USTL

INFO 202 – **2009/2010**

Architecture des ordinateurs : Fiche de TD 5

Lecture d'un code assembleur engendré par un compilateur

novembre 2009

1 Le jeu des Tours de Hanoï

En 1883, Édouard Lucas (1842–1891) publie sous le nom de Claus de Siam professeur au collège de Li-Sou-Tsian — anagramme de Lucas d'Amiens professeur à Saint-Louis — le jeu des Tours de Hanoï. Le jeu est constitué de trois piquets verticaux — 1, 2 et 3 — et de disques superposés de tailles strictement décroissantes enfilés autour du piquet 1 ; ces disques forment les fameuses tours. Il faut déplacer l'ensemble des disques pour que ceux-ci se retrouvent enfilés autour du

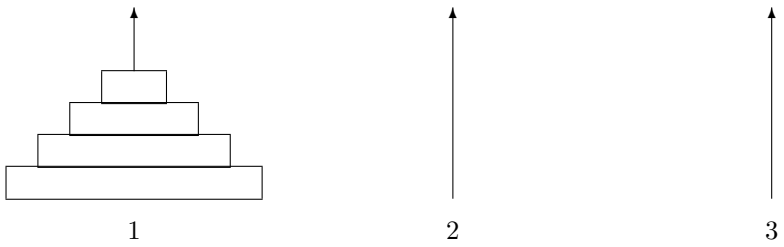

Figure 1: L'état initial du jeu des Tours de Hanoï pour $n = 4$

piquet 3 en respectant les règles suivantes :

- les disques sont déplacés un par un ;
- un disque ne doit pas se retrouver au-dessus d'un disque plus petit.

Lucas a montré que le problème est toujours résoluble et que pour une tour de n étages, il faut au minimum $2^n - 1$ coups pour déplacer la tour. La légende veut que les bonzes d'Hanoï passaient leur vie à résoudre ce problème pour $n = 127$, ce qui leur permettait d'attendre l'écroulement du temple de Brahma et donc la fin du monde. Pour mémoire, $2^{127} - 1$ est un nombre de Mersenne dont Lucas a démontré la primalité.

2 Raisonnement par récurrence

L'idée de la récursivité telle que nous l'employons aujourd'hui est attribuable à Stephen C. Kleene (1909–1994). Cette notion offre une solution élégante au problème des Tours de Hanoï.

Principe. Supposons notre problème résolu pour $n - 1$ disques i.e. que l'on sache transférer $n - 1$ disques depuis le piquet i jusqu'au piquet j en respectant les règles du jeu. Il existe alors une solution simple pour transférer n disques dans les mêmes conditions :

1. on amène les $n - 1$ disques du haut du piquet i sur le troisième piquet — représenté par $6 - (i + j)$;

2. on prend le dernier disque du piquet i et on le met seul en j ;
3. on ramène les $n-1$ disques de $6-(i+j)$ en j.

Implantation. Le programme suivant `Hanoi.c` en language C permet d'afficher les mouvements élémentaires à accomplir pour déplacer n disques du piquet i au piquet j (une procédure principale est fournie).

```
void Hanoi(int n, int i, int j){
        if (n>0){
                Hanoi(n-1,i,6-(i+j))  ;
                printf("%d --> %d\n",i,j) ;
                Hanoi(n-1,6-(i+j),j)  ;
        }
}
```

```
#include stdio.h
main(){
        Hanoi(4,1,3) ;
 }
```

