ABZmon, par Alain Birtz
version 2.0
Manuel de référence

Alain Birtz
650 Grand St-Charles,
St-Paul d'Abbotsford
P.Q., Canada, J0E-1A0
CompuServe [72467,2770]

Le dossier ABZmon

Ce document est la version française du guide d'utilisation dans le dossier ABZmon*ƒ*. Ce dossier est constitué de trois fichiers: ABZmon.doc.français, la documentation en français, ABZmon.doc.english, la documentation anglaise et ABZmon, un fichier de démarrage (INIT). Ce logiciel est gratuit et peut être distribué librement à la condition de donner l'ensemble des trois fichiers.

Installation

Vous devez placer le fichier ABZmon dans le dossier système et redémarrer l'ordinateur. Pour le système 7 (ou plus récent), ABZmon peut aussi être placé dans le dossier des extensions. Si l'installation se fait correctement vous verrez apparaître l'icône suivant:

ABZmon est en activité dès que vous voyez cette icône. Le symbole de cette icône apparaît sur le commutateur d'interruption des Mac II ...

Si le système ne peut pas installer ABZmon, par exemple, pour mémoire insuffisante, vous verrez plutôt l'icône ci-dessous:

Vous pouvez annuler l'installation de ABZmon en pressant simultanément les touches **Option** et = au démarrage.

Présentation

ABZmon est un "débogueur" de bas niveau, un utilitaire pour dépister les erreurs de programmation. Il possède toutes les fonctions habituelles d'un tel outil, comme les points d'arrêt, un mode de suivi pas à pas, visualisation de la mémoire, des registres internes du microprocesseur, de la pile, etc.

Mais ABZmon possède aussi des fonctions que l'on ne retrouve pas souvent chez les autres "débogueurs". Par exemple, ABZmon possède une interface graphique utilisant fenêtres, menus et souris. Un texte peut être parcouru à l'intérieur du "débogueur". On peut abandonner une application "gelée" et continuer normalement vers une autre application.

De nombreuses fonctions ont été implantées pour faciliter le travail du

programmeur. Dans une fenêtre de code, par exemple, on peut voir une opérande, disons un pointeur comme **$4(a2,d0.l)**, directement en option-cliquant sur son écriture. Dans cet exemple, on pourra voir l'adresse du pointeur et aussi la valeur pointée. On pourra aussi sauvegarder l'adresse (dans la réserve CLIP) et d'une simple pression de la souris ouvrir une fenêtre de code ou de mémoire à cette adresse.

L'usager peut lui-même choisir la portion de l'écran qui sera utilisée par ABZmon. On pourra même dédier un second moniteur pour l'affichage du "débogueur". D'une simple combinaison de clés, on peut faire conserver l'écran affiché par ABZmon.

Vous avez le contrôle sur plus de cinquante paramètres internes, qui déterminent le fonctionnement de ABZmon, incluant tous les vecteurs d'exception (applications et systèmes). Vous pouvez sauvegarder la disposition de vos fenêtres et menus, pour retrouver le même environnement ABZmon ultérieurement. Vous pouvez aussi sauvegarder des adresses ou valeurs utiles dans la réserve CLIP.

Pour entrer dans ABZmon

Il y a plusieurs façons d'entrer dans le "débogueur":

- En appuyant sur le commutateur d'interruption (programmer switch).

- Certains utilitaires, comme "Programmer's Key" de Paul Mercer, utilisent une combinaison de clés spéciale (la touche de mise en marche, "Power Key", par exemple) pour forcer l'entrée dans le "débogueur".

- En incluant, dans le code à compiler, l'instruction **Debugger.** Le "débogueur" est appelé immédiatement après l'exécution de cette instruction.

- En incluant l'instruction **DebugStr** dans le code. Le pointeur d'un message (format Pascal) doit être passé dans la pile. Ce message est affiché dans la fenêtre des messages de ABZmon.

- En incluant l'instruction **DebugNum** dans le code. Voir plus loin pour la description de cette nouvelle instruction.

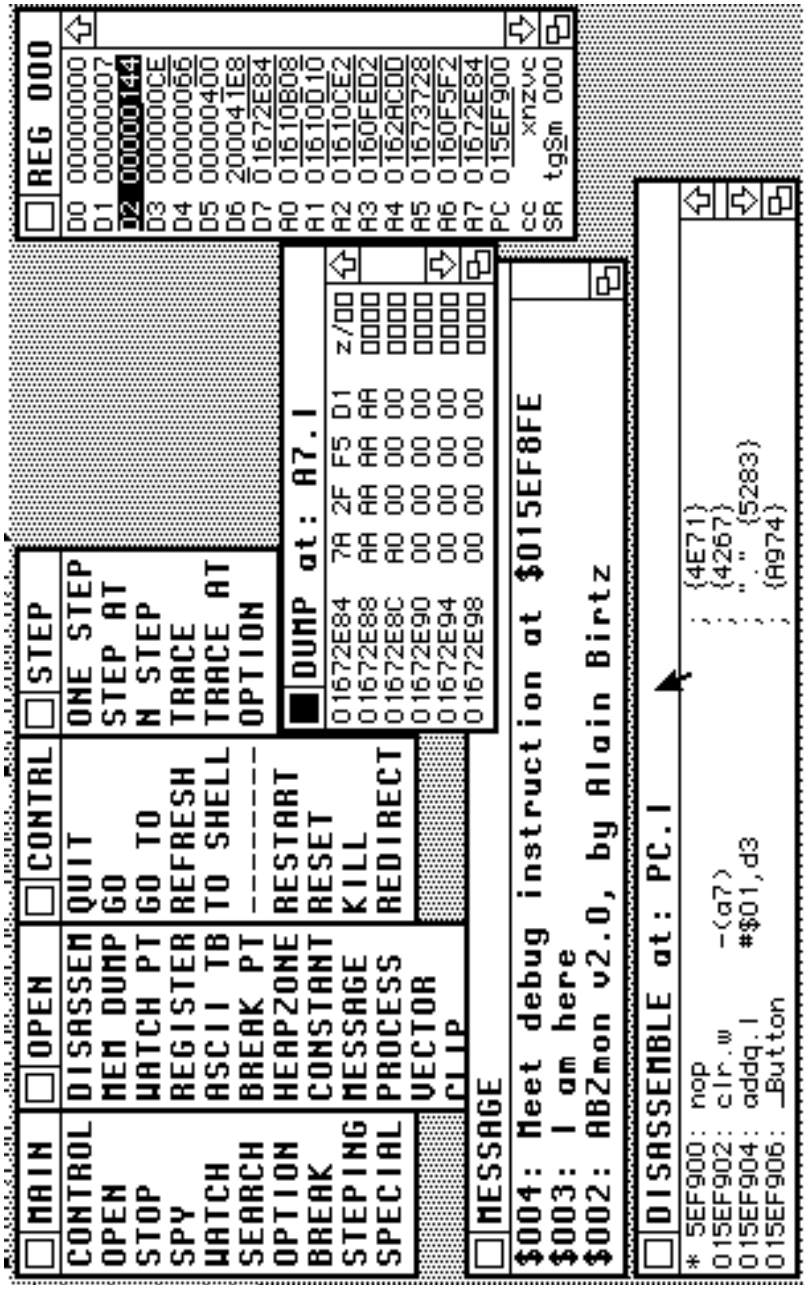
- Par la rencontre d'un point d'arrêt dans le code.

- Par le suivi pas à pas.

- Par une des formes d'arrêt proposée dans le menu "Stop" telle l'interception de certaines instructions ou certain changement dans une zone mémoire pré-déterminée.

- Lors d'une exception générée par le microprocesseur lui-même, durant une faute sur le "bus" ou une erreur d'adresse, par exemple.

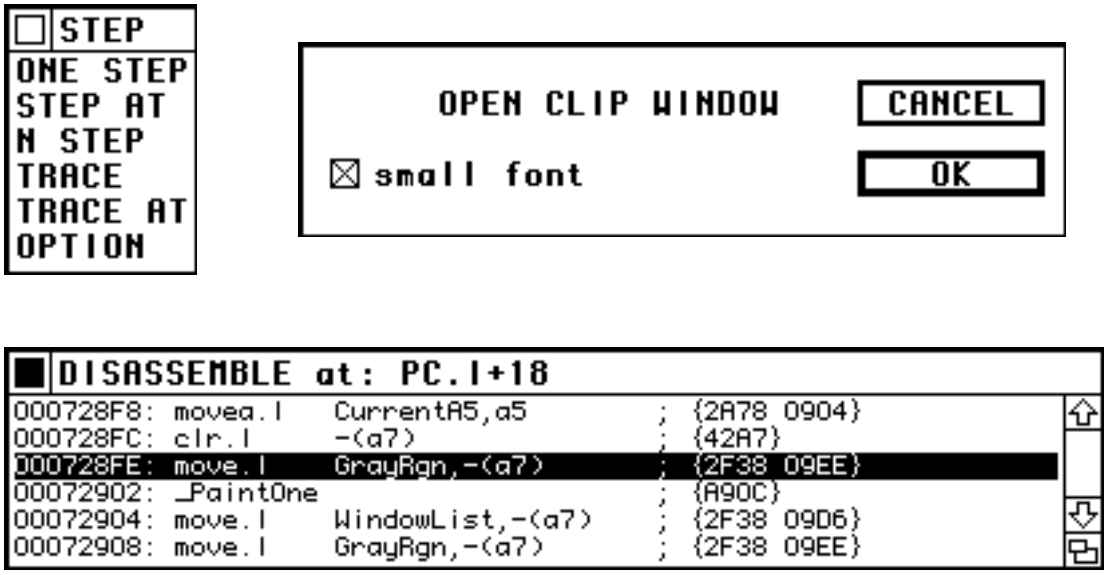
L'écran ABZmon par défaut



L'interface graphique de ABZmon

L'interface graphique ressemble à l'interface graphique du Mac, mais ne lui est pas identique. En fait ABZmon n'utilise aucune procédure de QuickDraw. ABZmon utilise ses propres routines d'écriture à l'écran, n'interférant ainsi d'aucune manière le fonctionnement normal du système d'opération. L'interface se compose de menus, de fenêtres et de boîtes de dialogue, gérés par la souris à la façon Macintosh.

Bien sûr, l'interface ABZmon n'est pas aussi raffiné que celle développée par Apple (elle tient sur 30 kilo-octets seulement), mais elle assurera quand même le confort auquel l'usager du Mac est habitué. Parmi les différences, on notera l'absence du couper-copier usuel. Le "débogueur" n'autorise que le transfert de nombres hexadécimaux (ce qui représente quant même la majorité des données utilisées, comme adresses mémoires). On notera aussi des différences dans la barre des ascenseurs. Elle ne contient que les flèches habituelles pour le défilement vers le haut ou vers le bas. Cette lacune est toutefois compensée par une vitesse foudroyante de défilement...



Exemple de menu, fenêtre et boîte de dialogue dans ABZmon

Notez que dans les boîtes de dialogue le bouton au contour plus épais est équivalent à la touche **Enter** ou **Return**.

La fenêtre active

Cette fenêtre affiche une case de fermeture noire. La première fois que vous cliquez dans une fenêtre (menu) elle devient active. Vous pouvez bouger une fenêtre (active ou non) de la manière habituelle en plaçant le curseur dans la barre titre. On ferme la fenêtre en cliquant dans la case de fermeture. Si la fenêtre possède des ascenseurs vous pouvez faire défiler le contenu en cliquant sur les icônes flèche vers le haut ou vers le bas, ou en utilisant les touches clavier flèche vers le haut ou vers le bas. Si la fenêtre possède une case d'expansion vous pouvez agrandir ou réduire la taille de la fenêtre de la façon habituelle.

Quelquefois, la fenêtre est vide. Ceci signifie que le contenu ne peut pas être affiché, probablement parce que ABZmon détecte une erreur bus ou une erreur adresse. Par exemple le "débogueur" ne peut pas afficher le contenu d'une fenêtre DUMP à l'adresse $AAAAAAAA parce que cette adresse ne correspond physiquement à aucune banque mémoire.

Dans l'exemple ci-dessus, la fenêtre DISASSEMBLE est active.

L'adresse par défaut

Vous pouvez sélectionner une ligne (elle s'inverse en noir) dans une fenêtre active en cliquant sur cette ligne, au début de la ligne. Dans une fenêtre de registres, par exemple, vous devez cliquer sur le nom du registre (la région hexadécimale est réservée pour changer la valeur du registre). De même dans une fenêtre DISASSEMBLE (ou une fenêtre DUMP), vous devez cliquer l'adresse au début de la ligne, la région des opérandes étant réservée pour afficher les valeurs des opérandes (les autres régions de la fenêtre DUMP étant réservées pour changer la mémoire).

La ligne sélectionnée détermine l'adresse par défaut. Cette adresse par défaut est utilisée dans plusieurs boîtes de dialogue demandant une adresse ou une valeur dans un champ d'édition.

Par exemple, pour placer un point d'arrêt vous sélectionnez une adresse dans une fenêtre DISASSEMBLE et cliquez l'item SET ONE du menu BREAK PNT. Pour exécuté le code à partir de l'adresse contenue dans le registre A3 vous sélectionnez la ligne A3 dans une fenêtre de registres et cliquez l'item GO TO du menu CONTRL.

Dans la fenêtre DISASSEMBLE illustrée ci-dessus, la troisième ligne est sélectionnée, l'adresse par défaut est donc $728FE.

Le menu MAIN

C'est le menu qui donne accès aux autres menus, et par conséquent, aux fenêtres et aux boîtes de dialogue. Comme tous les menus dans ABZmon, il peut être déplacé (on place le curseur dans le titre, en tenant le bouton de la souris enfoncé, et on déplace le curseur) et fermé (le petit carré dans le coin gauche supérieur). Si ce menu vient à être fermé, on peut le retrouver en fermant tous les autres et appuyant sur une touche du clavier ou cliquant de la souris...

Les items de ce menu ouvrent les menus suivants:

CONTROL: le menu de contrôle qui permet de retourner à l'application, de terminer l'application, de reprendre à une adresse donnée, etc.

OPEN: ce menu demande l'ouverture de fenêtre de mémoire, de code, des messages, de sélection de fichier, de texte...

STOP: les items de ce menu ordonnent l'arrêt pour diverses conditions sur les registres (par exemple si le pointeur de pile est incrémenté) ou si le programme exécute certaines instructions (par exemple LINK).

SPY: vérifie la mémoire à chaque instruction. Si la mémoire a changé, le programme est interrompu et ABZmon entre en jeu. Les items du menu déterminent le type de vérification à effectuer: sur un octet, un mot, un mot long, sur un segment de mémoire (par comparaison ou par somme).

WATCH: pour ajouter un nouvel élément d'observation pour un pointeur à indirection simple, double, triple...

SEARCH: pour trouver une séquence de lettres ou séquence de nombres hexadécimaux en mémoire.

OPTION: chaque item détermine l'apparence du contenu des fenêtres de code. Par exemple, le type d'étiquettes employées (locales ou celles du compilateur) ou le type de processeur (à 16 ou 32 bits)... Deux menus d'options sont ouverts simultanément.

BREAK: pour placer ou enlever des points d'arrêt.

STEPPING: gère le suivi pas à pas. Marche simple, à "n" pas, mode trace, etc

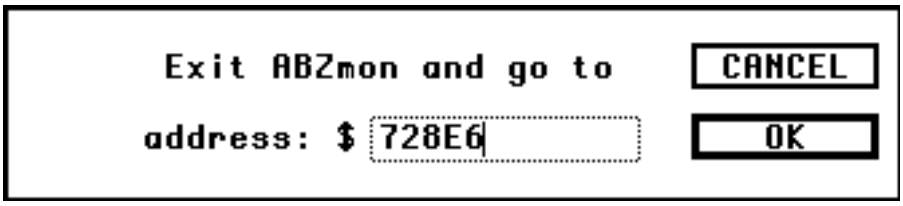
SPECIAL: différents items tels la sauvergarde de la disposition des menus et fenêtres, la sauvegarde de la réserve CLIP et de l'apparence des fenêtres de texte.

Le menu CONTRL

C'est le menu qui permet le retour à l'application courante ou de terminer l'application courrante.

QUIT et GO: pour quitter ABZmon et reprendre l'exécution du programme. Si l'arrêt du programme est demandé par une des trois instructions **Debug**, l'exécution reprend après l'instruction **Debug**. Si l'arrêt est demandé par un point d'arrêt, l'exécution reprend à l'instruction sous le point d'arrêt. Si l'entrée dans le "débogueur" s'est fait via un vecteur d'exception, l'exécution reprend à l'instruction fautive ou à l'instruction suivante selon le type d'exception rencontré. Raccourci clavier: **g**, **G** ou **commande G**.

GO TO: pour quitter ABZmon et reprendre l'exécution du programme à l'adresse donnée par la boîte de dialogue ci-dessous. L'adresse $728E6 est l'adresse par défaut (une ligne de code, à cette adresse, a été sélectionnée). Quant aucune ligne n'est sélectionnée, le champ d'édition est vide...



REFRESH: pour quitter ABZmon et forcer le système à reconstruire le bureau. Très utile pour la mise au point des programmes utilisant les routines QuickDraw.

RESTART: pour redémarrer l'ordinateur, en utilisant les fonctions du système.

RESET: pour redémarrer l'ordinateur, en utilisant l'instruction Reset du microprocesseur. Ré-initilise tous les périphériques.

KILL: pour quitter ABZmon et rendre le "débogueur" totalement inactif. Les vecteurs d'exceptions utilisés par ABZmon sont restitués et il n'y a plus aucun moyen de rapeler le "débogueur". A utiliser avec la plus grande prudence.

