TD n° 2 : ORGANISATION DES DONNEES EN MEMOIRE – INSTRUCTIONS MEMOIRE

1. Introduction

La première partie du TD-TP utilise le jeu d'instructions MIPS et QtSpim.
La seconde partie (***travail personnel***) utilise le jeu d'instructions ARM et ARMSIM.

2. Jeu d'instructions MIPS

A. Alignement mémoire

Soit la déclaration de variables C suivante

```
char X[9]=  {0x10, 0x32,0x54,0x76,0x98,0xBA,0xDC,0xEF,0x01}
short Y[2] = {0x1234, 0x5678}
int Z = 0xABCDEFACp
float A=1.5
double B=1.5
int C= 10
char D[]= "hello world"
char E[]= "fin de l'exercice"
```

Cette déclaration correspond à la zone .data du programme MIPS32 TD2L3E1.s fourni.

```
.data
X : .byte  0x10, 0x32,0x54,0x76,0x98,0xBA,0xDC,0xEF,0x01
Y : .half 0x1234, 0x5678
Z : .word 0xABCDEFAC
A : .float 1.5
B : .double 1.5
C : .word 10
D : .asciiz "hello world"
E : .asciiz "fin de l'exercice"
```

Avec le simulateur QtSpim, après chargement du programme, observer le contenu mémoire dans la zone « data » (user data segment).
Quelles sont les adresses des différentes variables X à E ? (par exécution du programme en pas à pas) ?
En déduire les règles d'alignement

B. Big endian et little endian – Implantation mémoire

Big Endian	Octet 0 MSB	Octet 1		Octet n LSB
Little Endian	Octet n MSB	Octet n-1		Octet 0 LSB

En observant l'implantation mémoire du programme TD2L3E1.s, peut-on en déduire la nature « big endian » ou « little endian » pour MIPS32 ?

C. Instructions mémoire

Soit le programme TD2L3E2.s dont la section .data est la suivante :

```
.data
X : .byte 0x10,0x32,0x54,0x76, 0x98,0xBA,0xDC,0xEF,0x01
Y : .word 0x76543210, 0xEFDCBA98
```

Quels sont les contenus des registres après exécution des instructions suivantes

```
la $t0,X
lw $t1,0($t0)
lw $t2, 8($t0)
lb $t3, 5($t0)
lh $t4, 2($t0)
lhu $t5, 6($t0)
lbu $t6, 3($t0)
la $t0, Y
lw $t7, 0($t0)
lw $s0, 4($t0)
```

D. Suite de Fibonnaci

Le programme MIPS 32 TD2L3E3.s range dans le tableau d'entiers 32 bits X les deux premières valeurs de la suite de Fibonnaci. Sans utiliser de boucle, compléter le programme pour écrire les 4 valeurs suivantes. Ecrire une variante qui rangera les 6 valeurs dans un tableau d'octets.

3. Jeu d'instructions ARM

Voir en annexe des détails sur le jeu d'instructions ARM.

E. Implantation mémoire

Le programme TD2L3A1.s a la zone donnée suivante :

```
.data
A: .byte 0x39, 0x32, 0x24, 0xAB, 0xDA
B: .word 0x98765432, 0xFA00, 0xFEDCBA98
C: .asciz "Hello world"
D: .word 345
```

Exécuter ce programme pour observer les règles d'alignement mémoire. ARM utilise-t-il « big endian » ou « little endian » ?

F. Instructions mémoire

Exécuter le programme TD2L3A2.s dont la zone donnée est la suivante :

```
.data
A: .byte 0x10, 0X32, 0x54, 0x76, 0x98, 0xBA, 0xDC, 0xEF, 0x01
```

Quels sont les contenus des registres après exécution du programme suivant :

```
LDR r0,=A       @chargement adresse de A
MOV r1, #4
LDR r2,[r0,#4]
LDR r3,[r0,#4]!
LDR r4,[r0],#8
LDR r5,[r0,-r1,LSL#1]
SWI 0x11 @ Stop program execution
```

G. Fibonnaci

Le programme ARM TD2L3A3.s range dans le tableau d'entiers 32 bits X les deux premières valeurs de la suite de Fibonnaci. Sans utiliser de boucle, compléter le programme pour écrire les 4 valeurs suivantes. Ecrire une variante qui range dans un tableau d'octets.

H. Annexe : jeu d'instructions ARM (simplifié)

Le jeu d'instructions ARM a 16 registres (R0 à R15) de 32 bits. R15=CP et R14=Registre de lien et R13 = Pointeur de pile. R0 est un registre normal (non câblé à 0)
Le format des instructions mémoire pour octets et mots de 32 bits est le suivant

31-28								19-16	15-12	11-0
Cond	01	I	P	U	B	W	L	n	d	Déplacement

Rs est le registre source (s est le numéro)
Rd est le registre destination (d est le numéro)
Lorsque I =0, le déplacement est la valeur sur 12 bits non signée
Lorsque I=1, le déplacement est obtenu comme suit : Déplacement = décalage [Rm]
Où les 11 bits sont interprétés comme suit

11-5		3-0
Décalage (nb de bits)	0	m (numéro de registre)

La signification des bits P, U, B, W et L n'est pas détaillée ici.
Les instructions mémoire sont les suivantes

Instruction	Signification	Action
LDR	Chargement mot	Rd ← Mem_{32}(AE)
LDRB	Chargement octet	Rd ← Mem_8 (AE)
STR	Rangement mot	Mem_{32}(AE) ← Rd
STRB	Rangement octet	Mem_8 (AE) ← Rd

Les modes d'adressage avec la syntaxe assembleur sont résumés dans la table ci-dessous.

Mode	Assembleur	Action
Déplacement 12 bits, Pré-indexé	[Rn, #deplacement]	Adresse = Rn + déplacement
Déplacement 12 bits, Pré-indexé avec mise à jour	[Rn, #deplacement] !	Adresse = Rn + déplacement Rn ← Adresse
Déplacement 12 bits, Post-indexé	[Rn], #deplacement	Adresse = Rn Rn ← Rn + déplacement
Déplacement dans Rm Préindexé	[Rn, ± Rm, décalage]	Adresse = Rn ± [Rm] décalé
Déplacement dans Rm Préindexé avec mise à jour	[Rn, ± Rm, décalage] !	Adresse = Rn ± [Rm] décalé Rn ← Adresse
Déplacement dans Rm Postindexé	[Rn], ± Rm, décalage	Adresse = Rn Rn ← Rn ± [Rm] décalé
Relatif		Adresse = R15 + déplacement