INTRO A L'ASSEMBLEUR THEORIQUE

A partir de cette semaine vous allez pouvoir trouver dans nos colonnes un cours d'assembleur décomposé en deux parties essentielles.
La première se composera d'un cours théorique sur le fonctionnement des ordinateurs, valable pour l'ensemble des micros-ordinateurs.
La deuxième formera un cours spécifique à un micro-processeur d'un micro-ordinateur, composé d'applications immédiates en langage machine.
Pour la seconde partie, un roulement de différentes machines est prévu. Ainsi chaque micro ne sera-t-il pas désavantagé ou avantagé. Pour commencer, les propriétaires de ZX81 vont pouvoir se réjouir: pendant deux semaines, le cours concernera le langage machine sur le Z80. Par la suite viendront le 6502 d'APPLE, le 6809 de Thomson, le 6510 du Commodore 64, le 6502 d'ORIC... Et bien d'autres micros! Nous espérons que cette formule satisfera l'ensemble des lecteurs.
Il est toutefois nécessaire de préciser que la partie théorique ne s'adresse pas aux "fort-en-maths" ni aux professionnels de l'informatique. Il est effectivement possible d'utiliser le langage machine d'un micro-ordinateur sans connaitre les méthodes de calcul de celui-ci -la majorité des autodidactes du langage machine se débrouillent très bien sans ces connaissances-, mais des bases solides à propos de la théorie concernant les micros-ordinateurs permettent une meilleure compréhension de la machine et ainsi une utilisation optimisée de celle-ci. Avis aux amateurs...

Michael THEVENET

P.S. Pour chaque micro-processeur, nous vous conseillerons un ouvrage de référence, ainsi que pour chaque micro-ordinateur nous vous proposerons un logiciel d'assemblage.

CONCEPTS DE BASE

Cette première partie du cours théorique va recenser un ensemble de connaissances utiles à la bonne compréhension du fonctionnement d'un micro-ordinateur. Un certain nombre de concepts et de définitions de base vont vous être exposés.
En appui à ces définitions viendront des exemples et des exercices pour vous permettre de mieux intégrer ces notions si elles sont nouvelles pour vous ou de vous les remettre en mémoire dans le cas contraire.

LA PROGRAMMATION, QU'EST-CE QUE C'EST ?

A tout problème il existe une solution qu'il s'agit de trouver. Cette recherche aboutit à la construction d'un ALGORITHME. Un algorithme se décompose en un ensemble d'instructions élémentaires qui permettent d'aboutir à la solution recherchée. Un algorithme ne se compose pas nécessairement de symboles mathématiques: il peut être écrit dans n'importe quel langage, à l'aide de n'importe quels symboles. En voici deux exemples:

1. Vous êtes devant la porte que vous avez fermée ce matin et vous voulez entrer. Voici l'algorithme que vous pouvez obtenir:

-Vous insérez la clé dans la serrure
-Vous tournez la clé d'un tour complet à gauche
-Vous tournez la poignée à gauche
-Vous poussez la porte
-Vous entrez.

2. Vous désirez calculer les racines d'une équation du second degré. Il vous faut tout d'abord savoir s'il existe des solutions. Puis, s'il en existe effectivement, les calculer.

-Vous avez une équation du type: A* X2+ B* X+ C= 0
-Vous calculez le discréminant: D= B2-4* A* C
-Si D est négatif il n'y a pas de solution
-Si D est nul il y a une racine double:
X'= X"= -B/(2* A)
-Si D est positif il y a deux solutions:
X'= (-B-RAC(D))/(2* A)
et X"= (-B+ RAC(D))/(2* A).

Une fois l'algorithme écrit, il va vous être possible de le faire exécuter par un micro-ordinateur. Mais, malheureusement, les ordinateurs ne comprennent pas encore le français. Il va donc falloir traduire l'algorithme dans un langage compréhensible par l'ordinateur. Ce langage se nomme LANGAGE DE PROGRAMMATION. La programmation signifie donc transcrire l'algorithme en une série d'instructions appartenant à un langage de programmation.

Il existe différents types de langages de programmation. Le seul à être compris par le micro sans traitement se nomme langage machine. Il est composé de codes en base 16 (héxadécimal) directement compréhensibles par le micro. Viennent ensuite deux catégories distinctes de langages EVOLUES:

-les langages interprétés (le basic implanté sur tous les micros familiaux par exemple)
-les langages compilés (ils font appel à un logiciel de traitement appelé compilateur, ainsi Pascal, Fortran, Cobol, PL1...)

La différence fondamentale entre ces deux sortes de langages se situe au niveau du traitement de l'information:

- un interpréteur basic traite les lignes une par une, au fur et à mesure de leur entrée au clavier. En effet dès qu'un Return (ou Newline ou Enter) est tapé au clavier, l'interpréteur basic se met en action pour analyser la ou les instructions tapées par l'utilisateur depuis le dernier Return. Les erreurs de syntaxe sont détectées immédiatement. La mise au point du programme, au niveau syntaxique est donc rapide. Par contre lors de l'exécution, l'interpréteur travaille à nouveau pour détecter les éventuelles erreurs de logique contenues dans le programme.
- à l'inverse, le programmeur qui utilise un langage compilé va créer un fichier contenant l'ensemble de son programme. Le fichier une fois terminé passe dans le compilateur qui va effectuer une double analyse: une analyse syntaxique, puis une analyse logique. Ce double travail terminé, on obtient un nouveau fichier directement compréhensible et exécutable par l'ordinateur. La mise au point d'un programme compilé est donc plus longue, mais le temps d'exécution est incomparablement plus court que pour un langage interprété.

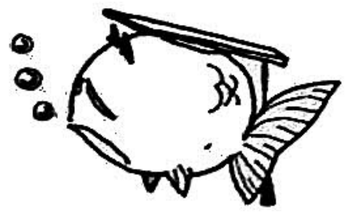
La question qui se pose maintenant est pourquoi utiliser des langages autres que le langage machine, ou pourquoi utiliser le langage machine de préférence à des langages évolués.

Il est bon de préciser que bien peu de personnes sont capables de programmer en langage machine directement. En effet, connaître la signification de codes hexadécimaux encombre la mémoire de connaissances non indispensables. C'est pour cette raison qu'a été créé le langage d'assemblage. Celui-ci est une représentation facilement mémorisable des codes hexadécimaux du langage machine. Par exemple pour un 6502 l'instruction LDA recouvre huit codes différents, suivant le mode d'adressage utilisé. Il paraît donc plus facile de retenir LDA (load accumulator) que AD, A5, A9, BD, B9, A1, B1 ou B5. Cette représentation sous forme alphanumérique du code machine se nomme MNEMONIQUE. Les personnes travaillant en langage machine, dans le langage courant, utilisent en fait des logiciels d'assemblage. Ceux-ci peuvent généralement être utilisés de deux manières:

-En mode d'assemblage immédiat l'assembleur se comporte comme un interpréteur
-En mode d'assemblage il tient lieu de compilateur.

Le choix que fait un programmeur au niveau du langage est dû à un certain nombre de critères de sélection. Ainsi, si votre problème ne doit se poser qu'une seule fois et que la rapidité d'exécution ne constitue pas un facteur primordial quant à la solution à apporter, un langage tel que le basic convient parfaitement (traduction de l'algorithme en basic aisée, mise au point rapide, programmation facilement compréhensible...). Par contre si vous devez utiliser le logiciel plusieurs dizaines de fois, que la rapidité d'exécution est essentielle, que l'occupation mémoire doit être minimum, alors le langage machine s'impose.

à suivre



INTRO A L'ASSEMBLEUR PRATIQUE

Pourquoi deux formations à l'assembleur différentes, d'abord? Parce que ceux qui connaissent déja un peu n'on' pas besoin de réapprendre, oeuf corse, on leur fournit donc une formation completement théorique (quand vous arriverez au complément a deux, vous allez rire! et c'est le deuxieme chapitre). Quant a ceux qui ne connaissent rien, il est totalement illusoire de vouloir apprendre dans des livres dont le titre est déjà incompréhensible. Or donc, voilà comment çà va se passer.
Nous allons pratiquer par machine (ZX81, SPECTRUM, ORIC, APPLE, COMMODORE, TO7...) en utilisant le plus grand nombre possibles d'exemples concrêts, mais spécifiques à cette machine. D'autre part, nous nous référerons à un assembleur précis pour chaque machine, car la syntaxe varie d'un assembleur à l'autre. Nous vous conseillerons aussi les livres utiles (ceux qui sont bons, pas chers, accessibles, et compréhensibles, oui, y'en a) et éventuellement nous mettrons en place un équivalent du softparade, mais réservé à l'assembleur, c'est-a-dire librairie et softs.
Cette formation sera très originale: elle sera compréhensible. La preuve? Le premier chapitre s'appelle "Bonjour". C'est pas trop difficile, non? Et on commence très fort avec le Z80, sur ZX 81. Ceux qui ont un Spectrum peuvent jeter un coup d'oeil, et les autres ont le choix entre acheter un ZX 81 et attendre les prochains articles.
De toutes façons, on débarquera à l'improviste chez des lecteurs pris au hasard pour vérifier qu'ils ont bien appris.

LANGAGE MACHINE... ZX81

Bon! Calez-vous bien en face du mini-clavier de votre ZX-81, allumez juste derriere lui le bloc énorme qui vous sert de moniteur video et chaussez les lunettes noires qui éviteront peut-etre a vos yeux de gonfler au fur et a mesure de votre progression dans les arcanes de la programmation machine. Il ne vous reste plus qu'a mettre sous tension votre ordinateur favori et a imaginer sous le petit boitier noir une gigantesque ruche faite de plusieurs milliers d'alveoles: autant de cases que vous, laborieuse petite abeille, vous efforcerez d'utiliser au mieux de vos intérêts. Mais avant la récolte du miel, si nous allions un peu butiner à l'exterieur...

En fait le jeu est tres simple. Chacune de ces cellules est...une case mémoire. Chaque case est numérotée (pour éviter toute confusion): son numéro est ce qu'on pourrait appeler (et même, j'y pense! ce qu'on appelle!) son adresse. Il y a donc la case numéro 1, la case numéro 10, et entre les deux, je vous le donne en mille...eh! oui! les adresses 2,3,4,5,6,7,8 et 9. Quand votre joli ZX est dépouillé de toute ornementation (pardon: extension!) vous pouvez considérer que vous êtes à la tête d'un empire de 17408 cases, empire peau-de-chagrin qui n'est déjà plus qu'une cité, plus qu'une rue même, et dont toutes les habitations sont des cases uniformes numérotées de 1 à 17408. Sur le plan urbanisme c'est un peu tristounet mais sur le plan gouvernemental -celui du programmeur que vous êtes- c'est beaucoup plus facile à gérer.

Vous voulez peut-être visiter? Une promenade au hasard? Passez donc au numéro 13 vous y verrez un superbe "64", au numéro 6 un agréable "203", au numéro 54 un authentique "136"... Ainsi de suite jusqu'à ce que les qualificatifs vous manquent! Comment faire? Enfantin: tapez un PRINT PEEK 13, un PRINT PEEK 6, un PRINT PEEK 54 et vous verrez par qui -soyons sérieux: par quoi la case en question est occupée.

Mieux. Aidez-vous du basic Sinclair pour explorer la mémoire de votre machine. Ce pitchounet programme suffit:
10 FOR I = 1 TO 17408
(succession des adresses)

20 SCROLL
(permet l'affichage en continu)

30 PRINT "A L ADRESSE ";I;" SE TROUVE "; PEEK I
(lecture du contenu de la case mémoire d'adresse I)

40 NEXT I
(au suivant!)

Faites tourner ça pendant quelques minutes (quelques heures!) et vous explorerez de façon systématique le contenu de chacune des cases mémoire de votre glorieuse machine. N'arretez que lorsque vous aurez constaté ceci: les valeurs obtenues s'échelonnent de 0 à 255. Jamais moins, jamais plus!

Pourquoi? C'est très simple: chacune de ces cases mémoire -et comme nous ne cesserons de le répéter: elles sont toutes identiques-chacune a une capacité d'accueil limitée. Comme un ascenseur qui ne peut plus grimper au-dela de x kilos de charge, notre case ne peut accueillir une valeur décimale supérieure a 255, et il va bien falloir s'en contenter...

Deuxième expérience: modifiez la ligne 10 de votre programme par:
10 FOR I = 17100 TO 17408
Autrement dit on file sans s'arrêter à l'autre bout de la rue et on recommence nos visites. Et la, surprise! Toujours le même resultat: 0,0,0... Plutôt desertique par ici, n'est-ce pas! Ne perdez pas plus de temps et "breakez" au bout d'une vingtaine de ces nullités.Explication: encore une fois c'est elémentaire: une partie de la mémoire (le début de la rue) est déja habitee par des êtres mathematiques dont les valeurs sont comprises entre 0 et 255. C'est Lord Sinclair qui les a logés là définitivement: c'est grâce a eux que votre ordinateur vous parle Basic: c'est d'eux que nous allons essayer de nous affranchir puisque notre but est de quitter le Basic pour parler directement à la machine dans son langage. On appelle cette zone, habitée a titre definitif par des locataires basiqueux, la ROM (ou MEMOIRE MORTE car tout y est figé). La ROM informatique, sachez-le, s'etend de l'adresse 1 a l'adresse 16383. Nous n'y pouvons plus rien. Le ZX 81 sans son extension c'est la crise du logement: la majeure partie des cases sont déja habitees! Voyons ce qui nous reste...

Que se passe t-il au-dela du numéro 16383? Rejouissez-vous c'est la que commence vraiment votre domaine, la RAM (ou MEMOIRE VIVE, la ou ca démenage!). C'est un MONOPOLY ideal: vous devenez proprietaire de toutes les cases memoire de la RAM et pouvez y loger a votre guise les locataires mathematiques de votre choix a condition toutefois que ceux-ci soient toujours compris entre les valeurs décimales 0 et 255. Bien sûr, vous êtes pressé de vérifier! Alors essayez donc: POKE 17100 , 255 (traduction française: place a l'adresse 17100 la valeur decimale 255). Si 255 vous déplait choisissez votre chiffre porte-bonheur: chacun est maitre dans sa RAM! Et constatez votre puissance de demiurge en trappant: PRINT PEEK 17100. Dans mon cas je le jure!-restitution du 255. Et dans le vôtre?

C'est le miracle de la RAM! Mais attention, une simple coupure de courant et c'est la catastrophe que, j'en suis sûr, vous connaissez deja... La désertification serait instantanée dans les cellules de la RAM! (Les occupants de la ROM étant, eux, d'une résistance a toute épreuve!)
Mais en dehors du jeu peu excitant qui consisterait à compter indéfiniment de 0 à 255, qu'allons-nous faire du fabuleux pouvoir des PEEK (lire dans une case mémoire) et des POKE (écrire dans une case mémoire de la RAM)? Eh bien nous pourrions faire autant que le Basic du Sinclair...Il n'est après tout composé que d'une succession de valeurs comprises entre 0 et 255. Tout est affaire de combinaisons. Mais rassurez-vous nous ne nous attellerons pas à cette tâche trop lourde. D'ailleurs copyright oblige! Non, le jeu passionnant auquel HEBDOGICIEL vous convie aujourd'hui et dans ses prochains numéros consistera à doter votre ZX-81 de fonctions ponctuelles qui viendront s'ajouter à celles dont son créateur l'a déjà pourvu. On appelle ces fonctions ponctuelles des "routines", mais vous verrez, rien de routinier dans ce travail d'exploration et de création. Il vous permettra, si vous le pratiquez régulièrement avec nous, de faire de votre ZX-81 une machine plus puissante (extension du basic) et plus attrayante (nous pourrons réaliser en langage machine de nombreuses routines graphiques:inversion vidéo paramétrable, spirale, rotation d'images etc.).
A bientôt! Bernard GUYOT

COURS D'ASSEMBLEUR

PRESENTATION

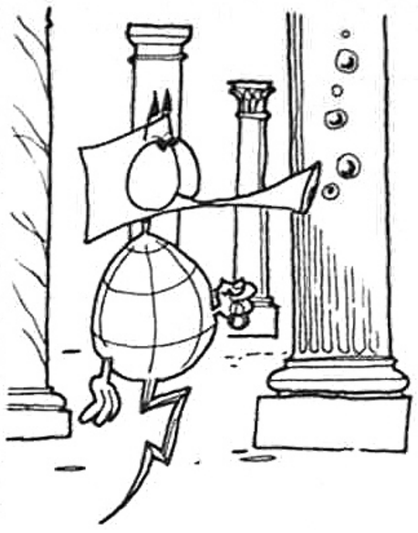
Depuis la semaine dernière vous pouvez trouver dans nos colonnes un cours d'assembleur décomposé en deux parties essentielles.

La première se compose d'un cours théorique sur le fonctionnement des ordinateurs, valable pour l'ensemble des micros-ordinateurs.
La deuxième forme un cours spécifique à un micro-processeur d'un micro-ordinateur, composé d'applications immédiates en langage machine.

Pour la seconde partie, un roulement de différentes machines est prévu. Ainsi chaque micro ne sera-t-il pas désavantagé ou avantagé. Pour l'instant, les propriétaires de ZX81 vont continuer à se réjouir: cette semaine, le cours concerne encore le langage machine sur le Z80. Par la suite viendront le 6502 d'APPLE, le 6809 de Thomson, le 6510 du Commodore 64, le 6502 d'ORIC... Et bien d'autres micros! Nous espérons que cette formule satisfera l'ensemble des lecteurs.

Il est toutefois nécessaire de préciser que la partie théorique ne s'adresse pas aux "fort-en-maths" ni aux professionnels de l'informatique. Il est effectivement possible d'utiliser le langage machine d'un micro-ordinateur sans connaître les méthodes de calcul de celui-ci -la majorité des autodidactes du langage machine se débrouillent très bien sans ces connaissances-, mais des bases solides à propos de la théorie concernant les micros-ordinateurs permettent une meilleure compréhension de la machine et ainsi une utilisation optimisée de celle-ci. Avis aux amateurs...

Michaël THEVENET



COMMENT EST REPRESENTEE L'INFORMATION DANS UN ORDINATEUR

Tout ordinateur, pour travailler, manipule des informations numériques ou sous forme de caractères. Nous allons nous attacher à étudier maintenant la représentation en machine de l'information.

REPRESENTATION INTERNE

Toutes les informations utilisées ou traitées par l'ordinateur sont stockées dans la mémoire de celui-ci sous forme de groupes de BITS. Bit est l'abréviation de BINARY DIGIT, ou encore chiffre binaire (c'est-à-dire que chaque bit ne peut prendre que deux valeurs: soit 0, soit 1). La quasi-totalité des micros-processeurs utilisés dans les ordinateurs familiaux travaillent sur des groupes de huit bits, groupes appelés OCTETS. Dans certaines représentations, on utilise ausi des groupes de quatre bits ou QUARTETS.

Un ensemble exécutable par un ordinateur se décompose en deux parties fondamentales:

-Le programme à proprement parler, composé d'une séquence d'instructions
-Les données sur lesquelles le programme va travailler qui peuvent être des nombres ou des caractères alphanumériques.

Il en résulte donc que notre étude va porter sur trois types de représentation:

-Le programme
-Les données numériques
-Les données alphanumériques.

1. Représentation d'un programme

Toutes les instructions d'un programme sont représentées par un ou plusieurs octets dans la machine. Deux types d'instructions sont à distinguer:

-Les instructions dites courtes sont constituées d'un octet unique, suffisant pour décrire l'instruction que le micro-processeur exécutera
-Les instructions dites longues ou composées qui comportent au minimum deux octets pour pouvoir être exécutables par le micro-processeur.

Il semble normal, à la lumière de ce qui précède, que les instructions courtes soient plus rapides à l'exécution que les instructions composées. Ainsi, une bonne maîtrise des instructions courtes permettra une meilleure optimisation des programmes.

2. Représentation des données numériques.

Depuis la maternelle nous utilisons la base 10 pour compter, effectuer des opérations et des calculs. Mais comme nous l'avons vu plus haut, un ordinateur travaille en binaire (base 2) ce qui met en valeur la conversion nécessaire pour se faire comprendre par le micro-ordinateur lorsque l'on veut traiter des données numériques. Cette conversion doit permettre de répondre à plusieurs critères:

a. Représenter des entiers
b. Représenter des nombres signés (c'est-à-dire des nombres positifs ou négatifs)
c. Représenter des nombres décimaux (en base 10)
d. Représenter des réels.

Cet ensemble de représentations des données numériques fait appel à de nombreuses notions mathématiques et logiques que nous allons expliciter pour une bonne compréhension de ces notions fondamentales.

a. Représentation des entiers

La première conversion qui peut venir à l'esprit consiste en une représentation sous forme BINAIRE DIRECTE. C'est-à-dire que l'on calcule la valeur binaire d'un nombre décimal. Par exemple 8 en décimal correspond à 1000 en binaire.
Comment, par quelle magie obtient-on ce résultat ? Un rappel de maths s'impose:

-dans le système décimal, le chiffre le plus à droite correspond à 10 à la puissance zéro, le chiffre immédiatement à gauche correspond à 10 à la puissance 1, le chiffre suivant à 10 puissance 2...
-dans le système binaire, le bit le plus à droite correspond à 2 puissance zéro, le bit immédiatement à gauche à 2 puissance 1, le bit à gauche à 2 puissance 2...

Nous avons donc 8 en décimal, ce qui donne effectivement 8* 10î0= 8 (rappelons qu'un nombre élevé à la puissance zéro est toujours égal à 1; par convention le symbole "î" intégré dans une expression mathématique indique l'élévation à la puissance).
D'autre part, nous avons 1000 en binaire ce qui nous donne en décimal: 1 * 2î3 + 0 * 2î2 + 0 * 2î1 + 0 * 2î0 = 2î3 = 8

Quel est le nombre maximum que nous pouvons représenter sur un octet (sous la forme binaire directe) ?
Pour effectuer ce calcul, nous allons tout d'abord numéroter les bits d'un octet. Cette numérotation devra être facile à mémoriser d'où celle utilisée dans l'ensemble de la littérature informatique:

les bits sont numérotés de DROITE A GAUCHE, de ZERO A SEPT.

Nous avons donc la représentation suivante:

b7 b6 b5 b4 b3 b2 b1 b0

Maintenant, nous pouvons aisément calculer la valeur du plus grand nombre décimal que nous pouvons coder en binaire direct:

N = b7 * 2î7 + b6 * 2î6 + b5 * 2î5 + b4 * 2î4 + b3 * 2î3 + b2 * 2î2 + b1 * 2î1 + b0 * 2î0

Pour que N soit maximum, il faut que tous les bits soient égaux à 1, donc:

N = 2î7 + 2î6 + 2î5 + 2î4 + 2î3 + 2î2 + 2î1 + 2î0
N = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1
N = 255

Voyons d'autres exemples de conversion binaire-décimal:

Que vaut, en décimal, le nombre binaire 10011001 ?

1* 1 = 1
0* 2 = 0
0* 4 = 0
1* 8 = 8
1* 16 = 16
0* 32 = 0
0* 64 = 0
1* 128 = 128

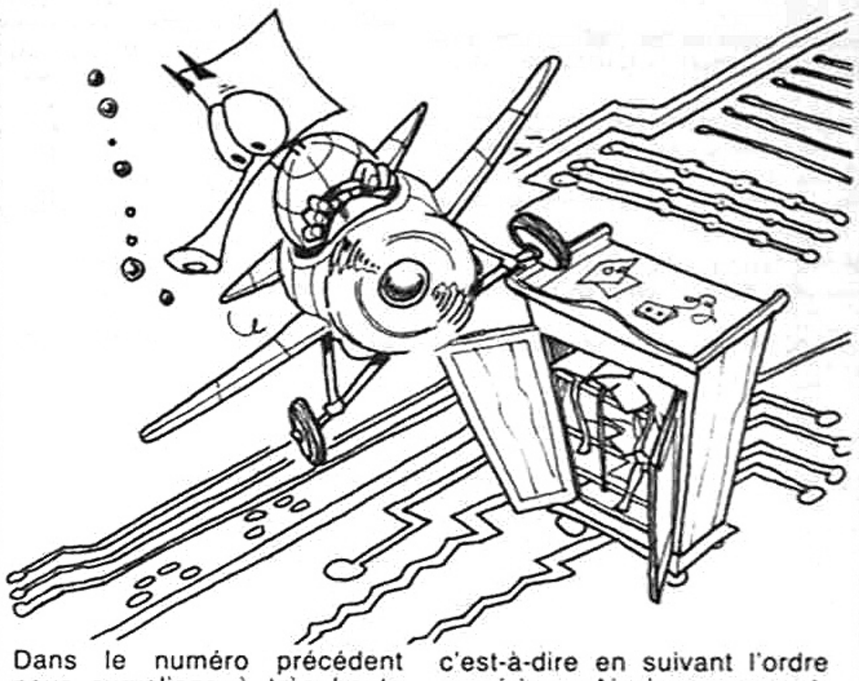
10011001 = 145

Maintenant, voici quelques exercices d'application dont vous pourrez trouver la correction dans le prochain numéro:

EX1: convertir en décimal les nombres binaires suivants
a. 11110000
b. 10000000
c. 10101010
d. 1001010011001010

INTRO A L'ASSEMBLEUR PRATIQUE

LANGAGE MACHINE... ZX81



Dans le numéro précédent nous survolions à très haute altitude les paysages internes du ZX81. Si vous ne vous souvenez pas, allez fouiller dans la vieille armoire où s'empilent semaine après semaine les listings et autres articles précieux de ce votre bon vieil Hebdo. Vous y trouverez l'esquisse topographique secrète qu'aujourd'hui je vous propose de détailler d'avantage. Autrement dit, cramponnez-vous! Nous amorçons la descente vers cet objectif ultime que sera pour nous l'unité centrale du ZX81.

On perd de l'altitude? Tant mieux! Là où nous ne distinguions que deux types de paysages: d'un côté la mégalopole Basic grouillante de valeurs numériques diverses -la ROM-, et de l'autre les terres vierges mais pourtant fertiles de la RAM. Nous sommes maintenant capables d'apercevoir une sorte de zone frontière entre elles, un troisième état des choses. De quoi s'agit-il? Chauffeur un peu plus bas s'il vous plaît...

Là. Nous sommes maintenant exactement à la verticale d'une toute petite zone, très restreinte comparativement à la ROM et qui, en vérité, fait déjà partie de la RAM. Une sorte d'oasis de chiffres à la lisière de ce désert de 0 (car vous venez de brancher votre ordinateur et sa mémoire vive est vierge). C'est seulement sur 124 cases mémoires (on dit aussi octets), sur 124 octets donc que cette zone s'étend; très exactement de l'adresse 16384 à l'adresse 16507. Cette partie de la mémoire vive du ZX81 a une fonction bien précise: c'est une messagerie utilisée par le système Basic de la ROM voisine. Chacun de ces 124 octets sert en quelque sorte de boîte postale au Basic du Sinclair, celui-ci venant y déposer des informations relatives au fonctionnement dynamique du système. Au fur et à mesure que votre ordinateur traite un programme en basic, il utilise cet ensemble de cellules mémoire pour y stocker ses propres repères, ceux-ci étant provisoires, mouvants, dynamiques pendant l'exécution d'un programme, il fallait bien évidemment utiliser la RAM qui seule permet d'écrire, puis d'effacer et de remplacer le contenu d'une case mémoire.

Un exemple simple qui vous fera immédiatement comprendre l'utilisation de ces "boîtes postales": lors de l'exécution d'un programme basic l'ordinateur traite les lignes du programme de façon séquentielle, c'est-à-dire en suivant l'ordre numérique. Ainsi commence-t-il par le numéro de ligne le plus bas pour terminer avec le numéro de ligne le plus élevé (si l'on excepte les instructions GOTO ou GOSUB). Pour l'aider dans cette tâche ingrate il se sert de deux octets dans lesquels il stocke, au fur et à mesure du déroulement du programme, le numéro de ligne de l'instruction en cours d'exécution. Ainsi ne risque-t-il pas "d'oublier" le numéro de ligne pendant le traitement quelques fois long et complexe d'une information; ainsi peut-il cheminer sereinement...et séquentiellement. Encore une fois soyons précis: les deux octets utilisés pour le travail particulier dont nous venons de parler se trouvent exactement aux adresses 16391 et 16392. Cet ensemble, bien utile reconnaissez-le, porte le joli nom de PPC. Mais si celui-ci ne vous plaît pas vous pouvez le rebaptiser car ce nom n'a de signification que pour nous (nous les individus férus de programmation en langage machine!). Il n'est pas reconnu comme nom de variable par le système Basic. Inutile donc d'interroger inlassablement votre pauvre machine en tapant et retapant au clavier PRINT PEEK PPC! Encore une fois PPC n'est pas un nom de variable, ce n'est qu'une mnémonique.

Vous vous demandez peut-être s'il y a beaucoup de mnémoniques à retenir dans la mémoire, vive ô combien, de votre encéphale? Autant que de chandelles à observer après vous être cogné de rage la boîte crâneuse contre la paroi visible de votre tube cathodique le plus proche. N'essayez pas! Ca fait trente six.

Parmi les plus aimables de ces mnémoniques citons pêle-mêle: aux adresses 16388 et 16389, RAMTOP, toujours disponibles pour indiquer au basic, ou à nous-mêmes, la capacité de la mémoire vive de votre machine et consécutivement l'endroit où commence le néant informatique; VARS, d'adresse 16400 et 16401, qui n'accepte de fournir d'information que sur l'emplacement où sont stockées, dans la RAM, tous les noms et toutes les valeurs des variables d'un quelconque programme basic; D-FILE enfin, qui rend visible l'invisible en précisant quelle zone de la mémoire le ZX va recopier sur l'écran de votre téléviseur, qui nous indique en fait l'adresse en mémoire de ce qu'on appelle le FICHIER D'AFFICHAGE.

Ainsi RAMTOP, VARS et D-FILE représentent-ils des valeurs qui sont à chaque moment susceptibles de varier. Ce sont donc des "variables", mais comme elles ne sont utilisées que par le système Basic, sans même que nous nous en rendions compte, on les nomme VARIABLES SYSTEME. Logique non? Ces variables ont donc un nom fictif (mnémoniques) et une adresse (généralement sur deux octets). Chacune de ces adresses contient une valeur qui représente en fait une autre adresse. Ce système qui peut au premier abord vous sembler d'une complication un peu gratuite est en réalité très commode et très utilisé en programmation machine. On appelle alors ce type de variable un POINTEUR. Pointeur?! Me direz-vous? Je m'explique en reprenant pour exemple l'une de ces variables système: RAMTOP. RAMTOP siège donc à l'adresse 16388 et 16389. Si j'interroge ces deux octets pour connaître la valeur qu'ils contiennent, ceux-ci vont me donner un nombre qui correspondra là encore à une adresse: pour RAMTOP cette adresse représentera l'emplacement du dernier octet de mémoire vive disponible additionné de 1. Soyons clair: avec les pointeurs on sait ce que l'on cherche sans savoir ce que l'on va y trouver! Je vous invite d'ailleurs à vérifier ce linpide axiome en lisant vous-même, grâce à l'instruction PEEK, le double contenu de ce pointeur du néant qu'est RAMTOP. Ce sera pour vous la possibilité concrète et immédiate de contrôler la quantité de mémoire réellement à votre disposition sur votre ordinateur, et à l'octet près s'il vous plaît!

Pour lire une valeur stockée sur deux octets d'adresse N, appliquez la formule générale suivante: PRINT PEEK N + 256 * PEEK (N+ 1)
soit pour RAMTOP : PRINT PEEK 16388 + 256 * PEEK 16389.

Retranchez un au résultat ainsi obtenu et vous ferez "pointer" RAMTOP sur l'adresse du dernier octet de votre mémoire. Et si vous trouvez que c'est un peu court il ne vous restera qu'à acheter un module d'extension mémoire supplémentaire pour votre ZX!

Mais vous, lecteur si pressé de programmer directement en langage machine, vous vous demandez peut-être pourquoi ce détour par les variables système qui, après tout, ne concernent que le fonctionnement du ZX sous Basic? Et là, impatient ami, je dois vous détromper, vous les yeux décillerl... Non! Nous ne referons pas le monde! Non, nous ne renierons pas totalement le travail admirable de Sir Clive! Nous mépriserons le Basic, certes; mais continuerons d'exploiter certaines bases du système Sinclair en-deçà du Basic lui-même. Notre prochain rendez-vous, qui sera consacré au fichier d'affichage (réalisation d'une inversion vidéo logicielle), suffira amplement à vous le prouver si, malgré notre plongée commune dans l'intimité des gouffres informatiques, vous ne m'accordez pas encore votre confiance.

Bernard GUYOT

COURS D'ASSEMBLEUR

Depuis six semaines déjà, vous pouvez trouver dans nos colonnes un cours d'assembleur décomposé en deux parties essentielles (l'une théorique, l'autre pratique).

Cette semaine, le cours pratique concerne à nouveau le Z80 du ZX 81. Le prochain numéro concernera pour la seconde fois le 6502 d'Apple.

Jusqu'à présent, les cours publiés sont:
N° 55 → ZX 81
N° 56 → ZX 81
N° 57 → APPLE
N° 58 → ORIC 1, ATMOS
N° 59 → TO7, TO7 70
N° 60 → COMMODORE 64

Michaël THEVENET

P.S. Pour tous ceux qui désirent acquérir les cours manquants, vous pouvez commander une photocopie de ceux-ci à l'hebdo, en précisant clairement lesquels vous intéressent. N'oubliez pas de joindre une enveloppe timbrée et libellée à vos nom et adresse!

Comment est représentée l'information dans un ordinateur

2. Représentation des données en machine

Nous vous donnons tout d'abord quelques exemples pour vous indiquer la marche à suivre.

```
   6        00000110
 + 8      + 00001000
= 14      = 00001110
            V= 0 C= 0
```

Le résultat est exact.

```
  127       01111111
 + 3      + 00000011
= 130     = 10000010
            V= 1 C= 0
```

Le résultat est faux. En effet, si l'on convertit le résultat binaire, on obtient: -126.

EX 8 effectuez les additions suivantes (après conversion en binaire) en indiquant le résultat, la retenue C, le débordement V, et en vérifiant l'exactitude du résultat.

a. 16 + 64 = ?
b. (-65) + (-63) = ?
c. (-6) + (-7) = ?

Jusqu'ici, nous avons étudié la représentation des nombres entiers en machine. Nous sommes parfaitement capables (grâce à la méthode de représentation en complément à deux) de manipuler des nombres positifs ou négatifs et d'effectuer des additions sur ces nombres. Nous avons aussi abordé le problème des grands nombres, sans le résoudre.

Nous allons étudier maintenant cet aspect du problème. Nous savons que pour effectuer des opérations arithmétiques sur des grands nombres, il est nécessaire de travailler sur plusieurs octets. Pour ces opérations nous devons choisir le nombre d'octets que nous utiliserons et nous y tenir. C'est ainsi que nous pourrons prétendre à une certaine efficacité dans les calculs. A chaque nombre d'octets que nous choisissons est associé un nombre maximum et un nombre minimum, automatiquement déductible de ce nombre d'octets. Le nombre d'octets choisi restera stable durant tous les calculs. Ce nombre est appelé FORMAT. La méthode qui consiste à utiliser un nombre d'octets stable s'appelle FORMAT FIXE.

EX 9: déterminez les plus grands et les plus petits nombres que l'on peut représenter en complément à deux
a. sur deux octets
b. sur trois octets

LA GRANDEUR DES NOMBRES :

Jusqu'à présent, nous n'avons effectué d'additions que sur des nombres de huit bits. Pourquoi travailler sur un seul octet? Tout simplement parce que l'ensemble des micros familiaux utilisent des micro-processeurs huit bits (tels les 6502, 6510, Z 80). Nous avons néanmoins besoin de nombres supérieurs à 127 ou inférieurs à − 128 dans de nombreux programmes.

Pour pouvoir traiter des nombres plus petits ou plus grands que ceux imposés par les micro-processeurs huit bits, nous allons utiliser une méthode de calcul appelée MULTI-PRECISION. Cette multi-précision fait appel à des nombres mémorisés sur deux, trois voire quatre octets. Vous constatez donc que le calcul en multi-précision fait appel à la notion de format fixe.

Nous allons, tout d'abord, examiner quelques exemples de représentation en DOUBLE PRECISION. Le format est alors de deux octets. La représentation s'effectue donc sur seize bits.

Décimal	Binaire
0	00000000 00000000
1	00000000 00000001
32765	01111111 11111101
− 1	11111111 11111111
− 3	11111111 11111101

Nous pouvons nous rendre compte assez rapidement que cette méthode de représentation pose des problèmes spécifiques:

1. Quel que soit le nombre que nous souhaitons représenter, nous sommes obligés d'utiliser seize bits, même si le nombre se contenterait de huit bits pour être codé (regardez par exemple 0 ou 100). Cette méthode est donc une grande consommatrice de mémoire, puisqu'il faudra tout stocker sur deux octets.

2. Les micro-processeurs utilisés par les micro-ordinateurs familiaux traitent majoritairement les calculs huit bits par huit bits. Il faut donc travailler en plusieurs étapes pour accomplir une opération multiprécision. Cela ralentit le traitement des calculs.

Pour ces deux raisons, on limite généralement la multiprécision à un format de quatre octets (trente deux bits) au maximum lorsque l'on utilise un micro-processeur huit bits.

Un autre point important est à souligner. Quel que soit le format de calcul choisi, si lors d'un calcul le résultat dépasse le format choisi, certains des bits du résultat seront perdus. Généralement les micro-processeurs sont programmés de telle manière qu'ils conservent les bits les plus significatifs (bits de "gauche") et perdent volontairement les bits les moins significatifs (bits de "droite"). Cette programmation des calculs s'appelle "TRONCATION DES RESULTATS" (ou troncature).

Pour éclaircir cette méthode, nous allons étudier un exemple d'opération en décimal. Nous choisissons un format à huit chiffres significatifs.

```
    12345678
 x       1,4
    49382712
  + 1234567
= 17283949,2
```

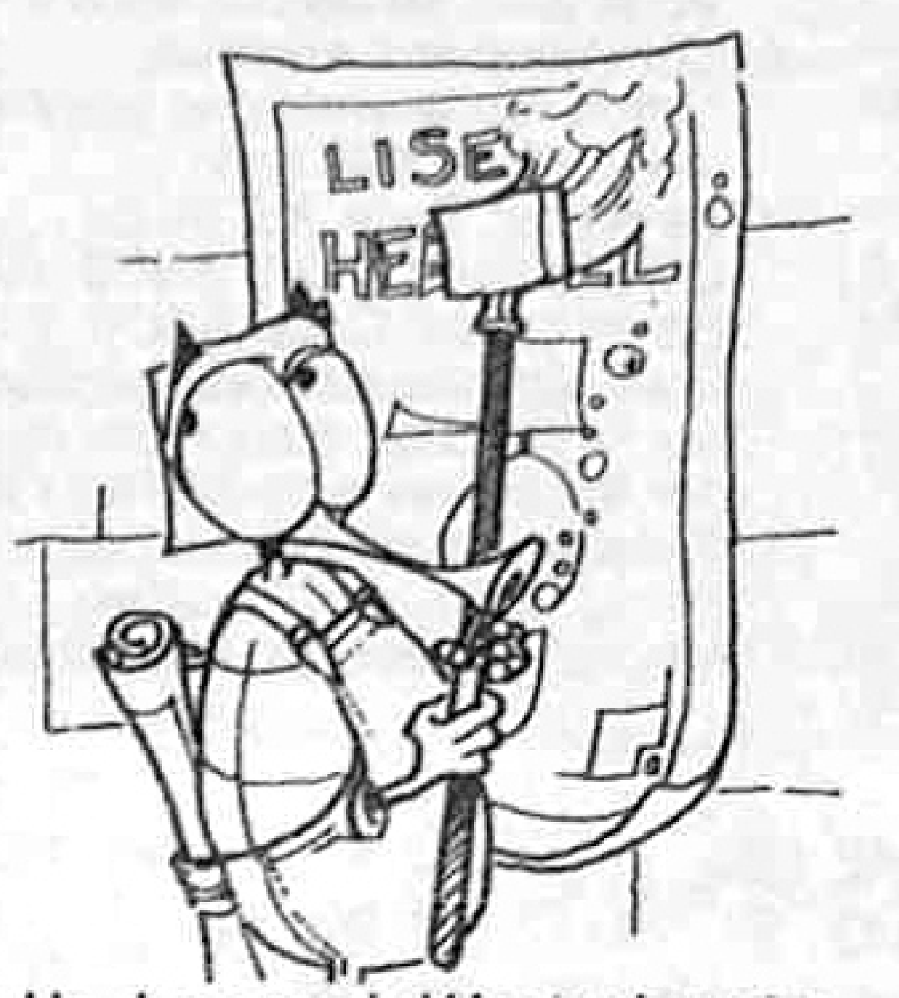
Vous vous rendez compte que le résultat exige neuf chiffres. Le deux qui suit la virgule sera éliminé par le processeur. Le résultat final de l'opération est donc: 17 283 949. Il a été tronqué pour pouvoir correspondre au format fixé au départ.
On utilise généralement cette méthode de troncation des résultats tant que la position de la virgule n'est pas perdue. On étend ainsi la gamme des opérations possibles en perdant toutefois la précision des calculs. La représentation des grands nombres en format fixe conduit donc à obtenir des résultats approchés et non plus exacts pour les multiplications et les divisions (les résultats intermédiaires étant tronqués au fur et à mesure). Malgré tout ces calculs restent suffisamment précis pour les utilisations mathématiques courantes.
Dans certains domaines, il est quand même nécessaire d'avoir en permanence des résultats corrects et non approximés.
Supposons par exemple que les caisses enregistreuses fonctionnent avec ce principe de troncature. Les sommes demandées aux clients ne seraient jamais exactes et cela au détriment des vendeurs. Vous comprenez donc que ce n'est pas admissible (au moins pour les vendeurs). C'est pour cette raison qu'un autre système de représentation des nombres a été développé, de manière que la précision soit toujours parfaite, quel que soit le calcul à accomplir. La solution normalement et couramment adoptée est la représentation "DCB". Cette abréviation signifie Décimal Codé Binaire.

REPRESENTATION DCB

Le principe de cette représentation est de coder chaque chiffre décimal en utilisant autant de bits que nécessaire pour représenter toute l'information. Combien de bits sont nécessaires pour coder les chiffres de 0 à 9? Il en faut quatre. En effet, si nous nous contentions de trois bits, nous disposerions de huit combinaisons seulement. Quatre bits nous permettent de disposer de seize combinaisons. Dix chiffres et seize possibilités? Nous avons donc six codes qui seront inutilisés. Ces six codes inutilisés risquent de provoquer des erreurs lors d'additions ou de soustractions. Il faudra donc prêter une attention particulière à la programmation de ces opérations. La représentation DCB ne nécissitant que quatre bits, certains ont pensé à coder deux chiffres décimaux par octet. Cette représentation est nommée "DCB compact".

à suivre

INTRO A L'ASSEMBLEUR PRATIQUE
Langage machine... ZX 81



Un des grand défauts de votre petite machine réside dans son système d'affichage sur l'écran familial du téléviseur. En effet, contrairement à la plupart des systèmes actuellement en vente sur le marché, le ZX 81 nous délivre une image en vidéo inversée. Non! Non! N'allez pas retourner votre télé les quatre pieds en l'air, je veux simplement dire par là que l'affichage normal des caractères s'effectue en noir sur fond blanc et non pas en blanc sur fond noir, ce qui est de règle habituellement avec les micros. Le nôtre semble prendre sa sortie vidéo pour une sortie imprimante et nous offre un texte peu encré sur une feuille d'un blanc trop lumineux et cela, sur un écran cathodique! Sommes-nous donc, pour toujours, voués aux verres filtrants et aux consultations ophtalmologiques?
Non, me répondrez-vous car on peut encore utiliser la fonction graphique: vous passez en curseur G et les lettres frappées au clavier s'afficheront alors en blanc sur fond de pavé sombre. Déjà mieux bien sûr, mais encore insuffisant car là où il n'y a pas de texte, nous retrouvons notre fond blanc aveuglant. De plus, si le ZX se met à écrire lui-même (le résultat d'une opération mathématique par exemple) il utilisera son mode normal d'écriture c'est-à-dire noir sur fond blanc.
La solution radicale à cet état de chose, consiste en la réalisation d'un inverseur vidéo en hard. Dur, dur! Circuit imprimé! Composants! Soudures! Bricolage interne annulant la garantie! Pour toutes ces raisons beaucoup ne s'y risquent pas...
Il nous reste une solution intermédiaire en soft. Comme toutes les solutions logicielles elle présente l'avantage d'être peu coûteuse (pas d'achat de matériel), peu "salissante" (pas besoin d'installer sur la table du salon son mini-atelier d'électronicien) et peu dangereuse (pas de court-circuit, au pire un "plantage" du processeur: fais RESET à ton programmeur et c'est oublié!). C'est cette solution que je vous propose de mettre en oeuvre aujourd'hui.
Si nous réussissons dans notre entreprise, à quels résultats pouvons-nous prétendre? A la réalisation d'un programme qui nous permette de substituer globalement aux caractères ou espaces figurant en mode normal sur l'écran, leurs répliques en vidéo inversée, c'est-à-dire en mode graphique.

On sait que toutes les lettres et la plupart des symboles mathématiques ou logiques sont disponibles dans leur version graphique c'est-à-dire en blanc sur fond noir; ils sont simplement situés à une adresse différente de celle de leurs homologues noir sur fond blanc dans la table de caractères du ZX, dans sa ROM. On sait également que l'adresse de chaque caractère est repérable par l'ordinateur grâce à son numéro de code CHR$. Ainsi au code CHR$ 20 correspond le symbole 0 en noir sur fond blanc (vérifier en tapant en mode direct PRINT$ 20), qu'au code CHR$ 38 correspond la lettre A en noir sur fond blanc. On peut aussi constater qu'au code CHR$ 156 correspond le 0 mais cette fois en blanc sur fond noir de même que code CHR$ 166 correspond le A en mode graphique. Si vous avez à proximité de vous la notice du Sinclair ouvrez-la vite à la page 181: ici commence la liste exhaustive des caractères figurant en mémoire morte et de leurs équivalents CHR$. Examinez-la attentivement car elle détient le secret de notre inversion vidéo logicielle...

Ca y est! Vous avez trouvé j'en suis sûr. Je ne vous révèle le truc qu'au titre de simple confirmation de votre géniale intuition:
1)Qu'ils soient en vidéo normale ou en vidéo inversée les différents caractères de la table se suivent dans un ordre identique. D'abord, les caractères graphiques, ensuite les symboles logiques, puis les 9 chiffres puis les lettres de l'alphabet dans leur ordre habituel.
2)La série des caractères en vidéo normale commence au code 0 tandis que la série des caractères en vidéo inverse commence au code 128.
De ces deux constatations il est aisé de déduire la solution de notre problème: pour obtenir la réplique, en vidéo inversée, d'un caractère en mode normal, il suffit d'ajouter à son code la valeur décimale 128. Vérifiez-le tout de suite en tapant:
PRINT CHR$ (38 + 128). 38 est le code de la lettre A en vidéo normale (VN), l'addition du facteur 128 va permettre de l'afficher en vidéo inverse (VI). C'est gagné ou presque...

N'oublions pas notre but qui est de convertir en VI la totalité des caractères présents sur l'écran en VN, espaces compris. Il faut donc "ratisser" toute la surface de l'écran, autrement dit modifier dans la RAM le contenu de chacune des cases mémoire du fichier d'affichage. Il vous faut donc au préalable savoir où se trouve, dans la RAM, le fichier d'affichage. Facile: il suffit de se reporter à la variabe système D-FILE étudiée dans notre précédent article. On eut alors en interrogeant les deux octets d'adresse 16396 et 16397 connaître l'adresse de début du fichier d'affichage. Il suffit pour cela d'appliquer la formule PRINT PEEK 16396 + 256 * PEEK 16297.

Possédons-nous enfin tous les éléments qui nous permettrons d'aboutir? Pas tout à fait. Il nous manque encore la structure du fichier d'affichage.

Sur l'écran du ZX nous avons à notre disposition 22 lignes de 32 caractères. Chacun de ces caractères vaut pour un octet, dans le fichier d'affichage et les adresses de ces octets se suivent dans l'ordre des caractères affichés sur une ligne et dans l'ordre des lignes affichées sur l'écran. Mais il faut savoir que ce fichier d'affichage (F.A.) commence en réalité par un caractère invisible sur l'écran mais qui n'en occupe pas moins de place en mémoire et dont le code est 118. Il faut également savoir que si en Basic nous disposons de 22 lignes le F.A. en comporte en fait 24, le ZX utilisant d'ailleurs la ligne 24 pour l'affichage de ses codes erreurs. Réjouissez-vous, grâce au langage machine ces deux dernières lignes nous deviendront accessibles. Il faut enfin savoir que chacune des lignes s'achève sur un caractère codé 118 invisible à l'écran mais qui vaut lui aussi un octet en RAM. Récapitulons la structure (essentielle!) du fichier d'affichage: 1 octet contenant 118, puis les 32 caractères de la première ligne 1 octet contenant 118 puis les 32 caractères de la seconde ligne, ainsi de suite jusqu'à l'octet contenant 118 qui termine tout à la fois la 24ème ligne et le F.A. lui-même.

Maintenant, nous pouvons élaborer l'algorithme de notre inversion vidéo logicielle:
1 -Localiser le début du fichier d'affichage grâce à D-FILE.
2 -Identifier le contenu (code) de chaque octet du F.A.
3 -Si sa valeur est différente de 118 lui ajouter 128.
4 -Renouveler l'opération jusqu'à la fin du F.A.
Voilà, il ne reste plus qu'à choisir notre langage de programmation. Je vous propose d'essayer d'abord le Basic pour le transposer ensuite en langage machine. Cette étude comparative mettra en valeur de façon concrète les avantages de l'assembleur et inaugurera la série de routines LM que je vous proposerais au fil de ces articles.

Notre programme d'inversion pourrait donc s'écrire en basic:
10 LET FA = PEEK 16396 + 256 * PEEK 16397
(localisation du fichier d'affichage).

15 LET FA = FA + 1
(on saute le premier caractère qui est un 118).

20 FOR I = FA TO FA + 792
(boucle recouvrant la totalité du F.A. soit 24 lignes de 33 caractères).

30 LET CAR = PEEK I
(Le code du caractère correspond au contenu de la case mémoire d'adresse I)

40 IF CAR < 128 AND CAR <>118 THEN LET CAR = CAR + 128
(S'il ne s'agit pas d'un caractère déjà en VI et si ce caractère n'est pas un Newline alors calculer son nouveau code...)

50 POKE I, CAR
(... puis l'afficher).

60 NEXT I
(Et ainsi de suite, jusqu'à la fin du F.A.)

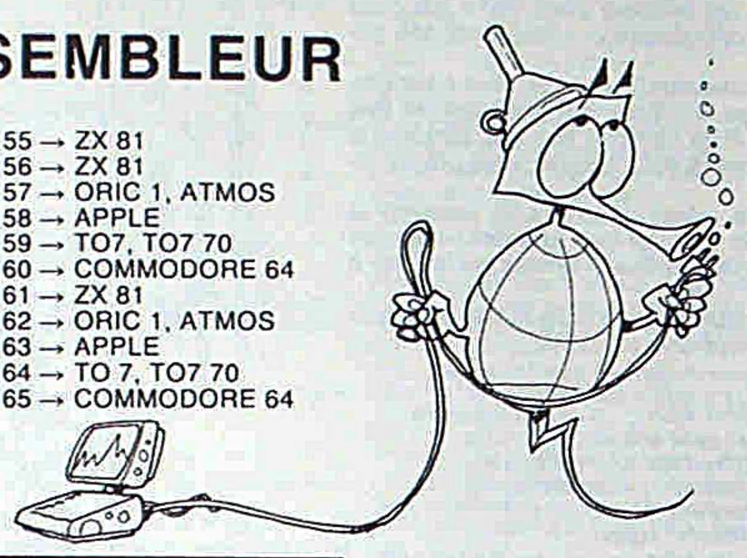
Pour essayer ce programme vous pouvez ajouter en tête du listing un PRINT "MESSAGE A VOTRE CONVENANCE" et vérifier ainsi l'effet d'inversion vidéo. Exemple:
1 PRINT "AVEC LE BASIC IL NE FAUT PAS ETRE PRESSE".

Phrase de circonstance car vous constaterez qu'il faudra plus d'une minute après le RUN pour que le travail de ce logiciel soit achevé. Notre confort visuel et la satisfaction de laisser notre empreinte sur les lignes 23 et 24 jadis interdites nous coûte décidément beaucoup trop de temps. C'est là qu'intervient le langage machine...

Bernard GUYOT

Formation à l'assembleur

COURS D'ASSEMBLEUR



Depuis plusieurs semaines, vous profitez d'un cours d'assembleur décomposé en deux parties principales (l'une théorique, l'autre pratique).

Cette semaine, le cours pratique concerne une nouvelle fois le Z80 du ZX 81. Le prochain numéro consacrera une pleine attention aux angoisses majeures du 6502 de l'Oric 1 et de l'Atmos.

Jusqu'à présent, les cours pratiques suivant ont été publiés:

N° 55 → ZX 81
N° 56 → ZX 81
N° 57 → ORIC 1, ATMOS
N° 58 → APPLE
N° 59 → TO7, TO7 70
N° 60 → COMMODORE 64
N° 61 → ZX 81
N° 62 → ORIC 1, ATMOS
N° 63 → APPLE
N° 64 → TO 7, TO7 70
N° 65 → COMMODORE 64

Comme je vous l'avais annoncé la semaine passée, avant de continuer le cours d'assembleur théorique du point de vue fonctionnement de la machine, nous allons étudier de plus près la conception Hardware (ou matérielle) d'un micro-ordinateur. Pour ce faire, je vais effectuer quelques rappels concernant la logique booléenne qui forme la base fondamentale du "raisonnement" informatique. En effet la conception d'un ordinateur doit se conformer à certains critères très restrictifs. L'un d'entre eux est qu'un ordinateur ne sais rien faire de lui-même. Lorsqu'un ingénieur câble une carte pour réaliser un additionneur (par exemple) il doit avoir en permanence à l'esprit que les seules opérations facilement réalisables sont celles effectuées grâce aux OPERATEURS LOGIQUES "ET", "OU" et "INVERSE".

La logique booléenne travaille sur deux valeurs: VRAI et FAUX. Vous pouvez vous rendre compte immédiatement de la similitude entre une variable booléenne et une variable binaire. Il suffit de considérer que "1" est égal à "VRAI" et que "0" est égal à "FAUX". Nous allons maintenant étudier ce que vous avez dû apprendre au moins une fois au cours de vos études: les tables de vérité. La première à laquelle nous allons nous consacrer fait appel à une notation particulière. Il s'agit de la fonction d'inversion. Si l'on utilise une variable de nom "a", son inverse sera notée "â" (que vous pourrez lire "non-a" ou "a-barre"). En voici la table de vérité:

a	â
0	1
1	0

Table de la fonction INVERSE.

En électronique, un circuit simple permet d'effectuer cet inversion. Il s'agit d'un INVERSEUR.
La deuxième fonction qui nous intéresse ne fait pas appel à des notions particulières. La fonction "ET" ou "INTERSECTION" (dans la terminologie des théories ensemblistes) est basée sur la comparaison de deux variables. En voici la table de vérité:

b \ a	0	1
0	0	0
1	0	1

Table de la fonction AND.

Lorsque nous voudrons écrire qu'une opération d'intersection a lieu entre deux variables, nous le noterons: (a . b). Cette opération "ET" se nomme aussi PRODUIT LOGIQUE (d'où la notation). En anglais, cette fonction s'appelle AND de la même manière que le circuit électronique correspondant.
La troisième opération, tout comme la deuxième, ne fait pas appel à une nouvelle notation. Cette fonction "OU" ou "REUNION" (toujours dans la terminologie ensembliste) fait elle aussi appel à la comparaison de deux variables. Voici la table de vérité qui lui correspond:

b \ a	0	1
0	0	1
1	1	1

Table de la fonction OR.

Lorsque nous désirerons noter qu'une opération de réunion a lieu entre deux variables, nous utiliserons la symbolisation suivante: (a + b). La fonction est aussi appelée SOMME LOGIQUE (d'où la notation). En anglais l'opération se nomme "OR", ainsi que le circuit électronique qui effectue cette opération.

Pour vous permettre de comprendre facilement comment fonctionnent les deux opérations "ET" et "OU", nous allons nous appuyer sur une équivalence simple avec des circuits électriques. Le premier schéma représentera la fonction "ET" alors que le deuxième symbolisera la fonction "OU".

Schéma 1

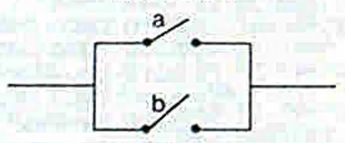


Schéma 2

La correspondance est simple: nous remplaçons chaque variable par un interrupteur (a et b). Si un interrupteur est ouvert, son état est équivalent à 0. Similairement, si ce même interrupteur est fermé son état sera 1. Dans le schéma 1, si nous voulons que le courant circule, il faut que les interrupteurs a ET b soient fermés. Si cette condition n'est pas vérifiée, le courant ne passe pas (état 0 pour l'ensemble du circuit). Pour le schéma 2 vous vous rendez compte tout de suite qu'il suffit que l'interrupteur a OU l'interrupteur b soit fermé pour que le courant circule dans le circuit (état 1 pour l'ensemble du circuit).

Nous allons maintenant voir rapidement un ensemble de règles concernant les opérations ci-dessus. Si elles vous paraissent rébarbatives au premier abord, n'y attachez pas trop d'importance: vous pourrez vous y référer au coup par coup.

(a + 0) = a (a . 0) = 0
(a + 1) = 1 (a . 1) = a
(a + a) = a (a . a) = a
(a + â) = 1 (a . â) = 0

Commutativité: (a + b) = (b + a)
Associativité: ((a + b) + c) = (a + (b + c))
Distributivité: (a . (b + c)) = ((a . b) + (a . c))
(a + (b . c)) = ((a + b) . (a + c))
Absorbtion: (a + (a . b)) = a
Absorbtion cachée: (a + (â . b)) = (a + b)

Théorèmes de Morgan
1. $(\hat{a} + \hat{b}) = \overline{(a . b)}$
2. $(\hat{a} . \hat{b}) = \overline{(a + b)}$

De ces premières fonctions "ET", "OU" et "INVERSE" on ne peut pas encore créer un ordinateur. Ce n'est qu'avec quatre autres fonctions (en relation directe avec les deux premières) que l'on peut envisager de modéliser n'importe quelle opération. Ces quatre nouvelles fonctions se nomment "NON ET" (ou "NAND" en anglais), "NON OU" (ou "NOR" en version anglaise), "OU EXCLUSIF" (ou encore "XOR" toujours en anglais) et enfin "NON OU EXCLUSIF" ("NXOR" in english sir). Nous allons voir les tables de vérité de ces fonctions pour nous éclairer sur leur rôle respectif. Tout d'abord regardons celle du "NON ET" ou "NAND":

b \ a	0	1
0	1	1
1	1	0

Table de la fonction NAND.

