Oric 1 & Oric Atmos

• Mickaël Pointier
Programmeur outils chez **Funcom**
Repousse les limites des « ordinosaures » de son enfance pendant son temps libre
mike@defence-force.org

Les ordinateurs Oric 1 et Atmos furent élus « ordinateur de l'année » respectivement en 1983 et 1984 en France. Leur succès fut probablement plus causé par leurs prix très raisonnables que par leurs performances, mais ces machines étaient probablement les choix idéaux pour apprendre à programmer.

1

Dans les années 80, l'essentiel du marché était occupé par des machines équipées de processeurs 8 bits (en grande majorité basés sur sur le microprocesseur MOS 6502 ou Z80 de Zilog), 64 kilo-octets de mémoire vive, et utilisant soit des cassettes audio, soit des disquettes 5.25 pouces pour stocker les informations. [1]

Ce qui différenciait les différents produits provenait surtout des composants supplémentaires pour aider le processeur à gérer l'affichage plus efficacement, ou à avoir la capacité à produire des sonorités plus évoluées que de simples « BIPs ». Cela provenait aussi de la qualité du logiciel dans la mémoire morte de la machine, ou de la présence de connecteurs d'extension.

Je ne vais pas maintenir le suspens plus longtemps : la raison pour laquelle les machines Oric étaient si peu chères est essentiellement due à la simplicité et aux limitations de la conception de la machine.

Minitel où es-tu ?

En France nous avions le Minitel, les Anglais avaient leur propre standard Vidéotex : le Prestel, abréviation signifiant « Press Telephone ». Originellement les machines Oric n'étaient pas conçues pour les joueurs, la cible était la petite entreprise, la comptabilité familiale ou encore l'utilisation comme terminal bancaire à distance.

Au cœur de la machine se trouve donc l'ULA HCS 10017, un composant capable d'afficher nativement des textes contenant des codes de contrôle vidéotex. L'ULA est aussi en charge de l'arbitrage de bus, rafraîchissement de la mémoire, permettant ainsi de réduire le nombre de composants sur la carte mère. Autour de l'ULA nous trouvons aussi un microprocesseur conçu par MOS Technologie le 6502 cadencé à 1 Mhz, 48 kilo-octets de mémoire vive utilisable (en fait il y a 64 kilo-octets de mémoire installée, mais elle n'est accessible que si un lecteur de disquette est connecté), plus 16 kilo-octets de mémoire morte hébergeant un BASIC Microsoft étendu.

Pour finir, le composant VIA 6522 est capable de fournir des chronomètres programmables, lancer des interruptions, et permet l'accès au générateur de son trois voix AY-3-8912 de General Instrument. A noter que le générateur de son lui-même est utilisé pour accéder au port imprimante (compatible Centronics) et de lire l'état des touches sur le clavier.

Idéal pour débuter

L'Oric est au Commodore 64 ce que l'Atari ST est à l'Amiga : juste un microprocesseur et de la mémoire directement accessible.

Pas de gestion de Sprites ou de Player de Missiles, pas de Copper ou de Display List, et pas non plus de Blitter. Pour les joueurs, le résultat est malheureusement immédiatement visible, les jeux sont souvent plus lents, moins jolis, et moins réactifs que sur les autres machines. Pour le programmeur potentiel par contre cela présente de nombreux avantages : il pourra se consacrer à l'apprentissage de la programmation grâce à la simplicité de la machine qui rend l'entreprise bien plus simple tout en limitant les espoirs quant aux résultats atteignables !

S'il n'en fallait qu'un...

Le manuel d'utilisation de l'Oric Atmos est une référence du genre, et il est bien dommage que l'on ne trouve plus que rarement ce genre de documents pour les produits modernes : complet, progressif, exhaustif et utilisable comme référence même lorsque l'on a atteint la maîtrise de la machine. [2]

2

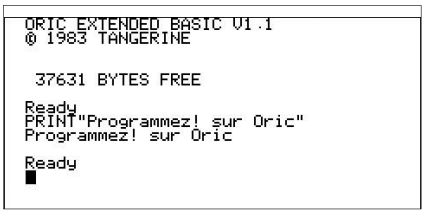
Lorsque l'utilisateur a finalement atteint la page 318 du manuel, il aura appris comment brancher sa machine (et les différents périphériques), ce que signifie le mot B.A.S.I.C. et les informations affichées au démarrage, comment charger et sauver des programmes, comment éditer du code et comprendre les messages d'erreur, la notion de variable et de type (numérique, chaîne de caractère), fait de petits calculs (TVA, circonférence), avant de monter en difficulté en abordant les limites nu-

mériques, encodage des valeurs flottantes, calcul hexadécimal et binaire, etc.
Les chapitres défilent, présentant la programmation structurée (boucles FOR, IF THEN ELSE, GOSUB RETURN, ON GOTO, etc.), les instructions graphiques et sonores, et même une partie de type « Inception » comme le film présentant un petit programme montrant comment le programme est stocké en mémoire par le système, une initiation à la programmation en assembleur 6502 et une série d'annexes contenant les schémas électroniques, emplacement des variables et fonctions systèmes, description de chaque broche de chaque connecteur, et même une section consacrée à la programmation de l'imprimante traceur.

