Microprocesseurs

Programmation en assembleur pour MSP430FR5669

TP2: Arithmétique et assembleur

Professeurs: **P. Buccella**
D. Bullot

Nom des étudiants:

Date du laboratoire:

Table des matières

1. Objectifs

Les objectifs de ce travail pratique sont:

0) Utiliser de documents de base tel que le manuel CPU-X et les supports du cours
1) Utiliser l'outil de développement Code Composer Studio pour développer et vérifier un programme assembleur
2) Expérimenter avec les mécanismes de base du calcul arithmétique, plus précisément l'addition, la soustraction des nombres entiers non signés et signés sur 8 et 16 bits.
3) Exécuter des additions de nombres fractionnaires

2. Documents de référence

Les documents utiles pour ce TP sont:

1) Jeu d'instructions MSP430
   - **slau391f-CPUX.pdf** → 1.6.2 MSP430 Instructions
     http://www.ti.com/lit/ug/slau391f/slau391f.pdf

2) Famille MSP430FR58xx, MSP430FR59xx et. MSP430FR6xx. Guide de l'utilisateur
   - **slau367o.pdf** - http://www.ti.com/lit/ug/slau367o/slau367o.pdf

3) Fichier modèle de code assembleur:
   - Micro-TP2_template.asm → à télécharger sur Moodle
4) Polycopié/exercices/notes de cours, spécialement
   - 01_Introduction → Codage des nombres
   - 03_Assembleur → slides « Exemple de programme assembleur » , « Format III: Sauts »
   - Microprocesseur MSP430 - Instruction et modes d'adressages

5) Liens utiles :
   - https://fr.wikipedia.org/wiki/Registre_état
   - https://fr.wikipedia.org/wiki/Indicateur_de_signe
   - https://fr.wikipedia.org/wiki/Indicateur_de_zéro
   - https://fr.wikipedia.org/wiki/Indicateur_de_débordement
   - https://fr.wikipedia.org/wiki/Indicateur_de_retenue

6) Guide de l'utilisateur de Code Composer Studio
   - https://software-dl.ti.com/ccs/esd/documents/users_guide/index.html

3. Résultat demandé

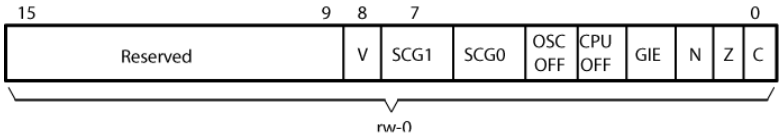
Remise à la fin du TP du projet avec le **code principal** (format compressé .zip)implémenté de la manière la plus fonctionnelle possible qui répond à toutes les questions et incluant tous les **commentaires utiles**.
Pour exporter le projet, voir :
https://software-dl.ti.com/ccs/esd/documents/ccs_sharing-projects.html

Note :1 fichier zip à **charger sur Teams** avec le rapport et les codes du TP. Le fichier est a nommer comme suit: TP2_Prénom1_Nom1_Prénom2_Nom2.zip

4. Programmation élémentaire en Assembleur

4.1. Généralités

L'ALU du processeur MSP430 exécute des opérations arithmétiques et logiques directement sur les bits de la donnée que l'on traite. Les opérations arithmétiques et logiques modifient l'état des drapeaux (flags) qui sont réunis dans le registre d'état SR (Status Register) .



SR	0x0000
V	0
SCG1	0
SCG0	0
OSCOFF	0
CPUOFF	0
GIE	0
N	0
Z	0
C	0

Le tableau ci-dessous montre la manière dont les bits d'état sont affectés suite à une opération effectuée par l'ALU.

Description des bits V, N, Z et C du registre SR

Bit	Description
V	**Overflow**. Ce bit est activé lorsque le résultat d'une opération arithmétique dépasse la *plage* autorisée selon le codage binaire en *complément* à 2. `ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B.A), ADDA` → **V=1** lorsque : positif + positif = négatif ; négatif + négatif = positif ; Autrement V=0 `SUB(.B), SUBX(.B,.A), SUBC(.B),SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA` → **V=1** lorsque : positif – négatif = négatif ; négatif – positif = positif ; Autrement V=0 Le dépassement ne se produit jamais lorsque les deux opérandes ajoutés sont de signe différent, ou lorsque le signe de deux opérandes de soustraction est le même
N	**Négatif**. Ce bit est activé (N=1) lorsque le résultat d'une opération est négatif et réinitialisé(N=0) lorsque le résultat est positif.
Z	**Zéro**. Ce bit est activé (Z=1) lorsque le résultat d'une opération est 0 et désactivé (Z=o) lorsque le résultat n'est pas 0.
C	**Carry**. Ce bit est activé (C=1) lorsque le résultat d'une opération produit un report et désactivé (C=0) lorsqu'il n'y a pas de report.

