الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research

Université Ibn Khaldoun de Tiaret
Faculté des sciences Appliquées
Département de Génie Electrique

Travaux Pratiques

µ-processeurs et µ-contrôleurs

Avec rappels de cours, corrigés et programmes types

Formation : Master Electrotechnique

Code Unité : UEM 1.1

Matière : TP - µ-processeurs et µ-contrôleurs

Par : Dr. TAHRI Ahmed

Expertisé par :

Pr. SEBAA Morsli, Professeur à l'université de Tiaret

Pr. NASRI Djilali, Professeur à l'université de Tiaret

Année : 2022/2023

Partie I Système à µ-processeur 8086

- Présentation générale d'un système à µ-processeur 8086
- TP1 : Prise en main d'un environnement de programmation sur µ-processeur **(01 semaine)**
- TP2 : Programmation des opérations arithmétiques et logiques dans un µ-processeur **(01 semaine)**
- TP3 : Utilisation de la mémoire vidéo dans un µ-processeur. **(01 semaine)**
- TP4 : Gestion de la mémoire du µ-processeur. **(02 semaines)**
- TP5 : Commande d'un moteur pas à pas par un µ-processeur **(02 semaines)**

Présentation générale d'un système à µ-processeur 8086

1. Présentation générale du 8086

L'Intel 8086 est un microprocesseur a 16 bits fabriqué par Intel à partir de 1978. C'est le premier processeur de la famille x86, qui est devenue l'architecture de processeur la plus répandue dans le monde des ordinateurs personnels, stations de travail et serveurs informatiques en raison du choix d'IBM de l'utiliser comme base de l'IBM PC sorti quelques années après.

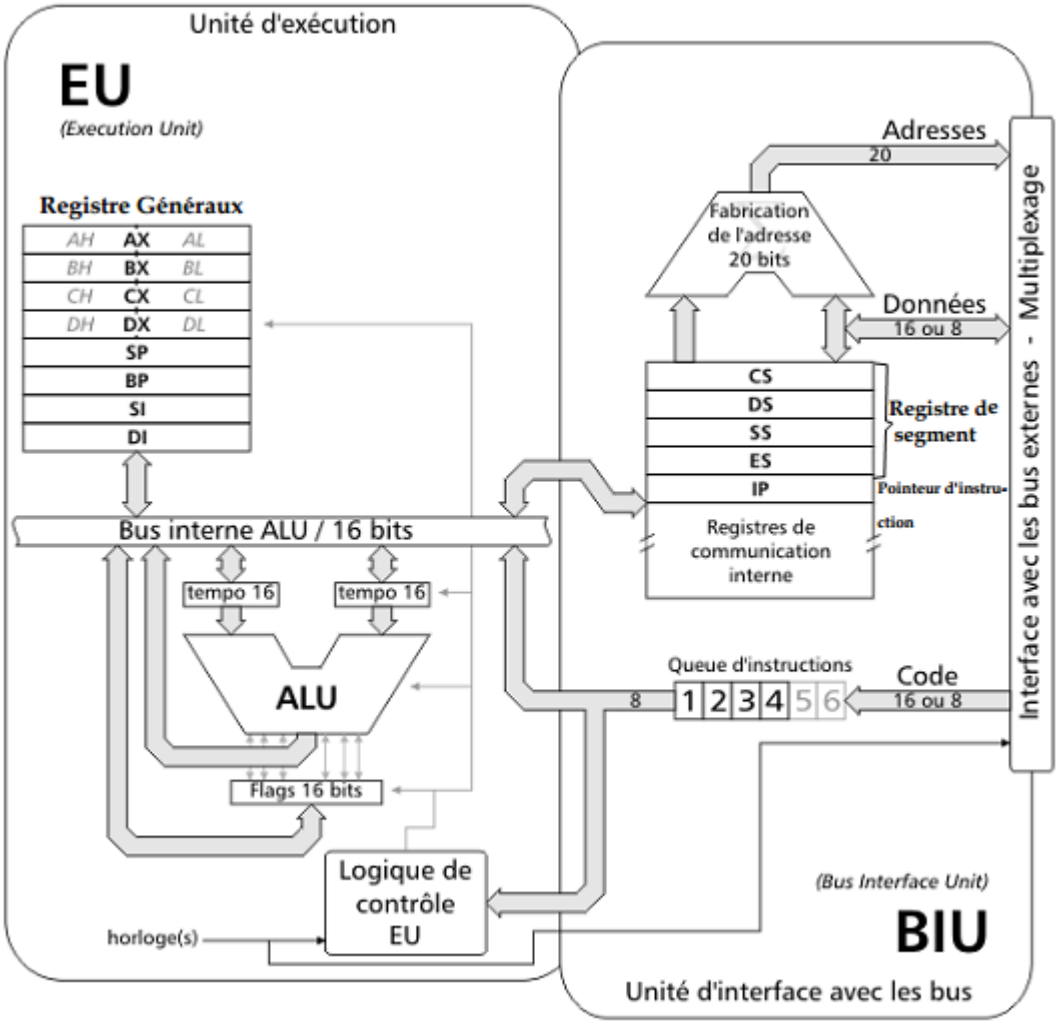


Figure 1. Registres du processeur 8086

1.1. Les registres :

La programmation en langage machine nécessite la connaissance au préalable des registres internes du processeur utilisé. En effet, en langage assembleur chaque instruction fait référence à une action élémentaire effectuée par le processeur. Il est mentionné pour chaque action qu'une des données manipulées doit obligatoirement se trouver au sein d'un registre du processeur. Nous présentons les différents registres du 8086 chacun dans sa catégorie. Les différents registres sont affichés la figure 1.

Les registres du 8086 sont séparé en cinq catégories, nommés *registres général, registres de segment* ***(segment registers)****, registres de pointer (****Pointer registers****), registres de la pile (****stack registers****) , et les registres des états(****status registers****)*, comme illustre la figure 1-3.

Les registres étant situés à l'intérieur du processeur (CPU), ils sont beaucoup plus rapides que la mémoire. L'accès à un registre ne prend généralement pas de temps. Par conséquent, vous devriez essayer de conserver les variables dans les registres.