Langages

Il faut environ une seconde pour démarrer un Oric et exécuter une instruction en langage BASIC. Lorsque le message Ready apparaît, il suffit d'entrer une instruction et valider. Par exemple :

```
PRINT"Programmez! sur Oric"
```



Durant la vie commerciale de la machine, l'essentiel des logiciels étaient écrits en BASIC, certains incorporant des routines en assembleur 6502 pour les parties critiques. Il était aussi possible de goûter aux joies de la programmation en LOGO ou Forth si vous pouviez vous procurer le logiciel adéquat. Nous sommes maintenant au 21e siècle, il serait dommage de se limiter à ces seuls choix !

Développement

Le problème du développement natif sur une machine limitée est qu'il est impossible d'utiliser 100% des potentialités. Sur une machine disposant de seulement 64 kilo-octets de mémoire, la mémoire utilisée par l'éditeur, le compilateur, le linker, le système d'exploitation, le débogueur, etc. ne sont pas disponibles pour l'application qui est développée.
Ce problème est résolu par l'utilisation d'un environnement de développement croisé qui est souvent composé d'un ensemble d'outils utilisés pour générer un produit prêt à l'usage sur la plateforme cible, ainsi qu'un émulateur et débogueur permettant de tester rapidement.
Dans cet article je vais utiliser le OSDK, mais il existe d'autres options, tel que CC65.

OSDK

Le OSDK contient un compilateur C, un assembleur 6502, un tokenizateur BASIC, des bibliothèques de fonctions standard, de la documentation et des exemples, ainsi que des outils pour convertir les données graphiques et sonores, compresser les fichiers, ou générer des disquettes au format émulateur.

Il contient aussi un émulateur et un débogueur symbolique. Tous les composants sont optionnels, et le code source est disponible.
Voyons comment écrire un simple programme, le fameux « Hello World », en utilisant le BASIC, le C et l'Assembleur.

Principes de fonctionnement

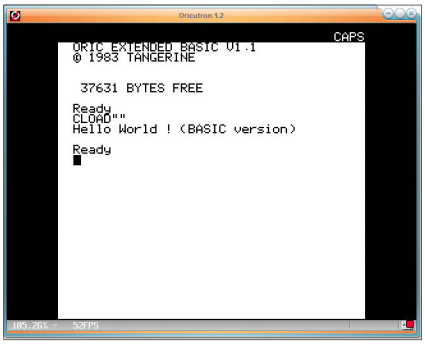
Les premières versions de l'OSDK ne fonctionnaient que sous MS-DOS, ce qui explique la présence de nombreux fichiers scripts BAT et de variables d'environnement.
Typiquement un projet OSDK est définit par trois fichiers qui sont l'équivalent de ce que pourrait faire un fichier script makefile, l'avantage étant que l'on peut facilement utiliser le système en mode graphique en double cliquant les fichiers avec l'extension BAT.
Le script osdk_built.bat sert à lancer la génération du projet et est rarement modifié.
Le script osdk_execute.bat est utilisé pour exécuter le programme dans une nouvelle session émulateur.
Le script osdk_config.bat contient les paramètres du projet à compiler, définis par l'intermédiaire de variables d'environnement, dont essentiellement:
OSDKNAME – Nom du programme
OSDKFILE – Modules à compiler
OSDKADDR – Adresse du programme
A noter : certains projets contiennent aussi un fichier osdk_makedata.bat qui doit être lancé pour générer les données graphiques et sonores.
Le répertoire /sample contient un certain nombre d'exemples prêts à compiler démontrant comment le système fonctionne.

Hello World (en BASIC)

Vous trouverez la version BASIC dans le sous répertoire basic/hello_world.
Le fichier main.bas contient le code source du programme :

```
10 '
20 ' This is the standard
30 ' "HELLO WORLD" sample
40 ' written in BASIC
50 '
60 PRINT "Hello World ! (BASIC version)"
```

En appelant osdk_build, le programme est converti au format fichier cassette utilisé par les émulateurs Oric et se trouve dans le sous répertoire BUILD sous le nom HW.tap.
La compacité est dû au format qui est « tokenizé » (opération qui consiste à remplacer les *mots clefs* par une valeur numérique unique les identifiants chacun), chaque instruction occupe seulement un octet en mémoire et c'est l'interpréteur BASIC en ROM qui se charge du décodage.
En lançant osdk_execute, l'émulateur Oricutron est appelé et le programme directement exécuté.



Notez que le programme complet occupe seulement 157 octets !

