Module PFSI

TD n° 9 - Programmation en langage d'assemblage

Objectifs:

*Savoir utiliser **un langage d'assemblage** et ses **directives**;*
*comprendre **l'assemblage d'un programme**;*
*utiliser les **adresses absolues ou relatives**;*
*écrire un programme **relogeable** et **translatable ;***
*gérer l'**initialisation***

On va réécrire le programme du TD n° 8 en langage d'assemblage, puis utiliser un assembleur qui va générer automatiquement le fichier de code machine exécutable qui avait précédemment été saisi manuellement.

1. Écriture du source en langage d'assemblage

1. **Saisir le programme en langage d'assemblage** dans un fichier "source" prog.src avec un éditeur de texte sous Unix, c'est à dire le contenu des colonnes "LABEL" et "ASM".

On insérera deux caractères de tabulation devant chaque mnémonique d'instruction afin de laisser un **champ** libre pour placer un **label** (dit aussi étiquette) quand nécessaire.

On ajoutera un **commentaire** aligné (après //) à la fin de chaque instruction avec ce qui avait été écrit originellement dans les colonnes "ACTION" puis "ETAT".

2. Directives d'assemblage

2.1. Compléter le programme avec les **directives d'assemblage ORG** et **START** indiquant l'adresse de chargement et l'adresse de démarrage (que l'assembleur placera en entête du fichier exécutable prog.iup).

2.2. **Remplacer les valeurs par des symboles** en utilisant la directive d'assemblage **EQU**
(e.g. SEUIL=3, RESET_ADDRESS=FFFA, LOAD_ADDRESS=FFB0) .

2.3. **Ajouter** des **étiquettes** (e.g. DEBUT, POSIT) et **utiliser** une **expression arithmétique constante** contenant le symbole **$** (dit "compteur de case d'assemblage" qui représente l'adresse de l'instruction où il se trouve) **calculant automatiquement le déplacement** dans les instructions de branchement relatif Jcc.

3. Assemblage

L'outil utilisé "assembleur-simulateur" est l'archive java microPIUPK.jar fonctionnant sur Linux (et Windows) développée par Karol Proch. Il est -entre autres- sur le serveur Neptune dans /home/depot/PFSI. En supposant que microPIUPK.jar est dans le chemin, et que prog.src est dans le "present working directory" on peut **assembler le fichier source assembleur prog.src** par la commande Linux, la clé -ass signifiant "assembler":

```
java -jar microPIUPK.jar -ass prog.src↵
```

On obtient alors le fichier exécutable sur simulateur prog.iup dans le "present working directory" .

Vérifier que ce fichier est exactement identique au fichier saisi manuellement dans le TD n°8.

4. Exécution

Lancer le simulateur (avec: `java microPIUPK.jar↵`) puis charger puis **exécuter prog.iup** (cf. aide intégrée).

5. Mode d'adressage direct

Utiliser LDW et STW en mode d'adressage direct pour accéder respectivement au port d'entrée et de sortie, avec moins d'instructions. (Dans le mode d'adressage direct, l'adresse de l'opérande est contenue dans le mot qui suit l'instruction). L'opérande est ici respectivement le contenu du port d'entrée ou de sortie.

6. Saut avec adresse relative ou absolue

6.1. **Reloger** le programme en code machine à une autre adresse, par exemple FFB0.
Pour ce faire, changer les adresses de chargement et de démarrage *après assemblage* dans le fichier exécutable, par exemple à FFB0, effacer la mémoire et recharger le fichier exécutable modifié.
Lancer l'exécution et vérifier que les sauts relatifs (e.g. Jcc) fonctionnent toujours, alors que le programme a été relogé ailleurs ! Le programme en code machine est donc **relogeable** (" relocatable").

6.2. Pour effectuer le saut de bouclage, à la place de :

`JMP déplacement` *// saute à l'instruction cible dont l'adresse est PC + l'opérande déplacement*

(***saut relatif***) utiliser maintenant **l'instruction :**

`JEA` `instruction_cible` *// saute à l'instruction cible qui est l'opérande*

(***saut absolu*** "Jump to Effective Address") dont l'opérande est l'instruction cible vers laquelle on branche. On utilisera le **mode d'adressage direct** : l'adresse de l'opérande (cet opérande est ici l'instruction cible) est alors contenue dans le mot qui suit l'instruction JEA.
Assembler puis tester ce programme: il doit fonctionner correctement puisque l'assembleur connaît l'adresse de l'instruction cible (début du programme) et a donc bien calculé le mot d'extension du JEA.
On note que l'adresse de l'instruction cible est absolue au lieu d'être relative au PC comme pour JMP *déplacement* (l'adresse de l'instruction cible serait alors PC + déplacement).

6.3. reloger le programme en code machine obtenu en 6.2 ailleurs comme en 6.1: on note que cette fois, le saut de boucle ne tombe plus sur la bonne instruction au début du programme (qui a bougé) mais toujours à l'ancienne adresse. Ce programme n'est donc pas relogeable: un système opérateur ne pourra donc pas le mettre tel que n'importe où en mémoire, ce qui compliquera le travail du système informatique.

7. Initialisation

Lorsque l'on clique sur "RESET", la machine démarre en FFFA et exécute l'instruction à cette adresse.
Du fait qu'il n'y a que des 0000 (code de l'instruction NOP qui ne fait rien qu'aller vers l'instruction suivante) à cet endroit, la machine effectue NOP NOP NOP puis sort de la mémoire après FFFE ... ce qui produit une erreur.

En FFFA, on devrait donc avoir un programme d'initialisation qui saute vers le début du programme de calcul précédent avec l'instruction d'initialisation "JEA @DEBUT" avec DEBUT = adresse de départ = FFB0h ici.

- On peut écrire directement le code de "JEA @" (i.e. 0950h) dans la case d'adresse FFFA puis l'adresse de début de programme de calcul (par exemple ici FFB0h) dans le mot (d'extension) suivant d'adresse FFFC.

- On peut aussi écrire un proramme d'initialisation init.src en langage d'assemblage qui saute vers le début du programme de calcul (en FFB0) avec un saut absolu JEA en mode direct, à placer et faire démarrer en RESETA=FFFA, puis l'assembler et le charger en mémoire (sans effacer la mémoire: les deux programmes co-existent).

Cette fois, quand on clique sur RESET, le programme d'initialisation effectue le saut vers le début du programme de calcul qui fonctionne alors correctement.