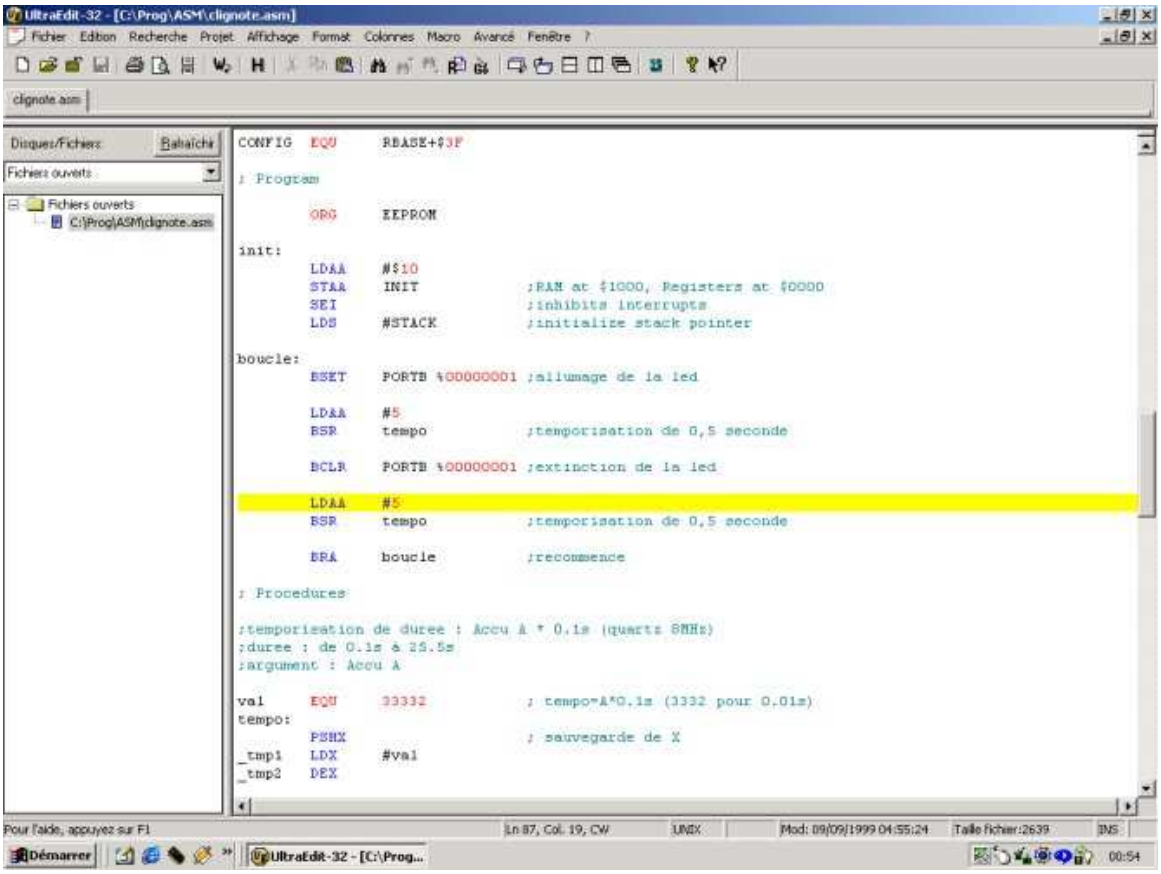
Manuel du compilateur as11

Nous allons voir ici comment se servir de l'exécutable as11.exe, qui permet de compiler votre source (le fichier .asm) en un fichier normalisé ayant pour extension .s19. Ce type d'extension, S-Record, est propriétaire à Motorola et est nécéssaire pour pouvoir télécharger votre programme dans le microcontrôleur. Je vous ai préparé un petit kit de programmation qui contient :

- as11.exe : l'exécutable proprement dit
- manuel.txt : ce manuel
- compile.bat : un fichier pour compiler facilement votre source
- WORDFILE.TXT : un fichier du logiciel UltraEdit pour avoir la coloration syntaxique
- model.asm : la structure minimale d'un programme asm, à compléter bien sûr :)



Oubliez le DOS ! UltraEdit est l'outil parfait pour programmer.