Hello World (en C)

Vous trouverez la version C dans le sous répertoire /c/hello_world_simple.
Le fichier main.c contient le code source du programme :

```
//
// This is the standard "HELLO WORLD" sample
//

#include <lib.h>

void main()
{
  printf("Hello World !\n");
}
```

La procédure est identique et le résultat similaire, a l'exception de la taille du programme de 1045 octets !

Comparé à la version BASIC, le code est directement exécutable par le microprocesseur, bien moins compact, mais aussi bien plus rapide à l'exécution. Si vous êtes curieux vous pouvez regarder le contenu du fichier linked.s dans le sous-répertoire osdk/TMP. Il contient le code assembleur généré par le compilateur C ainsi que les fonctions des bibliothèques utilisées par le programme avant que l'assembleur ne soit invoqué.

Hello World (en Assembleur)

La version 6502 est dans le sous répertoire /assembly/hello_world_assembly.
Le fichier main.s contient le code source du programme :

```
_main
  ldx #0

read
  lda message,x
  bne write
  rts

write
  sta $BB80,x

  inx
  jmp read

message
  .asc "Hello World !",0
```

Ce programme ne fait pas exactement la même chose que les versions en BASIC et en C : les exemples précédents utilisent la fonction d'affichage du système qui prend en charge le déplacement du curseur, et qui lorsque l'affichage a atteint la fin de l'écran lance le défilement vers le haut. Cette version assembleur se contente d'afficher le message Hello World à un emplacement fixe situé sur la première ligne de l'écran. La taille totale de l'exécutable est de 47 octets, incluant 14 octets d'informations d'entête du format cassette.

Performance et taille

Comparer la performance des différents langages est relativement difficile. Au niveau du code, le BASIC est de loin le langage le plus compact. Le format « tokenisé » permet de stocker une large quantité de code compliqué de façon efficace, ce qui rend aussi les programmes rapides à charger. D'un autre côté, la gestion des variables, tableaux, ou données en général, n'est pas particulièrement efficace, le

BASIC ne connaissant que deux types de données: chaîne de caractère, et valeurs en virgule flottante.
Les programmes C pourraient être plus efficaces, malheureusement nous avons ici un vieux compilateur qui n'est pas très à l'aise avec le jeu d'instruction limité du 6502. Malgré tout la performance des programmes en C est largement supérieure à l'équivalent en BASIC.
L'assembleur pur est de loin la meilleure solution sur cette machine, donnant accès à de nombreuses optimisations impossibles à effectuer en langage C; contrairement au BASIC, il peut se passer du système d'exploitation et donc permet de récupérer 16 kilo octets de mémoire vive supplémentaire. Pour donner un ordre de grandeur : la différence de vitesse entre le code C et le code assembleur fonctionnellement équivalent et d'un facteur 16 a 64 selon la complexité du code.
Sur du code simple la différence n'est pas trop importante, mais dès que l'on commence à jongler avec les variables, les boucles multiples, les conditions de sorties, etc. le code du compilateur devient totalement inefficace.
On retrouve le même ordre de grandeur entre le C et le BASIC. [3]
Exemple concret : remplir l'écran en cyclant avec toutes les couleurs disponibles

```
10 HIRES
20 FOR I=0 TO 8000
30 : POKE #A000+I,16+(I AND 7)
40 NEXT I
```

Cette simple version en BASIC prend environ 1 minute et 47 secondes pour remplir tout l'écran.

```
void main()
{
  char* screen;
  hires();
  for (screen=(char*)0xa000;
       screen<(char*)(0xa000+8000);
       screen++)
  {
    *screen=16+((int)screen & 7);
  }
}
```

Ce code C équivalent ne prend que deux secondes pour remplir l'écran, c'est à dire plus de 50 fois plus rapide que la version BASIC.

```
_main
  jsr _hires

  ldy #200
loop_y
  ldx #0
loop_x
  txa
  and #7
  ora #16
__write
  sta $a000,x
  inx
  cpx #40
  bne loop_x

  clc
  lda __write+1
  adc #40
  sta __write+1
  bcc skip
  inc __write+2
skip

  dey
  bne loop_y
  rts
```

Et pour terminer, cette version assembleur qui fait la même chose en environ une seconde et qui serait facilement optimisable en déroulant le code.

Ressources

Le but de cet article n'était pas de faire de vous un programmeur Oric mais simplement de vous donner un aperçu de ce qui est faisable.
Si vous être intéressé, n'hésitez pas à télécharger le OSDK, à vous inscrire sur le forum des développeurs de Defence Force, voire même à vous abonner au magazine mensuel du Club Europe Oric. •

Quelques liens utiles:

- OSDK : http://osdk.org
- Forums Defence Force : http://forum.defence-force.org
- Oric.org : https://www.oric.org
- Source code des projets : http://miniserve.defence-force.org/svn/users/dbug/programmez/

Vous êtes aussi le bienvenu sur notre canal IRC #Oric sur IRCnet!