M1 SESI – Architecture des Processeurs et Optimisation

TD4 – Superscalaire, Optimisation de codes

L'objectif de cette étude est de comparer deux réalisations différentes de l'architecture **Mips R3000**. La première est la réalisation classique sur un pipeline de 5 étages présenté en cours. Cette réalisation est appelée **Mips**. La seconde, appelée **SS2** est une réalisation superscalaire à deux pipelines. Il s'agit d'une réalisation à 5 étages : `IFC`, `DEC`, `EXE`, `MEM`, `WBK`. L'étude porte sur plusieurs programmes.

Exercice 1

*a - Quels sont les bypass nécessaires à l'exécution des instructions dans **SS2** ?*

*b - On considère l'étage `DEC` de la réalisation **SS2**. Proposer deux situations où la seconde instruction du buffer d'instructions ne peut pas être lancée (cas de Stall) ?*

On considère la boucle suivante :

```
Loop :
    Lw    r9 , 0(r4)
    Sll   r9 , r9 , 1
    Sw    r9 , 0(r4)
    Addiu r4 , r4 , 4
    Bne   r4 , r10, loop
    Nop
```

Il s'agit de la boucle principale d'une fonction qui multiplie la valeur d'un tableau d'entiers par 2.

```
    for (i = 0; i != size; i++) {
          v[i] = 2 * v[i];
    }
```

*c - Analyser l'exécution de cette boucle dans **SS2** en supposant que l'adresse du début de la boucle est alignée.*

On déroule la boucle de la question c (et on l'optimise légèrement) :

```
Loop :
    Lw    r9 , 0(r4)
    Lw    r19, 4(r4)
    Sll   r9 , r9 , 1
    Sll   r19, r19, 1
    Addiu r4 , r4 , 8
    Sw    r9 ,-8(r4)
    Bne   r4 , r10, loop
    Sw    r19, -4(r4)
```

*d - Analyser l'exécution de cette boucle dans **SS2** en supposant que l'adresse du début de la boucle est alignée.*

Exercice 2 :

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;
}
```

La boucle principale de cette fonction s'écrit en assembleur (on suppose que `R8` contient l'adresse de la chaîne `pt`) :

```
Loop :
    Lw    r8 , 0(r8)
    Bne   r8 , r0 , loop
    Addiu r2 , r2 , 1
```

*a - Analyser l'exécution de la boucle à l'aide d'un schéma simplifié dans **SS2**. On suppose que le buffer d'instructions est vide au début de cette séquence et que la première instruction est placée à une adresse alignée.*

*b - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

Exercice 3 :

Analyser l'exécution de la suite d'instructions suivante dans le processeur **SS2** à l'aide d'un schéma simplifié. On suppose que le buffer d'instructions est vide au début de cette séquence et que la première instruction est placée à une adresse alignée.

```
    Addi   r2 , r0 , 4
    Lui    r3 , 0x000c
    Add    r2 , r2 , r2
    Ori    r3 , r3 , 0x4568
    Lw     r2 , 0(r3)
    Lbu    r2 , 0(r2)
    Ori    r2 , r2 , 0x0001
    Bltzal r2 , suite
    Addu   r0 , r0 , r0
suite :
    Jr     r31
    Addu   r31, r31, -8
```

Exercice 4 :

La fonction strupper transforme une chaîne de caractères en majuscules :

```
char * strupper (char * src) {
    int i = 0;

    while (src[i] != '\0') {
        if ((src[i] >= 'a') && (src[i] <= 'z')) {
            src[i] = src[i] - 'a' + 'A';
        }
    i++;
    }
    return src;
}
```

La compilation de cette fonction pour un processeur non-pipeline a produit le code suivant :

```
_strupper:
    Addu  r2 , r0 , r4       ; valeur de retour
    Addi  r11, r0 , 'a'      ; pour la comparaison
    Addi  r12, r0 , 'z'      ; pour la comparaison

_strupper_loop:
    Lb    r8 , 0(r4)         ; lire src[i]
    Slt   r9 , r8 , r11      ; src[i] < 'a'
    Slt   r10, r12, r8       ; 'z' < src[i]
    Or    r10, r10, r9       ; et des 2 conditions
    Bne   r10, r0 , _strupper_endif ; si 1 des 2 cond vraie
    Addi  r8 , r8 , 'A'-'a'  ; transformer en majuscule
    Sb    r8 , 0(r4)         ; ecrire src[i]

_strupper_endif:
    Addiu r4 , r4 , 1        ; caractère suivant
    Bne   r8 , r0 , _strupper_loop

