Université Mohamed BOUDIAF- M'sila
Faculté de Technologie
Département d'Electronique
Options : ESEM ; INS
Année d'Etude : M1

. Année universitaire :2019/2020
. Chargé de Module : BENTOUMI .M
. Date : 28/04/2020

# TP5 PROGRAMMATION EN ASSEMBLEUR DES PROCESSEURS DE LA FAMILLE C6000

## 1- But :

- ✓ Se familiariser avec quelques instructions du DSP TMS320C6713 de Texas instrument.
- ✓ Savoir programmer la famille des processeurs **TI C6000** en assembleur.
- ✓ Savoir exécuter un programme en assembleur.
- ✓ Appliquer des programmes simples de traitement de signal.

## 2- Aperçu sur l'architecture C6000

Le **C6000** se compose de mémoire interne, de périphériques (port série, interface de mémoire externe, etc.) et, surtout, le CPU qui a les registres et les unités fonctionnelles pour l'exécution des instructions. Il est nécessaire de comprendre comment le CPU récupère et exécute les instructions d'assemblage pour écrire un programme d'assemblage hautement optimisé.

Dans le C6000, les nombres utilisés dans les opérations sont stockés dans les registres. Parce que les registres sont directement accessibles via le chemin de données de la CPU, l'accès aux registres est beaucoup plus rapide que l'accès aux données dans la mémoire externe. Le CPU C6000 possède deux fichiers de registre (A et B). Chacun de ces fichiers est composé de seize registres 32 bits (A0-A15 pour le A et B0-B15 pour le B). Les registres à usage général peuvent être utilisés pour les données, l'adresse des données, pointeurs ou registres de conditions. La figure suivante montre un schéma de principe des processeurs **C67x**. Cette structure de base est similaire à d'autres processeurs de la famille C6000.

Les fichiers de registre à usage général prennent en charge des données dont la taille varie de 16 bits à 40 bits en virgule fixe.

En plus des fichiers de registre à usage général, la CPU a un fichier de registre séparé pour les registres de contrôle. Les registres de contrôle sont utilisés pour contrôler diverses fonctions CPU telles que le mode d'adressage, les interruptions, etc.

