

Rapport de TP1

Architecture des ordinateurs

Groupe A – Mercredi 16h30/19h45

03/02/2021

RIGHI Racim

Exercice 1: prise en main de l'IDE

1. Création et configuration du projet

Le but de cette partie étant de jeter un premier coup d'œil à l'environnement de développement stm32cubeide, ce qui nous permet de créer un premier projet, et voir les étapes nécessaires, la configuration ainsi que la structure générale de l'arborescence des fichiers.

- **Que contient le fichier `startup_stm32f446retx.s` ?**

Ce fichier contient des instructions d'initialisation des adresses.

- **Que permet t'il?**

C'est le code qui est exécuté au premier démarrage du microcontrôleur, il initialise les piles, le tas, l'adresse d'entrée d'interruption et lance le programme main avec un branchement.

- **Ligne 126, une table des vecteurs est créée en mémoire. Que contient-elle ?**

Elle contient toutes les adresses des exceptions, y compris celles des interruptions

- **Trouver la section `Reset_Handler` (ligne 55). Que réalise-t-elle selon vous ?**

C'est le code exécuté à chaque fois que le processeur reçoit un événement de réinitialisation.

2. Les modes d'adressages

a. Analyse du fichier

- **Analyse du fichier `TP1_ex1_student.s`**

`.text`: déclare une zone de code (directive)

`.global main`: export du label main pour qu'il soit visible aux autres fichiers

`.type main, %function`: indication que main est une fonction

Main: @ de début (point d'entrée)

End: Label qui indique la fin de la fonction main et du programme

- **Que représente main: et stop:?**

Main et stop représentent respectivement le début et la fin du programme.

b. Un premier programme

```
.text
.global main
.type main, %function
```

main:

```
/* Initialisation des registres et addition */
mov r0, #1
mov r2, #3
mov r3, #5
add r4, r0, r2
add r4, r4, r3
```

c. Débogage

- **Quelle est l'extension du fichier binaire exécutable qui sera téléchargée sur la carte ?**

“.elf” Qui veut dire “Executable and Linkable Format”

Pour déboguer le programme, il suffit d'ajouter des points d'arrêt, ou “breakpoints” en faisant un double clic à la gauche des instructions, ceci nous permet de voir l'état des registres, ainsi que la mémoire en fournissant l'adresse mémoire recherchée dans l'onglet ‘memory’ du débogage.

- **Combien de registres génériques dispose le processeur ?**

13 registres

- **Que représente le registre PC ?**

PC: sert à sauvegarder l'adresse des instructions du programme, pas à pas lors de l'exécution.

- **Que contient-il ?**

Contient l'adresse de la prochaine instruction à exécuter

- **Pourquoi sa valeur est 0x80000204 (ou une valeur proche) ?**

Car c'est la première adresse disponible pour les programmes utilisateurs

- **Dans quelle zone mémoire se situe ce code ?**

- **Que contient le registre LR et à quoi sert-il ?**

Contient l'adresse de l'instruction courante (adresse de retour), contrairement à PC qui contient l'adresse de la prochaine instruction. Il sert lors des branchements pour pouvoir revenir là où en était le programme

- **Que contient le registre SP et à quoi sert-il ?**

Contient l'adresse courante du sommet de la pile.

Q : Que contient le registre xPSR et à quoi sert-il ?

PSR: Contient les flags indiquant la nature des résultats issus de l'ALU, conditionnant l'exécution du programme (appelé xPSR sur stm32cubeide)

- **Avant d'exécuter la première instruction, relever la valeur du registre PC. Faites du pas à pas et relever les différentes valeurs du PC. Que pouvez-vous conclure ?**

Que la valeur change (+2) après chaque instruction, commençant par <main> à 0x8000204 et se termine par <stop> à 0x800020

On peut conclure que la taille des instructions est de 2 bits.

A la fin de l'exécution, relever la valeur de R4. Est-ce cohérent ?