Ces registres sont facilement accessibles aux programmeurs, et chaque registre a une fonction spéciale qui doit être clairement comprise si vous voulez écrire en langage assembleur.

1.1.1. Les registres généraux

Quatre registres à usage général (AX, BX, CX, DX) ont une largeur de 16 bits mais sont accessibles sous forme d'octet ou de mot. Ces registres de données sont normalement utilisés pour stocker des résultats temporaires qui seront exploités par des instructions ultérieures.

Accumulateur (AX) : le registre se compose de 2 registres à 8 bits AL et AH qui peuvent être combinés ensemble et utilisés comme registre 16 bits AX. AL dans ce cas contient l'octet de poids faible du mot et AH l'octet de poids fort. L'accumulateur peut être utilisé pour les opérations d'E / S et la manipulation de chaînes de caractère.

Registre de base (BX) le registre se compose de 2 registres à 8 bits BL et BH qui peuvent être combinés ensemble et utilisés comme registre 16 bits BX. BL dans ce cas contient l'octet de poids faible du mot et BH l'octet de poids fort. Le registre BX contient généralement un pointeur de données utilisé pour l'adressage indirect basé, indexé basé ou registre.

Registre de comptage CX (*count register*) : le registre se compose de 2 registres à 8 bits CL et CH qui peuvent être combinés ensemble et utilisés comme registre 16 bits CX. CL dans ce cas contient l'octet de poids faible du mot et CH l'octet de poids fort. Le registre CX peut être utilisé comme un compteur dans la manipulation de boucle et les instructions de décalage / rotation.

Registres généraux		
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Registres de pointer
SI
DI
IP

Registres de segment
CS
DS
SS
ES

Registres d'état (status registers)	
FlagH	**Flag**L

Registres de la pile (Stack registers)
SP
BP

Figure 1-1 : les registres de 8086

Registre de donnée DX (*DATA register*) le registre se compose de 2 registres à 8 bits DL et DH qui peuvent être combinés ensemble et utilisés comme registre 16 bits DX. DL dans ce cas contient l'octet de poids faible du mot et DH l'octet de poids fort. Le registre DX peut être utilisé comme numéro de port dans les opérations d'E / S. Dans l'instruction de multiplication et de division d'entiers 32 bits, le registre DX contient un mot d'ordre élevé du nombre initial ou résultant.

1.1.2. Les registres de segment :

La plupart des registres contiennent des décalages(offsets) de données / instructions dans un segment de mémoire de 64 Ko (216). Il existe quatre segments différents de 64 Ko pour les instructions, la pile, les données et les données supplémentaires. Pour spécifier où dans 1 Mo de mémoire du processeur ces 4 segments sont situés, le processeur utilise quatre registres de segments.

Le segment de code (CS) est un registre de 16 bits contenant une adresse de segment de 64 Ko avec des instructions de processeur. Le processeur utilise CS pour tous les accès aux instructions référencées par le registre du pointeur d'instruction (IP). Le registre CS ne peut pas être modifié directement. Le registre CS est automatiquement mis à jour pendant les instructions de saut éloigné, d'appel éloigné et de retour éloigné.

Le segment de données (DS) est un registre de 16 bits contenant l'adresse d'un segment de 64 Ko avec des données de programme. Par défaut, le processeur suppose que toutes les données référencées par les

registres généraux (AX, BX, CX, DX) et le registre d'index (SI, DI) se trouvent dans **le segment de données**. Le registre DS peut être modifié directement à l'aide de l'instruction POP.

Le segment de pile ou stack (SS) est un registre de 16 bits contenant l'adresse d'un segment de 64 Ko avec une pile de programmes. Par défaut, le processeur suppose que toutes les données référencées par les registres du pointeur de pile (SP) et du pointeur de base (BP) se trouvent dans le registre du segment de pile et peuvent être modifiées directement à l'aide des instructions POP et LDS.

Extra Segment (ES) est un registre de 16 bits contenant une adresse de segment de 64 Ko, généralement avec des données de programme. Par défaut, le processeur suppose que le registre d'index de destination (DI) fait référence au segment ES dans les instructions de manipulation de chaîne. Le registre ES peut être modifié directement à l'aide des instructions POP et LES.

Encore une fois, notez que les quatre segments n'ont pas besoin d'être définis séparément. Les quatre segments peuvent se chevaucher complètement (CS = DS = SS = ES).

1.1.3. Registres de pointeurs

Les registres de cette catégorie ont tous une largeur de 16 bits et ne sont pas accessibles en tant qu'octet bas ou haut. Ces registres sont utilisés comme pointeurs de mémoire.

L'index de source (SI) est un registre de 16 bits. **SI** est utilisé pour l'adressage indirect indexé, indexé basé et de registre, ainsi qu'une adresse de données source dans les instructions de manipulation de chaînes de caractères,

L'index de destination (DI) est un registre 16 bits. DI est utilisé pour l'adressage indirect indexé, indexé basé et registre, ainsi qu'une adresse de données de destination dans les instructions de manipulation de chaîne de caractères.

Le pointeur d'instruction (IP) est un registre 16 bits. Le registre IP fonctionne toujours avec le registre CS et pointe vers le décalage (l'offset) ou l'instruction suivante.

Registres de pile

La pile est une zone de mémoire spéciale conçue pour le stockage rapide et temporaire des données du programme et les adresses de retour des sous-programmes.

Le pointeur de pile (SP) est un registre de 16 bits contenant un décalage (offset) dans le segment de pile (SS). La pile se développe vers le bas (vers les décalages de segment inférieur) à partir du pointeur de pile.

Le pointeur de base (BP) contient un décalage (offset) dans le segment de pile (SS) qui peut être utilisé comme pointeur vers une liste de paramètres.

1.1.4. Le registre d'état (Status)

15				**Flags**H			**8**	**7**			**Flags**L				**0**
				OF	**DF**	**IF**	**TF**	**SF**	**ZF**		**AF**		**PF**		**CF**

Indicateur de débordement (*overflow flag OF*) – **SET** (mise à 1) si le résultat arithmétique est un nombre positif trop grand ou un nombre négatif trop petit pour tenir dans l'opérande de destination.