Options de la ligne de commande

programme est maintenant terminé. Le but va être de transformer les mots clés, utiles lors de la création du amme, en chiffres compréhensibles par le downloader (pour télécharger le programme dans le 68HC11).
cela, motorola met à disposition des utilisateurs un compilateur qui va se charger de tout le boulot : c'est exe. Votre programme est du style mon_prog.asm. Pour le compiler, il faut ouvrir une fenêtre DOS et entrer la de commande suivante :

exe fichier1 fichier2 ... [- option1 option2 ...]

ptions possibles sont les suivantes :

tive la sortie du listing
désactive la sortie du listing (defaut)
énère la table des symboles utilisés
idem ci dessus plus la ligne des instructions utilisées
ctive le comptage de cycle
désactive le comptage de cycle

exemple : c:\prog\as11.exe mon_prog.asm -l c

Comme vous l'aurez remarqué, on peut très bien compiler plusieurs fichiers à la fois. Dans ce cas, le fichier S-Record (.S19) résultant portera le nom du premier fichier.

exemple : c:\prog\as11.exe mon_prog.asm procédure1.asm fichier2.asm -l s

Cela génère le fichier hexadécimal sous le nom de mon_prog.s19.
Astuce : si vous voulez consulter le listing (c'est un compte rendu présentant le programme et sa traduction ligne par ligne en hexadécimal) plus facilement, complétez la ligne de commande par l'instruction DOS suivante : > listing.txt.

exemple : c:\prog\as11.exe mon_prog.asm -l > listing.txt

Vous pouvez utiliser le fichier BAT fournit dans le kit (téléchargeable ci-dessus). A ce moment là, écrivez simplement, par exemple : c:\prog\as11.exe mon_prog (sans l'extension du fichier).

Votre programme aura (en fait, sûrement ;) des erreurs, soit de typographie, de "grammaire" ou autre. Le compilateur génère dans ce cas un fichier nommé STDOUT ou alors les erreurs sont indiquées dans le listing juste avant la ligne qui contient cette erreur. L'écriture est alors :

Numéro de la ligne: Description de l'erreur
ou
Numéro de la ligne: Warning --- Description de l'erreur

Traitement d'une compilation

Décortiquons l'exemple suivant :

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   petite demo
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

RAM       EQU   $1000
STACK     EQU   RAM+$FF   ;initialise le pointeur de pile a la fin de la RAM
EEPROM    EQU   $F800

; Registres

RBASE   EQU   $0000
     'A   EQU   RBASE+$00
          EQU   $103D   ; fixe avant utilisation

     gramme

     RG   EEPROM


     LDAA   #$10
     STAA   INIT   ;RAM en $1000, Registres en $0000
     SEI    ;interruptions inhibees
     LDS    #STACK   ;initialise le pointeur de pile

     t:  LDAA   #$FF
         STAA   PORTA
```

on compile ce source avec la commande *as11.exe demo.asm -l > demo.txt*. On demande au compilateur de nous ir un listing (option -l) qui est très utile : il nous montre comment il a compilé et à quelles adresses il a mis s les variables. Si une adresse est fausse, n'allez pas plus loin : inutile de télécharger de .S19 généré car il est

faux. Voici ce fichier, demo.txt dans notre exemple :

```
Freeware assember ASxx.EXE Ver 1.03.
0001 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
0002 ;
0003 ; petite demo
0004 ;
0005 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
0006
0007 1000    RAM EQU $1000
0008 10ff    STACK EQU RAM+$FF ;initialise le pointeur de pile a la fin de la RAM
0009 f800    EEPROM EQU $F800
0010
0011 ; Registres
0012
0013 0000    RBASE EQU $0000
0014 0000    PORTA EQU RBASE+$00
0015 103d    INIT EQU $103D ; fixe avant utilisation
0016
0017 ; Programme
0018
0019 f800    ORG EEPROM
0020
0021 init:
0022 f800 86 10    LDAA #$10
0023 f802 b7 10 3d    STAA INIT ;RAM en $1000, Registres en $0000
0024 f805 0f    SEI ;interruptions inhibees
0025 f806 8e 10 ff    LDS #STACK ;initialise le pointeur de pile
0026
0027 f809 86 ff    Debut: LDAA #$FF
0028 f80b 97 00    STAA PORTA #$FF


Number of errors 0
```

Voilà un fichier bien intéressant : la première colonne montre les numéros de lignes, utiles pour trouver les erreurs éventuelles. La deuxième colonne indique à quelle position dans la mémoire le compilateur a placé l'instruction correspondante sur la même ligne. Enfin, la troisième colonne montre *l'opcode* de l'instruction, c'est à dire de sa traduction en hexadécimal (et le second membre éventuel). Les opcodes de chaque instruction se trouvent dans les datasheets du 68HC11.

Le troisième fichier généré est sans doute le plus important car c'est lui que va utiliser le downloader (PCBUG11 est la version officielle de Motorola mais il en existe bien d'autres) afin de programmer la mémoire du 68HC11 (ou une mémoire externe si le composant est dépourvu d'une (E)EPROM interne). Voici le contenu du fichier DEMO.S19 :

