TD 1 : Langage assembleur et Calcul numérique

Exercices à préparer avant le TD : (à rendre au début de la séance)

Exercice 1 : Donner les définitions et équivalents en anglais des termes suivants :
Mot, demi-mot, octet :

Opérande :

Mnémonique :

Directive d'assemblage :

Compteur ordinal :

Exercice 2 : Donner pour chacune des instructions suivantes : le codage binaire, la description de l'opération et préciser les « flags » du registre APCSR potentiellement modifiés

```
add.w r1,r2,r3

adds.w r4,r5,#3

cmp.w r1,r2

ror.w r1,r2,r3

ands.w r1,r2,#4

tst.w r2,r3
```

Exercices traités en TD :

Exercice 3 : Chargement de constantes dans les registres
a) Donner un ou plusieurs équivalents assembleur des pseudo-instructions suivantes à l'aide de mov.w/mvn.w/movt/movw.

```
r2 ← 0x00000042


r4 ← 0xffffff24


r9 ← 0x0000deaf


r1 ← 0x12345678
```

b) Pour chaque solution, donner le code hexa correspondant, puis la suite d'octets telle qu'on la trouvera en mémoire.

Exercice 4 : Décodage d'instructions

Soit la suite d'octets suivante qui constitue le code d'un programme à l'adresse mémoire 0x200 :

```
4f f0 0e 00 4f f0 05 01 01 eb 00 02 a0 eb 01 03 03 fb 02 f4 04 f0 3e 05
```

Déterminer les instructions assembleur qui se cachent derrière cette suite de nombre en hexadécimal.
On commencera par identifier les instructions codées sur 16 bits et celles sur 32 bits.

Remplir le tableau suivant :

registre pc	Codage hexa	Instruction	r0	r1	r2	r3	r4	r5
(début du programme)			?	?	?	?	?	?
0x200								

Proposer un calcul équivalent avec une utilisation plus efficace des registres.

Exercice 5 : Calcul d'addition 64 bits
Réaliser un programme qui additionne les doubles mots (64 bits) contenus dans r1:r0 et r3:r2. On rangera le résultat dans r1:r0.

Exercice 6 : Calcul d'expressions
Écrire un programme en assembleur qui calcule chacune expressions suivantes, en précisant quels registres sont utilisés pour les variables apparaissant dans l'expression.

b^2-4ac (et positionner les indicateurs suivant la valeur finale de l'expression)
(m & 0xaaaaaaaa) >>1 | (m & 0x55555555) <<1
$(x-y)(x^2+xy-y^2+45)$

Exercice 7 : Écrire un programme qui prend un nombre entier entre 0 et 999 (inclus) dans r0 et écrit les codes ASCII ($30,…, $39) de son écriture décimale dans r1,r2,r3. Faire d'abord un organigramme du programme, puis réaliser le programme en assembleur.

TP 1 : Prise en main du matériel, Premiers programmes assembleur

Le but du TP est de se familiariser avec l'assembleur en réalisant des opérations simples sur les nombres binaires à l'aide des registres du processeur ARM cortex. Il est important de bien comprendre le fonctionnement du mode pas-à-pas, car il nous servira par la suite à débugger les programmes.

Documents techniques à lire et utiliser pour ce TP :
→ Micro-guide Linux
→ Environnement de travail pour le STM32

L'apéritif (le B-A, Ba)

Exercice 1 : Premier programme *(durée estimée : 2h)*
On rappelle la suite d'octets du programme inconnu de l'exercice 4 du TD.

```
4f f0 0e 00 4f f0 05 01 01 eb 00 02 a0 eb 01 03 03 fb 02 f4 04 f0 3e 05
```

On souhaite les insérer dans un programme assembleur pour les faire exécuter :

```
mystere:
     instruction1
     instruction2
     ...
```

En vous basant sur les fiches techniques :
– Créer un nouveau projet et y insérer les instructions décodées en TD.
– Compiler , charger et exécuter votre programme sur le STM32.
– Lancer une session de débuggage.
– Trouver et utiliser la commande de désassemblage pour vérifier que les instructions correspondent à celles saisies dans votre programme.
– Utiliser la commande `x` (examine) pour vérifier que les octets en mémoire correspondent bien à ceux de l'exercice ; trouver comment les afficher en hexadécimal par octets, puis par demi-mots, puis par instructions.
– Dans l'interface ddd, faire afficher les registres du STM32 et le code machine
– Exécuter le programme pas à pas et vérifier vos réponses (contenu du tableau de l'exercice 3 du TD)
– Ré-exécuter votre programme en changeant les valeur de r0 et r1 après leur initialisation à l'aide de gdb (22 et 21 au lieu de 14 et 5)

Exercice 2 : Chargement de constantes (bis)

Vérifiez sur le STM32 vos réponses à l'exercice 5 du TD.

Le plat de résistance (l'incontournable)

Exercice 3 : Les indicateurs arithmétiques (registre xPSR)
Nous souhaitons réaliser un programme qui charge des constantes dans r0 et r1 puis range dans r2 le résultat de l 'addition. Ensuite nous voulons isoler les 4 indicateurs N Z C V de APSR dans les registres r4-r7.

