

## Orion - Environnement de programmation pour LedMe avec le circuit Orion

### Introduction

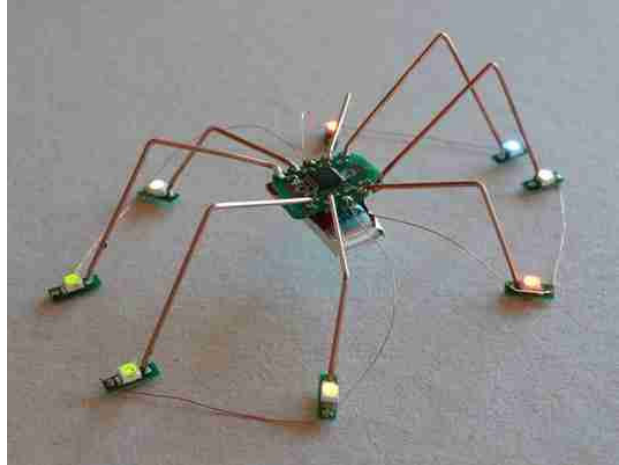
Plusieurs environnements de programmation similaire permettent de programmer des LEDs avec le langage simple en français. Voir

<http://www.bricobot.ch/docs/Smile.html>

Ce document liste les ordres pour le module Orion (aussi appelé LmP8 et mPro). Par rapport au Coeur4 et BimoLed, il y a deux ordres pour commander un moteur en option.

La documentation de câblage est sous <http://www.bricobot.ch/doc/LedMe.pdf>.

Le support des modules Lm8Out et Lm8In sera fait dès qu'il y aura de l'intérêt. Voir <http://www.bricobot.ch/docs/LedMeNim.pdf> comme exemple d'application.



### Instructions

Commençons par quelques définitions, qu'il faudra relire après avoir vu les exemples. En première lecture, c'est un peu abstrait.

Les instructions qui ont été définies permettent d'écrire et modifier des programmes qui agissent sur les LEDs. Une instruction définit une action, par exemple clignoter toutes les Leds. Mais il faut dire à quelle vitesse on veut qu'elles clignent, et combien de fois. ce sont les paramètres de l'instruction, qui sont soit des constantes, soit des variables.

Dans les programmes du début, on utilise surtout des constantes ; on corrige leur valeur dans le programme, on recharge les nouvelles instructions dans le processeur, et on vérifie l'effet.

Une constante doit toujours être précédée du signe # (prononcé valeur). C'est un nombre, auquel on peut donner un nom (c'est souvent préférable) en utilisant le signe = (on parle d'assignation).

Les variables sont dans une zone mémoire ou il faut réserver sa place ; elles nécessitent plus de réflexion, on y reviendra dans la 2<sup>e</sup> partie.

Les instructions sont alignées dans la mémoire programme. On peut donner un nom à une position mémoire programme (une étiquette, comme pour marquer des casiers superposés). On doit le faire si on veut sauter des instructions pour continuer à un endroit précis. Une étiquette est suivie d'un : (deux-points). Elle doit avoir un nom nouveau qui, comme pour tous les noms que vous devrez inventer, ne commence pas par un chiffre (et n'est pas déjà utilisé par le programme de traduction).

### □ Installer SmileNG à partir du CDrom

Le CDrom contient tous les programmes nécessaires, expliqués dans **Contenu.txt**. Créer un dossier appelé par exemple **Coeur4** au plus bas niveau (disque C). Copier dans ce dossier tous les dossiers et fichiers du dossier Coeur4 du CDrom.

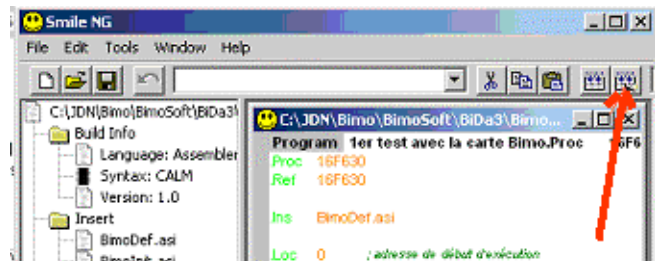
Ouvrir le sous-dossier **SmileNG**, cliquez une fois (bouton droite) sur **SmileNG.exe** pour créer un raccourci et le placer sur le bureau.



Ouvrir SmileNG en cliquant sur le raccourci.  
Si la fenêtre à droite n'affiche pas *SaveHex* cliquer dessus et choisir le bon texte.



