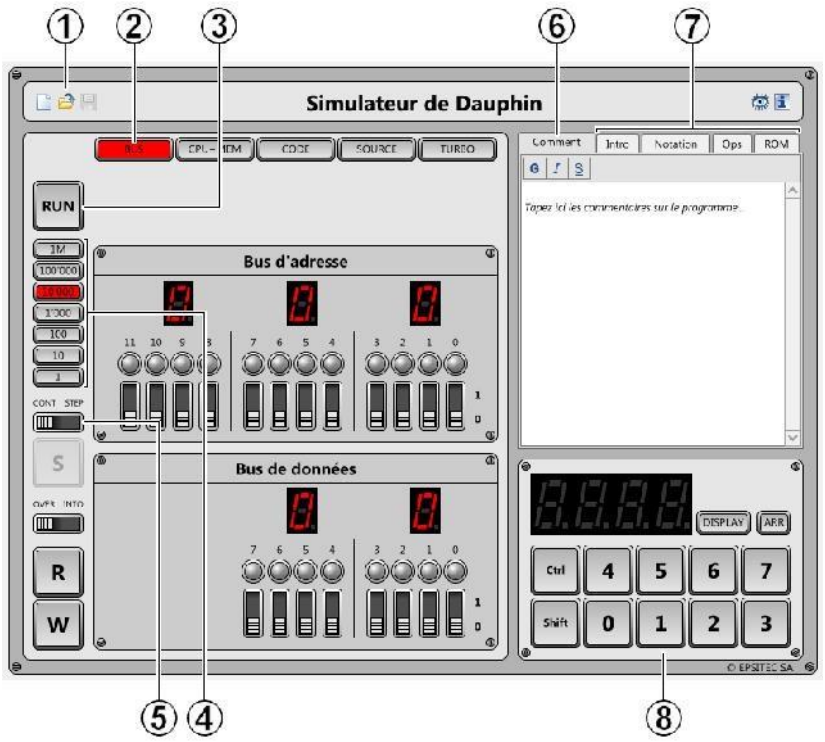
I. Le simulateur de Dauphin

Le simulateur de Dauphin est un logiciel qui fonctionne sur la plupart des PC récents équipés de Windows® 7, Vista ou XP. 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 acquerrez 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.

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



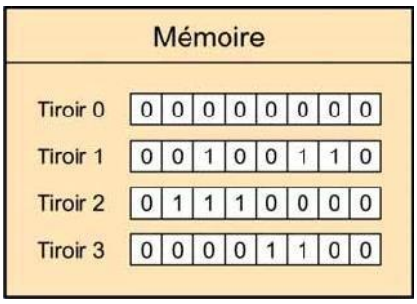
1. Icônes pour effacer, ouvrir ou enregistrer un programme.
2. Choix des panneaux affichés en dessous. [**BUS**] affiche les bus d'adresse et de données.
3. Bouton principal [**RUN/STOP**] pour démarrer ou arrêter le processeur.
4. Choix de la vitesse du processeur, en nombre d'instructions par secondes.
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.

II. 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:

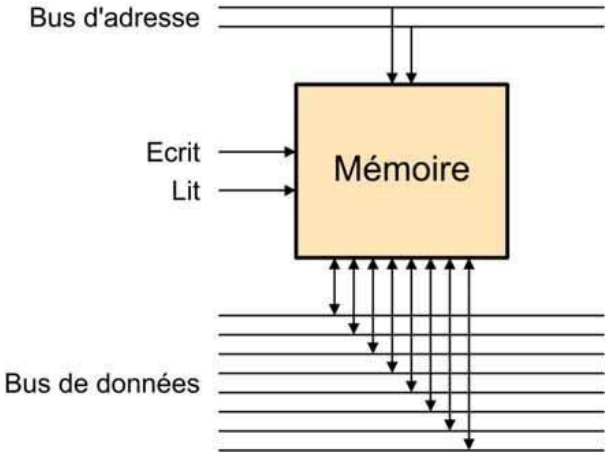
- Ecrire (*write*) consiste à mettre une valeur dans la mémoire.
- Lire (*read*) consiste à retrouver une valeur précédemment écrite.

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 4x8 = 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.
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.