3 Étude du code assembleur engendré

Plutôt que d'écrire directement en assembleur, nous allons interrompre la compilation du code C ci-dessus au moment de la création du code assembleur avant l'assemblage binaire et l'édition de liens (commande `gcc -S Hanoi.c` dans notre cas).

```
        .file "Hanoi.c"                     ; noms du fichier contenant Hanoi() et main()
        .section .rodata                    ; cuisine interne
.LC0:   .string "%d -> %d\n"                ; la chaîne et son format communiqué à printf
        .text                               ; cuisine interne
        .align 2                            ; fin de l'entête du fichier obtenu par gcc -S Hanoi.c
.globl Hanoi                                ; exporte le label Hanoi pour permettre son appel
        .type   Hanoi,@function             ; cuisine interne
Hanoi:  pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        cmpl    $0, 8(%ebp)
        jle     .L2
        subl    $4, %esp
        movl    16(%ebp), %eax
        movl    12(%ebp), %edx
        addl    %eax, %edx
        movl    $6, %eax
        subl    %edx, %eax
        pushl   %eax
        pushl   12(%ebp)
        movl    8(%ebp), %eax
        decl    %eax
        pushl   %eax
        call    Hanoi
        addl    $16, %esp
        subl    $4, %esp
        pushl   16(%ebp)                    ; les paramètres de la chaîne passés à printf
        pushl   12(%ebp)                    ; sont rangés en ordre inverse (convention du C)
        pushl   $.LC0                       ; équivalent de push OFFSET string
        call    printf                      ; appel de la procédure printf
        addl    $16, %esp
        subl    $4, %esp
        pushl   16(%ebp)
        movl    16(%ebp), %eax
        movl    12(%ebp), %edx
        addl    %eax, %edx
        movl    $6, %eax
        subl    %edx, %eax
        pushl   %eax
        movl    8(%ebp), %eax
        decl    %eax
        pushl   %eax
        call    Hanoi
        addl    $16, %esp
.L2:    leave                               ; instruction du Pentium ; prépare la sortie
        ret
.Lfe1:  .size   Hanoi,.Lfe1-Hanoi           ; permet de déterminer la taille du code
        .align 2                            ; cuisine interne
```

Le code de la procédure principale est :

```
.globl main                     ; exporte le label main pour permettre son appel
        .type   main,@function  ; cuisine interne

main:   pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp        ; 8 est le nombre d'octets nécessaire
                                ; au stockage des variables locales
        andl    $-16, %esp      ; aligne esp sur un multiple de 16
        movl    $0, %eax        ;
        subl    %eax, %esp      ;
        subl    $4, %esp
        pushl   $3
        pushl   $1
        pushl   $4
        call    Hanoi
        addl    $16, %esp
        leave                   ; instruction qui rétablit les bonnes valeurs de esp et ebp
        ret
.Lfe2:  .size   main,.Lfe2-main ; permet de déterminer la taille du code
        .ident "GCC: (GNU) 3.2 (Mandrake Linux 9.0 3.2-1mdk)"
```

Bien que la syntaxe de ce code assembleur soit différente de celui étudié en cours (assembleur 80x86), les principes généraux sont les mêmes. Nous allons tout de même expliciter quelques différences.

Le Pentium dispose de registres 32 bits construits sur le principe suivant :

	16 bits	8 bits	8 bits
EAX		AH	AL
EBX		BX	

Ainsi, le registre EAX — désigné par %eax — se comprend comme un registre AX étendu ; on peut manipuler des données 8, 16 et 32 bits. Lorsque l'on manipule les registres étendus, les commandes usuelles sont affublées de la lettre l — `pushl` place donc 32 bits sur la pile. L'instruction `movl` est presque équivalente à l'instruction `mov` de l'assembleur 80x86 si ce n'est que les opérandes sont inversées : l'instruction `add esp,12` en assembleur 80x86 correspond ici à l'instruction `addl $12,%esp`.

L'entier 0 est codé par le symbole $0. La commande `8(%ebp)` est l'équivalent de la commande `[ebp+8]` de l'assembleur 80x86.

Le code fournit utilise la procédure `printf` classique. Cette procédure prend en argument sur la pile — dans l'ordre — une chaîne de caractères indiquant le texte à afficher et les formats, suivit des paramètres.

Pour finir, on indique que le registre `%ebp` contient l'adresse du bas de la pile et que le registre `%esp` contient l'adresse du haut de celle ci. Une adresse référence un mot dans la mémoire.

Questions.