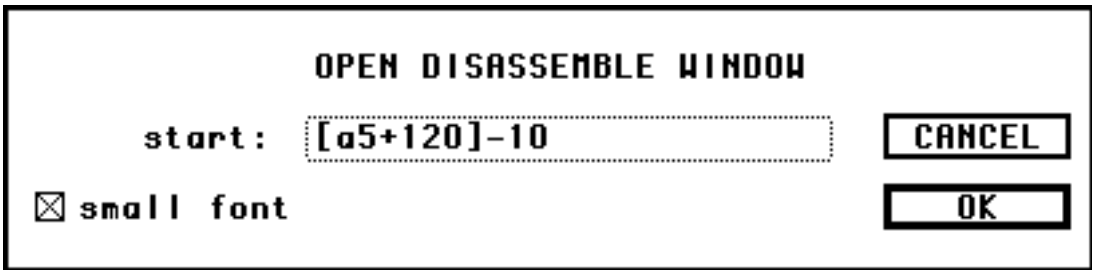
REDIRECT: pour quitter ABZmon et continuer en utilisant la routine d'exception du système. Avant de changer les vecteurs d'exceptions, ABZmon sauvegarde l'adresse des routines d'exceptions du système. Cet item permet de continuer à travers la routine d'exception du système, comme si le vecteur n'avait jamais été changé. A utiliser avec la plus grande prudence. Il est possible qu'un programme, chargé après ABZmon, change lui aussi certains vecteurs d'exceptions.

L'item ne contenant que des tirets est utilisé uniquement pour séparer les autres items: les quatres derniers items sont dangereux!

Le menu OPEN

C'est le menu qui demande l'ouverture de différentes fenêtres.

DISASSEM: demande l'ouverture d'une fenêtre de code désassemblé. On entre l'adresse de départ du code à désassembler dans la zone d'édition de la boîte de dialogue ci-dessous. Cette adresse peut être donnée de façon indirecte à l'aide de registres et déplacements relatifs (voir **Les adresses JB** plus loin). La case **small font** doit être cochée si on veut que le texte de la fenêtre soit affichée avec de petits caractères.



Si l'adresse n'est pas valide, le curseur de la souris se change en une flèche vide et le curseur de texte se positionne près du premier caractère invalide. On clique sur le bouton **OK** (alternativement, les touches **Return** ou **Enter**) pour confirmer l'ouverture de la fenêtre et sur le bouton **Cancel** pour annuler.

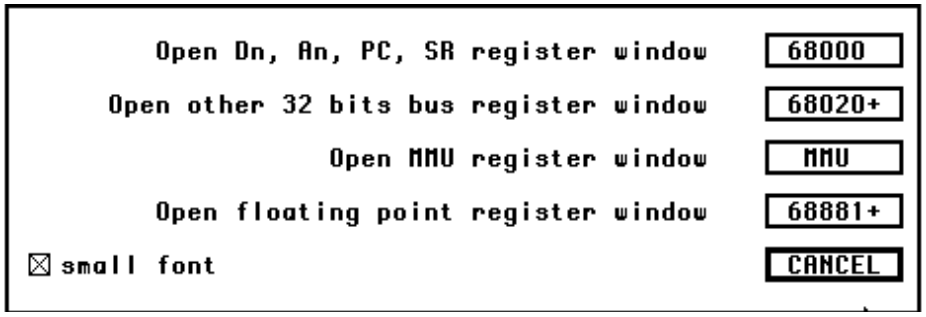
La combinaison **Command-D** est un raccourci clavier équivalent à cet item.

MEM DUMP: demande l'ouverture d'une fenêtre de visualisation de mémoire en valeur hexadécimale et code ascII. Une boîte de dialogue, semblable à celle utilisée pour les fenêtres de code, demande l'adresse de départ.

La combinaison **Command-M** est un raccourci clavier équivalent à cet item.

WATCH PT: demande l'ouverture d'une fenêtre d'observation pour pointeur à indirection simple, double ou triple. La boîte de dialogue ci-dessous est présentée pour confirmer l'ouverture et déterminer la taille de la fonte utilisée.

REGISTER: permet l'ouverture d'une des quatre fenêtres de registres. Pour le microprocesseur 68000, seule la fenêtre des registres Dn, An, PC et SR est disponible. Pour les autres types de microprocesseurs la fenêtre des autres registres 32 bits peut être ouverte, la fenêtre des registres MMU n'est ouverte que pour les machines possédant une unité de gestion de mémoires (68030-40 et 68851) tandis que la fenêtre des registres en virgule flottante n'est disponible que si une unité FPU est présente. La boîte de dialogue ci-dessous permet de choisir la fenêtre désirée.

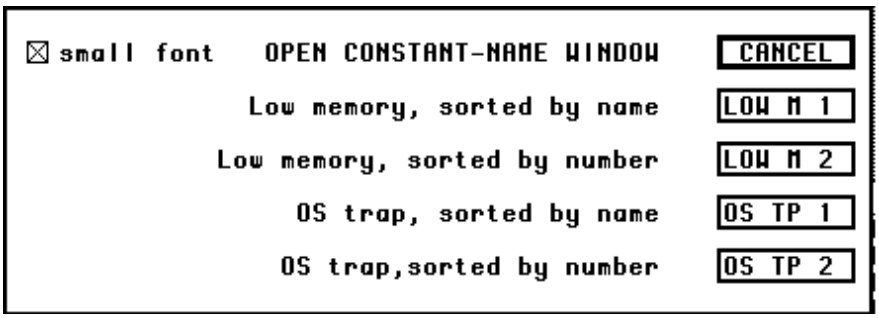


ASCII TB: demande l'ouverture de la table des codes ascII.

BREAK PT: demande l'ouverture d'une fenêtre décrivant les points d'arrêt.

HEAPZONE: demande l'ouverture d'une fenêtre décrivant les différentes zones mémoires utilisées par le système et les applications

CONSTANT: permet l'ouverture d'une des quatre fenêtres de constantes. Les constantes décrites sont celles des appels trappes du Toolbox et du OS, et les variables globales du système d'opération. Le classement des constantes est fait par la valeur des constantes ou par leur nom. La boîte de dialogue ci-dessous permet de choisir la fenêtre désirée.



MESSAGE: demande l'ouverture de la fenêtre des messages. Une seule fenêtre de ce type peut être ouverte. Si la fenêtre est déjà présente, ABZmon la place au premier plan en la mettant **active**.

PROCESS: demande l'ouverture d'une fenêtre décrivant toutes les applications en marche, incluant celle fonctionnant en tache de fond (sous le système 7).

VECTOR: demande l'ouverture d'une fenêtre donnant différentes informations sur les 58 vecteurs d'exception du microprocesseur.

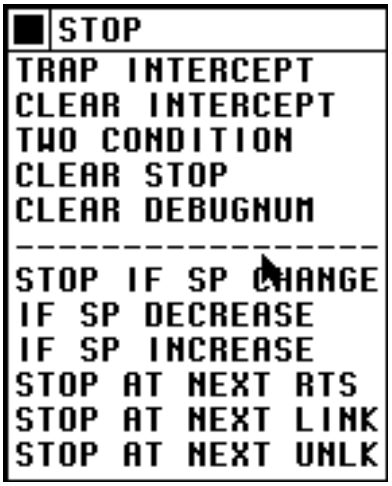
CLIP: demande l'ouverture d'une fenêtre qui donne le nom et la valeur de chacune des adresses de la réserve CLIP.

FILE SEL: demande l'ouverture d'une fenêtre qui permet de choisir un fichier (c'est l'analogue, dans l'interface du Mac, de la boîte de sélection des fichiers pour l'ouverture d'un document).

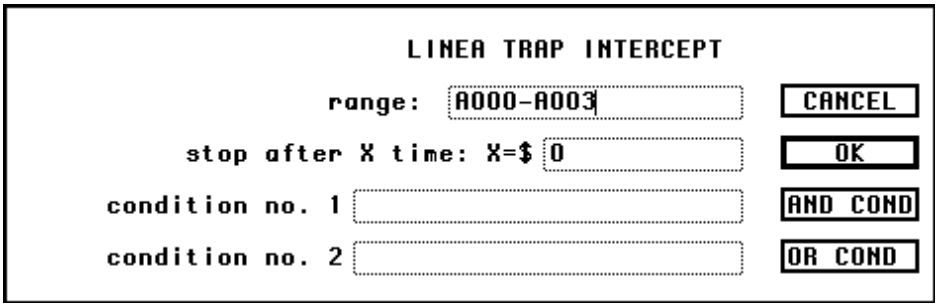
TEXTVIEW: demande l'ouverture d'une fenêtre qui affiche le texte d'un fichier choisi dans la fenêtre précédente.

Le menu STOP

C'est le menu qui ordonne l'arrêt pour diverses conditions.



TRAP INTERCEPT: Le système d'opération du Mac utilise le vecteur d'exception numéro 10 (A en hexadécimal) pour l'appel de ses fonctions de base. Cette exception est connue sous le nom de LineA. ABZmon impose un arrêt lorsque l'usager veut savoir si un programme utilise une fonction du OS passant par le LineA. La boîte de dialogue ci-dessous permet de définir le moment de l'arrêt.

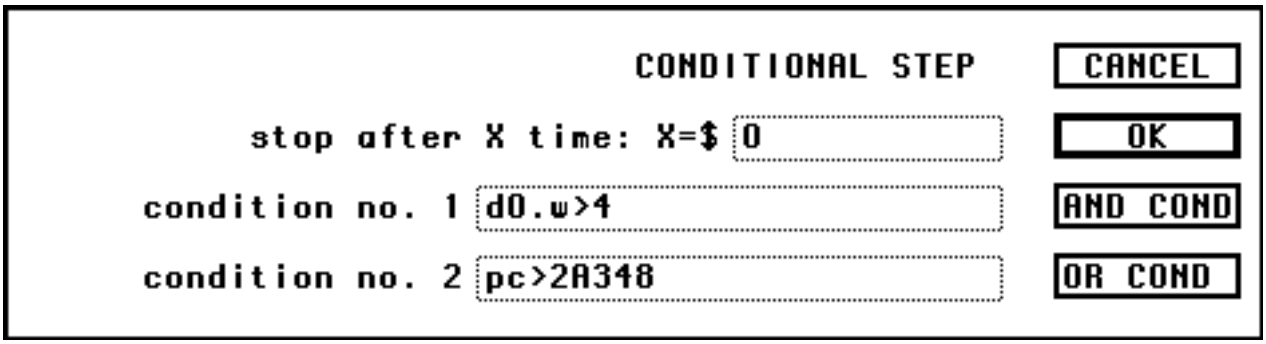


Dans le champ d'édition **range** on entre le numéro hexadécimal de la fonction LineA, par exemple A981 pour la fonction DrawDialog, ou l'intervalle de plusieurs fonctions LineA, par exemple A000-A003 pour les fonctions Open, Close, Read et Write. Dans le champ d'édition **stop after X time:** on entre un nombre hexadécimal qui détermine le nombre de rencontres **X** de cette ou ces fonctions du LineA avant l'arrêt. Les deux derniers champs d'édition imposent des conditions qui, si elles sont respectées, vont incrémenter le compteur **X** précédent (voir **Les conditions JB** plus loin). Ces champ peuvent être utilisés ou non. S'ils ne sont pas utilisés, aucune condition n'est imposée, et vous devez cliquer le bouton **OK**. Si un seul champ est utilisé, la condition sera imposée pour incrémenter le compteur **X**, et vous devez aussi cliquer le bouton **OK.** Si les deux champs sont utilisés, les deux conditions seront imposées

pour incrémenter le compteur **X**. Vous devez cliquer le bouton **AND COND** si les deux conditions doivent être vérifiées simultanément pour incrémenter le compteur **X**. Vous devez cliquer le bouton **OR COND** si seulement une des deux conditions doit être vérifiée pour incrémenter le compteur **X**.

CLEAR INTERCEPT: donne l'ordre à ABZmon de ne plus surveiller les appels LineA et de ne faire aucun arrêt s'y reportant.

TWO CONDITION: on demande au "débogueur" de surveiller le microprocésseur jusqu'à ce que la ou les conditions soit réalisées. Un arrêt est alors exigé. La boîte de dialogue ci-dessous permet de définir le moment de l'arrêt.



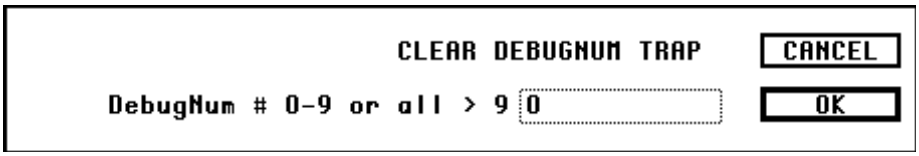
Dans le champ d'édition **stop after X time:** on entre un nombre hexadécimal qui détermine le nombre de fois **X** où ces conditions devront être realisées avant l'arrêt. Les deux derniers champs d'édition imposent des conditions qui, si elles sont respectées, vont incrémenter le compteur **X** précédent (voir **Les conditions JB** plus loin). Ces champs peuvent être utilisés ou non. S'ils ne sont pas utilisés, aucune condition n'est imposée (le compteur est incrémenté à chaque instruction), et vous devez cliquer le bouton **OK**. Si un seul champ est utilisé, la condition sera imposée pour incrémenter le compteur **X**, et vous devez aussi cliquer le bouton **OK.** Si les deux champs sont utilisés, les deux conditions seront imposées pour incrémenter le compteur **X**. Vous devez cliquer le bouton **AND COND** si les deux conditions doivent être vérifiées simultanément pour incrémenter le compteur **X**. Vous devez cliquer le bouton **OR COND** si seulement une des deux conditions doit être vérifiée pour incrémenter le compteur **X**.

Remarque: Puisque que ABZmon surveille chaque instruction une à une, la baisse de vitesse d'exécution du programme est très importante. Tout fonctionnera au ralenti. De plus, la rencontre de certaines instructions, comme les instructions qui changent le registre **SR** ou certaines fonctions LineA, va annuler le travail de surveillance de ABZmon. Cette surveillance s'effectue via le mode **trace** du microprocesseur. Ce mode est imposé par le bit 15 du registre **SR**. Si le registre **SR** change, ce bit 15 pourrait être mis à zéro et ABZmon n'aurait plus alors aucun moyen de continuer la surveillance.

L'exécution du programme se poursuit à la vitesse normale et aucun arrêt ne peut être imposé même si les conditions sont réalisées pour un tel arrêt.

CLEAR STOP: annule toute commande d'arrêt de ce menu.

CLEAR DEBUGNUM: annule la commande d'arrêt proposée par la rencontre d'une instruction **DebugNum** dans le code d'un programme en exécution. On entre le paramètre, de 0 à 9, de l'instruction à annuler pour annuler une seule instruction à la fois, ou une valeur supérieure à 9 pour annuler toute les instructions **DebugNum**, dans la boîte de dialogue présentée à cet effet.



STOP IF SP CHANGE: ABZmon surveille le pointeur de la pile **SP** à chaque instruction. S'il y a changement de valeur de **SP**, un arrêt est effectué. La remarque ci-dessus, concernant la vitesse d'exécution réduite, s'applique ici également.

IF SP DECREASE: ABZmon surveille le pointeur de la pile **SP** à chaque instruction. Si la valeur de **SP** diminue, un arrêt est effectué. Cette fonctionnalité est particulièrement utile pour atteindre la prochaine sous-routine appelée par les instructions **BSR**, **JSR** ou **LINK**. La remarque ci-dessus, concernant la vitesse d'exécution réduite, s'applique ici également.

IF SP INCREASE: ABZmon surveille le pointeur de la pile **SP** à chaque instruction. Si la valeur de **SP** augmente, un arrêt est effectué. Cette fonctionnalité est particulierement utile pour sortir d'une sous-routine amorcée par les instructions **BSR** et **JSR**. La remarque ci-dessus, concernant la vitesse d'exécution réduite, s'applique ici également.

STOP AT NEXT RTS:
STOP AT NEXT LINK:
STOP AT NEXT UNLK:
le "débogueur" surveille l'exécution des instructions du programme et force un arrêt si l'une de ces instructions RTS, LINK ou UNLK est rencontrée. La remarque ci-dessus, concernant la vitesse d'exécution réduite, s'applique ici également.

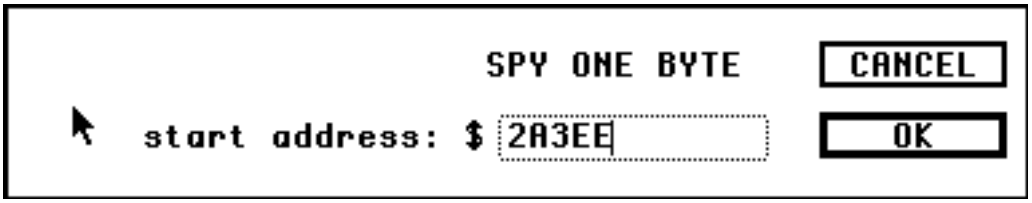
Note: les six derniers items du menu STOP peuvent être obtenus, en imposant des conditions sur les registres, par l'item TWO CONDITION. C'est d'ailleurs ce

que fait ABZmon pour ces six derniers items.

Le menu SPY

C'est le menu qui permet de donner l'ordre à ABZmon de rapporter les changements d'une certaine zone mémoire.

ONE BYTE: on demande au "débogueur" de surveiller un octet à chaque fois qu'une instruction est exécutée et d'imposer un arrêt s'il y a changement de la valeur de l'octet. La boîte de dialogue ci-dessous détermine l'adresse de l'octet.



La remarque ci-dessus, concernant la vitesse d'exécution réduite, s'applique ici également.

ONE WORD: comme pour l'item précédent, mais c'est la valeur d'un mot (2 octets) qui est surveillée.

ONE LONG: comme pour l'item précédent, mais c'est la valeur d'un mot long (4 octets) qui est surveillée.

ARRAY: on demande au "débogueur" de surveiller un segment de mémoire à chaque fois qu'une instruction est exécutée et d'imposer un arrêt s'il y a changement de la valeur d'au moins un des octets de ce segment. La boîte de dialogue ci-dessous détermine l'adresse du premier et du dernier octet du segment.

La longueur maximale du segment est fixé au moment du démarrage, de telle sorte que l'image du segment est conservé à l'intérieur même de l'espace mémoire occupé par ABZmon. Cette longueur maximal peut être changé dans la reaource S_UP (voir **Variables internes** plus loin).

