Débutant

Pierre Philippe Launay

COLLABORATORS

	TITLE : Débutant	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Pierre Philippe Launay	August 19, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Débutant

1.1 Les rudiments de HiSoft Devpac

```
                LES PREMIERS PAS

Un langage d'assemblage est un
                langage
                 non
                structuré
                 très proche de la

                machine
                . Les
                instructions
                 utilisées agissent directement sur le
                processeur
                alors que pour les autres langages chaque
                fonction
                 est en fait une
                suite
                d'instructions. On touche ici la qualité et le défaut d'une telle ↩
                   puissance.

LE PREMIER SOURCE : COMMENT ÉCRIRE UN PROGRAMME?

Il suffit de se jeter à l'eau.

Pour commencer il faut suivre et comprendre les
                exemples
                 du tiroir
"Exemples".

Le principe de création d'un programme est toujours le même. Dans un
premier temps il faut étudier ce que l'on désire faire, en réaliser un plan
puis déterminer les différentes
                actions
                 du programme.

Bien. Il convient ensuite d'entrer davatantage dans les détails de chaque
séquence. Au début, pour vous aider à comprendre, il faut regarder les
divers programmes que vous trouverez un peu partout.
```

On veut ouvrir un écran et une fenêtre. Il existe pour cela des fonctions spécialisées du système. Elles sont présentes dans intuition.library et dans graphics.library. Vous apprendrez rapidement à savoir utiliser ces diverses fonctions. Il nous faut donc
ouvrir
ces bibliothèques.

Les préliminaires étant terminés il ne reste plus qu'à appeler la fonction d'ouverture des bibliothèques. CALLEXEC OpenLibrary.

Cette fonction peut très bien échouer. Dans ce cas la valeur contenue dans le registre D0 sera nulle. Le test est réalisé par l'instruction
tst
soit
ici TST.l d0.

Il est maintenant temps de passer aux choses sérieuses...

LA STRUCTURATION

Revenons sur ce point assez important : il est tout à fait possible en assembleur d'écrire vos instructions les unes à la suite des autres sans aucun regroupement logique. Vous perdrez alors beaucoup de temps à la mise au point. Nous vous conseillons d'utiliser des blocs à la façon des autres langages :
- Do While = Étiquette » Action » Branchement conditionnel sur l'étiquette.
- For Next = Étiquette » Test » Sous routine d'action » Branchement.
- If else endif = Test » Action » Saut au Endif.
- appel aux procédures = Saut et branchements aux sous routines.
- indentation.

LES MACRO-INSTRUCTIONS

Plutôt que de devoir tout réinventer, il est plus judicieux d'écrire un nom d'instruction, par exemple "SETCOLOR 4,R,V,B" pour changer le crayon 4 de la palette avec les valeurs Rouge, Verte et Bleue plutôt que de réécrire la fonction à chaque fois. En
appelant
cette
macro
-instruction, l'assembleur
remplacera automatiquement la macro par le
code
correspondant quand il

assemblera
.

LE NOM DES ÉTIQUETTES

Une étiquette est une marque, un point de repère, que l'on écrit dans un source. Un peu comme une annotation sur une feuille de papier.

Il ne faut pas utiliser de termes ambigus ou des abréviations. Les noms à rallonge sont également à bannir. Une bonne méthode consiste à utiliser toujours les mêmes règles de création d'un nom. Nous conseillons

par exemple les suffixes "_txt" pour des chaînes de caractères explicatifs, "_help" pour les routines d'aide. Et nous créons les noms avec un préfixe indiquant leur origine comme "CalculatriceOK", "CalculatriceCancel",...

LES FICHIERS D'INCLUSION

Ce sont des fichiers standards regroupés par thèmes. Ils comportent des noms standardisés pour les valeurs des fonctions et pour la forme des différents objets utilisés.

Bonjour et bienvenue dans notre monde, le monde fantastique de la programmation et du développement.

1.2 actions

L'ANALYSE

helloworld devra afficher un message sibyllin sur l'écran.

En analysant plus en détails, cela signifie que l'on doit réaliser les séquences suivantes :
- Création d'un écran et ouverture.
- Création d'une fenêtre liée à l'écran et ouverture.
- Affichage d'un texte sur la fenêtre.
- Attente de la décision de l'utilisateur (déplacer la fenêtre, modifier sa taille, ou la fermer).
- Fermeture de la fenêtre.
- Fermeture de l'écran.

1.3 appelant

LES APPELS

On dit qu'on appelle une fonction quand on l'utilise dans le programme.

