

Apprenons à programmer le microcontrôleur du Bimo

Ce texte explique la programmation en ne supposant aucune connaissance au départ. Des programmes simples sont exécutés sur le Bimo pour expliquer les notions utiles l'une après l'autre. L'environnement SmileNG et le programmeur Pickit2 (à installer, c'est le plus difficile) permettent de passer un temps minimum pour écrire, assembler et exécuter les programmes. Le langage simplifié utilisé prépare à apprendre un langage informatique plus complet, assembleur, Basic, C. Il habitue un débutant à la discipline d'écriture de programmes, et consolide les notions de constantes, variables et structures de contrôle.

Les programmes décrits plus loin se trouvent sous <http://www.bricobot.ch/docs/Abimo.zip> si vous n'avez pas le CD Abimo entre les mains.

SmileNG peut se charger depuis <http://www.bricobot.ch/docs/SmileNG.zip>

et l'environnement Microchip depuis <http://www.bricobot.ch/docs/Microchip.zip>

Matériel et logiciel nécessaire

Bimo 45.-

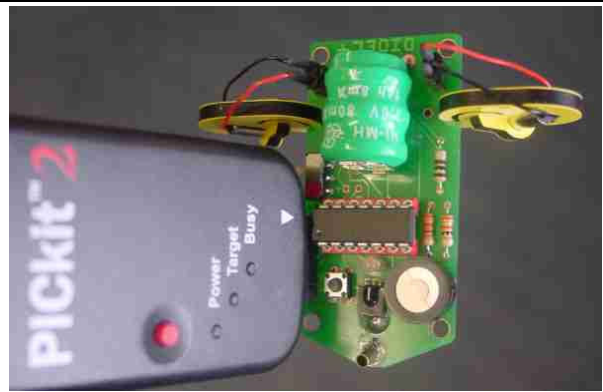
Pickit2 65.-

Bimo+Pickit2 100.- prix spécial avec CDrom contenant les programmes à installer et la documentation.

Note 1 Options pour souder le connecteur de programmation du Bimo.

Les notes sont dans un fichier séparé, pour éviter de surcharger avec une information plus détaillée. Voir AbimoNotes.pdf sur le CD ou

<http://www.bricobot.ch/docs/AbimoNotes.pdf>



Si vous devez installer sans avoir le CdRom sous la main, suivez les instructions sous <http://www.bricobot.ch/docs/Abimo07.pdf>

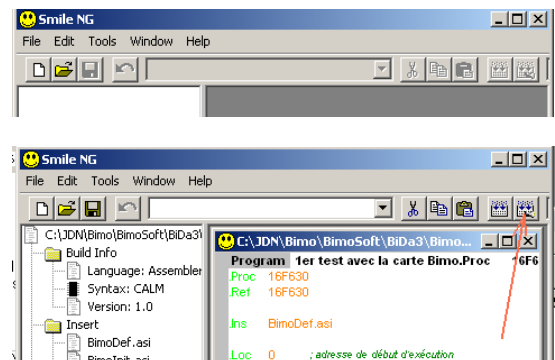
Le CDrom contient tous les programmes nécessaires, expliqués dans **Contenu.txt**.

Créer un dossier **Bimo** au plus bas niveau (disque C). Copier dans ce dossier tous les dossiers et fichiers du CDrom.

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

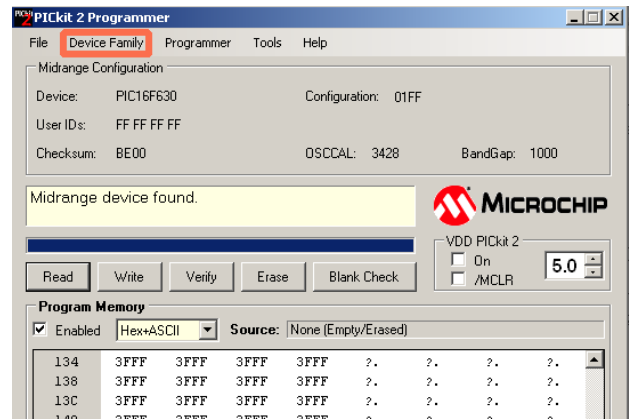
Note 2 Les fichiers de SmileNG07

Pour vérifier, ouvrir le fichier **B1.asm** dans le dossier **Bimo - Abimo** at assembler avec F5



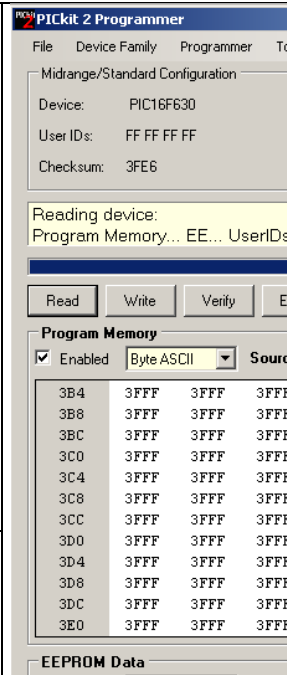
Ouvrir le dossier **Microchip**, cliquer sur **Pickit2.exe** pour créer un raccourci et placer sur le bureau.

