TD 2 : ARM

1 Instructions Arithmétiques Et Logiques

1.1 Avec le format d'instruction ARM, quelle est l'amplitude des constantes que l'on peut avoir dans les instructions arithmétiques avec immédiats ?

L'instruction de multiplication sur des entiers prend plus d'une dizaine de cycles d'horloge sur la plupart des processeurs. Les instructions arithmétiques et logiques prennent généralement un seul cycle d'horloge. Il est donc plus efficace d'implémenter la multiplication du contenu d'un registre par une constante en utilisant des opérations comme l'addition, la soustraction et les décalages.

Soit le code assembleur MIPS32 suivant :

```
.text

main:
    # A *= 17
    # TODO

    jr $ra

.data
A : .word 0x00010101
```

Et sa version ARM :

```
# A *= 17
# TODO

SWI 0x11 @ Stop program execution

.data
A: .word 0x00010101
```

On cherche à écrire le code assembleur MIPS32 qui correspond au code C suivant : `a *= 17` (où `a` est en entier). Ce code remplacera le commentaire `# TODO`.

1.2 En assembleur MIPS, écrire la solution la plus rapide en nombre de cycles.

1.3 Traduire le programme en assembleur ARM.

1.4 En assembleur ARM, écrire le code assembleur qui multiplie le registre `r1` par 33, par 37 et par 105.

2 Condionnelles

2.1 Écrire la séquence d'instructions qui place dans un registre la valeur absolue du contenu de ce registre. En MIPS32. En ARM avec un branchement (sans instruction conditionnelle). En ARM avec une instruction condionnelle (sans branchement).

3 Modes D'Adressage

ARM dispose de plusieurs modes d'adressage.

Mode	Assembleur	Action
Déplacement 12 bits Pré-indexé	`[reg, N]`	Accès à l'adresse `reg + N`
Déplacement 12 bits Pré-indexé avec mise à jour	`[reg, N] !`	Accès à l'adresse `reg + N` `reg += N`
Déplacement 12 bits Post-indexé	`[reg], N`	Accès à l'adresse `reg` `reg += N`
Déplacement dans `reg1` Pré-indexé	`[reg0, ±reg1, décalage]`	Accès à l'adresse `reg0 + (reg1` décalé selon `décalage)`
Déplacement dans `reg1` Pré-indexé avec mise à jour	`[reg0, ±reg1, décalage] !`	Accès à l'adresse `reg0 + (reg1` décalé selon `décalage)` `reg0 += (reg1` décalé selon `décalage)`
Déplacement dans `reg1` Post-indexé	`[reg0], ±reg1, décalage`	Accès à l'adresse `reg0` `reg0 += (reg1` décalé selon `décalage)`

On suppose que le contenu de la mémoire à partir de l'adresse `0xC000 0000` est le suivant :

Adresse	Contenu (en hexadécimal)
C000 0000	1020 3040
C000 0004	3223 2233
C000 0008	5445 4455
C000 000C	7667 6677

Adresse	Contenu (en hexadécimal)
C000 0010	9889 9988
C000 0014	BAAB BBAA
C000 0018	DCCD DDCC
C000 001C	EFFE ABCD

Initialement, `r0 = C000 0000` et `r1 = 4`

3.1 Quels sont les contenus des registres après exécution du programme suivant ?

```
LDR r3, [r0, 4] !
LDR r4, [r0], 14          // 14 en hexadécimal = 20 en décimal
LDR r5, [r0, -r1, LSL#1]
```

4 Boucles

```
for (i = 0; i < 8; ++i)
    s += X[i] + Y[i];
```

```
for (i = 1; i < 7; ++i)
    Y[i+1] += X[i] + X[i-1];
```

`X` et `Y` sont des tableaux d'entiers (de 32 bits). On utilisera le chargement et l'écriture mémoire à partir des adresses. L'adresse de `X` est `0x1000 0000`, l'adresse de `Y` est `0x2000 0000` et l'adresse de `S` est `0x0000 0100`. `r1` contient `0x1000 0000` et `r2` contient `0x2000 0000` et `r3` contient `0x0000 0100`.

4.1 Écrire l'assembleur ARM qui correspond aux boucles.