N7-1SN

Architecture des ordinateurs – S6 - TD2

Assembleur « Craps » : accès mémoire, pile, sous-programmes

1- Structure d'un programme assembleur

L'exemple suivant montre le code assembleur « craps » correspondant à l'instruction :

Resultat := Variable1 + Variable2

Les variables Resultat, Variable1 et Variable2, de type entier, sont réservées en mémoire à des adresses que l'on va désigner par Resultat, Variable 1 et Variable2. Contrairement aux langages évolués, les noms des variables que nous utilisons désignent leur adresse ; et les valeurs de ces variables sont désignées par [variable]. Les […] indiquent un accès mémoire.

```
Debut:      set     Variable1, %r2      // r2 <- adresse de Variable1
            set     Variable2, %r3      // r3 <- adresse de Variable2
            set     Resultat, %r4       // r4 <- adresse de Resultat
            ld      [%r2], %r5          // ld = load : r5 <- valeur mémoire de Variable1
            ld      [%r3], %r6          // r6 <- valeur mémoire de Variable2
            add     %r5, %r6, %r7       // %r7 <- %r5 + %r6
            st      %r7, [%r4]          // st = store : [Résultat] <- r7
Stop:       ba      Stop                // branchement sur place comme fin de programme
Variable1:  .word   123                 // réservation d'un mot mémoire initialisé à 123
Variable2:  .word   654
Resultat:   .word   0
```

Les Instructions de branchement : b'condition' adresse

- condition : a (always), eq (égal), ne, gt (greater than signé), lt (less than signé), lu (less unsigned), leu (less or equal unsigned), gu, geu, …
- la condition fait référence au résultat d'une opération précédente, dont l'état est stocké dans les indicateurs N, Z, V et C (seules les instructions se terminant par CC modifient les indicateurs : addcc, subcc, andcc, …).
- L'instruction cmp opérande1, opérande2 (traduite en subcc opérande1, opérande2, %r0) positionne les indicateurs et permet d'utiliser des conditions de branchement portant sur la position du résultat par rapport à 0 (= position de opérande1 par rapport à opérande2).

Les instructions de branchement nous permettront d'implanter les différentes structures de contrôle utilisées dans les langes évolués.

2- Si condition Alors Sinon

Algorithme	Solution 1	Solution 2

Si A > B Alors
  A <- A – B
Sinon
  B <- B – A
FinSi
	// A dans r1, B dans r2
	  cmp %r1, %r2
	  bgu AsupB
	AInfEqB : sub %r2, %r1, %r2
	  ba FinSi
	AsupB : sub %r1, %r2, %r1
	FinSi : …
		// A dans r1, B dans r2
		  cmp %r1, %r2
		  bleu AInfEqB
		AsupB : sub %r1, %r2, %r1
		  ba FinSi
		AInfEqB : sub %r2, %r1, %r2
		FinSi : …

3- Tant que condition Faire

```
Tantque :     Test de la condition d'entrée
              B'condition fausse'  Fintanque
              … actions            // condition vraie : on exécute les instructions de la boucle
              Ba Tantque           // et on recommence
Fintanque :   ….
```

Tantque A /= B Faire … Fintantque	A supposé dans r1 B supposé dans r2	Tantque : cmp %r1, %r2 beq Fintanque … ba Tantque Fintanque : …

Exercice : Ecrire le programme qui calcule le pgcd de deux nombres A et B, stockés en mémoire.

```
Pgcd:        …
Stop :       ba Stop
Nombre_A :   .word 90
Nombre_B :   .word 175
```

Exercice : Ecrire le programme qui calcule la factorielle d'un nombre stocké en mémoire.

4- Répéter

```
Repeter :    … actions
             Test de la condition de sortie
             B'condition fausse'  Repeter
             …
```

Par exemple, la séquence suivante permet de réaliser une boucle « répéter » de 10 passages

```
             set     10, %r1        // index : nombre de passages restant
Repeter:     … actions
             subcc   %r1, 1, %r1    // positionne les indicateurs N, Z, V, C
             bne     Repeter        // branchement à Repeter si %r1 /= 0
             …                      // instruction exécutée si %r1 = 0
```

Exercice : Ecrire une seconde version de factorielle en utilisant une boucle « Répéter »

4- Tableaux

Les éléments d'un tableau sont stockés les uns à la suite des autres en mémoire. L'adresse du tableau est celle du premier élément. Si l'indice du premier élément = 0, l'adresse de l'élément d'indice I = adresse_tableau + I*Taille_element. Sur craps, on travaillera avec Taille_element = 1