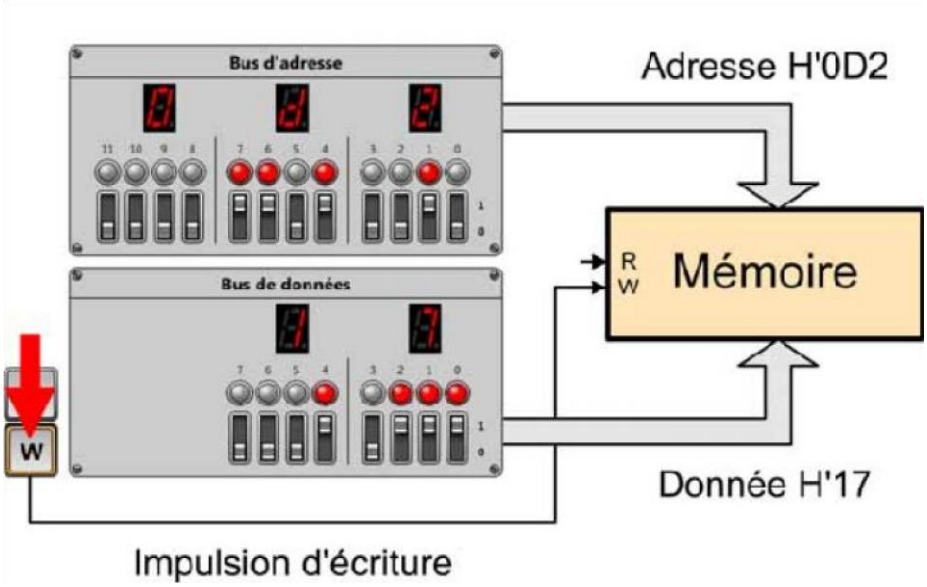
III. Une mémoire de Dauphin

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.

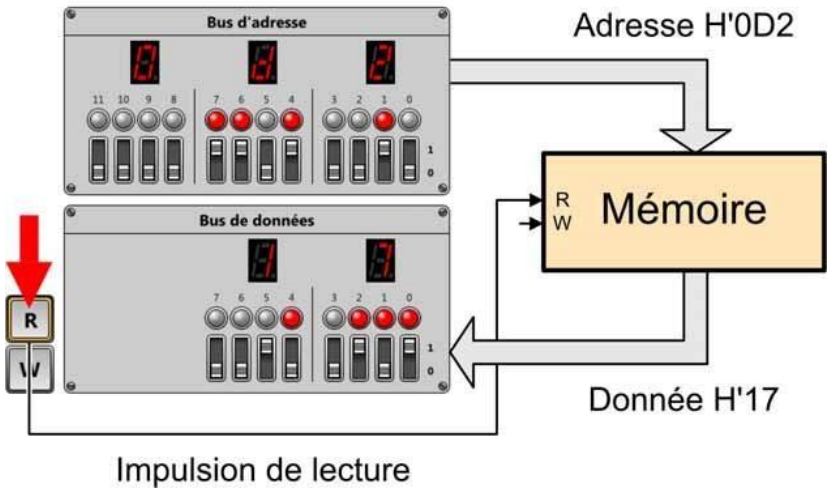


Ecrivons la valeur H'25 à l'adresse H'0D3:

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:

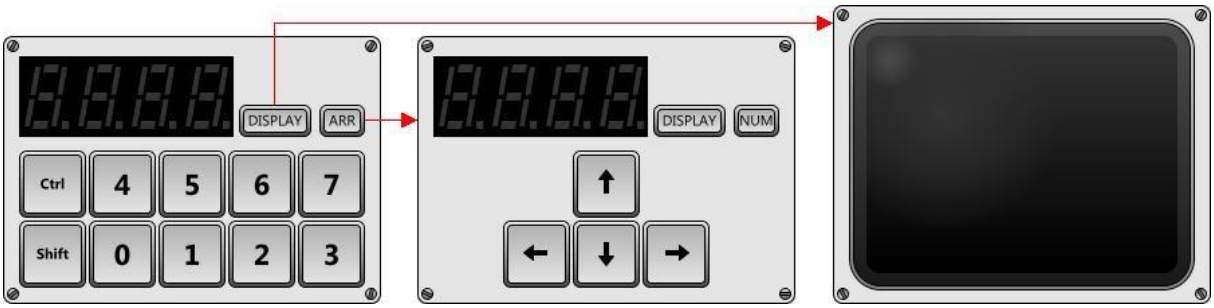
1. Sélectionnez l'adresse H'0D3 (0000 1101 0011).
2. Pressez sur le bouton poussoir [**R**]. Tant qu'il est maintenu pressé, on peut lire la valeur H'25 sur le bus de données.