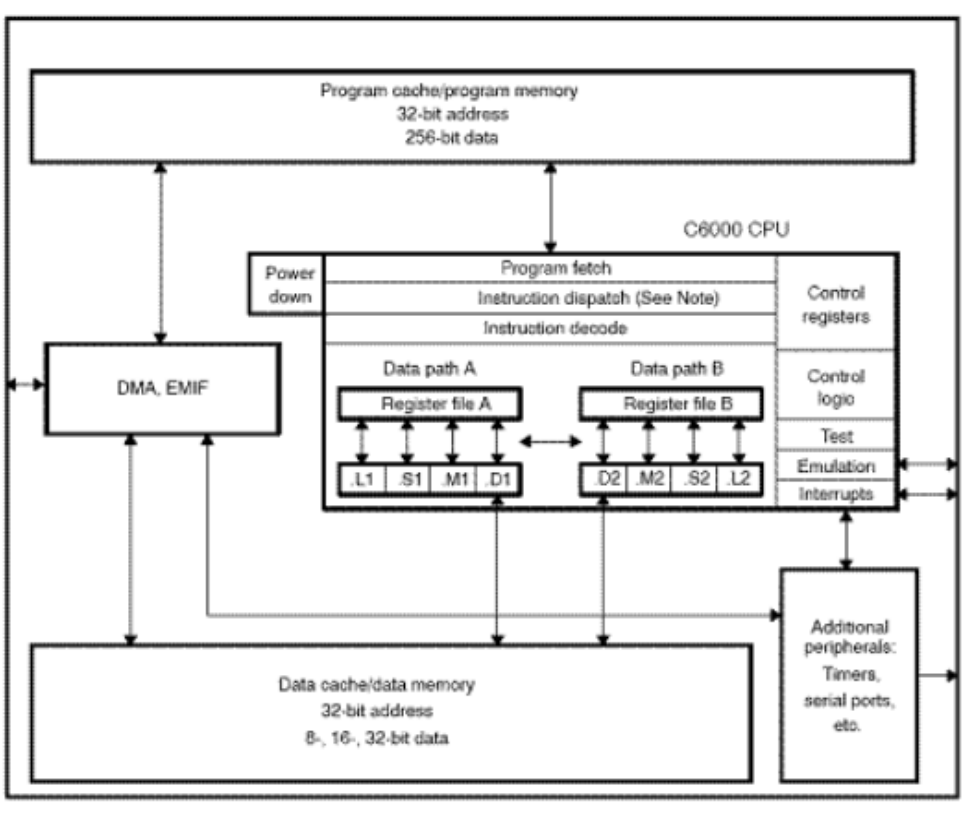



**Figure 1.** : Block diagramme du DSP TMS320C67x

**Unités fonctionnelles**

Le CPU C6000 possède plusieurs unités fonctionnelles qui effectuent les opérations actuelles. Chaque fichier de registre à 4 unités fonctionnelles nommées .M, .L, .S et .D. Les 4 unités fonctionnelles connectées au registre A sont nommées .M1, .L1, .S1 et .D1.Ceux connectés au registre B sont nommés .M2, .L2, .S2 et .D2. Par exemple, l'unité fonctionnelle.M1 effectue la multiplication sur les opérandes qui sont dans le registre A. Lorsque le CPU exécute le MPY.M1 A0, A1, A3 ci-dessus, l'unité fonctionnelle .M1 prend la valeur stockée dans A0 et A1, les multiplie ensemble et enregistre le résultat dans A3. Le .M1 en MPY .M1 A0, A1, A3 indique que cette opération est effectuée en unité .M1. L'unité .M1 a un multiplicateur de 16 bits et toutes les multiplications sont effectuées par le .M1 (ou.M2). Le schéma suivant montre l'architecture de base de la famille C6000 et des unités fonctionnelles.

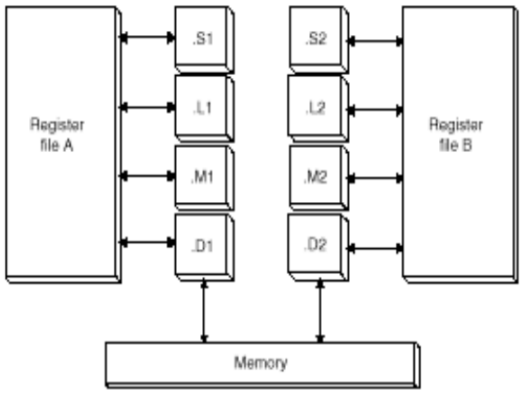



**Figure 2.** : Unités fonctionnelles du C6x

## 3. Stockage des données en mémoire

Le stockage du contenu du registre utilise les mêmes modes d'adressage. Les instructions en assembleur utilisées pour le stockage sont **: STB**, **STH**, et **STW**.

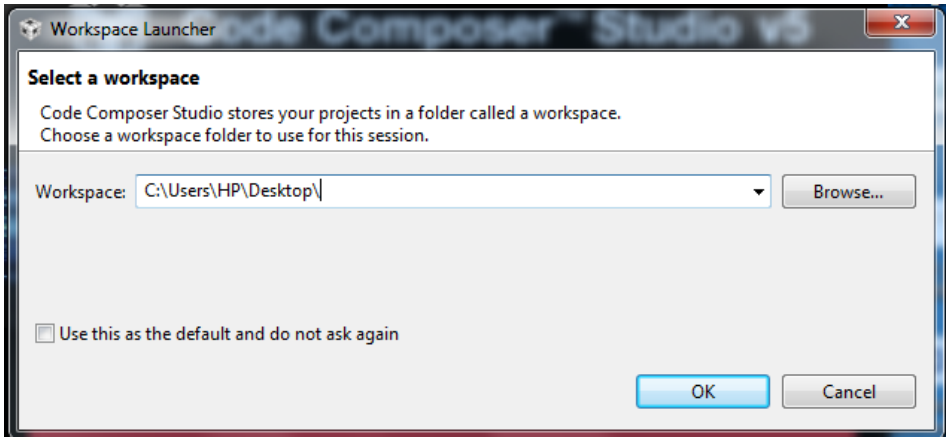
**Remarque** : Le stockage des données en mémoire dépend de l'**endian mode** du système mémoire. Avec **little endian mode** les adresses mémoire inférieures contiennent la partie LSB des données. Avec **big endian mode** les adresses mémoire inférieures contiennent la partie MSB des données.

