DevpacBasic

Didier Levet Launay

COLLABORATORS			
	TITLE : DevpacBasic		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Didier Levet Launay	August 19, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

DevpacBasic

1.1 Simulation Basic

```
MACRO-INSTRUCTIONS PERMETTANT L'ÉMULATION DU BASIC


 Avant propos

 REPEAT...UNTIL
  Répétition jusqu'à l'arrivé de l'évènement attendu.

 WHILE...WEND
  Répétion tant que l'évènement se produit.

 FOR ...NEXT
  Boucles un certain nombre de fois.

 SELECT...CASE
  Choix conditionnel multiples.

 BREAK
  Sortie de boucles.

 _IF
  Choix conditionnel alternatif.


 PROTO
  Prototypes et appels de fonctions.

 FUNCTION
  Définition de fonctions.


 WBSTARTUP
  Démarrage depuis le Workbench.
```

1.2 Avant propos

OBJECTIFS

Ces macro-instructions n'ont pas pour but de transformer HiSoft Devpac en un véritable langage Basic. Leur principal intérêt est de faciliter l'écriture d'un source et de démontrer par la même occasion la puissance du macro-assembleur. Nous vous encourrageons vivement à persévérer dans cette voie mais attention, la mise au point de telles macros n'est pas à la portée d'un débutant.

AVANTAGES

L'utilisation de ces macros devrait faciliter l'approche du langage assembleur :
- la structure du programme apparaît plus clairement;
- le fichier source est mieux structuré;
- les redondances entre les macros évitent certaines erreurs.

LIMITATION

Les macro-instructions sont très puissantes. Leur syntaxe doit par conséquent obéir strictement à quelques règles bien précises :
- Faites attention à l'extension (.b, .w ou .l) après le nom des macros lorsqu'elles acceptent des arguments.
- Placez les arguments entre crochets (< et >) lorsqu'ils contiennent des espaces.
- Respectez les règles d'écriture des modes d'adressage.

Enfin, nous vous recommendons de supprimer l'avertissement concernant l'optimisation des branchements en avant afin d'éviter l'apparition de ces messages, dus aux macros, pendant l'assemblage.

1.3 REPEAT/UNTIL

SYNTAXE

```
REPEAT
UNTIL.x Argument[,Code de condition[,Valeur]]
FOREVER
```

EXPLICATION

Les instructions assembleur encadrées par ces deux macros sont exécutées jusqu'à ce que la condition indiquée par UNTIL soit vraie.
La condition est testée en fin de boucle. En conséquence, les instructions composant la boucle sont exécutées au moins une fois.

Vous pouvez utiliser FOREVER à la place de UNTIL. Dans ce cas la boucle ne pourra être quittée qu'en utilisant la macro

Break
.

REMARQUE

Voir
les codes Conditions
pour une description
détaillée des arguments supportés par la macro-instruction UNTIL.

1.4 WHILE/WEND

SYNTAXE

WHILE.x Argument[,Code de condition[,Valeur]]
WEND

EXPLICATION

Les instructions assembleur encadrées par ces deux macros
sont exécutées tant que la condition indiquée par WHILE
est vraie.
À l'inverse de
REPEAT/UNTIL
, la condition est testée
au début de la boucle.

REMARQUE

Voir
les code de Conditions
pour une description
détaillée des arguments supportés par la macro-instruction WHILE.

1.5 FOR/NEXT

SYNTAXE

FOR.x Registre,Valeur initiale,Valeur finale[,Pas[,DOWN]]
NEXT

EXPLICATION

La valeur initiale est chargée dans le registre indiqué, qui doit
être un registre de donnée (d0 à d7). Ce registre joue le rôle de
compteur de boucle. Chaque fois que la macro NEXT est rencontrée,
le pas est ajouté au registre de donnée, puis ce registre est
comparé à la valeur finale.
L'exécution du programme reprend au début de la boucle tant que
le compteur est inférieure ou égal à la valeur finale (supérieur
ou égal si le pas est négatif).
L'argument DOWN indique que le pas est négatif, et doit être
écrit en majuscules. La valeur par défaut pour le pas est de 1.