IV. 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 droit 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.

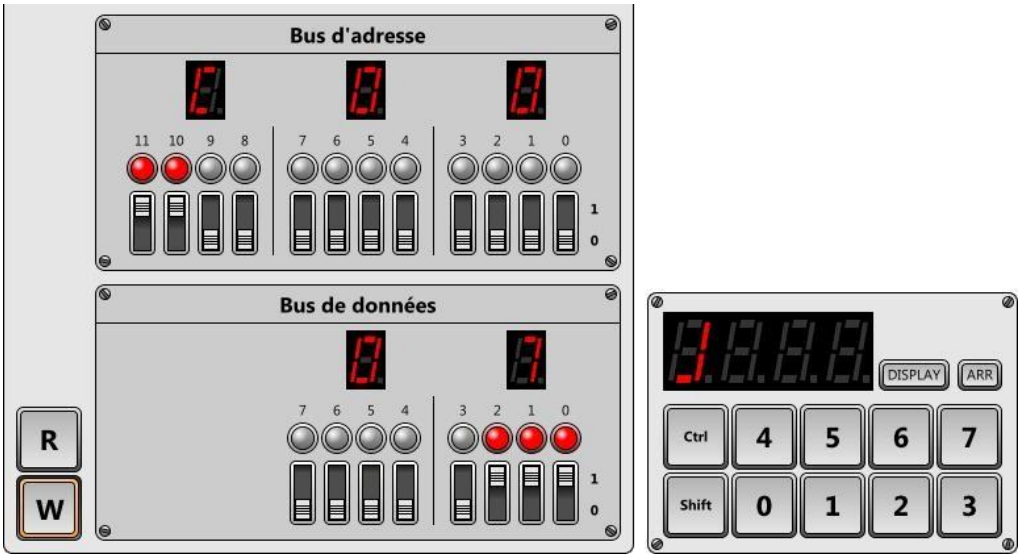
Peut-être aurez-vous remarqué que le dernier interrupteur (bit 11) du bus d'adresse semble inutile, puisque la mémoire répond aux adresses H'000 à H'7FF (0111 1111 1111 en binaire). En fait, il a bien une utilité précise: l'accès à la mémoire morte et aux périphériques.

Adresses	Contenu	Abréviation	Signification
H'000.. H'7FF	Mémoire vive	RAM	*Random accessmemory*
H'800.. H'BFF	Mémoire morte	ROM	*Readonlymemory*
H'C00.. H'C0F	Périphériques	PER	
H'C80.. H'CDF	Ecran bitmap	DIS	*Display*

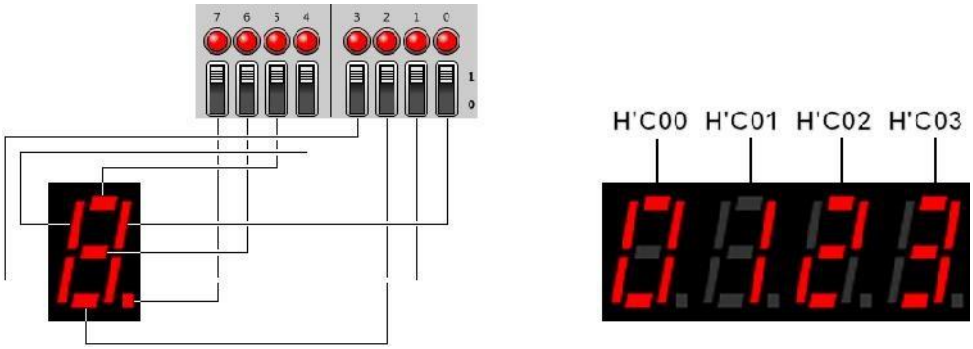
Lorsque nous utilisons les bus pour écrire à une adresse comprise entre H'000 et H'7FF, nous accédons à la mémoire vive (RAM). Pour interagir avec un périphérique, il suffit d'utiliser une adresse comprise entre H'C00 et H'C0F. C'est aussi simple que cela.