NOTE : Les manipulations de bits du registre SR doivent être effectuées par les instructions suivantes : MOV, BIS, and BIC.

Dans les prochaines étapes il s'agit de réaliser et exécuter des programmes courts avec les opérations suivantes :

- **Exécuter** les programmes en mode pas-à-pas
- **Observer** et **commenter** les valeurs prises par les registres (menu *View Register*).

4.2. Assignation de registre

Chargement d'une valeur de 8 bit (c.à.d. un *Byte*) dans le registre Rn:

MOV.B #valeur, Rn

Chargement d'une valeur de 16 bit (c.à.d. un *Word*) dans le registre Rn:

MOV.W #valeur, Rn

Dans chaque cas, observer le résultat dans le registre Rn et les bits d'état V, N, Z, C dans le registre SR.

Pour visualiser SR, sélectionner dans la barre d'état de l'outil de développement (Code Composer Studio CCS): Window → Show *View* → *Register* et dans le menu déroulant de la fenêtre Register: Core register.

Exemple :

```
MOV.W   #12, R12                ; Mettre une valeur à registre R12
```

R11	= 0x570D
R12	= 0x000C
R13	= 0x384B
R14	= 0x089D

4.3. Instructions d'addition

Les opérations arithmétiques se font de la même façon en binaire qu'en base décimale. C'est-à-dire que lorsque l'on dépasse la valeur maxi au niveau du bit *n* (lorsque l'on dépasse la valeur 1) on a une retenue ou un report au bit *n+1.*

Addition de deux nombres de 8 bits:	**ADD.B Rn, Rm**
Addition avec report de deux nombres de 8 bits (si le *carry* de l'opération précédente est 1, on le rajoute au résultat)	**ADDC.B Rn, Rm**
Addition de deux nombres de 16 bits:	**ADD.W Rn, Rm**
Addition avec report de deux nombres de 16 bits	**ADDC.WRn, Rm**

Exemple:

```
; début du programme pour le TP2
loop:
; - TP2: 6.2 Move
            mov.b  #55h, R5                 ; Move byte
            mov.w  #0xAA55, R10             ; Move word

; - TP2: 6.3 Addition
            mov.b  #11h, R6                 ; byte sans dépassement
            add.b  R5, R6                   ;
            addc.b R6, R6                   ;
            addc.b R6, R6                   ; dépassement...
            addc.b R6, R6                   ; byte avec dépassement
            nop                            ; to show result before next instr.

        ; a) word sans dépassement

        ; b) word avec dépassement
```

PC	0x004420
SP	0x002400
SR	0x0101
V	1
SCG1	0
SCG0	0
OSCOFF	0
CPUOFF	0
GIE	0
N	0
Z	0
C	1
R3	0x000000
R4	0x00FFFF
R5	0x000055
R6	0x000031

Les valeurs surlignées en jaune sont celles modifiées lors de la dernière opération.

Avec 55_h dans R5, la suite d'additions fait évoluer R6 de 11_H → 66_H → CC_H → 98_H → 31_H : Lors

- du passage de la valeur positive $+66_H$ à CC_H, on passe d'un nombre positif à un nombre négatif (N=1) et on a aussi un dépassement (V=1)
- du passage de R6 de CC_H à 98_H on reste dans les valeurs négatives (N=1) sans avoir de nouveau un dépassement (V=0), mais cette fois-ci nous avons un carry (C=1)
- du passage de R6 de 98_H à 31_H on passe d'une valeur négative de nouveau à une positive (N=0) avec un dépassement (V=1) et un carry (C=1)

Instruction	Valeur registre	V	N	Z	C
mov.b #11h, R6	11_h	0	0	0	0
add.b R5, R6	66_h	0	0	0	0
addc.b R6, R6	CC_h	**1**	**1**	0	0
addc.b R6, R6	98_h	**0**	**1**	0	**1**
addc.b R6, R6	31_h	**1**	**0**	0	**1**

Travail: Faire des opérations d'additions des words positifs :

a) ni dépassement (V) et ni Carry (C)

b) avec dépassement (V) et avec Carry (C).

Rapportez les résultats dans le tableau ci-dessous (avec indication des valeurs des registres et flags).

Instruction	Valeur registre	V	N	Z	C

4.4. Instructions de soustraction

Soustraction de deux nombres de 8 bits: **SUB.B Rn, Rm**

Soustraction avec report de deux nombres de 8 bits: **SUBC.B Rn, Rm**

Soustraction de deux nombres de 16 bits: **SUB.W Rn, Rm**

Soustraction avec report de deux nombres de 16 bits: **SUBC.W Rn, Rm**

Travail: Faire des opérations de soustractions des bytes et des words positifs sans dépassement (V) et une fois sans Carry (C), une fois avec Carry (C). Donner les résultats d'une manière similaire au tableau précédant (avec indication des valeurs des registres et flags).

