TD5 : Instructions de sauts et structures de contrôle

Objectif(s)

- ✯ Familiarisation avec les instructions de rupture de séquence : les branchements (saut conditionnels ou inconditionnels).
- ✯ Écriture de programmes contenant des conditionnelles et/ou des boucles.
- ✯ Parcours de tableau.

Remarque : l'exercice sur la manipulation de bits (intitulé "Manipulation de bits : décalage et masque") permet de manipuler les instructions de décalage et d'utiliser les masques avec lesquels vous n'êtes pas toujours très à l'aise. Il est fortement conseillé de le faire chez vous ou en TME s'il n'est pas traité en TD. Aucune correction n'est donnée.

Exercice(s)

Exercice 1 – Instructions de saut conditionnel et inconditionnel

A l'aide de votre mémento, dites ce que font les instructions suivantes et quel est leur format de codage :

```
beq  $8, $9, etiquette
bne  $8, $9, etiquette
blez $8, etiquette
bltz $8, etiquette
bgez $8, etiquette
bgtz $8, etiquette
j etiquette
jr r1
```

Quelles sont les instructions de saut conditionnel et de saut inconditionnel parmi les instructions ci-dessus ? Quelles sont les conditions que l'on peut exprimer dans les instructions de saut conditionnel ?

Solution:

Ce sont toutes des instructions de saut. Celles dont le mnémonique commence par `b` sont codées au format I et exécutent des sauts conditionnels. L'immédiat code le déplacement en mots (toutes les instructions font 4 octets) calculé par rapport à la valeur de `PC` (plus précisémenet par rapport à l'adresse de l'instruction suivant l'instruction de saut).

Celles dont le mnémonique commence par `j` sont des instructions de saut inconditionnel, le saut est donc toujours effectué. L'instruction `j` est codée au format J. L'immédiat sur 26 bits code l'adresse absolue de la cible (l'adresse est complétée avec 4 bits de poids fort qui sont ceux de `PC` et 2 bits de poids faible qui valent nécessairement 00). L'instruction `jr` effectue un saut à l'adresse correspondant à la valeur contenue dans le registre en paramètre.

Les conditions que l'on peut exprimer dans une instruction de saut conditionnel sont :

— les comparaisons à zéro : $>$ (`bgtz`), $\geq$ (`bgez`), $<$ (`bltz`), $\leq$ (`blez`)

— l'égalité (`beq`) ou la différence (`bne`) de valeur entre deux registres.

Donc, comme on le voit un peu plus loin, si l'on veut tester l'ordre de deux valeurs, on est obligé d'utiliser une instruction de comparaison (`slt`, `slti`, ...).

Dans les programmes écrits dans des langages de haut niveau, on utilise des structures de contrôle qui cassent l'exécution séquentielle des instructions d'un programme. Les deux types de structures de contrôle les plus employés sont l'alternative (if cond then ... [else ...]) et les boucles (boucles for ou boucles while).

Exercice 2 – L'alternative

Question 1

Donnez le code assembleur correspondant au programme C ci-dessous. Simulez l'exécution du programme et vérifiez que la valeur de `a` en mémoire à l'issue du programme est correcte. Faites de même avec une valeur initiale de `a` permettant de tester le deuxième chemin d'exécution possible.

```
int a = -5;
int b = 3;
void main() {
  if (a == 0) {
     a = a + b;
  }
  else {
     a = a - b;
  }
  exit();
}
```

Solution:

```
.data
a:      .word -5
b:      .word 3
.text
   lui   $9, 0x1001
   lw    $8, 0($9)         # lecture de  a
   bne   $8, $0, casfaux   # a != 0


   lw    $10, 4($9)       # lecture de b
   addu  $8, $8, $10      # a + b
   sw    $8, 0(9)         # a = a + b
   j     suite

casfaux:
   lw    $10, 4($9)       # lecture de b
   subu  $8, $8, $10      # a - b
   sw    $8, 0(9)         # a = a - b

suite:

   ori   $2, $0, 10
   syscall
```

Question 2

Donnez le calcul de la condition et le saut conditionnel dans les cas où la condition `a==0` est remplacée dans le code source par :

1. `a != 0`

2. `a > 0`
3. `a >= 2`
4. `a < b`

Solution:
Il faut remplacer `bne $8, $0, casfaux` par :

1. Si la condition est `a !=0`, il faut remplacer `bne $8, $0, casfaux` par `beq $8, $0, casfaux`.
2. Si la condition est `a > 0`, il faut remplacer `bne $8, $0, casfaux` par `blez $8, casfaux`.
3. Dans le cas d'une comparaison entre deux valeurs, il faut d'abord calculer la valeur de la condition en utilisant les instructions `slt`, `sltiu` ou `slti`. Pour la condition `a >= 2`, cela donne :

```
slti $11, $8, 2     # Le registre $11 vaut 1 si a < 2
bne $0, $11, casfaux
```

4. Pour la condition `a < b`, cela donne :

```
slt $11, $8, $10    # le registre $11 vaut 1 si a < b, valeur de b dans $10
beq $0, $11, casfaux
```

Exercice 3 – Boucles for

Question 1

Donnez un programme assembleur qui contient deux variables globales `p` et `q` initialisées (à 1 et 10 par exemple). Le programme principal calcule la somme des entiers compris entre `p` et `q` inclus et affiche cette somme.

