République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 8 Mai 1945 – Guelma

Faculté des Sciences et de Technologie

Département d'Electronique et

Télécommunications

Polycopie en Travaux Pratiques

"Microprocesseurs & DSP"

Présenté par :

1. DOGHMANE Hakim
2. BOUROUBA Hocine

2024/2025

Avant-propos

Ce polycopié de Travaux Pratiques "Microprocesseurs et DSP" s'adresse aux étudiants de la première année Master des parcours "Instrumentation". Il est constitué de deux grandes parties. La première partie est consacrée à la prise en main de la carte de développement MTS-86C, les techniques de programmation assembleur utilisant le microprocesseur 8086 et les différentes interfaces pour communiquer avec les divers périphériques d'entrées-sorties. Elle est composée de cinq manipulations expérimentales.

La deuxième partie de ce polycopié s'intéresse à la prise en main de la carte C6713DSK, la familiarisation avec le Code Composer Studio et l'implémentation de divers algorithmes de traitement du signal.

L'objectif principal de ce polycopié est de mettre à la disposition des étudiants un support de pédagogique basé sur des travaux pratiques afin qu'ils puissent apprendre les fondements de base de la programmation en assembleur et l'implémentation des algorithmes du traitement du signal sur une carte DSP.

Table des matières

Partie 01 : Microprocesseurs

Partie 02 : DSP

TP 01 : Présentation du Kit MTS-86C

1. Objectifs

L'objectif de ce TP est de présenter le Kit MTS-86C qui sera utilisé par la suite comme un outil de programmation en assembleur pour les séances de travaux pratiques suivantes. On procèdera par la suite à introduire la procédure d'utilisation du Kit MTS-86C en modes autonome et liaison (avec un PC). A la fin de la manipulation l'étudiant sera capable de :

- ✓ Utiliser le clavier du Kit MTS-86C pour saisir un programme en langage machine (mode autonome).
- ✓ Exécuter un programme assembleur en mode liaison.

2. Kit MTS-86C

Le kit **MTS-86C** permet de mettre en évidence les capacités de différents microprocesseurs du fabricant Intel à travers divers périphériques et accessoires reproduits sur le schéma de la Figure.1. Le kit MTS-86C est développé autour du microprocesseur 8086 d'INTEL. Il reprend dans son architecture les principales caractéristiques (matériel et logiciel) qui ont fait le succès du MTS-86C. Le kit MTS-86C est composé d´un microprocesseur 16 bits (8086 d'Intel) auquel sont connectés tous les constituants principaux de l'informatique industrielle :

- Une mémoire RAM de 64 Ko.
- Une mémoire EPROM Moniteur de 64 Ko.
- Une mémoire EPROM Utilisateur de 64 Ko.
- Une interface parallèle 8255 (3).
- Une interface série 8251 (2).
- Une interface TIMER 8253.
- Un contrôleur de clavier 8279.
- Contrôleur d'interruption 8259
- Un convertisseur analogique/numérique ADC0809.
- Un convertisseur numérique/analogique DAC0808.

Cette carte est équipée aussi de :

- Un clavier de 24 touches.
- Un afficheur LCD (16x2 lignes).
- Des diodes.
- Un afficheur 7 segments.
- Deux hauts parleurs.

- Deux connecteurs parallèles.
- Deux connecteurs séries.
- Un port d'extension.

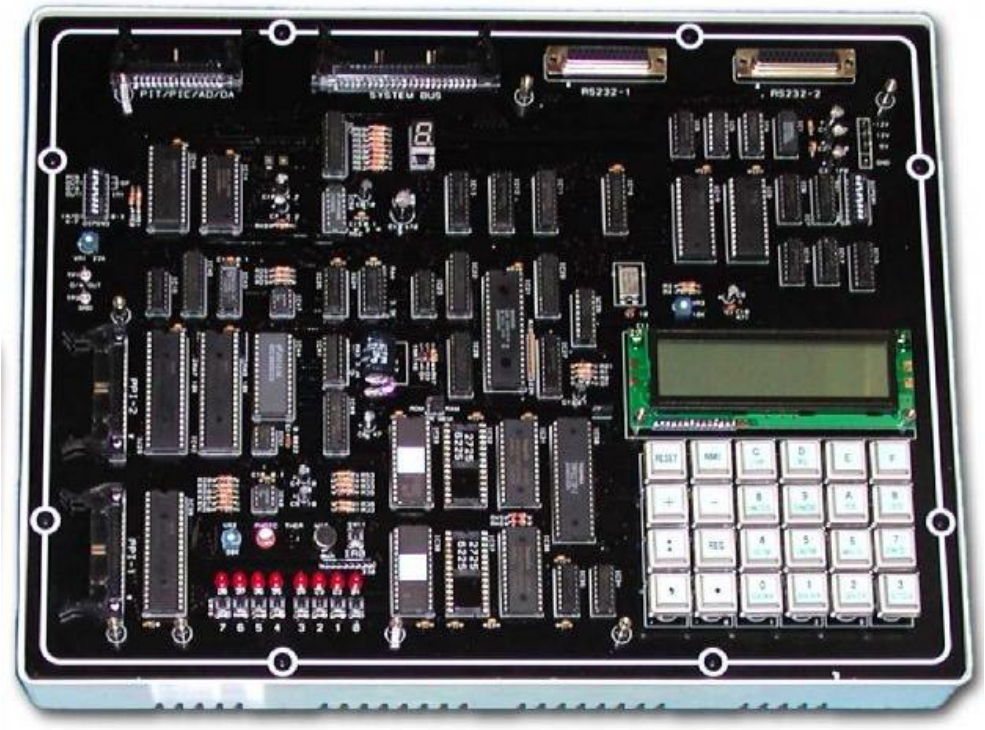


Figure.1 : Le Kit MTS-86C

3. Description du Microprocesseur 8086

Comme le Kit MTS-86C est conçu à base du microprocesseur 8086, il est préférable de faire un petit aperçu sur ce type de microprocesseur, avant toute présentation de ce Kit. Le 8086 est un circuit intégré de forme DIL (Dual In-Line : boitier de circuit intégrés) de 40 pattes comme le montre la Figure. 2. Il est équipé d'un bus de données de 16 bits et un bus d'adresses de 20 bits et fonctionne à des fréquences diverses selon plusieurs variantes : 5, 8 ou 10 MHz. Pour l'architecture interne du microprocesseur, le µ.P 8086 contient deux unités internes distinctes : l'**UE** (***Unité d'Exécution***) et l'**UIB** (***Unité d'Interfaçage avec le Bus***). Le rôle de l'UIB est de récupérer et stocker les informations à traiter, et d'établir les transmissions avec les bus du système, tandis que l'UE exécute les instructions qui lui sont transmises par l'UIB comme montrés dans la Figure.3.

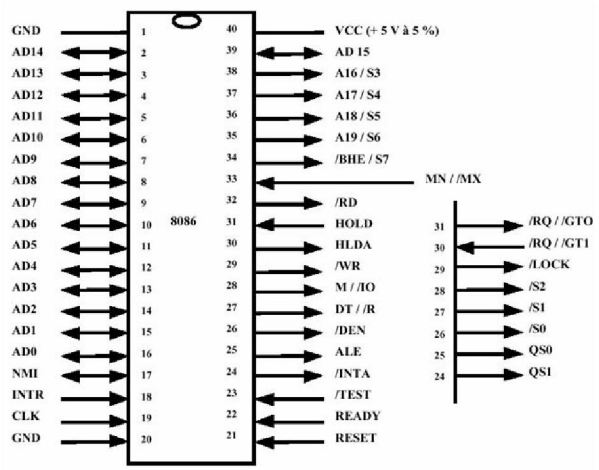
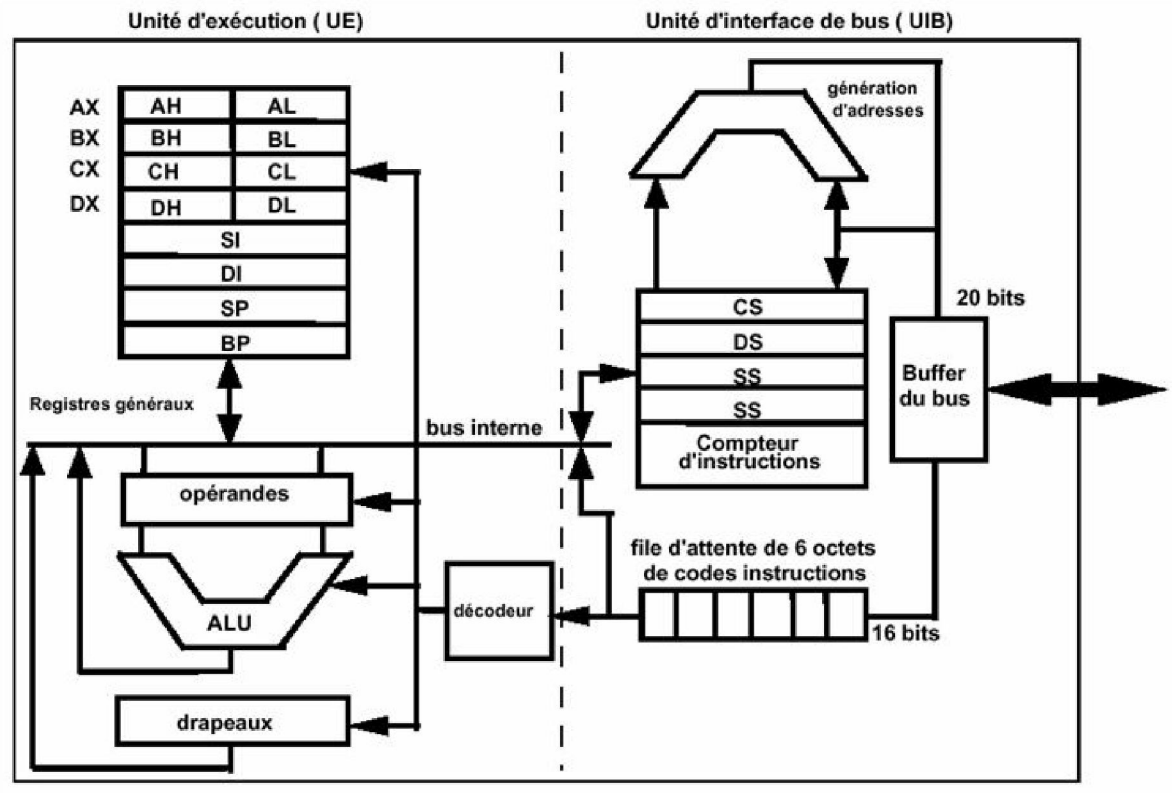


Figure.2 : Brochage du microprocesseur 8086.



3.1. Les registres du µP 8086

Dans ce paragraphe, on présente les différents registres du microprocesseur avec la description de leurs rôles.

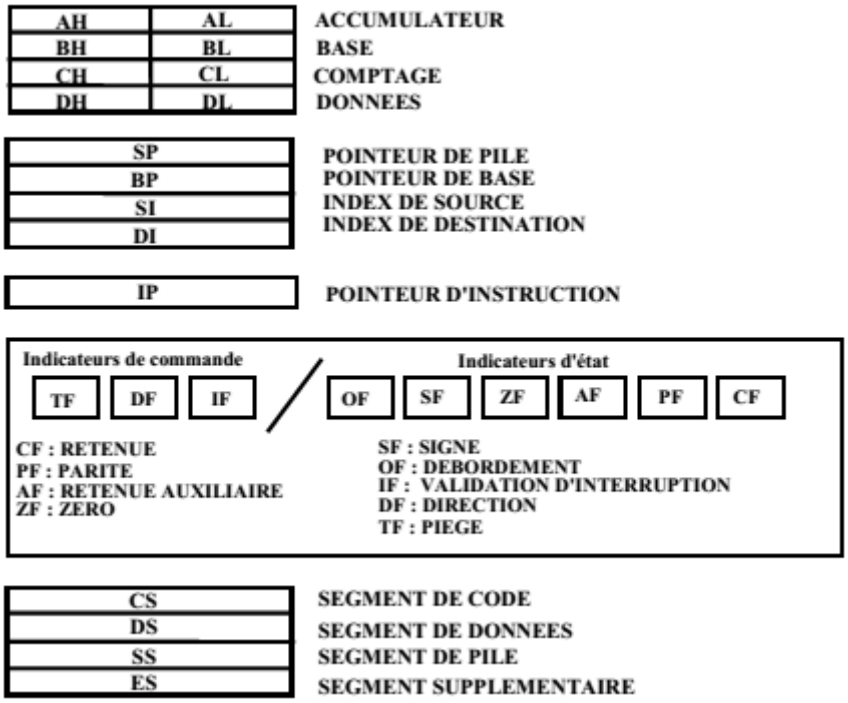


Figure.3 : Architecture interne du microprocesseur 8086.

Le microprocesseur 8086 contient 14 registres répartis en 4 groupes :

- **Registres généraux** : 4 registres sur 16 bits

  AX= (AH, AL);

  BX= (BH, BL);

  CX= (CH, CL);

  DX= (DH, DL).

- **Registres de segments** : 4 registres sur 16 bits.

  CS : Code Segment, registre de segment de code ;

  DS : Data Segment, registre de segment de données ;

  SS : Stack Segment, registre de segment de pile ;

  ES : Extra Segment, registre de segment supplémentaire pour les données

- **Registres de pointeurs et d'index** : 4 registres sur 16 bits.

  a) Pointeurs :

  SP : Stack Pointer, pointeur de pile (la pile est une zone de sauvegarde des données en cours d'exécution d'un programme) ;

  BP : Base Pointer, pointeur de base, utilisé pour adresser des données sur la pile.

  b) Index :

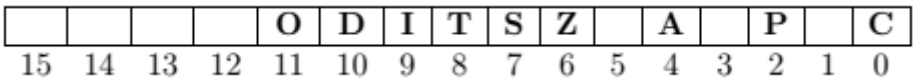
  SI : Source Index ; **DI** : Destination Index.

Ils sont utilisés pour les transferts de chaînes d'octets entre deux zones mémoire.

- **Pointeur d'instruction et indicateurs (flags) :** deux registres sur 16 bits.

IP : Pointeur d'instruction, contient l'adresse de la prochaine instruction à exécuter.

Flags :



CF : indicateur de retenue (carry) ;

PF : indicateur de parité ;

AF : indicateur de retenue auxiliaire ;

ZF : indicateur de zéro ;

SF : indicateur de signe ;

TF : indicateur d'exécution pas à pas ;

IF : indicateur d'autorisation d'interruption ;

DF : indicateur de décrémentation ;

OF : indicateur de dépassement (overflow).

3.2. Plan de l'espace mémoire

L'espace mémoire utilisé est divisé en trois parties. Une zone pour le programme moniteur et exemples d'application, la seconde pour le programme utilisateur et les vecteurs d'interruptions et la troisième, une zone extensible, elle peut être du type ROM ou RAM.

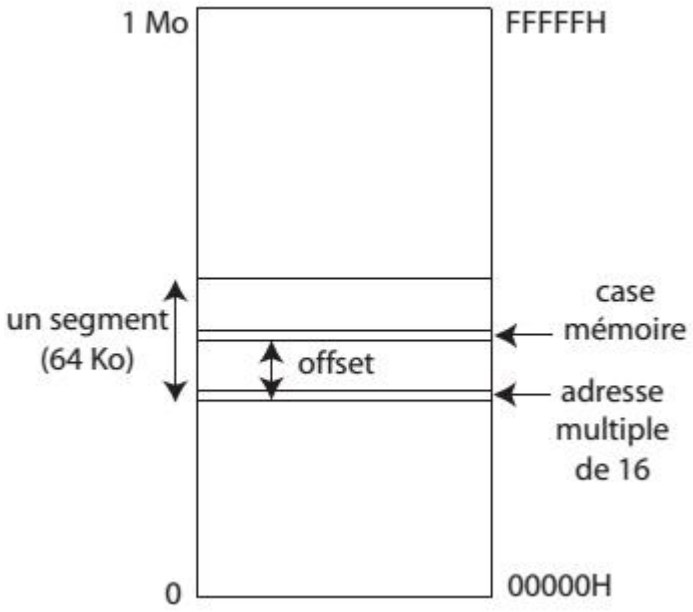
FFFFFH F8000H	Monitor Program	64 KB EPROM System Memory 27256 x 2
F0000H	Demo Program	
E0000H	User Memory	ROM, RAM
10000H	Non-Use	OPEN
00400H	Program and Data	64 KB SRAM Program Memory 62256 x 2
00000H	Interrupt Vector Table	

L'espace mémoire adressable par le 8086 est de 2^{20}=1 Mo (20 bits d'adresses). Cet espace est divisé en segments. Un segment est une zone mémoire de 64 Ko octets définie par son

adresse de départ qui doit être un multiple de 16. Dans une telle adresse, les 4 bits de poids faible sont à zéro. On peut donc représenter l'adresse d'un segment avec seulement ses 16 bit de poids fort, les 4 bits de poids faible étant implicitement à 0. Pour désigner une case mémoire parmi les 2^{16}= 65 536 contenues dans un segment, il suffit d'une valeur sur 16 bits.
Ainsi, une case mémoire est repérée par le 8086 au moyen de deux quantités sur 16 bits :

- ✓ L'adresse d'un segment ;
- ✓ Un déplacement ou offset (appelé aussi adresse effective) dans ce segment.

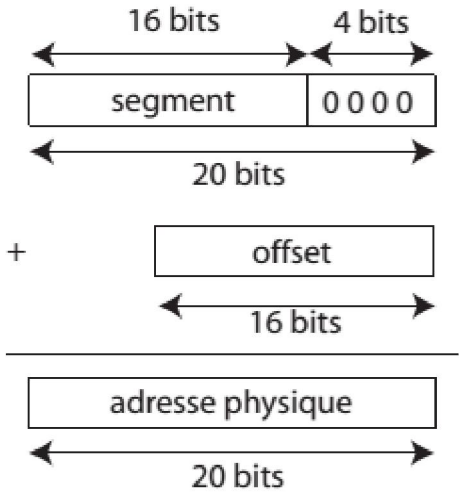
Cette méthode de gestion de la mémoire est appelée segmentation de la mémoire.



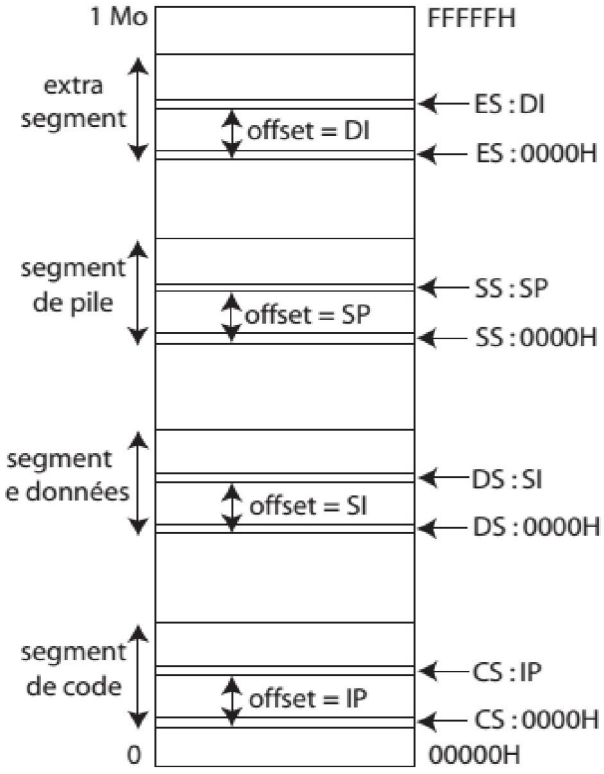
La donnée d'un couple (segment, offset) définit une adresse logique notée sous la forme

segment : offset.

L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexa) est appelée adresse physique car elle correspond à la valeur envoyée réellement sur le bus d'adresses A0 - A19. La correspondance entre adresse logique et adresse physique est illustré comme suit :



Les registres de segments, associés aux pointeurs et aux index, permettent au microprocesseur 8086 d'adresser l'ensemble de la mémoire.



