

- GEI 455 -  
Systèmes en temps réel

Utilisation du système de développement croisé SDS

Ce document décrit sommairement l'utilisation du système de développement croisé SDS exploité dans l'environnement Windows 95 pour produire les logiciels demandés en laboratoire. La plate-forme cible est le kit d'évaluation 68331. La version utilisée du logiciel SDS est une version de démonstration qui présente certaines limitations qui restent sans effets majeurs dans le cadre des laboratoires du cours.

Les outils en mode ligne de commande, donc dans une fenêtre DOS, seront utilisés car ils permettent la production d'une image mémoire en format Motorola S1-S9 qui peut être téléchargée dans le kit 68331. Les logiciels à développer le seront en C (ou en C++ selon votre choix) sauf pour quelques fonctions qui devront être écrites en assembleur. L'environnement de développement intégré de SDS pourra être utilisé mais le code produit ne semble pouvoir être exécuté que sur le simulateur.

Pour avoir accès aux outils en mode ligne de commande le répertoire `pc/windows/sds70/cmd` doit être ajouté dans le chemin (*path*). Il est également nécessaire de définir les symboles INC et LINKER au niveau de l'interpréteur (*shell*) de DOS; ces opérations sont réalisées dans le fichier de commandes indirectes `config.bat` du répertoire `cours/gei455/sds`.

Il est possible d'obtenir une description très sommaire des options de chaque outil à l'aide de l'option `-U`. Une description sommaire de ces outils est donnée dans les fichiers `readme` et `crosscod` du répertoire `sds70`. De plus une copie de la documentation des outils SDS intitulée *CrossCode User Guide* est disponible dans le laboratoire 303.

Procédure de développement

Les étapes du cycle de développement sont les suivantes:

- Édition du fichier source (`.c` ou `.cpp`);
- Compilation du fichier source pour produire un module objet (`.o`);
- Édition du fichier source assembleur (`.s`);
- Assemblage du fichier assembleur pour produire un module objet (`.o`);
- Édition des liens à partir des différents modules objets pour produire une image mémoire binaire (`load.out`);
- Traduction de l'image mémoire binaire pour produire un fichier format Motorola S1-S9 (`load.txt` ou `load.dwn`).

L'édition

L'édition de tous les fichiers source (`.c`, `.cpp`, `.s`, etc.) peut se faire en utilisant n'importe quel éditeur capable de manipuler les fichiers textes. On peut utiliser l'éditeur de DOS `Edit` ou bien `NotePad` ou `WordPad` dans Windows.

La compilation

Le compilateur est invoqué à l'aide de la commande `cc68000`. Il est possible d'obtenir un listing de l'assembleur généré par la compilation pour observer les instructions générées par le compilateur.

La ligne de commande à utiliser est la suivante:

```
cc68000 -V CPU32 -o list= exemple.c
```

Un listing de l'assembleur va être produit dans `exemple.lst`, et un fichier objet `exemple.o` sera également créé. L'option `-V CPU32` indique que le processeur utilisé est un CPU32.

### L'assemblage

Pour assembler du code source en assembleur de façon à produire un module objet il faut utiliser la ligne de commande suivante:

```
as68000 -V CPU32 exemple.s
```

Le module objet produit va être dans le fichier `exemple.o`. N'oublier pas de ré-assembler les modules que vous avez modifiés avant de les utiliser dans l'édition des liens (comme `start.s` ou `bug32.s`).

### L'édition des liens

L'édition des liens nécessite de spécifier tous les fichiers objets qui devront être intégrés au code. On y retrouve `start.o` qui est le code d'initialisation et de terminaison d'une application (indispensable en C et en C++); c'est ce code qui va appeler la fonction `main` de votre programme. On doit également y ajouter `bug32.o`, le module objet qui contient des fonctions d'accès aux services du moniteur `cpu32bug` présent dans la ROM du kit 68331. Ces fonctions vont constituer une micro bibliothèque d'exécution (*micro run time library*) pour remplacer la bibliothèque d'exécution standard dont nous ne disposons pas. Pour faire connaître ces fonctions au code C on peut y inclure le fichier `bug32.h` qui contient la description de leur interface. Les fonctions développées en assembleur devront être ajoutées dans un module objet supplémentaire. Compte-tenu d'une limitation de l'éditeur des liens de la version de démonstration, il sera nécessaire d'intégrer le code assembleur au `bug32.s`.

Un exemple de ligne de commande à utiliser serait la suivante:

```
linker -f link.spc exemple.o start.o bug32.o -o load.out
```

Le fichier `link.spc` est un fichier texte, donc éditable, qui contient une description de l'organisation en mémoire que doit avoir l'image mémoire créée par l'édition des liens. On y trouve où placer le code, les données initialisées, les données non initialisées, etc. La copie du fichier `link.spc` présente dans le répertoire `cours/gei455/sds` place le code à partir de l'adresse \$3000 pour le kit 68331. Des copies des modules `start` et `bug32` (sources assembleur et objets) sont également présentes dans le même répertoire; `bug32` est un embryon à compléter. Les vecteurs de départ (*reset*) et d'exceptions ne sont pas placés dans l'image mémoire car ils ne seraient pas utilisés.

### La conversion de l'image mémoire

Un outil permet la conversion de l'image mémoire binaire issue de l'édition des liens en image mémoire chargeable pour le kit 68331. Le format utilisé le format S1-S9 de Motorola. La ligne de commande à utiliser est la suivante:

```
down -d mot load.out -m data,DATA -o load.txt
```

L'option `-d` mot spécifie que le fichier de sortie `load.txt` doit être en format S1-S9. L'option `-m data,DATA` va provoquer une relocalisation des données initialisées dans l'image mémoire. L'option `-o` permet d'attribuer le nom et l'extension désirée au fichier S1-S9 généré (sinon, l'extension `.dwn` est utilisée). Ces données seront recopiées à leur emplacement d'utilisation lors du processus d'initialisation dans le module `start`.

### Chargement de l'image mémoire

L'image mémoire en format S1-S9 peut être chargée dans le kit 68331. Remarquez que ce fichier (`load.txt`) est du texte. Pour cela il faut utiliser un émulateur de terminal (`HyperTerminal` dans Windows 95 ou `BccTerm` dans le répertoire `apps/gei435`). Après avoir donné la commande `LO` au moniteur `cpu32bug`, il faut donner la commande à l'émulateur de terminal d'envoyer le fichier texte `load.dwn`. Il peut être nécessaire d'entre un ou deux retour à la fin de l'opération pour obtenir à nouveau la séquence de sollicitation (*prompt*) de `cpu32bug`. En principe un `G 3000` dans le moniteur doit provoquer l'exécution de votre programme.

### Utilisation

Un exemple élémentaire est dans le fichier `cours/gei455/sds/test.c` avec les fichiers de commandes pour le construire `blctest.bat` ainsi que les fichiers de commandes pour assembler les fichiers `start.s` (`blstart.bld`) et `bug32.s` (`bldebug32.bat`), tous dans le même répertoire.

Pour les laboratoires, il suffit de vous créer un répertoire à partir de ces fichiers, et de modifier les fichiers `test.c` (pour les fonctions en C) et `bug32.s` (pour les fonctions en assembleur) en conséquence.

Une documentation complète du moniteur `cpu32bug` est disponible dans le répertoire `cours/gei435/332doc`.