M1 SESI – Architecture des Processeurs et Optimisation

TD3 – Pipeline, Superpipeline, Optimisation de codes

Exercice 1 :

La fonction `GetListLength` calcule le nombre d'éléments d'une liste. Un élément d'une liste est une structure de donnée appelée `chain`. Une structure `chain` est composée de deux pointeurs (un pointeur est un mot de 4 octets qui contient l'adresse d'un autre objet). Le premier pointeur donne l'adresse de la structure `chain` suivante. Le deuxième pointeur donne l'adresse de la donnée.

```
struct chain {
    struct chain * NEXT;
    void * DATA;
};

int GetListLength (struct chain * pt) {
    int i = 0;
    while (pt != NULL) {
        i++;
        pt = pt->NEXT;
    }
    return i;
}
```

Ecrire en assembleur Mips-32 la boucle principale de `GetListLength`. On suppose que `R4` contient l'adresse de la liste (pt).

Analyser l'exécution de la boucle à l'aide d'un schéma simplifié.

Exercice 2 :

On considère un programme de traitement d'images. Une image est composée d'un ensemble de pixels organisés sous forme d'un tableau. Pour une image en 'niveaux de gris', chaque pixel est codé sur un octet. La valeur 0 représente la couleur noire, la valeur 255 la couleur blanche. Les valeurs comprises entre 0 et 255 représentent les différentes nuances de gris. Une fonction de traitement d'images consiste à appliquer un certain algorithme à un tableau de pixels. On considère le programme suivant.

```
loop:
    Lbu   r8 , 0(r4)        ; lire un pixel
    Addu  r8 , r8 , r5
    Lbu   r9 , 0(r8)        ; lire f(pixel)
    Sb    r9 , 0(r4)

    Addiu r4 , r4 , 1
    Bne   r4 , r10, loop
```

À l'entrée de la boucle, le registre `R4` contient l'adresse du tableau de pixels (l'image à traiter). Le registre `R10` contient l'adresse de la fin de l'image. Le registre `R5` contient l'adresse d'un tableau F de 256 octets. Ce tableau représente une fonction `f(x)` avec `0 <= x <= 255` et `0 <= f(x) <= 255`. Ce programme permet d'appliquer la fonction `f(x)` à chaque pixel de l'image. Si `x` est la valeur d'un pixel, celle-ci est remplacée par `f(x)`. Par exemple, si `f(x) = 255 - x`, l'application de ce programme à une image permet d'obtenir l'image négative.

a - Modifier le programme pour qu'il soit exécutable sur le Mips-32.

b - Analyser, à l'aide d'un schéma simplifié, l'exécution de ce programme dans le Mips. Calculer le nombre de cycles pour effectuer une itération.

c - Optimiser le code en changeant l'ordre des instructions de manière à obtenir un CPI et un CPI-utile de 1

d - Optimiser le code en utilisant la technique du "pipeline logiciel". Calculer le nombre de cycles par itération.

Exercice 3 :

On considère un processeur RISC pipeline, appelé **PROC**. **PROC** a le même jeu d'instructions que le Mips-32. PROC a un pipeline à 7 étages : `IF1`, `IF2`, `DEC`, `EXE`, `MM1`, `MM2`, `WBK`.

Au début de l'étage `IF1`, l'adresse de l'instruction est envoyée à la mémoire et l'instruction correspondante est récupérée à la fin du cycle `IF2`. De la même manière, l'accès à la mémoire de données se déroule en deux cycles. Comme dans Mips-32, le calcul de l'adresse d'instruction s'effectue dans `DEC`. De même, les étages `EXE` et `WBK` ressemblent aux étages `EXE` et `WBK` du Mips.

*a - Montrer, à l'aide d'un schéma détaillé, l'exécution de l'instruction `SW` dans **PROC**.*

PROC contient tous les bypass qui arrivent dans les étages `DEC` et `EXE`. On souhaite étudier l'impact de la mise en place d'un bypass dans l'étage `MM1` sur la performance du processeur. On considère donc deux versions de **PROC**.

Dans **PROC1**, il n'y a pas de bypass entre l'étage `MM2` de l'instruction `i` et l'étage `MM1` de l'instruction `i+2`. La période de l'horloge est de 2 ns pour **PROC1**. Dans **PROC2**, il y a un bypass entre l'étage `MM2` de l'instruction `i` et l'étage `MM1` de l'instruction `i+2`. La période de l'horloge est de 2,1 ns pour **PROC2**.

*b - Expliquer pourquoi la période de l'horloge est plus longue dans **PROC2** ?*

Pour comparer la performance des 2 versions de **PROC**, on considère le code de l'exercice 2.

