ATARI®

UTILITAIRES DE DEBOGAGE POUR ASSEMBLEUR ET MACRO-ASSEMBLEUR

David KANO
Jim DUNION

© 1982 David KANO
© 1982 Jim DUNION
Droits réservés pour tous pays.

ATARINSIDE

UTILITAIRES DE DEBOGAGE POUR ASSEMBLEUR ET MACRO-ASSEMBLEUR

David KANO
Jim DUNION

© 1982 David KANO
© 1982 Jim DUNION
Droits réservés pour tous pays.

Copyright et copie :

A l'achat de ce programme pour ordinateur et de sa documentation associée (le logiciel), vous obtenez le droit d'utiliser ce logiciel pour votre usage personnel seulement sans en effectuer de copies. Ce logiciel est déposé. Il vous est interdit de le reproduire, le traduire ou le dupliquer sans l'autorisation écrite d'ATARI Corp.

ATARINSIDE

GARANTIE

Conformément à la loi, la présente vente est soumise à la garantie légale des défauts et vices cachés.

Nous garantissons, pendant 30 jours après la date d'achat, que le support sur lequel ce programme ATARI® est enregistré, ne comporte aucun défaut.

Si toutefois, il se trouvait que ce programme ne puisse se charger normalement, et que la disquette, la cassette ou la cartouche ci-incluses en soit la cause, veuillez le rapporter à votre revendeur avec une preuve d'achat datée, afin qu'il puisse appliquer la garantie qui se borne strictement à l'échange de ce programme par un autre identique, dans les meilleurs délais.

Cette garantie ne s'applique plus dès lors que le support montre des signes évidents et anormaux d'usure, de contraintes mécaniques (pliures déformations dues à la chaleur, froissage de la bande, tentative de démontage ou d'ouverture du support, etc.) ou de mauvaise utilisation ou détérioration (renversement d'un liquide sur le support, empreinte de doigts sur les parties magnétiques, altération électromagnétique, etc.). La garantie est aussi exclue si ce produit n'est pas d'origine ATARI ou s'il a été modifié par quiconque autre que par les techniciens ou ingénieurs d'ATARI.

L'acheteur est tenu, dès son acquisition, de mettre à l'épreuve le logiciel de ce programme ATARI, de vérifier la véracité de ses résultats et de signaler sur le champ toute anomalie éventuelle à son vendeur afin que celui-ci puisse en faire vérifier l'exactitude par ATARI en le retournant pour son remplacement, dans les meilleurs délais.

ATARINSIDE

UTILITAIRES DE DEBOGAGE POUR ASSEMBLEUR
ET MACRO ASSEMBLEUR

(réf. : DXF 54012 D)

L'outil que vous venez d'acquérir contient deux programmes :

face 1 : **HEXABUG de David KANO**
(sommaire page iii)

face 2 : **D.D.T. de Jim DUNION**
(sommaire page 25)

ATARINSIDE

REMARQUE IMPORTANTE

POUR DES PROBLEMES DE COMPATIBILITE, L'ENSEMBLE DES PROGRAMMES CONTENUS DANS CE KIT FONCTIONNENT AVEC LE DOS 2.0, C'EST A DIRE EN SIMPLE DENSITE.
SI VOUS TRAVAILLEZ HABITUELLEMENT AVEC LE DOS.3 VOUS POURREZ CEPENDANT REUTILISER LES DIFFERENTS FICHIERS PRODUITS PAR CES DISQUETTES. POUR CELA, IL SUFFIT DE RECOPIER VOTRE FICHIER SOUS DOS 2 SUR UNE DISQUETTE DOUBLE DENSITE EN UTILISANT L'OPTION A DU DOS 3 (ACCESS DOS II).

ATARINSIDE

SOMMAIRE
HEXABUG

Un outil de mise au point de programmes en langage machine.

ATARINSIDE

INTRODUCTION

VUE GENERALE

Lorsque la vitesse d'exécution, l'utilisation optimale de la mémoire, et l'exploitation de toutes les possibilités des ordinateurs ATARI sont importantes dans un programme, il devient nécessaire d'utiliser le langage d'assemblage. Mais la nature non-interprétée des assembleurs les rend plus délicats à mettre au point que cette opération ne l'est pour des langages de haut niveau comme par exemple le BASIC. HEXABUG permet de contrer ce principal défaut des assembleurs.

Vous chargez HEXABUG, outil de mise au point simple à utiliser et simple à comprendre, simultanément avec votre programme cible, dans la mémoire de l'ordinateur. En utilisant des points d'arrets, vous suivez facilement le cheminement du programme ainsi que tous les résultats intermédiaires (cases mémoires ou contenus des registres du microprocesseur). En outre, l'image construite par votre programme cible est protégée ; lorsque vous relancerez le programme, son écran associé ré-apparaitra.

Cet outil utilise peu de commandes. La majeure partie de l'écran se compose de "bandes" ou "zones", chaque zone représentant une partie de la mémoire, un ensemble de registres, etc. Vous déplacez un curseur clignotant de zone en zone, ce qui vous permet de modifier le contenu de n'importe quel registre, point d'arret, adresse de début de zone ou case mémoire très simplement. Un défilement horizontal de chaque bande offre un accès rapide et simple à toute information. Enfin, quelques commandes vous donnent des possibilités supplémentaires : fonctionnement pas à pas, retour au DOS, recherche d'une suite d'octets en mémoire, reprise du déroulement du programme, etc. Les utilisateurs de l'Editeur de Programmes et de Textes (qui fait partie du programme Macro-Assembleur commercialisé par ATARI) constateront qu'HEXABUG est particulièrement simple à utiliser car le mouvement du curseur, les touches de fonction, et les fenetres d'erreur et de commande fonctionnent d'une manière similaire. Avec HEXABUG, vous vous concentrez sur la mise au point de votre programme, et non sur l'outil de mise au point.

ACCESSOIRES NECESSAIRES :

- 48 ko de mémoire vive
- Une Unité de Disquette ATARI

ACCESSOIRES OPTIONNELS

Macro-assembleur ATARI ou cartouche ASSEMBLEUR EDITEUR.

ATARINSIDE

MISE EN OEUVRE

CHARGEMENT DE HEXABUG DANS LA MEMOIRE DE L'ORDINATEUR.

1. Retirez toute cartouche de l'ordinateur ;

2. Eteignez votre ordinateur ;

3. Mettez sous tension votre unité de disquette ;

4. Insérez la disquette HEXABUG dans l'unité de disquette, l'étiquette marquée HEXABUG étant orientée vers vous et vers le haut. Fermez la porte de l'unité ;

5. Mettez sous tension l'ordinateur (en appuyant sur OPTION pour les ordinateurs XL) et votre téléviseur. Le programme se chargera automatiquement et s'exécutera immédiatement.

UTILISATION DE CE MANUEL

Ce manuel sous-entend que vous ayez une bonne connaissance du langage d'assemblage et des instructions du microprocesseur 6502. Chargez HEXABUG et utilisez chacune de ses possibilités tout en lisant ces pages. N'oubliez pas de lire le chapitre "PRINCIPES DE FONCTIONNEMENT". Enfin, un lexique rappelle la définition de quelques termes.

ATARINSIDE

UTILISATION DE HEXABUG

INTRODUCTION

Avant d'utiliser HEXABUG, vous devez posséder un programme écrit en assembleur que vous mettrez au point grace à HEXABUG. Chaque fois que votre programme exécutera une fonction BRK ($00), la main sera redonnée à HEXABUG. Une seule instruction BRK suffit : vous profiterez de cet arret pour positionner les points d'arrets d'HEXABUG. Par la suite, HEXABUG reprendra le controle à chaque point d'arret, indépendemment de l'instruction placée à cet endroit. En conséquence, nous vous conseillons de placer une instruction BRK au tout début de votre programme.

Lorsque vous avez sur disquette un fichier contenant le code objet de votre programme, vous pouvez charger HEXABUG comme nous l'avons vu précedemment. Dès que l'écran d'HEXABUG apparait, utilisez l'instruction DOS pour revenir au menu du DOS. En utilisant l'option L, chargez votre programme cible en mémoire. S'il est en "AUTORUN", l'instruction BRK redonnera la main à HEXABUG, sinon vous pouvez lancer l'exécution soit de votre programme cible soit de HEXABUG grace à l'option M du DOS II (dans le cas d'HEXABUG, l'adresse à taper est $BEF0 pour un ordinateur ATARI 800 équipé de 48 ko de mémoire minimum).

Si vous testez un sous-programme, il vous faudra certainement positionner certains registres du microprocesseur, la pile ou le contenu de certaines cases. Vous pouvez ainsi simuler un appel BASIC de sous-programme. Si vous travaillez sur un grand programme, vous avez ainsi la possibilité de le diviser en plusieurs plus petits et de mettre au point chaque sous partie individuellement. Cette technique modulaire vous fait gagner du temps :

. en vous facilitant le travail de mise au point,
. en créant une bibliothèque de sous programmes faciles à ré-utiliser sans d'autres programmes.

Si votre programme ne fonctionne pas correctement, déplacez les points d'arrets de manière à encadrer de plus en plus précisement la zone où se produit l'erreur. Une fois l'erreur située, vous pourrez (en modifiant registres et mémoire) simuler un fonctionnement correct et faire reprendre l'exécution du programme cible afin de déceler d'autres erreurs éventuelles. Plus vous trouverez d'erreurs en une seule séquence, plus vous gagnerez de temps car il est toujours long de revenir à l'assembleur, d'éditer le programme source, de ré-assembler, etc.

ECRAN DE HEXABUG

HEXABUG présente de nombreux renseignements sur un seul écran. La partie principale de l'écran se compose de "bandes" ou "zones", chaque zone représentant un groupe logique d'informations sur deux lignes ou plus. Dans le bas de l'écran, vous trouvez les lignes de messages et de commande. La position actuelle du curseur est indiquée par un carré clignotant.

ATARINSIDE

Le premier écran apparait ainsi :

STRIP NAME	SAMPLE SCREEN DISPLAY	NOTES
Register strip	PC A X Y S N V B D I Z C	(INVERSE)
	3304 0F 10 00 FB 0 0 0 0 0 0 0	Fields
Breakpoint strip	BREAKPOINTS	(INVERSE)
	1 2 3 4 5 6 7	(INVERSE)
	3304 0000 0000 0000 0000 0000 0000	Fields
Stack strip	STACK	(INVERSE)
	F7 F8 F9 FA FB FC FD FE FF 00 01 02 03	
	AA AB EE FF 1F 5F 23 31 1B C1 11 12 22	Fields
Code strip	Address: 3304 1 CODE	(INVERSE)
	00 01 02 03 04 05 06 07 08 09 0A 0B 0C	
	11 22 33 44 55 66 77 88 99 01 02 03 04	Fields
Memory strip	Address: XXXX 1 MEMORY	(INVERSE)
Memory strip	Address: XXXX 1 MEMORY	(INVERSE)
	Hex-A-Bug	
	Copyright 1982 David Kano	
Memory strip	Address: XXXX 1 MEMORY	Header line
		Label line
		Data line
Message line	Change PC to CONT or use DOS cmd.	(INVERSE)
Command line	DOS COMMANDS: CONT SS SKP SE /00/<	

ATARINSIDE

Chaque ligne de l'écran est divisée en deux ou plusieurs champs. Par exemple, la zone des registres dans le haut de l'écran a un champ par registre ; la zone des points d'arret a un champ par point d'arret. Chaque champ a un nom, un numéro d'ordre, ou l'adresse associée, marqué au-dessus de lui.

Zone courante : sur l'écran, une des zones est entourée d'un filet rouge plus large que celui entourant les autres zones. Cela permet de situer la zone dans laquelle est le curseur lorsque vous n'etes pas en mode commande. En appuyant sur la touche OPTION, vous faites passer le curseur de cette zone courante à la ligne commande et vice versa. En appuyant sur les touches de déplacement du curseur (et sur la touche CTRL), vous déplacez le curseur entre champs et zones.

DEPLACEMENT DU CURSEUR

Vous pouvez déplacer rapidement le curseur sur l'écran avec peu de manipulations. Le curseur d'HEXABUG ne se positionne que sur les champs que vous pouvez modifier. Après avoir amené le curseur à l'endroit choisi, vous taperez une nouvelle valeur. Celle-ci remplacera immédiatement celle qui était en mémoire et à l'écran. Si vous appuyez sur une touche qui n'a pas de sens, un message d'erreur s'affichera pendant quelques secondes dans la ligne message et le curseur ne se déplacera pas ; l'ancienne valeur ne sera pas changée.

HEXABUG utilise les touches suivantes pour déplacer le curseur (CTRL signifie que vous devez appuyer sur la touche CONTROL en meme temps que l'autre touche indiquée) :

- CTRL ↑ : déplace le curseur d'une ligne vers le haut ;
- CTRL ↓ : déplace le curseur d'une ligne vers le bas ;
- CTRL → : déplace le curseur d'une position vers la droite ;
- CTRL ← : déplace le curseur d'une position vers la gauche ;
- TAB : amène le curseur au champ suivant sur la meme ligne ;
- DELETE BACK: amène le curseur sur le caractère précédent dans la zone (et donne la valeur 0 sauf dans une ligne donnée).

ATARINSIDE

ZONES D'ENTREES :

Les zones pile, code et mémoire contiennent une seule zone d'entrée dans leurs zones de données : au milieu de la ligne. Aussi, pour modifier une valeur dans une ligne de données, utilisez les commandes de déplacement du curseur pour amener la case voulue sous le curseur.

ZONES D'AFFICHAGES MEMOIRE

L'écran possède trois zones permettant d'afficher le contenu de la mémoire. Les zones STACK et CODE constituent deux zones complémentaires et particulières dont nous reparlerons.

Chaque zone est constituée de trois lignes. En voici un exemple :

```
(1)      Address: 4004 1      MEMORY
(2)    00 01 02 03 04 05 06 07 08 09 0A 0B 0C
(3)    00 01 34 AF AA BB CC DD EE FF 11 22 33
```

La ligne 1 possède deux champs. Au chargement d'HEXABUG, le premier champ contient XXXX (adresse). Vous remplacez ces X par une adresse en hexadécimal, comme par exemple 4004. Le second champ indique le sens de défilement de la bande. Dans notre exemple, il est à 1. S'il est à 0, le sens est inversé.