Solution des exercices

EX 15 Donner les équivalences hexadécimales et binaires des nombres décimaux suivants

a. 28 = 1C (en hexa)
28 = 0001 1100 (en binaire)

b. 43 = 2B
43 = 0010 1011

c. 128 = 80
128 = 1000 0000

EX 16 Donner les équivalences décimales et binaires des nombres hexadécimaux suivants

a. AA = 170 (en décimal)
AA = 1010 1010 (en binaire)

b. F0 = 240
F0 = 1111 0000

c. FFCA = 65.482
FFCA = 1111 1111 1100 1010

EX 17 Donner les équivalences décimales et hexadécimales des nombres binaires suivants

a. 1100 = 12 (en décimal)
1100 = 0C (en hexa)

b. 0101 = 5
0101 = 05

c. 1110 0100 = 228
1110 0100 = E4

INTRO A L'ASSEMBLEUR PRATIQUE

Langage machine... sur ZX81

Les micros aussi ont envie de vivre vite... Je suis sûr que vous avez su déchiffrer cette exigence au travers de l'exécution, par le ZX, de notre programme d'inversion vidéo Basic. Souvenez-vous comme votre pauvre machine s'essoufflait, caractère après caractère à identifier puis recalculer les codes CHR$, tout cela pour retomber dans la même opposition manichéenne des blancs et des noirs à ceci près que les blancs étaient devenus noirs et les noirs blancs. Si l'on peut aboutir par là à de savoureuses utopies politiques, sur le plan de la stricte informatique le rapport de temps de travail/résultat nous inciterait plutôt à revenir à la bonne vieille machine à calculer de Blaise Pascal...

Décidément c'en est trop du Basic ! A bas la tyrannie interprétorienne ! A bas la lourdeur des procédures et la lenteur d'exécution ! A bas l'occupation des mémoires ! A bas l'administration basique ! A bas la bureaucratie, à bas (sic) !

Pour avoir toutes les chances de réussir cette révolution il est nécessaire que le citoyen informaticien se dote d'un outil efficace. L'instrument qui permettra à chacun de reprendre le contrôle du processeur, l'arme absolue de la libération a pour nom: Assembleur. Je vous propose donc de déterrer vos assembleurs clandestins ou d'acquérir un assembleur simple mais efficace tel le ZX AS. (Mais tout assembleur reprenant les mnémoniques du Z80 – assembler/Editeur Artic par exemple– fera parfaitement l'affaire).

Une partie importante du travail est déjà faite puisque nous avons précédemment, pour notre programme Basic, analysé le problème et défini la démarche à suivre pour aboutir à l'inversion vidéo globale de l'écran. Notre algorithme reste opérationnel, nous n'en changerons donc pas. Seuls varieront, avec le langage, les moyens de l'exprimer. Nous allons donc reprendre point par point cet algorithme et tenter de trouver sa traduction avec quelques instructions simples et courantes du processeur: le Z 80.

Il nous fallait d'abord localiser le début du fichier d'affichage dans la RAM. Pour cela il suffit d'interroger la variable système D-FILE et de conserver l'information qu'elle nous délivre afin que le processeur l'utilise dans son travail (ce que faisait le Basic avec ses variables, mais en langage machine: exit les variables !). Localiser le F.A. cela signifie trouver son adresse en RAM; or nous savons qu'une adresse dans l'ordinateur est stockée sur deux octets par nécessité. Il nous faut donc reprendre cette information dans son intégralité pour la mettre au service du Z80, c'est-à-dire la transférer dans deux des rares cellules mémoire du processeur lui-même, ce qu'on appelle ses registres. C'est uniquement à partir du contenu de ses registres que le processeur peut effectuer ses opérations. Toute la difficulté mais aussi le charme de l'assembleur tient précisément à cette économie de moyens et de mémoire. Il sera nécessaire de décomposer notre programme en petites unités et en opérations élémentaires. C'est seulement leurs nombreux et rapides traitements successifs qui pemettront d'aboutir au résultat recherché.

Je ne décrirais pas ici ce qu'on appelle "l'architecture interne" du processeur (nombre et affectations des registres, types d'opérations, synchronisation, etc...) Ce serait un exposé long et fastidieux qui sortirait du cadre de cette introduction à l'assembleur pratique. Nous allons plutôt, au fur et à mesure de nos besoins, découvrir les possibilités et les ressources multiples du Z80.

Revenons donc à notre problème: transférer au niveau des registres du processeur (autrement dit porter à sa connaissance) la valeur contenue aux adresses 16396 et 16397 de la RAM, valeur qui représente l'adresse du F.A. De même que D-FILE est stockée sur deux octets, de même il nous faudra utiliser deux registres du Z80, chacun d'entre eux ne pouvant accueillir plus de huit bits. Il faudra également que ces deux registres soient reconnus par le processeur comme contenant une seule et unique valeur codée sur deux octets (l'un dit de "poids fort" l'autre de "poids faible"). Fort heureusement, il existe au sein même du Z80 une variété particulière de registres qui se prête particulièrement bien à ce travail. Ces registres peuvent fonctionner par paires (on obtient ainsi 16 bits) et le processeur peut prendre en compte directement la valeur de cette paire. Ces paires de registres sont principalement BC (Registre huit bits B + registre huit bits C), DE et HL (même chose).
Le contenant étant identifié (prenons par exemple HL) comment y porter le contenu ? D'abord par une instruction spécifique de chargement d'un registre: LOAD (mnémonique d'assemblage LD), ensuite en choisissant un "mode d'adressage" approprié c'est-à-dire la manière dont sera spécifié au processeur l'adresse de l'opérande sur laquelle l'instruction (LD) devra travailler. Dans le cas qui nous intéresse le mode d'adressage est dit "absolu": après l'instruction LD HL codée sur un seul octet par votre assembleur (00101010) on précise sur les deux octets indispensables l'adresse de l'opérande (ici l'octet de poids faible 16396, le Z80 passera ensuite automatiquement à l'adresse 16396 + 1 pour lire l'octet de poids fort). Encore une fois c'est votre programme d'assemblage qui se chargera de convertir l'adresse décimale 16396 en valeur binaire sur deux octets. Il réalise ainsi, octet après octet, un programme binaire directement exécutable par le Z80; en résulte un gain de temps énorme dans l'exécution du programme et une substantielle économie de mémoire au niveau de son écriture. Constatez vous-même que notre instruction complète n'occupe que trois octets. Elle s'écrira en langage d'assemblage:
LD HL. (16396).
- Où le point n'est qu'un simple séparateur remplaçant la virgule officielle du code de ZILOG. (plus accessible sur le ZX 81).
- Où les parenthèses qui encadrent notre adresse codent ici le mode d'adressage choisi : l'adressage absolu. Elles indiquent au programme d'assemblage que la valeur précisée représente bien une adresse sur deux octets et non une valeur qu'il faudrait ranger tel quel dans HL (adressage immédiat). 16396 n'est ici qu'un pointeur.

Vous comprenez maintenant que le mode d'adressage est partie intégrante de l'instruction de chargement du registre HL. Il est donc en réalité déjà codé dans le premier des trois octets de notre instruction, les deux suivants étant exclusivement consacrés à la valeur numérique de cette adresse:

1° octet: LD HL:(nn)
2° octet: n adresse poids faible.
3° octet: n adresse poids fort.

Que fait le processeur en rencontrant l'instruction binaire LD HL.(16396): il charge le registre HL à partir de l'emplacement mémoire d'adresse 16396 de la façon suivante:

1° Il range dans L la valeur contenue à l'adresse 16396
2° Il range dans H la valeur contenue à l'adresse 16396 + 1

L'information précisant la position du fichier d'affichage en RAM est maintenant à la disposition du processeur...

Il nous faut maintenant ajouter 1 à cette mystérieuse valeur contenue dans HL afin de pointer sur le premier octet affiché à l'écran. Cette opération pourtant très simple porte un nom compliqué: INCREMENTATION. Et ça se conjugue ! J'incrémente, tu incrémentes etc... Incrémentons donc le registre HL par l'instruction d'assemblage INC HL. Votre curiosité ne sera totalement satisfaite que lorsque vous saurez que le "format" de cette instruction est celui d'un octet solitaire dont le triste matricule se prononce comme il s'écrit: 00100011. Face à une telle dérision les rouages du Zilog ne grincent que d'un cran: le contenu de HL est véhiculé par l'auto-bus (le bus du processeur lui-même) jusqu'à l'alu (l'unité de calcul qui n'est pas en aluminium); celle-ci sans poser de question ni émettre la moindre protestation à l'encontre de ce travail ingrat mais avec l'habileté du chinois sur son boulier... ajoute 1, puis s'en va vaquer à d'autres tâches qui sont en réalité les mêmes. Tandis que Zilog processeur à l'univers cité plus haut réempruntant la même ligne de bus, dépose précautionneusement au terminus HL la valeur ajoutée dont il s'enorgueuillit.

Bernard GUYOT

ERRATUM sur TO7
Cours assembleur n° 64
Une FN error in 20 empêche sans doute votre petit bonhomme de faire sa gymnastique. Rajoutez donc I, J ou NEXT de la ligne 30

Formation à l'assembleur

COURS D'ASSEMBLEUR

Depuis plusieurs semaines, vous profitez d'un cours d'assembleur décomposé en deux parties principales (l'une théorique, l'autre pratique).

Cette semaine, le cours pratique concerne une nouvelle fois le Z80 du ZX 81. Le prochain numéro consacrera une pleine attention aux angoisses majeures du 6502 de l'ORIC 1 et de l'ATMOS.

Jusqu'à présent, les cours pratiques suivant ont été publiés:

N° 55 → ZX 81
N° 56 → ZX 81
N° 57 → ORIC 1, ATMOS
N° 58 → APPLE
N° 59 → TO7, TO7 70
N° 60 → COMMODORE 64
N° 61 → ZX 81
N° 62 → ORIC 1, ATMOS
N° 63 → APPLE
N° 64 → TO 7, TO7 70
N° 65 → COMMODORE 64
N° 66 → ZX 81
N° 67 → ORIC 1, ATMOS
N° 68 → APPLE
N° 69 → TO7, TO7 70
N° 70 → COMMODORE 64

Comme je vous l'avais annoncé la semaine dernière, nous allons nous attacher durant quelques paragraphes à l'Unité Arithmétique et Logique (UAL ou ALU -version anglaise), véritable bastion des calculs au sein de l'ordinateur.

Pour étudier cet organe essentiel de votre machine, nous allons devoir accomplir un court retour en arrière, mettant ainsi notre mémoire à l'épreuve. Lorsque nous avons étudié l'additionneur binaire série, nous avons injecté les données sous forme de variables dans le circuit. Le système se contente alors d'additionner ces deux variables et de donner d'une part la somme d'autre part la retenue. Ce principe souffre malgré tout d'un champ d'activité restreint. C'est dans le but d'améliorer ce principe que l'idée de l'UAL surgit.

La différence fondamentale entre UAL et additionneur tient dans la nature des données que l'on fournit à chacun des deux circuits. Si l'on entre dans l'additionneur des variables, en revanche l'UAL reçoit une fonction de ces variables. Nous n'allons pas détailler le circuit de cette puce (car c'en est une) mais le survoler de manière à en avoir un aperçu significatif.

Nous allons tout d'abord admettre un certain nombre de règles logiques (que vous pouvez tenter de démontrer mathématiquement à l'aide des théorèmes vus au début de ce cours).

1. $A \cdot B \cdot REPORT$ représente une addition

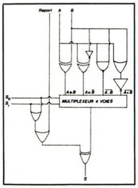
2. $A \cdot \bar{B} \cdot 1 = A \cdot B$

3. $(\overline{A \cdot B}) \cdot 1 = A \cdot B$

4. $(\overline{A + B}) \cdot 1 = A + B$

De ces différentes équations nous pouvons déduire la nécessité de multiplexer l'information à l'entrée du circuit, ce qui nous conduit à dessiner un schéma tel que le suivant pour créer une UAL.

UNITE ARITHMETIQUE ET LOGIQUE



Vous remarquerez les deux entrées de commande S0 et S1. Dans ce circuit nous ne disposons que de quatre fonctions logiques différentes. Si nous avions trois entrées de commande (par exemple) nous pourrions admettre une sélection par multiplexage entre huit fonctions logiques différentes (complémentation, extension de signe...). Vous trouverez ci-dessous la table de vérité correspondant au circuit que nous venons de schématiser.

TABLE DE VERITE

S_0	S_1	S
O	O	A * B * Report
O	I	A * B
I	O	A . B
I	I	A + B

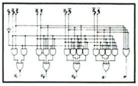
Le report est le seul calcul qui est pratiqué de manière identique dans l'additionneur binaire série et dans l'unité arithmétique et logique. De ce fait lorsque l'additionneur comporte un nombre de bits important (trente deux par exemple) le report va prendre un temps non négligeable (d'un point de vue électronique bien entendu, toutes ces durées étant inférieur au centième de seconde) pour se propager de bit en bit. Pour cela des ingénieurs ont mis au point un circuit d'anticipation de report.

Nous n'étudierons pas ce circuit attentivement. Il vous est seulement nécessaire de savoir qu'il est possible de calculer le report indépendamment du mécanisme de l'addition. En effet trois mécanismes interactifs se font jour lorsque l'on étudie attentivement le schéma d'un additionneur dans tout ce qui concerne le report (ou retenue). Les voici:

1. le report
2. le mécanisme de génération du report
3. Le mécanisme de propagation du report.

Les démonstrations qu'il vous faudrait littéralement élucider pour comprendre le principe de fonctionnement de ce mécanisme n'ont pas de place dans ces lignes. Vous pouvez malgré tout regarder le circuit autorisant l'anticipation du report.

CIRCUIT D'ANTICIPATION DU REPORT



Nous avons vu dans ces derniers paragraphes ce que peut contenir potentiellement une Unité Arithmétique et Logique.

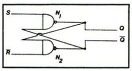
Nous arrivons maintenant à un stade essentiel des circuits de base en logique séquentielle. Je vous rappelle que la logique séquentielle, contrairement à la logique combinatoire, fait appel à une variable temporelle qui permet de synchroniser l'action de différents circuits.

La synchronisation des différents composants d'un micro-ordinateur est basée sur un circuit électronique appelé "HORLOGE". L'horloge (ou CLOCK en anglais) a un coeur composé d'un cristal de quartz qui bat à une vitesse de l'ordre du mégahertz. Je vous rappelle que le Hertz (Hz) est une unité de mesure de fréquence correspondant à un battement par seconde. Une horloge de 1 MHz (mégahertz) battra donc un million de fois par seconde soit un battement tout les millionièmes de seconde. Chaque battement correspond alternativement à un état actif (CK = 1) ou un état passif (CK = 0) pour l'horloge et pour tous les circuits assujettis à cette horloge.

Les premiers systèmes conçus sur le principe de la logique séquentielle que nous allons voir dans ce cours seront tous de la famille des BASCULES. Comme leur nom l'indique avec précision, ces circuits laisseront passer les informations à un instant donné, lorsque l'horloge indiquera au système que l'instant fatidique du basculement de l'état général du circuit est arrivé.

Pour assurer une transition agréable entre les circuits combinatoires et les circuits séquentiels nous pencherons durant quelques lignes notre étude en direction d'un circuit bien particulier qu'est le SET-RESET (S-R). Le circuit est dit bistable et est l'un des seul organes de mémorisation où l'horloge n'ait pas de rôle à jouer. Il est constitué de deux circuits NANDs (N1 et N2) rebouclés sur eux-même: en clair, la sortie est dédoublée en direction de la sortie du circuit d'une part, et d'autre part en direction des circuits NANDs. Mais trève de discours, le schéma parle de lui-même:

SET-RESET



r̄ et s̄ sont les entrées actives au niveau 0. Pour mémoire, je redirais qu'un fil est noté X lorsqu'il est actif à l'état 1, en revanche il est noté x̄ lorsqu'il est actif à l'état 0. Vous noterez que les deux sorties sont complémentaires car elles ne peuvent pas prendre une valeur identique au même moment.

L'ASSEMBLEUR PRATIQUE

Langage machine... sur ZX81

Processeur, où en étions-nous de notre travail sur le registre double HL ?
Pour seule réponse vous n'entendez comme moi qu'une rumeur qui va s'amplifiant, une protestation synthétique à l'accent de crécelle qui, péniblement, phonème par phonème, éructe: "Quoi HL ? Toujours HL ! Et les autres, combien de cycles à attendre ? Registres abandonnés, avez-vous donc une âme..."
Qu'on se rassure dans les cellules, chacun aura sa part ! La preuve A: ce registre dont nous n'avons encore jamais parlé, le plus important peut-être, l'un des rares d'ailleurs à porter un nom comme vous et moi: ACCUMULATEUR. En un mot celui dont la protestation est la plus légitime face à sa fierté bafouée à longueur de ligne, à longueur d'Hebdo, celui-là dis-je, sera aussitôt et avec mes plus limandes excuses, mis à contribution. Heureugistre !
Je vous le dis à vous, en aparté: il m'était difficile de faire autrement. Vous vous rendrez vite compte qu'on ne peut se passer bien longtemps de l'accumulateur. Il est omniprésent dans les programmes d'assemblage pour la bonne raison qu'il s'agit d'un registre... particulier. Il s'intercale entre l'ALU et les autres registres, toute opération doit obtenir son consentement et son bon accueil. Sa devise: "Vous voulez l'ALU ? Passez donc par l'Accu !". Ainsi l'Accu s'impose-t-il à nous tant et si bien que nous ne pensons plus qu'à ça...
Dans notre exemple, c'est lui qui va se charger de lire la valeur du premier octet affiché à l'écran dont l'adresse est, je vous le rappelle, contenue dans (rapprochez-vous) HL. "Comment ?" me demande déjà et sans ménagement le moins brillant de mes lecteurs. Mais par une instruction d'assemblage appropriée, pardi ! Vous auriez pu lui souffler, évidemment...
Dictée: "Il va se charger..." -LD A-"de lire HL..." -LD A.HL-"dont la valeur n'est pas à prendre en considération pour elle-même..." -je t'aime moi non plus-"mais comme pointeur..." -LD A.(HL)-"du premier octet visible à l'écran". Combien de fautes? Aucune, car ce sont des mots et une syntaxe que vous commencez à apprivoiser. LD A.(HL): ranger dans A la valeur de l'octet pointé par HL. Comme récré imaginative je vous laisse un instant inventer les instructions les plus fantaisistes à partir de LOAD, de lettres, parenthèses et ".".
Dring ! Dring ! Finie la pause. Rentrez dans vos salles de classe de...
Diable, quel désordre dans ma mémoire ! Vite ma prothèse électronique, que je me branche ! J'y suis: vous êtes confortablement installé chez vous et cherchez moyen d'ajouter 128 au code stocké dans l'accumulateur afin d'obtenir le nouveau code du caractère en vidéo inverse. Comment ? Avec ce merveilleux Z80 ma réponse sera toujours la même: par une instruction d'assemblage appropriée, en l'occurence ADD A.128 (ou bien en hexadécimal sur ZX AS: ADD A.$ 80). Notez que le résultat de cette addition est automatiquement rangé dans l'accumulateur en remplacement de son ancienne valeur. Une fois ce travail accompli il nous faudra remettre ce nouveau code de CHR $ à la place de l'ancien dans le fichier d'affichage. Comment ? Par etc... etc...: LD (HL).A. Notre adresse de Fichier d'Affichage étant toujours contenue dans la paire HL, il nous suffira de ranger la valeur de A à l'emplacement mémoire pointé par HL. Abracadabra ! Le tour est joué.
Oui, mais précisément pour un tour seulement. Car il va nous falloir répéter l'opération sur les 793 octets du fichier d'affichage. En basic, qu'utiliseriez-vous? Une boucle FOR NEXT, bien sûr. En assembleur sur Z80 idem mais avec une instruction spécifique, très importante, que nous allons maintenant détailler. Son nom de baptême: DJNZ figure sur le registre B de la paroisse Zilog. En effet cette instruction est indissociable de ce registre sur lequel elle travaille. Entendez-moi bien: qui dit boucle dit compteur; avec DJZN c'est B qui remplira cet office. Amen. En vérité je vous le dis, à chaque passage par l'instruction DJZN le registre B sera décrémenté (-1), sa valeur sera testée et le programme ne poursuivra sa route séquentiellement que si la valeur de B est égale à zéro, autrement il effectuera un branchement automatique (semblable à un GOTO basic) en amont ou en aval de cette instruction, de -126 à + 129 octets au choix. Cette instruction extraordinairement riche n'occupe que deux misérables octets en mémoire: le premier pour l'instruction elle-même, le second pour préciser la valeur du saut à réaliser. (En assembleur pas de GOTO, on parle de sauts -JUMP-et dans ce cas de saut relatif puisque l'adresse de branchement est calculée à partir de la position de l'instruction DJZN en mémoire). Remarquez au passage que le mode d'adressage est immédiat puisque la valeur du saut figure explicitement dans l'instruction elle-même.
Avant d'examiner comment mettre en oeuvre cette instruction dans votre programme, je vous rappelle que celui-ci serait incomplet sans le système de repérage des prudes octets de fin de ligne (code 118) auxquels il ne faut surtout pas toucher. Achevons donc l'inventaire des instructions nécessaires en abordant "CP", abréviation non ésotérique de "compare". Cette instruction intéressante permet d'effectuer une comparaison entre le contenu de l'accumulateur (toujours lui !) et une opérande qui peut être au choix: une valeur, un registre, un octet pointé par HL etc. Pour identifier les "118" il nous suffira donc d'écrire CP 118. Si la comparaison s'avère positive, un "drapeau" (bit d'un registre spécialisé appelé indicateur) sera hissé bien haut afin que notre programme puisse en tenir compte. Il le fera grâce à une instruction de saut conditionnel JRNZ.x. Si vous voulez des mots pour le dire: saut relatif à l'adresse x si et seulement si la comparaison s'avère négative (s'il ne s'agit pas du caractère 118).
Rassurez-vous, tout cela va devenir limpide avec le listing complet et documenté de notre programme d'inversion vidéo. Vous y retrouverez toutes les instructions détaillées au fil de ces pages avec en prime l'art de les combiner: la programmation.
NB: Attention, les L1 et L2 sont des bornes destinées aux branchements consécutifs à des sauts conditionnels. Elles font partie de l'univers impitoyable des JR. Pour ces "labels", j'ai repris ici les symboles utilisés par l'assembleur ZX AS où il est nécessaire de les faire précéder du symbole ":".
Il est très important que vous parveniez à une parfaite compréhension de ce petit programme. Si humble soit-il, il vous fait rentrer de plain-pied dans l'assembleur du Z80. Le reste ne sera plus alors que variations à l'infini, plus complexes et pourtant plus harmonieuses encore...

```
LISTING ZX81

-REM * programme d'inversion vidéo LM
-REM * Initialisations

     LD HL.(16396)   localisation du FA
     LD B.24         initialisation du compteur de lignes

-REM * lecture case par case du FA

:L2  INC HL          on passe à l'octet suivant
     LD A.(HL)       sa valeur est rangée dans l'accumulateur
     CP 118          on traque les "intouchables"
     JR NZ.L1        s'il ne s'agit pas d'un intouchable on saute
                     en L1
     DJ NZ.L2        sinon on retourne à la borne L2 après avoir
                     décompté une ligne sur le registre B
     RET             retour au basic (lorsque toutes les lignes
                     auront été parcourues, B=0)

-REM * calcul du nouveau code

:L1  ADD A.$80       ajouter le facteur hexadécimal 80 pour établir
                     le nouveau code
     LD (HL).A       remettre en place dans le FA le caractère en
                     vidéo inverse
     JR L2           retour à la borne L2 (saut non conditionnel)
```

Formation à l'assembleur

COURS D'ASSEMBLEUR

Depuis plusieurs semaines, vous profitez d'un cours d'assembleur décomposé en deux parties essentielles (l'une théorique, l'autre pratique).

Cette semaine le Z80 du ZX 81 en prend plein la mémoire. La semaine prochaine ce sera le tour du 6502 de l'ORIC.

Jusqu'à présent, les cours pratiques suivants ont été publiés:

N° 55 → ZX 81
N° 56 → ZX 81
N° 57 → ORIC 1, ATMOS
N° 58 → APPLE
N° 59 → TO7, TO7 70
N° 60 → COMMODORE 64
N° 61 → ZX 81
N° 62 → ORIC 1, ATMOS
N° 63 → APPLE
N° 64 → TO7, TO7 70
N° 65 → COMMODORE 64
N° 66 → ZX 81
N° 67 → ORIC 1, ATMOS
N° 68 → APPLE
N° 69 → TO7, TO7 70
N° 70 → COMMODORE 64
N° 71 → ZX 81
N° 72 → ORIC 1, ATMOS
N° 73 → APPLE
N° 74 → TO7, TO7 70
N° 75 → COMMODORE 64

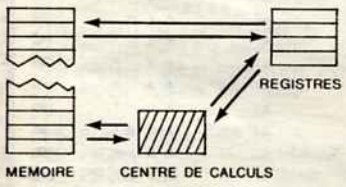
Cette fois-ci, nous atteignons un niveau praticable par n'importe quel type de machine: en effet, la série d'actions à accomplir est séquentielle. A aucun moment on ne demande au calculateur de pratiquer deux opérations simultanées. Nous pouvons considérer d'ailleurs que nous venons d'écrire un programme parfaitement exécutable par une machine.

QU'EST-CE QU'UN ORDINATEUR ?

L'ordinateur, tel que nous le pratiquons à notre niveau (micro-informatique personnelle ou professionnelle) n'est en fait qu'une machine de traitement séquentiel à programmes enregistrés. Les capacités d'un ordinateur se jugent par rapport au nombre de bits qu'il peut traiter d'un bloc. Nous avons déjà vu pourquoi les ordinateurs utilisaient une base de huit bits. Ainsi, dans l'escalade à la puissance de calcul *est-on* passé des processeurs à huit, puis seize, trente deux et même soixante quatre bits. Cette dimension que nous donnons en bits s'appelle aussi la "taille d'un mot machine". Nous pouvons imaginer un ordinateur sous la forme d'un centre de traitement des calculs en communication avec deux autres zones, non moins essentielles, la mémoire et les registres.

Nous pourrions représenter, d'une manière plus que simpliste, un ordinateur sous la forme suivante:

REPRESENTATION SCHEMATIQUE D'UN ORDINATEUR



Vous voyez l'ensemble des interconnexions entre les trois éléments vitaux de l'ordinateur. Mais, pour compliquer encore la tâche du concepteur d'ordinateur, il faut tenir compte des facteurs temps dans la communication entre ces différentes zones. Nous n'allons pas nous abrutir avec des chiffres, mais simplement considérer des qualificatifs.

Avant cela, précisons que les temps de communication que nous allons regarder concernent la circulation des informations entre le bloc de calcul (microprocesseur) d'une part et tous les organes de mémorisation d'autre part.

Registre (électronique): extrêmement rapide
Mémoire interne (non électronique): très rapide
Disque: lent
Bande magnétique: très lent
Terminal: extrêmement lent

Nous ne faisons pas ici un compte-rendu exhaustif de ce qui peut être raccordé à un ordinateur, mais je tiens à fixer des points de repère dans l'échelle des vitesses de travail des différents éléments d'un ordinateur.

Nous allons maintenant nous attaquer au chapitre le plus important de ce cours, qui concerne l'organisation matérielle et logicielle des différents éléments que nous avons pu étudier jusqu'à présent.

Nous pourrions imaginer de placer ces différents éléments dans une boîte, de remuer et de brancher pour voir si nous avons construit ainsi un ordinateur, mais comme nous l'avons vu précédemment, la logique de la conception informatique conduit à agir d'une manière plus rationnelle.

Nous savons jusqu'à présent fabriquer un registre et d'autres circuits de mémorisation. Reste à savoir comment gérer ces espaces de mémoire. Nous allons, pour résoudre ce problème, faire appel à nos souvenirs du début de ce cours.

Une **case mémoire** désigne soit le contenant soit le contenu, ce qui maintient un certain flou sur ce qu'est cette case mémoire. De par notre esprit cartésien, nous nous devons de supprimer cette ambigüité. La **case mémoire** est une boîte, d'où nous pouvons déduire subtilement que c'est le contenant. Chacune des cases mémoire d'un ordinateur possède un numéro différent de toutes les autres cases mémoire de ce même ordinateur. Ce numéro se nomme **adresse** de la case mémoire.

Maintenant que nous avons une idée précise de ce qu'est la boîte, ouvrons-la pour baptiser ce que nous y trouverons. A l'intérieur de chaque case mémoire, nous trouvons une série de chiffres binaires que nous appellerons dorénavant **information**. Le nombre de ces chiffres est fonction de la capacité de base de l'ordinateur. Ainsi lorsque l'on parle d'un **seize bits**, il s'agit d'un ordinateur dont chaque case mémoire contient une information sous la forme de groupes de seize chiffres binaires ou seize bits. Compte tenu des micros sur lesquels nous travaillons, nous considérerons (par un abus réel, mais aussi par commodité) qu'une case mémoire contient un octet, soit huit bits.

Reprenons le cours général de notre pensée. Nous avons actuellement une pile de boîtes, numérotées. Nous voulons, bien entendu, utiliser une machine de capacités raisonnables. Nous aurons donc 64 Kilo-octets de mémoire à gérer, ce qui signifie que nous allons avoir 65.536 adresses différentes parmi lesquelles nous devrons aller du premier coup et sans erreur extraire l'information que nous désirons. Le codage d'adresse est donc un exercice de style particulièrement intéressant, dans la mesure où nous devons pouvoir accéder à n'importe quelle case dans un minimum de temps et de bits (si possible) pour reconnaître cette case.

La première idée consisterait à numéroter bêtement chaque case. Nous aurions alors à mobiliser 16 bits (2 î 16 = 65536 = 64 Ko) et à parcourir toutes les boîtes jusqu'à trouver celle qui nous intéresse (cette solution demande une mobilisation temporaire du micro-processeur beaucoup trop longue). Une autre idée consisterait à dire que l'on peut effectuer un premier tri avant de s'attaquer à la recherche de la case elle-même. Le fait de décomposer la recherche ne nous permettra pas de "gagner des bits," mais elle raccourcit le temps de recherche de manière considérable. L'idée de différentes **pages** vient de là. Nous allons voir comment on aboutit à ce résultat.

Prenons une mémoire de 1 Ko. Nous avons 1024 adresses différentes à gérer, nous aurons donc besoin de 10 bits (2 î 10 = 1024) pour les représenter toutes. Les adresses iront donc de 00.0000.0000 jusqu'à 11.1111.1111. Imaginons que nous voulons mettre plusieurs mémoires de 1 Ko côte-à-côte, par exemple quatre. Pour trouver la bonne case mémoire, nous allons chercher dans laquelle des quatre mémoires elle se situe (donc nous devons savoir compter de 0 à 3) puis chercher la case dans la mémoire de 1 Ko, ce que nous savons faire avec dix bits. Pour compter de 0 à 3, il nous suffit de nous servir de deux bits. Faisons les comptes : deux bits pour reconnaître la bonne mémoire (ou la bonne page) et dix bits pour retrouver la case mémoire dans la mémoire.

ADRESSAGE SUR DOUZE BITS AVEC CODAGE DES PAGES SUR DEUX BITS

Revenons à notre micro hypothétique de 64 Ko. Nous savons coder une adresse sur dix bits, pour la reconnaître parmi 1024 de ses consoeurs. Il va donc nous falloir coder soixante quatre pages différentes. Pour avoir 64 codes différents nous avons besoin de six bits (2 î 6 = 64). Nous allons donc obtenir un codage sur seize bits (tout comme si nous adressions directement chaque case) dont six seront consacrés à la reconnaissance de la page.

ADRESSAGE SUR SEIZE BITS AVEC CODAGE DES PAGES SUR SIX BITS

L'ASSEMBLEUR PRATIQUE

Langage machine... sur ZX 81

A vos cahiers s'il vous plait ! Langage machine sur ZX81. Hebdogiciel n° 71. Vous avez tous devant vous le petit listing de votre premier programme d'assemblage. Nous allons, à partir de cette trame, introduire quelques variantes : transformer notre inversion vidéo totale en inversion partielle, utiliser notre savoir pour créer une fonction CLS ultra-rapide etc... Mais que pour l'écriture de ces modifications puisse s'imposer à votre esprit comme une évidence, parachevons votre intronisation en examinant ensemble le programme déjà réalisé, d'abord sous l'angle de sa structure, ensuite sous celui de son fonctionnement dynamique. Ces deux aspects bien assimilés il ne me restera plus qu'à laisser un blanc sur ces colonnes où vous pourrez écrire vous-même, d'un premier jet et sans rature, les variantes envisagées et, pourquoi pas, celles que vous imaginerez.

Principe général qui s'applique à tout langage et préside à toute oeuvre de programmation : la structuration. Le programme s'applique à isoler des **modules de traitement** dont le fonctionnement se veut autonome et le champ d'action délibérément limité. Dans nos listings chaque module est annoncé par son titre figurant (ZX AS oblige) après une instruction REM suivie d'une astérisque. Le module d'initialisation commence par remettre les pendules à l'heure, c'est-à-dire positionner les pointeurs (sur le fichier d'affichage) et initialiser les compteurs (le compteur de lignes).
Vient ensuite le moteur du système qui va permettre la lecture case par case, le **balayage** du F.A. Sa soupape : un CP 118 qui évitera les fins de ligne. Et pour finir le module d'inversion vidéo qu'on utilise chaque fois qu'il est nécessaire. Eh oui ! Plus c'est *carré* mieux c'est. Et comme dit le poète : "vous qui rentrez ici perdez toute ambigüité". Extirpez de votre pensée tout flou artistique, toute confusion poétique. Vous n'en rêverez que mieux une fois l'ordinateur éteint...

Si le programme est clairement structuré il est alors aisé d'en comprendre le fonctionnement dynamique. Contact ! *Initialisations* et le moteur se met en route. Grâce à la première incrémentation il saute d'emblée la case 118 qui ouvre le F.A. S'emparant du premier caractère affiché dont la comparaison avec 118 s'avère négative, il passe au module d'inversion vidéo. Celui-ci calcule le nouveau code et se charge de l'afficher à l'écran à la place de l'ancien code puis *redonne la main* au moteur qui fait un tour de roue supplémentaire (INC HL) et ainsi de suite... jusqu'à la fin de ligne où la comparaison avec 118 s'avère positive cette fois. Il ne se passe alors rien d'autre que la décrémentation du registre B qui accusera encore 23 lignes à inverser, et un nouveau tour de roue (INC HL) qui permettra d'atteindre la première case de la deuxième ligne. Et ainsi de suite jusqu'à l'ultime décrémentation de B par quoi s'achèvera l'inversion vidéo de la vingt quatrième ligne. Le programme poursuivant alors séquentiellement se jette tout droit sur l'instruction RET. Finis le trip machine, c'est la descente au basic qui commence... Pour vous aussi je l'espère les choses se sont inversées : c'était obscur, c'est devenu plus clair.

Nous n'en resterons pas là ! Vos brillantes facultés de compréhension n'en finissent pas de m'étonner, je soumets donc à votre perspicacité programmatique le problème suivant qui ne cessa de hanter mes nuits jusque vers ma onzième année : comment transformer ce système d'inversion vidéo globale en un traitement plus sélectif de la surface de l'écran, à savoir n'inverser la vidéo qu'à l'endroit précis où un message s'affiche et cette fois ne plus toucher aux *blancs* . Hugh !.. J'ai dit. Mais c'est vous qui allez fumer.

Au quatrième top deux laborieuses heures se seront précisément écoulées justifiant à mes yeux que je vous livre sans pudeur l'une des plus curieuses observations que j'ai pu faire sur la personne du rédacteur en chef de l'HHHebdo : il ne supporte pas le blanc dans ses colonnes... Renseignements pris auprès de personnalités compétentes il s'agirait là d'une phobie qui aurait jusqu'à présent échappé à la nosographie psychiatrique post-freudienne. C'est donc bien un scoop !

Vous me pardonnerez cher lecteur de continuer à monopoliser cette page que je vous aurais volontiers cédé. J'en profiterais pour vous donner la solution de notre petit problème et ferais appel à votre honnêteté intellectuelle en vous demandant de bien vouloir retourner votre journal de façon à ce que le corrigé se trouve à l'envers de l'énoncé du problème comme c'est le cas dans tout manuel édifiant et qui se respecte. Merci pour l'imprimeur à qui ça facilite bien la tâche.

Il faut préalablement savoir une chose c'est que le blanc sur l'écran est en fait un caractère **espace** qui, comme tout caractère, possède son numéro de code. On ne pouvait lui en attribuer de plus représentatif que le "0", on l'a donc fait. Notre problème se résume à identifier les caractères de code 0 et à partir de là à court-circuiter le module d'inversion vidéo, autrement dit à le sauter comme on le fait déjà pour les caractères de code 118 (fin de ligne). Il suffit donc d'intercaler entre l'instruction de chargement de l'accumulateur LD A.(HL) et avant l'identification des 118 (CP 118) une instruction CP 0 suivie d'un saut conditionnel en L2 (JR Z.L2), saut réalisé si et seulement si la comparaison du contenu de l'accumulateur avec 0 est positive. Vous constatez qu'avec seulement deux instructions supplémentaires nous avons doté notre programme d'une nouvelle et importante fonction.

Avec les quelques instructions que nous maîtrisons déjà et en modifiant légèrement la structure de notre programme il nous est possible de réaliser une routine d'effacement d'écran beaucoup plus efficace que la commande CLS du basic Sinclair. Mais d'abord en quoi consiste-t-elle ? Il s'agit tout simplement de remplacer tous les caractères du F.A. par le caractère de code 0 (sauf évidemment les 118 de fin de ligne). Le travail est donc sensiblement le même que celui exigé pour notre inversion vidéo globale. Il nous faudra balayer le F.A., éviter les 118 et placer notre valeur 0 dans tous les octets du F.A. Ce qui peut se traduire par le programme suivant :

LISTING ASSEMBLEUR n° 1

```
REM * CLS ULTRA-RAPIDE
REM * INITIALISATIONS
LD HL,(16396)          Localisation du F.A
LD B,24                Initialisation du
                       compteur de lignes
LD C,0                 Utilisation du registre
                       C pour stocker la
                       valeur d'échange
REM * BALAYAGE DU F.A
:L1 INC HL             On passe à l'octet
                       suivant
LD A,(HL)              Range la valeur dans
                       l'accumulateur
CP 118                 Traque les intouchables
JR NZ,L2               Si différent de 118
                       alors aller en L2
DJ NZ,L1               Sinon recommencer en L1
                       après décompte d'une
                       ligne (B = B-1)
RET                    Retour au basic (B = 0)
REM * EFFACEMENT
:L2 LD (HL),C          Effacer l'octet pointé
                       par HL en y mettant la
                       valeur de C
JR L1                  Continuer pour l'octet
                       suivant
```

Vous devinez aisément qu'il suffit de modifier la valeur du registre C pour obtenir un écran à la "couleur" de son choix. Pour les effets amusants, essayez les caractères graphiques codés de 1 à 10.

Une dernière variante intéressante consisterait à modifier le fond habituellement blanc sur lequel s'affiche les messages, c'est-à-dire à l'inverse de notre vidéo inverse partielle, de ne travailler que sur les blancs de l'écran afin de leur substituer le caractère de notre choix. Pour une utilisation plus souple de ce programme la "couleur" du fond sera stockée par *pokage* à l'adresse 16507 (inutilisée) où elle sera lue par la routine en temps utile.

LISTING ASSEMBLEUR n° 2

```
REM * FOND AUTOMATIQUE
REM * INITIALISATIONS
LD A,(16507)           Lecture de la "couleur"
                       choisie (ne pas oublier
                       le POKE 16507,X avant de
                       lancer la routine)
LD C,A                 Mise à disposition dans C
LD B,24                Compteur de lignes
LD HL,(16396)          Localisation du F.A
REM * BALAYAGE
:L1 INC HL             même
LD A,(HL)              séquence
CP 118                 que
JR NZ,L2               précédemment
DJ NZ,L1               eh
RET                    oui !
REM * MODIFICATION SELECTIVE
:L2 CP 0               S'il ne s'agit pas d'un
                       blanc...
JR NZ,L1               Poursuivre le balayage...
LD A,C                 Sinon transférer la
                       couleur de C dans A...
LD (HL),A              Puis de A dans le F.A
JR L1                  avant de reprendre le
                       balayage
```

Il ne tient plus qu'à votre imagination de multiplier ces variantes. Et surtout : conservez vos listings car nous verrons bientôt comment rendre aisément et simultanément toutes nos routines à partir de n'importe quel programme basic.

Bernard GUYOT

Formation à l'assembleur

COURS D'ASSEMBLEUR

Déjà des mois que vous planchez, les uns grâce au cours pratique, les autres sur les connaissances plutôt théoriques (et réthoriques ?) de l'informatique en général et de la programmation en assembleur en particulier.

Pour cette semaine un grand mouvement de réflexion du côté du Z 80 du ZX 81. Mais les fanatiques du 6502 de l'Oric ne perdent rien pour attendre : la semaine prochaine sera pour eux !

Depuis le début nous avons eu les cours suivants :

No 55 ----> ZX 81
No 56 ----> ZX 81
No 57 ----> ORIC 1, ATMOS
No 58 ----> APPLE
No 59 ----> TO7, TO7 70
No 60 ----> COMMODORE 64
No 61 ----> ZX 81
No 62 ----> ORIC 1, ATMOS
No 63 ----> APPLE
No 64 ----> TO7, TO7 70
No 65 ----> COMMODORE 64
No 66 ----> ZX 81
No 67 ----> ORIC 1, ATMOS
No 68 ----> APPLE
No 69 ----> TO7, TO7 70
No 70 ----> COMMODORE 64
No 71 ----> ZX 81
No 72 ----> ORIC 1, ATMOS
No 73 ----> APPLE
No 74 ----> TO7, TO7 70
No 75 ----> COMMODORE 64
No 76 ----> ZX 81
No 77 ----> ORIC 1, ATMOS
No 78 ----> APPLE
No 79 ----> TO7, TO7 70
No 80 ----> COMMODORE 64

D'un point de vue théorique, vous aurez deux parties dans ce programme d'affichage. La première gèrera un affichage, quel qu'il soit, à l'écran (souvent il suffit de se servir du sous-programme qui existe déjà dans la ROM de votre micro). La deuxième ressemblera à un tableau à une dimension qui contiendra votre message, chaque lettre étant rangée dans une case du tableau. A partir de cette structure (encore elle, mais elle est partout en langage machine !) de base vous aurez à choisir entre une routine qui ne vous servira que pour cet affichage spécifique ou une routine qui pourra gérer n'importe quel message, dont celui qui vous préoccupe.
Dans le premier cas il vous suffit de compter les lettres du message et de réaliser une boucle faisant exactement appel au tableau autant de fois qu'il y a de lettres dans cet affichage.
Dans le second, il vous faudra raisonner d'un point de vue général avant de vous attaquer au cas particulier qui vous inquiète.

En fin de compte je ne vous ai pas appris comment réaliser ce petit programme, mais j'ai essayé de vous faire saisir à quel point un problème inexistant en Basic peut devenir contraignant en langage machine. C'est pour cela que je tiens à vous répéter ce conseil (mille fois entendu peut-être) que lorsque vous vous lancez dans la programmation en langage machine, vous devez disséquer votre problème sur le papier jusqu'à l'obtention d'un algorithme transférable directement dans le micro-ordinateur. Fermons ici cette gigantesque parenthèse et revenons à nos questions plus générales sur la programmation en langage machine.

QUELQUES CONSEILS POUR PROGRAMMER EN LANGAGE MACHINE

Dans l'exemple précédent j'ai tenté de vous faire saisir l'extraordinaire complexité qui préside à la conception d'un programme en langage machine. Nous allons essayer de rationnaliser un peu plus nos connaissances de manière à éviter de sombrer dans le gouffre du programme impossible à réaliser ou à terminer.

Ne vous effrayez pas devant les nombreuses étapes que je vais vous décrire, certaines ne nécessiteront pas plus de quelques secondes pour être franchies.

1. Débranchez votre micro, prenez un stylo et beaucoup de papier.
2. Ecrivez en bon français ce que vous désirez réaliser, ainsi que le but que vous cherchez à atteindre.
3. Tentez de séparer votre problème en un ensemble de petits problèmes, sachant que chacune des parties de cet ensemble devra répondre aux critères suivants :

a. poser *un seul* problème
b. être (relativement) indépendant des autres problèmes
c. ne pas cacher un autre problème !!!

4. Essayez d'établir un ordre chronologique entre les différentes parties que vous aurez dégagé à l'étape 3.

5. Etablissez un organigramme (ou ordinogramme) des différentes parties que vous avez créées, en respectant l'ordre chronologique que vous venez de vous fixer.

6. A partir de là, programmez **sur le papier** chacune des différentes parties de votre problème.

7. Rallumez votre micro et testez chacune des petites routines que vous venez d'écrire. Deux cas se présentent alors :

a. vos routines fonctionnent et vous pouvez continuer (étape 8)
b. au moins une routine ne marche pas ou ne fait pas ce que vous souhaitiez. Dès lors vous devez effectuer une vérification essentielle : avez-vous mal dégrossi le problème (et de ce fait vous vous retrouvez obligé de reprendre à l'étape 3) ? Si vous n'avez effectivement qu'une tâche à accomplir d'après votre papier, reprenez votre programmation sur le papier pour la corriger (et n'oubliez pas de comparer ce que vous avez à l'écran avec ce que vous avez écrit sur le papier).

8. En vous basant sur votre organigramme, construisez sur le papier le programme qui vous permettra d'appeler chacune des routines que vous venez d'écrire au moment voulu.

9. Programmez sur votre micro et testez l'ensemble. Ca ne marche pas ? Revenez à l'étape 8 et reprenez pas à pas votre programme (sur le papier toujours) pour essayer de découvrir l'erreur. Ca ne marche toujours pas ? Reprenez votre organigramme (étape 4 et 5) et contrôlez que vous ne vous êtes pas abusé en vous lançant ainsi sur ce schéma-là.

10. Ca marche ! N'oubliez pas de sauvegarder votre oeuvre sur un support magnétique avant d'éteindre votre micro et de profiter d'un repos bien mérité.

Dans la série des conseils que l'on a plaisir à donner, mais que l'on trouve inutile de suivre, en voici de nouveaux :

1. Gardez la place d'introduire des commentaires dans vos listings (si votre programme d'assemblage ne permet pas la création de lignes de REM) et commentez-le le jour même où vous le terminez.

2. Conservez au moment de la programmation la structure que vous vous êtes fixé sur le papier, sinon gare à la perte de temps lorsque vous chercherez le grain de sable qui bloque la machine.

3. N'hésitez pas à résoudre un problème de deux ou trois façons différentes, vous garderez la meilleure pour d'autres programmes.

4. Tentez de programmer des routines à application générale et non spécifique, vous ne serez pas obligé de la réécrire pour chaque cas particulier, vous pourrez la transférer d'un programme à l'autre.
Vous vous doutez que si je donne ce genre de conseils, ce n'est pas seulement parce que je m'inspire d'auteurs célèbres (Zaks ou Leventhal par exemple) mais aussi en me fiant à mon expérience personnelle de ce sport intellectuel qu'est la programmation en langage machine. En effet, rien ne vous oblige à appliquer les méthodes que je vous indique, mais sachez que nombre de programmeurs n'appliquent aucune structure à leur travail et se retrouvent rapidement obligés de suivre pas à pas ce qu'ils ont écrit pour en comprendre la signification.
Je me fais l'effet d'un vieux gâteux à insister de la sorte, mais il me parait vraiment nécessaire, indispensable même, que vous ne vous lanciez pas tête baissée dans la programmation sans avoir franchi une seule des phases préparatoires qui lui sont utiles.
Dorénavant vous allez pouvoir trouver quelques conseils de programmation sur des problèmes classiques posés par les micro-ordinateurs. Vous ne serez en aucun cas tenu de vous y fier. Ce que je tiens à réaliser, c'est vous faire toucher du doigt des règles qui se retrouvent cycliquement dans différents cas de figure. J'irai même plus loin : si un problème se présente de telle façon que vous le trouvez insoluble, envoyez-moi la description précise de celui-ci, je pourrai peut-être vous aider à trouver une solution.
Enfin un dernier point me semble important à souligner : lors de l'utilisation du langage machine sur un micro-processeur donné par plusieurs personnes, vous pourrez découvrir autant de méthodes de programmation que de programmeurs pour résoudre un même problème. Je pourrai même dire qu'un programmeur chevronné programme un micro-processeur de manière si personnelle que sa programmation est reconnaissable entre toutes. Nous pouvons considérer que son oeuvre est signée, s'il a réalisé un programme entièrement sur ses idées et à l'aide d'astuces de programmation qui lui sont propres.

L'ASSEMBLEUR PRATIQUE

Langage machine... SUR ZX 81

Gauche ! Droite ! Gauche ! Droite ! Incroyable : le principe de l'alternance démocratique est aussi celui de la bonne marche des armées. C'est à y perdre son libéralisme, pas vrai ?

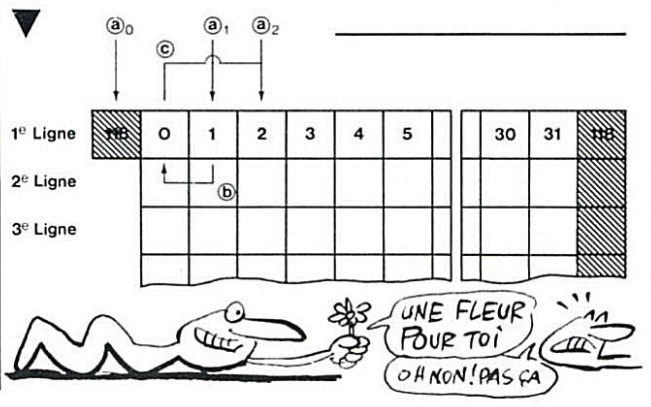
P'tits gars faut que j'vous dise la vérité; ouvrez grandes vos esgourdes, serrez les fesses, bombez-moi les écrans cathodiques et que j'voye pas un joystick sortir du rang ! J'aime l'ordre, l'ordre et tout ce qui s'y rapporte : ordonnance, ordonnateur et ordinateur. Compris ! Alors repos.

Quant à vous, vaste contingent des lecteurs de l'Hebdo, comme moi farouches partisans de l'ordre dans la programmation et... d'un joyeux désordre dyonisiaque partout ailleurs, je ne vous proposerai aujourd'hui que la mise au pas du SCROLL du bataillon d'octets dociles de votre fichier d'affichage. Saurez-vous, ex-réformés et futurs exemptés, vous faire obéir de cette armée masochiste qui ne connait comme mouvement d'ensemble que le CLS du Basic Sinclair qui l'anéantit ? Je suis là pour vous aider à gagner les seuls galons qui valent la peine d'en jouir...

Gauche ! Droite ! Définissons d'abord notre objectif : un SCROLLING c'est un mouvement d'ensemble de la page écran qui permet de décaler celle-ci dans une direction donnée. Vous connaissez l'instruction SCROLL du basic de notre ZX qui, en même temps qu'elle décale l'image d'une ligne vers le haut, positionne le pointeur d'écriture au début de la vingt deuxième ligne de l'écran. Nous laisserons de côté cette dernière fonction et nous nous consacrerons uniquement à trouver le moyen d'obtenir un décalage global de l'image d'une ou plusieurs colonnes vers la gauche ou vers la droite.

Gauche ! Droite ! Mais d'abord choisissons démocratiquement notre bord : pile à gauche, face à droite et la tranche à Le Pen. Allez ! Trois petits tours et... nous voilà dans le camp majoritaire. Reste à agir en bon politique : analyser le problème, définir une stratégie puis rédiger le texte qui sera ensuite soumis à l'approbation de la machine.

S'agissant d'un travail sur le fichier d'affichage (F.A.) il est nécessaire de se représenter clairement sa structure : 24 lignes de 32 caractères s'achevant toutes par un octet codé 118, et au début du fichier : un octet codé 118. La solution la plus simple pour obtenir notre effet de SCROLLING à gauche consistera à dupliquer à gauche chacune des cases du F.A. mis à part les invariables octets de fin de ligne (118). Nous utiliserons pour cela un pointeur à *trois* temps qui balayera le F.A. suivant le mouvement décrit sur notre schéma :



a. on pointe sur l'octet à dupliquer (à partir de a1)
b. on pointe sur la case où le dupliquer (simple décrémentation)

c. on pointe sur la case suivante (double incrémentation et redémarrage du cycle en a2)

Ce mouvement très simple permet *l'écrasement* des valeurs de la colonne 1 par les valeurs de la colonne 2, la substitution des valeurs de la colonne 2 par celles de la colonne 3 et ainsi de suite jusqu'à la colonne 31 qui, exception à la règle jusque-là suivie, ne voit pas sa valeur écrasée par un caractère de fin de ligne 118 mais doit purement et simplement disparaitre (caractère espace). On amorce de cette façon-là l'effacement de l'écran par un effet de sortie de l'image par la gauche et de dégagement simultané par la droite. Tous les 118 doivent, quant à eux, rester bien en place : pour obtenir un décalage global de l'image d'une case à gauche l'opération devra être répétée sur les 24 lignes du F.A. Si l'on veut une sortie complète de l'image, ce dernier mouvement devra être répété 32 fois. Rassurez-vous : avec l'assembleur tout cela va très vite.

C'est à ce stade de notre analyse, avant de rédiger le programme, qu'il convient d'examiner attentivement le mécanisme de notre scrolling afin d'en dégager toutes les structures répétitives, structures qui devront tendre vers un mouvement unique, vers une loi générale. On vise ici la simplification du programme, la concision de son écriture, le gain de temps dans son exécution. Dans notre exemple on peut retenir comme système dynamique principal le mode de balayage du *pointeur à trois temps*, c'est à dire que nous ne modifierons pas, tout au long de l'exécution du programme, les modalités de son avancée dans le F.A. : un pas à gauche, deux pas à droite !

Nous aboutissons donc au listing suivant :

Listing 1

```
REM * INITIALISATIONS
      LD B,24          compteur lignes
      LD HL,(16396)    adresse du F.A.(départ en
                       a0)
REM * POINTAGE SUR L'OCTET A DUPLIQUER
:L1   INC HL           étape c
      INC HL           idem
      LD A,(HL)        contenu mis à la
                       disposition du processeur
      CP 118           traque des intouchables
      JR NZ,L2         passer au module de
                       duplication
REM * EFFACEMENT DES OCTETS COLONNE 31
      DEC HL           pointe sur la colonne 31
      LD (HL),0        pour y mettre un espace
      INC HL           restaure l'état initial
                       du pointeur
      DJNZ,L1          après la 24ème ligne...
      RET              ...retour au basic
REM * MODULE DE DUPLICATION GAUCHE
:L2   DEC HL           temps b
      LD (HL),A        on duplique
      JR L1            retour au temps c
```

Vous remarquerez que la seule entorse à notre règle de déplacement à trois temps est due à l'écrasement forcé de la valeur de la colonne 31 par le code 0, manoeuvre inévitable puisqu'on ne peut y dupliquer le caractère 118.

Dans l'état actuel de notre programme, si nous souhaitons chasser totalement l'image de l'écran il suffira d'insérer notre routine à l'intérieur d'une boucle basic. On pourra aussi de cette façon régler à sa convenance le déplacement latéral de la page écran ce qui permet des effets amusants. Nous verrons, avec le scrolling à droite, comment introduire directement ce système de répétition dans notre routine.

Gauche ! Droite ! Le moment est bien choisi pour retourner sa veste, aussi basculerons-nous dans l'autre camp sans plus tarder. Scrolling à droite : il existe de nombreuses solutions permettant d'arriver au même résultat, cependant par souci de simplicité et de clarté il serait bien d'imaginer un programme exactement symétrique du précédent. Que les paresseux se retiennent malgré tout de contempler le même programme et ses effets réfléchis par un miroir judicieusement placé ! Car la solution est simple : elle consiste à prende le F.A. à l'envers, c'est à dire de commencer notre balayage à trois temps par l'avant-dernier des octets visibles à l'écran sur la 24ème ligne. Vous constaterez aisément qu'on pourra alors remplacer de façon parfaitement symétrique les INC par des DEC et inversement. Quant à la colonne à effacer il suffira de remplacer son ancien numéro 31 par celui de la première colonne de gauche n° 0. Il nous reste encore une modificaton : l'insertion d'un système de répétition automatique qui chassera complètement l'image de l'écran; il sera composé d'un classique compteur et d'une non moins classique décrémentation de celui-ci avec comparaison à chaque tour. Le programme se décompose donc comme suit :

Listing 2

```
REM * INITIALISATIONS
      LD C,32          compteur de tours (nombre
                       de colonnes)
:L3   LD B,24          compteur de lignes
      LD HL,(16396)    F.A.
      LD DE,792        nombre d'octets du F.A.
      ADD HL,DE        addition de HL avec DE,
                       résultat rangé dans HL,
                       on obtient l'adresse de
                       fin du F.A.
REM * POINTAGE SUR L'OCTET A DUPLIQUER
:L1   DEC HL           on
      DEC HL           recule
      LD A,(HL)        au
      CP118            lieu
      JR NZ,L2         d'avancer
REM * EFFACEMENT COLONNE 0
      LD A,0           l'accu prend le code
                       espace
      INC HL           pointe sur la colonne 0
      LD (HL),A        efface
      DEC HL           restaure le pointeur
      DJNZ,L1          compte les lignes en
                       décrémentant B
REM * COMPTEUR DE COLONNES
      DEC E
      CP C             C est comparé à A (A=0)
      JR NZ,L3         on recommence un tour...
      RET              ...sinon retour au basic
REM * MODULE DE DUPLICATION
:L2   INC HL           on avance
      LD (HL),A        au lieu de
      JR L1            reculer
```

Remarquez que la totalité des registres jusqu'alors évoqués (B, C, D, E, H, L) ont été utilisés. Nous atteignons déjà les limites de la capacité mémoire du Z80 ! Sommes-nous à la fin de nos aventures ? Non grâce à la... Pile je continue, face j'arrête... Ah ! Alors à la prochaine !

Bernard GUYOT

Formation à l'assembleur

COURS D'ASSEMBLEUR

Vous êtes, grâce à l'HHHHebdo, devenus des quasi-cracks de l'assembleur. Mais ne partez pas ! Les bonnes surprises et les bons plans vont se multiplier dans les prochains cours, vous transportant jusqu'au nirvana des programmeurs. Comme toujours le cours théorique branchera les fans du tordu alors que les bidouilleurs se brancheront directement sur leur cours rien qu'à eux !

Vous avez déjà eu droit aux discours suivants, dans votre page chérie par-dessus tout :

ZX 81 ----> 55 56 61 66 71 76 81
ORIC ----> 57 62 67 72 77 82
APPLE ----> 58 63 68 73 78 83
THOMSON ----> 59 64 69 74 79 84
COMMODORE ----> 60 65 70 75 80 85

Récemment, nous avons brièvement survolé l'exemple d'une structure d'édition de messages. Nous allons maintenant tenter de formuler (d'un point de vue tout à fait général) la démarche à suivre pour aboutir à une routine (portable sur n'importe quel matériel) d'édition de messages ou de caractères. Pour cela vous devez impérativement disposer d'une adresse de la ROM de votre ordinateur : celle du sous-programme qui gère l'édition d'un caractère sur un périphérique. Nous aurons à nous en servir car nos connaissances et notre maîtrise du langage machine ne nous permet pas de concevoir mieux que cette procédure installée par les concepteurs du micro.