Attention: ceci calcule [val(Rm) ─ val(Rn)], résultat assigné dans Rm . Veuillez aussi voir le description de l'instruction sub dans le manuel CPU-X (slau391f-CPUX.pdf)

Instruction	Valeur registre	V	N	Z	C

4.5. Instructions de comparaison

Comparaison d'un registre avec une valeur immédiate: **CMP.W #X, Rn**

Rn > X: V=0, **N=0**, Z=0, C=1

Rn = X: V=0, **N=0**, **Z=1**, C=1

Rn < X: V=0, **N=1**, Z=0, C=0

Travail: Faire des opérations de comparaison d'un registre avec des valeurs immédiates et rapportez les résultats dans le tableau ci-dessous (avec indication des flags).

Instruction	Valeur registre	V	N	Z	C

4.6. Instructions de Test

Comparaison d'un registre avec zéro: **TST.W Rn**

Rn>0: V=0, **N=0**, Z=0, C=1

Rn=0: V=0, **N=0**, **Z=1**, C=1

Rn<0: V=0, **N=1**, Z=0, C=1

Travail: Faire des opérations de test d'un registre avec différentes valeurs immédiates et rapportez les résultats dans le tableau ci-dessous (avec indication des flags).

Instruction	Valeur registre	V	N	Z	C

4.7. Instructions Logiques

Les instructions logiques permettent de faire des opérations bit-à-bit sur des nombres binaires (c'est-à-dire en considérant chacun des bits indépendamment des autres, sans se soucier de la retenue).

Les opérations par bit sont très utiles pour manipuler les bits individuels dans un mot.

Nous définissons les bits à l'aide des opérations OR, nous les effaçons à l'aide de AND et du complément, et nous les basculons à l'aide de XOR.

Exemples d'applications pour la manipulation de bits :

Mettre un/plusieurs bit à 0: :il est possible de mettre à zéro tous les bits qui ne nous intéressent pas dans un nombre. Pour ce faire, nous devons créer une valeur de « masque », un nombre dont les bits pondérés qui nous intéressent sont à 1, les autres à 0. Il faut ensuite faire un **AND** entre le nombre à masquer et le masque pou ne garder que les bits qui nous intéressent.

- **Inverser la valeur d'un/plusieurs bit** : Il est possible d'inverser tous les bits d'un nombre en faisant un **XOR** avec un nombre ne contenant que des 1. De plus, l'opération XOR d'un opérande avec lui-même change l'opérande en 0. Ceci est utilisé pour effacer un registre.

- **Mettre un bit à 1** L'instruction **BIS** peut eut être utilisée pour mettre à un un ou plusieurs bits. C'est effectivement la même chose qu'une opération OU

- **Tester la valeur d'un bit** : l'instruction **BIT** peut eut être utilisée pour tester si un ou plusieurs bits dans un nombre sont à 1, on utilise le AND bit à bit entre le masque et l'entier.

Travail:: Effectuer des opérations logiques pour :

a) mettre a zéro les 4 bits de poids faible (les 4 derniers bits) d'un nombre codé sur 8 bits . Effectuer l'opération en utilisant l'instruction AND et BIS.

b) Inverser tout les bits d'un nombre codé sur 16 bits.

c) mettre a un les 4 bits de poids fort d'un nombre codé sur 16 bits .

d) tester si le bit de position 15 de poids fort du nombre #0xFFEE, est a 1.

Rapportez les résultats dans le tableau ci-dessous(avec indication des flags).

Instruction	Valeur registre	V	N	Z	C

4.8. Additions d’opérandes en format entier signé de 16 bits

Dans cette partie on veut réaliser des d’**opérations d’addition et soustraction**, avec des opérandes **en format entier signé** à 16 bits, donnant des résultats positifs, négatifs

Travail:

a) Effectuer une opération d'addition sur 16 bits avec dépassement (V) et sans limitation

b) Effectuer une opération de soustraction sur 16 avec dépassement (V) et sans limitation

c) Limitation de l'addition avec les mêmes valeurs que dans a) [1)]

d) Limitation de la soustraction avec les mêmes valeurs que dans b) [1)]

e) Effectuer ensuite les opérations **sur 32 bits** suivantes:

65'539 + 65'540 = 131’079

3 – 65'544 = – 65’541

[1)] Pour le cas de dépassement (V) modifier les additions et soustractions pour limiter les valeurs positives à 2^{15}-1 (0x7FFF) et les valeurs négatives à –(2^{15}) (0x8000). Dans ce travail il est conseillé d'utiliser une calculatrice ou un programme tel que calc (sous Windows) pour s’aider avec les conversions décimales/binaires/hex