La ligne 2 contient les deux derniers chiffres de l'adresse hexadécimale pour chaque champ d'affichage. Le poids fort de l'adresse est donné par l'adresse complète indiquée à la ligne 1.

La ligne 3 indique en hexadécimal le contenu de chaque case mémoire dont les adresses sont précisées par la ligne 2. Ainsi, dans notre exemple, nous voyons le contenu des adresse 4000 à 400C ; la valeur enregistrée à l'adresse 4004 est AA, celle à l'adresse 4009 est FF, etc. Lorsqu'on amène le curseur sur cette troisième ligne, il se positionne au milieu de la bande et reste fixe. Si l'on veut modifier le contenu de l'adresse 4008, il faut donc faire défiler la bande de quatre positions à gauche.

Lorsque l'écran d'HEXABUG apparait pour la première fois, les lignes 2 et 3 sont utilisées par le nom du programme, de l'auteur, et par le copyright.. Les informations disparaissent dès que vous donnez une adresse à la zone, en remplacement des XXXX.

ATARI INSIDE

- Affichage en hexadécimal ou en ASCII

L'adresse principale et les derniers chiffres des adresses sur la ligne 2 sont toujours représentés en hexadécimal. Vous pouvez voir par contre les données en hexadécimal ou en ASCII. Appuyez sur CTRL C pour passer d'un affichage hexa à un affichage ASCII et vice versa (pour la zone courante).

- Contenu de la mémoire

Positionnez le curseur sur le champ adresse de la zone. Tapez l'adresse voulue et déplacez le curseur hors de cette zone. A ce moment, la zone d'affichage est remise à jour et vous voyez apparaitre le contenu de cette case et de ses voisines.

- Changement d'adresse

En faisant défiler la zone (CTRL→ et CTRL←), le curseur étant dans la partie données de la zone, vous pouvez examiner les autres cases mémoire voisines. Si vous maintenez la pression sur ces touches pendant quelques secondes, la vitesse de défilement double. L'adresse en ligne 1 est remise à jour à chaque déplacement de curseur.

Chaque zone mémoire constitue une fenetre dans l'ensemble de l'espace mémoire adressable de l'ordinateur ATARI.

- Modification du contenu de la mémoire

Lorsque le curseur est au centre de la ligne de données, il vous suffit de taper une nouvelle valeur hexa pour modifier immédiatement le contenu de cette case. Puis la bande se déplace automatiquement afin que la case suivante apparaisse sous le curseur. Cela simplifie beaucoup l'entrée de nombreuses valeurs hexa. Le déplacement se fait selon le sens programmé (valeur 1 ou 0 placée à droite de l'adresse, en ligne 1 de la zone). HEXABUG vérifie que vous ne tentez pas de modifier une valeur en mémoire morte. Si oui, le message "SORRY, ADRESS IS IN ROM" s'affiche et HEXABUG attend un nouvel ordre.

ATARINSIDE

- Inverseur de sens

Nous écrivons normalement de gauche à droite. C'est-à-dire que nous modifions généralement le contenu de la mémoire commençant par les adresse basses et en remontant vers le haut de la mémoire. En conséquence, la zone se déplace de droite à gauche, le curseur étant fixe. Cela n'est toutefois pas parfait car dans certains cas, il est nécssaire d'inverser cet ordre. Ainsi la pile du système utilise la mémoire de haut en bas ; les mots sur 16 bits sont stockés poids faibles en premier. Afin de vous permettre de changer ce sens de défilement de la zone, HEXABUG a un inverseur par zone, placé à droite du champ d'adresse (dans la première ligne de la zone). Si vous tapez "0" dans ce champ, le curseur se déplace sur la gauche dès que vous avez modifié la donnée. Si vous tapez 1 dans le champ direction, votre curseur se déplacera vers la droite. Ce sens joue aussi sur la touche TAB.

Le meilleur moyen pour comprendre le fonctionnement de cet indicateur est de l'utiliser sur une zone libre de la mémoire, déplacez le curseur, changez le sens, modifiez à nouveau certains octets. Vous constaterez que HEXABUG est un outil très puissant pour modifier ou construire des tables, modifier des codes, etc.

En raison de la facilité avec laquelle vous pouvez modifier la mémoire, faites attention à ne pas faire d'erreur.

ZONE D'AFFICHAGE DE LA PILE

La zone d'affichage de la pile est une zone mémoire particulière servant à afficher et à modifier le contenu de la pile de votre programme. Quand votre programme s'arrête sur un point d'arrêt, HEXABUG recopie les données de la pile dans une page de la mémoire HEXABUG. La zone d'affichage représente cette page mémoire. Lorsque vous relancez le programme, HEXABUG transfère les données de sa page mémoire dans la zone réservée à la pile par le 6502.

La taille de la pile telle qu'elle est définie par le microprocesseur 6502 représente une limitation importante du système. En effet, les autres microprocesseurs ont des piles qui peuvent utiliser toute la mémoire de l'ordinateur. Le registre S du 6502 ne contient que 8 bits, si bien qu'il ne peut adresser que 256 octets, soit une page mémoire. Si votre programme excède cette capacité, il ré-écrira pardessus les premiers octets de la pile. Après avoir sauvegardé la pile, HEXABUG repositionne le registre S à son point de départ ($FF) pour sa propre utilisation. Ainsi, même si votre programme utilise toute la pile, HEXABUG ne provoquera pas de dépassement de capacité.

ATARINSIDE

La sauvegarde de la pile vous permet aussi d'ajouter des éléments à la pile sans modifier la pile d'HEXABUG. Ceci est très utile pour tester des sous-programmes appelés depuis le BASIC (avec la fonction USR). Quand une donnée est placée dans la pile, le pointeur de pile est décrémenté. Cela implique que les données sont rangées depuis les adresses hautes vers les adresses basses. La zone d'affichage de la pile a son indicateur de sens initialisé à zéro, si bien que vous pouvez placer des données dans la pile en respectant le sens normal. Notez qu'il vous suffit d'ajouter des octets à partir de la position pointée par le registre S. La zone d'affichage est automatiquement initialisée à cette adresse. Si vous entrez des données, la dernière que vous aurez tapée sera la première utilisée par votre programme. Vérifiez bien que vous entré vos valeurs dans le champ S.

La zone d'affichage de la pile fonctionne comme la pile du 6502, c'est-à-dire qu'arrivée en 00, elle reprend à son point de départ ($FF). En d'autres termes, la page mémoire affichée à l'écran est toujours la même et la ligne d'adresse représente toujours la zone -$100-$1FF. Cela est imposé par le fonctionnement du 6502 et voila pourquoi il n'existe pas de champ adresse dans la première ligne de cette zone.

ZONE D'AFFICHAGE DES INTRUCTIONS

Cette zone est également particulière. Lorsqu'un point d'arrêt est rencontré, HEXABUG initialise automatiquement l'adresse de départ de cette zone par l'adresse du point d'arrêt. En tapant vous-même une adresse dans le champ de cette zone, vous pouvez créer un huitième point d'arrêt.

- Désassemblage

La zone d'affichage des instructions peut donner une représentation des données en hexa ou en ASCII, comme les autres zones, mais elle peut aussi désassembler les codes des instructions. En appuyant sur CTRL W, vous ouvrez une "fenêtre d'instructions" affichant onze lignes d'instructions désassemblées à la fois. Cette fenêtre ne sert qu'à l'affichage ; vous ne pouvez pas positionner le curseur directement sur une instruction et la modifier.

Chaque ligne possède quatre champs, en voici un exemple :

```
FF9F   20600E   JSR   $0E60
```

Ces champs sont :

- l'adresse de l'instruction
- l'instruction placée à partir de cette adresse pouvant utiliser plusieurs octets et représentée en hexadécimal.
- le mnémonique du langage assembleur correspondant à cette instruction.
- l'opérande de l'instruction.

Si l'instruction est un branchement, un champ additionnel représente l'adresse à laquelle sautera le programme si la condition est vraie. Par exemple:

```
FFC9  D020  BNE  $20    $FFEB
```

ATARINSIDE

Si la donnée n'est pas une instruction valide, ??? s'affichera dans le champ mnémonique.

Défilement des adresses :

Pour ouvrir cette fenêtre, votre curseur doit se situer au-dessus de la zone code ou sur la première ligne de cette zone mais pas en-dessous (toutefois, il peut être sur la ligne de commande). Appuyez sur CTRL W pour ouvrir ou fermer la fenêtre de désassemblage. Lorsque la fenêtre est ouverte, vous pouvez examiner les instructions suivantes en appuyant sur CTRL N (pour "NEXT").

Après être allé de l'avant, vous pouvez revenir en arrière en utilisant CTRL P (pour "PREVIOUS"). Mais comme d'une manière générale il est impossible de désassembler en reculant dans la mémoire, CTRL P s'arrêterea de fonctionner lorsque vous aurez à nouveau atteint l'adresse de départ.

L'adresse de départ de la zone code est utilisée comme argument implicite par la commande SKP décrite plus loin.

ZONE D'AFFICHAGE DES REGISTRES

La zone d'affichage des registres représente les valeurs contenues dans les registres du 6502 lorsque votre programme s'arrête sur un point d'arrêt. Chacun des drapeaux du registre P (registre d'état) est affiché individuellement pour vous rendre leur lecture plus facile. Pour changer le contenu d'un registre, amenez le curseur dans le champ correspondant et tapez directement la nouvelle valeur. Notez que n'importe quelle valeur différente de zéro tapée pour un des drapeaux du registre P, positionne ce drapeau à 1. Pour relancer l'execution de votre programme à partir d'une nouvelle adresse, changez le compteur ordinal PC. Si vous relancez votre programme depuis le début, vous devrez certainement repositionner le pointeur de pile S à $FF.

Si vous modifiez l'adresse du compteur ordinal PC, HEXABUG exécute deux opérations après que vous ayez déplacé votre curseur en dehors de la zone d'affichage des registres :

1) Il vérifie que la donnée placée à la nouvelle adresse constitue bien une instruction valide pour le 6502. Si ce n'est pas le cas, le message DATA AT ADRESS NOT INSTRUCTIONS s'affiche et votre curseur se repositionne sur le champ du compteur ordinal, si bien que vous pouvez changer sa valeur immédiatement. HEXABUG ne peut par contre évidemment pas savoir si un code valide constitue réellement une instruction ou s'il s'agit simplement d'une donnée.

2) La zone d'affichage des instructions est initialisée à cette nouvelle adresse.

Si vous modifiez le contenu du registre, HEXABUG modifiera en conséquence l'affichage dans la zone de la pile, dès que vous aurez déplacé le curseur en dehors de la zone d'affichage des registres.

ATARINSIDE

Notez qu'après avoir modifié des registres vous pouvez directement appuyer sur START pour exécuter une commande sans avoir à déplacer votre curseur en dehors de la zone d'affichage des registres. Par exemple, vous pouvez modifier le compteur ordinal PC et appuyer sur START pour exécuter une commande CONT sans avoir à déplacer le curseur.

ZONE D'AFFICHAGE DES POINTS D'ARRET

Cette zone comporte sept points, un pour chaque point d'arrêt. Le fait d'entrer une adresse dans un de ces champs produit un arrêt de votre programme et un retour à HEXABUG lorsque l'exécution de votre programme passe par cette adresse. Lorsque vous tapez une adresse dans un champ point d'arrêt, quatre évènements se produisent :

1) Si l'adresse est 0000, alors le point d'arrêt est supprimé. Notez que cela signifie que vous ne pouvez placer de point d'arrêt à l'adresse 0000.

2) L'adresse est vérifiée afin d'être sûr qu'elle contient bien une instruction valide du 6502. Si ce n'est pas le cas, le message d'erreur DATA AT ADRESS NOT INSTRUCTIONS s'affiche.

3) Une vérification est effectuée afin d'être sûr que l'adresse se situe en mémoire vive (RAM). Dans la négative, le message SORRY,THE ADRESS IS IN ROM s'affiche. En effet, HEXABUG remplace temporairement l'instruction du programme par l'instruction BRK ; en conséquence, tous les points d'arrêt doivent être en RAM.

4) Si tous les tests précédants sont passés avec succès, l'instruction du programme est sauvegardée dans une mémoire tampon.

Si dans les cas 2 et 3 le test n'est pas vérifié, alors un message d'erreur s'affiche, un bip se fait entendre et le curseur positionné au début du champ adresse afin que vous puissiez corriger la valeur.

Si vous désirez modifier une instruction, vérifiez bien qu'il n'existe pas de points d'arrêt à cette adresse.
Sinon annulez-le, modifiez l'instruction et replacez le point d'arrêt. Ainsi vous etes sur que HEXABUG a bien remis en place la bonne instruction à la bonne adresse.

LIGNE DE MESSAGE

HEXABUG utilise cette ligne pour afficher des messages. La plupart des messages d'erreur s'affichent pendant quelques secondes. D'autres messages restent à l'écran tant qu'un autre message ne vient pas les remplacer. Vous pouvez toujours effacer cette ligne en appuyant simultanément sur les touches SHIFT et CLEAR.

ATARINSIDE

LIGNE DE COMMANDE

- Controle du curseur :

Appuyez sur la touche OPTION pour amener votre curseur sur la ligne de commande. Appuyez de nouveau sur OPTION pour revenir aux zones d'affichage.

Dans la ligne de commande, HEXABUG insère un nouveau caractère à l'emplacement du curseur lorsque vous tapez au clavier. Tous les caractères situés à droite du curseur sont déplacés vers la droite afin de faire de la place pour les caractères que vous tapez. Les touches CTRL → et CTRL ← déplacent le curseur sur la ligne: Les touches CTRL ↑ et CTRL ↓ amènent le curseur au début de la ligne de commande. DELETE BACKS efface le caractère situé à gauche du curseur. CTRL DELETE BACKS efface le caractère sous le curseur et SHIFT DELETE BACKS efface toute la ligne de commande.

- Syntaxe générale :

Toutes les commandes ont une syntaxe commune. Les commandes et leurs arguments doivent être tapés en lettres capitales. En fait, vous pouvez utiliser les minuscules dans un seul cas : lorsque vous modifiez le code ASCII du contenu de la mémoire.

Au démarrage d'HEXABUG, la variable SHFLOK (à l'adresse $02BE), qui contrôle le mode majuscule/minuscule, est sauvegardée et HEXABUG force le mode majuscule. La valeur originale de SHFLOK est remise en place lorsqu'on quitte HEXABUG. Toutes les commandes doivent commencer dans la première colonne à gauche dans la ligne de commande et doivent être suivies d'un espace.