Comme je suis partisan de la méthode du clou enfoncé, nous allons suivre scrupuleusement les différentes étapes que je vous avais fortement conseillé lors d'une précédente causerie.

1. Nous débranchons en rythme notre micro-ordinateur, nous nous saisissons d'une pile de papier et de différents stylos (plusieurs couleurs peuvent aider à la compréhension de schémas).

2. Nous devons maintenant définir clairement le problème que nous nous posons, et ce en français et non pas en instructions diverses et incompréhensibles.
Nous voulons créer une routine en langage machine qui puisse gérer et éditer des messages, quel que soit le contenu et la longueur de ces messages. Cette routine devra, d'autre part, pouvoir être transférée d'un programme à l'autre sans modification. Enfin, elle devra contenir le minimum d'impératifs du type : zone mémoire à réserver pour le stockage des messages, encombrement mémoire minimum, rapidité d'exécution pas trop ringarde...

3. Pour résoudre sans anarchie le problème que nous venons de nous poser, nous devons impérativement disséquer les différentes phases de ce travail pour arriver à une structure cohérente. Chacune des parties que nous allons dégager dans le traitement d'un message à éditer devra répondre aux critères suivants : poser un problème à la fois et un seul, être relativement indépendant des autres problèmes et enfin ne pas cacher d'autre problème derrière une banalité de mauvais aloi.

L'édition d'un message est d'une difficulté relativement peu élevée. Malgré tout, nous aurons à différencier deux parties principales dans ce travail :

a. la recherche du prochain caractère à éditer.
b. l'édition du caractère proprement dite, à l'aide de la routine interne de votre micro.

Si la seconde partie ne présente aucune difficulté, la première par contre mérite d'être approfondie. En effet, il est hors de question d'admettre comme immédiat la recherche du prochain caractère à éditer. Le simple fait de parler de *recherche* sous-entend que nous allons avoir d'une part une zone de stockage des caractères et d'autre part un caractère spécial qui nous permettra de juger que nous sommes arrivés en fin de message. Notre premier problème de la recherche du prochain caractère se décompose donc en deux nouvelles parties : saisie du caractère et comparaison avec le caractère de fin de message. Ce que nous pouvons résumer de la sorte :

a. recherche du prochain caractère à éditer :

a1. saisie du caractère
a2. test sur le caractère : fin de message ou non.

Nous nous retrouvons donc à la tête d'une structure (j'insiste sur ce mot, vous l'auriez peut-être déjà oublié) parfaitement opérationnelle, à un détail majeur près ! Nous n'avons pas encore résolu la méthode que nous utiliserons pour détecter la zone de début de message.

Pour réaliser ce repérage, nous avons deux solutions au choix, l'une meilleure que l'autre bien entendu :

Solution 1 : nous pouvons imposer une zone de la mémoire comme étant le stock de messages à éditer. Nous aurons alors besoin d'un pointeur qui nous indiquera l'endroit où nous en sommes dans cette zone. Deux reproches viennent rapidement à l'esprit : la zone mémoire est obligatoirement bloquée dans sa totalité, monopolisant du même coup un espace mémoire dont nous pourrions avoir besoin par ailleurs et le pointeur devra être sur deux octets si nous pensons éditer des messages de plus de 256 caractères, et comme nous voulons une routine universelle pour notre bécane, il nous faudra donc un pointeur seize bits.

Solution 2 : elle consiste à placer les messages complètement n'importe où dans la mémoire. Cette fois nous nous servirons

obligatoirement d'un pointeur de deux octets (il permet de pointer une zone dans les 64 Ko courants dans les micros familiaux). Nous n'aurons donc que le problème de la gestion du pointeur à régler. Cette solution, si elle n'est pas gourmande en place mémoire (les zones réservées contenant juste les messages et rien d'autre), implique un nouvel impératif : avant d'appeler la routine d'édition de caractères, nous devrons initialiser le pointeur pour lui donner l'adresse de début du message à éditer.

Pour notre routine nous choisirons bien sûr la seconde solution, pour le repérage de la zone du message. De ce fait, nous nous retrouvons dans l'obligation de résoudre un nouveau problème : la gestion d'un pointeur de deux octets. En effet, la quasi-totalité des micros familiaux n'offre pas la possibilité de pratiquer des opérations sur seize bits. Nous avons donc découvert un nouveau problème, que nous nous devons de résoudre avant de considérer la partie de recherche algorithmique comme terminée.

Je ne vous bombarderai pas le cerveau d'informations inutiles ou superfétatoires avec la recherche de la méthode optimale pour les opérations sur seize bits. Je vous donnerai une possibilité de résoudre ce problème qui n'est ni la seule, ni forcément la meilleure, mais elle offre l'incomparable avantage de fonctionner dans tous les cas de figure. Je vous confierai l'algorithme de cette méthode dans la suite de ma démonstration.

4. Nous arrivons ici à la phase *chronologique* de nos recherches. Nous nous devons d'établir un schéma général de la séquence de déroulement des différents points que nous avons dégagés précédemment. Nous allons donc, ni plus ni moins, classifier les différentes étapes que le micro-processeur aura à suivre pour aboutir à l'édition d'un message à l'écran.

Nous allons donc aboutir à un schéma proche de celui que je vous livre :

a. initialisation du pointeur : mettre dans ce vecteur l'adresse de début de message.

b. appel de la routine : sans commentaire.

c. recherche du caractère à éditer : notre initialisation sert pour le premier caractère.

d. test sur le caractère : si le caractère est celui correspondant à la fin du message alors fin de la routine.

e. édition du caractère : appel à la routine spécifique de votre micro-ordinateur.

f. incrémentation du pointeur : l'opération se passe sur deux octets.

g. aller en c : continuer l'édition des caractères jusqu'à la rencontre du caractère de fin de message.

L'ASSEMBLEUR PRATIQUE

Langage machine... sur ZX 81

Vous ne pouvez pas l'encadrer et cette situation vous est insupportable ! Quoi de plus triste en effet que de ne pouvoir souligner d'une bordure aux motifs variés et gracieux les contours de votre écran d'affichage, que ce soit pour la présentation BCBG du titre en gros caractères de votre petit dernier programme ou l'affichage plus intimidant de vos meilleurs scores au Big Invaders. Découragé par la lenteur basique de l'opération vous avez trop souvent négligé ce bel et noble ornement : le cadre. Remarquez que son esthétique sobre et rigoureuse a triomphé de toutes les modes, accompagnant à travers les âges l'art pictural, l'art photographique et maintenant... l'informatique. Indispensable hier, il ne l'est pas moins aujourd'hui et vous ne sauriez le nier plus longtemps sans risquer gros de m'échauffer la bile ! J'affirme donc que ce ne sera point sortir du cadre de cet exposé éminemment contemporain que d'en esquisser ici l'analyse formelle et même -allez, je vous fais un cadeau ! - structurale.
Si nous ramenons le cadre aux éléments essentiels qui le compose que voyons-nous ? Respectivement, de haut en bas et de gauche à droite :
1 - Sa limite supérieure.
2 - Son bord gauche et son bord droit.
3 - Sa limite inférieure.
Rassurez-vous cette complexité n'est qu'apparente et si je n'hésite pas à tenter de vous la rendre intelligible c'est pour mieux servir notre objectif ultime : le tracé d'un cadre en langage machine. Vaste entreprise qui, héritière d'une tradition séculaire puissance 10, porte au panthéon de la modernité cette forme inaltérable, inépuisable qu'est le cadre. Le cadre, la plus belle invention de l'homme après ce qu'il met à l'intérieur.
Pour tracer un cadre en langage machine il vous faut :
-Un abonnement EDF (très cher)
-Un ZX81 sans faux contacts (très rare)
-Un écran cathodique (très laid)
-Un algorithme approprié (très juste !)
Si nous supposons les trois premières conditions remplies, c'est à dire que vous êtes fauché mais veinard et dans un environnement particulièrement moche, il ne vous reste plus qu'à satisfaire à la dernière condition. Pour cela je me permettrai d'insister sur un point très important : n'arrêtez pas ici votre lecture; vous êtes excédé, soit, mais courage ! Les trois quarts du travail sont d'ores et déjà accomplis et ce serait trop dommage de renoncer maintenant. Continuez ! Continuez jusqu'à l'algorithme suivant :
si nous voulons commodément choisir la "couleur" de notre bordure, il faudra préalablement la stocker dans une case mémoire où le programme viendra la lire. Pour cela utilisons l'adresse libre 16381 pourtant en-deça de la mémoire utilisateur, mais qu'importe ! Il suffira d'y poker le code correspondant au caractère choisi comme motif de notre cadre. Les motifs baroques sont codés de 1 à 10, les classiques de 11 à 20, les modernes de 28 à 37, etc... On trouve même les Lettristes codés de 38 à 63 ce qui, chers amateurs d'art contemporain, représente un véritable tour de force sur une machine aussi commune.
Ceci fait, il nous suffira de tracer successivement les différents éléments du cadre tels que je les ai déjà décrits. D'abord le tracé de la limite supérieure, ce que nous savons faire sans difficulté; puis le tracé des côtés qui pourrait s'obtenir facilement par l'astuce suivante : après avoir pointé sur le premier octet de la deuxième ligne du fichier d'affichage, je saute 31 colonnes afin de pointer directement sur le dernier octet de cette même ligne et je recommence l'opération jusqu'à l'avant-dernière ligne comprise. Ce saut pourra être facilement réalisé à partir de l'addition (instruction ADD) d'une valeur constante représentant l'écart (31).
Puis nous conclurons en traçant la limite inférieure sur la vingt-quatrième ligne de l'écran, avant de "rendre la main au Basic".
Il ne s'agira là que d'une simple bordure d'écran dont les dimensions resteront fixées définitivement dans le programme. La seule fonction paramétrable introduite étant le choix possible de la "couleur". Nous verrons plus tard comment améliorer ce programme, d'abord en lui ajoutant le clignotement, ensuite en le rendant totalement paramétrable à partir d'une ligne de Basic. Mais voyons d'abord le programme de base :

Listing ASSEMBLEUR

```
REM * BORDURE D'ECRAN

    LD A,(16381)    Couleur de la bordure
    LD C,A          dans le registre C
    LD B,32         initialisation nombre
                    de colonnes
    LD HL,(16396)   localisation fichier
                    d'affichage

L0  INC HL          saute le 118
    LD (HL),C       trace la
    DJNZ,L0         limite supérieure
    INC HL

    LD DE,31        remise à jour des
    LD B,22         compteurs

L1  INC HL          tracé simultané des
                    deux colonnes
    LD (HL),C       1ère case de la ligne
    ADD HL,DE       ajoute 31
    LD (HL),C       dernière case de la
                    ligne
    INC HL
    DJNZ,L1
    INC HL

    LD B,32         remise à jour des
                    compteurs

L2  LD (HL),C       tracé de la
    INC HL          limite
    DJNZ,L2         inférieure

    RET             retour au basic
```

J'espère que vous êtes désormais en mesure de suivre sans difficulté ce programme au demeurant très linéaire. Si oui je vous invite à passer sans plus attendre à l'étape suivante où nous tâcherons de faire clignoter à un rythme acceptable et pendant une durée déterminée notre encadrement. Mais ceci va nous obliger à aborder un des outils essentiels du langage machine : la Pile.

Si nous nous sommes fort bien passé de la pile dans nos programmes précédents c'est sans doute qu'elle n'est pas tout à fait indispensable ? Oui, mais au sens où n'est pas indispensable au bon fonctionnement de votre ZX son extension 16K : pour de mini-programmes vous vous contentez des quelques milliers d'octets que vous offre sa version de base mais dès que vous en arrivez à développer des programmes élaborés et plus longs elle vous devient indispensable.
La Pile c'est donc de la mémoire supplémentaire ? Pas vraiment... La mémoire du microprocesseur se trouve strictement limitée à ses quelques paires de registres et rien d'autre. Les registres représentent en quelque sorte les antichambres du traitement de l'information. Les données n'y accèdent qu'au dernier moment en fonction des besoins du processeur, lors de leur utilisation immédiate. Mais imaginez un instant que celui-ci ait besoin quasi-simultanément d'une masse d'information dépassant largement la capacité mémoire de ses registres. Que faire alors ? Sans possibilité d'extension, durement sollicité par un programme trop gourmand, le microprocesseur, plongé dans une détresse difficile à imaginer pour des humains, devra-t-il se résoudre à un "plantage" suicidaire ? Non car il reste encore une solution : la Pile.
L'idée est simple : le processeur à court de mémoire va tout simplement utiliser, pour ses besoins immédiats, la mémoire vive du système lui-même (RAM). Il l'exploite à l'intérieur d'une zone protégée d'une façon systématique et particulière qui lui vaut son nom de PILE (en anglais LIFO : last in-first out : dernier entré, premier sorti). Il " empile" les unes sur les autres les données qu'il veut stocker temporairement et n'en retient que l'adresse en mémoire du dernier élément empilé. C'est cette adresse qu'il range dans un registre spécifique à seize bits nommé "pointeur de pile" (SP).
Ainsi chaque opération d'empilement (instruction PUSH) s'accompagne d'une remise à jour du pointeur (décrémentation de SP) tandis que chaque opération de dépilement, d'extraction d'une donnée (instruction POP) s'accompagne d'une incrémentation de SP.
Il y a "empilage" lorsque, devant la saturation de ses registres, le microprocesseur (en réalité le programme d'assemblage) décide de "faire de la place" et libère un registre en en sauvant le contenu à l'adresse pointée par SP. Si un deuxième registre devait être libéré, son contenu serait alors empilé sur le précédent c'est à dire stocké à l'adresse SP-2 (deux octets étant nécessaire pour ranger des adresses sur 16 bits); et ainsi de suite.
Il y a "dépilage" lorsque le microprocesseur, voulant récupérer les données précédemment empilées, les restitue à l'un ou l'autre de ses registres dans l'ordre inverse où il les a empilées. Le pointeur de pile est alors à chaque fois doublement incrémenté (SP+ 2).
Lorsque, pour les besoins d'un programme assembleur, on est amené à se servir de la pile il faut toujours avoir à l'esprit cette règle d'or : après tout usage personnel, la pile doit avoir impérativement retrouvé son niveau initial. Concrètement : avant tout retour au basic comptez toujours les PUSH et les POP de votre programme sous peine d'irrémédiable plantage. Vous voilà avertis pour la prochaine fois !

Bernard GUYOT

Formation à l'assembleur

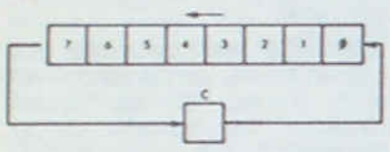
COURS D'ASSEMBLEUR

Vous êtes, grâce à l'HHHHebdo, devenus des quasi-cracks de l'assembleur. Mais ne partez pas ! Les bonnes surprises et les bons plans vont se multiplier dans les prochains cours, vous transportant jusqu'au nirvana des programmeurs. Comme toujours le cours théorique branchera les fans du tordu alors que les bidouilleurs se brancheront directement sur leur cours rien qu'à eux !

Vous avez déjà eu droit aux discours suivants, dans votre page chérie par-dessus tout :

ZX 81 ----> 55 56 61 66 71 76 81 86
ORIC ----> 57 62 67 72 77 82 87
APPLE ----> 58 63 68 73 78 83 88
THOMSON ----> 59 64 69 74 79 84 89
COMMODORE ----> 60 65 70 75 85 90

Rotation à gauche d'une position

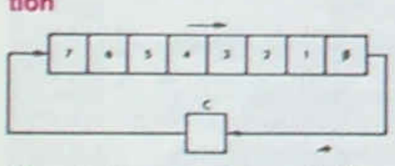


Nous nous servirons de cette instruction pour modifier le multiplicande à chaque phase de calcul du produit partiel. Comme nous allons travailler sur une multiplication sur seize bits, nous aurons besoin de pouvoir effectuer seize décalages à gauche du multiplicande. Nous devrons donc réserver une place mémoire suffisante pour stocker celui-ci soit quatre octets. Avant la rotation du premier octet nous devrons mettre la Carry à zéro, par la suite (pour les trois octets suivants) le bit 7 éjecté de l'octet précédent arrivera automatiquement à la place du bit 0 de l'octet suivant.

Nous devrons aussi nous servir de l'algorithme d'addition donné précédemment dans ce cours pour réaliser les calculs intermédiaires du résultat. Cet algorithme devra être étendu selon le principe indiqué à la suite de l'algorithme.

Vous croyez que notre recherche est terminée et que nous allons pouvoir balancer l'organigramme maintenant ? Malheureusement non ! Il nous reste à trouver comment nous allons tester les bits du multiplicateur un à un. Nous travaillons sur des nombres de seize bits et notre résultat tiendra, au maximum, sur trente deux bits. Si vous vous souvenez bien de ce qui est écrit un peu plus haut, vous verrez que l'instruction de rotation nous sera d'un grand secours car elle ne fonctionne pas seulement vers la gauche mais aussi vers la droite. D'ailleurs en voici la confirmation schématique.

Rotation à droite d'une position



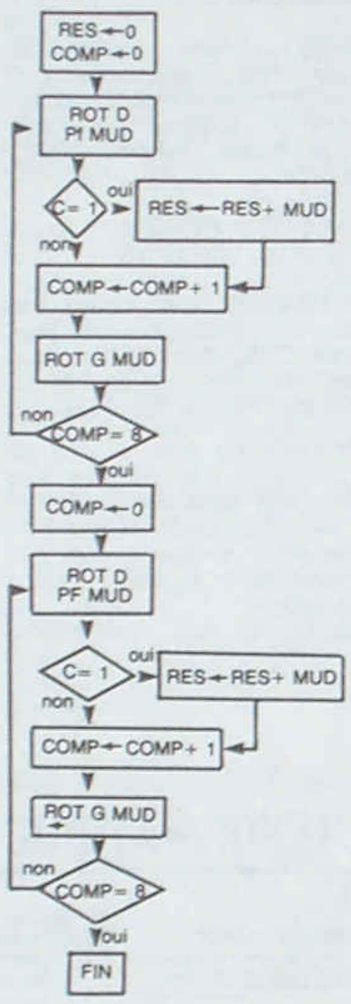
Nous traiterons notre multiplicateur en deux phases, chacune consistant à extraire un à un les huit bits du premier puis du second octet du multiplicateur. La rotation placera dans la Carry le bit qu'il nous intéresse de tester à chaque phase du calcul du produit intermédiaire.

Nous avons maintenant tous les éléments pour traiter notre multiplication sur seize bits. Regardons d'abord la structure de données qui nous sera nécessaire pour la réalisation de notre projet :

* quatre octets pour le multiplicande
* deux octets pour le multiplicateur
* quatre octets pour le résultat

Nous connaissons dorénavant le terrain sur lequel nous allons nous aventurer, reste à organiser la structure de calcul. Nous commencerons notre découverte par l'organigramme.

Multiplication seize bits



RES résultat
COMP compteur
ROT D rotation à droite d'une position
ROT G rotation à gauche d'une position
MUT multiplicateur
MUD multiplicande

Nous déduirons facilement de ce qui précède l'algorithme que je vais vous exposer maintenant. Encore une fois, je ne prétends nullement vous fournir le meilleur algorithme pour effectuer des multiplications sur seize bits, mais ce que j'essaye de vous faire comprendre c'est la démarche à suivre pour aboutir à un algorithme tel que celui que je vous expose maintenant.

Dans cet algorithme, j'utiliserai des noms de routines tels que ADD16 (pour addition sur 16 bits) ou ROLMUD (pour rotation à gauche d'une position du multiplicande) qui correspondent à des algorithmes précis et ponctuels, que nous n'avons pas forcément étudiés, mais d'une simplicité extrême. Encore une fois, je vous rappelle que la numérotation que j'utilise ne correspond

absolument pas à celle que vous imposera votre programme d'assemblage, mais elle clarifie la lecture du schéma de programme.

Algorithme de la multiplication sur seize bits

100 mettre à 0 les quatre octets du résultat

110 mettre à 0 l'octet de comptage (ce peut être un registre d'index)

120 rotation à droite d'une position de l'octet de poids faible du multiplicateur

130 si C= 1 alors le résultat est augmenté de la valeur actuelle du multiplicande (par l'appel à la routine ADD16)

140 le compteur est incrémenté d'une unité

150 traitement du multiplicande par la routine ROLMUD

160 si la valeur du compteur est inférieure à 8 alors aller en 120

170 mettre le compteur à 0

180 rotation à droite d'une position de l'octet de poids fort du multiplicateur

190 si C= 1 alors le résultat est augmenté de la valeur actuelle du multiplicande

200 le compteur est incrémenté d'une unité

210 traitement du multiplicande par la routine ROLMUD

220 si la valeur du compteur est inférieure à 8 alors aller en 180

230 fin

Vous voyez que cette multiplication n'est pas si complexe à calculer. Un seul point obscur reste : nous n'avons pas réfléchi au fonctionnement de la routine nommée ROLMUD dans cet algorithme. Celle-ci doit gérer la rotation à gauche d'une position de l'ensemble des quatre octets que compte le multiplicande.

Vous vous souvenez certainement du principe de fonctionnement de la rotation. Le contenu de la Carry est injecté à droite ou à gauche de l'octet, tous les bits sont décalés, dans le sens de l'injection, et le sortant est stocké dans la Carry. Cette instruction tourne effectivement puisque nous ne perdons à aucun moment une information.

Pour pratiquer la rotation des bits sur quatre octets nous sommes dans l'obligation de nous méfier du contenu de la Carry au moment de l'opération sur le premier octet. Par la suite nous n'aurons plus à la modifier pour profiter de la réinjection, dans l'octet suivant, du bit extrait de l'octet précédent.

Je pense que vous êtes suffisamment armé dans la compréhension des organigrammes pour comprendre la méthode choisie dans cette routine.

L'ASSEMBLEUR PRATIQUE

Langage machine... Sur ZX81

L'heure est donc venue pour nous les risque-tout du langage machine, les artificiers de l'assembleur pratique, de mettre à l'épreuve de la réalité les élucubrations trop théoriques de notre précédent rendez-vous. Rappelez-vous que notre sagacité s'exerçait alors à comprendre le fonctionnement d'un accessoire presque indispensable au microprocesseur : la pile. Si vous pensez "bien sûr ! Sans pile pas de courant" ne forcez pas d'avantage sur votre mémoire et sur votre jugement, ça pourrait avoir des conséquences encore plus fâcheuses pour vous. Il vaut mieux sortir de l'azote liquide le numéro soigneusement cryogénisé de l'HHHebdo en question; laissez-le doucement dégeler sur le rebord de la fenêtre en l'exposant thermostat 4 environ. Bien l'essorer avant d'enlever à l'épluche-légumes les peaux successives qui dissimulent la précieuse Page Pédago. Faites ensuite revenir les caractères d'imprimerie trop pâles à l'aide d'une grande poêle n'attachant pas et de votre renforçateur habituel. Servez dans un vaste plat comme garniture de votre micro habituel, ou bien... en salade, cela va sans dire !
Mais ne nous égarons pas et essayons de retomber pile sur notre sujet : celle du Z80 est une structure permettant au processeur de gérer d'une façon originale la mémoire vive d'un système informatique (pile software) pour ses besoins personnels les plus urgents. Ces besoins peuvent être regroupés en trois catégories distinctes :

1-Les interruptions : le processeur en cours de travail est brutalement sollicité par un périphérique pour un travail qui devient alors prioritaire.

2-Les sous-programmes : nous verrons qu'il existe en langage symbolique d'assemblage des instructions analogues aux GOSUB / RETURN du Basic.

3-Le stockage des données immédiatement accessibles. C'est sur ce point que portera notre application.

Dans tous les cas ce qui est demandé à la pile, c'est la sauvegarde rapide dans la mémoire vive du système (RAM) du contenu d'un ou plusieurs des registres du processeur. Ceci dans un premier temps, le second étant la restitution, au moment opportun, de ces mêmes valeurs sur le ou les registres concernés. Rappelons que la sauvegarde s'effectue par empilage des données les unes sur les autres dans chaque octet de la RAM et que leur restitution a lieu de manière inverse par dépilage, le dernier élément rentré étant le premier à sortir; c'est là ce qui vaut à la chose son nom de PILE puisque son fonctionnement est analogue à celui d'une pile d'assiettes creuses, d'une pile de codes civil, d'une pile de lingeries fines, d'une pile de meubles d'époque ou d'une pile de piles... Deux procédures sont caractéristiques de toutes les piles : d'abord on a accès qu'au dernier élément empilé, ensuite on est souvent intéressé par un élément qui se trouve au beau milieu de la pile. C'est précisément là que réside la difficulté de manipulation de la pile : où mettre les éléments dont on a nul besoin avant d'arriver à celui qui nous intéresse de suite ! La hauteur (ou profondeur) quelquefois impressionnante de la pile et le peu de registres disponibles apparente la stratégie à utiliser à celle du casse-tête célèbre : "tours de Hanoï".

N'oubliez pas honorables machinistes que le moteur de cette pile est constitué par un registre double SP autrement dit Stack Pointer, dit autrement Pointeur de Pile. Important : il contient l'adresse en mémoire du dernier élément empilé !
Avant de sortir la trousse à outils encore deux mots : PUSH j'empile, POP je dépile. PUSH et POP emploient toutes deux comme argument le registre double sur lequel elles vont travailler. On ne peut PUSHER ou POPER un registre simple à 8 bits. Les paires en question peuvent être indifféremment BC, DE, HL et même AF (Accumulateur + Indicateur). La sauvegarde urgente de AF intervient lors des interruptions et/ou de l'exécution des sous-programmes. En effet le microprocesseur abandonnant provisoirement un travail en cours pour un autre plus urgent, devra au préalable sauvegarder toutes les valeurs de tous les registres de façon à pouvoir ultérieurement retrouver les paramètres nécessaires à la reprise de son travail initial.

Examinons concrètement la troisième catégorie d'utilisation : le Z80 ne possédant, pour ses opérations courantes, que les trois paires de registres déjà citées, on arrive rapidement à leur saturation et donc à la nécessité de stocker ailleurs les données sur lesquelles notre programme machine va travailler. Si nous reprenons notre listing précédent "tracé d'une fenêtre", nous constatons qu'en l'espace d'une vingtaine d'instructions tous les registres doubles courants ont été utilisés. Si nous voulons maintenant apporter à ce petit programme l'un des perfectionnements prévus de longue date -inversion vidéo à fréquence et durée déterminées- nous nous voyons déjà contraints d'utiliser la pile.
Tout d'abord quels modules de traitement devrons-nous adjoindre au programme principal déjà réalisé ? Ils sont au nombre de trois :

1-Une temporisation qui se chargera de régler la cadence du clignotement (vidéo normale/vidéo inverse).

2-L'inversion vidéo du caractère choisi pour la bordure d'écran.

3-Un compteur de tours qui déterminera la durée de fonctionnement de notre routine.

Le programme de base et ses adjonctions pourraient se présenter de la façon suivante :

LISTING ASSEMBLEUR

```
REM * PROGRAMME PRINCIPAL
    LD A.(16381)    initialisation
    LD C.A          et sauvegarde
    LD E.150        du compteur de
:L5 PUSH DE         clignotements
    LD B.32
    LD HL.(16396)
:L0 INC HL
    LD (HL).C
    DJNZ.L0
    INC HL
    LD DE.31
    LD B.22
:L1 INC HL
    LD (HL).C
    ADD HL.DE
    LD (HL).C
    INC HL
    DJNZ.L1
    INC HL
    LD B.32
:L2 LD (HL).C
    INC HL
    DJNZ.L2

REM * TEMPORISATION
    LD D.12
:L4 LD E.255
:L3 DEC E
    JR NZ.L3
    DEC D
    JR NZ.L4

REM * INVERSION VIDEO
    LD A.C
    ADD A.$80
    LD C.A

REM * DECOMPTE DES CLIGNOTEMENTS
    POP DE
    DEC E
    JR NZ.L5
```

Vous remarquerez que nous n'avons ajouté au programme de base que l'initialisation du compteur de clignotements. C'est le registre E qui nous servira à ranger cette valeur; mais ce registre nous est également indispensable pour d'autres tâches : addition du facteur 31, temporisation. Afin de le libérer pour ces travaux, nous sauvegardons notre valeur 150 par un PUSH DE. Le registre DE se trouve dès lors disponible pour accueillir d'autres données. Nous ressortirons de la Pile la valeur 150 lorsque nous effectuerons en fin de programme sa décrémentation au niveau du module de décompte des clignotements (POP DE; DEC E).

Une instruction de saut conditionnel (JR NZ.L5) nous permet de sauvegarder cette nouvelle valeur dans la Pile à l'emplacement mémoire où était stockée la valeur précédente, ceci tant que le compteur de clignotements n'aura pas atteint la valeur 0; nous revenons donc au PUSH DE du programme principal pour une nouvelle exécution. La règle d'or qui préside à toute manipulation de la pile se trouve ici parfaitement respectée : avant de rendre la main au Basic le programme ne peut manquer d'exécuter le POP DE qui restaure la pile à son niveau initial. Attention ! Il ne suffit pas toujours de compter les PUSH et les POP d'un programme LM pour s'assurer du respect de cette règle, encore faut-il que ces instructions aient été exécutées en nombre égal pendant le déroulement du programme, ce qui n'est pas toujours évident dans des programmes traversés de sous-programmes et de JUMP.
Il y avait bien d'autres façons d'arriver à nos fins; notamment un programme mieux construit ne ferait pas répéter systématiquement au processeur la construction de la fenêtre, il se contenterait d'inverser la vidéo d'un dessin réalisé une fois pour toute. Mais ce programme très linéaire n'a d'autre but que de vous rendre transparent l'emploi des instructions du langage symbolique d'assemblage.
Vous avez bien mérité que je vous laisse la concision et le style !

Bernard Guyot

L'ASSEMBLEUR PRATIQUE

Langage machine... Sur THOMSON

Pour en finir avec les indicateurs arithmétiques et logiques du registre d'état, nous allons tenter de parfaire vos connaissances en la matière. Dernièrement, nous avons parlé des instructions de branchement propres à chaque indicateur. Il nous reste à parler, entre autres, de celles qui testent la combinaison de "plusieurs" bits du registre d'état. Voici donc ces fameuses intructions de branchement (en CPL2 et relatif au PC); à noter qu'elles concernent les indicateurs N, Z, V, C :

BGE (branch if greater than or equal to) : un branchement "si supérieur ou égal à" est effectué, si les indicateurs N et V sont tous les deux à 1 ou à 0 et ce lors de l'utilisation de valeurs en CPL2 (comparaison ou opération). Donc si (N (XOR) V)= 0 (revoir à ce propos, les opérateurs logiques du n° 79).

BGT (branch if greater than) : un branchement "si supérieur à" est effectué, si les indicateurs N et V sont tous les deux à 1 ou à 0 et si Z est à 0, lors de l'emploi de valeurs en CPL2. Donc si Z AND (N (XOR) V)= 0.

BHI (branch if higher) : un branchement "si supérieur" est réalisé, si les indicateurs C et Z sont tous les deux à 0, lors de l'emploi de valeurs non signées. Donc si (C OR Z)= 0.

BLE (branch if less than or equal to) : un branchement "si inférieur ou égal à" est réalisé, si l'un OU l'autre des bits N et V est à 1 et si le bit Z est à 1, lors de l'utilisation de valeurs en CPL2. Donc si Z OR (N (XOR) V)= 1.

BLS (branch if lower or same) : un branchement "si inférieur ou égal à" est effectué, si l'un ou l'autre ou bien les deux indicateurs C et Z sont à 1, lors de l'emploi de valeurs non signées. Donc si (C OR Z)= 1.

BLT (branch if less than) : un branchement "si inférieur à" est réalisé, si l'un OU l'autre des indicateurs N et V est à 1, lors de l'emploi de valeurs en CPL2. Donc si (N (XOR) V)= 1.

De plus, il est bon de noter que l'instruction BHS (branch if higher or same) "branchement si supérieur ou égal", est IDENTIQUE à l'instruction BCC (déjà étudiée dans notre exemple du N° 89), en ce sens qu'elle effectue un branchement lorsque le bit C est à 0. De même, l'instruction BLO (branch if lower) "branchement si inférieur", est identique à BCS (N° 89), mais cette fois le branchement s'effectue si C est à 1. Donc les instructions BHS-BCC possèdent un même code opératoire, ainsi que BLO-BCS. BRA (branch always), branche dans tous les cas (branchement inconditionnel). BSR (branch to subroutine), branche à un sous programme également dans tout les cas. BRN (branch never), comme son nom l'indique ne branche jamais.

- Ouf ! on passe à autre chose ?..

Attendez, encore un instant d'attention. Les branchements que nous venons d'étudier sont de type COURT, il nous reste à examiner les branchements de type LONG.

- ???

Oui, nous avons déjà expliqué comment s'effectue un branchement relatif au PC en mode CPL2. A la suite d'une instruction de branchement, on indique une valeur en CPL2 codée sur "un octet". Cette valeur, ajoutée à l'adresse que contient le PC, précise l'adresse où doit s'effectuer le branchement. Donc dans le cas d'un branchement court, cette valeur signée est située entre + 127 et -128. Ce qui signifie que l'on peut avancer d'un maximum de 127 octets dans la mémoire ou reculer au plus de 128. Ce branchement est de type COURT. Il est possible de couvrir les 65.536 adresses de la mémoire par un branchement de type LONG. Pour cela, l'instruction de branchement précédée de la lettre "L", spécifie un branchement LONG et la valeur à rajouter au PC doit être de 16 bits; donc codée sur "2 octets" et signée par l'octet de poids fort (vous savez, celui de gauche). Ainsi, on peut avancer dans la mémoire d'un maximum de 32767 octets ou reculer de 32768.

- Donc, contrairement à ce que vous disiez, on ne peut accéder à toute la mémoire.

Mais si, admettons que vous désiriez effectuer un branchement négatif inférieur à -32768 octets. Un branchement positif peut être équivalent de la façon suivante : monté jusqu'en FFFF (65535) on retombe (bing !) à 0 et on "remonte" la mémoire, jusqu'à l'adresse voulue. De la sorte, un branchement positif suffisant, remplace un branchement négatif impossible et inversement.

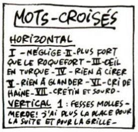
Comme un exemple est toujours plus explicite qu'une tonne de charabia, nous avons pensé à un petit exemple évidemment très décomposé. Voici en bref, l'algorithme pas possible qui va permettre de vous familiariser avec les branchements expliqués plus haut. Notre programme commence par tester le clavier et effectue une suite de comparaisons entre des valeurs déterminées et la valeur ASCII d'une touche éventuellement tapée. Le résultat de ces cogitations nous vaut l'affichage de la catégorie de la touche en question : majuscule, minuscule, ponctuation etc... Pour cela, on utilise deux routines système :

- GETC$ en $ E806, scrute le clavier et place dans le registre B le code ASCII d'une touche enfoncée. Si aucune touche n'est enfoncée, B contient 0.
- PUTC$ en $ E803 déjà employée, affiche à l'écran le caractère dont le code ASCII est dans B.

A noter que nous utilisons en début de programme, la pseudo instruction EQU qui assigne une étiquette à chacune de ces routines. De plus, la routine GETC$ se trouvant à plus de 127 octets de l'instruction de branchement, nous n'avons pas hésité à effectuer un branchement long (LBSR) à celle-ci pour justifier notre propos (nous aurions pu utiliser un JSR >$ E806). Pour le reste, diverses comparaisons branchent ou ne branchent pas aux sous-routines d'affichage (PRINT1, PRINT2 etc...) faciles à comprendre. Les différents messages sont stockés dans des tables et X contient l'adresse du début de la table concernée. Il suffit d'afficher le contenu de l'adresse pointée par X, qui est incrémenté jusqu'à la fin du message (comparaison avec 20, code hexa de 32 pour espace) et on recommence...

Question : savez-vous pourquoi le message "inexistant" ne s'affiche jamais ?..

Les testeurs fous habituels plus Nicolas BOURDIN le brillant petit nouveau.




```
7D01                  ORG    $7D01
        E806   GETC   EQU    $E806
        E803   PUTC   EQU    $E803
7D01 17 6B02   DEBUT  LBSR   GETC
7D04 C1 02            CMPB   #$2
7D06 27 29            BEQ    FIN
7D08 C1 00            CMPB   #0
7D0A 27 F5            BEQ    DEBUT
7D0C C1 21            CMPB   #33
7D0E 2D 22            BLT    PRINT1
7D10 C1 2F            CMPB   #47
7D12 2F 23            BLE    PRINT2
7D14 C1 39            CMPB   #57
7D16 23 24            BLS    PRINT3
7D18 C1 3A            CMPB   #58
7D1A 2C 0E            BGE    CHIF
7D1C C1 5A     SUITE  CMPB   #90
7D1E 2E 03            BGT    MINU
7D20 7E 7D41          JMP    PRINT4
7D23 C1 7A     MINU   CMPB   #122
7D25 22 1F            BHI    PRINT5
7D27 7E 7D4B          JMP    PRINT6
7D2A C1 41     CHIF   CMPB   #65
7D2C 2D 09            BLT    PRINT2
7D2E 7E 7D1C          JMP    SUITE
7D31 3F        FIN    SWI
7D32 8E 7D85   PRINT1 LDX    #MESS1
7D35 20 17            BRA    PRINT
7D37 8E 7D79   PRINT2 LDX    #MESS2
7D3A 20 12            BRA    PRINT
7D3C 8E 7D95   PRINT3 LDX    #MESS3
7D3F 20 0D            BRA    PRINT
7D41 8E 7D65   PRINT4 LDX    #MESS4
7D44 20 08            BRA    PRINT
7D46 8E 7D8A   PRINT5 LDX    #MESS5
7D49 20 03            BRA    PRINT
7D4B 8E 7D6F   PRINT6 LDX    #MESS6
7D4E E6 80     PRINT  LDB    ,X+
7D50 C1 20            CMPB   #32
7D52 27 05            BEQ    SAUT
7D54 BD E803          JSR    PUTC
7D57 20 F5            BRA    PRINT
7D59 C6 0D          SAUT   LDB    #13
7D5B BD E803               J3R    PUTC
7D5E C6 0A                 LDB    #$A
7D60 BD E803               JSR    PUTC
7D63 20 9C                 BRA    DEBUT
7D65 4D 41 4A 55    MESS4  FCC    'MAJUSCU
7D69 53 43 55 4C
7D6D 45 20
7D6F 4D 49 4E 55    MESS6  FCC    'MINUSCU
7D73 53 43 55 4C
7D77 45 20
7D79 50 4F 4E 43    MESS2  FCC    'PONCTUA
7D7D 54 55 41 54
7D81 49 4F 4E 20
7D85 52 49 45 4E    MESS1  FCC    'RIEN '
7D89 20
7D8A 49 4E 45 58    MESS5  FCC    'INEXIST
7D8E 49 53 54 41
7D92 4E 54 20
7D95 43 48 49 46    MESS3  FCC    'CHIFFRE
7D99 46 52 45 20
        0000               END
```

```
10 CLEAR,32000:CLS
20 FOR I=&H7D01 TO &H7D9C
30 READ A$:POKEI,VAL("&H"+A$)
40 NEXT:EXEC &H7D01
50 DATA 17,6B,02,C1,02,27,29,C1,00,27,F5
,C1,21,2D,22,C1,2F,2F,23,C1,39,23,24,C1,
3A,2C,0E,C1,5A,2E,03,7E,7D,41,C1,7A,22,1
F,7E,7D,4B,C1,41,2D,09,7E,7D,1C,39,8E,7D
,85,20,17,8E,7D,79,20,12,8E,7D,95,20,0D
60 DATA 8E,7D,65,20,08,8E,7D,8A,20,03,8E
,7D,6F,E6,80,C1,20,27,05,BD,E8,03,20,F5,
C6,0D,BD,E8,03,C6,0A,BD,E8,03,20,9C,4D,4
1,4A,55,53,43,55,4C,45,20
70 DATA 4D,49,4E,55,53,43,55,4C,45,20,50
,4F,4E,43,54,55,41,54,49,4F,4E,20,52,49,
45,4E,20,49,4E,45,58,49,53,54,41,4E,54,2
0,43,48,49,46,46,52,45,20
```

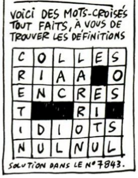
L'ASSEMBLEUR PRATIQUE

Langage machine... Sur ZX 81

Nous nous sommes déjà servi de la pile dans le cours précédent mais nous n'avons fait qu'illustrer le principe de son fonctionnement sans vraiment aborder la complexité possible de sa mise en oeuvre. Notre programme ne possédait en tout et pour tout qu'un PUSH et corollairement son POP. Autant dire que nous n'avons empilés puis dépilés que deux malheureux octets, initiation oblige !

Est-ce à dire que je vous propose aujourd'hui un casse-tête à base de PUSH et de POP ? L'inextricable enchevêtrement des différents niveaux de fonctionnement de la pile où s'égarerait le peu de raison qui vous reste encore à investir dans la conquête du langage symbolique d'assemblage. Vous l'avez échappé belle : non ! Et si la tête vous tournait ce ne serait que la conséquence naturelle de l'exécution de notre programme d'aujourd'hui : la rotation de l'image de votre ZX. Ce projet ambitieux peut se concevoir de deux façons différentes selon votre sensibilité personnelle. Certains -très techniques- le présenteront comme un programme simulant le dérèglement de la stabilité horizontale d'un téléviseur, tandis que d'autres, plus optimistes et esthètes dans l'âme y verront un moyen supplémentaire d'attirer sur leur écran cathodique toutes les attentions. Quoiqu'il en soit l'algorithme et le programme restent les mêmes. Nous allons aussi, à cette occasion, tenter de jongler un peu plus habilement avec la pile, d'empiler et de dépiler à bon escient et sur une "hauteur" plus conséquente. Ah ! vivre dangereusement...

Vous avez tous en mémoire la structure très simple du fichier d'affichage du ZX81; la succession linéaire, en mémoire, des octets qui le constitue ainsi que les caractères codés 118 qui ouvrent le fichier et terminent chacune de ses 24 lignes. La rotation de l'image s'apparente à un



travail que nous avons déjà réalisé : le scrolling. Il s'agit en fait d'un scrolling dans lequel on récupère à chaque tour la colonne qui sort de l'écran pour la reconstituer de l'autre côté. Si par exemple je décide de faire tourner mon image de gauche à droite, son premier décalage vers la droite provoquera la sortie du contenu de la colonne 31 et sa reduplication immédiate sur la colonne 0. Cette même opération renouvelée trente-deux fois donnera l'impression d'une rotation complète de l'image.

Comme toujours en matière de programmation, nombreux sont les chemins qui permettent d'arriver au même résultat. Celui que je vous propose d'emprunter ici permet peut-être plus qu'un autre de rendre transparente la démarche logique et la structure du programme.

Notre rotation d'image se composera simplement des 24 rotations successives de toutes les lignes d'écran. Le programme s'articule autour d'un module de décalage d'une case vers la droite avec duplication de la dernière case à droite dans la première case à gauche. Le reste est constitué de différents compteurs (lignes, colonnes, nombre de décalages de l'image) pour lesquels la pile nous sera d'un précieux secours.

Le programme se présente donc de la façon suivante :

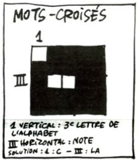
```
REM * ROTATION DE L'IMAGE

REM * INITIALISATIONS
    LD E.32  compteur décalage global
             de l'image
    PUSH DE
:L2 LD HL.(16396)
    INC HL
    PUSH HL  première case visible FA
    LD C.24  compteur lignes
:L0 LD A.(HL)
    LD B.31  compteur colonnes

REM * ROTATION LIGNE 1 CASE
:L1 INC HL
    LD D.(HL)
    LD (HL).A
    LD A.D
    DJNZ.L1
    POP HL
    LD (HL).A

REM * CHANGEMENT DE LIGNE
    LD DE.33
    ADD HL.DE
    PUSH HL  première colonne
             ligne suivante
    DEC C    décompte lignes
    JR NZ.L0

REM * TEMPORISATION
    LD D.15
:L8 LD E.255
:L9 DEC E
    JR NZ.L9
    DEC D
    JR NZ.L8

 *  POP HL   pour retrouver...
    POP DE   le nb décalages image
    DEC E    décompte...
    PUSH DE  ...et nouvelle sauvegarde
    JR NZ.L2

 *  POP DE   restauration de la pile

    RET
```

Vous serez peut-être surpris de constater, dès le début du programme, au niveau des initialisations, la présence d'instructions d'empilage PUSH. Non pas que nos registres soient déjà encombrés par l'exécution d'une tâche, mais la sauvegarde immédiate des valeurs de départ des compteurs donne à ces registres la disponibilité dont nous aurons besoin immédiatement après. C'est bien le moment de sauvegarder ces valeurs dans la pile puisque nous n'aurons à travailler sur elles qu'une fois le programme exécuté dans sa quasi-totalité. Ceci est tout à fait évident pour la décrémentation du compteur de décalage global de l'image qui intervient à l'extrême fin du programme, après le module de temporisation (POP DE; DEC E).

Pour éviter de se perdre dans l'utilisation des différents niveaux de la pile, on peut agir avec elle suivant une stratégie analogue à celle des boucles imbriquées du Basic : les premiers PUSH sont à mettre en rapport avec l'exécution d'un travail qui englobera les suivants, et ainsi de suite ...

Si nous reprenons ce mode de représentation pour notre programme nous obtenons le schéma suivant :

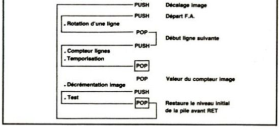
On voit ainsi clairement comment se combinent sans jamais s'enchevêtrer les différents niveaux de la pile. Les instructions POP précédées de "* " sont signalées parce que caractéristiques de la stratégie à mettre en oeuvre sur les structures LIFO (dernier entré, premier sorti). Ces deux * POP n'ont d'autre utilité que de faire pointer SP sur le niveau de pile qui nous intéresse à ce moment précis de l'exécution du programme. Autrement dit * POP HL et * POP DE utilisent respectivement les registres HL et DE comme de simples exutoires où les valeurs déposées ne sont d'aucune utilité. La première (* POP HL) permet à l'instruction suivante POP DE de ranger dans DE la valeur du compteur image que l'on doit décrémenter; la seconde (* POP DE) redonne à la pile le niveau qui était le sien avant que nous l'ayons utilisé. En somme la philosophie de la pile pourrait se résumer dans cet axiome universellement connu : "prière de laisser ces lieux dans l'état où vous les avez trouvé en entrant".



Je suppose qu'avec les connaissances déjà acquises le reste du programme n'est pas trop difficile à digérer. Voici quand même pour les plus constipés la description succinte du fonctionnement dynamique du programme. Il n'y aura plus ensuite qu'à tirer la chasse...

Après les indispensables initialisations on effectue le décalage successif des 24 lignes. Pour cela on sauvegarde provisoirement dans D l'octet qui sera occuppé par A (LD (HL).A) puis A prend la valeur de D avant que celui-ci ne soit chargé avec le contenu de l'octet suivant. Ainsi de suite jusqu'à la dernière case de la ligne où la restauration de l'adresse du début de ligne (POP HL) permet le chargement de la première case par le contenu de cette dernière. L'addition du facteur 33 permet de pointer sur le début de la ligne suivante qu'on sauvegarde à nouveau comme référence de début de ligne. La répétition de cette opération sur les 24 lignes aboutit à la rotation globale de l'image d'une case vers la droite. La temporisation obtenue grace à deux boucles imbriquées est facilement réglable. Elle permet aux curieux de disséquer le fonctionnement du programme et aux plus pressés de s'étourdir la vue. Tournez manège !

Bernard Guyot



Formation à l'assembleur

COURS D'ASSEMBLEUR

Vous êtes, grâce à l'HHHHebdo, devenus des quasi-cracks de l'assembleur. Mais ne partez pas ! Les bonnes surprises et les bons plans vont se multiplier dans les prochains cours, vous transportant jusqu'au nirvana des programmeurs. Comme toujours le cours théorique branchera les fans du tordu alors que les bidouilleurs se brancheront directement sur leur cours rien qu'à eux !

Vous avez déjà eu droit aux discours suivants, dans votre page chérie par-dessus tout :

ZX 81 → 55 56 61 66 71 76 81 86 91 94
ORIC → 57 62 67 72 77 82 87 92 94
APPLE → 58 63 68 73 78 83 88 93 94
THOMSON → 59 64 69 74 79 84 89 94 98
COMMODORE → 60 65 70 75 85 90 94 99

Nous en sommes resté, la dernière fois, à une question particulièrement angoissante : comment résoudre le problème de priorité dans les interruptions, en particulier celles provoquées par les périphériques. Pour cette dernière exploration dans le cerveau des concepteurs d'ordinateurs, nous devrons faire revenir à notre mémoire certaines des règles évoquées lors du discours sur les sous-programmes.

Déjà, avant d'étudier à fond cette énigme, nous allons devoir établir une catégorisation des interruptions pour établir une hiérarchie de celles-ci. En effet, il paraît difficilement imaginable de laisser le micro-processeur se débrouiller sans aucun point de repère dans ce qui peut devenir une tempête de signaux électriques en provenance des ports d'entrées-sorties.

Jusqu'à présent, nous n'avons tenu compte que des seules interruptions en provenance de l'extérieur, celles venant de périphériques en somme. De fait ces interruptions possèdent généralement l'immense avantage d'être programmables par l'utilisateur. En revanche une seconde catégorie, celle-ci interne à l'ordinateur, restera inaliénable et intouchable autrement qu'en se saisissant de son fer à souder et en modifiant certains des branchements sur les cartes. De ces deux catégories, que nous allons nommer pour plus de commodité, nous tirerons une première échelle de priorité dans le traitement des interruptions.

Une chose paraît certaine : les interruptions internes au système, celles qui sont dites **non masquables**, ne pourront jamais passer après une interruption d'origine extérieure à l'ordinateur. Les interruptions provenant des périphériques sont dites, elles, **masquables**. Ce terme sous-entend une chose très simple : dans le registre d'état, que nous avons vu dans ce cours il y a fort longtemps, se trouve un bit essentiel à la manipulation des interruptions. Bien entendu suivant les machines, d'autres registres interviendront dans le traitement de ces interruptions, registres programmables directement par l'utilisateur. L'organisation de ces registres reste propre à chacune des machines, variant même avec un même micro-processeur : les interruptions ne se programmeront pas forcément de la même façon sur un Atmos ou un Apple alors que le 6502 règne à l'intérieur des deux machines.

Que se passe-t-il donc lorsque vous décidez, en temps que programmeur, de neutraliser temporairement les interruptions en provenance de l'extérieur du micro-ordinateur ? Les périphériques n'ont aucune raison de s'arrêter de formuler des requêtes en direction du micro-processeur. Celui-ci étant protégé contre ces troubles, les demandes sont stockées dans un endroit de la mémoire jusqu'à ce que vous autorisiez à nouveau la circulation de ces requêtes jusqu'au micro-processeur.

Dans les interruptions non masquables il existe aussi une hiérarchie interne. Ainsi une interruption demandant la réinitialisation du système (reset tiède) passera devant toutes ses collègues, quel que soit le contexte. Dans les gros systèmes informatiques, l'interruption la plus prioritaire sera celle qui provoquera la sauvegarde automatique de tous les travaux en cours sur disque dur lors de la détection d'une chute de tension sur le réseau d'alimentation du système. L'importance des travaux en cours ne permet pas de perdre la moindre information, aussi dès la chute de tension captée par le micro-processeur, celui-ci interrompt tous les travaux en cours et effectue une sauvegarde d'urgence de toute la mémoire volatile.

Revenons à un niveau plus général du traitement d'exception qu'est celui des interruptions. Comme nous avions pu le voir pour les sous-programmes, à l'intérieur d'une routine en cours d'exécution, on peut effectuer un appel à un autre sous-programme. Le nombre d'appels imbriqués autorisés par un micro-processeur est déterminé par la capacité de stockage de la pile système. Cette capacité à accepter un nombre donné d'appels imbriqués se nomme la profondeur d'appel. De la même façon, une routine d'interruption peut être elle-même interrompue pour laisser la priorité à l'exécution d'une interruption plus urgente à traiter. De la même façon que pour les sous-programmes, la pile sert alors de zone de stockage des adresses de retour en fin d'exécution des différentes routines.

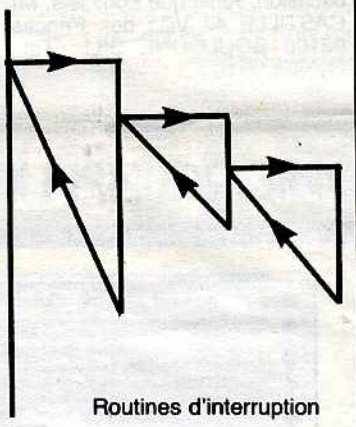
Pour qu'une routine d'interruption puisse interrompre sans vergogne une autre routine d'interruption en cours de traitement, il faut nécessairement que celle en cours d'exécution soit strictement moins prioritaire que la nouvelle à se présenter à la porte du micro-processeur. De cette règle, les concepteurs ont tiré des méthodes de codage qui permettent au micro-processeur, en testant simplement le contenu de son registre d'état, de savoir si la requête présentée est recevable ou non à l'instant où elle arrive.

Dans les nouveaux micro-systèmes informatiques (Amiga, Atari 520 ST ou Macintosh) des micro-processeurs esclaves du 68000 gèrent les interruptions de telle manière que le 68000 ne s'occupera que du traitement effectif des routines d'interruption. Si une requête de moindre importance se présente, les co-processeurs s'occuperont de la faire patienter jusqu'à ce que l'état du registre de status change et autorise cette interruption.

Ces interruptions d'interruptions paraissent fort complexe, à contempler sur le papier un descriptif littéraire. Une schématisation graphique de la chose simplifiera sans aucun doute le raisonnement de manière à éclairer définitivement votre lanterne.

Le premier dessin illustrera la méthode d'imbrication des routines d'interruption. A chaque interruption nouvelle, le processeur sauvegarde l'ensemble des registres dans la pile ainsi que l'adresse à laquelle il devra reprendre le travail une fois la routine exécutée.

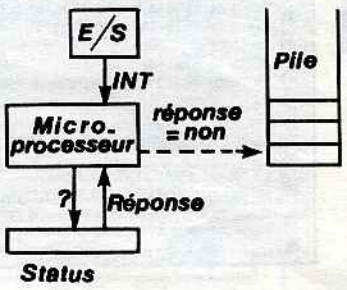
Programme



Routines d'interruption

Imbrication de routines d'interruption

La seconde illusration montrera le cheminement suivi par le micro-processeur lors d'une requête d'interruption. L'interrogation du registre d'état lui indiquera s'il doit accomplir la routine d'interruption correspondant à la requête ou s'il devra mettre celle-ci en attente.



Cheminement du micro-processeur pour une interruption

Vous êtes dorénavant armé pour faire face aux multiples problèmes posés par la programmation des interruptions. Pour tirer un parti maximum de votre machine, vous serez tenu de suivre les spécifications qui lui sont propres, mais le schéma général exposé ici suffit amplement à exposer le schéma de fonctionnement général de cette méthode de programmation du micro-processeur.

L'ASSEMBLEUR PRATIQUE

Langage machine... Sur ZX 81

Enrichir son vocabulaire est une façon sûre de progresser dans la connaissance du langage symbolique d'assemblage. L'agencement judicieux des instructions en vue de la bonne exécution d'un programme relève d'une pratique souvent longue à acquérir, mais l'apprentissage d'une fonction nouvelle est heureusement quasi-instantané. Posséder le vocabulaire le plus large possible est une condition indispensable à l'élaboration de programmes optimisés, c'est-à-dire qui exploitent au mieux les possibilités d'un microprocesseur donné. Je vous propose aujourd'hui la mise en oeuvre pratique de nouvelles instructions d'assemblage dans le cadre d'un petit programme dont l'objectif pourrait vous sembler paradoxal. Pourquoi ? Parce qu'il vise à la suppression d'une instruction du langage concurrent : le SAVE du Basic. Considérez comme déclarée la guerre des langages informatiques !
En situation de guerre; c'est bien connu, tous les coups sont permis, les plus vicieux étant généralement considérés comme les plus héroïques. Ainsi l'infiltration d'espions dans les lignes ennemies, les substitutions d'identité, la disparition subite d'un chef d'état-major au sein même de son QG, tout cela vous fait courir un frisson délicieux au long de l'échine, de la base du képi au talon de la botte !
Notre mission est à peu près analogue : il s'agit d'infiltrer souterrainement le listing Basic ennemi, de remonter insidieusement jusqu'à la tête-de-pont constituée par l'instruction SAVE et de la neutraliser en lui substituant un trompe-l'oeil, un élément qui fera diversion. Vous n'êtes pas sans ignorer, Messieurs, l'importance de cette mission. Privée de son SAVE-conduit un programme Basic ne sait plus s'orienter. Où commence-t-il ? En ligne 20, en ligne 1000 ou 2000 ? Nul ne saura plus le dire ! Il sera dès lors facile de multiplier les fausses pistes, de tendre des pièges, des chausse-trappes à tous ceux qui s'aventureraient à dupliquer le logiciel ainsi protégé ! Ah !Ah !Ah ! -(rire inquiétant...). Préparez le commando d'élite binaire pour l'opération MATATARI ! Messieurs vous pouvez disposer ! Ah !Ah !Ah ! -(rire satanique résonnant longuement sous les voûtes humides et sombre du bunker).

Pour le programmeur néophyte l'intérêt de l'opération réside également dans la découverte d'une possibilité d'action, par le langage machine, sur le corps même d'un programme Basic déjà installé en mémoire; l'interaction de l'assembleur et du Basic démultipliant les possibilités des deux langages.

Entrons, baïonette au canon, dans le vif du sujet.
Un programme Basic stocké dans la mémoire du ZX81 est en réalité composé d'une succession d'octets codés représentant, dans leur ordre respectif :
1. Le numéro de ligne (sur deux octets)
2. La longueur de la ligne (sur deux octets)
3. L'instruction Basic avec son argument.
Cette suite d'octets commence obligatoirement à l'adresse 16510 et s'achève au début du fichier d'affichage. C'est donc dans les limites de cet espace que nous chercherons l'instruction SAVE. Chaque instruction étant codée sur un octet unique, à SAVE correspond la valeur 248. Mais cette valeur peut se retrouver fortuitement dans un numéro de ligne ou encore dans l'argument d'une fonction, c'est la raison pour laquelle, afin de limiter les risques de confusion, notre recherche portera sur une chaîne d'octets composé de l'octet "SAVE", du guillemet qui le suit fatalement, et de deux caractères choisis arbitrairement qui seront placés impérativement après le guillemet (ici "* " en vidéo inversée).
Une fois le SAVE localisé, il suffira de lui substituer un caractère quelconque; j'ai choisi pour de machiavéliques raisons l'innocente instruction REM Ah !Ah !Ah ! Et pour donner à cette ligne de REM un aspect tout à fait normal il convient de supprimer le guillemet et les deux "$ ". Je les supprime sans l'ombre d'un remords Ah !Ah !Ah !-(dangereux)- N'oublions pas non plus l'inversion vidéo automatique de la dernière lettre de l'argument de SAVE. Je la rétablis dans sa "couleur" initiale en en soustrayant le facteur 128. Il ne reste plus qu'à effacer le dernier guillemet et le mauvais tour est joué...
Cela donne le programme suivant dont nous allons examiner certaines instructions inédites à ce jour : CPIR, XOR, SBC.