```
S110F8008610B7103D0F8E10FF86FF970095
      0000FC
```

voyez que l'on retrouve les opcodes de notre programme. Bien sûr, il est possible (!) de modifier ce fichier un fichier texte classique), à condition de savoir ce que l'on fait. Ainsi, à partir d'un langage évolué (asm et e mieux, le C), on code directement chaque bit de la mémoire. Bien sûr, l'assembleur et -dans une plus grande re- le C offrent une vitesse et une souplesse de développement incomparables.

Syntaxe et structure d'un programme assembleur

ode est donc tapé dans un fichier texte et contient des caractères ASCII. La structure d'un programme est ituée d'une suite d'instructions écrites ligne par ligne. Chaque ligne ne peut contenir que 4 champs au mum, séparés par au moins un espace. Ces champs sont :

label
e opération (instruction ou directive de compilation)

- Une opérande
- Un commentaire

Le label

Un symbole commençant à la première colonne est un label, et peut être éventuellement terminé par deux points (:). C'est un mot, généralement explicite, qui permet de signifier le commencement d'une partie particulière du programme : le début, la fin, un sous-programme, une procédure etc. Cependant :

- Si le premier caractère de la ligne est un astérisque (*), le reste de la ligne est vue comme un commentaire ; tout ce qui est écrit n'est pas pris en compte par le compilateur et est laissé à titre indicatif.
- Si la première colonne est constituée d'un (ou plusieurs) espace ou tabulation elle n'est pas prise en compte. Cela permet généralement de bien indenter le programme.
- Les symboles peuvent être constitués de lettres, en minuscule ou en majuscule, de digits (0 à 9) ou de caractères spéciaux (dollar $, point ., souligné (underscore) _). Les symboles sontiennent de 1 à 15 caractères, dont le premier doit être une lettre ou un caractère spécial. Mais **attention**, le compilateur fait la différence entre les minuscules et les majuscules. Enfin, un programme ne doit pas contenir plus de 2000 labels différents.

Ainsi, d'après les règles ci-dessus, les deux codes suivants sont identiques :

Source 1	**Source 2**
Label: DECA	Label DECA
BNE Label	BNE Label

L'opération

Le champ opération suit le champ Label ; ces deux champs doivent être séparés par au moins un espace. L'opération doit être une instruction ou une directive **connue** par le compilateur. Il est possible d'écrire une opération en minuscule ou en majuscule, sachant que de toute manière le compilateur transforme la casse en minuscule avant de tester la validité de l'instruction. Notez qu'il est plus clair d'écrire ces opération en majuscule. Ainsi, le mnémonique 'nop' peut s'écrire indifféremment 'NOP', 'NoP' etc. Comme nous l'avons dit plus haut, l'opération est de deux types :

- L'instruction (ou le mnémonique) : cela correspond au langage propre au microcontrôleur. Notez que le registre apparait quelques fois à la fin de l'instruction. Atention donc à ne pas confondre 'CLRA' qui efface l'accumulateur A et 'CLR A' qui efface la position mémoire A. L'espace est donc très important ici.
- La directive : cela correspond à des ordres pour diriger la compilation : définir un emplacement mémoire où placer le code par exemple.

L'opérande

ıamps est complètement lié au champ opération ; c'est en sorte le second membre de l'instruction totale formée ɔpération et de l'opérande. Et comme plus haut, au moins un espace doit séparer les deux champs (une ation est préférable afin d'aligner tous les champs de cette troisième colonne). L'opérande est constituée de ɔles et de mots, quelques fois séparés par une virgule. Le mode d'adressage du 68HC11 est alors dépendant du ɔle utilisé (voir le cours correspondant aux modes d'adressages).

ıat de l'opérande :	**Mode d'adressage :**
ıe opérande	accumulateur et inhérent
ression>	direct, étendu et relatif
:pression>	immédiat
ression>,X ou Y	indexé avec le registre X ou Y

sûr remplacez <expression> par une expression valide, c'est à dire sans mettre les crochets : seuls sont ɔtés les lettres (minuscules ou majuscules, la casse étant gérée), le point (.), le souligné (underscore _) et le - ($). Le nombre de caractères est limité à 15. Un programme ne doit pas contenir plus de 2000 expressions

différentes.

L'opérande peut être une **constante** : une constante est une valeur qui ne change pas durant l'exécution du programme. L'assembleur 68HC11 autorise 5 formats de constantes (le défaut étant le décimal) :

- **ASCII** : un caractère unique est exprimé en ASCII si celui-ci est précédé d'une quote (').
Exemples : 'A 'p '1 '* sont valides mais 'LORD_ASRIEL est trop beaucoup trop long (11 caractères) et renvoie une erreur.

- **Hexadécimal** : un nombre est exprimé en hexadécimal s'il est précédé d'un dollar ($). Le nombre doit être compris entre $0000 et $FFFF.
Exemples : $69 $ACDB $08FA sont valides mais $U37 (U n'est pas hexadécimal) ACDB (pas de dollar) et $6B44F9 (plus de 4 caractères après le dollar) renvoient une erreur.

- **Octal** : un nombre est exprimé en octal s'il est précédé d'une arobase (@). Le nombre doit être compris entre @0 et @177777.
Exemples : @1234 @7 @177760 sont valides mais @125546722 (plus de 6 digits) @1239 (un digit dépsse 7) et @277777 (en dehors des limites) retournent une erreur.

- **Binaire** : un nombre est exprimé en binaire s'il est précédé d'un poucentage (%). Il doit faire au maximum 16 digits composés exclusivement de zéro et de un.
Exemples : %1 %1001110110001100 %001 sont valides mais 0001101110 (pas de %) %110101110000010111110110 (plus de 16 digits) et %1001254 (contient autre chose que 0 ou 1) renvoient une erreur.

- **Décimal** : il n'y a pas de préfixe. Le nombre doit être compris entre 0 et 65535 (5 digits au maximum).
Exemples : 12 3655 sont valides mais 9436879 (plus de 5 digits) et 37.9 (caractère non valide) renvoient une erreur.

Personnellement je ne vois pas très bien l'utilité d'utiliser l'octal. Le binaire est privilégié lors des opérations de masquage (on voit tout de suite quel digit est affecté ou non par le masquage) alors que l'hexadécimal est le plus répendu (c'est une représentation plus courte du binaire) avec le décimal (on utilise souvent le décimal quand il s'agit d'un compteur par exemple, ce qui est plus naturel).

Le commentaire

La dernière colonne est réservée à un commentaire éventuel ce qui s'avère pratique, voire indispensable. Comme dans tout langage, le commentaire n'est pas compilé mais est ignoré pendant la phase de compilation. Comme toujours maintenant, le commentaire doit être espacé d'au moins un espace (ou tabulation) avec l'opérande. Un commentaire est réalisé en utilisant le caractère astérisque (*) ou point-virgule (;). Notons que dans ces cours on utilisera excusivement le point-virgule étant donné que c'est un signe couramment employé en assembleur (sur les autres plates-formes).