4. Manipulations sur le kit MTS-86C

Pour écrire et exécuter un programme, la communication entre l'utilisateur et le kit MTS-86C est assuré par le clavier (Figure.4) et le moniteur (l'afficheur). Le clavier et l'afficheur permettent une utilisation du système MTS-86C en mode autonome. On peut rentrer le programme en hexa sur le clavier et visualiser le contenu de la mémoire sur le moniteur

- **Le clavier** : le clavier comporte 16 touches pour les digits hexadécimaux et 8 touches fonctions et il apparaît comme suit :

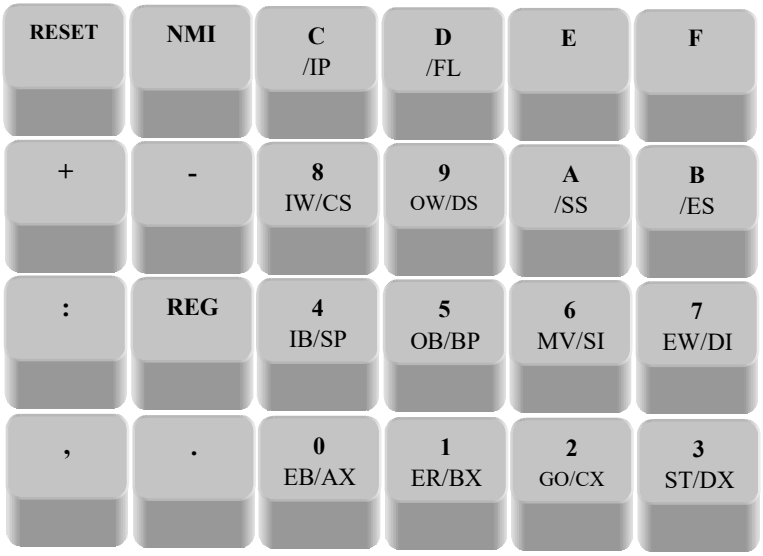


Figure.4 : Le clavier du Kit MTS-89C

- **Le moniteur** : le système MTS-86C est piloté par un moniteur en ROM. ***En mode autonome***, le moniteur assure la gestion de la saisie du clavier, l'affichage du contenu de la mémoire ou des messages de dialogues et l'exécution des commandes de l'utilisateur. ***En mode liaison***, le moniteur assure le dialogue avec le PC et l'exécution des requêtes de celui-ci (chargement de programme utilisateur en RAM, exécution du programme, lecture/écriture de la mémoire et des ports, etc.).

  - **Les touches de fonctions**

Touches	Description
RESET	La touche RESET vous permet d'initialiser la carte MTS-86C. Quand la touche est pressée le 8086 envoie un message sur le LCD et le monitor est près.
NMI	La touche NMI génère une interruption non masquable de type 2.
+	La touche+ vous permet d'additionner 2 valeurs hexadécimales.
-	La touche - vous permet de substituer une valeur hexadécimale d'un autre.
:	Cette touche vous permet de séparer une adresse entrée sur deux parties ; la valeur de segment ou de l'offset.
REG	Cette touche vous permet d'utiliser les contenus des registres du 8086 comme une adresse ou une donnée en entrée.
,	Cette touche sépare les entrées saisies à partir des touches du clavier et incrémente l'adresse mémoire à chaque entrée.

.	Lorsque cette touche est appuyée la commande courante est exécutée. Notez que lorsque la touche GO est appuyée, le 8086 lance l'exécution à partir de l'adresse spécifiée quand la touche est pressée.

- **Les touches hexadécimales**

Touche Hex	Commande		Registre	
	Acronyme	Nom	Acronyme	Nom
0 EB/AX	EB	Examine par Octet	AX	Examine par Octet
1 ER/BX	ER	Examine le registre	BX	Base
2 GO/CX	GO	Go	ex	Compteur
3 ST/DX	ST	Un pas (STEP)	DX	Données
4 IB/SP	IB	Entrée Octet	SP	Stacker Pointer
5 OB/BP	OB	Sortie Octet	BP	Base Pointer
6 MV/SI	MV	Move	SI	Source Index
7 EW/DI	EW	Examine par Word	Dl	Destination Index
8 IW/CS	IW	Entrée Word	CS	Code Segment
9 OW/DS	ow	Sortie Word	DS	Data Segment
A/SS	non	N/A	SS	Stack Segment
B/ES	non	N/A	ES	Extra Segment
C/IP	non	N/A	IP	Pointeur d'instruction
D /FL	non	N/A	FL	Flag
E	non	N/A	non	N/A
F	non	N/A	non	N/A

5. Manipulation en mode autonome

5.1 Chargement du code hexadécimal d'un programme en assembleur

Soit le programme en assembleur suivant :

```
MOV  AX, 1254H    ; affecter au registre AX la valeur 1254 en Hexadécimale
MOV  BX, 4567H    ; affecter au registre BX la valeur 4567 en Hexadécimale
MOV  CX, AX       ; recopie le contenu du registre AX dans le registre CX
MOV  AX, BX       ; mettre le contenu du registre BX dans le registre AX
MOV  BX, CX       ; recopie le contenu du registre CX dans le registre BX
HLT               ; Fin du programme
```

L'équivalent en hexadécimal de ce code est donné dans le tableau ci-dessous :

Programme en assembleur	**L'équivalent en hexadécimal**
MOV AX,1254H	B8 54 12
MOV BX,4567H	BB 67 45
MOVCX,AX	8B C8
MOV AX,BX	8B C3
MOVBX,CX	8B D9
HLT	F4

Ce petit programme permet de faire la permutation du contenu de deux registres AX et BX.

L'édition de ce code hexadécimal est possible grâce au clavier de notre Kit MTS-86C. L'insertion du programme à partir de l'adresse physique 01000h (0100 :0000h adresse

logique) se fait en appuyant sur la séquence des boutons suivants :

- ✓ La **RESET**
- ✓ La touche **EB**/AX (Examine par Octet), le moniteur affiche

Seg. Off	DATA
0000 :0000	BB

- ✓ Entrer l'adresse segment 0100 par le clavier 0→**1**→**0**→**0**→**0**
- ✓ Appuyer sur la touche (:**)**
- ✓ Entrer l'adresse offset 0000
- ✓ Appuyer sur la touche (,**)** pour examiner le contenu de la case mémoire
- ✓ Pour modifier le contenu de la case mémoire courante, enter le premier octet de notre programme (B8) par claver B→**8.**
- ✓ Appuyer sur la touche (**,**)
- ✓ Entrer le deuxième octet 54
- ✓ Continuer avec la même façon jusqu'au dernier octet (de l'octet 3 à l'octet n)
- ✓ Appuyer sur la touche (**.**)

Cette procédure nous permet aussi d'examiner et de visualiser le contenu d'une case mémoire.

5.2 Exécuter un programme en mode autonome

Dans le paragraphe précédent, Nous avons vu comment faire entrer le programme en code machine à partir de l'adresse (0100 :0000h). Pour exécuter ce programme, il suffit d'appuyer sur la séquence des boutons suivants :

- ✓ La touche **RESET**
- ✓ La touche **GO**/AX (Examine par Octet), le moniteur affiche

Seg. Off	DATA	
0000 :0000	XX	GO

- ✓ Entrer l'adresse segment 0100
- ✓ Appuyer sur la touche (**:**)
- ✓ Entrer l'adresse offset 0000
- ✓ Appuyer sur la touche (**.**)

Pour voir le résultat de ce programme, on a besoin d'examiner le contenu des registres du 8086 ou les case mémoires qui contiennent les opérandes et les résultats du programme.

5.3 Examiner et modifier le contenu d'un registre

On examine le contenu d'un registre AX par exemple en appuyant sur les touches suivantes :

- ✓ La touche **RESET**
- ✓ La touche **ER**/BX (Examine par Octet), le moniteur affiche

```
Examine Register

_
```

- ✓ Appuyer sur la touche (GO/**CX**) pour examiner le contenu du registre CX
- ✓ Appuyer sur la touche (,) pour examiner le contenu du prochain registre (DX)

```
Examine Register

DX:0100
```

- ✓ Pour modifier le contenu d'un registre (DX dans ce cas) à 1234, en appuyant sur la séquence **1→2→3→4**
- ✓ Appuyer sur la touche pour terminer la commande (.**)**

On peut aussi examiner le contenu d'un registre par la commande **(REG)** comme suit :

- ✓ Appuyer sur la touche (REG)
- ✓ Appuyer sur la touche N°4(IB/**SP**) pour examiner le contenu du registre SP.
- ✓ Appuyer sur la touche (.**)** pour terminer la commande.

5.4 Exécuter un programme pas à pas en mode autonome

Pour exécuter un programme pas à pas (mode STEP), il suffit d'appuyer sur la séquence des boutons suivants :

- ✓ La touche **RESET**
- ✓ La touche **ST**/DX

```
Seg. Off        DATA

0000 :0000    XX     GO
```

- ✓ Entrer l'adresse segment 0100
- ✓ Appuyer sur la touche (:**)**
- ✓ Entrer l'adresse offset 0000
- ✓ Appuyer sur la touche (**,**) pour exécuter la commande MOV AX,1254H
- ✓ Appuyer sur la touche (**,**) pour exécuter la commande MOV BX,4567H

.

.

.

- ✓ Appuyer sur la touche pour terminer la commande (**.**)

5.5 Exécuter une instruction et visualiser le résultat

Pour exécuter l'instruction **MOV AX, 1254H** par exemple, et visualiser le contenu du registre AX en appuyant sur les touches suivantes :

- ✓ La touche **RESET**
- ✓ La touche **ST**/DX

```
Seg. Off        DATA

0000 :0000      XX      GO
```

- ✓ Entrer l'adresse segment 0100
- ✓ Appuyer sur la touche (**:**)
- ✓ Entrer l'adresse offset 0000
- ✓ Appuyer sur la touche (**,**) pour exécuter la commande MOV AX,1254H
- ✓ Appuyer sur la touche pour terminer la commande (**.**)
- ✓ Appuyer sur la touche ER/BX et le moniteur affiche

```
Examine Register

_
```

- ✓ Appuyer sur la touche (EB/**AX**) pour visualiser le contenu de AX
- ✓ Appuyer sur la touche (,) si vous voulez examiner d'autre registres si non, appuyer sur la touche (.) pour terminer la commande
- ✓ Appuyer à nouveau sur la touche **ST**/DX si vous voulez exécuter d'autre instructions

6. Manipulation en mode liaison (PC ---> kit MTS 86-C à travers RS232)

Le kit MTS-86C ne supporte que les fichiers de type **'. hex'**. Pour créer ce fichier à partir du ficher source **'.asm'**, il faut faire appel à un logiciel assembleur qui permet la traduction d'un code source écrit en langage mnémoniques (jeu d'instructions) au langage machine (codes binaires correspondant aux instructions). Il contient toujours des programmes de compilation, d'édition de lien et de conversion. Pour programmer en assembleur, trois phases sont nécessaires :

- Saisir le code source avec un éditeur de texte.
- Compiler le code source pour obtenir le code objet.
- Editer les liens pour avoir un programme exécutable.

Plusieurs logiciels permettent le passage entre ces trois phases comme :

- MASM (Microsoft Assembler : avec LINK comme éditeur de lien)
- TASM (Turbo assembler : avec TLINK comme éditeur de lien)
- NASM, etc.

a) Le programme MASM de Microsoft ou TASM de Borland, permettent de créer un fichier intermédiaire appelé « fichier Objet ». La commande suivante génère la ficher .obj. Taper la commande suivante :

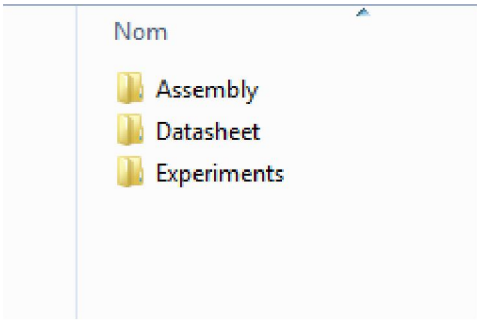
C:\ASM86> masm Tp0

b) Le programme « LINK » de Microsoft ou « TLINK » de Borland créent à partir du fichier « .obj » un fichier « .exe ». Tapez la commande suivante :

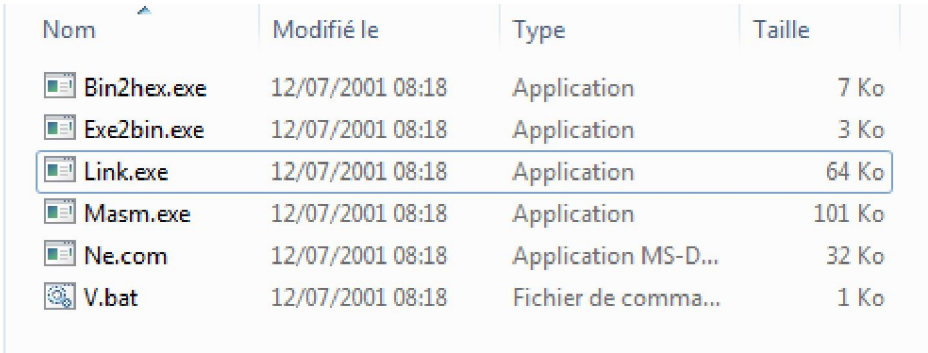
C:\ASM86> link Tp0

c) Le programme «EXE2BIN » convertit le fichier .exe en un fichier .bin. Tapez la commande suivante : C:\ASM86> exe2bin Tp0

Le kit MTS-86C possède ses propres programmes de compilation, d'édition de lien et de conversion. Ces programmes sont fournis avec le kit dans un CD qui contient les dossiers suivants :



Le dossier « Assembly » contient les programmes suivants :



6.1 Conversion d'un code assembleur (.asm) au code hexadécimal (.hex)

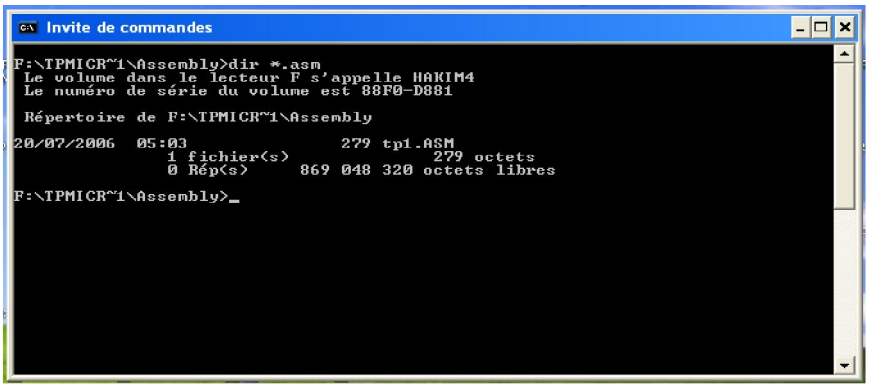
Les étapes suivantes permettent de convertir un code assembleur (.asm) en code machine (.hex) :

a) Installation du logiciel Assembleur du Kit MTS-86C

Pour installer ce logiciel, il suffit de copier le répertoire "Assembly" sur votre poste de travail.

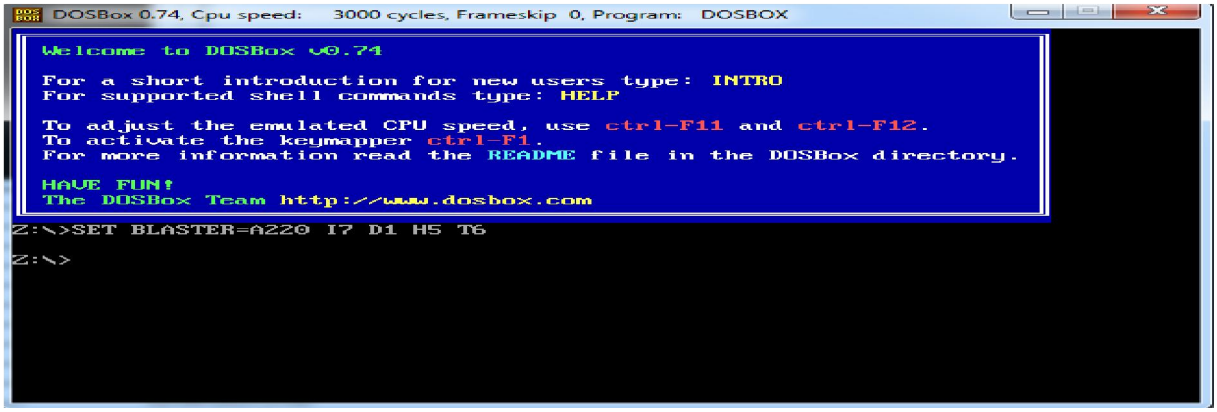
b) Lancer l'invite de la commande DOS

Basculer d'abord en mode DOS : Cliquer sur programmes/Accessoires/Invites de commandes. Puis change le dossier vers "Assembly"



Les logiciels assembleurs tel que NASM, TASM et les programmes du kit MTS-86C ne fonctionnent pas sur Windows 7 facilement en raison de quelques problèmes de compatibilité. Pour résoudre le problème de compatibilité, nous allons utiliser un logiciel open source nommé « **DOSBox** ». Après l'installation, du **DOSBox** **« version 0.74 »** un raccourci de DOSBox sera créer sur le bureau.

Après l'exécution de **DOSBox 0.74**, un écran noir apparaît :



Maintenant, entrer la commande suivante :

MOUNT C C:\ Assembly (and press Enter)

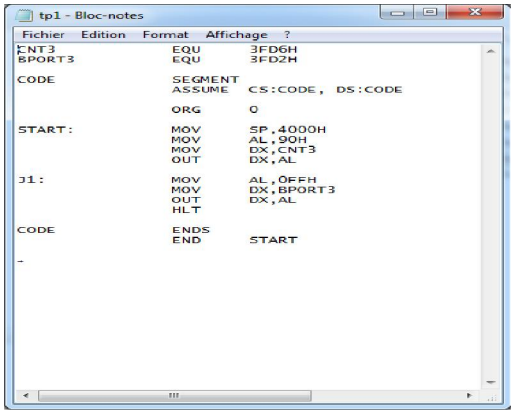
Un message apparaîtra disant « **Drive C : is mounted as Local Directory C:\Assembly** » ; C'est-à-dire le lecteur C : est associé au répertoire local C:\Assembly\

Ensuite, tapez la commande **C :**

Dès maintenant, nous pouvons travailler avec les programmes du kit.

c) Saisir le code source avec un éditeur de texte

Supposons que nous avons un programme assembleur. Pour saisi ce programme, on utilise n'importe quel éditeur de texte.



Nous avons maintenant un fichier assembleur « tp1.asm »

d) Compiler, éditer les liens et Convertir en fichier «. Hex »

Avant d'exécuter un programme, il faut d'abord l'assembler puis éditer les liens. Pour assembler le programme source portant l'extension « .asm », il faut utiliser toujours sous DOS la commande « masm ». Si le fichier source ne contient pas d'erreur syntaxique, un fichier «. Obj » sera créer. Par exemple :

masm nom_fichier.asm

```
C:\>masm tp1
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [tp1.OBJ]: tp1
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51768 + 464776 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

C:\>
```

Comme le fichier « *nom_fichier.obj* » est inutilisable par le DOS, on utilise sous DOS la commande « link », pour convertir ce fichier en un fichier exécutable « *nom_fichier.*exe ».

