TP 1 : Introduction au microcontrôleur – langage d'assemblage

---

Objectifs :

-Découvrir l'environnement de développement STM32CubeIDE
-Développement d'un premier programme en langage d'assemblage
-Utilisation des fonctions de debug lors de l'exécution sur le processeur ARM
-Compréhension de l'architecture interne du microcontrôleur

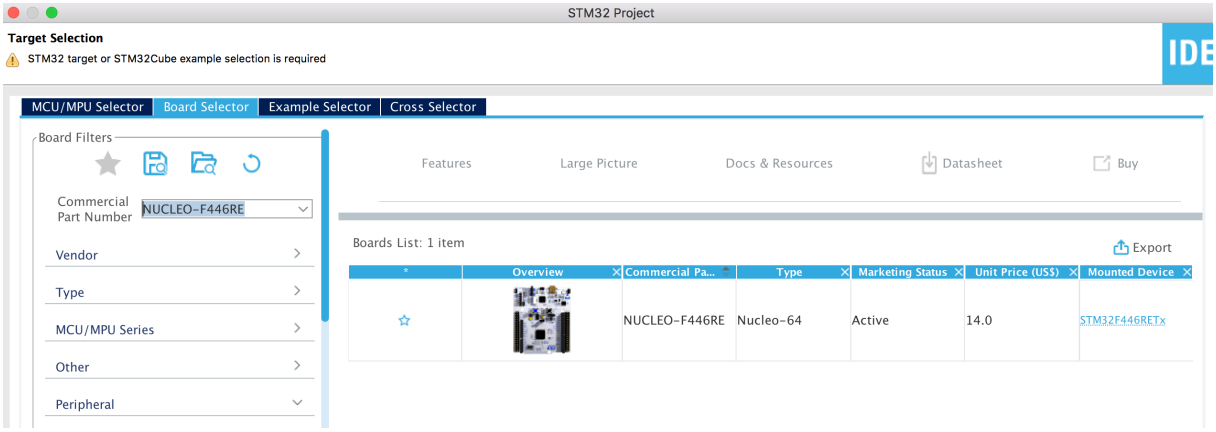
Introduction

Exercice 1 : Prise en main de l'IDE

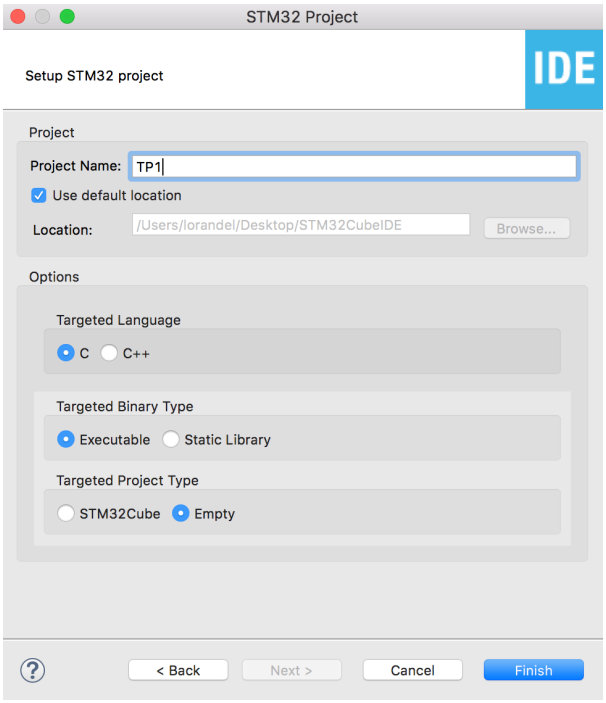
1. Création et configuration du projet STM32CubeIDE

Après avoir lancé le logiciel, sélectionner un répertoire dans votre espace local qui sera votre « workspace ». Puis, dans l'onglet File > New > STM32 Project.

Une fenêtre de sélection de la plateforme cible s'ouvre. Dans notre cas, via l'onglet Board Selector, choisissez NUCLEO-F446RE puis **NEXT.**



Dans la fenêtre de configuration du projet, donner un nom à votre projet (TP1) et saisissez les paramètres indiqués ci-dessous. Attention, choisissez bien **Empty** dans **Targeted Project Type** puis **FINISH.**



Le projet est maintenant créé, nous allons pouvoir passer à l'ajout de nos fichiers sources. Avant cela, répondez aux questions suivantes :

Question : Que contient le fichier startup_stm32f446retx.s ? Que permet-il selon vous ?
Ligne 126, une table des vecteurs est créée en mémoire. Que contient-elle ?
Trouver la section Reset_Handler (ligne 55). Que réalise-t-elle selon vous ?

