TP 5

PENNSIM- Programmation, routines

Les exercices ci-dessous seront conçus sur papier *puis* testés à l'aide du logiciel PENNSIM.

Fichiers fournis : `tp5_mult6.asm`, `tp5_codage.asm`, `tp5_chaine.asm`, `tp5_saisie.asm`

5.1 Programmes plus complexes en langage d'assemblage LC-3

EXERCICE 1 ▶ **Multiplication par 6 des entiers d'un tableau**
Dans le fichier `tp5_mult6.asm`, vous devez compléter la routine `mul6tab` pour qu'elle multiplie les entiers d'un tableau par 6 modulo 16 : en entrée `R0` contient l'adresse de la première case du tableau, `R1` l'adresse de la dernière case. Vous traduirez pour cela le pseudo-algorithme suivant :

```
R2 <- R0
while( R2 <= R1 ) {  // (R2 <= R1) <=> (R2-R1 <= 0)
    R3 <- mem[R2];
    R3 <- 2*R3+4*R3;   // R3 <- 6*R3
    R3 <- R3 & 0x000F; // R3 <- R3 modulo 16
    mem[R2] <- R3;
    R2++;
}
```

1. Qu'est-il prévu dans le programme principal du fichier source `tp5_mult6.asm` ?
2. En utilisant uniquement `R4` comme registre intermédiaire, montrez comment traduire la ligne `R3 <- 2*R3+4*R3`.
3. A la ligne `R3 <- R3 & 0x000F`, le `&` désigne le AND bit-à-bit : justifier le fait que l'opération `R3 & 0x000F` calcule bien le reste dans la division euclidienne de `R3` par 16. Comment traduirez-vous cette ligne en langage d'assemblage ?
4. Traduisez l'algorithme proposé. Vous pouvez utiliser R4 et/ou R5 pour les calculs intermédiaires.
5. Exécutez pas-à-pas votre programme, et assurez-vous qu'il fonctionne correctement : vérifiez que la routine multiplie bien par 6 modulo 16 les entiers du tableau !

EXERCICE 2 ▶ **Un message codé**
Récupérez le fichier `tp5_codage.asm` sur la page web du cours. Il s'agit de compléter la routine `dechiffre` pour qu'elle permette de déchiffrer le message qui se trouve rangé à partir de l'adresse `msg` sous la forme d'une chaîne de caractères se terminant par 0. La routine prend comme paramètres l'adresse du début du message dans `R0`, et la clé du chiffrement `k` dans `R1`. Pour décoder, la routine remplacera chacun des caractères `c` du message par `k^c` (ou-exclusif bit-à-bit entre `k` et `c`), sauf le caractère 0 final.

1. Assemblez et exécutez une première fois le programme : quel est le message affiché ?
2. Si a et b sont deux variables booléennes, on rappelle que $a \oplus b$ désigne le ou-exclusif entre a et b. En utilisant les lois de Morgan, vérifiez que

$$a \oplus b = \overline{\overline{(\overline{a} \cdot b)} \cdot \overline{(a \cdot \overline{b})}}$$

3. On suppose que `R1` contient la clé et `R3` un caractère du message. Donnez un morceau de code en langage d'assemblage pour remplacer `R3` par `R1^R3`, en utilisant `R4` et `R5` comme variables intermédiaires du calcul.
4. En utilisant `R2` comme un pointeur pour parcourir le message, donnez sur papier un pseudo-code pour la routine `dechiffre`. Vous référencerez simplement le code de la question précédente par (*).

5. Complétez la routine `dechiffre` dans le fichier `tp5_codage.asm`, d'après le pseudo-code de la question précédente. Testez votre programme, en l'assemblant et en l'exécutant : quelle est la chaîne de caractères affichée ?
6. Comment faire pour coder un message avec la clé fournie ? Complétez le programme (`tp5_codage.asm`) pour tester votre proposition.

5.2 Entrées - sorties

EXERCICE 3 ► **Saisie d'une chaîne de caractères**
Récupérez `tp5_chaine.asm`. Le système d'exploitation du LC-3 fournit une interruption permettant d'afficher une chaîne de caractères (`PUTS` ≡ `TRAP x22`), mais on n'a pas la possibilité de saisir une chaîne de caractères. Le but est d'écrire une routine permettant cela. Vous compléterez progressivement le programme fourni.

Listing 5.1 – tp5_chaine.asm

```
          .ORIG x3000
; Programme principal
          LEA R6,stackend ; initialisation du pointeur de pile
          ; *** A COMPLETER ***
          HALT
; Pile
stack:    .BLKW #32
stackend: .FILL #0
          .END
```

1. Avant de déclencher une interruption vers le système d'exploitation dans une routine, il est important de sauvegarder l'adresse de retour contenue dans R7. Pourquoi ?
2. Écrivez une routine `saisie` permettant de saisir une chaîne de caractères au clavier, en rangeant les caractères lus à partir de l'adresse contenue dans R1. La saisie se termine lorsqu'un retour à la ligne (code ASCII 10) [1] est rencontré, et la chaîne de caractères doit être terminée par un caractère `'\0'` (de code ASCII 0).
3. Testez la routine en écrivant un programme qui affiche "Entrez une chaîne : ", effectue la saisie d'une chaîne en la rangeant à une adresse désignée par une étiquette `ch`, puis affiche "Vous avez tapé : " suivi de la chaîne qui a été saisie.

EXERCICE 4 ► **Saisie d'un entier au clavier**
Vous modifierez et compléterez progressivement le fichier `tp5_saisie.asm`.

1. La routine `saisie` permet de lire un entier naturel en base 10 au clavier, et place l'entier lu dans `R1`. Modifiez-la pour qu'elle affiche « `Entrez un entier naturel :` » avant d'effectuer la saisie.
2. Complétez la routine `aff` de façon à ce qu'elle affiche autant d'étoiles « `*` » que l'entier naturel contenu dans `R1` lors de son appel. Après avoir affiché `R1` étoiles, la routine doit aussi afficher un retour à la ligne. Suivez les consignes données en commentaire dans le fichier.
3. Modifiez le programme principal `main` pour qu'il effectue la lecture d'un entier au clavier avec `saisie`, puis son affichage avec `aff`.
4. La routine `mul10` fournie permet de multiplier par 10 le contenu de `R1`. Mais, telle qu'elle vous est fournie, elle exécute 10 instructions : modifiez cette routine de façon à ce qu'elle exécute au plus 6 instructions, tout en calculant toujours le même résultat.
5. L'adresse de retour contenue dans `R7` est sauvegardée et restaurée à l'aide de la pile dans `saisie` et `aff`, mais pas dans `mul10`. Pourquoi ?

1. Dans une ancienne version du simulateur, le retour chariot (code ASCII 13) était utilisé, d'où la présence de ce code dans certains des exercices de TD : en tout cas, avec LOGISIM, c'est bien le retour à la ligne qui est lu quand on tape « Entrée. »