Le premier module de traitement a pour objectif essentiel de délimiter l'espace sur lequel portera notre recherche, c'est-à-dire la totalité du listing Basic en mémoire. Pour cela on prépare les registres doubles HL et BC. Le premier reçoit l'adresse de départ de la recherche (16510), le second, qui servira de compteur, devra contenir le nombre d'octets maxi sur lequel portera la recherche. Pour cela il suffit de soustraire de l'adresse de fin de recherche (celle du FA) l'adresse de début de recherche. Cette opération se fait grâce à l'instruction SBC HL.BC, son résultat étant rangé dans HL. Mais il faut toutefois prendre garde à l'indicateur de report (Carry) qui est systématiquement pris en considération dans cette opération. Afin d'éliminer tout risque d'erreur on le positionne préalablement à zéro grâce à l'instruction XOR A. C'est ici la seule utilité de cette instruction qui, en réalité, effectue sur son opérande un OU exclusif logique. Pour des raisons inhérentes à la construction du programme les valeurs de HL et BC sont sauvegardées pour être immédiatement après sorties de la pile par les instructions POP BC, POP HL qui ouvrent la boucle L0 et permutent les valeurs des deux registres. Nous verrons pourquoi plus précisément en abordant le deuxième module de traitement : l'identification de la chaîne d'octets. CPIR en est l'instruction centrale. C'est une instruction extrêmement élaborée dont le fonctionnement mérite d'être décrit en détail. CPIR exécute une comparaison du contenu de l'accumulateur avec l'octet pointé par HL; mais à la différence d'une *simple instruction CP elle* effectue en cas de résultat négatif une opération d'incrémentation sur le registre HL et simultanément une décrémentation du registre BC; après quoi le cycle redémarre jusqu'à ce que la recherche aboutisse (comparaison positive) ou que BC soit égal à zéro. BC joue donc le rôle de compteur d'octets, tandis qu'HL pointe successivement, par incrémentation automatique, sur toute la zone à tester. Vous comprenez maintenant pourquoi nous avons donné à HL l'adresse 16510 et à BC le nombre d'octets du listing !
Il vous suffit d'exécuter cette routine, immédiatement après le SAVE :
10 SAVE "PROGRAMME"
20 RAND USR x (x peut prendre une valeur quelconque puisque la routine est entièrement relogeable)

...et vous étonnerez tous les ceusses qui croient qu'on ne peut exister sans avoir d'origine.

Bernard GUYOT

```
REM * DISPARITION DE SAVE
REM * LIMITE DE LA RECHERCHE
    LD HL.(16396)      F.A.
    LD BC.16510        début listing
    PUSH BC
    XOR A
    SBC HL.BC
    PUSH HL
:L0 POP BC             permute
    POP HL             HL avec BC

REM * IDENTIFICATION DE SAVE
    LD A.248           SAVE
    CPIR
    JP NZ.963          Retour ROM
    PUSH HL
    PUSH BC
    LD A.(HL)
    CP 11              Retour ROM
    JR NZ.L0
    INC HL
    LD A.(HL)
    CP 141             $
    JR NZ.L0
    INC HL
    LD A.(HL)
    CP 141             $
    JR NZ.L0
    POP BC             restaure la
    POP HL             Pile

REM * REMPLACE PAR REM
    DEC HL
    LD (HL).234
    LD A.0
    LD B.3
:L4 INC HL
    LD (HL).A
    DJNZ.L4

REM * RECHERCHE DERNIER "
    LD A.11
:L5 INC HL
    LD D.(HL)
    CP D
    JR NZ.L5

REM * INVERSE DERNIER CARACTERE
    DEC HL
    LD A.(HL)
    SUB 128
    LD (HL).A
    INC HL
    LD (HL).0

    RET
```

Formation à l'assembleur

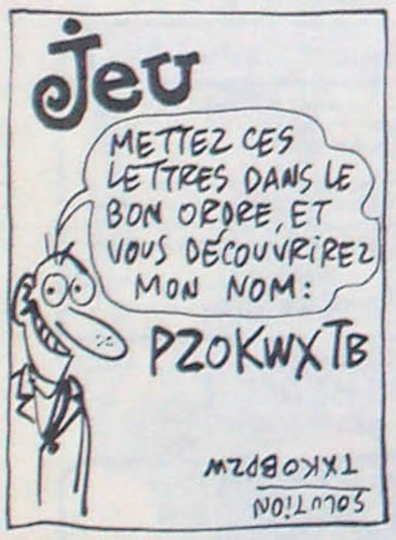
COURS D'ASSEMBLEUR

Vous êtes grâce à l'HHHHEBDO devenus des quasi-cracks de l'assembleur. Mais ne partez pas ! Les bonnes surprises et les bons plans vont se multiplier dans les prochains cours, vous transportant jusqu'au nirvana des programmeurs. Comme toujours le cours théorique branchera les fans du tordu alors que les bidouilleurs se brancheront directement sur leur cours rien qu'à eux !

Vous avez déjà eu droit aux discours suivants dans votre page chérie par-dessus tout :
ZX 81 → 55 56 61 66 71 76 81 86 91 94 100
ORIC → 57 62 67 72 77 82 87 92 94 101
APPLE → 58 63 68 73 78 83 88 93 94 102
THOMSON → 59 64 69 74 79 84 89 94 98 103
COMMODORE → 60 65 70 75 85 90 94 99 104

Lors de nos précédentes rencontres, nous avons découvert avec stupéfaction et surprise les différences fondamentales existant entre les langages interprétés et les langages compilés. Cette (relativement) brève étude comparative nous a amené à distinguer le côté pédagogique des premiers par rapport à l'aspect de puissance des seconds. Il est certain qu'un langage interprété ouvre les portes de l'informatique aux néophytes alors que les langages compilés s'adressent plus directement aux développeurs et programmeurs confirmés.

Notre étude nous a, jusqu'à présent, conduit à nous préocupper uniquement de programmation linéaire, dans la mesure où l'assembleur permet plus facilement ce type d'écriture. Malgré tout, certains professionnels de la programmation arrivent à écrire des programmes en langage machine simulant le fonctionnement de langages récursifs. Cette prouesse ne sera malheureusement pas à la portée du premier venu, aussi tous ceux que la programmation récursive tente devront plutôt se pencher sur l'étude d'un langage plus approprié, tel le Pascal, le C ou Ada. Je sens que vous commencez à patauger dans mes explications, aussi nous allons voir ce que sous-entend le terme de programmation récursive.



Récursivité

Si vous vous jetez avidement sur votre dictionnaire favori, vous découvrirez sous ce mot une référence au terme **récurrent**. Pour ce nouveau mot, vous lirez une définition approchant de ceci : *série récurrente*, série dont chaque terme est fonction des termes immédiatement précédents. Le processus récursif, pour sa part, stipule : qui peut être répété un nombre indéfini de fois par l'application de la même règle.

Vous vous sentez mieux, vous voyez clairement ce que permet un raisonnement récursif ? Non ? Diable ! Vous avez peut-être de vagues souvenirs d'anciens cours de maths au cours desquels votre professeur se servait de démonstrations par récurrence pour arriver à ses fins. Encore une fois nous rencontrons ce terme, mais il me paraît plus judicieux de traiter le problème par le biais d'exemples simples que par un exposé théorique particulièrement obscur et de plus abscons.

Revenons à un niveau mathématique élémentaire et souvenons-nous de ce qu'est une factorielle. Je vous rappelle à toutes fins utiles la notation généralement usitée pour cette représentation mathématique : a ! (se lit factorielle de a). A quoi est égale cette fameuse factorielle ? Regardez l'équation suivante :

a ! = a x (a-1) x (a-2) x...x 3 x 2 x 1

Vous désirez écrire un programme qui résolve ce simple calcul. Vous pouvez bien entendu effectuer ce calcul à grands renforts de logarithmes, exponentielles et autres fonctions trigonométriques, de manière à obtenir un résultat approximé. Regardez par exemple la formulation que nous propose Bruno de la Boisserie en deux lignes :

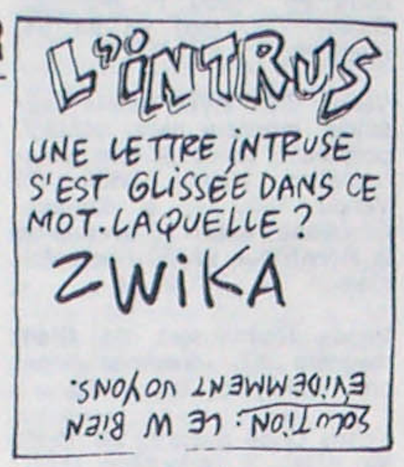
Factorielle en Basic

```
1 DEFFNL(X)=LOG(X)/LOG
  (10):INPUTN:B=N*(F
  NL(N)-FNL(EXP(1)))+
  FNL(8*ATN(1)*N)/2
2 E=INT(B):D=10^(B-E):
  F=(1/12+(1/288-1/3
  73/N)/N)/N:A!=D+D*F:
  PRINT"N!="A!"E+"E
```

Si je ne vous avais pas donné ce listing vous auriez sans doute éprouvé quelques difficultés à programmer ce calcul. Malheureusement cette méthode ne permettra jamais d'avoir un résultat exact à coup sûr : en effet, à partir du moment où l'on utilise les fonctions trigonométriques ou logarithmiques, on ne peut pas représenter en mémoire l'ensemble des chiffres significatifs, aussi perd-on de l'information et, perte plus importante, la justesse du résultat en pâtit. Mais rassurez-vous : il existe une formulation récurrente pour obtenir la factorielle d'un nombre qui elle donnera des résultats exacts, dans la mesure où elle ne fait rentrer en ligne de compte que la muliplication. Nous allons la découvrir maintenant grâce à la série d'équations suivante :

a ! = a x ((a-1) !)
a ! = a x (a-1) x ((a-2) !)
a ! = a x (a-1) x (a-2) x ((a-3) !)
.....
a ! = a x (a-1) x (a-2) x (a-3) x...x ((a-(a+ 1)) !)

Vous voyez bien, dans cet exemple, que la factorielle d'un nombre est égale à ce nombre multiplié par la factorielle du nombre moins 1. Nous nous trouvons donc en face d'une série récurrente (une factorielle est fonction d'une autre factorielle et ainsi de suite).
La principale caractéristique d'un programme récursif réside dans la possibilité, pour un sous-programme, de s'appeler lui-même. A chaque fois qu'une routine s'appelle elle-même, un empilement des données présentes en mémoire lors de l'appel s'effectue dans une pile. En retour, ces données seront dépilées dès que le micro sera revenu à ce stade du calcul.



Nous allons voir maintenant comment nous exploiterons cette capacité d'auto-appel des routines dans le cas qui nous préoccupe actuellement soit le calcul d'une factorielle. Nous allons écrire le programme sous la forme d'une routine. Celle-ci ne nécessite pas de calculs compliqués comme vous allez le voir.

Routine FAC(n)

Si A= 0 Alors 1
Sinon AxFAC(A-1)

Fin routine

Bien sûr, si vous programmiez cette routine en Pascal, elle n'aurait pas exactement cette allure, mais elle est déjà fonctionnelle sur le papier. Admettons que vous vouliez calculer la factorielle de 10. Vous appelez la routine par FAC(10). Le test s'effectue et comme A est différent de 0, la routine effectue le calcul de AxFAC(9). Nous nous retrouvons devant un appel de FAC. Ce nouvel appel conduit l'ordinateur à s'intéresser à FAC(9), où 9 est différent de 0, ce qui aboutit au calcul de 9xFAC(8). Les appels se poursuivent jusqu'à A= 0. A quoi arrivons-nous à ce moment ? Nous allons essayer de le résumer dans le tableau ci-dessous.

APPEL CALCUL

1	10xFAC(9)
2	9xFAC(8)
3	8xFAC(7)
4	7xFAC(6)
5	6xFAC(5)
6	5xFAC(4)
7	4xFAC(3)
8	3xFAC(2)
9	2xFAC(1)
10	1xFAC(0)
11	1

Dans cette succession d'appels, l'ordinateur mémorise juste le nombre d'appels effectués. Dès lors que le test donne un résultat ne mettant plus en cause une routine, c'est qu'il est arrivé en fin de calcul. Ainsi, dans notre cas, lors du onzième appel le test A= 0 se révèle exact, donc FAC(0) donne comme résultat 1. A partir de cet instant, le programme va **remonter** les appels successifs (n'oubliez pas qu'il a stocké le nombre d'appels effectués). Cette remontée se passera de la manière suivante.

RETOUR CALCUL

1	1x1
2	2x(1x1)
3	3x(2x1)
4	4x(3x2)
5	5x(4x6)
6	6x(5x24)
7	7x(6x120)
8	8x(7x720)
9	9x(8x5040)
10	10x(9x40320)

Lors de la dernière remontée, on obtient bien le résultat correct de 10 !, soit 3628800. Vous voyez que dans cette méthode de calcul (récursive, je vous le rappelle) seule la multiplication intervient ce qui limite les calculs à la capacité intrinsèque de votre ordinateur. Nous verrons sur un autre exemple à quel point la programmation récursive peut se montrer supérieure à la programmation linéaire.

L'ASSEMBLEUR PRATIQUE

Langage machine... Sur ZX 81

C'est un truisme de dire de la rentrée qu'elle est un moment difficile, mais quand même ! Permettez ! J'arrive à l'instant des mers chaudes, de contrées lointaines où nul micro jamais n'a prétendu dicter son inhumaine loi au quêteur de soleil, à l'amateur de plaisirs simples et de sensations voluptueuses, à celui qui est libre enfin de rêver son quotidien... Alors s'il vous plait molo la reprise ! Sans doute l'avion vient d'atterrir mais moi, moins sophistiqué que lui, j'ai des paliers de décompression à respecter impérativement sous peine de déprime du genre : c'est gris, ça pue et ça m'emmerde ! Vous ne voudriez pas me voir sombrer dans une telle vulgarité n'est-ce pas ?

Bon c'est vrai que l'année dernière sur conseil de mon psychiatre de famille j'avais repris le collier vraiment très très progressivement... Tiens quelle coïncidence pratiquement jusqu'aux congés de cette année. L'étalement des vacances c'est bien, mais l'étalement de la rentrée ça a du bon aussi croyez-moi.

Pourtant combien de flippés, de retour dans leur rassurant cloaque, avant même de poser les valises, la femme et les enfants pour se rouler tranquille un joint là où passe l'unique rayon de soleil de leur appart, se ruent sur courrier, dossiers, travaux en retard, branchent fébrilement leur micro, le gavent déjà de disquettes, le tout montre en main !

Ca vous ennuie pas j'espère que j'prenne mon temps...

Au fait mon assembleur ? Dans quel tiroir poussiéreux l'avais-je abandonné, sous quelle pile de linge douteux se cache-t-il ? Ah ! Une cassette ! C'est peut-être mon logiciel chéri qui dort là jusqu'à ce que son prince charmant etc... Mais non ! En lieu et place des bits désirés voilà que le magnéto me renvoie la dernière rengaine de Gainsbourg. Il est vrai qu'elle plaide elle aussi en faveur de l'assemblage, un assemblage tout aussi excitant, qu'en pensez-vous ?

Bon je vois que la clientèle de l'HHHebdo a bien changé en mon absence et qu'il faut maintenant faire dans le sérieux. Je m'en tiendrai donc désormais aux deux passes de mon assembleur abstrait, désincarné : le ZX AS.

Juste le temps de boucher toutes les issues par lesquelles le petit soleil parisien pourrait encore m'appeler, de faire taire la douce mélopée vaguement exotique qui berce encore mes souvenirs et j'exhume de mes dossiers, pour votre plus grande jouissance, un listing en langage symbolique d'assemblage qui porte pompeusement et en gros caractères le titre suivant : inversion vidéo en scrolling.

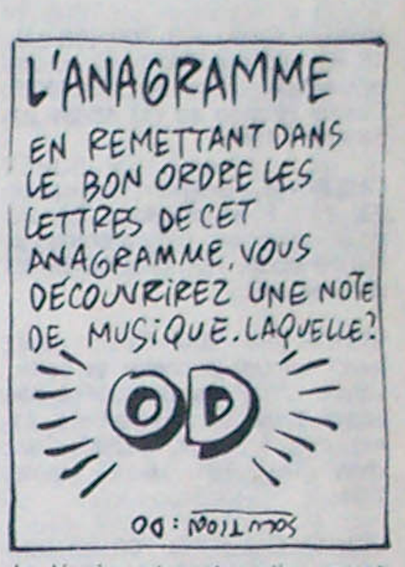
Ca vous dit quelque chose ? Non ? C'est que vous êtes comme moi parti trop longtemps en vacances (vous avez raison), ou bien que vous n'avez pas lu l'HHHebdo depuis le début (vous avez tort). Et si ces termes barbares vous sont tout à fait familiers, attention, peut-être en êtes-vous devenu un à votre insu !

Il me souvient lointainement que dans une existence antérieure -ou était-ce un numéro anté-rieur ? nous avions réalisé un système d'inversion vidéo logiciel, puis à côté, indépendemment, une routine effectuant un scrolling latéral de l'ensemble de la page-écran. Je vous propose aujourd'hui de combiner ces deux logiciels dans un seul et même programme dont l'écriture devra être particulièrement soignée, concise et structurée, page pédago oblige. Ce petit exercice fort simple, idéal pour un redémarrage en douceur, devra aboutir sur l'écran de votre ZX à un résultat élégant : l'inversion vidéo progressive de la page-écran par un balayage latéral de gauche à droite. Si cette routine est exécutée plusieurs fois consécutivement elle offre à nos regards la rotation d'un masque d'inversion vidéo au format de la page-écran. Spectaculaire non ?

Le système de balayage en scrolling de l'écran tient compte de la représentation quasi-linéaire, en mémoire, du fichier d'affichage, l'addition du facteur 33 (32 colonnes + un octet codé 118) permettant de remplir une colonne par le haut avant de passer à la suivante immédiatement à sa droite.

Le travail le plus délicat réside toujours dans l'emploi de la pile. La complexité de sa mise en oeuvre s'accroît en proportion de sa profondeur d'action, ou si vous préférez du nombre d'éléments empilés. Si nous laissons de côté tout ce qui permet la répétition de notre routine, nous observons qu'il ne saurait y avoir d'utilisation de la pile plus simple que dans ce petit exercice de "rentrée". Un seul niveau de profondeur ! Rassurant non ? Pourquoi la pile ? Simplement parce qu'aucun des registres BC, DE, HL ne sera disponible. Voyez : le registre B est utilisé comme compteur de lignes couplé avec l'instruction DJNZ qui

```
REM * INVERSION VIDEO EN SCROLLING

REM * INITIALISATIONS
    LD C.32          nb colonnes
    LD HL.(16396)    F.A.
    INC HL           saute le 118
:L1 PUSH HL
    LD B.24          nb lignes
    LD DE.33         facteur chgt lignes

REM * INVERSION 1 COLONNE
:L2 LD A.(HL)
    ADD A.128
    LD (HL).A
    ADD HL.DE
    DJNZ.L2          chgt ligne
                     me'me col.

REM * TEMPORISATION
    LD D.3
:L3 LD E.255
:L4 DEC E
    JR NZ.L4
    DEC D
    JR NZ.L3

REM * COLONNE SUIVANTE
    POP HL           case départ
    DEC C            décompte col.
    RET Z
    INC HL
    JR L1



    LD E.10
:L5 PUSH DE

PROGRAMME INITIAL

REM * COLONNE SUIVANTE
    POP HL
    DEC C
    JR Z.L6
    INC HL
    JR L1
:L6 POP DE
    DEC E            décompte tours
    RET Z
    JR L5
```



le décrémente automatiquement. Le registre C contient quant à lui le nombre de colonnes sur lequel porte le travail (pour tout l'écran 32). Et HL comme d'habitude sert de pointeur d'adresse. Il pointera successivement sur les adresses respectives de chacun des octets à inverser.

Vous vous étonnerez sans doute de voir le registre double DE sollicité pour supporter une valeur qui pourrait tenir sans problème sur un seul octet. Cette valeur (33) devra être ajoutée à chaque tour à l'adresse de pointage initial (tête de colonne) contenue dans HL, ceci pour pointer dans la case immédiatement au-dessous. C'est précisément en raison de cette addition, effectuée très fréquemment par le processeur, qu'on utilise le registre DE; on peut en effet ajouter directement à HL la valeur contenue dans DE grâce à l'instruction ADD HL.DE. Il n'existe pas d'instruction permettant d'ajouter à HL la valeur d'une constante ou d'un registre simple; par contre HL est la seule des trois paires de registres ordinaires du Z80 à pouvoir accueillir le résultat d'une addition avec un autre registre double, ce qui fait parfaitement notre affaire ici.

Examinons le fonctionnement dynamique du programme et plus particulièrement le travail de la pile.
Le module d'initialisation ajuste les valeurs de départ des compteurs de colonnes, lignes, et place dans DE le facteur constant 33 qui permettra de travailler par colonnes. Vous remarquez qu'on sauve immédiatement l'adresse de la tête de colonne; il nous suffira plus tard, lorsque toute la colonne aura été inversée, de restaurer cette adresse, de l'incrémentrer (+ 1), pour pointer sur la tête de colonne suivante. Cette opération a lieu au niveau du module final; POP HL précédent dans tous les cas le retour au Basic (RET Z), la pile retrouvera toujours son niveau initial au sortir de la routine. Essentiel !

Il est intéressant de greffer sur cette routine un système permettant son exécution automatique plusieurs fois de suite. Il suffira pour cela de prendre en tenaille la totalité de notre programme, en amont par l'initialisation d'un compteur d'exécutions, en aval par la décrémentation de celui-ci avant retour conditionnel au basic lorsque tout sera accompli. Pour cela le plus commode est de rajouter un niveau supplémentaire à la pile.

L'initialisation du registre E autorise ici 10 exécutions successives.

Bon maintenant je rouvre les persiennes, troque ma cassette d'assembleur pour celle dont la douce mélopée, vaguement exotique, berce encore mes souvenirs...

Formation à l'assembleur

COURS D'ASSEMBLEUR

Vous êtes grâce à l'HHHHEBDO devenus des quasi-cracks de l'assembleur. Mais ne partez pas ! Les bonnes surprises et les bons plans vont se multiplier dans les prochains cours, vous transportant jusqu'au nirvana des programmeurs. Comme toujours le cours théorique branchera les fans du tordu alors que les bidouilleurs se brancheront directement sur leur cours rien qu'à eux !

Vous avez déjà eu droit aux discours suivants dans votre page chérie par-dessus tout: ZX 81 → 55 56 61 66 71 76 81 86 91 94 100 105
ORIC → 57 62 67 72 77 82 87 92 94 101 106
APPLE → 58 63 68 73 78 83 88 93 94 102 107
THOMSON → 59 64 69 74 79 84 89 94 103 108
COMMODORE → 60 65 70 75 80 85 90 94 104

Ce cours vous surprendra par bien des côtés ! Tout d'abord, il me faut vous annoncer la fin prochaine de ce discours théorique qui vous a tous tenus de longues semaines en haleine. En effet, il me semble avoir exposé l'ensemble des connaissances essentielles et utiles pour vous. Dorénavant, vous pourrez découvrir prochainement dans les colonnes de la page pédagogique trois nouveaux cours pratiques pour trois machines, jusqu'à maintenant oubliées de l'HHHHebdo. Je vous fais la surprise ? Non, je sens que vous allez encore m'envoyer des tonnes de courrier pour me demander le pourquoi et le comment de ces nouveaux micros. En fait vous allez découvrir les arcanes secrets du Spectrum, du MSX et de... l'Amstrad !

Déjà, je pressens que vous vous sentez nettement mieux, maintenant que vous savez cela. Alors tous ceux qui s'impatientaient à l'idée que leur HHHHebdo unique et préféré ignorait tout de ces machines se rassurent : l'impossible va être fait (pour les miracles comptez deux jours d'attente minimum).

Le deuxième point que je voulais traiter avec vous a été provoqué, à l'origine, par une de mes réflexions au cours du cours (sic) concernant la programmation récursive. Je vous avais humblement avoué ne pas m'être attardé sur la programmation, en Basic, d'un algorithme de reconnaissance de palindrome. Comme je m'y attendais un peu, vous avez été nombreux à m'envoyer votre solution à cet épineux problème.

Comme je ne suis pas rancunier (malgré l'agressivité du courrier de certains des lecteurs) je vous livre ici le fruit des réflexions de ceux que j'estime les mieux pensés. Bien sûr, il ne m'est pas possible d'exposer en détail le cheminement de chacun, aussi vais-je tenter de réaliser une synthèse de toutes ces réalisations.

Guy Beteille formule une remarque particulièrement judicieuse sur la différence fondamentale de raisonnement entre la pensée humaine et l'activité mécanique de l'ordinateur : lorsque vous lisez une phrase et que vous cherchez à déterminer s'il s'agit d'un palindrome, vous supprimez automatiquement les espaces, contrairement au micro qui les considère comme des caractères à part entière.

L'ensemble de vos programmes débute donc par la compression de la chaîne de caractères entrée par l'utilisateur en une suite de lettres privée d'espaces. Cette première étape sera fondamentale. Sans celle-ci, aucune vérification ne pourra avoir lieu correctement.

La seconde phase de votre algorithme est elle aussi basée sur un principe simple, mais efficace. Il s'agit en effet de construire, à partir de la phrase entrée au clavier, une image miroir qui servira à la vérification proprement dite. Vous avez tous utilisé un principe fort similaire. Seuls les associés Nalet et Barthes ont songé à une ruse d'enfer pour éviter d'avoir à créer cette phrase miroir.

La troisième et dernière étape se résume à effectuer la vérification elle-même. Elle consiste à comparer caractère par caractère la chaîne et son image miroir. Si les deux chaînes sont identiques, le palindrome est reconnu, sinon le programme annonce que la phrase ou le mot entré par l'utilisateur n'est pas un palindrome. Cette solution de vérification est celle exploitée par Philippe Sanchez (eh ! Philippe, tu n'as pas encore pigé l'orthographe d'algorithme !), Michel Goux et Guy Beteille.

La solution proposée par nos spécialistes de l'emballage (ils bossent chez Tetra Pak, vous savez le lait longue conservation) Nalet et Barthes se résume à bidouiller avec les fonctions de traitement de chaîne offertes par certains basics (en particulier celui de Microsoft). Il suffit en effet d'utiliser avec subtilité les fonctions MID$, LEFT$ et RIGHT$. Je vous donne les deux propositions dans les listings qui suivent. A vous de choisir celle que vous préférez !

LISTING 1

```
5  CLEAR
10  HOME : PRINT "ENTRE TA PHRASE": PRINT
    "JE VERIFIE SI C'EST UN PALINDROME "
20  INPUT A$:A =  LEN (A$)
30  FOR I = 1 TO A
40 B$ =  MID$ (A$,I,1): IF B$ = " " THEN  GOTO
    60
50 C$ = C$ + B$
60  NEXT I
70 C =  LEN (C$)
80  FOR I = C TO 1 STEP  - 1:B$ =  MID$ (C
    $,I,1):D$ = D$ + B$: NEXT I
90  IF C$ = D$ THEN  PRINT "T'AS GAGNE, C'
    EST UN PALINDROME": GOTO 110
100  PRINT "HOUSSU! CA PAS ETRE PALINDROME
    "
110  END
```

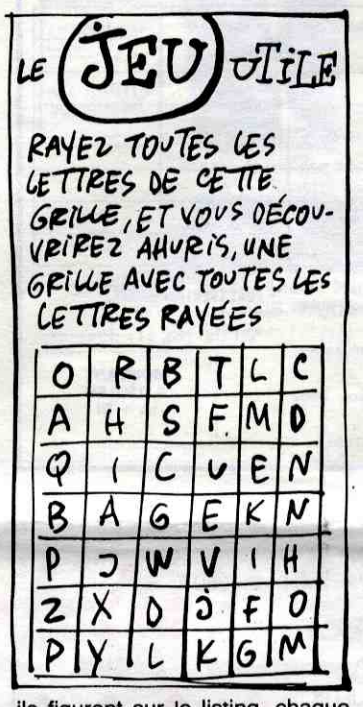
LISTING 2

```
5  CLEAR
10  HOME : PRINT "ENTRE TA PHRASE": PRINT
    "JE VERIFIE SI C'EST UN PALINDROME "
20  INPUT A$
30  IF  LEN (A$) < 2 THEN  PRINT "DE QUI S
    E MOQUE-T-ON?": GOTO 20
40  FOR I = 1 TO  LEN (A$): IF  MID$ (A$,I
    ,1) = " " THEN A$ =  LEFT$ (A$,(I - 1)
    ) +  MID$ (A$,(I + 1))
50  NEXT I
60  IF  ASC ( RIGHT$ (A$,1)) =  ASC ( LEFT$
    (A$,1)) GOTO 80
70  PRINT "RATE!!! CE N'EST PAS UN PALINDR
    OME": GOTO 20
80 A$ =  MID$ (A$,2,( LEN (A$) - 2))
90  IF  LEN (A$) <  = 1 THEN  PRINT "BRAVO
    !!! VOUS EN AVEZ COINCE UN!": GOTO 20
100  GOTO 60
110  END
```

L'ASSEMBLEUR PRATIQUE

Langage machine... Sur ZX 81

Les actuels possesseurs de ZX81 sont décidément des gens hors du commun, des ascètes de la micro, des mystiques, adorateurs du Rien informatique, tenus de pratiquer le bit-à-bit (technique de réanimation à partir du langage machine) pour sauver de la médiocrité et de la désuétude leurs pauvres programmes en basic Sinclair. A ceux-là je tire mon chapeau d'autant plus bas que j'en suis, et que je n'en suis pas à une vanité près. Mis à part l'homme de Néanderthal -un de mes voisins de palier- qui pourrait encore aujourd'hui se contenter d'un ordinateur sans clavier mécanique, sans moniteur couleur, sans HRG, sans allume-cigare intégré et surtout sans instruction READ-DATA-RESTORE ! Après avoir été le premier micro familial ne reste-t-il plus au ZX81 que la gloire d'être le premier micro-ordinateur jetable ? Et le nom tant respecté de Lord Sinclair va-t-il bientôt faire son entrée au panthéon des Bic et autres Kleenex ? Epargnons-lui cette honte mes frères et prolongeons artificiellement, une fois encore, la vie de ce malheureux ZX. Non ! Ne débranchons pas tout de suite la machine !
Attaquons-nous plutôt au manque fondamental du ZX, à cette absence, si cruellement ressentie par vous tous, d'instructions READ-DATA-RESTORE. Puisque vos moyens financiers, chers ZXistes, ne vous ont jamais permis de courtiser d'autres ordinatrices, je commencerai par faire un bref rappel de ce qu'est cette noble fonction, n'hésitant pas à vous dévoiler l'une de mes plus intimes expériences. C'est une adorable petite anglaise qui la première fois a chuchoté à mon oreille les doux mots de READ et de DATA. Comme je faisais semblant de ne pas comprendre elle insista jusqu'à m'en donner la traduction francaise. Je rougissais comme une alim surchauffée lorsqu'elle me dit avec un air mutin :
- "Les DATAS représentent des données qui sont inscrites une fois pour toute dans le corps d'un programme basic...
- Dans le corps... balbutiais-je troublé.
- Oui, elles figurent sur le listing même sous la forme d'une succession de valeurs numériques ou de chaînes de caractères. Ces données sont inscrites sur une ou plusieurs lignes du programme et séparées par des virgules comme par exemple : DATA Lundi,Mardi,Mercredi". Et elle ajouta avec un accent coquin : "tous les jours de la semaine si tu veux". Et comment que je voulais...
"Le programme utilisera ces données en les appelant au moment opportun par l'instruction READ. Le DATA appelé correspond à l'introduction manuelle d'une donnée par INPUT". Je n'osais lui avouer que je préférais l'introduction à la main avec les INPUT et la laissais continuer : "Les Datas sont lus séquentiellement dans l'ordre où



ils figurent sur le listing, chaque READ appelant le DATA suivant. Lorsque tous les DATAS ont été lus, une instruction RESTORE permet de réinitialiser le pointeur le cas échéant". J'imaginais déjà la douce et impérieuse nécessité qu'il y aurait bientôt de réinitialiser mon pointeur... De fait nous passâmes peu de temps après aux exercices pratiques.

C'est ce que je vous propose aujourd'hui avec ce programme simulant les fonctionnalités d'un READ-DATA-RESTORE; le cahier-décharge est le suivant :
Les DATAS devront être inscrits sur une ou plusieurs lignes de REM distribuées à n'importe quel endroit du listing. Un point fera office de séparateur entre les datas. Le pointeur sera réinitialisé automatiquement après la lecture du dernier data et à n'importe quel moment de l'exécution du programme. Celui-ci n'étant qu'une simple simulation, il se contentera d'afficher à l'écran le data à l'emplacement pointé par DF-CC c'est-à-dire à l'adresse de la position d'écriture PRINT dans le fichier d'affichage. Et maintenant RUN...

Commençons par l'organe essentiel : le pointeur. Il est constitué par un dispositif de balayage de la RAM, plus précisément de l'espace réservé au listing basic. On sait que cette zone a pour limite inférieure l'adresse invariable du début de listing (16510) et pour limite supérieure le départ du fichier d'affichage dont l'adresse fluctuante réside toujours en (16396). La différence de ces deux bornes (limites CPIR) définit l'espace sur lequel porte la recherche de la chaîne discontinue des DATAS, ceux-ci devant toujours se présenter sous la forme :
REM DATA.LUNDI.MARDI.MERCREDI.

REM DATA.JEUDI.VENDREDI.SAMEDI.

REM DATA.DIMANCHE
Il nous faudra détecter la succession des caractères REM,D,A,T (laissons tomber le dernier A voulez-vous) pour ensuite lire entre les points nos différents datas. J'ai placé à l'adresse 16417 le compteur de datas qui mémorise le numéro du prochain data à lire. Il s'agit d'un octet disponible dans la zone RAM des variables système. On peut facilement le réinitialiser à tout moment par une simple instruction POKE 16417,x.
Le comptage des datas s'effectue de la façon suivante : l'octet 16417 est incrémenté après chaque lecture. Le registre E prélève cette valeur et effectue à chaque appel de la routine le décompte de sa valeur (DEC E) jusqu'à atteindre le data correspondant à la valeur du compteur au moment de l'appel. Exemple : On veut le deuxième data (mardi). (16417) contient donc 2. Le pointeur va successivement balayer le premier data (E= E-1= 1) puis le second (E= E-1= 0) sur lequel il s'arrêtera (CP 0). Juste avant le module d'écriture on prend soin d'incrémenter l'octet 16417 afin que le compteur prenne sa nouvelle valeur (2+1) et pointe ainsi sur le mercredi. Si en raison de vos convictions religieuses vous souhaitiez passer directement au Dimanche, il vous suffirait de faire POKE 16417,7.

Le module d'écriture s'appuie sur deux pointeurs (encore !) : le registre double HL qui contient successivement l'adresse de chacun des caractères composant le data à afficher et le registre double DE qui tire sa valeur de celle contenue à l'adresse 16398 à savoir la variable système DF-CC, pointeur elle-même de l'instruction basic PRINT (Ouf !). DF-CC est remise à jour après l'affichage d'un caractère par :L3 LD(16398).DE. Le branchement sur le module RESTORE s'effectue à partir d'un test sur le caractère 118 (fin de ligne), le dernier data de la liste -qui pourra en contenir jusqu'à 255- ne devant pas, paradoxalement, être suivi d'un "." final.
En ce qui concerne les instructions utilisées vous reconnaîtrez au passage le CPIR, puissante instruction permettant le balayage d'une zone mémoire de taille BC et d'adresse de départ HL, la reconnaissance d'un caractère s'effectuant par le truchement de l'accumulateur. Vous ne reconnaîtrez pas en revanche l'instruction EXX que nous utilisons ici pour la première fois; elle permute les valeurs des registres doubles BC, DE, HL, avec leurs homologues clandestins et sinon inaccessibles que sont BC', DE', HL'. Les deux instructions EXX encadrent ici la simple défausse de la pile (restauration de son niveau initial avant passage à un autre module de traitement).

Cette routine longue de 105 octets a l'avantage d'être entièrement relogeable. Donc mettez-la où vous voulez pourvu que vous y preniez du plaisir.

Bernard Guyot

```
REM * READ DATA RESTORE

REM * COMPTEUR DATAS
    LD A.(16417)
    LD E.A

REM * LIMITES CPIR
    LD HL.(16396)     F.A.
    LD BC.16510       départ listing
    PUSH BC
    XOR A
    SBC HL.BC
    PUSH HL
:L0 POP BC            inverse BC
    POP HL            et HL

REM * DETECTION
:L5 LD A.234          "REM"
    CPIR
    RET NZ
    PUSH HL           pour repartir là
    PUSH BC           où on s'était arrêté
    LD A.(HL)
    CP 41             "D"
    JR NZ.L0
    INC HL
    LD A.(HL)
    CP 38             "A"
    JR NZ.L0
    INC HL
    LD A.(HL)
    CP 57             "T"
    JR NZ.L0
    INC HL            saute "A"
    INC HL            pointe sur "."
    EXX
    POP DE            défausse de
    POP BC            la pile
    EXX               valeurs retrouvées

REM * COMPTAGE DATAS
:L1 LD A.27           "."
    CPIR
    RET NZ
    LD A.(HL)
    CP 118            test fin de ligne
    JR Z.L5           autre ligne DATA?
    DEC E             compteur data
    LD A.E
    CP 0
    JR NZ.L1

REM * ECRITURE
:L4 LD A.(16417)
    INC A
    LD (16546).A
    LD DE.(16398)     DF-CC
:L3 LD (16398).DE     mise à jour DF-CC
    LD A.(HL)         caractère à afficher
    CP 118
    JR Z.L6
    CP 27             "."
    RET Z
    PUSH AF
    LD A.(DE)
    CP 118            Si PRINT pointe
    JR NZ.L2          sur une fin de ligne
    INC DE
:L2 POP AF
    LD (DE).A         écrit sur DF-CC
    INC HL            suivant à lire
    INC DE            suivant à écrire
    JR L3

REM * RESTORE
:L6 LD A.1
    LD (16417).A
    RET
```

Formation à l'assembleur pratique

Langage machine... Sur ZX 81

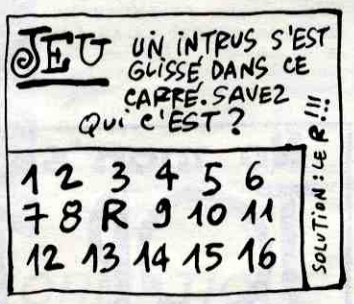
LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110
ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111
AMSTRAD -» 111
APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112
SPECTRUM -» 112
THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113
MSX -» 113
COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109

Les grilles, barreaux et autres herses pourraient-ils devenir des éléments décoratifs, voire même les supports d'une nouvelle créativité ? C'est la question gravissime que je soulevais avec lourde peine lors d'un récent colloque sur l'informatisation des établissements pénitentiaires. Fort heureusement celle-ci trouva un écho favorable et donc totalement dépourvu de sonorités métalliques ou ferrugineuses dans la haute assemblée des détenus - jusqu'à- 65 ans, je veux dire du personnel administratif bien entendu. C'est pourquoi, au hasard d'un séjour à Fleury-la-belle ou à la Bonne Santé, vous pourrez trouver dans la salle de Hard, égarées entre des logiciels classés X, ces deux routines en langage symbolique d'assemblage qui ont pour noms : DOUBLE HERSE HORIZONTALE et DOUBLE HERSE VERTICALE. Rassurez-vous ces herses n'ont de rapports que très lointains avec l'écroûteuse-émotteuse de nos campagnes ou même avec la herse norvégienne dont les dents, je vous le rappelle, sont fixées à des cylindres rotatifs. Pas de rapport également avec l'herpès qui ne se trouve pourtant qu'à sept mots en amont dans le Petit Robert. Il s'agit là, vous le savez, d'une affection cutanée particulièrement dégoûtante qui justifie plus que jamais le célèbre "touche pas à mon herpès" lancé pour la première fois à la face d'un immigré par un leader d'extrême-droite bien connu des services de police. Non ! Ces deux doubles herses dont nous allons maintenant dévoiler la complexité limitent leur influence à l'espace d'un écran cathodique et leurs possibilités à celles du clavier de votre ZX81. Pas trop déçu j'espère ?

A quoi ressemble une double herse horizontale ? Au départ à strictement rien; ensuite à des rayures qui naissent simultanément à gauche et à droite de l'écran, une ligne sur deux et croissent jusqu'à atteindre le côté opposé. A la fin la double herse horizontale ne ressemble plus à rien. Vous constatez donc qu'elle n'existe qu'à travers son dynamisme vivant. C'est je crois cet anthropomorphisme de la Double Herse qui la rend si attachante...



Le programme de Double Herse Verticale aboutit au même résultat mais de haut en bas, et ceci sans obligation de faire reposer le téléviseur sur l'un de ses côtés ce qui, m'a-t-on dit de source autorisée, ramènerait la réception des trois chaînes au niveau de Canal + après 20h 30. Elle sera donc particulièrement appréciée par vous, propriétaires de "grands écrans", gigantesques bahuts inesthétiques et ridicules.

C'est avec votre vivacité d'esprit coutumière, cher lecteur, que vous constaterez la structure symétrique de ces deux programmes, mais comme on est pas tous les jours au mieux de sa forme j'en accuse ici les traits pour la rendre plus évidente encore, en attendant pour vous des jours meilleurs et surtout une Bonne et Heureuse Année 1986. C'est de tout coeur que je vous souhaite d'échapper aux saisies d'huissier et au SIDA afin de poursuivre dans des conditions décentes, au clavier de votre micro, la palpitante aventure de la Vie !...

Pour chacun de ces programmes nous trouvons :
1 - Un module d'initialisation qui met les pointeurs à jour (fichier d'affichage, adresses de départ des processus de "coloration")
2 - Successivement, deux modules de coloration symétriquement inverses qui travaillent alternativement (droite/gauche ou bas/haut) par coloration d'une ligne ou d'une colonne une case sur deux.
3 - Un module de temporisation qui permet d'appréhender le processus spécifique de coloration en "double herse".
4 - Un retour conditionnel au Basic une fois le travail terminé, toujours précédé de l'indispensable restauration du niveau initial de la pile.

```
REM * DOUBLE HERSE HORIZONTALE

REM * INITIALISATIONS
    LD HL.(16396)          F.A.
    INC HL
    PUSH HL
    LD DE.64
    ADD HL.DE
    PUSH HL
    LD C.32
    LD A.(16507)           stock couleur

REM * COLORATION A DROITE
:L0 LD (HL).A              col. 1°case
    LD B.11                compteur
    LD DE.66               1 case/2
:L1 ADD HL.DE
    LD (HL).A
    DJNZ.L1
    POP DE
    DEC DE
    POP HL
    PUSH DE
    PUSH HL

REM * COLORATION A GAUCHE
    LD (HL).A              1°case
    LD B.11
    LD DE.66
:L2 ADD HL.DE
    LD (HL).A
    DJNZ.L2
    POP DE
    INC DE
    POP HL
    PUSH DE
    PUSH HL

REM * TEMPORISATIONS
    LD D.12
:L3 LD E.255
:L4 DEC E
    JR NZ.L4
    DEC D
    JR NZ.L3
    DEC C
    JR NZ.L0
    POP HL                 défausse
    POP DE                 pile
    RET
```

```
REM * DOUBLE HERSE VERTICALE

REM * INITIALISATIONS
    LD HL.(16396)
    INC HL
    PUSH HL
    LD DE.790
    ADD HL.DE
    LD C.24                compteur lignes
    LD A.(16508)

REM * COLORATION BAS
:L0 LD B.16
:L1 LD (HL).A
    DEC HL                 une case...
    DEC HL                 sur deux
    DJNZ.L1
    DEC HL

REM * COLORATION HAUT
    POP DE                 départ FA
    PUSH HL                départ bas
    LD B.16
:L2 LD (DE).A              coloration
    INC DE                 une case...
    INC DE                 sur deux
    DJNZ.L2
    INC DE

    DEC C
    JR Z.L3
    POP HL                 départ haut
    PUSH DE                départ bas

REM * TEMPORISATIONS
    LD D.12
:L4 LD E.255
:L5 DEC E
    JR NZ.L5
    DEC D
    JR NZ.L4
    JR L0

:L3 POP BC                 défausse pile
    RET
```

Examinons de plus près le travail de la pile et la stratégie employée pour la coloration sélective des cases du F.A. Pour la double herse horizontale le registre HL pointe d'abord sur la première case en haut à gauche. C'est notre premier point de départ, tout de suite sauvegardé dans la pile. L'addition du facteur 64 fait ensuite pointer HL sur la dernière case à droite de la deuxième ligne. Nous sauvegardons à nouveau notre seconde adresse de départ et le travail de coloration va commencer à partir de celle-ci avec le module suivant. Le choix du "motif" de notre Herse est stocké à l'adresse libre en RAM 16507, c'est donc là que l'accumulateur ira la prélever. Après coloration de la case départ l'opération doit être répétée 11 fois ((24 lignes/2)-1). L'initialisation du registre B offre alors la possibilité d'utiliser l'instruction DJNZ comme boucle conditionnelle. L'addition du facteur 66 à chaque tour permet de pointer sur l'octet de même colonne mais en sautant une ligne à chaque fois (2X(32col + 1 car. fin de ligne)). Avant de passer la main au module de coloration à gauche on remet à jour le pointeur de coloration à droite en ressortant de la pile son adresse initiale pour la décrémenter -POP DE, DEC DE. On prend soin aussi de sortir le deuxième niveau de pile (adresse de départ à gauche)- POP HL, avant de sauver l'adresse du nouveau départ à droite -PUSH DE, et celle -vous me suivez ? !- du départ à gauche qu'on devra retrouver tout à l'heure -PUSH HL.
Cette succession de POP et de PUSH est un peu difficile a assimiler, c'est pourtant une gymnastique à laquelle vous devrez vous habituer. Notez qu'ici on utilise la pile sur deux niveaux de profondeur seulement. C'est très peu !

Le module de coloration à gauche reprend exactement la structure du module précédent et on retrouve la séquence des POP et PUSH à la différence près d'une incrémentation là où on avait une décrémentation (la progression à partir de la gauche suppose une addition tandis qu'elle nécessite une soustraction à partir de la droite).

Avant d'en finir, comptez une fois encore les PUSH et les POP et vous constaterez que nous sommes redevables de 2 niveaux à la pile du Z80 ce qui implique une défausse de 2 POP avant le RET final.

Le fonctionnement de la pile dans la DOUBLE HERSE VERTICALE a été sensiblement simplifié. N'oubliez pas qu'une écriture plus concise est toujours préférable et, si ce n'est pas trop compliqué, cherchez vous aussi à faire plus simple.

Bernard Guyot

Langage machine... Sur COMMODORE 64

0H24. Une dépêche tombe sur mon télescripteur : *cours attendu de toute urgence !* C'est particulièrement gênant, je rentre à peine d'un dîner aux chandelles avec ma dulcinée, prêt à m'endormir, la tête pleine de souvenirs... Que faire ? Ce cours va-t-il subir les assauts de mon coeur, qui ne demande qu'à s'exprimer (pauvre de vous). STOP ! Changement d'idées obligatoire... Rien de tel qu'un petit listing pour attaquer dans la joie ce nouveau cours.

Routine Joystick 1

```
.,  9000 AD 00 DC LDA $DC00
.,  9003 A2 00    LDX #$00
.,  9005 A0 00    LDY #$00
.,  9007 4A       LSR
.,  9008 B0 01    BCS $900B
.,  900A 88       DEY
.,  900B 4A       LSR
.,  900C B0 01    BCS $900F
.,  900E C8       INY
.,  900F 4A       LSR
.,  9010 B0 01    BCS $9013
.,  9012 CA       DEX
.,  9013 4A       LSR
.,  9014 B0 01    BCS $9017
.,  9016 E8       INX
.,  9017 4A       LSR
.,  9018 60       RTS
```

Ah ! Je me sens nettement mieux. Rentrons dans le vif du sujet : nous allons étudier aujourd'hui les techniques de lecture des ports joystick. Désormais, vous ne trouverez plus d'explications sur les instructions du 6510, mais vous les découvrirez dans le cadre de petits programmes, dans leur milieu naturel en quelque sorte.

La lecture des ports joystick s'effectue au niveau de deux registres en $DC00 (port 2) et en $DC01 (port 1). Les données sont contenues de la manière suivante :

Bit 0 en haut
Bit 1 en bas
Bit 2 à gauche
Bit 3 à droite
Bit 4 bouton de feu

Le programme que vous venez de voir (ou d'entrer dans votre CBM) se décompose ainsi :

9000 l'accumulateur prend la valeur du port 2
9003 X=Y=0
9007 rotation de A vers la droite : le bit 0 passe dans la retenue
9008 test de la retenue. Quand tous les bits sont à 1, le joystick est inactif. Si le joystick n'est pas en haut (bit 0 = 1) on saute en $900B
900A Y est décrémenté. A l'écran, l'origine des axes est en haut à gauche. Quand Y diminue, le curseur monte d'une ligne (ce qui correspond bien à la position du joystick)
900B le bit 1 du registre passe dans C
900C test de C : joystick en bas ?
900E oui : incrémentation de Y
900F le bit 2 du registre passe dans C
9010 test de C : joystick à gauche ?
9012 oui : décrémentation de X
9013 le bit 3 du registre passe dans C
9014 test de C : joystick à droite ?
9016 oui : incrémentation de X
9017 le bit 4 du registre passe dans C
9018 fin de la routine

Les données testées au cours de cette routine se retrouvent, à la fin du traitement, dans les registres X et Y (pour la direction) et dans C (pour le tir).

Nous allons réaliser une application directe de ces caractéristiques : une souris pour le Commodore 64... Avec le joystick bien sûr !

Routine Joystick 2

```
.,  C000 48       PHA
.,  C001 8A       TXA
.,  C002 48       PHA
.,  C003 98       TYA
.,  C004 48       PHA
.,  C005 A5 FE    LDA $FE
.,  C007 C5 CF    CMP $CF
.,  C009 F0 28    BEQ $C033
.,  C00B A5 CF    LDA $CF
.,  C00D 85 FE    STA $FE
.,  C00F AD 00 DC LDA $DC00
.,  C012 A2 00    LDX #$00
.,  C014 4A       LSR
.,  C015 B0 02    BCS $C019
.,  C017 A2 91    LDX #$91
.,  C019 4A       LSR
.,  C01A B0 02    BCS $C01E
.,  C01C A2 11    LDX #$11
.,  C01E 4A       LSR
.,  C01F B0 02    BCS $C023
.,  C021 A2 9D    LDX #$9D
.,  C023 4A       LSR
.,  C024 B0 02    BCS $C028
.,  C026 A2 1D    LDX #$1D
.,  C028 E0 00    CPX #$00
.,  C02A F0 07    BEQ $C033
.,  C02C 8E 77 02 STX $0277
.,  C02F A2 01    LDX #$01
.,  C031 86 C6    STX $C6
.,  C033 68       PLA
.,  C034 A8       TAY
.,  C035 68       PLA
.,  C036 AA       TAX
.,  C037 68       PLA
.,  C038 4C 31 EA JMP $EA31
```

Pour pouvoir lancer ce petit programme, vous aurez besoin de celui-ci :

Routine Joystick 3

```
.,  C100 78       SEI
.,  C101 A9 00    LDA #$00
.,  C103 8D 14 03 STA $0314
.,  C106 A9 C0    LDA #$C0
.,  C108 8D 15 03 STA $0315
.,  C10B 58       CLI
.,  C10C 60       RTS
```

Ces deux routines mettent directement en application le dernier cours que vous avez subi sur les interruptions. Pour l'utiliser, lancez le programme par un J C100 ou par un G C100 (suivant l'assembleur).

C000 sauvegarde de A, X et Y...
C004 ... reportez-vous au cours du numéro 109
C005 routine de temporisation, CF représente la valeur de clignotement du
C00D curseur, FE un octet de référence
C00F lecture du port 2
C012 X = 0
C014 C = bit 0
C015 test de C
C017 X = curseur haut
C019 C = bit 1
C01A test de C
C01C X = curseur bas
C01E C = bit 2
C01F test de C
C021 X = curseur gauche
C023 C = bit 3
C024 test de C
C026 X = curseur droite
C028 si X n'a pas été modifié...
C02A ... alors saut en $C033
C02C buffer clavier ($0277 à $0280) = valeur de X
C02F un caractère dans le...
C031 ... buffer clavier
C033 récupération de A, X et Y...
C038 ... reportez-vous au cours du numéro 109

Après avoir exécuté le programme en $C100, le manche de votre joystick dirige le curseur à l'écran. Bien entendu, cet exercice de style n'a qu'un intérêt limité, mais vous pouvez grâce au premier programme diriger le sprite 0 avec le joystick. Pour ce faire, modifiez la routine 1 de la manière suivante :

Routine Joystick 4

```
.,  9000 AD 00 DC LDA $DC00
.,  9003 AE 00 D0 LDX $D000
.,  9006 AC 01 D0 LDY $D001
.,  9009 4A       LSR
.,  900A B0 01    BCS $900D
.,  900C 88       DEY
.,  900D 4A       LSR
.,  900E B0 01    BCS $9011
.,  9010 C8       INY
.,  9011 4A       LSR
.,  9012 B0 01    BCS $9015
.,  9014 CA       DEX
.,  9015 4A       LSR
.,  9016 B0 01    BCS $9019
.,  9018 E8       INX
.,  9019 4A       LSR
.,  901A B0 04    BCS $9020
.,  901C A9 01    LDA #$01
.,  901E 85 FF    STA $FF
.,  9020 8E 00 D0 STX $D000
.,  9023 8C 01 D0 STY $D001
.,  9026 60       RTS
```

Aux registres X et Y sont affectées les coordonnées d'origine du sprite 0 (en $D000 et $D001) puis, après avoir été modifiées par le joystick, elles seront replacées dans ces deux octets. Ainsi vous déplacerez votre sprite à volonté à l'écran. Le bouton de tir est placé en FF et mis à 1 lorsqu'il est actionné.

Alors, que pensez-vous de l'exploitation de vos connaissances ? C'est facile le langage machine, hein ? Eh bien allez réfléchir un peu sur ce petit problème : modifiez la routine souris (routine 2) de façon à ce que l'appui sur le bouton de feu fasse un RETURN (réponse dans quatre semaines).

Vous vous êtes sans doute rendu compte que les routines machine peuvent être installées n'importe où en mémoire (ou presque). Ainsi dans nos exemples, elles se trouvaient en $9000 ou en $C000. Vous vous demandez maintenant comment est organisée la mémoire, nous allons donc apporter ici une réponse à cette question dramatique.

Je vous rappelle, à tout hasard, qu'il existe deux types de mémoire : les RAM (celles que vous pouvez modifier à volonté) et les ROM (vous ne pouvez que les lire). Votre Commodore propose 64 Ko de RAM et 16 Ko de ROM. Le processeur ne peut gérer que 64 Ko au total, du coup les 16 Ko de ROM sont placés pardessus 16 Ko de RAM, suivant le principe des pages mémoire. Lors de la mise sous tension de votre ordinateur, la mémoire est configurée ainsi.

Schéma de la mémoire

De $0000 à $9FFF se trouve la mémoire utilisateur, là où se placent vos programmes Basic.
De $A000 à $BFFF se loge la ROM qui contient le Basic.
De $C000 à $CFFF se trouve une RAM où vous pourrez implanter vos routines machine.
De $D000 à $DFFF résident les entrées/sorties.
De $E000 à $FFFF loge le KERNAL, soit toutes les routines de gestion des périphériques.

Cette configuration de base donne 44 Ko de RAM et 20 Ko de ROM. Si vous voulez récupérer l'ensemble de la mémoire en RAM, il vous faudra jouer sur le contenu de l'adresse $0001 qui sert d'interrupteur soft pour la mémoire. Le bit correspondant à une page mémoire donnée est à 1 si elle contient de la ROM et à 0 si elle contient de la RAM. Dans l'octet $0001, nous avons trois bits qui servent d'interrupteurs :

Bit 0 : page $A000 à $BFFF
Bit 1 : page $E000 à $FFFF
Bit 2 : page $D000 à $DFFF

Attention : si vous isolez le processeur en transformant la ROM qui gère les entrées/sorties en RAM, vous ne pourrez plus rien en faire (essayez un POKE1,PEEK(1)AND253).

Vous pourrez, grâce à ce superbe octet 1, modifier la ROM Basic (pour le franciser ou pour lui rajouter des instructions par exemple). Cette manipulation se passera en deux phases : recopie de la ROM en RAM par une boucle du style :

FOR I = $A000 TO $BFFF : POKEI, PEEK(I) : NEXT

Dès que cette boucle est terminée, commutez cette page mémoire en RAM grâce au bit 0 de l'octet $0001. Dès lors, vous pourrez magouiller tranquillement la table des mots-clé dans la zone $A09E à $A19B. Bonne chance et réveillez-moi pour le prochain cours.

Sébastien MOUGEY

Formation à l'assembleur pratique

Langage machine... Sur COMMODORE 64

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114
COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114
ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115
AMSTRAD -» 111 115
APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116
SPECTRUM -» 112 116
THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117
MSX -» 113 117

BONJOUR...

Ça c'est de l'intro à la Mourousi comme je les aime. Attaquons d'entrée avec le listing que vous avez dû pondre pendant les vacances de Noël.

Listing Réponse

```
C000 48        PHA
C001 8A        TXA
C002 48        PHA
C003 98        TYA
C004 48        PHA
C005 A5 FE     LDA $FE
C007 C5 CF     CMP $CF
C009 F0 2D     BEQ $C038
C00B A5 CF     LDA $CF
C00D 85 FE     STA $FE
C00F AD 00 DC  LDA $DC00
C012 A2 00     LDX #$00
C014 4A        LSR
C015 B0 02     BCS $C019
C017 A2 91     LDX #$91
C019 4A        LSR
C01A B0 02     BCS $C01E
C01C A2 11     LDX #$11
C01E 4A        LSR
C01F B0 02     BCS $C023
C021 A2 9D     LDX #$9D
C023 4A        LSR
C024 B0 02     BCS $C028
C026 A2 1D     LDX #$1D
C028 4A        LSR
C029 B0 02     BCS $C02D
C02B A2 0D     LDX #$0D
C02D E0 00     CPX #$00
C02F F0 07     BEQ $C038
C031 8E 77 02  STX $0277
C034 A2 01     LDX #$01
C036 86 C6     STX $C6
C038 68        PLA
C039 A8        TAY
C03A 68        PLA
C03B AA        TAX
C03C 68        PLA
C03D 4C 31 EA  JMP $EA31
```

Nos comptes sont maintenant réglés : vous avez ma réponse au dernier exercice du numéro 114. Pour l'exécuter (la peine de mort est-elle vraiment abolie ?) vous aurez à utiliser la routine qui débutait en $C100 (routine joystick 3). Je signale enfin que l'instruction CMP #$00 est inutile, quand l'accumulateur est à 0, le bit Z est automatiquement positionné à 1, donc BEQ et BNE fonctionnent systématiquement dans ce cas. La ligne $C02D est alors superfétatoire, supprimez-la (encore ! vraiment il serait temps de dresser l'échafaud). Le dernier cours traitait principalement du port joystick et le résultat était affiché à l'écran. Horreur ! Je vous ai pris pour des êtres omniscients (ou pas loin) : je ne vous ai donné aucun renseignement sur cet affichage.

