TP 2 : Opérations arithmétiques/logiques – Mixage d'ASM/C

Objectifs :

- Représentation numérique
- Comprendre les opérations arithmétiques et logiques de base ainsi que le masquage
- Savoir mixer langage C et assembleur

Introduction

Exercice 1 : Représentation des données et opérations arithmétiques/logiques

1.1 Représentation numérique des données

Remplissez le tableau suivant, en précisant le résultat et le positionnement des bits du registre d'état N, Z, C, V

Op1	Op2	Type d'opération	Résultat	N	Z	C	V
0x08000000	0x07000000	+					
		-					
0x40000000	0x40000000	+					
0x40000000	0x80000000	-					
0x00F00000	0xFFFFFFFF	+					
0x7F000000	0x0F000000	+					
		-					
0x0F000000	0x7F000000	+					
		-					

Rappel sur les bits du registre d'état :

N : Négative = bit positionné à 1 si le résultat renvoyé par l'UAL est négatif.
Z : Zero = bit positionné à 1 si le résultat de l'UAL est nul.
C : Carry = bit positionné à 1 si l'opération de l'UAL a générée une retenue.
V : Overflow = bit positionné à 1 si l'opération a engendrée un débordement

-A partir du code template donné TP2_Ex1_1.s, vérifié l'exactitude de vos résultats.

1.2 Addition d'entiers longs sur 64 bits

On cherche à effectuer l'addition de deux nombres entiers de 64 bits, chacun étant contenu sur deux registres 32 bits à la fois (préalablement stockés en mémoire). L'opérande 1 sera donc placée dans deux registres de votre choix, tout comme l'opérande 2. Le résultat sera placé obtenu sera ensuite envoyé en mémoire dans la variable prévue à cet effet.

-Réaliser le code assembleur correspondant

1.3 Masquage

Une opération de masquage consiste à forcer à 0 ou à 1 certains bits d'un mot tout en conservant les autres bits intacts. Ce type de procédé est très fréquente lors de la programmation bas-niveau. Il est aussi possible de tester la valeur d'un seul bit dans les registres du processeur et de ses périphériques.

-Selon vous, quels sont les résultats des opérations suivantes :

```
MOV R0, #0x3A
MOV R1, #0x0F
MOV R2, #0x10
AND R1, R0, R1
AND R0, R0, R2
ORR R0, R1, R0
BIC  R0, R0, R2
```

-Indiquer le contenu de chaque registre destination à l'issue de chaque instruction.
-Consigner vos résultats dans votre rapport.
-Même question pour les instructions suivantes :

```
LDR R0, =VAL          (on supposera VAL constante valant 0x87654321)
LDR R0, [R0]
MOV R1, #0xFF
LSL R2, R1, #4
ASR R3, R2, #2
ASR R4, R0, #2
LSR R4, R1, #1
EOR R2, R4, R2
BIC R4, R4, R1
```

-Consigner vos résultats dans votre rapport.
-Sous STM32CubeIDE, reprenez le code TP2_ex1_3_student.s et vérifier vos conclusions.

Exercice 2 : Suite de Fibonacci

La suite de Fibonacci est une suite d'entiers dont chaque terme est la somme des deux termes précédents. Les deux premiers termes permettent de démarrer le calcul et sont initialisés à 0 et 1 respectivement.
On note donc que : $F_n = F_{n+1} + F_{n+2}$

2.1 Suite de Fibonacci C

Réaliser le code en langage C en vous basant sur le template de code existant (TP2_ex2_1.c). Consigner vos développements dans votre rapport.

2.2 Mixer assembleur et C

On souhaite remplacer la description de la fonction fibo() actuellement en C par une description en langage d'assemblage, fibo_asm().

-Faites les modifications nécessaires à votre code afin de pouvoir intégrer le code assembleur.
-Vérifier le bon fonctionnement.
-Consigner vos développements dans votre rapport.

Exercice 3 : Une histoire de caractère…

On souhaite développer un programme qui compte le nombre de caractère 'e' présent dans une chaîne de caractère.

-A partir du template de cet exercice TP2_ex3_student.c, compléter la fonction count_e() en C
-Compléter ensuite la fonction équivalente en Assembleur, asm_count_e() ;
-Consigner vos développements dans votre rapport.

Conclusion

A travers ce premier travail pratique, vous avez pu approfondir la représentation des données, le masquage binaire et la possibilité au sein d'un code C d'intégrer des fonctions assembleur.

Références

[1]STMicroelectronics, '*STM32F446xx advanced ARM –based 32-bit MCUs*', RM0390 Reference Manual revision 4 February 2018, https://www.st.com/resource/en/reference_manual/dm00135183-stm32f446xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[2] ARM, Cortex-M4 Instructions, ARM Developer Guide, https://developer.arm.com/documentation/ddi0439/b/Programmers-Model/Instruction-set-summary/Cortex-M4-instructions

[3] Texas Instruments, Cortex-M3/M4F Instruction Set, Technical User's Manual, 2010-2011 http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/CortexM_InstructionSet.pdf

Annexe – Guide de l'assembleur ARM

En plus des références 2 et 3, voici la liste des principales instructions du processeur ARM Cortex-M4 :

Instructions ARM les plus usuelles		
Type d'instruction	**Langage d'assemblage ARM**	**Description du transfert**
ADD	ADD r0, r1, r2	[r0] = [r1] + [r2]
	ADD r0, r1, #0x32	[r0] = [r1] + 0x32
	ADC r0, r1, #0x32	[r0] = [r1] + 0x32 + C
	STR r0, [r3, #2]	[[r3]+2] <- [r0] ; adressage indirect avec offset ; stock en mémoire le contenu de r0 à l'adresse contenue dans r3 incrémentée de 2
ADDS	ADD r0, r1, r2	Similaire au ADD mais avec modification des flags du registre d'état (N,Z,C,V)
Déplacement vers un registre	MOV r0, #100	[r0] <- valeur immédiate passée dans l'instruction (ici 100 en décimal) ; attention valeur immédiate faible.
	MVN r0, r1	[r0] <- not([r1])
ET logique	AND r0,r1,r2	[r0] = [r1] and [r2]
OU logique	ORR r0,r1,r2	[r0] = [r1] or [r2]
OU Exclusif (XOR)	EOR r0,r1,r2	[r0] = [r1] xor [r2]
Bit clear = masquage inverse	BIC R1, R0, #0x01	[R1] <- R0 AND not(#0x01) ;
	BIC R1, R0, R2	[R1] <- R0 AND not(R2) ;