Par exemple :

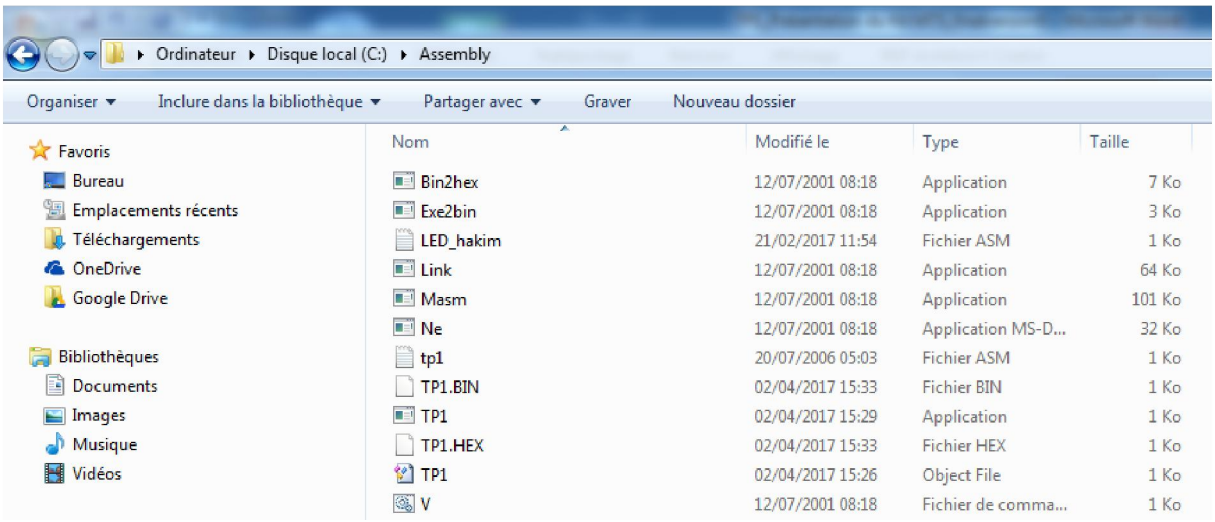
link nom_fichier.obj

```
C:\>link tp1.obj

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [TP1.EXE]: tp1
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

C:\>
```

Pour créer le fichier « *nom_fichier.hex »,* on utilise les deux commandes exe2bin et bin2hex respectivement.

```
C:\>exe2bin tp1.exe

C:\>bin2hex tp1.bin

================================================================================
Converter from BIN to INTEL HEX file V 1.0          made by Sigma Intelligence
This program is the Public ware. Anyone can use this.  HAK LIM Micro Instrument
================================================================================
Input BIN file name : tp1.bin
Input start address :
Start address is $0000

C:\>_
```

Après l'exécution de ces instructions, les fichiers "tp1.exe", "tp1.bin" et "tp1.hex" seront créer dans le répertoire "Assembly", comme indiquer dans la figure ci-dessous :

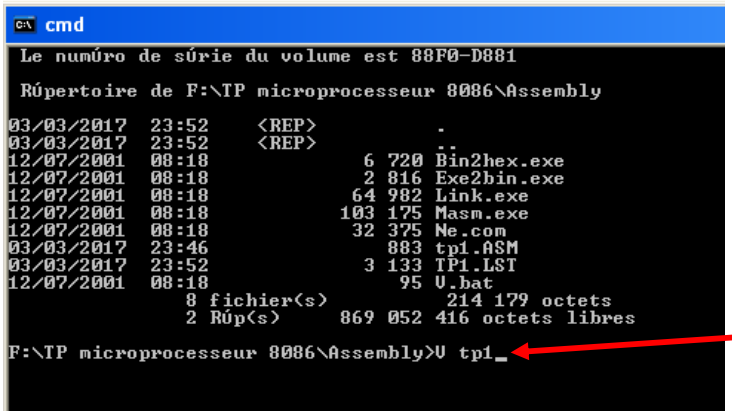


Les étapes précédentes peuvent être résumées comme suit :

Afin d'éviter de reprendre toutes ces étapes à chaque fois qu'il y'a création d'un fichier « **.hex** », un fichier batch « **V.bat** » est utilisé pour crée le fichier hex

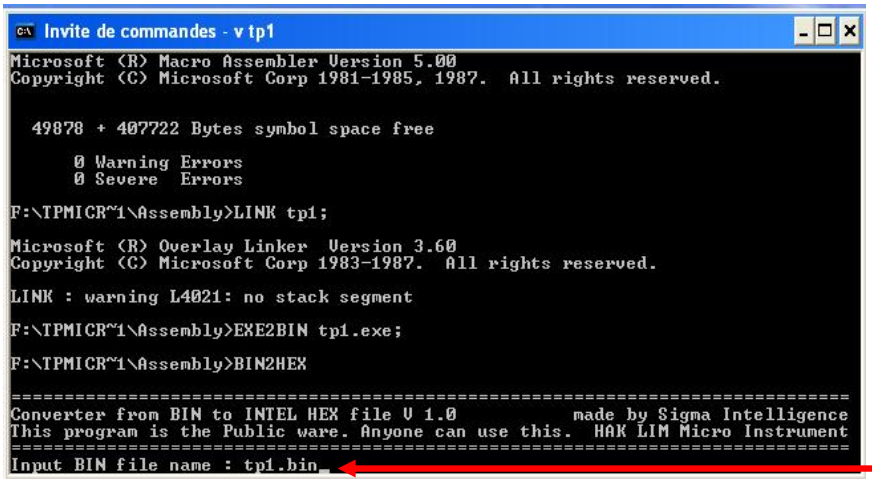
Entrer la commande "V *nom_fichier* " (V est une application de type Batch), *nom_fichier* est le nom du fichier ".asm".

Nous avons un fichier assembleur "tp1.asm", et nous voulons créer son code machine (tp1.hex)



Taper la commande

Après l'exécution de la commande "V tp1", la machine affiche le message de la figure ci-dessous.



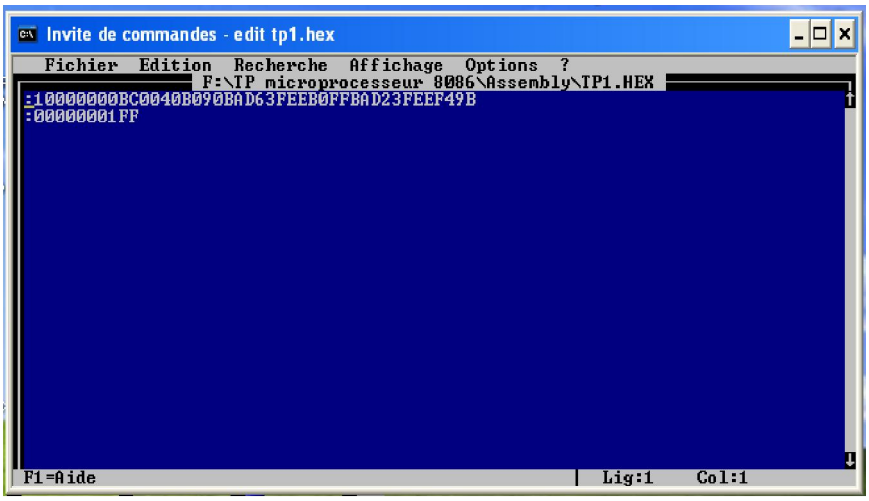
Le fichier binaire

Lorsque la machine demande à l'utilisateur d'introduire le nom du fichier binaire, il faut introduire le même que ce du fichier « .asm », c'est-à-dire, il faut introduire le nom : "tp1.bin".

La commande V tp1 exécute les opérations suivantes :

- Compile le programme « tp1.asm » pour obtenir le code objet « tp1.obj »
- Éditer les liens pour obtenir le fichier « tp1.exe »
- Converti le fichier « tp1.exe » pour obtenir le fichier « tp1.Hex »

La figure suivante, montre le contenu du fichier "tp1.hex"



6.2 Chargement d'un programme en code hexadécimal

Cette section permet la description des commandes qui permettent la communication avec le KIT MTS-86C à travers une liaison série utilisant l'Hyper terminal, dont le but de :

- Envoyer un programme sous forme hexadécimale.
- Visualiser, modifier les registres du microprocesseur 8086, etc.

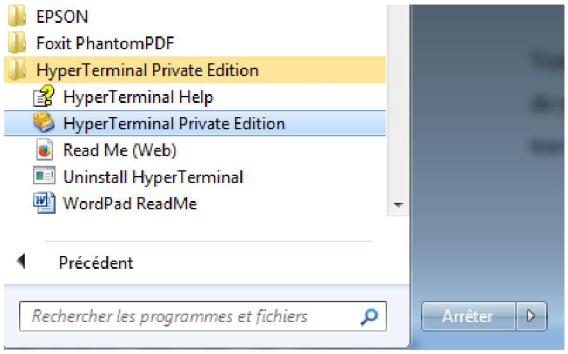
Au-delà du Windows XP, le logiciel Hyper terminal n'est plus fournit. On peut télécharger une version gratuite.

Etape 1 : Démarrez le programme HyperTerminal

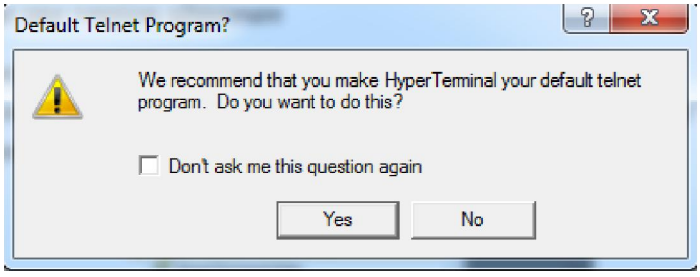
Cliquez sur le bouton "Démarrer",

Aller sur "Tous les programmes \ HyperTerminal Private Edition"

Cliquer sur "HyperTerminal Private Edition"

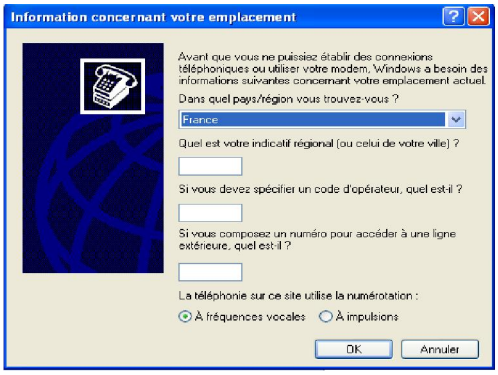


✓ Un dialogue propose d'utiliser HyperTerminal pour du réseau TCP/IP distant.



✓ Répondre Non

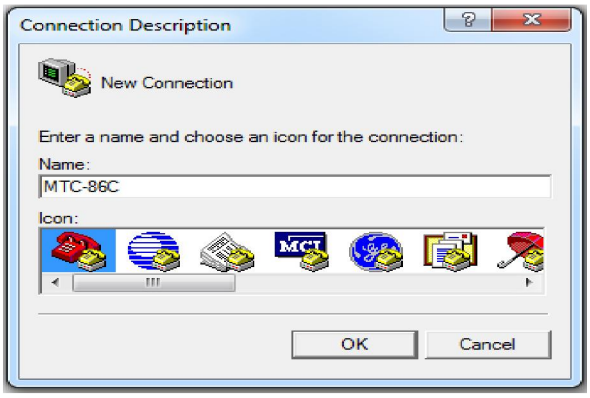
✓ La première fois, il peut vous demander le téléphone, et le code zone (c'est pour la partie modem qui ne nous intéresse pas).



✓ Ensuite une boîte de dialogue peut avoir un aspect légèrement différent pour d'autre version de Windows. La boîte de dialogue « Connection Description » apparaît dans la zone de travail d'HyperTerminal.

Etape 2 : Nommer la session HyperTerminal

✓ Mettre un nom quelconque dans la fenêtre « Name », tapez par exemple **" MTS-86C ",** puis choisir une icône (optionnel)**.**



Etape 3 : Spécifiez l'interface de connexion de l'ordinateur

✓ Un dialogue demande quel port de communication est utilisé. Choisir le port de communication (COM **1** par exemple). Vous pouvez aller dans le gestionnaire de périphérique pour en savoir plus sur vos ports.

✓ Utilisez la flèche de déroulement dans le champ **Se connecter en utilisant** : pour sélectionner COM1, puis cliquez sur OK pour afficher la boîte de dialogue Propriétés

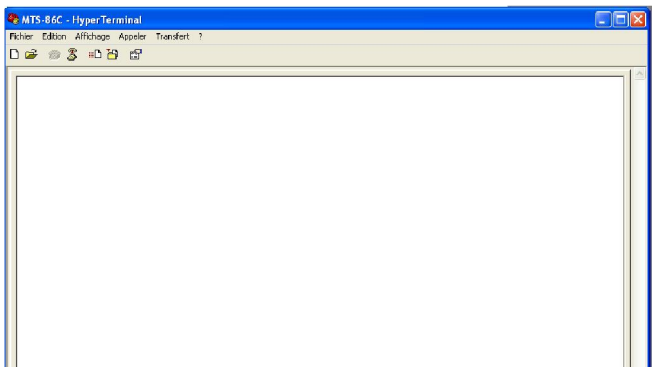
Etape4 : Spécifiez les propriétés de connexion de l'interface

✓ Une fenêtre de dialogue demande la configuration des paramètres de la liaison série (débit, nombre de bits des données, etc.).

✓ Pour notre cas, utiliser les flèches de déroulement pour sélectionner les réglages montrés ci-dessous :

Bits par seconde	**19200 bauds**
Bits de données	8 bits de données
Parité	None
bit de stop	1 bit de stop
Contrôle de flux	Xon/Xoff

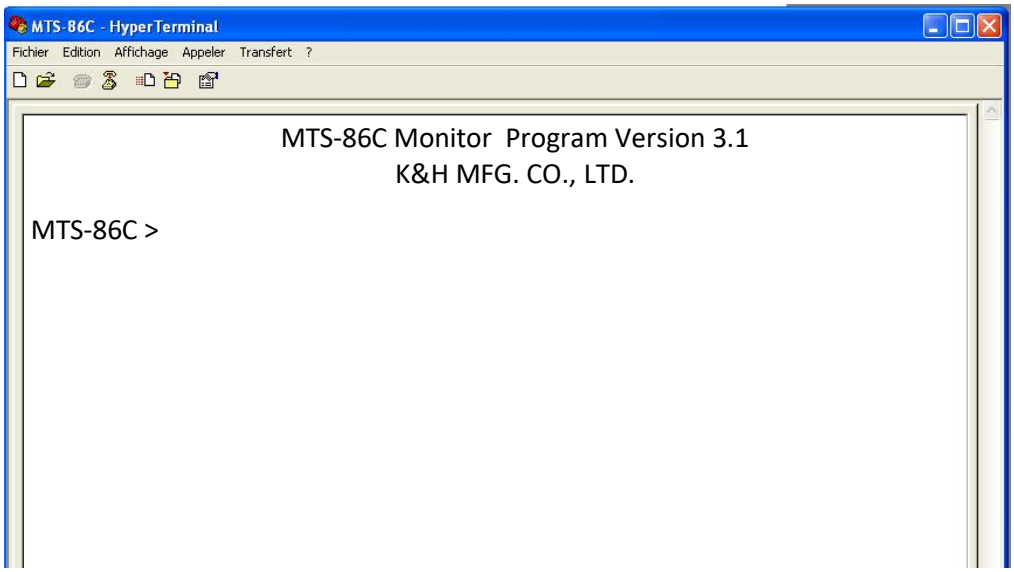
✓ Cliquez sur OK. Une zone de travail HyperTerminal vide apparaît.



✓ Maintenant le PC est en mode terminal, connecter le PC avec le Kit MTS-86C via la liaison RS232.

Etape 5 : Lorsque la fenêtre de la session HyperTerminal apparaît, mettre le Kit 8086 sous tension, si ce n'est déjà fait.

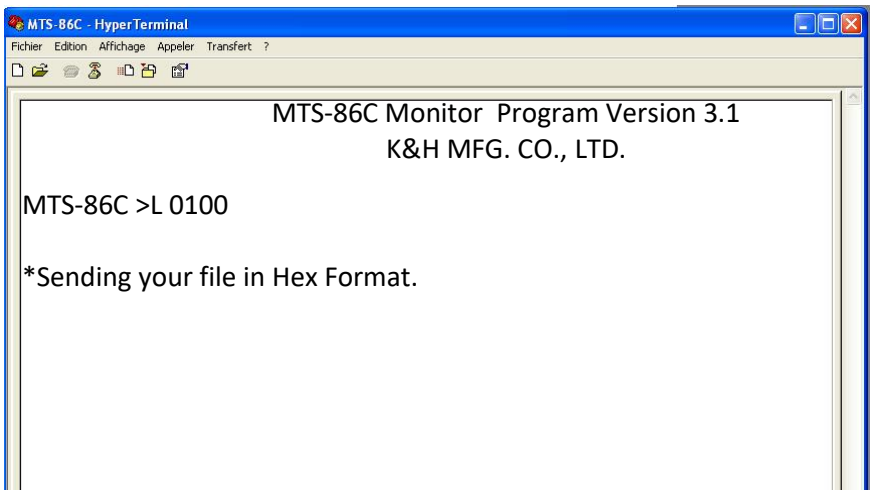
- ✓ La connexion s'est alors déroulée avec succès**.** Appuyer ensuite sur la touche RESET, puis sur n'importe quelle touche alphabétique (A, …, F) jusqu'à ce que le menu principal apparaisse et on obtient la fenêtre de l'HyperTerminal suivante :



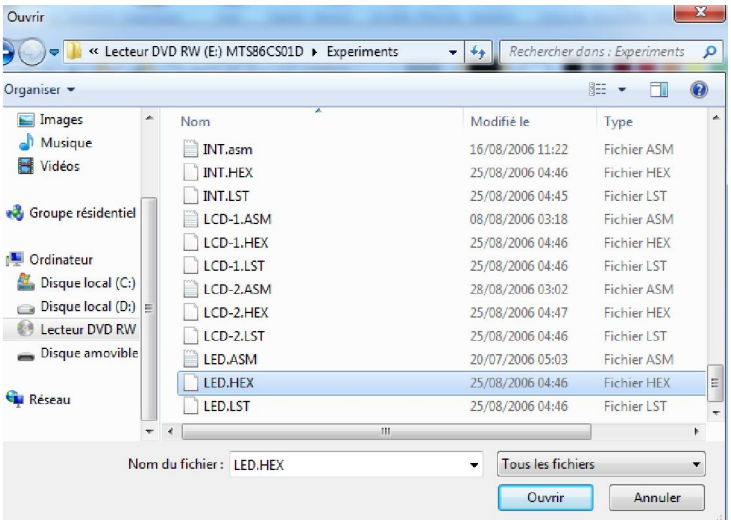
Etape 6 : Chargement d'un programme

Pour charger un programme vers la mémoire du kit MTS-86C, il faut spécifier l'adresse de début. Pour charger par exemple le programme dans la zone mémoire définit par l'adresse de début (0100 :0000h). On écrit

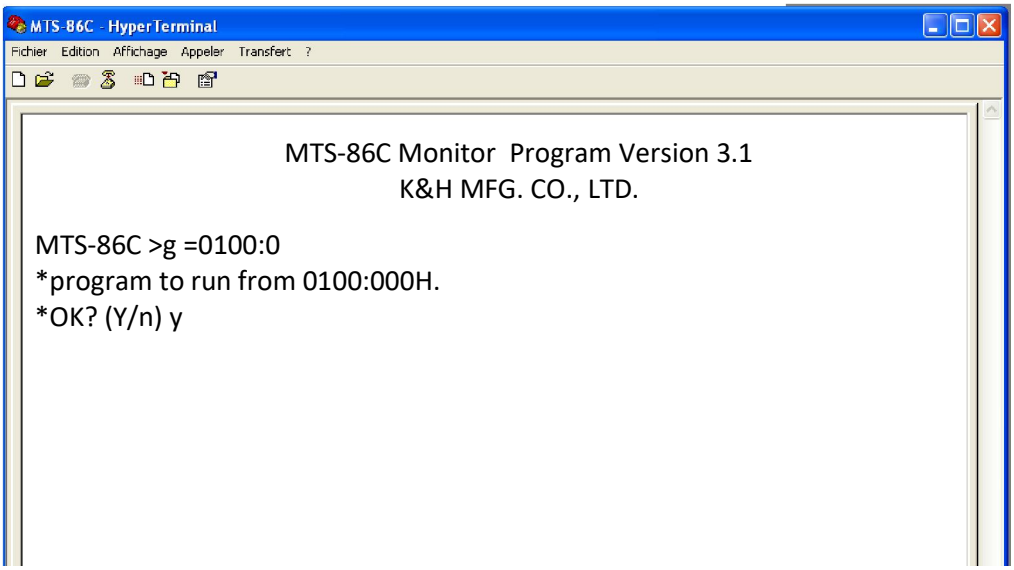
- ✓ MTS-86C > L 0100
- ✓ Valider par entrée. Maintenant le système est près de transféré le fichier Hex.



