|  |  |
| --- | --- |
| Ordinateurs: structure et applications | Nom/Matricule : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  |
| GIF-1001, Hiver 2015 |  |

## Travail Pratique 2

## Modes d’adressage en assembleur ARM

# Ce travail pratique vaut 3% de la note totale du cours. À faire individuellement, il contient 20 questions obligatoires, comptant chacune pour 5 points, pour un total de 100 points. Le travail est à remettre au plus tard mardi le 10 février à 23h59. La procédure de retard détaillée dans le plan de cours s’applique.

## OBJECTIFS

# Ce travail pratique vise les objectifs suivants:

# Comprendre ce qu’est un code d’opération (« opcode ») et comprendre la relation entre le contenu de la mémoire, les instructions exécutées et les données en mémoire;

# Comprendre comment les variables et constantes sont représentées en code machine;

# Comprendre quelques instructions d’assembleur ARM;

# Approfondir la connaissance sur les microprocesseurs;

# S’initier au logiciel *IAR Embedded Workbench*

## PRÉPARATION

# Dans ce laboratoire, vous aurez à répondre à des questions relatives à un programme écrit en assembleur ARM. Pour ce faire, vous devrez tout d’abord installer le programme *IAR Embedded Workbench* sous Windows (les autres plateformes ne sont pas supportées). Rendez-vous à l’adresse <http://supp.iar.com/Download/SW/?item=EWARM-EVAL>, téléchargez et installez ensuite l’exécutable. Après l’installation, le gestionnaire de licence vous demandera d’enregistrer votre logiciel. Choisissez alors l’option « Kickstart, size-limited evaluation ». Vous pourrez alors utiliser le logiciel indéfiniment, avec des limitations imposées sur la mémoire disponible. Cette limitation ne causera pas de problème dans le cours.

# Après avoir installé le logiciel, suivez les étapes suivantes :

# Tout d’abord, créez un nouveau « workspace » : « File 🡪 New 🡪 Workspace ».

# Ensuite, créez un nouveau projet asm : « Project 🡪 Create new project », et sélectionnez le template « asm » sous l’option « asm ». Enregistrez le projet et le « workspace ».

# Vous devez remplacer le fichier asm.s dans votre « workspace » par le fichier tp2.asm sur le site du cours.

# Répondez ensuite aux questions ci-après.

# Vous devez obligatoirement décocher la case « Stack pointer(s) not valid until program reaches : » dans « Tools 🡪 Options 🡪 Stack ».

# Il peut aussi être utile d’afficher les numéros de lignes. Pour ce faire, suivez les étapes suivantes :

# Allez dans « Tools 🡪 Options ». Sous l’onglet « Editor », cochez « Show line numbers ».

## QUESTIONS

# Pour chaque question, le numéro de la ligne de code correspondante (dans le fichier asm.s) est inscrit entre parenthèses.

# 1. (ligne 13) Que se passe-t-il si l'on ne fait pas l'instruction B main (« branch ») ? Pourquoi ?

# 2. (ligne 16) Quelle est la dimension de la constante \_tableau ?

# 3. (ligne 17) Pourquoi met-on l'adresse d'une variable dans une constante ?

# 4. (ligne 26) Quelle est la valeur du registre PC (« Program Counter ») à cette instruction ? Rappel : la valeur affichée par le « débuggeur » de IAR ne représente pas exactement le comportement de PC lors de l’exécution du programme (voyez notes de cours)…

# 5. (ligne 27) Quelle est l’adresse de cette instruction ?

# 6. (ligne 28) Pourquoi faut-il soustraire 128 de PC pour mettre 0x00100000 dans R2 ?

# 7. (ligne 28) Pourquoi ne met-on pas l'adresse de la variable \_pointeur (0x00100000) directement dans R2 avec une instruction MOV ?

# 8. (ligne 28) Quelle est la représentation binaire de cette instruction ? Vous pouvez écrire votre réponse en hexadécimal.

# 9. (ligne 29) Quels bits de l'instruction désignent le numéro de registre de destination ?

# 10. (ligne 29) À quoi correspond la valeur mise dans R4 ?

# 11. (ligne 30) Quelle est la représentation binaire de cette instruction ? Vous pouvez écrire votre réponse en hexadécimal.

# 12. (ligne 31) Quelle variable est modifiée par cette instruction ?

# 13. (ligne 33) Quel(s) registre(s) sont affecté(s) par cette instruction ? De quelle façon ?

# 14. (ligne 34) Quel est le contenu de R2 après cette instruction ? Qu'est-ce que cette valeur représente ?

# 15. (ligne 35) Quelle est la taille de cette instruction ?

# 16. (ligne 36) Que vaut la variable \_pointeur après l’exécution de cette instruction ?

# 17. (ligne 37) Est-ce que la représentation binaire de cette instruction est la même que pour l’instruction en ligne 34 (soit : LDR R2, [R0, R5]) ? Expliquez pourquoi.

# 18. (ligne 38) Cette ligne contient une instruction placée volontairement en commentaire, car elle est illégale/invalide. Pourquoi ? Comment faudrait-il corriger l’instruction ?

# 19. (ligne 40) À quoi sert cette ligne ?

# 20. (ligne 50) À quelle adresse se retrouve \_variable ? Après avoir exécuté le programme, quel(s) registre(s) contienne(nt) cette adresse ?

# Bon travail !