TD 1 : Modèles D'Exécution - Instructions Arithmétiques Et Logiques

Prénom : _____ **Nom :** _____

1 Quatre modèles d'exécution

Le modèle d'exécution des instructions est donné parle couple (n, m) où n est le nombre d'opérandes spécifié par instruction, et m le nombres d'opérandes mémoires.

Soient les quatre modèles :

(0, 0) Machine à pile. Dans une machine à pile, une instruction arithmétique ou logique dépile les deux opérandes en sommet d'une pile, effectue l'opération et empile le résultat. L'instruction `push` empile un opérande mémoire. L'instruction `pop` dépile un opérande mémoire.

(1, 1) Machine à accumulateur. Un seul opérande de type mémoire est spécifié dans les instructions arithmétiques et logiques, l'autre étant un registre (implicite) : accumulateur.

(2, 1) Les instructions arithmétiques et logiques ont deux opérandes, l'un dans un registre (qui est à la fois source et destination) et l'autre étant un opérande mémoire. Ce modèle inclus le modèle (2, 0) où les deux instructions sont dans des registres.

(3, 0) Machine chargement-rangement. Les accès mémoires ne sont possibles que par les instructions de chargement `load` et rangement `store`. Les instructions arithmétiques et logiques ne portent que sur des opérandes situés dans des registres.

Les instructions disponibles dans les quatre processeurs considérés avec ces modèles d'exécution sont données dans le tableau suivant. Pour les instructions mémoire, on ne se préoccupe pas des modes d'adressage. Pour les multiplications, on considère que le produit de deux registres ou d'un registre et d'un mot mémoire peut être contenu dans le registre résultat.

Modèle (0, 0)	Modèle (1, 1)	Modèle (2, 1)	Modèle (3, 0)
`push X`	`load X` $accu \leftarrow X$	`load Ri, X` $Ri \leftarrow X$	`load Ri, X` $Ri \leftarrow X$
`pop X`	`store X` $X \leftarrow accu$	`store Ri, X` $X \leftarrow Ri$	`store Ri, X` $X \leftarrow Ri$
`add`	`add X` $accu \leftarrow accu + X$	`add Ri, X` $Ri \leftarrow Ri + X$	`add Ri, Rj, Rk` $Ri \leftarrow Rj + Rk$
`sub`	`sub X` $accu \leftarrow accu - X$	`sub Ri, X` $Ri \leftarrow Ri - X$	`sub Ri, Rj, Rk` $Ri \leftarrow Rj - Rk$
`mul`	`mul X` $accu \leftarrow accu \times X$	`mul Ri, X` $Ri \leftarrow Ri \times X$	`mul Ri, Rj, Rk` $Ri \leftarrow Rj \times Rk$

Les variables `A`, `B`, `C` *et* `D` *sont initialement en mémoire.*

1.1 Écrire les séquences de code pour les quatre machines pour `A = B + C`. Donner le nombre d'instructions et le nombre d'accès mémoire.

1.2 Écrire les séquences de code pour les quatre machines pour la suite d'instructions suivantes. Donner le nombre d'instructions et le nombre d'accès mémoire.

```
A = B + C
B = A + C
D = A - B
```

1.3 Écrire les séquences de code pour les quatre machines pour l'expression suivante. Donner le nombre d'instructions et le nombre d'accès mémoire.

W = (A + B) × (C + D) + (D × E)

2 Instructions arithmétiques et logiques MIPS

Pour cette partie, on utilise le jeu d'instructions MIPS32.
Les instructions `add` *et* `sub` *effectuent l'opération sur des données en complément à deux. S'il n'y a pas de débordement, le résultat est rangé dans le registre destination. S'il y a débordement, alors il y a une exception et le registre destination est inchangé.*
Les instructions `addu` *et* `subu` *effectuent des opérations modulo* 2^{32}*. Il n'y a jamais d'exception.*

2.1 On considère l'instruction `add`. Donner la plus grande valeur et la plus petite valeur représentable dans les registres qui contiennent des entiers signés en complément à deux. Pour les deux valeurs, on demande la forme hexadécimal et son équivalent décimal.

2.2 On considère l'instruction `addu`. Donner la plus grande valeur et la plus petite valeur représentable dans les registres qui contiennent des entiers non signés. Pour les deux valeurs, on demande la forme hexadécimal et son équivalent décimal.

On suppose maintenant que les registres du processeur contiennent initialement les valeurs suivantes :

Registre	Contenu (en hexadécimal)
R0	0x0000~0000
R1	0x3000~1000
R2	0x5000~1016
R3	0x8432~A380

Registre	Contenu (en hexadécimal)
R4	0xECDF~1234
R5	0x8976~5432
R6	0xC234~5678
R7	0xA012~3456

2.3 En partant à chaque fois du contenu de la table précédente, donner le contenu des registres après exécution des instructions MIPS32. On considérera que les registres contiennent des entiers signés en complément à deux. Dans chaque cas, on considérera que l'instruction est à l'adresse 0x1000 0000

2.3.1 `add R8, R1, R2`

2.3.2 `add R9, R6, R7`

2.3.3 `sub R10, R1, R2`

2.3.4 `sub R11, R6, R7`

2.3.5 `addu R8, R1, R2`

2.3.6 `addu R9, R6, R7`

3 Prise en main avec le simulateur ARM

Installer `ARMSim` http://armsim.cs.uvic.ca/Downloads.html et ne pas hésiter à lire la documentation http://armsim.cs.uvic.ca/Documentation.html

Dans `ARMSim`, charger le fichier `arm_exemple.s` (à droite) et l'exécuter pas à pas (`Debug`, `Step Over (F10)`). Observer les valeurs en hexadécimal et en décimal signé.

```
        MOV r0,#100
        MOV r1,#112
        ADD r2,r0,r1
        ADD r3,r1,r2
        MOV r4,r3
        SUB r5,r1,r2
        RSB r6,r2,r3
        EOR r7,r1,r2      @ OU exclusif
FIN: SWI 0x11 @ Stop program execution
```

3.0.7 Écrire un programme qui charge une valeur N (comprise entre 0 et 255) dans le registre `R0` puis calcule les valeurs suivantes sans utiliser l'instruction de multiplication

`R1` $= 8 \times$ `R0`
`R2` $= 17 \times$ `R0`
`R3` $= 15 \times$ `R0`
`R4` $= 10 \times$ `R0`

Pour charger une constante comprise entre 0 et 255, on utilise `MOV Reg, #N`. Pour charger une constante 32 bits, on utilise la pseudo instructions `LDR Reg, =0x12345678`.

4 Place libre (remarques, impressions, fin de réponse...)