- Paramètres des commandes

Si une commande nécessite un argument, celui-ci doit commencer après l'espace finissant la commande. HEXABUG ignore le reste de la ligne situé à droite de l'argument. En conséquence, vous pouvez utiliser la place restante pour des remarques, ou d'autres commandes en préparation. Voici un exemple de la commande DOS utilisée avec l'argument A :

DOS A <AUTRES CARACTERES IGNORES>

Appuyez sur la touche START pour faire exécuter la commande. La position du curseur dans la ligne n'a pas d'importance.

Voici une description des commandes connues par HEXABUG :

-DOS : passage au Système d'Exploitation de la Disquette.

La commande DOS produit le même effet que lorsque vous tapez DOS en BASIC. Vous pouvez ainsi charger en mémoire, par l'option L, un programme objet que vous voulez tester avec HEXABUG. Pour revenir à HEXABUG depuis le DOS, utilisez l'option M (RUN AT ADDRESS l'adresse de départ étant $BEF0).

ATARINSIDE

La commande DOS accepte un argument optionnel. Si vous tapez A après la commande (et après l'espace nécessaire), HEXABUG libère toute la mémoire en repositionnant RAMTOP et RAMSIZ avant de passer au DOS.
Remarque : la commande DOS utilise le vecteur DOSVEC ($000A) du système d'exploitation. Si votre programme modifie ce vecteur, la dommande DOS ne fonctionnera pas correctement.

-CONT : continue l'exécution de votre programme.

C'est la commande principale pour revenir à votre programme depuis HEXABUG. Cela produit les effets suivants :

1- Cette commande recharge les registres du microprocesseur avec les valeurs affichées.

2- La pile est remise à jour en fonction de la zone d'affichage correspondante.

3- Les cases mémoire modifiées par HEXABUG retrouvent leurs valeurs initiales. L'écran redevient celui de votre programme.

4- Votre programme reprend à l'adresse précédemment indiquée par le compteur ordinal PC.

5- Tous les points d'arrêt sont mis en place.

Les quatre premières étapes ci-dessus sont toujours réalisées. Mais l'instruction CONT accepte un argument qui peut être Z ou A. CONT Z n'effectue pas la cinquième étape. Cela revient à ignorer tous les points d'arrêt en une seule commande.

L'option A, comme l'option Z, n'utilise aucun point d'arrêt et en outre, elle libère la zone mémoire réservée à HEXABUG, comme le fait l'argument A dans la zone DOS.

-SS : fonctionnement pas à pas de votre programme.

La commande SS réalise les trois premières étapes de la commande CONT puis exécute l'instruction placée à l'adresse du compteur ordinal. SS place un point d'arrêt temporaire à l'instruction suivante, ce qui signifie que vous ne pouvez pas utiliser le mode pas à pas à travers la mémoire morte (ROM). La commande SKP décrite ci-dessous vous permet d'exécuter tous les sous-programmes placés en mémoire morte et d'arrêter votre programme dès son retour en mémoire vive. Vous vous concentrez ainsi sur le déroulement de votre programme sans tenir compte des sous-programmes du système d'exploitation ATARI (sous-programmes qui fonctionnent déjà correctement et qu'il n'est pas nécessaire de contrôler).

ATARINSIDE

-SKP : Place un point d'arrêt temporaire à l'adresse de début de la zone d'affichage du code.

La commande SKP est similaire à la commande CONT mais en outre elle condidère l'adresse de début de la zone d'affichage "code" comme un point d'arrêt.

Cette commande est très utile dans de nombreux cas. Par exemple, en positionnant cette adresse sur la première instruction placée après une boucle, vous évitez de contrôler à nouveau cette boucle qui s'exécute peut être des douzaines de fois. En somme vous faites exécuter des sous-programmes qui fonctionnent correctement par le microprocesseur à la pleine vitesse. Vous arrivez ainsi rapidement à la zone que vous voulez étudier. Cette instruction est également très utile pour passer par dessus de nombreux appels à des sous-programmes.

- Commandes de défilements des zones d'affichage

Les commandes suivantes concernent la zone d'affichage dans laquelle se situe le curseur. Si vous exécutez une de ces commandes alors que la zone concernée n'est pas une zone mémoire, le message d'erreur SCROLLING STRIP ONLY s'affiche. Si la zone concernée n'a pas d'adresse de départ (elle n'a pas encore été utilisée) le message STRIP NOT IN USE s'affiche.

Chacune des commandes suivantes modifie la zone mémoire dont le contenu apparait à l'écran. Chacune existe dans une version rapide (elle est suivie de la lettre F). Il n'est pas possible d'utiliser les versions rapides dans la zone d'affichage de la pile, mais vous avez rarement besoin de faire défiler cette bande de plus de quelques octets à la fois. Pour arrêter le défilement horizontal de la fenêtre (version lente de la commande), appuyez sur la touche BREAK.

-SE et SEF : recherche d'une suite de données.

La commande SE recherche dans la mémoire la suite d'octets que vous donnez comme argument, en commençant à l'adresse de la zone d'affichage considérée. SE peut aussi bien rechercher dans le sens ascendant que descendant. La syntaxe générale est :

SE [DELEMITEUR][HEXA][ESPACE][HEXA][ESPACE]...[DELIMITEUR] OPTIONNEL<

ATARINSIDE

Chaque zone entre crochets représente un caractère ou un nombre. Le premier caractère dans l'argument est un délimiteur. Vous pouvez utiliser n'importe quel caractère comme délimiteur tant que vous le conservez jusqu'à la fin de l'argument. Le premier octet de l'ensemble de données que vous recherchez est ensuite tapé. Les données inférieures à $10 doivent commencer par un zéro. Un espace est nécessaire entre chaque octet. La recherche peut s'effectuer sur une chaine de 9 octets (longueur maximale de la ligne de commande). La recherche dans la mémoire s'effectue par défaut des adresses basses vers les adresses hautes. Vous pouvez rechercher dans l'autre direction en ajoutant le symbole < en fin de ligne, après le second délimiteur.

SE /01 23 45 67 89 AB/<

Dans cet exemple, le programme recherche la chaine "01 23 45 67 89 AB" des adresses hautes vers les adresses basses. La zone d'affichage défile en synchronisme avec la recherche.

La commande SEF recherche beaucoup plus vite dans la mémoire (quelques secondes suffisent pour 48 KO). La syntaxe est la même. Toutefois la zone d'affichage n'est pas modifiée et il n'est pas possible de relancer la commande SEF après avoir trouvé la première occurence de la chaine (alors que la commande SE le permet). Si la chaine n'est pas trouvée, le message SEARCH FAILED apparait.

-CR et CRF : déplacement par pas vers des adresses mémoire supérieures.

La commande CR (CURSOR RIGHT) fait défiler la zone d'affichage contenant le curseur par bonds. Le nombre hexadécimal à deux chiffres placé après cette commande indique le nombre d'adresses sautées à chaque fois. La syntaxe est : CR [HEXA]

Lorsque la valeur Hexa est inférieure à 10, le zéro d'entête n'est pas nécessaire.

Cette commande est utile pour examiner le contenu de tables. Utilisez la longueur de chaque entrée dans la table comme un argument pour l'instruction CR. Vous pouvez ainsi examiner chaque entrée successivement. La version rapide de cette commande est utile pour accéder à une adresse en simulant l'adressage indexé. Placez l'adresse de base (début de la table) dans la zone d'affichage concernée et utilisez CRF avec la valeur d'offset pour afficher immédiatement l'adresse désirée.

ATARINSIDE

En raison des modes d'adressage du 6502, l'examen de tables revient souvent. HEXABUG a donc été conçu pour faciliter l'examen de listes de données.

-CL et CLF : déplacemnt par pas vers des adresses mémoire inférieures.

La commande CL (CURSOR LEFT) est la symétrie de CR, le déplacement s'effectuant des adresses hautes vers les adresses basses.

-SMOOTH : mise en/hors service du défilement fin.

Utilisez cette commande pour supprimer ou réactiver la possibilité de défilement fin dans les zones d'affichage mémoire. HEXABUG valide cette fonction à sa mise en service. Pour la supprimer, utilisez cette commande sans argument. Pour réactiver cette fonction, tapez SMOOTH 1. Cette fonction est indépendante et sans effet sur le contenu même de la zone d'affichage.

Le registre matériel utilisé pour contrôler le défilement fin horizontal est un registre à écriture seulement. En d'autres termes, bien que vous puissiez placer des valeurs dans ce registre, il n'est pas possible de les relire. En conséquence, HEXABUG ne peut donc pas sauvegarder et recharger les valeurs de votre programme si celui-ci utilise les possibilités de défilement fin.

Pour éclaircir cela, examinons l'adresse $D404 qui est celle du registre HSCROL en utilisant HEXABUG. La dernière valeur placée dans ce registre lorsque la zone d'affichage est immobile est $00, mais vous voyez $FF à l'affichage. Maintenant essayez de modifier le contenu de cette adresse en écrivant $0F. Le message d'erreur SORRY, ADDRESS IS IN ROM s'affiche et la zone mémoire se déplace de 4 caractères vers la droite. Le message d'erreur apparait car HEXABUG découvre qu'il n'obtient pas à la relecture de ce registre la valeur qu'il y a placée. Il croit donc qu'il s'agit d'une adresse en mémoire morte mais en réalité, ce registre a bien reçu la valeur $0F puisque l'affichage a changé.

L'intérêt de ne pas utiliser la fonction de défilement fin d'HEXABUG réside donc dans le fait qu'il n'y aura pas d'interférences avec votre programme s'il utilise le registre HSCROL. Vous pouvez toujours utiliser les zones d'affichage mais vous constaterez que leur lecture devient beaucoup moins agréable lorsque la bande défile.

ATARINSIDE

MESSAGES D'ERREUR

CHANGE PC TO CONT OR USE DOS CMD

Ce message vous rappelle que l'adresse du compteur ordinal vaut zéro lorsque HEXABUG vient de se charger. Aussi, vous devez y placer l'adresse de départ de votre programme à moins que vous désiriez utiliser la commande DOS pour revenir au menu du système d'exploitation de la disquette.

CIO ERROR NUMBER XXX

L'erreur affichée est apparue après un retour CIO (reportez-vous au manuel DE RE pour description du CIO). Notez qu'HEXABUG utilise le CIO pour saisir les caractères en provenance du clavier, en ouvrant le périphérique K:

L'IOCB numéro 7 peut avoir déjà été utilisé. Reportez-vous à votre manuel DOS pour une liste complète des codes d'erreurs. Une erreur CIO est généralement fatale, c'est-à-dire que votre programme ne peut redémarrer. Vérifiez bien que votre programme n'utilise pas l'IOCB numéro 7.

DATA AT ADDRESS NOT INSTRUCTION

Vous essayez de placer un point d'arrêt à une adresse qui ne contient pas une instruction valide du 6502. Ce message apparait également si vous placez une mauvaise adresse dans le compteur ordinal.

DELIMITER ERROR

La syntaxe utilisée pour une commande SE ou SEF est mauvaise. Vous devez utiliser le même délimiteur pour le début et la fin de la chaine.

NO FAST COMMANDS IN STACK

Vous tentez d'utiliser la version rapide d'une commande de défilement alors que vous consultez la zone d'affichage de la pile. Utilisez la commande normale à la place (par exemple SE au lieu de SEF).

NO SUCH COMMAND

Vous avez tapé une commande qui n'existe pas dans HEXABUG. N'oubliez pas que toute commande doit être suivie par un espace.

ATARINSIDE

PLEASE ENTER HEX NUMBER : 0-F

Vous avez tapé un caractère qui n'est pas dans la gamme 0 à 9 et A à F. Le curseur ne se déplace pas, tapez un chiffre hexadécimal valide.

SCROLLING STRIP ONLY

Vous utilisez une commande de défilement dans une zone d'affichage qui ne l'autorise pas.

SEARCH FAILED

La commande SEF retourne ce message lorsque la chaine recherchée n'a pas été trouvée en mémoire.

SORRY, ADDRESS IS IN ROM

Vous tentez de changer la valeur d'une adresse mémoire qui est en mémoire morte (ROM). Cette opération est impossible. Ce message apparait également si vous tentez de placer un point d'arrêt en ROM. Si vous utilisez la commande SS, ce message apparait si l'instruction suivante est un appel à un sous-programme placé en mémoire morte car SS remplace l'instruction suivante par BRK. Utilisez la commande SKP pour arrêter l'exécution au retour du sous-programme placé en ROM.

STRIP NOT IN USE

Vous tentez d'utiliser une commande de défilement dans une zone d'affichage qui n'est pas encore utilisée. Donnez d'abord une adresse de départ à cette zone.

ATARINSIDE

PRINCIPES DE FONCTIONNEMENT

HEXABUG se charge automatiquement dans la mémoire grâce à un fichier AUTORUN.SYS. Il utilise les 12 K supérieurs de la mémoire ($9000-$C000) en déplaçant les pointeurs RAMSIZ et RAMTOP afin que le programme utilisateur ne puisse utiliser cette région. Tout programme utilisant au maximum 32 K de mémoire peut donc être mis au point par HEXABUG sur un ordinateur ATARI possédant au moins 48 K. Si votre programme déplace lui aussi RAMSIZ et RAMTOP, vous pouvez descendre ces pointeurs en partant de l'adresse inférieure d'HEXABUG puis après avoir manipulé ces pointeurs "à la main", vous chargez votre programme depuis le DOS. Bien sur, il ne devra pas y avoir de recouvrement entre HEXABUG et la zone réservée utilisée par votre programme. La zone mémoire située entre $B8D1 et $BEEF est libre et réservée pour de futures améliorations. De nombreux programmes supposent que RAMTOP et RAMSIZ sont sur un multiple de 4k, HEXABUG se conforme à cette habitude.

Vous pouvez utiliser HEXABUG en même temps que le macro-assembleur ATARI ou que l'Editeur de Programmes et de Textes. Cela vous fait gagner du temps dans les cycles de mise au point mais la mémoire libre pour le programme d'application diminue.

Si vous testez un programme qui utilise l'unité de disquette, n'oubliez pas de placer dans l'unité une disquette sans valeur, dans le cas où votre programme l'endommagerait.