2. Exercice 1 : Les modes d'adressage

Nous allons créer notre premier fichier source en assembleur. Commencer par supprimer le fichier main.c qui est ajouté par défaut.
Pour ajouter la première source, le plus simple est de cliquer-déposer le fichier **TP1_ex1_student.s** dans le répertoire *Src* dans la fenêtre Project Explorer
Le fichier est fourni sur l'ENT.

1) Analyse du fichier

A partir du code fourni, essayer de répondre aux questions suivantes et consigner les réponses dans votre rapport :
Analyser les différentes lignes de code, ainsi que tous les mots clés.
Que représente « main: » ainsi que « stop: »? Que représentent main et stop d'un point de vue du microprocesseur ?

Attention, en assembleur, l'indentation est importante et est à respecter ».

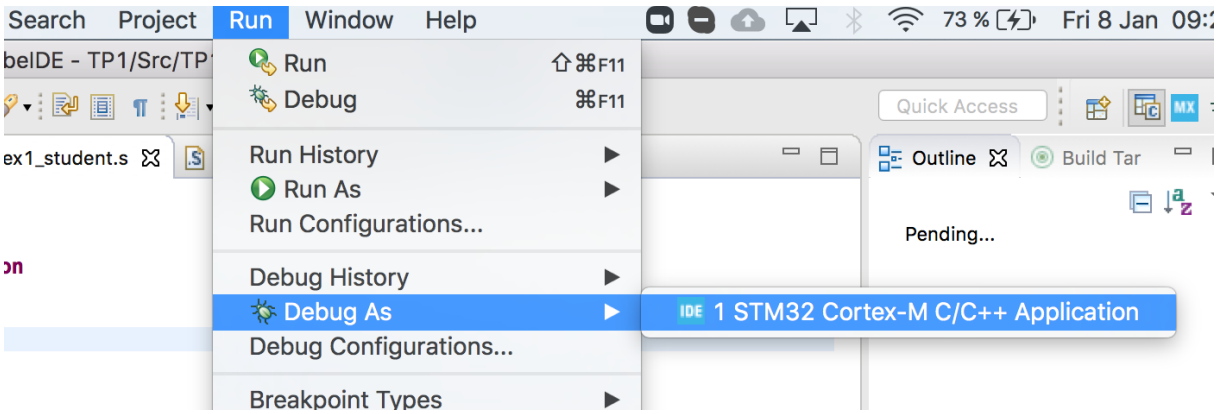
2) Un premier programme – v1

Modifier le programme afin d'initialiser le registre R0 avec la valeur 1, le registre R2 à 3 et le registre R3 avec la valeur 5. Les valeurs 1,3 et 5 seront passés en immédiat. Puis réaliser l'addition de ces 3 valeurs et stocker le résultat final dans R4.

3) Débogage

A présent, nous allons générer le fichier exécutable. Pour cela, faire un build du projet (Project > Build all). En observant la fenêtre *CDT Build Console*, vérifier qu'il n'y a pas d'erreur.
Q : Quelle est l'extension du fichier binaire exécutable qui sera téléchargée sur la carte ?

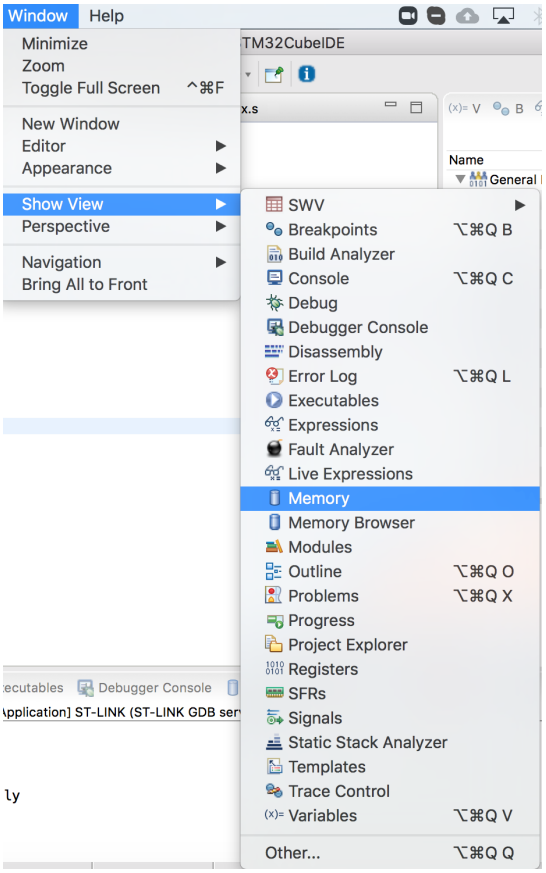
Pour valider votre code sur la carte, il faut connecter votre microcontrôleur en USB puis lancer le debugger. Pour cela, dans l'onglet *Run > Debug As > STM32 Cortex-M C/C++ Application*. Vous pouvez aussi faire un clic droit sur votre projet dans la fenêtre Project Explorer puis *Debug As > STM32 Cortex-M C/C++ Application*.