```
HPRIO   EQU     RBASE+$3C
INIT    EQU     $103D           ; fixed before use
TEST1   EQU     RBASE+$3E
CONFIG  EQU     RBASE+$3F

; Program

        ORG     EEPROM

init:
        LDAA    #$10
        STAA    INIT            ;RAM at $1000, Registers at $0000
        SEI                     ;inhibits interrupts
        LDS     #STACK          ;initialize stack pointer

boucle:
        BSET    PORTB %00000001 ;allumage de la led

        LDAA    #5
        BSR     tempo           ;temporisation de 0,5 seconde

        BCLR    PORTB %00000001 ;extinction de la led

        LDAA    #5
        BSR     tempo           ;temporisation de 0,5 seconde

        BRA     boucle          ;recommence
```

Voici les quatre colonnes ; remarquez comme l'indentation facilite la lecture.

Les directives

Comme nous l'avons vu, nous avons la possibilité (nan, on doit en fait ;) d'utiliser des directives de compilation. Voici la liste, elles ne doivent pas obligatoirement toute se trouver dans le programme mais ORG est lui obligatoire.

- ORG : définit l'adresse du commencement du programme qui suit. Le programme est soit en mémoire externe, soit en interne ; dans tous les cas il faut dessiner le plan mémoire de son système et définir quels sont les endroits où se situent le début du programme, la RAM, les vecteurs d'interruptions etc.
Exemple : ORG $F800

- EQU : défini une équivalence, c'est à dire une constante (à la manière d'un #define en C). Très utile pour faciliter la programmation, la relecture et le debuggage.
Exemple : RAM EQU $1000

Utilisez les directives suivantes pour réserver des emplacements mémoires, c'est à dire des variables. Ces directives sont équivalentes aux int, char... en C.

Z Block storage of zero; single bytes
3 Form constant byte
C Form constant character string
B Form constant double byte
L Initialize a block of memory to a constant
B Reserve memory; single bytes
B Zero memory bytes; same as BSZ

ıples :

MB 1 ; réserve 1 octet
_Asriel RMB 2 ; réserve 2 octets
age FCB
$77,$77,$2E,$70,$72,$6F,$67,$72,$61,$6D,$6D,$61,$74,$69,$6F,$6E,$77,$6F,$72,$6C,$64,$2E,$63,$6F,$6D

ıez ce qui est écrit en hexadécimal ? Réponse : www.ezzaitouni.moh.com bien sûr !! ;)

[Introduction | Top | Annexe suivante]

par ezzaitouni mohammed
Dernière mise à jour: 03/01/2009