Pour vérifier, ouvrir le fichier **C0.asm** dans le dossier **Coeur** et assembler avec F5 ou en cliquant sur l'icône (flèche rouge).



SmileNG se souviendra du sous-dossier dans lequel vous travaillez. Cela sera plus facile la prochaine fois.

Ne pas modifier les fichiers avec l'extension **.asi**.

## □ Ecrire le programme

Les instructions qui ont été définies permettent d'écrire et modifier des programmes qui agissent sur les LEDs. Une instruction définit une action, par exemple faire clignoter toutes les Leds. Mais il faut dire à quelle vitesse on veut qu'elles clignotent, et combien de fois. ce sont les paramètres de l'instruction, qui sont ici des constantes.

Une constante doit toujours être précédée du signe # (prononcé valeur). La valeur maximale est 255. On peut modifier la valeur d'une constante, et c'est une partie du jeu que l'on va faire, mais il faut chaque fois re-traduire le programme, recharger les nouvelles instructions dans le processeur, et vérifier l'effet en exécutant le programme.

Les instructions sont alignées dans la mémoire programme. On peut donner un nom à une position mémoire programme (une étiquette, comme pour marquer des casiers superposés). On doit le faire si on veut sauter des instructions pour continuer à un endroit précis. Une étiquette est suivie d'un : (deux-points). Elle doit avoir un nom nouveau qui, comme pour tous les noms que vous devrez inventer, ne commence pas par un chiffre (et n'est pas déjà utilisé par le programme de traduction).

### Structure générale des programmes

Pour que nos instructions s'exécutent correctement, il faut dire plusieurs choses au programme de traduction, dans un certain ordre. Les définitions et instructions sont insérées par des ordres **.Ins** . Il faut avoir bien compris le processeur et l'assembleur pour modifier ces fichiers insérés.

Les signes \ et ; caractérisent le début d'un commentaire. La fin de la ligne est ignorée. On peut mettre des lignes vides et, pour aligner, on utilise de préférence la touche « tab » (à gauche, marquée par une flèche).

```
\prog :NomDuFichier.asm et brève explication  
.Ins OrionDef.asi
```

```

.Ins OrionInit.asi
; eventuellement mettre ici le nom du fichier, codé
.Ins OrionProg.asi
; mettre le programme ici
; le programme doit se terminer par un AllerA, autrement le processeur
; va exécuter des mots binaires dans la mémoire qui feront on ne sait quoi !
.Ins OrionFin.asi
.End

```

### Exemple : programme demo

<pre> \prog:O1.asm demo .Ins OrionDef.asi .Ins OrionInit.asi .16 "O","1"," "," " (suite colonne droite) </pre> <p>Note : le fichier OrionDef.asi est associé au module Orion avec possibilité d'extension moteur. Les autres versions suivantes acceptent les mêmes programmes, mais avec d'autres fichiers insérés.</p> <p>Les Orion sont programmés d'origine avec le programme O1.asm.</p> <p>Appelez vos programmes de test O2.asm, O3.asm etc avec un nom plus explicite pour les versions « finales ».</p>	<pre> .Ins OrionProg.asi Boucle : Motif #2'11001001,#2'00100110 Allume #20 Motif #0,# 0 Allume #5 Motif #2'01010101,#2'01010101 Clignote #10,#4 AllerA BouCle .Ins OrionFin.asi .End </pre>
--	---

Pour assembler pressez sur F5.

### Instructions spécifiques

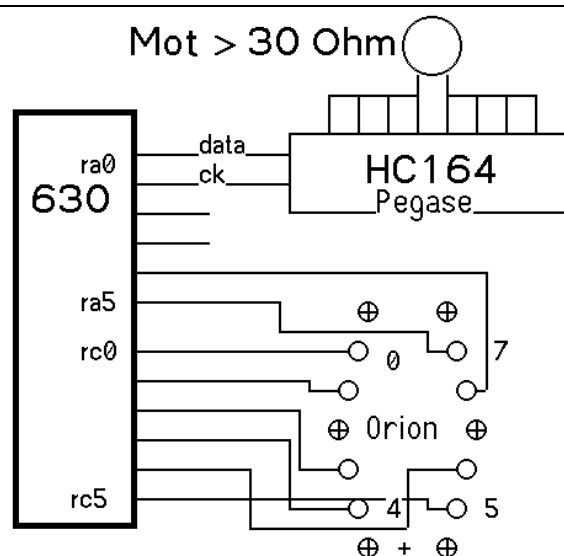
Avec l'instruction **Motif**, on prépare ce qui sera affiché. Il faut décrire les 8 bits dans l'ordre, et noter deux nombres binaires que le processeur comprendra et enverra dans l'électronique qui commande les diodes lumineuses.