La syntaxe des valeurs de début et de fin de boucle est identique

à la syntaxe des modes d'adressage.

EXEMPLES

```
FOR.w d0,#0,#10     ; FOR D0=0 TO 10
FOR.b d7,start(pc),#0,#4  ; FOR D7=start(pc) TO 0 STEP 4
FOR.l d3,debut,fin,#-1,DOWN ; FOR D3=debut TO fin STEP -1
```

REMARQUE

La valeur du compteur de boucle est transférée dans le registre de donnée à chaque passage, avant la première instruction de la boucle. Il est tout à fait possible d'utiliser le registre de donnée à d'autres fins, sans modifier pour autant le compteur de boucle. Par contre, le registre A7 ne DOIT PAS être modifié, ou DOIT être restauré avant la macro NEXT de fin de boucle.

1.6 SELECT/CASE/ENDSELECT

SYNTAXE

```
SELECT Registre
CASE.x Valeur1[,TO[,Valeur2]
CASE.x ,TO,Valeur2
[DEFAULT]
ENDSELECT
```

EXPLICATION

L'argument de SELECT doit être un registre de donnée (d0 à d7). La valeur contenue dans ce registre est examinée par le premier CASE. Les instructions suivant la macro CASE sont exécutées lorsque la condition est remplie, c'est à dire lorsque la valeur du registre est égale à Valeur1, ou fait partie de l'intervalle Valeur1-Valeur2. Si cette condition est fausse, l'examen du registre se poursuit avec l'instruction CASE suivante. Les instructions assembleur situées après DEFAULT son exécutées lorsqu'aucune des condition n'à été réalisée.

EXEMPLE

```
SELECT  d0
CASE.b  Code1(pc) ; Si d0=Code1(pc)
CASE.b  ,TO,#31   ; Si -128<=d0<=31v
      ; Attention à la 1ère virgule !
CASE.b  #"a",TO,"z" ; si "a"<=d0<="z"
CASE.b  #0,TO   ; Si 0<=d0<=127v
```

REMARQUE

Par défaut, aucun branchement n'est effectué vers ENDSELECT à la fin d'un bloc CASE. Si ce mode de fonctionnement n'est pas souhaité, utilisez la macro

```
BREAK
pour terminer
```

les blocs CASE.

La syntaxe des valeurs définissant l'intervalle est identique
à la syntaxe des modes d'adressage.

1.7 BREAK

SYNTAXE

BREAK

EXPLICATION

Cette macro vous permet de quitter prématurément une boucle
FOR/NEXT, REPEAT/UNTIL ou WHILE/WEND. Elle permet en outre
de terminer l'exécution d'un bloc CASE.

Lorsqu'elle est utilisée pour quitter une boucle, elle doit
être écrite dans un bloc IF/ENDIF.

EXEMPLES

```
FOR d0,#0,#10  ; FOR D0=0 TO 10   | SELECT  d0
  ...          | CASE  #1
  _IF d7  ; Si d7 non nul.  |   ...
    BREAK      |   BREAK
  _ENDIF         | CASE  #2,TO,#5
  ...          |   ...
NEXT             | ENDSELECT
```

1.8 PROTO/GOSUB

SYNTAXE

PROTO NomDeFonction,Type Argument 1, Type Argument 2, ...
GOSUB NomDeFonction,Arg1,Arg2,...

EXPLICATION

PROTO permet de définir le prototype d'une fonction, et doit
être utilisé avant toute référence à cette fonction. Le type
des arguments peut être choisi parmis les types de base, à
savoir BYTE, UBYTE, WORD, UWORD, LONG, ULONG, CPTR, etc. La
taille des arguments (définie par leur type) ne peut excéder
4 octets. Toute référence à la fonction ainsi prototypée
devra compter le même nombre d'arguments.

GOSUB permet d'appeler la fonction indiquée. Le prototype de
cette fonction doit avoir été défini au préalable, et le
nombre d'arguments doit être conforme. Le passage des arguments
à la fonction est géré automatiquement.