Le 16F630 est dans la catégorie « midrange ». Il est parfois automatiquement reconnu si le Bimo est connecté (choix dans "Programmer").



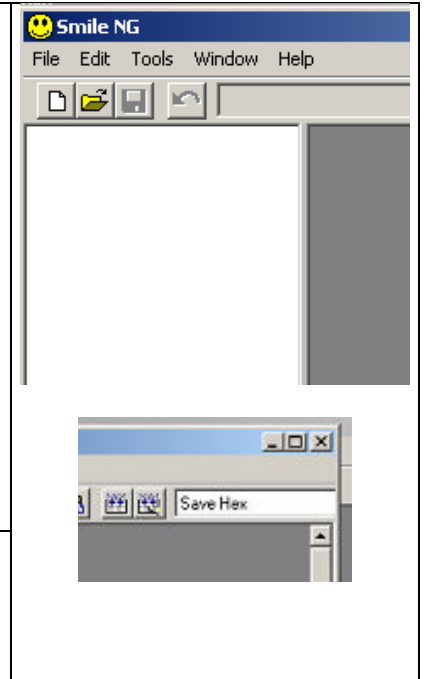
Après installation et après avoir cliqué dans l'icône Pickit2 l'écran a l'allure suivante (leBimo est connecté). Cliquer sur **Read** Choisir le mode **Byte ASCII**

Note3 : Que faire si le Pickit2 ne se charge pas et si le processeur n'est pas reconnu



Après installation et après avoir cliqué dans l'icône SmileNG l'écran a l'allure ci-contre. Choisir le mode **SaveHex** si ce n'est pas le cas.

Note 4 Les menus de SmileNG



Maintenant que les outils sont en place, on peut apprendre à programmer le Bimo.

Notre langage de programmation

Un processeur est très simple et il faut lui dire beaucoup de choses pour qu'il fasse ce que l'on veut. Notre méthode est de cacher les détails dans des fichiers insérés à l'exécution, et que l'on ne va pas chercher à comprendre pendant quelque temps.

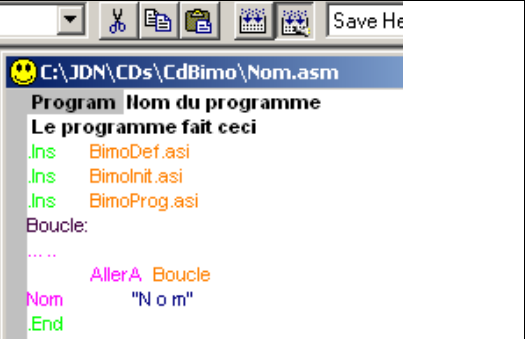
Il faut demander à l'assembleur d'insérer au début trois fichier :

BimoDef.asi contient des définitions (comment sont câblées les LEDs, etc) et des variables Bimolnit.asi démarre le processeur. BimoProg.asi contient les toutines, La coupure en trois permet d'insérer des variables et tables là ou le processeur les veut.

Notre langage est traduit par l'assembleur, mais les instructions ne sont pas des instructions du processeur 16F630, mais des groupes d'instructions (appelées macro-instructions) qui font le travail. Nos instructions ne connaissent que des nombres de 8 bits, que l'on écrira en décimal, de 0 à 255.

Un programme a donc l'allure suivante :

<i>Ce qui est tapé au clavier</i>	<i>Ce que montre SmileNG</i>
<pre> \prog ;Nom du programme \b; Le programme fait ceci .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: AllerA Boucle </pre>	

<pre>Nom "N o m" .End</pre>	
----------------------------------	---

On voit une spécialité intéressante de SmileNG : les ordres d'affichage sont mis en évidence ou ignorés par la touche du clavier F8. **Note 5** : Ordres de mise en évidence de SmileNG.. Les minuscules sont identifiées aux majuscules, mais les lettres accentuées ne sont pas acceptées comme noms (dans les commentaires, oui). Ce qui suit un ; ou un \ est un commentaire. La touche Tab aligne en colonne 8. Par exemple ci-dessus, il ne faut pas taper 8 espaces devant le AllerA, mais un Tab..

La ligne Nom permet de voir le nom du programme dans le hex affiché par Pickit2. Le nom doit avoir un espace entre chaque lettre, car les instructions du Pic sont 14 bits, et le code d'une lettre 8 bits. Il faut compléter avec, par exemple, un espace.