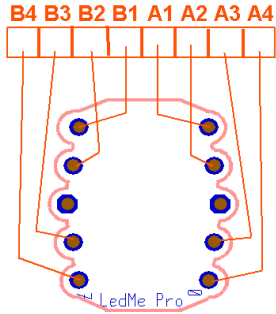
Le processeur travaille avec des mots de 8 bits. Il suffit de préparer un mot pour commander les 8 diodes.

**Regarder l'ordre des bits dans la figure.**

La notation #2' devant un nombre binaire (formé de 0 et de 1) dit au programme que c'est une constante écrite en binaire. Un nombre usuel (décimal) s'écrit directement et a une valeur maximale de 255.

On a  $2'00000000 = 0$ ,  
 $2'00000001 = 1$ ,  $2'0000010 = 2$



<b>Allume</b> #Duree Exemple : Motif #2'11001001 Allume #20 ; env 2 sec	Allume le motif pendant 20 unités de durée	
<b>Clignote</b> #Duree,#NombreDeFois Exemple : Motif #2'11110000 Clignote #1,#10	Allume le motif pendant la durée, éteint pendant la même durée, répète NombreDeFois	

Question : Comment remplacer **Clignote #2,#4** en utilisant uniquement l'instruction **Allume**?

Pour créer un nouveau programme, le plus simple est de reprendre un programme existant et d'en faire une copie sous un autre nom (onglet Fichier – Enregistrer sous).

<b>Varie</b> #DureeEvolution, #NbdeFois Exemple Varie #1,#20 ; Pulse 20 fois le plus rapidement possible ; (env 1 battement/seconde)	Augmente et diminue l'intensité du motif préparé, NbdeFois, selon DuréeEvolution	
---	--	--

<b>TourneG</b> #Durée,#NombreDeFois Exemple : Oscille: Repete #4 TourneG #4,#1 Attend #2 Tourned #4,#1 Attend #2 FinRepete Oscille	Allume une LED après l'autre en tournant dans le sens inverse des aiguilles de la montre, toutes les Durée, et répète NombreDeFois (pas de motif à préparer)  Dans cet exemple, on tourne à gauche et on répète 4 fois.	
<b>TourneD</b> #Durée,#NombreDeFois	Idem à droite	

<b>Chenillard</b> #NbDeLeds,#Durée,#NombreDeFois	Allume NbDeLeds en les faisant tourner une fois pour les éteindre derrière.	
--	---	--

Que se passe-t-il si NbDeLeds est supérieur à 16 ?

### Instruction de structures, identiques à Abimo

<b>AllerA</b> Adresse C'est le <b>Goto</b> du Basic, le <b>Jump</b> d'autres langages	Exemples Boucle : ; Des instructions que l'on veut répéter indéfiniment AllerA Boucle
--	--

<b>Stop</b> Un programme se termine toujours par un AllerA ou par un Stop. Autrement, le processeur va exécuter des bits en mémoire, et on ne sait pas ce qu'il va faire	A noter que les instructions et les paramètres sont décalés et alignés pour faciliter la lecture. Minuscules et majuscules dont identifiées. Un commentaire est précédé d'un ; ou d'un \
--	--

Etiquette : <b>Repete</b> Nombre de fois ; instructions répétées <b>FinRepete</b> Etiquette	
--	--

<b>Attente</b> DuréeEnDixièmesdeSec	Attente #10 ; pour une seconde
-------------------------------------	--------------------------------

### Instructions supplémentaires pour l'option moteur

Le moteur, électro-aimant ou diode lumineuse puissante reçoit un courant qui va dans un sens ou dans l'autre (ou pas de courant). Il y a donc 3 instructions.

<b>Mpos</b> active dans le sens positif	Exemple : Mpos Motif #2'11000011,W Allume #20 Mneg Motif #2'00111100,W Allume #20 Mstop
<b>Mneg</b> active dans le sens négatif	
<b>Mstop</b> désactive	

### Utilisation des variables

Les variables permettent de faire des boucles dans lesquelles des paramètres changent.