Dans le CPU du C6x, il faut exactement un cycle d'horloge CPU pour exécuter chaque instruction. Cependant, les instructions telles que LDW doivent accéder à la mémoire externe lente et les résultats du chargement ne sont pas disponibles immédiatement à la fin de l'exécution. Ce retard des résultats d'exécution est appelé créneaux de retard ou **delay slots**.

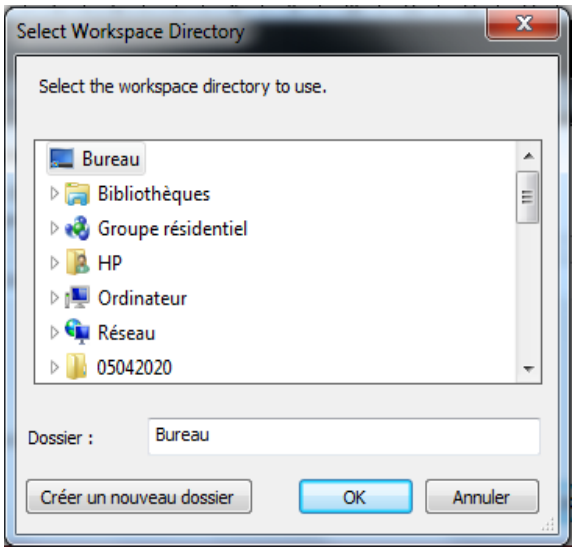
| **Description** | **Instructions** | **Delay slots** |
|---|---|---|
| Simple Cycle | Toutes les instructions sauf les suivantes | 0 |
| Multiplication | MPY, SMPY etc. | 1 |
| Chargement | LDB, LDH, LDW | 4 |
| Branchement | B | 5 |

## 4. Programmation en assembleur sous CCSV5.5 en utilisant le simulateur TMS320C67xx.

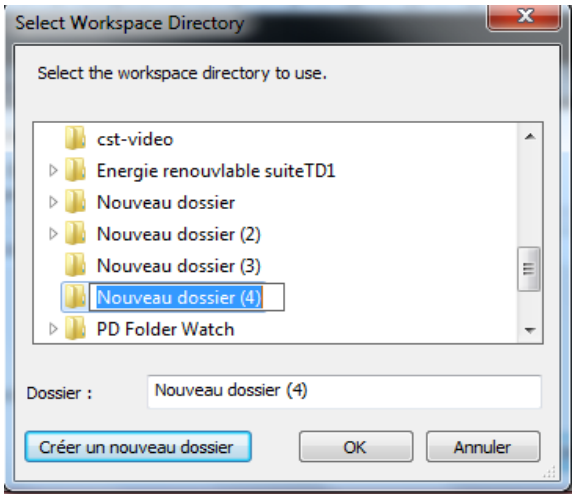
4.1 Double clic sur l'icône de **Code Composer studio 5.5.0,** la fenêtre **Workspace Launcher** apparait.



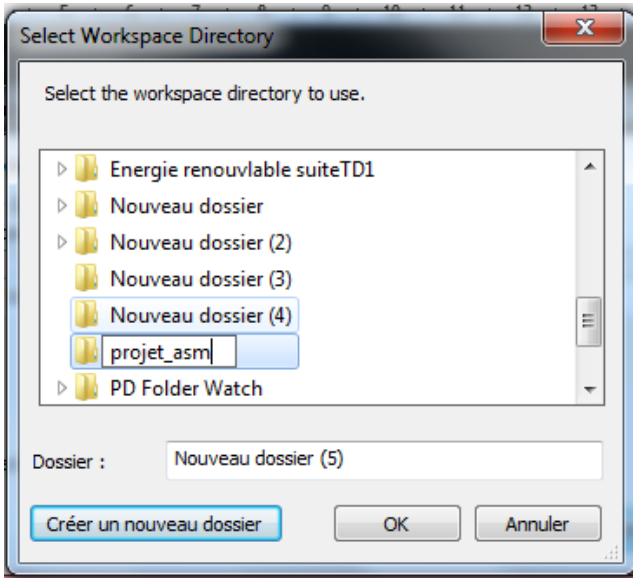

Dans cette fenêtre cliquer sur **Browser** la fenêtre **Select Workspace directory** apparait