• **Drapeau de direction (DF)** - si SET, les instructions de manipulation de chaîne décrémenteront automatiquement les registres d'index. Si EFFACÉ (mise à 0), les registres d'index seront auto-incrémentés.

• **Indicateur d'activation d'interruption (IF)** - lorsque SET ce bit, les interruptions masquables amèneront la CPU à transférer le contrôle vers un emplacement spécifié par le vecteur d'interruption.

Drapeau de piège (Trape) (TF) - si SET, une interruption en mode pas à pas se produira après l'instruction suivante. TF est EFFACÉ par l'interruption à pas à pas.

Drapeau de Signe (SF) - SET si le bit le plus significatif du résultat est négatif. EFFACÉ si le bit le plus significatif du résultat est positif.

Drapeau Zéro (ZF) - SET si le résultat opéré est zéro.

Drapeau de retenu auxiliaire (AF) - SET s'il y avait retenu ou emprunt aux bits 0-3 dans le registre AL, EFFACE autrement.

Drapeau de parité (PF) - SET si 8 bits de poids faible du résultat contiennent un nombre pair de 1 bit, EFFACE dans le cas contraire.

Drapeau de Retenu (carry) (CF) - SET s'il y a eu report ou emprunt au bit le plus significatif lors du dernier calcul de résultat.

2. Méthodes de programmation

2.1. 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 ?

• 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 fonctionnant de manière satisfaisante.

2.2. Langage machine et assembleur :

- Langage machine : codes binaires correspondant aux instructions ;
- Assembleur : logiciel de traduction du code source écrit en langage assembleur (mnémoniques).

Réalisation pratique d'un programme :

Rédaction du code source en assembleur à l'aide d'un éditeur (logiciel de traitement ASCII) :

– édit 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,

– A86 disponible en shareware sur Internet, ...

Pour obtenir le code objet : code machine exécutable par le microprocesseur ;

Chargement en mémoire centrale et exécution : rôle du système d'exploitation.

Pour la mise au point (débogage) du programme, on peut utiliser un programme d'aide à la mise au point (comme DEBUG sous MS-DOS) permettant :

• l'exécution pas à pas ;

• la visualisation du contenu des registres et de la mémoire ;

• la pose de points d'arrêt ...

2.3. L'assemblage :

L'assembleur pour le microprocesseur peut être utilisé de deux manières : (1) avec des modèles qui sont uniques à un assembleur particulier, et (2) avec des définitions de segment complet qui permettent contrôle sur le processus d'assemblage et sont universels pour tous les assembleurs. Cette section présente les deux méthodes et explique comment organiser l'espace mémoire d'un programme à l'aide de l'assembleur. Il explique également le but et l'utilisation de certaines des directives les plus importantes utilisées avec cet assembleur.

2.3.1. Structure de Programme avec modèle :

==========

.model small ; ou /tiny/medium/large

.Stack <nombre>

.data

; Initialisation des données utilisées dans le programme.

; La déclaration de variable est ici.

.code

; Initialisation du segment de données,

; La logique du programme est ici.

End

2.3.2. Structure basé sur la définition segment complet :

==========

DATA_SEG SEGMENT 'DATA'

; Début de définition du segment DATA_SEG

DATA_SEG ENDS

Code SEGMENT 'CODE'

; Début de définition du segment DATA_SEG

ASSUME CS: CODE_SEG ,DS: DATA_SEG ;

ORG 100H

; Instructions

CODE_SEG ENDS ; Fin du segment CODE_SEG

END ; fin de programme

==========

Remarques :

1-Tous ce qui est après la point-virgule " ;" est considéré comme commentaire (n'est pas pris en compte par l'assembleur)

2- **.model, .stack, .data, .code**, **End** sont des directives (pseudo-opérations) de l'assembleur ; Les directives indiquent comment un opérande ou une section d'un programme doit être traité par l'assemble qui sont définis comme suit :

.MODEL Sélectionne le modèle de programmation

.STACK Sélectionne le début du segment de pile (modèles uniquement)

.DATA Indique le début du segment de données (modèles uniquement)

.CODE Indique le début du segment de code (modèles uniquement)

.STARTUP Indique l'instruction de démarrage dans un programme (modèles uniquement)

END Termine un fichier programme.

ASSUME Informe l'assembleur de nommer chaque segment (segments complets uniquement)

ORG Définit l'origine dans un segment

SEGMENT Démarre un segment pour des segments complets

• Origine du programme en mémoire : ORG offset

Exemple : org 1000H

• Définitions de constantes : nom constante EQU valeur

Exemple : escape equ 1BH

• Réservation de cases mémoires :

nom variable DB valeur initiale

nom variable DW valeur initiale

DB : Define Byte, réservation d'un octet ;

DW : Define Word, réservation d'un mot (2 octets).

Travaux pratiques N°01 : Prise en main d'un environnement de programmation sur µ-processeur

1. Objectives

- Programmation des µ-processeur par le langage assembleur
- Se familiariser avec le logiciel de programmation Emu8086.
- Débuter avec le langage Assembleur.

2. Premier pas en programmation

Parmi les opérations de base qu'on réalise en programmation Assembleur d'un µp **8086**, le transfert des données. Ceci se fait grâce à l'instructions : **MOV**.

2.1. Instruction MOV

L'instruction MOV (En anglais : move est la traduction de mot français déplacer). En assembleur, Cette instruction réalise un transfert d'une source vers une destination :

MOV destination, source

Les transferts possibles sont :

Destination	**Source**
Registre	Registre
Registre	Mémoire
Mémoire	Registre
Registre	Valeur immédiate
Mémoire	Valeur immédiate

2.2. Exemples

MOV AX, BX : charge le contenu du registre **Bx** dans le registre **Ax**. Dans ce cas, le transfert se fait du **registre Bx** (source) vers un autre **registre Ax (**destination).

MOV BL, [1200H] : cette instruction réalise le transfert du contenu de la **case mémoire** (source) d'adresse effective (offset) **1200H** vers le **registre Bl (**destination). L'instruction comporte l'adresse de la case mémoire où se trouve la donnée.

