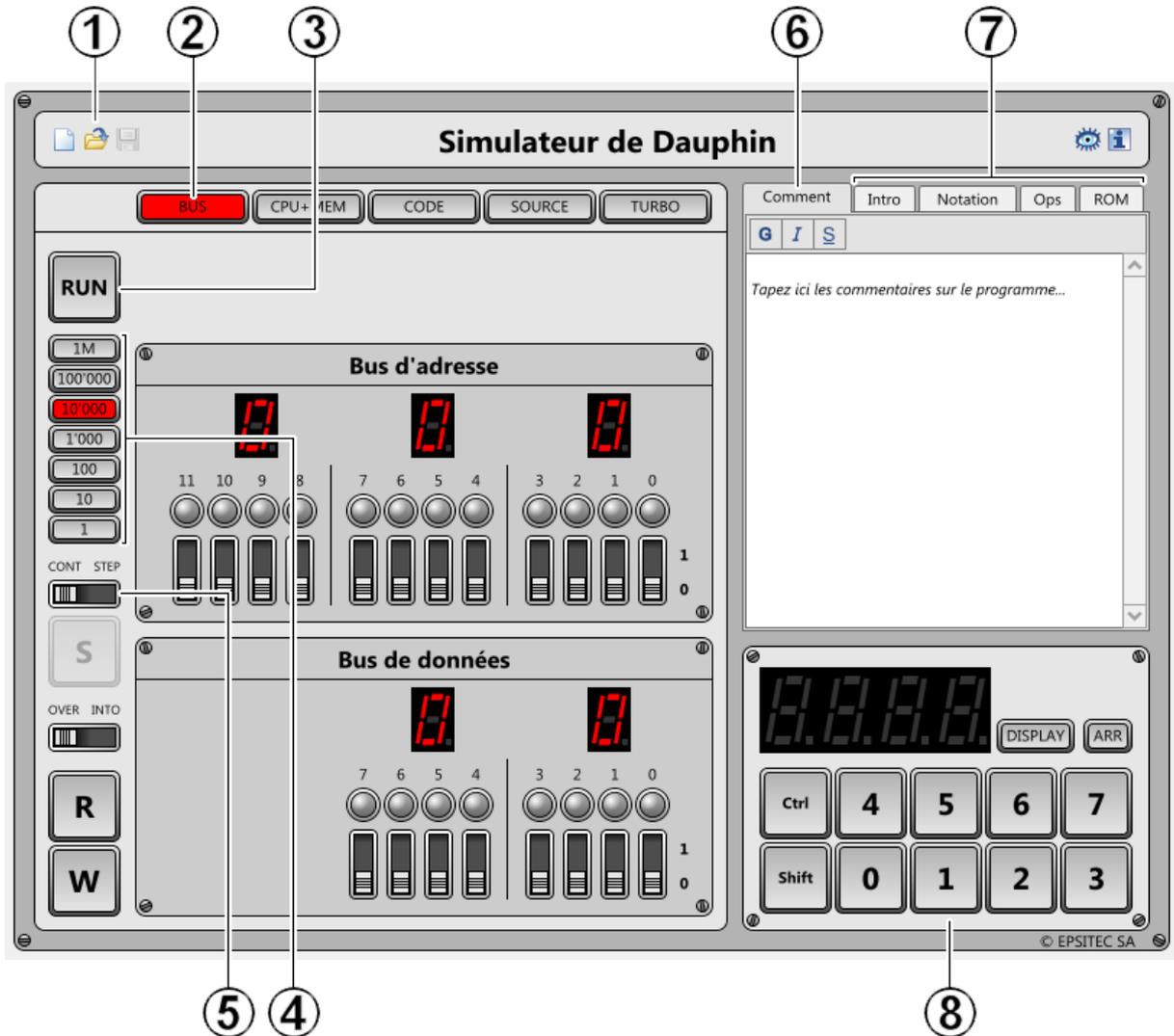
Le simulateur de Dauphin est un logiciel qui fonctionne sur la plupart des PC récents équipés de Windows® Vista, XP ou 2000. Vous pouvez le télécharger gratuitement à l'adresse www.epsitec.ch/dauphin.

Avec ce simulateur, vous vous trouvez face à un ordinateur rudimentaire, vierge de tout logiciel, qui ne sait strictement rien faire, exactement comme à l'époque des pionniers de l'informatique. C'est vous qui lui donnez toutes les instructions qu'il doit exécuter. Même les tâches les plus simples telles qu'afficher la valeur correspondant à la touche pressée doivent être programmées. Vous acquerez ainsi les bases de la programmation en « langage machine », une chose presque totalement oubliée de nos jours.

Bien que le Dauphin soit infiniment moins puissant qu'un ordinateur actuel, tous les principes que vous apprendrez ici restent valables avec l'informatique moderne. Nous sommes persuadés que vous comprendrez mieux comment fonctionne votre PC, à la lecture de ce manuel.

Nous nous efforcerons d'utiliser les termes techniques français, en mentionnant leurs équivalents anglais entre parenthèses et en italique.

Après avoir installé et exécuté le logiciel, l'écran ressemble à ceci :



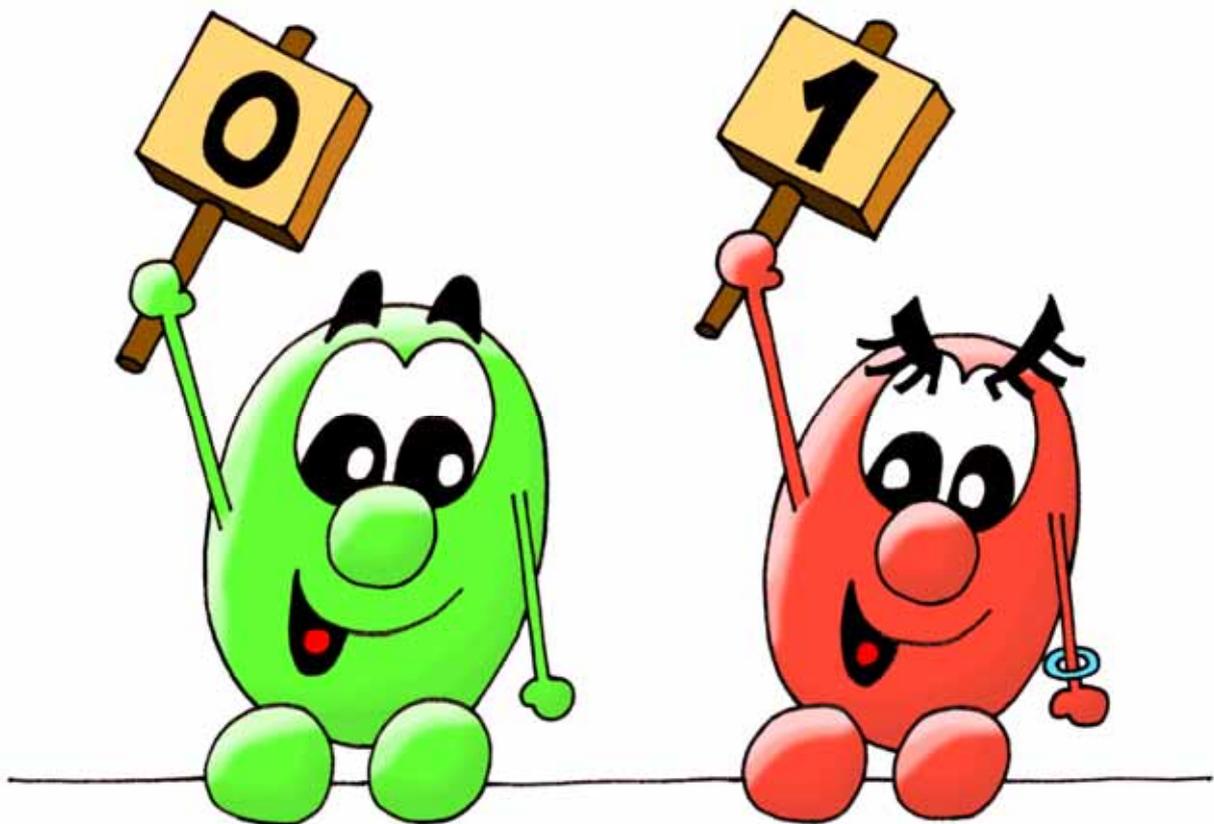
1. Icônes pour effacer, ouvrir ou enregistrer un programme.
2. Choix des panneaux affichés en dessous. [**BUS**] affiche les bus d'adresse et de données.
3. Bouton principal [**RUN/STOP**] pour démarrer ou arrêter le processeur.
4. Choix de la vitesse du processeur, en nombre d'instructions par secondes.
5. Commutateur [**CONT STEP**] pour choisir le mode continu ou « pas à pas » (*step by step*).
6. Commentaires sur le programme ouvert.
7. Résumé des instructions du microprocesseur.
8. Clavier et affichage du Dauphin, activés par des accès bus et certaines instructions.

2 Binaire et hexadécimal

Un ordinateur moderne est composé de plusieurs milliards de transistors, qui ont la particularité de fonctionner en « tout ou rien ». Un transistor conduit l'électricité ou ne la conduit pas. L'information véhiculée par un fil peut donc prendre deux états : 0 ou 1. On parle de système binaire.

L'information la plus petite gérée par un ordinateur est le bit, qui peut prendre les états 0 ou 1. Un bit permet de représenter deux valeurs. Par exemple, on pourrait utiliser un bit pour déterminer si un personnage est vert ou rouge. Si le bit est à zéro, le personnage est vert, et s'il est à un, il est rouge :

0	Vert
1	Rouge



Il est facile de comprendre que deux bits permettent de représenter quatre valeurs :

00	Trèfle
01	Carreau
10	Pique
11	Cœur

Si on ajoute un troisième bit, on pourra représenter huit valeurs. Par exemple, trois bits permettraient de déterminer un jour de la semaine, la huitième valeur étant ici inutilisée :

000	Lundi
001	Mardi
010	Mercredi
011	Jeudi
100	Vendredi
101	Samedi
110	Dimanche
111	Inutilisé

En fait, l'ajout d'un bit double à chaque fois le nombre de valeurs possibles. Avec seulement huit bits, on arrive déjà à représenter 256 valeurs, ce qui est suffisant pour définir un caractère (lettre minuscule ou majuscule, chiffre, signe de ponctuation, etc.). En informatique, un ensemble de huit bits est appelé un octet (*byte*). C'est une grandeur souvent utilisée.

La notation binaire n'est pas pratique, car on écrit de longues suites de 0 et de 1. Par exemple, le nombre 250 s'écrit 11111010. On préfère une notation condensée appelée hexadécimal. Les bits y sont regroupés par quatre. Un groupe de quatre bits permet de représenter 16 valeurs. On utilise les chiffres 0 à 9 pour les dix premières, puis les lettres A à F pour les six dernières :

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Par exemple, le nombre décimal 250 (équivalent binaire de 1111 1010) s'écrit FA en hexadécimal. Le nombre décimal 19 (équivalent binaire de 0001 0011) s'écrit 13 en hexadécimal. Afin d'éviter les confusions, on précède généralement un nombre décimal de « D' » et un nombre hexadécimal de « H' ». Donc, le nombre décimal D'19 s'écrit H'13 en hexadécimal. Si rien n'est précisé, il s'agit d'un nombre décimal.

3 Pour ne pas perdre la mémoire

La mémoire d'un ordinateur est un composant essentiel. C'est là que sont stockés les données et les programmes. Aussi grande et complexe que soit la mémoire d'un ordinateur, on n'y effectue que deux opérations élémentaires, écrire ou lire :

- Ecrire (*write*) consiste à mettre une valeur dans la mémoire.
- Lire (*read*) consiste à retrouver une valeur précédemment écrite.

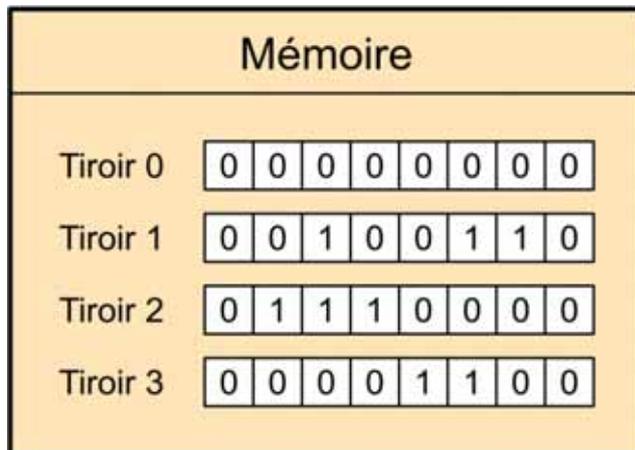
Prenons par exemple une mémoire de 32 bits. Les mémoires sont souvent regroupées en octets. Notre mémoire sera donc composée de 4 octets (4 octets de 8 bits chacun, ce qui donne bien $4 \times 8 = 32$ bits).



On peut considérer cette mémoire comme une commode ayant 4 tiroirs. Les tiroirs sont numérotés de 0 à 3, et chaque tiroir contient 8 bits.

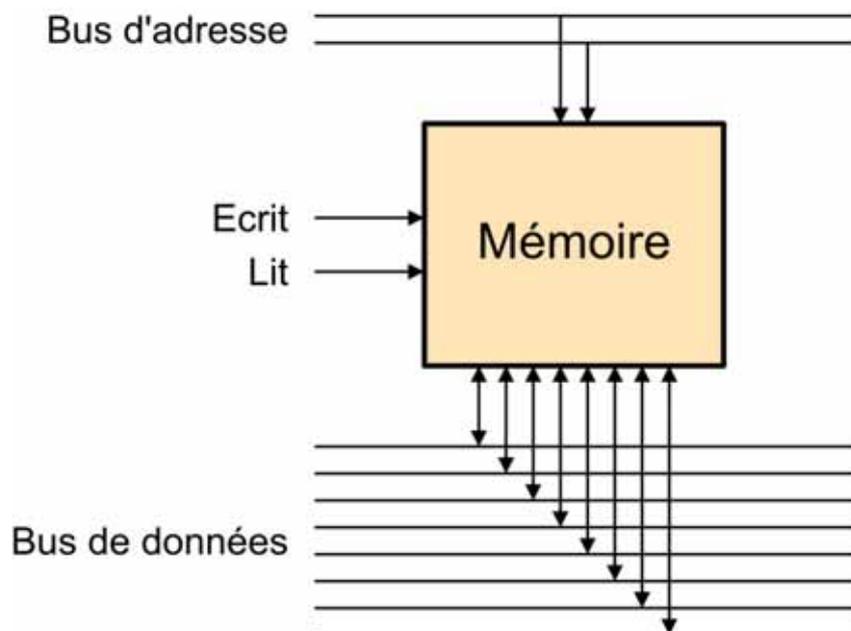
Note : En informatique, il est usuel de commencer à numéroté à partir de zéro, ce qui peut être troublant !

Dans la figure ci-dessous, le tiroir numéro 1 (donc le deuxième depuis le haut) contient la valeur binaire 00100110, c'est-à-dire H'26 en hexadécimal :



Pour écrire un octet, il faut donner deux informations : la valeur à écrire et le numéro du tiroir. La valeur à écrire est appelée donnée (*data*) et le numéro du tiroir est appelé adresse (*address*).

Physiquement, notre mémoire de 32 bits serait reliée au monde extérieur par 8 connections pour les données et 2 connections pour les adresses 0 à 3. Un ensemble de connexions est appelé *bus*. La mémoire est donc accédée avec un bus d'adresse et un bus de données.



Deux fils supplémentaires, appelés *Ecrit* et *Lit* dans la figure ci-dessus, permettent d'agir sur la mémoire.

- Une impulsion sur le premier fil écrit la valeur présente sur le bus de données dans le tiroir désigné par le bus d'adresse.
- Une impulsion sur le deuxième fil place le contenu du tiroir choisi par le bus d'adresse sur le bus de données; on lit la mémoire.