P'TIT DEJ'

Nous allons nous mettre sous la dent le circuit vidéo 6567 et ses routines, pour réparer l'oubli du cours précédent. Si vous voulez vous monter un clone de votre Commodore, sachez par avance que tous les circuits de numéro 65xx valent la peau des fesses (180 francs pour un 6526 et votre C64 en compte deux !). Mais seuls des fondus d'électronique pourraient s'intéresser à des détails pareils, ils n'ont qu'à se reporter au Programmer's Reference Guide, page 436.

DÉJEUNER

La mémoire écran de votre micro préféré est divisée en deux parties parfaitement inégales :

★ La mémoire des caractères de $0400 à $07E7

★ La mémoire couleur de $D800 à $DBFF

La première adresse correspond au premier caractère en haut à gauche de l'écran. En $0401 vous trouverez le caractère immédiatement à droite du précédent. Le dernier caractère de la ligne sera mémorisé à l'adresse $0418, le premier de la seconde ligne en $0419 et ainsi de suite jusqu'au millième caractère qui sera stocké en $07E7. La mémoire couleur fonctionne selon le même principe, nul, mais efficace. Je sens que vous commencez à vous endormir dans le doux ronron du ventilateur, à côté du radiateur au fond de votre chambre. Alors au boulot la jeunesse ! Vous avez tous les éléments en main pour réussir à résoudre l'énigme que je vais maintenant vous coller en travers du chemin.

Sujet : écrivez une routine qui change la couleur de tous les caractères à l'écran, en gris par exemple.

Vous avez trouvé une superbe solution de course ? Comparez avec celle que je vous propose maintenant.

Listing 1

```
C000 A9 00     LDA #$00
C002 8D 0D C0  STA $C00D
C005 A9 D8     LDA #$D8
C007 8D 0E C0  STA $C00E
C00A A9 0F     LDA #$0F
C00C 8D 00 D8  STA $D800
C00F EE 0D C0  INC $C00D
C012 AD 0D C0  LDA $C00D
C015 F0 03     BEQ $C01A
C017 4C 0A C0  JMP $C00A
C01A EE 0E C0  INC $C00E
C01D AD 0E C0  LDA $C00E
C020 C9 DC     CMP #$DC
C022 D0 E6     BNE $C00A
C024 60        RTS
```

Commentaire autorisé : vous devez remplir près de mille octets avec la même valeur. Lorsque vous avez plus de 256 octets à adresser, le mode étendu indexé n'est plus utilisable. Il faut donc ruser comme un sioux et se servir d'une petite astuce qui consiste à modifier directement le contenu de l'adresse qui suit votre mnémonique (ou votre instruction). Dans notre exemple c'est le STA $D800 de l'adresse $C00C qui est incrémenté. On modifie en premier lieu le poids faible et, quand il a fait un tour complet et revient à $00, le poids fort. Comme la zone de mémoire à transformer se termine en $DBFF, il suffit d'établir un test d'arrêt qui compare la valeur à $DC00 pour stopper la routine au moment opportun. Si vous voulez réutiliser votre routine, n'oubliez pas de réinitialiser le STA de l'octet $C00C à $D800 (en plaçant $00 en $C00D et $D8 en $C00E).

GOÛTER

Voici maintenant de quoi rigoler avec les routines système du micro de telle façon que l'écran en prenne plein la vue.

★ Pour réinitialiser l'écran, vous pouvez utiliser la routine en ROM d'adresse d'appel $E518.

★ Pour retrouver un fond bleu, le sous-programme à appeler commence en $E5A0.

★ Pour voir votre écran scroller à tout va, c'est en $E8EA qu'il faut aller.

★ Pour afficher un caractère contenu dans A à une position de votre choix, placez le numéro de ligne en X et appelez la routine qui réside à partir de l'adresse $E9F0 puis installez en $D3 le numéro de colonne et effectuez l'affichage par l'appel au sous-programme résidant en $EA13.

★ Vous voulez remonter d'une ligne à l'écran ? Pas de problème vous y arriverez par la routine située en $E8B0. Comme nous ne sommes pas sectaires, il nous arrive aussi de descendre d'une ligne par l'appel au sous-programme d'adresse $E8C2.

★ Maintenant, pour afficher un texte, vous devez placer l'adresse de début de votre texte en $22 et $23, sa longueur en X et enfin effectuer un saut souple et précis en direction de l'adresse $AB25.

★ Pour modifier le jeu de caractères en cours d'utilisation, il suffit de jouer sur le contenu de l'octet $D018. Si vous y placez $15, vous vous retrouvez en caractères majuscules, en y installant $17 ce sont les minuscules qui reviennent en force.

Allez ! Une petite routine pour se détendre et pour comprendre à quoi servent toutes les indications que je viens de vous tartiner au-dessus.

Listing 2

```
C000 A9 17     LDA #$17
C002 8D 18 D0  STA $D018
C005 A9 9E     LDA #$9E
C007 85 22     STA $22
C009 A9 A1     LDA #$A1
C00B 85 23     STA $23
C00D A2 FF     LDX #$FF
C00F 20 25 AB  JSR $AB25
C012 60        RTS
```

DÎNER SURPRISE

Tapez ce superbe listing numéro 3 (puisqu'il est le troisième du cours) et lancez l'exécution grâce au listing 4 subtilement placé à la suite de ce troisième listing.

Listing 3

```
C000 A9 00     LDA #$00
C002 8D 0E C0  STA $C00E
C005 A9 D8     LDA #$D8
C007 8D 0F C0  STA $C00F
C00A AE 12 D0  LDX $D012
C00D 8E 00 DC  STX $DC00
C010 EE 0E C0  INC $C00E
C013 AD 0E C0  LDA $C00E
C016 F0 03     BEQ $C01B
C018 4C 0D C0  JMP $C00D
C01B EE 0F C0  INC $C00F
C01E AD 0F C0  LDA $C00F
C021 C9 DC     CMP #$DC
C023 D0 E8     BNE $C00D
C025 4C 31 EA  JMP $EA31
```

Eh ! N'oubliez pas de rentrer aussi ce listing 4, sinon le 3 ne risque pas de fonctionner !

Listing 4

```
C100 78        SEI
C101 A9 00     LDA #$00
C103 8D 14 03  STA $0314
C106 A9 C0     LDA #$C0
C108 8D 15 03  STA $0315
C10B 58        CLI
C10C 60        RTS
```

Au fait ! J'ai oublié de vous expliquer comment couper l'écran : mettez $0B en $D011 et replacez $1B à cette même adresse pour rétablir l'aspect habituel de votre télé.

SOUPER

Maintenant que vous savez où vous adresser, il ne vous reste plus qu'à faire joujou avec toutes ces adresses en mémoire pour vous concocter des présentations complètement infernales. La prochaine édition du cours concernera encore partiellement votre écran puisque je vous expliquerai en long, en large et en travers comment bidouiller, grâce à la table des adresses écran, les routines qui gèrent l'affichage et même comment en implanter de nouvelles en mémoire.

Bien. Je vous laisse digérer les listings de la semaine en toute tranquillité, à dans quatre semaines.

Sébastien MOUGEY

Langage machine... Sur ZX 81

Vous avez tous utilisé l'instruction basic INKEY$ dans vos programmes, que ce soit pour piloter un vaisseau spatial en *temps réel* ou simplifier la procédure de réponse à une question posée par la machine. Avec INKEY$ inutile de valider une entrée au clavier par le traditionnel NEW LINE, le seul effleurement d'une touche suffit pour que l'ordinateur réagisse aussitôt. Le revers de cette sensibilité exacerbée c'est bien sûr la pauvreté d'information ainsi transmise au micro : un caractère pas plus et qui, sur le ZX, n'est même pas codé en ASCII !

Utilisateurs maniaques du ZX 81 vous n'aurez pas manqué de remarquer les faiblesses de l'instruction INKEY$. Si, en soi, la routine de balayage du clavier est efficace, sa mise en œuvre dans le cadre d'un programme basic l'alourdit considérablement, notamment lorsqu'il s'agit d'identifier, à travers plusieurs lignes de basic, la touche pressée. Concrètement cela se traduit par un appui prolongé sur la touche pour que la détection se produise. Il est également dommageable, dans la plupart des cas, que la touche BREAK reste opératoire sous INKEY$. On interrompt ainsi malencontreusement le déroulement d'un programme qui passait d'une page écran à l'autre, à notre ordre.

Vous allez constater que notre routine d'aujourd'hui pallie à tous ces défauts; mais surtout elle offre l'avantage d'associer à un scanning puissant la possibilité d'afficher un message clignotant et ceci simultanément au balayage du clavier. Ce message apparaît centré sur une fenêtre de 28 caractères au niveau de la 24ème ligne de l'écran, habituellement inaccessible. Il s'inscrit et s'efface à un rythme susceptible d'attirer l'attention des plus distraits jusqu'à ce qu'on ait pressé une quelconque touche, mis à part BREAK qui resterait sans effet. Il est possible, par une simple lecture d'octet (PEEK) d'identifier, si cela est nécessaire, la touche pressée. Fin du fin ! Lorsqu'on quitte la routine, la fenêtre d'affichage du message prend la couleur choisie à l'avance par vos petits soins ! Si vous voyez quelqu'autre perfectionnement à y ajouter, n'hésitez pas, les octets de votre RAM sont là pour ça...

Le texte du message figurera à un endroit quelconque du listing basic sous la forme d'une ligne de REM identifiable par la routine grâce à la séquence REM INKEY$ = x où "x" représente la *couleur* à afficher au sortir de la routine, le texte du message venant immédiatement après.

Examinons le programme : les habitués des cours d'assembleur de l'HHHHebdo reconnaîtront d'emblée l'instruction CPIR dont on ne peut décidément plus se passer lorsqu'on y a goûté. C'est à cette puissante instruction qu'est dévolu le soin de localiser, dans la masse des octets du listing basic, la séquence REM INKEY$ = , une fois déterminées les limites de l'espace sur lequel portera la recherche par Comparaison, Incrémentation, Répétition, la succession reconnue des caractères codés 234, 65, 20 nous donnera dans HL l'adresse de début du message.

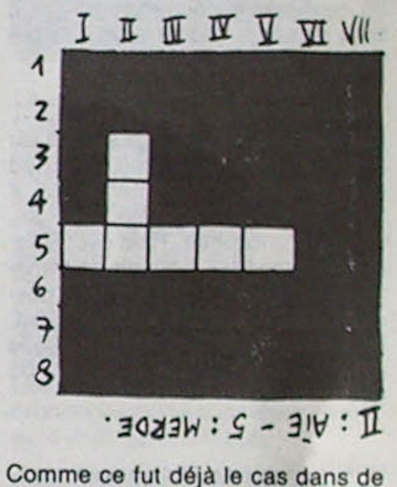
Rien que de plus classique au chapitre *Écriture* dans lequel, après permutation, DE pointe sur l'adresse du message tandis qu'HL pointe sur la fenêtre d'affichage à l'écran.

Plus original, par contre, est le compteur de scannings qui, à défaut d'appui sur une touche, va compter 200 passages par la routine (CALL $02BB) du Basic Sinclair avant d'effacer ou d'inscrire le message. Remarquez l'instruction CALL qui effectue l'appel du sous-programme d'adresse $02BB. Le symbole $ signifie, dans l'assembleur que j'utilise (ZX AS), que l'adresse est exprimée en hexadécimal. Le sous-programme est en fait ici une routine du basic présente en mémoire morte (ROM). L'alternance d'inscriptions et d'effacements est obtenue grâce à une instruction CPL inédite à ce jour. CPL, en effectuant la complémentation ou si vous préférez l'inversion du contenu de l'accumulateur (les 0 deviennent des 1 et vice-versa), commande avec le concours de l'instruction BIT 0 ,A l'aiguillage du programme une fois suivante vers l'effacement (0). BIT 0,A effectue un test du bit 0 de l'accumulateur et de ce fait positionne l'indicateur 2. L'instruction de saut relatif JR Z,L9 réalise ensuite l'aiguillage alternatif sur l'un ou l'autre des modules de traitement.

MOTS-CROISÉS

II VERTICAL : CRI DE DOULEUR

5 HORIZONTAL : CRI DE DOULEUR



Comme ce fut déjà le cas dans de précédentes routines, l'octet d'adresse 16417 joue le rôle de boîte postale en recueillant le code de la touche pressée... s'il y en a une.

Mais quoi ? Il n'y en a pas ! Et le message *va te faire f...* ne cesse de clignoter. Plus personne. Je crois qu'il est temps que je me tire aussi. Salut...

Bernard GUYOT.

```
LISTING ASSEMBLEUR ZX

REM     * SCANNING CLAVIER ET
          MESSAGE CLIGNOTANT

REM     * LIMITES CPIR
        LD HL.(16396)
        LD DE.16510
        PUSH DE
        XOR A
        SBC HL.DE
        PUSH HL

:L1     POP BC          inverse HL
        POP HL          et BC

REM     * CHERCHE INKEY$=
        LD A.234        REM
        CPIR
        RET NZ
        PUSH HL
        PUSH BC
        LD A.(HL)
        CP 65           INKEY$
        JR NZ.L1
        INC HL
        LD A.(HL)
        CP 20           =
        JR NZ.L1
        INC HL          pointe sur la couleur
        INC HL          pointe sur le message
        POP DE          défausse...
        POP DE          ... de la pile
        PUSH HL         départ message

REM     * ECRITURE
        LD HL.(16396) F.A.
        LD DE.762       24ème ligne
        ADD HL.DE       3ème colonne
        LD B.28         dimension fenêtre

:L9     POP DE          HL sur DE
        PUSH DE         départ message

:L2     LD A.(DE)
        CP 118
        JR Z.L6         fin de message
        LD (HL).A       écriture
        INC DE          pointeur message
        INC HL          pointeur F.A.
        DJ NZ.L2

REM     * COMPTEURS SCANNINGS

:L6     PUSH AF         accu et indic
        LD E.200        200 exécutions

:L5     DEC E
        JR Z.L7
        PUSH DE         compteur scanning

REM     * SCANNING
        CALL $02BB      routine Sinclair
        LD B.H
        LD C.L
        LD D.C
        INC D
        JR NZ.L4

:L13    POP DE          compteur scanning
        JR L5           nouvelle exécution

REM     * APPUI DETECTE

:L4     CALL $07BD
        LD A.(HL)
        CP 0            protection...
        JR Z.L13        ... contre BREAK
        LD (16417).A    code touche pressée
        POP BC          défausse...
        POP HL          ... de la pile

REM     * RETOUR
        LD HL.(16396)
        LD DE.762
        ADD HL.DE
        POP DE          adresse début de message
        DEC DE          pointe sur la couleur
        LD A.(DE)
        LD B.28         fenêtre à remplir

:L11    LD (HHL).A      coloriage de...
        INC HL          ... la...
        DJ NZ.L11       ... fenêtre
        RET

REM     * EFFACE OU REECRIT

:L7     POP AF
        CPL             complémentaire de A
        BIT 0.A         test
        LD HL.(16396)
        LD DE.762
        ADD HL;DE
        LD B.28
        JR Z.L9         aiguillage

:L8     LD (HL).0       efface...
        INC HL          ... le...
        DJ NZ.L8        message
        JR L6

ROUTINE RELOGEABLE DE 128 OCTETS
```

Formation à l'assembleur pratique

Langage machine... Sur COMMODORE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118
COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118
ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119
AMSTRAD -» 111 115 119
APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120
SPECTRUM -» 112 116 120
THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121
MSX -» 113 117 121

Le Lundi 27 janvier 1994, après une légère électrocution, Madame Soleil retrouvait ses esprits dans sa salle de bain, un simple problème d'isolation de son sèche-cheveux, mais elle venait d'avoir un rêve prémonitoire : elle savait que le lendemain, un incident d'intérêt international aurait lieu, peut-être la cinquième guerre mondiale, un tremblement de terre au Mont Saint Michel, le rachat d'IBM par Jack Tramiel, une sécheresse à Venise, la fin d'Hebdogiciel ou encore une bombe atomique sur Hiroshima...
Le mardi 28 janvier, dans l'après-midi, Challenger XXIV explosait dans le ciel de Cap Canaveral, à son bord une centaine de Commodore 64 (ordinateur démodé depuis 13 ans) destinés aux écoliers de Saturne, dans le cadre du plan informatique pour tous. Cet horrible accident aurait pu être évité si la NASA et CBM avaient écouté les conseils de notre voyante.
D'après ce texte, découvert dans un journal ramené par le dernier pilote de la machine à avancer dans le temps que possède votre hebdomadaire préféré, le petit ordinateur sur lequel vous vous excitez le vendredi soir devrait encore exister dans une dizaine d'années grâce à cet *excellent* cours, vous pourrez alors créer des programmes pédagogiques qui seront vendus sur Saturne (de quoi devenir très riche).
Vous pouvez éventuellement recréer des routines d'affichage, mais je vous conseille d'utiliser les tables mémoire. Votre écran contient 40 caractères par ligne, la première ligne commence en $0400 mais pour trouver l'adresse de début de la quinzième ligne (par exemple), il y a plus simple que calculer 15x40 ! En ECF0 et 00D9 se trouvent les tables d'adresses (LSB et MSB).

```
C000 A9 00     LDA #$00
C002 8D 0E C0  STA $C00E
C005 A9 D8     LDA #$D8
C007 8D 0F C0  STA $C00F
C00A AD 86 02  LDA $0286
C00D 8D 00 DC  STA $DC00
C010 EE 0E C0  INC $C00E
C013 AD 0E C0  LDA $C00E
C016 D0 F2     BNE $C00A
C018 EE 0F C0  INC $C00F
C01B AD 0F C0  LDA $C00F
C01E C9 DC     CMP #$DC
C020 D0 E8     BNE $C00A
C022 A9 00     LDA #$00
C024 85 FB     STA $FB
C026 A9 04     LDA #$04
C028 85 FC     STA $FC
C02A 85 FE     STA $FE
C02C A9 01     LDA #$01
C02E 85 FD     STA $FD
C030 A2 00     LDX #$00
C032 A0 00     LDY #$00
C034 B1 FB     LDA ($FB),Y
C036 48        PHA
C037 B1 FD     LDA ($FD),Y
C039 91 FB     STA ($FB),Y
C03B C8        INY
C03C C0 28     CPY #$28
C03E D0 F7     BNE $C037
C040 88        DEY
C041 68        PLA
C042 91 FB     STA ($FB),Y
C044 A5 FB     LDA $FB
C046 18        CLC
C047 69 28     ADC #$28
C049 85 FB     STA $FB
C04B 90 02     BCC $C04F
C04D E6 FC     INC $FC
C04F A5 FD     LDA $FD
C051 18        CLC
C052 69 28     ADC #$28
C054 85 FD     STA $FD
C056 90 02     BCC $C05A
C058 E6 FE     INC $FE
C05A E8        INX
C05B E0 19     CPX #$19
C05D D0 D3     BNE $C032
C05F 60        RTS
```

En étudiant cette routine ROM vous avez un exemple parfait de cette utilisation. Par l'appel de celle-ci (X contenant le numéro de ligne) vous obtenez le poids faible en D1 et le poids fort en D2.

Maintenant que vous connaissez un peu mieux l'affichage, nous allons pouvoir réétudier la routine de scrolling à gauche du 6 septembre (n° 99). En voici une version légèrement modifiée.

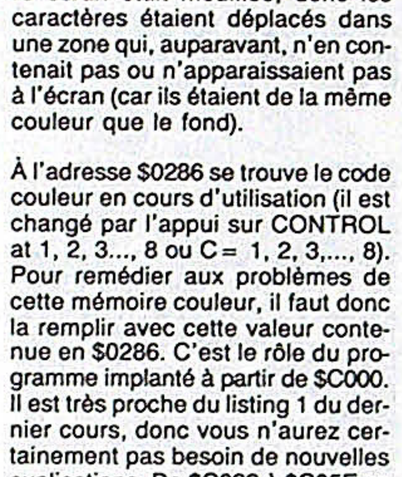
```
C060 A9 00     LDA #$00
C062 8D 6E C0  STA $C06E
C065 A9 D8     LDA #$D8
C067 8D 6F C0  STA $C06F
C06A AD 86 02  LDA $0286
C06D 8D 00 DC  STA $DC00
C070 EE 6E C0  INC $C06E
C073 AD 6E C0  LDA $C06E
C076 D0 F2     BNE $C06A
C078 EE 6F C0  INC $C06F
C07B AD 6F C0  LDA $C06F
C07E C9 DC     CMP #$DC
C080 D0 E8     BNE $C06A
C082 A9 00     LDA #$00
C084 85 FB     STA $FB
C086 A9 04     LDA #$04
C088 85 FC     STA $FC
C08A 85 FE     STA $FE
C08C A9 01     LDA #$01
C08E 85 FD     STA $FD
C090 A2 00     LDX #$00
C092 A0 28     LDY #$28
C094 B1 FD     LDA ($FD),Y
C096 48        PHA
C097 B1 FB     LDA ($FB),Y
C099 91 FD     STA ($FD),Y
C09B 88        DEY
C09C C0 00     CPY #$00
C09E D0 F7     BNE $C097
C0A0 C8        INY
C0A1 68        PLA
C0A2 91 FD     STA ($FD),Y
C0A4 A5 FB     LDA $FB
C0A6 18        CLC
C0A7 69 28     ADC #$28
C0A9 85 FB     STA $FB
C0AB 90 02     BCC $C0AF
C0AD E6 FC     INC $FC
C0AF A5 FD     LDA $FD
C0B1 18        CLC
C0B2 69 28     ADC #$28
C0B4 85 FD     STA $FD
C0B6 90 02     BCC $C0BA
C0B8 E6 FE     INC $FE
C0BA E8        INX
C0BB E0 19     CPX #$19
C0BD D0 D3     BNE $C092
C0BF 60        RTS
```

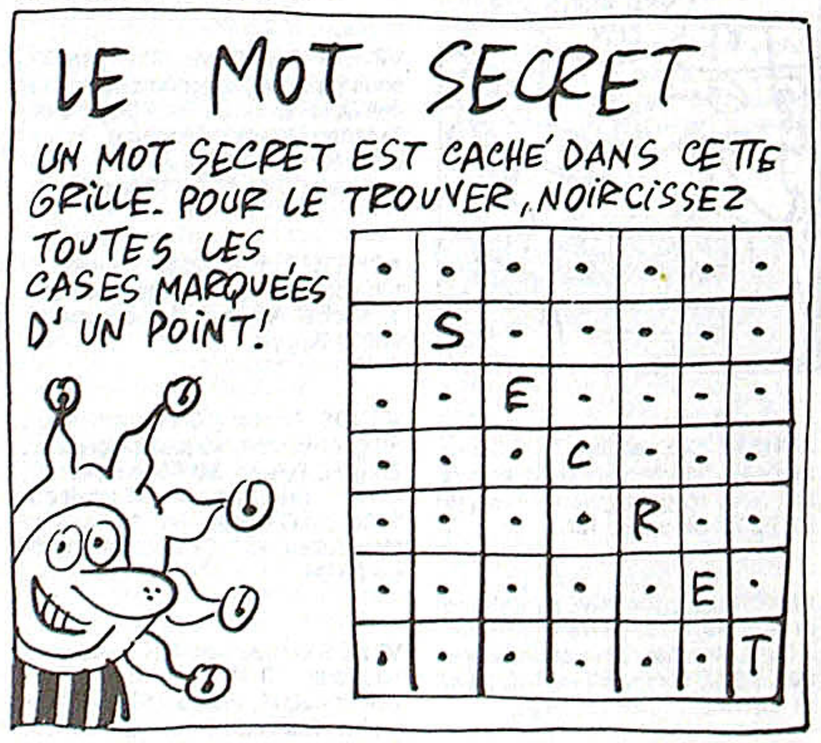
Quand vous appelez une routine d'affichage (comme $EA13), celle-ci se charge de modifier le code du caractère dans la mémoire écran, mais aussi le code couleur dans la mémoire couleur (formidable !). Dans la précédente version de notre routine d'affichage, seule la mémoire écran était modifiée, donc les caractères étaient déplacés dans une zone qui, auparavant, n'en contenait pas ou n'apparaissaient pas à l'écran (car ils étaient de la même couleur que le fond).

À l'adresse $0286 se trouve le code couleur en cours d'utilisation (il est changé par l'appui sur CONTROL at 1, 2, 3..., 8 ou C= 1, 2, 3,..., 8). Pour remédier aux problèmes de cette mémoire couleur, il faut donc la remplir avec cette valeur contenue en $0286. C'est le rôle du programme implanté à partir de $C000. Il est très proche du listing 1 du dernier cours, donc vous n'aurez certainement pas besoin de nouvelles explications. De $C022 à $C05E se trouve le programme de décalage. Reportez-vous au paragraphe nommé ESPLICASSIONNESSES du cours correspondant.

Une petite variation : un scrolling à droite. Il est réalisé à l'aide des mêmes astuces que son petit frère vu au-dessus.

```
C200 A5 C5     LDA $C5
C202 C9 40     CMP #$40
C204 F0 04     BEQ $C20A
C206 C5 C5     CMP $C5
C208 F0 FC     BEQ $C206
C20A 20 00 C0  JSR $C000
C20D A5 C5     LDA $C5
C20F C9 40     CMP #$40
C211 F0 F7     BEQ $C20A
C213 A5 C5     LDA $C5
C215 C9 40     CMP #$40
C217 F0 04     BEQ $C21D
C219 C5 C5     CMP $C5
C21B F0 FC     BEQ $C219
C21D 20 60 C0  JSR $C060
C220 A5 C5     LDA $C5
C222 C9 40     CMP #$40
C224 F0 F7     BEQ $C21D
C226 C9 3E     CMP #$3E
C228 D0 D6     BNE $C200
C22A 60        RTS
```





Pour utiliser ces deux programmes, vous les appelez par un Jump C000 ou un Jump C060. Si les effets de ces programmes ne vous paraissent pas évident, rendez-vous compte de l'influence de ceux-ci grâce à une nouvelle routine que voici :

```
C000 A5 C5     LDA $C5
C002 C9 40     CMP #$40
C004 D0 05     BNE $C00B
C006 85 FB     STA $FB
C008 4C 31 EA  JMP $EA31
C00B C5 FB     CMP $FB
C00D F0 F9     BEQ $C008
C00F 85 FB     STA $FB
C011 A9 00     LDA #$00
C013 8D 04 D4  STA $D404
C016 8D 05 D4  STA $D405
C019 8D 06 D4  STA $D406
C01C 8D 00 D8  STA $D800
C01F 8D 00 D8  STA $D800
C022 A9 0F     LDA #$0F
C024 8D 18 D4  STA $D418
C027 8D 06 D4  STA $D406
C02A A9 02     LDA #$02
C02C 8D 05 D4  STA $D405
C02F A9 11     LDA #$11
C031 8D 00 D4  STA $D400
C034 8D 01 D4  STA $D401
C037 8D 04 D4  STA $D404
C03A 4C 31 EA  JMP $EA31
```

En $C5 se trouve le code de la touche appuyée, quand le clavier est au repos (aucune allusion à quoi que ce soit) $C5 prend la valeur $40. Quand vous lancez cette routine par un Jump C200 (ou équivalent) votre doigt reste quelque temps sur RETURN, donc le programme attend que vous ayez relâché cette touche avant de s'occuper du contenu de $C5.

C202 Si le code correspond au clavier normal, il saute en $C02A (aucune touche appuyée)

C206 Comparaison du clavier avec lui-même, si son état change entretemps, c'est que vous avez relâché RETURN

C20A Scrolling à gauche

C20D à C211 Si aucune touche n'est pressée, on recommence le scrolling

C213 à C21B On attend de nouveau que le clavier retourne à son état initial (repos)

C21D Scrolling à droite

C220 à C224 Identique à la zone débutant en C20D

C226 à C228 Si vous avez appuyé sur Q le programme s'arrête, sinon...

Vous êtes maintenant des professionnels de l'affichage basse résolution sur Commodore, je n'ai plus rien à vous apprendre à ce sujet, alors en guise de conclusion, cette petite routine qui n'a absolument aucun rapport avec le circuit vidéo, mais qui introduit prématurément un des prochains cours : *le circuit sonore !*

```
C100 78        SEI
C101 A9 00     LDA #$00
C103 8D 14 03  STA $0314
C106 A9 C0     LDA #$C0
C108 8D 15 03  STA $0315
C10B 58        CLI
C10C 60        RTS
```

Avant de faire quoi que ce soit, utilisez le programme chargé de réaliser la modification de l'IRQ.

```
E9F0 BD F0 EC  LDA $ECF0,X
E9F3 85 D1     STA $D1
E9F5 B5 D9     LDA $D9,X
E9F7 29 03     AND #$03
E9F9 0D 88 02  ORA $0288
E9FC 85 D2     STA $D2
E9FE 60        RTS
```

N'oubliez pas le Jump C100 pour lancer le programme.

Je vous souhaite maintenant de très bonnes vacances (pour ceux qui y sont encore). Si vous partez au ski, je vous conseille de vous casser une jambe et d'emporter votre Commodore, vous pourrez le bidouiller 24 sur 24 pendant que votre petite copine se fera draguer par les beaux moniteurs de ski ! Bonne convalescence...

Sébastien MOUGEY

Langage machine... Sur ZX 81

Trois petits tours et puis s'en vont, ainsi font font font... 767 petits décalages puis s'en ira, ainsi fera fera fera votre fichier d'affichage... grâce à ce programme destiné aux adultes en bas âge de nos grandes écoles d'informatique et autres établissements d'enseignement supérieur à la moyenne nationale qui, comme chacun sait, assoie le débile moyen entre le débile léger et le débile grave. Mais quelle que soit la catégorie à laquelle vous appartenez : ni honte, ni fierté excessive ! Dans ces colonnes, en effet, nulle trace de sectarisme intellectuel, pas de morgue à l'encontre des smicars du Q.I., des chômeurs de l'encéphale, pas d'avantage de considération pour les stakhanovistes du concept ou les trôneurs du spipirituel. Non ! Rassurez-vous cher é-lecteur, mon éducation soignée, mon expérience intense, riche et déjà longue de la vie que j'étirais par surcroît et paresseusement, ainsi que mes talons de deux centimètres qui me permettent aisément de m'élever au-dessus du débat politique et des partis, tout cela me laisse à penser aujourd'hui qu'aucun électeur ne vaut moins qu'un autre, ni surtout mieux qu'un autre, ni enfin mieux que moi-même. Beaucoup parmi vous chercheront vainement mon nom sur les listes électorales. Inutile : j'ai toujours recours au pseudonyme d'une poudre à laver qui m'assure les suffrages des bonnes ménagères et des hommes célibataires propres. Dois-je avouer ici que je suis l'éminence grise de l'éminence grise d'une éminence grise, le point de départ d'un réseau d'influences que vous ne soupçonnez pas et qui, à travers MT, MD, GC et quelques autres aboutit jusqu'à... Vous ne saurez rien de plus, j'en ai d'ailleurs déjà trop dit. Selon ma garde du corps il serait même préférable que je ne signe pas ce papier et que je me serre au plus près d'elle pour raisons de sécurité.
C'est pourquoi je décrirai maintenant de façon détaillée et sans digression la routine de ROTATION/DISPARITION qui je le souhaite recueillera tous vos suffrages. Voici donc le programme et d'abord son objectif : il s'agit de générer sur l'écran de votre ZX une rotation de l'image de la gauche vers la droite, rotation qui s'accompagne du nettoyage progressif de l'écran du haut vers le bas. En fait l'image donne l'impression de se visser dans le bas de l'écran ce qui n'est qu'une illusion puisque le sens habituel du pas de vissage est inversé. Il s'agit donc bien d'un logiciel original, cent pour cent soft et qui ne doit rien à la mécanique.
Ce programme très simple se décompose facilement en une succession de modules de traitement tous accessibles par une instruction de saut conditionnel (JR NZ.L), raison pour laquelle ils débutent tous par le label obligé du ZX ASSEMBLER (:L).
Les premières lignes de cette routine effectuent un travail facilement identifiable : effacement du contenu de la première case du fichier d'affichage; cependant sa signification par rapport à l'ensemble du programme vous échappe peut-être. Il faut pour la comprendre connaître le principe de fonctionnement de la routine.
Avant de trouver les moyens logiciels de réaliser notre objectif, il convient d'analyser le mécanisme mis en jeu pour obtenir cet effet de rotation/disparition : c'est l'Analyse qui précède toujours la Programmation.
Pour obtenir un décalage de l'image vers la droite combiné avec sa sortie progressive par le bas de l'écran, il nous suffira de déplacer d'une case en aval le contenu de chaque octet du F.A., autrement dit de pointer sur l'adresse qui suit immédiatement celle de l'octet à déplacer afin de l'y installer là et pas ailleurs. Ceci semble extrêmement simple et d'ailleurs l'est à deux difficultés près : La plus facile à surmonter consiste à éviter les cases du F.A. marquées d'un indélébile 118 (fin de lignes, début et fin du F.A.). Pour les détecter et donc modifier le déroulement du programme on a recours aux classiques instructions d'assemblage "CP" dont l'opérande sera évidemment la valeur recherchée : CP 118. Le positionnement automatique de l'indicateur Z à l'issue de ce test de comparaison permet de modifier le séquencement des opérations grace à l'instruction "JR NZ.L". Ce sont là deux instructions courantes que les habitués de l'HHHebdo connaissent bien.

```
REM * ROTATION/DISPARITION

      LD HL,(16396)     F.A.
      INC HL
      LD A,0            efface 1'case
      LD (HL),A
      LD DE,767         (24 X 32)-1
      PUSH DE

REM * INITIALISATIONS
:L1   LD HL,(16396)
      LD DE,799         (24 x 32)-2
      ADD HL,DE
      LD DE,767         compteur décalages

REM * EVITE LES 118
:L2   LD A,(HL)
      CP 118
      JR NZ,L3
      DEC HL
      JR L2

REM * PRELEVEMENT/TEST
:L3   LD C,A            sauvegarde
      PUSH HL           sauvegarde
      INC HL            pointe case aval
      LD A,(HL)
      CP 118
      JR NZ,L4
      INC HL

REM * ECRITURE ET COMPTEURS
:L4   LD (HL),C         écrit en aval
      POP HL            double...
      DEC HL            ...décrémentation
      DEC DE            compteur décalages
      LD A,D
      CP 0
      JR NZ,L2          travail...
      LD A,E            ...terminé?
      CP 0
      JR NZ,L2

      POP DE            compteurs passes
      DEC DE
      PUSH DE

      LD A,D
      CP 0
      JR NZ,L1          image...
      LD A,E            ...évacuée?
      CP 0
      JR NZ,L1

      POP DE            défausse de pile
      RET
```

La seconde difficulté ne doit pas, elle non plus, effrayer le programmeur débutant. Celui-ci serait d'emblée tenté de commencer le travail au début du F.A. en incrémentant successivement les adresses des octets a déplacer. Il est pourtant préférable d'entamer le travail par la fin du F.A. en utilisant des décrémentations d'adresses; cela évite d'avoir à sauvegarder le contenu de l'octet aval avant que l'octet amont n'y soit redupliqué (mais si ! réfléchissez bien).

Notre routine aura donc la structure suivante :
1 - pointer sur l'avant-dernière case du F.A.
2 - prélever son contenu
3 - pointer sur l'adresse suivante (incrémentation)
4 - dupliquer l'octet amont
5 - Effectuer une double décrémentation d'adresse...
6 - ...et retour à la phase 2

Il suffit d'ajouter à ceci deux "compteurs", l'un pour comptabiliser le nombre d'octets à décaler d'une adresse vers la droite (24 lignes X 32 colonnes) = 768 octets, l'autr?e de passes nécessaires à la disparition totale de l'image : ici aussi 768 passes. Vous constatez que cela nous fait un nombre impressionnantd'opérations à effectuer pour arriver à nos fins : 768 puissance 2 = 589824 opérations ! Inutile donc de prévoir des boucles de ralentissement !
Il est temps maintenant d'élucider l'énigme de l'effacement du premier octet du F.A. ! Encore une fois réfléchissez : s'il était dupliqué à chaque passe, le contenu de cet octet finirait par recouvrir la totalité de l'écran puisque rien ne vient jamais "l'écraser". Alors autant choisir la couleur de l'écran que vous laisserez derrière vous en modifiant la troisième ligne du programme (LD A,x).
Extrêmement linéaire, ce programme pourra aisément être compris par les débutants. Quant aux plus chevronnés, ils se feront un devoir d'en réécrire une mouture plus concise. N'est-ce pas ?

Bernard Guyot

Formation à l'assembleur pratique

Langage machine... Sur ZX 81

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122
COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122
ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123
AMSTRAD -» 111 115 119 123
APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124
SPECTRUM -» 112 116 120 124
THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125
MSX -» 113 117 121 125

D'aucuns prétendront que c'est au mépris de toute rationnalité scientifique, contraire à l'éthique professionnelle d'un spécialiste patenté de la micro-informatique, indigne d'un responsable exerçant ses hautes fonctions dans les colonnes du plus important des hebdomadaires spécialisés, inacceptable de la part d'une autorité universellement reconnue dont l'influence ne cesse de croître d'HHHebdo en HHHebdo, pourtant c'est sans crainte des huées de mes détracteurs, des quolibets proférés contre moi dans les chaumières préfabriquées contemporaines, des jets de fruits et légumes dont les cours peuvent brutalement s'effondrer, d'éventuels attentats dont ma personne pourrait être l'objet dans le métro aux heures de pointe, ou pire encore de la prise d'otage qui pourrait m'arracher à l'affection des biens meubles et immeubles dont j'ai pu faire état lors de ma dernière déclaration fiscale, c'est donc sans craindre aucune de ces calamités où l'homme contrarié excelle et sans cliquetis de prothèse (ma jambe droite écrasée par un chauffard le 23 juin 82), mais en redressant/resserrant mon corset (coups et blessures - correctionnel du 6 sept 84) et réorientant ma minerve (claque dans la gueule avant-hier par mon voisin de palier) que je proclame aujourd'hui solennellement qu'à l'instar de mon ours en peluche, de mon chien et de deux de mes petites amies, l'ordinateur familial et familièrement appelé ZX81 a une âme.

Oui ! Ce qui aurait pu apparaître aux yeux du technicien borné comme n'étant qu'un sandwiche de bakélite fourré à la puce nipponne ou californienne, ou pour l'utilisateur simple comme un fatras de mauvais contacts engendrant péniblement sur le téléviseur du salon des programmes dignes de la cinquième chaîne, ou encore pour ma concierge une machine qui, par l'intérêt qu'elle suscite en elle, s'apparente aux rasoirs électriques double lames, ce petit boîtier noir, je n'hésite pas à l'affirmer, offre à mes yeux toutes les manifestations possibles de ce qu'il faut bien appeller une personnalité.

Il y a d'abord sa langue, son vocabulaire, sa capacité à élaborer, tout comme nous, des chaînes de caractères (à propos, je vous conseille d'ajouter de façon systématique aux vôtres des guillemets au début et à la fin pour des raisons évidentes de clarté). Ensuite, rivalisant avec l'humanité profonde d'un rond-de-cuir du ministère des finances ne peut-il construire lui aussi de merveilleux tableaux de chiffres ? (Fonctionnaires de toutes administrations prenez exemple sur le ZX : dimensionnez vos tableaux préalablement ! Vous nous épargnerez les habituels déficits budgétaires !). Et s'il vous fallait des preuves supplémentaires de l'évidente parenté qu'il existe entre l'homme et l'ordinateur je pourrais encore citer deux instructions du langage Basic qui, comme diraient nos éligibles de Mars, sont à cet égard significatives : "RND" qui peut à tout moment faire éclater la logique rigoureuse de la machine en lui substituant une démarche totalement aléatoire. N'est-ce pas là l'équivalent du hasard ou de la folie qui peuvent à tout moment modifier notre destin si merveilleusement planifié de la crèche à la retraite en passant par l'école, la caserne et l'usine sans oublier tous les jours la demi-baguette moulée pas trop cuite chez le boulanger du coin.
Et cette instruction "PAUSE" qui nous renvoie directement à ce qui est sans aucun doute le propre de l'homme : la fatigue. L'ennui, la lassitude qu'engendre le travail, manifestation naturelle d'une humanité accomplie sans laquelle nous nous verrions ravalés au rang des japonais et de leur éblouissante réussite économique et technico-industrielle !

Vous le constatez vous-même cher lecteur, le doute désormais n'est plus permis, et si nous voulons nous situer dans une perspective darwinienne nous pouvons affirmer sans la moindre ambiguïté et avec un énorme haut-le-cœur : L'ordinateur descend de l'homme !

Pour en finir une bonne fois avec le mythe tenace de l'ordinateur nonpensant, l'idée m'est venue de sonder plus avant sa personnalité binaire grâce à une technique éprouvée depuis l'abbé Michon (plus connu pour ses conserves de porc) et actualisée par des graphologues dont le sérieux et la compétence ne sont plus mis en question que par les machines à écrire. C'est pourquoi, à l'émission de José Arthur qui, aidé d'une graphologue doit identifier un inconnu célèbre, je laissais mon ZX répondre à ma place et soumettais à Nœlle Robert un exemplaire d'écriture tiré avec difficulté de mon imprimante Sinclair. Le diagnostic fut le suivant : "écriture aérée, espacée, juxtaposée et typographique, mais caractères peu lisibles, tremblés, tordus, hampes et jambages inclinés sur la gauche, écriture de type centripète plus haut que vous mon cher José." - Qu'en pouvez-vous déduire jextraordinaire graphologue ?
"Personnalité intéressante dans sa petite enfance mais aujourd'hui fatiguée, dépassée. Esprit logique et calculateur mais dysfonctionnements fréquents. Utilisation vraisemblable d'une prothèse en guise de main droite. Adrénaline en faible quantité et pas de trace de testostérone dans l'écriture : c'est donc une femme."

Voilà une grande nouvelle pour tous les mâles amoureux du ZX : ils sont hétéros...

PS : L'écriture du ZX étant un peu trop petite au goût des graphologues, je vous communique cette routine en langage machine permettant d'obtenir, cette fois à l'inverse des humains, des grands à partir des petits -caractères bien sûr ! Très simple a utiliser : il suffit de rentrer le message à afficher en "géant" dans la variable chaîne de caractères C$ en ayant soin de le faire précéder de deux signes "$" en vidéo inversée; la fin du message sera signalée par un pavé en vidéo inversée codé 128. La forme est donc la suivante :
C$ = "$$....message....■"
Le fonctionnement est le suivant : un buffer est créé en RAM à l'adresse 16335 pour les 64K ou 16444 (PRBUFF) pour les 16K. Après localisation de la chaîne C$ en RAM on duplique le message dans le buffer avant de le "traiter" caractère par caractère dans la routine "grands caractères". Cette routine fait appel à des sous-programmes de la ROM Sinclair, je ne rentrerai donc pas dans le détail de son fonctionnement. Appliquez-vous surtout à saisir le système de localisation d'une chaîne de caractères dans l'espace RAM affecté aux variables (VARS).

Bernard Guyot

```
REM * GRANDS CARACTERES

REM * CHARGEMENT BUFFER
:L30  LD DE.16335        départ buffer
      PUSH DE            ou 16444
      LD HL.(16400)      VARS

:L20  LD A.(HL)
      CP 72              C$ codé chaine
      INC HL
      JR NZ.L20
      PUSH HL            pile niv.2
      INC HL
      INC HL
      LD A.(HL)
      CP 141             $ inversé
      JR NZ.L21
      INC HL
      CP 141             $ inversé
      JR Z.L22

:L21  POP HL             poursuit en
      JR L20             en cas d'échec

:L22  POP DE             départ chaine C$
      INC HL             début message C$
      POP DE             départ buffer

:L23  LD A.(HL)          remplissage du
      LD (DE).A          buffer avec C$
      CP 128             fin de message
      JR Z.L24
      INC HL             suivant C$
      INC DE             next buffer
      JR L23             jusqu'au 128

:L24  LD HL.16335        ou 16444 (PRBUFF)
      PUSH HL
      D BC.$28F8

:L10  PUSH BC

REM * GRANDS CARACTERES
      POP BC
      POP HL
      LD A.(HL)
      CP 128             fin de message
      RET Z              terminé
      INC HL
      PUSH HL
      PUSH BC
      LD H.0
      LD L.A
      ADD HL.HL
      ADD HL.HL
      ADD HL.HL
      LD DE.$1E00
      ADD HL.DE
      LD D.08
      LD E.D
:L1   LD A.(HL)
:L2   SLA A              décalage arithmétique
      POP BC             à gauche
      JR NC L0
      PUSH AF
      PUSH BC
      PUSH DE
      PUSH HL
      CALL $0BB2         appel de sous-progr.
      POP HL
      POP DE
      POP BC
      POP AF
:L0   INC C
      PUSH BC
      DEC E
      JR NZ.L2
      INC HL
      POP BC
      DEC B
      PUSH BC
      DEC D
      JR NZ.L1
      POP BC
      LD A.C
      CP $38
      JR C.L5
      LD C.$F8
      DEC B
      RET M              si S=1
      JR L10
:L5   LD A.B
      ADD A.8
      LD B.A
      JR L10
```

Langage machine... Sur COMMODORE 64

Le son est une sensation physiologique, une vibration du tympan provoquée par des variations de pression de l'air qui se propagent de la source sonore jusqu'à l'oreille.
Son : effet des vibrations rapides des corps, se propageant dans les milieux matériels et excitant l'organe de l'ouïe.
Tympan : membrane située au fond du conduit auditif, qui transmet les vibrations de l'air aux osselets de l'oreille moyenne.
Pression : action d'un fluide sur une surface.
Source sonore : petit appareil beige comportant 66 bouts de plastique de couleur marron, un voyant sur la partie supérieure droite (de couleur rouge) et une étiquette sur la partie supérieure gauche. Le tout relié à un haut-parleur via un amplificateur.

Comme vous ne l'avez absolument pas deviné, nous traiterons aujourd'hui des sons. La petite surprise du dernier cours n'était qu'une introduction.
Mais avant toute chose, vous devez savoir que le CI de son, 6581 en l'occurrence, a une valeur de 174,34 francs TTC (tarif à fin Janvier). l'HHHebdo décline toute responsabilité en cas de destruction de celui-ci !
Ce cours sera beaucoup plus théorique que pratique, mais je vous promets que le prochain n'aura pratiquement pas de texte, que des listings !

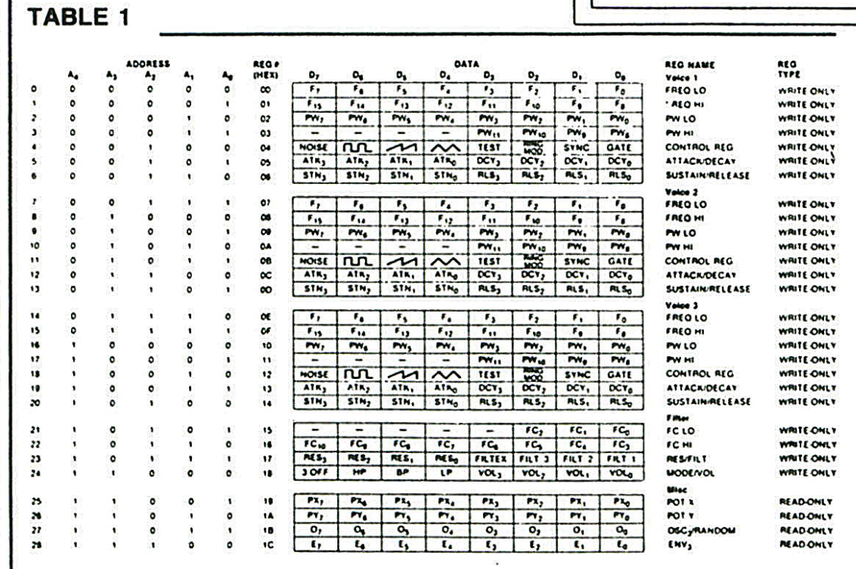
Le circuit sonore est adressé dans la mémoire Commodore de D400 à D41C comme suit :

TABLE ADRESSES

D400-D41C	54272-54300	Sound chip (SID).
D400	54272	Voice 1 low frequency.
D401	54273	Voice 1 high frequency.
D402	54274	Low pulse (pulse waveform only) voice 1.
D403	54275	High pulse (pulse waveform only) voice 1.
D404	54276	Waveform voice 1.
D405	54277	Attack/decay voice 1.
D406	54278	Sustain/release voice 1.
D407	54279	Low frequency voice 2.
D408	54280	High frequency voice 2.
D409	54281	Low pulse (pulse waveform only) voice 2.
D40A	54282	High pulse (pulse waveform only) voice 2.
D40B	54283	Waveform voice 2.
D40C	54284	Attack/decay voice 2.
D40D	54285	Sustain/release voice 2.
D40E	54286	Low frequency voice 3.
D40F	54287	High frequency voice 3.
D410	54288	Low pulse (pulse waveform only) voice 3.
D411	54289	High pulse (pulse waveform only) voice 3.
D412	54290	Waveform voice 3.
D413	54291	Attack/decay voice 3.
D414	54292	Sustain/release voice 3.
D415	54293	Low cutoff frequency (0-7).
D416	54294	High cutoff frequency (0-255).
D417	54295	Resonance (bits 4-7). Filter voice 3 (turn off) bit 2. Filter voice 2 (bit 1). Filter voice 1 (bit 0).
D418	54296	High pass filter (bit 6). Low pass filter (bit 4). Band pass filter (bit 5). Master volume control (bits 0-3 (0-15)).

Mais les octets D419, D41A, D41B et D41C ne jouent pas sur la programmation (utilisés en lecture). L'utilisation est très proche du mode basic, mais tout de même simplifiée par la profusion de mnémoniques ou d'instructions du 6510.

TABLE 1



Voici l'explication bit par bit des adresses, la colonne REG représente le poids faible, il faut donc prendre D4 comme poids fort. Pour le calcul des fréquences, voir page 152 du Manuel de l'Utilisateur ou l'Appendice M. Dans le cas du signal rectangulaire, PW Lo et PW Hi commandent le rapport cyclique. Les temporisations de l'ADSR sont les suivantes :

TABLE 2

VALUE		ATTACK RATE	DECAY/RELEASE RATE
DEC	(HEX)	(Time/Cycle)	(Time/Cycle)
0	(0)	2 ms	6 ms
1	(1)	8 ms	24 ms
2	(2)	16 ms	48 ms
3	(3)	24 ms	72 ms
4	(4)	38 ms	114 ms
5	(5)	56 ms	168 ms
6	(6)	68 ms	204 ms
7	(7)	80 ms	240 ms
8	(8)	100 ms	300 ms
9	(9)	250 ms	750 ms
10	(A)	500 ms	1.5 s
11	(B)	800 ms	2.4 s
12	(C)	1 s	3 s
13	(D)	3 s	9 s
14	(E)	5 s	15 s
15	(F)	8 s	24 s

Voie 1

GATE (D404 bit 0)
A l'état haut, démarrage du cycle attack/decay/sustain.
A l'état bas, cycle release.
SYNC (bit 1)
A l'état haut, synchronisation des oscillations entre la voie 1 et la voie 3.
RING MOD (bit 2)
Effet produisant un son de *timbre de cloche*, basé sur la création de fréquences non harmoniques.
Voie 1 en triangle : variation de fréquence de la voie 1 simultanément avec la voie 3.
TEST (bit 3)
A l'état haut, initialisation de la voie 1 (mise à 0 de tous les bits).

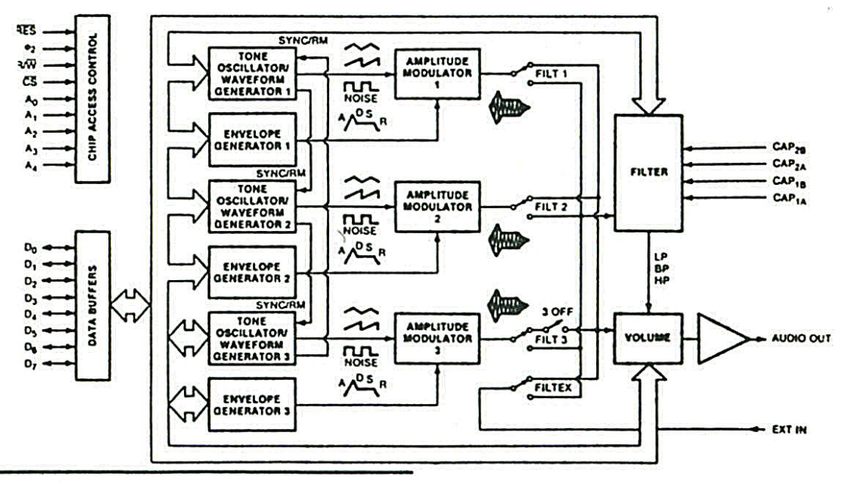
Voie 2

SYNC & RING MOD (D40B bits 1 et 2)
Mêmes effets que pour la voie 1, mais accord entre les voies 2 et 1.

Voie 3

SYNC & RING MOD (D4112, bits 1 et 2)
Mêmes effets que précédemment mais accord entre les voies 3 et 2.

SCHEMA 1



FILTRES
FClo/FChi (D415,D416)
Réglage de la fréquence de coupure des filtres.
MODE (D418)
LP (bit 4)
Filtre passe bas. Toutes les fréquences inférieures à FClo/FChi sont transmises normalement, les fréquences supérieures sont atténuées.
BP (bit 5)
Filtre passe bande. Les fréquences proches de FClo/FChi ne sont pas atténuées.

HP (bit 6)
Les fréquences supérieures à FClo/FChi ne sont pas atténuées.

RES/FILT (D417)
FILT 1 (bit 0)
A l'état haut, la voie 1 est filtrée.

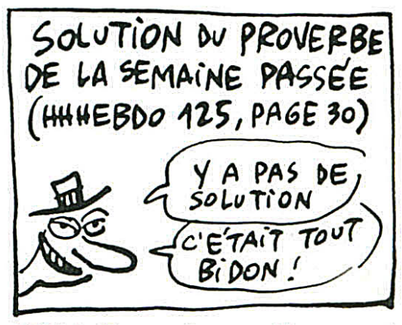
FILT 2 (bit 1)
A l'état haut, la voie 2 est filtrée.

FILT 3 (bit 2)
A l'état haut, la voie 3 est filtrée.

FILT EX (bit 3)
Le signal extérieur est filtré à l'état haut. Entrée disponible sur la broche 5 de la DIN audio/vidéo. L'impédance d'entrée est de 100 kiloohms. Ne dépassez pas 3 volts crête à crête si vous ne voulez pas tout flinguer dans votre micro.

RES (bits 4 à 7)
Effet de résonnance sur les filtres, réglables de 0 à 15. Enfin pour les fous du bricolage, voici le schéma interne :

J'espère que vous arriverez à digérer toutes ces informations. Mine de rien, si vous pigez tout ça, vous allez pouvoir passer pour les mozarts du microprocesseur musical, que dis-je le Mozart ? Le Beethoven du 6581 ! Bon, allez, suffisamment déliré, apprenez tout ça bien comme il faut, dans quatre semaines on fera joujou avec tous (tous ?) les bits dont je viens de vous entretenir (les bits ?... entré ? tenir ?).



Sébastien MOUGEY

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126
COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126
ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128
AMSTRAD -» 111 115 119 123 127
APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128
SPECTRUM -» 112 116 120 124 127
THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129
MSX -» 113 117 121 125 129

LANGAGE MACHINE SUR ZX 81

Âmes transparentes et virginales, regards clairs, esprits lumineux et sereins, ardents défenseurs de la veuve même joyeuse, de l'orphelin même délinquant et des allocations familiales pour tous, vous qui croyez encore en l'HHHHebdo et pouvez faire l'amour sans arrière-pensée, hommes de grande volonté qui avez foi en la multiplication des pains, la diminution consécutive des coûts de production et le sens de l'HHHHebdo, je vous le dis : tournez la page ou détournez votre regard car ces lignes ne sont pas pour vous.

Aujourd'hui je m'adresse aux vicieux. J'élargis mon audience en ratissant du côté des faux-culs, des esprits retors et hypocrites. Je rameute les lâches, les fourbes, toute une faune malsaine qui lorgne avec concupiscence la femme du voisin et convoite son micro-ordinateur comme l'ensemble de sa logithèque. A ceux-là, que j'appelle mielleusement *mes frères* sachant qu'ils ne sont au mieux que mes semblables, à ceux-là seulement j'adresse, avec l'expression de mes sentiments les moins respectueux, cette routine en langage machine qui, en l'espace d'une centaine d'octets et d'un dixième de seconde, transformera n'importe quel honnête listing basic en véritable champ de mines, vrai piège à cons, arme universelle...

Imaginez un listing où tout apparemment soit normal, où les instructions du langage basic sont convenablement alignées, sagement suivies de leurs arguments et des opérateurs logiques ou mathématiques qui semblent n'avoir d'autre vocation que soulager l'humanité laborieuse de la difficulté de penser que (x-2) < (y/3) mais voilà qu'une fois lancé le fameux RUN -dont, à en croire les plus récentes statistiques, la fréquence d'utilisation dépasse aujourd'hui nettement celle du mot de confiance ce qui marque un fabuleux tournant dans l'histoire de la civilisation- voilà que le résultat affiché à l'écran est une colossale erreur, une insulte à Boole, Pascal, Euclide et bien d'autres jusqu'au programmeur boutonneux qui en perd son basic et finirait même apr douter de la réalité de son acné post-juvénile. Il n'en faut pas d'avantage pour qu'une navette spatiale se transforme en feu d'artifice ou que le baron Empain soit déclaré non imposable par l'administration fiscale, si l'on en reste à la stricte analogie. Cette routine offre donc des perspectives au moins aussi excitantes pour l'esprit que le spectacle d'une catéchumène reconvertie au peep-show par un habile *cœur vaillant*.

Voyons plus précisément de quoi il retourne : cette routine a pour objet de substituer à certains opérateurs arithmétiques ou logiques présent sur le listing original, d'autres opérateurs qui eux n'ont rien à y faire sinon fausser complètement et d'une façon idéalement sournoise les résultats escomptés. Il s'agit donc de truquer rapidement et discrètement le listing d'un programme basic afin de rendre son utilisation ou sa copie inopérante. Cette routine extrêmement simple, à la trace linéaire et répétitive, est accessible à tous les débutants et je l'espère amènera au langage machine ceux qui n'ont pas encore osé s'y risquer. Pour eux nous allons détailler sa structure et analyser *pas à pas* son fonctionnement.

Attention ! Cette routine est dangereuse ! C'est pourquoi je propose avant toute utilisation d'effectuer un test simple qui décidera soit de l'activer, soit au contraire de la court-circuiter. C'est un feu de signalisation mis au *vert :* pas de danger, mis au *rouge :* trucage obligé du listing ! Il s'agit en réalité d'un octet disponible au début de la mémoire RAM, en amont du listing basic, dans cette zone dite des *variables système.* Cet octet sera le dépositaire d'une valeur *verte* (exemple : 100) ou *rouge* (toute autre valeur que 100). Une simple instruction Poke permettra de le colorer à sa guise. N'oublions pas que la valeur ainsi pokée sera sauvegardée en cas de copie. Votre garde-fou consiste donc en un test du contenu de cet octet suivi d'un retour conditionnel au basic en cas de feu vert. Dans le cas contraire on poursuit séquentiellement sur la routine de substitution.

Sa structure est bipartite. D'abord la délimitation de l'espace en RAM sur lequel interviendront d'éventuelles substitutions voulues. Ici trois modules qui métamorphoseront les ★ en / puis les + en - enfin les > en <.