V. L'affichage

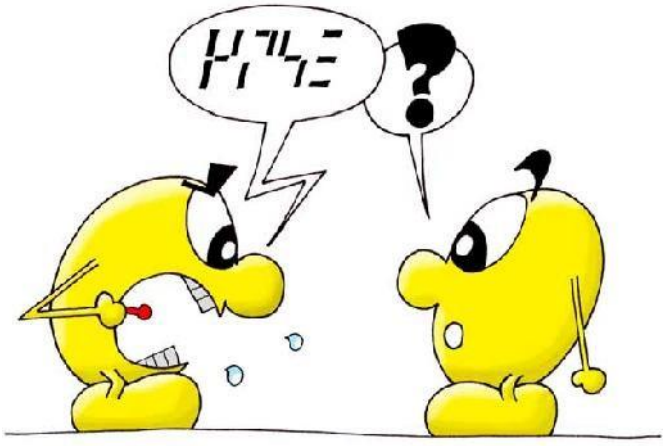
1. Sélectionnez l'adresse H'C00 (1100 0000 0000) avec les interrupteurs du bus d'adresse.
2. Sélectionnez la donnée H'07 (0000 0111) avec les interrupteurs du bus de données.
3. Pressez sur le bouton poussoir [**W**] (*write*= écrire) en bas à gauche. Une sorte de « J » apparaît sur l'afficheur de gauche.



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:



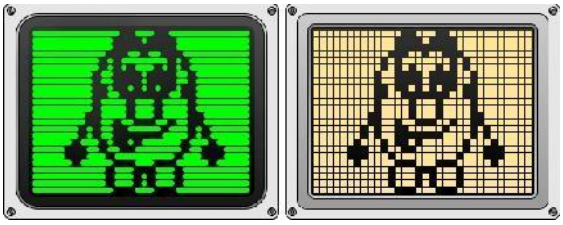
Expérimentez ceci avec différentes adresses et différentes données, en appuyant à chaque fois sur [**W**]. Vous pouvez ainsi afficher vraiment n'importe quoi:



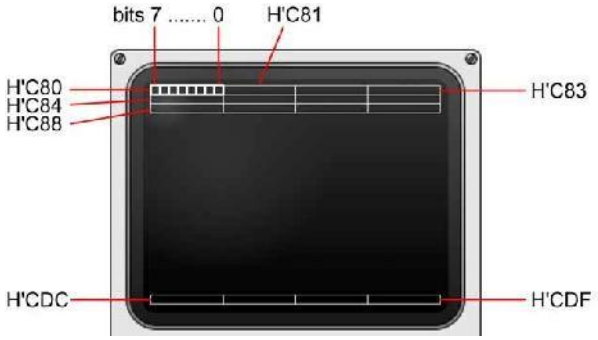
VI. L'écran bitmap

L'écran bitmap est l'ancêtre des écrans vidéo d'aujourd'hui. Il permet d'afficher

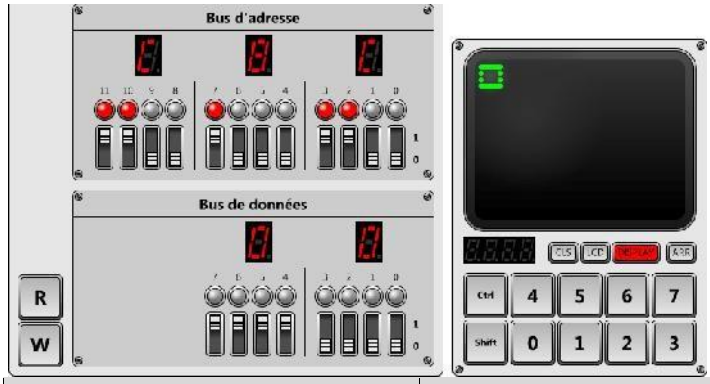
32 x 24 points (*pixels*) monochromes. Pour voir l'écran bitmap, cliquez sur le bouton [**DISPLAY**]. Les quatre afficheurs à sept segments sont alors réduits, mais ils sont toujours fonctionnels. Le bouton [**CLS**] (*CLearScreen*) efface l'écran, si nécessaire. Le bouton [**LCD**] (*LiquidCrystalDisplay*) change la technologie de l'écran simulé. [**CRT**] (*CathodeRayTube*) 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**]:



