

Rapport de TP2

Opérations arithmétiques/logiques – Mixage
d'ASM/C

Architecture des ordinateurs

Groupe A – Mercredi 16h30/19h45

17/02/2021

RIGHI Racim

• Exercice 1 :

1. Représentation numérique des données

Op1	Op2	Type d'opération	Résultat	N	Z	C	V
0x08000000	0x07000000	+	0xF0000000	0	0	0	1
		-	0x10000000	0	0	1	0
0x40000000	0x40000000	+	0x80000000	1	0	0	1
0x40000000	0x80000000	-	0xC0000000	1	0	0	1
0x00F00000	0xFFFFFFFF	+	0xFFFFFFFF	0	0	1	0
0x7F000000	0x0F000000	+	0x8E000000	1	0	0	1
		-	0x70000000	0	0	1	0
0x0F000000	0x7F000000	+	0x8E000000	1	0	0	1
		-	0x90000000	1	0	0	0

2. Addition d'entiers longs sur 64 bits

Pour faire la somme d'entiers 64 bits, il suffit de faire la somme normale des bits de poids faible, ensuite la somme avec retenue des bits de poids fort.

```

8 .text
9 .global main
10 .type main, %function
11
12 main:
13     /*A COMPLETER*/
14     /* Const1 64 bits 1 */
15     LDR R0, =const1_64bits_p1
16     LDR R0,[R0]
17     LDR R1, =const1_64bits_p2
18     LDR R1,[R1]
19     /* Const2 64 bits */
20     LDR R2, =const2_64bits_p1
21     LDR R2,[R2]
22     LDR R3, =const2_64bits_p2
23     LDR R3,[R3]
24     /* Addition poids faible normal
25        et poids fort avec retenue "carry" */
26     ADD R4,R0,R2
27     ADC R1,R1,R3
28
29
30 stop:  B stop
31       BX LR
32 .end

```

3. Masquage

Instruction	R0	R1	R2	Justification
MOV R0, #0x3A	0x3A	?	?	/
MOV R1, #0x0F	0x3A	0x0F	?	/
MOV R2, #0x10	0x3A	0x0F	0x10	/
AND R1, R0, R1	0x3A	0x0A	0x10	0011 1010 AND 0000 1111 = 0000 1010

AND R0, R0, R2	0x10	0x0A	0x10	0011 1010 AND 0001 0000 = 0001 0000
ORR R0, R1, R0	0x1A	0x0A	0x10	0000 1010 AND 0001 0000 = 0001 1010
BIC R0, R0, R2	0x0A	0x0A	0x10	Les bits à 1 de R2 seront mis à 0 dans R0 0001 1010 BIC 0001 0000

	R0	R1	R2	R3	R4	Justification
LDR R0, =VAL	@val	?	?	?	?	/
LDR R0, [R0]	0x87654321	?	?	?	?	/
MOV R1, #0xFF	0x87654321	0xFF	?	?	?	/
LSL R2, R1, #4	0x87654321	0xFF	0xFF0	?	?	Décalage de R1 de 4 bits vers la gauche
ASR R3, R2, #2	0x87654321	0xFF	0xFF0	0x3FC	?	Décalage de R2 de 2 bits vers la droite
ASR R4, R0, #2	0x87654321	0xFF	0xFF0	0x3FC	0xE1D950C8	1000 0111 0110 0101 0100 0011 0010 0001 Devient 1110 0001 1101 1001 0101 0000 1100 1000
LSR R4, R1, #1	0x87654321	0xFF	0xFF0	0x3FC	0x7f	Décalage de R4 de 1 bit vers la droite
EOR R2, R4, R2	0x87654321	0xFF	0xF8F	0x3FC	0x7f	Ou exclusif entre R2 et R4
BIC R4, R4, R1	0x87654321	0xFF	0xF8F	0x3FC	0x0	Les bits à 1 de R1 sont mis à 0 dans R4

• Exercice 2 :

1. Suite de Fibonacci C

On effectue le calcul de la suite de Fibonacci, on commençant avec 2 entiers de départ x et y, et en mettant les résultats de la suite dans un tableau de 10 éléments, on s'arrête par conséquent au calcul de U9.

```

11 int main (void){
12
13     // Perform call to fibo function
14     int res = fibo(0,1);
15
16     //Infinite loop
17     while(1){
18     }
19
20     return 0;
21 }
22
23 int fibo(int x, int y){
24     //A completer
25     fibo_suite[0] = x;
26     fibo_suite[1] = y;
27     int tmp;
28     for (int i = 2; i < TAB_SIZE ; i++)
29     {
30         tmp = x+y;
31         x = y;
32         y = tmp;
33         fibo_suite[i] = y;
34     }
35     return y;
36 }

```

Code source de la fonction de calcul en C de Fibonacci

On remarque dans le registre R2 le résultat de la suite en commençant par 0 et 1, qui est 34.