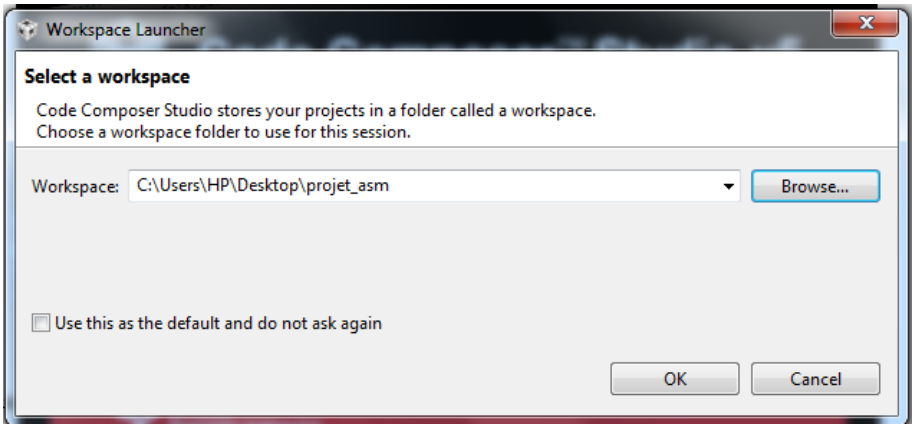

Cliquer sur **Bureau** ensuite sur **Créer un nouveau dossier**, il apparait un dossier




Donner un nom a ce dossier comme exemple **projet_asm** appuyer sur **OK**

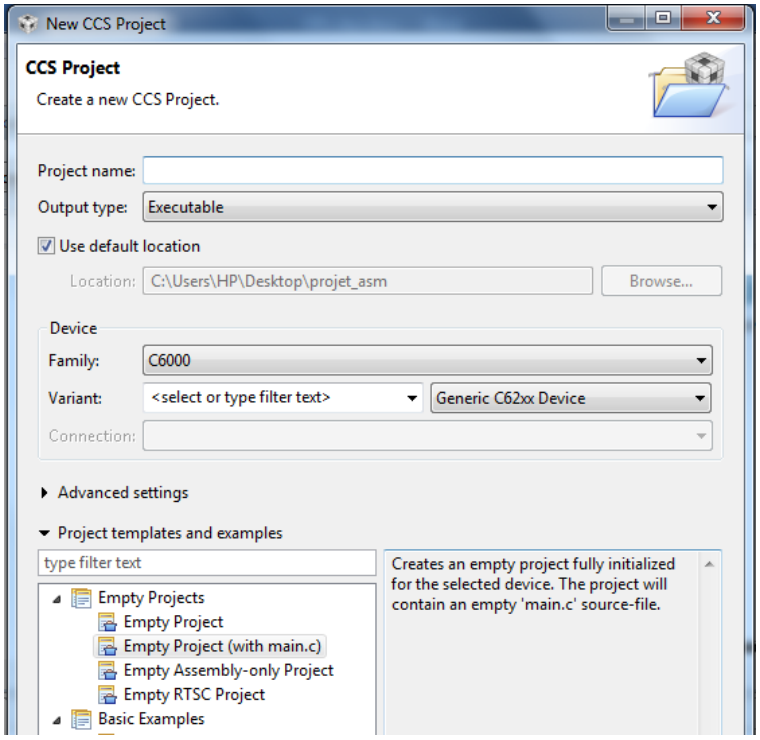



Le nom du projet s'ajoute au chemin puis cliquer sur **OK** du **Workspace Launcher.**