La remarque ci-dessus, concernant la vitesse d'exécution réduite, s'applique ici également.

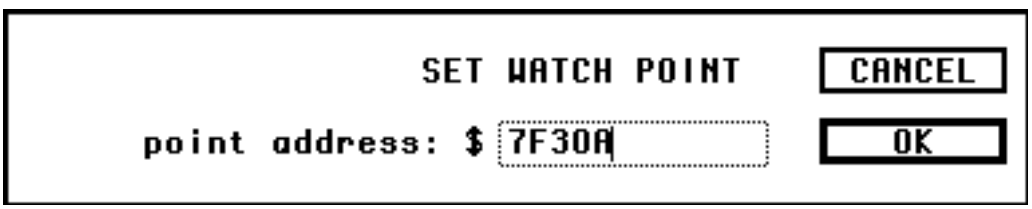
CHECKSUM: comme pour l'item précédent, mais c'est seulement lorsque la somme des octets du segment est changé que l'arrêt intervient. Bien que moins précise que la méthode précédente, cette méthode a l'avantage de ne requérir aucun espace mémoire et peut donc s'appliquer à un segment de n'importe quelle longueur.

CLEAR: annule la surveillance de la zone mémoire.

Le menu WATCH

C'est le menu qui permet d' ajouter un nouvel élément d'observation pour un pointeur à indirection simple, double, triple. Ces éléments sont visibles dans une fenêtre **WATCH PT**.

POINT: ajoute un nouvel élément d'observation pour un point. La boîte de dialogue ci-dessous détermine l'adresse du point.



HANDLE: ajoute un nouvel élément d'observation pour un pointeur à indirection double. Par exemple si le mot long à l'adresse $36662 contient la valeur $24ED4 et que le mot long à $24ED4 est $FFFFFFFF alors la fenêtre **WATCH PT** affichera: **36662 => 24ED4 => FFFFFFFF**

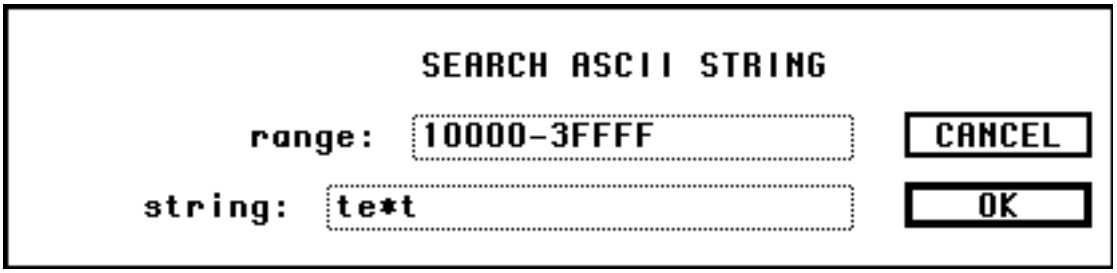
DOUBLE: ajoute un nouvel élément d'observation pour un pointeur à indirection triple.

REMOVE: enlève tous les élements d'observation.

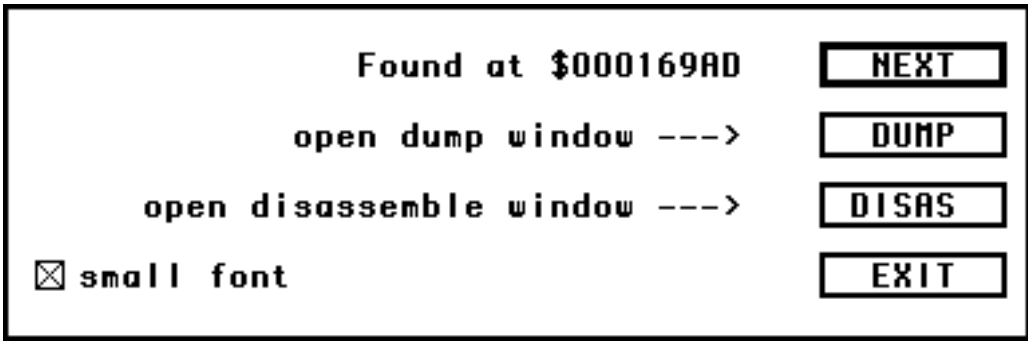
Le menu SEARCH

C'est le menu qui permet de trouver une séquence de lettres ou séquence de nombres hexadécimaux en mémoire.

ASCII STR: pour trouver une séquence de lettres en mémoire. Dans la boîte de dialogue ci-dessous, le premier champ d'édition **range** contient l'adresse du premier et du dernier octet du segment mémoire où la recherche doit se faire. Le second champ **string** contient la séquence de lettres recherchée. Le symbole * est le caractère de remplacement (joker) qui tient pour n'importe quel caractère. ABZmon recherche donc un mot comme **test** ou une partie de mot comme dans **tente**.



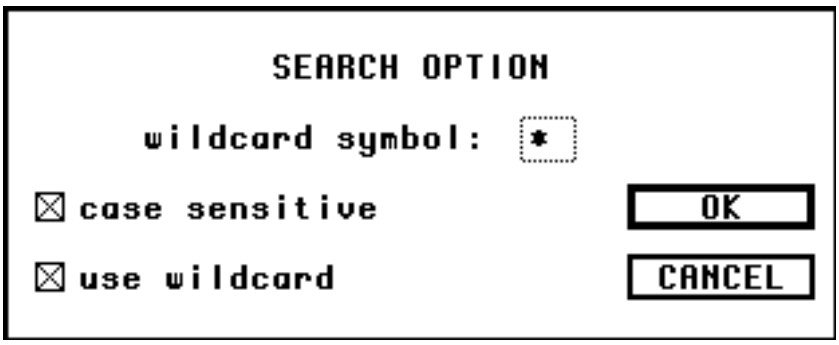
Dans cet exemple ABZmon a trouvé le mot **text** à l'adresse $169AD. Il affiche alors la boîte de dialogue suivante:



Le bouton **NEXT** demande la poursuite de la recherche. Le bouton **DUMP** demande d'ouvrir une fenêtre de visualisation de la mémoire à cette adresse, tandis que bouton **DISAS** demande d'ouvrir une fenêtre de code. Le bouton **EXIT** demande l'annulation de la recherche.

HEXA STR: pour trouver une séquence de nombres hexadécimaux en mémoire. Dans la boîte de dialogue ci-dessous, le premier champ d'édition **range** contient l'adresse du premier et du dernier octet du segment mémoire où la recherche doit se faire. Le second champ **string** contient la séquence de nombres hexadécimaux recherchée. Elle correspond à la séquence ascII de l'item précédent. Remarquez que le symbole *, qui est le caractère de remplacement (joker), peut aussi être employé dans une séquence de nombres hexadécimaux.

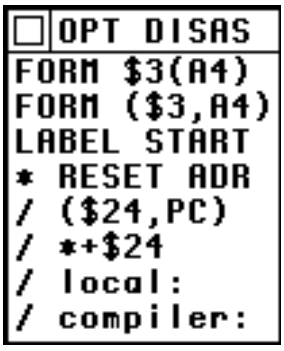
OPTION: détermine les options de recherche. Dans la boîte de dialogue ci-dessous, le caractère de remplacement (joker) est choisi dans le champ d'édition **wildcard symbol**. La case **case sensitive**, si elle est cochée, est sensible à la différence entre majuscule et minuscule (dans ce cas, la recherche n'aurait pas trouvé **Test**, par exemple). La case **use wildcard**, si elle est cochée, demande de faire la recherche en utilisant le caractère de remplacement.

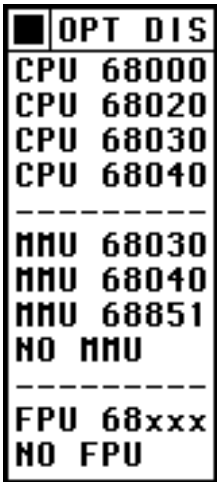


FIND NEXT: demande la poursuite de la recherche.

Les menus OPT DISAS et OPT DIS

Ce sont les menus qui déterminent l'apparence du contenu des fenêtres de code. Plusieurs autres caractéristiques importantes des fenêtres de code peuvent être établies avec la ressource S_UP (voir **Variables internes** plus loin). Le second menu détermine le type de microprocesseur central, d'unité de gestion de mémoire et de co-processeur mathématique utilisé pour le désassemblage du code.

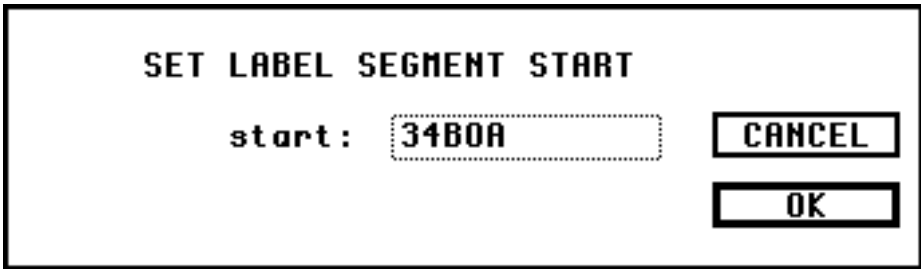




FORM $3(A4): Motorola, le fabricant des microprocesseurs de la famille des 68000 définit deux types d'écriture pour les opérandes pour l'assembleur: la forme $3(A4) avec déplacement à l'extérieur des parenthèses (généralement pour le microprocesseur 68000) et la forme ($3,a4) avec déplacement à l'intérieur des parenthèses (généralement pour les microprocesseurs 68020 et ultérieurs). Cet item demande la première forme.

FORM ($3,A4): demande la deuxième forme.

LABEL START: ABZmon peut désassembler le code en utilisant des étiquettes pour les branchements (**BSR, BRA,...**), pour les sauts (J**MP, JSR**) ou les reférences relatives au compteur de programme PC, afin de faciliter la lecture du code. ABZmon utilise un tampon spécial, le tampon des étiquettes, pour conserver l'adresse et le nom des étiquettes. Bien entendu, ABZmon ne conserve pas les étiquettes pour l'ensemble de tous les programmes en mémoire. Seul un certain segment mémoire est analysé pour en extraire les étiquettes. La boîte de dialogue ci-dessous donne l'adresse de départ de ce segment.



RESET ADR: il arrive souvent, durant la mise au point d'un programme, que seulement quelques octets du programme soit modifiés. Mais cette correction peut avoir déplacer le segment mémoire pour lequel ABZmon a extrait les étiquettes. On a besoin alors de recalculer l'adresse de ces étiquettes. C'est le rôle de cet item. Si vous voyez une étiquette avec le numéro 0, c'est que l'adresse des étiquettes a besoin d'être recalculée. L'équivalent clavier de cet item est la touche '*'. Recalculer l'adresse des étiquettes ne prend qu'une fraction de seconde (à moins d'avoir un tampom des étiquettes très grand), ne prenez pas de risque: utilisez la touche '*' dès que vous soupçonnez que le code a pu être modifié...

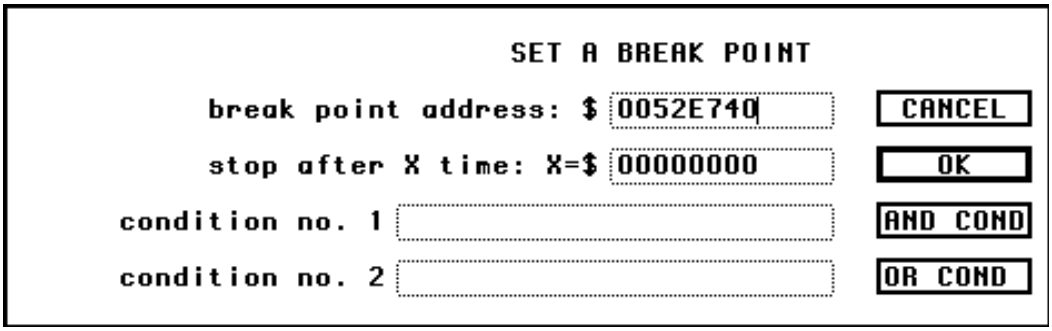
/ ($24,PC):
/ *+$24
/ local:
/ compiler

ces 4 items déterminent la forme de déplacement utilisé dans le désassemblage du code. Les deux premiers sont évidents. Le troisième utilise une étiquette locale de la forme **c_17** ou **d_17** selon que le déplacement relatif au compteur de programme **PC** est dans un segment de code ou dans un segment de données. Le dernier utilise les étiquettes sauvegardées par le compilateur. La touche '/' permet de passer successivement à chacune de ces quatre formes.

Le menu BREAK PNT

C'est le menu qui permet de placer des points d'arrêt dans un programme.

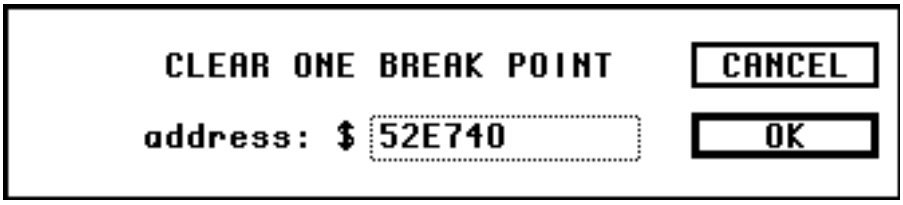
SET ONE: va placer un point d'arrêt dans le code. La boîte de dialogue ci-dessous donne les caractéristiques du point d'arrêt.



Dans le champ d'édition **break point address** on entre l'adresse où doit se faire l'arrêt. Notez que l'arrêt se fait avant l'exécution de l'instruction à cette adresse. Dans le champ d'édition **stop after X time:** on entre un nombre hexadécimal qui détermine le nombre de passages **X** à cette adresse avant l'arrêt. Les deux derniers champs d'édition imposent des conditions qui, si elles sont respectées, vont incrémenter le compteur **X** précédent (voir **Les conditions JB** plus loin). Ces champ peuvent être utilisés ou non. S'ils ne sont pas utilisés, aucune condition n'est imposée, et vous devez cliquer le bouton **OK**. Si un seul champ est utilisé, la condition sera imposée pour incrémenter le compteur **X**, et vous devez aussi cliquer le bouton **OK.** Si les deux champs sont utilisés, les deux conditions seront imposées pour incrémenter le compteur **X**. Vous devez cliquer le bouton **AND COND** si les deux conditions doivent être vérifiées simultanément pour incrémenter le compteur **X**. Vous devez cliquer le bouton **OR COND** si seulement une des deux conditions doit être vérifiée pour incrémenter le compteur **X**.

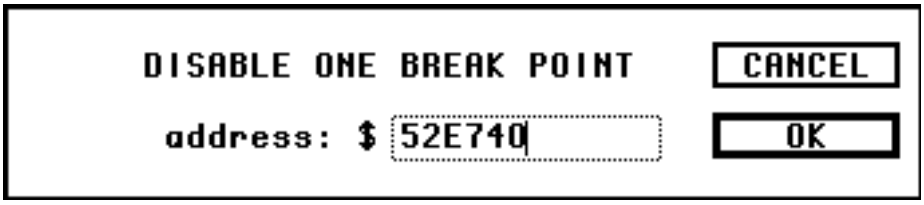
Dans une fenêtre de code, un point d'arrêt est signalé par la lettre **b** en début de ligne. De même l'adresse correspondant au registre PC est signalée par le caractère *****. Si un point d'arrêt est à la même adresse que le compteur de programme, c'est plutôt le tiret qui sera utilisé.

CLEAR ONE: pour enlever un point d'arrêt. La boîte de dialogue ci-dessous demande l'adresse du point d'arrêt.



CLEAR ALL: pour la suppression de tous les points d'arrêt.

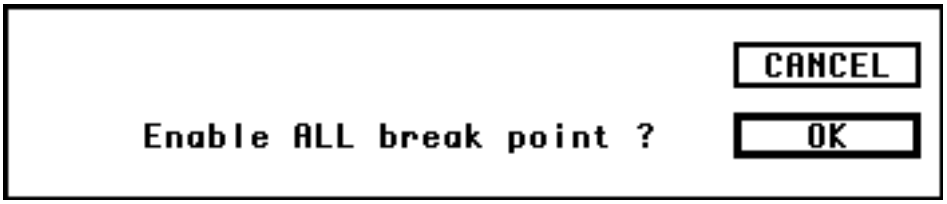
DISABLE ONE: pour rendre un point d'arrêt inactif. Si vous pensez devoir utiliser ce point d'arrêt ultérieurement... La boîte de dialogue ci-dessous demande l'adresse du point d'arrêt



ENABLE ONE: pour rendre un point d'arrêt à nouveau actif. Une boîte de dialogue semblable à celle de l'item précédent demande l'adresse du point d'arrêt.

DISABLE ALL: pour rendre tous les point d'arrêt inactifs.

ENABLE ALL: pour rendre tous les points d'arrêt à nouveau actifs.

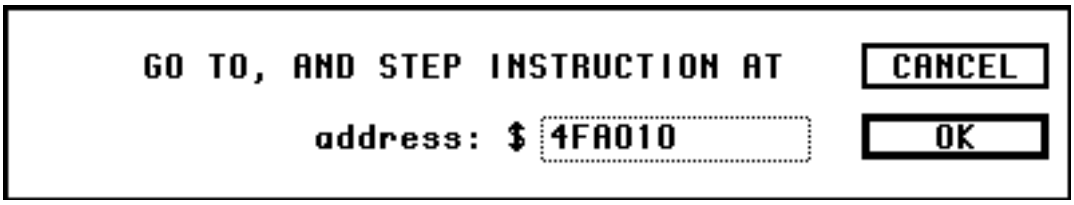


Le menu STEP

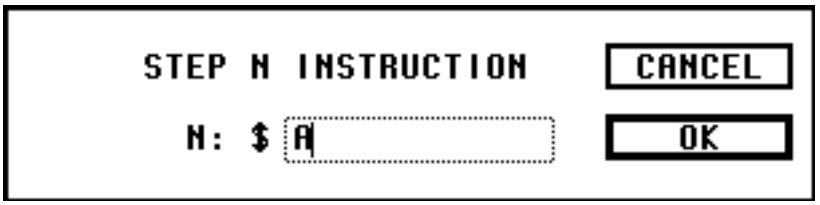
C'est le menu qui permet de faire le suivi pas à pas.