- ✓ Aller vers Transfert→Send Text File
- ✓ Choisir votre fichier Hex (LED.HEX par exemple)
- ✓ Appuyer sur ouvrir



Etape 7 : pour exécuter le programme envoyé. On peut utiliser le clavier du kit 8086 (mode autonome) ou bien comme montre la figure ci-dessous (mode liaison)



Lorsque vous taper "y", les huit (08) LEDs du kit MTS 86-C seront allumées

Etape 8 : Fermeture de la session

- ✓ Pour mettre fin à la session en mode console à partir d'une session HyperTerminal, sélectionnez : Fichier > Quitter
- ✓ Lorsque la boîte d'avertissement de déconnexion HyperTerminal apparaît, cliquez sur Oui
- ✓ L'ordinateur demande ensuite si la session doit être enregistrée. Cliquez sur Oui.

6.3 Examine et modifier le contenu de la mémoire : La commande E

Syntaxe E <adresse> ou E <adresse> [<liste d'octets>]

Pour examiner le contenu de mémoire à l'adresse [0100:0], taper l'instruction suivante:

```
MTS-86C<E 0100:0
0100:0000 00 _
```

Si, nous désirons que l'octet 00 sera remplacé par le caractère « A » en code ASCII qui correspond en code hexadécimal 41H. Alors, nous devons toujours travailler avec les codes hexadécimaux. Ainsi, nous devons taper après l'espace l'octet 41 et le curseur change sa position vers l'octet suivant.

```
MTS-86C<E 0100:0
0100:0000 00  41
0100:0001 00 _
```

Pour aller à l'adresse suivante sans faire du changement, taper la touche "espace"

```
MTS-86C<E 0100:0
0100:0000 00  41
0100:0001 00
0100:0002 00 _
```

Si vous entrez à une adresse, un format non hexadécimal, la machine vous affiche le message "Not Hex", et elle vous demande de corriger le format de données.

```
MTS-86C<E 0100:0
0100:0000 00  BC
0100:0001 00
0100:0002 00  2Y    NOT HEX !
0100:0002 00 _
```

6.4 Afficher les données de mémoire sur l'écran de PC : La commande D

Syntaxe D ou D <adresse> ou D <bloc>

Cette commande donne la suite d'octets du bloc défini. Si on spécifie une adresse seule, le bloc fera 128 octets. D sans paramètres "dump" les 128 octets suivants.

Si on veut par exemple visualiser le contenue de la mémoire à partir de l'adresse 0100, on écrit : D 0100:0, comme montrer dans la figure ci-dessous

```
MTS-86C > D 0100:0
0100:0000 BC 00 40 B0 90 BA D6 3F-EE B0 FF BA D2 3F EE F4    ..@...?.
0100:0010  00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0020 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0030 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0040 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0050 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0060 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0070 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
```

Sur la figure ci-dessus trois zones sont visibles :

- ✓ Une zone des adresses segment : offset
- ✓ Une zone qui contient des données en représentation hexadécimale
- ✓ Une dernière zone montrant le contenu des cases mémoires en représentation ASC II.

Travaillons maintenant sur les données avec le registre Data Segment (DS). La commande **D** (dump) pour extraire le contenu du segment des données commençant à partir de l'adresse 0000H. Alors, nous devons taper l'instruction comme elle est montrée sur l'écran suivant :

```
MTS-86C > D ds:0
0100:0000 BC 00 40 B0 90 BA D6 3F-EE B0 FF BA D2 3F EE F4    ..@...?.
0100:0010  00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0020 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0030 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0040 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0050 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0060 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
0100:0070 00 00 00  00 00  00  00  00-00  00 00  00   00  00 00 00    .....
```

Pour visualiser le contenu des autres segments par exemple le code segment, nous devons taper l'instruction MTS-86C > D CS:0 [Entrer]

6.5 Consultation et mise à jour de contenus des registres : La commande R

Syntaxe R [<registre>]

La commande R (registre) sert à :

- ✓ Afficher tous les registres si aucun nom de registre n'est spécifié.

```
MTS-86C > R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FF00  BP=0000  SI=0000  DI=0000
DS=0200  ES=0100  SS=0000  CS=0100  IP=0000  NV UP DI PL NZ NA PO NC
```

- ✓ Afficher le contenu du registre et modifier son contenu : Afin de changer le contenu d'un registre, par exemple le registre AX, on tape le code suivant : **R AX**

Alors on obtient :

```
MTS-86C > R AX
AX 0000
:1234
```

En affichant une autre fois le contenu par l'intermédiaire de la commande R, on obtiendra :

```
MTS-86C > R
AX=1234  BX=0000  CX=0000  DX=0000  SP=FF00  BP=0000  SI=0000  DI=0000
DS=0200  ES=0100  SS=0000  CS=0100  IP=0000  NV UP DI PL NZ NA PO NC

MTS-86C > _
```

Nous pouvons remarquer que le contenu du registre AX a été modifié est devenu 1234.

6.6 Exécuter un programme : La commande G

Syntaxe G [=<adresse1>] [<adresse2>...]

Exécute le programme à partir de l'adresse1 si elle est spécifiée (elle est facultative) ou à partir de CS : IP si elle n'est pas indiquée. Par exemple

```
MTS-86C > G=0200:0000
      * Program to run from 0200:0000H
      * Ok? (Y/n) Y

_
```

Le kit MTS86-C exécute le programme à partir de l'adresse 0200h jusqu'à la fin du programme (ce programme permet d'allumer les 8 LEDs de kit).

On peut stopper l'exécution d'un programme par l'instruction :

```
MTS-86C > G=0100:0003,0009
      * Program to run from 0100:0003H
      * Ok? (Y/n) Y
AX=0090 BX=0000 CX=0000 DX=3FD6 SP=FF00 BP=0000 SI=0000 DI=0000
 DS=0100 ES=0100  SS=0000  CS=0100  IP=0009  NV UP DI PL NZ NA PO NC
MTS-86C > _
```

Le programme s'exécute à partir de l'adresse (0100:0003) et s'arrête à l'adresse (0100:0009), puis le kit affiche le contenu des tous les registres.

Une autre façon de stopper l'exécution du programme est d'assigner le registre segment CS (CS=0100 par exemple), puis on écrit l'instruction :

```
MTS-86C > G=0003,0005
      * Program to run from 0100:0003H
      * Ok? (Y/n) Y
AX=0090 BX=0000 CX=0000 DX=3FD6 SP=FF00 BP=0000 SI=0000 DI=0000
 DS=0100 ES=0100  SS=0000  CS=0100  IP=0009  NV UP DI PL NZ NA PO NC
MTS-86C > _
```

Le programme s'exécute à partir de l'adresse (CS :0003) et s'arrête à l'adresse (CS :0005), puis le kit affiche le contenu des tous les registres.

Remarque :

Dans le cas où le système est rebooter (reset), le registre de segment CS=0100 par défaut.

6.7 Trace de programmes : La commande T

Syntaxe T[=<adresse>] [<N>...]

Exécute <N> instructions (1 si N n'est pas spécifié), en commençant à <adresse> ou en CS:IP si <adresse> n'y est pas. Ceci permet de suivre le cheminement du programme en cours (visualiser les contenus de registres et d'indicateurs suite à l'exécution de N instructions pour chaque instant). Il suffit de continuer à taper T pour voir ce qui se passe à chaque étape.

Par exemple :

```
MTS-86C > T=0100:0000,2
     * Program to run from 0100:0003H
     * Ok? (Y/n) Y
AX=0090 BX=0000 CX=0000 DX=3FD6 SP=FF00 BP=0000 SI=0000 DI=0000
 DS=0100 ES=0100  SS=0000  CS=0100  IP=0009  NV UP DI PL NZ NA PO NC
MTS-86C > _
```

Le programme exécute deux instructions à la fois à partir de l'adresse (0100:0000).

Exemples :

T=0000,2 le programme exécute deux instructions à la fois à partir de l'adresse (CS:0000).

T=2 le programme exécute deux instructions à la fois à partir de l'adresse (CS:IP).

T le programme exécute une instruction à partir de l'adresse (CS:IP).

6.8 Remplissage d'un bloc mémoire par des données : La commande F

Syntaxe F <adresse début>, <adresse finale>, <data>

Remplit toutes les cellules du segment mémoire compris entre les adresses début et adresse finale par le caractère dont le code ASCII est indiqué.

Par exemple :

Avant de commencer, on affiche le contenu du bloc mémoire **(0300 :0000)**

```
MTS-86C > D 0300:0
0300:0000  B8 12 54 F4 36 00 36 00-36 00 36 00 36 00 36 00   ..T.6.6.6.6.6.
0300:0010  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0020  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0030  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0040  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0050  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0060  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0070  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
```

Puis, on exécute l'instruction de remplissage, comme indiquer ci-dessous :

```
MTS-86C > F 0300:0,20,11
```

Cette commande remplie la zone mémoire entre 0300 :0000 jusqu'à 0300 :0020 par la valeur 11. En affichant une autre fois le contenu de cette zone mémoire par l'intermédiaire de la commande D, on obtiendra :

```
MTS-86C > D 0300:0
0300:0000  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11   ................
0300:0010  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11   ................
0300:0020  11 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   ..6.6.6.6.6.6.
0300:0030  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0040  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0050  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0060  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0070  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.

MTS-86C > _
```

Nous pouvons remarquer que le contenu des cases mémoire a été modifié est devenu 11.

6.9 Déplacement d'un bloc mémoire : La commande M

Syntaxe M <adresse début>, <adresse finale>, <destination >

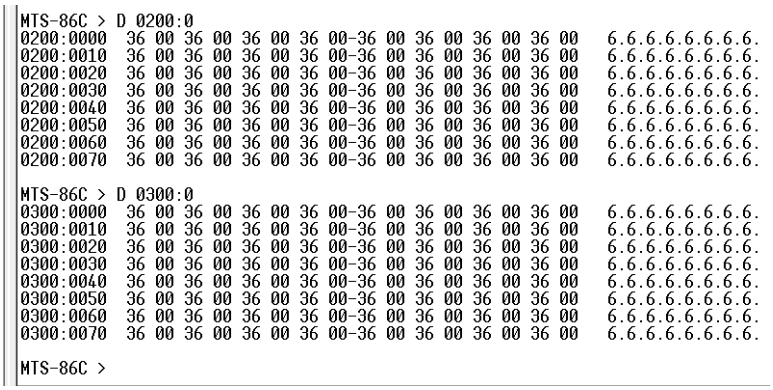
Avant de commencer en affiche le contenu du deux bloc mémoire bloc1 **(0200 :0000 jusqu'à 0200 :0020) et bloc2 (0300 :0000 jusqu'à 0300 :0020)** par l'intermédiaire de la commande D, on obtiendra :

```
MTS-86C > D 0200:0
0200:0000  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0010  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0020  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0030  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0040  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0050  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0060  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0200:0070  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.

MTS-86C > D 0300:0
0300:0000  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11   ................
0300:0010  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11   ................
0300:0020  11 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   ..6.6.6.6.6.6.
0300:0030  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0040  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0050  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0060  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.
0300:0070  36 00 36 00 36 00 36 00-36 00 36 00 36 00 36 00   6.6.6.6.6.6.6.

MTS-86C >
```

Maintenant , on exécute l'instruction suivante

```
MTS-86C > M 0200:0,0020,0300:0
```

Cette commande permet de copie le bloc mémoire (0200 :0000 jusqu'à 0200 :0020) vers le bloc mémoire qui commence par l'adresse 0300 :0000. En affichant une autre fois le contenu des deux blocs :



Nous pouvons remarquer que le bloc1 et bloc2 sont identique.

6.10 Envoi d'un octet vers un périphérique d'entrée /sortie : La commande O

Syntaxe O <adresse périphérique >, <data>

6.11 Réception d'un octet à travers un périphérique d'entrée /sortie : La commande I

Syntaxe I <adresse périphérique >

TP 02 : Techniques de programmation 1

1. Objectifs

L'objectif de ce TP est de concevoir des programmes assembleurs intervenant les instructions les plus utilisées, ainsi que les différents modes d'adressages de microprocesseur 8086, utilisant le Kit MTS-86C.

2. Rappels théoriques

Les instructions peuvent être classées en groupes :

- Instructions de transfert de données.
- Instructions arithmétiques.
- Instructions logiques.
- Instructions de branchement, etc.

2.1. Les instructions de transfert

Elles permettent de déplacer des données d'une source vers une destination :

- Registre vers mémoire.
- Registre vers registre.
- Mémoire vers registre.

Remarque : Le microprocesseur 8086 n'autorise pas les transferts de mémoire vers mémoire (pour ce faire, il faut passer par un registre intermédiaire).

Syntaxe :

mov destination, source

2.2.1. Mode d'adressage

Il existe différentes façons de spécifier l'adresse d'une case mémoire dans une instruction :

a. Adressage par registre

Exemple : mov ax,bx ; charge le contenu du registre bx dans le registre ax. Dans ce cas, le transfert se fait de registre à registre.

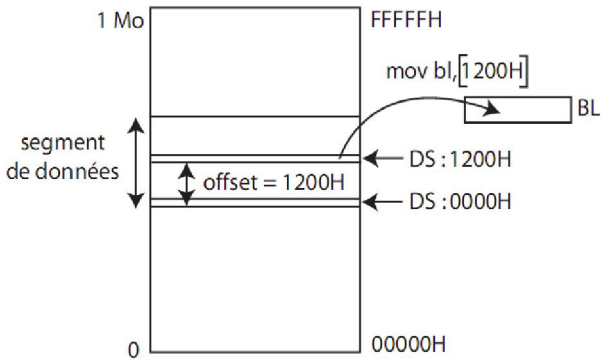
b. Adressage immédiat

Exemple : mov al,12H ; charge le registre al avec la valeur 12H. La donnée est fournie immédiatement avec l'instruction.

c. Adressage direct

Exemple : mov bl,[1200H] ; transfère le contenu de la case mémoire d'adresse effective (offset) 1200H vers le registre bl.

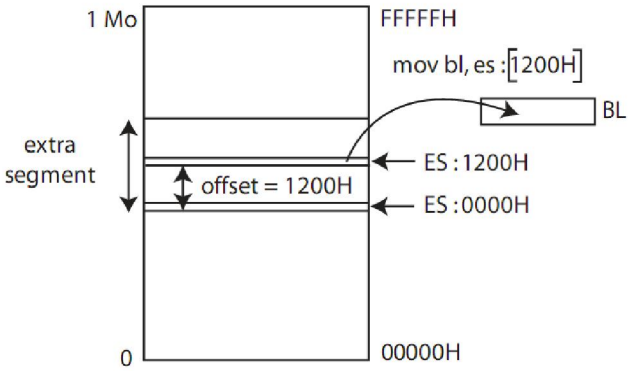
L'adresse effective représente l'offset de la case mémoire dans le segment de données (segment dont l'adresse est contenue dans le registre DS) : segment par défaut.



On peut changer le segment lors d'un adressage direct en ajoutant un préfixe de segment.

Exemple **:** mov bl,es:[1200H]

On parle alors de forçage de segment.

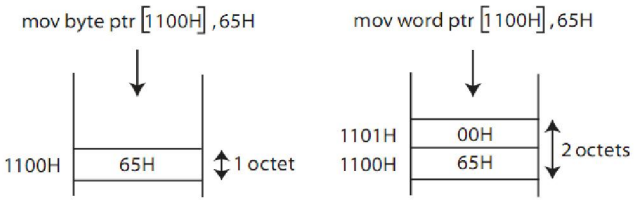


Remarque **:** Dans le cas de l'adressage immédiat de la mémoire, il faut indiquer le format de la donnée : octet ou mot (2octets) car le microprocesseur 8086 peut manipuler des données sur 8 bits ou 16 bits. Pour cela, on doit utiliser un spécificateur de format :

Exemples

mov byte ptr [1100H],65H ; transfère la valeur 65H (sur un octet) dans la case mémoire d'offset 1100H ;

mov word ptr[1100H],65H ; transfère la valeur 0065H (sur deux octets) dans les cases mémoires d'offset 1100H et 1101H.



d. Adressage basé : l'offset est contenu dans un registre de base BX ou BP.

Exemples : mov al,[bx] ; transfère la donnée dont l'offset est contenu dans le registre de base bx vers le registre al. Le segment associé par défaut au registre bx est le segment de données : on dit que l'adressage est basé sur DS.

mov AL,[BP] ; le segment par défaut associé au registre de base BP est le segment de pile. Dans ce cas, l'adressage est basé sur SS.

Remarque

Toute combinaison segment : offset où l'offset désigne un registre pointeur ou un registre index, doit satisfaire l'une des conditions suivantes :

- **IP :** travaille exclusivement avec CS pour adresser les instructions du programme.
- **SP :** travaille exclusivement avec SS pour les opérations de la pile.
- **BP :** travaille de préférence avec SS, peut être associe à DS, CS et ES.
- **BX :** travaille de préférence avec DS, peut être associe à SS, ES et CS.
- **DI :** travaille de préférence avec DS, mais obligatoirement avec ES pour les instructions répétitives, peut être associe à SS et CS.
- **SS :** travaille de préférence avec DS, peut être associe à SS, ES et CS.

e. Adressage indexé : semblable à l'adressage basé, sauf que l'offset est contenu dans un registre d'index SI ou DI, associés par défaut au segment de données.

Exemples **:**

mov al,[si] ; charge le registre AL avec le contenu de la case mémoire dont l'offset est contenu dans SI.

mov[di],bx ; charge les cases mémoire d'offset DI et DI+1 avec le contenu du registre BX.

f. Adressage basé et indexé : l'offset est obtenu en faisant la somme d'un registre de base, d'un registre d'index et d'une valeur constante. Ce mode d'adressage permet l'adressage de structures de données complexes : matrices, enregistrements, ...etc.

Exemples

mov ah,[bx+si+100H] ; mettre le contenu de la case mémoire (DS :BX+SI+100) dans le registre AH.

mov bx,10 ; BX joue le rôle de l'indice ligne de la matrice et il contient la valeur 10

mov si,15 ; SI joue le rôle de l'indice colonne de la matrice et il contient la valeur 15

mov byte ptr matrice[bx][si],12H ; mettre la valeur 12H dans la position [10][15].

2.3. Les instructions arithmétiques

Les instructions arithmétiques de bases sont l'addition, la soustraction, la multiplication et la division qui incluent diverses variantes. Plusieurs modes d'adressage sont possibles.

2.3.1. Addition

Syntaxe : ADD opérande1, opérande2

opérande1← opérande1+ opérande2

***Exemples* :**

- add ah,[1100H] ; ajoute le contenu de la case mémoire d'offset 1100H à l'accumulateur AH (adressage direct).
- add ah,[bx]; ajoute le contenu de la case mémoire pointée par BX à l'accumulateur AH (adressage basé).
- add byte ptr [1200H],05H; ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H (adressage immédiat).

2.3.2. Soustraction

***Syntaxe* :** SUB opérande1, opérande2

opérande1← opérande1- opérande2

2.3.3. Multiplication

***Syntaxe* :** MUL opérande

Cette instruction effectue la multiplication du contenu de AL par un opérande (sur un octet ou du contenu de AX par un opérande sur deux octets). Le résultat est placé dans AX si les données à multiplier sont sur un octet (résultat sur 16 bits), dans (DX,AX) si elles sont sur deux octets (résultat sur 32 bits).

***Remarque* :** L'opérande peut être un registre ou une case mémoire.

2.3.4. Division

***Syntaxe* :** DIV opérande

Cette instruction effectue la division du contenu de AX par un opérande sur un octet ou le contenu de (DX,AX) par un opérande sur deux octets. Si l'opérande est sur un octet, alors AL=quotient et AH=reste; si l'opérande est sur deux octets, alors AX=quotient et DX=reste.

2.3.5. Autres instructions arithmétiques

- **ADC** : Addition avec retenue.
- **SBB** : Soustraction avec retenue.
- **INC** : Incrémentation d'une unité.
- **DEC** : Décrémentation d'une unité.
- **IMUL** : Multiplication signée.
- **IDIV** : Division signée.

2.4. Les instructions logiques

Sont des instructions qui permettent de manipuler des données au niveau de bits. Les opérations logiques de base sont : ET, OU, OU exclusif, Complément à 1, Complément à 2, Décalages et rotations.