Trouver les adresses de début et de fin de listing ne pose pas de difficulté. On sait que le corps d'un programme basic commence invariablement à l'adresse 16514, immédiatement après l'espace réservé aux variables système. en revanche, il nous faut interroger l'une de ces variables système pour localiser l'adresse du dernier octet d'un listing dont les dimensions sont variables à l'infini ou presque, c'est la raison d'être de D.FILE qu'il suffit de questionner là où elle réside, à l'adresse 16396. Cette variable nous renvoie aussitôt l'adresse de l'octet qui suit immédiatement le dernier octet du listing. Il suffit donc de décrémenter cette valeur pour coïncider avec ce que nous recherchons.

Adresses de départ et de fin ne sont pas suffisantes si l'on désire utiliser la puissante instruction CPDR, lisez : comparaison par bloc avec décrémentation et répétition, c'est en effet l'instruction la mieux adaptée au type de travail que nous souhaitons mettre en œuvre : comparer chacun des octets du listing avec des valeurs types (exemple : "★" codée 23) qui seront remplacées à chaque rencontre par d'autres signes (exemple : 23 par 24, "/"). CPDR effectuera une comparaison de l'octet pointé par HL avec le contenu de l'accumulateur suivi d'une décrémentation automatique de HL ainsi que BC, ces opérations se répèteront jusqu'à ce que BC soit égal à 0 ou que la comparaison (HL)/ACCU se révèle positive. Il sera donc nécessaire par CPDR de placer la valeur correspondant à (D.FILE-16514). C'est le sens de l'instruction SBC HL.BC pour laquelle il est indispensable de remettre préalablement à zéro l'indicateur de report C grâce à l'instruction XOR A.

Au terme de ce module de délimitation du listing nous aurons donc stocké dans la pile :
- au premier niveau le nombre d'octets du listing
- au deuxième niveau (D.FILE-1)
Nous somme prêts désormais à utiliser CPDR.

C'est au niveau des modules de substitution que nous sortons de la pile ces deux valeurs pour les placer sur les paires de registres utilisées par CPDR. HL passe sur BC tandis que (D.FILE-1) retrouve HL. Il suffit maintenant de placer dans l'accumulateur l'opérande qui servira d'élément de comparaison et d'attendre la sortie automatique de l'instruction CPDR, celle-ci signifiant que la recherche a abouti et que l'on a exploré tous les octets voulus. Dans le premier cas, une incrémentation nous repositionnera au niveau de l'octet à modifier tandis qu'une décrémentation, une fois la substitution opérée, réamorcera CPDR au bon endroit.

Vous vous interrogez peut-être sur la présence de CP 30. Cette instruction n'a d'autre utilité que de remettre à 0 l'indicateur Z qui venait d'être positionné. Notez pour finir que la sauvegarde immédiate dans la pile des valeurs de départ de la recherche CPDR permet la réinitialisation facile de cette instruction au début de chaque module de substitution.

Maintenant que vous avez tout compris il me reste à vous souhaiter de faire le plus mauvais usage possible de cette routine.

Bernard Guyot

```
LISTING ASSMEBLEUR ZX 81

REM * TRUCAGE LISTING

REM * VERIFICATION FEU

    LD A.(16417)    sauvegardable
    LD B.A
    LD A.100        vert
    CP B            feu vert?
    RET Z           oui: retour au basic

REM * DELIMITATION LISTING

:L0 LD HL.(16396)   D.FILE
    DEC HL
    PUSH HL         fin de listing
    XOR A           met l'indicateur
    LD BC.16514     C à 0 pour SBC
    SBC HL.BC
    PUSH HL         espace listing

REM * SUBSTITUTION * ---) /

    POP BC          inverse BC
    POP HL          et HL...
    PUSH HL         replacés dans
    PUSH BC         la pile
    LD A.23         *
:L2 CPDR
    JR NZ.L1
    INC HL
    LD (HL).24
    DEC HL
    CP 30           met Z à 0
    JR L2

REM * SUBSTITUTION + ---) -

:L1 POP BC          on reprend
    POP HL          au début
    PUSH HL
    PUSH BC
    LD A.21         +
:L3 CPDR
    JR NZ.L4
    INC HL
    LD (HL).22      -
    DEC HL
    CP 30
    JR L3

REM * SUBSTITUTION ) ---) (

:L4 POP BC
    POP HL
    LD A.18         )
:L5 CPDR
    JR NZ.L6
    INC HL
    LD (HL).19      (
    DEC HL
    CP 30
    JR L5

:L6 RET
```

LANGAGE MACHINE SUR COMMODORE 64

Comment gagner un séjour en prison grâce à HEBDOGICIEL ? ? (Grand jeu gratuit sans obligation d'achat)

Pour profiter de cette offre fantastique, c'est vraiment très simple ! Suivez les 5 étapes suivantes dans le bon ordre et sans vous perdre.

1- Munissez-vous d'un Commodore 64.
2- Reliez ce CBM à un ampli de 100 watts minimum et ce dernier à une enceinte de qualité supérieure.
3- Tapez du bout de vos petits doigts gras une des routines suivantes.
4- A 22H01 mettez l'ampli sous tension, et à l'aide d'un JMP C000 lancez cette merveilleuse routine sus-citée.
5- Quelques minutes plus tard, un représentant des forces de l'ordre déposera son index sur le bouton-poussoir de votre sonnette d'entrée.

Inculpation pour tapage nocturne, merci HHHH...

Ce cours sera beaucoup plus pratique que théorique, mais je vous promets que celui du mois dernier n'avait pratiquement pas de listings, que des textes !
(Voir première colonne, trente sixième ligne du dernier article, c'est pratique de pomper ses propres phrases. Quel manque d'inspiration !)
Voici des routines bruyantes pour votre instrument de silicicum, le fameux, l'incomparable, l'inénarrable 6581. Mais avant toute chose, l'initialisation de ce circuit doit être effectuée. Ce n'est pas très compliqué, la mise a zéro de ses bits est suffisante.
Voici un subtil exemple de programme remédiant à ce douloureux problème.

```
C000 A2 18     LDX #$18
C002 A9 00     LDA #$00
C004 9D 00 D4  STA $D400,X
C007 CA        DEX
C008 D0 FA     BNE $C004
```

Donc avant chaque exécution de l'une des routines suivantes, ce programme doit être appelé, c'est pourquoi il est placé de C000 à C00A, et que toutes les autres (les huit restant en attente, à la suite de ce qui précède et ne voulant pas s'impatienter plus longtemps toutes seules dans leur coin, hurlent chacune plus fort que sa voisine pour attirer votre attention, cher et fidèle lecteur) débutent en C00A. (A relire plusieurs fois)

Les hurlements de vos voisins vont commencer.

```
C00A A9 1E     LDA #$1E
C00C 8D 01 D4  STA $D401
C00F A9 07     LDA #$07
C011 8D 05 D4  STA $D405
C014 A9 32     LDA #$32
C016 8D 0F D4  STA $D40F
C019 A9 0F     LDA #$0F
C01B 8D 18 D4  STA $D418
C01E A9 13     LDA #$13
C020 8D 04 D4  STA $D404
C023 A9 05     LDA #$05
C025 A2 00     LDX #$00
C027 8E 0F D4  STX $D40F
C02A 8E 04 D4  STX $D404
C02D E8        INX
C02E A0 FF     LDY #$FF
C030 88        DEY
C031 D0 FD     BNE $C030
C033 E0 FF     CPX #$FF
C035 D0 F0     BNE $C027
C037 E9 01     SBC #$01
C039 D0 EA     BNE $C025
C03B 60        RTS
```

Vous êtes déjà de retour de cette belle prison ? ! ? Alors comment était le voyage ?

```
C00A A9 05     LDA #$05
C00C 8D 0E D4  STA $D40E
C00F A9 10     LDA #$10
C011 8D 12 D4  STA $D412
C014 A9 0A     LDA #$0A
C016 8D 03 D4  STA $D403
C019 A9 8F     LDA #$8F
C01B 8D 18 D4  STA $D418
C01E A9 F0     LDA #$F0
C020 8D 06 D4  STA $D406
C023 A9 41     LDA #$41
C025 8D 04 D4  STA $D404
C028 A9 01     LDA #$01
C02A 85 A2     STA $A2
C02C 85 A1     STA $A1
C02E A5 A2     LDA $A2
C030 8D 00 D4  STA $D400
C033 A5 A1     LDA $A1
C035 8D 01 D4  STA $D401
C038 C9 08     CMP #$08
C03A D0 F2     BNE $C02E
C03C A2 18     LDX #$18
C03E A9 00     LDA #$00
C040 9D 00 D4  STA $D400,X
C043 CA        DEX
C044 D0 FA     BNE $C040
C046 60        RTS
```

Cette fois-ci, il y a récidive, vous avez donc prolongé vos vacances, et pas de sursis.

```
C00A A9 F0     LDA #$F0
C00C 8D 00 D4  STA $D400
C00F A9 21     LDA #$21
C011 8D 01 D4  STA $D401
C014 A9 08     LDA #$08
C016 8D 05 D4  STA $D405
C019 A9 68     LDA #$68
C01B 8D 16 D4  STA $D416
C01E A9 01     LDA #$01
C020 8D 17 D4  STA $D417
C023 A9 4F     LDA #$4F
C025 8D 18 D4  STA $D418
C028 A2 20     LDX #$20
C02A A9 81     LDA #$81
C02C 8D 04 D4  STA $D404
C02F A9 00     LDA #$00
C031 85 A2     STA $A2
C033 A5 A2     LDA $A2
C035 C9 05     CMP #$05
C037 D0 FA     BNE $C033
C039 A9 80     LDA #$80
C03B 8D 04 D4  STA $D404
C03E A9 00     LDA #$00
C040 85 A2     STA $A2
C042 A5 A2     LDA $A2
C044 C9 02     CMP #$02
C046 D0 FA     BNE $C042
C048 CA        DEX
C049 D0 DF     BNE $C02A
C04B 60        RTS
```

Maintenant c'est terminé, la prison à vie. Emmenez donc votre ordinateur et abonnez-vous à l'HHHHebdo (publicité très mal déguisée, la salope) !

```
C00A A9 82     LDA #$82
C00C 8D 01 D4  STA $D401
C00F A9 07     LDA #$07
C011 8D 05 D4  STA $D405
C014 A9 1E     LDA #$1E
C016 8D 0F D4  STA $D40F
C019 A9 0F     LDA #$0F
C01B 8D 18 D4  STA $D418
C01E A2 0B     LDX #$0B
C020 A9 15     LDA #$15
C022 8D 04 D4  STA $D404
C025 20 35 C0  JSR $C035
C028 A9 14     LDA #$14
C02A 8D 04 D4  STA $D404
C02D 20 35 C0  JSR $C035
C030 CA        DEX
C031 D0 ED     BNE $C020
C033 60        RTS
C034 EA        NOP
C035 A9 00     LDA #$00
C037 85 A2     STA $A2
C039 A5 A2     LDA $A2
C03B C9 07     CMP #$07
C03D D0 FA     BNE $C039
C03F 60        RTS
```

Sans commentaire.


```
C00A A6 C5     LDX $C5
C00C E0 40     CPX #$40
C00E D0 FA     BNE $C00A
C010 A9 0F     LDA #$0F
C012 8D 05 D4  STA $D405
C015 A9 00     LDA #$00
C017 8D 06 D4  STA $D406
C01A A9 0F     LDA #$0F
C01C 8D 18 D4  STA $D418
C01F A9 11     LDA #$11
C021 8D 04 D4  STA $D404
C024 AD 12 D0  LDA $D012
C027 8D 01 D4  STA $D401
C02A A6 C5     LDX $C5
C02C E0 40     CPX #$40
C02E F0 F4     BEQ $C024
C030 A9 00     LDA #$00
C032 8D 18 D4  STA $D418
C035 60        RTS
```

Grâce à ce merveilleux son, vous avez droit au quartier de haute sécurité (QHS pour tous les diplômés de Fleury et autre Fresnes).

```
C00A A9 0F     LDA #$0F
C00C 8D 05 D4  STA $D405
C00F 8D 0C D4  STA $D40C
C012 A9 00     LDA #$00
C014 8D 06 D4  STA $D406
C017 8D 0D D4  STA $D40D
C01A A9 0F     LDA #$0F
C01C 8D 18 D4  STA $D418
C01F A9 21     LDA #$21
C021 8D 04 D4  STA $D404
C024 A9 25     LDA #$25
C026 8D 0B D4  STA $D40B
C029 A9 10     LDA #$10
C02B 8D 01 D4  STA $D401
C02E 8D 08 D4  STA $D408
C031 60        RTS
```

Le directeur vous a confisqué votre micro (mais il vous a laissé la télé, sympa le mec, non ?).

```
C00A A9 0F     LDA #$0F
C00C 8D 05 D4  STA $D405
C00F 8D 0C D4  STA $D40C
C012 8D 13 D4  STA $D413
C015 A9 00     LDA #$00
C017 8D 06 D4  STA $D406
C01A 8D 0D D4  STA $D40D
C01D 8D 14 D4  STA $D414
C020 A9 0F     LDA #$0F
C022 8D 18 D4  STA $D418
C025 A9 25     LDA #$25
C027 8D 04 D4  STA $D404
C02A 8D 0B D4  STA $D40B
C02D 8D 12 D4  STA $D412
C030 A9 20     LDA #$20
C032 8D 00 D4  STA $D400
C035 A9 60     LDA #$60
C037 8D 07 D4  STA $D407
C03A A9 A0     LDA #$A0
C03C 8D 0E D4  STA $D40E
C03F A9 10     LDA #$10
C041 8D 01 D4  STA $D401
C044 8D 08 D4  STA $D408
C047 8D 0F D4  STA $D40F
C04A 60        RTS
```

Et ...

```
C00A A9 0F     LDA #$0F
C00C 8D 05 D4  STA $D405
C00F 8D 0C D4  STA $D40C
C012 A9 00     LDA #$00
C014 8D 06 D4  STA $D406
C017 8D 0D D4  STA $D40D
C01A A9 0F     LDA #$0F
C01C 8D 18 D4  STA $D418
C01F A9 21     LDA #$21
C021 8D 04 D4  STA $D404
C024 A9 25     LDA #$25
C026 8D 0B D4  STA $D40B
C029 A0 80     LDY #$80
C02B B9 00 A0  LDA $A000,Y
C02E 8D 01 D4  STA $D401
C031 B9 00 B0  LDA $B000,Y
C034 8D 08 D4  STA $D408
C037 A9 00     LDA #$00
C039 85 A2     STA $A2
C03B A5 A2     LDA $A2
C03D C9 03     CMP #$03
C03F D0 FA     BNE $C03B
C041 88        DEY
C042 D0 E7     BNE $C02B
C044 A9 00     LDA #$00
C046 8D 18 D4  STA $D418
C049 60        RTS
```

Quelques commentaires pour ce gentil directeur de cette belle prison (mon papa est policier, ma maman contractuelle, ça explique tout)... (ceci n'est qu'une honteuse note d'humour, je tiens à le préciser !)

- Exécution des programmes : JMP C000
- Vous retournez à l'assembleur automatiquement après un certain temps (qui ne durera jamais plus d'un temps certain, c'est certain car c'est calculé) correspondant à l'émission des sons, sauf dans le cas du programme 6 au cours duquel vous devrez appuyer sur une touche pour interrompre le déroulement (c'est l'exemple classique de l'exception qui confirme la règle).

Pour toute réclamation, veuillez joindre votre numéro de cellule et de costume rayé. Entonnons encore une fois le chant de libération des commodoristes assoiffés de bruitages et à dans quatre semaines, le maton arrive !

Sébastien Mougey

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130

COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130

ORIC -» 57 62 67 72 77 82 87 92 96

101 106 111 115 119 123 128 131

AMSTRAD -» 111 115 119 123 127 131

APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132

SPECTRUM -» 112 116 120 124 127 132

THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133

MSX -» 113 117 121 125 129 133

LANGAGE MACHINE SUR COMMODORE 64

Pas d'introduction extraordinaire aujourd'hui !
Le simple cours, normal, habituel, routinier, sans intérêt, monotone, ... Cependant la nouveauté du jour est : un JEU !
Après avoir passé en revue les lutins (vive la France), l'écran, les sons, et tout le Commodore piste par piste, résistance par résistance, CI par CI, voici une application toute béèète... Pour débuter ce jeu, voici les dessins composant notre merveilleux mercenaire, aventurier ou tout simplement personnage.
Il est constitué de deux sprites, le premier tout bleu, pour le corps, et le second, tout rose, pour la tête et les jambes... Heuuu pardon, les mains.
Donc à chaque affichage de celui-ci, deux sprites seront superposés. Voici les emplacements de ceux-ci :

ADRESSE NO DEFINITION
8192 128 Vue de face - corps
8256 129 Vue de face - tête
8320 130 Vue de gauche - corps
8384 131 Vue de gauche - tête
8448 132 Vue de gauche - 2e corps
8512 133 Vue de gauche - 2e tête
8576 134 Vue de droite - corps
8640 135 Vue de droite - tête
8704 136 Vue de droite - 2e corps
8768 137 Vue de droite - 2e tête
8832 138 Vue de face - 2e corps
8896 139 Vue de face - 2e tête

Voici les données en hexa.

LISTING 1

```
:2000 00 00 00 00 38 00 00 28
:2008 00 00 00 00 00 00 00 00
:2010 10 00 00 FE 00 00 FE 00
:2018 01 FF 00 01 FF 00 03 7D
:2020 80 03 7D 80 06 7C C0 00
:2028 6C 00 00 6C 00 00 C6 00
:2030 00 C6 00 01 83 00 01 83
:2038 00 03 01 80 03 01 80 00
:2040 00 00 00 00 00 00 00 10
:2048 00 00 7C 00 00 38 00 00
:2050 00 00 00 00 00 00 00 00
:2058 00 00 00 00 00 00 00 00
:2060 00 00 00 00 00 00 00 06
:2068 00 C0 00 00 00 00 00 00
:2070 00 00 00 00 00 00 00 00
:2078 00 00 00 00 00 00 00 00
:2080 00 00 00 00 70 00 00 40
:2088 00 00 00 00 00 00 00 00
:2090 20 00 01 FC 00 03 FC 00
:2098 07 FE 00 0C FE C0 18 FB
:20A0 80 0C F9 00 06 F8 00 00
:20A8 F8 00 00 DC 00 00 C6 00
:20B0 01 83 00 03 01 80 0E 01
:20B8 80 0C 01 80 08 01 C0 00
:20C0 00 00 00 00 00 00 00 30
:20C8 00 00 78 00 00 70 00 00
:20D0 00 00 00 00 00 00 00 00
:20D8 00 00 C0 00 00 00 00 00
:20E0 00 00 00 00 00 00 00 06
:20E8 00 00 00 00 00 00 00 00
:20F0 00 00 00 00 00 00 00 00
:20F8 00 00 00 00 00 00 00 00
:2100 00 00 00 00 70 00 00 40
:2108 00 00 00 00 00 00 00 00
:2110 20 00 01 FC 00 03 FC 00
:2118 03 F8 00 06 F8 00 06 FC
:2120 00 03 FE 00 01 F8 00 00
:2128 F8 00 00 D8 00 00 D8 00
:2130 00 D8 00 01 98 00 01 98
:2138 00 03 18 00 03 9C 00 00
:2140 00 00 00 00 00 00 00 30
:2148 00 00 78 00 00 70 00 00
:2150 00 00 00 00 00 00 00 00
:2158 00 00 00 00 00 00 00 00
:2160 00 00 00 00 00 03 00 00
:2168 00 00 00 00 00 00 00 00
:2170 00 00 00 00 00 00 00 00
:2178 00 00 00 00 00 00 00 00
:2180 00 00 00 00 70 00 00 10
:2188 00 00 00 00 00 00 00 00
:2190 20 00 01 FC 00 01 FE 00
:2198 03 FF 00 1B F9 80 0E F8
:21A0 C0 04 F9 80 00 FB 00 00
:21A8 F8 00 01 D8 00 03 18 00
:21B0 06 0C 00 0C 06 00 0C 03
:21B8 80 0C 01 80 1C 00 80 00
:21C0 00 00 00 00 00 00 00 60
:21C8 00 00 F0 00 00 70 00 00
:21D0 00 00 00 00 00 00 00 00
:21D8 18 00 00 00 00 00 00 00
:21E0 00 00 00 00 00 00 00 00
:21E8 03 00 00 00 00 00 00 00
:21F0 00 00 00 00 00 00 00 00
:21F8 00 00 00 00 00 00 00 00
:2200 00 00 00 00 70 00 00 10
:2208 00 00 00 00 00 00 00 00
:2210 20 00 01 FC 00 01 FE 00
:2218 00 FE 00 00 FB 00 01 FB
:2220 00 03 FE 00 00 FC 00 00
:2228 F8 00 00 D8 00 00 D8 00
:2230 00 D8 00 00 CC 00 00 CC
:2238 00 00 C6 00 01 CE 00 00
:2240 00 00 00 00 00 00 00 60
:2248 00 00 F0 00 00 70 00 00
:2250 00 00 00 00 00 00 00 00
:2258 00 00 00 00 00 00 00 00
:2260 00 00 00 00 06 00 00 00
:2268 00 00 00 00 00 00 00 00
:2270 00 00 00 00 00 00 00 00
:2278 00 00 00 00 00 00 00 00
:2280 00 00 00 00 38 00 00 28
:2288 00 00 00 00 00 00 00 00
:2290 10 00 00 7E 00 00 7F 00
:2298 07 5D 80 03 5C C0 01 9D
:22A0 80 00 D8 00 00 3C 00 00
:22A8 7C 00 00 6C 00 00 C6 00
:22B0 00 C6 00 01 83 00 01 83
:22B8 00 03 01 80 03 01 80 00
:22C0 00 00 00 00 00 00 00 10
:22C8 00 00 7C 00 00 38 00 00
:22D0 00 00 02 00 00 07 00 00
:22D8 00 00 00 00 00 00 00 00
:22E0 00 00 06 00 00 00 00 00
:22E8 00 00 00 00 00 00 00 00
:22F0 00 00 00 00 00 00 00 00
:22F8 00 00 00 00 00 00 00 00
:2300 2D FF 00 E1 00 FF 0D FF
```

Et maintenant en DATAs.

LISTING 2

```
1 FORF=8192TO8959:READA:POKEF,A:NE
XT
10 DATA 0,0,0,0,56,0,0,40,0,0,0,0,
0,0,0,0
11 DATA 16,0,0,254,0,0,254,0,1,255
,0,1,255,0,3,125
12 DATA 128,3,125,128,6,124,192,0,
108,0,0,108,0,0,198,0
13 DATA 0,198,0,1,131,0,1,131,0,3,
1,128,3,1,128,0
14 DATA 0,0,0,0,0,0,0,16,0,0,124,0
,0,56,0,0
15 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
16 DATA 0,0,0,0,0,0,6,0,192,0,0,
0,0,0,0
17 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
18 DATA 0,0,0,0,112,0,0,64,0,0,0,0
,0,0,0,0
19 DATA 32,0,1,252,0,3,252,0,7,254
,0,12,254,192,24,251
20 DATA 128,12,249,0,6,248,0,0,248
,0,0,220,0,0,198,0
21 DATA 1,131,0,3,1,128,14,1,128,1
2,1,128,8,1,192,0
22 DATA 0,0,0,0,0,0,0,48,0,0,120,0
,0,112,0,0
23 DATA 0,0,0,0,0,0,0,0,0,0,192,0,
0,0,0,0
24 DATA 0,0,0,0,0,0,0,6,0,0,0,0,0,
0,0,0
25 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
26 DATA 0,0,0,0,112,0,0,64,0,0,0,0
,0,0,0,0
27 DATA 32,0,1,252,0,3,252,0,3,248
,0,6,248,0,6,252
28 DATA 0,3,254,0,1,248,0,0,248,0,
0,216,0,0,216,0
29 DATA 0,216,0,1,152,0,1,152,0,3,
24,0,3,156,0,0
30 DATA 0,0,0,0,0,0,0,48,0,0,120,0
,0,112,0,0
31 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
32 DATA 0,0,0,0,0,3,0,0,0,0,0,0,0,
0,0,0
33 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
34 DATA 0,0,0,0,112,0,0,16,0,0,0,0
,0,0,0,0
35 DATA 32,0,1,252,0,1,254,0,3,255
,0,27,249,128,14,248
36 DATA 192,4,249,128,0,251,0,0,24
8,0,1,216,0,3,24,0
37 DATA 6,12,0,12,6,0,12,3,128,12,
1,128,28,0,128,0
38 DATA 0,0,0,0,0,0,0,96,0,0,240,0
,0,112,0,0
39 DATA 0,0,0,0,0,0,0,24,0,0,0,0,0
,0,0,0
40 DATA 0,0,0,0,0,0,0,0,3,0,0,0,0,
0,0,0
41 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
42 DATA 0,0,0,0,112,0,0,16,0,0,0,0
,0,0,0,0
43 DATA 32,0,1,252,0,1,254,0,0,254
,0,0,251,0,1,251
44 DATA 0,3,254,0,0,252,0,0,248,0,
0,216,0,0,216,0
45 DATA 0,216,0,0,204,0,0,204,0,0,
198,0,1,206,0,0
46 DATA 0,0,0,0,0,0,0,96,0,0,240,0
,0,112,0,0
47 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
48 DATA 0,0,0,0,6,0,0,0,0,0,0,0,0,
0,0,0
49 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
50 DATA 0,0,0,0,56,0,0,40,0,0,0,0,
0,0,0,0
51 DATA 16,0,0,126,0,0,127,0,7,93,
128,3,92,192,1,157
52 DATA 128,0,216,0,0,60,0,0,124,0
,0,108,0,0,198,0
53 DATA 0,198,0,1,131,0,1,131,0,3,
1,128,3,1,128,0
54 DATA 0,0,0,0,0,0,0,16,0,0,124,0
,2,56,0,2
55 DATA 0,0,2,0,0,7,0,0,0,0,0,0,0,
0,0,0
56 DATA 0,0,6,0,0,0,0,0,0,0,0,0,0,
0,0,0
57 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
58 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0
```

Un petit programme très simple de traitement (moins complexe que celui du jeu).

LISTING 3

```
C100 A2 00    LDX #$00
C102 86 FF    STX $FF
C104 AD 00 DC LDA $DC00
C107 29 10    AND #$10
C109 D0 04    BNE $C10F
C10B A2 01    LDX #$01
C10D 86 FF    STX $FF
C10F AD 00 DC LDA $DC00
C112 29 0F    AND #$0F
C114 C9 0F    CMP #$0F
C116 D0 0B    BNE $C123
C118 A2 80    LDX #$80
C11A 8E F8 07 STX $07F8
C11D A2 81    LDX #$81
C11F 8E F9 07 STX $07F9
C122 60       RTS
C123 4A       LSR
C124 B0 0D    BCS $C133
C126 AC 01 D0 LDY $D001
C129 C0 31    CPY #$31
C12B F0 06    BEQ $C133
C12D CE 01 D0 DEC $D001
C130 CE 03 D0 DEC $D003
C133 4A       LSR
C134 B0 0D    BCS $C143
C136 AC 01 D0 LDY $D001
C139 C0 E5    CPY #$E5
C13B F0 06    BEQ $C143
C13D EE 01 D0 INC $D001
C140 EE 03 D0 INC $D003
C143 4A       LSR
C144 B0 2A    BCS $C170
C146 AC 00 D0 LDY $D000
C149 C0 15    CPY #$15
C14B F0 23    BEQ $C170
C14D CE 00 D0 DEC $D000
C150 CE 02 D0 DEC $D002
C153 A5 A2    LDA $A2
C155 29 08    AND #$08
C157 D0 0D    BNE $C166
C159 A9 86    LDA #$86
C15B 8D F8 07 STA $07F8
C15E A9 87    LDA #$87
C160 8D F9 07 STA $07F9
C163 4C 70 C1 JMP $C170
C166 A9 88    LDA #$88
C168 8D F8 07 STA $07F8
C16B A9 89    LDA #$89
C16D 8D F9 07 STA $07F9
C170 4A       LSR
C171 B0 2A    BCS $C19D
C173 AC 00 D0 LDY $D000
C176 C0 FF    CPY #$FF
C178 F0 23    BEQ $C19D
C17A EE 00 D0 INC $D000
C17D EE 02 D0 INC $D002
C180 A5 A2    LDA $A2
C182 29 08    AND #$08
C184 D0 0D    BNE $C193
C186 A9 82    LDA #$82
C188 8D F8 07 STA $07F8
C18B A9 83    LDA #$83
C18D 8D F9 07 STA $07F9
C190 4C 9D C1 JMP $C19D
C193 A9 84    LDA #$84
C195 8D F8 07 STA $07F8
C198 A9 85    LDA #$85
C19A 8D F9 07 STA $07F9
C19D 60       RTS
```

Mais il ne fait que diriger les sprites, il faut avant tout les initialiser.

LISTING 4

```
C000 A5 C5    LDA $C5
C002 C9 40    CMP #$40
C004 D0 FA    BNE $C000
C006 A9 03    LDA #$03
C008 8D 15 D0 STA $D015
C00B A9 00    LDA #$00
C00D 8D 20 D0 STA $D020
C010 8D 21 D0 STA $D021
C013 A9 06    LDA #$06
C015 8D 27 D0 STA $D027
C018 A9 0A    LDA #$0A
C01A 8D 28 D0 STA $D028
C01D A9 50    LDA #$50
C01F 8D 00 D0 STA $D000
C022 8D 01 D0 STA $D001
C025 8D 02 D0 STA $D002
C028 8D 03 D0 STA $D003
C02B EA       NOP
C02C EA       NOP
C02D EA       NOP
C02E EA       NOP
C02F EA       NOP
C030 20 00 C1 JSR $C100
C033 A5 FF    LDA $FF
C035 F0 0A    BEQ $C041
C037 A9 8A    LDA #$8A
C039 8D F8 07 STA $07F8
C03C A9 8B    LDA #$8B
C03E 8D F9 07 STA $07F9
C041 A5 A2    LDA $A2
C043 C5 A2    CMP $A2
C045 F0 FC    BEQ $C043
C047 A5 C5    LDA $C5
C049 C9 40    CMP #$40
C04B F0 E3    BEQ $C030
C04D 60       RTS
```

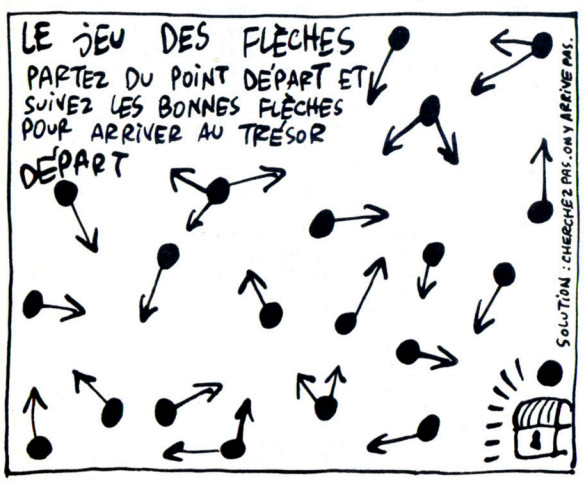
Dans la routine de déplacement, à chaque action sur le joy 2 (droite ou gauche), les coordonnées des sprites 1 & 2 sont modifiées, ainsi que l'adresse à laquelle se trouvent leurs dessins, et comme les mouvements sont décomposés en deux fois deux dessins (tête et corps, 2e tête et 2e corps), l'impression de mouvement peut paraître plus réaliste.
Maintenant, appuyez sur le bouton de tir : surprise ! ! ! (interdit au moins de 18 ans).

La prochaine fois, la véritable routine de déplacement...

Sébastien Mougey

LANGAGE MACHINE SUR ZX 81

L'idée de doter le ZX 81 d'une puissante paire de mâchoires ne me serait pas venue à l'esprit sans un étrange concours de circonstances que je m'en vais ici vous narrer succinctement. Savez-vous que l'hiver 85 fût particulièrement rigoureux à Oulan Bator, capitale de la Mongolie centrale, cher pays où je vis le jour avec, tradition locale, un chromosome surnuméraire. Fort heureusement je n'y étais plus cette année-là, ayant fuit deux ans auparavant le désert de Gobi où j'avais des difficultés à m'approvisionner en rouleaux de papier pour mon imprimante Sinclair. Je m'étais établi à Paris, capitale qui, elle aussi, dut subir les assauts du gel et de la merde.
Souvenez-vous de toute cette tuyauterie en fonte-garantie-à-vie qui, fissure après fissure, laissait échapper sur les façades déjà dégradées de nos immeubles les miasmes rénaux et intestinaux d'une population désemparée, calfeutrée dans la pièce la plus petite et la mieux chauffée et qui trop souvent se disputait le privilège du seul siège disponible. Il n'y avait pas d'autre échappatoire et il fallait bien passer le temps sans prendre autant de place qu'à la pétanque et plus intelligemment qu'au tarot. Dans de telles circonstances les inhibitions scatologiques des constipés, refoulés et autres névrosés disparaissent plus efficacement qu'après quinze ans de psychanalyse et le moindre avorton parisien s'offre à peu de frais le luxe de marquer son territoire de son odeur à l'instar des grands fauves noirs des savanes africaines !



Mais ne nous laissons pas gagner par un lyrisme trop facile en matière fécale, et revenons à une approche plus prosaïque de cette réalité si dure quand il gèle !
J'habitais à l'époque dans une studette au septième étage d'un immeuble cossu accessible uniquement par l'escalier de service encore encombré des cadavres des boniches du siècle précédent. On ne comprenait pas, spontanément, pourquoi au centre de mon unique pièce se trouvait une cuvette de WC d'une propreté irréprochable. C'est simplement que j'étais très à cheval sur la propreté des cuvettes de WC et aussi parce que j'avais tellement bien réaménagé ce qui était autrefois les toilettes de l'étage qu'en dehors de l'exiguité des lieux, des bruits intempestifs de la chasse d'eau et surtout du petit verrou de type "occupé ou libre" on ne pouvait se douter de rien. C'est là que je coulais des jours paisibles dans le relâchement total et permanent de tous mes sphincters, me livrant à mes deux activités favorites : noircir le papier en rouleau de mon distributeur et pianoter sur mon ZX entre deux faux-contacts.

Avec l'arrivée du gel les choses allaient pour moi se gâter considérablement et j'envisageais alors de demander le secours de mon ZX à travers l'élaboration d'un logiciel spécifique de lutte contre l'obstruction des tuyaux de vidange. Programme ambitieux qui, alliant robotique et informatique, devait aboutir à la création de puissantes mâchoires dévoratrices brisant la glace et avalant la merde. Cette géniale invention coprophage connût parmi ses plus beaux prolongements industriels nos splendides motos vertes parisiennes, souvent chevauchées par de romantiques Walkyries dont les blonds cheveux flottant aux gaz d'échappement laissent dans leur sillage une odeur insoutenable qui ne permet plus à personne de douter de l'efficacité du dispositif dont elles ont la glorieuse charge.

Voici donc la version logicielle de ce système :

```
Fichier: ZXLIST

REM * MACHOIRES HORIZONTALES
REM * FERMETURE MACHOIRES
      LD HL,(16396)   D-FILE
      INC HL
      PUSH HL         départ haut
      LD DE,790
      ADD HL,DE       départ bas
      LD C,12         compteur lignes
      LD A,(16444)    prend la couleur
:L10  LD B,32         compteur colonnes
:L11  LD (HL),A       coloration lignes
      DEC HL
      DJNZ,L11
      DEC HL          saute le 118
      POP DE          haut
      PUSH HL         bas
      LD B,32         compteur colonnes
:L12  LD (DE),A       coloration haut
      INC DE          en avancant
      DJNZ,L12
      INC DE          saute le 118
      DEC C           compteur lignes
      JR Z,L13
      POP HL          bas
      PUSH DE         haut
      LD D,10
:L14  LD E,255
:L15  DEC E
      JR NZ,L15       temporisation
      DEC D
      JR NZ,L14
      JR L10
:L13  POP BC          défausse pile
      LD D,45
:L50  LD E,255
:L51  DEC E
      JR NZ,L51       temporisation
      DEC D
      JR NZ,L50
      RET

REM * OUVERTURE MACHOIRES HORIZONT.
      LD A,(16417)
      LD B,A          feu vert
      LD A,100
      CP B
      JR Z,L55
:L54  LD B,90         si feu rouge...
      JR L54          boucle sans fin'
:L55  LD HL,(16396)   D-FILE
      LD DE,395       moitié d'écran
      ADD HL,DE
      PUSH HL         départ haut
      LD DE,2
      ADD HL,DE       départ bas
      LD C,12         compteur lignes
      LD A,(16444)    couleur effacement
:L0   LD B,32         compteur colonnes
:L1   LD (HL),A       effacement bas
      INC HL          en avancant
      DJNZ,L1
      INC HL          saute 118
      POP DE          haut
      PUSH HL         bas
      LD B,32         compteur colonnes
:L2   LD (DE),A       effacement haut
      DEC DE          en reculant
      DJNZ,L2
      DEC DE          saute 118
      DEC C           compteur lignes
      JR Z,L3
      POP HL          bas
      PUSH DE         haut
      LD D,10
:L4   LD E,255
:L5   DEC E
      JR NZ,L5        temporisation
      DEC D
      JR NZ,L4
      JR L0
:L3   POP BC          défausse pile
      RET

REM * MACHOIRES VERTICALES
REM * FERMETURE
      LD HL,(16396)   D-FILE
      INC HL
      PUSH HL         départ gauche
      LD DE,31
      ADD HL,DE
      PUSH HL         départ droite
      LD C,16         compteur colonnes
      LD A,(16445)    prend la couleur
:L20  LD (HL),A       coloration
      LD B,23         compteur lignes
      LD DE,33        pour obtenir...
:L21  ADD HL,DE       la case en-dessous
      LD (HL),A       coloration
      DJNZ,L21
      POP DE          décalage du...
      DEC DE          départ à droite
      POP HL          départ gauche
      PUSH DE         départ droite
      PUSH HL         départ gauche
      LD (HL),A       coloration
      LD B,23         compteur lignes
      LD DE,33        pour obtenir...
:L22  ADD HL,DE       case en-dessous
      LD (HL),A       coloration
      DJNZ,L22
      POP DE          décalage du...
      INC DE          départ à gauche
      POP HL          départ à droite
      PUSH DE         à gauche
      PUSH HL         à droite
      LD D,10
:L23  LD E,255
:L24  DEC E
      JR NZ,L24       temporisation
      DEC D
      JR NZ,L23
      DEC C           compteur colonnes
      JR NZ,L20
      POP HL          défausse de...
      POP DE          la pile
      LD D,40
:L52  LD E,255
:L53  DEC E
      JR NZ,L53       temporisation
      DEC D
      JR NZ,L52
      RET

REM * OUVERTURE
      LD A,(16417)
      LD B,A          feu vert
      CP B
      JR Z,L57
:L56  LD C,A          si rouge...
      JR L56          boucle sans fin'
:L57  LD HL,(16396)   D-FILE
      LD DE,16        facteur divis. vert.
      ADD HL,DE       départ à gauche...
      PUSH HL         16* colonne
      INC HL          départ à droite...
      PUSH HL         17*colonne
      LD C,16         compteur colonnes
      LD A,(16445)    prend la couleur
:L35  LD (HL),A       coloration
      LD B,23         compteur lignes
      LD DE,33        pour obtenir la...
:L36  ADD HL,DE       case en-dessous
      LD (HL),A       coloration
      DJNZ,L36
      POP DE          décalage départ..
      INC DE          à droite
      POP HL          départ à gauche
      PUSH DE         départ à droite
      PUSH HL         à gauche
      LD (HL),A       coloration
      LD B,23         compteur lignes
      LD DE,33        pour obtenir la..
:L37  ADD HL,DE       case en-dessous
      LD (HL),A       coloration
      DJNZ,L37
      POP DE          décalage du...
      DEC DE          départ à gauche
      POP HL          droite
      PUSH DE         gauche
      PUSH HL         droite
      LD D,10
:L38  LD E,255
:L39  DEC E
      JR NZ,L39       temporisation
      DEC D
      JR NZ,L38
      DEC C           compteur colonnes
      JR NZ,L35
      POP HL          défausse de...
      POP DE          la pile
      RET
```

Pour les MACHOIRES HORIZONTALES on établit deux points de départ de tracés, l'un en haut à gauche de l'écran, l'autre en bas à droite. La couleur de la mâchoire est stockée en 16444 (buffer de l'imprimante). Le tracé du haut s'effectue sur une ligne par incrémentations, puis c'est ensuite à celui du bas sur une ligne par décrémentations successives. Après une nécessaire temporisation on renouvelle quinze fois l'opération de façon à recouvrir la totalité de l'écran. Lorsqu'une ligne est tracée on conserve dans la pile la prochaine adresse de départ.

Pour l'OUVERTURE HORIZONTALE le schéma est à peu près le même mais les adresses de départ, cette fois au milieu de l'écran, diffèrent. A noter que "l'ouverture" ne se produira que si le "feu est mis au vert" (octet 16417 mis à la valeur 100), ceci au cas où vous souhaiteriez pièger vos programmes.
Pour la FERMETURE VERTICALE le tracé s'effectue colonne après colonne par addition du facteur 33, départ à droite.
L'OUVERTURE VERTICALE est soumise elle aussi au "feu vert".

Que ce soit pour l'horizontale ou la verticale, l'effet "mâchoires" est très élégant. Plus encore si vous parvenez à les faire sourire...

Bernard Guyot

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130 134
COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130 134
ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128 131 136
AMSTRAD -» 111 115 119 123 127 131 135
APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132 135
SPECTRUM -» 112 116 120 124 127 132 136
THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133 137
MSX -» 113 117 121 125 129 133 137

LANGAGE MACHINE SUR COMMODORE 64

Suite du petit héros, alias sprite. Il faut maintenant le faire sauter en haut et en bas de l'écran.

8960 140 Vu de face, accroupi - corps
9024 141 Vu de face, accroupi - tête
9088142 Saut - corps
9152 143 Saut - tête
9216 144 Vu de face, accroupi - 2e corps
9280 145 Vu de face, accroupi - 2e tête
9344 146 Chute - corps
9408 147 Chute - tête

Les petits dessins :

LISTING 1

```
:2300 00 00 00 00 00 00 00 00'
:2308 00 00 00 00 00 00 00 00'
:2310 00 00 00 38 00 00 28 00'
:2318 00 00 00 00 00 00 00 10'
:2320 00 00 FE 00 01 FF 00 03'
:2328 7D 80 96 38 C0 0C 38 60'
:2330 01 BB 00 00 EE 00 00 44'
:2338 00 00 6C 00 00 EE 00 00'
:2340 00 00 00 00 00 00 00 00'
:2348 00 00 00 00 00 00 00 00'
:2350 00 00 00 00 00 00 10 00'
:2358 00 7C 00 00 38 00 00 00'
:2360 00 00 00 00 00 00 00 00'
:2368 00 00 00 00 00 00 00 00'
:2370 18 00 30 00 00 00 00 00'
:2378 00 00 00 00 00 00 00 00'
:2380 00 00 00 01 83 00 01 BB'
:2388 00 01 AB 00 01 83 00 01'
:2390 83 00 01 93 00 00 FE 00'
:2398 00 FE 00 00 7C 00 00 7C'
:23A0 00 00 7C 00 00 7C 00 00'
:23A8 6C 00 00 6C 00 00 C6 00'
:23B0 00 C6 00 00 C6 00 01 83'
:23B8 00 01 83 00 01 83 00 00'
:23C0 01 83 00 00 00 00 00 00'
:23C8 00 00 10 00 00 7C 00 00'
:23D0 38 00 00 00 00 00 00 00'
:23D8 00 00 00 00 00 00 00 00'
:23E0 00 00 00 00 00 00 00 00'
:23E8 00 00 00 00 00 00 00 00'
:23F0 00 00 00 00 00 00 00 00'
:23F8 00 00 00 00 00 00 00 00'
:2400 00 00 00 00 00 00 00 00'
:2408 00 00 00 00 00 00 00 00'
:2410 00 00 00 38 00 03 29 80'
:2418 03 01 80 01 83 00 01 93'
:2420 00 00 FE 00 00 7C 00 00'
:2428 7C 00 00 38 00 00 38 00'
:2430 01 BB 00 00 EE 00 00 44'
:2438 00 00 6C 00 00 EE 00 00'
:2440 00 00 00 00 00 00 00 00'
:2448 00 00 00 00 00 00 00 00'
:2450 00 00 03 01 80 00 10 00'
:2458 00 7C 00 00 38 00 00 00'
:2460 00 00 00 00 00 00 00 00'
:2468 00 00 00 00 00 00 00 00'
:2470 00 00 00 00 00 00 00 00'
:2478 00 00 00 00 00 00 00 00'
:2480 00 00 00 06 38 C0 06 28'
:2488 C0 03 01 80 03 01 80 01'
:2490 93 00 01 D7 00 00 FE 00'
:2498 00 FE 00 00 7C 00 00 7C'
:24A0 00 00 00 7C 00 00 7C 00'
:24A8 6C 00 00 6C 00 00 C6 00'
:24B0 00 C6 00 01 83 00 01 83'
:24B8 00 03 01 80 03 01 80 00'
:24C0 06 00 C0 00 00 00 00 10'
:24C8 00 00 7C 00 00 38 00 00'
:24D0 00 00 00 00 00 00 00 00'
:24D8 00 00 00 00 00 00 00 00'
:24E0 00 00 00 00 00 00 00 00'
:24E8 00 00 00 00 00 00 00 00'
:24F0 00 00 00 00 00 00 00 00'
:24F8 00 00 00 00 00 00 00 00'
:2500 FF 00 FF 00 FF 00 FF 00'
```

Les mêmes deformés (alias DATAs) :

LISTING 2

```
1 FORF=8960TO9472:READA:POKE F,A:NEXTF
10 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
11 DATA 0,0,0,56,0,0,40,0,0,0,0,0,0,0,0,16
12 DATA 0,0,254,0,1,255,0,3,125,128,6,56,192,12,56,96
13 DATA 1,187,0,0,238,0,0,68,0,0,108,0,0,238,0,0
14 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
15 DATA 0,0,0,0,0,0,16,0,0,1
16 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
17 DATA 24,0,48,0,0,0,0,0,0,0,0,0,0,0,0,0
18 DATA 0,0,0,1,131,0,1,187,0,1,171,0,1,131,0,1
19 DATA 131,0,1,147,0,0,254,0,0,254,0,0,124,0,0,124
20 DATA 0,0,124,0,0,124,0,0,108,0,0,108,0,0,198,0
21 DATA 0,198,0,0,198,0,1,131,0,1,131,0,1,131,0,0
22 DATA 1,131,0,0,0,0,0,0,0,0,16,0,0,124,0,0
23 DATA 56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
24 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
25 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
26 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
27 DATA 0,0,0,56,0,3,41,128,3,1,128,1,131,0,1,147
28 DATA 0,0,254,0,0,124,0,0,124,0,0,56,0,0,56,0
29 DATA 1,187,0,0,238,0,0,68,0,0,108,0,0,238,0,0
30 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
31 DATA 0,0,3,1,128,0,16,0,0,124,0,0,56,0,0,0
32 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
33 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
34 DATA 0,0,0,6,56,192,6,40,192,3,1,128,3,1,128,1
35 DATA 147,0,1,215,0,0,254,0,0,254,0,0,124,0,0,124
36 DATA 0,0,124,0,0,124,0,0,108,0,0,108,0,0,198,0
37 DATA 0,198,0,1,131,0,1,131,0,3,1,128,3,1,128,0
38 DATA 6,0,192,0,0,0,0,16,0,0,124,0,0,56,0,0
39 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
40 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
41 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
42 DATA 148,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Lorsque la merveilleuse routine de déplacement (presque définitive) sera terminée, les 6 NOPs seront remplacés par des JSR BRUITAGE, l'adresse bruitage n'existant pas encore !

LISTING 3

```
C100 A2 00     LDX #$00
C102 86 FF     STX $FF
C104 AD 00 DC  LDA $DC00
C107 29 10     AND #$10
C109 D0 04     BNE $C10F
C10B A2 01     LDX #$01
C10D 86 FF     STX $FF
C10F AD 00 DC  LDA $DC00
C112 29 0F     AND #$0F
C114 C9 0F     CMP #$0F
C116 D0 03     BNE $C11B
C118 4C 50 C1  JMP $C150
C11B 4A        LSR
C11C B0 05     BCS $C123
C11E 48        PHA
C11F 20 A0 C1  JSR $C1A0
C122 68        PLA
C123 4A        LSR
C124 B0 05     BCS $C12B
C126 48        PHA
C127 20 00 C2  JSR $C200
C12A 68        PLA
C12B 4A        LSR
C12C B0 05     BCS $C133
C12E 48        PHA
C12F 20 50 C2  JSR $C250
C132 68        PLA
C133 4A        LSR
C134 B0 03     BCS $C139
C136 20 A0 C2  JSR $C2A0
C139 60        RTS
```

A compléter de :

LISTING 4

```
C150 A9 80     LDA #$80
C152 8D F8 07  STA $07F8
C155 A9 81     LDA #$81
C157 8D F9 07  STA $07F9
C15A 60        RTS
```

LISTING 5

```
C1A0 AD 01 D0  LDA $D001
C1A3 C9 3D     CMP #$3D
C1A5 F0 42     BEQ $C1E9
C1A7 A9 8C     LDA #$8C
C1A9 8D F8 07  STA $07F8
C1AC A9 8D     LDA #$8D
C1AE 8D F9 07  STA $07F9
C1B1 A5 A2     LDA $A2
C1B3 69 15     ADC #$15
C1B5 C5 A2     CMP $A2
C1B7 D0 FC     BNE $C1B5
C1B9 A9 8E     LDA #$8E
C1BB 8D F8 07  STA $07F8
C1BE A9 8F     LDA #$8F
C1C0 8D F9 07  STA $07F9
C1C3 A2 28     LDX #$28
C1C5 EA        NOP
C1C6 EA        NOP
C1C7 EA        NOP
C1C8 CE 01 D0  DEC $D001
C1CB CE 03 D0  DEC $D003
C1CE A5 A2     LDA $A2
C1D0 C5 A2     CMP $A2
C1D2 F0 FC     BEQ $C1D0
C1D4 CA        DEX
C1D5 D0 EE     BNE $C1C5
C1D7 A9 8C     LDA #$8C
C1D9 8D F8 07  STA $07F8
C1DC A9 8D     LDA #$8D
C1DE 8D F9 07  STA $07F9
C1E1 A5 A2     LDA $A2
C1E3 69 10     ADC #$10
C1E5 C5 A2     CMP $A2
C1E7 D0 FC     BNE $C1E5
C1E9 60        RTS
```

LISTING 6

```
C200 AD 01 D0  LDA $D001
C203 C9 DD     CMP #$DD
C205 F0 42     BEQ $C249
C207 A9 90     LDA #$90
C209 8D F8 07  STA $07F8
C20C A9 91     LDA #$91
C20E 8D F9 07  STA $07F9
C211 A5 A2     LDA $A2
C213 69 15     ADC #$15
C215 C5 A2     CMP $A2
C217 D0 FC     BNE $C215
C219 A9 92     LDA #$92
C21B 8D F8 07  STA $07F8
C21E A9 93     LDA #$93
C220 8D F9 07  STA $07F9
C223 A2 28     LDX #$28
C225 EA        NOP
C226 EA        NOP
C227 EA        NOP
C228 EE 01 D0  INC $D001
C22B EE 03 D0  INC $D003
C22E A5 A2     LDA $A2
C230 C5 A2     CMP $A2
C232 F0 FC     BEQ $C230
C234 CA        DEX
C235 D0 EE     BNE $C225
C237 A9 8C     LDA #$8C
C239 8D F8 07  STA $07F8
C23C A9 8D     LDA #$8D
C23E 8D F9 07  STA $07F9
C241 A5 A2     LDA $A2
C243 69 10     ADC #$10
C245 C5 A2     CMP $A2
C247 D0 FC     BNE $C245
C249 60        RTS
```

LISTING 7

```
C250 AC 00 D0  LDY $D000
C253 C0 15     CPY #$15
C255 F0 23     BEQ $C27A
C257 CE 00 D0  DEC $D000
C25A CE 02 D0  DEC $D002
C25D A5 A2     LDA $A2
C25F 29 08     AND #$08
C261 D0 0D     BNE $C270
C263 A9 86     LDA #$86
C265 8D F8 07  STA $07F8
C268 A9 87     LDA #$87
C26A 8D F9 07  STA $07F9
C26D 4C 7A C2  JMP $C27A
C270 A9 88     LDA #$88
C272 8D F8 07  STA $07F8
C275 A9 89     LDA #$89
C277 8D F9 07  STA $07F9
C27A 60        RTS
```

LISTING 8

```
C2A0 AC 00 D0  LDY $D000
C2A3 C0 FF     CPY #$FF
C2A5 F0 23     BEQ $C2CA
C2A7 EE 00 D0  INC $D000
C2AA EE 02 D0  INC $D002
C2AD A5 A2     LDA $A2
C2AF 29 08     AND #$08
C2B1 D0 0D     BNE $C2C0
C2B3 A9 82     LDA #$82
C2B5 8D F8 07  STA $07F8
C2B8 A9 83     LDA #$83
C2BA 8D F9 07  STA $07F9
C2BD 4C CA C2  JMP $C2CA
C2C0 A9 84     LDA #$84
C2C2 8D F8 07  STA $07F8
C2C5 A9 85     LDA #$85
C2C7 8D F9 07  STA $07F9
C2CA 60        RTS
```

Pour la démo, le listing 4 du mois dernier est toujours valable, une petite modif toutefois : en C01D, mettre LDA # $DD au lieu de # $50.

Sébastien MOUGEY

Alias - je ne serai pas à la mode et ne mettrai pas alias quelque chose.

LANGAGE MACHINE SUR ZX 81

Comment améliorer, sur notre ZX, la gestion des chaînes de caractères et la rapidité de leur affichage à l'écran ? Quel est l'informaticien profondément humain et un tantinet littéraire qui n'a jamais souhaité, un jour, émailler ses programmes de nombreux messages de convivialité et de longs commentaires personnalisés ? Malheureusement avec le ZX le nombre de chaînes de caractères possible est limité aux 26 lettres de l'alphabet (on fait pas un poème avec ça !); quand à l'affichage de ces chaînes il est lent et l'usage de la tabulation oblige a multiplier les lignes de programmation avec leurs instructions PRINT AT.
Je vous propose donc de remplacer tout ce dispositif extrêmement gourmand de mémoire par une simple instruction RND x, où "x" prendra la valeur de l'adresse de départ de notre routine. Pour plus de commodité, celle-ci, ne comprenant que des JR (saut relatif), sera parfaitement relogeable et vous pourrez donc l'implanter où bon vous semblera dans votre 16 ou 64K, et surtout l'ajouter à la suite des routines qui, je l'espère pour vous, s'y trouvent déjà. Mais qu'allons-nous réaliser exactement ?

Une variable chaîne de caractères unique, T$, devra contenir à elle seule tout le texte écrit et pourquoi pas la partie graphique et illustrative d'un même programme. Les différents messages à afficher figureront les uns à la suite des autres dans cette chaîne, isolés par des séparateurs "»" et identifiables chacun par des numéros d'ordre qui suivront immédiatement ces séparateurs.
Avant le message lui-même, différents symboles seront interprétés par la routine afin d'assurer tabulation et mise en page. J'ai choisi "$" pour le changement de ligne, "+" pour la tabulation colonne 10, "£" pour la tabulation colonne 21. Vous pourrez aisément remplacer mes symboles par les vôtres, il vous suffira de leur substituer de nouveaux codes CHR$; de plus, vous pourrez rajouter de nouvelles possibilités à cette routine en y intercalant des modules de traitement supplémentaires ; sa structure s'y prête parfaitement.
Enfin, pour assurer un repérage sans failles de la chaîne T$ dans l'espace mémoire affecté aux variables, je propose d'amorcer celle-ci par un signe "+" redoublé. A titre d'exemple, la chaîne T$ pourrait se présenter de la façon suivantes :

T$ = "++)1£VOICI LE PREMIER MESSAGE)2+VOICI LE SECOND)3$LE TROISIEME)4ET LE DERNIER)" (dans laquelle 1,2,3,4 seront en réalité représentés par leurs équivalents CHR$.

Sauts de lignes et tabulations devront toujours s'effectuer en tenant compte de la position actualisée du pointeur d'écriture DF-CC. En l'absence d'indications c'est cette position qui sera retenue pour l'affichage du tronçon de T$. Le choix de ce tronçon s'effectuera par le truchement d'un octet qu'il conviendra préalablement de poker au numéro choisi. C'est l'octet d'adresse 16417 qui remplira cet office. Exemple : si je POKE 16417,3 puis RND x, la routine m'affichera le troisième tronçon de T$ c'est-à-dire dans notre chaîne, après un saut de ligne, le message "LE TROISIEME". Une simple boucle Basic permettra l'incrémentation automatique de cet octet.
Rappelons pour finir à tous ceux qui auraient à rentrer un texte important dans T$ qu'il convient d'utiliser la procédure dite de CONCATENATION et donc de créer une variable de transit U$ qui accueillera le texte de chaque tronçon avant de le placer à la suite de ses antérieurs dans T$ par une simple opération d'addition de chaînes : LET T$ = T$ + U$.

Vous constatez que la routine fait amplement appel aux variables-système du ZX81; E-LINE et VARS représentent les limites inférieures et supérieures de l'espace mémoire affecté aux variables. C'est dans cet intervalle que la routine cherchera le code de T$ (57 (T) + 32 (constante d'identification des variables chaînes) = 89) suivi, trois octets plus loin, des codes de "++". La recherche du tronçon à afficher est suivie d'une "sécurité" empêchant l'affichage de déborder au-delà de la zone mémoire réservée à l'écran. Vient ensuite le MODULE D'AIGUILLAGE qui, après repérage des symboles, oriente le travail vers les différents MODULES DE TRAITEMENT. Ils correspondent successivement aux tabulations, sauts de lignes et séparateurs. Vous remarquerez qu'au fur et à mesure de l'AFFICHAGE SUR HL on réactualise la position du pointeur d'écriture en "pokant" à l'adresse DF-CC, en même temps qu'on prend garde à ne pas déborder sur une fin de ligne (118). Le retour au Basic s'effectue au niveau du module d'aiguillage après repérage du symbole "»", aussi ne jamais l'oublier à la fin du dernier tronçon de T$!

Je ne doute pas qu'économie de mémoire et confort d'utilisation seront suffisants pour faire de vous d'excellents, sinon programmeurs, du moins prosateurs...

Bernard Guyot

```
REM ★ GESTION D'UNE CHAINE T$

REM ★ LIMITES
 :L31 LD HL.(16404) E-LINE
 PUSH HL
 LD HL.(16400) VARS

REM ★ IDENTIF. T$, ++
 :L0 LD A.(HL)
 CP 89
 INC HL
 JR NZ.L0
 PUSH HL
 INC HL saute nb car.
 INC HL de T$
 LD A.(HL)
 CP 21 +
 JR NZ.L1
 INC HL
 CP 21 +
 JR Z.L2
 :L1 POP HL redépart
 JR L0
 :L2 POP DE défausse pile.
 INC HL pointe sur »
 POP BC E-LINE dans BC

