TD n° 1 : MODELES D'EXECUTION INSTRUCTIONS ARITHMETIQUES ET LOGIQUES

1. Quatre modèles d'exécution

Le modèle d'exécution des instructions est donné par le couple (n, m) où n est le nombre d'opérandes spécifié par instruction, et m est le nombre d'opérandes mémoire.
Soient les quatre modèles

a) Modèle (3,0) : machine chargement –rangement. Les accès mémoire 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.
b) Modèle (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) : l'accumulateur
c) Modèle (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. Le modèle (2,1) inclut le modèle (2,0) où les deux instructions sont dans des registres
d) Modèle (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.

Les instructions disponibles dans les quatre processeurs considérés avec ces modèles d'exécution sont données dans la table ci-dessous. 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. :

M0 (0,0)	M1 (1,1)	M2 (2,1)	M3 (3,0)
PUSH X	LOAD X (accu ← X)	LOAD Ri,X (Ri ← X)	LOAD Ri,X
POP X	STORE X (X ← accu)	STORE Ri,X (X ← Ri)	STORE Ri,X
ADD	ADD X (accu ← accu + X)	ADD Ri,X (Ri ← Ri + X)	ADD Ri,Rj,Rk (Ri ← Rj+Rk)
SUB	SUB X (accu ← accu - X)	SUB Ri,X (Ri ← Ri - X)	SUB Ri,Rj,Rk (Ri ← Rj - Rk)
MUL	MUL X (accu ← accu * X)	MUL Ri,X (Ri ← Ri * X)	MUL Ri,Rj,Rk (Ri ← Rj * Rk)

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

Q 1) Ecrire les séquences de code pour les quatre machines pour A = B + C. Donner le nombre d'instructions et le nombre d'accès mémoire

Q2) Ecrire 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 ;

Q3) Ecrire les séquences de code pour les quatre machines pour l'expression. 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.
NB : les instructions ADD et SUB effectuent l'opération sur des données en complément à 2. 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.
NB : les instructions ADDU et SUBU effectuent des opérations modulo 2^{32}. Il n'y a jamais d'exception.

Q 4) 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 hexadécimale et son équivalent décimal.

Q 5) 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 hexadécimale et son équivalent décimal.

On suppose maintenant que les registres du processeur contiennent initialement les valeurs données dans la table 1 :

Registre	Contenu (hexa)
R0	00000000
R1	30001000
R2	50001016
R3	8432A380
R4	ECDF1234
R5	89765432
R6	C2345678
R7	A0123456

Table 1

Q 6) En partant à chaque fois du contenu de la table 1, 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 1000 0000H.

a) ADD R8,R1,R2
b) ADD R9,R6,R7
c) SUB R10, R1,R2
d) SUB R11, R6,R7
e) ADDU R8,R1,R2
f) ADDU R9,R6,R7

3. Travail personnel (à faire avant le prochain TD).

On utilise le simulateur ARM, chargeable à l'adresse
http://armsim.cs.uvic.ca/DownloadARMSimSharp.html
Le mode d'emploi (en anglais) est disponible à l'adresse
http://armsim.cs.uvic.ca/AttachedFiles/ARMSim_UserGuide4Plus.pdf

3.1 Utilisez le fichier Exemple.s ci-dessous.

```
@  Exemple de programme TD1
      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
```

1. Sauvegarder le fichier sous forme exemple.s
2. Charger Exemple.s via File, puis Load
3. Exécuter le programme pas à pas via Debug, puis Step into (F10)
4. On observera les résultats en hexadécimal et en décimal signé.

3.2 Ecrire des programmes.s qui chargeront une valeur N (comprise entre 0 et 255) dans le registre r0, puis chargeront dans les registres les valeurs suivantes (sans utiliser l'instruction de multiplication) :

- r1 = 8*r0
- r2 = 17*r0
- r3 = 15*r0
- r4 = 10*r0

NB :
Pour charger une constante comprise entre 0 et 255, on utilise mov reg, #N
Pour charger une constante 32 bits, on utilise la pseudo instruction ldr reg, =0x12345678