Modules 1010 0101 Registers			
Name	Value	Description	
1010 0101 General Register		General Purpose	
1010 0101 r0	0x22 (Hex)		
1010 0101 r1	0x2000001c (Hex)		
1010 0101 r2	34 (Decimal)		
1010 0101 r3	34 (Decimal)		
1010 0101 r4	0x20000044 (Hex)		

Contenu des registres à la fin de l'exécution

Et le contenu du tableau dans la zone mémoire qu'on trouve en récupérant l'adresse du registre R1, on remarque bien les 10 valeurs de 0 à 34.

Monitors				
537001968		0x2000001c <Hex>	0x2000001c <Traditional>	0x2000001c : 0x2000001C <Sig
0x2000001c				
Address	0 - 3	4 - 7	8 - B	C - F
20000010	0	0	0	0
20000020	1	1	2	3
20000030	5	8	13	21
20000040	34	-754593784	-1089982717	-536337400
20000050	-536468479	-536599550	-536730620	455940518

Contenu du tableau dans la mémoire

2. Mixer assembleur et C

Pour pouvoir utiliser les variables x et y en assembleur, on les déclare comme variables globales

```

40 void fibo_asm(){
41     //Adresse du tableau
42     asm("LDR r0, =fibo_suite");
43     // Récupération de x et y et initialisation du compteur (debut:2*4 octets)
44     asm("MOV r1,#8\n\t"
45         "LDR r2, =x\n\t"
46         "LDR r3, =y\n\t"
47         "LDR r2, [r2]\n\t"
48         "LDR r3, [r3]\n\t"
49     );
50     // Stockage des 2 premières valeurs du tableau
51     asm("STR r2,[r0]\n\t"
52         "STR r3,[r0,#4]\n\t");
53     // Effectuer la somme de fibonacci
54     asm("loop:\n\t"
55         "ADD r4,r3,r2\n\t"
56         "MOV r2,r3\n\t"
57         "MOV r3,r4\n\t"
58         "STR r4,[r0,r1]\n\t"
59         "ADD r1,r1,#4\n\t"
60         "CMP r1,#40\n\t"
61         "BNE loop\n\t"
62     );
63 }

```

Code source de la solution assembleur de fibonacci

Remarque : On a aussi la possibilité de faire la boucle en C pour faciliter la solution, de cette manière on aurait qu'à récupérer la valeur de i à chaque itération et éviter tout les branchements conditionnels en assembleur.

• Exercice 3 :

1. En C

La fonction de calcul est relativement simple, on parcourt la chaîne de caractères et on incrémente un compteur à chaque 'e', on peut utiliser la fonction `strlen` de `string.h` pour récupérer la taille de la chaîne. A la fin on peut voir le résultat en récupérant l'adresse de la variable 'nb_e_c' et mettre son contenu dans un registre.

```
11
12 int main (void){
13     taille = strlen(string1);
14     //Init Tab Content randomly
15     nb_e_c = count_e(string1);
16     // int success = count_e_asm(string1);
17     //Infinite loop
18     while(1){
19     }
20
21     return 0;
22 }
23
24 int count_e(char* tab){
25     /*A COMPLETER */
26     char ch;
27     for(int i=0 ; i < taille ; i++)
28     {
29         ch = string1[i];
30         if(string1[i] == 'e')
31             nb_e_c = nb_e_c + 1;
32     }
33     asm("LDR r0,=nb_e_c");
34     asm("LDR r0,[r0]");
35 }
```

Name	Value	Description
General Register		
r0	6	
r1	536870948	
r2	34	
r3	34	
r4	536870984	
r5	0	
r6	0	
r7	537001952	
r8	0	
r9	0	
r10	0	
r11	0	

No details to display for the current selection.

Fonction de calcul du nombre de 'e' et résultats

2. En assembleur

On peut voir sur la capture d'écran suivante le contenu des registres à une étape intermédiaire, où le compteur est au caractère 29 qui a pour code ascii 116, donc 't' dans « Universitéeee! ». Le nombre de 'e' actuel étant de 2


```

37 int count_e_asm(char* tab){
38     // Compteur de 'e'
39     asm("LDR r3,=nb_e_c\n\t"
40         "LDR r3, [r3]\n\t"
41     );
42     // Taille de la chaîne
43     asm("LDR r4,=taille\n\t"
44         "LDR r4, [r4]\n\t"
45     );
46     // Adresse du tableau et compteur de parcours
47     asm("LDR r0, =string1\n\t"
48         "MOV r2, #0\n\t");
49     // Boucle de parcours et vérification si caractère == 'e' == 101 (ascii)
50     asm(
51         "loop:\n\t"
52         "LDRB r1, [r0,r2]\n\t"
53         "CMP r1, #101\n\t"
54         "BNE check_cpt\n\t"
55         // si caractère égale on incrémente le compteur de e
56         "ADD r3,r3,#1\n\t"
57         "check_cpt:\n\t"
58         // Vérification si fin de chaîne
59         "ADD r2,r2,#1\n\t"
60         "CMP r2,r4\n\t"
61         "BNE loop\n\t"
62     );
63     asm("LDR r5, =nb_e_c\n\t"
64         "STR r3, [r5]\n\t"
65     );
66
67
68     return 0;
69 }

```

Code source de la solution assembleur

- Conclusion

Dépôt github de toutes les solutions :

<https://github.com/RacimRgh/TP-Archi-STM32>