UPMC
SORBONNE UNIVERSITÉS

Écriture de programmes de test. semaine 3

Objectif(s)

★ Se constituer un ensemble de petits programmes de test.

Exercice(s)

Exercice 1 – Codage et décodage d'indtructions assembleur ARM

Question 1

Donnez le codage mémoire des instructions suivantes (vous pouvez tricher) :

```
L1:     MOV r6, #63176704
        ADC r10, r11, r12, ASR r13
        SUBGES r4, r5, r6, RRX
        BEQ L1
        LDR r7, [PC, - r0, LSR #2]
```

Question 2

À quelles instructions correspondent les codes mémoires suivants :

```
e1a04001
e3510001
e92d4070
```

Exercice 2 – Programmes de test en assembleur

Dans un premier temps vous allez écrire des programmes n'utilisant pas de variable ni de fonction.

Pour tester vos programmes vous allez utiliser le le petit simulateur d'instructions ARM développé pour cette UE. L'exécutable se trouve dans le répertoire `/users/enseig/jeanlou/VLSI2017oct/bin`. Par défaut ce simulateur fonctionne en mode interactif (instruction par instruction) mais en ajoutant l'option `-run` à la ligne de commande vous l'exécution se fait automatiquement. L'exécution vous permet de visualiser l'instruction qui va être exécutée, la valeur des registres et des *flags*, elle se poursuit jusqu'à ce qu'au chargement de l'instruction se trouvant au label `_good` ou `_bad`.

Pensez à ajouter le répertoire à votre **PATH**, modifiez votre `.bashrc`.

En partant d'un fichier source en *assembler* vous allez devoir utiliser `as` pour obtenir un fichier `elf` qui correspond à l'image mémoire de votre programme. Par défaut le code mémoire généré est localisé à l'adresse 0. Si vous devez lier ce morceau de programme à d'autres vous allez devoir utiliser `ld` pour déplacer ces segments de code et mettre à jours les liens.

Prenez garde à bien utiliser les versions pour ARM de ces outils et non celles pour Intel installées par défaut sur les machines !

Voici le programme minimal exécutable par le simulateur :

```
/*-------------------------------------------------------------
//           Mon premier programme                            //
-------------------------------------------------------------*/
        .text
        .globl  _start
_start:
        /* 0x00 Reset Interrupt vector address */
        b       _good

        /* 0x04 Undefined Instruction Interrupt vector address */
        b       _bad

_bad :
        add r0, r0, r0
_good :
        add r1, r1, r1
```

Au reset le processeur exécute l'instruction se trouvant à l'adresse 0, le fichier `sections.lds` indique le positionnement des différentes sections. Un fichier `Makefile` vous est fournit comme source d'inspiration.

Question 1

Écrire la séquence d'instruction permettant de calculer le pgcd de deux entiers contenus dans 2 registres (`r0` et `r1`). Simulez l'exécution de ce programme avec `arm_sim`.

Question 2

Minimisez le nombre d'instruction et de branchements en utilisant l'exécution conditionnelle des instructions.

Question 3

Pour toutes les instructions (traitement de donnée, mémoire simple et multiples et branchements) écrire un programme assembleur permettant de tester son fonctionnement. Vous prendrez soin utiliser les différentes variantes disponibles d'opérandes pour chaque instruction.

Question 4

Les appels de fonctions pour les architectures ARM suivent les conventions suivantes :

- Les registres `r0` à `r3` peuvent être utilisés pour transmettre les 4 premiers arguments de fonction ;
- Le registre `r0` est utilisé pour véhiculer la valeur de retour ;
- Le contenu des registres `r4` à `r11` doivent être préservés par les fonctions (donc enregistrées sur la pile si utilisés) ;
- La pile se remplit dans le sens des adresses décroissantes.

Écrivez et testez quelques fonctions :

- pgcd,
- multiplication à la russe,
- factoriel récursif (pour utiliser la pile).

Exercice 3 – Programmes de test en C

Question 1

Écrivez quelques programmes de test en C, compilez-les avec `gcc` pour ARM avec les options `-march=armv2a -mno-thumb-interwork` : `strlen`, tri à bulle, tri rapide ...