ONE STEP: exécute l'instruction pointée par le PC et revient au "débogueur". L'équivalent clavier est la touche **s**.

STEP AT: exécute une instruction à une adresse donnée. On entre cette adresse dans la boîte de dialogue ci-dessous.



N STEP: exécute "N" instructions à partir de l'instruction pointée par le PC et revient au "débogueur". La boîte de dialogue ci-dessous demande la valeur de "N"



Pour exécuter "N" instructions à une adresse autre que celle pointée par le PC vous devez auparavent changer la valeur du registre PC dans une fenêtre de registre.

La remarque (dans le menu STOP, item TWO CONDITION) concernant la vitesse d'exécution réduite, s'applique ici également.

TRACE: exécute l'instruction pointée par le PC et revient au "débogueur". Cet item ne diffère de ONE STEP que pour les instructions **BSR** et **JSR**. Avec TRACE la routine est exécutée au complet alors que pour ONE STEP c'est

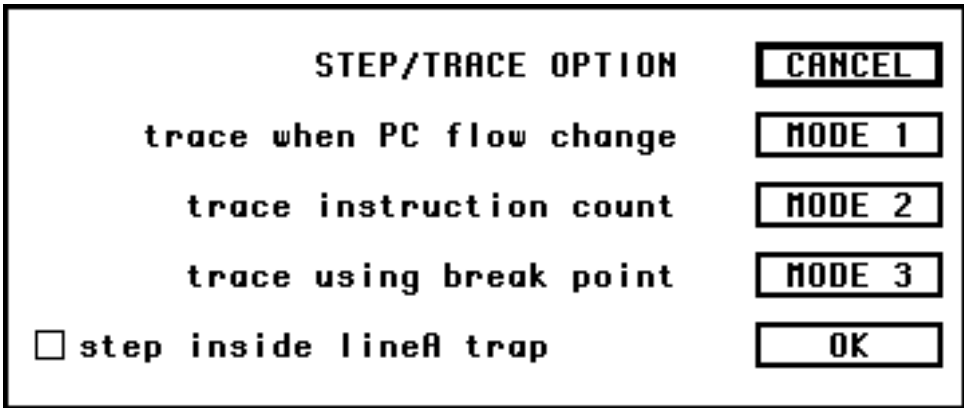
seulement la première instruction de la routine qui est exécutée. L'équivalent clavier est la touche **t**.

La remarque (dans le menu STOP, item TWO CONDITION) concernant la vitesse d'exécution réduite, s'applique pour le suivi des instructions **BSR** et **JSR**.

TRACE AT: exécute une instruction (comme pour TRACE) à une adresse donnée. On entre cette adresse dans la boîte de dialogue ci-dessous.

La remarque (dans le menu STOP, item TWO CONDITION) concernant la vitesse d'exécution réduite, s'applique pour le suivi des instructions **BSR** et **JSR**.

OPTION: détermine les caractéristiques du suivi pas à pas.



Le bouton **MODE 1** n'est valable que pour les microprocesseurs à 32 bits (68020 ou ultérieur). Le mode **trace** utilisé est celui qui exécute toutes les instructions qui ne changent pas la valeur séquentielle du compteur de programme en une seule étape. Les instructions telles JMP, BRA ou RTS changent la valeur séquentielle du PC. Par conséquent, ce mode fait le suivi d'une instruction de ce type, à l'autre, du même type seulement.

Le bouton **MODE 2** utilise le mode trace commun à tous les microprocesseurs de la famille 68000. Lorsque ABZmon rencontre une instruction **BSR**, **JSR ou lineA**, il compte le nombre d'instructions dans la routine et affiche ce nombre dans la fenêtre des messages.

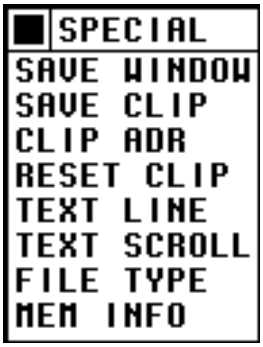
Le bouton **MODE 1** place un point d'arrêt après chaque instruction à être

exécutée. ABZmon apparaît donc après l'exécution et on a ainsi un suivi pas à pas. On utilise avantageusement ce mode pour les instructions **BSR** et **JSR** qui entrent dans des routines très longues. Le mode 2 est très lent pour de telles routines.

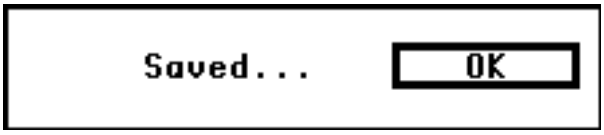
La case **step inside lineA trap**, si elle est cochée, force ABZmon à entrer dans les sous-routines d'une instruction LineA, par une commande ONE STEP. On peut donc ainsi examiner tout le code des appels du ToolBox et du OS (comme DrawDialog, FillRect ou DragWindow). Normalement le microprocesseur en mode "trace" exécute ces instructions en une seule étape, le niveau de priorité d'exception du LineA étant supérieur à celui du mode "trace".

Le menu SPECIAL

C'est le menu pour diverses opérations.



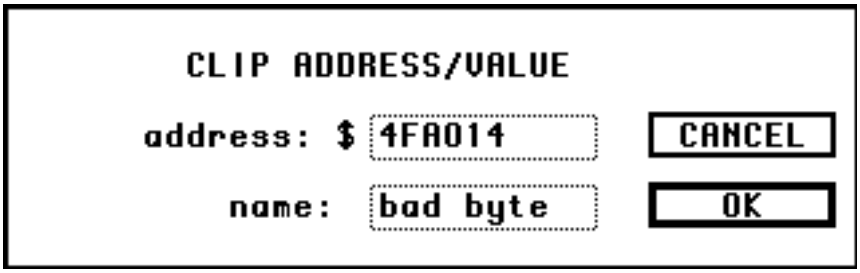
SAVE WINDOW: sauvegarde la disposition actuelle des fenêtres et menus. L'usager retrouvera le même environnement ABZmon au sessions suivantes. La boîte de dialoque suivante informe que la sauvergarde s'est effectuée correctement. L'équivalent clavier est la combinaison **commande s**.



Remarque la sauvegarde devrait se faire lorsque l'ordinateur est 'tranquille'. Il faut éviter les moments où les périphériques (disque dur, scanner, ...) sont trop sollicités.

SAVE CLIP: sauvegarde la réserve CLIP. La boîte de dialogue ci-dessus informe que la sauvergarde s'est effectuée correctement (seulement si la réserve CLIP n'est pas vide).

CLIP ADR: enregistre une nouvelle adresse dans la réserve CLIP.



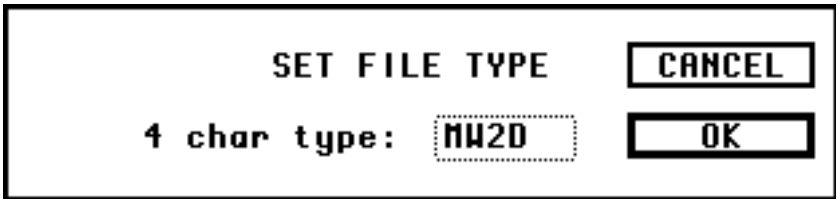
Dans le premier champ d'édition on entre l'adresse et dans le second le nom (jusqu'à 8 caractères). L'équivalent clavier est la combinaison **commande c**.

RESET CLIP: vide la réserve CLIP. ABZmon demande confirmation s'il y a au moins une adresse dans la réserve CLIP, sinon il ne fait rien.

TEXT LINE: dans une fenêtre de texte on peut voir chaque fin de ligne de deux façons différentes. La première affiche le texte en "brisant" la ligne (qui se poursuit plus bas). La seconde va simplement afficher la partie de la ligne qui peut être contenue dans la fenêtre (le reste est invisible mais peut être parcourru en utilisant les touches flèches la guauche et vers la droite). Cet item fait passer d'un mode à l'autre.

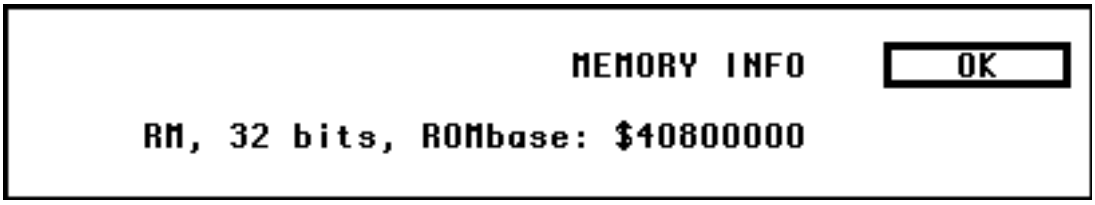
TEXT SCROLL: les icônes flèche vers le bas ou vers le haut font dérouler le texte d'une fenêtre de texte. Le texte peut être déroulé une ligne à la fois ou plus rapidement, une page à la fois. Cet item fait passer d'un mode à l'autre.

FILE TYPE: détermine le second type de fichier dans la fenêtre de sélection de fichiers. ABZmon reconnaît tous les fichiers de type TEXT. Pour reconnaître un autre type de fichier, il faut entrer son type dans la boîte de dialogue ci-dessous:



Dans cet exemple, on a entré le type MW2D, pour les documents de MacWriteII. Si vous n'entrez aucun type, ABZmon reconnaîtra tous les types de fichiers (Attention à ne pas laisser de caractères invisibles comme les espaces blancs).

MEM INFO: affiche une boîte de dialogue donnant les informations suivantes sur la mémoire de l'ordinateur: **RM** ou **VM** -> mémoire réelle ou mémoire virtuelle, **32 bits** ou **24 bits** pour l'adressage mémoire et finalement l'adresse du début des mémoires mortes (ROM).



Les fenêtres

Si une fenêtre ne peut être affichée, à la suite d'une erreur d'adresse (adresse inexistante) dans une fenêtre de code par exemple, ABZmon laisse cette fenêtre ouverte et signale cette anomalie dans la fenêtre des messages. Vous devez alors fermer cette fenêtre. Une telle fenêtre est toujours vide.

Généralement on peut sélectionner une ligne en cliquant sur le début de cette ligne. Elle apparaît alors en inversé et l'adresse de cette ligne devient l'adresse par défaut, c-a-d, l'adresse qui apparaît à l'ouverture de la plupart des boîtes de dialogue. On peut dé-sélectionner cette ligne en cliquant à nouveau sur la ligne sélectionnée.

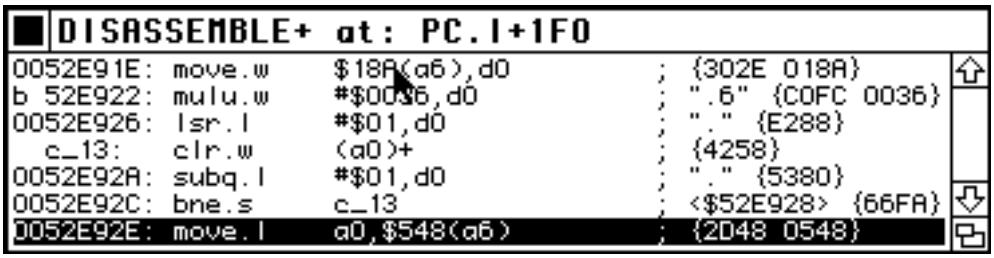
La fenêtre active est celle qui a l'icône de fermeture en noir. Une fenêtre devient la fenêtre active dès que l'on clique à l'intérieur. On peut aussi utiliser les touches "." et ",". La première permet une rotation de toutes les fenêtres et menus. La seconde alterne entre la dernière et l'avant dernière.

On peut déplacer une fenêtre, sans la rendre active, en plaçant le curseur dans le titre et en déplaçant la souris lorsque le bouton de la souris est enfoncé.

La plupart des fenêtres peuvent être agrandies ou diminuées en plaçant le curseur dans l'icône en forme de deux rectangles (en bas, à gauche) et en déplaçant la souris lorsque le bouton de la souris est enfoncé. La dimension de la fenêtre est calculée de façon à ne pas couper un caractère en son milieu. Certaines fenêtres, comme les fenêtres de registre 68000, ne peuvent être agrandies ou diminuées que dans une seule direction.

Les fenêtres de code

Ce sont les fenêtres qui permettent de voir le code désassemblé d'un segment de mémoire.



Le titre de cette fenêtre est dépendant de l'adresse JB de la première ligne de cette fenêtre. L'instruction **move.w**, à l'adresse mémoire $52E91E, correspond à la valeur du compteur de programme **PC** plus $1F0. Cette adresse JB, dans le titre, va évidemment changer si on clique dans les flèches de déroulement de la fenêtre, mais elle correspondra toujours à l'adresse de la première ligne. Si le déroulement de la fenêtre est trop rapide, vous pouvez utiliser les touches flèche vers le haut ou vers le bas, ou les touches **u** et **d** (**u**p et **d**own) pour les claviers dépourvus de ces flèches. Les touches flèche vers la droite et flèche vers la gauche, ou les touches **r** et **l** (**r**ight and **l**eft), sont utilisées pour forcer le désassemblage deux bytes avant ou après l'addresse de la premère ligne.

La deuxième ligne débute par la lettre **b** pour indiquer la présence d'un point d'arrêt à cette adresse. Si la ligne débute par le caractère *, l'adresse de cette ligne correspond à la valeur du compteur de programme PC. Si l'adresse de la ligne est à la fois la valeur du compteur de programme PC et l'adresse d'un point d'arrêt, la ligne débutera par un tiret.

La quatrième ligne débute par une étiquette (locale). Elle correspond à une boucle qui se termine à la sixième ligne. La dernière ligne a été sélectionnée, l'adresse par défaut (celle qui apparaît dans la plupart des boîtes de dialogue) est donc l'adresse de cette ligne, c-a-d $52E92E. On sélectionne une ligne en cliquant dans la zone contenant l'adresse ou l'étiquette en début de ligne.

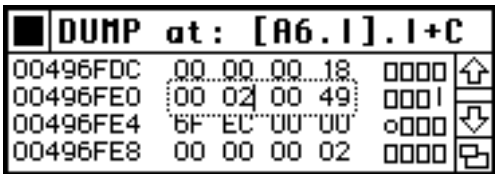
A la première ligne, le curseur pointe sur l'opérande $18A(a6). En pressant sur le bouton de la souris, on obtient alors les informations de cette opérande. La boîte de dialogue ci-dessous montre que l'adresse effective de cette opérande est $2506DA4 et que le mot pointé à cette adresse est $14. Notez que l'instruction **move.w** déplace un mot (deux octets) seulement et par conséquent la valeur pointée est aussi un mot. Si l'instruction **move** déplace un octet ou un mot long (4 octets) la valeur pointée est un octet ou un mot long.

Les valeurs entre accolades (après le point-virgule) sont les codes hexadécimaux de l'instruction désassemblée. Les symboles entre guillemets sont les représentations ascII des constantes pour les opérandes du mode "immédiat". A la deuxième ligne, par exemple, la valeur $36 correspond à la valeur ascII du chiffre 6. Si une valeur "immédiate" n'a pas de représentation ascII (comme à la ligne 5), le symbole entre guillemets est un point. Les valeurs entre "<" et ">" sont les adresses correspondant aux déplacements relatifs. A la sixième ligne, par exemple, le branchement se fait à l'adresse $52E928, soit l'adresse de l'étiquette c_13.

Plusieurs caractéristiques importantes des fenêtres de code peuvent être établies avec la ressource S_UP (voir **Variables internes** plus loin).

Les fenêtres de visualisation de la mémoire

Ce sont les fenêtres de visualisation de mémoire, en valeur hexadécimale et code ascII, d'un segment de mémoire.



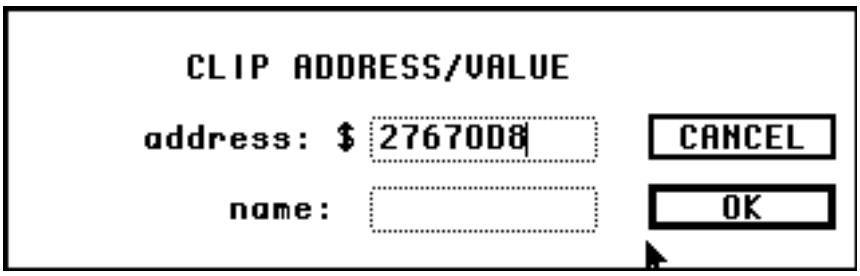
Le titre de cette fenêtre dépend de l'adresse JB du premier octet de la première ligne de cette fenêtre. Dans la fenêtre ci-dessus le registre **A6** a la valeur $495BE0. A cette adresse on retrouve la valeur $496FD0. L'adresse $496FDC = $495BE0 + C de la première ligne correspond à [A6.l].l+C. Cette adresse JB, dans le titre, va évidemment changer si on clique dans les flèches de déroulement de la fenêtre, mais elle correspondra toujours à l'adresse du premier octet de la première ligne. Si le déroulement de la fenêtre est trop rapide, vous pouvez utiliser les touches flèche vers le haut ou vers le bas, ou les touches **u** et **d** (**u**p et **d**own) pour les claviers dépourvus de ces flèches. Les touches flèche vers la droite et flèche vers la gauche, ou les touches **r** et **l** (**r**ight and **l**eft), sont utilisées pour un déphase d'un octet seulement.

La zone hexadécimale correspond à la zone ascII. Par exemple le dernier caractère de la deuxième ligne, le "i", correspond à la valeur hexadéxiamal $49. Si la valeur hexadécimale ne correspond à aucun caractère ascII affichable, c-a-d les valeurs inférieuress à $20 ou supérieures à $7E, le symbole correspondant dans la zone ascII est un rectangle vide.