Dans ce mode, vous pouvez debugger en pas à pas votre programme assembleur. Ce serait évidemment similaire pour un code décrit en C/C++.

A l'aide de ces boutons, vous pouvez respectivement (en partant de la gauche), ajouter des breakpoints (point d'arrêt de l'exécution de votre programme), Terminer+Relancer votre programme, Lancer l'exécution, stoper l'exécution, déconnecter la plateforme, Faire un Step Into (peut permettre de rentrer dans une fonction en C par exemple), et un step Over (pour se déplacer à l'instruction suivante). Le dernier bouton est un step Out permettant de sortir d'une fonction (après un step in par exemple)

Observer les différentes vues possibles via Window > Show View > …



L'environnement de debug est très performant, vous permettant de visualiser toutes sortes d'élément comme par exemple, le contenu mémoire (Memory), la vue Disassembly, la Console, les problèmes, les variables monitorées, etc.

Sur votre droite, vous pouvez observer à droite les registres internes du microprocesseur et leurs contenus.
Q : Combien de registres génériques dispose le processeur ? Que représente le registre PC ?
Q : Que contient-il ? Pourquoi sa valeur est 0x80000204 (ou une valeur proche) ? Dans quelle zone mémoire se situe ce code ?
Q : Que contient le registre LR et à quoi sert-il ?
Q : Que contient le registre SP et à quoi sert-il ?
Q : Que contient le registre xPSR et à quoi sert-il ?

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

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

Consigner toutes vos réponses dans votre rapport. Vous pouvez quitter le mode *Debug* en cliquant sur l'icône appropriée (terminate).

4) Modification du programme – version 2 – stockage mémoire

Nous allons chercher à effectuer des déplacements de données entre la mémoire et les registres. Il s'agira donc de mettre en œuvre un des modes d'adressage disponible.

- Modifier le programme précédent afin de pouvoir stocker le résultat dans une variable SUM.

-Consigner votre code dans votre rapport ; (Le commenter !! /* comment */)

5) Modification du programme – version 3 : adressage indirect

- Modifier votre code pour initialiser un tableau tab en mémoire contenant les valeurs suivantes :

tab[5]= 1,12,28,4,3.

-Modifier votre programme pour lire ensuite les éléments du tableau tab et en faire la somme.

-Le résultat sera stocké dans une variable mémoire RES.

6) Modification du programme – version 4 : Branchement conditionnel

-Modifier le programme afin d'inclure une condition en fonction du résultat de la somme :

-Si la somme est supérieure ou égale à 30,

alors la variable mémoire VALEUR_SUP30=1, 0 sinon.

-Modifier le contenu de tab pour pouvoir tester les deux cas possibles.

-Consigner vos développements dans votre rapport

Exercice 2 : Boucle et Tri de tableau

Dans cet exercice, on souhaite appréhender la réalisation des boucles. Celles-ci vont permettre de réaliser des algorithmes plus évolués et optimiser votre code

1) Boucle

Réaliser un programme (et non une variable directement initialisée en mémoire) permettant d'initialiser le contenu d'un tableau de données, avec les valeurs allant de 1 à 9. Vous adopterez une description mettant en œuvre une boucle. En assembleur, cela se traduit par des branchements (conditionnels ou non) et des étiquettes (label).

2) Algorithme de tri

On souhaite maintenant effectuer le tri des éléments d'un tableau. On se propose d'utiliser un algorithme simple à implémenter et intuitif, basé sur l'insertion.

Il s'agit de parcourir un tableau de taille N que l'on va trier par ordre croissant. L'idée consiste à parcours les éléments et de faire la comparaison de deux éléments à chaque fois. Si l'élément de rang i+1 est inférieur à celui de rang i, on permute les deux éléments. On décrémente les indices pour vérifier que l'élément précédent, s'il y en a un, ne soit pas aussi inférieur, et ainsi

de suite. Si l'élément de rang i+1 est supérieur à l'élément i, alors on incrémente les indices. On s'arrête dès que l'on atteint la fin du tableau.