Lorsque HEXABUG est en mémoire, vous pouvez le relancer de deux manières : soit vous utilisez l'option M du DOS et l'adresse $BEF0; soit vous exécutez un petit programme qui possède une instruction BREAK. Mais ces deux procédures ne sont pas identiques. Voici pourquoi :

Lorsque votre programme rencontre un point d'arrêt (instruction BRK), HEXABUG exécute un certain nombre d'opérations avant de vous donner le contrôle du curseur :

1- il sauvegarde tous les registres internes du 6502 ;
2- il sauvegarde la pile (page 1) ;
3- il repositionne les instructions d'origine à la place des points d'arrêt ;
4- il sauvegarde toutes les variables du système d'exploitation qu'il utilise ;
5- il remplace le sous-programme de gestion des interruptions de synchronisation verticale par le sien (Blanking Vertical) ;
6- l'affichage est remis à jour.

En relançant HEXABUG par l'adresse $BEF0 :

1- tous les registres sont initialisés à 0, sauf le registre S qui est initialisé à $FF ;
2- HEXABUG sauvegarde la pile ;
3- il sauvegarde toutes les variables qu'il utilise ;
4- il met en place sa propre gestion des interruptions verticales (Blanking vertical) ;
5- la commande DOS est placée dans le champ commande;
6- un message vous rappelle que vous devez initialiser le compteur ordinal PC;
7- l'écran est mis à jour.

ATARINSIDE

Notez que l'étape 3 dans la première procédure n'est pas exécutée dans la seconde. Cela est important si vous désirez mettre au point deux programmes différents qui se chargent l'un après l'autre dans la même zone mémoire. Si vous ne faites pas attention, HEXABUG risque avec la première procédure, de placer les instructions correspondant aux points d'arrêt du premier programme dans le second. La procédure la plus saine dans ce cas consisterait à effacer tous les points d'arrêt avant de lancer le second programme.

Toute la mémoire vive est initialisée à $00 à la mise en route de l'ordinateur, ainsi le fait de lancer l'exécution à n'importe quelle adresse non utilisée provoquera l'initialisation d'HEXABUG.

Quand votre programme rencontre un point d'arrêt, HEXABUG sauvegarde toutes les variables que votre programme utilise de manière à pouvoir les restaurer à la reprise du programme. Voici une liste des variables sauvegardées par HEXABUG :

	Adresse de sauvegarde	**Adresse réelle**	
ATACHR	$95F0	$02FB	Caractère sauvegardé par le sous-programme de gestion du clavier.
BRKKEY	$95F2	$0011	Drapeau de la touche BREAK
CH	$95F6	$02FC	Octet utilisé par le sous-programme d'interruption du clavier
INVFLG	$95F1	$02B6	Drapeau d'inversion vidéo
IOCBAS	$95E3	$0020	Base en page 0 de l'IOCB douze octets sont sauvegardés
SHFLOK	$95F3	$02BE	Fonction Majuscule/minuscule
VBREAK	$95DF	$0206	Vecteur BREAK
VDSLST	$95DC	$0200	Vecteur d'interruption de Display List
VVBLKI	$95E1	$0222	Vecteur d'interruption de Blanking vertical

Pour charger le contenu d'une de ces adresses, modifiez dans l'adresse de sauvegarde, et non dans l'adresse réelle.

Notez que le sous-programme de gestion des interruptions de Blanking vertical est remplacé par un sous-programme d'HEXABUG. Cela veut dire que les compteurs de temps logiciel ne seront pas incrémentés avec HEXABUG en fonctionnement. Par contre, les compteurs matériels continuent de fonctionner normalement.

ATARINSIDE

HEXABUG positionne les players-missiles en dehors de l'écran. Généralement, les programmes n'utilisent pas directement les registres matériels mais plutôt les registres cache placés en mémoire vive. Le sous-programme du système d'exploitation utilisé pendant les périodes de Blanking vertical transfère normalement le contenu des mémoires cache dans les registres matériels. Comme HEXABUG utilise sa propre gestion, les players restent en dehors de l'écran pendant que vous utilisez HEXABUG. Lorsque vous revenez à votre programme, votre vecteur VVBLKI est restauré et les players missiles fonctionnent à nouveau normalement.

Un autre registre, en lecture seulement, est utilisé par HEXABUG : il s'agit de NMIEN. Ce registre autorise ou non les interruptions non masquables. Lorsque vous revenez à votre programme, les interruptions de Display List seront autorisées car HEXABUG les utilise.

Chaque fois que vous entrez dans HEXABUG, l'IOCB 7 est ouvert pour K:. Quand vous quittez HEXABUG pour revenir à votre programme, l'IOCB est fermé. Toutefois votre programme ne devrait pas utiliser cet IOCB.

HEXABUG utilise les adresses $80 à $9A en page zéro. En conséquence, votre programme ne peut utiliser ces adresses. La première moitié de la page zéro ($00 à $80) est utilisée par le système d'exploitation. Les adresses $9A à $FF sont libres.

ATARINSIDE

RESUME DES COMMANDES

TOUCHES DE FONCTION

TOUCHE	FONCTION
OPTION	Passe le curseur de la ligne de commande aux zones d'affichage et vice-versa
SELECT	Commute entre l'écran de votre programme et l'écran d'HEXABUG
START	Exécute la commande placée à gauche dans la ligne de commande
CTRL ↑	Déplace le curseur d'une ligne vers le haut
CTRL ↓	Déplace le curseur d'une ligne vers le bas
CTRL ←	Déplace le curseur d'un caractère vers la gauche
CTRL →	Déplace le curseur d'un caractère vers la droite
TAB	Déplace le curseur jusqu'au champ suivant
CTRL C	Affiche les données en ASCII
CTRL W	Permet le désassemblage dans la zone code
CTRL N	Fait défiler la fenêtre de désassemblage vers les instructions suivantes
CTRL P	Fait revenir en arrière la fenêtre de désassemblage

ATARI INSIDE

NOM : CONT (continue le programme)
SYNTAXE : CONT ou CONT Z ou CONT A
UTILISATION : Permet la reprise du programme sous test à partir de l'adresse indiquée par le compteur ordinal.
Initialise tous les points d'arrêt si cette commande est utilisée sans argument

NOM : DOS
SYNTAXE : DOS ou DOS A
UTILISATION : Passe au menu du DOS par l'intermédiaire du vecteur DOSVEC.
L'argument A annule HEXABUG.

NOM : SS
SYNTAXE : SS
UTILISATION : Exécute une seule instruction à partir de l'adresse placée dans le compteur ordinal, et retour à HEXABUG.

NOM : SKP
SYNTAXE : SKP
UTILISATION : Similaire à CONT mais place un point d'arrêt temporaire à l'adresse de début de la zone d'affichage du code. Si vous y placez l'adresse de la première instruction située après l'appel d'un sous-programme en ROM (mémoire morte), vous ferez exécuter à pleine vitesse le sous-programme et reprendrez le mode pas à pas ensuite (instruction SS).

NOM : SE et SF
SYNTAXE : SE/XX'XX...XX/<
UTILISATION : Recherche la suite d'octets placée entre les deux délimiteurs dans la mémoire. La touche BREAK annule la commande. SE fait défiler la zone d'affichage tandis que SEF permet une recherche rapide.

NOM : CR et CRF
SYNTAXE : CR XX (où XX est une valeur Hexadécimale
UTILISATION : Fait défiler la zone d'affichage vers les adresses supérieures.

NOM : CL ET CLF
SYNTAXE : CL XX (où XX est une valeur Hexadécimale)
UTILISATION : Fait défiler la zone d'affichage vers les adresses inférieures.

NOM : SMOOTH
SYNTAXE : SMOOTH (suppression) ou SMOOTH 1 (activation)
UTILISATION : Supprime ou active le défilement fin des zones d'affichage.

Adresse de relance d'HEXABUG : $BEF0

ATARINSIDE

SOMMAIRE D.D.T.

ATARINSIDE

INTRODUCTION

l'ART DE CREER UN PROGRAMME DE DEBOGAGE

En terme simple, un programme est une séquence d'instructions qui demande à l'ordinateur de faire quelque chose. Programmer revient à préparer les instructions pour que l'ordinateur les exécutent. Celà semble assez simple mais la programmation conserve une atmosphère mystérieuse, ainsi le plus vétérant des programmeurs aborde une discussion sur la programmation avec déférence et hésitation. Essayez de trouver un programmeur qui sait quand son programme est terminé et vous comprendrez ce que je veux dire.La première loi de DUNIO est que les choses ne sont jamais aussi simples qu'elles devraient l'être. Celà signifie que les programmes prennent toujours plus de temps qu'ils ne devraient--pour qu'ils fonctionnent effectivement dans tous les cas.Qu'est ce qui rend difficile la programmation sur ordinateur? La réponse est principalement dans le fait que, en tant qu'humain, nous ne sommes pas habitués a penser aussi précisément que ce qu'il y a faire dans un programme d'ordinateur. Même l'homme le plus rationnel a une pensée influencée par l'émotion, l'intuition et la perspicacité. Le résultat final s'en retrouve toujours plus grand que la suite entièrement logique et rationnelle. L'ordinateur ne se programme pas encore, malheureusement, suivant nos critères. Il doit etre programmé de manière très précise. Avec des machines qui exécutent un demi million d'instructions par seconde, le temps que vous vous aperceviez que quelque chose ne va pas et l'ordinateur s'est déjà bloqué. Le problème est d'aller du concept au concret, de partir d'une idée et de la changer en programme.

Qui n'a jamais pensé à la meilleure souricière logicielle. De façon générale, il est plus facile d'avoir des idées que de les programmer, mais il ne doit pas en être ainsi. Comme le dirait un artisan, il faut avoir de bons outils pour faire un bon ouvrier. La programmation en tant qu'entreprise humaine se situe entre l'art et la science et personne ne sait ou se trouve le juste milieu. La programmation est gênée par des outils inadéquats, mais la plus grande partie du problème vient de l'approche de l'acte de programmation en lui même. Je me considère moi même être autant artiste que technicien. Chaque nouveau programme est une oeuvre artistique. Non seulement le programme doit fonctionner correctement, mais il doit aussi sembler juste. L'ordinateur est un instrument d'imagination, un pinceau sans comparaison, un crayon rempli de milliers d'histoires pas encore écrites---l'ultime instrument attendant que l'utilisateur le fasse vivre. Je nomme cette attitude "l'Art de la programmation créative".

ATARI INSIDE

J'ai horreur de l'admettre, mais d'une façon ou d'une autre, des erreurs se glissent dans mes programmes, en particulier quand j'essaye quelque chose de nouveau.---par exemple travailler en temps réel avec du graphique couleur et des sons. Un système comme l'ATARI 800 en est un bon exemple. Pour atteindre la pleine potentialité de ce système, nous devons parfois travailler en langage assembleur. A ce niveau de familiarité avec l'ordinateur, chaque petite erreur est amplifiée d'une manière considérable et la retrouver est difficile. Ce peut être une erreur de syntaxe, une erreur sémantique, une erreur d'horloge, une erreur de matériel..., la liste est longue. Comme Piet Hein le disait dans un de ses Grooks: "les problèmes dignes d'être attaqués prouvent leur valeur par leurs résistance".

Chers amis, je suis fatigué de tout celà. Ce dont nous avons besoin, c'est de quelque chose qui puisse nous laisser faire un peu plus de débogage, de débogage créatif. Ce dont nous avons besoin c'est de quelque chose comme ---- DUNION'S DEBUGGING TOOL!

GUIDE UTILISATEUR DE DDT

DESSEINS PHILOSOPHIQUE DE DDT

L'ordinateur ATARI 800 a des caractéristiques qui le distinguent des autres ordinateurs. Malheureusement, essayer d'accéder à ces caractéristiques à partir du BASIC ou de PILOT est frustrant. Dans beaucoup de cas, la seule solution est d'écrire au moins une partie du programme en langage assembleur. Celà ne serait encore pas trop mauvais si nous avions un outil de développement décent pour le langage assembleur, mais nous n'en n'avions pas jusqu'à très récemment. Cette situation a changée voici peu de temps avec l'apparition du Macro Assembleur ATARI, un outil de programmation très puissant. Cependant, étant donné que les programmes assembleurs ont l'habitude d'être d'abord infestés d'erreurs, le Macro Asembleur accentue encore ce défaut. Que faisons nous actuellement pour déboguer les programmes assemblés. La solution idéale, bien sur, serait de pouvoir accéder à quelque chose comme un analyseur logique ou un autre type de système de développement hardware. Cependant la plupart d'entre nous n'y arrive pas. Il y a toujours la cartouche Assembleur-Editeur ATARI. Sans renier ce produit celà ne me semblait pas encore idéal. Que faire donc? La réponse semblait être de développer un instrument de débogage spécifiquement développé pour être utiliser avec le Macro Assembleur. Ainsi est né DDT.

DDT est un outil de débogage du langage source flexible et extensible. Celà signifie que vous pourriez assembler DDT avec votre code source comme une sorte de parasite.Vous pouvez connecter DDT avec tout ce qui fonctionne à l'intérieur de votre ordinateur ATARI 800. Ces connections ou "crochets" permettent au DDT de coexister avec votre programme à tester. Cette flexiblité est utile dans un certain nombre de cas. D'abord, celà vous permet de déterminer l'endroit où DDT sera placé en mémoire; ce qui peut varier, dépendant de ce qui est à déboguer. Ensuite celà vous permet d'utiliser l'assembleur pour établir plusieurs caractéristiques de DDT. Notez encore que DDT est assez flexible pour ne pas avoir à l'assembler avec votre programme à chaque fois. Les exemples inclus dans la disquette de DDT vous donnerons une idée des possibilités offertes (par ex., connecter DDT à un programme).

La plupart des erreurs émanent de suppositions (implicites ou explicites) qui s'avèrent inexactes. Si c'est le cas, un outil de débogage qui vous oblige à voir des locations mémoire, des registres, des points d'arrêt, etc... passe à côté d'un point crucial. La plupart du temps, vous n'avez d'abord aucune idée de la cause d'un problème. L'idée centrale dans DDT est de mettre autant de renseignements possibles sur l'écran et ensuite laisser votre système visuel (c'est à dire vos yeux et l'hémisphère droit de votre cerveau) fonctionner, bref de laisser l'ordinateur faire ce qu'il fait le mieux et de laisser les programmeurs faire ce qu'ils font le mieux.