Un premier programme – B1.asm
Faisons clignoter les LEDs (il n'y en a qu'une sur les anciens Bimo, les programmes sont compatibles) . Il faut 3 ordres pour :
Allumer/éteindre les LEDs
Attendre avant de changer
Recommencer ou s'arrêter

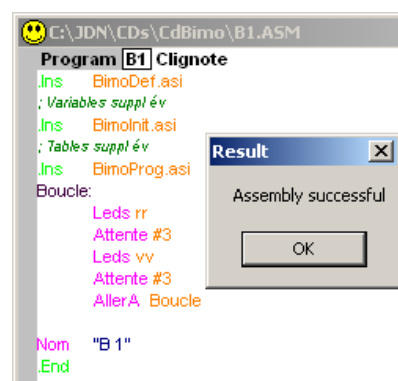
L'ordre **Leds xy** a un paramètre formé de 2 lettres qui donnent la position et la couleur. Par exemple,
Leds 0R ; (zero-R) Led éteinte à gauche, rouge à droite
Leds RV ; (R V) Led rouge à gauche, verte à droite

L'ordre **Attente nn** a comme paramètre un nombre entre 0 et 255 qui donne la durée de l'attente en dixièmes de secondes.
L'ordre **AllerA Nom** fait continuer le programme à l'adresse repérée par **Nom** :

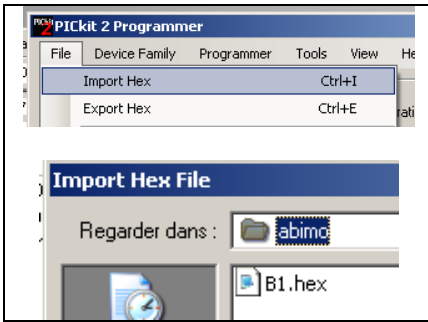
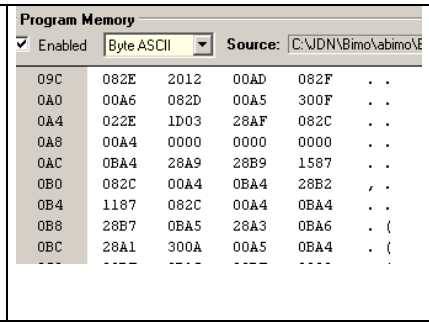
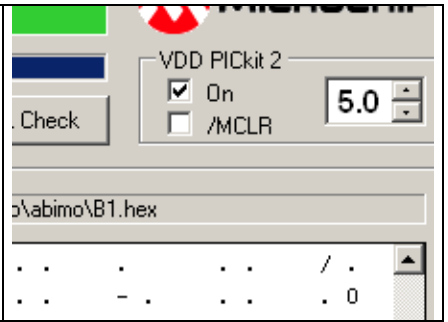


```
\prog;B1|Clignote
.Ins      BimoDef.asi
.Ins      Bimolnit.asi
.Ins      BimoProg.asi
Boucle:
    Leds   vv ; vert-vert ,ce n'est pas un w !
    Attente #3
    Leds   00 ; éteint
    Attente #3
    AllerA Boucle
Nom "B 1"
.End
```

Editer le programme ou l'ouvrir depuis le disque.
(Le signe # se fait avec AltGr-3)



La touche F5 (ou en cliquant dans l'icône) crée le fichier B1.hex s'il n'y a pas d'erreur.

		
<p>Charger le fichier .hex créé par l'assembleur dans Pickit2 : File – Import Hex – dossier abimo - B1.hex Cliquez sur Write pour programmer. Après modification du même fichier dans Smile, il suffit de cliquer sur Write, pas besoin de recharger, Pickit a vu le changement !</p>		<p>Cliquez sur On pour alimenter le Bimo. Laissez la tension à 5V. Ne pas enclencher l'accumulateur du Bimo s'il est connecté au Pickit2</p>

Modifiez v v en r r, 0 r etc. Ajoutez des lignes pour un clignotement plus riche. Les modifications sont très rapides : F5 dans SmileNG, passer sur Pickit2, Write (le fichier modifié est rechargé avant programmation).

S'il y a une faute, il suffit de la corriger, pas besoin de supprimer le message d'erreur. Mais la détection d'erreur n'est pas toujours parfaite ! Note 6 Anomalies de la détection d'erreur

Avez-vous remarqué qu'au démarrage la LED pulse 2 fois et la sirène du Bimo se fait entendre avant que la Boucle s'exécute ?

C'est un bout de programme appelé au moment de l'initialisation, qui prouve que le processeur a démarré correctement. Si cela ne réagit pas juste après, il faut aller relire son programme, ce n'est pas la faute du Bimo !