R4= 9, oui c'est cohérent car $3+1+5=9$, et on a stocké ce résultat dans R4.

d. Modification du programme – version 2 – stockage mémoire

Pour sauvegarder une valeur dans une variable, il suffit de charger son adresse avec "ldr", la mettre dans un registre, et ensuite avec "str" et l'adressage indirecte, on sauvegarde dans cette adresse mémoire sauvegardée

```
.data
SUM: .2byte 0

.text
.global main
.type main, %function

main:
    /* Initialisation des registres et addition */
    mov r0, #1
    mov r2, #3
    mov r3, #5
    add r4, r0, r2
    add r4, r4, r3
    /* Récupération de l'adresse de SUM, et stockage du résultat*/
    ldr r1, =SUM
    str r4, [r1]
end:
    STR R4, [R5]

stop:    B stop
        BX LR

.end
```

Code source de la version 2

e. Modification du programme – version 3 – adressage indirect

```
.data
SUM: .2byte 0
RES: .2byte 0
TAB: .4byte 1, 12, 28, 4, 3

.text
.global main
.type main, %function

main:
/* -----
-----
Récupérer l'adresse de TAB dans R1 */
ldr r0, =TAB
ldr r5, =RES
/* Initialiser les registres à 0 pour faire la somme
r1 contient l'indice courant, et r4 le cumul de la somme */
mov r1, #0
mov r4, #0
/* Effectuer la somme */
loop:
/*On compare le compteur avec 20, car on a 5 valeurs de 4 bytes
Si ==20 on arrête le parcours et on fait un branchement vers end, sinon on fait la somme
*/
cmp r1, #20
beq end
ldr r3, [r0, r1]
add r4, r4, r3
add r1, r1, #4
b loop
/* Mettre le résultat de la somme dans la variable RES */
end:
str R4, [R5]

stop: B stop
BX LR

.end
```

Code source de la version 3

Name	Value
<div> <div> <div>1010</div> <div>0101</div> </div> <div>General Registers</div> </div>	
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r0</div> </div>	536870916
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r1</div> </div>	20
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r2</div> </div>	536870964
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r3</div> </div>	3
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r4</div> </div>	48
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r5</div> </div>	536870914
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r6</div> </div>	0
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r7</div> </div>	0
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r8</div> </div>	0
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r9</div> </div>	0
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r10</div> </div>	0
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r11</div> </div>	0
<div> <div> <div>1010</div> <div>0101</div> </div> <div>r12</div> </div>	0

Contenu des registres à la fin de l'exécution (R4 étant la somme totale)

f. Modification du programme – version 4 – Branchement conditionnel

```

.data
VALEUR_SUP30: .byte 0
SUM: .2byte 0
RES: .2byte 0
TAB: .4byte 1, 12, 28, 4, 3

.text
.global main
.type main, %function

main:
/* -----
   -----
   Récupérer l'adresse de TAB dans R1 */
   ldr r0, =TAB
   ldr r5, =RES
   /* Initialiser les registres à 0 pour faire la somme
   r1 contient l'indice courant, et r4 le cumul de la somme */
   mov r1, #0
   mov r4, #0
   /* Effectuer la somme */
loop:
/*On compare le compteur avec 20, car on a 5 valeurs de 4 bytes
   Si ==20 on arrête le parcours et on fait un branchement vers end, sinon on fait la somme
   */
   cmp r1, #20
   beq sup30
   ldr r3, [r0, r1]
   add r4, r4, r3
   add r1, r1, #4
   b loop
   /* Vérification si la somme >= 30 */
sup30:
   cmp r4, #30
   /* Si <30 on change pas la variable car elle est déjà initialisée à 0*/
   bcc end

   /* Chargement et modification de la variable */
   ldr r6, =VALEUR_SUP30
   mov r7, #1
   str r7, [r6]
   /* Mettre le résultat de la somme dans la variable RES */
end:
   str R4, [R5]

stop: B stop
      BX LR
.end

