

# M1 SESI – Architecture des Processeurs et Optimisation

## TD2 – Pipeline MIPS

Ce TD se focalise sur les points suivants :

- Le schéma d'exécution des instructions dans la réalisation pipeline du Mips
- La dépendance de données dans le pipeline
- Le problème des branchements dans le pipeline
- Évaluation du nombre cycles nécessaires pour l'exécution d'un code, CPI, CPI-utile

### Exercice 1 :

Montrer à l'aide d'un schéma détaillé l'exécution de l'instruction `SLLV rd, rt, rs` (Shift Left Logic Variable) dans le pipeline du Mips-32.

### Exercice 2 :

Montrer à l'aide d'un schéma détaillé l'exécution de l'instruction `BLTZAL rs, label` (Branch if Less Than Zero And Link) dans le pipeline du Mips-32.

### Exercice 3 :

On considère une nouvelle instruction de branchement conditionnel : l'instruction `BEQPI` (Branch if equal and post increment).

```
Beqpi    rs, rt, Label
```

Il s'agit d'une instruction de format I. Cette instruction réalise l'opération suivante. Comme pour `BEQ`, le contenu de RS est comparé au contenu de RT. En cas d'égalité, on se branche à l'instruction portant l'étiquette Label, sinon le branchement échoue et on continue en séquence. De plus, une fois le test effectué, le contenu de RT est incrémenté.

Le schéma d'exécution du pipeline du Mips convient-il à l'exécution de cette instruction ? Si oui, proposer un schéma détaillé montrant l'exécution de cette instruction. Si non, expliquer pourquoi.

### Exercice 4 :

On considère une nouvelle instruction de branchement conditionnel : l'instruction `BEQPD` (Branch if equal and pre-decrement).

```
Beqpd    rs, rt, Label
```

Il s'agit d'une instruction de format I. Cette instruction réalise l'opération suivante. Comme pour `BEQ`, le contenu de RS est comparé au contenu de RT. En cas d'égalité, on se branche à l'instruction portant l'étiquette Label, sinon le branchement échoue et on continue en séquence. En plus, avant d'effectuer le test, le contenu de RT est décrémenté.

Le schéma d'exécution du pipeline du Mips convient-il à l'exécution de cette instruction ? Si oui, proposer un schéma détaillé montrant l'exécution de cette instruction. Si non, expliquer pourquoi.

### Exercice 5 :

Montrer à l'aide d'un schéma détaillé l'exécution de la suite d'instructions suivante dans le pipeline du Mips-32.

```
Add      r3 , r2 , r1
Add      r3 , r3 , r1
```

Quelle est la condition du déclenchement des bypass ?

## Exercice 6 :

Montrer à l'aide d'un schéma détaillé l'exécution de la suite d'instructions suivante dans le pipeline du Mips-32.

```
Add    r0 , r2 , r1
Add    r3 , r0 , r1
```

Donner l'expression exacte de la condition du déclenchement des bypass ?

## Exercice 7 :

Montrer à l'aide d'un schéma détaillé l'exécution de la suite d'instructions suivante dans le pipeline du Mips-32.

```
Lw      r3 , 0 (r2)
Add     r3 , r3 , r1
```

## Exercice 8 :

Analyser l'exécution de la suite d'instructions suivante dans le processeur Mips-32 à l'aide d'un schéma simplifié.

```
Addiu   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 9 :

La réalisation du processeur Mips-32 nécessite la mise en œuvre de 8 bypass (raccourcis). Pour chacun de ces bypass, proposer un exemple de suite d'instructions qui illustre la nécessité de ce bypass.

## Exercice 10 :

La fonction strlen calcule le nombre de caractères :

```
unsigned int strlen (char * str) {
    unsigned int i = 0;
    while (str [i] != '\0') {
        i++;
    }
    return (i);
}
```

Différents codes assembleur peuvent être produits pour cette fonction pour un processeur non-pipeline :

1ère version :