2.4.1. ET logique

***Syntaxe* :** AND opérande1, opérande2 ; opérande1← opérande1 ET opérande2

2.4.2. OU logique

Syntaxe : OR opérande1, opérande2 ; opérande1← opérande1 OU opérande2

2.4.3. Complément à 1

Syntaxe : NOT opérande ; opérande ←$\overline{\text{opérande}}$

2.4.4. Complément à 2

***Syntaxe* :** NEG opérande ; opérande ←$\overline{\text{opérande}}$ + 1

2.4.5. OU exclusif

***Syntaxe* :** XOR opérande1, opérande2 ; opérande1← opérande1 XOR opérande2

2.4.6. Instructions de décalages et de rotations

Ces instructions déplacent d'un certain nombre de positions les bits d'un mot vers la gauche ou vers la droite. Dans le cas de décalage, les bits qui sont déplacés sont remplacés par des zéros. Il y a les décalages logiques (opérations non signées) et les décalages arithmétiques (opérations signées). Dans le cas de rotation, les bits déplacés dans un sens où seront réinjectés de l'autre côté du mot.

2.4.6.1. Décalage logique vers la droite (Shift Right)

***Syntaxe* :**

SHR opérande, n ; Cette instruction décale l'opérande de n positions vers la droite.

***Exemple* :**

mov al,11001011B

shr al,1

2.4.6.2. Décalage logique vers la gauche (Shift Left)

Syntaxe

SHL opérande, n ; Cette instruction décale l'opérande de n positions vers la droite.

Exemple

mov al,11001011B

shl al,1

2.4.6.3. Décalage arithmétique vers la droite

Syntaxe

SAR opérande, n ; Ce décalage conserve le bit de signe bien que celui-ci soit décalé.

Exemple

mov al,11001011B

sar al,1

2.4.6.4. Décalage arithmétique vers la gauche

Syntaxe

SAL opérande, n ; Identique au décalage logique vers la gauche.

2.4.6.5. Rotation à droite (Rotate Right)

Syntaxe

ROR opérande, n ; Cette instruction décale l'opérande de n positions vers la droite et réinjecte par la gauche les bits sortant.

Exemple :

mov al,11001011B

ror al,1

2.4.6.6. Rotation à gauche (Rotate Left)

Syntaxe :

ROL opérande, n; Cette instruction décale l'opérande de n positions vers la gauche et réinjecte par la droite les bits sortant.

Exemple :

mov al,11001011B

rol al,1

2.4.6.7. Rotation à droite avec passage par l'indicateur de retenue (Rotate Right through Carry)

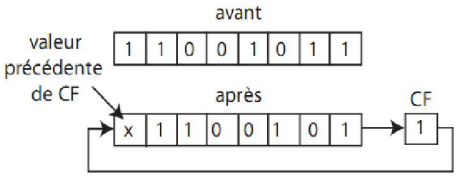
Syntaxe :

RCR opérande, n; Cette instruction décale l'opérande de n positions vers la droite en passant par l'indicateur de retenue CF.

Exemple :

mov al,11001011B

rcr al,1



2.4.6.8. Rotation à gauche avec passage par l'indicateur de retenue (Rotate Left through Carry)

Syntaxe :

RCL opérande, n; Cette instruction décale l'opérande de n positions vers la gauche en passant par l'indicateur de retenue CF.

Exemple :

mov al,11001011B

rcl al,1

2.5. Les instructions de branchement

Les instructions de branchement (ou saut) permettent de modifier l'ordre d'exécution des instructions du programme. Il existe trois types de saut :

- Saut inconditionnel.

- Saut conditionnel.
- Appel de sous-programmes.

2.5.1. Instruction de saut inconditionnel

Syntaxe :

JMP label ; Cette instruction effectue un saut (jump) vers le label spécifié. Un label (étiquette) est une représentation symbolique d'une instruction en mémoire.

2.5.2. Instructions de sauts conditionnels

Syntaxe **:**

Jcondition label ; Un saut conditionnel n'est exécuté que si une certaine condition est satisfaite. Si non l'exécution se poursuit séquentiellement à l'instruction suivante. La condition du saut porte sur l'état de l'un (ou plusieurs) des indicateurs d'état (flags) du microprocesseur.

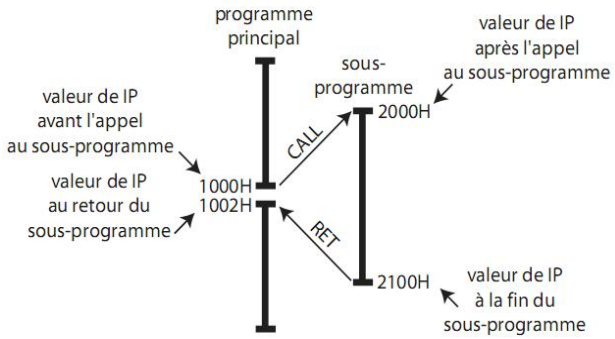
instruction	nom	condition
`JZ label`	Jump if Zero	saut si ZF = 1
`JNZ label`	Jump if Not Zero	saut si ZF = 0
`JE label`	Jump if Equal	saut si ZF = 1
`JNE label`	Jump if Not Equal	saut si ZF = 0
`JC label`	Jump if Carry	saut si CF = 1
`JNC label`	Jump if Not Carry	saut si CF = 0
`JS label`	Jump if Sign	saut si SF = 1
`JNS label`	Jump if Not Sign	saut si SF = 0
`JO label`	Jump if Overflow	saut si OF = 1
`JNO label`	Jump if Not Overflow	saut si OF = 0
`JP label`	Jump if Parity	saut si PF = 1
`JNP label`	Jump if Not Parity	saut si PF = 0

Remarque : Il existe un autre type de saut conditionnel, les sauts arithmétiques. Ils suivent en général l'instruction de comparaison : CMP opérande1, opérande2

condition	nombres signés	nombres non signés
=	`JEQ label`	`JEQ label`
>	`JG label`	`JA label`
<	`JL label`	`JB label`
≠	`JNE label`	`JNE label`

2.5.3. Appel de sous-programmes

Afin d'éviter d'écrire la même séquence d'instructions plusieurs fois dans un programme, on rédige la séquence une seule fois en lui attribuant un nom (au choix) et on l'appelle lorsqu'on en a besoin. Le programme appelant est le programme principal. La séquence appelée est un sous-programme ou procédure.



2.6. Méthodes de programmation

2.6.1. Etapes de la réalisation d'un programme

- Définir le problème à résoudre : que faut-il faire exactement ?
- Déterminer des algorithmes, des organigrammes : comment faire ? Par quoi commencer, puis poursuivre ?
- Rédiger le programme (code source) :
    - Utilisation du jeu d'instructions (mnémoniques).
    - Création de documents explicatifs (documentation).
- Tester le programme en réel.
- Corriger les erreurs (bugs) éventuelles : déboguer le programme puis refaire des tests jusqu'à obtention d'un programme fonctionnant de manière satisfaisante.

2.6.2. Langage machine et assembleur

- Langage machine : est un code binaire correspondant aux instructions.
- Assembleur : est un logiciel de traduction du code source écrit en langage assembleur (mnémoniques).

2.6.3. Réalisation pratique d'un programme

- Rédaction du code source en assembleur à l'aide d'un éditeur (logiciel de traitement de texte ASCII) :
    - Edit sous MS-DOS.
    - Notepad (bloc-note) sous Windows.
    - Assemblage du code source (traduction des instructions en codes binaires) avec un assembleur :
        - MASM de Microsoft.
        - TASM de Borland.

2.6.4. Structure d'un fichier source en assembleur

Pour faciliter la lisibilité du code source en assembleur, on le rédige sous la forme suivante :

```
labels          instructions        commentaires
label1 :        mov ax,60H          ; ceci est un commentaire ...
    ⋮               ⋮                   ⋮
sous_prog1      proc near           ; sous-programme
    ⋮               ⋮                   ⋮
sous_prog1      endp
    ⋮               ⋮                   ⋮
```

2.6.4.1. Directives pour l'assembleur

- Origine du programme en mémoire : ***Syntaxe*** **:** ORG offset. ***Exemple*** **:** org 1000H.
- Définitions de constantes : ***Syntaxe*** **:** nom_constante EQU valeur. ***Exemple*** **:** escape equ 1BH.
- Réservation de cases mémoires : ***Syntaxe*** **:** nom_variable1 DB valeur initiale ou nom_variable DW valeur initiale.

Exemples **:**

nombre1 db 25

nombre2 dw?; pas de valeur initiale

buffer db100 dup(?); réservation d'une zone mémoire de 100 octets.

3. Exercices de compréhension

Exercice 1

Ecrire un programme assembleur qui permet stocker la valeur 02BCH dans la case mémoire d'adresse 0200:0000

Exercice 2

Ecrire un programme assembleur qui permet copie le contenu de la case mémoire d'adresse 0200:0000 vers la case mémoire 0100:0000.

Exercice 3

Soit le programme assembleur suivant :

```
MOV AL, 0FFH    ; Charger 0xFF dans AL
SHL AL, 1       ; Décalage logique à gauche de AL d'un bit
SHR AL, 1       ; Décalage logique à droite de AL d'un bit
MOV AL, 088H    ; Charger 0x88 dans AL
MOV CL, 03H     ; Charger 0x03 dans CL
SAR AL, CL      ; Décalage arithmétique à droite de AL par CL bits
MOV AL, 080H    ; Charger 0x80 dans AL
ROL AL, 1       ; Rotation à gauche de AL d'un bit
ROR AL, 1       ; Rotation à droite de AL d'un bit
```

MOV AL, 080H ; Charger 0x80 dans AL

RCL AL, 1 ; Rotation à gauche avec retenue de AL d'un bit

RCR AL, CL ; Rotation à droite avec retenue de AL par CL bits

HLT ; Arrêter le processeur

- Insérer le programme ci-dessus depuis l'adresse offset 0000H et CS=0200H
- Exécuter le programme en mode step by step, en lit à chaque exécution les valeurs du registre Ax et du resistre d'état.

Exercice 4

Soit le programme assembleur suivant :

MOV AX, 0003H ; Charger 3 dans le registre AX

DEC AX ; Décrémenter AX (AX = AX - 1)

DEC AX ; Décrémenter AX (AX = AX - 1)

DEC AX ; Décrémenter AX (AX = AX - 1)

DEC AX ; Décrémenter AX (AX = AX - 1)

DEC AX ; Décrémenter AX (AX = AX - 1)

DEC AX ; Décrémenter AX (AX = AX - 1)

HLT ; Arrêter le processeur

- Inséerer le programme à l'adresse 100H:0H
- Exécuter le programme en mode step by step, en lit à chaque exécution les valeurs du registre Ax.

Exercice 5

Taper le programme suivant:

Programme Assembleur	Code machine
MOV AX,30H	B8 30 00
MOV BX,20H	BB 20 00
XCHG AX,BX	93
HLT	F4

Exercice 6

Taper le programme suivant:

Programme Assembleur	Code machine
MOV AX,0EH	B8 0E 00
PUSH AX	50
MOV Ax,12H	B8 12 00
POP AX	58
HLT	F4

Exécuter le programme pas à pas, en visualisant le contenu du registre SP avant et après l'exécution du programme et justifier vos réponses.

4. Manipulations

Manip 01

Ecrire un programme qui fait une addition immédiate de deux valeurs (chaque valeur est de 1 octet). La même chose pour la soustraction, en consultant le registre d'état.

Manip 02

Écrire un programme qui additionne deux nombres stockés en mémoire en utilisant le mode d'adressage direct.

Manip 03

Écrire un programme qui utilise le mode d'adressage indirect pour accéder à un tableau et calculer la somme de ses éléments.

Manip 04

Utiliser le mode d'adressage indirect pour accéder à une valeur dans un tableau de 5 octets.

Manip 05

En utilisant uniquement les deux registres AX et BX, écrire un programme en langage assembleur permettant d'évaluer l'expression arithmétique suivante :

$$X = \big((3 * B + A\%5)/(C + 5)\big) * (B * C)$$

Où :

- Les variables A, B, C, X sont déclarées dans le programme.
- A, B, C sont des octets non signés.

X est une variable sur quatre octets.

Manip 06

Evaluer l'expression suivante et stocker le résultat en mémoire à l'adresse 0300H:

$$Y = 2x^2 + 3x - 1$$

Tel que x est un nombre positif (1 octet) stocké en mémoire a l'dresse 0100H.

Manip 07

Écrire un programme qui compare deux nombres et utilise un saut conditionnel pour afficher un message selon le résultat de la comparaison.

TP 03 : Techniques de programmation 2

1. Objectifs

L'objectif de ce TP est de concevoir des programmes assembleurs faisant intervenir les structures de contrôles et itératives de microprocesseur 8086.

2. Rappels théoriques

2.1. Structures de contrôle

Les structures de contrôle dans le microprocesseur 8086 permettent de gérer le flux d'exécution des programmes. Les instructions de contrôle incluent :

- **Instructions de saut (Jumps)** : Ces instructions permettent de modifier le flux d'exécution en sautant à une autre adresse dans le programme. Les types de sauts incluent les sauts inconditionnels (par exemple, JMP) et conditionnels (par exemple, JE, JNE), qui dépendent des résultats d'opérations précédentes.
- **Instructions de boucle (Loops)** : Le 8086 utilise des instructions spécifiques pour gérer les boucles, comme LOOP, qui décrémente le registre CX et saute à une étiquette si CX n'est pas égal à zéro. Cela permet de répéter un bloc d'instructions un nombre déterminé de fois.
- **Instructions de condition (Conditionals)** : Les instructions conditionnelles permettent d'exécuter des blocs de code en fonction de l'état des drapeaux (flags) du processeur, qui reflètent les résultats des opérations arithmétiques ou logiques. Par exemple, les instructions CMP (comparaison) et TEST (test logique) modifient les drapeaux, influençant ainsi les sauts conditionnels.

2.2. Structures itératives

Les structures itératives sont essentielles pour la répétition d'instructions. Dans le 8086, les boucles sont généralement mises en œuvre à l'aide de :

- **Boucles basées sur le registre CX** : Le registre CX est souvent utilisé pour compter le nombre d'itérations. L'instruction LOOP est utilisée pour répéter un bloc d'instructions tant que CX n'est pas égal à zéro.
- **Boucles avec des instructions de saut** : En combinant des instructions de saut avec des comparaisons, il est possible de créer des boucles plus complexes. Par exemple, une boucle peut continuer jusqu'à ce qu'une certaine condition soit remplie, en utilisant des instructions comme JZ (jump if zero) ou JNZ (jump if not zero).

3. Manipulations

Manip 01

Charger les registres AX et BX par des valeurs, puis comparer ces deux valeurs :

- Si AX > BX, charger le registre CX par la valeur `1`.
- Si non, charger `0` dans le registre `CX`.

Manip 02

Définir un tableau de 6 octets en mémoire. Ecrire un programme qui permet de rechercher la valeur `8`. Si la valeur est trouvée, stocker son index dans `BX`. Sinon, charger `-1` dans `BX`.

Manip 03

Ecrire un programme assembleur qui permet d'inverser les éléments du tableau précédent.

Manip 04

Écrire un programme qui utilise une boucle pour remplir un tableau avec les carrés des nombres de 1 à 4 en utilisant le mode d'adressage basé sur l'index.

Manip 05

Ecrire un programme assembleur qui ajoute six fois au registre AX la valeur 5.

Manip 06

Ecrire un programme assembleur qui permet de calculer :

- La somme des nombres entiers pairs (de 0 à 10) et stocker le résultat dans le registre BX.
- La somme des nombres entiers impairs (de 0 à10) et stocker le résultat dans le registre DX

Manip 07

Ecrire un programme assembleur qui permet de compter les nombres nuls d'un tableau de longueur 5 octets, et débutant à partir de l'adresse 300H, le résultat sera placé à l'adresse 400H.

Manip 08

Ecrire un programme assembleur qui permet de déterminer le maximum d'un tableau d'octets mémoire de longueur 100H et débutant à partir de l'dresse 300H.

Manip 09

Ecrire un programme en assembleur qui permet de calculer le factoriel d'un nombre entier n de 8 bits stocké en mémoire à l'adresse 0210H

Manip 10

Ecrire un programme en assembleur qui permet de calculer :

- La somme des nombres entiers pairs (de 0 à 10) et stocker le résultat dans le registre BX.

- La somme des nombres entiers impairs (de 0 à10) et stocker le résultat dans le registre DX

Manip 11

Ecrire un programme en assembleur 8086, qui permet de compter les nombres nuls dans un tableau d'octets mémoire de longueur 0Ah et débutant à l'adresse [300h], le résultat sera placé à l'adresse [400h].

Manip 12

Ecrire un programme en assembleur 8086, qui permet de déterminer le maximum dans un tableau d'octets mémoire de longueur 10h et débutant à l'adresse [300h], le résultat sera placé à l'adresse [400h]

Manip 13

Ecrire un programme qui permet de trier par ordre croissant un tableau de longueur 0Fh débutant à l'adresse [300h]

Manip 14

Modifier le programme précédent pour que le tableau sera trié dans l'ordre décroissant.

Manip 15

Ecrire un programme qui indique la présence de la chaîne de caractères ‘M1ELN’ qui débute à l’adresse [300h], dans un tableau de N = 96 éléments qui débute à l’adresse [200h], (utiliser deux procédures ‘AFICH_OK’ et ‘AFFICH_NK’ pour afficher le résultat).

TP 04 : Programmation de l'interface parallèle 8255

1. Objectifs

L'objectif de ce TP est de se familiariser avec l'utilisation de l'interface de communication parallèle 8255 intégrée au système MTS-86C, et d'apprendre à programmer l'interface parallèle 8255 du kit MTS-86C pour contrôler et interagir avec des périphériques externes (LEDs, boutons poussoirs, etc.).

2. Présentation de l'interface parallèle :

Les échanges entre le microprocesseur et son environnement externe (clavier, écran, imprimante, …) se font à l'aide des circuits d'interfaces. Ces échanges peuvent être effectués en parallèle ou en série. Dans le cadre de ce TP, nous allons utiliser la liaison parallèle de la famille « Intel » : il s'agit du PPI (Programmable Peripheral Interface) 8255.

2.1. PPI 8255

Le PPI 8255 est une interface programmable d'entrée/sortie destiné à l'utilisation avec les microprocesseurs Intel. Elle possède 24 pins d'entrées/sorties programmable par deux groupes de 12 pins chacun et elle peut être utilisée en 3 modes.

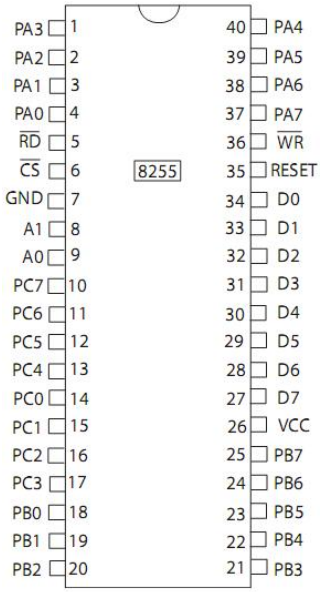


Figure.1 : Brochage du 8255.

Dans le MODE 0, chaque groupe de 12 pins peut être programmé par lot de 4 broches en entrée ou en sortie. Dans le MODE 1, chaque groupe peut être programmé de sorte que 8 lignes soient configurées en entrées ou en sorties. Le MODE 2, est un mode bidirectionnel où 8 lignes sont utilisées comme bus bidirectionnel.

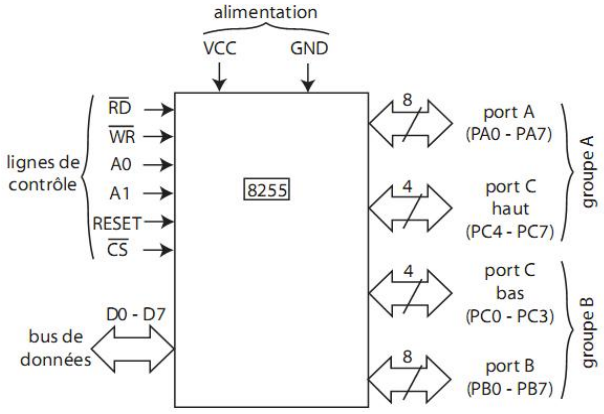


Figure.2 : Schéma fonctionnel du 8255.