ATARINSIDE

Une conséquence de cette approche est que DDT se concentre autour du contrôle de son écran. A ce contrôle s'ajoute la capacité de changer et de gérer facilement l'état interne de la machine de sorte que vous pouvez avoir une image bien plus nette de ce qui se passe vraiment à l'intérieur du système à chaque instant. La plupart du temps, corriger une erreur de programme est facile, le problème est de la trouver et c'est là que DDT intervient.

La prochaine partie décrit chaque caractéristique de DDT. Ce qui suit est un chapitre expliquant comment il faut commencer à utiliser DDT et qui décrit les exemples. En dernière partie, un appendice technique contient davantage de renseignements sur la façon dont les caractéristiques de DDT sont implémentées. Lisez rapidement le manuel en entier pour avoir une vue d'ensemble de DDT, ensuite revenez en arrière et lisez chaque page plus attentivement. Finalement avant de commencer l'expérience, prenez une disquette vierge, formattez là, et écrivez y les fichiers du DOS. Ensuite copiez y tous les modules en code objet ou en code source qui vous intéressent. Si vous voulez expérimenter un des modules en code objet, renommer le AUTORUN.SYS. Alors tout ce qui vous reste à faire est d'arréter la machine puis de la remettre en route pour charger et initialiser le code automatiquement. Rien de plus simple.

Bonne chasse!

L'AFFICHAGE A L'ECRAN DDT

L'écran DDT vous montre l'état interne de la machine. L'écran est divisé en plusieurs parties, chacune d'elles montre un aspect différent de ce qui se passe à l'intérieur de l'ordinateur à l'instant.

Ces différentes parties sont appelées:

CONTENU DES REGISTRES - le contenu des registres courants du 6502.

FENETRE DE VISUALISATION - le contenu d'une partie de mémoire.

AFFICHAGE DE LA PILE - le contenu des 15 enregistrements du haut de la pile.

MINI TABLE DE SYMBOLES - la table des noms et des valeurs des symboles les plus courants.

TABLE DES POINTS D'ARRET - la table des registres des points d'arrêt

FENETRE DES COMMANDES - une fenêtre montrant les commandes entrées au clavier.

Les sections suivantes montrent chaque partie de visualisation.

```
-------------------------------------------------------------
:   LOC VAL INSTRUCTION  : STK :  VAR.     : VALUE          :
:------------------------:-----:----------:----------------:
:  1E2F  20              :  81 : LOMEM    : 33E0           :
:  1E30  75              :  96 : MEMPTOP  : 34E4           :
:  1E31  20              :  23 : SYMB1    : B8             :
:  1E32  00  PLA         :  45 : LABEL1   : A9             :
:  1E33  AA  TAX         :  76 : LABEL2   : 00             :
:  1E34  68  PLA         :  97 :          :                :
:  1E35  40  RTI         :     :          :                :
:  1E38  4C  JMP  LABEL1 :     :          :                :
:  1E39  BE              :     :          :                :
:  1E3A  FF              :     :          :                :
:  1E3B  A9  LDA  #53    :     :          :                :
:  1E3C  53              :     :          :                :
:  1E3D  30  BMI  2  1E41:     :          :                :
:-------------------------------------------------------------:
: BKP1 :BKP2 :BKP3 :BKP4 :BKP5 :BKP6 :TRP1 :TRP2            :
:------:-----:-----:-----:-----:-----:-----:----------------:
: ABCD :0000 :0000 :0000 :0000 :0000 :ABC6 :0000            :
:-------------------------------------------------------------:
:  PC  : ACC:NV BDIZC :X :Y :SP :    COMMAND                  :
:------:----:---------:--:--:---:-------------------------:
: 1E32 : 1E :00110100 :00:12:F9 : S 302122                  :
-------------------------------------------------------------
```

CONTENUS DES REGISTRES

La partie du bas de l'écran montre le contenu des registres du microprocesseur 6502. Quand DDT est entré, les contenus de ces registres sont copiés dans des registres fantômes qui sont alors visualisés. Ces registres fantômes sont copiés dans les registre du 6502 avant que le contrôle revienne au programme qui est testé.

Le contenu de ces registres est donné en Hexadécimal:

PC = compteur de programmes (deux octets)
ACC = accumulateur
X = registre d'index X
Y = registre d'index Y
SP = pointeur de pile

Le registre d'état du processeur est sous forme binaire où:

N = indicateur Négatif
V = Indicateur de débordemant de capacité
B = Indicateur d 'instruction BRK
D = indicateur de mode décimal
I = Indicateur d'interruption non permise
Z = Indicateur de zero
C = indicateur de retenue

ATARI INSIDE

FENETRE DE VISUALISATION

La fenêtre de visualisation montre une partie de l'espace mémoire adressable. Cette fenêtre est du côté gauche en haut de l'écran et occupe plus d'un quart de l'écran. Cette fenêtre est positionnée par DDT ou peut être déplacée pas à pas soit par "E", soit par la flèche vers le haut ou la flèche vers le bas.

La fenêtre peut être vue comme ayant l'un des trois filtres possibles devant lui. Vous changez de filtre grace à la commande "W" (voyez la section Interprétation des commandes). Le premier filtre qui est placé initialement est un filtre opaque. Il a un résumé des instructions en cours. Avec ce filtre, de nombreuses commandes semblent ne rien faire.

Le second filtre est un filtre de désassemblage. Le symbole ">" pointe ce qui est appelé la position courante. Quand DDT est chargé, il correspondra à la valeur du PC. La position courante peut être modifiée par "E", la flèche vers le haut ou la flèche vers le bas.

Le troisième filtre est le filtre hexadécimal. La fenêtre montre la valeur hexadécimale et la représentation ATASCII de jusqu'à 48 positions mémoire. A nouveau le signe > indique la position courante.

Il y a toujours 3 octets sous la position courante. Ils sont donnés en format hexadécimal.

En affichage hexadécimal, chaque ligne de la position courante est donnée dans un format similaire: d'abord l'adresse en hexadécimal de la location, ensuite son contenu, et ensuite le code désassemblé lu.
Les codes mnémoniques standards du 6502 sont employés, avec les indications sur le mode d'adressage conventionnellles.

Plusieurs choses ont été ajoutées pour aider au débogage. Un mnémonique en inverse vidéo indique qu'un point d'arrêt a été placé à cet endroit. Si vous regardez effectivement ce qu'il a dans cette case mémoire, c'est 0. Une instruction BRK en inverse vidéo signifie qu'une instruction BRK particulière n'a pas été placée par DDT. Celà arrive, par exemple, en regardant la mémoire toute à 0.

Deuxièmement, si l'instruction est une instruction de branchement, et que la flèche vers le haut ou la flèche vers le bas est ajoutée au code désassemblé pour indiquer le sens du branchement conditionnel, l'adresse calculée du branchement conditionnel est aussi montrée.

Enfin ,si la partie adresse d'une instruction est une adresse définie dans la mini table des symboles, on verra sur l'écran le nom du symbole plutôt que sa valeur hexadécimale. La caractéristique du symbole peut être utilisée pour localiser les références du symbole ou dans le code simplement comme étiquettes pour rendre plus lisible le désassemblage.

Si le filtre hexadécimal est en place, chaque ligne qui se trouve après la position courante commence sur un multiple de 4 octets. Celà signifie que la ligne de la position courante peut avoir de 1 à 4 valeurs associées. Les valeurs d'une ligne de position courante seront toujours cadrées à gauche.

ATARINSIDE

AFFICCHAGE DE LA PILE.

La partie supérieure centrale de l'écran montre les valeurs du haut de la pile. Si le pointeur de pile est placé en $E0 ou plus haut (c'est à dire qu'il y a moins de 15 entrées dans la pile), alors on ne verra que les valeurs de la pile. L'affichage se fait du haut vers le bas. Si plus de 15 éléments sont dans la pile, alors on ne verra que les 15 derniers entrés.

Exemples

```
      .--.          .--.          .--.          .--.          .--.
SP=$FF:  :  SP=$FE:B9:  :  SP=$FD:B9:  SP=$F0:B9:  SP=$EF:A8:
      :  :          :  :          :A8:          :A8:          :A7:
      :  :          :  :          :  :          :A7:          :A6:
      :  :          :  :          :  :          :A6:          :A5:
      :  :          :  :          :  :          :A5:          :A4:
      :  :          :  :          :  :          :A4:          :A3:
      :  :          :  :          :  :          :A3:          :A2:
      :  :          :  :          :  :          :A2:          :A1:
      :  :          :  :          :  :          :A1:          :A0:
      :  :          :  :          :  :          :A0:          :B9:
      :  :          :  :          :  :          :B9:          :B8:
      :  :          :  :          :  :          :B8:          :B7:
      :  :          :  :          :  :          :B7:          :B6:
      :  :          :  :          :  :          :B6:          :B5:
      :  :          :  :          :  :          :B5:          :B4:
       --            --            --            --            --
```

MINI TABLE DE SYMBOLES

La partie supérieure droite est réservée à la mini table de symboles. Il y a de la place pour 15 variables dans cette table. Cette caractéristique vous permet de visualiser le contenu de variables choisies sans perturber leurs implantations réelles. Deux octets sont visualisés dans l'ordre poids forts, poids faibles (même ceux qui sont placé physiquement dans l'ordre poids faibles- poids forts). Cette table est implantée 3 octets après le début de DDT. Les 3 premiers octets contiennent JMP à l'entrée de DDT. 135 places mémoires sont réservées à la mini table de symboles. Chaque symbole de la table a le format suivant:

NOM	LOCATION	OCTETS A MONTRER
6 caractères pour le nom du symbole	adresse du symbole 2 octets	1 ou 2 octets

Un exemple de la configuration de la mini table de symboles avec l'utilisation de AMAC (Macro Assembleur ATARI) serait:

```
        ORG DDT+3       ; celà nous positionne au début
                        ; de la table des symboles
        DB  'VAR1  '    ; 6 caractères SVP
        DW  VAR1        ; laissons l'assembleur donner
                        ; la valeur a mettre ici
        DB  1           ; soit 1 ou 2 pour indiquer
                        ; que la variable sera montrée comme une   valeur au
                        ra 1 ou 2 octets.
```

Vous pouvez utiliser la mini table de symboles pour garder et montrer des variables standards du système

```
        DB  'COLPF2'
        DW  710
        DB  1
```

Vous pouvez gérer une petite zone de mémoire en plaçant des variables fictives, chacune pointant sur une ou deux cases mémoires.

La mini table de symboles a d'autre utilisations qui vous font parfois faire des découvertes heureuses. Vous pouvez, par exemple, utiliser une étiquette d'un programme comme un symbole. La valeur montrée n'aura aucun sens, mais le code de désassemblage s'en trouve plus lisible:

```
      DB   ':LOOP1 '
      DW   :LOOP1
      DB   1
```

Vous pouvez même définir un symbole comme "------" ou autre chose pour séparer différentes zones de la table des symboles. Et vous pouvez utiliser la table des symboles pour localiser une partie de votre programme: il faut pour celà enregistrer des positions mémoires fictives.

```
LCODE DW  :CODE
```

ATARINSIDE

Vous pourrez ainsi définir le symbole dans la table

```
DB  'LCODE '
DW  LCODE
DB  2
```

La valeur affichée sera alors l'adresse du module CODE

Vous ne devez pas définir plus de symboles que ceux dont vous avez besoin. Regardons les petits programmes d'illustration pour mieux nous rendre compte de l'utilisation de la mini table des symboles dans différents domaines. Notez que vos définitions seraient les dernières entrées dans le programme principal. Ceci pour être sûr que les définitions des symboles viennent après DDT, qui initiallement remplit la table ainsi:

```
ORG  :SSYMT
ECHO 15
DB   '
DW   0
DB   1
ENDM
```


ATARI INSIDE

TABLE DES POINTS D'ARRETS.

La table des points d'arrêt est située juste au dessus de l'affichage des registres. On peut définir jusqu'à 6 points d'arrêt et 2 points d'arrêt sur erreur, chacun est montré avec sa position courante. Si un registre est effacé (non positionné), la valeur indiquée sera 0000. Si un registre de point d'arrêt est positionné, la valeur de ce registe est l'adresse mémoire de l'instruction BRK correspondante. Cependant, dans le cas d'un point d'arrêt sur défaut, l'instruction BRK n'est pas utilisée. Les valeurs sont utilisées en mode interprété pour créer l'équivalent d'une instruction BRK.

FENETRE DES COMMANDES

La partie inférieure droite de l'écran est réservée pour la fenêtre des commandes, cette zone affiche la commande que vous entrez.

PIEGE (TRAP)

Les points d'arrêt sur "piège" sont réservés pour le mode interactif. Dans ce mode les points d'arrêt en mémoire sont ignorés, puisque DDT a le contrôle du système. A la place, DDT vérifie les valeurs dans les registres "piège". Si en outre la valeur de ce registre est égale à l'adresse de la prochaine instruction à exécuter, DDT stoppe le mode interprété. Celà permet de placer des pseudo points d'arrêts dans la ROM, par exemple. Il est alors plus facile et plus rapide d'atteindre un certain endroit d'un code dans la ROM, en positionnant un "piège", et lançant le mode interprété, plutôt que procéder pas à pas pour atteindre la position désirée.

ATARINSIDE

LES POINTS D'ARRETS

L'une des techniques les plus classiques du débogage est de placer dans le programme des points d'arrets. Supposez que vous débogguiez un programme qui détruit le système. Une des première chose que vous pouvez faire est de regarder le programme et de dire, "Je me demande où il est arrivé". Vous allez alors placer une instruction BRK qui appellera DDT. Lorsque vous lancerez alors le programme, il y a deux solutions: Si le programme atteint le point d'arrêt, c'est que le problème est après cet endroit. Si par contre le programme saute le point d'arret et ne s'arrête pas à ce point c'est que le problème est situé avant. Vous faites ainsi une première localisation du problème et répétant l'opération, vous arriverez éventuellement à voir où est le problème.

Le méchanisme du point d'arrêt est la voie la plus commune pour transférer le contrôle à DDT. Quand le programme s'exécute, le point d'arrêt donne le contrôle à DDT, ce dernier a été initialisé. L'écran de DDT est alors visualisé et les touches de fonction sont validées ainsi que l'interpréteur des commandes. Le point d'arrêt reste même après qu'il ait été rencontré au cours de l'exécution.