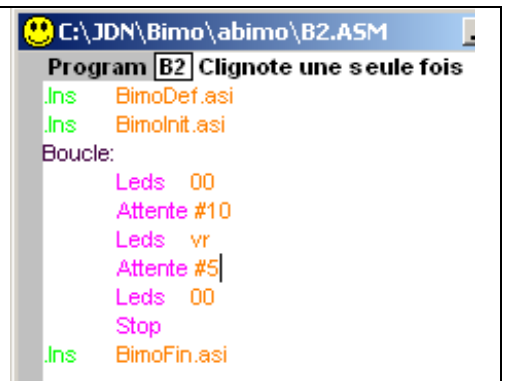
Attention ! Ne pas oublier le signe # quand il s'agit d'une constante. L'assembleur ne signale pas l'erreur, il croit que c'est une variable, et exécute une valeur imprévisible.

Le clignotement s'arrête – B2.asm

Le programme est le même, mais au lieu du AllerA Boucle, on met une instruction Stop qui est en fait un Ici : AllerA Ici

On ne peut pas stopper un processeur ! il doit toujours exécuter l'instruction suivante, qui peut être la même éternellement.

Enlevez l'instruction stop et observez. Si vous refaites la modif après avoir chargé d'autres programmes, le comportement sera différent, car il y a d'autres instructions résiduelles dans la mémoire.




```

C:\JDN\Bimo\abimo\B2.ASM
Program [B2] Clignote une seule fois
.ins BimoDef.asi
.ins BimoInit.asi
Boucle:
    Leds 00
    Attente #10
    Leds vr
    Attente #5|
    Leds 00
    Stop
.ins BimoFin.asi

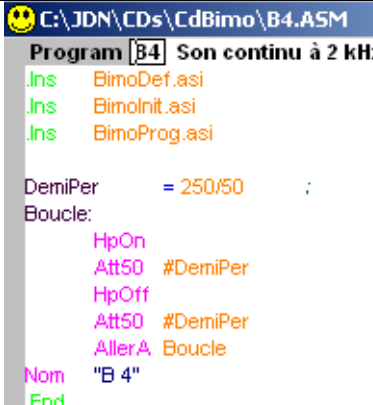
```

Et maintenant on veut faire deux fois la séquence ou plus? On ne va quand même pas retaper plusieurs fois la même chose !

<p>La paire d'instruction Etiquette : Repete #n FinRepete Etiquette est utilisée pour répéter n fois les instructions à l'intérieur, avec nn entre 2 et 255 Le programme B3.asm utilise cet ordre.</p> <p>On ne peut pas avoir un autre Repete dans un Repete.</p> <p>Les valeurs 0 et 1 sont acceptées, mais 1 n'est pas utile, et 0 fait faire 256 fois la boucle.</p> <p>Il serait plus élégant de donner des noms aux deux valeurs d'attente : DureeOn = 5 DureeOff = 2</p>	
---	---

Son continu

Pour faire un son continu, il suffit d'activer et désactiver le haut-parleur avec la bonne demi-période. Pour une fréquence de 2kHz, qui s'entend bien, cela fait une période de 0.5ms (2 kHz veut dire 2000 impulsions par seconde, 2 impulsions en 1 ms) donc une demi-période de 0.25ms = 250 microsecondes (si vous ne comprenez pas encore ces calculs, ce n'est pas grave).. Les ordres HpOn HpOff attirent et relâchent la membrane du petit haut-parleur. On programme comme pour les Leds (allumer-éteindre), mais il faut une boucle d'attente plus rapide. L'instruction Att50 #n attend n fois 50microsecondes. Il faut donc pour 2 kHz attendre n = 250/50 = 5 fois l'attente élémentaire). Avec n=2 on a 5 kHz, avec n=3 on a 3.3 kHz. On voit que l'on ne pourra pas générer n'importe quelle fréquence. Avec n =1 on a la fréquence maximale de 10 kHz, assez désagréable.

<pre> \prog;B4.asm Son continu a 2 kHz .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: HpOn Att50 #5 HpOff Att50 #5 AllerA Boucle Nom "B 4" .End </pre>	
--	--

Question : Si on fait des attentes inégales, avec le même total, la fréquence est la même ; mais est-ce que l'intensité du son varie ?

Sirènes

Comment programmer la sirène que l'on entend au démarrage du Bimo ?

La période varie, il faut donc faire intervenir dans ce programme une variable. Les variables doivent être déclarées entre les deux fichiers insérés. Elles valent zéro si on ne les initialise pas. Partons d'une fréquence basse, avec une attente de 20, soit une demi-période de 2,5 millisecondes, une période de 5 ms, une fréquence de 200 Hz. Diminuons ensuite la période, donc la fréquence va augmenter.

Pour pouvoir diminuer la période, il faut qu'elle varie, donc que cela soit une variable. Il faut la déclarer et l'initialiser avant de l'utiliser. L'ordre Copie permet de l'initialiser à la valeur 20, et l'ordre Diminue permet de la diminuer de 1, mais pas trop souvent. Toutes les 60 périodes semble convenir.