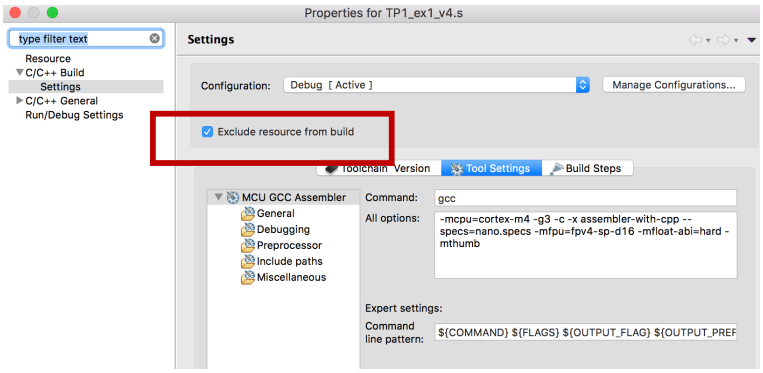
-Rédiger un pseudo-code pour cet algorithme sur votre rapport
-Récupérer le code TP1_ex2_2_student.s et ajouter le à votre projet
-Développer le code assembleur correspondant et vérifier vos développements
-Consigner tous vos développements dans votre rapport.

Exercice 3 : Chaînes de caractères

On souhaite faire un programme qui compare le contenu de 2 chaînes de caractère stockées en mémoire. Si les deux chaînes sont identiques, la variable SAME sera à 1, 0 sinon.

-Récupérer le code TP1_ex3_student.s et ajouter le à votre projet
-Développer le code assembleur correspondant et vérifier vos développements. Vous veillerez à tester les 2 cas possibles.
-Consigner tous vos développements dans votre rapport.

Note : Si vous avez plusieurs fichier .s contenant chacun un 'main', vous pouvez exclure certains fichiers du build pour éviter les soucis via : ***clic droit sur le fichier > propriétés > C/C++ Build < exclude resource from build.***



Exercice 4 : Pour aller plus loin, …

- Mixer C et assembleur

Une des fonctionnalités intéressantes est de pouvoir intégrer au sein d'un code C, du code assembleur. Pour cela, il faut respecter une syntaxe précise vue en cours.

-Récupérer le code source TP2_ex4.c
-Réaliser la fonction somme des éléments d'un tableau en ASM directement dans le code C.
-Simuler et vérifier la fonctionnalité
-Consigner tous vos développements dans votre rapport.

Conclusion

A travers ce premier travail pratique, vous avez pu découvrir l'environnement STM32CubeIDE de ST Microelectronics, ainsi qu'effectuer vos premiers développements en Assembleur afin de mieux appréhender la structure interne d'un microprocesseur, son jeu d'instruction.

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**
Accès mémoire	LDR r0, Mem	[r0] <- [Mem] ; avec Mem label d'une variable ; Attention non supportée (@32bits)
	STR r0, Mem	[Mem] <- [r0] ; Attention non supportée (@ 32bits)
	LDR r0, [r3]	[r0] <- [[r3]] ; adressage indirect ; charge r0 avec le contenu mémoire dont l'adresse est fournie par r3
	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
Déplacement entre registre	MOV r0, #100	[r0] <- valeur immédiate passée dans l'instruction (ici 100 en décimal)
	MOV r0, r1	[r0] <- [r1]
Instruction arithmétique	ADD r0, r1, r2	[r0] = [r1] + [r2]
	MUL r0, r1, r2	[r0] = [r1] * [r2]
	SUB r0, r1, r2	[r0] = [r1] - [r2]
Comparaison	CMP r0, r1	Réalise r0-r1 et affecte les bits N,Z,C, du registre de status (xPSR)
Branchement conditionnel	BGT label (BGE, BLT, BLE, BEQ, BNE,…)	Branchement au label si condition satisfaite
Branchement non conditionnel	B Label	Branchement vers le label

Annexe – Mode d'adressage

Le processeur supporte plusieurs modes d'adressage possibles en Assembleur ARM. On notera que *offset* dans les exemples illustratifs suivants peut correspondre à une valeur immédiate (#litéral), ou un registre (Rn) ou un registre avec décalage (Rn, décalage)

-Indirect

```
LDR/STR  Rd, [Rn]                ; accès à mem[Rn]
```

-pré-indexé

```
LDR/STR Rd, [Rn, offset]         ; accès à mem[Rn+offset]
```

-Pré-indexé automatique

```
LDR/STR  Rd, [Rn, offset]!       ; Rn <- Rn + offset, puis accès à mem[Rn]
```

- Post-indexé automatique

```
LDR/STR  Rd, [Rn] , offset       ; accès à mem[Rn] puis Rd <- Rn + offset
```

-Immédiat (non supporté pour LDR/STR sur Cortex-M4)