2.1.1. Fonctions des broches

- Data Bus Buffer : Ce bus bidirectionnel à 3 états est utilisé pour interfacer le 8255 au bus du système. Les données sont transmises et reçues via le buffer suivant l'instruction de « input » ou de « output » générée par le microprocesseur. Le Control Word et les informations de configuration sont aussi transmis via ce bus.
- Read/Write et Control logic : La fonction de ce bloc est de gérer tous les transferts, internes et externes, des données, de contrôle ou du Status Words.
- CS : Cette broche permet la communication entre le 8255 et le CPU.
- RD : Cette broche permet au 8255 de transférer les données ou les informations de configurations au CPU via le data bus.
- WR : permet au microprocesseur d'envoyer des données ou le Control Words vers le 8255.
- A_0, A_1 : Ces broches d'entrées contrôlent la sélection des ports (A, B ou C) ou du registre du Control Word.

A1	A0	RD	WR	CS	Opérations d'Input (Lire)
0	0	0	1	0	Port A->Data Bus
0	1	0	1	0	Port B->Data Bus
1	0	0	1	0	Port C->Data Bus
					Opérations d'Output (Ecrire)
0	0	1	0	0	Data Bus -> Port A
0	1	1	0	0	Data Bus -> Port B
1	0	1	0	0	Data Bus -> Port C
1	1	1	0	0	Data Bus -> Control
					Fonctions de désactivation
X	X	X	X	1	Data Bus -> 3 états
1	1	0	1	0	Condition illégale
X	X	1	1	0	Data Bus -> 3 états

2.1.2. Le contrôle des groupes

Le CPU envoie un Control Word vers le 8255 pour configurer les ports. Le Control Word contient

des informations comme : le mode, le bit set, le bit reset, ... etc. Chaque bloc de contrôle (Groupe A et Groupe B) accepte les commandes depuis le Read/Write Control logic, et reçoit le Control Word depuis le bus interne des données et envoie les commandes au port qui lui est associé.
Le Groupe A de Contrôle ---> Port A et quartet du Port C (de poids fort CP4 à CP7).
Le Groupe B de Contrôle ---> Port B et quartet du Port C (de poids faible CP0-CP3].

2.1.3. Sélection des modes

Il existe trois modes d'utilisation de l'interface 8255 : Mode 0, Mode 1 et Mode 2. Quand l'entrée RESET est à l'état haut, tous les ports sont configurés en entrée. Si l'entrée RESET est désactivée, l'interface 8255 garde son état (en entrée) sans aucune initialisation préalable. Les modes des ports A et B peuvent être configurés séparément. Cependant le port C est divisé en deux groupes configurés suivants les besoins des ports A et B. (voir annexe)

3. Manipulations

Le système MTS-86C dispose de trois interfaces parallèles programmable 8255. Les interfaces PPI-1 et PPI-2 du kit sont utilisées pour communiquer avec l'extérieur de la carte, alors que l'interface PPI-3 est utilisée pour l'expérimentation aussi et elle est connectée aux diodes L_i et aux boutons poussoirs SW_i du kit MTS-86C, suivant les schémas électroniques disponible en annexe. D'après ces schémas (annexe), le port A de PPI-3 est connecté aux boutons poussoirs SW et le port B de PPI-3 est connecté aux diodes via les circuits 74LS240.

3.1. Exemple de programmation de l'interface parallèle

Supposons que l'on veut écrire un programme en assembleur qui nous permet d'allumer la diode LED_0 en appuyant sur le bouton poussoir SW_0 correspondant. Le programme s'écrit alors comme suit :

```
CNT3 EQU 3FD6H ; Assignation de l'adresse du control word du PPI-3 à la variable CNT3.
APORT3 EQU 3FD0H ; Assignation de l'adresse du port A du PPI-3 à la variable APORT3.
BPORT3 EQU 3FD2H ; Assignation de l'adresse du port B du PPI-3 à la variable BPORT3.
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 0
START : MOV SP,4000H ; Configuration de la pile
MOV AL, 90H ; Configuration du control word du 8255 (port A: Input, port B et port C: output).
MOV DX, CNT3 ; Copie l'adresse du control word dans le registre DX.
OUT DX, AL ; Configuration des ports du 8255 (écriture dans le registre du control word).
J1: MOV DX, APORT3 ; Copie l'adresse du port A dans le registre DX.
```

IN AL, DX ; Copie le contenu du port A dans le registre AL.

NOT AL ; Inverser le signal

MOV DX, BPORT3 ; Copie l'adresse du port B dans le registre DX.

OUT DX, AL ; Copie le contenu du registre AL dans le port B.

JMP J1 ; Reboucler vers J1

CODE ENDS

END START

- Tester le bon fonctionnement du programme précédant sur la carte MTS-86C.

3.2. Programmation de l'interface 8255

Manip 1

Ecrire des programmes en assembleur permettant :

- D'allumer toutes les LEDS.
- D'afficher le nombre hexadécimal 95 sur les huit Leds.
- D'alterner l'allumage des diodes de droite à gauche en continu.
- Modifier le programme précédent pour réaliser l'alternance cinq (5) fois seulement.

Manip 2

Soit à réaliser un chenillard sur le port B de la PPI 8255. Le principe est le suivant :

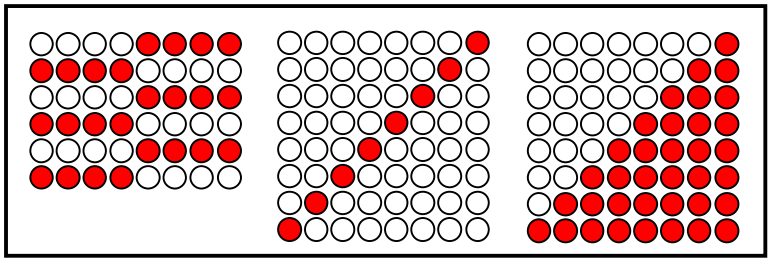
Etape1	0	0	0	0	0	0	0	0
Etape2	1	0	0	0	0	0	0	0
Etape3	0	1	0	0	0	0	0	0
Etape4	0	0	1	0	0	0	0	0
Etape5	0	0	0	1	0	0	0	0
Etape6	0	0	0	0	1	0	0	0
Etape7	0	0	0	0	0	1	0	0
Etape8	0	0	0	0	0	0	1	0
Etape9	0	0	0	0	0	0	0	1

Le "1" représente une Led éteinte et le "0" une Led allumée.

- ✓ Ecrire ce programme, en utilisant l'instruction de décalage adéquate.
- ✓ Modifier le programme précédent pour n'exécuter ce défilement que trois fois.
- ✓ Conclure.

Manip 3

Ecrire un programme assembleur qui permet de générer la séquence de la figure ci-dessous de manière répétée.



Manip 4

Ecrire un programme assembleur qui permet de commander un moteur pas à pas, utilisant deux boutons poussoirs permettant de commander le sens de rotation.

Manip 5

Refaire manipulation 4 pour commander maintenant un moteur à courant continu.

TP 05 : Programmation de l'interface série 8251

1. Objectifs

L'objectif de cette manipulation est d'initier l'étudiant aux principes fondamentaux de la communication série et de la programmation du contrôleur de communication universel programmable 8251 (USART) du kit MTS-86C. L'étudiant devra développer des programmes qui permet l'interaction avec le 8251, dans le but de :

- Configurer le 8251 (débit en bauds, format des données, parité).
- Transmettre des caractères depuis le kit MTS-86C vers un PC.
- Recevoir des caractères envoyés par le PC.

2. Matériel requis

- Le kit MTS-86C, basé sur le microprocesseur 8086.
- Documentation technique de l'interface série 8251 UART.
- Câble série (RS232) pour connecter le kit au PC.
- Logiciel Hyper terminal pour tester la communication série.

3. Rappel théorique

L'interface série 8251 USART (Universal Synchronous and Asynchronous Receiver-Transmitter) est un circuit intégré utilisé pour gérer la communication série entre un microprocesseur et des périphériques externes. Elle prend en charge les communications synchrones et asynchrones, et offre une flexibilité dans la configuration des formats de données et des protocoles de transmission.

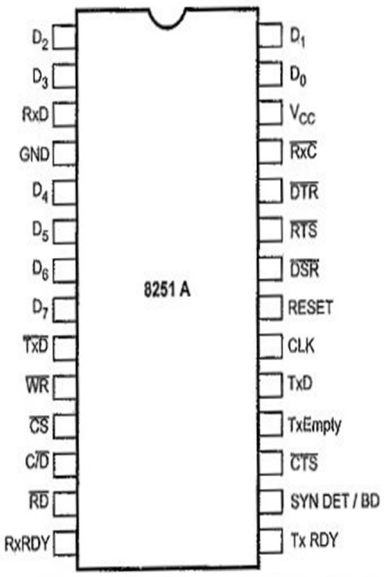


Figure.1 : Brochage de 8251.

Il convertit les données parallèles (issues du microprocesseur) en données série pour la transmission, et inversement pour la réception. Il est utilisé dans des applications nécessitant une communication série, comme :

- Transmission de données entre le microprocesseur et un PC.
- Interfaces avec des capteurs ou des actionneurs.
- Communication entre deux microprocesseurs via RS232.

3. 1. Modes de fonctionnement

Le 8251A dispose de 5 registres divisés en deux groupes : **registres de contrôles** et **registres de données** (sélectionnés par la ligne C/D) :

- Le groupe contrôle comporte le **registre de mode**, le **registre de contrôle** et **le registre d'état**. Ces registres sont sélectionnés lorsque la ligne C/D= 1.
- Le groupe de données comporte le **registre Buffer Data In** et **Buffer Data Out**. Ils sont sélectionnés lorsque la ligne C/D= 0.

Le Tableau ci-dessous présente la sélection des registres.

CS	C/D	RD	WR	Sélection
1	X	X	X	8251A n'est pas sélectionné
0	1	0	1	Lecture du registre d'état
0	1	1	0	Ecriture dans le registre de mode ou de contrôle.
0	0	0	1	Lecture du registre Buffer Data In
0	0	1	0	Ecriture dans le registre Buffer Data Out

4. Manipulations

Manip 01

Utilisez la console HyperTerminal pour téléverser le programme ci-dessous sur la mémoire du kit MTS-86C.

```
D_PORT          EQU   0FFF0H
CS_PORT         EQU   0FFF2H
CNT3            EQU        3FD6H
BPORT3          EQU    3FD2H
CODE      SEGMENT
          ASSUME     CS:CODE
          ORG   0
          MOV   AL,90H
```

```
        MOV  DX,CNT3
        OUT  DX,AL
        MOV  AL,00H
        MOV  DX,CS_PORT
        OUT  DX,AL
        OUT  DX,AL
        OUT  DX,AL
        MOV  AL,40H
        OUT  DX,AL
        MOV  AL,4EH
        OUT  DX,AL
        MOV  AL,27H
        OUT  DX,AL
J1:     CALL IN_RS
        CMP  AH,0DH
        JE   J3
J2:     CALL OUT_RS
        JMP  J1
J3:     CALL OUT_RS
        MOV  AH,0AH
        JMP  J2
IN_RS:      MOV  DX,CS_PORT
        IN   AL,DX
        AND  AL,2
        JZ   IN_RS
        MOV  DX,D_PORT
        IN   AL,DX
        MOV  DX,BPORT3
        OUT  DX,AL
        MOV  AH,AL
        RET
OUT_RS:     MOV  DX,CS_PORT
        IN   AL,DX
        AND  AL,1
        JZ   OUT_RS
        MOV  DX,D_PORT
        MOV  AL,AH
        OUT  DX,AL
        RET
CODE    ENDS
        END
```

- Le programme précédent permet d'envoyer des caractères via le 8251 du kit MTS-86C. Exécuter le programme. Que se passe-t-il ? Pourquoi ?

- Identifier la valeur de l'instruction de commande. En déduire l'état de configuration actuel du port série.
- Identifier la valeur de l'instruction de mode. Déduire alors le mode de communication, la longueur du caractère, la parité (activée ou non activée), le bit de parité et le facteur de débit.
- Commenter les routines IN_RS et OUT_RS et expliquer l'utilité des instructions AND AL, 2 dans IN_RS et AND AL, 1 dans OUT_RS.

Manip 02

Soit le programme assembleur suivant :

```
D_PORT          EQU   0FFD0H
CS_PORT         EQU   0FFD2H
CNT3            EQU        3FD6H
BPORT3          EQU    3FD2H
CODE      SEGMENT
          ASSUME     CS:CODE
          ORG   0
          MOV   AL,90H
          MOV   DX,CNT3
          OUT   DX,AL
          MOV   AL,00H
          MOV   DX,CS_PORT
          OUT   DX,AL
          OUT   DX,AL
          OUT   DX,AL
          MOV   AL,40H
          OUT   DX,AL
          MOV   AL,4EH
          OUT   DX,AL
          MOV   AL,27H
          OUT   DX,AL
J1:       CALL  IN_RS
          CMP   AH,0DH
          JE    J3
J2:       CALL  OUT_RS
          JMP   J1
J3:       CALL  OUT_RS
          MOV   AH,0AH
          JMP   J2
IN_RS:          MOV   DX,CS_PORT
          IN    AL,DX
          AND   AL,2
```

```
                JZ      IN_RS
                MOV     DX,D_PORT
                IN      AL,DX
                MOV     DX,BPORT3
                OUT     DX,AL
                MOV     AH,AL
                RET
OUT_RS:         MOV     DX,CS_PORT
                IN      AL,DX
                AND     AL,1
                JZ      OUT_RS
                MOV     DX,D_PORT
                MOV     AL,AH
                OUT     DX,AL
                RET
CODE            ENDS
                END
```

- Identifier la valeur de l'instruction de commande. En déduire l'état de configuration actuel du port série.
- Identifier la valeur de l'instruction de mode. Déduire alors le mode de communication, la longueur du caractère, la parité (activée ou non activée), le bit de parité et le facteur de débit.

TP 01 : Prise en main de la carte C6713DSK

1. Objectifs

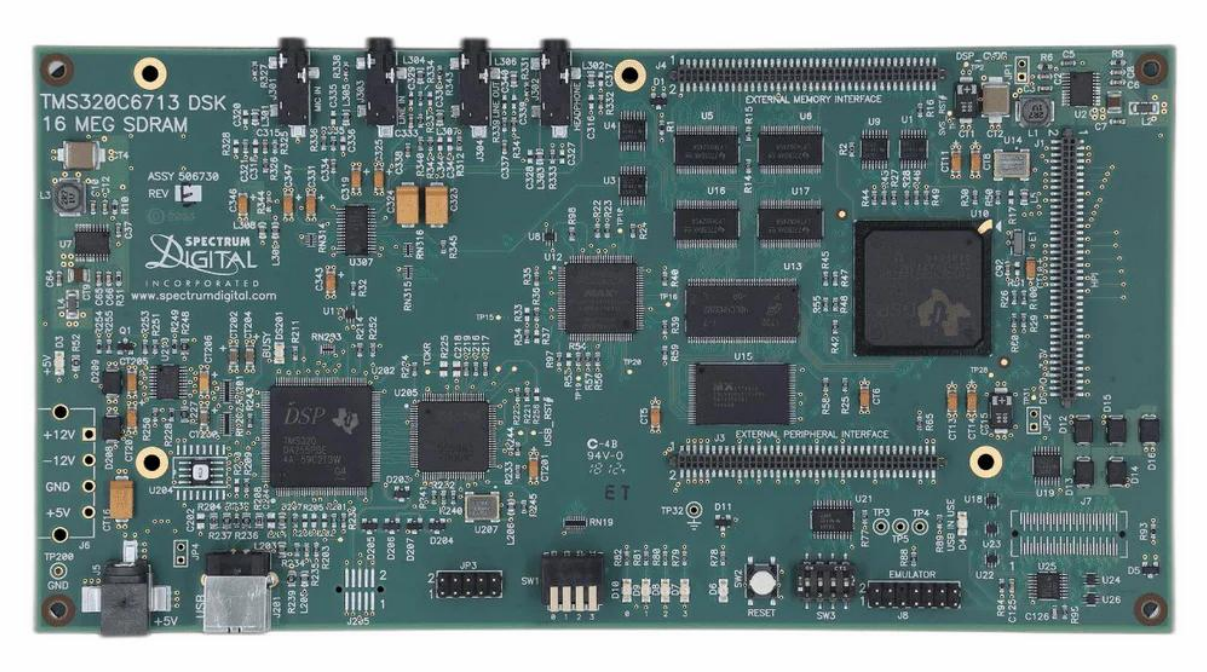
Pour réaliser et concevoir un programme, les outils suivants sont nécessaires :

Le kit DSK (DSP Starter Kit) comprend :

- Code Composer Studio (CCS) fournit un environnement de développement intégré (IDE), qui regroupe le compilateur C, l'assembleur, l'éditeur de liens et le débogueur.
- La carte illustrée à la Figure.1 contient le processeur de signaux numériques à virgule flottante (C6713) TMS320C6713.
- Un câble de bus synchrone universel (USB) qui relie la carte DSK à un PC.
- Une alimentation de 5 V pour la carte DSK.

2. Introduction

Le DSK est un ensemble puissant, doté des outils matériels et logiciels nécessaires au traitement des signaux en temps réel. Il s'agit d'un système DSP complet. La carte DSK C6713 recevant les consignes est une plate-forme de développement de faible coût conçue par Texas Instruments (DSP Starter Kit) permettant une mise en œuvre rapide pour des applications dédiées. Elle s'articule autour d'un microprocesseur (Digital Signal Processor) TMS320C6x.



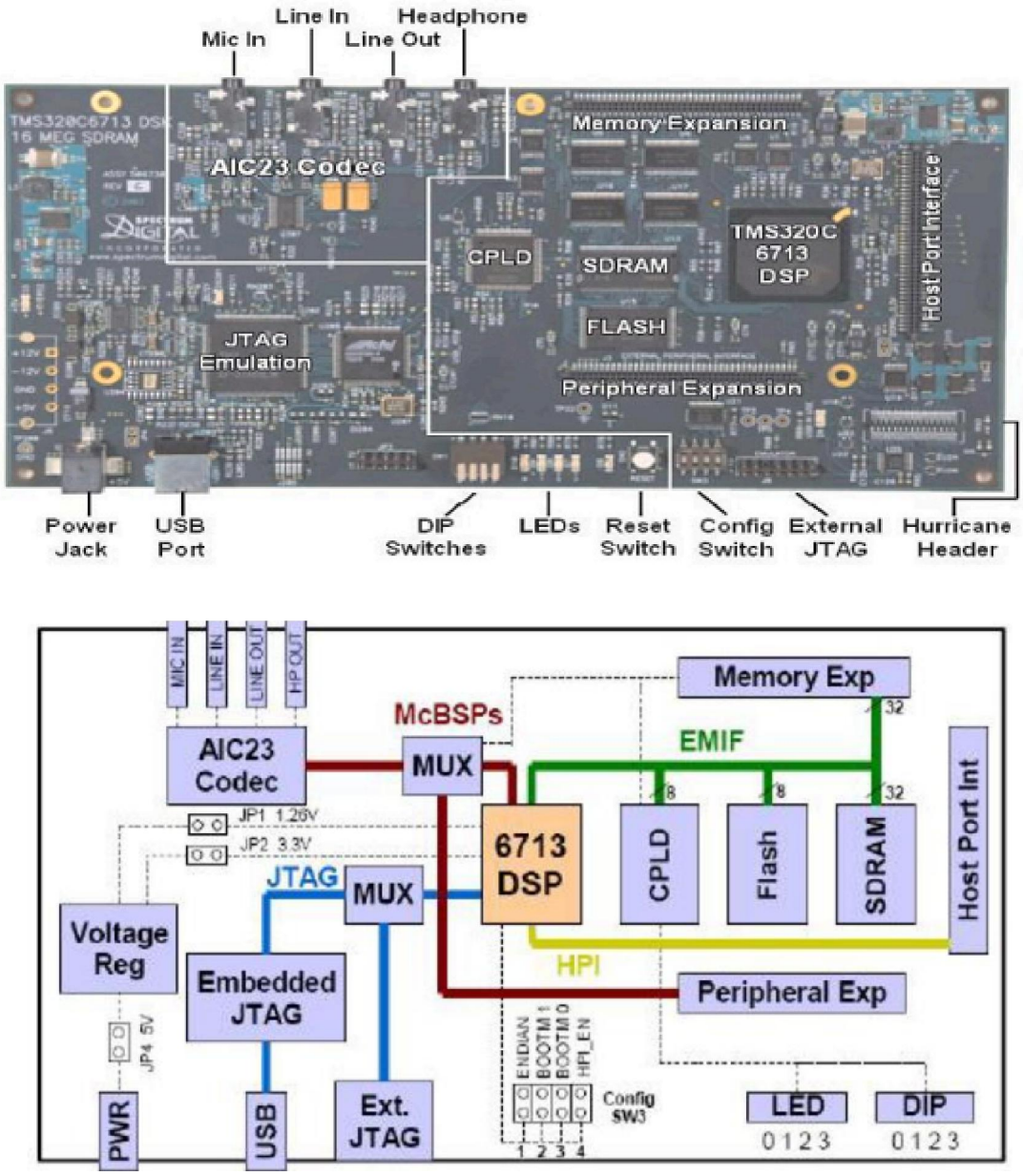


Figure.1 : Carte DSK à base de TMS320C6713 : (a) carte ; (b) diagramme

L'outil logiciel nécessaire pour générer des fichiers exécutables TMS320C6x s'appelle Code Composer Studio (CCS). Il intègre les utilitaires assembleur, éditeur de liens, compilateur, simulateur et débogueur. En l'absence d'une carte cible, qui permet d'exécuter un fichier exécutable sur un processeur C6x réel, le simulateur peut être utilisé pour vérifier la fonctionnalité du code en utilisant des données déjà stockées dans un fichier de données.

