Université Joseph Fourier – Grenoble 1

Architecture des ordinateurs

Exercices sur le langage machine

Avril 2012

I. Commentaires …

On considère le programme suivant écrit en langage C :

```
1       char tab [11]= "aa(*aaa*)a";
2       int i;
3
4       main () {
5         printf (" chaine initiale : %s\n", tab);
6         for (i=0; i<10; i++) {
7           if ( (tab[i]=='(') && (tab[i+1]=='*') ) {
8             tab[i] = '/';
9             }
10          }
11        printf (" chaine modifiee : %s\n", tab);
12        }
```

Il s'agit d'une version partielle et très simplifiée d'un programme qui remplacerait les débuts et fins de commentaires de la forme "(* ....*)" par la forme "/*....*/". En fait, on ne traite ici que le début des commentaires.
On vous donne une zone data contenant la déclaration de la chaîne à modifier dans le tableau tab.

```
        .data
tab:    .ascii  "aa(*aaa*)a"
        .byte   0x00            @ le caractere '\0'
```

Q1. Ecrire en langage d'assemblage ARM la partie de programme nécessaire à la traduction des lignes 7 à 9 (ou 6 à 10 si vous avez le temps, ou même tout si le printf ne vous fait pas peur) du programme donné. N'oubliez pas de commenter votre programme. En particulier, précisez pour chaque variable ou information manipulée dans quel registre elle est implantée.

Q2. Le « && » logique peut avoir deux traductions (par calcul booléen, et par composition de branchements), reprendre la traduction précédente pour en produire une seconde. Généraliser et abstraire pour obtenir deux schémas de traduction généraux des formules booléennes.

II. Matrice carrée

On étudie la réalisation de l'algorithme suivant qui incrémente chaque élément d'une matrice d'entiers N × P (N lignes, P colonnes). Une telle matrice peut être implantée par un tableau de N tableaux (les lignes) de taille P.

```
mat: un tableau sur [0..N-1] de tableaux sur [0..P-1] d'entiers naturels représentés sur 4 octets.

pour i=0 a N-1
        pour j=0 a P-1
                mat[i,j] ← mat[i,j] + 1
```

La matrice est représentée dans la zone data dans une zone de mémoire de taille N * P * 4.

```
        .set N, 10      @ nombre de lignes, la valeur 10 est arbitraire
        .set P, 7       @ nombre de colonnes, la valeur 7 est arbitraire
        .data
```

```
admat: .skip N*P*4    @ zone de mémoire pour la matrice
```

L'élément mat[i,j] est situé à l'adresse admat + (i*P + j) * 4.

Q1. Récrire l'algorithme proposé en termes de structures de contrôle "tant que"

Q2. Proposez une traduction en ARM de cet algorithme. Vous pourrez définir et utiliser une procédure annexe qui donne l'adresse d'un élément mat[i,j] (on se donnera alors, une procédure MUL qui calcule dans le registre r2 le produit des entiers contenus dans les registres r0 et r1).

III. Multiplication récursive

On considère la fonction suivante :

```
1:      unsigned int mult(unsigned int a, b) {
2:        unsigned int n;
3:        n = b;
4:        if (n == 0)
5:                return 0 ;
6:        else {
7:                n = n-1;
8:                return ( a + mult(a, n) );
9:        }
10:     }
```

Q1. Les paramètres et le résultat de la fonction mult sont passés par la pile. La variable locale n est stockée dans la pile. Dessinez la structure de la pile lors d'une exécution de la fonction mult. Donnez un code ARM pour la fonction mult en vous aidant du patron suivant :

```
mult:
        @ sauvegarde de l'adresse de retour dans la pile
        @ sauvegarde du frame pointer dans la pile
        @ mise en place du nouveau frame pointer
        @ reservation de place pour la variable locale
        @ empiler les variables temporaires
        @ corps de la fonction
        @ mise en place de la valeur de retour
        @ depiler les variables temporaires
        @ liberer place allouee a la variable locale
        @ restaurer l'ancien frame pointer
        @ recuperer l'adresse de retour
        @ retour
```

On suppose maintenant l'existence d'une procédure dec et d'une fonction add :

- dec décrémente de 1 la valeur de la variable dont l'adresse lui est donnée en paramètre.
- plus retourne le résultat de l'addition de ses deux paramètres donnés.

Q2. Le paramètre de la procédure dec, les paramètres et le résultat de la fonction plus sont passés par la pile. Donnez les parties de code ARM à modifier pour remplacer la ligne 7 de l'algorithme donné pour la fonction mult par dec (adresse de n); (* en langage C : dec(&n) *) et la ligne 8 par return (plus(a, mult(a,n)))