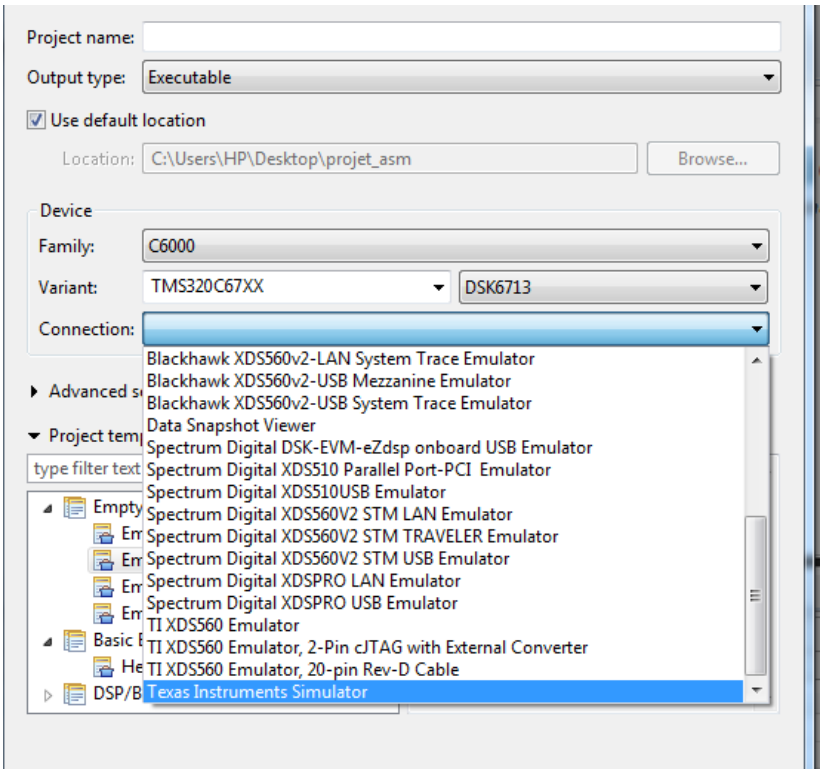

Le logiciel ouvre la fenêtre **CCS Edit- TI Resource Explorer – Code Composer Studio**.

Sur cette fenêtre aller à **Project** et cliquer sur ce dernier puis aller à New CCS project et cliquer sur cette dernière une fenêtre **New CCS Project** apparait