On peut changer la valeur d'un octet (en mémoire) directement dans une fenêtre de visualisation de mémoire. Pour changer la valeur de l'octet vous cliquez dans la ligne contenant cet octet. Selon que vous cliquez dans la zone hexadécimale ou ascII, vous verrez la zone devenir une zone d'édition et vous pourrez alors changer la valeur hexadécimale ou le caractère ascII. Dans la fenêtre ci-dessus on a cliqué dans la zone hexadécimale de la deuxième ligne

près de la valeur $02. Le curseur de texte apparaît à cet endroit, on peut alors changer la valeur de cet octet (ou de tout autre octet de cette ligne). On doit valider le changement en appuyant sur la touche **Return** ou **Enter**. Le changement en mémoire se fait seulement après la validation à l'aide de l'une ou l'autre de ces touches. Si la ou les valeurs changées ne sont pas valides (une valeur hexadécimale incorrecte ou plus de valeurs que ne doit en contenir la ligne) le curseur de la souris se change en flèche vide et se positionne près d'un caractère incorrect.

Une autre technique des fenêtres de visualisation de mémoire est très utile pour passer une adresse à la réserve CLIP. On presse la touche Option en plaçant le curseur de la souris sur le premier chiffre de l'adresse désirée, dans la zone hexadécimale. ABZmon prend alors ce chiffre et le sept suivant pour former l'adresse (à huit caractères) et passe cette adresse dans une boîte de dialogue comme ci-dessous. Il n'est pas nécessaire que le premier caractère de l'adresse soit en début de ligne ou même le premier caractère d'un octet.

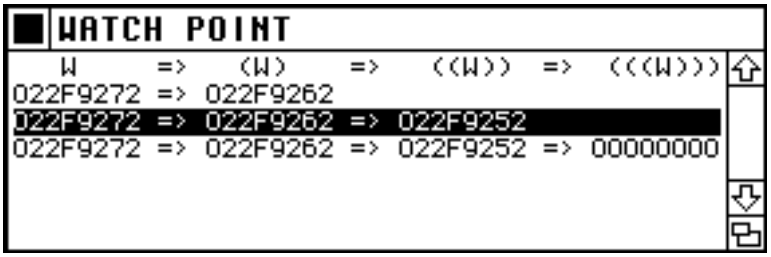


On utilise cette technique avantageusement lorsque la fenêtre contient l'adresse d'un pointeur. On passe l'adresse du pointeur à la réserve CLIP, on sélectionne cette adresse dans une fenêtre CLIP et on ouvre une nouvelle fenêtre de visualisation de mémoire, à cette adresse, par les touches **Commande M**. On a ainsi une représention de la région mémoire de ce pointeur .

En cliquant dans l'adresse en début de ligne, on sélectionne cette ligne. Elle apparaît alors en inversé et l'adresse du début de la ligne devient l'adresse par défaut, c-a-d, l'adresse qui apparaît à l'ouverture de la plupart des boîtes de dialogue.

Les fenêtres d'observation

Ce sont les fenêtres d'observation pour un pointeur à indirection simple, double, triple. On peut entrer jusqu'à dix éléments d'observation.

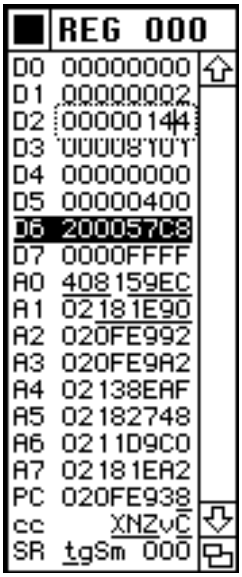


La première ligne contient le titre des colonnes. Les lignes suivantes sont les éléments d'observation. A la deuxième ligne on a une indirection simple. A l'adresse $22F9272 on retrouve la valeur $22F962. La troisième ligne montre que la valeur $22F962 est en fait un pointeur, la valeur pointée par cette adresse étant $22F9252. La quatrième ligne est une indirection triple qui montre que la valeur $22F9252 est également un pointeur, vers la valeur $0.

On peut sélectionner une ligne en cliquant n'importe où dans cette ligne. L'adresse par défaut est celle de la première colonne. Evidemment la ligne des titres ne peut être sélectionnée (de même que les lignes vides).

Les fenêtres de registres 68000

Ce sont les fenêtres qui montrent les registres Dn, An, SR et PC.



Les registres Dn, An et PC ont leurs valeurs affichées en hexadécimal. Le registre SR occupe les deux dernières lignes et se décompose en deux parties. L'octet bas de SR (bits 0 à 7) se nomme CCR et est représenté à l'avant-dernière ligne. L'octet haut (bits 8 à 15) est représenté à la dernière ligne. Pour l'octet CCR, chaque bit est représenté par la lettre définie au standard Motorola, soit X -> extend, N -> negative, Z -> zero, V -> overflow et C -> carry (les autres bits ne sont pas utilisés). Pour les bits 8 à 15 de SR, les bits de masque d'interruption, I2, I1 et I0 (bit 8 à 10) sont représentés par les chiffres 0 ou 1 selon l'état du bit. Les lettres S (supervisor) et M (master) sont au standard Motorola. Les bits de mode "trace", T1 et T0, sont respectivement représentés par les lettres T et G (le bit 11 n'est pas utilisé). Lorsque la lettre est en majuscule, le bit correspondant est à 1. Si la lettre est en minuscule, le bit correspondant est à 0.

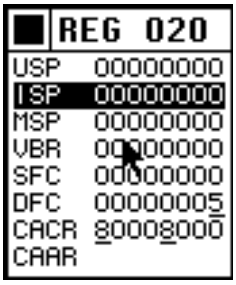
Pour mettre en évidence le changement des registres, ABZmon conserve une copie des registres. Lorsque le "débogueur" est à nouveau sollicité, il compare cette copie des registres aux nouvelles valeurs des registres et souligne les différences dans la fenêtre. Pour les registres Dn, An et PC les chiffres des valeurs hexadécimales qui diffèrent sont soulignés (par exemple le dernier chiffre du compteur de programme PC). Pour le registre SR, ce sont les changements de bits qui sont soulignés (dans la fenêtre ci-dessus les bits X, N, Z et C du CCR ont passé de 0 à 1).

On peut changer la valeur d'un registre directement dans une fenêtre de registre 68000. Pour changer la valeur du registre vous cliquez dans la ligne contenant ce registre, sur la valeur (ou représentation, dans le cas de SR) du registre. Vous verrez la valeur (ou représentation) de ce registre devenir une zone d'édition et vous pourrez alors changer la valeur (ou représentation) de ce registre. Dans la fenêtre ci-dessus on a cliqué dans la valeur du registre D2, entre les deux 4. Le curseur de texte apparaît à cet endroit, on peut alors changer la valeur (ou représentation) de ce registre. On doit valider le changement en appuyant sur la touche **Return** ou **Enter**. Le changement du registre se fait seulement après la validation à l'aide de l'une ou l'autre de ces touches. Si la valeur (ou la représentation) changée n'est pas valide (une valeur hexadécimale incorrecte ou plus de valeurs que ne doit en contenir la ligne ou bit inadéquat) le curseur de la souris se change en flèche vide et se positionne près d'un caractère incorrect.

Pour sélectionner une ligne de registre, on doit cliquer dans le nom de ce registre. Dans la fenêtre ci-dessus, le nom est D6, à la septième ligne. L'adresse par défaut devient la valeur du registre. Les deux dernière lignes, pour le registre SR, ne pourront être sélectionnées.

Les fenêtres des autres registres 32 bits (CPU)

Ce sont les fenêtres qui montrent les registres USP, ISP, MSP, VBR, DFC, CACR et CAAR. Ces registres ne se retrouvent que dans les microprocesseurs 68020, 68030 ou 68040. Certains peuvent être absents (comme CAAR, pour les 68040, dans la fenêtre ci-dessous). Dans ce cas, aucune valeur n'est affichée vis-à-vis le nom de ce registre.



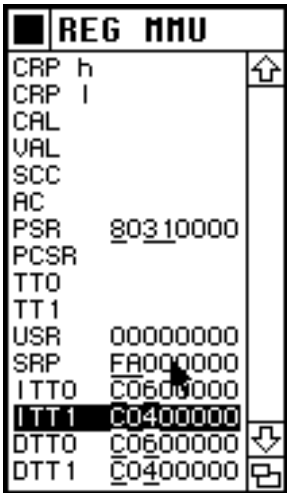
Comme pour les registres 68000, les chiffres soulignés indiquent le changement par rapport aux valeurs des registres précédents. Pour sélectionner une ligne de registre, on doit cliquer dans le nom de ce registre.

Dans la fenêtre ci-dessus, le nom est ISP, à la deuxième ligne. L'adresse par défaut devient la valeur du registre. Les lignes vides peuvent être sélectionnées mais ne changent pas la valeur par défaut.

En cliquant sur la valeur du registre (devant le nom du registre) on peut changer cette valeur. La boîte de dialogue suivante permet de changer la valeur du registre VBR, choisi dans la fenêtre ci-dessus.

Les fenêtres de registres MMU

Ce sont les fenêtres qui montrent les registres MMU. C'est registres ne se retrouvent que dans les microprocesseurs 68030 et 68040 et dans le coprocesseur 68851. Certains peuvent être absents (comme CAL, VAL, SCC pour les 68040, dans la fenêtre ci-dessous). Dans ce cas, aucune valeur n'est affichée vis-à-vis le nom de ce registre.



Comme pour les registres 68000, les chiffres soulignés indiquent le changement par rapport aux valeurs des registres précédentes. Pour sélectionner une ligne de registre, on doit cliquer dans le nom de ce registre. Dans la fenêtre ci-dessus, le nom est ITT1, à la troisième ligne de la fin. L'adresse par défaut devient la valeur du registre. Les lignes vides peuvent être sélectionnées mais ne change pas la valeur par défaut.

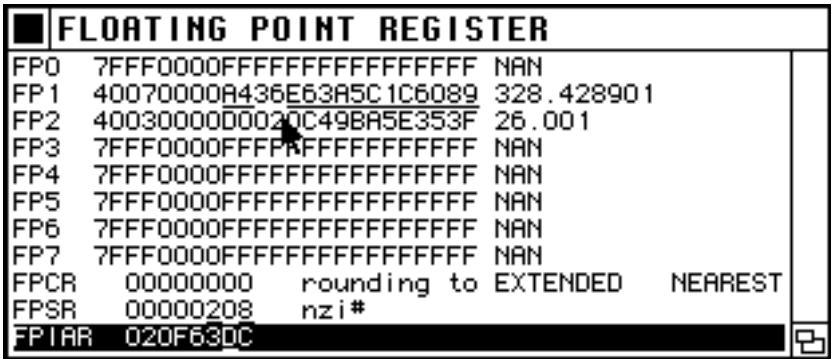
En cliquant sur la valeur du registre (devant le nom du registre) on peut changer cette valeur. La boîte de dialogue suivante permet de changer la

valeur du registre SRP, choisi dans la fenêtre ci-dessus.

Certains registres, DRP, SRP et CRP, sont des registres 64 bits. ABZmon les visualise en partie haute, avec la lettre **h** (high), et en partie basse, avec la lettre **l** (low). Pour modifier la valeur de ces registres vous devez procéder en deux étapes, une pour chaque partie.

Les fenêtres de registres FPU

Ce sont les fenêtres qui montrent les registres des nombres en virgules flottantes. Ces registres ne se retrouvent que dans le microprocesseur 68040 et les coprocesseurs 68881 et 68882. Comme pour les registres 68000, les chiffres soulignés indiquent le changement par rapport aux valeurs des registres précédentes.



Les registres de données FPn

Le huit premières lignes sont pour les registres généraux FPn. A droite du nom de ces registres, vous voyez la valeur hexadécimale du registre et ensuite, sa représentation en virgule flottante (ou une expression telle NAN (not a number) ou INFINITY (infini) si une telle representation en virgule flottante n'existe pas). En cliquant sur une de ces huit lignes on obtient la boîte de dialogue suivante qui permet de changer la valeur du registre correspondant.

On peut aussi utiliser la notation scientifique, comme dans 2.6001e1, pour représenter le nombre ci-dessus. Le nombre à droite du "**e**" est l'exposant de 10. Le bouton **HEXA** de cette boîte de dialogue permet le changement de valeur sous la forme hexadécimale.

On revient à la représentation en virgule flottante par le bouton **DECI**.

Le registre de contrôle FPCR

A droite du nom du registre, on trouve la valeur hexadécimale du registre. Les deux octets bas seulement sont utilisés bien que le registre soit sur 32 bits. Plus à droite encore, on trouve le type d'arrondi pour les valeurs en virgules flottantes. Le mode est représenté par NEAREST, ZERO, -INFINIT et +INFINIT (au plus près, à zéro, à moins l'infini et à plus l'infini). La précision est représentée par EXTENDED, SINGLE, DOUBLE ou ???????? (étendue, simple, double ou inconnue).

On peut changer la valeur de ce registre en cliquant n'importe où sur la ligne. Une boîte de dialogue semblable à celle utilisée pour les registres MMU permet d'entrer la nouvelle valeur du registre.

Le registre de status FPSR

A droite du nom du registre, on trouve la valeur hexadécimale du registre. L'octet supérieur n'est pas utilisé mais le registre est sur 32 bits. Plus à droite encore, on trouve la représentation des quatre bits les plus significatifs (soit les bits 27 à 24) l'octet des condition de code.

Le bit 27 (valeur négative) est représenté par la lettre "n" (majuscule si le bit est à 1, c-à-d, valeur négative et minuscule si le bit est à 0, c-à-d, valeur positive).

Le bit 26 (valeur zéro) est représenté par la lettre "z" (majuscule si le bit est à 1, c-à-d, valeur nulle et minuscule si le bit est à 0, c-à-d, valeur non nulle).

Le bit 25 (valeur infinie) est représenté par la lettre "i" (majuscule si le bit est à 1, c-à-d, valeur infinie, et minuscule si le bit est à 0, c-à-d, valeur finie).

Le bit 24 est représenté par le caractère "#" si le bit est à 1 signifiant non-valeur, et par "-" si le bit est à 0.

Le registre d'adresse FPIAR

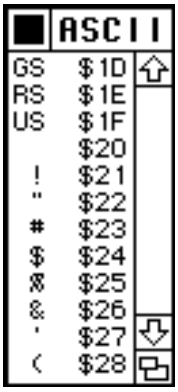
Pour les instructions qui génèrent des exceptions "traps", le registre FPIAR conserve l'adresse de ces instructions, adresse qui est alors utilisée par la routine d'exception.

A droite du nom du registre, on trouve la valeur hexadécimale du registre. La ligne de ce registre est la seule, dans la fenêtre, qui peut être sélectionnée, pour donner l'adresse par défaut. Pour ce faire on doit cliquer dans le nom du registre.

On peut changer la valeur de ce registre en cliquant sur la valeur hexadécimale du registre. Une boîte de dialogue semblable à celle utilisée pour les registres MMU permet d'entrer la nouvelle valeur du registre.

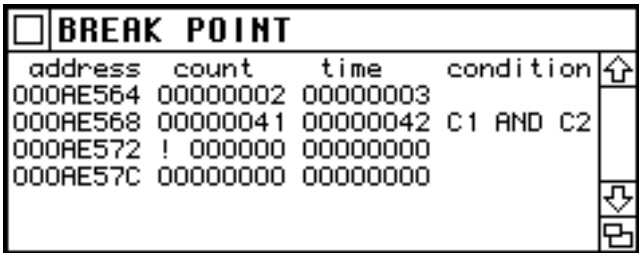
Les fenêtres de caractère ascII

Ce sont les fenêtres qui montrent, dans l'ordre, tous les caractères ascII (ou leurs noms s'ils ne sont pas représentables) ainsi que leurs valeurs hexadécimales.



Les fenêtres de points d'arrêt

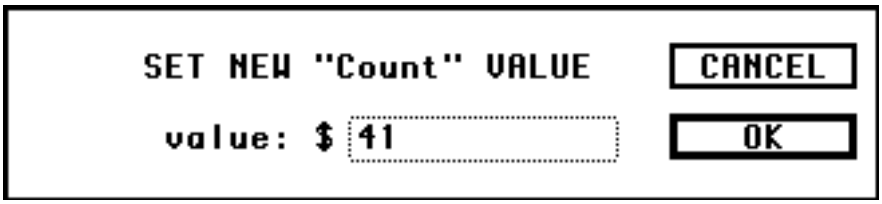
Ce sont les fenêtres qui montrent tous les points d'arrêt précédemment fixés par le menu **BREAK PNT**.



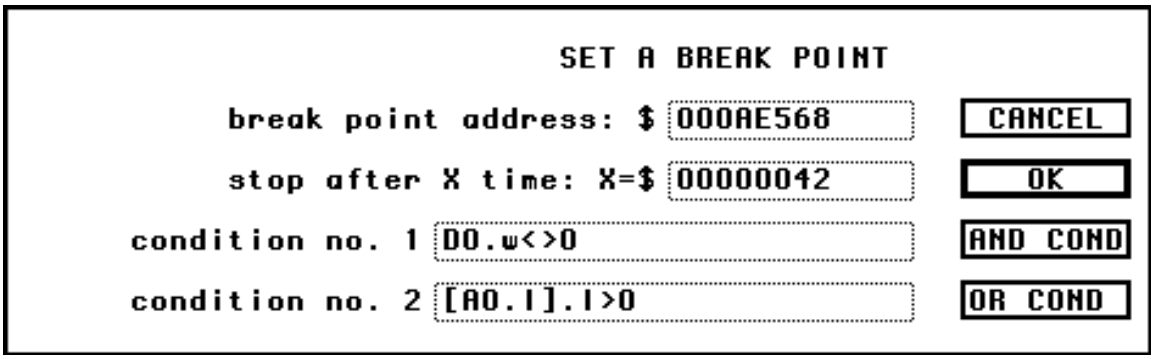
La première ligne contient le nom des colonnes. La première colonne à gauche, **address**, contient l'adresse du point d'arrêt. La seconde colonne, **count**, le nombre de passages au point d'arrêt, avant l'arrêt. La troisième colonne, **time**, le nombre initial de passages requis avant l'arrêt. La dernière colonne, **condition**, donne les conditions pour décrémenter la deuxième colonne **count**. S'il n'y a pas de condition. le compteur est décrémenté à chaque passage.