```

Code source de la version 4

Exercice 2: Boucle et tri de tableau

1. Boucle

```
1 .data
2 TAB: .skip 20
3
4 .text
5 .global main
6 .type main, %function
7
8 main:
9 /* -----
10  -----
11  Récupérer l'adresse de TAB dans R1 */
12  ldr r0, =TAB
13  /* Initialiser r1 à 1: la valeur courante, et r2 à 0: l'indice courant */
14  mov r1, #1
15  mov r2, #0
16  /* Remplir le tableau */
17 loop:
18  cmp r1, #10
19  beq stop
20  str r1, [r0, r2]
21  add r1, r1, #1
22  add r2, r2, #2
23  b loop
24
25 stop: B stop
26      BX LR
27 .end
28
```

Code source de la boucle et remplissage du tableau

2. Algorithme de tri

- Pseudo code

```
I = 0
while( i < N - 1 )
{
    If( tab[i] > tab[i+1]
    {
        tmp = tab[i]
        tab[i] = tab[i+1]
        tab[i+1] = tmp
        if( i > 0 ) i = i - 1
    }
    I = i + 1
}
```

- Ajout du fichier et développement de l'algorithme

Pour pouvoir “build” correctement le projet, on indique que les fichiers contenant main utilisés précédemment ne doivent pas être compilés.


```

1.data
2N: .byte 5 /*Taille */
3tab: .byte 14, 25, 2, 16, 5 /*déclaration d'un tableau tab */
4
5.text
6.global main
7.type main, %function
8
9main:
10    LDR R0,=N /*taille*/
11    LDRB R0, [R0]
12    LDR R1,=tab
13    /* Compteur */
14    MOV R2, #0
15    SUB R0,R0,#1
16loop:
17    /*Si compteur = taille -11 alors fin du programme
18    et ce pour ne pas vérifier la dernière valeur et celle d'après qui ne fait pas partie du tableau*/
19    CMP R2,R0
20    BEQ stop
21    /* Comparer tab[compteur] et tab[compteur+1]*/
22    LDRB R3,[R1,R2]
23    ADD R2,R2,#1
24    LDRB R4,[R1,R2]
25    CMP R3,R4
26    /* Si inférieur alors continuer, sinon permuter*/
27    BLT loop
28    /* Permutation */
29    STRB R3,[R1,R2]
30    SUB R2,R2,#1
31    STRB R4,[R1,R2]
32    /* Si on est pas à l'indice 0, revenir en arrière pour traiter la valeur précédente dans le nouveau tableau, sinon continuer */
33    CMP R2,#0
34    BEQ loop
35    SUB R2,R2,#1
36    B loop
37

```

Code source de l'algorithme de tri

```

536870913 : 0x20000001 <Traditional> x + New Renderings...
0x20000001  100E0502 00000019 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000021  06000000 40604121 8171FFF6 02200160 F0E92DBD 0D468041 1F461646 37240046
0x20000041  082F02B1 03D305D0 08BF082F 01E00824 02E00624 04E00424 A6E00224 A01B2D19
0x20000061  57B1FD44 122F02B1 03D30CD0 D6BF022F 402300E9 2CF00046 31E00DFA 00464078
0x20000081  08FAB6F0 408831E0 5CF00046 31E003FA 00464068 C0FA3BF0 DE2808B2 BD2000D0
0x200000A1  0181F0E8 F0E8BD20 F8E92D81 E34CC143 0307DB68 234BC0D5 234BC060 E3AD0060

```

Affichage de la mémoire après le tri

On voit bien que les valeurs sont triées dans l'ordre (02 05 0E 10 19)hex = (2 5 14 16 25)dec

Exercice 3: Chaînes de caractères

```

.data /*mot clé de déclaration d'une zone de données */
SIZE: .byte 7 /*Taille chaîne -1 */
Hello: .asciz "Bonjour"
Hello2: .asciz "Bonjour"
SAME: .byte 0

.text
.global main
.type main, %function

main:
/* Chargement des variables */
LDR R0,=Hello
LDR R1,=Hello2
LDR R2,=SIZE
LDRB R2,[R2]
/* on initialise un compteur */
MOV R3, #0
loop:
/* Si taille = compteur alors mots égaux, on va vers égale pour changer la variable SAME*/
CMP R3,R2
BEQ egale
/* On charge une lettre de chaque mot */
LDRB R4,[R0,R3]
LDRB R5,[R1,R3]
ADD R3,R3,#1
CMP R4,R5
BEQ loop
/* Si on trouve un caractère différent, on saute vers la fin, et SAME reste à 0 */
B stop

/* Si la variable SAME a changé vers 0, on arrête (on a trouvé une lettre différente) */
egale:
LDR R6,=SAME
MOV R7,#1
STRB R7,[R6]

stop: B stop
      BX LR
.end