<pre> \prog;B5.asm Sirene .Ins BimoDef.asi Var DemiPer ; variable demi-période </pre> <p>Abimo 18.04.2009</p>	<p>On entend bien les pas d'augmentation de fréquence. Avec la sirène du Bimo, pourquoi est-ce plus régulier ? C'est parce que cette</p>
--	--

<pre>.Ins Bimolnit.asi .Ins BimoProg.asi Debut : Copie #20,DemiPer Boucle: Attente 5 ; pour séparer les 2 sirènes RR: Repete #60 HpOn Att50 DemiPer HpOff Att50 DemiPer FinRepete RR Diminue DemiPer AllerA Boucle</pre>	<p>sirène est programmée avec les instructions du processeur, ce qui permet de prendre des décisions plus précises qu'avec notre langage. Plus le langage est évolué, plus il est lent. Que se passe-t-il quand la sirène a apparemment fini ? On entend un son bas. En fait, le programme tourne toujours. La période a passé de zéro à 255 (donc 25,5ms) et la valeur de DemiPer augmente gentiment, très lentement car la période est longue. Si vous attendez 14 minutes, le programme repasse par les conditions initiales (DemiPer=20) et vous entendez la sirène pour 2 secondes.</p>
---	--

Comment éviter ces 14 minutes d'attente, et avoir le programme qui passe à autre chose, ou recommence ? Il faut tester que DemiPer vaut zéro, ce qui se fait avec l'ordre **SiEgalAllerA**. Cet ordre répond à la question : est-ce que 2 constantes ou variables sont égales (même valeur) et si oui, à quelle étiquette faut-il sauter ? Il y a donc trois paramètres à cet ordre : une première constante (donc avec le #) ou une variable, puis la 2^e variable à comparer, et enfin l'étiquette (adresse du saut). Le programme devient :

<pre>\prog;B6.asm Sirene qui recommence .Ins BimoDef.asi Var DemiPer ; variable demi-période .Ins Bimolnit.asi .Ins BimoProg.asi Debut: Copie #20,DemiPer Boucle: RR: Repete #30 HpOn Att50 DemiPer HpOff Att50 DemiPer FinRepete RR Diminue DemiPer SiEgalAllerA #0,DemiPer,Debut AllerA Boucle</pre>	<p>On déteste voir des nombres dans un programme. Il faut déclarer les valeurs, c'est plus facile à comprendre et à modifier.</p> <pre>ValDemiPer = 20 ; Valeur initiale PerEvolution = 30 ; Définit la vitesse d'évolution</pre> <pre>DebutProgramme Debut: Copie #ValDemiPer,DemiPer Boucle: RR: Repete #PerEvolution HpOn</pre>
--	--

Au lieu de s'arrêter à #0, on peut s'arrêter à #6 ou toute autre valeur inférieure à ValDemiPer. Cela serait bien de faire une petite attente entre ces sirènes en ajoutant l'instruction Att50 #2.

A vous de jouer

Comment faire un son qui monte et descend ? Il faut définir les périodes min et max, faire une première sirène qui monte, donc part de max et s'arrête (avec le SiEgalAllerA) à min. On enchaîne avec la sirène qui descend (et l'instruction Augmente pour augmenter). Bravo si vous pouvez le faire sans charger le programme **B7.asm**.

Soyons clairs et précis

Pour une bonne programmation, il faut inventer des noms qui soient explicites. Si on relit le programme plusieurs mois plus tard, il faut tout de suite savoir de quoi il s'agit ! Quelques noms sont réservés, et l'assembleur signalera par un commentaire « \error:5]Symbol double defined » .

Note 7 pour la liste de noms réservés en plus des ordres de notre langage.

Les noms doivent commencer par une lettre, et continuent par des lettres, des chiffres, ou les signes _ et ? .

On donne autant que possible des noms pour les **constantes**, et les constantes sont toujours précédées dans les instructions par le signe # (prononcé valeur). Notre langage simplifié n'accepte que des nombres de 0 à 255.

On donne des noms aux **variables** et on doit les déclarer, comme dans les deux exemples précédents. Une variable doit être initialisée. Une variable est une position mémoire, comme un tiroir ; on donne un nom au tiroir, mais cela ne dit pas son contenu, que l'on veut justement varier.

On donne des noms aux **étiquettes** qui sont suivies par un : quand elles sont définies.
 On peut mettre librement des espaces et des Tabs entre les parties d'une instruction, et des retours à la ligne entre les instructions. Il faut en profiter pour faire un programme agréable à lire.

Poussoir

Deux ordres permettent de tester le bouton poussoir du Bimo.

AttendPoussoir ne fait rien d'autre que surveiller le poussoir et continuer dès qu'il est pressé.

SiPoussoirAllerA Etiquette saute à l'adresse donnée si le poussoir est activé, mais continue autrement. Si on veut clignoter tout en surveillant le poussoir, ou interrompre un morceau de musique au milieu, c'est cet ordre qu'il faut utiliser.