4 Une mémoire de Dauphin

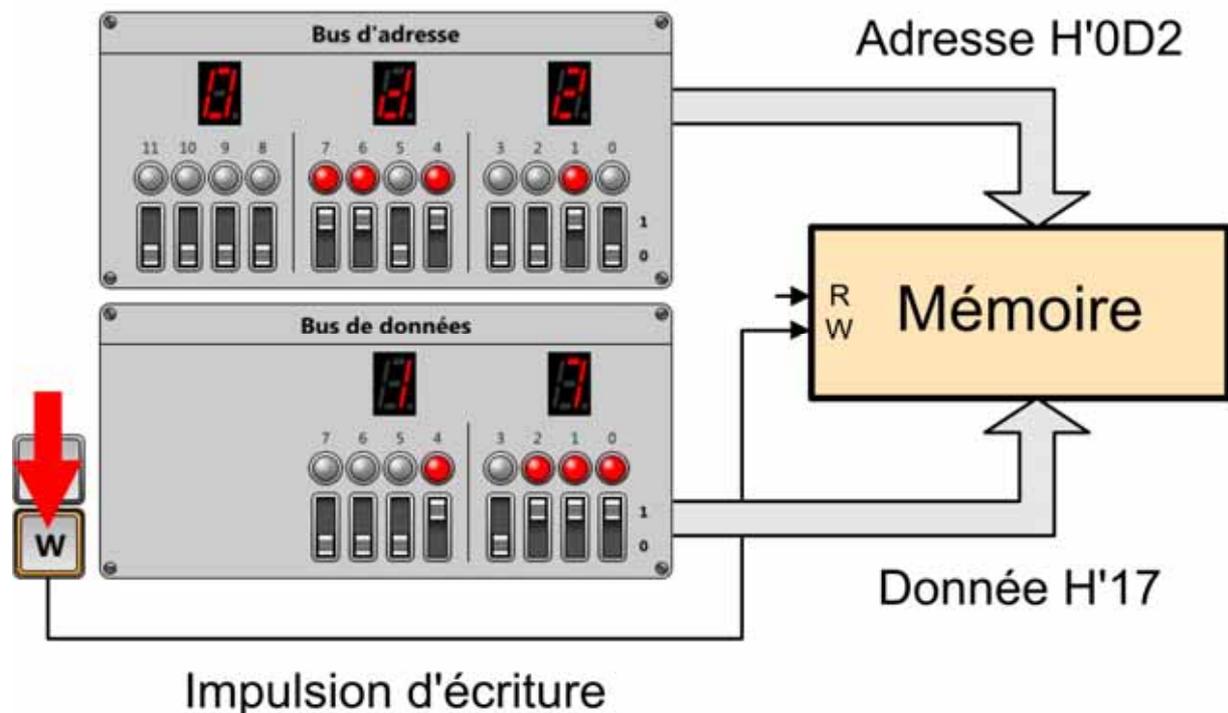
Pour bien comprendre ces notions, rien de tel qu'un exercice pratique avec le simulateur de Dauphin. La mémoire du Dauphin est bien plus grande que la mémoire de 32 bits à 4 tiroirs vue plus haut. Elle comporte 2'048 tiroirs (soit un total de 16'384 bits), sélectionnés grâce à un bus d'adresse de 11 bits.

Pourtant, cette mémoire est ridiculement petite face aux mémoires des ordinateurs actuels, qui atteignent facilement un milliard d'octets (autrement dit mille millions ou 1'000'000'000) !



Ecrivons la valeur H'17 à l'adresse H'0D2, à l'aide du panneau de contrôle :

1. Sélectionnez l'adresse H'0D2 (0000 1101 0010) avec les interrupteurs du bus d'adresse. Les petites lampes et les trois afficheurs montrent cette valeur.
2. Sélectionnez la donnée H'17 (0001 0111) avec les interrupteurs du bus de données. Les petites lampes et les deux afficheurs continuent de montrer H'00; c'est normal, les données ne sont pas encore sur le bus.
3. Pressez sur le bouton poussoir [W] (*write* = écrire) en bas à gauche. Pendant que le bouton est maintenu pressé, les petites lampes et les deux afficheurs montrent la valeur H'17.

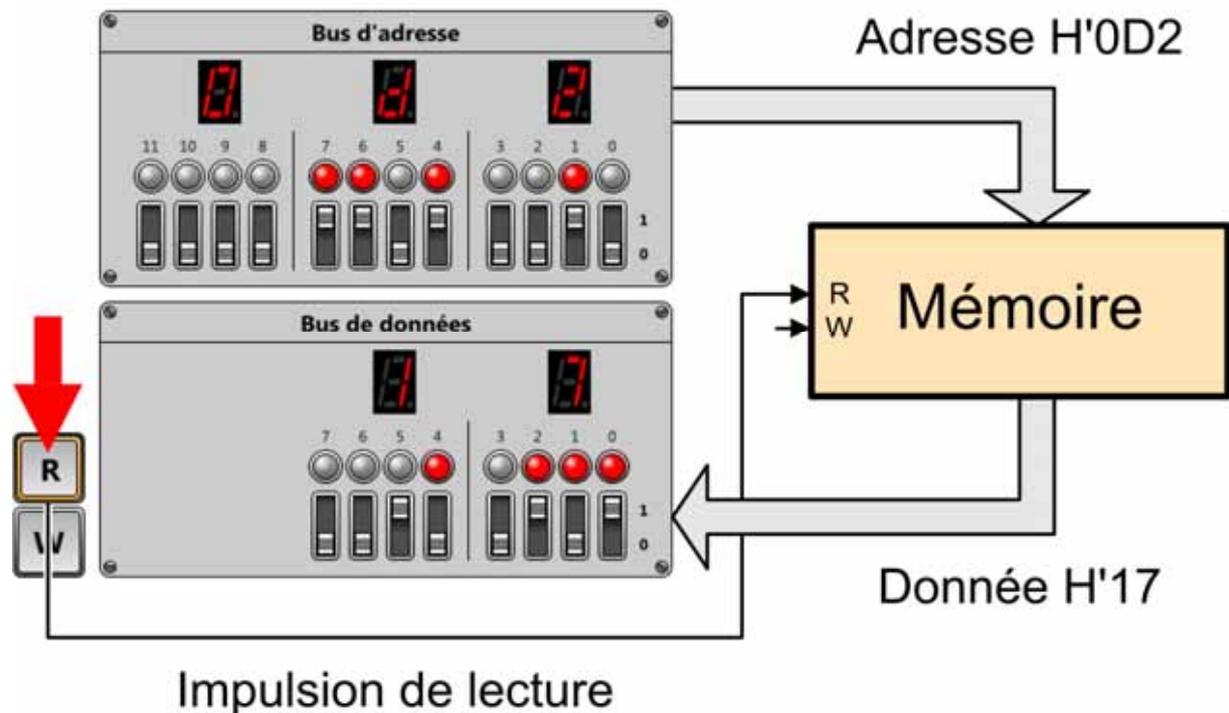


Ecrivons la valeur H'25 à l'adresse H'0D3 :

1. Sélectionnez l'adresse H'0D3 (0000 1101 0011) avec les interrupteurs du bus d'adresse. Il suffit de bouger l'interrupteur du bit 0, tout à droite.
2. Sélectionnez la donnée H'25 (0010 0101) avec les interrupteurs du bus de données.
3. Pressez sur le bouton poussoir [**W**] en bas à gauche.

Lisons la première valeur stockée à l'adresse H'0D2 :

1. Sélectionnez à nouveau l'adresse H'0D2 (0000 1101 0010).
2. Pressez sur le bouton poussoir [**R**] (*read* = lire). Tant qu'il est maintenu pressé, on peut lire la valeur 17 sur le bus de données. Vous pouvez presser [**R**] autant de fois que vous le désirez, pour lire la valeur, qui ne changera pas tant que vous n'écrirez pas une autre valeur.



Lisons la valeur stockée à l'adresse H'0D3 :

1. Sélectionnez l'adresse H'0D3 (0000 1101 0011).
2. Pressez sur le bouton poussoir [**R**]. Tant qu'il est maintenu pressé, on peut lire la valeur H'25 sur le bus de données.

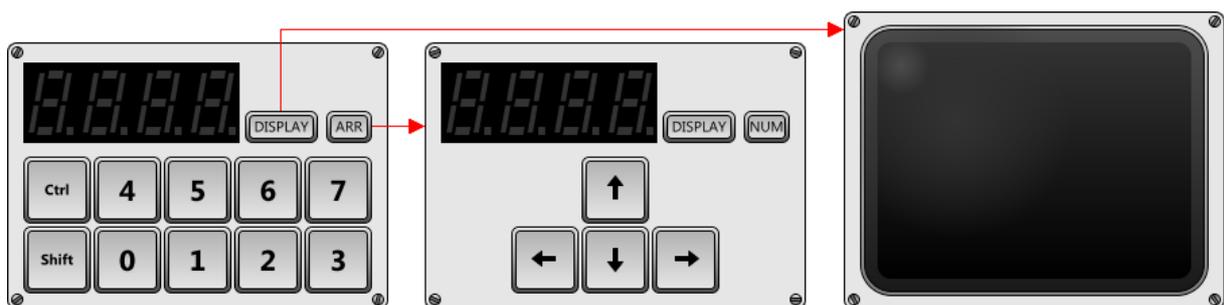
4.1 Les périphériques

Pour interagir avec un ordinateur moderne, vous utilisez principalement un clavier, une souris et un écran. Ce sont des périphériques. Le clavier et la souris sont les moyens dont dispose l'utilisateur pour dire à l'ordinateur ce qu'il doit faire (utilisateur → ordinateur). L'écran est le moyen dont dispose l'ordinateur pour communiquer avec l'utilisateur (ordinateur → utilisateur).



Le Dauphin dispose de trois périphériques rudimentaires pour dialoguer avec l'utilisateur :

- Un clavier de dix **[NUM]** ou quatre **[ARR]** touches (utilisateur → Dauphin).
- Un affichage de quatre valeurs (Dauphin → utilisateur).
- Un écran *bitmap* **[DISPLAY]** de 32 x 24 points (Dauphin → utilisateur).



Ces périphériques sont dans le panneau inférieur droite du simulateur. Si vous cliquez sur les touches, il ne se passe rien; l'affichage reste désespérément muet. C'est tout à fait normal. Au stade actuel, aucun programme ne fonctionne. Ces périphériques sont donc inactifs.

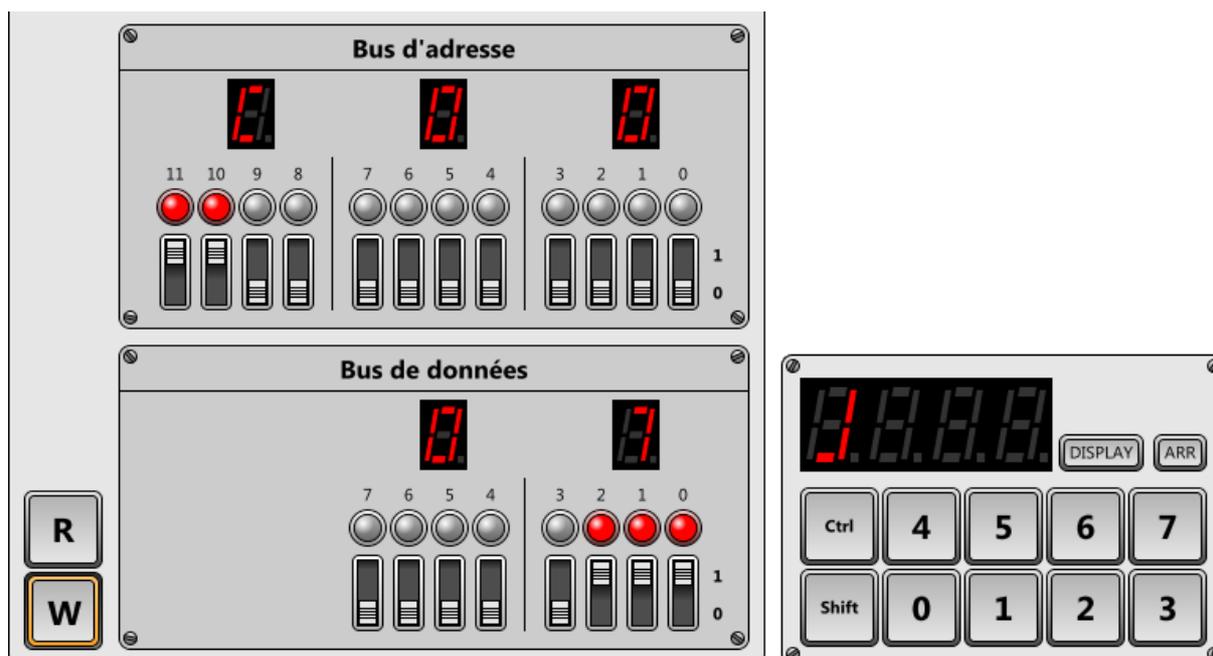
Peut-être aurez-vous remarqué que le dernier interrupteur (bit 11) du bus d'adresse semble inutile, puisque la mémoire répond aux adresses H'000 à H'7FF (0111 1111 1111 en binaire). En fait, il a bien une utilité précise : l'accès à la mémoire morte et aux périphériques.

Adresses	Contenu	Abréviation	Signification
H'000.. H'7FF	Mémoire vive	RAM	<i>Random access memory</i>
H'800.. H'BFF	Mémoire morte	ROM	<i>Read only memory</i>
H'C00.. H'COF	Périphériques	PER	
H'C80.. H'CDF	Ecran bitmap	DIS	<i>Display</i>

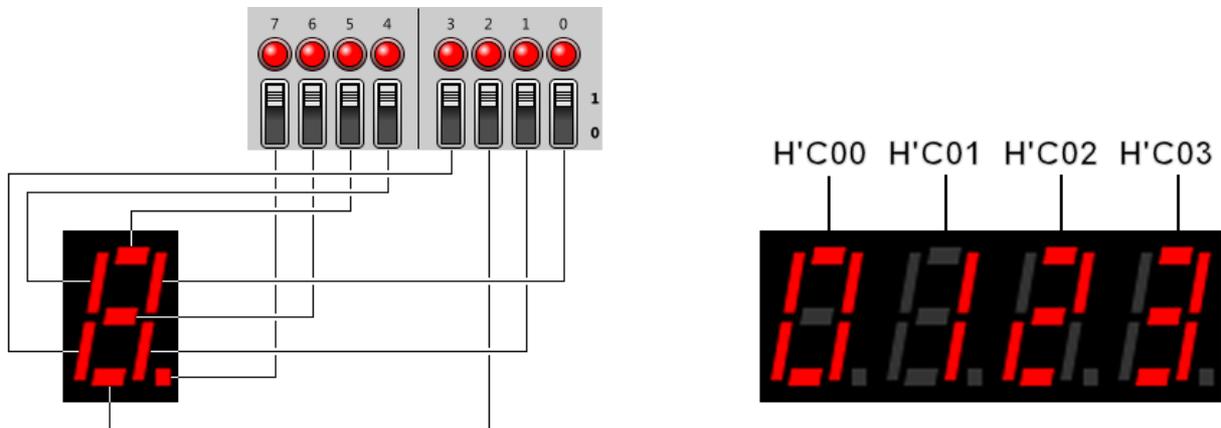
Lorsque nous utilisons les bus pour écrire à une adresse comprise entre H'000 et H'7FF, nous accédons à la mémoire vive (RAM). Pour interagir avec un périphérique, il suffit d'utiliser une adresse comprise entre H'C00 et H'COF. C'est aussi simple que cela.