Ce qui suit est un exemple d'appel de la macro Circle :

```
CIRCLE  RastPort,rayon,CentreX,CentreY
```

1.4 assemblera

L'ASSEMBLAGE

L'assemblage est l'opération qui transforme le texte de votre programme en une suite d'ordres simples compréhensibles par le processeur. Le résultat en est une action très rapide et totalement indépendante du langage qui l'a créée.

1.5 code

LE LISTAGE

Un code est le texte, le scénario, des commandes compréhensible par
l' utilisateur et que comprend un logiciel (un langage) servant
d'intermédiaire entre l'utilisateur et l'ordinateur.

Ce qui suit est un exemple de code :

```
WBStartup
DefaultValue
ScreenOpenTag 1,10,10,640,256,Hires
WindowOpenTag 3,15,15,200,100
Title 3,20,20,<La force créatrice en action.>
```

1.6 exemples

LES PROGRAMMES DE PRISE EN MAIN DU MANUEL

- le source demo.s ("Démonstration") est expliqué dans le livre et à
  pour ambition de vous aider à la prise en main de HiSoft Devpac 3.50.
- le source helloworld.s ("Allo, allo,... y'a quelqu'un?") vous
  montre comment ouvrir une fenêtre sur l'écran. Avec ce source vous verrez
  combien, malgré une fausse apparence de difficulté, la programmation en
  assembleur est en fin de compte assez aisée. Par la suite, quand vous aurez
  compris comment utiliser les macros, l'écriture du même programme sera
  encore plus rapide. Il vous faudra tout d'abord comprendre comment corriger
  l'erreur portant sur le texte affiché.
- le source freemem.s ("mémoire libre") vous montre un autre aspect de
  la programmation.

1.7 fonction

LE REGROUPEMENT DES ORDRES SIMPLES

Une fonction est une séquence plus ou moins fréquente d'actions simples
et répétitives. Par exemple, le chargement d'une image ou bien
l'affichage d'un texte sont généralement écrites une bonne fois pour
toutes. Il suffira alors d'appeler les fonctions correspondantes pour
exécuter toute la séquence prévue. En assembleur on préférera parler
de programme ou de routine plutôt que de fonctions car l'assembleur
n'est pas un langage structuré à la différence du C ou du Pascal.

Ce qui suit est un exemple de fonction :

```
LoadIff <mon image>
```

1.8 instructions

L'ORDRE

Une instruction est l'élément de base qui sert à créer des programmes.

Ce qui suit est un exemple d'instruction :

```
MOVE.l A0,A2
```

1.9 langage

LE DIALOGUE

Un langage est un système de communication entre deux entités. Ici l'homme et la machine. Autrement dit, entre le vivant et l'inerte... ou presque!

Notez aussi qu'on préfère souvent dire "langage assembleur", voire carrément "assembleur", plutôt que le véritable terme de "langage d'assemblage".

1.10 machine

LES ORDINATEURS

Une machine est un objet pourvu d'un mécanisme permettant de réaliser une action. Un ordinateur est une machine ayant pour objet de traiter les informations.

1.11 macro

LE GAIN DE TEMPS

Une macro est une sorte de raccourci à l'écriture. Les macros sont définies au début du programme puis, par la suite, il suffit de les nommer pour que l'assembleur remplace automatiquement le nom de la macro par son contenu lors de l'opération d'assemblage.

Ce qui suit est un exemple d'une macro CIRCLE qui appele elle même une autre macro ELLIPSE. On peut ainsi réaliser des "routines" extrêmement sophistiquées.

Le source devient alors beaucoup plus clair et concis, à condition toutefois de regrouper toutes les macros dans un fichier séparé. Une simple ligne d'appel du fichier correspondant permettra un énorme gain de temps à l'écriture et une plus grande facilité de relecture et de débogage (voir plus loin la cellule code.

```
CIRCLE  MACRO     * RastPort, rayon, Centre X, Centre Y
  ELLIPSE \1,\2,\2,\3,\4
  ENDM
```