```

Code source de la comparaison des mots

Pour tester le programme, regarde au fur et à mesure de l'exécution le contenu des registres R4 et R5, qui vont contenir respectivement la lettre actuelle de Hello et lettre actuelle de Hello2, en arrivant au 5ème caractère (o et O) on remarque tout de suite la différence, et mon programme saute directement vers la fin. Si on remplace le 'O' par 'o' dans Hello2, on remarque que on aura bien 1 dans R7, et par conséquent 1 dans la variable SAME.

The screenshot shows a debugger interface with two main panels. The top panel displays assembly code with comments in French. The bottom panel shows the 'Monitors' window with a memory dump and register values.

Assembly Code:

```

10
11 main:
12 /* Chargement des variables */
13 LDR R0,=Hello
14 LDR R1,=Hello2
15 LDR R2,=SIZE
16 LDRB R2,[R2]
17 /* on initialise un compteur */
18 MOV R3, #0
19 loop:
20 /* Si taille = compteur alors mots égaux, on va vers égale pour changer la variable SAME*/
21 CMP R3,R2
22 BEQ egale
23 /* On charge une lettre de chaque mot */
24 LDRB R4,[R0,R3]
25 LDRB R5,[R1,R3]
26 ADD R3,R3,#1
27 CMP R4,R5
28 BEQ loop
29 /* Si on trouve un caractère différent, on saute vers la fin, et SAME reste à 0 */
30 B stop
31
32 /* Si la variable SAME a changé vers 0, on arrête (on a trouvé une lettre différente) */
33 egale:
34 LDR R6,=SAME
35 MOV R7,#1
36 STR R7,[R6]

```

Register Values:

Name	Value	Description
General Registers		General Purpos
r0	536870913	
r1	536870921	
r2	7	
r3	4	
r4	111	
r5	48	
r6	0	

Memory Dump:

Address	0 - 3	4 - 7	8 - B	C - F
20000000	07426F6E	6A6F7572	00426F6E	6A307572
20000010	00000000	00000000	00000000	00000000
20000020	00000000	00000000	00000000	00000000
20000030	02B02DE9	F0418046	0D461646	1F460024
20000040	37B1022E	00000000	00000000	00000000

Affichage de la mémoire et des registres après l'exécution, cas mots différents

Exercice 4: Pour aller plus loin

Pour réaliser cet exercice, j'ai essayé d'utiliser la syntaxe étendue des instructions assembleur en C suivante:

```

/* Extended inline assembly syntax */ __asm [volatile] (code_template
: output_operand_list          [: input_operand_list          [:
clobbered_register_list]]    );

```

Cependant je n'ai pas pu tester le programme et vérifier qu'il fonctionne.

```

11= int main (void){
12
13     //Init tab Content with random values
14     init_tab();
15     //call to somme_tab function
16     int success = somme_tab_asm();
17     //printf("\nSomme: %d", Somme);
18
19     //Infinite loop
20     while(1){
21     }
22
23     return 0;
24 }
25
26= void init_tab(){
27     for (int i=0;i<TAB_SIZE;i++){
28         tab[i]=rand()%10;
29     }
30 }
31
32= int somme_tab_asm(void){
33     for(int i = 0; i<TAB_SIZE; i++)
34     {
35         __asm ("ADD %[result], %[input1], %[input2]"
36             : [result] "=r" (Somme)
37             : [input1] "r" (Somme), [input2] "r" (tab[i])
38             );
39     }
40     __asm("MOV R0,%[input]"
41         :/* empty operands */
42         :[input] "r" (Somme));
43
44     return 0; // return 0 if success
45 }
46

```

Dépôt github pour voir l'historique des modifications (chaque commit correspond à une question)

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