Cependant, lors de l'utilisation du simulateur, une routine de service d'interruption (ISR) ne peut pas être utilisée pour lire des échantillons du signal. Pour pouvoir traiter les signaux en temps réel sur un processeur C6x réel, un kit de démarrage DSP (DSK) ou une carte EVM (EValuation Module) est nécessaire pour le développement du code. L'équipement de test recommandé est

un générateur de fonctions, un oscilloscope, un microphone, un boom box et des câbles avec prises audio. Une carte DSK peut facilement être connectée à un PC hôte via son port parallèle ou USB.

L'interface du signal avec la carte DSK se fait via ses deux prises audios standard. Une carte EVM doit être installée dans un emplacement PCI pleine longueur à l'intérieur d'un hôte PC. L'interface du signal avec la carte EVM se fait via ses trois prises audios standard. Pour effectuer les travaux pratiques, la connaissance du langage C est supposée. La programmation de la plupart des processeurs DSP peut être effectuée en C ou en assembleur. Bien que l'écriture des programmes en C nécessite moins d'efforts, l'efficacité obtenue est normalement inférieure à celle des programmes écrits en assembleur. L'efficacité signifie avoir le moins d'instructions ou le moins de cycles d'instructions possible en utilisant au maximum les ressources de la puce. La Figure.2 montre les étapes à suivre pour passer d'un fichier source (extension .c pour C, .asm pour l'assembleurs et .sa pour l'assembleur linéaire) à un fichier exécutable (extension .out).

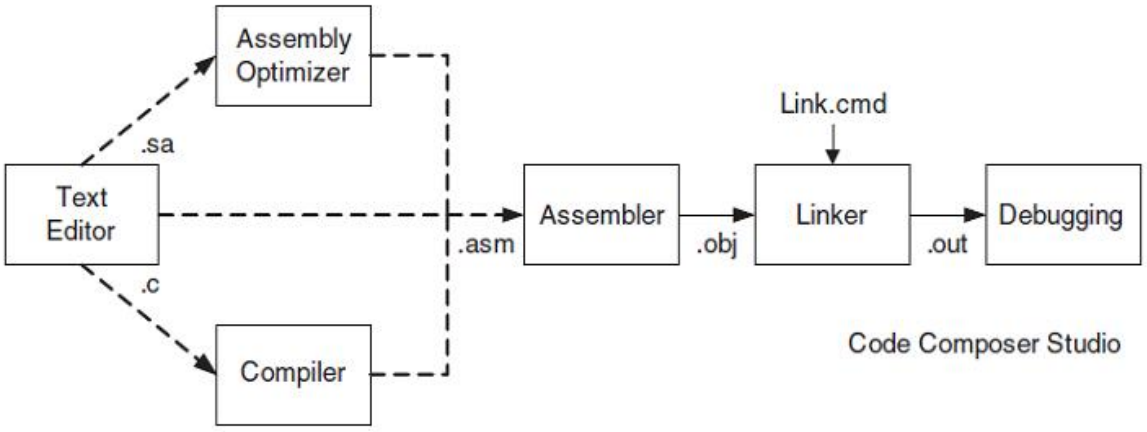


Figure. 2 : Outils logiciels du C6x

.c = fichier source C

.sa = fichier source d'assemblage linéaire

.asm = fichier source de l'assembleur

.obj = fichier objet

.out = fichier exécutable

.cmd = fichier de commande de l'éditeur de liens

L'exemple suivant :

void main()
{
y = DotP((int *) a, (int *) x, 40);
}
	.title "dot product"
	.def dotp
	.sect code
	dotp: .proc A4,B4,A6,B6,A8,B3

int DotP(int *m, int *n, short count)
{
int sum, i;
sum = 0;
for(i=0;i<count;i++)
sum += m[i] * n[i];
return(sum);
}
	.reg a, ai, b,bi,r,prod,sum,c,ci,i;
	MV A4,c
	MV B4,b
	MV A6,a
	MV B6,r
	MV A8,i
	loop: .trip 40
	LDH *a++, ai
	LDH *b++,bi
	MPY ai,bi,prod
	SHR prod,15,sum
	ADD ai,sum,ci
	STH ci, *c++
	[i] SUB i,1,i
	[i] B loop
	.endproc B3

L'assembleur est utilisé pour convertir un fichier d'assembleur en un fichier objet (extension .obj). L'optimiseur d'assembleur et le compilateur sont utilisés pour convertir, respectivement, un fichier d'assembleur linéaire et un fichier C en un fichier objet. L'éditeur de liens est utilisé pour combiner des fichiers d'objets, comme indiqué par le fichier de commande de l'éditeur de liens (extension .cmd), dans un fichier exécutable.

Toutes les étapes d'assembleur, de liaison, de compilation et de débogage ont été intégrées dans un environnement de développement intégré (IDE) appelé Code Composer Studio.

3. Cartes cibles C6x DSK / EVM

Le DSK est un ensemble puissant, doté des outils matériels et logiciels nécessaires au traitement des signaux en temps réel. Il s'agit d'un système DSP complet. Lors de la disponibilité d'une carte DSK ou EVM, un fichier exécutable peut être exécuté sur un processeur C6x réel. En l'absence de telles cartes, CCS peut être configuré pour simuler le processus d'exécution. La carte DSK C6713 est un système DSP de taille approximative de 5×8 pouces qui comprend une puce DSP C6713 fonctionnant à 225 MHz avec une mémoire de 4/4/256 Ko pour le cache de données L1D / cache de programme L1P / mémoire L2, respectivement, 8 Mo de la SDRAM intégrée (RAM dynamique synchrone), 512 Ko de mémoire flash. La carte DSK, comprend aussi un codec stéréo 32 bits TLV320AIC23 (AIC23) pour l'entrée et la sortie. Il utilise une technologie sigma-delta qui fournit une ADC (Analog Digital Converter) et un DAC (Digital Analog Converter). Il est connecté à une horloge système de 12 MHz. Des taux d'échantillonnage variables de 8 à 96 kHz peuvent être réglés facilement.

4. Programmation du TMS320DSK6713

La programmation du kit DSK pour effectuer une certaine tâche de traitement du signal peut se faire en code C via Code Composer Studio ou via MATLAB en utilisant Simulink. Dans les séances des TPs, nous utilisons les deux méthodes de programmation.

Tout projet créer par le compositeur de code doit contenir plusieurs fichiers de types différents. Ces types de fichiers sont énumérés ci-dessous.

- **file.pjt :** pour créer et construire un projet nommé file
- **file.c :** Programme source en C
- **file.asm :** programme source d'assemblage créer par l'utilisateur, par le compilateur C ou par l'optimiseur linéaire optimiseur linéaire.
- **file.sa :** programme source d'assemblage linéaire. L'optimiseur linéaire utilise file.sa en tant qu'entrée pour produire un programme d'assemblage file.asm
- **file.h :** fichier de support d'en-tête
- **file.lib :** fichier de bibliothèque, tel que la bibliothèque de support d'exécution filerts6700.lib
- **file.cmd :** fichier de commandes de l'éditeur de liens qui associe les sections à la mémoire
- **file.obj :** fichier objet créé par l'assembleur
- **file.out :** fichier exécutable créé par l'éditeur de liens pour être chargé et exécuté sur le processeur C6713
- **file.cdb :** fichier de configuration lors de l'utilisation du DSP/BIOS

Le CCS utilise des fichiers de support situés dans le dossier support (à l'exception des fichiers de bibliothèque). Les fichiers supports sont :

- **C6713dskinit.c :** contient des fonctions pour initialiser le DSK, le codec, les ports série, et pour les E/S.
- **C6713dskinit.h :** fichier d'en-tête avec des prototypes de fonctions. Des fonctions telles que celles utilisées pour sélectionner l'entrée micro au lieu de l'entrée ligne (par défaut), le gain d'entrée, etc.
- **C6713dsk.cmd :** exemple de fichier de commande de l'éditeur de liens.
- **Vectors_intr.asm :** version modifiée d'un fichier vectoriel fourni avec CCS pour gérer les interruptions. Douze interruptions, de INT4 à INT15, sont disponibles. Elles sont utilisées pour les programmes pilotés par les interruptions.
- **Vectors_poll.asm :** fichier vectoriel pour les programmes utilisant le polling.

- **rts6700.lib, dsk6713bsl.lib, csl6713.lib :** fichiers de bibliothèque d'exécution, de carte et de support de puce, respectivement. Ces fichiers sont inclus dans le CCS et se trouvent respectivement dans **C6000 \cgtools\lib/C6000\dsk6713\lib et c6000\bios\lib**.

TP 02 : Familiarisation avec le Code Composer Studio : Création de projets, outils de débogage, cible, EVM, simulateur.

1. Objectifs

Les objectifs de ce projet sont les suivants :

- Présenter aux étudiants le fonctionnement du kit DSK6713.
- Lancer un projet simple pour familiariser les étudiants avec l'environnement du compositeur de code.

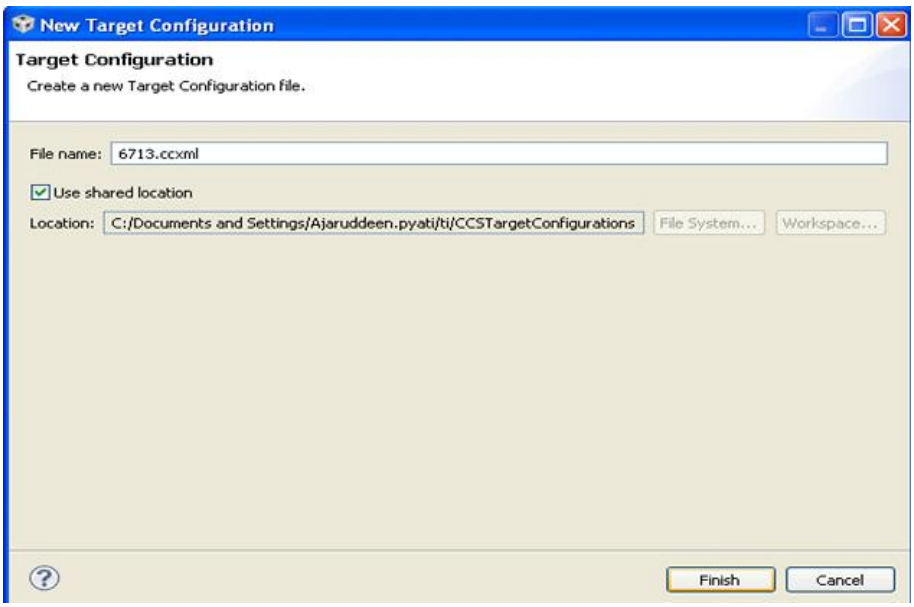
A la fin de la manipulation l'étudiant sera capable de créer des projets.

2. Procédure de travail avec CCS-V5 (utilisant TMS320C6713)

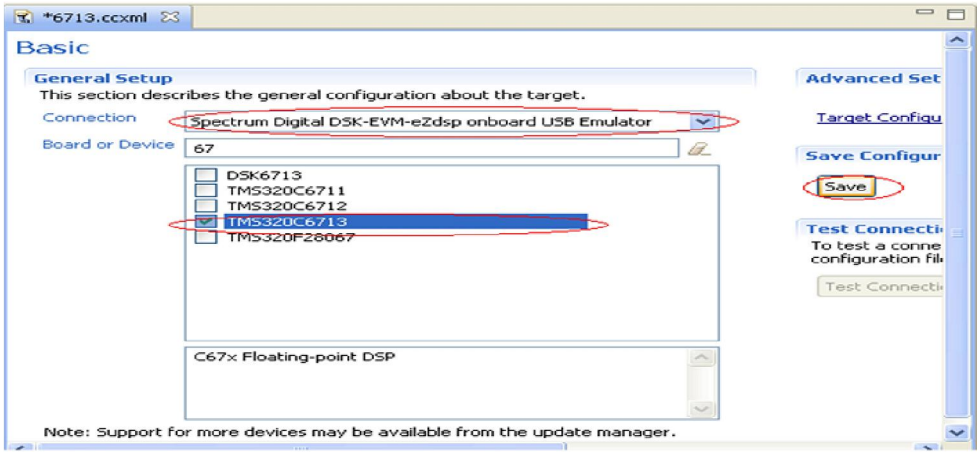
Étape 1 : Création d'une configuration cible

File→new→target configuration

Vous obtenez la fenêtre illustrée à la figure ci-dessous. Vous pouvez donner n'importe quel nom de cible avec l'extension « **.ccxml »** , puis cliquez sur finish.



Une fois que vous aurez cliqué sur finish, vous obtiendrez la fenêtre de configuration générale comme indiqué ci-dessous.

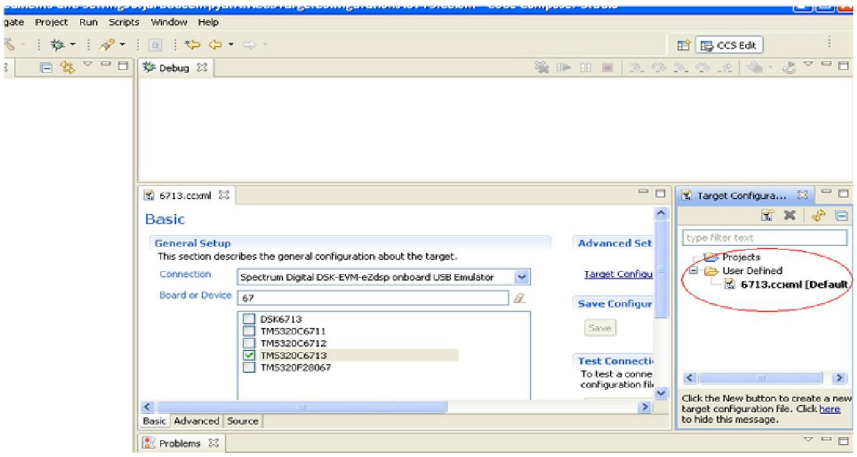


Dans la configuration générale, vous devez sélectionner l'appareil et la connexion comme indiqué dans la figure ci-dessus. Connexion : Emulateur USB embarqué Spectrum Digital DSK-EVM-eZdsp Carte ou dispositif : TMS320C6713.

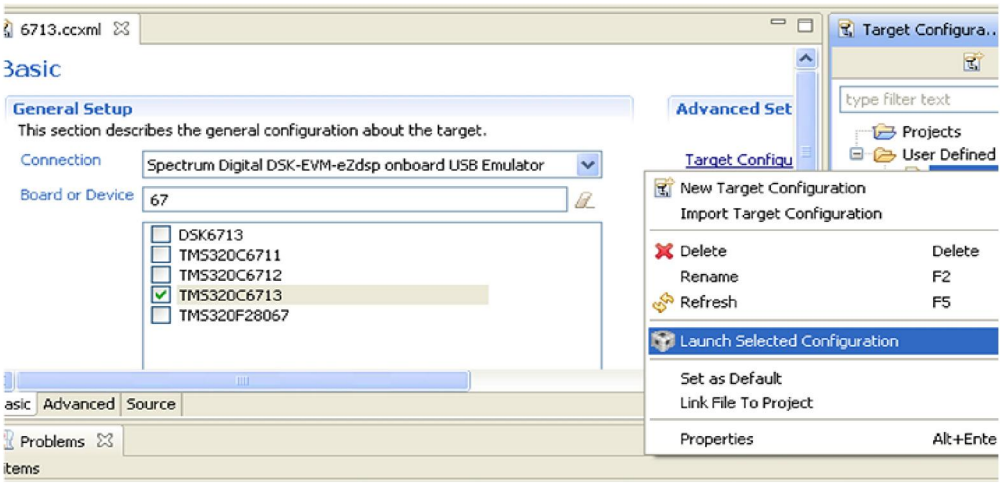
Étape 2 : Lancer la configuration de la cible

Allez à « View » dans la barre d'outils

View→target configuration, vous pouvez voir le nom de votre cible sous « user defined » comme indiqué dans la ci-dessous.

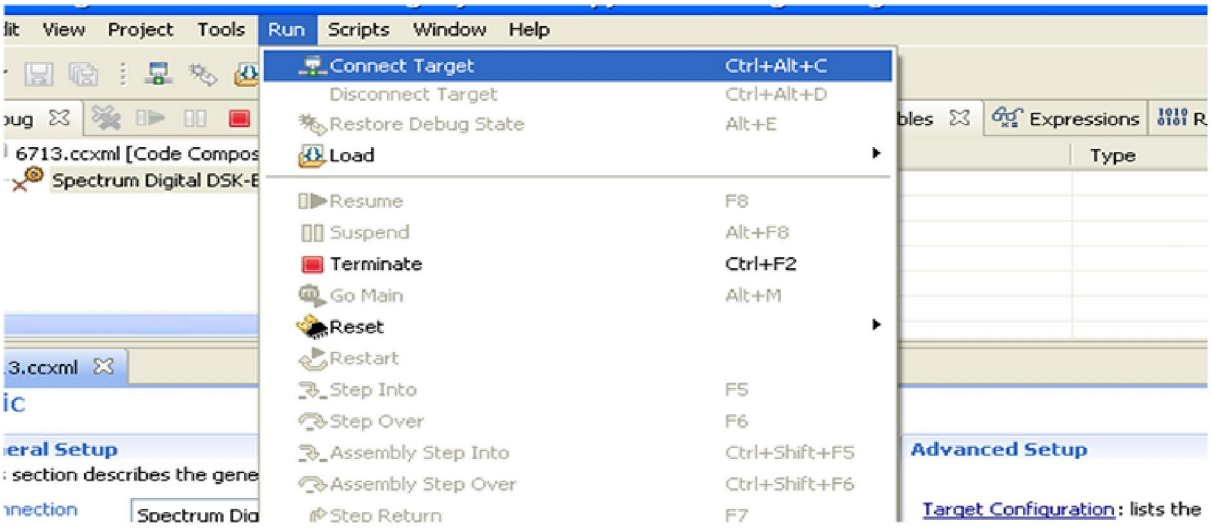


Ici, sous « user defined », il y a 6713.ccxml qui est ma cible. Cliquez ensuite sur SAVE. Faites un clic droit sur votre cible (6713.ccxml), puis lancez la configuration sélectionnée comme indiqué dans la figure ci-dessous.