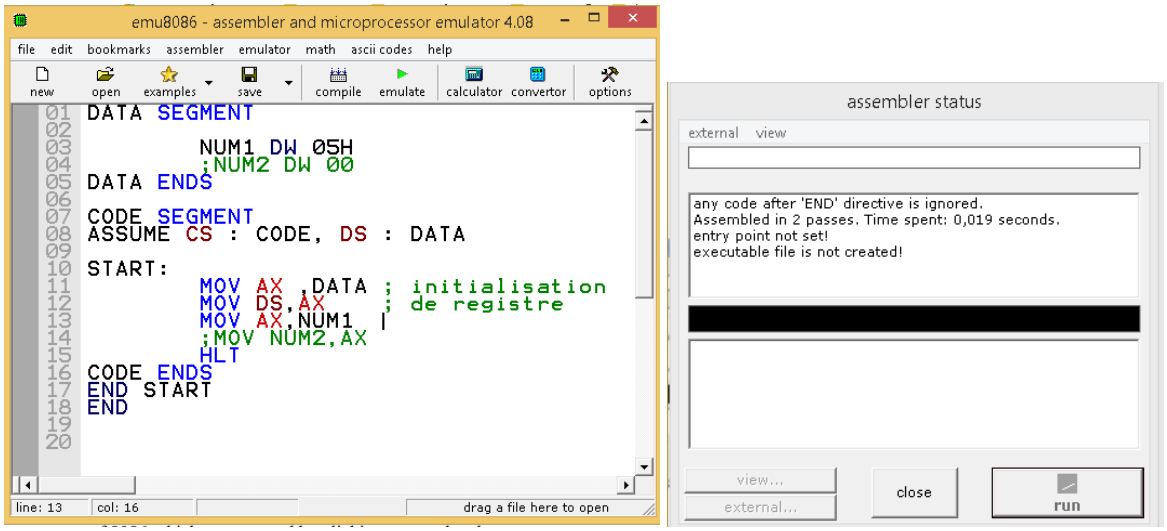
MOV [1200H], AL : cette instruction réalise le transfert du contenu du **registre_Al** (source) vers la **case mémoire** (destination) d'adresse effective (offset) **1200H**.L'instruction comporte l'adresse de la case mémoire ou la donnée va être transférer.

MOV AL, 12H : charge le registre **AL** (destination) avec la valeur immédiate (source) 12H. La donnée est fournie immédiatement avec l'instruction.

MOV [1200H], 12H : cette instruction réalise le transfert de la **valeur immédiate** (source) **12H** vers la **case mémoire** (destination) d'adresse effective (offset) **1200H**. L'instruction comporte l'adresse de la case mémoire ou la donnée va être transférer.

3. Présentation du Emu8086 :

Emu86 est un IDE éducatif pour le développement du programme assembleur. C'est un programme Windows. Il comporte un éditeur de source spécialisé qui identifie les mnémoniques 8086, les nombres hexadécimaux et les étiquettes par différentes couleurs, comme illustré à la figure 1.



(a) (b)

Figure 1. a) Éditeur de source EMU8086, et b) fenêtres de rapport d'état de l'assembleur.

Le bouton de compilation de la barre des tâches lance l'assemblage et la liaison du fichier source. Une fenêtre de rapport s'ouvre une fois le processus d'assemblage terminé. La figure 2 montre l'émulateur de 8086 qui s'ouvre en cliquant sur le bouton émuler.

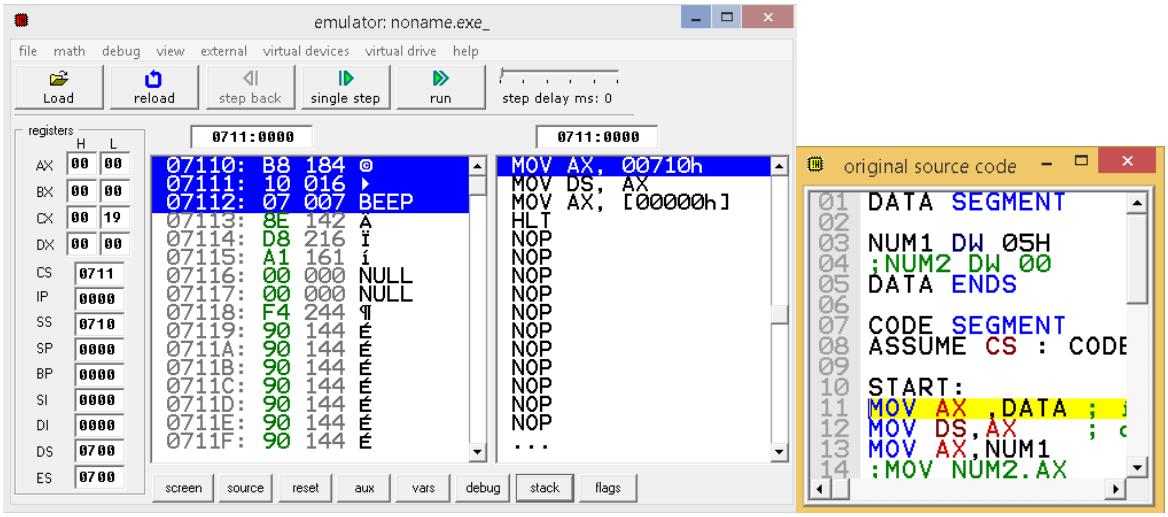


Figure 2 : la fenêtre de l'émulateur de l'environnement de débogage EMU8086

Deux fenêtres s'afficheront. Une fenêtre (**Original source code**) contient le code qu'on vient d'écrire. Une seconde fenêtre (**emulator**) qui nous permettra d'exécuter le code, de voir les contenus des différents registres, la mémoire, le programme, etc.

4. Ecrire votre premier programme en assembleur

1. Écrire un programme de langage d'assemblage pour déplacer les données de l'emplacement source à l'emplacement de destination (NUM1=05h à NUM2=00).
2. Exécuter le code dans emu8086 en mode pas-à-pas,
3. Reporter les modifications apportées sur les registres et la mémoire par chaque instruction exécutée. Tirez des conclusions.

   Remarque : L'émulateur indique les modifications de chaque pas d'exécution en le colorant en bleu.