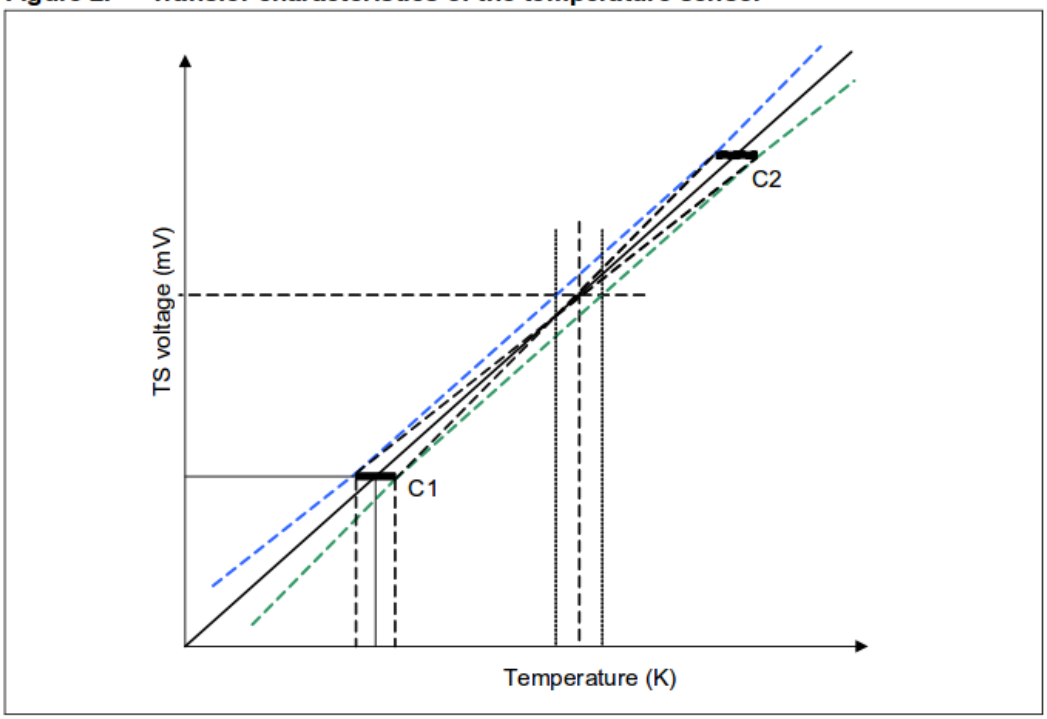
Trouver des valeurs pertinentes de r0 et r1 pour obtenir les différentes combinaisons de valeurs pour N, Z, C et V. On pourra utiliser les commandes GDB pour changer les valeurs de r0 et r1 à chaque essai plutôt que de recompiler.

La cerise sur le gâteau (le défi de la semaine !)

Exercice 4 : Calcul de température à partir d'un capteur du STM32

L'objectif du défi est de réaliser l'affichage en décimal d'un résultat de mesure de température : le STM32 est équipé d'un capteur de température. Celui-ci délivre une tension (V_{sensor}) dont la variation est proportionnelle aux écarts de température : le capteur est dit *linéaire*. On peut mesurer la tension Vsensor à l'aide d'un convertisseur analogique intégré au STM32. Ce convertisseur fournit une valeur numérique entière proportionnelle à la tension mesurée.
Lors de la fabrication en usine, on effectue un calibrage du capteur en mesurant deux valeurs du capteur à 30°C et 110°C. Les valeurs obtenues sont dans le tableau ci-dessous :

Figure 2. Transfer characteristics of the temperature sensor



The temperature can be evaluated from the digital value, ValTS, sampled by the ADC using linear approximation. It can be applied if the coordinates of two calibration points C1 and C2 are known as shown in *Figure 2*.

The current temperature can be evaluated as follows where the cold temperature coordinate pair is designated as (TC1, ValC1) and the hot temperature pair as (TC2, ValC2):

Temp = (TC2 – TC1) / (ValC2 – ValC1) * (ValTS – ValC1) + TC1

	Calibrage C1	Calibrage C2	Mesure du capteur
Valeur du convertisseur	ValC1=0x03b5	ValC2=0x04b0	donnée ValTS dans r1
Température	TC1=30°C	TC2=110°C	résultat Temp dans r0

–Écrire un programme assembleur qui calcule la temperature mesurée Temp (dans r0) en fonction de la valeur de tension mesurée ValTS (dans r1).
–Quelle est la température qui correspond à ValTS = 0x0396 ?
–Quelle est la précision de la mesure (plus petit écart mesurable en °C) ?
–Combien de chiffres après la virgule est-il pertinent d'afficher pour cette température ?
–Proposer un programme qui permet de calculer ce(s) chiffre(s) après la virgule.

Pour aller plus loin : *(s'il vous reste du temps)*

L'exercice qui suit utilise des connaissances tirées du cours 2 : boucles et branchements conditionnels.

Exercice 5 : Calcul de racines carrées par approximations successives

On souhaite écrire une routine permettant de calculer une approximation par défaut d'un entier 32bits non-signé. Pour cela on va procéder par approximations successives avec l'algorithme suivant (donné ici en python) :

```
def sqrt(x):
    # x est un entier non signé sur 32 bits
    racine=0
    bit=1<<15
    while (bit!=0):
        rb=racine+bit
        bit=bit>>1
        if (rb*rb <= x):
            racine=rb
    return racine
```

–Programmer cet algorithme en assembleur (on placera argument x et résultat dans le registre r0).

–Testez votre algorithme. en particulier vérifier qu'il répond bien pour 0,1,2,0xffffffff ...

–Proposez un algorithme similaire pour calculer la racine cubique d'un entier non-signé, puis d'un entier signé.