1.9 FUNCTION/ENDFUNC

```
                SYNTAXE

FUNCTION NomDeFonction, Liste de registres, Nom arg1, Nom arg2, ...
ENDFUNC

EXPLICATION

FUNCTION défini le point d'entrée d'une fonction (sous-programme),
et prend en charge la gestion des arguments. La fonction doit avoir
été prototypée au préalable. (voir
                 PROTO/GOSUB
                )
La liste de registres correspond aux registres que vous voulez
sauvegarder sur la pile afin de retrouver leur contenu à la fin
de la fonction. La syntaxe est identique à celle utilisée par
l'instruction 'movem'.
Les arguments suivants sont les noms que vous voulez attribuer
aux différents paramètres de la fonction.

Lorsque ENDFUNC est atteint, la liste des registres est
récupérée sur la pile, puis les arguments sont retirés de cette
même pile. La valeur retournée par la fonction, si elle existe,
peut être transmise dans un registre, à condition qu'il ne fasse
pas partie de la liste des registres sauvegardés. On utilise
généralement le registre d0.

EXEMPLE

PROTO MaFonction,ULONG,ULONG,BOOLW
...
GOSUB MaFonction,cpt(a0),#1000,FALSE
...
FUNCTION  MaFonction,d2-d5/a3-a4,Compteur,Limite,Flag
move.l  Compteur(sp),d0
move.l  Limite(sp),d1
move.w  Flag(sp),d7
...
ENDFUNC

REMARQUE

La macro ENDFUNC génère le rts final.
```

1.10 _IF/_ELSE/_ELSEIF/_ENDIF

```
                SYNTAXE

_IF.x Argument[,Code de confition[,Valeur]]
_ELSE
_ELSEIF.x Argument[,Code de confition[,Valeur]]
_ENDIF
```

```
EXPLICATION

L'existance dans Devpac de directives utilisant des noms identiques
nécessite la présence du trait de soulignement devant le nom de ces
macros.

La condition définie par _IF est testée. Si cette condition
est vraie, les instructions situées juste après _IF sont exécutées.
Lorsqu'elle est fausse, les instructions situées après _ELSE ou _ELSEIF
sont exécutées. S'il n'existe pas de branche _ELSE ou _ELSEIF,
une condition fausse entraîne l'exécution des instructions placées
après _ENDIF.

EXEMPLES

STYLE CLASSIQUE   AVEC _ELSEIF

_IF d0      _IF d0
  ...     ...
_ELSE     _ELSEIF d1
  _IF d1      ...
    ... _ENDIF
  _ENDIF
_ENDIF

REMARQUE

Voir
                les code de Conditions
                pour une description
détaillée des arguments supportés par les macro-instructions _IF
et _ELSEIF.
```

1.11 WBSTARTUP

```
SYNTAXE

WBSTARTUP

EXPLICATION

Cette macro est destinée à remplacer le fichier easystart.i
Elle vous permet de démarrer votre programme depuis le
Workbench ou depuis le CLI, et d'obtenir les arguments
correspondant. Pour les puristes, l'usage de cette macro
n'interdit pas la création de code ré-entrant.

Vous obtenez ces arguments dans les registres d0 et a0.
Le registre d0 vaut -1 lorsque le programme est lancé
depuis le Workbench, auquel cas le registre a0 est
initialisé avec le pointeur vers le message envoyé par
le Workbench.

REMARQUE
```

L'utilisation de cette macro interdit le fonctionnement
de programme sur les machines équipées du système 1.3 ou
inférieur.

1.12 Conditions

Argument[,Code de condition[,Valeur]]

L'argument indique l'objet de la comparaison. Lorsque
la valeur de comparaison n'est pas indiquée, la valeur
"0" est prise par défaut, et l'argument peut être un des
modes d'adressage supporté par l'instruction "tst".

Lorsque la valeur de comparaison est indiquée, la syntaxe
de cette valeur et la syntaxe de l'argument doivent
correspondre à une des combinaison de modes d'adressage
de l'instruction "cmp". En effet, la comparaison est
effectuée par "cmp Valeur,Argument".

Les codes de condition peuvent être les suivants :
NE : Différent
EQ : Egal
GT : Strictement supérieur
GE : Supérieur ou égal.
LT : Strictement inférieur
LE : Inférieur ou égal
= : Egal
!= : Différent.