TP 4

PennSim- Prise en main, introduction à l'architecture LC-3

Nous allons utiliser le simulateur LC-3 nommé PennSim dont la documentation est disponible à l'adresse :

`http://castle.eiu.edu/~mathcs/mat3670/index/Webview/pennsim-guide.html`

Fichiers fournis : `tp4_add_simple.asm`, `tp4_puts.asm`, `tp4_bin.asm`, `tp4_puts2.asm`, `tp4_loop.asm`, `lc3os.asm`

Téléchargez `PennSim.jar` sur `http://sylvain.brandel.pages.univ-lyon1.fr/archi/PennSim.jar` (ou sur la page web de PennSim) et mettez-le dans votre répertoire de TP. Le programme se lance de la même manière que Logisim (cf. Chapitre 1).

4.1 Jouons avec le simulateur de LC-3

Exercice 1 ▶ **Installation, documentation**

Commencez par récupérer `tp4_add_simple.asm` et `tp4_puts.asm`.

1. Lisez rapidement, mais attentivement, la documentation en ligne du logiciel.
2. Assemblez, chargez et exécutez pas-à-pas le programme de test `tp4_add_simple.asm` rappelé ci-dessous. Comme nous n'avons pas chargé l'OS, il faut amener PC à la première adresse du programme à l'aide de la commande `set PC x3000`. Observez l'évolution de l'état des registres et de la mémoire au cours de l'exécution du programme. Pourquoi est-il possible d'exécuter ainsi le programme sans charger l'OS ?

Listing 4.1 – tp4_add_simple.asm

```
        .ORIG X3000      ; spécifie l'adresse de chargement du programme
        LD R1,a
        LD R2,b
        ADD R0,R1,R2
        ADD R0,R0,-1
        ST R0, r
stop:   BR stop          ; bloque l'exécution ici (boucle sur soi-même)
r:      .BLKW 1
a:      .FILL 10
b:      .FILL 6
.END
```

3. Remettez le simulateur à zéro avec la commande `reset`. Assemblez et chargez l'OS, puis le programme `tp4_puts.asm`. Exécutez pas-à-pas le programme, et observez bien l'exécution de l'appel système PUTS : à quel programme appartiennent les instructions exécutées ?

Listing 4.2 – tp4_puts.asm

```
        .ORIG X3000      ; spécifie l'adresse de chargement du programme
        LEA R0,chaine
        PUTS
        HALT             ; rend la main à l'OS
chaine: .STRINGZ "hello\n"
.END
```

EXERCICE 2 ► **Exécution d'un programme en langage machine**
Soit le programme suivant, écrit en langage machine dans le langage machine du LC-3 (fichier `tp4_bin.asm`) :

Listing 4.3 – tp4_bin.asm

```
.ORIG X3000      ; where to load the program in memory
.FILL x5020
.FILL x1221
.FILL xE404
.FILL x6681
.FILL x1262
.FILL x16FF
.FILL x03FD
.FILL xF025
.FILL x0006
.END
```

Le décodage de ce programme se trouve Figure 4.1. Parcourrez rapidement ce décodage et répondez aux questions suivantes :

— À l'aide de quelles instructions récupère-t-on une donnée en mémoire dans ce programme ?
— Pouvait-on faire autrement ?
— Comment est réalisé le saut de compteur de programme pour réaliser la boucle ?
— Que deviennent les labels dans le programme assemblé ?

Ensuite, assemblez et lancez la simulation pas à pas sur le fichier `tp4_bin.asm`. Bien que l'on ait "assemblé" à la main, il faut quand-même effectuer avec la commande `as` la transformation en un fichier objet `.obj`. Suivez bien toutes les étapes lors d'une exécution pas-à-pas du programme. On remarquera que le simulateur LC-3 donne l'équivalent en langage d'assemblage des instructions machine considérées.