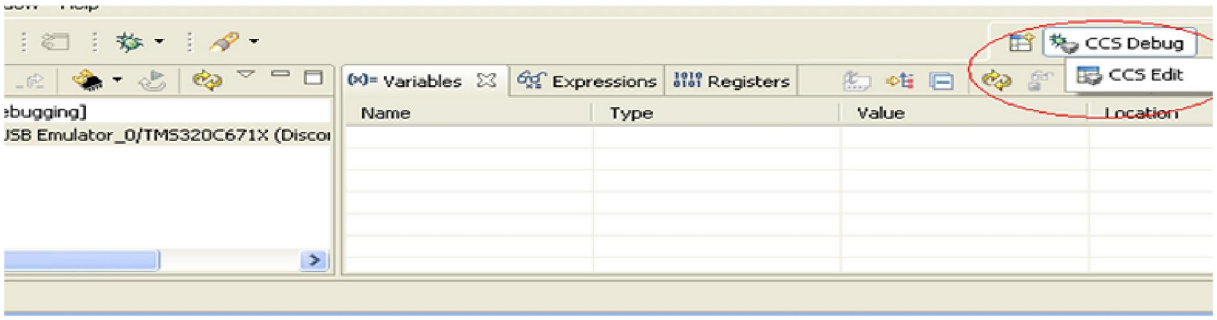


Étape 3 : Connexion de la cible

Allez sur Run→connect target comme indiqué ci-dessous.



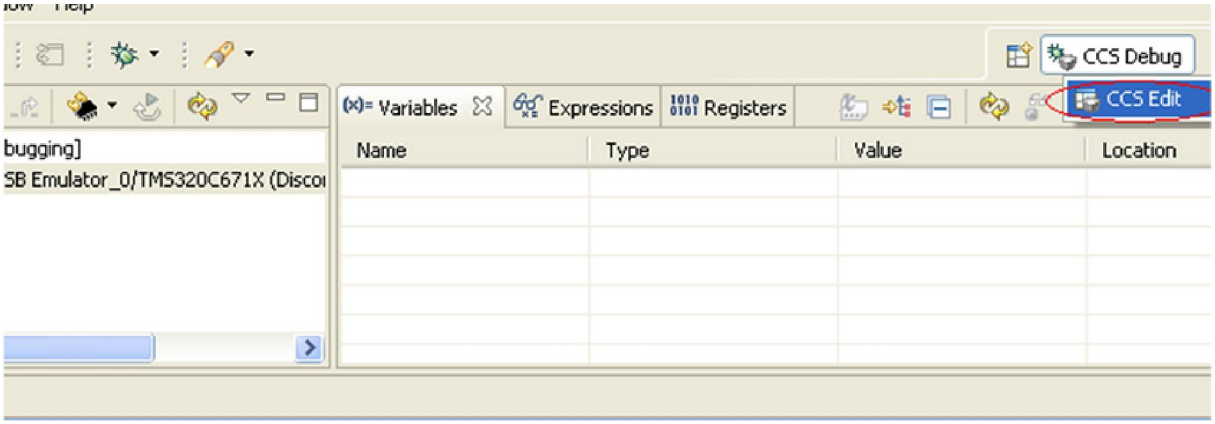
Note : dans la fenêtre CCS mail, il y a deux perspectives, l'une est CCS Debug et l'autre est CCS edit comme le montre la figure ci-dessous.



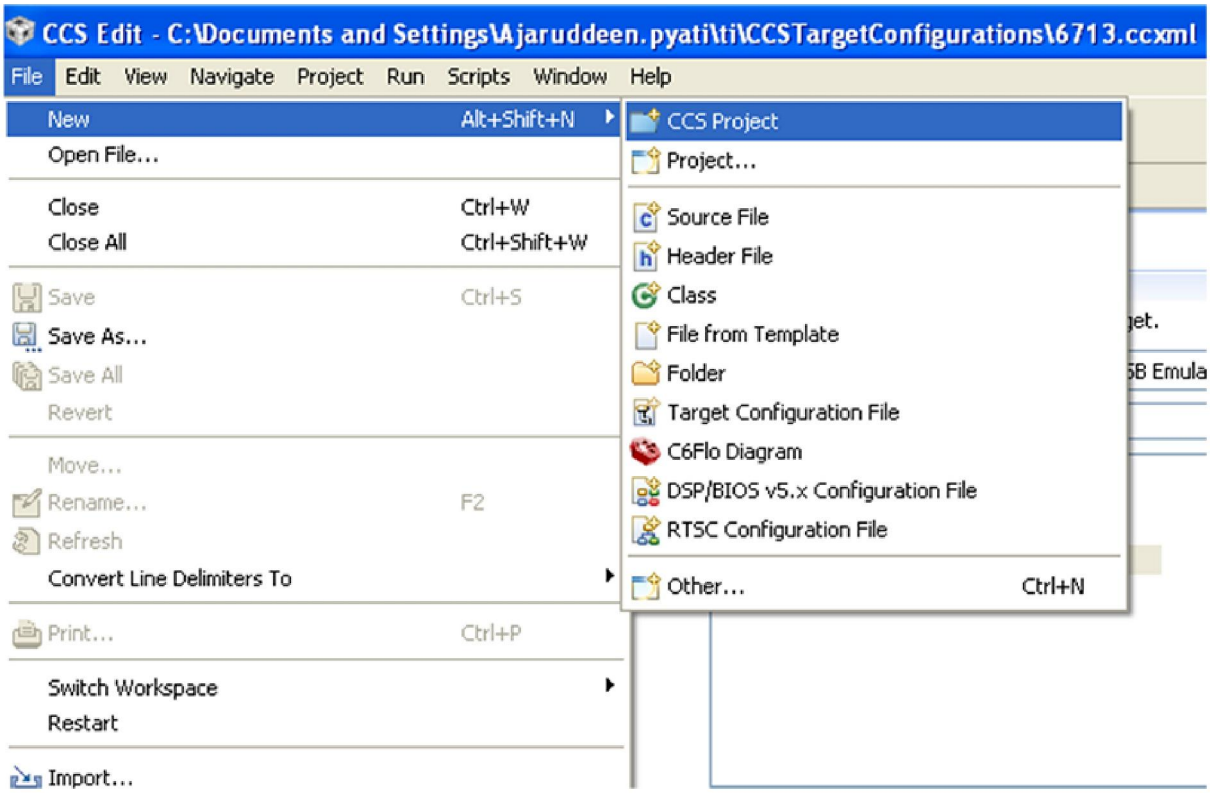
Dans CCS debug il y a toutes les options liées à la cible et dans CCS edit il y a toutes les options liées aux projets (source, lib etc.).

Étape 4 : Création d'un projet

Premièrement, cliquez sur « CCS edit » comme indiqué ci-dessous



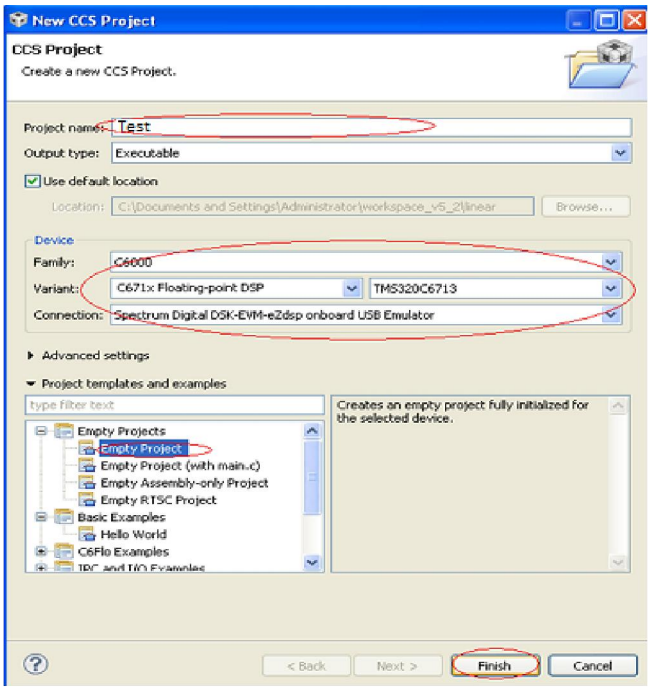
Ensuite, aller sur File→new→CCS Project comme indiqué dans la figure ci-dessous



Ensuite, apporter les modifications mentionnées.

- **Nom du projet** : n'importe quel nom (par exemple, j'ai donné Test)
- **Famille** : C6000
- **Variante** : C671xDSP à virgule flottante TMS320C6713
- **Connexion** : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator

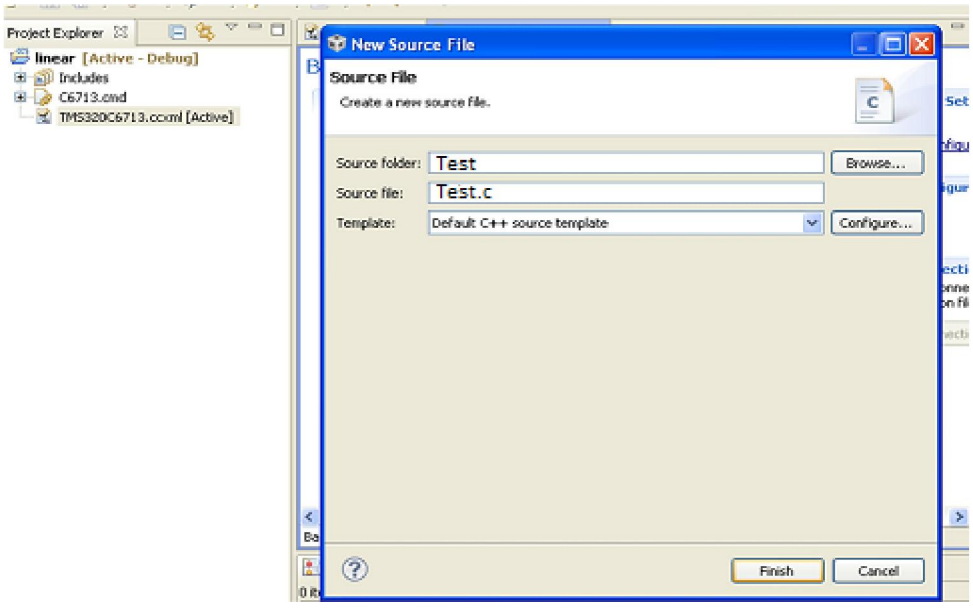
Sélectionnez ensuite un projet vide, puis cliquez sur finish.



Étape 5 : Créer un fichier source

File→new→source fichier

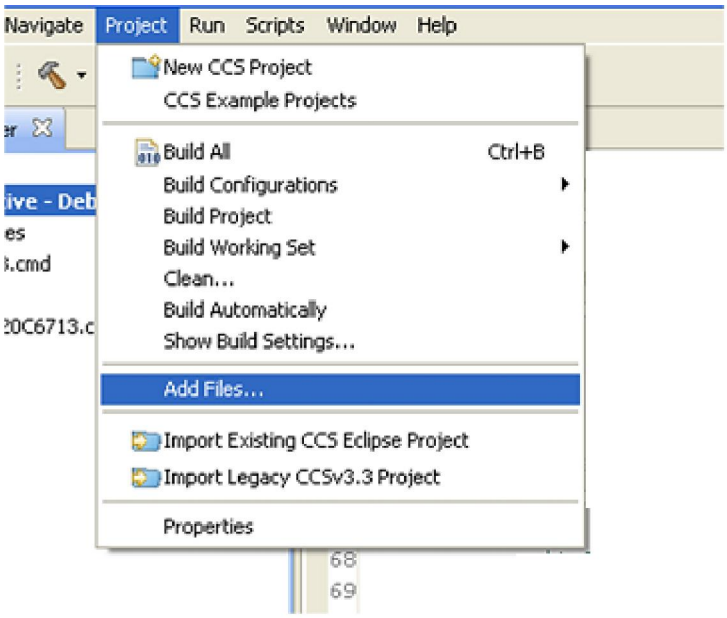
Fichier source : donnez un nom avec l'extension .c (ici j'ai donné Test.c).



Maintenant, écrivez un code sur l'espace de travail, puis allez sur file→save.

Étape 6 : Ajout d'un fichier de bibliothèque

Allez sur project→add files comme indiqué dans la figure ci-dessous.



Puis ajouter les fichiers de la bibliothèque bsl et csl. Ces fichiers se trouvent dans les chemins mentionnés ci-dessous.

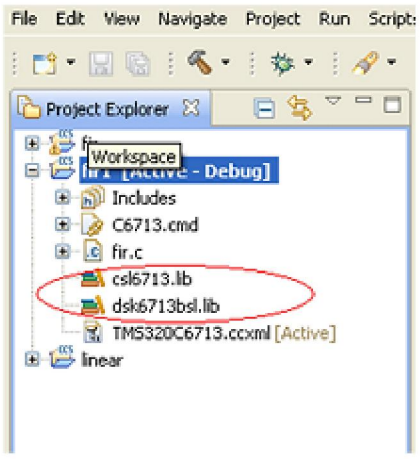
- BSL (fichier de bibliothèque de support de carte)

Chemin: C:\CCStudio_v3.1\C6000\dsk6713\lib\dsk6713bsl.lib

- CSL (Chip support library file)

Chemin d'accès : C:\CCStudio_v3.1\C6000\csl\lib\csl6713.lib

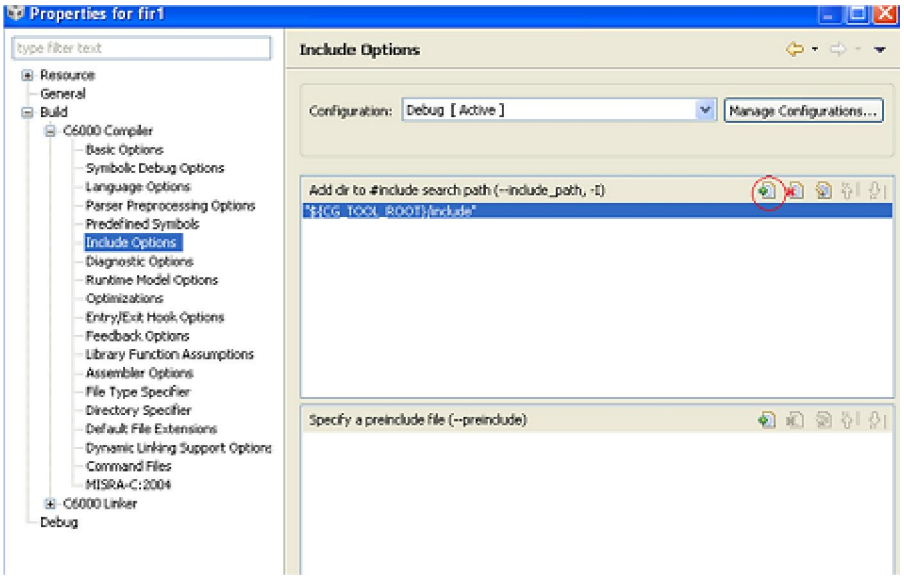
Vous pouvez voir que ces fichiers ont été ajoutés à votre projet comme le montre la figure ci-dessous.



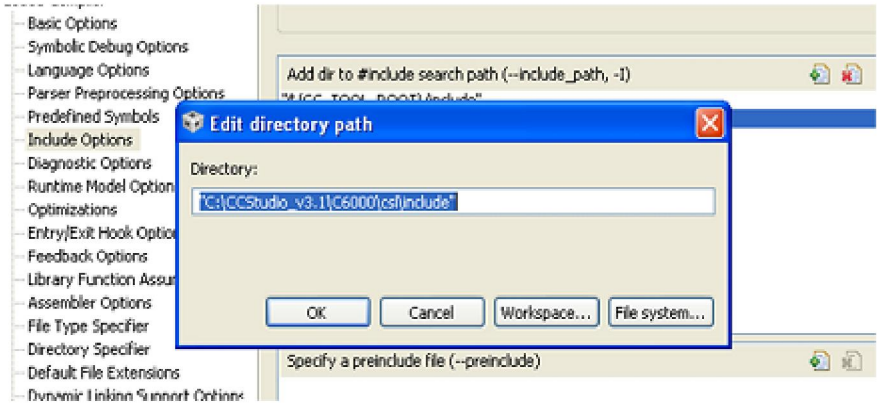
Étape 7 : Définition des propriétés de construction

Allez sur project→properties, sous Build→C6000 Compiler→include options now

Cliquez sur ajouter comme indiqué dans la figure ci-dessous

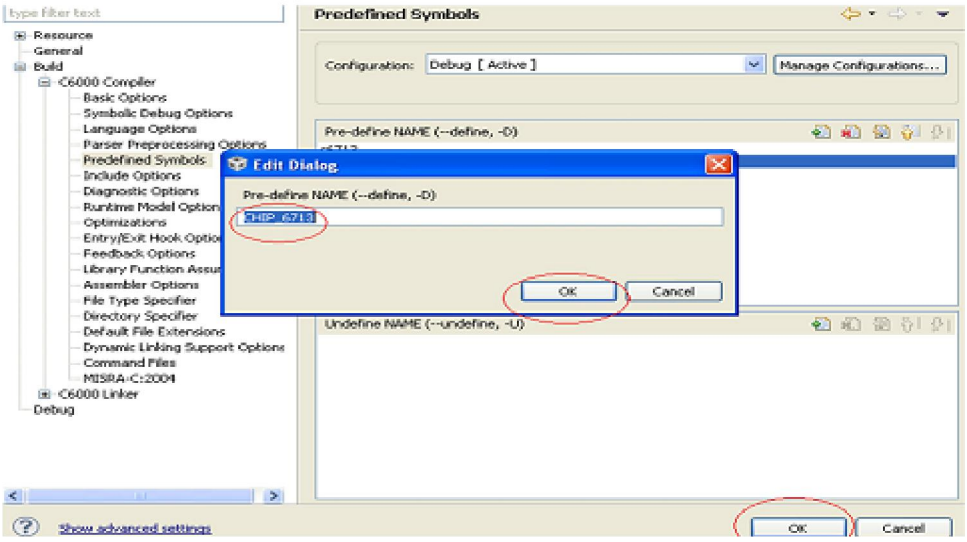


Cliquez ensuite sur le système de fichiers et accédez au chemin C:\CCStudio_v3.1\C6000\csl\include, puis cliquer sur ok.



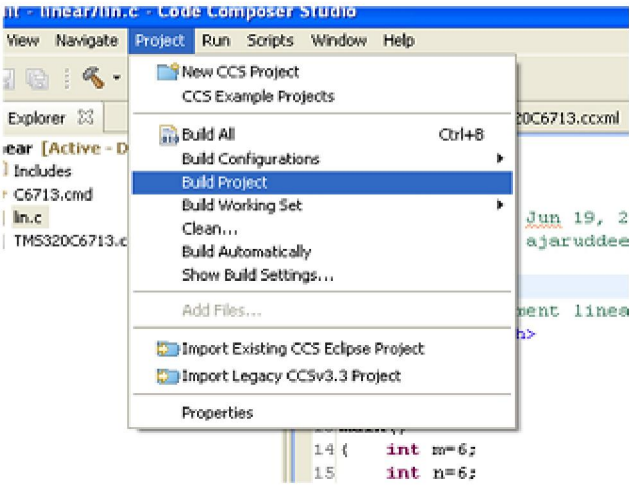
Maintenant, sous le même build, allez à

Build→C6000 Compiler→Predefined symboles, puis cliquez à nouveau sur ajouter (en haut à droite). Puis tapez CHIP_6713 et cliquez ok.



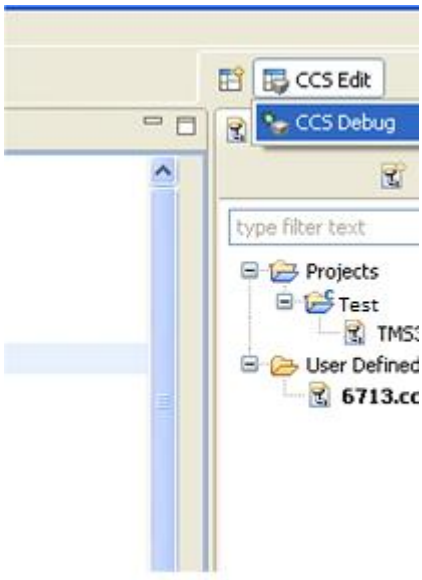
Étape 8 : Construire le projet

Allez sur Project→Build Project comme le montre la ci-dessous.



Étape 9 : exécuter le projet

Basculer vers la prospective de débogage CCS comme indiqué dans la figure ci-dessous.



Aller ensuite sur Run→Resume pour exécuter le programme