L'ordre Cligno par exemple a un paramètre qui est la durée du clignotement. Cette durée peut être fixe, ou variable

Si on écrit Cligno #2, la durée est constante, environ 0.2 secondes

Si on écrit Cligno Dur1, on ne peut pas dire quelle est la durée en lisant

l'instruction. La valeur de la durée est dans la position mémoire Dur1, ce sont les instructions précédentes qui ont préparé la valeur dans Dur1.

Si on écrit

Copie #2,Dur1

Cligno Dur1

alors on connaît la durée du clignotement.

L'avantage des variable, c'est qu'elles peuvent varier! Ecrivons

Copie #2,Dur1

Rr1: Repete #20

Cligno Dur1

Augmente Dur1

FinRepete Rr1

Le clignotement va se répéter 20 fois avec une durée qui augmente.

Ceci permet des effets plus spectaculaires en écrivant moins d'instructions.

Les variables doivent être déclarées, pour que le traducteur leur réserve de la place en mémoire. On doit donc inventer un nom, aussi explicite que possible.

A l'emplacement prévu pour les variables, on écrira

Var Dur1

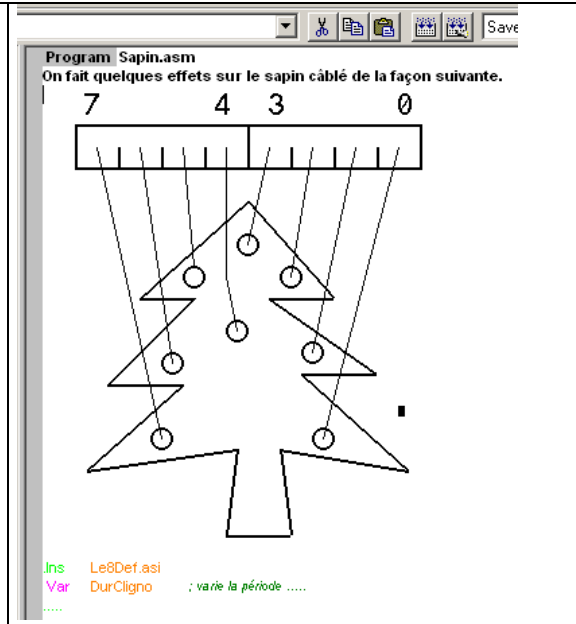
L'art d'un programme bien écrit, que l'on relit facilement après quelques mois, tient beaucoup dans le choix des noms donnés aux variables et aux étiquettes. Dans les long programme, un commentaire doit annoncer un groupe d'instruction.

A noter que dans les programmes SmileNG, on peut insérer des images. Le fichier doit être un .bmp et on écrit au début d'une ligne

\image:Mondessin.bmp (pas d'espaces autour du : )  
La touche F8 fait apparaître ce texte ou le dessin

### Exemple

```
\prog:Sapin.asm
;On fait quelques effets sur le sapin câblé de
la façon suivante.
\image:Sapin.bmp
.Ins OrionDef.asi
Var DurCligno ; varie la période
.Ins OrionInit.asi
; mettre les tables ici ; pas de tables
.Ins OrionProg.asi
Boucle:
.....
AllerA Boucle
.Ins OrionFin.asi
```



### Instructions utiles avec des variables

#### Copie Source, Destination

On prend la valeur de la source et on la met dans la variable destination (nécessairement une variable)  
La copie ne modifie pas la source.

#### Copie #7, Décompteur

Copie la valeur 7 dans la variable Décompteur (on dit initialise cette variable)

#### Copie NbdeFois, Décompteur

Copie une variable dans une autre

#### Augmente Destination

#### Diminue Destination

#### DécaleADroite Destination

#### DecaleAGauche Destination

Exemples dans la 2<sup>e</sup> partie

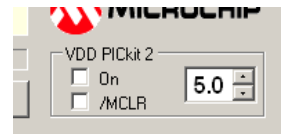
Exemples à faire !

## □ Préparer le programmeur

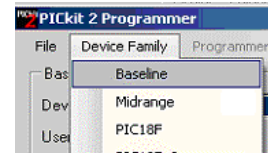
Ouvrir le sous-dossier **Microchip**, cliquer sur **Pickit2V2.exe** pour créer un raccourci et le placer sur le bureau. Double-cliquez sur l'icône.