4.1. Algorithme à suivre:

1. Définir de segment data par l'instruction SEGMENT
2. Déclaré les variables à 16bits NUM1 et NUM2a
3. Définition de segment code
4. Attribuées les étiquettes des segments de code et de données aux segments CS et DS par la directive ASSUME
5. Initialiser le registre Ds : L'emplacement de mémoire qui est défini par le nom Data est chargé dans le registre de segment DS via le registre AX. Le registre AX est ici utilisé car les registres de segment ne peuvent pas être chargés avec les données immédiates.

6. Déplacer le contenu de NUM1 vers Ax
7. Déplacer le contenu de Ax vers NUM2
8. Arrêter le programme par HLT

4.2. Programme à compléter :

```
DATA SEGMENT                ; Définition de segment DATA
        _____ ; déclarer le variable NUM1
        _____; déclarer le variable NUM2
DATA ENDS
CODE SEGMENT
ASSUME DS : DATA, CS : CODE
START :
        MOV AX ,DATA ; Initialisation de registre Ds
        MOV DS,AX
        _____
        _____
        _____; Arreter
CODE ENDS
END START
END
```

Travaux pratiques N°02 : Programmation des opérations arithmétiques et logiques dans un µ-processeur.

1. Objectifs :

- Écrire un programme en langage assembleur pour effectuer des opérations arithmétiques telles que l'addition, la soustraction, la multiplication et la division de deux nombres.

2. Préparation théorique :

2.1. Rappel:

Le 8086 permet d'effectuer les quatre opérations arithmétiques de base, l'addition, la soustraction, la multiplication et la division. Les opérations peuvent s'effectuer sur des nombres de 8 bits ou de 16 bits signé ou non signé. Les nombres signés sont représenté en complément à 2.

2.2. Les instructions Arithmétiques

Instructions	Opérandes	Description
ADD Op1,Op2	**Registre, mémoire** **Mémoire, Registre** **Registre, Registre** **Mémoire, Immédiate** **Registre, Immédiate**	**Addition** **Algorithme** : opérande1 = opérande1 + opérande2 **Exemple :** MOV AL,5 ; AL=5 ADD AL,2 ; AL=5+2 **Voir aussi:** ADC, INC
SUB Op1,Op2	**Registre, mémoire** **Mémoire, Registre** **Registre, Registre** **Mémoire, Immédiate** **Registre, Immédiate**	**Soustraction** **Algorithme** : opérande1 = opérande1 - opérande2 **Exemple** : MOV AL, 7 ; AL=7 SUB AL,2 ; AL=7-2 **Voir aussi:** SBB, DEC, NEG, CMP
MUL Op	**Registre** **Mémoire**	**Multiplication non signé.** Multiplier le contenu de Registre/ Mémoire avec le contenu de registre **AL**. **Algorithme :** Lorsque l'opérande est un **octet(8bits)**: **AX = AL * opérande.** Lorsque l'opérande est un **mot(16bits):** **(DX: AX) = AX * opérande.** **Exemple :** MOV AL,70 ; AL=70 MOV BL,10 ; BL=10 MUL BL ; AL=70*10

		Voir aussi: IMUL
DIV Op	**Registre** **Mémoire**	**Division non signé** **Algorithme :** Lorsque l'opérande est un octet : **AL = AX / opérande** AH = reste Lorsque l'opérande est un mot : **AX = (DX : AX) / opérande** DX = reste Example: MOV AX,241; AX=124 MOV BL,11; BL=11 DIV BL ; Quotient=AL=21 Reste=AH=10 **Voir aussi:** IDIV, CBW, CWD

2.3. Les instructions logiques

Instructions	Opérandes	Description
NOT Op	**Registre** **Mémoire**	**Négation :** Complément à 1de l'opérande **Algorithme** : opérande = ! opérande **Example:** MOV AL,5; AL=0101 NOT AL ; AL=1010
AND Op1,Op2	**Registre, mémoire** **Mémoire, Registre** **Registre, Registre** **Mémoire, Immédiate** **Registre, Immédiate**	**ET logique** **Algorithme** : opérande1 = opérande1 & opérande2 **Exemple** : MOV AL, 5; AL=0101 AND AL,3 ; AL=0101 & 0011=0001 **Voir aussi:** TEST
OR Op1,Op2	**Registre, mémoire** **Mémoire, Registre** **Registre, Registre** **Mémoire, Immédiate** **Registre, Immédiate**	**OU logique** **Algorithme** : opérande1 = opérande1 | opérande2 **Exemple** : MOV AL, 5; AL=0101 AND AL,3 ; AL=0101 | 0011=0111 **Voir aussi**, XOR

3. EXERCICE 01

Écrivez un programme pour additionner deux nombres de 16 bits. Le résultat doit être sauvegarder dans une variable nommée « *Result* »

Tout d'abord, le segment de données est défini par l'instruction de définition de segment. Dans le segment de données, les deux nombres sont entrés. Ensuite, le segment de code est défini et les étiquettes des segments de code et de données sont attribuées aux segments CS et DS par la directive assume.

L'emplacement de mémoire qui est défini par le nom Data est chargé dans le registre de segment DS via le registre AX. Le registre AX est ici utilisé car les registres de segment ne peuvent pas être chargés avec les données immédiates.

Un numéro est chargé dans le registre AX, puis le deuxième numéro est ajouté avec le contenu de AX. Le résultat est placé dans l'emplacement de mémoire nommé « *Result* ».

4. Exercice 02 : Ecrire un programme pour

(a) soustraction de 8 bits.

Un nombre est chargé dans le registre AL, puis le deuxième nombre est chargé dans le registre BL, et une soustraction est effectuée entre eux. Le résultat est placé dans l'emplacement de mémoire nommé "SUBRES1".

(b) soustraction de 16 bits.

Un nombre est chargé dans le registre AX, puis le deuxième nombre est chargé dans le registre BX, et une soustraction est effectuée entre eux. Le résultat est placé dans l'emplacement de mémoire nommé "SUBRES2".

(c) Écrivez un programme pour multiplier deux nombres de 8 bits.