1.12 ouvrir

```
L'UTILISATION DES FONCTIONS DU SYSTÈME D'EXPLOITATION

De nombreuses fonctions sont disponibles dans différentes bibliothèques
spécialisées. Le principe d'ouverture d'une bibliothèque est toujours le même :
- On remplit les registres spécialisés avec des valeurs bien spécifiques,
  (toujours les mêmes).
- On appelle la fonction du système d'exploitation permettant l'ouverture
  d'une bibliothèque. C'est la fonction OpenLibrary.
- Une fois la fonction d'ouverture exécutée, elle retournera dans le
  registre D0 une valeur indiquant à quel endroit de la mémoire se trouve
  le début de la bibliothèque en question.

Pour remplir un registre, on utilise :
- soit l'instruction MOVE qui recopie (déplace) une valeur précise (par
  exemple la valeur #INTUITION_REV) dans le registre. MOVEQ #INTUITION_REV,d0.
- soit l'instruction LEA qui charge l'adresse d'une "case mémoire" (par
  exemple la case mémoire int_name(pc) qui contient le texte "intuition.library")
  dans le registre d'adresses donné. LEA int_name(pc),a1.


* OUVERTURE DE LA BIBLIOTHÈQUE intuition.library
  MOVEQ   #INTUITION_REV,d0 version
  LEA   int_name(pc),a1
  CALLEXEC  OpenLibrary
  TST.l   d0      Test d'ouverture d'Intuition :
  BEQ   exit_false    si ça échoue on s'en va
  MOVE.l    d0,_IntuitionBase sinon sauvegarde du pointeur
```

1.13 processeur

```
LES PROCESSEURS

Un processeur est l'unité de gestion des informations dans un
ordinateur :
- Certains processeurs sont des généralistes comme les processeurs
  68000, 68010, 68020, 68030, 68040 et 68060.
- D'autres processeurs sont spécialisés dans la gestion de certaines
  mémoires comme le co-processeur 68851.
- D'autres encore sont des processeurs spécialisés dans les calculs
  comme les co-processeurs 68881 et 68882.
```

1.14 suite

```
Une suite est une séquence, un enchainement.

(-:
```

1.15 structuré

LA STRUCTURATION

Un langage est dit structuré quand il réalise des blocs d'instructions. Chaque bloc peut être imbriqué dans d'autres blocs comme l'oeuf dans sa coquille. Avec un langage structuré on ne peut pas accéder à un niveau d'emboitement sans passer par les niveaux supérieurs.

Un langage d'assemblage ne connait pas toutes ces contraintes. On peut tout à fait écrire les instructions les unes à la suite des autres sans s'imposer de règles rigoureuses. Pour une bonne compréhension des sources il est cependant primordial de respecter au maximum la philosphie des langages structurés.

Ce qui suit est un exemple de programme écrit avec un langage structuré :

```
WHILE condition
  REPEAT
    ...
    action
    ...
  UNTIL résultat valable
WEND
```

On ne peut pas accéder à l'action avant d'avoir accédé au bloc REPEAT...UNTIL. On ne peut pas accéder au bloc REPEAT...UNTIL avant d'avoir accédé au bloc WHILE...WEND.

En respectant ce type de structuration dans vos programmes, vous aurez immédiatement une meilleure perception du déroulement du codage.

1.16 tst

L'ADAPTATION AVEC CHAQUE SITUATION

LE TEST DE RETOUR

Si la valeur D0 est nulle alors il faut sortir du programme et rediriger d'urgence la suite de l'exécution du source vers la porte de sortie. Ce branchement est réalisé par l'instruction BEQ (branchement si la valeur testée est nulle) à l'endroit indiqué, ici exit_false. BEQ exit_false.

Dans le cas contraire, on récupère la valeur retournée en D0 et on la recopie là où on saura la récupérer à l'envie, ici une case mémoire créée par nous et qui s'appelle _IntuitionBase. MOVE.l d0,_IntuitionBase.

LA LONGUEUR DE LA CIBLE

Oui mais qu'est-ce donc que ces "q" et autres ".l"? Ce sont des suffixes qui précisent si l'instruction doit utiliser la forme rapide (q) ou agit sur des mots longs (l) (en réalité MOVEQ est une instruction à part entière légèrement différente de l'instruction MOVE). La plupart des instructions possèdent ainsi plusieurs variantes. On distingue ainsi les suffixes

indiquant à l'instruction d'agir sur des octets (s ou b), des mots (w) ou des mots longs (l).

```
* OUVERTURE DE LA BIBLIOTHÈQUE intuition.library
  MOVEQ   #INTUITION_REV,d0 version
  LEA   int_name(pc),a1
  CALLEXEC  OpenLibrary
  TST.l   d0      Test d'ouverture d'Intuition :
  BEQ   exit_false    si ça échoue on s'en va
  MOVE.l    d0,_IntuitionBase sinon sauvegarde du pointeur
```