1. De manière à comprendre le code assembleur fournit, compléter les commentaires associés au code en expliquant la fonction des instructions — commencer par la procédure principale ;
2. Expliciter l'évolution de la pile lors de l'appel de la procédure `Hanoi(3,1,3)`.

```
.globl main                     ; exporte le label main pour permettre son appel
        .type   main,@function  ; cuisine interne

main:   pushl   %ebp            ; sauve l'adresse du bas de la pile sur la pile
        movl    %esp, %ebp      ; crée un nouveau bas de pile
        subl    $8, %esp        ; 8 est le nombre d'octets nécessaire
                                ; au stockage des variables locales
        andl    $-16, %esp      ; aligne esp sur un multiple de 16
        movl    $0, %eax        ;
        subl    %eax, %esp      ;
        subl    $4, %esp
        pushl   $3              ; placer le piquet de destination dans la pile
        pushl   $1              ; placer le piquet de départ dans la pile
        pushl   $4              ; placer le nombre de disques dans la pile
        call    Hanoi           ; appel de la procédure Hanoi
        addl    $16, %esp       ; rends les paramètres de l'appel à Hanoi inaccessibles
                                ; 3 × 4 mots + subl = 16 mots
        leave                   ; instruction qui rétablit les bonnes valeurs de esp et ebp
        ret                     ; retour à l'appelant
.Lfe2:  .size   main,.Lfe2-main ; permet de déterminer la taille du code
        .ident "GCC: (GNU) 3.2 (Mandrake Linux 9.0 3.2-1mdk)"
```

```
        .file "Hanoi.c"                 ; noms du fichier contenant Hanoi() et main()
        .section .rodata                ; cuisine interne
.LC0:   .string "%d -> %d\n"            ; la chaîne et son format communiqué à printf
        .text                           ; cuisine interne
        .align 2                        ; fin de l'entête du fichier obtenu par gcc -S Hanoi.c
.globl Hanoi                            ; exporte le label Hanoi pour permettre son appel
        .type   Hanoi,@function         ; cuisine interne
Hanoi:  pushl   %ebp                    ; sauve l'adresse du bas de la pile sur la pile
        movl    %esp, %ebp              ; crée un nouveau bas de pile afin d'utiliser 16(%ebp)
        subl    $8, %esp                ; 8 est le nombre d'octets nécessaire
                                        ; au stockage des variables locales
        cmpl    $0, 8(%ebp)             ; si le mombre de disques est nul
        jle     .L2                     ; allez à la fin de la procédure
        subl    $4, %esp
        movl    16(%ebp), %eax          ; %eax= j, num. piquet de destination
        movl    12(%ebp), %edx          ; %edx= i, num. piquet de départ
        addl    %eax, %edx              ; %edx = i + j
        movl    $6, %eax                ; %eax= 6
        subl    %edx, %eax              ; %eax= 6 − (i + j)
        pushl   %eax                    ; empile 6 − (i + j) (passage de paramètres)
        pushl   12(%ebp)                ; empile i (piquet de départ)
        movl    8(%ebp), %eax           ; dépile n dans %eax
        decl    %eax                    ; %eax=%eax−1
        pushl   %eax                    ; empile n − 1 (passage de paramètres)
        call    Hanoi                   ; appel recursif de la procédure Hanoi
        addl    $16, %esp               ; rends les paramètres de l'appel à Hanoi inaccessibles
                                        ; 3 × 4 mots + subl = 16 mots
        subl    $4, %esp
        pushl   16(%ebp)                ; empile j (passage de paramètres)
        pushl   12(%ebp)                ; empile i
        pushl   $.LC0                   ; équivalent de push OFFSET string
                                        ; les paramètres de la chaîne passés à printf
                                        ; sont rangés en ordre inverse (convention du C)
        call    printf                  ; appel de la procédure printf
        addl    $16, %esp               ; rends les paramètres de l'appel à printf inaccessibles
                                        ; 3 × 4 mots = 16 mots
        subl    $4, %esp
        pushl   16(%ebp)                ; empile j (passage de paramètres)
        movl    16(%ebp), %eax          ; %eax= j
        movl    12(%ebp), %edx          ; %edx= i
        addl    %eax, %edx              ; %edx = i + j

        movl    $6, %eax                ; %eax= 6

        subl    %edx, %eax              ; %eax= 6 − (i + j)
        pushl   %eax                    ; %eax= 6 − (i + j)
        movl    8(%ebp), %eax           ; dépile n dans %eax
        decl    %eax                    ; %eax=%eax−1
        pushl   %eax                    ; empile n − 1 (passage de paramètres)
        call    Hanoi                   ; appel recursif de la procédure Hanoi
        addl    $16, %esp               ; rends les paramètres de l'appel à Hanoi inaccessibles
                                        ; 3 × 4 mots + subl = 16 mots
.L2:    leave                           ; instruction qui rétablit les bonnes valeurs de esp et ebp
        ret
.Lfe1:  .size   Hanoi,.Lfe1-Hanoi       ; permet de déterminer la taille du code
        .align 2                        ; cuisine interne
```