Un nombre est chargé dans le registre AL et le deuxième nombre est chargé dans le registre BL, puis la multiplication entre ces deux nombres est effectuée. L'octet de poids faible du résultat est placé dans l'emplacement mémoire nommé « MULRES1 »et l'octet de poids fort est placé dans l'emplacement mémoire nommé « MULRES2 ».

(d) Ecrire un programme pour effectuer une division sur 16 bits

Un nombre est chargé dans le registre AX et le deuxième nombre est chargé dans le registre BX, puis la division entre ces deux nombres est effectuée. Après division, le contenu de AX est placé dans le « QUOTIENT » et le contenu de DX est placé dans le « RESTE ».

5. Exercice 03

Ecrire un programme qui effectue les fonctions suivantes entre deux nombre à 16 bits : AND, OR, XOR, NOT.

Travaux pratiques N°03 : Utilisation de la mémoire vidéo dans un µ-processeur.

1. Objectifs

- L'accés à la mémoire vidéo
- L'écriture dans la mémoire vidéo
- Affichage sur un écran en mode text

2. Préparation théorique

La mémoire vidéo est une zone mémoire qui constitue une image de l'écran (Figure 1). Si on écrit quelque chose dans cette mémoire, elle apparaît à l'écran. En mode texte, à chaque position de l'écran, correspondent deux positions(octets) de la mémoire vidéo. Le premier octet correspond au caractère affiché, le deuxième correspond à son attribut de couleur (voir tableau 1) L'attribut de caractère est une valeur de 8 bits, une couleur d'arrière-plan définie par 4 bits et une couleur de premier plan définie à 4 bits. La première paire d'octets représente le caractère en haut à gauche de l'écran. Pour le codage de la couleur, voir int 10h, fonction 09

La mémoire écran commence à l'adresse B8000h correspondant à l'adresse **Segment:Offset = B800:0000**

Si l'écran est configuré en mode 80 caractères par ligne. Chaque ligne correspond à 160 octets dans la mémoire vidéo. Pour écrire un "A" en rouge sur noir à la colonne 20 de la ligne 10, il faudra écrire 'A'=65=41h (code ascii de A) à la position 10*160 +20*2 =1640 et 04 dans la position suivante. La ligne 0 débute a la position mémoire 0, la ligne 1 à la position 160..., la ligne 10 à la position 1600.

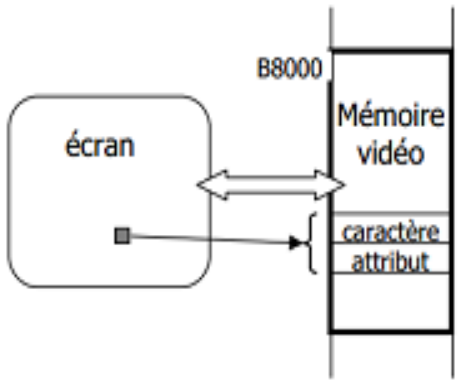


Figure 1 Mémoire vidéo

Tableau 1. Code couleur

Hex	Binaires	Couleur
0	0000	Noir
1	0001	bleu
2	0010	vert
3	0011	cyan
4	0100	rouge
5	0101	magenta
6	0110	marron
7	0111	gris clair
8	1000	gris foncé
9	1001	bleu clair
A	1010	vert clair
B	1011	cyan clair
C	1100	rouge clair
D	1101	Magenta clair
E	1110	jaune
F	1111	blanc

3. Exercice

1. Compléter le programme suivant pour afficher le mot « hello word ! » dans l'écran avec coloration (en rouge claire avec un arrière-plan jaune):
2. Quels changements doivent être apportés au programme pour afficher le mot « hello word ! » au milieu de l'écran.

==========Programme TP 02==========

CODE **SEGMENT**

ASSUME CS:CODE, DS:DATA

START:

MOV AX, 3 ; regler le mode video ; mode text80x25, 16 couleurs, 8 pages (ah=0, al=3)

INT 10H ; appliquer!

MOV AX, 1003H ; annuler le clignotement et activer les 16 couleurs:

MOV BX, 0

INT 10H

; initialiser le segment de données (DS) (il doit pointe sur la mémoire vidéo)

_____

_____

;Afficher "hello world"

; le premier octet est le code ascii, le second octet est le code de couleur.

MOV […], 'H'; écrire « H »dans la position 1 de l'écran

MOV [….], 11101100B ; colorer le caractère « H » en rouge claire sur jaune.

_____

_____

_____

_____

_____

_____

_____

MOV AH, 0; attendez n'importe quelle touche:

INT 16H

MOV AX, 4CH ; stop

INT 21H

ENDS

END START

Travaux pratiques N°04 : Gestion de la mémoire du µ-processeur

1. Objectifs :

- **Mode d'adressage du µp 8086**
- **Gestion de la mémoire du µ-processeur.**

2. Préparation théorique :

2.1. Segmentation de la mémoire :

Le 8086 possède 20 bits d'adresse, il peut donc adresser 2^{20} octets soit 1 Mo, de 00000 à FFFFF.

- La mémoire totale adressable de 1 Mo est fractionnée en pages de 64 ko appelés segments.
- La représentation de ces adresses au sein du µP (puisque les registres ne font que 16 bits) se fait par deux registres. Un registre pour adresser le segment qu'on appelle **registre segment** et un registre pour adresser à l'intérieur du segment qu'on désignera par **registre d'adressage ou offset**. Une adresse se présente toujours sous la forme **segment : offset**
- **Remarque :** Si le registre segment n'est pas spécifié (cas rien), alors le processeur l'ajoute par défaut selon l'offset choisit.

Segment	Adresse début	Adresse fin	Pointeur de segment
Segment 0	00000	0FFFF	00000
Segment 1	10000	1FFFF	10000
Segment 2	20000	2FFFF	20000
Segment 14	E0000	EFFFF	E0000
Segment 15	F0000	FFFFF	F0000

Offset utilisé	Registre segment par défaut qui sera utilisé par le CPU
Valeur	DS
DI	
SI	
BX	
BP	SS

Tableau : segments par défaut

2.2. Adresse physique

L'adresse physique (ou absolue) est calculée comme suit :

Ex: Adresse de segment -------→ 1005H
Adresse de décalage(offset) ----------→ 5555H

Adresse de segment -------→ 1005H ---------- 0001 0000 0000 0101
Décalé à gauche de 4 positions ---------- 0001 0000 0000 0101 0000
+
Adresse d'Offset ----------5555H ---------- 0101 0101 0101 0101
=
Adresse Physique---------155A5H ----0001 0101 0101 1010 0101

Adresse physique = Adresse de segment * 10H + Adresse de décalage(offset).

