

Architecture des ordinateurs et systèmes d'exploitation

Corrigé du TD 6: Des structures de contrôle du langage C vers l'assembleur

Marcel Bosc

Christophe Dehlinger
Benoît Meister

Arnaud Giersch
Nicolas Passat

Mathieu Haeefele

1. If-then-else

Traduire le programme C suivant en assembleur SPARC :

```
#include <stdio.h>

int main(void)
{
    int a, b;
    a = 5;
    scanf("%d", &b);
    if (a < b)
        printf("c'est plus petit\n");
    else
        printf("c'est plus grand\n");
    return 0;
}
```

Correction :

```
! --- donnees ---
.section ".rodata"
    .align 1
.SCANF0:
    .asciz "%d"
.PRINTF0:
    .asciz "c'est plus petit\n"
.PRINTF1:
    .asciz "c'est plus grand\n"

! --- instructions ---
.section ".text"
    .align 4
    .global main

main:
    save %sp, -96, %sp    ! réserve un cadre de pile de 96 octets
    set 5, %l0           ! l0 = 5
    set .SCANF0, %o0     ! premier argument pour scanf = SCANF0
    add %fp, -4, %o1     ! deuxieme argument pour scanf = [fp-4]
    call scanf           ! appel de scanf, qui stocke
    nop                 ! donc la variable lue dans [fp-4]

                                ! recupere le resultat de scanf
                                cmp %l0, %l1                ! compare l0 avec l1
                                bge .PLUS_GRAND           ! si l0 > l1 va au label PLUS_GRAND
                                nop

                                set .PRINTF0, %o0        ! appel du printf
                                call printf              !
                                nop

                                ba .FIN                  ! evite la partie .PLUS_GRAND
                                nop

.PLUS_GRAND:
    set .PRINTF1, %o0    ! appel du printf
    call printf          !
    nop                 ! un nop apres tout branch ou call

.FIN:
    clr %l0
    ret
    restore
```

2. Boucle while

Traduire le programme C suivant en assembleur SPARC :

```
#include <stdio.h>

int main(void)
{
    int a, b;
    a = 0;
    scanf("%d", &b);
}
```

```

while(a <= b) {
    printf("%d\n", a);
    a++;
}
return 0;
}

```

Correction :

<pre> ! --- donnees --- .section ".rodata" .align 1 .SCANFO: .asciz "%d" .PRINTFO: .asciz "%d\n" ! --- instructions --- .section ".text" .align 4 .global main main: save \$sp, -96, \$sp ! réserve un cadre de pile de 96 octets mov \$g0, \$l0 ! l0 = 0 set .SCANFO, \$o0 ! appel de scanf add \$fp, -4, \$o1 ! </pre>	<pre> call scanf ! nop ld [%fp-4], \$l1 ! récupère le résultat du scanf .WHILE: cmp \$l0, \$l1 ! compare les valeurs bg .FIN ! si l0 > l1, ne rentre pas dans la boucle nop set .PRINTFO, \$o0 ! affiche la valeur de l0 mov \$l0, \$o1 call printf nop add \$l0, 1, \$l0 ! incrémente l0 ba .WHILE ! retourne au début de boucle nop .FIN: clr \$i0 ret restore </pre>
--	--

3. Fonction récursive

(i) Écrire une fonction récursive en C qui à partir de deux entiers positifs x et y calcule x^y .

Correction :

```

#include <stdio.h>

int puissance(int n, int puiss)
{
    if (puiss > 0)
        return n * (puissance(n, puiss - 1));
    else
        return 1;
}

int main (void)
{
    int x, y;
    scanf("%d", &x);
    scanf("%d", &y);
    printf("x^y = %d\n", puissance(x, y));
    return 0;
}

```

(ii) Traduire cette fonction en assembleur SPARC.

Correction :

<pre> ! --- instructions --- .section ".text" .align 4 ! 1 instruction = 4 octets .global puissance ! puissance : symbole global puissance: save \$sp, -96, \$sp cmp \$l1, \$g0 ble .PAS_DE_RECUR nop mov \$l0, \$o0 ! place l'argument n° 0 sub \$l1, 1, \$o1 ! place l'argument n° 1 call puissance ! appel de puissance(o0, o1, ...) nop smul \$o0, \$l0, \$o1 ! i0 = l0 * o0 ba .FIN nop </pre>	<pre> .PAS_DE_RECUR: set 1, \$i0 .FIN: ret restore ! --- donnees --- .section ".rodata" .align 1 .SCANFO: .asciz "%d" .PRINTFO: .asciz "x^y = %d\n" ! --- instructions --- .section ".text" </pre>
--	--

```

.align 4
.global main
main:
save %sp, -104, %sp      ! réserve un cadre de pile de
                        ! 96+8 = 104 octets

set .SCANFO, %0         ! 1er scanf
sub %fp, 8, %01         ! place la valeur lue
call scanf              ! a l'adresse donnée
nop                    ! par fp-8

set .SCANFO, %0         ! 2eme scanf
sub %fp, 4, %01         ! place la valeur lue
call scanf              ! a l'adresse donnée
nop                    ! par fp-4

```

```

ld [%fp-8], %00        ! charge le 1er argument
ld [%fp-4], %01        ! le 2eme
call puissance         ! appelle la fonction
nop

mov %00, %01           ! met le resultat en arg 1
set .PRINTF0, %00      ! la chaine en arg 0
call printf            ! et appelle printf
nop

clr %i0
ret
restore

```