*c - Modifier le code du programme pour qu'il soit exécutable sur **PROC**.*

*d - Analyser, à l'aide d'un schéma simplifié, l'exécution de ce code dans **PROC1**. Combien de cycles sont nécessaires pour l'exécution d'une itération de la boucle ? Calculer le CPI et le CPI-utile. Combien dure (en seconde) le traitement d'une image de 1024 pixels ?*

*e - Analyser, à l'aide d'un schéma simplifié, l'exécution de ce code dans **PROC2**. Combien de cycles sont nécessaires pour l'exécution d'une itération de la boucle. Calculer le CPI et le CPI-utile. Combien dure (en seconde) le traitement d'une image de 1024 pixels ?*

*f - Pour **PROC1**, réordonner le code afin d'éviter au maximum les cycles perdus. Combien de cycles sont maintenant nécessaires pour l'exécution d'une itération de la boucle. Calculer le CPI et le CPI-utile. Combien dure (en seconde) le traitement d'une image de 1024 pixels ?*

*g - Pour **PROC2**, réordonner le code afin d'éviter au maximum les cycles perdus. Combien de cycles sont maintenant nécessaires pour l'exécution d'une itération de la boucle. Calculer le CPI et le CPI-utile. Combien dure (en seconde) le traitement d'une image de 1024 pixels ?*

*h - Pour améliorer la performance du traitement, on décide de dérouler une fois la boucle (traitement de 2 pixels par itération). Proposer un code optimisé pour **PROC1**. Combien de cycles sont maintenant nécessaires pour l'exécution d'une itération de la boucle. Calculer le CPI et le CPI-utile. Combien dure (en seconde) le traitement d'une image de 1024 pixels ?*

*i - Pour améliorer la performance du traitement, on décide de dérouler une fois la boucle (traitement de 2 pixels par itération). Proposer un code optimisé pour **PROC2**. Combien de cycles sont maintenant nécessaires pour l'exécution d'une itération de la boucle. Calculer le CPI et le CPI-utile. Combien dure (en seconde) le traitement d'une image de 1024 pixels ?*

Exercice 4 :

On considère le processeur pipeline P9. Il a le même jeu d'instructions que le Mips. Il est

construit autour d'un pipeline à 9 étages :

`IF1, IF2, DEC1, DEC2, EXE1, EXE2, MEM1, MEM2, WBK`

Le découpage en étages de ce pipeline a été obtenu à partir du pipeline du processeur Mips. Chacun des étages `IFC`, `DEC`, `EXE` et `MEM` a été divisé en deux étages. Le calcul de l'adresse de l'instruction suivante s'achève dans l'étage `DEC2`.

On souhaite étudier le fonctionnement de ce processeur. Le code suivant est la boucle principale d'une fonction qui calcule la valeur absolue des éléments d'un tableau d'entiers. A l'entrée de la boucle `R4` contient l'adresse du tableau, le registre `R9` contient l'adresse de la fin du tableau.

```
loop:
    Lw    r8 , 0(r4)        ; lire un élément
    Bgez  r8 , _endif
    Sub   r8 , r0 , r8      ; calculer l'opposé
    Sw    r8, 0(r4)

_endif:
    Addiu r4 , r4 , 4
    Bne   r4 , r9 , loop
```

a - Le découpage du pipeline en 9 étages a-t-il un impact sur le compilateur ?

b - Dans le processeur Mips, il existe des dépendances de données d'ordre 1, 2 et 3. Dans P9 quelles sont les dépendances de données ?

c - Quels sont les bypass nécessaires à l'exécution des instructions dans ce P9 ? Illustrer pour chaque bypass son utilisation à l'aide d'un exemple d'une suite d'instructions.

d - Quelles sont les situations qui provoquent des cycles de gel ? Illustrer chaque cas à l'aide d'un exemple d'une suite d'instructions.

e - Modifier le code de la boucle pour qu'il soit exécutable sur le processeur P9.

f - Analyser l'exécution d'une itération de la boucle à l'aide d'un schéma simplifié.

g - Calculer le CPI et le CPI-utile en supposant que 50% des nombres sont négatifs.

h - Réordonner ce code de manière à éviter au maximum les cycles perdus.

i - Optimiser ce code à l'aide de la technique "software pipeline".

j - Le temps de cycle du processeur P9 est égal à 0,6 fois le temps de cycle du processeur Mips. Comparer le temps nécessaire à l'exécution d'une itération de la boucle entre le Mips et P9. Quelle conclusion en tirez-vous ?

Last modified on Nov 3, 2020, 12:30:37 PM