



Systèmes Embarqués 1 & 2

Classes T-2/I-2 // 2017-2018

a.15 – C - Interfaçage assembleur - C

Exercices

Exercice 1

Compilez le code ci-dessous à l'aide du compilateur gcc de GNU et désassemblez le code objet.

```
$ arm-none-eabi-gcc -g -Wall -Wextra -O0 -std=c11 -c -o as2c.o as2c.c
```

```
$ arm-none-eabi-objdump -dS as2c.o
```

Analysez le code obtenu pour les optimisations -O0, -O1 et -Og

Fichier: as2c.c

```
extern int fnct2(int* b);

int fnct(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8, int a9, int a10)
{
    int v[]={ [0]=a1, [1]=a2, [2]=a3, [3]=a4, [4]=a5, [5]=a6, [6]=a7, [7]=a8, [8]=a9, [9]=a10, };

    return a1+a2+a3+a4+a5+a6+a7+a8+a9+a10 + fnct2(v);
}

int main ()
{
    return fnct (1,2,3,4,5,6,7,8,9,10);
}
```

Exercice 2

Pour les deux structures struct S1 et struct S2 ci-dessous, indiquez la convention utilisée pour

- (a) le retour de ces structures par les fonctions f1 et f2
- (b) le passage par valeur de ces structures par les fonctions f3 à f7

```
struct S1 {int a;};
struct S2 {int a; int b[100];};

struct S1 f1();
struct S2 f2();

void f3 (int a, int b, int c, int d, struct S2 s);
void f4 (int a, int b, int c, int d, const struct S2* s);
void f5 (struct S2 s, int a, int b, int c, int d);
void f6 (const struct S2* s, int a, int b, int c, int d);
void f7 (struct S1 s);
```

**Exercice 3**

Soit le code assembleur suivant:

```
.data
value1 : .word 10
value2 : .word 100

.bss
result : space 4

.text
add :   ldr r0,=value1
        ldr r0,[r0]
        ldr r1,=value2
        ldr r1,[r1]
        add r0, r0, r1
        ldr r1,=result
        str r0,[r1]
```

- Modifiez le code assembleur ci-dessus afin de le transformer en sous-routine réentrante. Les paramètres (*value1* et *value2*) ainsi que la valeur de retour (*result*) seront stockés dans la pile. Aucune référence à des valeurs globales ne sera tolérée. Tous les registres utilisés dans le traitement de la sous-routine doivent être rendus dans leur état initial.
- Dessinez l'état de la pile.
- Implémentez le programme principal permettant d'appeler la routine « *add* ».
- Refaites l'exercice en utilisant également le frame pointer (FP / R11).
- Transformez votre programme selon les conventions C et indiquez le prototype de la fonction « *add* ».

Exercice 4

Développez une bibliothèque en assembleur permettant d'appeler des fonctions C selon les spécifications ci-dessous

Spécifications:

- Signature des fonctions à appeler

```
int function (void* param, int arg1, int arg2, void* arg3);
```

- La bibliothèque doit permettre d'attacher et de détacher 100 fonctions différentes. Chaque méthode est identifiée par une clef unique, p.ex. un nombre entre 0 et 99.
- Lors de l'attachement de la fonction, il est possible de spécifier une valeur/un pointeur *param*, lequel sera passé à la fonction lors de son appel
- Le choix de la fonction à appeler se fait à l'aide de la clef d'attachement, les arguments *arg1* à *arg3* sont spécifiés lors de l'appel

Traitement des exceptions:

- Pas de crash si le nombre de fonctions attachées dépasse la valeur maximale
- Pas de crash si aucune fonction n'a été attachée pour une clef donnée lors de l'appel
- Si une fonction a déjà été attachée pour une clef donnée, une erreur doit être signalée