Exercice : Ecrire le programme qui calcule la somme des éléments d'un tableau

```
N = 10                  // nombre d'éléments
Somme_tab :  ….
Stop :       ba      Stop
Somme :      .word 0
Tab :        .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

5- Sous-programmes

L'appel d'un sous-programme nécessite les différentes opérations suivantes :

- préparation des paramètres, on parlera de passage de paramètres
- mémorisation de l'adresse de retour
- branchement à la première instruction du sous-programme,
- exécution du sous-programme, puis retour au programme appelant

Le passage des paramètres et la mémorisation de l'adresse de retour peut s'effectuer de deux manières :

- dans des registres,
- en utilisant une pile : une pile est une structure de données, sur laquelle on peut effectuer deux opérations :
    - empiler : ajout d'une nouvelle valeur en sommet de pile : **push %ri**
    - dépiler : récupération de la valeur au sommet de pile : **pop %ri**

La pile de craps est gérée par un pointeur, qui est un registre dédié appelé %sp (%r29), et qui pointe sur l'élément au sommet de la pile. Chaque programme doit initialiser sa propre pile avec une adresse assez éloignée du bloc instructions pour que la pile n'écrase pas les instructions.

```
PILE = 0x200
Prog :      set PILE, %sp
            …
```

La pile est aussi utilisée pour :

- sauvegarder les registres qui sont modifiés par le sous-programme afin que ces modifications n'affectent le contexte du programme appelant
- réserver des variables locales au sous-programme : c'est ce qui est fait par les compilateurs

Pour des raisons d'efficacité (l'accès aux registres est plus rapide que l'accès mémoire), et de facilité de programmation, nous utiliserons en priorité des registres pour le passage des paramètres et les variables locales (plus de 20 registres à la disposition du programmeur).

Dans craps, l'appel à un sous-programme s'effectue par l'instruction « ***call »***, qui sauvegarde l'adresse de l'instruction courante dans le registre **%r28**. Le retour du sous-programme s'effectue par l'instruction « ***ret*** » qui récupère l'adresse de l'instruction d'appel dans le registre %r28, et lui ajoute 1 pour passer l'instruction qui suit.

Nos sous-programme en assembleur craps prendront donc la forme :

```
// rôle : …
// IN paramètre1 : …… dans r1
// IN paramètre2 : …… dans r2
// …
// OUT résultat : ..... dans r3
Sous_prog :   push %rX
              push %rY      // rX et rY sont modifiés dans le sous-programme
              …             // R3 non sauvegardé car sert pour retourner le résultat
              pop %rY       // attention : dépilement dans l'ordre inverse de l'empilement
              pop %rX
              ret           // retour au programme appelant
```

Soit le code suivant :

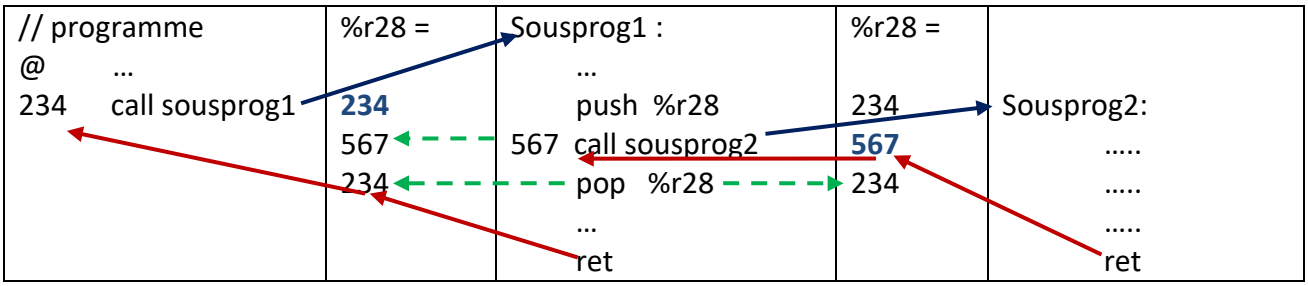
```
PILE = 0x200
N = 10
        set  PILE, %sp
        set  Tab, %r1
        call Ordonner
Stop:   ba Stop
Tab:    .word  11, 10, 9, 8, 7, 6, 4, 5, 3, 2

// Rôle : … à compléter…
// IN : r1, adresse d'un élément de tableau
Ordonner:
        ld  [%r1], %r2
        ld  [%r1+1], %r3
        cmp  %r2, %r3
        ble  o_suite
        st  %r3, [%r1]
        st  %r2, [%r1+1]
o_suite:
        ret
```

1- Que fait le sous-programme Ordonner ? Compléter sa spécification : rôle
2- Quel est le contenu des registres r1, r2, et r3 après le retour du sous-programme ?
3- Comment corriger cela ?

Appels en cascade

Lorsqu'un sous-programme a besoin d'appeler un autre sous-programme, le registre **%r28** est déjà occupé, car il contient l'adresse de retour du premier appel. Pour ne pas perdre cette adresse, il faut la sauvegarder dans la pile avant le nouvel appel et la récupérer au retour de cet appel.



Ecrire le sous-programme « bulle_max » qui, par appels successifs à « Ordonner », fait remonter le max d'un tableau à la place du dernier élément.
Ecrire le programme qui effectue un tri croissant d'un tableau de N entiers signés.