4.2 L'affichage

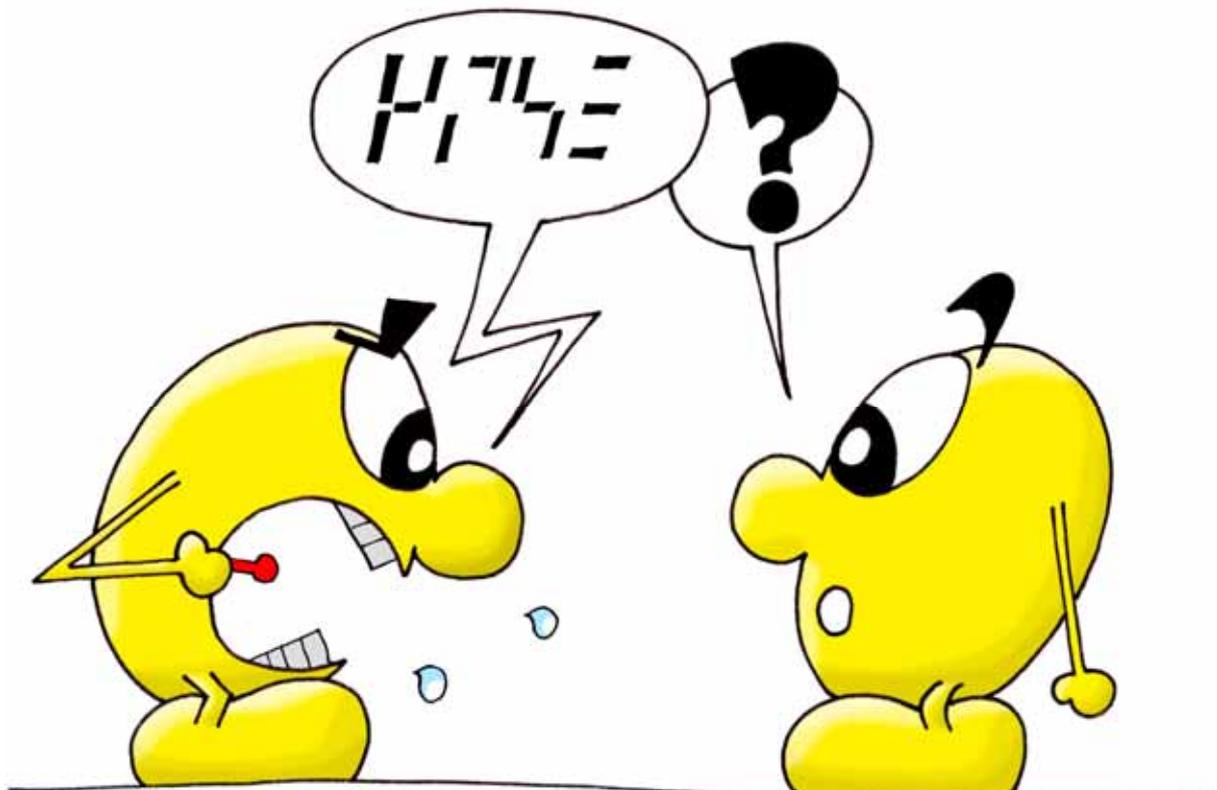
1. Sélectionnez l'adresse H'C00 (1100 0000 0000) avec les interrupteurs du bus d'adresse.
2. Sélectionnez la donnée H'07 (0000 0111) avec les interrupteurs du bus de données.
3. Pressez sur le bouton poussoir [W] (*write* = écrire) en bas à gauche. Une sorte de « J » apparaît sur l'afficheur de gauche.



En fait, chaque bit du bus de données correspond à un segment de l'afficheur. Il est ainsi possible d'afficher toutes sortes de combinaisons étranges qui ne ressemblent à rien. Les autres afficheurs sont accessibles avec les adresses H'C01, H'C02 et H'C03 :

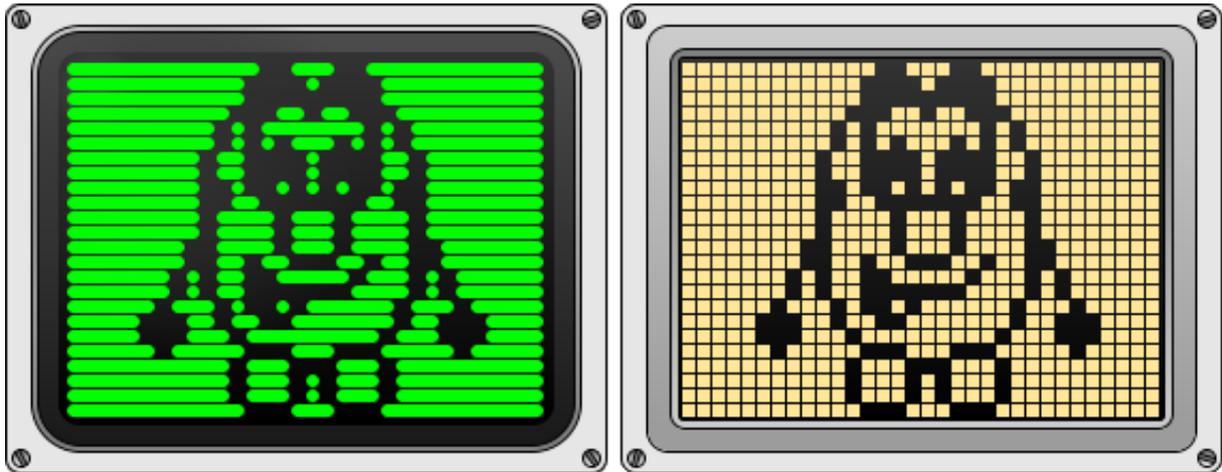


Expérimentez ceci avec différentes adresses et différentes données, en appuyant à chaque fois sur [W]. Vous pouvez ainsi afficher vraiment n'importe quoi :

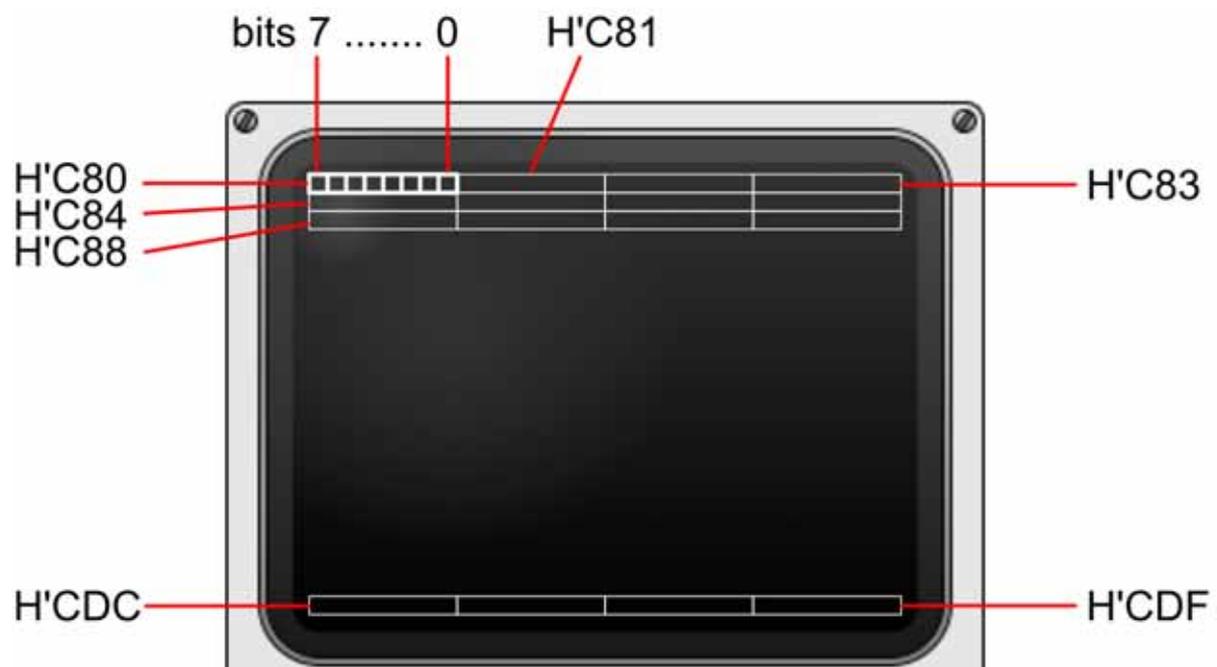


4.3 L'écran bitmap

L'écran bitmap est l'ancêtre des écrans vidéo d'aujourd'hui. Il permet d'afficher 32 x 24 points (*pixels*) monochromes. Pour voir l'écran bitmap, cliquez sur le bouton [**DISPLAY**]. Les quatre afficheurs à sept segments sont alors réduits, mais ils sont toujours fonctionnels. Le bouton [**CLS**] (*CLear Screen*) efface l'écran, si nécessaire. Le bouton [**LCD**] (*Liquid Crystal Display*) change la technologie de l'écran simulé. [**CRT**] (*Cathode Ray Tube*) revient à une simulation de tube cathodique.

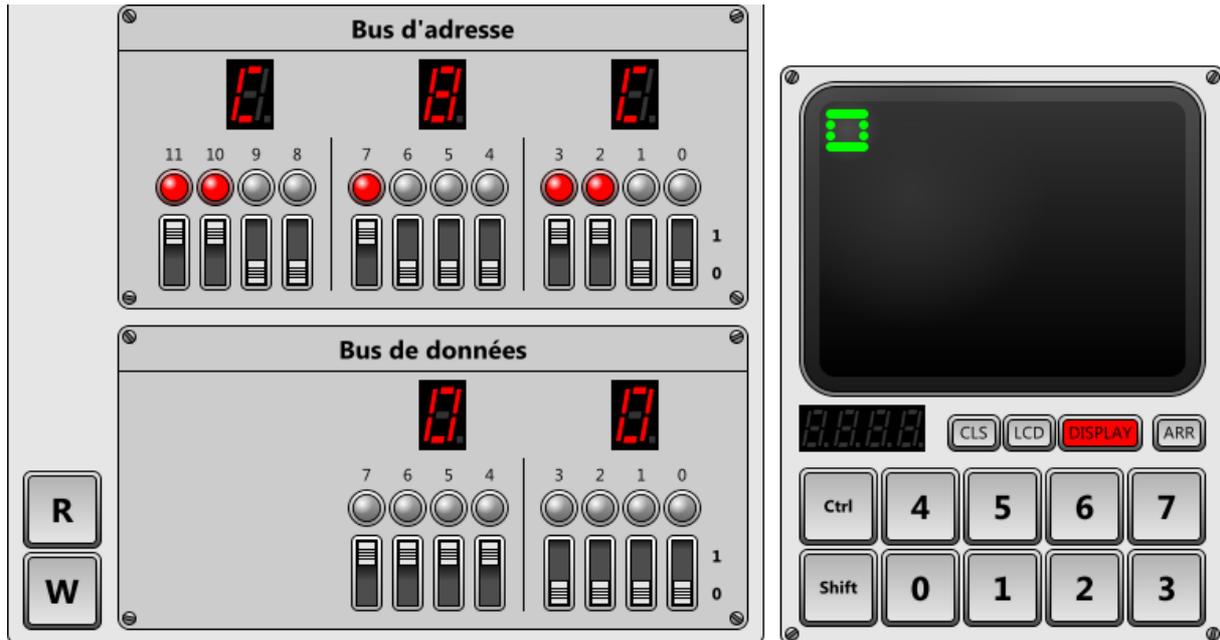


Les adresses H'C80 à H'CDF permettent d'accéder aux 32 x 24 = 768 points de l'écran. Chaque point correspond à un bit. Un octet donne donc accès à huit points. La première adresse H'C80 (1100 1000 0000) correspond à la ligne horizontale de huit points tout en haut à gauche. La deuxième adresse H'C81 correspond aux huit points sur la droite. Pour trouver l'octet en dessous d'un autre, il faut ajouter quatre à l'adresse. Dans une ligne horizontale de huit points, le bit 7 correspond au point de gauche, et le bit 0 à celui de droite.



Le panneau de contrôle permet de s'amuser à allumer des points dans l'écran. Pour dessiner un petit carré en haut à gauche, écrivez les données suivantes, en appuyant à chaque fois sur **[W]** :

Adresse	Donnée
H'C80 (1100 1000 0000)	H'F0 (1111 0000)
H'C84 (1100 1000 0100)	H'90 (1001 0000)
H'C88 (1100 1000 1000)	H'90 (1001 0000)
H'C8C (1100 1000 1100)	H'F0 (1111 0000)



Dessinez d'autres motifs simples pour bien comprendre le système d'adressage des points.

4.4 Le clavier

Le clavier correspond à l'adresse H'C07 (1100 0000 0111). Cela n'a aucun sens (et aucun effet) d'écrire une donnée à cette adresse. On l'utilise uniquement en lecture.

1. Sélectionnez l'adresse H'C07 (1100 0000 0111) avec les interrupteurs du bus d'adresse.
2. Cliquez sur la touche **[6]** du clavier dans le panneau inférieur droite.
3. Pressez sur le bouton poussoir **[R]** (*read* = lire) en bas à gauche. La valeur H'86 (1000 0110) apparaît tant que le bouton est maintenu pressé. Une deuxième pression sur **[R]** affiche H'06. Etrange, non ?

Lorsqu'une touche **[0]** à **[7]** est cliquée, le bit 7 (valeur H'80 ou 1000 000 en binaire) est mis à un à l'adresse H'C07. Si la touche est relâchée avant que quelqu'un ne lise cette adresse, elle y reste mémorisée. Dès que l'adresse H'C07 est lue, le bit 7 est remis à zéro. Cette astuce est nécessaire pour éviter de

perdre une touche pressée brièvement, si on tarde à lire l'adresse H'C07, et également pour éviter de lire deux fois la même touche.

Les touches [**Shift**] et [**Ctrl**] correspondent aux bits 3 et 4. Elles ne sont pas mémorisées et n'activent pas le bit 7. Un clic de la souris enfonce l'une de ces touches (fond rouge). Un nouveau clic la relâche. Pour vous faciliter la vie, la valeur lue à l'adresse H'C07 est affichée entre parenthèses lorsque l'une de ces touches est enfoncée.

Le bouton [**ARR**] transforme le clavier en quatre flèches, très utiles dans certains jeux. Les touches flèches correspondent aux bits 3 à 6. Elles ne sont pas mémorisées et n'activent pas le bit 7.

4.5 La mémoire morte

La mémoire morte (ROM = *read only memory*) répond aux adresses H'800 à H'BFF. On peut y lire des données préinscrites, mais pas les modifier. En d'autres termes, on peut lire (*read*) des données à ces adresses, mais pas les écrire (*write*). L'utilité de la mémoire ROM sera abordée plus loin (voir chapitre 5.10).



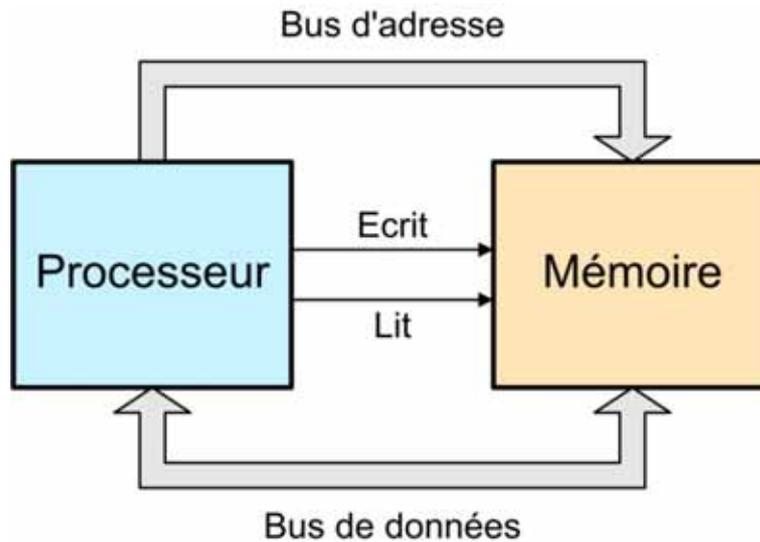
La petite manipulation ci-dessous tente d'écrire une valeur en mémoire ROM :

1. Sélectionnez l'adresse H'800 (1000 0000 0000) avec les interrupteurs du bus d'adresse.
2. Sélectionnez la donnée H'38 (0011 1000) avec les interrupteurs du bus de données.
3. Pressez sur le bouton poussoir [**W**] (*write* = écrire) en bas à gauche. La valeur H'03 (0000 0011), différente de celle que vous tentez d'écrire, apparaît tant que le bouton est maintenu pressé. Cette valeur correspond au contenu de la ROM, que vous ne pouvez pas modifier.

