POLYTECH ANGERS

3A - Microcontrôleur
TD3 - Assembleur et architecture des ordinateurs avec Little Computer 3

Nicolas Delanoue

Exercice 1 (Prise en main du Simulateur)
L'ordinateur *Little Computer 3 (LC-3)* est un ordinateur minimaliste qui a été inventé pour la pédagogie. Il est un bon compromis entre puissance de calculs et simplicité. Malheureusement, il n'existe pas réalisation physique de cet ordinateur mais il existe des simulateurs comme `PennSim.jar`.

1. Télécharger et exécuter le programme java nommé `PennSim.jar` disponible via https://perso-laris.univ-angers.fr/~delanoue/polytech/microcontroleur/
   La figure 1 décrit l'interface graphique du logiciel de simulation d'un ordinateur LC-3 nommé PennSim.

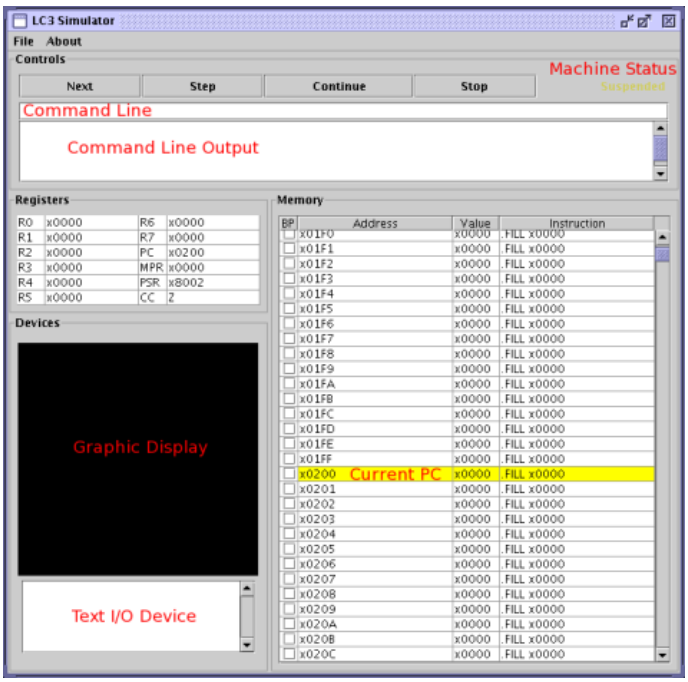

FIGURE 1 – Interface graphique du logiciel PennSim.

— La zone de texte *Command Line* permet d'exécuter des commandes.

— La zone de texte *Command Line Output* permet de connaître les résultats de commandes passées dans la zone précédente.
— Les 4 boutons *Next, Step, Continue et Stop* permettent contrôler l'exécution des programmes.
— Le label *Machine Status* permet de connaître l'état d'exécution de la machine.
— La zone nommée *Register* permet de connaître l'état des registres du processeur.
— La zone nommée *Memory* nous donne l'état de chaque case mémoire de notre ordinateur. De plus, la case mémoire surlignée en jaune permet de connaître la prochaine instruction qui va être exécutée (pointeur d'instruction aussi appelé PC pour Program Counter).
— La zone nommée *Devices* propose deux équipements un écran et une console permettant d'afficher des valeurs.

Il est possible d'en savoir plus sur le simulateur via le lien [1].

2. Télécharger le fichier `ex1.asm` [2] dont le contenu est le code suivant :

```
.orig x3000
ADD R1,R1,#5
ADD R2,R2,#4
ADD R3,R1,R2
.end
```

3. Dans la zone *Command Line*, exécuter la commande `as ex1.asm`.
4. Vérifier que cette commande a bien produit deux fichiers `ex1.obj` et `ex1.sym`.
5. Charger le programme `load ex1.obj` avec la commande `load ex1.obj`.
6. Observer l'état de la mémoire autour de la zone mémoire `x3000`.
7. Déplacer le pointeur d'instruction avec une commande `set PC x3000`.
8. Exécuter le programme en appuyant sur les boutons de l'interface.
9. Observer l'état des registres au fur et à mesure de l'exécution.
10. A la fin de cet exercice, je vous invite à taper la commande `reset` afin de réinitialiser les registres du microprocesseur ainsi que sa mémoire.

Exercice 2 (Avec des constantes)

1. Observer le code suivant :

```
.orig x0000
      LD R1,a0
      LD R2,a1
      ADD R4,R1,R2
      ST R4,a2
a0:   .FILL #12
```

1. https://www.cis.upenn.edu/~milom/cse240-Fall06/pennsim/pennsim-guide.html
2. Tous les fichiers à télécharger sont disponibles via l'url : https://perso-laris.univ-angers.fr/~delanoue/polytech/microcontroleur/

```
a1:    .FILL #10
a2:    .BLKW 1
a3:    .FILL #6
.end
```

2. Donner un sens à chacune des lignes.
3. Quelle sera la valeur stockée dans la case repérée par le label `a2` à la fin de l'exécution ?
4. Quelle sera la valeur stockée dans la case repérée par le label `a3` à la fin de l'exécution ?
5. Vérifier en assemblant puis chargeant le programme dans le simulateur.
6. Pour quelle raison le simulateur lève-t-il une exception à la fin de l'exécution ?