3. **Application** :

Écrire un programme en langage assembleur pour déplacer un tableau de 8 octets vers un autre tableau d'adresse effective(offset) 1000H dans la mémoire.

3.1. Algorithme :

1. Définir le segment de données par l'instruction de définition de **SEGMENT**.
2. Dans le segment de données, définir un tableau de 8 octets avec le nom « Tab »,
3. Définir le segment de code. Les étiquettes (les noms) des segments de code et de données sont attribuées aux segments CS et DS par la directive **ASSUME**.
4. L'emplacement de mémoire qui est défini par le nom Data est chargé dans le registre de segment DS via le registre AX. Le registre AX est ici utilisé car les registres de segment ne peuvent pas être chargés avec les données immédiates.
5. Chargez l'adresse de début du tableau dans le registre SI (instruction **OFFSET**).
6. Chargez l'adresse du destination « 1000H » dans le registre DI.
7. Chargez le nombre d'octets du tableau dans le registre CL.
8. Obtenez le premier octet du tableau dans le registre AL. (Donner un étiquette « **Next :** » dans cette ligne)
9. Déplacer le contenu de AL vers l'octet de l'adresse de destination (chargé déjà dans DI).
10. Incrémentez (**INC**) le pointeur de tableau (registre SI).
11. Incrémentez le pointeur de tableau (registre DI).
12. Utiliser « ***LOOP*** » pour boucler l'opération vers étiquette « Next ». pour aller vers l'étape 8.

15. Arrêtez

NB : L'instruction ***LOOP*** fait un saut vert l'étiquette précisé si le registre CX≠0, décrément CX également.

3.2. Travail demandé :

1. Faire un organigramme à partir de l'algorithme donnée.
2. Ecrire le programme en assembleur.
3. Vérifier le déroulement de programme sous l'émulateur 8086.

Solution :

==========Programme==========

```
DATA SEGMENT ; start data segment
    TAB DB 06H
        DB 04EH
        DB 02DH
        DB 030H
        DB 098H
        DB 0ACH
        DB 0FEH
        DB 02DH
 DATA ENDS   ; end of data segment
CODE SEGMENT ;  Start of code segment.
ASSUME CS:CODE DS:DATA ;Assembler directive.
START:  MOV AX,DATA
        MOV DS,AX
        MOV SI,OFFSET TAB ;Set SI-register as pointer for array.
        MOV DI,1000H ;Set DI-register as pointer for result.
        MOV CL,8 ;Set CL as count for elements in the array.
Next:   MOV AL,[SI] ;Set first data as smallest.
        MOV [DI],AL
        INC SI ;Increment the address pointer.
        INC DI
        LOOP Next
        HLT ;Halt program execution.
CODE ENDS ;End of code segment.
END START ;Assembly end.
```

==========

Travaux pratiques N°05 : Commande d'un moteur pas-à-pas par un µ-processeur

1. Objectifs :

- Conception de programme en assembleur faisant intervenir des boucles,
- L'utilisation du port E/S virtuel pour commander un moteur pas à pas.

2. Préparation théorique :

2.1. Transfert de contrôle ou instruction de branchement :

Les instructions de transfert de contrôle transfèrent le flux d'exécution du programme vers une nouvelle adresse spécifiée dans l'instruction directe ou indirecte. Lorsque ce type d'instruction est exécuté, les registres CS et IP sont chargés avec de nouvelles valeurs correspondant à l'emplacement où le flux d'exécution va être transféré.

Ce type d'instructions est classé en deux types :

- **Instructions de transfert de contrôle Inconditionnel (branch):**

En cas d'instructions de transfert de contrôle inconditionnelles ; le contrôle d'exécution est transféré à l'emplacement spécifié indépendamment de tout statut ou condition. Le CS et l'IP sont modifiés inconditionnellement aux nouveaux CS et IP. Ce sont les instructions suivantes

CALL, RET, JUMP,IRET,INT N, INT O, LOOP

- **Instructions de transfert de contrôle conditionnel (branch):**

Lorsque ces instructions sont exécutées, elles transfèrent le contrôle d'exécution à l'adresse spécifiée relativement dans l'instruction, à condition que la condition dans l'**opcode** soit satisfaite, sinon, l'exécution se poursuit séquentiellement. Les conditions, ici signifie l'état des indicateurs (flags) de code de condition.

Conditional Jump (Branch) Instructions

Instruction	Description	Condition
JZ , JE	Jump on Zero, or Equal	ZF = 1
JNZ , JNE	Jump on Non-Zero or Not Equal	ZF = 0
JS	Jump on sign Set	SF = 1
JNS	Jump on sign clear	SF = 0
JO	Jump on Overflow	OF = 1
JNO	Jump on No Overflow	OF = 0
JP , JPE	Jump on Parity set, *or* Parity Even	PF = 1
JNP , JPO	Jump on Parity clear, *or* Odd Parity	PF = 0
JB , JNAE , JC	Jump on Below, *or* Not Above or Equal (unsigned)	CF = 1
JNB , JAE , JNC	Jump on Not Below, *or* Above or Equal (unsigned)	CF = 0
JBE , JNA	Jump on Below or Equal, *or* Not Above (unsigned)	CF <OR> ZF = 1
JNBE , JA	Jump on Not Below or Equal, *or* Above (unsigned)	CF <OR> ZF = 0
JL , JNGE	Jump on Less, *or* Not Greater or Equal (signed)	SF <XOR> OF = 0
JNL , JGE	Jump on Not Less, *or* Greater or Equal (signed)	SF <XOR> OF = 1
JLE , JNG	Jump on Less or Equal, *or* Not Greater (signed)	(SF <XOR> OF) <or> ZF = 0
JNLE , JG	Jump on Not Less or Equal, *or* Greater (signed)	(SF <XOR> OF) <or> ZF = 1

3. Application : Commande d'un moteur pas-à-pas

Cette application contrôle un moteur pas-à-pas virtuel par l'écriture des valeurs sur le port E/S virtuel 07 à l'aide de l'instruction OUT et IN. Elle fait intervenir également des boucles réalisées à l'aide des instructions de transfert de control conditionnel et inconditionnels.