L'exemple suivant utilise l'ordre Bip, pas encore vu, mais évident à comprendre.

<pre>\prog;B8.asm Bip si on agit sur le poussoir .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: AttendPoussoir Bip AllerA Boucle</pre>	<pre>\prog;B9.asm Coupe le clignotement .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: B1: Attente #1 Leds VV Attente #1 Leds 00 SiPoussoirAllerA B2 AllerA B1 B2: Stop</pre>
--	--

A noter que SiPoussoirAllerA n'attend pas que le poussoir soit relâché. Comparez les deux programmes suivants. Ba.asm n'a pas un fonctionnement fiable, car il dépend de la durée avec laquelle on presse . Bb.asm est fiable parce que l'on attend que le poussoir soit relâché.

A noter que dans les deux cas, il faut appuyer quelques dixièmes de seconde, car le processeur ne teste pas pendant les attentes de clignotement.

<pre>\prog;Ba.asm Modifie la couleur .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: B1: Attente #2 Leds VV Attente #2 Leds 00 SiPoussoirAllerA B2 AllerA B1</pre>	<pre>(suite) B2 : Attente #2 Leds RR Attente #2 Leds 00 SiPoussoirAllerA B1 AllerA B2 Nom "B a" .End</pre>
---	--

<pre>\prog;Bb.asm Modifie la couleur .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: B1: SiRelacheAllerA B11 AllerA B1 B11: Attente #2 Leds VV Attente #2 Leds 00 SiPoussoirAllerA B2 AllerA B11</pre>	<pre>(suite) B2 : SiRelacheAllerA B21 AllerA B2 B21: Attente #2 Leds RR Attente #2 Leds 00 SiPoussoirAllerA B1 AllerA B2 Nom "B b" .End</pre>
---	--

Un peu de musique

L'ordre Note permet de jouer une note. Il y a 15 notes, numérotées de 0 à 14, (auxquelles on a attribué les noms Do, Re, ... Do3) et un silence de la durée d'une note, appelé Sil. On peut donc jouer un morceau, enchaîner des parties répétées, etc. Ce n'est pas aussi bon qu'un portable, qui a un circuit spécial pour jouer exactement une fréquence.

<pre> \prog;B10.asm Jouer des notes .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: Attente #2 Note #Do Note #Sol Note #Do2 Note #Sil Note #Re Stop </pre>	<pre> \prog;B11.asm Mélodie .Ins BimoDef.asi .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: Attente #2 RR : Repete #3 Note #Do Note #Re FinRepete RR Note #Do2 Note #Sil Note #Re Stop </pre>
---	---

L'exemple suivant est important pour comprendre la différence entre constante et variable. On a dit que les notes sont en fait représentés par les nombres de 0 à 14 (15 notes). On peut les jouer dans l'ordre pour avoir la gamme.

<pre> \prog;B12.asm La gamme .Ins BimoDef.asi Var NumeroNote .Ins Bimolnit.asi .Ins BimoProg.asi Boucle: Attente #2 Copie #Do,NumeroNote RR : Repete #15 Note NumeroNote Augmente NumeroNote FinRepete RR Stop </pre>	<p>Humm !! Les fréquences sont justes, mais pas l'intensité. Ce qui se passe, c'est que le haut-parleur, comme tous les systèmes mécaniques, préfère osciller à des fréquences dites de résonance. Pour notre petit haut-parleur, il y a de fortes irrégularités dans l'intensité.</p>
---	---

Dépannage et erreurs

Si un programme ne marche pas, il faut localiser l'erreur en réfléchissant, et si on ne voit pas, en coupant le programme, insérant des instructions supplémentaires.

Les ordres Bip, BipNfois #nombre, CliNfois #nombre sont pratiques pour aider au dépannage de longs programmes. On n'a jamais parlé de ces ordres. Faites des petits programmes pour les tester, ou rajoutez ces instructions dans un programme précédent.

On peut insérer ces ordres pour vérifier que le programme passe bien là où on croit. On peut les enlever quand c'est au point.

Attention aux # manquants. Un opérande manquant est parfois signalé, parfois un paramètre par défaut est utilisé. Une adresse oubliée est signalée par un **Illegal symbol in expression**, sauf dans l'ordre FinRepete, où c'est à l'exécution que des bips se font entendre avant de tout bloquer.

Les moteurs

La vitesse des moteurs est commandée par des impulsions de largeur variable, mais il faut des fréquences élevées et régulières que le processeur génère par interruption : toutes les millisecondes, le processeur ne s'occupe plus de votre programme, mais du programme qui définit la vitesse des moteurs. Quand ces interruptions sont activées, votre programme est ralenti (imaginez votre efficacité si vous recevez un téléphone toutes les 5 minutes). Cela s'entend bien avec les sons, mais c'est peu visible avec les clignotements.