REM ★ IDENTIF NO D'ORDRE
 :L3 LD A.L
 CP C si HL=BC
 JR NZ.L25 et BC=E-LINE
 LD A.H alors
 CP B retour
 RET Z
 :L25 LD A.(HL)
 CP 18 »
 INC HL
 JR NZ.L3
 LD D.(HL) n°tronçon T$
 LD A.(16417) n°compteur
 CP D
 JR NZ.L3

REM ★ SAUVE HL, DF-CC
 :L4 PUSH HL
 LD HL.(16396) D-FILE
 LD DE.792
 ADD HL.DE fin du FA
 LD B.H
 LD C.L
 LD HL.(16398) DF-CC

REM ★ AIGUILLAGE
 POP DE départ message
 :L14 LD A.H
 CP B si DF-CC
 JR C.L26 en fin de
 LD A.L FA...
 CP C impossible...
 RET NC donc retour.
 :L26 LD (16398).HL DF-CC.
 INC DE saute le n°
 LD A.(DE)
 CP 13 $
 JR Z.L15
 CP 21 +
 JR Z.L16
 CP 12 £
 JR Z.L17
 CP 18 » fin du
 RET Z tronçon.
 JR L20

REM ★ "$" = SAUT DE LIGNE
 :L15 LD A.(HL)
 CP 118 retour
 DEC HL début de
 JR NZ.L15 ligne.
 PUSH DE libère DE
 LD DE.35 pour obtenir
 ADD HL.DE début de ligne
 POP DE suivante.
 JR L14

REM ★ "+" = TAB 10
 :L16 LD A.(HL)
 CP 118
 DEC HL
 JR NZ.L16
 PUSH DE libère DE.
 LD DE.12 tabulation
 ADD HL.DE à 10.
 POP DE
 JR L14 caract. next

REM ★ "£" = TAB 21
 :L17 LD A.(HL)
 CP 118
 DEC HL
 JR NZ.L17
 PUSH DE
 LD DE.23 tabulation
 ADD HL.DE à 21
 POP DE
 JR L14

REM ★ AFFICHAGE SUR HL
 :L20 LD (HL).A écriture
 INC HL inc DF-CC
 LD A.(HL) vérification
 CP 118
 JR NZ.L14
 INC HL
 JR L14

REM ★ n'oubliez pas l'initialisation de l'octet 16417 !
```

LANGAGE MACHINE SUR MSX

```
                      .Z80
0000' 00        BASC: DB   0
0001' 9000     ADLEC: DW   9000H
0003' 0001     COMPT: DW   1
0005' F5        DEB:: PUSH AF
0006' C5              PUSH BC
0007' D5              PUSH DE
0008' E5              PUSH HL
0009' CD 0011'        CALL MUSIC
000C' E1              POP  HL
000D' D1              POP  DE
000E' C1              POP  BC
000F' F1              POP  AF
0010' C9              RET
0011' 21 0003' MUSIC: LD   HL,COMPT
0014' ED 4B 0003'     LD   BC,(COMPT)
0018' 0B              DEC  BC
0019' ED 43 0003'     LD   (COMPT),BC
001D' 78              LD   A,B
001E' B1              OR   C
001F' C0              RET  NZ
0020' 3E 00           LD   A,0
0022' CD 005F'        CALL LI
0025' CD 0093         CALL 93H
0028' 3C              INC  A
0029' CD 005F'        CALL LI
002C' CB 7B           BIT  7,E
002E' 28 03           JR   Z,MU1
0030' 32 0000'        LD   (BASC),A
0033' CD 0093   MU1:  CALL 93H
0036' 06 07           LD   B,7
0038' CD 005F'  MU2:  CALL LI
003B' 3C              INC  A
003C' CD 0093         CALL 93H
003F' 10 F7           DJNZ MU2
0041' CD 005F'        CALL LI
0044' 4B              LD   C,E
0045' CD 005F'        CALL LI
0048' 43              LD   B,E
0049' ED 43 0003'     LD   (COMPT),BC
004D' 3A 0000'        LD   A,(BASC)
0050' A7              AND  A
0051' C8              RET  Z
0052' 3E 00           LD   A,0
0054' 32 0000'        LD   (BASC),A
0057' 01 9000         LD   BC,9000H
005A' ED 43 0001'     LD   (ADLEC),BC
005E' C9              RET
005F' E5         LI:  PUSH HL
0060' C5              PUSH BC
0061' ED 4B 0001'     LD   BC,(ADLEC)
0065' 2A 0001'        LD   HL,(ADLEC)
0068' 03              INC  BC
0069' ED 43 0001'     LD   (ADLEC),BC
006D' 5E              LD   E,(HL)
006E' C1              POP  BC
006F' E1              POP  HL
0070' C9              RET
                      END
```

Ciel ! Un listing ! Déjà ?..

Eh oui très chers lecteurs, mais rassurez-vous, comme nous vous savons (de Marseille, ha, ha !) ou plutôt vous espérons (et ron, petit patapon) en vacances, notre cours d'aujourd'hui ne sera guère abscons (comme un balai, hi, hi !). Notre clémence proverbiale ne vous vaudra point cette fois de magistral discours sur les bienfaits incontestables du mnémonique ZIMBABOUE HL, à préférer au vulgaire KRAKBUM BC, mais plutôt une mélodieuse démonstration des possiblités enchanteresses du remarquable PSG (Programmable Sound Générator).

Sans plus attendre, tapez et lancez le listing 2 (Basic), non sans avoir au préalable, poussé sensiblement le volume de votre téléviseur (ou moniteur pour les nobles et fortunés). Dès lors, toutes les 30 secondes et quoi que vous fassiez (NEW, programmation, pipi ou caca), votre MSX interprètera avec emphase, le début d'un thème céleste fort connu (du moins de nos frères allumés).

Tout de même quelques précisions : Les codes du listing source (putain ça donne soif...), sont à l'ombre des lignes 30 à 40 du listing Basic. La partition s'étale sur la plage des DATA à partir de la ligne 70. Les frétillants codes hexadécimaux qui composent cette partition, ont été pêchés au large du listing 3. A noter que ce dernier a été réalisé par l'extraordinaire programme EDITEUR MUSICAL des numéros 78-79.

Il ne vous reste plus qu'à enchanter vos oreilles et celles de vos proches de quelques délicieuses et personnelles tonitruances, en sachant que :

chaque ligne de DATA détermine un son complet de la manière suivante :

DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

1 : octet faible de la fréquence du canal A.
2 : octet fort de la fréquence du canal A.
3 : octet faible de la fréquence du canal B.
4 : octet fort de la fréquence du canal B.
5 : octet faible de la fréquence du canal C.
6 : octet fort de la fréquence du canal C.
7 : période du générateur de bruit.
8 : bascule de contrôle de l'ouverture des canaux.
9 : volume du canal A.
10 : volume du canal B.
11 : volume du canal C.
12 : octet faible de la période de l'enveloppe.
13 : octet fort de la période de l'enveloppe.
14 : type d'enveloppe.
15 : octet faible de la durée de la note.
16 : octet fort de la durée de la note.

Pour les valeurs à entrer autres que les notes, veuillez consulter votre manuel de référence, parce que nous, hein...

Les estivants Nicolas Bourdin et Jean-Claude Paulin.

LISTING 2

```
10 POKE&HFD9F,&HC9
20 FORI=0TO&H70 READA$ POKE&H8F00+I,V
AL("&H"+A$) NEXT POKE&HFDA0,5 POKE&HF
DA1,&H8F POKE&HFD9F,&HC3
30 DATA1,0,90,10,0,F5,C5,D5,E5,CD,11,
8F,E1,D1,C1,F1,C9,21,3,8F,ED,4B,3,8F,
B,ED,43,3,8F,78,B0,C0,3E,0,CD,5F,8F,C
D,93,0,3C,CD,5F,8F,CB,7B,28,3,32,0,8F
,CD,93,0,6,C,CD,5F,8F,3C,CD,93,0,10,F
7,CD,5F,8F,4B,CD,5F,8F,43,ED,43,3,8F,
3A,0,8F,A7,C8,3E,0,32,0,8F
40 DATA1,0,90,ED,43,1,8F,C9,E5,C5,ED,
4B,1,8F,2A,1,8F,3,ED,43,1,8F,5E,C1,E1
,C9
50 I=0
60 READA$ IFA$="END"THENENDELSEPOKE&H
9000+I,VAL("&H"+A$) I=I+1 GOTO60
70 DATA53,1,AB,1,D5,0,0,38,8,8,8,F,F,
0,30,0
80 DATA53,1,1D,1,8E,0,0,38,8,8,8,F,F,
0,30,0
90 DATA7C,1,40,1,A0,0,0,38,8,8,8,F,F,
0,8,0
100 DATAAB,1,53,1,A9,0,0,38,8,8,8,F,F
,0,8,0
110 DATAC4,1,7C,1,BE,0,0,38,8,8,8,F,F
,0,8,0
120 DATA1D,1,D5,0,6A,0,0,38,8,8,8,F,F
,0,30,0
130 DATA53,1,1D,1,8E,0,0,38,8,8,8,F,F
,0,10,0
140 DATA7C,1,40,1,A0,0,0,38,8,8,8,F,F
,0,8,0
150 DATAAB,1,53,1,A9,0,0,38,8,8,8,F,F
,0,8,0
160 DATAC4,1,7C,1,BE,0,0,38,8,8,8,F,F
,0,8,0
170 DATA1D,1,D5,0,6A,0,0,38,8,8,8,F,F
,0,30,0
180 DATA53,1,1D,1,8E,0,0,38,8,8,8,F,F
,0,10,0
190 DATABE,0,A0,0,50,0,0,38,8,8,8,F,F
,0,8,0
200 DATAD5,0,A9,0,54,0,0,38,8,8,8,F,F
,0,8,0
210 DATABE,0,A0,0,50,0,0,38,8,8,8,F,F
,0,8,0
220 DATAE2,00,BE,0,5F,0,0,38,8,8,8,F,
F,0,30,0
230 DATAE2,00,BE,0,5F,0,0,38,0,0,0,F,
F,0,FF,5
240 DATAEND
```

LISTING 3

```
LA        =    55.00 Hz  -->  FE3
LA#;SIb   =    58.27 Hz  -->  EFF
SI        =    61.74 Hz  -->  E27
DO        =    65.41 Hz  -->  D5C
DO#;REb   =    69.30 Hz  -->  C9C
RE        =    73.42 Hz  -->  BE7
RE#;MIb   =    77.78 Hz  -->  B3C
MI        =    82.41 Hz  -->  A9A
FA        =    87.31 Hz  -->  A02
FA#;SOLb  =    92.50 Hz  -->  972
SOL       =    98.00 Hz  -->  8EA
SOL#;LAb  =   103.83 Hz  -->  86A
LA        =   110.00 Hz  -->  7F1
LA#;SIb   =   116.54 Hz  -->  77F
SI        =   123.47 Hz  -->  713
DO        =   130.81 Hz  -->  6AE
DO#;REb   =   138.59 Hz  -->  64E
RE        =   146.83 Hz  -->  5F3
RE#;MIb   =   155.56 Hz  -->  59E
MI        =   164.81 Hz  -->  54D
FA        =   174.61 Hz  -->  501
FA#;SOLb  =   185.00 Hz  -->  4B9
SOL       =   196.00 Hz  -->  475
SOL#;LAb  =   207.65 Hz  -->  435
LA        =   220.00 Hz  -->  3F8
LA#;SIb   =   233.08 Hz  -->  3BF
SI        =   246.94 Hz  -->  389
DO        =   261.63 Hz  -->  357
DO#;REb   =   277.18 Hz  -->  327
RE        =   293.66 Hz  -->  2F9
RE#;MIb   =   311.13 Hz  -->  2CF
MI        =   329.63 Hz  -->  2A6
FA        =   349.23 Hz  -->  280
FA#;SOLb  =   369.99 Hz  -->  25C
SOL       =   392.00 Hz  -->  23A
SOL#;LAb  =   415.30 Hz  -->  21A
LA        =   440.00 Hz  -->  1FC
LA#;SIb   =   466.16 Hz  -->  1DF
SI        =   493.88 Hz  -->  1C4
DO        =   523.25 Hz  -->  1AB
DO#;REb   =   554.37 Hz  -->  193
RE        =   587.33 Hz  -->  17C
RE#;MIb   =   622.25 Hz  -->  167
MI        =   659.26 Hz  -->  153
FA        =   698.46 Hz  -->  140
FA#;SOLb  =   739.99 Hz  -->  12E
SOL       =   783.99 Hz  -->  11D
SOL#;LAb  =   830.61 Hz  -->  10D
LA        =   880.00 Hz  -->  FE
LA#;SIb   =   932.33 Hz  -->  EF
SI        =   987.77 Hz  -->  E2
DO        =  1046.50 Hz  -->  D5
DO#;REb   =  1108.73 Hz  -->  C9
RE        =  1174.66 Hz  -->  BE
RE#;MIb   =  1244.51 Hz  -->  B3
MI        =  1318.51 Hz  -->  A9
FA        =  1396.91 Hz  -->  A0
FA#;SOLb  =  1479.98 Hz  -->  97
SOL       =  1567.98 Hz  -->  8E
SOL#;LAb  =  1661.22 Hz  -->  86
LA        =  1760.00 Hz  -->  7F
LA#;SIb   =  1864.66 Hz  -->  77
SI        =  1975.53 Hz  -->  71
DO        =  2093.00 Hz  -->  6A
DO#;REb   =  2217.46 Hz  -->  64
RE        =  2349.32 Hz  -->  5F
RE#;MIb   =  2489.02 Hz  -->  59
MI        =  2637.02 Hz  -->  54
FA        =  2793.83 Hz  -->  50
FA#;SOLb  =  2959.96 Hz  -->  4B
SOL       =  3135.96 Hz  -->  47
SOL#;LAb  =  3322.44 Hz  -->  43
LA        =  3520.00 Hz  -->  3F
LA#;SIb   =  3729.31 Hz  -->  3B
SI        =  3951.07 Hz  -->  38
DO        =  4186.01 Hz  -->  35
DO#;REb   =  4434.92 Hz  -->  32
RE        =  4698.64 Hz  -->  2F
RE#;MIb   =  4978.03 Hz  -->  2C
MI        =  5274.04 Hz  -->  2A
FA        =  5587.65 Hz  -->  28
FA#;SOLb  =  5919.91 Hz  -->  25
SOL       =  6271.93 Hz  -->  23
SOL#;LAb  =  6644.88 Hz  -->  21
LA        =  7040.00 Hz  -->  1F
LA#;SIb   =  7458.62 Hz  -->  1D
SI        =  7902.13 Hz  -->  1C
DO        =  8372.02 Hz  -->  1A
DO#;REb   =  8869.84 Hz  -->  19
RE        =  9397.27 Hz  -->  17
RE#;MIb   =  9956.06 Hz  -->  16
MI        = 10548.08 Hz  -->  15
FA        = 11175.30 Hz  -->  14
FA#;SOLb  = 11839.82 Hz  -->  12
SOL       = 12543.85 Hz  -->  11
SOL#;LAb  = 13289.75 Hz  -->  10
```

Les estivants Nicolas Bourdin et Jean-Claude Paulin.

LANGAGE MACHINE SUR ZX 81

Voici un logiciel de 139 octets qui, s'il avait été écrit en basic, aurait occupé une place bien plus importante dans la mémoire de votre ZX. C'est d'ailleurs la raison pour laquelle on voit peu de BANDE ANNONCE dans les programmes tout-en-basic des programmeurs néophytes que vous n'êtes plus... Mais vous vous demandez sans doute ce que recouvre cette appellation qui évoque déjà dans votre imagination, sur le modeste écran cathodique de votre ZX, les fastes de la vidéo. Hélas ! ne planez pas plus haut, revenez à l'âge de pierre des ordinateurs, lorsque la HRG et la couleur n'existaient pas plus que la possibilité d'incrustation vidéo et ne méprisez pas pour autant votre ZX puisque nous allons pouvoir réaliser ,malgré tout, nos "bandes annonces".

Il s'agira en réalité de faire défiler sur une fenêtre de 29 caractères, par un scrolling de gauche à droite, les différents tronçons d'une chaîne de caractères D$. Chacun des tronçons sera numéroté de 1 à 255 max, l'affichage pouvant être séquentiel (incrémentation automatique) ou sélectif à chaque tour. Nous raffinerons d'avantage en nous offrant le luxe de choisir la ligne sur laquelle se situera notre fenêtre d'affichage. Pour pouvoir faire varier à notre guise ces deux paramètres je propose une solution facile à implanter et pratique à utiliser : deux octets dans la zone RAM des variables-système, les emplacements 16417 pour le numéro de ligne et 16507 pour stocker le numéro d'ordre à partir duquel la séquence pourra se dérouler. Une simple instruction POKE permettra à tout moment de réinitialiser ces deux octets.

Grâce à cette routine vous pourrez émailler vos logiciels d'amples et abondants commentaires qui, par leur mode original et dynamique d'apparition à l'écran, ne susciteront pas les baillements et n'empiéteront pas sur une illustration ou un tableau de chiffres déjà présent à l'écran. Vous êtes convaincus ? On commence ?

Puisque nous utilisons une chaîne de caractères, il faudra tout d'abord la rechercher dans la zone RAM où le système sinclair range indifféremment tous les noms et valeurs de variables. Cet espace a des limites précises : en amont VARS (16400) et en aval E-LINE (16404); aussi utiliserons-nous les contenus de ces adresses comme bornes nécessaires à la puissante instruction CPIR. Celle-ci effectue des comparaisons successives de l'accumulateur avec le contenu de l'octet pointé par HL et automatiquement incrémenté jusqu'à ce que la comparaison s'avère positive. C'est ainsi que nous recherchons d'abord le symbole représentant le nom de la chaîne -(D = 41) + (chaîne de caractères = 32) = 73-, puis, deux octets plus loin, le symbole doublé "$$" (13,13) dont la présence évitera tout risque de confusion avec d'autres octets marqués 73.

Arrivés à ce point nous pointons l'adresse du premier caractère de la chaîne D$; reste à pointer sur le tronçon qui nous intéresse dans cette chaîne. On l'identifie en 16507 et on le recherche tout au long de D$ grâce, une fois encore, à l'instruction CPIR. Une fois trouvée l'adresse de départ du bon tronçon, il faut alors se préparer à son affichage. Pour cela il nous faut un deuxième paramètre : sur quelle

ligne situer notre fenêtre d'affichage. L'octet d'adresse 16417 apportera la réponse à cette question, tandis qu'un dispositif de sécurité veillera à ce que cette réponse ne déborde pas, par erreur, la limite supérieure du fichier d'affichage. Si c'était le cas, l'affichage aurait lieu automatiquement sur la vingtquatrième ligne de l'écran.

On teste ensuite chaque caractère du tronçon avant de l'afficher sur le dernier octet à droite de notre fenêtre; en effet si le signe "$" réapparaissait il signalerait cette fois la fin de ce tronçon et nécessiterait donc la sortie de notre routine.

Le travail trop rapide de l'assembleur oblige à recourir à une temporisation avant de procéder au décalage global vers la gauche de la totalité de la fenêtre d'affichage. Vous remarquez pour ce petit scrolling le travail du registre C qui permet le transit, octet par octet, de la gauche vers la droite, en évitant "l'écrasement" pur et simple de l'octet-à-gauche par l'octet-à-droite.

Une fois le tronçon lu dans sa totalité, il convient pour des raisons esthétiques, de faire évacuer la fenêtre d'affichage en poursuivant 29 fois le scrolling; ceci aura pour effet de dupliquer 29 fois le dernier signe du tronçon c'est pourquoi je vous recommande, afin de simuler la disparition, de terminer tous les messages de vos tronçons par une case blanche (0) avant le signe de fin ($).

La routine s'achève sur l'incrémentation de l'octet 16507 qui permet un séquencement automatique des différents messages.

Ah ! Une dernière chose... Si vous ne savez pas avec quoi remplir D$, je vous conseille les œuvres complètes de Gérard Ceccaldi !

Bernard Guyot

LANGAGE MACHINE SUR APPLE

VIVE LES VACANCES ?

EH, oh on fait bronzette sur la plage, on conte fleurette aux jeunes filles, et bien évidemment on n'a pas emporté son APPLE en vacances : où va-t-on je vous le demande; de mon temps c'était mieux ! C'est tout juste si on achète le numéro quadruple de l'HHHHEBDO pour faire bonne figure, ou pour mettre dans sa collection. Ressaisissez-vous que diable ! (pauvre France !). Vous n'allez tout de même pas perdre le bénifice des cours assembleur précédents en vous pavanant sous les paulmiers en kléber-muda !

VOUS AVEZ DIT 65C02 ?

Que ceux qui possèdent un APPLE IIc aillent faire un tour ailleurs car ce qui suit ne devrait pas les concerner : il va en effet être question de la simulation de certaines instructions propres au 65C02 sur un vulgaire 6502. Voici tout d'abord la liste de ces dites instructions (ça mange pas de pain et ça permet de remplir la page sans se fatiguer) : BRA, PHX, PHY, PLX, PLY, STP, STZ, TRB, TSB, WAI, INC A, DEC A. Commençons par le commencement (original non ?) :

- BRA qui signifie BRanch Always indique au processeur qu'il faut toujours effectuer le branchement (d'où le nom). C'est ce qu'on l'habitude de rendre (pouah !) par un CLV BVC ou encore par un CLC BCC (les deux conditions précédentes étant remplies il y a bien branchement). Pour ce qui est du déplacement, tout ce passe comme pour les BNE, BEQ... et autres BPL.
- PHX, PHY (PusH X register, PusH Y register) : ce sont respectivement les équivalents de PHA pour les registres ixe et igrécque (?). Quand on n'a pas la chance de posséder ces instructions on se contente de faire : STA adresse TXA PHA LDA adresse, où adresse désigne un emplacement libre de la mémoire. Le principe est le même pour simuler PHY.
- PLX, PLY (PuLl X register, PuLl Y register) : auxiliaires indispensables de PHX et PHY, ils sont de la même famille que PHA et PHP. Voici une simulation maison de PLX : STA adresse PLA TAX LDA adresse. On emploie la même méthode pour faire un PLY (il suffit de remplacer TAX par TAY).
- STZ (STore Zero) : c'est l'équivalent du CLR (clear) du 68000. Mais j'imagine que cela ne vous dit pas grand chose. En fait STZ est une instruction stockant zéro à l'adresse désirée. Ainsi STZ ADR peut être rendu par STA adresse LDA £$00 STA ADR LDA adresse.
- WAI (WAIt) : c'est la classique routine d'attente. Surr 6502 apple-ez la routine en $FCA8 de la ROM. Fastoche ! (on peut également utiliser des NOP)
- INC A, DEC A : très intéressantes (INCrémente A, DECrémente A) car évitent les traditionnels CLC ADC £$01 et SEC SBC £$01. Bon ben j'crois qu'on a fini pour cette fonction.

Je vous entend déjà réclamer les STP (SToP), TRB et autres TSB; mais pour ne pas vous lasser nous en reparlerons une prochaine fois (je sens que les possesseurs de IIc se languissent car ils viennent juste de rentrer de leur promenade estivale).

WHAT ELSE ?

Revenons au bon vieux 6502. Il reste an corps beau coût de choses à dire (j'expérimente l'écriture phonétique pour faire travailler les méninges). Parlons par exemple de la pile du micro-processeur. Mais caisse ? Il s'agit d'une zone mémoire se comportant comme une pile d'assiettes : on ne peut accéder directement qu'à l'assiette du dessus, c'est ce qu'on appelle le

sommet de la pile. C'est aussi la dernière assiette qui a été empilée. Tiens, on vient d'employer le terme "empiler" : cela signifie déposer une assiette sur le dessus. De la même façon, on défini le terme "dépiler" comme étant l'action de retirer une assiette du dessus (c'est également de la que vient le qualificatif employé dans l'expression "crème DÉPILATOIRE" : c'est une crème qui enlève les poils du dessus). Munis de ces deux opérations, on peut accéder à tout élément de la pile (pour les non anglicistes, PUSH correspond à empiler et PULL à dépiler). Pour que le pile ne perde pas la face, on lui alloue la zone mémoire entre $100 et $1FF. De plus, il existe un registre interne du processeur (appelé S pour Stack register (Stack veut dire "pile" (j'imbrique les parenthèses pour embrouiller tout le monde (sauf moi (c'est fait exprès))))) qui pointe sur le premier emplacement libre de la pile, S-1 correspondant alors au sommet de la pile. On est en présence d'une pile montante, l'écriture se faisant dans le sens des adresses décroissantes. Ainsi, pour empiler, on décrémente S, tandis que pour dépiler, on l'incrémente. Initialement, S est à $FF.

Voici un dessin pour fixer le peu d'idées que vous avez :

$100
.
.
.
.
$1FC emplacement libre pointé par S
$1FD DONNEE 1 sommet de la pile
$1FE DONNEE 2
$1FF DONNEE 3 bas de la pile

MANIPULATIONS DIVERSES

Comme vous vous en doutez, il existe plusieurs instructions permettant de manipuler la pile. En voici la liste : PHA, PLA, PHP, PLP, TSX, TXS.

Commençons par changer les deux dernières : TSX et TXS. Elles ont respectivement pour rôle de transférer le registre S dans X et réciproquement. Ainsi, en faisant TSX LDA $101,X, on tombe pile (ah ah ah) sur le sommet de la pile : je vous rappelle que S pointe sur le prochain emplacement libre, c'est'pourquoi il faut faire un LDA $101,X et non pas un LDA $100,X.

Passons maintenant aux instructions d'empilement PHA et PHP. Lorsque le processeur rencontre un PHA (de code $48), il stocke dans le premier emplacement libre de la pile le contenu de l'accumulateur, puis décrémente S. Tout se passe de la même façon avec PHP (PusH P register) excepté que cette fois c'est le registre d'état P qui est emplié. C'est tout aussi simple avec PLA et PLP (instructions de dépliement). On récupère alors le contenu du sommet de la pile, en le place dans A ou P selon le choix, et on incrémente S (l'adresse de l'ancien sommet correspondant maintenant à celle du prochain emplacement libre).

FONCTION FONDAMENTALE

A quoi sert la pile, me direz-vous ? Il faut savoir que celle-ci ne change pas : non contente de servir de lieu de stockage temporaire, elle joue un rôle déterminant dans l'exécution des sous-programmes : elle est par exemple sollicitée lors d'un JSR ou d'un RTS. Examinons de plus près ces deux fonctions.

1) Un JSR ADRESSE a pour effet de sauver PC + 2 (compteur ordinal : celui qui pointe sur les instructions que l'on exécute froidement) sur la pile, et de placer ADRESSE dans PC.

2) Lors d'un RTS, on récupère dans PC le sommet de la pile (PC + 2), on ajoute deux à S et on incrémente PC (qui pointe alors sur la nouvelle instruction à exécuter).

Vous trouverez ci-dessous un exemple expliqué dans tous ses détails. Supposons que vous exécutiez le programme suivant (présentant un intérêt purement pédagogique !) :

300 JSR $800
303 BRK
800 JSR $908
803 RTS
908 RTS

Décrivons l'état de la pile après chacune de ces opérations (on suppose que la valeur initiale de S est bien $FF).

300 JSR $800 1FF = 02, 1FE = 03, S = FD, PC = 800
800 JSR $908 1FF = 02, 1FE = 03, 1FD = 02, 1FC = 08, S = FB, PC = 908
908 RTS 1FF = 02, 1FE = 03, 1FD = 02, 1FC = 08, S = FD, PC = 803
803 RTS 1FF = 02, 1FE = 03, 1FD = 02, 1FC = 08, S = FF, PC = 303

THE END

Voilà. Vous savez tout sur la pile, et vous avez donc la possibilité de réaliser des programmes entièrement relogeables, grâce à d'habiles manipulations de la pile. Ce sera tout pour cette fois-ci, et rendez-vous en septembre pour la rentrée, si l'avion qui me ramènera des Bahamas n'est pas détourné. D'ici là, glandez bien.

Philippe PIERNOT (ricard)

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 → 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130 134 138 142 146
COMMODORE → 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130 134 138 142 146
ORIC → 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128 131 136 141 144 146
AMSTRAD → 111 115 119 123 127 131 135 139 143 146
APPLE → 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132 135 139 143 146
SPECTRUM → 112 116 120 124 127 132 136 140 144 146
THOMSON → 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133 137 141 146
MSX → 113 117 121 125 129 133 137 140 146

LANGAGE MACHINE SUR MSX

AU BOULOT

La frappe fébrile et hâtive de l'aimable listing du numéro d'août engendra une étrange et lyrique mélopée qui bouleversa bon nombre de lecteurs vacanciers. Ces derniers sans trop réfléchir (on est en vacances, merde), s'empressèrent de triturer maladroitement quelques malheureux datas, dans l'espoir éperdu d'égaler les maitres bienveillants, partis depuis se dorer la pilule ou se cailler les miches dans quelque lointaine contrée. Contrastant avec le doux bruit du ressac, des sons anarchiques s'envolèrent dans le ciel de France, lequel ne manqua pas de réagir à l'outrage, en marquant sa contrariété par quelques intempéries bien senties et largement méritées.
Bref, il manquait quelque chose aux brutes électroniques qui visiblement ne se souciaient guère de l'harmonie des choses : la comprenette. Sans plus attendre, les maitres bronzés et pimpants se remirent au labeur, car comme dit le précepte : comprend le son que tu te fourres dans l'oreille, comme le reste que tu te fourres ailleurs (aïe !). Tiens, aïe justement...

ALLEZ LE PSG !..

Le processeur sonore AY-3-8910 (oui, ça ressemble au cri de douleur d'un informaticien le jour de paye), cause de tout ce tintamarre et qui équipe par miracle nos beaux MSX, est appelé plus communément PSG (Programmable Sound Générator). Puisqu'il s'agit d'un processeur programmable et que le sujet du cours reste quand même la programmation, nous allons voir comment que c'est que ça se programme un PSG donc (charabia, viré !).

ALORS, ÇA VIENT ?

Le PSG, c'est tout d'abord 3 canaux sonores indépendants et réglables, ensuite un générateur de bruit, puis un générateur d'enveloppe, un sytème de mixage de tout ça et accessoirement, deux ports d'entrées/sorties.

Tout d'abord, parlons de ce dont on ne parlera pas (ouais d'accord, je vous avais prévenu, vi-ré ! !), les ports d'entrées/sorties : ils servent, entre autres, à la gestion des joysticks. On aura donc surement l'occasion d'en recauser dans le poste si on est pas viré avant. Voilà.

Pour le reste, le contrôle du 8910 se fait à travers les registres de ce même. Donc, un petit zyeutage de leur côté va nous faire le plus grand bien.

Il faut savoir, qu'en tant que puce qui se respecte, le PSG met 16 registres à votre disposition, fonctionnant un peu comme ceux du Z80. A l'inverse de ces précieux derniers, ils ne se nommeront pas par des lettres de l'alphabet, mais plus humblement par des R (pour 'registre' : étonnant !) suivis de leur numéro d'ordre. Pour plus de commodité, les canaux sonores seront quant à eux, nommés par des lettres de l'alphabet. On aura donc de R0 à R15 pour contrôler les mélodies électroniques produites à travers les canaux A, B et C.

On aurait pu se contenter de vous renvoyer à l'HHHHebdo numéro 146-147-148-149 pour savoir ce que contrôle le registre R(n), où 'n' est un entier compris entre 0 et 15, en précisant que tout naturellement, la définition de l'usage de ce registre était précisée dans les commentaires numérotés du bas gâuche de la page 60, avec le commentaire 1 pour le registre R0, le blabla 2 pour le registre R1, etc... Donc en précisant que le numéro de commentaire correspondant à un 'n' donné, était égal à n + 1. Mais en agissant ainsi, on aurait pu passer (aux yeux des perspicaces) pour des roublards, des faussaires, parcequ'alors, on vous aurait faussement renseigné.

Comme jamais nous n'oserions (tu parles), voici ce qu'il convient encore de rajouter :

- R0 : s'exprime sur 8 bits.
- R1 : ne mémorise qu'un quartet (les 4 bits de poids faible de l'octet que vous lui enverrez).
- R2 : octet, comme R0.
- R3 : quartet, comme R1.
- R4 : octet.
- R5 : quartet.

Donc, en ayant fait les rapprochements qui s'imposent (R0 et R1, R2 et R3, R4 et R5, forment des couples de registres contrôlant respectivement les fréquences des canaux A, B et C. Ces fréquences s'expriment sur 12 bits et donc par des nombres compris entre 0 et 4095.), vous auriez attendu la formule qui permet de déterminer la valeur nécessitée pour obtenir une fréquence donnée en hertz et qui est : valeur = 3579545/(16 ★ fréquence). Et bien, attendez...

- R6 : ce sont seuls les 5 bits de poids faibles de l'octet transmis à ce registre qui importent pour contrôler la fréquence du générateur de bruit. La formule de calcul de la valeur de ces 5 bits en fonction de la fréquence désirée, étant la même que celle déterminant la fréquence des canaux A, B et C. Il va sans dire que la plage des fréquences utilisables ici est bien plus réduite.

- R7 : lui est spécial, il ouvre (1) ou ferme (0) les canaux et gère le mixage du bruit et du son. Chacun de ses bits a un rôle spécifique : bit 0 : à 1, détermine la présence de son sur le canal A, à 0, c'est une absence qui est marquée.
bit 1 : même chose que le bit 1 mais pour le canal B.
bit 2 : même chose, mais pour le canal C.
bit 3 : comme le bit 0, mais pour le bruit sur le canal A.
bit 4 : le bruit du canal B.
bit 5 : le bruit du canal C.
bits 6 et 7 : en rapport avec les ports d'entrées sorties.

- R8 : son quartet de poids faible permet de moduler le volume du canal A, son bit 4 détermine si l'enveloppe agira ou non sur le son provenant de ce canal.
- R9 : comme R8 mais pour le canal B.
- R10 : comme R8 mais pour le canal C.

- R11 et R12 : rien à ajouter, ils gérent effectivement la période de l'enveloppe et, la période, c'est le temps qui se passera entre deux crêtes du volume, crêtes mises en évidences par les représentations des courbes d'enveloppes que vous pouvez admirez dans vos manuels préférés (On note d'ailleurs que sur ces représentations graphiques, on ne précise jamais que l'abscisse est censée représenter le temps et, l'ordonnée, le volume.).

- R13 : le quartet de poids faible de ce registre mémorise (nous y voilà), la forme d'enveloppe. Alors là : manuel. On ne va pas vous dire : "Alors là, il y a une montée, puis ça se met à redescendre, mais, ho suspens, va-ce remonter à nouveau ?" (c'est Français ça ? Viré !).

- R14 et R15 : les ports d'entrées-sorties déjà signalés.

D'accord, c'est bien beau tout ça, mais il n'y a pas d'instructions du type LD Rn,valeur ou OUT (Rn),A, alors comment que c'est-y qu'on y accède à ces sacrés registres : par deux instructions OUT. La première enverra le numéro du registre sur lequel on veut agir, au port d'adressage du PSG. La seconde enverra la valeur que l'on veut transmettre à ce registre au port de contrôle du PSG. Mais puisque nous avons pu constater des variations des numéros de ces ports d'un MSX à l'autre, nous nous contenterons d'affirmer que sur tous les MSX, on peut procéder à cette savante opération en appelant le vecteur 93H, avec le numéro du registre à trafiquer dans l'accumulateur et la valeur à envoyer dans le registre E du Z80.

C'était la délicate mission du programme de notre dernier cours, en choisissant habilement et sous interruptions, les valeurs de chargement des registres du processeur musiqueux.

tralalilalère...

Nibourdin Colas et
Jeaulin Pandeclau

LANGAGE MACHINE SUR ZX 81

On ne le dira jamais assez : une bonne routine est une routine paramètrable ! Elle doit offrir à son usager la plus grande souplesse d'utilisation possible, c'est à dire la capacité de tendre, à partir d'un objectif général, vers un résultat plus spécifique au programme Basic dans laquelle elle viendra s'insérer. En voici un bon exemple aujourd'hui avec ce programme d'INVERSION VIDEO PARAMETRABLE. Si l'objectif principal de cette routine reste en tout état de cause la transformation du noir-sur-fond-blanc en blanc-sur-fond-noir, le travail effectué d'un appel (RAND USR) à l'autre, pourra chaque fois se prêter à des "interprétations" fort différentes. Il suffira pour cela de modifier quelques indices, quelques "paramètres". Je sens bien que j'excite votre curiosité, alors voyons de plus près de quoi-qu'il-s'agit-là...

D'abord, cette routine vous offrira pour le même prix un autre service : un scanning clavier performant, auto-protégé contre la touche "break", et capable d'identifier facilement la touche pressée; ensuite, et comme cadeau de rentrée, je vous offre la possibilité de faire se succéder les inversions vidéo afin d'obtenir un effet de clignotement. Nul besoin de préciser que toutes ces fonctions sont simultanées avant d'examiner ce qui fait l'intérêt majeur de cette routine : le mode de paramètrage utilisé. Nous disposerons de cinq indices qui permettront de faire varier, dans la limite des possibilités du ZX, à la fois l'emplacement et le format de la zone-écran frappée d'inversion vidéo, et la durée d'activation du scanning clavier. Innovation de taille : les paramètres ne seront plus modifiés par la lourde procèdure du "pokage" d'octet mais figureront directement sur le listing sous forme de codes CHR$ dans une ligne REM réservée à cet effet. Donc initialisation immédiate et facile de ces paramètres et surtout considérable économie de mémoire ! Il y a de quoi quitter sans regrets les plages aoûtiennes pour s'enfermer dans un profitable tête-à-tête avec son bon vieux ZX pas vrai ? !

Vous n'aurez aucun mal à suivre sur le listing ci-contre le déroulement des opérations.
Le premier module détient en fait la clé de ce système de paramètrage. Vous remarquez qu'il fait appel à la variable-système CH-ADD. Celle-ci permet de localiser en mémoire l'adresse du prochain caractère à fournir à l'interpréteur basic lors de sa lecture d'un listing. C'est là que réside l'astuce : si j'interroge CH-ADD, à partir d'un appel sur listing basic de type RAND USR x (ou x est l'adresse de cette routine d'interrogation), cette variable-système me renverra forcément l'adresse de la ligne de basic suivante. Si cette ligne supporte une REM, je peux alors y faire figurer sous forme de CHR$ différentes valeurs dont les adresses deviennent facilement repérables, donc exploitables en tant que paramètres. On aurait par exemple le fragment de listing suivant :

```
100 RAND USR 16514
110 REM >"$★ =
```

où 16514 serait l'adresse de départ de notre routine et les symboles présents en REM les cinq paramètres nécessaires.

L'utilisation d'une autre variable-système (PRBUFF) va nous permettre, par le truchement de l'instruction LDIR (chargement par bloc avec incrémentation et répétition) de transférer les onze premiers octets (BC = 11) à partir de CH-ADD vers les emplacements libres constitués par les octets du buffer d'imprimante. Il s'agit ici de simplifier, à chaque lecture d'un paramètre, le calcul de son adresse.
Le module suivant, "ADRESSE LIGNE", a pour charge de lire le septième octet du buffer d'imprimante (BI) qui détient à ce moment le premier paramètre de notre routine : l'adresse de la ligne d'écran à partir de laquelle se fera l'inversion vidéo. En cas de "blanc" (0) c'est la ligne 0 qui sera retenue. Vient ensuite, suivant le même procédé, l'identification de la colonne de départ, puis la longueur (nombre de colonnes) de la ligne d'écran qui sera inversée. C'est en 16453 (dixième octet du BI) qu'on pourra lire le nombre de lignes sur lesquelles cette inversion sera répétée. Cet ensemble de quatre paramètres définira donc à chaque fois un format original d'inversion vidéo. Le module de "comparaison" vérifiera que ce format ne déborde pas de l'espace alloué au fichier d'affichage; il supprime ainsi les risques de "plantage" en retournant au basic (RET M) en cas d'erreur.
Ce sont les trois modules suivant qui exécuterons le travail d'inversion vidéo proprement dit, tandis que le quatrième, prenant en tenaille, dans une boucle indicée par le registre E, la routine de scrutation du clavier, en assurera 255 répétitions avant de passer la main au module "durée". C'est ici que nous exploiterons notre cinquième et dernier paramètre, lequel vise à reproduire (de 0 à 255 fois, c'est au choix) l'ensemble des opérations déjà décrites.

Les deux derniers modules correspondent aux deux modes de sortie de cette routine :
1 - Le temps imparti est épuisé, aucune touche n'ayant été pressée : l'octet 16417 prend la valeur 0.
2 - Une touche a été pressée. Si c'est la touche BREAK on n'en tient pas compte; si c'est une autre touche on dépose son code en 16417 avant de retourner au Basic.

...Et maintenant, à vos paramètres !

Bernard Guyot


```
REM * INVERSION VIDEO PARAMETRABLE
      AVEC SCANNING CLAVIER

REM * CHARGT PRBUFF
      LD HL,(16406)   CH-ADD
      LD DE,16444     PRBUFF
      LD BC,11
      LDIR

REM * ADRESSE LIGNE
      LD HL,(16396), D-FILE
      INC HL
      LD A,(16450)    départ inv,
      CP 0
      JR Z,L8
      LD B,A
      LD DE,33
:L0   ADD HL,DE
      DJNZ,L0

REM * ADRESSE COLONNE
:L8   LD D,0
      LD A,(16451)    colonne
      LD E,A
      ADD HL,DE
      PUSH HL

REM * CONTROLE F.A.
      LD D,0
      LD A,(16452)    longueur
      LD E,A
      ADD HL,DE
      LD A,(16453)    nb lignes
      LD B,A
      LD DE,33
:L9   ADD HL,DE
      DJNZ,L9
      XOR A
      SBC HL,DE
      EX DE,HL

REM * COMPARAISON
:L10  LD HL,(16396)
      LD BC,792
      ADD HL,BC       fin F.A,
      XOR A
      SBC HL,DE
      POP HL          si ret
      RET M

REM * LIGNES ET LONGUEUR
:L4   PUSH HL
      PUSH HL
      LD A,(16453)
      LD B,A
:L1   LD A,(16452)
      LD C,A
REM * INVERSION VIDEO
:L2   LD A,(HL)
      CP 118
      JR NZ,L7
      INC HL
      LD A,(HL)
:L7   ADD A,$80
      LD (HL),A
      INC HL
      DEC C
      JR NZ,L2

REM * REPETITIONS
      POP HL
      LD DE,33
      ADD HL,DE
      PUSH HL
      DJNZ,L1
      POP HL          défausse

REM * TEMP. & SCANNING
      LD E,255
:L3   CALL $02BB
      LD B,H
      LD C,L
      LD D,C
      INC D
      JR NZ,L5
      DEC E
      JR NZ,L3

REM * DUREE
:L6   POP HL
      LD A,(16454)
      DEC A
      LD (16454),A
      CP 0
      JR NZ,L4

REM * RETOUR NUL AUTOM.
      LD A,0
      LD (16417),A
      RET

REM * SORTIE PAR SCANNING
:L5   CALL $07BD
      LD A,(HL)
      CP 0            break
      POP HL
      JR Z,L4
      LD (16417),A
      RET

CARACTERISTIQUES
- 148 octets relogeables
- structure de la REM qui devra
  être placée impérativement
  après l'appel de routine:
     1° octet = ligne
     2° octet = colonne
     3° octet = longueur
     4° octet = nb lignes
     5° octet = durée
Attention: pour chacun de ces
octets les valeurs doivent être
représentées par leurs
équivalents CHR$ (de 0 à 255).
```

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130 134 138 142 146 150

COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130 134 138 142 146 152

ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128 131 136 141 144 146 153

AMSTRAD -» 111 115 119 123 127 131 135 139 143 146 151

APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132 135 139 143 146 151

SPECTRUM -» 112 116 120 124 127 132 136 140 144 146 153

THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133 137 141 146 152

MSX -» 113 117 121 125 129 133 137 140 146 150

LANGAGE MACHINE SUR ZX 81

Il a suffi que je m'écarte quelques semaines de la rue Baron (le siège d'Hebdogiciel, au cas où des terroristes l'ignoreraient) en direction des chaudes tropiques du cancer (quoi de plus légitime depuis le front populaire) pour que la chienlit s'installe dans les colonnes de votre rubrique préférée : le cours d'assembleur ZX81.
Ainsi les aoûtiens ont-ils été frustrés du listing de la routine "BANDE ANNONCE" qui, bien qu'annoncée, n'est cependant jamais parue ! Vous n'avez pu, entre deux baignades, qu'en savourer le commentaire et d'aucuns auront mis cette étrange disparition sur le compte d'un début d'insolation ou d'hydrocution qui sont, comme chacun sait, les seuls moyens de ne pas s'ennuyer sur les plages pendant qu'on emmagasine des ultraviolets pour l'hiver. Qu'ils se rassurent : le soleil, pas plus que le string de leur préoccupante voisine, n'a été responsable d'un quelconque état hallucinatoire ! Non, c'est seulement qu'il n'y avait pas davantage de place dans les colonnes de l'HHHHHebdo d'août que vous n'en aviez vous-même à l'époque sur les bords de la Méditerranée (et ce n'était certainement pas à votre voisine, la fille au string, que vous vous en seriez plaint...).

Bon, heureusement depuis mon retour, mon coup de gueule et surtout mon rachat de la majorité des parts de la société HHHHebdogiciel and Co, les choses ont bien changé. Tout est réorganisé, l'ordre nouveau s'épanouit harmonieusement dans le doux ronronnement des disquettes; tout est luxe, calme et volupté et, je le dis au cas où ça vous intéresserait : l'équipe est au complet.

Vous en étiez frustrés et bien La voici : dans sa sublime transparence, sa nudité provocante, parée de ses labels de qualité. Elle est là, devant vous celle qui a si longtemps excité votre curiosité, qui a mobilisé tous vos désirs de l'été; la voici qui s'offre enfin après cette insupportable attente où votre désir exacerbé ne pouvait trouver son exutoire, coincés que vous étiez entre les fesses de l'une et les seins de l'autre, sans pouvoir rien faire d'autre qu'attendre sous le soleil implacable la sortie du prochain numéro. La voici donc dans toute sa beauté, telle que vous l'imaginiez, telle que vous rêviez de la posséder un jour, avec ses 139 octets tous bien alignés, son architecture irréprochable, ses sous-programmes rondelets, jolis, rebondis...
Alors allez-y, n'attendez plus et surtout... ne vous gênez pas pour moi.

Pour faire bonne mesure avec le numéro d'août je vous propose, en pendant, le listing d'une routine mais cette fois privée de son commentaire; l'équilibre ainsi rétabli permettra de reprendre un bon départ pour le prochain cours.
Vous énervez pas, je vous en dis un peu plus : Il s'agit en fait de la première partie d'une routine dite de "codage réversible". Elle vous permettra de conserver en mémoire magnétique tous vos programmes en langage machine sous une forme codée, indéchiffrable et inutilisable par quiconque n'aura pas à sa disposition la routine de décodage, autrement dit par quiconque n'aurait pas lu le prochain numéro.

A bon lecteur, salut...

Bernard Guyot

```
REM * BANDE ANNONCE

REM * LIMITES CPIR
      LD HL,(16404)    E-LINE
      LD DE,(16400)    VARS
      PUSH DE
      XOR  A
      SBC HL,DE        nb octets
      PUSH HL
:L0   POP BC
      POP HL

REM * RECHERCHE D$+$$
      LD A,73          D+32
      CPIR
      RET NZ
      PUSH HL
      PUSH BC
      INC HL
      INC HL
      LD A,(HL)
      CP 13            $
      JR NZ,L0
      INC HL
      LD A,(HL)
      CP 13            $
      JR NZ,L0
      INC HL           pointe le n°

REM * DEFAUSSE PILE
      POP DE           limites CPIR
      POP DE           inutiles

REM * RECHERCHE $ + NUM.
:L7   LD A,(16507)     n° message
      LD  D,A
      LD A,(HL)
      CP D
      JR Z,L8
      LD A,13
      CPIR
      RET NZ
      JR L7

REM * POSITION D'ECRITURE
:L8   INC HL
      PUSH HL
      LD HL,(16396)    D-FILE
      DEC HL           col 30...
      DEC HL           ...virtuel
      LD A,(16417)     n°ligne
      LD B,A
      LD A,24
      CP B
      JR NC,L6         si B<A
      LD B,23
:L6   LD DE,33         affichage
:L1   ADD HL,DE        24*ligne
      DJNZ,L1
      POP DE

REM * TEST END ET ECRIT
:L5   LD A,(DE)
      CP 13
      JR Z,L10
      LD (HL),A        écriture
      INC DE           next car.

REM * TEMPORISATION
      LD B,25
:L2   LD C,255
:L3   DEC C
      JR NZ,L3
      DEC B
      JR NZ,L2

REM * SCROLLING A GAUCHE
      PUSH HL
      LD B,29
:L4   DEC HL
      LD C,(HL)
      LD (HL),A
      LD A,C           transit
      DJNZ,L4
      POP HL
      JR L5

REM * FIN DE SCROLL
:L10  LD D,35
:L11  PUSH HL
      LD A,(HL)        dernier
      LD B,29          octet
:L12  DEC HL           scroll ...
      LD C,(HL)        remplacement
      LD (HL),A        par le contenu
      LD A,C           de la denière
      DJNZ,L12         case.
      POP HL
      DEC D
      JR Z,L13         fene^tre vide


REM * TEMPORISATION
      LD B,25
:L14  LD C,255
:L15  DEC C
      JR NZ,L15
      DEC B
      JR NZ,L14
      JR L11

REM * INC COMPTEUR
:L13  LD HL,16507
      INC (HL)
      RET

------------------

1000 LET D$="$$1ME VOILA $2ME REVOILA
     $3C EST LA DERNIERE... $"
1006 FOR I = 1 TO 100
1007 RAND
1008 POKE 16507,I
1009 POKE 16417,(INT(RND*20)+1)
1010 RAND USR 8192
1015 IF I=4 THEN LET I=0
1016 NEXT I

Attention: dans ce listing 1,2,3, doivent
être représentés par leurs
équivalents CHR$. La routine est ici logée
à l'adresse 8192 (sur mémoire 64K)

-----------------

CARACTERISTIQUES:
- Routine relogeable
- Longueur: 139 octets
- Adresses obligatoires:
    16417 = N° de ligne
    16507 = N° d'ordre
- La REM utilise D$="$$1... $2... $3... $"
  (où 1,2,3, sont représentés par leurs
  équivalent CHR$)

_________

REM * CODAGE LM SELECTIF
      ET REVERSIBLE

REM * LIMITES
      LD A,0
      LD (14083),A
      LD DE,14084
      LD HL,16300
      LD BC,14167
      XOR A
      SBC HL,BC
      PUSH HL

REM * DETECTION
      LD A,1
:L0   LD HL,14167
      POP BC
      PUSH BC
      CPIR
      JR NZ,L1
      INC A
      CP 255
      JR Z,L2
      JR L0

REM * CHARGEMENT
:L1   LD (DE),A
      PUSH AF
      LD HL,14083
      LD B,(HL)
      INC B
      LD A,10
      CP B
      JR Z,L3
      LD (HL),B
      INC DE
      POP AF
      INC A
      JR L0

REM * RETOUR
:L3   POP AF
:L2   POP BC

REM * TRUCAGE
REM * DEPART BUFFER 2
      LD DE,14094
:L10  PUSH BC
      LD HL,14167

REM * COMPTEUR
:L11  LD A,(DE)
      CPIR
      JR Z,L12
      LD A,(14083)
      DEC A
      LD (14083),A
      CP 0
      JR Z,L13
      INC DE
      POP BC
      JR L10

REM * BUFFER 1
:L12  PUSH DE
      LD DE,14084
      LD A,(14083)
      PUSH HL
      LD H,0
      LD L,A
      ADD HL,DE
      LD A,(HL)
      POP HL
      DEC HL
      LD (HL),A
      INC HL
      POP DE
      JR L11
:L13  POP BC
      RET
```

LANGAGE MACHINE SUR MSX

TRUC BIDULE etc...

Chers lecteurs :
- Listing 1 : truc bidule machin chouette en Basic - flèches haut et bas.
- Listing 2 : truc bidule machin chouette en Assembleur. Rajout d'un effet magnétique (nous sommes des scientifiques) à proximité de la plate-forme.
- Listing 3 : chargeur Basic des codes machine du listing source. Evasion par ESPACE.
- Mais qu'est-ce ? Pourquoi ? Quand ? Comment ? Où ça ? Qui es-tu étranger ? Les réponses dans le prochain catalogue de 'La Redoute".

Nicolas BOURDINET et Jean-Claude PAULINOU.

LISTING 1

```
10 '************************************
20 '*  Truc bidule machin chouette  *
30 '************************************
40 KEYOFF:DIMA$(7):COLOR,1,1:SCREEN1,
2:X=112:Y=96
50 'Redéfinition des caractères
60 RESTORE70:FORI=0TO6:READAD:FORJ=0T
07:READA$:VPOKEAD*8+J,VAL("&H"+A$):NE
XTJ,I
70 DATA 97,01,03,07,0F,1F,3F,7F,FF,10
4,FF,FF,FF,FF,E0,C0,80,,105,FF,FF,FF,
FF,1F,3F,7F,FF,112,FF,FF,FF,FF,F0,E0,
C0,80,120,FE,FC,F8,F0,E0,C0,80,,130,F
F,FF,FF,FF,FF,FF,FF,FF,151,FF,FE,FC,F
8,F0,E0,C0,80
80 'Définition des sprites
90 RESTORE100:FORI=1TO7:FORJ=1TO8:REA
DA$:A$(I)=A$(I)+CHR$(VAL("&H"+A$)):NE
XTJ,I
100 DATA,,,,,,,,FF,FF,FF,FF,FF,FF,FF,
FF,,,,,1F,3F,7F,FF,,,,,E0,C0,80,,,,,,
10,30,70,F0,F0,F0,F0,F0,F0,E0,C0,80,F
,F,F,1F,3F,7F,FF
110 SPRITE$(1)=A$(1)+A$(2)+A$(1)+A$(1
)
120 SPRITE$(2)=A$(3)+A$(1)+A$(4)+A$(1
)
130 SPRITE$(3)=A$(1)+A$(1)+A$(5)+A$(6
)
140 SPRITE$(4)=A$(7):SPRITE$(5)=A$(2)
150 'Mise en couleur des caractères
160 RESTORE170:FORI=0TO5:READAD,C:VPO
KEAD,C:NEXT
170 DATA8204,129,8205,141,8206,209,82
07,135,8208,65,8210,113
180 'Affichage de tout le bazar
190 LOCATE11,7:PRINT"ahix"
200 LOCATE11,8:PRINT"ép ù"
210 PUTSPRITE1,(120,55),8,4:PUTSPRITE
2,(120,63),4,5
220 S=STICK(0):Y=Y+(S=1)-(S=5)
230 PUTSPRITE3,(X,Y),4,1
240 PUTSPRITE4,(X,Y),8,2
250 PUTSPRITE5,(X,Y),7,3
260 GOTO 220
```

LISTING 2

ASSEMBLEUR de luxe

```
                    ORG  $C000
C000 CD6F00         CALL $6F
C003 010701         LD   BC,$0107
C006 CD4700         CALL $47
C009 0101E2         LD   BC,$E201
C00C CD4700         CALL $47
C00F 11BDC0         LD   DE,TCAR
C012 0607           LD   B,7
C014 1A       REMPLI LD  A,(DE)
C015 6F             LD   L,A
C016 13             INC  DE
C017 1A             LD   A,(DE)
C018 67             LD   H,A
C019 13             INC  DE
C01A C5       REMPL PUSH BC
C01B 0608           LD   B,8
C01D CD94C0         CALL REMP
C020 C1             POP  BC
C021 10F1           DJNZ REMPLI
C023 0614           LD   B,$14
C025 212038         LD   HL,$3820
C028 113BC1         LD   DE,SPR1
C02B 1A       LOOP  LD   A,(DE)
C02C D5             PUSH DE
C02D E5             PUSH HL
C02E 1600           LD   D,0
C030 5F             LD   E,A
C031 2103C1         LD   HL,TDS1
C034 19             ADD  HL,DE
C035 E5             PUSH HL
C036 D1             POP  DE
C037 E1             POP  HL
C038 C5             PUSH BC
C039 0608           LD   B,8
C03B CD94C0         CALL REMP
C03E C1             POP  BC
C03F D1             POP  DE
C040 13             INC  DE
C041 10E8           DJNZ LOOP
C043 114FC1         LD   DE,TOTO
C046 210C20         LD   HL,$200C
C049 0C07           LD   B,7
C04B CD94C0         CALL REMP
C04E 21ED18         LD   HL,$18ED
C051 1156C1         LD   DE,ECR1
C054 0604           LD   B,4
C056 CD94C0         CALL REMP
C059 210D19         LD   HL,$190D
C05C 0604           LD   B,4
C05E CD94C0         CALL REMP
C061 21041B         LD   HL,$1B04
C064 0614           LD   B,$14
C066 CD94C0         CALL REMP
C069 3E08     CLAV  LD   A,8
C06B CD4101         CALL $141
C06E 2172C1         LD   HL,Y
C071 CB6F           BIT  5,A
C073 2001           JR   NZ,HAUT
C075 35             DEC  (HL)
C076 CB77     HAUT  BIT  6,A
C078 2001           JR   NZ,SUIT
C07A 34             INC  (HL)
C07B CB47     SUIT  BIT  0,A
C07D C8             RET  Z
C07E 210C1B         LD   HL,$1B0C
C081 110400         LD   DE,4
C084 0603           LD   B,3
C086 3A72C1         LD   A,(Y)
C089 CD4D00   TITI  CALL $4D
C08C 19             ADD  HL,DE
C08D 10FA           DJNZ TITI
C08F CD9DC0         CALL RALEN
C092 18D5           JR   CLAV
C094 1A       REMP  LD   A,(DE)
C095 CD4D00         CALL $4D
C098 23             INC  HL
C099 13             INC  DE
C09A 10F8           DJNZ REMP
C09C C9             RET
C09D C5       RALEN PUSH BC
C09E F5             PUSH AF
C09F 3A72C1         LD   A,(Y)
C0A2 ED44           NEG
C0A4 C636           ADD  A,$36
C0A6 CB7F           BIT  7,A
C0A8 2802           JR   Z,RA
C0AA ED44           NEG
C0AC CB3F     RA    SRL  A
C0AE CB3F           SRL  A
C0B0 F601           OR   1
C0B2 47             LD   B,A
C0B3 0E00           LD   C,0
C0B5 0B       RAL   DEC  BC
C0B6 78             LD   A,B
C0B7 B1             OR   C
C0B8 20FB           JR   NZ,RAL
C0BA F1             POP  AF
C0BB C1             POP  BC
C0BC C9             RET
C0BD 0803     TCAR  DW   $0308
C0BF 0103070F       DB   $01,$03,$07,$0F
C0C3 1F3F7FFF       DB   $1F,$3F,$7F,$FF
C0C7 4003           DW   $0340
C0C9 FFFFFFFF       DB   $FF,$FF,$FF,$FF
C0CD E0C08000       DB   $E0,$C0,$80,$00
C0D1 4803           DW   $0348
C0D3 FFFFFFFF       DB   $FF,$FF,$FF,$FF
C0D7 1F3F7FFF       DB   $1F,$3F,$7F,$FF
C0D9 8003           DW   $0380
C0DD FFFFFFFF       DB   $FF,$FF,$FF,$FF
C0E1 F0E0C080       DB   $F0,$E0,$C0,$80
C0E5 C003           DW   $03C0
C0E7 FEFCF8F0       DB   $FE,$FC,$F8,$F0
C0EB E0C08000       DB   $E0,$C0,$80,$00
C0EF 1040           DW   $0410
C0F1 FFFFFFFF       DB   $FF,$FF,$FF,$FF
C0F5 FFFFFFFF       DB   $FF,$FF,$FF,$FF
C0F9 B804           DW   $04B8
C0FB FFFEFCF8       DB   $FF,$FE,$FC,$F8
C0FF F0E0C080       DB   $F0,$E0,$C0,$80
C103 00000000 TDS1  DB   $00,$00,$00,$00
C107 00000000       DB   $00,$00,$00,$00
C10B FFFFFFFF TDS2  DB   $FF,$FF,$FF,$FF
C10F FFFFFFFF       DB   $FF,$FF,$FF,$FF
C113 00000000 TDS3  DB   $00,$00,$00,$00
C117 1F3F7FFF       DB   $1F,$3F,$7F,$FF
C11B 00000000 TDS4  DB   $00,$00,$00,$00
C11F E0C08000       DB   $E0,$C0,$80,$00
C123 00000000 TDS5  DB   $00,$00,$00,$00
C127 103070F0       DB   $10,$30,$70,$F0
C12B F0F0F0F0 TDS6  DB   $F0,$F0,$F0,$F0
C12F F0E0C080       DB   $F0,$E0,$C0,$80
C133 0F0F0F0F TDS7  DB   $0F,$0F,$0F,$0F
C137 1F3F7FFF       DB   $1F,$3F,$7F,$FF
C13B 00080000 SPR1  DB   $00,$08,$00,$00
C13F 10001800 SPR2  DB   $10,$00,$18,$00
C143 00002028 SPR3  DB   $00,$00,$20,$28
C147 30000000 SPR4  DB   $30,$00,$00,$00
C14B 08000000 SPR5  DB   $08,$00,$00,$00
C14F 818DD187 TOTO  DB   $81,$8D,$D1,$87
C153 410071         DB   $41,$00,$71
C156 61686978 ECR1  DB   'ahix'
C15A 82702097 ECR2  DB   $82,'p ',$97
C15E 37781008 TAS1  DB   $37,$78,$10,$08
C162 3F781404 TAS2  DB   $3F,$78,$14,$04
C166 60700404 TAS3  DB   $60,$70,$04,$04
C16A 60700808 TAS4  DB   $60,$70,$08,$08
C16E 60700C07 TAS5  DB   $60,$70,$0C,$07
C172 60       Y     DB   $60
```

LISTING 3

```
10 KEYOFF:COLOR,1:CLEAR200,&HBFFF:L=6
0:FORI=&HC000 TO &HC172 STEP8
20 S=0:FORAD=ITOI+7:READA$
30 C=VAL("&H"+A$):POKEAD,C:S=S+C:NEXT
40 READS$:IFS<>VAL("&H"+S$)THENPRINT"
Erreur DATA ligne";L:STOP
50 L=L+10:NEXT:DEFUSR=&HC000:X=USR(0)
60 DATACD,6F,00,01,07,01,CD,47,259
70 DATA00,01,01,E2,CD,47,00,11,209
80 DATABD,C0,06,07,1A,6F,13,1A,240
90 DATA67,13,C5,06,08,CD,94,C0,36E
100 DATAC1,10,F1,06,14,21,20,38,255
110 DATA11,3B,C1,1A,D5,E5,16,00,2F7
120 DATA5F,21,03,C1,19,E5,D1,E1,3F4
130 DATAC5,06,08,CD,94,C0,C1,D1,486
140 DATA13,10,E8,11,4F,C1,21,0C,259
150 DATA20,06,07,CD,94,C0,21,ED,35C
160 DATA18,11,56,C1,06,04,CD,94,24B
170 DATAC0,21,0D,19,06,04,CD,94,272
180 DATAC0,21,04,1B,06,14,CD,94,27B
190 DATAC0,3E,08,CD,41,01,21,72,2A8
200 DATAC1,CB,6F,20,01,35,CB,77,393
210 DATA20,01,34,CB,47,C8,21,0C,25C
220 DATA1B,11,04,00,06,03,3A,72,0E5
230 DATAC1,CD,4D,00,19,10,FA,CD,3CB
240 DATA9D,C0,18,D5,1A,CD,4D,00,37E
250 DATA23,13,10,F8,C9,C5,F5,3A,3FB
260 DATA72,C1,ED,44,C6,36,CB,7F,4AA
270 DATA28,02,ED,44,CB,3F,CB,3F,36F
280 DATAF6,01,47,0E,00,0B,78,B1,280
290 DATA20,FB,F1,C1,C9,08,03,01,3A2
300 DATA03,07,0F,1F,3F,7F,FF,40,235
310 DATA03,FF,FF,FF,FF,E0,C0,80,61F
320 DATA00,48,03,FF,FF,FF,FF,1F,466
330 DATA3F,7F,FF,80,03,FF,FF,FF,53D
340 DATAFF,F0,E0,C0,80,C0,03,FE,5D0
350 DATAFC,F8,F0,E0,C0,80,00,10,514
360 DATA04,FF,FF,FF,FF,FF,FF,FF,6FD
370 DATAFF,B8,04,FF,FE,FC,F8,F0,69C
380 DATAE0,C0,80,00,00,00,00,00,220
390 DATA00,00,00,FF,FF,FF,FF,FF,4FB
400 DATAFF,FF,FF,00,00,00,00,1F,31C
410 DATA3F,7F,FF,00,00,00,00,E0,29D
420 DATAC0,80,00,00,00,00,00,10,150
430 DATA30,70,F0,F0,F0,F0,F0,F0,640
440 DATAE0,C0,80,0F,0F,0F,0F,1F,27B
450 DATA3F,7F,FF,00,08,00,00,10,1D5
460 DATA00,18,00,00,00,20,28,30,090
470 DATA00,00,00,08,00,00,00,81,089
480 DATA8D,D1,87,41,00,71,61,68,360
490 DATA69,78,82,70,20,97,37,78,339
500 DATA10,08,3F,78,14,04,60,70,1B7
510 DATA04,04,60,70,08,08,60,70,1B8
520 DATA0C,07,60,FF,00,00,FF,FF,370
```

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130 134 138 142 146 150 154

COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130 134 138 142 146 152 156

ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128 131 136 141 144 146 153 157

AMSTRAD -» 111 115 119 123 127 131 135 139 143 146 151 155

APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132 135 139 143 146 151 155

SPECTRUM -» 112 116 120 124 127 132 136 140 144 146 153 157

THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133 137 141 146 152 156

MSX -» 113 117 121 125 129 133 137 140 146 150 154

LANGAGE MACHINE SUR ZX 81

```
REM * DETRUCAGE
REM * ADRESSES
       LD HL,14094    buffer 2
       PUSH HL
       LD HL,14083    compteur
       LD A,0
       LD (HL),A
       LD DE,14084    buffer 1
       LD C,$C9       201
       LD B,10
:L20   LD A,(DE)
       CP C
       JR Z,L21
       INC DE         next vrai
       INC (HL)
       DJNZ,L20
:L21   DEC DE
       PUSH DE

REM * COMPTEUR SUR BC
:L22   LD HL,16300
       LD BC,14167
       PUSH BC
       XOR A
       SBC HL,BC
       PUSH HL
       POP BC
       POP HL

REM * IDENTIF BUF 1
       POP DE
:L23   LD A,(DE)
       CPIR
       JR Z,L24
       LD A,(14083)   compteur
       DEC A
       CP 0
       JR Z,L25
       LD (14083),A
       DEC DE
       POP BC
       INC BC
       PUSH BC
       PUSH DE
       JR L22

REM * PERMUTATION
:L24   EXX            échange..
       POP DE         registres..
       LD A,(DE)      auxiliaires.
       PUSH DE
       EXX            restauration
       DEC HL
       LD (HL),A
       INC HL
       JR L23

REM * RETOUR
:L25   POP BC         défausse
       RET

---------------

CARACTERISTIQUES
- Trucage réversible d'un listing LM
- Routine relogeable de 181 octets
- Chargements des buffers à GOTO 2000
- Adresses réservées: 14083 à 14103
- Jusqu'à 10 caractères sélectionnés
  par ordre croissant et sauvés
  dans un buffer situé de 14084
  à 14093 puis permutés avec
  une sélection de 10 caractères
  résidant sur buffer de 14094
  à 14109.
- L'octet 14083 est réservé au
  comptage.