Attention : si `q` est strictement inférieur à `p`, la somme doit valoir `0`; si les deux entiers sont égaux, alors somme doit valoir `p`.

Solution:
Le test de sortie de boucle doit se faire au début de la boucle et pas à la fin. S'il est effectué à la fin de la boucle cela suppose qu'il y a toujours une itération au moins alors qu'on ne le sait pas a priori...

```
.data
p: .word 1
q: .word 10

.text
lui $8, 0x1001  #adresse de p
lw $9, 0($8)        # valeur de p
lw $10, 4($8)       # valeur de q
xor $11, $11, $11   # somme = 0

loop:
    slt $12, $10, $9    # $12 vaut 1 si valeur q < valeur courante de p
    bne $12, $0, finloop

    addu $11, $11, $9    # somme += valeur courante de p
    addiu $9, $9, 1      # incrementation valeur courante de p
    j loop
finloop:
    ori $4, $11, 0
    ori $2, $0, 1
    syscall

    ori $2,$0, 10
    syscall
```

Exercice 4 – Boucle while avec conditionnelle et manipulation de tableau

Question 1

Donnez un programme qui calcule le maximum d'un tableau d'entiers relatifs (codés en compléments à 2) strictement positifs et affiche ce maximum ; le mot à 0 marque la fin du tableau.

Rappel : un tableau de `N` entiers correspond à `N` entiers rangés consécutivement en mémoire. Ainsi, il vous faut allouer un ensemble d'entiers non nuls en mémoire et indiquer la fin du tableau en allouant un mot initialisé à 0 après la déclaration des entiers du tableau.

Solution:

```
.data
tab: .word 3, 123,  12, 1024, 5, 35, 200, 0

.text

        lui $8, 0x1001  # adresse tab
        xor $9, $9, $9  # max = 0

loop:
        lw $10, 0($8)           # chargement de l'element courant
        beq $10, $0, finloop    # chargement du mot 0 qui finit le tableau

        slt $11, $9, $10        # $11 = 1 si max < element courant
        beq $11, $0, suite

        ori $9, $10, 0         # max prend la valeur de l'element courant
suite:
        addiu $8, $8, 4        # adresse de l'element suivante
        j loop

finloop:
        ori $4, $9, 0
        ori $2, $0, 1
        syscall
        ori $2, $0, 10
        syscall
```

Exercice 5 – Manipulation de bits : décalage et masque (à faire en TME si non traité en TD)

Question 1

Déclarez un entier `n` en variable globale et initialisez-le à la valeur 123. On souhaite écrire un programme qui calcule puis affiche le nombre de bits à 1 dans le mot binaire représentant l'entier.

Quelle valeur doit afficher ce programme quand `n` vaut 123 ? quand `n` vaut -1 ou 0xFEDCBA98 ?

Pour écrire ce programme, il est nécessaire de tester la valeur des 32 bits du mot binaire correspondant à `n`. Il faut donc écrire l'équivalent d'une boucle `for` en assembleur qui traite le ième bit du mot à la ième itération. Si ce bit vaut 1, il faut le compter dans le résultat.

Pour récupérer la valeur du bit de poids faible d'un mot binaire, il suffit de réaliser une opération de masquage (opération booléenne ET) avec la valeur 1 :

$$a_3a_2a_1a_0 \ \& \ 0b0001 = 000a_0$$

Pour traiter tous les bits d'un mot, il suffit de décaler le mot vers la droite de manière logique ce qui permet de positionner sur le bit de poids faible du mot résultant le bit voulu :

$$a_3a_2a_1a_0 \gg_{unsigned} 1 = 0a_3a_2a_1$$

$$a_3a_2a_1a_0 \gg_{unsigned} 2 = 00a_3a_2$$

Solution:

123 se code en binaire sur 8 bits 0b01111011 donc le programme doit afficher 6. Si `n` vaut -1, alors le programme doit afficher 32. Si `n` vaut 0xFEDCBA98 alors le programme doit afficher 4+3+3+2+3+2+2+1 = 20.

Il est conseillé de chercher cet exercice et de le tester avec MARS (voir note au début), la correction assembleur n'est pas donnée.