Exercice 3 (On recommence)

1. Observer et commenter le code suivant :

```
.orig x0000
        LD R0,a0
        LD R1,a1
debut:  ADD R0,R0,#1
        ADD R1,R1,#2
        ST R0,a0
        ST R1,a1
        BR debut
a0:    .FILL #0
a1:    .FILL #0
.end
```

2. Donner une trace de l'exécution de ce programme.
3. Traduire ce programme en c.
4. Tester dans le simulateur.
5. Combien de temps va durer l'exécution de ce programme ?

Exercice 4 (On recommence, enfin ça dépend.)

1. Le code suivant effectue un saut conditionnel

```
.orig x0000
        LD R0,a0
debut:  ADD R0,R0,#-1
        ST R0,a0
        BRp debut
a0:    .FILL #5
.end
```

2. Commenter chacune des lignes.
3. Traduire ce programme en c.

4. Donner une trace de l'exécution de ce programme.
5. Tester dans le simulateur.

Exercice 5 (Qui est le plus grand)
L'objectif de cette exercice est de trouver le maximum de deux valeurs a et b et de stoker le résultat dans la variable c. On donne le canevas suivant :

```
1  .orig x0000
2          LD R0,a
3          LD R1,b
4          NOT R1,R1     ; R1 <- NOT(R1)
5          ADD R1,R1,1   ; R1 <- R1 + 1
6                        ; Donc R1 contient -b
7          ADD R2,R1,R0
8
9          --------------------
10         LD R3,b
11         ST R3,c
12         BR FIN
13 ELSE:   LD R3,a
14         ST R3,c
15 FIN:    ADD R1,R1,0   ; NOP
16 a:      .FILL #4
17 b:      .FILL #6
18 c:      .BLKW 1
19 .end
```

1. Quel est le contenu du registre `R2` après l'exécution de l'instruction 6 ?
2. Dans quel état seront les registres `CC` avec les valeurs de a et b données ?
3. Quel est le rôle des instructions 9 et 10 ?
4. Quel est le rôle des instructions 12 et 13 ?
5. Compléter la ligne 7 de ce programme assembleur afin que la case mémoire numéro c contienne la valeur du plus grand des deux nombres.
6. Tester dans le simulateur avec différentes valeurs de a et b.

Exercice 6 (Fibonnacci)
Écrire un programme qui calcule le dixième élément de la suite de Fibonnaci définie par

$$u_0 = 1, u_1 = 1 \text{ et } u_{n+2} = u_{n+1} + u_n$$

Exercice 7 (Multiplication)
Écrire un programme qui calcule le produit de deux nombres a et b.

Exercice 8 (Intégrale)
1. Écrire un programme assembleur LC-3 qui calcule les sommes partielles d'une suite de 10 entiers écrits en mémoire (à la fin du programme), en écrasant cette même suite en mémoire. Par exemple, (5,2,10,3,6,120,120,10,0,1) est remplacée par (5,7,17,20,26,146,266,276,276,277).
2. Simulez-le.

Exercice 9 (Avec un système d'exploitation)
Le fichier `lc3os.asm` contient le code source d'un système d'exploitation minimaliste. Avec ce système d'exploitation, il est possible de réaliser des *appels systèmes*. On pourra alors facilement afficher des chaînes de caractères sur la console ou bien d'afficher des images sur l'écran.

1. Télécharger le et assembler le fichier `lc3os.asm`.
2. Charger en mémoire le fichier `lc3os.obj`.
3. Assembler et charger le programme assemblé suivant nommé `helloWorld.asm`.

```
1           ;;
2           ;; Author: Nicolas Delanoue
3           ;;
4           ;; A first assembly-language LC-3 program
5           ;;
6           ;; This is a variation on the world-famous
7           ;; "Hello, world" program
8           ;; known to many programmers.
9           ;;
10          ;; Notes:
11          ;; 1. HALT is equivalent to TRAP x25
12          ;; 2. PUTS is equivalent to TRAP x22
13          ;; 3. LEA (Load Effective Address) uses the
14          ;;    IMMEDIATE addressing mode
15          ;; 4. \n represents the "newline" character
16
17          .ORIG x3000          ; specify the "origin";
18
19          LEA R0,HELLO         ; R0 = address of output string
20          PUTS                 ; write("Hello, world!\n")
21
22          LEA R0,COURSE        ; R0 = address of output string
23          PUTS                 ; write("SAGI")
24
25          LEA R0,UNIV          ; R0 = address of output string
26          PUTS                 ; write("Polytech")
27          HALT                 ;
28
29  HELLO   .STRINGZ "Hello, world!\n"
30  COURSE  .STRINGZ "SAGI\n"
31  UNIV    .STRINGZ "Polytech\n"
32          .END
```

4. Exécuter et observer.
   Note : On prendra garde de bien placer le pointeur d'instruction.