```

_strlen:
    Addiu    r29, r29, -1*4
    Addu     r2, r0 , r0      ; initialisation
_strlen_loop:
    Lb       r9 , 0(r4)       ; lire 1 caractere
NOP → Beq     r9 , r0 , _strlen_end_loop ; fin de chaine ?
    Addiu    r4 , r4 , 1      ; caractere suivant
    Addiu    r2 , r2 , 1      ; caractere suivant
    J        _strlen_loop    ; boucle

_strlen_end_loop:
    Addiu    r29, r29, 1*4
    Jr       r31

```

2ème version :

```

_strlen:
    Addiu    r29, r29, -1*4
    Addiu    r2, r0 , -1      ; initialisation
_strlen_loop:
    Lb       r9 , 0(r4)       ; lire 1 caractere
    Addiu    r4 , r4 , 1      ; caractere suivant
    Addiu    r2 , r2 , 1      ; caractere suivant
    Bne      r9 , r0 , _strlen_loop ; fin de chaine ?

    Addiu    r29, r29, 1*4
    Jr       r31

```

3ème version :

```

_strlen:
    Addiu    r29, r29, -1*4
    Addiu    r8, r4 , 1      ; sauvegarde adresse debut
_strlen_loop:
    Lb       r9 , 0(r4)       ; lire 1 caractere (octet)
    Addiu    r4 , r4 , 1      ; caractere suivant
    Bne      r9 , r0 , _strlen_loop ; fin de chaine ?
    Subu     r2 , r4 , r8      ; calculer nbr de carac

    Addiu    r29, r29, 1*4
    Jr       r31

```

Pour chacune des trois versions, transformer le code de telle façon qu'il puisse être exécuté sur le processeur Mips pipeline à 5 étages.

Analyser l'exécution de la boucle sur le processeur Mips à l'aide d'un schéma simplifié.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle.

Calculer le CPI et le CPI-utile de la boucle.

## Exercice 11 :

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:
    Addiu    r29, r29, -1*4

    Addu     r2, r0, r4      ; valeur de retour
    Addiu    r11, r0, 'a'    ; pour la comparaison
    Addiu    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
    Addiu    r8, r8, 'A'-'a'; transformer en majuscule
    Sb       r8, 0(r4)

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

    Addiu    r29, r29, 1*4
    Jr       r31

```

Transformer ce code de telle façon qu'il puisse être exécuté sur le processeur Mips pipeline à 5 étages.

Analyser l'exécution de la boucle sur le processeur Mips à l'aide d'un schéma simplifié dans le cas où le `if` (du code source) réussit, puis dans le cas où le `if` échoue.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas où le `if` réussit.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas où le `if` échoue.

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

## Exercice 12 :

La fonction suivante calcule le PGCD de deux nombres positifs non nuls :

```

unsigned int pgcd (unsigned int a, unsigned int b) {
    while (a != b) {
        if (a < b) {
            b = b - a;
        }
        else {
            a = a - b;
        }
    }
    return a;
}

```

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

```

_pgcd:
    Beq      r4, r5, _pgcd_end

```

```

_pgcd_loop:
    Sltu    r8 , r4 , r5    ; a < b
    Beq     r8 , r0 , _pgcd_else
    Subu    r5 , r5 , r4
    J       _pgcd_endif

_pgcd_else:
    Subu    r4 , r4 , r5
_pgcd_endif:
    Bne     r4 , r5 , _pgcd_loop

_pgcd_end:
    Add     r2 , r4 , r0    ; valeur de retour
    Jr      r31

```

Transformer ce code de telle façon qu'il puisse être exécuté sur le processeur Mips pipeline à 5 étages.

Analyser l'exécution de la boucle sur le processeur Mips à l'aide d'un schéma simplifié, dans le cas où le `if` (du code source) réussit, puis dans le cas où le `if` échoue.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas où le `if` réussit.

Calculer le nombre de cycles nécessaires à l'exécution d'une itération de la boucle dans le cas où le `if` échoue.

Sachant que dans un cas sur deux le `if` échoue, calculer le CPI et le CPI-utile de la boucle.