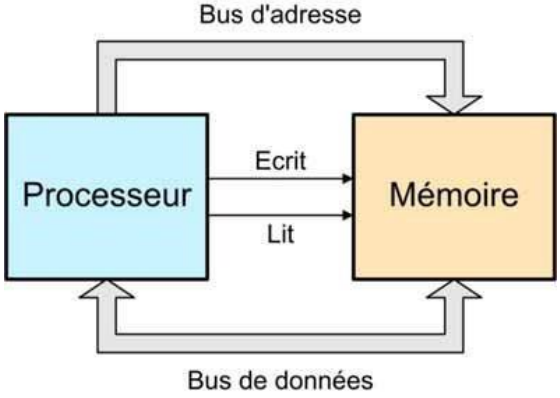
Adresse	Donnée
H'C80 (1100 1000 0000)	H'F0 (1111 0000)
H'C84 (1100 1000 0100)	H'90 (1001 0000)
H'C88 (1100 1000 1000)	H'90 (1001 0000)
H'C8C (1100 1000 1100)	H'F0 (1111 0000)

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

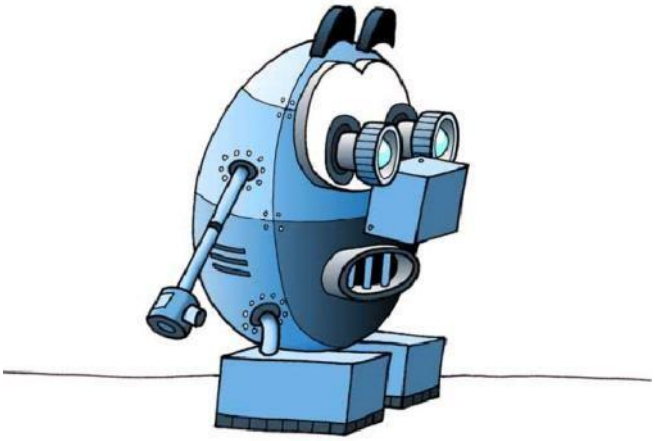
VII. Le processeur

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.

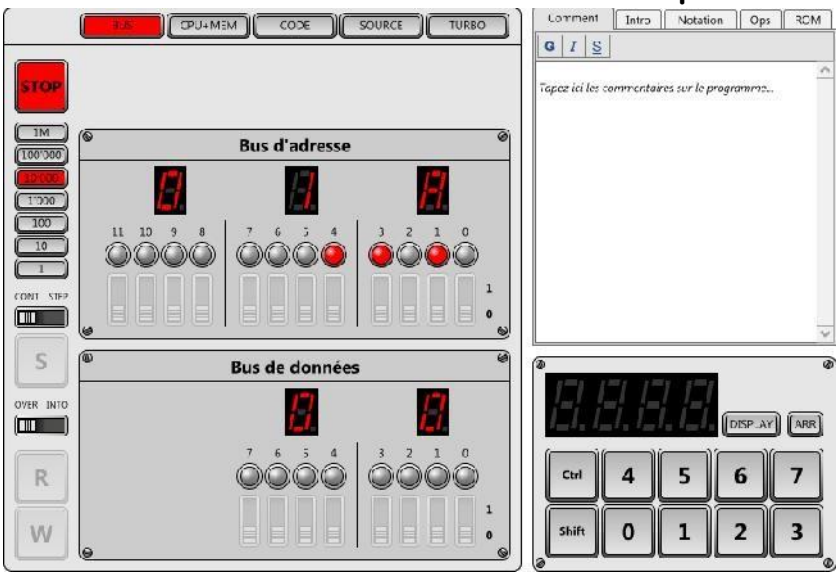


Le processeur est une sorte d'automate sophistiqué.



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 retenez 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.

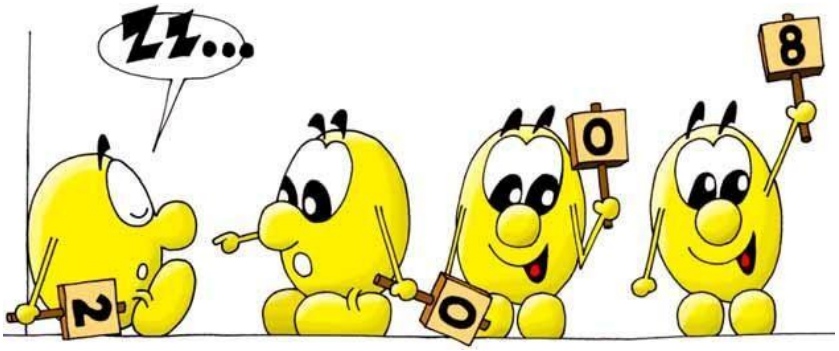
La programmation consiste à remplir la mémoire avec des instructions plus complexes que les « NOP » initiaux. Ce sont ces instructions que le processeur va suivre à la lettre, et qui vont déterminer les tâches effectuées par le programme.

a) 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.

