Poly d'exercices

1 - Codage des instructions assembleur

En utilisant les documents sur le codage des instructions du MIPS, donner le codage de chacune des instructions du programme assembleur suivant sachant que la première instruction est placée à l'adresse 0x400020.

```
Loop:   add  $t1, $s3, $s3
        add  $t1, $t1, $t1
        lw   $t0, 0($t1)
        j    Exit
```

400 004	0	19	19	9	0	32
400 008	0	9	9	9	0	32
400 00C	35	9	8	0		
400 010	2	100008				

En binaire

000000	10011	10011	01001	00000	100000	0x02734820
000000	01001	01001	01001	00000	100000	0x01294820
100011	01001	01000	0000 0000 0000 0000			0x8d280000
000010		00 0100 0000 0000 0000 0010 0000				0x08100008

2 - Désassemblage

Retrouver les instructions assembleur correspondant au code machine ci-dessous :

1	**0x00AF8020**					*add*	*$s0, $a1, $t7*
3	**0x23BD0004**				*addi*	*$sp, $sp, 4*	
7	**0x0085402A**	*slt*	*$8, $4, $5*			*slt*	*$t0, $a0, $a1*

3 – Compilation

On considère le bout de code C suivant :

```
for (i = 0; i < 100; i++)
     h = h + c;
```

On suppose que a et b sont des tableaux de mots dont l'adresse des premiers éléments est respectivement dans les registres $a0 et $a1. On suppose aussi que le registre $t0 est associé à la variable i et que le registre $s0 est associé à la variable c. Donner le code assembleur équivalent pour le MIPS. Combien d'instructions sont exécutées pour cette boucle ? Combien y a-t-il d'accès à la mémoire ?

```
     addiu    $t0, $zero, 0
Loop:
     slti     $t2, $t0, 100       # $t2 = 1 si i < 100,
     beq      $t2, $zero, Exit
     add      $s1, $s1, $s0       # $h = h + c
     addiu    $t0, $t0, 1         # on incrémente i
```

```
        j     Loop
    Exit :
```

Pour 100 itérations, il y a 5 instructions dans la boucle + 1 avant, soit 500+1+2 instructions, soit 503.
Une instruction de moins pour le do – While dans la boucle, soit 401 (et pas de test).

4 – Architecture du MIPS

On considère l'architecture du chemin de données du processeur MIPS étudiée en cours et donnée sur le document joint en annexe. Sur ce document, les lettres A .. P sont associées à certains bus. Le but de cet exercice est d'indiquer les valeurs présentes sur ces bus pour 3 instructions assembleur.

Pour déterminer ces valeurs, on supposera que l'instruction présente sur le bus de sortie de la mémoire d'instructions (bus repéré avec la lettre A) reste aussi longtemps qu'on le souhaite, pour obtenir une valeur constante et stable sur les bus. On laissera donc le temps à toutes les unités de cette architecture d'effectuer leur opération.

On considère les 3 instructions suivantes (avec $a1 = 0x0000FF00 et $t7=0) :

```
Pgm:    add  $s0, $a1, $t7
        andi $s1, $s0, 15
        beq  $s1, $zero, Exit
        ...
Exit:
```

L'étiquette Pgm est à l'adresse 0x00400024 et l'étiquette Exit est à l'adresse 0x00400040.

On pourra remplir le tableau de la page suivante. On respectera les notations conseillées, soit hexadécimale pour des valeurs de 16 à 32 bits, et binaire pour les autres. Vous indiquerez sur votre copie les différents calculs si nécessaire. Les valeurs sans importance pour l'instruction sont notées *inutile* dans le tableau.

	add $s0, $a1, $t7	andi $s1, $s0, 15	beq $s1, $zero, Exit
A (hexa)	0x00AF8020	0x3211000F	0x12200004
B (binaire)	000000	001100	000100
C (binaire)	00101	10000	10001
D (binaire)	01111	10001	00000
E (binaire)	10000	00000	00000
F (hexa)	0x8020	0x000F	0x0004
G (binaire)	10000	10001	ind
H (hexa)	0x0000FF00	0x0000FF00	0x0
I (hexa)	0xFFFF8020	0x0000000F	0x00000004
J (hexa)	0x0	0x0000000F	0x0
K (hexa)	0x0000FF00	0x0	0x0
L (binaire)	0	1	1
M (hexa)	0x0000FF00	0x0	ind
N (hexa)	0x00400028	0x0040002C	0x00400030
O (hexa)	0x003E00A8	0x00400068	0x00400040
P (hexa)	0x00400028	0x0040002C	0x00400040