---------------

2000 PRINT "CHARGEMENT AUTOMATIQUE
     DES 10 CODES A PERMUTER"
2050 DIM A(10)
2060 LET A(1)=201
2062 LET A(2)=32
2064 LET A(3)=40
2066 LET A(4)=119
2068 LET A(5)=50
2070 LET A(6)=229
2072 LET A(7)=193
2074 LET A(8)=42
2076 LET A(9)=16
2078 LET A(10)=237
2100 FOR I=0 TO 9
2110 POKE (I+(14094)),A(I+1)
2130 NEXT I
2150 PRINT "CODES PLACES"
2200 PRINT "CHRGT REPERES LIMITES"
2210 FOR I=0 TO 9
2220 POKE (I+14084),201
2230 NEXT I
2240 PRINT "TOUT EST OK"
```

Cette routine de "détrucage", fait suite au logiciel présenté lors du dernier cours d'assembleur sur ZX. Ces deux routines, une fois rassemblées, feront le bonheur de tous les fanatiques du copyright, des paranoïaques de la copie illicite, des maniaques de la dissimulation, des idolâtres du code secret. Que ceux qui n'ont jamais rien eu à cacher jettent immédiatement ce papier : il pourrait nuire à leur probable béatification par le Saint-Siège. Qu'ils le jettent ou... qu'ils le cachent.
Est-ce aussi satanique que j'en ai l'air ? Philosophiquement : non. Quand le bien ressemble au Bien et le mal au Mal, faire l'un ou l'autre c'est demeurer toujours dans l'innocence. Ainsi, lorsqu'avec cette routine vous métamorphoserez un programme bien écrit, structuré, efficace, en une succession aberrante d'instructions inutiles, votre supposée victime ne sera nullement trompée sur la marchandise qui ressemblera en réalité à ce qu'elle sera devenue.
Comment refaire le monde à l'envers ? C'est très simple : d'abord vous prenez le listing de votre routine de "trucage réversible"; vous le parcourez rapidement de votre regard incomparablement vif et vous saisissez dans l'instant son principe de fonctionnement : Mais Bon dieu ! Mais si bien sûr ! Il suffit, dans une routine en langage machine, d'isoler des valeurs d'octets qui n'ont pas été utilisées - et vous verrez on en trouve toujours - puis de mettre de côté ces précieuses valeurs qui vont nous servir de monnaie d'échange dans la suite des opérations.
Reprenons à partir de l'idée de base : Si dans le corps d'un listing je procède à des substitutions d'octets, j'empêche mon programme LM de tourner normalement. Transformez seulement un POP en PUSH et vous m'en direz des nouvelles ! Si ensuite je n'ai pas les moyens de restaurer dans son intégrité première mon logiciel, il peut être considéré comme définitivement perdu pour Jean, François, Paul, les autres et surtout pour moi. Ça peut être gênant dans presque tous les cas et surtout dans le mien. Que si j'ai les moyens de rétablir l'ordre initial en échangeant cette fois les bons octets contre les mauvais, je dispose alors d'un système de codage simple et efficace.

Pour cette routine, j'ai choisi de permuter une dizaine d'octets dont les valeurs codent des instructions très courantes de l'assembleur du Z80; je les appelle "valeurs vraies". Si toutefois l'une ou plusieurs d'entre elles ne figuraient pas sur le listing que vous souhaitez coder, la ou les suivantes seraient retenues. Vous trouverez en ligne 2600 un petit programme Basic qui vous permettra de placer automatiquement ces valeurs dans le buffer qui leur est attribué à partir de l'adresse 14094.
C'est dans cette partie de la mémoire 64K que la routine viendra les prélever en temps utile.
J'en profite pour signaler aux malheureux ZXistes qui n'auraient qu'une pauvre extension 16K à leur disposition, que cette routine est entièrement relogeable et qu'en conséquence, ils peuvent établir les deux buffers là où ils le désirent - à partir de 16514 - sans omettre l'octet de comptage. Il suffit pour cela de modifier les adresses de départ des buffers et du compteur.

Pour que notre codage soit réversible, il faut substituer à chacune de ces "valeurs vraies", une autre valeur différente de toutes celles déjà présente en listing. C'est l'objet de la partie "détection" de la routine de codage. Elle procède à dix examens attentifs de la routine à coder et isole un maximum de dix valeurs qui n'y figurent pas. Si elle n'en pouvait trouver que trois le codage s'effectuerait alors seulement à partir de ces trois valeurs. J'appelle les octets ainsi sélectionnés, des "valeurs fausses". La routine les range dans un buffer qui commence à l'adresse 14084. Le codage intervient dans le module de "trucage" qui échange systématiquement, dans le corps du listing, la valeur du premier octet des "valeurs vraies" avec la valeur du premier octet des "valeurs fausses". Le travail se poursuit avec les octets seconds des deux buffers, puis troisièmes etc... jusqu'à épuisement de toutes les "valeurs fausses", au terme de quoi votre listing, truffé "d'erreurs", est méconnaissable même pour son auteur.
C'est dans cet état d'indéchiffrabilité que vous ferez voyager vos copies magnétiques, sans craindre qu'elles ne tombent entre des mains ennemies puisque vous seul détenez la clef du décodage (avec moi, tant que vous n'aurez pas établi votre grille personnelle en modifiant les "valeurs vraies" des lignes 2060 à 2078).

La routine de "détrucage" effectue bien évidemment le même travail mais en sens inverse. On ne retrouve plus le module de "détection" puisque les valeurs fausses résident toujours en buffer. Elles sont successivement localisées sur le listing codé et permutées avec leurs homologues du buffer des "valeurs vraies". Juste avant le RET, votre logiciel a miraculeusement retrouvé sa physionomie originelle. Ouf !

Attention : pour chaque routine ou paquet de routines la clef sera différente. En effet, si c'est vous qui choisissez les "valeurs vraies", en revanche, c'est le module de détection qui détermine, en fonction de la routine à coder, la liste des "valeurs fausses". Le secret est ainsi partagé entre l'ordinateur et vous. Juste retour des choses...

Bernard Guyot

LANGAGE MACHINE SUR MSX

TRUC BIDULE etc... (suite).

Bien que somptueusement paré de trois listings sibyllins, le laconisme obscur et un tantinet gouailleur de notre dernier cours, laissait présager une suite pédagogique à la mesure de sa singularité. Nous nous devions d'éclaircir sans tarder le lecteur, sur la signification profonde de ces trois formulations synthétiques, résultat tangible d'un mode de pensée logique et structuré, paradoxalement issu des esprits malades et tourmentés que sont les nôtres.

LISTING HEIN

Compréhensible par tous car en bon vieux Basic, ce listing pouvant donc se passer de commentaires, est la base du grand tout, frère aîné du petit. En effet, le source (listing 2) en est directement déduit. D'ailleurs, en suivant les explications du listing assembleur, il n'est pas interdit de se référer à ce premier listing Basic, histoire de saisir bien fermement le pourquoi glissant du comment. Donc, une frappe minutiéuse vous a sans doute permis d'entrevoir un cube nationaliste (tricolore), traversant verticalement selon votre bon vouloir, une plate-forme rectangulaire du plus bel effet. Voyons cela en assembleur...

LISTING D'EUX

Donc, un listing conséquent, en assembleur de surcroît, qui à n'en pas douter s'organise à partir de $C000 et se destine à la création d'une foule de codes objets à la moindre directive d'assemblage.

$C000 : tout d'abord, initialisation du VDP en SCREEN 1 par la magie d'un CALL $6F. Voici donc nos tables correctement positionnées et la région de communication initialisée.

$C003 : bon c'est pas tout ça, maintenant va falloir préciser en quelle couleur on désire travailler et avec quelle taille de sprites. Dans le cas qui nous occupe, on aurait pu (on a oublié), remplir la table des couleurs avec du noir (histoire de se dispenser du COLOR,1 de la ligne 10 du listing 3) pour compléter la mise en noir de notre bordure d'écran. Bref, créer l'équivalent en assembleur d'un COLOR,1,1. Eh bien voici :

Assembleur Zarbi (de luxe).

```
C000              ORG  $C000
C000 CD6F00       CALL $6F
C003 210020       LD   HL,$2000
C006 0620         LD   B,$20
C008 3E01   FNOIR LD   A,1
C00A CD4D00       CALL $4D
C00D 23           INC  HL
C00E 10F8         DJNZ FNOIR
C010 010701       LD   BC,$0107
C013 C34700       JP   $47
```

En SCREEN1, la TC (table des couleurs) débute en $2000 (soit 8192) et se termine en $201F (8223), soit 32 octets. La valeur de chacun de ces octets, détermine la couleur de fond (bits 0 à 3) et de forme (Bits 4 à 7) de 8 configurations. Ici, HL est chargé de $2000 (adresse du début de table) et B est chargé de $20 (soit 32, nombre d'octets de la table). On place ensuite 1 (couleur de fond noire et de forme transparente) dans le registre A.

Coul.forme : Coul.fond

0 0 0 0 : 0 0 0 1

0 = transp. : 1 = noir

Après un CALL $4D chargé d'écrire à l'adresse de HL dans la VRAM (oui, sans cela on écrit en RAM), on incrémente HL et on se rebranche en FNOIR (étiquette pour "fond noir") jusqu'à ce que B soit égal à 0 (subtile action de DJNZ).
Ce travail effectué, occupons-nous de la bordure d'écran. Rien de plus simple lorsqu'on sait que sa couleur est déterminée (toujours en SCREEN1) par la valeur des bits 0 à 3 (dits bits de poids faible) du registre 7 du VDP, qu'il faut placer cette valeur dans B, puis le numéro du registre dans C avant d'effectuer un CALL $47 (écriture dans un registre VDP). A noter qu'ici, le JP à la place du CALL nous a dispensés d'un RET final (oui, car un JP n'empile pas d'adresse en se branchant. Ainsi, le RET rencontré en fin d'exécution de la routine appelée par CALL, dépile l'adresse de retour au programme principal).
Il faut savoir que l'affichage du contenu des touches de fonction est maintenu, mais a disparu du fait de la couleur de forme transparente. On peut toutefois supprimer purement et simplement cet affichage par rajout d'un CALL $CC. Faites l'expérience suivante sous Basic :
- DEFUSR1 = &HCF : DEFUSR2 = &HCC
- Apparition des touches de fonction : PRINTUSR1(0).
- Disparition des touches de fonction : PRINTUSR2(0).

Mais voilà qu'on perd le fil de notre programme, continuons. Rappelez-vous notre cours du numéro 129 concernant les registres du VDP. Le registre 1 par ses bits 0 et 1, détermine la taille des sprites. Ou plus précisément :
- Le bit 0 : lorsqu'il est à 0, les sprites sont de taille normale. Lorsqu'il est à 1, les sprites sont 2 fois plus grand (élargis). On appelle ça le facteur d'agrandissement.
- Le bit 1 : à 0, la taille des sprites est fixée à 8 ★ 8. A 1, la taille des sprites est fixée à 16 ★ 16.
Dans notre programme, nous avons fixé la taille des sprites à 16 ★ 16 sans agrandissement, donc chargé le registre 1 de la valeur binaire : 11100010, soit $E2 (les bits 2 à 7, eh bien nous en avons déjà parlé dans le N° 129).

LD BC,$E201 ; B contient $E2 et C le numéro du registre à charger de cette valeur.
CALL $47 ; appel à la routine d'écriture dans le registre VDP dont le numéro est dans C.

$C00F : alors là attention, nous allons sauter ! Du calme, pour le maintien de la logique (rapport du capitaine Kirk de l'Entreprise : à l'HHHHEBDO, ils ont de grandes oreilles pointues), nous allons nous rendre en $C094, où œuvre la routine REMP, qui à partir de maintenant va être fréquemment appelée. Cette dernière est destinée, comme son étiquette l'indique vaguement, à remplir les tables du VDP. Notez au passage, qu'elle transfère un nombre d'octets déterminé par le registre B : de la RAM pointée par DE, en VRAM pointée par HL et ce par l'utilisation du fameux sous-programme $4D de la ROM (voir ci-dessus), qui se charge d'envoyer l'octet contenu dans l'accumulateur, à l'adresse HL du VDP en toute compatibilité MSX (les vecteurs de la basse ROM, rappelez-vous).

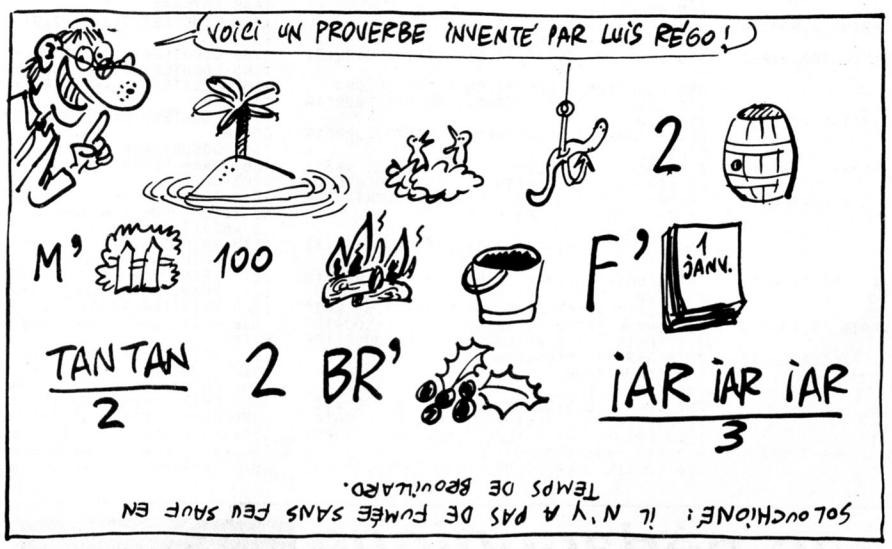
$C00F : en fait nous y voilà vraiment, juste avant c'était une prémisse. Qu'allons-nous y faire ? C'est bien simple. Nous avons 7 caractères à redéfinir, soit : a, h, i, x, é, p et ù (visibles dans le listing 1), d'où la nécessité d'une table, TCAR ($C0BD à $C102), constituée de 7 groupements d'octets ainsi constitués :
- Adresse du premier octet du caractère à modifier (premier octet du groupe de huit octets de définition dans la TGC). Précisons que cette adresse est égale à l'adresse de départ de la TGC, plus 8 fois le code ASCII du caractère, exemple : pour le premier caractère de code ASCII 97 et d'adresse TGC = 0 (BASE(7)), cela nous donne : 0 + 97 ★ 8 = 776, soit $0308.
- Les 8 octets qui vont déterminer, après transfert, la nouvelle forme du caractère pointé par l'adresse définie précédemment.

Cette table est pointée avec DE. On répète 7 fois l'opération qui consiste à transférer 8 octets de RAM en VRAM, DE pointant sur la RAM et HL initialisé de manière à pointer convenablement en VRAM. Après exécution, nos caractères à redéfinir sont en principe redéfinis et on se retrouve en, heu...

$C023 : voyez-vous, c'est comme si nous étions en ligne 80 du listing 1 et de $C023 à $C042, c'est du kif les lignes Basic 80 à 150. En utilisant les tables TDS1 à TDS7 (de $C103 à $C13A) et les tables SPR1 à SPR5 (de $C13B à $C14E), nous allons nous appliquer à définir nos sprites selon une méthode assez proche de celle utilisée dans le listing 1. Sachez que l'on peut assimiler TDS1 à A$(1) et que SPR1 est quasiment la représentation de la ligne 110.
Nous avons 5 sprites à définir, chacun basé sur 4 chaînes de 8 octets (4 ★ 8 = 32 octets) puisque nous allons utiliser des sprites de 16 ★ 16 (oui, rappelez-vous, on a décidé ça en $C009).

Bon, voilà atteint notre quota de caractères et puisque détailler tout ça réclame de l'espace... la suite dans quelques lustres.
Ah ! Au fait avant de se quitter : savez-vous ce qu'est un squelette dans un placard ? Ben c'est un journaliste du magazine "Micros MSX", qui a joué à cache-cache avec l'HHHHEBDO et qui a gagné (hilarant !)...

Nicolas BOURDIN, Jean-Claude PAULIN et SINED le barbare.



FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !

Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130 134 138 142 146 150 154 158

COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130 134 138 142 146 152 156 160

ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128 131 136 141 144 146 153 157 161

AMSTRAD -» 111 115 119 123 127 131 135 139 143 146 151 155 159

APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132 135 139 143 146 151 155 160

SPECTRUM -» 112 116 120 124 127 132 136 140 144 146 153 157 162

THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133 137 141 146 152 156 159

MSX -» 113 117 121 125 129 133 137 140 146 150 154 158

LANGAGE MACHINE SUR ZX 81

```
REM * REGISTRE AUXILIAIRE D'AFFICHAGE

REM * COPIE D'ECRAN
      LD HL,(16396)
      LD DE,9000
      LD BC,793
      LDIR
      RET

REM * LECTURE INSTANTANEE
      LD HL,9000
      LD DE,(16396)
      LD BC,793
      LDIR
      RET

REM * LECTURE EN SCROLLING A DROITE
REM * COMPTEURS ADRESSES
      LD H,32
      LD L,24
      LD BC,9000
      LD DE,(16396)
:L11  INC BC
      INC DE
      PUSH DE
      PUSH BC
:L10  PUSH HL

REM * 1 COLONNE
      LD A,(BC)
      LD (DE),A
      LD HL,33
      ADD HL,BC
      PUSH HL
      POP BC
      LD HL,33
      ADD HL,DE
      PUSH HL
      POP DE

REM * DEC COMPTEUR HL
      POP HL
      DEC L
      JR NZ,L10
      DEC H
      JR NZ,L10

REM * TEMPORISATION
      LD B,8
:L13  LD C,255
:L14  DEC C
      JR NZ,L14
      DEC B
      JR NZ,L13

REM * CHANGEMENT COLONNE
      POP BC
      POP DE
      JR L11

REM * RETOUR
:L12  POP BC
      POP DE
      RET

REM * PERMUTATION VISIBLE
      EN SCROLLING A GAUCHE
REM * POINTEUR DE COLONNE
      LD BC,1
:L5   PUSH BC

REM * 1E COL FA1 DANS PRBUFF
      LD HL,(16396)
      INC HL
      PUSH HL
      LD DE,16444
      LD C,24
:L0   PUSH BC
      LDI
      LD BC,32
      ADD HL,BC
      POP BC
      DEC C
      JR NZ,L0

REM * SCROLLING GAUCHE 1 TOUR
      LD B,31
      POP HL
      INC HL
:L2   PUSH HL
      LD C,24
:L1   PUSH BC
      LD A,(HL)
      DEC HL
      LD (HL),A
      POP BC
      DEC C
      LD DE,34
      ADD HL,DE
      JR NZ,L1
      DEC B
      POP HL
      INC HL
      JR NZ,L2

REM * 32E COL FA2 DANS FA1
      LD HL,9000
      POP BC
      ADD HL,BC
      INC C
      PUSH BC
      PUSH HL
      LD HL,(16396)
      LD DE,32
      ADD HL,DE
      EX DE,HL
      POP HL
      PUSH HL
      LD C,24
:L3   PUSH BC
      LDI
      LD BC,32
      ADD HL,BC
      PUSH HL
      EX DE,HL
      ADD HL,BC
      EX DE,HL
      POP HL
      POP BC
      DEC C
      JR NZ,L3

REM * TEMPORISATION
      LD D,20
:L6   LD E,255
:L7   DEC E
      JR NZ,L7
      DEC D
      JR NZ,L6

REM * BUFFER DANS FA2
      LD HL,16444
      POP DE
      LD C,24
:L4   PUSH BC
      LDI
      PUSH HL
      LD HL,32
      ADD HL,DE
      EX DE,HL
      POP HL
      POP BC
      DEC C
      JR NZ,L4

REM * FIN OU CODA
      POP BC
      LD A,C
      CP 33
      RET Z
      JR L5

-------------------------

Caractéristiques:
- Registre auxiliaire d'affichage
- Chargement (copie)
- Lecture immédiate
- Lecture en scrolling à droite
- Permutation visible en scrolling
  à gauche
- Longueur = 203 octets
- Routine relogeable (ici FA2 est
  situé entre 9000 et 9793)
- Adresses relatives:
  . Chargement:          0
  . lecture immédiate:   +12
  . Lecture en scroll:   +25
  . Permutation:         +81
  . Fin:                 +203
```

De pauvres usagers du ZX81 souffrent chaque jour un peu plus des limites que leur impose la conception volontairement simplifiée d'un matériel trop souvent déconsidéré par les ignorants. J'ai tâché, dans ces colonnes, de leur insuffler un peu de courage pour affronter l'ascétisme technologique auquel leur bourse plate les condamne. Pourtant, je sais que certains d'entre eux, épuisés par le minimalisme informatique du Bon Lord, dérivent, hagards, hors de l'étroit chemin du purisme intellectuel, pour s'effondrer, à la limite de l'anémie, dans les emballages de cartons et polystyrène expansé d'un nouveau micro qui, hélas, n'amortira qu'imparfaitement leur chute !
Ils ont cédé à la tentation du "tout beau, tout neuf", du clinquant, de l'inutile. Jadis anges du logi-ciel ils ont chû au milieu des étrons de la société de consommation et, ce qui est beaucoup plus grave, ils ont du même coup abandonné la sainte liturgie de l'assembleur sur ZX, auquel je les conviais périodiquement. A ces RENEGATS je souhaite les pires déconvenues : des tubes pas cathodiques, des mémoires complètement mortes, des claviers pince-doigts et de nombreux illogiciels ! Qu'ils sachent bien que j'ai d'ores et déjà pris de diaboliques dispositions, pour que les listings de l'HHHHebdo qui ne concernent pas le ZX81, soient truffés d'erreurs et les pages semées de coquilles où leur bel enthousiasme néophyte viendra se casser les dents !
Fallait pas m'laisser comme ça...

Quant aux autres, vous tous mes très très chers amis, comment allez-vous ? Et madame votre mère ? Bien, très bien j'espère. Donnez-moi vite de vos nouvelles, ça me fera tellement plaisir. Et surtout, si je peux quelque chose pour vous, n'hésitez pas, demandez-moi n'importe quoi; je serais tellement heureux de vous satisfaire. Quoi ? vous voudriez un registre auxiliaire d'affichage pour votre ZX. Mais bien sûr ! Mais avec plaisir !... Quoi ? Vous aimeriez qu'il soit permutable avec le fichier d'affichage et que son apparition à l'écran s'effectue en scrolling à gauche, derrière l'image initiale qu'il viendrait remplacer, tandis qu'elle serait simultanément stockée dans ce registre auxiliaire ? Mais avec plaisir ! Tout de suite !...

Le listing du REGISTRE AUXILIAIRE D'AFFICHAGE se compose de quatre programmes indépendants :
- Le premier permet d'obtenir une copie conforme de l'écran. L'espace mémoire alloué à cette copie constitue le registre auxiliaire d'affichage. Il est situé dans notre exemple, aux adresses 9000 à 9793 d'une MEMOPAK 64K, mais vous pouvez aussi bien l'installer sur votre 16K dans un espace protégé par une RAMTOP modifiée suivant la procédure habituelle.
- Le deuxième programme permet d'afficher instantanément à l'écran le contenu de ce registre auxiliaire d'affichage (FA2).
- Le troisième programme autorise l'apparition à l'écran (en scrolling à droite) du contenu de FA2 après qu'il ait été transféré dans le fichier d'affichage habituel (FA1). A ce moment le contenu de FA1 se voit écrasé par celui de FA2.
- Le quatrième programme vise a supprimer cet écrasement et transforme notre registre auxiliaire d'affichage en véritable mémoire graphique, capable de stocker l'information contenue dans le fichier d'affichage, avant de venir la remplacer à l'écran dans un bel effet de scrolling à gauche. Le spectacle offert à l'utilisateur devient alors l'objectivation du processus mis en jeu dans les mémoires de la machine : permutation des contenus de FA1 et FA2, colonne après colonne, une mémoire d'écran venant chasser l'autre qui se réfugie dans la place ainsi libérée.

L'action se décompose en quatre mouvements qui structurent le programme lui-même. Cet ensemble de mouvements sera répété autant de fois qu'il y a de colonnes à modifier pour travailler sur la totalité du fichier d'affichage, soit 32 colonnes. Détaillons-les :
1 - Duplication de la colonne 0 de FA1. Pour ce faire, on a recours aux bons offices de PRBUFF, qui tient à notre disposition 33 octets dans la zone des variables-système.
2 - Scrolling à gauche de FA1.
3 - Transfert du contenu des espaces-mémoire correspondant à la colonne 31 de FA2 dans la colonne 31 de FA1.
4 - Le contenu de PRBUFF est déposé dans la "colonne 32" du registre auxiliaire d'affichage (FA2). Un pointeur, régulièrement décrémenté, aiguillera successivement le chargement sur les colonnes 31-30-29 etc.

Mais j'oubliais de vous demander des nouvelles de votre père. Ça va bien j'espère ?..

Bernard Guyot

LANGAGE MACHINE SUR MSX

CLAUSTROPHOBIE

Dans ce luxueux hebdomadaire de renommée internationale, l'espace restreint accordé à la pédagogie, nous oblige bien souvent à d'abruptes et ignominieuses séparations. Il n'est pas rare, en effet, qu'un brillant développement se trouve tronqué par l'injuste exiguïté de la surface allouée. Ah qu'il eut été agréable de pouvoir disposer d'une étendue au moins égale à celle dont dispose d'ordinaire la télé, le cinéma ou la musique. Mais bon, il est vrai qu'il n'y a pas que l'informatique dans la vie et que tout le monde a droit à l'existence..,

MÉMÉ SIXE ET SES LUTINS

Donc, à la fin de notre cours du numéro 158, nous étions sur le point de définir 5 sprites de 16 ★ 16 aux alentours de $C023.

LD B,$14 : la valeur décimale 20 qui servira de compteur au DJNZ situé en $C041, est chargée allègrement dans le registre B. Pourquoi 20 ? Tout simplement parce que nous avons à définir 5 sprites par 4 chaînes (tiens voilà, ça fait 20) de 8 octets chacun, soit 32 octets par sprite.

LD HL,$3820 : mine de rien, on place dans HL, afin de la pointer, le début de la table de définition (dessin) des sprites, soit $3820 ou 14336 en décimal (BASE(14)).

LD DE,SPR1 : DE est chargé avec l'index des datas, soit $C13B (l'équivalent de RESTORE si vous voulez). En effet, une même ligne de datas étant utilisée plusieurs fois, on stocke dans DE l'index qui pointera la ligne à utiliser pour définir le sprite en cours.

LD A,(DE) : A se trouve chargé du contenu de l'index qui est dans DE. la valeur de A est en fait le décalage de HL par rapport à TDS1. Bon, il est grand temps de détailler un peu tout ça, vu le cirage pressenti. Voyons comment les tables SPR et TDS permettent d'accéder aux 32 octets de définition d'un sprite. Tiens, Le sprite 2 par exemple :

Examinons SPR2 ($C13F) : nous trouvons les quatre octets $10, $00, $18 et $00. Ces derniers sont en fait des index à ajouter à l'adresse de TDS1. Ainsi, on pointe sur les quatre chaînes de 8 octets qui vont former les 32 octets du sprite 2.

L'adresse de TDS1 + $10 = l'adresse de TDS3 ($C103 + $10 = $C113).
L'adresse de TDS1 + $00 = l'adresse de TDS1 ($C103 + $00 = $C103).
L'adresse de TDS1 + $18 = l'adresse de TDS4 ($C103 + $18 = $C11B).
L'adresse de TDS1 + $00 = l'adresse de TDS1 ($C103 + $00 = $C103).

Donc, les chaînes qui définissent le sprite 2 sont : TDS3, TDS1, TDS4 et TDS1. D'où une flagrante similitude avec la ligne 120 du programme Basic qui est :
120 SPRITE$(2) = A$(3) + A$(1) + A$(4) + A$(1).
Conclusion : en procédant de la sorte, nous avons économisé de nombreux octets tout en obtenant le résultat escompté. Les 32 octets du sprite 2 sont bien :

$00,$00,$00,$00,$1F,$3F,$7F,$FF
$00,$00,$00,$00,$00,$00,$00,$00
$00,$00,$00,$00,$E0,$C0,$80,$00
$00,$00,$00,$00,$00,$00,$00,$00

PUSH DE et PUSH HL : Les registres pointeurs sont sauvegardés (empilés), histoire de ne pas perdre le fil en cours, ou le cours du fil au cours du cours trop court.

LD D,0 et LD E,A : de la sorte, le registre double DE se trouve chargé de la valeur de décalage que contient A (donc, DE = A).

LD HL, TDS1 : on place dans le registre HL, l'adresse de début des données ($C103) définissant les sprites.

ADD HL,DE : le contenu de HL est additionné au contenu du registre DE. Le résultat est placé dans HL. Ainsi, on pointe avec HL les datas désirés (le principe est expliqué ci-dessus).

PUSH HL et POP DE : on empile HL et on dépile DE. Ruse d'enfer qui permet d'échanger le contenu de ces deux registres (SWAP). On aurait pu faire LD D,H et LD E,L ou plus simplement EX DE,HL, mais nous devons rester fidèles à notre réputation de génies méconnus.

POP HL : on récupère l'adresse de définition des sprites ($3820), 'pushée' rappelez-vous en $C035.

PUSH BC : on sauvegarde BC, qui contient le nombre de paquets (de 8 octets chacun) qu'il nous reste à transférer.

LD B,8 : on charge B du nombre d'octets (8) constituant chaque paquet. Cela bien sûr, à l'intention du DJNZ de la routine REMP que nous allons appeler sans plus tarder.

CALL REMP : tiens, quand on parle du loup : ici, on envoie tout ce joli monde en VRAM par la grâce de la routine REMP située en $C094. Mais comment se fait-il que nous n'ayons pas encore détaillé cette routine ? Alors voilà :
- A est chargé du contenu de DE.
- CALL $4D place la valeur de A à l'équivalent en VRAM de l'adresse contenu dans HL.
- HL est incrémenté ainsi que DE.
- DJNZ REMP boucle le total avec décrémentation de B, jusqu'à ce que B = 0.

POP BC et POP DE : on dépile nos valeurs préalablement sauvegardées, soit BC le nombre de paquets et DE l'index (SPR) des datas des sprites.

INC DE : DE est incrémenté afin d'accéder au paquet suivant.

DJNZ LOOP : le compteur B initialisé à $14 (20) en $C023 est décrémenté par l'action de DJNZ et on boucle en $C02B s'il nous reste des paquets à transférer (tant que B n'est pas égal à 0).

Si vous n'avez rien compris, relisez tout cela bien calmement depuis le début et vous sentirez naître en vous la grâce de la suprême révélation.

LES COLORIAGES DE MÉMÉ SIXE

LD DE,TOTO : de $C043 à $C04D, nous allons nous activer à remplir la table des couleurs (TC), comme en lignes 160 et 170 du programme Basic. DE est chargé de $C14F (étiquette TOTO), où figurent nos 6 couleurs ou plutôt 7 (vu qu'il y en a une qui compte pour du beurre), soit : $81, $8D, $D1, $87, $41, $00 (rien à cirer en 8209) et $71.

LD HL,$200C : on place dans HL, l'adresse 8204 ($200C) de la table des couleurs.

LD B, 7 : et dans B le nombre de couleurs.

CALL REMP : le mécanisme de REMP n'est plus à décrire, vu qu'il a déjà été décrit lorsqu'on l'a décrit.

L'AFFICHAGE DE MÉMÉ SIXE

LD HL,$18ED : de $C04E à $C060, nous allons assurer l'équivalent des deux PRINT des lignes 190 et 200 du programme Basic. Les LOCATE sont ici réalisés d'après les adresses chargées dans HL, en $C04E et $C059. Ces dernières sont créées par la formule :
Adresse d'un emplacement de caractère de coordonnées X et Y = adresse de début de la table des noms de configurations (TNC, soit $1800, valeur décimale 6144 de BASE(5) en SCREEN 1), plus 32 que multiplie Y - à condition toutefois d'être en 32 colonnes - auquel est ajouté X. C'est un peu compliqué hein ? Alors lancez le petit programme Basic suivant :

```
10 SCREEN1 :WIDTH32
20 LOCATE11,7 :PRINT"a"
30 VPOKE6144 + (32★7) + 11,97
40 GOTO 40
```

On constate qu'un seul "a" est affiché à l'écran, puisque les lignes 20 et 30 font double emploi : C.Q.F.D.

A noter que : ?HEX$(6144 + (32 ★7) + 11) = $18EB et non pas $18ED comme dans le source. Pourquoi ? Tout simplement parce que nous ne sommes pas en 32 colonnes, alors on s'est permis de rajouter 2 octets histoire de tomber juste. Faites donc :

```
10 SCREEN1 :WIDTH29
20 LOCATE11,7 :PRINT"a"
30 VPOKE&H18ED,97
40 GOTO 40
```

Un seul "a" est affiché : C.Q.F.D dis donc.

LD DE,ECR1 : DE est chargé de l'adresse pointant les 4 caractères de la première ligne à afficher, soit $C156. A noter que l'assembleur utilisé nous a permis de rentrer entre apostrophes, ces caractères tels quels dans le programme source. La valeur placée en mémoire lors de l'assemblage étant, bien sûr, de forme ASCII.

LD B,4 : on place dans B, le nombre de caractères de cette ligne, soit 4.

CALL REMP : branchement à le routine REMP, qui n'en peut plus de remplir tellement elle remplit.

On réitère l'opération pour la seconde ligne de caractères, à placer à partir de l'adresse TNC $190D.

Allez, il suffit pour aujourd'hui. Ah, au fait, avant de se quitter : A quoi reconnaît-on qu'un journaliste de 'Micros MSX' s'est servi d'un traitement de texte ? Aux traces de Tipp-Ex sur l'écran du moniteur (Mmmpppfff, hilarant !).

Nicolas BOURDIN, Jean-Claude PAULIN et Sined le Barbare.

FORMATION A L'ASSEMBLEUR PRATIQUE

LA REVOLUTION CONTINUE !
Les micros, ci-après nommés, ont déjà hérité d'un nombre de cours conséquents dans les numéros ci-dessous décrits.

ZX 81 -» 55 56 61 66 71 76 81 86 91 95 100 105 110 114 118 122 126 130 134 138 142 146 150 154 158 162

COMMODORE -» 60 65 70 75 80 85 90 95 99 104 109 114 118 122 126 130 134 138 142 146 152 156 160 164

ORIC -» 57 62 67 72 77 82 87 92 96 101 106 111 115 119 123 128 131 136 141 144 146 153 157 161 166

AMSTRAD -» 111 115 119 123 127 131 135 139 143 146 151 155 159 163

APPLE -» 58 63 68 73 78 83 88 93 97 102 107 112 116 120 124 128 132 135 139 143 146 151 155 160

ATARI -» 163

SPECTRUM -» 112 116 120 124 127 132 136 140 144 146 153 157 161 164 165

THOMSON -» 59 64 69 74 79 84 89 93 98 103 108 113 117 121 125 129 133 137 141 146 152 156 159 165

MSX -» 113 117 121 125 129 133 137 140 146 150 154 158 162 166

LANGAGE MACHINE SUR *APPLE*

BONNE ANNÉE

Ha ! Le nouvel an et son cortège de promesses et de bonnes résolutions. Tiens, par exemple, je promets à Jean-Claude de rendre mes cours à temps et ceci dès maintenant (ça ne mange pas de pain, puisque l'article est déjà paru !). Je fais également le serment de publier des routines qui fonctionnent bien (ce qui signifie en d'autres termes, que je me porte garant des personnes qui les tapent : je cite Stéphane et Cyrille; mais n'est-ce pas leur accorder une trop grande confiance ?). Bon, mais ce n'est pas tout, après ce préambule, il s'agit de programmer en langage machine.

TRI SIMPLE

Comme l'indique le titre de ce paragraphe, je vais vous parler de tri. Voyons voir la façon de procéder. On part de 'n' nombres répartis dans un ordre quelconque. Ensuite, on considère le 'n'ième (le dernier de la liste) et on le compare au 'n'ième - 1. S'il est plus petit, on permute les deux nombres, sinon, on laisse les choses telles quelles sont.

Puis on renouvelle l'opération : comparaison du 'n'ième nombre avec le 'n'ième - 2 et ainsi de suite jusqu'au premier. Enfin, on recommence tout depuis le début, mais en partant du 'n'ième - 1 et ceci jusqu'au premier nombre. Voici un petit exemple pour clarifier vos idées, qui je n'en doute pas, doivent être embrouillées. Supposons que vous désiriez classer en ordre croissant les nombres : 7, 6, 1, 4, 2. Nous aurons comme étape :

7, 6, 1, [4 2]

7, [6 1, 2, 4]

[7 4, 1, 2, 6]

6, [4 1, 2] 7 : on ne touche plus au 7.

[6 2, 1, 4] 7

4, [2 1] 6, 7 : on ne touche plus au 6.

[4 1, 2] 6, 7

[2 1] 4, 6, 7 : on ne touche plus au 4.

1, 2, 4, 6, 7 : c'est fini !

Les paires encadrées, correspondent aux nombres à permuter. Il ne vous reste plus qu'à mettre la chose en pratique. Voici le listing :

```
$1000  98          TYA
$1001  AA          TAX
$1002  CA          DEX
$1003  B9 00 60    LDA$6000,Y
$1006  DD 00 60    CMP$6000,X
$1009  B0 0B       BCS$1016
$100B  48          PHA
$100C  BD 00 60    LDA$6000,X
$100F  99 00 60    STA$6000,Y
$1012  68          PLA
$1013  9D 00 60    STA$6000,X
$1016  8A          TXA
$1017  D0 E9       BNE$1002
$1019  88          DEY
$101A  D0 E4       BNE$1000
$101C  60          RTS
```

Il est bon de préciser que Y contient le nombre d'éléments moins un (Y ≠ 0), les nombres étant rangés à partir de $6000. Cette routine effectue un classement par ordre croissant, occupe 20 octets et est relogeable.

DETAILS

Expliquons le déroulement de la routine en pas à pas : aux adresses $1000 - $1001, Y est recopié dans X, puis en $1002, X est décrémenté de 1. La comparaison du dernier nombre au précédent, se fait en $1003 - 100A : s'il est plus grand, on se rend en $1016, sinon, on permute les deux nombres ($100B - $1015), en stockant temporairement le premier dans la pile. Puis on arrive en $1016, où on tranfère X dans A, afin de placer les indicateurs du registre d'état comme si on avait fait un CPX # $00 (mais on gagne un octet par cette méthode; comme on dit : misons petit, optimisons !). Si X est non nul, on se branche en $1002 pour faire une nouvelle boucle (on décrémente X...), sinon (le BNE en $1017 n'étant pas réalisé), on décrémente Y et on va en $1000 pour effectuer la boucle principale ($1019 - $101A) tant que Y≠ 0. La fin de la routine s'effectue par le 9e désormais classique RTS en $101C.

SUGGESTIONS

Pour terminer avec le tri simple, vous pouvez modifier l'adresse d'implantation des nombres (actuellement $6000) et classer les octets non plus par ordre croissant, mais par ordre décroissant (remplacer le BCS en $1009 par un BCC). Enfin, précisons qu'il existe d'autres méthodes de tri beaucoup plus rapides comme le tri à bulle, le tri par insertion, le tri arborescent, etc. Nous y reviendrons une prochaine fois.

SAISIE D'UN CARACTERE

Voici une routine qui devrait remplacer le KEYIN de la ROM. Elle affiche un curseur, attend qu'une touche soit frappée et retranscrit le caractère à l'écran. De plus, on utilise le temps mis entre le démarrage et la frappe de la touche pour créer un nombre aléatoire de 0 à 255.

L'utilisation de ce programme est simple :
Initialiser $06, $07 avec l'adresse de la ligne écran où placer le curseur. Placer dans Y, le numéro de la colonne où apparaîtra ledit curseur. Enfin sachez que le nombre aléatoire (un octet) est situé en $4E (page zéro).
Allez, je ne vous laisse pas languir plus longtemps et je vous offre le listing :

```
$900   2C 10 C0   BIT$C010
$903   E6 4E      INC$4E
$905   A9 A0      LDA#$A0
$907   91 06      STA($06),Y
$909   A9 39      LDA#$39
$90B   20 A8 FC   JSR$FCA8
$90E   AD 00 C0   LDA$C000
$911   30 0E      BMI$921
$913   A9 20      LDA#$20
$915   91 06      STA($06),Y
$917   A9 39      LDA#$39
$919   20 A8 FC   JSR$FCA8
$91C   AD 00 C0   LDA$C000
$91F   10 E2      BPL$903
$921   91 06      STA($06),Y
$923   60         RTS
```

Soit : 36 octets et le tout entièrement relogeable.

ANALYSE LIGNE PAR LIGNE

Le BIT$C010 ordonne au clavier d'attendre une nouvelle touche ($C010 est un 'softswitch'). Ensuite, on incrémente le conteur de $4E (nombre aléatoire) et on stocke dans l'écran le curseur éteint (espace), grâce aux instructions : LDA # $A0 et STA($06),Y. On exécute ensuite une temporisation (utilisation de la routine WAIT de la ROM, logée en $FCA8): adresses $909 à $90D. On lit alors le clavier par un LDA$C000 et si le résultat est négatif (qui signifie qu'une touche a été enfoncée), on fait un 'jump' en $921 (BMI$921). La séquence LDA # $20, STA($06),Y qui suit, affiche à l'écran un pavé blanc (le curseur). On effectue de nouveau une temporisation (LDA # $39 et JSR$FCA8) et on teste l'état du clavier. Si le résultat est positif, i e, aucune n'a été pressée : on saute en $903 (retour au début, incrémentation du nombre aléatoire, etc.). Sinon (le BPL$903 n'ayant pas eu lieu), on stocke à l'écran, le caractère qui a été rentré (STA($06),Y). Enfin, as usual, on termine par un RTS.
Libre à vous de modifier le curseur (adresses $906 et $914), ou de changer les temporisations (adresses $90A et $918) pour obtenir un clignotement plus rapide. Vous pouvez aussi rajouter un 'beep' quand une touche est validée. Cette routine peut constituer le noyau d'un éditeur (prendre alors en compte les touches ←, →, RETURN), avec autorepeat automatique et réglable.

FAIN

Après cet article d'une haute tenue et d'un intérêt pédagogique certain, je vous salu bien bas. Quant à moi, il ne me reste plus qu'à mettre la viande dans le torchon (expression paysanne bien de chez nous).

Philippe PIERNOT
(surnommé JACQUET).

LANGAGE MACHINE SUR *ZX 81*

```
REM * DOUBLE CLEF

REM * MESSAGE D'OUVERTURE
      LD DE,(16396)
      INC DE
      LD HL,10440     adr mess
      LD BC,32
      LDIR

REM * ADRESSE COMPT/BOITES
:L8   LD BC,7
      PUSH BC
      LD DE,16444     PR-BUFF
      PUSH DE

REM * SAISIE DE TOUCHE
:L2   BIT 0,(IY+$3B)
      JR Z,L2
      LD BC,($4025)
      CALL $0F4B
      CALL $07BD
      LD A,(HL)

REM * CHARGE/DEC C
      ADD A,6B
      POP DE
      LD (DE),A
      INC DE
      POP BC
      DEC C
      JR Z,L3
      PUSH BC
      PUSH DE
      JR L2

REM * COMPARAISON
:L3 LD HL,16444     PR-BUFF
      LD BC,16526   adr clef 1
      LD D,8
:L5   LD A,(BC)
      DEC D
      JR Z,L6
      CPI
      JR NZ,L4
      JR L5

REM * ECHEC->INV VIDEO
:L4   LD HL,(16396)  F,A,
      LD B,24
:L7   INC HL
      LD A,(HL)
      ADD A,$80       inv vidéo
      LD (HL),A
      DJNZ,L7
:L14 JR L6

REM * REUSSI->PERMUT CLEF
:L6   LD DE,16532     adr clef 2
      LD HL,16520     adr clef 1
      LD B,7
:L15 LD A,(HL)
      PUSH AF
      LD A,(DE)
      LD (HL),A
      POP AF
      LD (DE),A
      INC DE
      INC HL
      DJNZ,L15

REM * VERIF D-FILE
      LD DE,(16396)   D-FILE
      LD A,(16544)    octet 1
      CP D
      JR NZ,L14
      LD A,(16543)    octet 2
      CP E
      JR NZ,L14

REM * 3 FEUX MIS AU VERT
      LD HL,(16554)
      LD A,(HL)
      PUSH DE
      LD DE,(16549)
      LD (DE),A
      LD DE,(16552)
      LD (DE),A
      POP DE

REM * DECODAGE ET
      AFFICHAGE MESSAGE
      LD HL,1
      ADD HL,DE
      LD DE,10472     adr cpght
      LD B,32
:L20 LD A,(DE)
      SUB 37
      LD (HL),A
      INC HL
      INC DE
      DJNZ,L20

REM * DEROUL.AFFICH.
      LD D,2
:L17 LD HL,(16396)
      LD B,32
:L15 INC HL
      LD A,(HL)
      ADD A,$80
      LD (HL),A
      LD E,18
:L18 LD C,255
:L16 DEC C
      JR NZ,L16
      DEC E
      JR NZ,L18
      ADD A,$80
      LD (HL),A
      DJNZ,L15
      DEC D
      JR NZ,L17
      RET           ou enchaîner sur
                    la routine de trucage LM
                    réversible.
```

```
CARACTERISTIQUES:
- Longueur: 256 octets (y compris la
  routine de détrucage)
- Routine non-relogeable
- Départ en 10184
- Adresses réservées:
      D-FILE:      16543/16544
      COULEUR FEU: 16527
      FEUX:        16319/16551
      BUFFER:      10440 à 10523
- Feux mis au "vert":
      16319:       16549/16550
      16551:       16552/16553
      16527:       16554/16555
-40 octets réservés en début de
 programme:        16514 à 16554
```

Voici le temps venu d'apporter à la jeunesse pré-délinquante de notre beau pays post-démocratique, ma modeste mais essentielle contribution à sa formation littératurotechnique. Conscient de l'importance de mon rôle de pédagogue pour génies déclassés - parias des universités et laissés-pour-compte de toutes réformes passées, présentes et à venir - je resterais, quant à moi, fidèle à ma ligne pédagogique, tout en l'adaptant à la progression flagrante de nos cours d'assembleur. Cette ligne successivement brisée, discontinue, souterraine puis latente, s'épanouit aujourd'hui dans l'entrelacs d'une pédagogie diffuse, les rets d'un savoir qui ne peut plus désormais vous échapper.
Ô vision lumineuse de ces cohortes d'Enseignés, de bienheureux aux visages calmes dont l'humble livrée de chauffeur-livreur ou d'aide-magasinier est la trop modeste parure d'un savoir gigantesque ! Je vous le dis, votre humilité fera de vous des saints, mais ne comptez pas pour autant sur une place dans le calendrier : là aussi c'est bouché !

Par contre les estimations prévisionnelles de l'ANPE pour les dix prochaines années laissent apparaître deux secteurs d'activité en forte expansion : d'une part les ANPE, d'autre part la serrurerie électronique dans laquelle, d'ores et déjà, de nombreux jeunes ont trouvé une raison de vivre. Le programme qui suit est destiné à susciter d'autres vocations...

Notre routine dite "DOUBLE CLEF" autorisera l'accès aux programmes qu'elle "protégera" à partir d'un mot de passe alphanumérique que vous choisirez à votre convenance. Elle sera directement appelée à partir du listing basic, par une instruction RAND USR 10184 qui suivra immédiatement l'instruction SAVE à partir de laquelle l'ensemble du programme aura été sauvegardé. A la lecture de votre cassette, le programme démarre automatiquement sur la routine LM qui affiche, sur la première ligne du fichier d'écran, un message convivial du genre : "votre clef sinon ce programme s'auto-

détruira dans les dix secondes". L'usager doit alors taper un code composé de sept signes alphanumériques. La saisie des touches frappées est automatique pour chaque caractère et ne nécessite donc pas de "new-line". Si le code n'est pas conforme à la clef - que vous aurez mis en place préalablement dans une ligne REM située en tête de listing - la question initiale sera reposée en vidéo inversée et ainsi de suite jusqu'à ce que la bonne réponse soit fournie au logiciel. Si le code s'avère être correct, le travail de la routine se poursuit à travers plusieurs étapes.
La première consiste à substituer au code qui vient d'être utilisé, une seconde clef de sept signes, elle aussi tenue en réserve dans la première ligne de REM, très précisément aux adresses 16532 à 16538 (tandis que la clef 1 réside de 16520 à 16526).
L'étape suivante vérifie que le programme protégé n'a pas été modifié à l'occasion d'une copie, en comparant la valeur actuelle de D-FILE, à celle que vous aurez préalablement sauvegardée aux adresses 16543 et 16544 prévues à cet effet. En cas de comparaison négative, le programme "boucle" et bblige l'usager à... débrancher son ZX. Si, au contraire, D-FILE est resté ce qu'il était à l'origine, le programme se poursuit par la mise au "vert" de trois "feux". Il s'agit de trois octets bien cachés dont les adresses sont rangées respectivement en... Mais voyez ça sur le listing. Vous remarquerez que deux de ces octets sont situés sur la première ligne de REM, tandis que le troisième est installé, invisible, en amont des variables-système; tout ça dans l'intention machiavélique de brouiller les pistes.
Vous déciderez de la "couleur verte" en pokant avec la valeur de votre choix l'octet 16527. Ces feux peuvent, à tout moment, être contrôlés par le programme principal, afin de prévenir toute utilisation intempestive de l'une ou l'autre de ses parties.
Ceci fait, la routine de DOUBLE CLEF ne rendra pas la main au basic avant d'avoir affiché un message de bienvenue ou un quelconque copyright. Ce message dispose des 32 octets de la première ligne du fichier d'affichage et son apparition est rendue plus attrayante par un balayage, case par case, de chacun des caractères qui le compose, dont la vidéo s'inverse un instant. Deux tours sont ainsi accomplis grâce au module de "déroulement d'affichage".

Vous pourrez utiliser cette routine pour protéger tous les programmes qui supporteront l'ajout d'une première ligne de REM, soit 40 octets réservés à partir de l'adresse 16514, et qui contiendront vos codes sous une forme dissimulée (une série de points d'interrogation obtenus par addition du facteur 68 à leurs valeurs CHR§), ainsi que de nombreux leurres pour fausser la piste sur laquelle pourraient s'aventurer de malins pirates.
Il vous faut également disposer d'une 64K puisque, dans la version présentée, les différents messages doivent être stockés dans une zone RAM inexistante sur 16K. Je signale à ceux qui n'auraient que 16K à leur disposition que, si la routine n'est pas relogeable, elle est en revanche facilement transposable moyennant quelques changements d'adresses et la procédure classique d'une zone RAM réservée et inaccessible au NEW. On peut également implanter les différents messages sur la ligne de REM, en les codant d'une façon analogue aux clefs et aux leurres.
Un programme basic simplifiant l'entrée des codes, leurres et messages paraîtra dans le prochain cours.

Bernard Guyot