TD ASR2 (Programmation assembleur) 1ère Année

Analyse et Optimisation de programmes en assembleur

Exercice 1 Le but est d'optimiser le code source assembleur ci dessous

```
1   _calcul:
2       stw r30,-8(r1)
3       stwu r1,-64(r1)
4       mr r30,r1
5       stw r3,88(r30)
6       stw r4,92(r30)
7       li r0,1
8       stw r0,32(r30)
9       li r0,1
10      stw r0,36(r30)
11  L2:
12      lwz r0,36(r30)
13      lwz r2,92(r30)
14      cmpw cr7,r0,r2
15      ble cr7,L5
16      b L3
17  L5:
18      lwz r2,32(r30)
19      lwz r0,88(r30)
20      mullw r0,r2,r0
21      stw r0,32(r30)
22      lwz r2,36(r30)
23      addi r0,r2,1
24      stw r0,36(r30)
25      b L2
26  L3:
27      lwz r0,32(r30)
28      mr r3,r0
29      lwz r1,0(r1)
30      lwz r30,-8(r1)
31      blr
```

1°) S'agit-il d'un programme ou d'un sous-programme (fonction) ? Pourquoi ?

2°) Combien de paramètres comporte ce (sous-) programme ? Quels sont leurs types et leurs natures (entrée ou sortie) ? Où sont-ils rangés ?

3°) Repérer précisément la boucle en notant les lignes correspondantes. Quelle variable (registre ou case mémoire) contrôle son déroulement ? Comment est-elle initialisée ? Quelle est la condition d'arrêt ? Comment évolue-t-elle ?

4°) La boucle utilise-t-elle des registres de travail ? Si oui, lesquels et pourquoi faire ?

5°) Donner l'équivalent C de ce (sous-) programme. Que calcule-t-il ?

6°) Traduire ce programme C en langage assembleur PowerPC 604 optimisé en supprimant toute instruction de chargement et de déchargement dans la pile. Chaque instruction sera clairement explicitée.

7°) Tester votre programme en suivant les étapes suivantes :

- Ecrire la fonction en C
- Ecrire le programme principal permettant de tester le précédent (gcc *.c maincal.c –o calcul)
- Compiler sur euphor la fonction en assembleur non optimisé (gcc-3.4 –S -mcpu=604 *.c)
- Remplacer les instructions assembleur du fichier *.s par votre programme assembleur et le tester (gcc-3.4 *.s maincal.c –o calcul)

- Comparer votre version avec la version optimisée du compilateur (gcc-3.4 –S –o *opt.s –O -mcpu=604 *.c)

Exercice 2 Le but est de fournir un équivalent C de la fonction assembleur suivante :

```
1       _apr:
2           li r0,1
3       L2:
4           andi. r2,r4,1
5           beq- cr0,L5
6           mullw r0,r0,r3
7       L5:
8           mullw r3,r3,r3
9           srawi r4,r4,1
10          addze. r4,r4
11          bne+ cr0,L2
12          mr r3,r0
13          blr
```

1°) Détailler pour [R3] = 2 et [R4] = 4, l'exécution pas à pas de cette fonction. Quelle est la valeur retournée en sortie ? Même question pour [R3] = x et [R4] = 9.
2°) En appelant x le paramètre passé dans R3 et n celui passé dans R4, que calcule apr(x,n) ?
3°) Donner l'équivalent C de cette fonction.
4°) Quelles remarques peut on faire sur l'efficacité de cet algorithme ?
5°) Tester votre programme en suivant les mêmes étapes que précédemment :

- Ecrire la fonction en C
- Ecrire le programme principal permettant de tester le précédent (gcc *.c main.c –o puissance)
- Compiler sur euphor la fonction en assembleur optimisé (gcc-3.4 –S –O -mcpu=604 *.c)
- Remplacer les instructions assembleur du fichier *.s par votre programme assembleur et le tester (gcc-3.4 *.s main.c –o puissance)

N.B. 1 : Les divisions entières successives d'un nombre pair par deux donnent toujours au moins un nombre impair parmi les quotients obtenus.
N.B. 2 : srawi RA,RS,SI : Décalage à droite d'un mot RS de SI bits dans le registre RA.