La différence entre la troisième et la deuxième colonne donne le nombre de passages effectués. A la deuxième ligne (premier point d'arrêt à l'adresse $AE564), par exemple, on voit qu'il reste 2 passages (sur 3) à effectuer avant l'arrêt. Il y a eu un passage à cette adresse. A la dernière ligne, le point d'arrêt va effectivement ordonner l'arrêt car le compte est à zéro. Le point d'arrêt à l'adresse $AE568 n'aura lieu qu'au 65 ième passage ($41) pour lequel les conditions C1 et C2 sont simultanément réalisées. Si le point d'arrêt est inactif (voir l'item **DISEABLE ONE** dans le menu **BREAK PNT**) un point d'exclamation apparaît à la deuxième colonne. C'est le cas pour le point d'arrêt à l'adresse $AE572 (quatrième ligne).

Si on clique une adresse de point d'arrêt à la première colonne, cette adresse devient l'adresse par défaut. Si on clique dans la deuxième colonne, **count**, on fait apparaître une boîte de dialogue qui permet de changer le nombre de passages à effectuer avant l'arrêt.

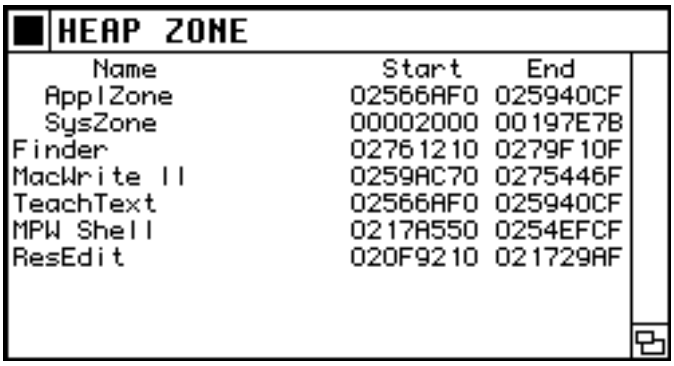


Si on clique sur le reste de la ligne on peut changer tous les paramètres du point d'arrêt (voir l'item **SET ONE** dans le menu **BREAK PNT**).



Les fenêtres de zones

Ce sont les fenêtres qui montrent les différentes zones mémoires utilisées par le système et les applications.

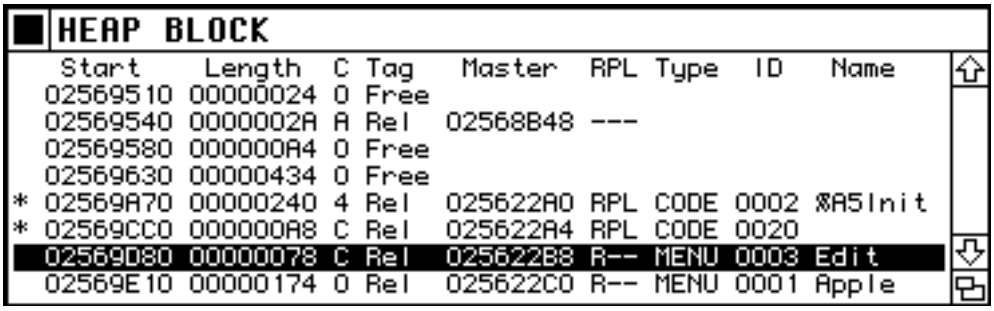


La première ligne contient le nom des colonnes. La première colonne à gauche, **Name**, contient le nom du programme (application, accessoire de bureau, tableau de contrôle, etc). La seconde, **Start**, et la dernière, **End**, contiennent respectivement l'adresse du premier et du dernier octet du programme.

En cliquant sur n'importe quelle ligne non vide, on fait apparaître une boîte de dialogue pour l'ouverture d'une fenêtre qui décrit chaque bloc du programme correspondant.

Les fenêtres de blocs mémoire

Ce sont les fenêtres qui montrent les différents blocs mémoire d'un programme choisi dans une fenêtre de zone.

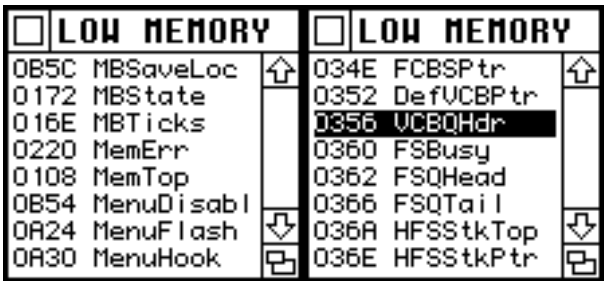


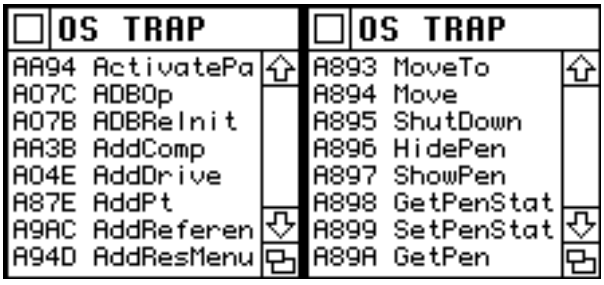
La première ligne contient le nom des colonnes. La première colonne à gauche, **Start**, contient l'adresse du bloc. La seconde, **Length**, contient la longueur du bloc. La troisième, **C**, est le facteur de correction, pour que le bloc soit aligné sur 16 octêts dans la mémoire. La colonne **Tag** identifie la sorte de bloc: **Free** (libre), **NonR** (non-relocalisable), **Rel** (relocalisable) et **????** (inconnue). Pour les blocs relocalisables, la colonne **Master** contient l'adresse du maître-pointeur (handle). La colonne **RPL** indique si le bloc est une ressource (**R** ressource), si le bloc peut être purgé (**P** purgeable) et si le bloc est verrouillé (**L** lock). Une tiret indique que le bloc ne possède pas la propriété. Pour les blocs ressources, la colonne **Type** donne le type de ressource, la colonne **ID,** le numéro hexadécimal de la ressource et la colonne **Name**, le nom de la ressource (si elle est nommée). Ces dernières informations ne sont pas toujours disponibles. Dans ce cas le message **Ressource not found** est affiché.

Si le bloc ne peut pas bouger, parce qu'il est verrouillé ou non-relocalisable, une étoile est affichée en début le ligne, pour la ligne de ce bloc. On peut sélectionner une ligne en cliquant n'importe où dans cette ligne. L'adresse par défaut est celle de la première colonne.

Les fenêtres de constantes

Ce sont les fenêtres qui montrent les numéros et noms des variables globales du OS et les numéros et noms des "trappes" du LineA.



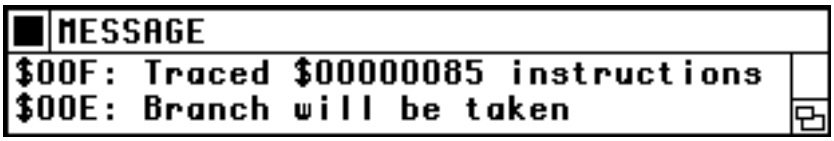


La présentation se fait par ordre alphabétique des noms ou par valeurs croissantes des constantes. On peut sélectionner une ligne en cliquant n'importe où dans cette ligne. La valeur par défaut est celle de la constante.

La fenêtre de messages

Ce sont les fenêtres qui affichent les messages de ABZmon:

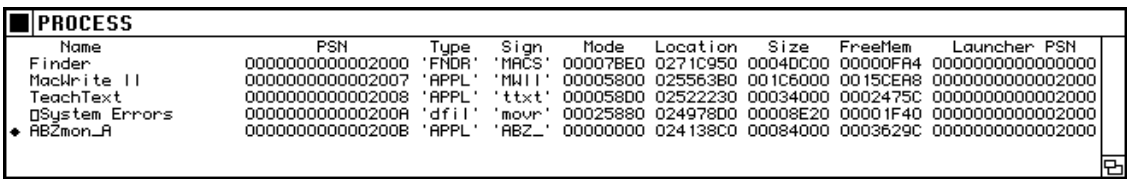
- rencontre d'un point d'arrêt (voir le menu **BREAK PNT**)
- interception de LineA (voir le menu **STOP**)
- rencontre d'une instruction Debug (voir **Pour entrer dans ABZmon**)
- DebugNum inactif (voir le menu **STOP**)
- conditions satisfaites (voir le menu **STOP**)
- nombre d'instructions exécutées (voir aussi le menu **STEP**)
- une fenêtre ne peut être affichée (voir **Les fenêtres**)
- utilisation du commutateur d'interruption (voir **Pour entrer dans ABZmon**)
- point d'arrêt effacé (on a probablement oublié de l'enlever...)
- la boucle continue ou s'arrête
- nouvelle définition d'un ancien point d'arrêt
- branchement va être effectué ou non (instructions Bcc et DBcc)
- l'octet va être mis à -1ou 0 (instructions Scc)
- on va procéder (ou non) à l'instruction TRAPcc



La ligne débute par le numéro du message. Le message le plus récent est toujours à la première ligne. Aucune ligne ne peut être sélectionnée. Pas plus d'une fenêtre de messages peut être ouverte en tout temps.

Les fenêtres d'applications

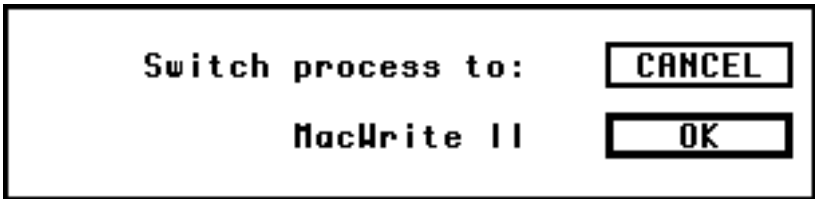
Ce sont les fenêtres qui donnent la liste des applications ouvertes et diverses informations sur ces applications. Ne fonctionne que pour le sytème 7 ou plus récent. L'intérêt principal est de pouvoir changer l'application courante pour une autre application lorque l'application courante est fautive.



La première ligne contient le nom des colonnes. La première colonne à gauche, **Name**, contient le nom de l'application. La seconde, **PSN**, contient le numéro de l'application (process serial number). La troisième, **Type**, contient le type d'application. La quatrième, **Sign**, est la "signature" (créateur) de l'application. La colonne **Mode** donne des informations contenues dans la ressource "**SIZE**" de l'application. L'adresse du début du code pour l'application est à la colonne **Location** tandis que la colonne **Size** donne la

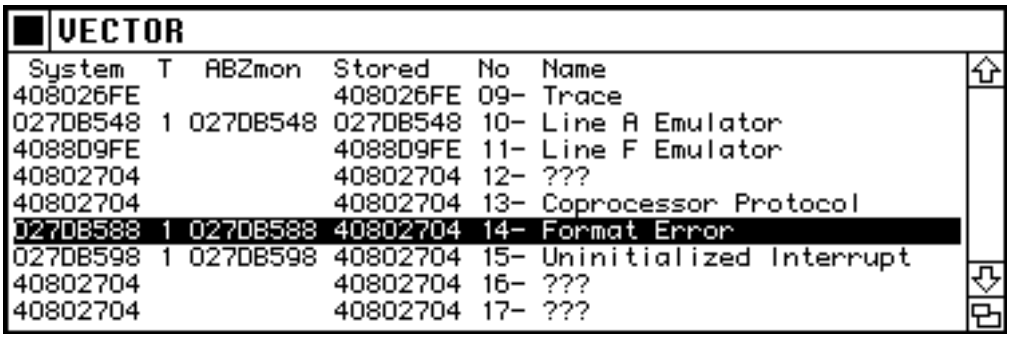
taille de cette application. **FreeMem** indique la mémoire disponible pour l'application et **Laucher PSN** est le numéro de l'application qui a lancé cette application (généralement le Finder).

On cliquant sur une ligne d'application on fait apparaître la boîte de dialogue ci-dessous qui permet de "passer" à une autre application.



Les fenêtres des vecteurs d'exception

Ce sont les fenêtres qui donnent différentes informations sur les 58 vecteurs d'exception du microprocesseur. On peut aussi y changer la valeur de ces vecteurs.



Le Mac utilise généralement deux ensembles de vecteurs. Pour le système d'opération, le premier vecteur est à l'adresse **VBR** (vector base address). Pour les applications, le premier vecteur est à l'adresse **0**. ABZmon permet de passer facilement de l'un à l'autre en cliquant sur la première ligne. Dans la fenêtre ci-dessus, en cliquant sur cette ligne, on passe au deuxième ensemble. Le nom **System** change alors pour **Applic** et les nouvelles valeurs des vecteurs application sont affichées.

La première ligne contient le nom des colonnes. La première colonne à gauche, **System** ou **Applic**, contient la valeur du vecteur c-a-d l'adresse de la routine d'exception.

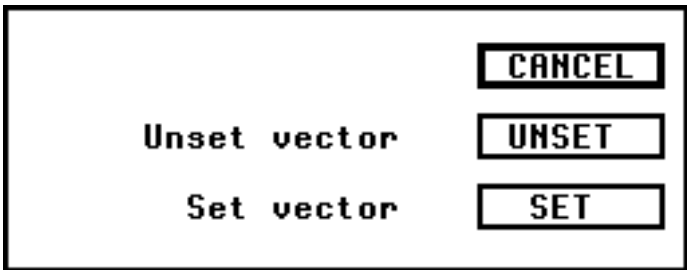
La seconde, **T**, contient le type de vecteur. Si le vecteur de cette ligne n'a pas été changé (par ABZmon, au démarrage, ou par l'usager) cette colonne et la suivante sont vides. La valeur **1** signifie que le vecteur a été changé une fois. La lettre **i** signifie que ce vecteur est changé à chaque fois qu'il y a une interruption (hardware) c-a-d à tout les 1/60 de seconde environ. La lettre **m** signifie que le vecteur est changé à chaque fois que ABZmon apparaît.

La troisième colonne, **ABZmon**, donne la valeur du vecteur tel que fixé par le "débogueur". Si cette valeur n'est pas la même que celle de la première colonne, c'est le signe qu'un programme a modifié le vecteur après le "débogueur".

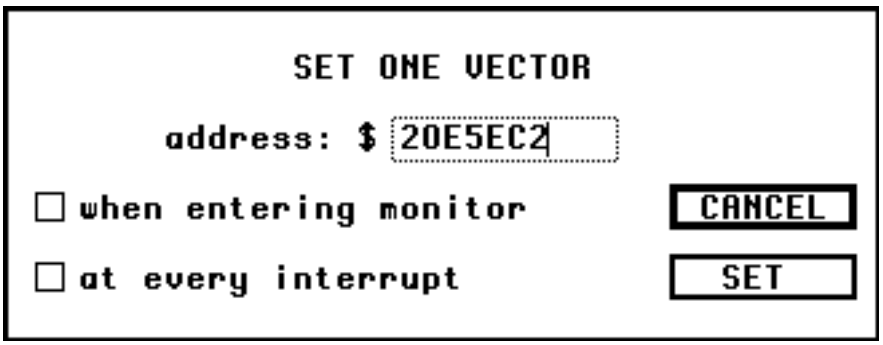
La colonne **Stored** contient la valeur du vecteur au démarrage de ABZmon, la colonne **No** contient le numéro du vecteur et la colonne **Name** le nom du vecteur tel que défini par le fabricant Motorola.

On peut sélectionner une ligne en cliquant n'importe où dans cette ligne, sauf à la colonne **T**. La valeur par défaut est celle de la première colonne.

Pour changer la valeur d'un vecteur vous devez cliquer à l'intersection de la colonne T et de la ligne de ce vecteur. La boîte de dialogue suivante apparaît:



Le bouton **CANCEL** annule la démarche. Le bouton **UNSET** restitue la valeur originale si le vecteur a été changé (au démarrage ou par l'usager) et affiche un message d'erreur dans le cas contraire. Le bouton **SET** montre la boîte de dialogue suivante:

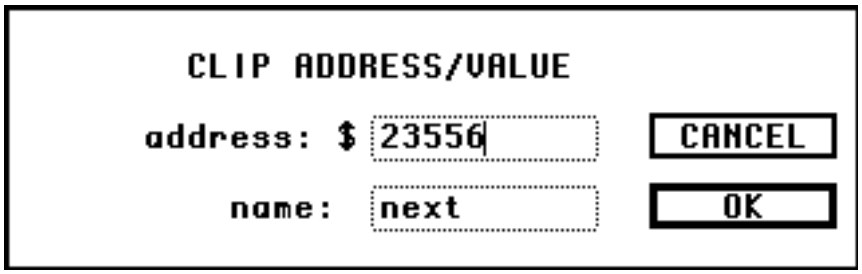


Dans le champ d'édition **address**, vous entrez l'adresse pour la nouvelle routine d'exception. La case **when entering monitor**, si cochée, change la valeur du vecteur chaque fois que ABZmon apparaît tandis que la case **at every interrupt**, si cochée, change la valeur du vecteur à chaque fois qu'il y a une interruption (hardware) c-a-d à tout les 1/60 de seconde environ. Vous validez avec le bouton **SET**.

Les fenêtres CLIP

Ce sont les fenêtres qui contiennent des adresses ou valeurs utiles et fréquement utilisées (jusqu'à dix). On peut associer un nom à ces adresses ou valeurs et les modifier à partir de la fenêtre.