Après qu'un point d'arrêt ait été rencontré et que le contrôle ait été transféré à DDT, il y a plusieurs façons de quitter DDT. La commande "C" positionne un point d'arrêt à la position courante et le programme continue ensuite son exécution. START relance simplement l'exécution. "G" peut être utilisé pour transférer le déroulement des opérations à une autre place mémoire.

On peut à chaque fois donner jusqu'à 6 points d'arrêts. La position des points d'arrêts est donnée dans l'affichage des registres points d'arret. Si un point d'arrêt est supprimé, il est indiqué 0000. Lorsqu'un registre de point d'arrêt prend une nouvelle valeur et que ce registre contenait déjà un point d'arrêt, l'instruction correspondante est revalidée. Notez qu'il y a un système interne de point d'arrêt utilisé par la commande "C". Si n'importe quel point d'arrêt est rencontré, y compris le point d'arrêt de "C", le contrôle est donné à DDT. Le point d'arrêt pour "C" est supprimé.

TOUCHES DE FONCTION POUR LE CONTROLE

START - utilisé pour continuer l'exécution du programme à l'adresse indiquée par le registre PC.

SELECT - utilisé pour montrer alternativement l'écran DDT et l'écran visualisé avant l'appel de DDT. Un effort a été fait pour permettre la plupart des cas possibles, comme des modes mixtes dans la display list, les routines VBLANK, autres sets de caractères,interruption de Display List, changement dans les dimensions du champ visuel et les player-missiles.

OPTION - utilisé pour exécuter le programme pas à pas. Cela met en marche le filtre de désassemblage mais il ne supprimera pas forcément l'écran du programme. Référez vous à la partie PAS A PAS pour plus d'informations.

ATARINSIDE

L'INTERPRETEUR DES COMMANDES

L'interpréteur des commandes est le module qui vous permet d'entrer au clavier des commandes pour DDT. La fenêtre des commandes est visualisée dans la partie inférieure droite de l'écran. La partie inférieure gauche de l'écran étant destinée à montrer l'état des registres de la machine.

Chaque commande est donnée par une simple touche. Cependant, selon la commande, des arguments supplémentaires peuvent être nécessaires. Si la touche sélectionnée n'est pas valide pour DDT, elle sera simplement ignorée.

Les touches de commandes de DDT sont:

```
E <adr> ..... Contenu de l'adresse adr
C ..... continuer et quitter le point d'arrêt
G <adr> ..... allez à l'adresse adr
B <1-6>,<adr> ..... mettre le point d'arrêt 1-6 à adr
R <PC,A,P,X,Y,S>,<val> mettre val dans le registre choisi
D <hchaine> ..... mettre chaîne de caractère hexadécimal
v ..... Descendre la fenêtre de visualisation
è ..... Monter la fenêtre de visualisation
I ..... Mode interprété
W ..... Changement de filtre de la fenêtre
T <1-2>,<adr> ..... piège à l'adresse
S <hchaine> ..... Cherche la chaîne de caractère hchaine
```

Ces commandes sont décrites dans les pages suivantes.

ENTREE D'UNE VALEUR

Certaines des commandes du clavier demandent l'entrée de une ou plusieurs valeurs. L'entrée d'une valeur est indiquée par un délimiteur (espace, virgule ou RETURN). Quand deux valeurs sont nécessaires comme dans la commande B, une virgule doit être placée entre les deux valeurs. Taper un délimiteur sans taper une valeur sans avoir entré de valeur annulera la commande (sauf -voir les commandes points d'arrêt et piège)

Dans les explications suivantes les abréviations suivantes sont adoptées:

```
<adr>           = l'adresse sur 4 digits (seulement en hexadécimal)
<1-6>           = un des chiffres 1, 2, 3, 4, 5 ou 6
<PC,A,P,X,Y,S>  = soit PC, A, P, X, Y ou S
<val>           = une valeur qui peut être un octet, ou une adresse,
selon le registre choisi
<hchaîne>       = une chaîne de jusqu'à 10 caractères hexadécimaux
```

L'interpréteur des commandes (CI) ignorent les caractères autres que 0-9 et A-F en entrée. Pour supprimer un caractère, utilisez la clef Del.

A chaque fois qu'une commande est attendue, le CI positionne un champ correspondant au nombre maximum de digits hexadécimaux qui peuvent être entrés. (ex. 4 digits pour une adresse). Quand cette limite est atteinte, plus aucun digit n'est accepté. Vous pouvez enlever des caractères et alors en remettre d'autres. Supprimer après le point d'entrée annulera purement et simplement la commande.

ATARINSIDE

EXAMINER E<adr>

Utiliser la commande EXAMINER pour positionner la fenêtre de visualisation sur une plage de mémoire. A l'extrémité gauche de l'écran sur la quatrième ligne, est placé le signe >. Ce signe pointe l'adresse courante qui a été entrée dans la commande E. Notez que cette commande ne modifie pas l'état du filtre de la fenêtre de visualisation et ne modifiera pas l'instruction qui vient d'être exécutée par une commande d'exécution pas à pas.

CONTINUER C

Utilisez la commande CONTINUER pour revenir au code qui a appelé DDT et continuer l'exécution du programme. Il fonctionne de manière similaire que la touche de fonction START dans le fait qu'il ira à l'adresse indiquée par le PC. Cependant "C" laisse derrière lui un système additionnel de point d'arrêt. De manière interne, cela se fait par le saut à l'instruction suivante puis par le positionnement d'un registre de point d'arrêt invisible à l'adresse juste passée. Seul un point interne est maintenu. Si un point d'arrêt était déjà enregistré, l'instruction correspondante sera d'abord rétablie avant que le nouveau point d'arrêt soit validé. Le point d'arrêt sera supprimé si n'importe quel point d'arrêt est rencontré pendant l'exécution du code (y compris le point d'arrêt de C lui même).

ALLER A G<adr>

Utilisez l'instruction ALLER A pour exécuter une partie de programme commençant à l'adresse adr. Avant que le contrôle soit transféré à cette position mémoire, tous les registres du microprocesseur sont chargés par les registres visibles dans l'écran DDT. cela se vérifie pour toutes les commandes qui lancent l'exécution d'un code.

ATARI INSIDE

POINT D'ARRET B<1-6>,<adr>

Utilisez la commande POINT D'ARRET pour positionner l'un des 6 registres de point d'arret à une adresse mémoire. Si un chiffre différent de 1-6 est entré comme numéro de registre la commande est terminée immédiatement. Notez que deux valeurs doivent être entrées pour cette commande (le numéro de registre et l'adresse du point arrêt). Chacun de ces champs doit se terminer par un délimiteur (p.e. tapez B1 A000 puis appuyez sur la touche RETURN.) Souvenez vous que n'importe quel délimiteur est traité de manière identique (espace, virgule et RETURN).

Quand un point d'arrêt est positionné, son contenu est indiqué dans le registre du point d'arrêt à l'écran. Physiquement, un 0 (correspondant à l'instruction BRK) est placé en mémoire à l'adresse indiquée. Si par une instruction EXAMINER, on regarde le contenu de la mémoire à cet endroit, on verra un 0 même si le code mnémonique reste le même que précédemment. Si un point d'arrêt est placée à une location mémoire, son code mnémonique apparaitra en inverse vidéo. Si le registre d'un point d'arrêt est déjà utilisé quand un nouveau point d'arrêt est demandé, l'instruction est d'abord rétablie au niveau du premier point d'arrêt.

Pour supprimer le registre d'un point d'arrêt et replacer le code initial, taper n'importe quel délimiteur après l'entrée du numéro de registre (p.e. tapez "B1,,", le code source sera rétabli). si vous essayez de supprimer un registre de point d'arrêt qui est à 0, il ne se passera rien du tout. Notez cependant, que si vous mettez un point d'arrêt en ROM ou dans une place mémoire inexistante, cela peut provoquer des choses intéressantes, mais sûrement pas ce que vous attendiez.

REGISTRE R<PC,A,P,X,Y,s>,<val>

Utilisez la commande REGISTRE pour modifier n'importe quel registre du microprocesseur 6502. Après avoir tapé R, vous ne pouvez entrer que PC ou A ou P ou X ou Y ou S. Tout autre caractère provoquera l'annulation de la commande R. SI le caractère A, X, Y ou S est entré , aucune autre lettre n'est permise jusqu'au délimiteur. La touche Del vous permet toutefois de supprimer le caractère entré. Si P est tapé, l'entrée additionnelle d'un C sera acceptée et indiquera le compteur de programme. P seul indique quand à lui le registre d'état. Avec A, P, X, Y et S, vous ne pouvez entrer que des valeurs hexadécimales sur 1 octet. Alors que PC accepte jusqu'à deux octets. Notez encore que cette commande demande deux arguments et deux délimiteurs.

ATTENTION! L'utilisation inconsidérée de cette commande, en particulier avec "P", "PC" et "S" peut être très dangereux pour le déroulement correct des opérations.

ATARINSIDE

DEPOSER D<hchaine>

Utilisez DEPOSER pour placer en mémoire une chaîne d'octets. Une chaîne de caractères hexadécimaux (10 caractères, soit 5 octets) peut être entrée. Les valeurs entrées seront entrées dans les places mémoires successives à partir de la position courante indiquée dans la fenêtre de visualisation et remplace ce qu'il y avait. La chaîne de caractères hexadécimaux entrée est décodée deux caractères par deux caractères. S'il reste un caractère seul à la fin, il sera interprété comme la partie poids faibles d'un octet. Si, par exemple, on entre la chaine 01AAB0, on verra enregistré 01, AA, B0 dans la mémoire. Si maintenant on entre 01AAB on verra dans la mémoire les octets 01, AA et 0B. Notez que le chargement des différentes cellules de mémoire ne changera pas la position de la fenêtre de visualisation. On contrôle la fenêtre avec les commandes EXAMINER, VOIR LE HAUT et VOIR LE BAS.

VOIR LE BAS (flèche vers le bas)

Cette commande vous permet de déplacer la fenêtre de visualisation vers le bas. Selon le filtre mis en place, le déplacement se fera d'un octet (filtre hexadécimal) ou d'une série d'octets (filtre de désassemblage). Notez que la répétition automatique de la touche est valide et si vous appuyez continuellement sur cette touche, vous verrez un défilement vers le bas (il n'est pas nécessaire de tenir appuyée la touche CTRL).

Si vous appuyez sur la touche SHIFT en même temps que vous appuyez sur la touche "flèche vers le bas", là fenêtre se déplacera d'un écran complet à chaque fois.

VOIR LE HAUT (flèche vers le haut)

Cette commande vous permet de déplacer la fenêtre de visualisation vers le haut. Selon le filtre mis en place, le déplacement se fera d'un octet (filtre hexadécimal) ou d'une série d'octets (filtre de désassemblage). Notez que la répétition automatique de la touche est valide et si vous appuyez continuellement sur cette touche, vous verrez un défilement vers le haut (il n'est pas nécessaire de tenir appuyée la touche CTRL).

Si vous appuyez sur la touche SHIFT en même temps que vous appuyez sur la touche "flèche vers le haut", là fenêtre se déplacera d'un écran complet à chaque fois.

Un problème survient parfois, quand vous visualiser le contenu de la mémoire avec le filtre désassembleur. Si vous essayez de voir le haut de la mémoire, l'ordinateur n'est pas capable de savoir si l'instruction précédant le point actuel fait 1, 2 ou 3 octets. DDT garde en mémoire le nombre d'octet dont il se déplace lorsqu'il regarde vers le bas de la mémoire. Ceci est fait à chaque fois. Il est alors possible de déplacer la fenêtre de visualisation vers le haut et de voir défiler les instruction correctement. Référez vous à l'apendice technique pour de plus amples informations.

ATARI INSIDE

MODE INTERPRETE I

Utilisez le mode INTERPRETER pour placer le système dans un mode d'exécution automatique pas à pas. Après l'exécution de chaque instruction, l'écran est mis à jour si l'on se trouve dans l'écran DDT. La fenêtre de visualisation est automatiquement placée en mode interprété. En appuyant sur la touche BREAK, on arrête le déroulement du programme . Il est possible d'exécuter ainsi des routines de la ROM, par exemple du BASIC, de manière interprétée, mais des problèmes de visualisation peuvent survenir en essayant de lancer des routines du Système d'Exploitation de manière interprétée. Le registre Piège est utilisé pour positionner l'équivalent d'un point d'arrêt dans ce mode. Le mode interprété est beaucoup plus rapide si on le lance avec l'écran de l'utilisateur plutôt que l'écran de DDT. Ceci est dû au fait que DDT n'a pas à mettre à jour son écran lorsque ce dernier est déconnecté.

FENETRE W

Utilisez la commande FENETRE pour changer de filtre dans la fenêtre de visualisation. Trois filtres sont possibles: un filtre opaque donnant le code d'exécution de DDT, le filtre de désassemblage et le filtre hexadécimal.

PIEGE T<1-2>,<adr>

Utilisez la commande PIEGE pour positionner l'un des points d'arrêt de piège à une adresse précise. L'adresse entrée doit être entrée dans un registre spécifique piège. Notez que piege n'est valide qu'en mode interprété. Pour supprimer le piège, tapez T puis 1 ou 2 pour le registre piège que vous voulez supprimer et tapez ensuite deux délimiteurs. 0000 apparaitra dans le registre.

CHERCHER S<hchaine>

Utilisez CHERCHER pour localiser une séquence spécifique de caractères hexadécimaux en mémoire. Vous pouvez entrez jusqu'à 10 caractères hexadécimaux (5 octets). La mémoire sera parcourue depuis la position courante indiquée dans la fenêtre de visualisation et vers le haut, jusqu'à l'adresse C000. Cette limite est la limite permise par le système. La recherche ne se fait pas dans l'espace mémoire au dessus de C000. NOtez que vous pouvez quand même commencer la recherche plus haut que la limite. Si vous examiner par exemple le contenu de l'adresse F111 et que vous lancez une recherche à partir de ce point. La recherche se fera de F111 jusqu'à FFFF, puis repartira de 0 jusqu'à C000. Si l'ordinateur trouve la chaîne de caractères, la fenêtre de visualisation sera repositionnée, sinon la fenêtre des commandes sera simplement effacée.

