TP 2 : MIPS

1 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    = 0xABCDEFAC;
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_align.s` :

```
.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 `xspim` ou `QtSpim`, après chargement du programme `mips32_align.s`, observer le contenu mémoire dans la zone `.data` (user data segment) en utilisant l'exécution pas à pas.

1.1 Quelles sont les adresses des différentes variables `X` à `E` ? En déduire les règles d'alignement.

2 Big Endian Et Little Endian - Implémentation Mémoire

Big endian	Octect 0 MSB	Octect 1	···	Octect N LSB
Little endian	Octect N MSB	Octect N - 1	···	Octect 0 LSB

2.1 En observant l'implémentation mémoire du programme `mips32_align.s`, peut-on en déduire la nature big endian ou little endian pour MIPS32 ?

3 Instructions Mémoire

Soit le programme `mips32_reg.s` dont la section `.data` est la suivante :

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

3.1 Quels sont les contenus des registres après l'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 $t7, Y
lw $t8, 0($t0)
lw $t9, 4($t0)
jr $ra
```

4 Suite De Fibonnaci

Le programme `mips32_fibonnaci.s` range dans le tableau d'entiers 32 bits `X` les deux premières valeurs de la suite de Fibonnaci.

4.1 Sans utiliser de boucle, compléter le programme pour écrire les 4 valeurs suivantes.

5 Condionnelles

5.1 Écrire le code assembleur MIPS32 qui place dans un registre la valeur absolue du contenu de ce registre (compléter le fichier `mips32_abs.s`).

En code ASCII, les caractères 0 à 9 sont représentés sur 8 bits par `0x30` à `0x39`. A et B sont chacun un caractère ASCII.

5.2 Écrire le code assembleur MIPS32 qui écrit dans la variable C le maximum entre A et B. On écrira deux versions : une en utilisant les noms des variables (compléter le fichier `mips32_max_a_b_var.s`) et une en utilisant uniquement les calculs d'adresses en les repérant dans le simulateur (compléter `mips32_max_a_b.s`).

6 Procédures Simples

```
int a, b, c;
int max(int x, int y)
{
    if (x > y) { return x; }
    else { return y; }
}
```

```
int main()
{
    a = 10; b = 15;
    c = max(a, b);
    return 0;
}
```

6.1 Traduire ce programme C en assembleur MIPS32 en utilisant les registres (compléter le fichier `mips32_fmax_reg.s`) puis en utilisant la mémoire (`mips32_fmax_stack.s`).

7 Boucles

```
for (i = 0; i < 8; ++i)
    s += X[i] + Y[i];
```

```
for (i = 0; i < 16; ++i)
    if (X[i] > 0) { s += X[i]; }
```

```
for (i = 1; i < 7; ++i)
    Y[i-1] += X[i] + X[i-1];
```

`X` et `Y` sont des tableaux d'entiers (de 32 bits). On utilisera le chargement et l'écriture mémoire à partir des adresses.

7.1 Écrire le code des trois boucles en assembleur MIPS32 sans utiliser les noms de variable pour la première boucle (compléter les fichiers `mips32_loop_*.s`).

7.2 Écrire le programme MIPS32 puis ARM qui écrit dans `min` la valeur minimale et dans `max` la valeur maximale du tableau `X` (en utilisant les noms de variables) (compléter `mips32_loop_min_max.s`).

8 Procédures Imbriquées

Soit le programme C suivant, qui trie (ordre croissant) un tableau de `N` octects signés :

```
char v[10];

void change(char v[], int i, int j)
{
    int tmp;
    tmp = v[i];
    v[i] = v[j];
    v[j] = tmp;
}

void tri(char v[], int n)
{
    for (int i = n - 1; i > 0; --i);
    {
        for (int j = i - 1; j >= 0; --j);
        {
            if (v[j] > v[i]) { change(v, i, j); }
        }
    }
}
```

```
int main()
{
    v[0] = 26;
    v[1] = 25;
    v[2] = 24;
    v[3] = 23;
    v[4] = 22;
    v[5] = 21;
    v[6] = 20;
    v[7] = 19;
    v[8] = 18;
    v[9] = 17;

    tri(v, 10);

    return 0;
}
```

8.1 Traduire ce programme en assembleur MIPS32 (compléter le fichier `mips32_tri.s`).