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

1. IMPLANTATION MEMOIRE

Soit la déclaration de variables C suivante

```
unsigned char  toto [17] ;
short a,b,c, d, e, f ;
double w[10], y[8][8];
float z[10], foo[4][ 5];
int ouf, cest, fini ;
```

Q 1) : Si l'on suppose que la variable toto[0] est à l'adresse 1000 0000_H, donnez les adresses hexadécimales des variables toto [16], a, f, y[0][0], foo[0][0], fini

	Décimal	Hexadécimal	Adresse
toto	0	0	10000000
toto(16)	16	10	10000010
a	18	12	10000012
b	20	14	10000014
c	22	16	10000016
d	24	18	10000018
e	26	1A	1000001A
f	28	1C	1000001C
w(0)	32	20	10000020
Y(0)(0)	112	70	10000070
Z(0)	624	270	10000270
foo(0)(0)	664	298	10000298
ouf	744	2E8	100002E8
cest	748	2EC	100002EC
fini	752	2F0	100002F0

Toto[16] : 1000 0010
A : 1000 0012
F : 1000001C
Y[0][0] : 10000070
Foo [0][0] : 10000298
Fini : 100002F0

2. 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

La déclaration en C suivante définit une occupation mémoire; les valeurs sont notées en hexadécimal en commentaire. Le placement est supposé aligné, i.e un nombre adéquat d'octets

est sauté pour que chaque objet soit aligné sur ses bornes naturelles. Donner le contenu des octets mémoire concernés, en supposant que la structure `data` est implantée à partir de l'adresse 0, pour les deux cas,big endian, et little endian.

```
struct {
        int     a;      //%x11121314
        double  b;      //%x2122232425262728
        char*   c;      //%x31323334
        char    d[7];   //'A', 'B', 'C','D', 'E', 'F', 'G'*/
        short   e;      //%x5152
        int     f;      //%x61626364
        } data;
```

Adresse	Donnée en BE		Adresse	Données en LE
00	11		00	14
01	12		01	13
02	13		02	12
03	14		03	11
04	x		04	x
05	x		05	x
06	x		06	x
07	x		07	x
08	21		08	28
09	22		09	27
0A	23		0A	26
0B	24		0B	24
0C	25		0C	25
0D	26		0D	23
0E	27		0E	22
0F	28		0F	21
10	31		10	34
11	32		11	33
12	33		12	32
13	34		13	31
14-1A	'A'-'G'		14-1A	'A'-'G'
1B	x		1B	x
1C-1D	51-52		1C-1D	52-51
1E	x		1E	x
1F	x		1F	x
20	61		20	64
21	62		21	63
22	63		22	62
23	64		23	61

3. Instructions mémoire

Jeu d'instructions MIPS32

On suppose que le contenu de la mémoire à partir de l'adresse $C0000000_H$ est le suivant :

Adresse	Contenu
C000 0000	10
C000 0001	32
C000 0002	54
C000 0003	76
C000 0004	98
C000 0005	BA
C000 0006	DC
C000 0007	EF
C000 0008	01

Quels sont les contenus des registres après exécution du programme suivant (on suppose que l'ordre de rangement des octets en mémoire est « little endian »). :

LUI R1, R0, 0xC000 R1 = C000 0000
LW R2, 0(R1) R2 = 76543210
LB R3, 5(R1) R3 = FFFFFFBA
LH R4, 2(R1) R4 = 00007654
LHU R5, 6(R1) R5 = 0000EFDC
LBU R6, 3(R1) R6 = 00000076

Jeu d'instructions ARM

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 et leur effet 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 = CP + déplacement

On suppose que le contenu de la mémoire à partir de l'adresse C0000000 est le suivant :

Adresse	Contenu
C000 0000	10203040
C000 0004	32232233
C000 0008	54454455
C000 000C	76676677
C000 0010	98899988
C000 0014	BAABBBAA
C000 0018	DCCDDDCC
C000 001C	EFFEABCD

Initialement R0 = C0000000 ; R1=4 ;

Quels sont les contenus des registres après exécution du programme suivant :
LDR R3, [R0, 4] !
LDR R4, [R0], 14_H // 20 décimal = 14_H
LDR R5, [R0,-R1,LSL#1]

Instruction	Adresse	
LDR R3, [R0, 4] !	Adresse = C0000000+4	R3= 32232233 R0=C0000004
LDR R4, [R0], 20	Adresse = C0000004	R4=32232233 et R0=C00000018
LDR R5, [R0,-R1,LSL#1]	Adresse = C00000010	R5=98899988 R0 = C0000010