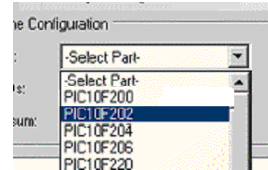
Vérifier que la tension indiquée à droite est 5V comme sur la figure (agir sur l'ascenseur si nécessaire)



Choisir dans **Device Family** **baseline**

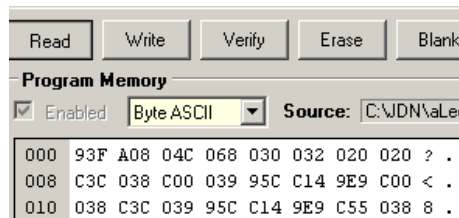


puis dans la liste **Device** **Pic10F202**



## □ Programmer Orion

On peut maintenant connecter Orion sur l'adaptateur et faire une lecture ( **Read** ) pour vérifier. Bien localiser la lin1, pastilla carrée avant d'insérer, et ne jamais forcer. Mailler légèrement le connecteur pendant la programmation, pour garantir le contact. Si le processeur n'est pas vide, on peut voir le contenu de la mémoire, noté en hexadécimal.

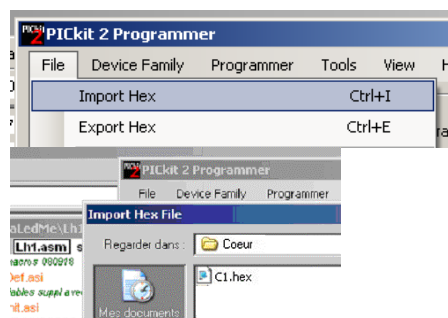


## Programmation

Une fois que SmileNG a annoncé **Assemblage correct**, un fichier du même nom avec l'extension **.hex** est créé. Il faut alors basculer sous Pickit2 et charger ou recharger ce fichier.

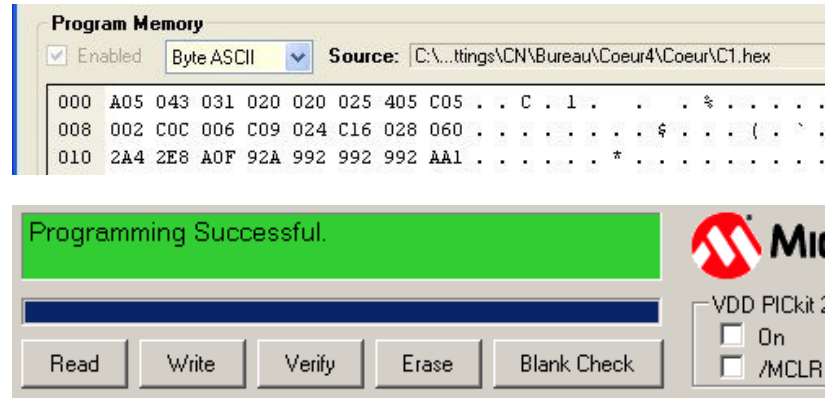
Par exemple, pour programmer le cœur avec C1, cliquer sous **File** et choisir **Import Hex**.

Choisir le dossier Coeur et donner le nom de fichier C1.hex.



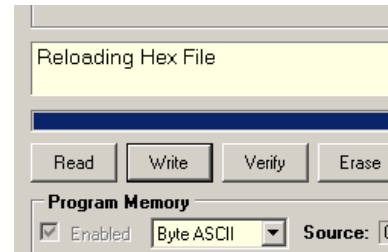
Le programme se charge et on peut voir le nom du fichier dans le code, s'il a été déclaré dans le source.

Pour programmer, cliquer sur **Write**.



Si on recharge le même programme, qui a été modifié, la séquence est plus rapide, il suffit de cliquer sur **Write**. Le Pickit2 voit que le .hex a été modifié et reprogramme sans demander le nom du fichier.

Attention, bien tenir à la main pendant la programmation. Un mauvais contact ou une inversion peut mettre le processeur dans un mode qui nécessite ensuite une procédure de récupération (voir <http://www.bricobot.ch/docs/RecupPic.pdf> )



Pour des informations supplémentaires, voir <http://www.bricobot.ch/docs/AbimoNotes.pdf> , en particulier

**Note 3** : Que faire si le Pickit2 ne se charge pas et si le processeur n'est pas reconnu

**Note 4** : Les menus de SmileNG

**Note 6** : Dépannage et erreurs

Une initiation à l'assembleur vous intéresse ? voir <http://www.epsitec.com/dauphin/>

Vous voulez apprendre l'assembleur du 16F87x, similaire au 16F630 ?

<http://www.didel.com/picg/picg87x/CoursPicg87x.html>

Novembre 2008