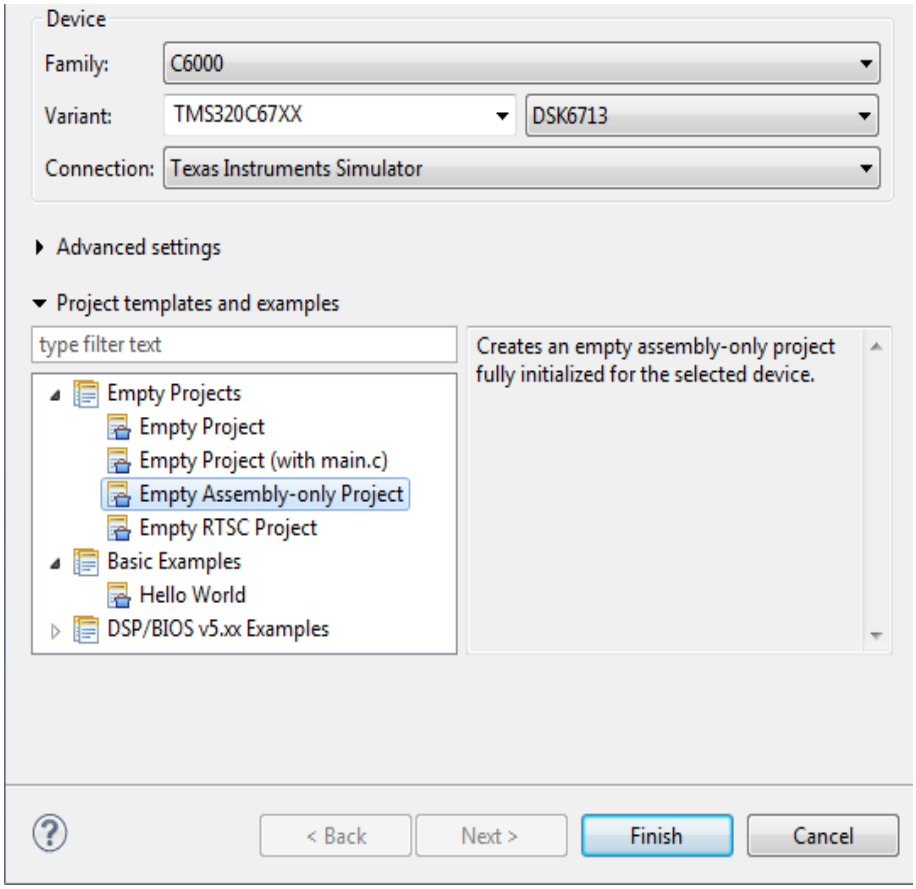



Donner un nom à votre projet comme nous avons mentionné au début. Choisir dans **Family** : **C6000**.

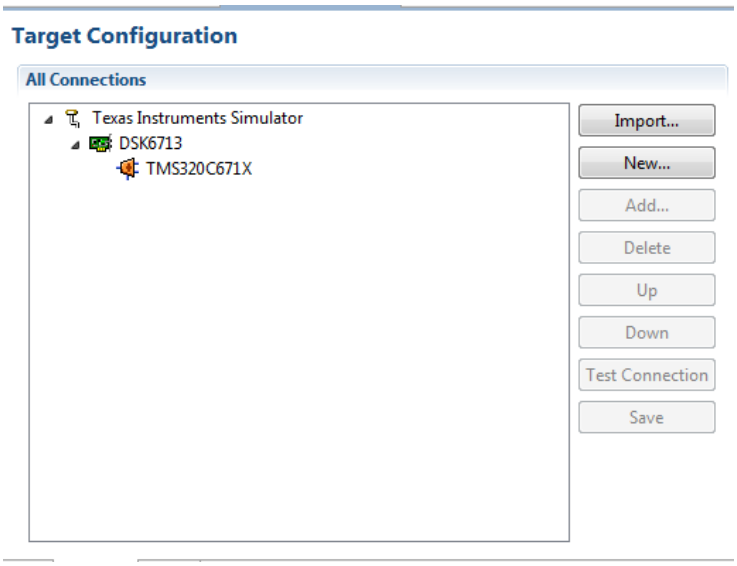
Sur **Variant** choisir : **TMS320C67XX** Puis sur **Connection** : **Texas Instruments Simulator**


Project name:
Output type:
Executable
Use default location
Location:
C:\Users\HP\Desktop\projet_asm
Browse...
Device
Family:
C6000
Variant:
TMS320C67XX
DSK6713
Connection:
Blackhawk XDS560v2-LAN System Trace Emulator
Blackhawk XDS560v2-USB Mezzanine Emulator
Blackhawk XDS560v2-USB System Trace Emulator
Data Snapshot Viewer
Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
Spectrum Digital XDS510 Parallel Port-PCI Emulator
Spectrum Digital XDS510USB Emulator
Spectrum Digital XDS560V2 STM LAN Emulator
Spectrum Digital XDS560V2 STM TRAVELER Emulator
Spectrum Digital XDS560V2 STM USB Emulator
Spectrum Digital XDSPRO LAN Emulator
Spectrum Digital XDSPRO USB Emulator
TI XDS560 Emulator
TI XDS560 Emulator, 2-Pin cJTAG with External Converter
TI XDS560 Emulator, 20-pin Rev-D Cable
Texas Instruments Simulator
type filter text