Adresse	Contenu	Contenu binaire	Détails des instructions	pseudo-code
x3000	x5020	0101 000 000 1 00000	AND R0, R0, 0	$R_0 \leftarrow R_0 \& 0 = 0$
x3001	x1221	0001 001 000 1 00001	ADD R1, R1, 1	$R_1 \leftarrow R_0 + 1 = 1$
x3002	xE404	1110 010 0 0000 0100	LEA R2, Offset9=4	$R_2 \leftarrow x3007$ (label fin)
x3003	x6681	010 011 010 00 0001	LDR R3, R2, 1	$R_3 \leftarrow mem[R2+1]$ (label donnee → $x3008$)
boucle:x3004	x1262	0001 001 001 1 00010	ADD R1, R1, 2	$R_1 \leftarrow R_1 + 2$
x3005	x16FF	0001 011 011 1 11111	ADD R3, R3, -1	$R_3 \leftarrow R_3 - 1$
x3006	x03FD	0000 001 1 1111 1101	BRp Offset9=-3	si $R_3 > 0$ aller à boucle
fin:x3007	xF025	1111 0000 0010 0101	TRAP x25	*HALT*
donnee:x3008	x0006	donnée	-	

FIGURE 4.1 – Un programme en binaire/hexadécimal (`tp4_bin.asm`)

Vous pouvez exécuter plusieurs fois le programme sans avoir à le recharger : pour cela, ramenez PC à l'adresse x3000 avec `set PC x3000`. D'autre part, vous pouvez modifier le contenu de la mémoire "à la main" : il suffit d'éditer le contenu de la colonne `Value` à l'adresse dont vous souhaitez modifier le contenu. En procédant ainsi, modifiez le programme pour qu'il effectue 8 tours de boucle et qu'il a ajoute 5 à `R1` à chaque itération.

EXERCICE 3 ► **Assemblage à la main**
Sur papier d'abord :

1. Écrivez un programme en langage d'assemblage LC-3 qui écrit 10 fois le caractère 'Z' sur l'écran.

2. Assemblez ce programme à la main, puis sur le modèle du Listing 4.3, créez un programme “pré-assemblé”.
3. Utilisez le simulateur pour tester votre programme. Attention, comme vous allez devoir faire appel au système d’exploitation du LC3, il faudra le charger en mémoire avant de tenter d’exécuter votre programme (télécharger, assembler et charger `lc3os`).

4.2 Écriture et simulation de programmes en langage d’assemblage LC-3

Jusqu’à présent nous avons écrit des programmes en remplissant la mémoire directement avec les codages des instructions. Nous allons maintenant écrire des programmes de manière plus simple, en écrivant les instructions en *langage d’assemblage LC-3*.

EXERCICE 4 ► **Exécution d’un programme donné**
Prévoyez le comportement des fichiers `tp4_puts2.asm` et `tp4_loop.asm`. Vérifiez avec le simulateur. Quelle est la différence entre les primitives PUTS et OUT, mises à votre disposition par le système d’exploitation ?

Listing 4.4 – tp4_puts2.asm

```
        .ORIG x3000       ; specify where to load the program in memory
        LEA R0,HELLO
        PUTS
        LEA R0,COURSE
        PUTS
        HALT
HELLO:  .STRINGZ "Hello, world!\n"
COURSE: .STRINGZ "Bienvenue en ARCHI\n"
        .END
```

Listing 4.5 – tp4_loop.asm

```
        .ORIG x3000
        LD R1,N
        NOT R1,R1
        ADD R1,R1,#1     ; R1 = -N
        AND R2,R2,#0
LOOP:   ADD R3,R2,R1
        BRzp ELOOP
        LD R0,STAR
        OUT
        ADD R2,R2,#1
        BR LOOP
ELOOP:  LEA R0,NEWLN
        PUTS
STOP:   HALT
N:      .FILL 6
STAR:   .FILL x2A        ; the character to display
NEWLN:  .STRINGZ "\n"
        .END
```

EXERCICE 5 ► **Min et max de deux entiers**
Écrivez un programme en langage d’assemblage LC-3 qui calcule le min et le max de deux entiers, et stocke le résultat à un endroit précis en mémoire, de label `min`. Testez avec différentes valeurs.