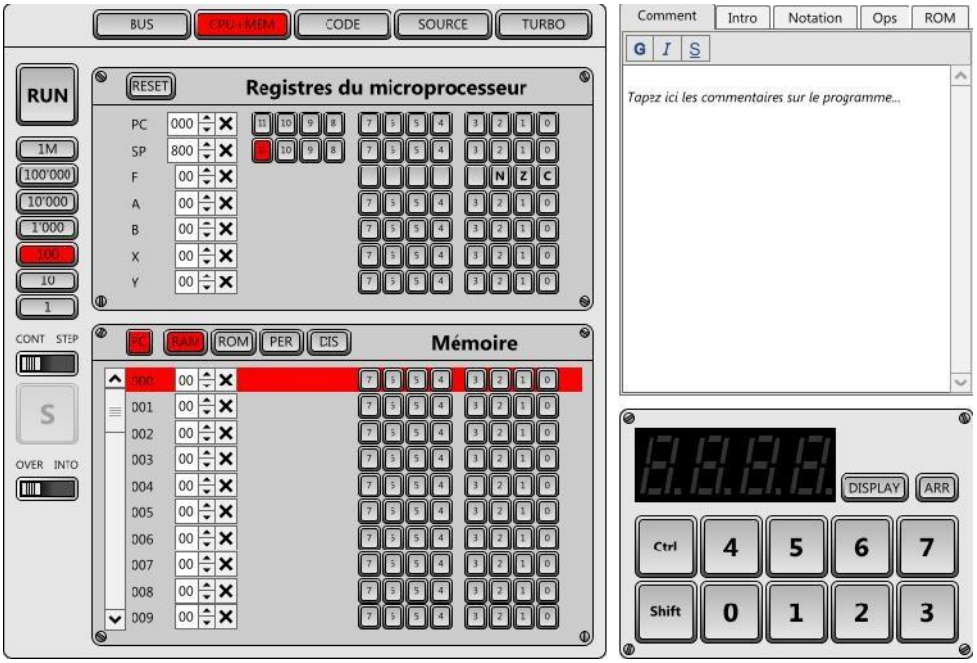
Ce programme très simple affiche « 2008 », à la fréquence de 100 instructions par seconde, puis stoppe. On voit nettement que les quatre chiffres n'apparaissent pas instantanément, mais de droite à gauche: 8, 0, 0 puis finalement 2. Et pourtant, 100 instructions par seconde semblent déjà une belle vitesse. 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'exceptionnelaujourd'hui !

b) 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 (*CentralProcessingUnit*) et la mémoire.



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** (*programcounter*) 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.

c) 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.

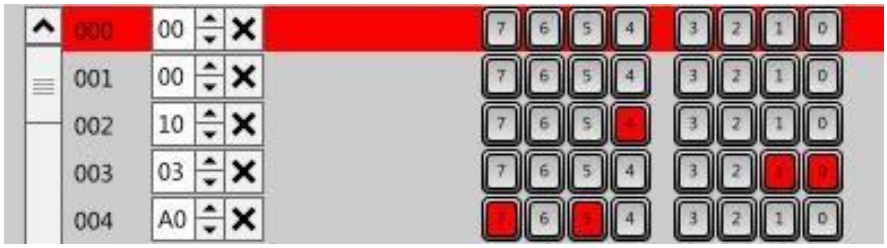
d) Premier saut dans l'inconnu

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.



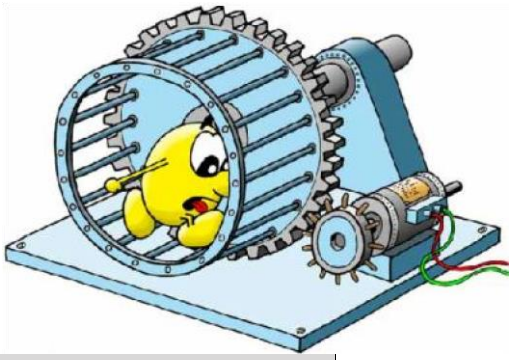
Pour pouvoir observer calmement ce premier programme rudimentaire, nous allons l'exécuter en « pas à pas » (*step bystep*). Cela signifie que le processeur ne va chercher et exécuter la prochaine instruction que lorsque vous appuyez sur le bouton [**S**] (*step*). Pour passer dans ce mode, basculez le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

e) 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 !

f) 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.

g) 2 exercices d'application

Exercice 1

Écrire une séquence d'instructions qui multiplie par 5 le nombre contenu dans la case mémoire d'adresse 100 et stocke le résultat dans la case mémoire d'adresse 110.