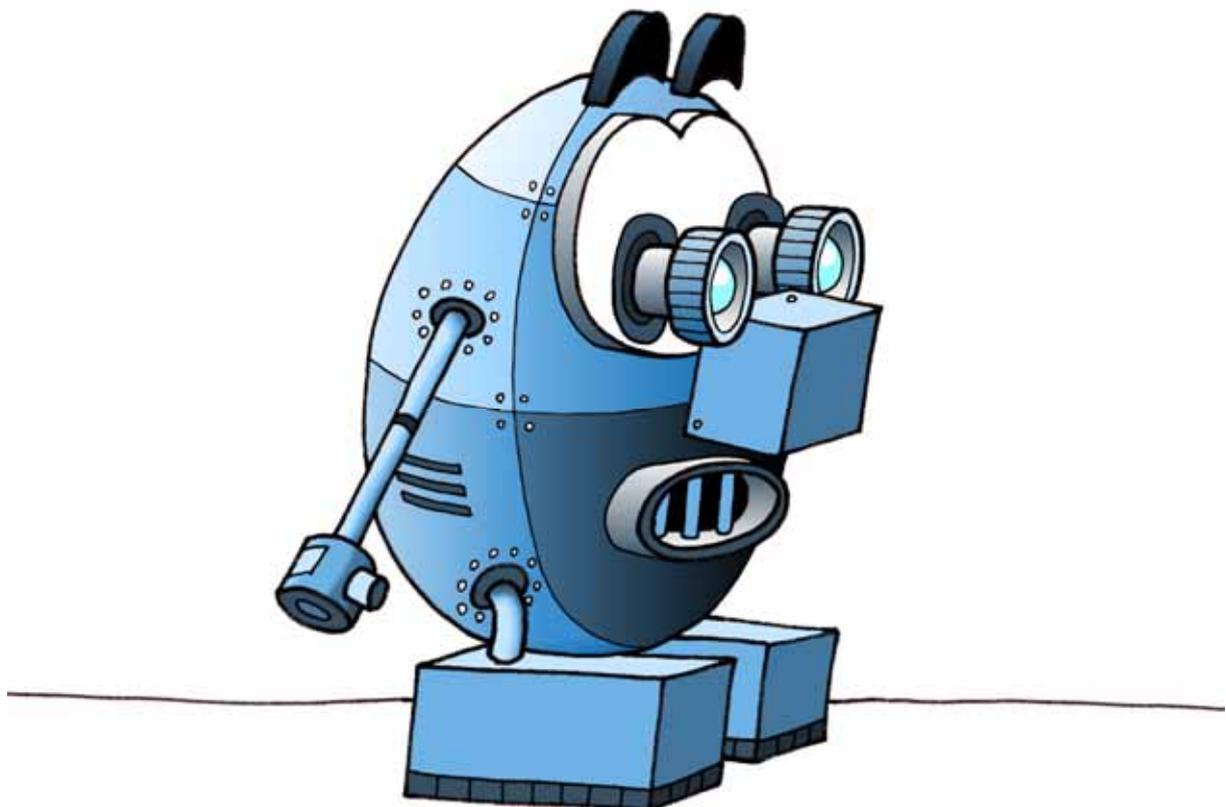
5 Le processeur

Le processeur étudié dans ce manuel n'existe pas dans la réalité. Il s'agit d'un processeur didactique baptisé PSI30.

Au chapitre précédent, nous avons accédé à la mémoire avec le panneau de contrôle. C'est lui qui a pris le contrôle des bus d'adresse et de données. Pendant ces opérations, le processeur était arrêté. Mais lorsque l'ordinateur fonctionne, c'est le processeur qui prend le contrôle des bus.

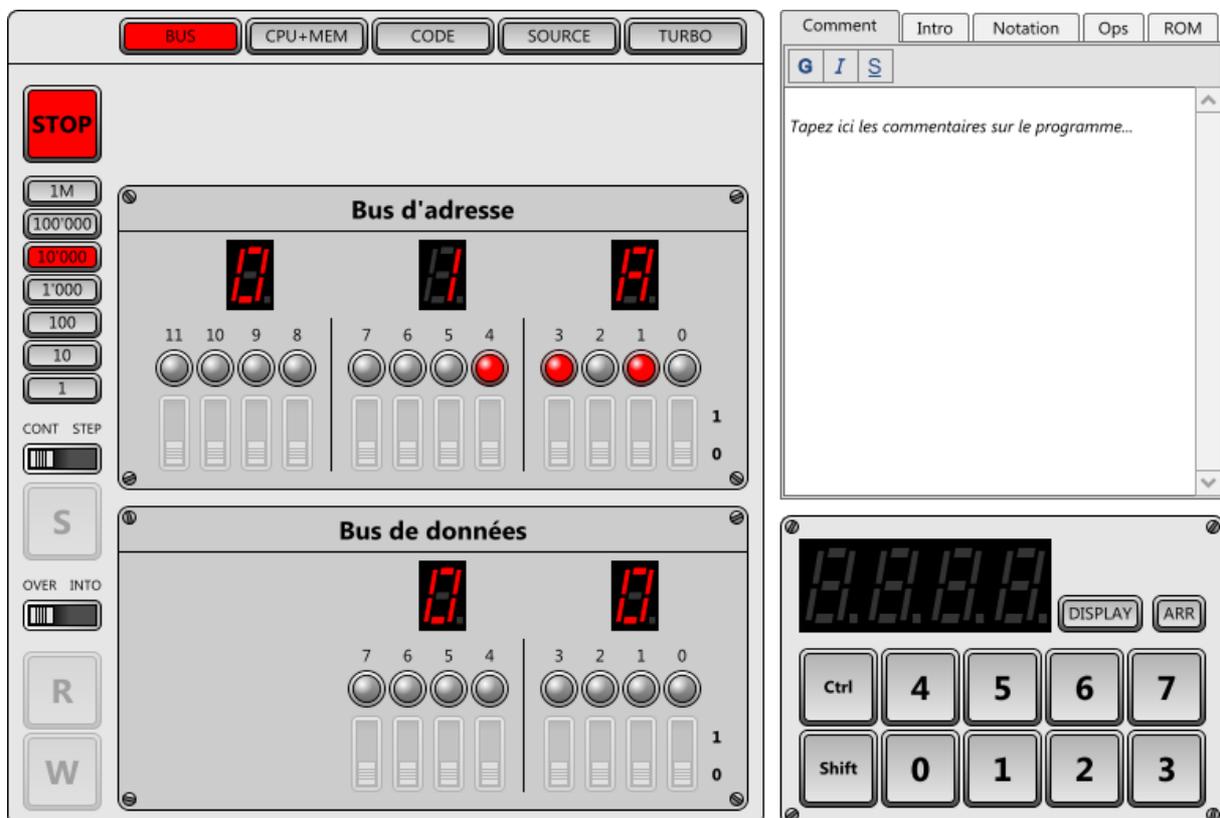


Le processeur est une sorte d'automate sophistiqué.



Heureusement, il est aisé d'en comprendre le principe. L'idéal est de le mettre au travail immédiatement, et d'observer son comportement :

1. Cliquez sur l'icône  « Nouveau », tout en haut à gauche, puis répondez « Non », si nécessaire, pour ne pas enregistrer le programme. Cela efface la mémoire, qui ne contient alors plus que des zéros.
2. Si ce n'est pas déjà fait, basculez le commutateur [**CONT STEP**] sur **CONT** (*continuous*).
3. Enfoncez le bouton [**10**], pour que le processeur ne travaille pas trop vite. Ce bouton signifie que le processeur exécutera 10 instructions par seconde.
4. Cliquez le bouton [**RUN**]. Le bus d'adresse se met à compter (H'001, H'002, H'003, H'004, etc.) et le bus de données ne bouge pas (H'00).
5. Après avoir observé ce comportement, cliquez le bouton [**STOP**] pour stopper le processeur.



Que s'est-il passé ? Il vaut la peine de comprendre ce comportement dans le détail :

1. Lorsque le bouton [**RUN**] est enfoncé, le processeur démarre. Il prend le contrôle des bus d'adresse et de données, pour lire le contenu de la mémoire à l'adresse H'000. C'est toujours la première tâche effectuée au démarrage du processeur.
2. L'information qu'il lit est appelée instruction. Actuellement, tous les bits de la mémoire sont à zéro. Le processeur PSI30 lit donc l'instruction H'00, qui

est un peu spéciale. Cette instruction s'appelle « NOP » (*no operation*) et elle signifie « rien de spécial à faire ».

3. Le processeur lit alors l'instruction suivante, à l'adresse H'001. Il y trouve la même instruction « NOP », et continue donc de parcourir toute la mémoire séquentiellement aux adresses H'002, H'003, H'004, etc.

Bien entendu, ce comportement basique n'a qu'un intérêt didactique. Mais reprenez le principe de base d'un processeur :

1. Lire une instruction en mémoire.
2. Exécuter l'instruction et passer à l'adresse suivante.
3. Recommencer à l'infini.

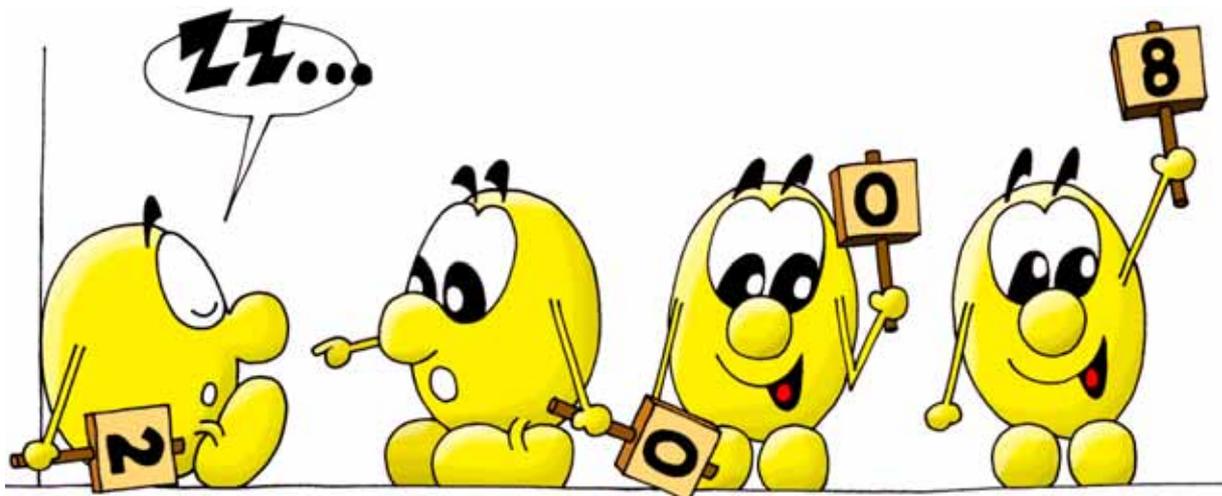
La programmation consiste à remplir la mémoire avec des instructions plus complexes que les « NOP » initiaux. Ce sont ces instructions que le processeur va suivre à la lettre, et qui vont déterminer les tâches effectuées par le programme.

5.1 Tout de suite ?

On remarque qu'un processeur effectue les instructions les unes après les autres. Il est incapable d'effectuer deux choses en même temps. Pour s'en convaincre, effectuez les manipulations suivantes :

1. Cliquez sur l'icône  « Ouvrir » en haut à gauche, et sélectionnez le programme « 2008.dolphin ».
2. Cliquez sur le bouton **[RUN]** pour démarrer le programme.

Ce programme très simple affiche « 2008 », à la fréquence de 100 instructions par seconde, puis stoppe. On voit nettement que les quatre chiffres n'apparaissent pas instantanément, mais de droite à gauche : 8, 0, 0 puis finalement 2. Et pourtant, 100 instructions par seconde semblent déjà une belle vitesse. Cela vous donne une idée de la complexité requise pour effectuer une opération apparemment aussi simple que d'afficher un nombre de quatre chiffres.

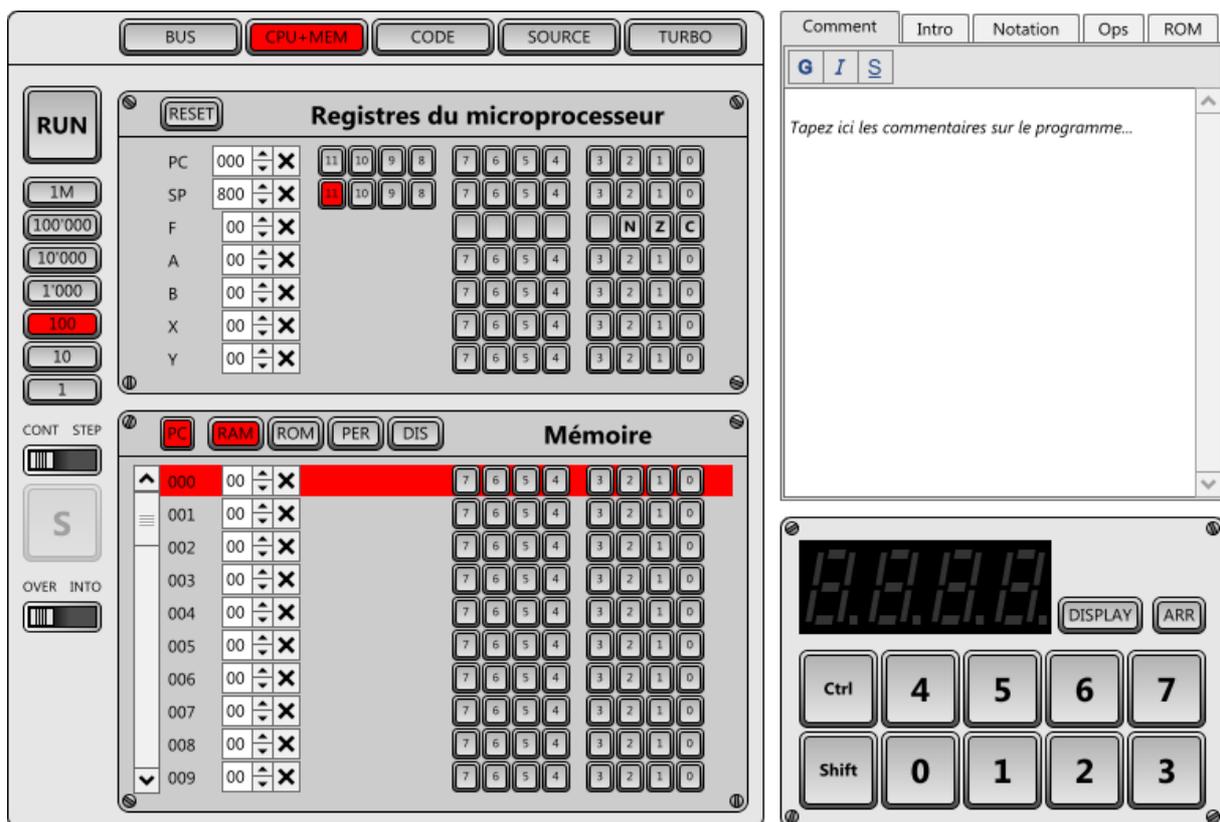


Il peut sembler invraisemblable qu'un ordinateur moderne, qui génère des images en trois dimensions époustouflantes et qui corrige vos fautes de frappe, effectue toutes ces tâches uniquement à l'aide d'instructions rudimentaires. C'est pourtant bien ainsi que fonctionnent tous les ordinateurs. Le secret vient de la très grande quantité d'instructions effectuées chaque seconde : plus de 1'000'000'000 (un milliard, soit mille millions) n'a rien d'exceptionnel aujourd'hui !