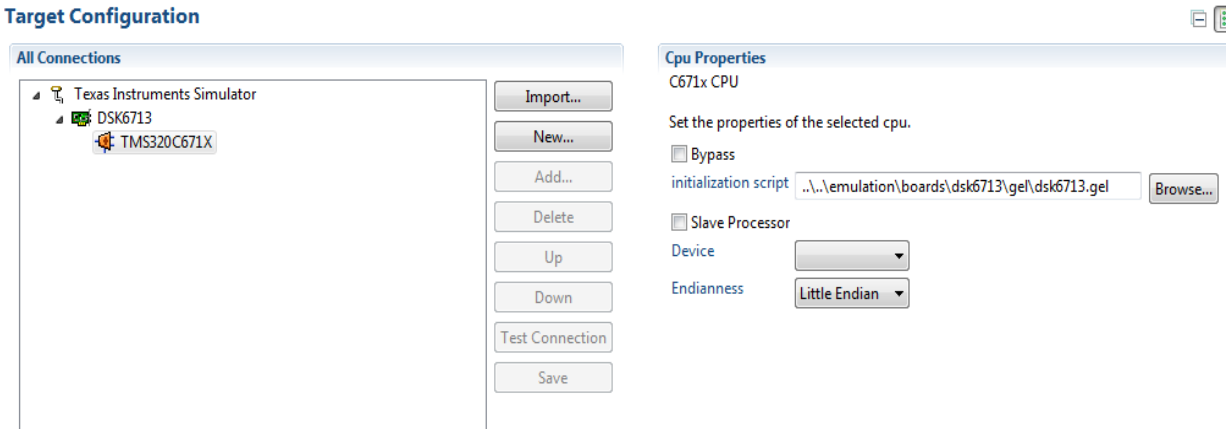
Sur **Empty projects** choisir **Empty Assembly-only Project** puis appuyer sur Finish




Sur Project Explorer appuyer sur la flèche projet_asm puis sur la flèche TargetConfigs puis aller à DSK6713.ccxml double clic sur cette dernière une fenêtre Target Configuration apparait