ATARI INSIDE

LES POINTS D'ENTREE DANS DDT

Il y a trois façons d'entrer dans DDT:

ENTREE DIRECTE
ENTREE EN COURS
ENTREE PAR POINT D'ARRET

ENTREE DIRECTE

Le point d'entrée est prévu pour permettre l'entrée immédiate dans DDT sans se préhocuper d'autres considérations. C'est un caractère spécial du clavier qui est initialement donné par CTRL, SHIFT et ESC (c.à.d. en appuyant simutanément sur les touches CTRL, SHIFT et ESC). Quand DDT est initialisé, la partie du Système d'Exploition qui gère le clavier est modifiée de sorte qu'il vérifie d'abord le caractère spécial avant de prendre en main les entrées normales du claviers. Si ce caractère est trouvé, on entre directement dans DDT par le point d'ENTREE DIRECTE.

La commande "C" ou l'appui sur START rend le contrôle à l'endroit où le processeur était quand le caractère spécial de DDT a été tapé. Pour plus d'information sur le méchanisme de l'ENTRE DIRECTE, voir la partie sur le balayage du clavier dans l'appendice des détails techniques.

ATTENTION! Ne jamais utiliser l'ENTREE DIRECTE 2 fois pour appeler DDT sans être d'abord sorti de celui-ci. Si vous le faisiez, vous pouvez rendre impossible le retour au point d'appel original.

Quand vous utilisez l'ENTREE DIRECTE vous noterez que la position courante indiquée est une séquence de code comme suit:

```
PLA
TAX
PLA
TAY
PLA
RTI
```

C'est une partie du code de DDT qui simule l'entrée d'un point d'arrêt dans DDT. Pour atteindre l'instruction actuelle en code machine qui serait exécuté, écrivez simplement les 6 instructions ci-dessus.

ATARINSIDE

ENTREE EN COURS

Ce point d'entrée est le point de départ pour le code de DDT.Les 3 premiers octets repréentent l'instruction JMP DDT ENTRY. Si cette adresse mémoire est appelée par une instruction JSR, alors l'appui sur la touche de fonction START rendra le contrôle au point d'appel. Ceci permet à DDT d'être appelé à des positions variées du programme pour établir des points d'arrêts, changements de valeurs, etc...

Exemple

```
        :
        :
        :
---- votre code ---
        :
        PHA          ; ceci n'est qu'un exemple
        JSR DDT
---- appuyez sur START pour revenir ici-----
```

Quand vous utilisez l'ENTREE EN COURS, la position courante donnera une instruction RTS. Comme avec l'entrée directe, c'est dans ce cas une partie de DDT qui est utilisée pour implémenter le mécanisme d'entrée. Il ne faut alors exécuter que quelques instructions pour arriver au code de votre application.

ENTREE PAR POINT D'ARRET

Les entrées par points d'arrêt sont les solutions les plus courantes pour entrer dans DDT. Les point d'arret doivent d'abord être établis via une ENTRE DIRECTE ou une ENTREE EN COURS à DDT. Après qu'ils soient positionnés, DDT sera appelé si ces instructions spécifiques sont exécutées. Les sorties des points d'arret de DDT reviennent à la séquence de code où le point d'arrêt a été placé. Notez que les points restent en place à moins qu'ils ne soient spécifiquement supprimés. Celà est vrai, meme quand le point d'arrêt est passé.

Rappelez vous également que si un registre piège est positionné dans un mode interprété, si vous essayez alors d'exécuter l'instruction à cette adresse, le mode interprété sera interrompu. Aussi, pour passer au dessus d'un point d'arrêt piège, vous devez soit supprimer le piège, ou passer au dessus de l'instruction piège et entrer alors dans le mode interprété.

ATARINSIDE

COMMENT UTILISER DDT

LES EXEMPLES

DDT contient plusieurs programmes d'exemple pour placer DDT dans différentes positions. Allumer votre ordinateur et jouez avec DDT pendant que vous lisez.

CHARGEMENT DE DDT DANS LA MEMOIRE DE L'ORDINATEUR.

1 . Placez la cartouche BASIC ATARI dans son connecteur pour l'ATARI 800. Le Basic est intégré pour le 800 XL.

2 . L'ordinateur est-il bien éteint.

3 . Allumez votre unité de disquette.

4 . Quand le voyant BUSY est éteint, ouvrez la porte d'accès et insérez la diskette DDT avec l'étiquette en haut à droite dans le coin le plus proche de vous. (Utilisez l'unité 1 si vous avez plusieurs unités de disquette).

5 Allumez votre ordinateur et l'écran. Le programme se chargera en mémoire et affichera le READY propre à BASIC.

Est-ce-que tout est normal à ce niveau? Vous pouvez même taper un petit programme comme suit:

```
10 FOR I=1 TO 1000
20 PRINT "I=";I
30 NEXT I
40 GOTO 10
```

Tapez RUN pour lancer le programme et appuyez maintenant sur les 3 touches suivantes en même temps: SHIFT, CTRL et ESC (elles sont toutes dans la partie gauche du clavier). Apparaît le message "Welcome to DUNION's DEBUGGING TOOL", plus connu sous le nom de DDT.

Il y a plusieurs programmes "SHELL" en langage assembleur que vous devrez regarder. Il est alors nécessaire d'utiliser l'Editeur de Texte ATARI (MEDIT). L'idée de base derrière le concept "Interpréteur du langage de commande" (SHELL) est de laisser tel quel les modules de code source (DDT.MAC, DDTLST.MAC, et le module source que vous débogguez), en les modifiant le moins possible. Avec "l'interpréteur du langage de commande", vous pouvez réaliser tous les changements nécessaires dans le programme "SHELL" sans changer les autre fichiers. chacun de ces "SHELLS" est décrit dans la prochaine section.

ATARI INSIDE

RELIER VOTRE PROGRAMME A DDT

Le programme en langage assemblé appelé SHELL.MAC est le programme général que vous devrez utiliser pour assembler votre programme avec DDT. Un listage de ce programme se trouve dans la partie DETAILS TECHNIQUES de ce manuel. Jetez un coup d'oeil à ce listage. Comme vous pourrez le voir, le programme SHELL est un guide pour attacher votre programme à DDT. Considérons un programme assemblé normalement par le macro-assembleur :

```
D:VOTPROG.MAC S=D:SYSTEXT.MAC
```

La procédure générale que voudriez suivre serait de charger SHELL.MAC avec MEDIT, l'éditer suivant les instructions de SHELL.MAC, sauver le fichier et l'assembler avec la commande

```
D:SHELL.MAC S=SYS/SYSTEXT.MAC.
```

cela produira un fichier objet appelé SHELL.OBJ que l'on rappele AUTORUN.SYS pour permettre le chargement automatique.

D'autres programmes SHELL montrent comment se familiariser à ce procédé. Chacun des programmes SHELL indique comment ils ont été intégrés. Pour voir comment chacune de ces versions travaille, renommez le code objet choisit AUTORUN.SYS (p.e. renommer SHELL1.OBJ en AUTORUN.SYS). Malheuresement, à cause des contraintes d'espace, je n'ai pas pu laisser le code objet de chaque module SHELL. SHELL2.OBJ est considéré comme le fichier courant AUTORUN .SYS, et SHELL3.OBJ n'est pas présent du tout. Pour créer ce fichier vous devez assembler le fichier SHELL3.MAC.

SHELL1.MAC est la version automatique créée tout d'abord pour vous familiariser avec DDT. Les variables de la mini table de symboles sont certaines de celles qu'utilisent le Système d'Expoitation pour contrôler le système. Cette version de DDT est fort utile pour comprendre les particularités graphiques et autres caractéristiques du système. Vous pouvez facilement visualiser et changer la mémoire écran, les registres fantômes, et ainsi de suite. Vous pouvez même écrire en mémoire un petit programme en langage machine avec la commande DEPOSER.

Il y a plusieurs choses à noter dans cette version. D'abord, si vous utilisez la touche de fonction START ou la commande "C", alors le fichier DUP.SYS sera chargé au dessus de DDT. Après cela, vous devez recharger DDT pour le réutiliser.

Ensuite, comme le mécanisme d'ENTREE DIRECTE est utilisé pour entrer dans DDT,vous ne pouvez pas utiliser le point d'ENTREE EN COURS pour réentrer dans DDT. cela rendra impossible le retour par la voie normale au fichier DUP.SYS.

ATARINSIDE

SHELL2.MAC est la version qui vous permet d'examiner le fonctionnement interne du programme BASIC. Notez que les variables dans la mini table des symboles sont les variables BASIC utilisées pour gérer la mémoire. Une chose intéressante à faire et de lancer un programme BASIC puis d'appuyer en meme temps sur les touches SHIFT, CTRL et ESC pour accéder à DDT, appuyer sur SELECT pour sélectioner l'écran BASIC et ensuite appuyer sur I pour lancer le programme en mode interprété. Le programme BASIC sera ralenti dans un rapport de 100. Laisser le programme s'exécuter jusqu'à un point qui vous semble intéressant, puis appuyez sur la touche BREAK pour stopper le mode interprète et revenir à l'écran DDT. Utilisez alors DDT pour voir ce que fait BASIC dans son fonctionnement interne.

SHELL3.MAC est la version développée pour tester un programme écrit en langage assembleur. Une routine appelée PSEUDO.MAC dans la disquette est l'implémentation d'un générateur de nombres pseudo aléatoires. En bref, cette routine va générer un nombre pseudo aléatoire inférieur ou égal à une limite appelée "limite supérieure". Pour plus d'inforamtion sur l'écriture de cette routine appelez le code source grace à MEDIT. Après avoir assemblé SHELL3.MAC, renommer le code objet obtenu SHELL3.OBJ en AUTORUN.SYS. En initialisant à nouveau l'ordinateur, le système chargera DDT et PSEUDO, initialisera DDT et fera un JSR DDT au premier point d'arrêt positionné.

Avec cette version, vous commencerez à vous rendre compte de la puissance de DDT. Par exemple, si vous testez un sous programme travaillant avec des valeurs numériques (comme le fait PSEUDO), Il n'est alors pas nécessaire de placer une routine d'impression pour connaître la sortie du sous programme. Il est très simple de placer le résultat dans une case mémoire et de placer cette case mémoire en entrée dans la mini table des symboles.

Ensuite, remarquez comme la mini table de symboles peut être utile dans de nombreuses circonstances. Un symbole peut servir à gérer une sortie de routine (VALUE) des parametres d'entrée, (UPPER et DEGRAN) et meme des zones de mémoire (RANNUM+RANNM2=4 octets côtes à côte). Un symbole peut encore etre simplement défini comme une place mémoire (ex. ETIQU) dans votre code source. Leurs valeurs à l'écran est sans intérêt, mais le listage du désassemblage n'en n'est que plus lisible. Vous pouvez même utiliser une variable appelée "....." pour séparer un symbole variable et un symbole étiquette.

Pour avoir une idée de la façon d'utiliser DDT, copiez SHELL3.OBJ dans AUTORUN.SYS, enlevez les cartouches de votre système, puis réinitialisez le système. Vous entrerez directement dans DDT. Tapez sur "W" pour changer l'écran, puis appuyez sur OPTION deux fois pour passer directement au début du programme de PSEUDO. Placez un point d'arrêt à l'instruction JMP LOOP (vous verrez cette instruction en déplaçant la fenêtre vers le bas, elle se trouve à l'adresse $4018). Appuyez maintenant sur START. L'écran va clignoter puis reviendra avec le PC à $4018. Continuez ainsi. Notez les moments où le contenu de VALUE est inférieur ou égal à UPER. Positionnez un piège (TRAP) à $4018 et lancez le mode interprété, etc..

SHELL4.MAC a été développé pour déboguer un programme hybride (avec une partie BASIC et une autre en langage assembleur par exemple). Le code objet correspond ici à la routine générateur de nombres pseudo aléatoires., le lien avec BASIC et DDT. Pour utiliser cette version, renommer SHELL4.OBJ en AUTORUN.SYS puis recharger le système. Quand vous voyez le READY apparaître, taper alors RUN"D:PSEUDO" et appuyez sur la touche RETURN.

Dans le programme BASIC PSEUDO, vous pouvez remettre à 0 le noyau ou le point de départ pour le générateur de nombres pseudo aléatoires. Positionnez le noyau à une valeur donnée, et entrez la valeur de la limite supérieure ainsi que le nombre de valeurs à générer. Notez les nombres pseudo aléatoires générés. Maintenant vous recommencez et vous replacer le noyau à la même valeur qu'il y avait précédemment. Replacez également la même limite supérieure et le nombre de valeurs à obtenir. Vous obtiendrez la même liste de nombres générés. C'est de fait l'intérêt d'un générateur de nombres pseudos aléatoires, la capacité de générer les mêmes séries de nombres qui semblent aléatoires.

INTERACTION AVEC DOS

Si vous décidez de placer l'origine de DDT juste en haut de la partie FMS de DDT, vous devez vous assurer que c'est exactement là que DUP.SYS sera chargé. Ainsi si vous essayer de charger DUP.SYS (avec la commande DOS de BASIC par exemple), il couvrira DDT. Aucun problème réel ne résultera de cette opération, mais vous rencontrerez de sérieuses difficultées en essayant de recharger DDT à partir du DOS. Vous devez par exemple créer un MEM.SAV avant que le système d'exploitation vous laisse couvrir DUP.SYS. En général, si vous devez utiliser DUP.SYS, placez DDT en dessous de l'espace mémoire où DUP.SYS sera chargé.

Si vous voulez appeler DOS depuis DDT, vous avez plusieurs possiblités. Une solution est de placer dans votre code une instruction du genre:

```
DOSCALL   JMP  (DOSVEC)      ; DOSVEC = $0A
```

Vous appelez alors DOS par la commande "G" avec l'adresse de DOSCALL.

ATARINSIDE

APPENDICE - DETAILS TECHNIQUES

BALAYAGE DU CLAVIER

Pendant l'initialisation de DDT, le vecteur de la gestion clavier est repositionné vers un préprocesseur qui vérifie le caractère spécial pour ENTREE DIRECTE. Si ce caractère est reconnu, le contrôle est transféré vers le point ENTREE DIRECTE. Sinon le contrôle est rendu à la routine normale de scrutation clavier.

Quand vous écrivez votre application il faut savoir certaines choses au sujet du travail du préprocesseur.

1 . les interruptions clavier doivent être validées.

2 . Le caractère recherché est enregistré dans une table interne et il peut être changé. Dans le code source l'adresse est à DBCHR et est initialement placé à $DC.

PAS A PAS

DDT est pourvu d'un système de scrutation pas-à-pas pour analyser l'exécution d'un code. Celui ci est activé en appuyant sur la touche de fonction OPTION ou à partir de la commande I. La commande I active un mode automatique de scutation pas-à-pas qui s'arrête lorsque l'on appuie sur la touche BREAK.