5.2 Dans les entrailles



Pour la suite des exercices, cliquez sur le bouton **[CPU+MEM]**. Ceci modifie les panneaux de contrôle, qui montrent maintenant l'intérieur des deux composants principaux d'un ordinateur : le processeur (*Central Processing Unit*) et la mémoire. Il y a 30 ans, cela n'était hélas pas possible à réaliser avec un véritable Dauphin !



Le panneau du processeur PSI30 montre que ce dernier possède sept registres (nommés PC, SP, F, A, B, X et Y). Certains registres contiennent des valeurs de 12 bits, et d'autres de 8 bits seulement. Chaque registre a une tâche bien précise. Par exemple, le premier registre nommé **PC** (*program counter*) détermine l'adresse en mémoire de la prochaine instruction à lire. On l'appelle parfois « pointeur d'instruction ».



De gauche à droite, vous trouvez :

- **PC** : Le nom du registre.
- **029** : La valeur hexadécimale (H'029) contenue actuellement dans le registre. Vous pouvez éditer cette valeur.
- Deux petits triangles pour ajouter ou soustraire un au contenu.
- **X** : Une croix pour remettre le registre à zéro.
- **11..0** : La représentation du registre en binaire. Les cases rouges correspondent aux bits à un. Vous pouvez cliquer sur ces petits boutons pour changer l'état d'un bit.

5.3 Le saut

Vous avez déjà compris le fonctionnement de l'instruction « NOP ». Nous allons maintenant étudier une première instruction véritablement utile, l'instruction de saut (*jump*). On parle parfois aussi d'instruction de branchement (*branch*).



Certaines instructions simples telles que « NOP » sont codées avec un seul octet. D'autres instructions plus complexes nécessitent deux, trois ou quatre octets. L'instruction « JUMP » demande trois octets.

5.4 Premier saut dans l'inconnu

L'instruction « JUMP » a la valeur H'10. Elle est suivie de deux octets qui déterminent l'adresse à laquelle doit se poursuivre le déroulement du

programme. Par exemple, si vous désirez poursuivre l'exécution à l'adresse H'3A0, vous devez écrire une instruction de trois octets : H'10 H'03 H'A0.

Ecrivez les valeurs H'10, H'03 et H'A0 à partir de l'adresse H'002 :

1. Cliquez sur l'icône  « Nouveau », tout en haut à gauche, puis répondez « Non », si nécessaire, pour ne pas enregistrer le programme.
2. Dans le panneau de la mémoire, double-cliquez dans la valeur à l'adresse H'002 (actuellement H'00).
3. Tapez 10, puis sur la touche Entrée (*Enter*).
4. Le curseur s'est déplacé à l'adresse suivante. Tapez 3 (il n'est pas nécessaire de taper 03) puis Entrée.
5. Tapez A0 puis Entrée.



Pour pouvoir observer calmement ce premier programme rudimentaire, nous allons l'exécuter en « pas à pas » (*step by step*). Cela signifie que le processeur ne va chercher et exécuter la prochaine instruction que lorsque vous appuyez sur le bouton [**S**] (*step*). Pour passer dans ce mode, basculez le commutateur [**CONT STEP**] (*continuous ou step by step*) sur **STEP**.

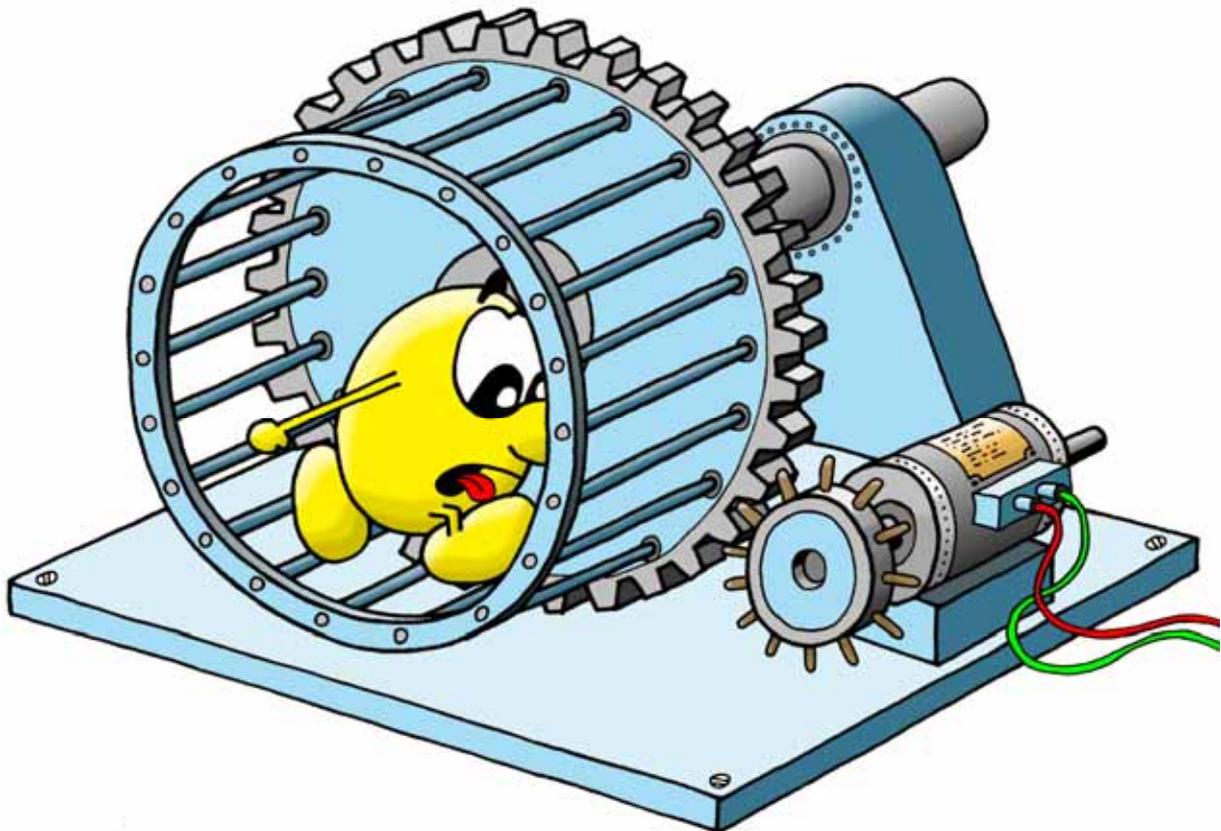
1. Cliquez [**RUN**]. Le processeur démarre à l'adresse H'000, mais s'arrête aussitôt. Le registre PC montre la valeur H'000.
2. Cliquez le bouton [**S**]. L'instruction H'00 « NOP » contenue à l'adresse H'000 est lue et exécutée. Le registre PC contient maintenant H'001, qui correspond à l'adresse de la prochaine instruction à exécuter. C'est maintenant la ligne 001 du panneau inférieur de la mémoire qui est en rouge. La ligne rouge correspond toujours à la prochaine instruction qui sera exécutée, autrement dit à l'adresse pointée par le registre PC.
3. Cliquez le bouton [**S**]. Une deuxième instruction « NOP » est exécutée, et le registre PC vaut H'002. L'instruction de saut H'10 H'03 H'A0 est prête à s'exécuter.
4. Cliquez le bouton [**S**]. Les trois octets de l'instruction « JUMP » ont été lus, et le registre PC contient maintenant H'3A0. L'exécution se poursuivra désormais à partir de cette adresse.
5. Cliquez le bouton [**S**]. L'instruction « NOP » de l'adresse H'3A0 s'est exécutée, et le registre PC vaut H'3A1.

Il est peu commode d'écrire un programme sous forme d'instructions codées en hexadécimal. Dès que le programme se complexifie, il devient très difficile de s'y retrouver et de déceler les erreurs. C'est la raison pour laquelle il faut prendre

l'habitude d'écrire son programme sous une autre forme, sur une simple feuille de papier. De cette façon, notre programme s'écrirait ainsi :

Adr.	Donnée	Instruction	Commentaire
000	00	NOP	Ne fait rien
001	00	NOP	Ne fait rien
002	10	JUMP H'3A0	Saute à l'adresse H'3A0
003	03		
004	A0		
Nom du programme : jump1.dolphin			

5.5 Mouvement perpétuel



Nous allons expérimenter un deuxième programme très simple, qu'on pourrait appeler « mouvement perpétuel » :

Adr.	Donnée	Instruction	Commentaire
000	00	NOP	Ne fait rien
001	00	NOP	Ne fait rien
002	00	NOP	Ne fait rien
003	00	NOP	Ne fait rien
004	10	JUMP H'000	Saute à l'adresse H'000
005	00		
006	00		
Nom du programme : jump2.dolphin			

1. Cliquez sur l'icône  « Nouveau », tout en haut à gauche, puis répondez « Non », si nécessaire, pour ne pas enregistrer le programme.
2. Basculez le commutateur [**CONT STEP**] sur **CONT** (*continuous*).
3. Enfoncez le bouton [**10**], pour que le processeur ne travaille pas trop vite.
4. Enfoncez le bouton [**RUN**]. Les cinq instructions du programme s'exécutent en boucle, et le registre PC affiche successivement H'000, H'001, H'002, H'003 et H'004 puis recommence à l'infini.
5. Pendant l'exécution du programme, vous pouvez basculer le commutateur [**CONT STEP**] sur **STEP**, puis appuyer autant de fois que vous le désirez sur [**S**] pour exécuter le programme en pas à pas.
6. Vous pouvez également modifier le registre PC, pour décider quelle sera la prochaine instruction exécutée en appuyant sur [**S**].
7. Lorsque vous avez terminé, appuyez sur [**STOP**] pour stopper le programme.

Ce genre de comportement est souvent désigné par le terme de « boucle infinie ». Il n'est généralement pas souhaité, puisque l'ordinateur ne fera plus rien et semblera mort !

5.6 L'addition s'il vous plait



En plus du registre PC, le processeur PSI30 contient d'autres registres. Les registres A, B, X et Y permettent de manipuler des valeurs (les registres SP et F ne seront qu'effleurés dans ce manuel). Voici quelques opérations possibles avec ces registres :

- Initialiser un registre avec une valeur fixe (ce que l'on appelle une constante).
- Copier le contenu d'un octet en mémoire dans un registre.

- Copier le contenu d'un registre dans un octet en mémoire.
- Copier un registre dans un autre.
- Additionner ou soustraire des registres entre eux.
- Effectuer des opérations binaires (et, ou, ou exclusif, rotations, etc.).

L'exemple ci-dessous montre une utilisation possible des registres A et B :

Adr.	Donnée	Instruction	Commentaire
000	50	MOVE #H'12, A	Copie la valeur H'12 dans le registre A
001	12		
002	51	MOVE #H'5, B	Copie la valeur H'5 dans le registre B
003	05		
004	84	ADD B, A	Ajoute B au contenu de A
005	10	JUMP H'004	Saute à l'adresse H'004
006	00		
007	04		
Nom du programme : addhexa.dolphin			

Exécutez ce programme en mode « pas à pas » (commutateur [**CONT STEP**] sur **STEP**). Vous observerez le registre A prendre successivement les valeurs H'12, H'17, H'1C, H'21, H'26, etc.

Pendant le fonctionnement du programme, vous avez tout loisir de modifier le contenu d'un registre. Par exemple, essayez de modifier le contenu de B pour additionner une autre valeur. Il est également très utile de modifier le registre PC, pour continuer l'exécution à une autre adresse.

5.7 Silence, on tourne

Evidemment, le processeur peut agir avec les périphériques, clavier et afficheurs. Il doit simplement utiliser des adresses comprises entre H'C00 et H'COF.

Le petit programme ci-dessous fait tourner un segment sur l'afficheur de gauche, dans le sens des aiguilles d'une montre :

Adr.	Donnée	Instruction	Commentaire
000	50	MOVE #H'1, A	Copie la valeur 1 dans le registre A (premier segment vertical supérieur droite)
001	01		
002	58	MOVE A, H'C00	Copie la valeur dans A à l'adresse H'C00, c'est-à-dire dans l'afficheur de gauche
003	0C		
004	00		
005	30	RL A	Décale A d'un bit vers la gauche
006	10	JUMP H'002	Saute à l'adresse H'002
007	00		
008	02		
Nom du programme : shift1.dolphin			

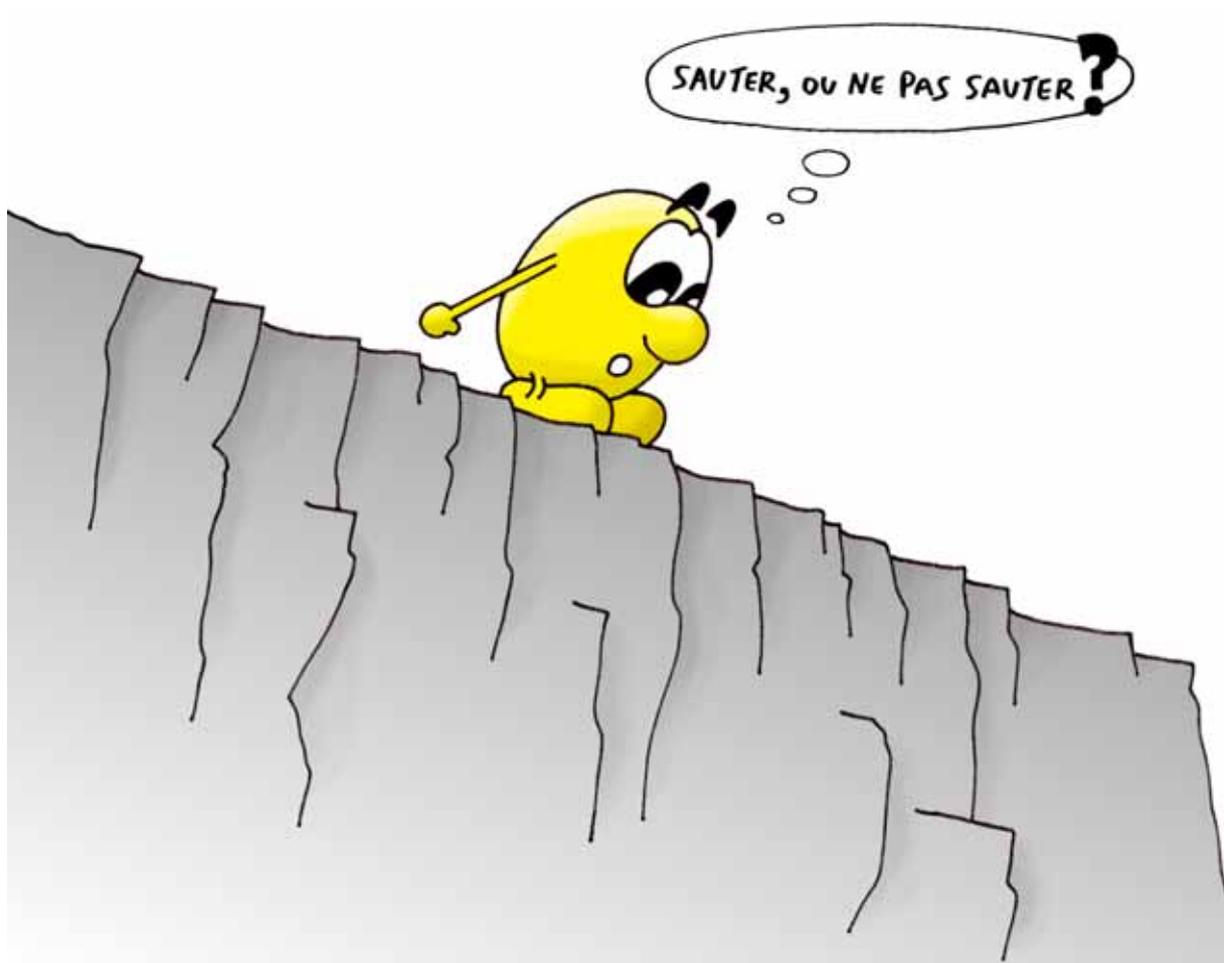
Exécutez ce programme en mode continu (commutateur [**CONT STEP**] sur **CONT**), à 10 instructions par seconde [**10**]. Il est intéressant d'observer la valeur binaire du registre A et de voir le bit se décaler vers la gauche :



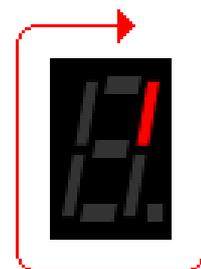
L'instruction « RL » (*rotate left*) décale tous les bits d'une position vers la gauche. Le 7^{ème} et dernier bit de gauche revient dans le premier bit de droite. Ainsi, le registre A prendra les valeurs H'01, H'02, H'04, H'08, H'10, H'20, H'40, H'80, H'01, H'02, etc.