    Jr    r31
```

*a - Transformer ce code de telle façon qu'il puisse être exécuté sur le processeur **SS2**.*

*b - Analyser l'exécution de la boucle sur le processeur **SS2** à l'aide d'un schéma simplifié. On suppose qu'au début de l'exécution de la boucle le buffer d'instructions est vide et que l'étiquette `_strupper_loop` se trouve à une adresse non-alignée.*

c - Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas ou le `if` (du code source) réussit.

d - Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas ou le `if` (du code source) échoue.

e - Sachant que 30% des caractères sont minuscules calculer le CPI et le CPI-utile de la boucle.

*f - Comparer le CPI au CPI obtenu par l'exécution du même code par la réalisation **Mips**.*

Exercice 5 :

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   r10, 0(r8)     ; lire un pixel
    Addu  r10, r10, r9
    Lbu   r11, 0(r10)    ; lire f(pixel)
    Sb    r11, 0(r8)

    Addiu r8 , r8 , 1
    Bne   r8 , r12, loop
```

A l'entrée de la boucle, le registre `R8` contient l'adresse du tableau de pixels (l'image à traiter). Le registre `R12` contient l'adresse de la fin de l'image. Le registre `R9` 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 **SS2**.*

*b - Analyser, à l'aide d'un schéma simplifié, l'exécution de ce programme dans le **SS2**. On suppose qu'au début de l'exécution de la boucle le buffer d'instructions est vide et que l'étiquette `Loop` se trouve à une adresse alignée. Calculer le nombre de cycles pour effectuer une itération.*

c - Comparer le nombre de cycles nécessaires à une itération avec la réalisation Mips.

d - Optimiser le code en changeant l'ordre des instructions. Calculer le nombre de cycles par itération.

*e - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

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

*g - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

h - Dérouler la boucle 1 fois (traitement de deux éléments par itération) et optimiser. Calculer le nombre de cycles par itération et par élément.

*i - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

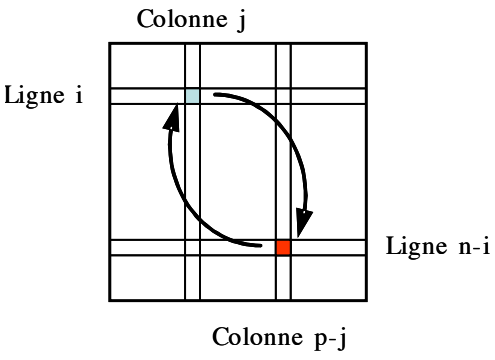
Exercice 6 :

On considère le programme suivant :

```
loop:
    Lbu   r10, 0(r5)
    Lbu   r11, 0(r6)
    Sb    r11, 0(r5)
    Sb    r10, 0(r6)

    Addiu r6 , r6 , -1
    Addiu r5 , r5 ,  1
    Bne   r5 , r7 , loop
```

Ce programme réalise une rotation de 180° de l'image. L'image contient `n` lignes et `p` colonnes. Le pixel de la colonne `j` d'une ligne `i` est échangé avec le pixel de la colonne `p-j` de la ligne `n-i`.



À l'entrée de la boucle, le registre `R5` contient l'adresse du premier pixel de la ligne `i`. Le registre `R7` contient l'adresse du dernier pixel de la ligne `i`. Le registre `R6` contient l'adresse du dernier pixel de la ligne `n-i`. On suppose que l'étiquette `loop` correspond à une adresse multiple de 8.

*a - Modifier le programme pour qu'il soit exécutable sur le **SS2**.*

*b - Analyser, à l'aide d'un schéma simplifié, l'exécution de ce programme dans le **SS2**. On suppose qu'au début de l'exécution de la boucle le buffer d'instructions est vide et que l'étiquette `Loop` se trouve à une adresse alignée. Calculer le nombre de cycles pour effectuer*

une itération.

c - Comparer le nombre de cycles nécessaires à une itération avec la réalisation Mips.

d - Optimiser le code en changeant l'ordre des instructions. Calculer le nombre de cycles par itération.

e - Comparer le nombre de cycles nécessaires à une itération avec la réalisation Mips.

Exercice 7 :

Le code suivant est la boucle principale d'une fonction qui calcule la valeur absolue des éléments d'un tableau d'entiers. À l'entrée de la boucle `R5` contient l'adresse du tableau, le registre `R6` contient l'adresse de la fin du tableau.

```
loop:
    Lw    r7 , 0(r5)    ; lire un élément
    Bgez  r7 , _endif
    Sub   r7 , r0 , r7  ; calculer l'opposé
    Sw    r7 , 0(r5)

_endif:
    Addiu r5 , r5 , 4
    Bne   r5 , r6 , loop
```

a - Modifier le programme pour qu'il soit exécutable sur le SS2.

*b - Analyser, à l'aide d'un schéma simplifié, l'exécution de ce programme dans le **SS2**. On suppose qu'au début de l'exécution de la boucle le buffer d'instructions est vide et que l'étiquette `Loop` se trouve à une adresse alignée. Calculer le nombre de cycles pour effectuer une itération.*

*c - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

d - Optimiser le code en changeant l'ordre des instructions. Calculer le nombre de cycles par itération.

*e - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

f - Dérouler la boucle 1 fois (traitement de deux éléments par itération) et optimiser. Calculer le nombre de cycles par itération et par élément.

*g - Comparer le nombre de cycles nécessaires à une itération avec la réalisation **Mips**.*

Last modified on Nov 22, 2017, 2:41:19 PM

▶ **Attachments** (1)