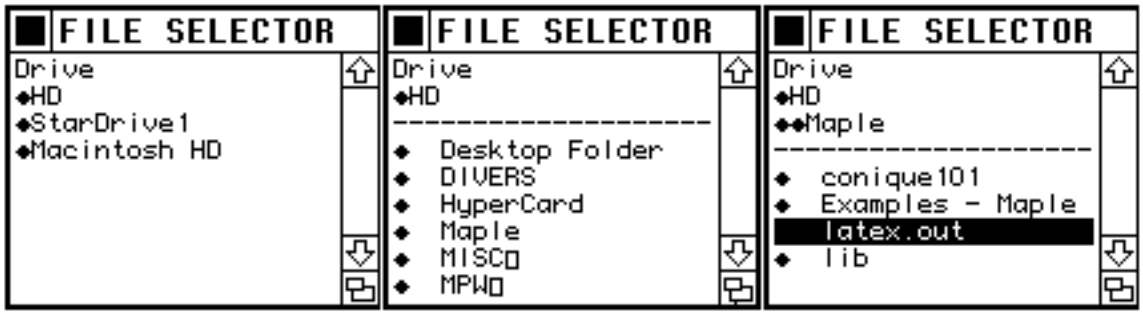
La première colonne contient l'adresse ou la valeur utile et la seconde le nom associé. On peut sélectionner une ligne en cliquant sur l'adresse ou la valeur utile. La valeur par défaut est cette adresse ou valeur utile. En cliquant sur le nom on fait apparaître la boîte de dialogue suivante:



On peut alors changer l'adresse ou la valeur utile et le nom associé.

Les fenêtres de sélection de documents

Ce sont les fenêtres qui permettent de choisir un fichier (c'est l'analogue, dans l'interface du Mac, de la boîte de sélection des fichiers pour l'ouverture d'un document).



La première fois qu'une fenêtre de sélection de document est ouverte elle donne la liste de tous les disques connectés, y compris ceux en réseau. C'est

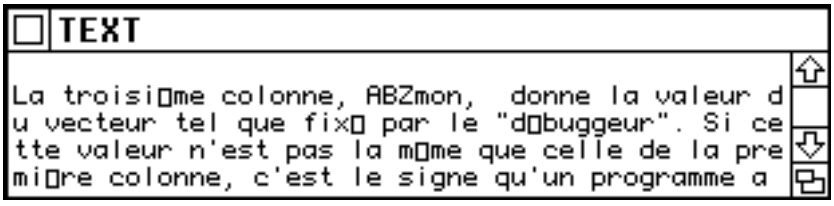
la première fenêtre à gauche ci-dessus. Le symbole en forme de losange identifie un disque ou un dossier. En cliquant sur le disque nommé **HD** on obtient la fenêtre du milieu qui donne, après la ligne pointillée, le contenu du premier niveau hiérarchique du disque **HD**. Le disque **HD** contient les dossiers **Desktop Folder**, **Divers**, **Hypercard**, **Maple**... On a cliqué sur le dossier Maple pour voir son contenu. C'est la fenêtre la plus à droite. Le contenu, après la ligne pointillée, est contitué des dossiers **conique101**, **Examples - Maple**, **lib** et du document **latex.out**. Ce fichier a été sélectionné en cliquant sur son nom.

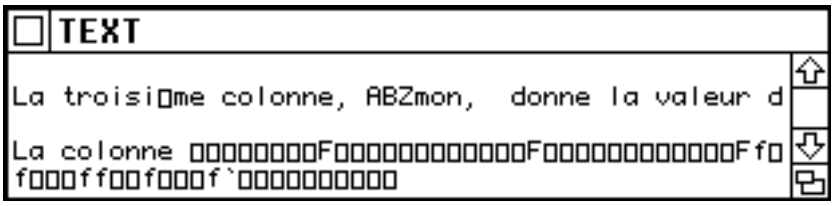
En résumé, les lignes avant les pointillés donnent la hiérarchie des dossiers et les lignes après les pointillés donnent le contenu du dernier dossier dans la hiérarchie. En cliquant sur les lignes après les pointillés on ouvre le dossier si la ligne est un dossier (qui s'ajoute aux dossiers de la hiérarchie, juste avant la ligne pointillée) et on sélectionne le document si la ligne est un document. Si on clique sur une ligne avant les pointillés, dans la hiérarchie des dossiers, on ouvre (après le pointillé) le dossier ou disque de cette ligne. Par exemple, si dans la troisième fenêtre, on clique sur HD, on va ouvrir le contenu du disque HD tel que montré à la fenêtre du milieu.

Le type de document apparaissant dans la fenêtre dépend de la valeur de la variable interne que l'on peut changer dans le menu **SPECIAL**, à l'item **FILE TYPE**. ABZmon montre tous les documents **TEXT** et tous les documents du type de cette variable interne. Par exemple, si le type est **MW2D**, tous les documents **MacWrite II** sont affichés. Si la variable interne ne contient rien tous les fichiers sont montrés.

Les fenêtres de textes

Ce sont les fenêtres qui permettent de visualiser un fichier. L'apparence de la fenêtre dépend d'une variable interne que l'on peut changer dans le menu **SPECIAL**, à l'item **TEXT LINE**. Lorsque cette variable interne est à zéro, la ligne du texte original est brisée et se continue à la ligne suivante si cette ligne dépasse la fenêtre. C'est le cas de la première fenêtre ci-dessous (le texte provient de la version intilal du document **MacWrite II** que vous lisez présentement). Lorsque la variable interne n'est pas zéro, la ligne n'est pas brisée. La ligne suivante de la fenêtre débute après la rencontre du dernier caractère "retour de chariot". Comme il s'agit d'un document **MacWrite II**, ce caractère marque la fin d'un paragraphe. La ligne suivante sépare les paragraphes et l'autre ligne débute le second paragraphe comme on peut le voir à la seconde fenêtre ci-dessous. Les caractères non-imprimables sont représentés par de petits rectangles.

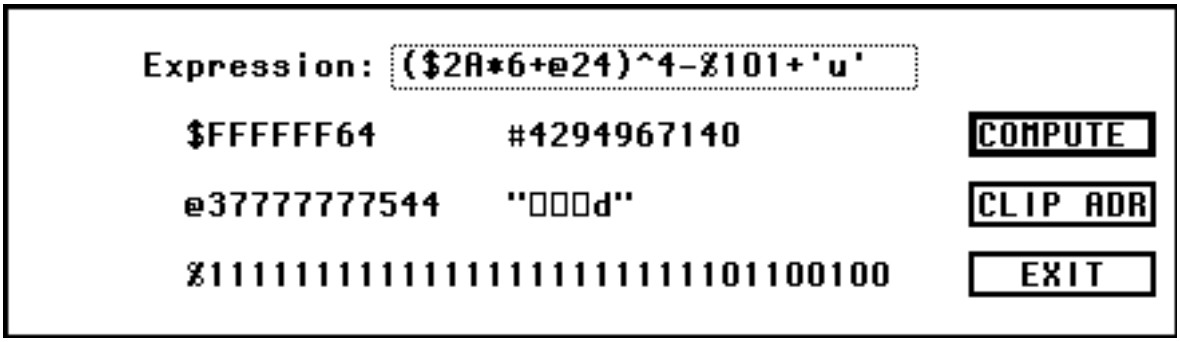




Une deuxième variable interne détermine le type de déroulement du texte avec les flèches vers le haut et vers le bas. On peut changer cette variable interne dans le menu **SPECIAL**, à l'item **TEXT SCROLL**. Le déroulement se fait une ligne à la fois ou une page à la fois selon l'état de cette variable. L'item **TEXT SCROLL** fait passer d'un mode à l'autre.

Le calculateur

C'est une boîte de dialogue qui permet de faire des calculs et conversions en système hexadécimal, décimal, octal, binaire et représentatation ascII. On appelle le calculateur en appuyant sur la touche "=".



Dans le champ d'édition intitulé **Expression** on entre l'expression à calculer. On valide avec les touches **Return** ou **Enter** ou en utilisant le bouton **COMPUTE**. En cas d'erreur le curseur se change en flèche creuse et se positionne près du caractère fautif. Le bouton **CLIP ADD** ajoute la valeur calculée à la réserve CLIP. La valeur calculée est aussi conservée en mémoire et devient la valeur par défaut pour cette boîte de dialogue si aucune ligne n'est sélectionnée dans une fenêtre. On termine en utilisant le bouton **EXIT**.

Le calculateur affiche les cinq conversions dans le corps de la boîte de dialogue, précédés des symboles identifiant le système de nombre utilisé, tel que défini ci-dessous. La conversion ascII, entre guillemets, affiche un petit rectangle vide si le caractère correspondant n'est pas imprimable.

Le calculateur reconnaît les nombres hexadécimaux (précédés de **$**), les nombres décimaux (précédés ou non de **#**), les nombres octals (précédés de **@**) et les nombres binaires (précédés de **%**). Il reconnaît aussi des nombres sous forme ascII, tel "**abCd**" ou '**zx**' entre guillemets ou apostrophes, de une à

quatre lettres. Chaque lettre représente un octet d'un entier long sur quatre octets. Dans les deux exemples précédents les valeurs de ces entiers sont $61624364 et $00007A78.

Le calculateur utilise les opérateurs suivants, en ordre de priorité croissante, tel que défini par le langage C.

- ~	moins unitaire et opérateur non binaire (biwise not)
/ *	division et multiplication
+ -	addition et soustraction
<< >>	décalage logique à gauche et à droite (bitwise shift)
&	'et' logique (bitwise 'and')
^	'ou' exclusif (bitwise 'xor')
|	'ou' logique (bitwise 'or')
()	parenthèses

A l'ouverture du calculateur, la valeur par défaut est la dernière utilisée par le calculateur, à moins qu'une adresse soit sélectionnée dans une fenêtre. Dans ce cas c'est la valeur sélectionnée qui devient la valeur par défaut. Pour obtenir la valeur décimale d'un registre, par exemple, il suffit de sélectionner le registre et de presser la touche '**=**'.

Les adresses JB

Ce sont des adresses données avec indirection simple ou double des registres et déplacements. Il y a plusieurs formes reconnues par le "débogueur". La forme générale est:

[Rn.s+dép1].s+dép2

ou **Rn** est un registre d'adresse (A0-A7, SP) ou de donnée (D0-D7) ou le compteur de programme (PC), **dép1** et **dép2** des déplacements et .**s** la taille de ces élements (.b -> un octet, .w -> deux octets, .l -> quatre octets). Si **.s** n'est pas spécifié, la taille par défaut est l'entier long sur quatre octets. **Rn**, **dép1** et **dép2** sont facultatifs.

Voici des exemples de forme d'adresses JB:

[PC+5000E]+20

si PC = 20000 et que la valeur de l'entier long à l'adresse 7000E (c-a-d 20000+5000E) est 44444, l'adresse JB est 44464 (c-a-d 44444+20)

[D5.w]+20

si D5 = 701234 (alors D5.w = 1234) et que la valeur de l'entier long à l'adresse 1234 est 44444, l'adresse JB est 44464 (c-a-d 44444+20)

[22222].b-20

si la valeur de l'octet à l'adresse 22222 est 84 alors la valeur JB est 64 (c-a-d 84-20)

[A6]

si A6 = 44444 et que l'entier long à cette adresse est 12345 alors l'adresse JB est 12345

SP+20

si le pointeur de pile à la valeur 66666 alors l'adresse JB est 66686 (c-a-d 66666+20)

A4

L'adresse JB est l'adresse du registre A4 (et non la valeur hexadécimale A4)

E3678

L'adresse JB est E3678

Les conditions JB

Ce sont des expressions qui comparent deux adresses JB. Les comparaisons se font sur des entiers non-signés. Elles sont de la forme:

JB1 op JB2

ou **JB1** et **JB2** sont des adresses JB et **op** un des opérateurs:

<	**>**	strictement plus petit, grand
<=	**=<**	plus petit ou égal
>=	**=>**	plus grand ou égal
=		égal
<>	**><**	différent

Voici quelques exemples:

D5.b > 6

Pour que la condition soit réalisée il faut que l'octet bas du registre D5 soit strictement supérieur à 6.

[22222].w=[22224].w

Pour que la condition soit réalisée il faut que le mot sur deux octets à l'adresse 22222 soit le même que celui à l'adresse 22224.

[PC].w>=F000

Pour que la condition soit réalisée il faut que l'instruction courante soit une instruction de co-processeur (lineF).

Touches claviers spéciales

Option et **flèche**: simule la souris, dans la direction des flèches

Option, **commande** et **flèche**: simule la souris, dans la direction des flèches, comme si le bouton de la souris était enfoncé.

flèche: vers le haut et vers le bas pour le déroulement plus lent de la fenêtre;
vers la gauche et vers la droite pour déplacer le curseur de texte dans une zone d'édition.

'**l**','**r**','**u**','**d**' ou '**L**','**R**','**U**','**D**',: simule flèche vers la gauche (left), vers la droite (right), vers le haut (up), vers le bas (down) pour les claviers dépourvus de flèches.

Commande et '**m**': demande l'ouverture d'une fenêtre de visualisation de la mémoire (équivalent de **MEM DUMP** dans le menu **OPEN**).

Commande et '**d**': demande l'ouverture d'une fenêtre de code (équivalent de **DISASSEM** dans le menu **OPEN**).

Commande et '**s**': pour sauvegarder la disposition de fenêtres (équivalent de **SAVE WINDOW** dans le menu **SPECIAL**)

Commande et '**c**': pour ajouter à la réserve CLIP (équivalent de **CLIP ADR** dans le menu **SPECIAL**).

Commande et '**q**': pour la sortie du "débogueur" (équivalent de **GO** ou **QUIT** dans le menu **CONTRL**).

Contrôle et '**m**': ré-initialise le port ADB. Utile quant la souris (**m**ouse) gèle...

Option, Comande et '-': capture d'écran à l'intérieur de ABZmon.

'**g**' ou '**G**': pour la sortie du "débogueur" (équivalent de **GO** ou **QUIT** dans le menu **CONTRL**).

'**s**' ou '**S**': demande le suivi pas à pas (équivalent de **ONE STEP** dans le menu **STEP**).

'**t**' ou '**T**': demande aussi le suivi pas à pas (équivalent de **TRACE** dans le menu **STEP**).

'.' : (point) pour mettre active la fenêtre suivante. Utile pour faire voir toutes les fenêtres, y compris celles masquées par les autres

',' : (virgule) pour mettre active la fenêtre précédente.

'%' : la fenêtre active est continuellement rafraîchie. Le curseur se change en flèche courte. Observez le résultat en ouvrant une fenêtre de visualisation de mémoire à l'adresse 16A et en appuyant sur cette touche. En appuyant une nouvelle fois sur cette touche on annule cet effet.

'*' : recalcule l'adresse des étiquettes (équivalent de **RESET ADR** dans le menu **OPT DISAS**).

'/' : change la forme de déplacement utilisé dans le désassemblage du code. (voir le menu **OPT DIS**).

'=' : pour faire apparaître le calculateur

'~', '`' ou **Escape**: (tilde ou apostrophe inverse) fait voir l'écran normal. En pressant une autre touche ou le bouton de la souris on revient à l'écran ABZmon. A utiliser lors du suivi d'une routine QuickDraw par exemple.

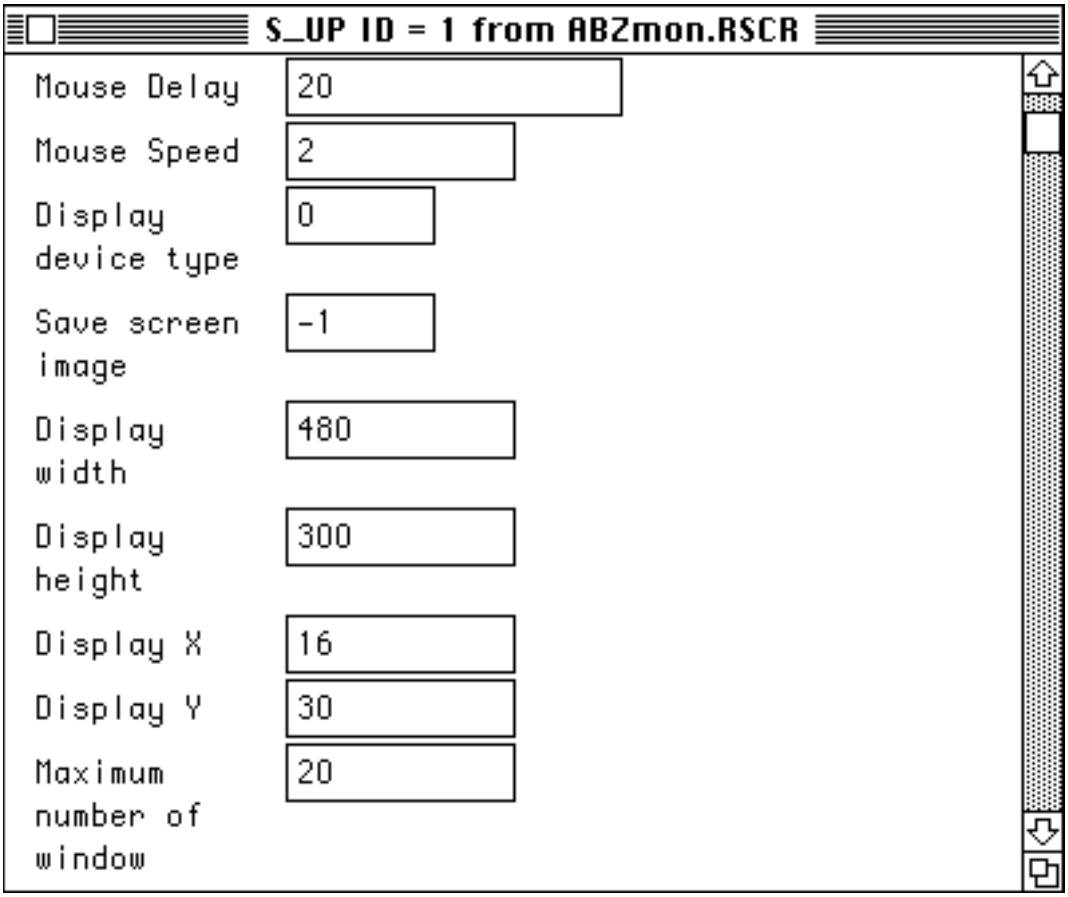
Return ou **Ente**r: pour valider un changement dans une fenêtre de visualisation de mémoire ou une fenêtre de registre, et pour simuler le bouton au contour plus épais dans une boîte de dialogue. Aussi pour répéter la dernière commande **GO**, **TRACE** ou **STEP**

Bar d'espacement: répète la dernière commande **GO**, **TRACE** ou **STEP.**

Variables internes

Ces variables servent à changer certaines caractéristiques de ABZmon comme le moniteur vidéo sur lequel apparaîtra le "débogueur", la dimension de l'écran ABZmon, l'apparence des fenêtres de code, la taille des caractères dans les menus, etc. Certaines de ces variables doivent être manipulées avec la plus grande précaution, comme les vecteurs d'exception ou la taille de la pile interne de ABZmon.