5.8 Sauter ou ne pas sauter, telle est la question

Ce chapitre explique le fonctionnement d'une instruction essentielle, le saut conditionnel. Elle va permettre au programme d'effectuer une chose ou une autre, en fonction du résultat d'un test.



Le programme du chapitre précédent n'est pas très joli. Il serait plus esthétique de ne pas allumer le segment horizontal du milieu, ni le point décimal.



Pour cela, il faut ajouter deux nouvelles instructions, aux adresses H'006 et H'008 :

Adr.	Donnée	Instruction	Commentaire
000	50	MOVE #H'1, A	Copie la valeur 1 dans le registre A
001	01		
002	58	MOVE A, H'C00	Ecrit la valeur A dans l'afficheur de gauche
003	0C		
004	00		
005	30	RL A	Décale A d'un bit vers la gauche
006	74	AND #H'3F, A	Efface les bits 6 et 7 (H'3F vaut 0011 1111 en binaire)
007	3F		
008	12	JUMP,ZS H'000	Saute à l'adresse H'000 si le résultat de l'instruction précédente était nul (si A contient H'00)
009	00		
00A	00		
00B	10	JUMP H'002	Saute à l'adresse H'002
00C	00		
00D	02		
Nom du programme : shift2.dolphin			

Le principe est le suivant :

1. On décale A d'un bit vers la gauche (RL A).
2. On remet à zéro les deux derniers bits (6 et 7), correspondant au segment du milieu et au point décimal (AND #H'3F, A).
3. Si tous les bits restants sont à zéro, on saute à l'adresse H'000, pour remettre le premier bit dans A (valeur H'01).
4. Dans le cas contraire, on saute à l'adresse H'002, comme d'habitude, pour continuer de décaler A vers la gauche.

L'opération 2. se résout avec une instruction logique appelée « AND ». On ne conserve que les bits qui sont à un, à la fois dans le registre A et dans l'opérande (ici la valeur H'3F à l'équivalent binaire 0011 1111 à l'adresse H'007). De plus, l'instruction « AND » met à un le bit Z du registre F (fanions, *flags*) si le résultat est nul. Sinon, elle met ce bit à zéro.

L'opération 3. est un saut conditionnel, ce qui signifie qu'on saute parfois, et parfois pas... En fait, le saut « JUMP,ZS » (*Zero Set*) ne s'effectue que si le bit Z du registre F est à un. Dans le cas contraire, c'est l'instruction suivante qui s'exécute, ici l'instruction à l'adresse H'00B (un saut à l'adresse H'002).

5.9 Lecture du clavier

Le petit programme ci-dessous attend que l'utilisateur clique sur une touche [0] à [7], puis montre le résultat sur l'afficheur de gauche :

Adr.	Donnée	Instruction	Commentaire
000	54	MOVE H'C07, A	Lecture du clavier dans A
001	0C		
002	07		

003	D4	TCLR	A: #7	Teste puis efface le bit 7
004	07			
005	12	JUMP,EQ	H'000	Si le bit était à zéro (donc qu'aucune touche n'était pressée), saute au début
006	00			
007	00			
008	58	MOVE	A, H'C00	Ecrit la valeur lue dans l'afficheur de gauche
009	0C			
00A	00			
00B	10	JUMP	H'000	Revient au début
00C	00			
00D	00			
Nom du programme : key1.dolphin				

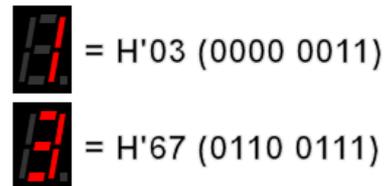
1. Exécutez ce programme en mode continu (commutateur [**CONT STEP**] sur **CONT**) en appuyant sur [**RUN**].
2. Si vous cliquez sur [**0**], rien ne s'affiche.
3. Si vous cliquez sur [**3**], la valeur « 1 » s'affiche.
4. Si vous cliquez sur [**5**], deux segments affichent un résultat qui ne ressemble à rien.
5. Si vous cliquez sur [**7**], une sorte de « J » s'affiche.



Voilà qui semble bien étrange. Pourtant, notre programme fonctionne à la perfection. Lorsque vous appuyez sur la touche [**3**], la valeur H'83 est lue. L'instruction « TCLR » teste puis efface le bit 7. La valeur devient donc H'03. Cette valeur est ensuite transférée dans le premier afficheur (adresse H'C00). Il

faut comprendre que cette valeur H'03 a l'équivalent binaire 0000 0011. Elle signifie donc « allumer les segments correspondant aux bits 0 et 1 », c'est-à-dire les deux segments verticaux de droite, qui semblent ainsi former le chiffre « 1 ».

Pour afficher le chiffre 3, il faudrait allumer cinq segments avec la valeur H'67 (0110 0111 en binaire).



Pour afficher une valeur ayant la même apparence que la touche cliquée, il faut compiler notre programme :

Adr.	Donnée	Instruction	Commentaire
000	54	MOVE H'C07, A	Lecture du clavier dans A
001	0C		
002	07		
003	D4	TCLR A: #7	Teste puis efface le bit 7
004	07		
005	12	JUMP,EQ H'000	Si le bit était à zéro (donc qu'aucune touche n'était pressée), saute au début
006	00		
007	00		
008	74	AND #H'07, A	Masque les bits inutiles
009	07		
00A	42	MOVE A, X	Copie A dans X
00B	54	MOVE H'016+{X}, A	Copie la valeur contenue à la Xème position de la table à l'adresse H'016 dans A
00C	10		
00D	16		
00E	58	MOVE A, H'C00	Copie la valeur lue dans la table dans l'afficheur de gauche
00F	0C		
010	00		
011	10	JUMP H'000	Revient au début
012	00		
013	00		
014	FF	TABLE #8	Signale une table de 8 octets
015	08		
016	3F	BYTE #H'3F	Digits pour le chiffre 0 (0011 1111)
017	03	BYTE #H'03	Digits pour le chiffre 1 (0000 0011)
018	6D	BYTE #H'6D	Digits pour le chiffre 2 (0110 1101)
019	67	BYTE #H'67	Digits pour le chiffre 3 (0110 0111)
01A	53	BYTE #H'53	Digits pour le chiffre 4 (0101 0011)
01B	76	BYTE #H'76	Digits pour le chiffre 5 (0111 0110)
01C	7E	BYTE #H'7E	Digits pour le chiffre 6 (0111 1110)
01D	23	BYTE #H'23	Digits pour le chiffre 7 (0010 0011)
Nom du programme : key2.dolphin			

Les adresses H'016 à H'01D contiennent une table de huit valeurs, qui donnent les combinaisons exactes de bits pour allumer les segments permettant de représenter les chiffres 0 à 7.

5.10 Routines en mémoire morte

Tous les exemples précédents montrent qu'il est complexe d'effectuer des tâches apparemment très simples. C'est pourquoi les tâches universelles comme attendre que l'utilisateur appuie sur une touche, ou afficher une valeur, sont écrites une bonne fois pour toutes, sous forme de routines. Votre programme peut faire appel à ces routines, qui sont programmées dans la mémoire morte (ROM).

Le programme ci-dessous fait l'écho des touches pressées sur les afficheurs successifs, de gauche à droite.

Un appel à une routine de la mémoire ROM s'effectue avec l'instruction « CALL », qui vaut H'01. Elle est suivie de deux octets qui correspondent à l'adresse de la routine. Par exemple, `_WaitKey` est à l'adresse H'803. Un appel à cette routine s'écrit donc H'01, H'08, H'03.

Adr.	Donnée	Instruction	Commentaire
000	51	MOVE #H'00, B	Commence sur le digit de gauche
000	00		
002	01	CALL H'803	<code>_WaitKey</code> Attente et lecture du clavier dans A
003	08		
004	03		
005	01	CALL H'80C	<code>_DisplayHexaDigit</code> Affiche le digit contenu dans A sur l'afficheur déterminé par B
006	08		
007	0C		
008	29	INC B	Passes à l'afficheur suivant (+1)
009	10	JUMP H'002	Revient au début
00A	00		
00B	02		
Nom du programme : rom1.dolphin			

La documentation des routines peut être consultée à l'aide du panneau supérieur droite, onglet ROM. On y apprend que `_DisplayHexaDigit` affiche la valeur contenue dans le registre A sur l'afficheur désigné par le registre B.

L'instruction « INC » augmente de 1 le contenu de B. Cette opération s'appelle « incrémenter ». Ainsi, chaque pression sur une touche utilise l'afficheur suivant, de gauche à droite.

Le registre B doit contenir une valeur comprise entre 0 et 3. Après quatre touches pressées, B vaudra donc 4, ce qui devrait causer un problème, puisqu'il n'existe pas de cinquième afficheur. Mais heureusement, `_DisplayHexaDigit` est suffisamment malin pour considérer l'afficheur 4 comme le premier, l'afficheur 5 comme le deuxième, etc. C'est l'un des avantages d'utiliser des routines; on peut les écrire très soigneusement, pour résoudre un maximum de cas tordus.

Lorsque vous exécutez un programme en mode « pas à pas », l'appel d'une routine par l'instruction « CALL » exécute toute la routine d'un coup. Si vous

souhaitez entrer dans la routine, il faut placer le commutateur [**OVER INTO**] sur **INTO** (*over* signifie « par-dessus » et *into* signifie « dans »).

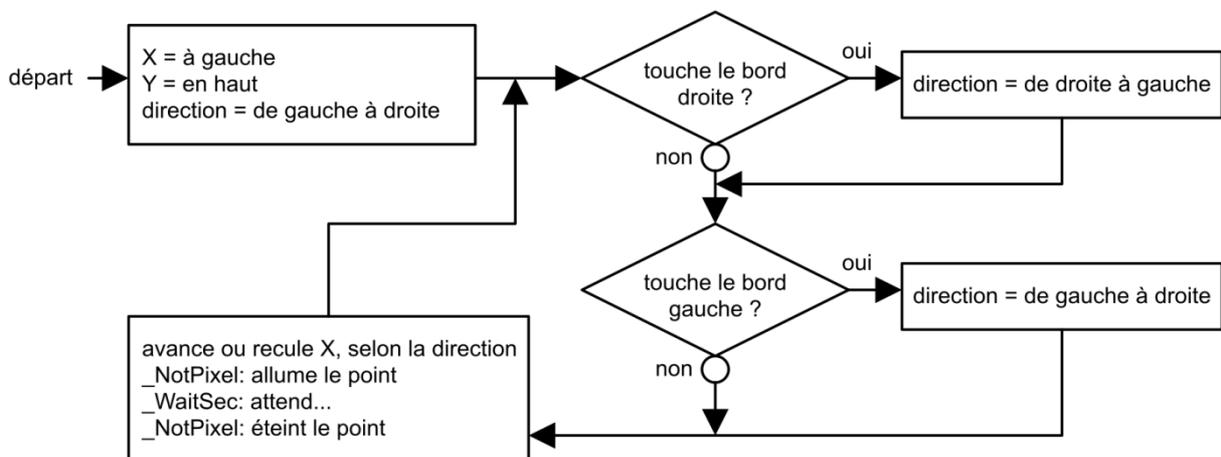
5.11 Codage manuel ou automatique



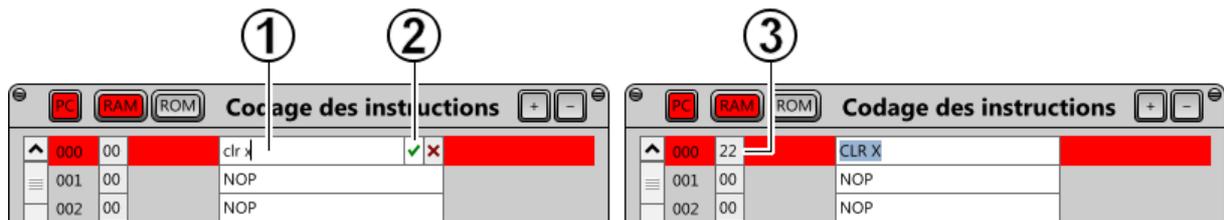
Le codage manuel des instructions vu jusqu'à présent était réellement utilisé avec les premiers systèmes à microprocesseurs. C'est un travail fastidieux et source de nombreuses erreurs. Heureusement, le bouton [**CODE**] permet de s'affranchir de cette étape.



Ecrivons un programme qui fait rebondir une petite balle sur les bords gauche et droite de l'écran bitmap. La balle fera un mouvement horizontal de va-et-vient.



Cliquez sur l'icône  « Nouveau », tout en haut à gauche, puis répondez « Non », si nécessaire, pour ne pas enregistrer le programme.



1. Double-cliquez dans l'instruction « NOP » à l'adresse H'000, puis tapez « clr x ». L'emploi de majuscules ou de minuscules n'a pas d'importance.
2. Cliquez le petit « vu » vert.
3. L'instruction est codée automatiquement s'il n'y a pas de faute de frappe. Elle occupe entre un et quatre octets. Dans cet exemple, elle occupe un octet (H'22) à l'adresse H'000.

Procédez de même pour chaque instruction de la 3^{ème} colonne, puis validez avec la touche Entrée pour passer à la ligne suivante :

Adr.	Donnée	Instruction	Commentaire
000	22	CLR X	Position horizontale (à gauche)
001	23	CLR Y	Position verticale (en haut)
002	51 01	MOVE #H'01, B	Direction (de gauche à droite)
004	72 1F	COMP #H'1F, X	Touche le bord droite ?
006	14 00 0B	JUMP,LO H'00B	Non -> saute à l'adresse H'00B
009	51 FF	MOVE #H'FF, B	Nouvelle direction (de droite à gauche)
00B	72 00	COMP #H'00, X	Touche le bord gauche ?
00D	17 00 12	JUMP,HI H'012	Non -> saute à l'adresse H'012
010	51 01	MOVE #H'01, B	Nouvelle direction (de gauche à droite)
012	86	ADD B, X	Avance ou recule X horizontalement
013	01 08 1B	CALL H'81B	_NotPixel allume le nouveau point
016	50 0A	MOVE #H'0A, A	Copie la durée à attendre dans A
018	01 08 06	CALL H'806	_WaitSec attend...
01B	01 08 1B	CALL H'81B	_NotPixel éteint l'ancien point
01E	10 00 04	JUMP H'004	Recommence

Nom du programme : rebond1.dolphin

Ce programme est basé sur la routine H'81B _NotPixel qui inverse sur l'écran bitmap le point dont on donne les coordonnées x;y dans les registres de mêmes noms X et Y. Comme l'écran bitmap mesure 32 x 24 points, le registre X doit varier entre 0 et 31 (H'00 et H'1F).

La direction est stockée dans le registre B. Elle vaut 1 (H'01) de gauche à droite ou -1 (H'FF) de droite à gauche. L'instruction « ADD B, X » augmente ou diminue de un le contenu de X. Au départ, X prend les valeurs 0, 1, 2, ..., 30 et 31. A ce moment, la balle touche le bord droite. La direction est alors inversée en mettant la valeur -1 dans B. Les prochaines valeurs de X seront donc 30, 29, 28, etc.

adresses H'00B, H'012 et H'004 utilisées pour les sauts ne sont plus correctes, ce qui est mis en évidence par les points d'exclamation sur fond jaune.