Exercice 2

Écrire un programme qui lit deux valeurs *x*et*y*contenues respectivement dans les cases mémoires 11 et 12, calcule la différence *y*-*x*et stocke le résultat à l'adresse 13.

On suppose que ces deux valeurs sont des nombres entiers positifs.

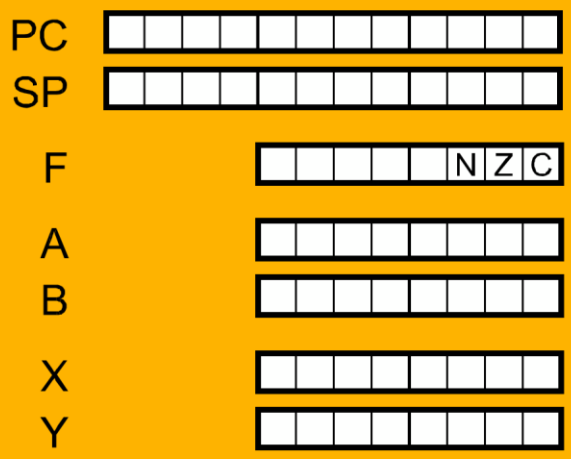
Compléter ce programme pour qu'il stocke la valeur 0 à l'adresse 15 si *x*est égal à*y*, ou la valeur *x*sinon.

a) D'autres exemples de programmes

Ouvrir des programmes exemples du logiciel (extension .dolphin), les exécuter et étudier les codes sources des programmes.

(Des programmes sont téléchargeables à cette adresse : http://www.epsitec.ch/dauphin/concours)

Résumé PSI30



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] [ll]	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)

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] [ll]	MOVE ADDR, r	(N, Z)
[58+r] [mh] [ll]	MOVE r, ADDR	(N, Z)
[DC] [vv] [mh] [ll]	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] [ll]	ADD ADDR, r	(N, Z, C)
[B8+r] [mh] [ll]	ADD r, ADDR	(N, Z, C)
[DE] [vv] [mh] [ll]	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] [ll]	SUB ADDR, r	(N, Z, C)
[BC+r] [mh] [ll]	SUB r, ADDR	(N, Z, C)
[DF] [vv] [mh] [ll]	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] [ll]	AND ADDR, r'	(N, Z)
[F0+r'] [mh] [ll]	AND r', ADDR	(N, Z)

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] [ll]	OR ADDR, r'	(N, Z)
[F2+r'] [mh] [ll]	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] [ll]	XOR ADDR, r'	(N, Z)
[F4+r'] [mh] [ll]	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] [ll]	TEST ADDR: r'	(Z)
[D8] [vv] [mh] [ll]	TEST ADDR: #val	(Z)

Test de bit et mise á un

[C2]	TSET B: A	(Z)
[C3]	TSET A: B	(Z)
[D2+r'] [vv]	TSET r': #val	(Z)
[CA+r'] [mh] [ll]	TSET ADDR: r'	(Z)
[D9] [vv] [mh] [ll]	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] [ll]	TCLR ADDR: r'	(Z)
[DA] [vv] [mh] [ll]	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] [ll]	TNOT ADDR: r'	(Z)
[DB] [vv] [mh] [ll]	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] [ll]	COMP ADDR, r	(N, Z, C)
[DD] [vv] [mh] [ll]	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] [ll]	CLR ADDR	(N=0, Z=1)
[A9] [mh] [ll]	NOT ADDR	(N, Z)
[AA] [mh] [ll]	INC ADDR	(N, Z)
[AB] [mh] [ll]	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] [ll]	RL ADDR	(N, Z, C)
[AD] [mh] [ll]	RR ADDR	(N, Z, C)
[AE] [mh] [ll]	RLC ADDR	(N, Z, C)
[AF] [mh] [ll]	RRC ADDR	(N, Z, C)

Sauts

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

Appels de routines

[01] [mh] [ll]	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