MI41 final

Durée 1h30. Les fonctions écrites en assembleur ARM doivent respecter la convention d'appel AAPCS, les accès mémoire doivent être minimisés (sauvegardes/restaurations des registres).

1. Exercice ASM

1. Une case mémoire 32 bits (registre d'interface mappé en mémoire) contient une donnée 8 bits B, fragmentée tel que B1 B0 sont respectivement sur les bits 1 et 0 de la case mémoire 32 bits, B3 B2 sur les bits 10 et 9, B7 B6 B5 B4 sur les bits 19 18 17 16 : contenu case mémoire

`x x x x x x x x x x x x B7 B6 B5 B4 x x x x x B3 B2 x x x x x x x B1 B0`.

Donnez le code d'une fonction prenant en paramètre l'adresse de la case mémoire et renvoyant l'octet B :

char receivebyte (unsigned int * A) ;

2. Convention d'appel : donnez le code assembleur correspondant au code C suivant :

```
extern int fonction_externe (int a) ;
int fonction (int a, int b)
{
        int i ;
        for (i=0 ; i < b ; i++)
                a = fonction_externe (a) ;
        return a ;
}
```

2. Interface

Une interface de périphérique permet d'envoyer des données 32 bits via une ligne de transmission. D'un point de vue utilisateur, lorsque l'interface est configurée (on considérera l'interface configurée dans cette exercice), l'envoi des données consiste à placer les données les unes après les autres dans un registre mappé en mémoire. L'interface dispose d'un tampon mémoire interne pour stocker temporairement les données avant et pendant leur transmission. Sachant que la transmission des données sur la ligne étant bien plus lente que le système et que le tampon de l'interface est limité, un bit d'un registre d'état de l'interface permet de savoir s'il reste au moins une place dans le tampon de l'interface pour pouvoir y placer une nouvelle donnée. Ce registre d'état est également un registre mappé en mémoire.
Il s'agit ici d'écrire une fonction en C puis en assembleur ARM permettant de transmettre N données via l'interface. Le prototype de la fonction en C est :

*unsigned int transmit (unsigned int * data, unsigned int len) ;*

avec :

- data : adresse où se trouvent les données à transmettre.
- len : nombre de donnée 32 bits à transmettre
-

l'adresse du registre de donnée de l'interface est 0xFF004000, c'est un registre 32 bits appelé **data_reg**
l'adresse du registre d'état de l'interface est 0xFF004004, c'est un registre 32 bits appelé **state_reg**

Les bits du registres d'état ont différents rôle. Ici, seul nous intéresse le bit N° 3 qui lorsqu'il est à 1 indique que le tampon de l'interface est plein, lorsqu'il est à 0, l'interface peut accepter une nouvelle donnée, une nouvelle donnée peut alors être placée dans le registre de donnée de l'interface.

Le principe de la fonction est alors :

- tant que le bit N° 3 du registre d'état de l'interface est à 1, attendre qu'il passe à 0
- lorsqu'il est est à 0, placer une nouvelle donnée dans le registre de données de l'interface
- lorsque toutes les données ont été transmises, la fonction revient en retournant le nombre de données envoyées.

1. Donnez le code C de la fonction, en définissant bien les adresses des registres de l'interface

2. Donnez le code ASM correspondant

3. Que pourrait-on faire pour que lorsque le bit du registre d'état reste trop longtemps à 1, la fonction ne reste pas bloquée et retourne tout de même ('time out').