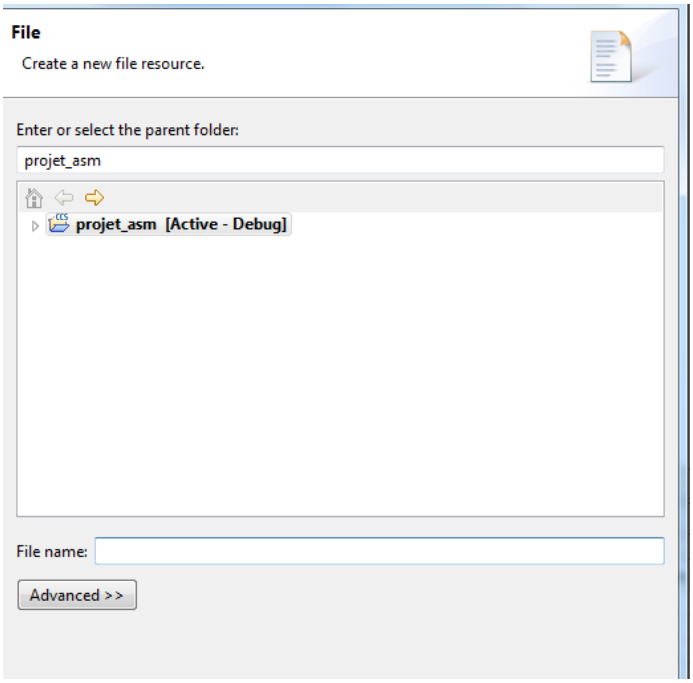


Cliquer sur le symbole de la prise (TMS320C671X) on aura



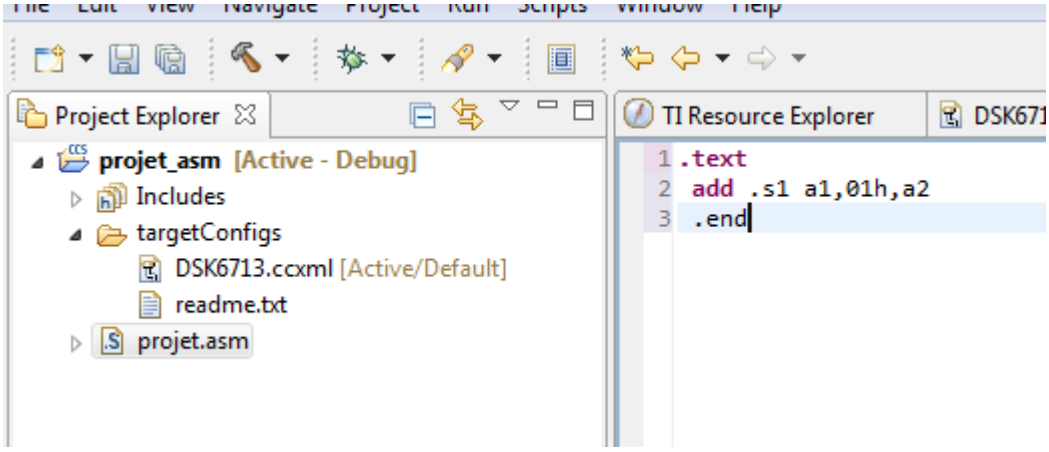

Aller à Device et choisir C67xx la fenêtre Recommended connection name apparait. Cilquer sur Yes. Ensuite appuyer sur Save.

Appuyer par le bouton à droite de la souris sur projet_asm ensuite sur New et sur File une nouvelle fenêtre apparait.




Choisir un nom avec une extension .asm dans notre cas projet.asm. Cliquer sur Finish.

Ecrire un programme en assembleur qui additionne le contenue du registre A1 avec le nombre 01h et mettre le résultat dans le registre A2.




Aller à File et cliquer sur Save.

Aller à Project puis sur Build all. S'il n'y a pas d'erreur appuyer sur Débuggeur.

Cliquer sur Registres puis sur Core Registres tous les registres vont apparaitre. Mettre une valeur dans A1.

Appuyer sur View Disassembly. Aller à la flèche step Into et exécuter le programme instruction par instruction. Remarquer que la valeur de A2 change après l'exécution de l'instruction ADD.

Les changements apparaissent en Jaune.

## Travail demandé

**1-** Supposons que les valeurs suivantes sont stockées dans des adresses mémoire :

Loc 32- bit value

100h fe54 7834h

104h 3459 f34dh

108h 2ef5 7ee4h

10ch 2345 6789h

110h ffff eeddh

114h 3456 787eh

118h 3f4d 7ab3h

On Suppose que **A10 = 0000 0108h**. Trouver le contenu de A1 et A10 après l'exécution de chacune des instructions suivantes.

1. LDW .D1 *A10, A1
2. LDH .D1 *A10, A1
3. LDB .D1 *A10, A1
4. LDW .D1 *-A10[1], A1
5. LDW .D1 *+A10[1], A1
6. LDW .D1 *+A10[2], A1

7. LDB .D1 *+A10[2], A1
8. LDW .D1 *++A10[1], A1
9. LDW .D1 *- A10[1], A1
10. LDB .D1 *++A10[1], A1
11. LDB .D1 *-A10[1], A1
12. LDW .D1 *A10++[1], A1
13. LDW .D1 *A10- [1], A1

**2-** Énumérez toutes les unités fonctionnelles que vous pouvez affecter à chacune de ces instructions :

1. ADD .?? A0,A1,A2
2. B .?? A1
3. MVKL .?? 000023feh, B0
4. LDW .?? *A10, A3
5. ADD .??? B0,A1,B2
6. MPY .??? A1,B2,A4

**3-** Par exemple, considérons le chargement du contenu de la mémoire à l'adresse pointée par A10 vers A1, puis le déplacement des données chargées vers A2.

Le code pour ce cas est :

1. LDW .D1 *A10 , A1
2. MV .D1 A1 ,A2

Le résultat de l'instruction LDW n'est pas disponible immédiatement après l'exécution de LDW. Par conséquent, l'instruction MV ne copie pas la valeur souhaitée de A1 dans A2 et le résultat est aléatoire et faut. Pour remédier à ça on rectifie le programme.

1 LDW .D1 *A10, A1
2 **NOP 4**
3 MV .D1 A1,A2

Il y a plusieurs instructions l'addition, la soustraction et la multiplication sur le CPU C6x. Les instructions de base sont ADD, SUB et MPY.

ADD et SUB ont 0 retard (ce qui signifie que les résultats de l'opération sont immédiatement disponibles), mais le MPY a 1 intervalle de retard (le résultat de la multiplication est valide après 1 cycle d'horloge supplémentaire).

Ecrire un programme en assembleur qui calcule

**(0000 ef35h + 000033dch - 0000 1234h) * 0000 0007h**