Pour que le programme fonctionne à nouveau, il faut corriger manuellement les adresses des sauts :

1. Cliquez sur l'instruction « JUMP,LO H'00B » à l'adresse H'007, et corrigez-la en « JUMP,LO H'**00C** ».
2. Cliquez sur l'instruction « JUMP,HI H'012 » à l'adresse H'00E, et corrigez-la en « JUMP,HI H'**013** ».
3. Cliquez sur l'instruction « JUMP H'004 » à l'adresse H'01F, et corrigez-la en « JUMP H'**005** ».

Le programme devrait maintenant fonctionner à la perfection, avec le petit point dessiné au milieu de l'écran.

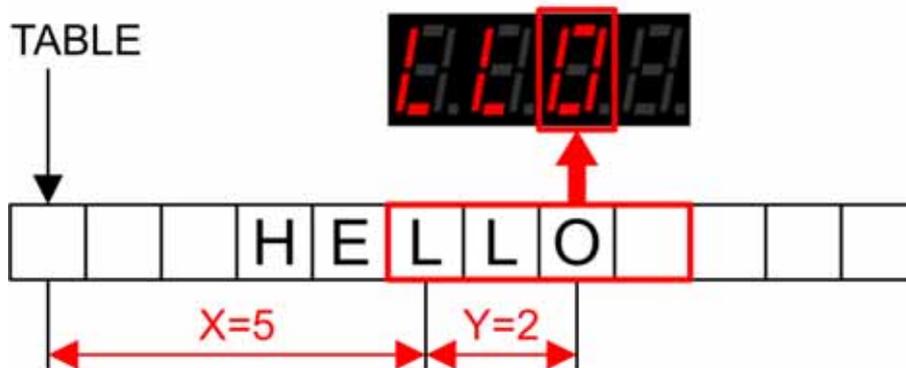
5.12 Bonjour



Ce petit programme fait défiler le mot « HELLO » sur les afficheurs. Il illustre bien la puissance des registres X et Y du processeur PSI30.

- X correspond à l'index de la lettre dans la table. Il varie de 0 à 8.
- Y correspond à l'afficheur. Il varie de 0 à 3.

L'instruction à l'adresse H'004 est particulièrement intéressante. Elle met la bonne lettre dans le registre A, en fonction du rang dans le mot (registre X) et du rang de l'afficheur à utiliser (registre Y).



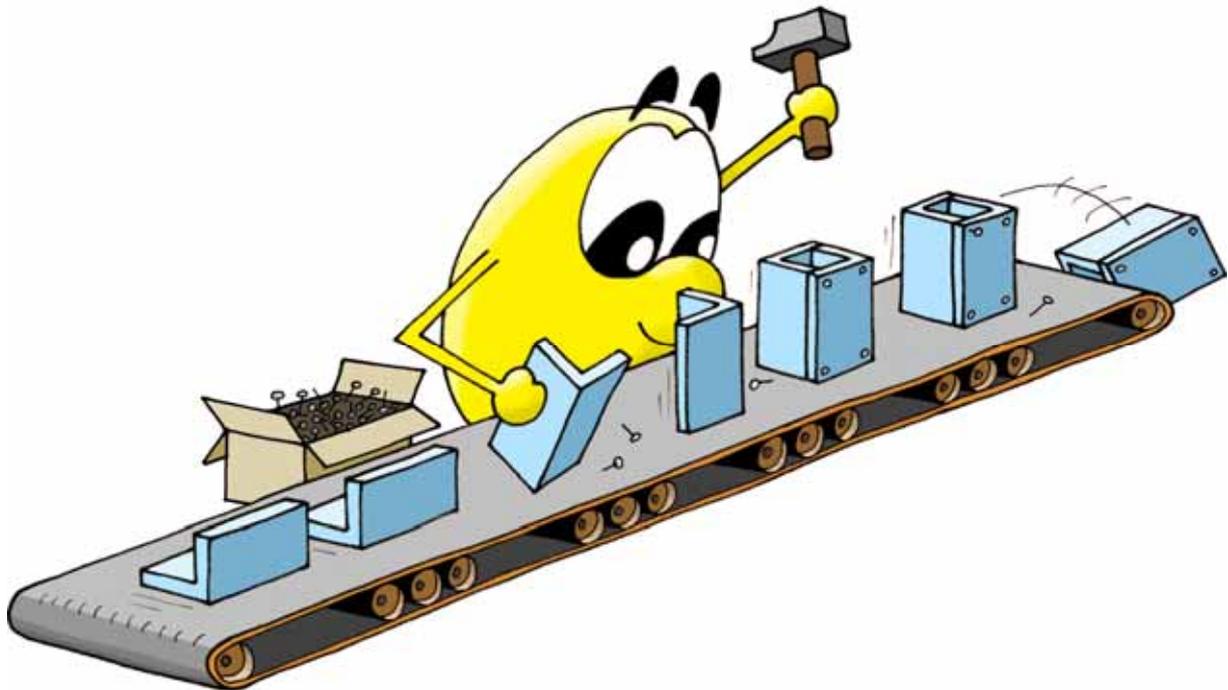
La figure ci-dessus montre le contenu des registres X et Y lors de l'affichage de la lettre « O » sur le troisième afficheur.

Adr.	Donnée	Instruction	Commentaire
000	52 00	MOVE #H'00, X	Rang de la lettre
002	53 00	MOVE #H'00, Y	Rang du digit
004	54 30 20	MOVE H'020+{X}+{Y}, A	Cherche la lettre
007	58 2C 00	MOVE A, H'C00+{Y}	Affiche la lettre sur le digit
00A	2B	INC Y	Digit suivant
00B	73 04	COMP #H'04, Y	4 ^{ème} digit atteint ?
00D	14 00 04	JUMP,LO H'004	Si non, saute en H'004
010	50 03	MOVE #H'03, A	Si oui, attend 1.5 secondes
012	01 08 06	CALL H'806	_WaitSec
015	2A	INC X	Lettre suivante
016	72 08	COMP #H'08, X	Dernière lettre atteinte ?
018	16 00 02	JUMP,LS H'002	Si non, saute en H'002
01B	10 00 00	JUMP H'000	Si oui, saute au début
01E	FF 0C	TABLE #H'0C	Début d'une table de 12 octets
020	00	BYTE #H'00	Caractère espace
021	00	BYTE #H'00	Caractère espace
022	00	BYTE #H'00	Caractère espace
023	5B	BYTE #H'5B	Caractère « H » (0101 1011)
024	7C	BYTE #H'7C	Caractère « E » (0111 1100)
025	1C	BYTE #H'1C	Caractère « L » (0001 1100)
026	1C	BYTE #H'1C	Caractère « L » (0001 1100)
027	3F	BYTE #H'3F	Caractère « O » (0011 1111)
028	00	BYTE #H'00	Caractère espace
029	00	BYTE #H'00	Caractère espace
02A	00	BYTE #H'00	Caractère espace
02B	00	BYTE #H'00	Caractère espace

Nom du programme : hello1.dolphin

Exécutez ce programme à la fréquence de [10'000] instructions par seconde.

5.13 L'assembleur d'instructions

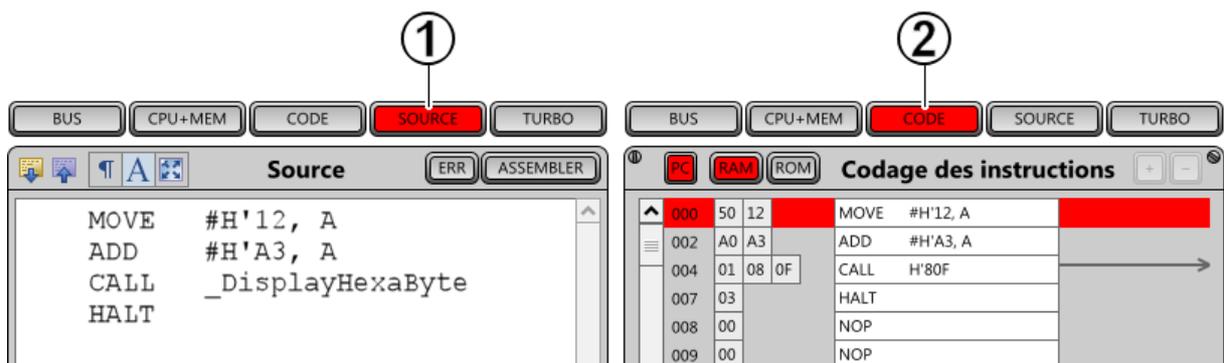


Pour la suite des exercices, nous allons utiliser une ultime méthode, encore plus souple : l'assembleur d'instructions.



Le principe est le suivant :

- Vous éditez un texte qui décrit les instructions sous une forme la plus claire possible. Ce texte est appelé « source ».
- Un outil appelé « assembleur » traduit ce texte en instructions binaires.



1. **Texte source.** Les routines en ROM sont prédéfinies, ce qui permet d'écrire « CALL _DisplayHexaByte », qui est bien plus clair que « CALL H'80F » (qui reste cependant possible).
2. **Code généré.**

La création d'un programme s'effectue dans cet ordre :

1. Edition du programme source.
2. Assemblage automatique, en cliquant sur le bouton [**ASSEMBLER**].
3. Si des fautes de frappe sont détectées pendant l'assemblage, les lignes incriminées sont marquées spécialement. Vous devez alors corriger les erreurs, puis recommencer au point 2. Le bouton [**ERR**] déplace le curseur sur la prochaine erreur.
4. Exécution du programme avec le bouton [**RUN**], souvent en mode **STEP**, pour analyser les dysfonctionnements, avec l'onglet [**CODE**].
5. Correction des éventuels problèmes en recommençant le cycle au point 1.

Un gros avantage de l'assembleur réside dans la définition de constantes. Par exemple, le programme en haut de la page précédente pourrait s'écrire ainsi :

```
NUMBER1 = H'12
NUMBER2 = H'A3
    MOVE    #NUMBER1, A
    ADD     #NUMBER2, A
    CALL    _DisplayHexaByte
    HALT
```

Le programme du chapitre 5.8 créé en mode [**CPU+MEM**], qui fait tourner un segment sur l'afficheur de gauche, s'écrit ainsi avec l'assembleur :

```
.LOC    0
START:
    MOVE    #1, A           ; met le bit initial
LOOP:
    MOVE    A, _DIGIT0     ; allume le bon segment
    RL     A                ; décale le bit à gauche
    AND     #H'3F, A       ; masque les bits inutiles
    JUMP,ZS START          ; si plus rien -> START
    JUMP    LOOP           ; recommence à l'infini
```

Vous constatez immédiatement une énorme simplification dans la gestion des sauts. Par exemple, la dernière instruction du programme s'écrit « JUMP LOOP », ce qui signifie « saute à l'étiquette LOOP ». N'importe où ailleurs dans le programme, vous définissez une étiquette « LOOP: » et le tour est joué : l'instruction de saut continue l'exécution du programme sur l'instruction qui suit l'étiquette. L'assembleur se charge de calculer les bonnes adresses, y compris si vous ajoutez ou supprimez des instructions entre le saut et l'étiquette.

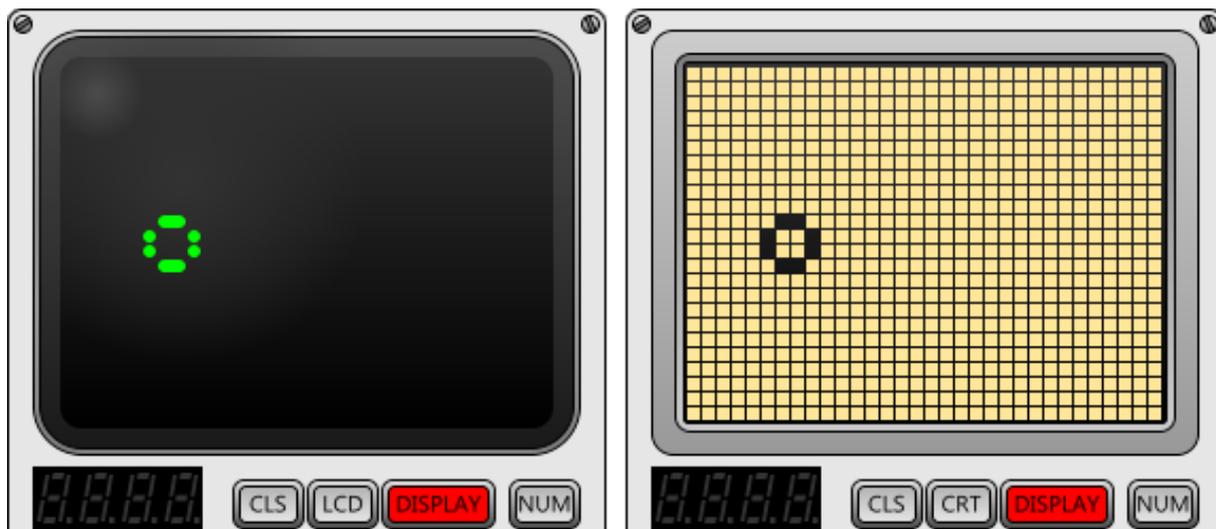
Les périphériques sont prédéfinis (voir l'onglet « Intro »), ce qui permet d'écrire « MOVE A, _DIGIT0 », qui est bien plus clair que « MOVE A, H'C00 ».

L'instruction spéciale « .LOC 0 » est facultative. Elle détermine l'adresse à laquelle est généré le code.

Le texte compris entre le point-virgule et la fin de la ligne est ignoré par l'assembleur. Ceci permet d'écrire des commentaires, afin d'expliquer le fonctionnement du programme.

5.14 Encore des rebonds

Au chapitre 5.11, nous avons fait rebondir un point sur les bords gauche et droite de l'écran bitmap. Nous allons maintenant faire rebondir une balle composée de huit points.



Le programme du chapitre 5.11 créé en mode **[CODE]** s'écrit ainsi avec l'assembleur :

```

        .LOC      0
        CLR       X           ; X à gauche
        MOVE     #12, Y      ; Y au milieu
        MOVE     #1, B       ; de gauche à droite
LOOP:
        COMP     #31, X      ; touche le bord droite ?
        JUMP,LO RIGHT      ; non -> RIGHT
        MOVE     #-1, B      ; de droite à gauche
RIGHT:
        COMP     #0, X       ; touche le bord gauche ?
        JUMP,HI LEFT       ; non -> LEFT
        MOVE     #1, B       ; de gauche à droite
LEFT:
        ADD      B, X        ; avance ou recule X
        CALL     _NotPixel   ; allume le nouveau point

        MOVE     #10, A      ; durée à attendre
        CALL     _WaitSec    ; attend...

        CALL     _NotPixel   ; éteint l'ancien point
        JUMP     LOOP

```

Le dessin de la balle s'effectue par les deux instructions « CALL _NotPixel », qui inversent simplement un pixel dans l'écran. Mais rien n'empêche de remplacer l'appel de cette routine en ROM par une routine définie dans la mémoire RAM, par votre programme.

La routine qui dessine une balle de huit points s'écrit ainsi :

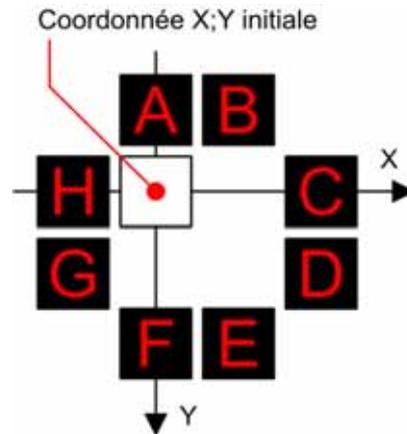
```

; Inverse la balle.
; in  X position horizontale
;     Y position verticale
; out -
; mod F
NOTBALL:
    PUSH    X
    PUSH    Y

    DEC     Y
    CALL    _NotPixel    ; A
    INC     X
    CALL    _NotPixel    ; B
    INC     X
    INC     Y
    CALL    _NotPixel    ; C
    INC     Y
    CALL    _NotPixel    ; D
    DEC     X
    INC     Y
    CALL    _NotPixel    ; E
    DEC     X
    DEC     Y
    CALL    _NotPixel    ; G
    DEC     Y
    CALL    _NotPixel    ; H

    POP     Y
    POP     X
    RET

```



Les cinq premières lignes qui commencent par un point-virgule sont facultatives. Il s'agit de commentaires destinés à faciliter la compréhension du fonctionnement de la routine. C'est une bonne habitude que de préciser ce qu'utilise une routine en entrée (*in*), ce qu'elle fournit éventuellement en sortie (*out*), et les registres qu'elle modifie (*mod*).