Vous pouvez changer ces variables en utilisant ResEdit. Vous ouvrez ABZmon et sélectionnez la ressource S_UP numéro 1. Vous verrez apparaître une fenêtre donnant le nom des variables et leurs valeurs:



Vous trouverez ci-dessous la signification de chaque variable:

Spy Buffer Size: la taille maximale du segment mémoire à enregistrer et à comparer à chaque instruction par la commande **ARRAY** du menu **SPY**.

Stack Size: la taille de la pile interne de ABZmon. En utilisant cette pile le débogueur garantie l'intégrité de la pile au dessus et aussi au dessous du pointeur de pile.

Key Delay: le delai requis pour répéter la touche clavier en maintenant la clé enfoncée.

Mouse Delay: le delai requis pour faire reconnaître un second "clic" de la souris.

Mouse Speed: la vitesse de déplacement du curseur par rapport à la souris.

Display Device Type: si vous disposez de plusieurs moniteurs vidéo, cette variable permet de choisir celui sur lequel apparaîtra l'écran ABZmon. La valeur 0 fait apparaître l'écran ABZmon sur le moniteur vidéo principal. Les valeurs 1, 2, 3,... feront apparaître l'écran ABZmon sur le second, troisième, quatrième... moniteur vidéo. Si vous avez des problèmes avec ces valeurs (Lisa, XL...), essayer la valeur -1: ABZmon tentera de régler le problème en utilisant les définitions écran de QuickDraw plutôt que ceux du NuBus.

Save screen image: L'image en dessous de l'écran ABZmon sera restituée si cette variable a la valeur -1. Si vous utilisez un seul moniteur vidéo, vous devez utiliser cette valeur. Si vous utilisez un second moniteur, il n'est pas nécessaire de conserver cette image, vous sauverez un peu de mémoire et vous verrez l'écran ABZmon même après avoir quitté le "débogueur".

Display width: la largeur de l'écran ABZmon. Ne jamais laisser cette valeur inférieure à 480. Certaines boîtes de dialogue ont cette largeur.

Display height: la hauteur de l'écran ABZmon. Ne jamais laisser cette valeur inférieure à 240. Certaines boîtes de dialogue ont cette hauteur.

Display X: la coordonnée X du coin droit supérieur de l'écran ABZmon. Une valeur 20 indique que le coin droit supérieur de l'écran ABZmon est à 20 pixels du coté gauche du moniteur vidéo.

Display Y: la coordonnée Y du coin droit supérieur r de l'écran ABZmon. Une valeur 20 indique que le coin droit supérieur de l'écran ABZmon est à 20 pixels du haut du moniteur vidéo.

Maximum number of window: le nombre maximum de fenêtres qui peuvent être ouvertes simultanément.

Local label buffer size: la taille du tampon où sont emmagasinées les étiquettes locales et leurs adresses.

Compiler symbol buffer size: la taille du tampon où sont emmagasinées les étiquettes générées par un compilateur et leurs adresses.

Use big font in Menu window: si 0, les menus utilisent des petits caractères et si -1, des caractères plus visibles.

Use big font in Message window: si 0, la fenêtres des messages utilise des petits caractères et si -1, des caractères plus visibles.

Use wildcard in Search: les recherches (du menu **SEARCH**) se font en utilisant un caractère de remplacement si la variable est -1; si la variable est 0, ABZmon n'utilise pas de caractère de remplacement.

Wildcard symbol in Search: le caractère de remplacement dans les recherches (du menu **SEARCH**).

Case sensitive in Search: les recherches (du menu **SEARCH**) se font en ne tenant pas compte de la différence entre minuscule et masjuscule si la variable est 0; si la variable est -1, ABZmon tient compte de la différence.

Hide/Show cursor: le curseur QuickDraw est manipulé durant les interruptions. Il va arriver quelquefois que ce curseur change de forme pendant que ABZmon prend le contrôle de l'écran. Au retour, il y aura conflit entre l'image du curseur conservée et la nouvelle image du curseur. Le résultat est un mélange disgracieux à l'écran, à l'endroit du curseur. Si le bit est à 1, ABZmon corrige automatiquement. Si vous travaillez sur des routines affectant le curseur, laissez ce bit à 0. ABZmon n'interviendra plus pour faire la correction.

Mouse coupled: quant la variable **CrsrCouple** ($8CF) est non nulle, le curseur du Mac bouge en même temps que la souris. Puisque ABZmon utilise son propre curseur, la souris n'a pas besoin d'être synchronisée au curseur du Mac. Si le bit est mis à 1, le débogueur annulle la variable **CrsrCouple** quant l'écran ABZmon apparaît et remet **CrsrCouple** à -1 au retour. De cette façon le curseur du Mac n'est pas affecté. Vous pouvez mettre le bit à 0 quant le bit **Hide/Show cursor** est 1.

Report OS event: une autre source potentielle de problème provient du systême d'opération qui cherche à redessiner la souris ou une fenêtre quant l'écran ABZmon apparaît. Une façon de circonvenir le probème est de mettre à 0 la variable **SysEvtMask** ($144). Aucun évènement du sysême n'est alors rapporté. Quant le bit est à 1, le débogueur annulle **SysEvtMask** lorsque l'écran ABZmon apparaît et restitue à **SysEvtMask** sa valeur quant il retourne. Si le bit est à 0, le débogueur ne change pas la valeur de la variable.

File type: la fenêtre de sélection de fichier affiche toujours les fichiers TEXT. Entrez un autre type à cette variable pour faire afficher en plus ce type de fichier. Si cette variable est nulle (aucun type) tous les types de fichiers seront affichés. Attention: ResEdit place 4 caractères blancs quant ce champ est vide. Pour entrez la valeur zero vous devez utiliser l'éditeur hexadécimal.

Les variables suivantes déterminent l'apparence des fenêtres de code.

Code origin in Disassembly: on additionne cette valeur à toutes les reférences (directes ou indirectes) au compteur de programme qui sont données sous forme d'adresses hexadécimales.

Tabulation length in Disassembly: les différents champs dans une ligne de code désassemblée sont séparés par des tabulations de longueur donnée par cette variable.

Target CPU in Disassembly: fixez cette variable en accord avec le type de microprocesseur que vous utilisez: 0 -> 68000, 20 -> 68020, 30 -> 68030, 40 -> 68040 (peut être changé au besoin pour examiner du code s'adressant à d'autres microprocesseurs). Si cette variable est à -1, ABZmon choisit le microprocesseur de l'ordinateur.

Target MMU in Disassembly: fixez cette variable en accord avec le type de coprocesseur (pour gestion de mémoire) que vous utilisez: 51 -> 68851, 30 -> 68030, 40 -> 68040, 0 -> non utilisé (peut être changé au besoin pour examiner du code s'adressant à d'autres microprocesseurs). Si cette variable est à -1, ABZmon choisit le coprocesseur de l'ordinateur.

Target FPU in Disassembly: fixez cette variable en accord avec le type de coprocesseur arithmétique que vous utilisez: 81 -> 68881, 82 -> 68882, 40 -> 68040, 0 -> non utilisé (peut être changé au besoin pour examiner du code s'adressant à d'autres microprocesseurs) Si cette variable est à -1, ABZmon choisit le coprocesseur de l'ordinateur.

MC68020 addressing form: Motorola définit deux formes d'opérandes: 0 -> (1234,pc), -1 -> 1234(pc).

Default displacement form: détermine la forme (par défaut) de déplacement d'une opérande: 0 -> (1234,pc), 1 -> *+32, 2 -> étiquette_locale+22, 3 -> étiquette_compilateur+22.

Hexa address before Disassembly: la ligne débute par l'adresse hexadécimale de l'instruction si cette variable est à -1 (0 pas d'adresse).

Use low memorw: si à -1, les variables globales du Mac sont remplacées par les noms définis par Apple (0 si représentées seulement la valeur hexadécimale).

Append size in absolute operand: une opérande en mode absolue a deux tailles possibles: mot ou mot long. Si la variable est à -1 on ajoute **.w** ou **.l** à l'opérande selon le cas (on ajoute rien si 0).

instruction hexa. code: on ajoute le code hexadécimal de l'instruction, entre accolades, à la fin de la ligne si ce bit est 1.

displacement -> addr. in comment: on ajoute l'adresse du déplacement relatif (ou du branchement) de l'opérande, entre < >, à la fin de la ligne si ce bit est 1.

immédiat: on ajoute la représentation asccII d'une opérande "immédiate", entre apostrophes, à la fin de la ligne si ce bit est 1.

2 blank before label/symbol: on ajoute deux espaces blancs avant les étiquettes, en début de ligne, pour rendre le code plus lisible, si cette variable est -1 (pas de blancs pour 0).

SYMBOL variable length: on utilise les étiquettes de type "longueur variable" du compilateur si ce bit est 1.

SYMBOL fixed 8 byte: on utilise les étiquettes de type "longueur fixe 8 octets" du compilateur si ce bit est 1.

SYMBOL fixed 16 byte: on utilise les étiquettes de type "longueur fixe 16 octets" du compilateur si ce bit est 1.

Heap Block Size (24/32 bits): si 0, on laisse le "débogueur" déterminer la dimension des blocs mémoires (application ou système). On peut cependant forcer ABZmon à utiliser une des deux dimensions 24 ou 32 bits en donnant une de ces deux valeurs à la variable.

Build Stack Frame: si cette variable est -1, ABZmon construit une nouvelle pile d'exception pour le retour au programme. Si la variable est nulle on se sert de la pile existante. A changer en cas de problème.

Walk in LineA: si cette variable est -1, la commande **ONE STEP** du menu **STEP** entre dans la routine d'exception d'une instruction LineA (OS ou ToolBox). Si 0, la routine d'exception est exécutée comme une seule instruction.

Trace mode: détermine le type de suivi pas à pas:

valeur 0: le mode **trace** utilisé est celui qui exécute toutes les instructions qui ne changent pas la valeur séquentielle du compteur de programme en une seule étape. Les instructions telles JMP, BRA ou RTS changent la valeur séquentielle du PC. Par conséquent, ce mode fait seulement le suivi d'une instruction de ce type, à une autre du même type.

valeur 1: lorsque ABZmon rencontre une instruction **BSR**, **JSR ou lineA**, il compte le nombre d'instructions dans la routine et affiche ce nombre dans la fenêtre des messages.

valeur 2: on place un point d'arrêt après chaque instruction à être exécutée. ABZmon apparaît donc après l'exécution et on a ainsi un suivi pas à pas.

Skip non-used debug: si une des trois instructions "debug" ci-dessous n'est pas utilisée par ABZmon (bit à 1 pour être utilisée) alors ABZmon ignore cette instruction si le bit est 1 et passe le contrôle à la routine d'exception du Mac si le bit est 0.

_DebugNum: on utilise cette instruction si le bit est 1 (voir Pour entrer dans ABZmon).

_DebugStr: on utilise cette instruction si le bit est 1 (voir Pour entrer dans ABZmon).

_Debugger: on utilise cette instruction si le bit est 1 (voir Pour entrer dans ABZmon).

TRAPcc message: affiche un message de prise d'exception selon la condition si le bit est à 1 et n'affiche rien si le bit est 0.

Scc message: affiche un message d'état de l'octet selon la condition si le bit est à 1 et n'affiche rien si le bit est 0.

DBcc message: affiche un message de répétition de boucle selon la condition si le bit est à 1 et n'affiche rien si le bit est 0.

TBcc message: affiche un message de branchement selon la condition si le bit est à 1 et n'affiche rien si le bit est 0.

Force CPU: si le CPU au moment du chargement de ABZmon n'est pas le même que le CPU après le démarrage, en utilisant une carte accélératrice par exemple, on peut forcer ABZmon à reconnaître le nouveau CPU en donnant les deux derniers chiffres du CPU à cette variable: 00 -> 68000, 20 -> 68020, 30 -> 68030, 40 -> 68040

Force FPU: si le FPU au moment du chargement de ABZmon n'est pas le même que le FPU après le démarrage, par l'ajout d'un coprocesseur mathématique par exemple, on peut forcer ABZmon à reconnaître le nouveau FPU en donnant les deux derniers chiffres du FPU à cette variable: 81 -> 68881, 82 -> 68882, 40 -> 68040, -1 -> pas de changement.

Force MMU: si le MMU au moment du chargement de ABZmon n'est pas le même que le MMU après le démarage, en utilisant une carte accélératrice par exemple, on peut forcer ABZmon à reconnaître le nouveau MMU en donnant les deux derniers chiffres du MMU à cette variable: 51 -> 68851, 30 -> 68030, 40 -> 68040, -1 -> pas de changement.

Broke Line: dans une fenêtre de texte, la ligne trop longue pour être contenue dans la fenêtre est brisée et se continue plus bas si la variable est -1. Si la variable est 0 le reste de la ligne n'est pas visualisée.

One page: dans une fenêtre de texte, le déroulement se fait une page à la fois (grandeur d'une fenêtre) si la variable est -1 et une ligne à la fois si la variable est 0.

68040 MEM Protect: cette option n'est que pour les ordinateurs 68040 (Quadra, Centris,...). Le code du débogueur est protéger contre toute tentative d'écriture (accidentelle) quant cette variable est -1. Si un programme essaie d'écrire dans la zone mémoire occupée par le débogueur, une erreur d'accès est générée et vous voyez apparaître ABZmon. Vous savez alors que le programme courrant est fautif. Cette option requiert environ 24K de mémoire supplémentaire.

Le reste des variables sont les vecteurs d'exception du microprocesseur. Les valeurs associées déterminent le type d'action que ABZmon doit faire au démarrage pour ces vecteurs. Le type d'action est déterminé par les 5 bits faibles de la valeur de la variable:

bit 0 --> le vecteur est changé dans la banque des vecteurs partant de 0
bit 1 --> le vecteur est changé dans la banque des vecteurs partant de VBR
bit 2 --> le vecteur est changé une seul fois, au démarrage
bit 3 --> le vecteur est changé chaque fois que ABZmon apparaît

bit 4 --> le vecteur est changé à chaque interruption (chaque 1/60 seconde)

Par exemple, pour changer un vecteur dans les deux banques de vecteurs une seule fois, au démarrage, on donne à la variable la valeur 7.

DebugNum

Cette procédure fonctionne comme Debugger ou DebugStr, mais plutôt que d'interrompre le programme et de forcer l'entrée dans le "débogueur" juste après cette instruction dans le code, DebugNum demande l'interruption seulement après un certain nombre de passage à cette instruction. DebugNum requiert deux paramètres. Le premier, appellé '**count**', est un entier long sur quatre octets. Si **count=0**, l'arrêt se fait à la première rencontre de DebugNum. Si **count=1**, la première rencontre de DebugNum est ignorée l'arrêt se fait à la seconde rencontre, etc. L'autre paramètre, appellé '**Number'**, est un entier sur deux octets, entre 0 et 9, qui identifie le numéro du DebugNum. Vous disposez donc de dix DebugNum différents.

Le code d'appel de cette procédure pour le langage assembleur est

```
move.l      #70,-(sp)       ; count
move.w      #4,-(sp)        ; DebugNum #
dc.w        $AAFF
```

ou, si vous utilisez l'assembleur MPW

```
move.l      #70,-(sp)       ; count
move.w      #4,-(sp)        ; DebugNum #
_DebugNum
```

en ayant en en-tête de votre code la ligne suivante

```
_DebugNum     opword     $AAFF
```

Pour le langage C de MPW vous avez l'en-tête suivante

```
void        DebugNum(count,number) long number; integer count; extern 0xAAFF;
```

et le code d'appel de la procédure est

```
DebugNum(70L,4);
```

Ré-initialisation de DebugNum

ABZmon conserve en mémoire dix octets drapeau et dix entiers longs, associés à chacun des dix DebugNum. Quant le "débogueur" rencontre un DebugNum pour la première fois, il met l'octet drapeau à 1 et met l'entier long à la valeur de '**count**'. Si **count=0,** ABZmon interrompt le programme, sinon la valeur de l'entier long est diminué de 1. A chaque rencontre de DebugNum, ABZmon vérifie l'entier long. Si l'entier long est 0, ABZmon interrompt le programme (et remet le drapeau à 0), sinon la valeur de l'entier long est diminué de 1.

Supposons que votre programme se termine avant que l'entier long soit 0. La prochaine fois que vous utiliserez DebugNum (pour le même '**number**'), après recompilation de votre programme par exemple, ABZmon va ré-initialiser l'entier long , avec la valeur '**count**' seulement si l'octet drapeau est à 0. Vous pouvez remettre à zéro un ou tous les octets drapeau à l'aide de la commande **CLEAR DEBUGNUM** dans le menu **STOP**. Assurez vous que chaque fois que vous utiliserez DebugNum, l'octet drapeau soit à 0.

Rapport d'évaluation:

ABZmon à été testé sur Mac SE, SE-30, LC, LC II, LC III, Si et Quadra.
Si vous avez des difficultés à installer le débogueur ou si vous rencontrez certaines difficultés de fonctionnement, svp, laissez un message décrivant les difficultés rencontrées (et le type de matériel utilisé) sur le réseau CompuServe:

[72467,2770]

ou écriver moi à:

Alain Birtz
650 Grand St-Charles,
St-Paul d'Abbotsford
P.Q., Canada, J0E-1A0

Remerciements:

à Hélène Préfontaine, pour son remarquable travail sur la documentation
à Clément Ross, pour ses nombreuses suggestions techniques