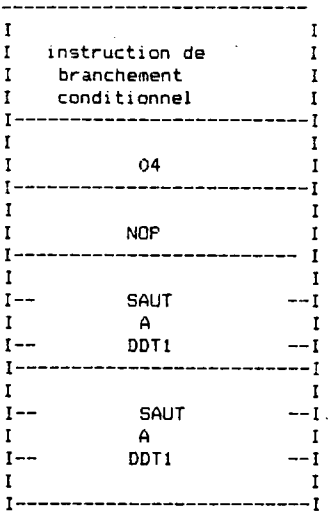
Quand la requête d'un fonctionnement pas-à-pas est prise en compte, DDT fait l'analyse de l'instruction actuellement pointée par le registre PC. Si ce n'est pas une instruction interdite (c.à.d. une instruction qui détruirait DDT), l'instruction est transférée vers un banc de test, les registres du 6502 sont chargés depuis les registres fantôme, et alors l'instruction est exécutée. Après l'exécution, le registre d'état est sauvegardé, l'écran est mis à jour, et le contrôle revient à DDT.

Si DDT ne permet pas directement l'instruction qui doit être exécutée (par exemple une instruction JMP) alors l'instruction est simulée, l'état et l'affichage des registres sauvegardés sont mis à jour avant que le contrôle soit rendu à DDT. Les instructions interdites comprennent les instructions de branchement JMP, branchement indirect, JSR, RTI, RTS et BRK.

Si un point d'arrêt est rencontré pendant une exécution pas-à-pas, DDT prend l'instruction qui devait être à cet emplacement mémoire avant l'exécution. Si pour une raison quelconque, une instruction BRK ne correspond pas à un point d'arrêt que vous avez défini, un NOP sera chargé à la place. C'est aussi le cas pour une instruction non définie.

ATARI INSIDE

Les instructions de branchement sont traitées de manière hybride . L'instruction de branchement est placée sur un banc de test comme nous l'avons vu précédemment. Après l'exécution de l'instruction de branchement, DDT peut mesurer la distance entre l'adresse de l'instruction de branchement et l'adresse de branchement. Cette valeur sert à calculer l'adresse qui sera placée dans le PC.



ATARINSIDE

UTILISATION DES RESSOURCES SYTEMES DE DDT

Le code DDT utilise par lui même 6K de RAM, et l'écran 1 autre K. Un soin particulier a pris pour assurer à DDT un fonctionnement parallèle au système fonctionnant normalement. En mode interprété, par exemple, vous pourrez utiliser toutes les possiblités de l'ordinateur y compris le clavier et les clés de fonction (sauf la touche BREAK que DDT se réserve). Une fonction sous-jacente de DDT est que votre programme fonctionne généralement dans un environement

En accord avec les protocoles établis par le Système d'Exploitaion en place. Il y a 6 adresses en page 0 que DDT utilise quand il est actif (2-7). Le système d'exploitation ne doit pas utiliser ces adresses pendant que DDT est activé. Dans le cas où votre programme utilise ces adresses (ce n'est pas grave!), elles sont sauvegardées avant l'entrée dans DDT et remises en place avant la sortie de DDT. Si cependant elles sont utilisées pendant que DDT a le contôle, ce sont les valeurs de DDT que vous verrez et non celles de votre programme.

DDT n'utilise que 2 variables globales DDTI et ECODE. Les deux sont utilisées dans SHELL.MAC. Sinon, toutes les autres variables sont locales. Les programmes SHELL utilisent aussi les variables globales DDT et ICODE.

MOUVEMENT DE LA FENETRE D'AFFICHAGE

DDT place une pile d'entrée quand le filtre de désassemblage est placé. cela signifie qu'à chaque fois que vous allez descendre la fenêtre de visualisation, DDT place dans la pile le nombre d'octets qui sont pris pour le déplacement. Alors, pour remonter la fenêtre de visualisation, DDT regarde s'il reste des valeurs dans la pile. Si c'est le cas vous pouvez remonter la fenêtre, sinon il ne se passe rien. La pile est vidée quand on entre dans DDT, ou quand une commande EXAMINER est entrée. Pour garder en mémoire le nombre d'octets de déplacement, on place 4 valeurs par octet de la pile. 1, 2 ou 3 sont les valeurs qui peuvent être enregistrées. 64 octets sont réservés pour la pile. Vous pouvez donc déplacer la fenêtre de 256 instructions avant que la pile soit remplie, auquel cas on commence par perdre les premières valeurs entrées et vous ne pourrez remonter au maximum que de 256 instruction. En terme informatique, la pile est implémentée en un buffer circulaire.

ATARI INSIDE

CE QU'IL FAUT REGARDER ATTENTIVEMENT

A mon grand regret, il reste des petites erreurs dans DDT. Celà arrive en général, quand vous travaillez en pas à pas ou en mode interprété. Si votre programme travaille sur la display list ou avec ANTIC ou le GTIA/CTIA, vous pouvez alors être bloqué avec un écran DDT transformé. Ceci n'est en général pas fatal, seulement un intermède. Pour retrouver l'écran DDt normal, il suffit d'appuyer sur la touche BREAK pour arréter le mode interprété, puis d'appuyer 2 fois sur la touche SELECT.

Essayer de de faire des entrées/sorties sur disque ou tout autre système ayant atrait à une activité temps réel en mode interprété ou en exécution pas à pas va probablement avoir des effets inatendus. Vous pourrez placer des points d'arrêt de telle sorte que les Entrées/sorties soient traitées en temps réel, et appeler ensuite DDT.

Attention à l'utilisation du point d'entree DIRECT (entré par l'appui simultané des touches SHIFT, CTRL et ESC). Il faut être d'abord sorti de DDT avant de pouvoir le rappeler.

Certains programmes peuvent être trop gros pour pouvoir tourner avec DDT. Si celà arrivait, AMAC se bloque simplement et se détruit. Vous pouvez remédier à celà en assemblant un SHELL qui contiendra DDT et un autre avec le programme de test. Il est vrai que vous devrez prendre certaines précautions: Les valeurs de ORG doivent être correctes, le code de test doit connaître où se trouve le code d'initialisation de DDT, ainsi que la mini table de symboles. Mais ce n'est pas tout. Après avoir produit les deux codes objets, renommez celui qui contient DDT en AUTORUN.SYS. Copiez ensuite l'autre code en AUTORUN.SYS avec l'option APPEND. DUP.SYS copiera cet autre code à la fin du code de DDT. Ne vous préhocupez pas de ce que les segments du codes soient localisés à des endroits différents. Le chargeur de programmes binaires placera les différents segments à leurs bonne place. Tout ce que vous avez à faire à faire est d'être sûr que la bonne mini table de symboles est chargé en dernier et que le dernier segment a la bonne adresse d'initialisation chargée dans le vecteur RUN.

Finalement pour passer successivement de DDT à DUP.SYS (s'ils se recouvrent l'un l'autre), il semble que des événements bizares se déroulent dans le système. Si celà arrive, essayez tout d'abord d'appuyer sur SYSTEM RESET, et si celà ne donne aucun résultat, rechargez simplement le système. Je sais que ce n'est pas une forme très élégante pour résoudre un problème, mais quand il n'y a pas le choix!

ATARINSIDE

```
**************************************************
*
*     CECI EST LE PROGRAMME GENERAL SHELL
*
*    VOUS DEVEZ ASSEMBLER CE PROGRAMME POUR
*    ATTACHER VOTRE PROGRAMME DE TEST
*    AU PROGRAMME DE DEBOGGAGE
*
*    REFEREZ VOUS A LA DOCUMENTATION DE DDT
*    POUR LES INSTRUCTIONS ET ADAPTER
*    CE PROGRAMME A VOS BESOINS PROPRES
*
**************************************************
*
*
*                 ETAPE  1
*
*     VOUS DEVEZ D'ABORD ETABLIR OU VOUS ALLEZ
*     PLACER VOTRE PROGRAMME ET DDT
*
*     UNE SOLUTION EST DE LAISSER DDT PLACE
*     EN HAUT DU DOS, ET ETRE DANS UN SENS
*     UNE EXTENSION DE IT
*
*     DANS CE CAS, L'INSTRUCTION ORG PLACE DDT
*     POUR QU'IL COMMNENCE JUSTE LA OU LE DOS
*     S'ARRETE. NOTEZ QUE C'EST POUR LA CONFIGURATION
*     STANDARD DOS 2
*
*     SI VOUS VOUS TROUVEZ DANS UNE CONDITION
*     PARTICULIERE ( PLUS DE BUFFERS D'UNITE DE
*     DISQUE, LE 850 ALLUME,...), CHANGEZ LE ORG
*     POUR QU'IL SOIT ADAPTE A VOTRE CONFIGURATION.
*
            ORG $1CFC
*
**********************************************************
*
*
*                 ETAPE  2
*
*     VOUS DEVEZ ETRE MAINTENANT SUR QUE VOTRE CODE DDT
*     EST ASSEMBLE. IL EST SUPPOSE QUE TOUS LES
*     FICHIERS NECESSAIRES SONT DANS LE LECTEUR DE
*     DISQUE 1.
*
*     VOUS POUVEZ CEPENDANT CHANGER LES SPECIFICATIONS
*     DE VOS FICHIERS POUR QU'ILS S'ADAPTENT A VOTRE
*     DEVELOPPEMENT.

            PROC
DDT         =           *
            INCLUDE     D:DDT.MAC
*
```


ATARI INSIDE

```
*****************************************************
*
*                 ETAPE 3
*
*   L'ETAPE SUIVANTE CONSTRUIT LA DISPLAY LIST
*   ET L'ECRAN POUR DDT.
*
*   LE CODE CORRESPONDANT PREND 1K JUSTE EN DESSOUS DE
*   L'ESPACE MEMOIRE ET A DES CONDITIONS DE LIMITE DE
*   CROISEMENT.
*
*   NOTEZ QUE LE ORG SUIVANT ASSURE QUE LA DISPLAY LIST NE
*   CROISE PAS UNE LIMITE DE 1K ET QUE LA MEMOIRE D'ECRAN NE
*   CROISE PAS UNE LIMITE DE 4K.
*
*   SI POUR UNE RAISON OU UNE AUTRE, VOUS VOULEZ DEPLACER
*   L'ECRAN, CHANGEZ CETTE INSTRUCTION.
*
*   NOTEZ EGALEMENT QUE L'ETIQUETTE ICODE EST UTILISEE POUR
*   DEFINIR UN POINT POUR ENREGISTRER LE CODE
*   D'INITIALISATION. 9 OCTETS SONT RESERVES A CELA.
*
          IF (((((((HIGH *)/4+1)*1024-*)<33) OR ((((((HIGH
          *)/8+1)*1024)-*)
          ORG (((HIGH*)/4+1)*1024
          ENDIF
*
          INCLUDE       D/DDTLST.MAC
          EPROC
   ICODE  =             *
          ORG           *+9
*
*
*****************************************************
*
*                 ETAPE  4
*
*   LE PROGRAMME D'INITIALISATION DE DDT PLACE
*   UNE ROUTINE QUI MODIFIE LE POINTEUR MEMLO
*   QUAND LA TOUCHE SYSTEM RESET EST APPUYE.
*
*   CECI EST NORMALEMENT UTILISE POUR PROTEGER LE
*   CODE DE DDT ET COMMENCER L'ESPACE LIBRE JUSTE APRES DDT.
*
*   POUR CHANGER CELA, VOUS DEVREZ DEFINIR UNE VALEUR ECODE
*   QUI SERA PLACEE DANS LE POINTEUR MEMLO.
*
*   UNE SUGGESTION SERAIT DE METTRE TOUT SIMPLEMENT A CET
*   ENDROIT LA VALEUR QUI Y SERAIT DE TOUTE FACON.
*
*   PAR EXEMPLE DANS LE DOS STANDARD, VOUS DEVEZ METTRE
*   ECODE     =          $1CFC
*
*
```

ATARINSIDE

```
*******************************************************
*
*                ETAPE 5
*
*   COMME VOUS DEVEZ ATTACHER VOTRE PROPRE CODE, UN CERTAIN
*   NOMBRE DE CHOSES PEUT ETRE NOTE.
*
*   1. VOUS DEVREZ ENLEVER TOUTES LES INSTRUCTIONS ORG DE
*   VOTRE PROGRAMME ET LES PLACER ICI. AVEC AUCUNE NOUVELLE
*   INSTRUCTION ORG, VOTRE CODE SUIVRA LE CODE DE DDT. CELA
*   SIGNIFIE QUE NORMALEMENT, VOTRE CODE COMMENCERA AUX
*   ALENTOUR DE L'ADRESSE $3715
*
*   2. ENLEVEZ TOUTES LES INSTRUCTIONS END DE VOTRE PROGRAMME.
*   SINON , TOUT CE QUI SUIVRA SERA INEFFICACE.
*
          ORG          VOTORIG
          INCLUDE      D:VOTPROG
*
*******************************************************
*
*                ETAPE 6
*
*   SI VOUS VOULEZ DEFINIR UNE TABLE DE MINISYMBOLES,
*   C'EST LE MOMENT. L'INSTRUCTION ORG DOIT PLACER
*   L'INSTRUCTION A TOUT ENDROIT OU DDT EST +3.
*
*   RAPPELEZ VOUS QUE CHAQUE SYMBOLE DOIT ETRE DEFINI COMME:
*
          DB           'SYMBOL'  ; 6 CARACTERES
          DW           SYMBOLE   ; ADRESSE DU SYMBOLE
          DB           1         ; 1 OU 2

          ORG          DDT+3
          ORG          DDT+3
          DB           '      '
          DW           0
          DB           1
*
*******************************************************
*
*                ETAPE  7
*
*   IL FAUT MAINTENANT DIRE AU SYSTEME OU
*   COMMENCER L'EXECUTION DU PROGRAMME
*
*   LA CONFIGURATION QUE NOUS AVONS ICI INITALISE DDT,
*   APPELLE DDT POUR VOUS PERMETTRE L'INITIALISATION DE POINTS
*   D'ARRETS, ET SE BRANCHE ENSUITE AU DEBUT DE VOTRE
*   PROGRAMME
*
*
          END          ICODE
*
*******************************************************
```


ATARI INSIDE

ATARI®

ATARINSIDE