Une routine est toujours terminée par l'instruction « RET » (*return*).

1. Cliquez sur l'icône  « Ouvrir » en haut à gauche, et sélectionnez le programme « rebond2.dolphin ».
2. Cliquez sur le bouton [**SOURCE**].
3. A la fin du programme, tapez le code de la routine NOTBALL décrit à la page précédente.
4. Effectuez les corrections marquées en rouge ci-dessous.
5. Cliquez sur [**ASSEMBLER**] et corrigez les éventuelles erreurs.
6. Cliquez sur [**100'000**] puis sur [**RUN**].

```

        .LOC      0
        CLR       X           ; X à gauche
        MOVE     #11, Y      ; Y au milieu
        MOVE     #1, B       ; de gauche à droite
LOOP:
        COMP     #29, X      ; touche le bord droite ?
        JUMP,LO  RIGHT      ; non -> RIGHT
        MOVE     #-1, B      ; de droite à gauche
RIGHT:
        COMP     #1, X       ; touche le bord gauche ?
        JUMP,HI  LEFT       ; non -> LEFT
        MOVE     #1, B       ; de gauche à droite
LEFT:
        ADD      B, X        ; avance ou recule X
        CALL     NOTBALL     ; allume nouvelle balle

        MOVE     #10, A      ; durée à attendre
        CALL     _WaitSec    ; attend...

        CALL     NOTBALL     ; éteint l'ancienne balle
        JUMP     LOOP

NOTBALL:
        etc...
        RET

```

Si vous souhaitez aller plus loin, ouvrez le programme **rebond4.dolphin**, qui fait rebondir une balle à 45 degrés sur les quatre bords de l'écran bitmap. Il a suffi de quelques instructions supplémentaires pour gérer le déplacement vertical.

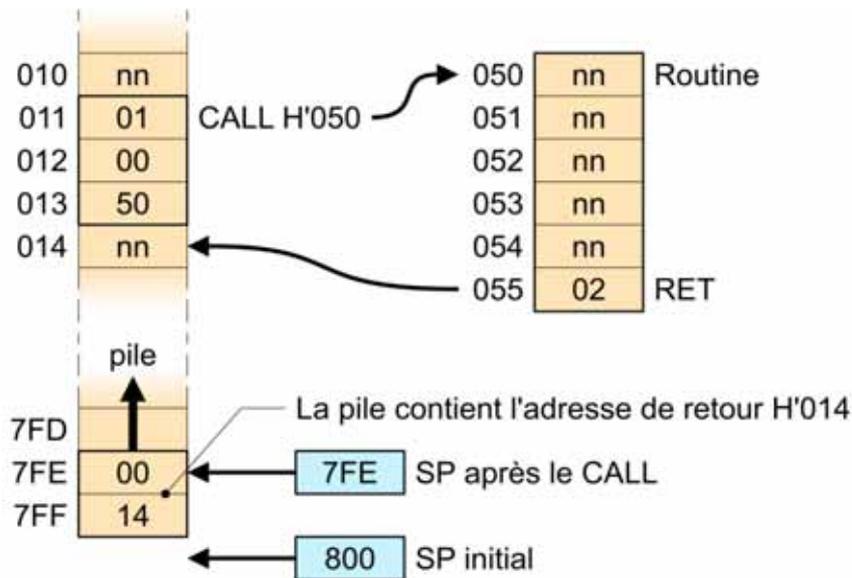
5.15 La pile

Pour comprendre le fonctionnement du programme du chapitre précédent, quelques explications sur la pile (*stack*) s'imposent.



La pile est une zone mémoire pointée par le registre SP (*stack pointer*). Au départ, SP est initialisé juste après la fin de la RAM, à l'adresse H'800. L'appel d'une routine sauvegarde l'adresse de retour sur la pile à la fin de la RAM. Supposons que l'adresse H'011 contienne l'instruction « CALL H'050 ». Lors de son exécution, le processeur effectue les tâches suivantes :

1. Le registre SP est diminué de 2. Il vaut donc H'7FE.
2. L'adresse de retour H'014 est sauvegardée sur la pile. C'est l'adresse de l'instruction qui suit le « CALL ».
3. Saut au début de la routine appelée, à l'adresse H'050.



La routine se termine par l'instruction « RET », qui effectue les tâches suivantes :

1. L'adresse de retour H'014 est récupérée sur la pile.
2. Le registre SP est augmenté de 2. Il vaut donc à nouveau H'800.
3. Saut à l'adresse de retour H'014.

Les instructions « PUSH » et « POP » sauvegardent et restituent des registres sur la pile. « PUSH » effectue les tâches suivantes :

1. SP est diminué de 1.
2. La valeur contenue dans le registre est sauvegardée sur la pile.

« POP » effectue les tâches suivantes :

1. La valeur récupérée sur la pile est copiée dans le registre.
2. SP est augmenté de 1.



6 A l'aide



Ce petit manuel ne fait qu'aborder le vaste sujet de la programmation des microprocesseurs. Si vous êtes curieux, vous aurez remarqué que le panneau supérieur droite donne le résumé de toutes les instructions du processeur PSI30, dans l'onglet Ops. Vous pouvez ainsi facilement expérimenter d'autres instructions que celles vues ici.

Si vous souhaitez aller plus loin, ouvrez les programmes **hello2.dolphin** et **hello3.dolphin** et essayez d'en comprendre le fonctionnement. Plusieurs petits jeux sont également disponibles : **pingpong**, **mur**, **snake** et **life**.

Sur le site www.epsitec.ch/dauphin, vous trouverez la suite de ce manuel.

7 C'est méga

Les divers chiffres qui caractérisent les ordinateurs modernes sont tellement grands qu'on a pris l'habitude de parler de kilos, de mégas et de gigas.

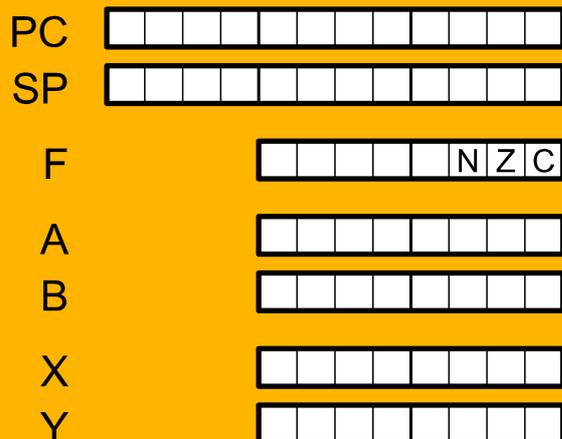
Abréviation	Nom	Quantité	Valeur approximative	Valeur exacte
K	kilo	mille	1'000	1'024
M	méga	un million	1'000'000	1'024 kilos
G	giga	un milliard	1'000'000'000	1'024 mégas
T	téra	un billion	1'000'000'000'000	1'024 gigas

Voici quelques capacités usuelles :

Média	Capacité	Abréviation
Disquette	1.4 méga octet	1.4 Mo
CD	700 méga octets	700 Mo
Mémoire vive	1 giga octet	1 Go
DVD	9 giga octets	9 Go
Disque dur	200 giga octets	200 Go

On parle souvent de Mb (méga-byte) ou Gb (giga-byte). Il s'agit de termes anglo-saxons exactement équivalents à Mo et Go.

Résumé PSI30



Transferts

[40+r]	MOVE A, r	(N, Z)
[44+r]	MOVE B, r	(N, Z)
[48+r]	MOVE X, r	(N, Z)
[4C+r]	MOVE Y, r	(N, Z)
[50+r] [vv]	MOVE #val, r	(N, Z)
[54+r] [mh] [II]	MOVE ADDR, r	(N, Z)
[58+r] [mh] [II]	MOVE r, ADDR	(N, Z)
[DC] [vv] [mh] [II]	MOVE #val, ADDR	(N, Z)

Additions

[80+r]	ADD A, r	(N, Z, C)
[84+r]	ADD B, r	(N, Z, C)
[88+r]	ADD X, r	(N, Z, C)
[8C+r]	ADD Y, r	(N, Z, C)
[A0+r] [vv]	ADD #val, r	(N, Z, C)
[B0+r] [mh] [II]	ADD ADDR, r	(N, Z, C)
[B8+r] [mh] [II]	ADD r, ADDR	(N, Z, C)
[DE] [vv] [mh] [II]	ADD #val, ADDR	(N, Z, C)

Soustractions

[90+r]	SUB A, r	(N, Z, C)
[94+r]	SUB B, r	(N, Z, C)
[98+r]	SUB X, r	(N, Z, C)
[9C+r]	SUB Y, r	(N, Z, C)
[A4+r] [vv]	SUB #val, r	(N, Z, C)
[B4+r] [mh] [II]	SUB ADDR, r	(N, Z, C)
[BC+r] [mh] [II]	SUB r, ADDR	(N, Z, C)
[DF] [vv] [mh] [II]	SUB #val, ADDR	(N, Z, C)

ET logique

[E0]	AND A, B	(N, Z)
[E1]	AND B, A	(N, Z)
[74+r] [vv]	AND #val, r	(N, Z)
[E8+r'] [mh] [II]	AND ADDR, r'	(N, Z)
[F0+r'] [mh] [II]	AND r', ADDR	(N, Z)

Registre

[xx+r]	r	[xx+r']	r'
r=0	A	r'=0	A
r=1	B	r'=1	B
r=2	X		
r=3	Y		

Valeur immédiate

[vv]	#val
vv	Valeur positive 8 bits.

Adresse

[mh] [II]	ADDR
m=0	Adresse absolue 12 bits
m=1	+{X}
m=2	+{Y}
m=4	{SP}+depl (± adresse relative)
m=8	{PC}+depl (± adresse relative)

OU logique

[E2]	OR A, B	(N, Z)
[E3]	OR B, A	(N, Z)
[78+r] [vv]	OR #val, r	(N, Z)
[EA+r'] [mh] [II]	OR ADDR, r'	(N, Z)
[F2+r'] [mh] [II]	OR r', ADDR	(N, Z)

OU exclusif logique

[E4]	XOR A, B	(N, Z)
[E5]	XOR B, A	(N, Z)
[7C+r] [vv]	XOR #val, r	(N, Z)
[EC+r'] [mh] [II]	XOR ADDR, r'	(N, Z)
[F4+r'] [mh] [II]	XOR r', ADDR	(N, Z)

Test de bit

[C0]	TEST B: A	(Z)
[C1]	TEST A: B	(Z)
[D0+r'] [vv]	TEST r': #val	(Z)
[C8+r'] [mh] [II]	TEST ADDR: r'	(Z)
[D8] [vv] [mh] [II]	TEST ADDR: #val	(Z)

Test de bit et mise à un

[C2]	TSET B: A	(Z)
[C3]	TSET A: B	(Z)
[D2+r'] [vv]	TSET r': #val	(Z)
[CA+r'] [mh] [II]	TSET ADDR: r'	(Z)
[D9] [vv] [mh] [II]	TSET ADDR: #val	(Z)

Test de bit et mise à zéro

[C4]	TCLR B: A	(Z)
[C5]	TCLR A: B	(Z)
[D4+r'] [vv]	TCLR r': #val	(Z)
[CC+r'] [mh] [II]	TCLR ADDR: r'	(Z)
[DA] [vv] [mh] [II]	TCLR ADDR: #val	(Z)

Test de bit et inversion

[C6]	TNOT B: A	(Z)
[C7]	TNOT A: B	(Z)
[D6+r] [vv]	TNOT r': #val	(Z)
[CE+r'] [mh] [II]	TNOT ADDR: r'	(Z)
[DB] [vv] [mh] [III]	TNOT ADDR: #val	(Z)

Comparaisons

[60+r]	COMP A, r	(N, Z, C)
[64+r]	COMP B, r	(N, Z, C)
[68+r]	COMP X, r	(N, Z, C)
[6C+r]	COMP Y, r	(N, Z, C)
[70+r] [vv]	COMP #val, r	(N, Z, C)
[F8+r] [mh] [II]	COMP ADDR, r	(N, Z, C)
[DD] [vv] [mh] [III]	COMP #val, ADDR	(N, Z, C)

Opérations unaires

[20+r]	CLR r	(N=0, Z=1)
[24+r]	NOT r	(N, Z)
[28+r]	INC r	(N, Z)
[2C+r]	DEC r	(N, Z)
[A8] [mh] [II]	CLR ADDR	(N=0, Z=1)
[A9] [mh] [II]	NOT ADDR	(N, Z)
[AA] [mh] [II]	INC ADDR	(N, Z)
[AB] [mh] [II]	DEC ADDR	(N, Z)

Rotations

[30+r]	RL r	(N, Z, C)
[34+r]	RR r	(N, Z, C)
[38+r]	RLC r	(N, Z, C)
[3C+r]	RRC r	(N, Z, C)
[AC] [mh] [II]	RL ADDR	(N, Z, C)
[AD] [mh] [II]	RR ADDR	(N, Z, C)
[AE] [mh] [II]	RLC ADDR	(N, Z, C)
[AF] [mh] [II]	RRC ADDR	(N, Z, C)

Sauts

[10] [mh] [II]	JUMP ADDR
[12] [mh] [II]	JUMP,EQ ADDR
[12] [mh] [II]	JUMP,ZS ADDR
[13] [mh] [II]	JUMP,NE ADDR
[13] [mh] [II]	JUMP,ZC ADDR
[14] [mh] [II]	JUMP,LO ADDR
[14] [mh] [II]	JUMP,CS ADDR
[15] [mh] [II]	JUMP,HS ADDR
[15] [mh] [II]	JUMP,CC ADDR
[16] [mh] [II]	JUMP,LS ADDR
[17] [mh] [II]	JUMP,HI ADDR
[18] [mh] [II]	JUMP,NS ADDR
[19] [mh] [II]	JUMP,NC ADDR

Appels de routines

[01] [mh] [III]	CALL ADDR
[02]	RET

Utilisation de la pile

[08+r]	PUSH r	
[FC]	PUSH F	
[0C+r]	POP r	
[FD]	POP F	(N, Z, C)
[07] [vv]	SUB #val, SP	
[06] [vv]	ADD #val, SP	
[54+r] [40] [dd]	MOVE {SP}+depl, r	(N, Z)
[58+r] [40] [dd]	MOVE r, {SP}+depl	(N, Z)

Gestion des fanions

[04]	SETC	(C=1)
[05]	CLRC	(C=0)
[FE]	NOTC	(C)

Divers

[00]	NOP
[03]	HALT
[5C]	EX A, B
[5D]	EX X, Y
[5E]	SWAP A
[5F]	SWAP B

Exemples

MOVE A, X [42]

[40+r] avec r=X=2 donne [42]

SUB #10, B [A5] [0A]

#10 est en décimal et donne H'0A en hexa

TEST H'C07: #3 [D8] [03] [0C] [07]

Instruction de 4 octets avec la valeur du bit en premier [03] suivi de l'adresse [0C] [07]

ADD H'1F4+{Y}, B [B1] [21] [F4]

[mh] avec +{Y} donne m=2, le poids fort de l'adresse H'1F4 est 1, donc [21]

JUMP,LS {PC}+7 [16] [80] [07]

Saut relatif 7 octets plus loin

[mh] avec {PC}+depl donne m=8

JUMP {PC}-3 [10] [8F] [FD]

Saut relatif 3 octets en arrière (boucle infinie)

[mh] avec {PC}+depl donne m=8, -3 vaut H'FFD

MOVE {SP}+2, A [54] [40] [02]

[mh] avec {SP}+depl donne m=4