Pour faire tourner les moteurs, il faut mettre au début l'ordre ActiveMoteurs et utiliser ensuite l'ordre Vitesse #nombre,#nombre. Ces deux nombres sont les vitesses des moteurs gauche et droite, dans l'ordre. La valeur zero est la vitesse nulle, et il y a 7 vitesses positives et 7 vitesses négatives. Pour que le Bimo tourne sur lui-même à pleine vitesse, il faut écrire l'instruction

Vitesse #7,# -7

Si on donne une vitesse supérieure à 7 ou -7, la valeur prise par le programme est le reste de la division par 8. La vitesse 16 est nulle !

Même si le programme s'arrête à cause de l'instruction Stop, le moteur va toujours tourner. Les interruptions se font toujours. Il faut assigner la vitesse nulle ou couper les interruptions, mais on n'a pas prévu d'ordre pour ça dans notre langage simplifié.

Pour vérifier, faites des programmes simples qui démarrent et arrêtent les moteurs. Après programmation, il faut naturellement déconnecter le programmeur et activer la batterie du Bimo, si on veut vérifier le déplacement.

<pre>\prog;B13 Avance-recule .Ins BimoDef.asi Var Vit .Ins Bimolnit.asi .Ins BimoProg.asi ActiveMoteur Boucle: \b;Avance en accélérant Leds RR : Vitesse initiale minimum Copie #1,Vit AA: Repete #7 ; augm vitesse Vitesse Vit,Vit Augmente Vit Attente #3 Bip FinRepete AA</pre>	<pre>(suite) \b;Reculé en accélérant Leds VV Copie #-1,Vit RR: Repete #7 ; augm vitesse Vitesse Vit,Vit Diminue Vit Attente #3 Bip FinRepete RR AllerA Boucle Nom "B 1 3" .End</pre>
---	--

Pour programmer des figures, il suffit d'aligner les ordres de vitesse et d'attente.

Exercices : Programmer une étoile, un polygone.

Exemple plus compliqué : un détecteur de réflexe.

L'ordre Hasard Variable,ValeurMax donne un nombre entre 1 et ValeurMax<255 dans la variable (par défaut ValeurAuHasard si on ne précise pas). On peut utiliser cet ordre pour faire un retard imprévisible et allumer une LED. Il faut alors presser sur le poussoir le plus rapidement possible. Un compteur tourne pour mesurer le temps d'attente, et lorsque l'on a pressé, ce temps d'attente commande un nombre de bips avant de recommencer.

La structure générale est donc :

- Activer le poussoir pour recommencer une partie
- Eteindre la LED, lire le nombre au Hasard et démarrer l'attente
- Allumer la LED quand l'attente est terminée
- Compter (variable Temp) toutes les 10ms (par exemple en faisant un son) et tester le poussoir
- Si le poussoir est activé, jouer le nombre de bips dans la variable Temp.
- Recommencer

Si le poussoir n'est pas activé dans les 2 secondes, réagir avec un sirène (par exemple) et recommencer.

Le programme se trouve sous **Br.asm**

Dé électronique (1 à 6 bips)

Chaque fois que l'on presse sur le poussoir, on veut 1 à 6 bips. L'instruction Hasard ,#6 donne un nombre entre 1 et 6 dans ValeurAuHasard. C'est facile de clignoter ou bipper cette valeur quand on a agi sur le poussoir. Le programme se trouve sous **Bc.asm**.

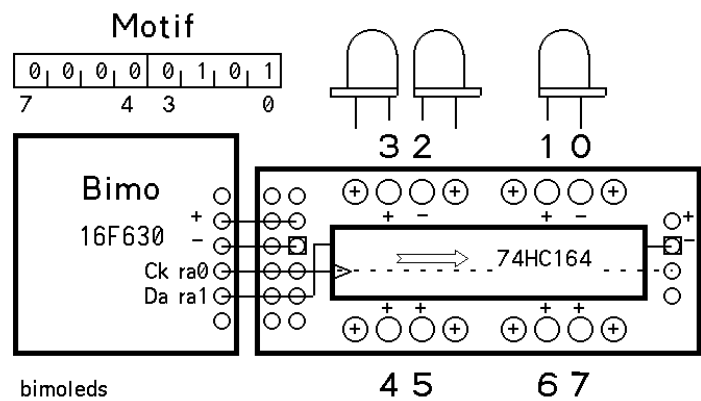
Jouez 100 fois en mettant des coches pour chaque valeur. Est-ce du bon hasard ?

BimoLeds – kit avec 8 Leds branchées sur le connecteur de programmation du Bimo

Pour le montage de ce kit, voir <http://www.bricobot.ch/docs/BimoLeds.pdf>

Le logiciel agit sur le mot de 8 bits qui correspond aux LEDs branchées sur le registre à décalage.

On déclare un motif binaire, écrit 2'0000101 par exemple. Les "1" allument les LEDs monocouleurs. Les paires 10 et 01 allument les LEDs bicouleurs. Avec le motif ci-contre, la LED bicouleur est verte car le courant sort par la patte longue (côté arrondi).



Les ordres disponibles sont

Allume #Motif, #Duree
Clignote #Motif, #Duree, #NbDeFois
Varie #Motif, #Duree, #NbDeFois

Par exemple, le programme B21 s'écrit :

```
\prog;B21|Effet sur BimoLeds
.Ins BimoDef.asi
.Ins BimoInit.asi
Boucle:
  Allume #2'11111111,#10
  ; allume tout si unicolor
  Varie #2'10011001,#3,#2
  Clignote #2'01100110,#3,#2
  AllerA Boucle
.Ins BimoFin.asi
```

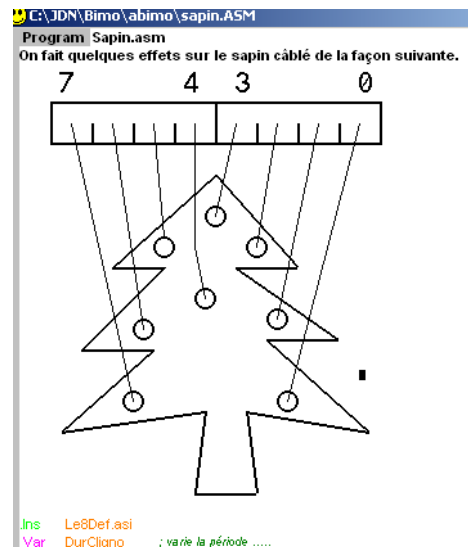
Conseils

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 longs programmes, un commentaire doit annoncer un groupe d'instructions.

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 soit ce texte soit le dessin

```
\prog:Sapin.asm
;On fait quelques effets sur un sapin câblé de la
façon suivante sur les sorties de BimoLed.
\image:Sapin.bmp
.Ins BimoDef.asi
Var DurCligno ; varie la période
.Ins BimoInit.asi
.Ins BimoProg.asi
.....
```



Syntaxe et résumé des instructions

Noms des constantes et variables: commencent par une lettre, pas d'espace dans le nom

Nombres : décimal 0..255

Nombres et constantes toujours précédés du signe # (valeur)

Assignment des constantes: Toto = 3

Etiquettes suivies d'un : (sans espace)

Déclaration des variables: Var Nom

Structure d'un programme :

```

.Ins      BimoDef.asi
...variables supplémentaires éventuelles
.Ins      BimoInit.asi
...tables
.Ins      BimoProg.asi
... programme
Nom      "N o m"
.End

```

<p>Copie <i>Const/Var, Variable</i> (Source --> Destination)</p> <p>Inverse <i>Var</i></p> <p>Attente <i>#Const/Var</i> Durée mult de 0.1 s max 255= 25.5 s</p> <p>Att50 <i>#Const/Var</i> Durée mult de 50 us</p> <p>Leds xy (VV RR 0V V0 0R R0 00)</p> <p>CliNfois <i>#Const/Var</i></p> <p>Bip BipNfois <i>#Const/Var</i></p> <p>SireneMonte SireneDescend</p> <p>HP0n HpOff</p> <p>Vitesse <i>#Const/Var, #Const/Var</i> VitG, VitD de -7 à +7 0= arrêt</p>	<p>Stop</p> <p>AllerA <i>Etiqu</i> Continue à l'adresse Etiqu :</p> <p>SiEgalAllerA <i>#Const/Var, Variable, Adresse</i></p> <p>SiPlusGrandAllerA <i>#Const/Var, Variable, Adresse</i></p> <p>Div2 Var Mul2 Var</p> <p>Hasard (valeur dans ValeurAuHasard)</p> <p>SiPoussoirAllerA Adresse</p> <p>AttendPoussoir</p> <p><i>Etiqu:</i> Repete <i>#Const/Var</i> nombre de fois</p> <p>Finrepete <i>Etiqu</i></p> <p><i>Extension pour BimoLeds :</i></p> <p>Allume <i>#Motif, #Duree</i></p> <p>Clignote <i>#Motif, #Duree, #NbDeFois</i></p> <p>Varie <i>#Motif, #Duree, #NbDeFois</i></p>
--	---

Une initiation à l'assembleur vous intéresse ? voir <http://www.epsitec.com/dauphin/>
Cett initiation vous permet de passer à l'assembleur du Bimo ou BimoPlus <http://www.didel.com/Dauphin3.pdf>
Vous voulez apprendre l'assembleur du 16F87x, similaire au 16F630 ?
<http://www.didel.com/picg/picg87x/CoursPicg87x.html>

Jdn 080904/090219/090418