3.1. Algorithme

1. Définir le segment de données par l'instruction de définition de **SEGMENT**.
2. Dans le segment de données, définir un tableau de 4 octets avec le nom « DATCW» contient les ordre de commande du moteur pas-a-pas : 00000110, 00000100, 00000011, 00000010.
3. Définir le segment de code. Les étiquettes (les noms) des segments de code et de données sont attribuées aux segments CS et DS par la directive **ASSUME**.
4. L'emplacement de mémoire qui est défini par le nom Data est chargé dans le registre de segment DS via le registre AX.
5. Le port 07 est relié au moteur, il est prés à recevoir des données si le bit N°07 est activé. Alors créer une boucle « Patienter » de test pour attendre le port 07 qu'il soit prêt.
   -Lire le port 7 (Instruction IN)
   -Tester le bit 7 s'il est à « 1 » (Instruction Test, AND)
   -Sauter vers « Patienter » si le résultat de test égale 0.
6. Créer une boucle pour envoyer les ordres de commande (octet par boucle) au port N°07 c-d au moteur en utilisant l'instruction **OUT**
   -Créer un étiquète de boucle extérieur « Quatre_pas» avant la boucle
   -Obtenir l'offset de Tableau DATCW dans le registre SI
   -Initialiser CL avec le nombre d'octet de DATCW
   -Créer un étiquète de boucle intérieure « Pas »
   -Insérer la boucle « Patienter » crée à l'étape 5
   -Charger un octet de DATCW dans le registre AL
   -Envoyer le Contenu de AL vers le port 07 (instruction OUT)
   -Incrémenter SI
   -boucler vers « Pas » et décrémenter CL
   -Sauter vers « Quatre_pas » sans condition
7. Terminer

3.2. Travail demandé :

1. Donner l'organigramme à partir de l'algorithme donnée
2. Ecrire le programme assembleur est exécuter le sous l'emulateur8086.

Remarque : pour voir le moteur pas-a-pas virtuel, dans la barre d'outils de emu8086, allez dans ***Virtual_devices*** puis chercher ***stepper_motor***.

3.3. Solution

==========Programme ==========

```
DATA        SEGMENT
; Bin data for clock-wise
; half-step rotation:
    DATCW    DB    0000_0110B
             DB    0000_0100B
             DB    0000_0011B
             DB    0000_0010B
DATA ENDS
CODE SEGMENT
             ASSUME CS:CODE, DS:DATA
START:
             MOV    AX,DATA
             MOV    DS,AX
             MOV    BX, OFFSET DATCW ; start from clock-wise half-step.

Quatre_pas:
             MOV CL,4
             MOV    SI, OFFSET DATCW
             Pas:
Patienter:   IN AL,7; motor sets top bit when it's ready to accept new command
             TEST AL, 10000000B
             JZ  Patienter
             MOV AL, [SI]
             OUT 7, AL
             INC SI
             LOOP Pas
             JMP Quatre_pas
CODE ENDS
END START
==========================================================================
```

Partie II Système à microcontrôleur PIC

- Présentation générale du microcontrôleur PIC
- TP6: Programmation du µ-microcontrôleur PIC (I) **(01 semaines)**
- TP7: Programmation du µ-microcontrôleur PIC (II) **(01 semaines)**
- TP8: Commande d'un moteur pas à pas par un µ-microcontrôleur PIC **(02 semaines)**

Présentation générale du µ-contrôleur PIC

Les PIC sont des microcontrôleurs RISC (de l'anglais, Reduced Instructions Set Computer) ; qui signifie : calculateur à jeu réduit d'instructions. Ils sont dédiés aux applications qui ne nécessitent pas une grande une grande quantité de calculs complexes, mais qui demandent beaucoup de manipulations d'entrées/sorties. En effet, il existe plusieurs familles de microcontrôleurs, dont les plus connues sont : Atmel, Motorola, Microship, Intel,…etc. La famille des PIC à bus de données 8 bits est subdivisée, à l'heure actuelle, en 3 grandes catégories :

- Base-ligne : utilisent des mots d'instruction de 12 bits.
- Mid-Range : utilisent des mots d'instruction de 14 bits.
- High-End : utilisent des mots d'instruction de 16 bits.

1. Brève découverte du PIC 16F84A

16F84 dont le numéro 16 signifie qu'il fait partie de la famille "MID-RANGE". C'est la famille de PIC qui travaille sur des mots de 14 bits.

La lettre F indique que la mémoire programme de ce PIC est de type "Flash".

Les deux derniers chiffres permettent d'identifier précisément le PIC, ici c'est un PIC de type 84A.

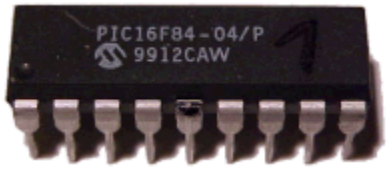


Figure 1. Image du PIC16F84A

L'alimentation du circuit est assurée par les pattes VDD (4,5 à 6V) et VSS (GND). Elle permet le fonctionnement du PIC. Il est possible d'utiliser un oscillateur avec un quartz allant jusqu'à 20 Mhz relié avec 2 condensateurs de découplage (OSC1/CLKI et OSC2/CLKO), du fait de la fréquence importante du quartz utilisé. Par conséquent, quelque soit l'oscillateur utilisé, l'horloge système base est obtenue en divisant la fréquence par 4.

La broche MCLR (Master Clear) a pour effet d'exciter la réinitialisation du PIC (RESET) lorsqu'elle est connectée à 0. Lorsque le RESET est activé, tous les registres sont initialisés.

Le PIC 16F877A, dispose de 2 ports (figure 2):

- Port A : 5 pins I/O numérotées de RA0 à RA4. En effet, ces broches (sauf RA4) sont multiplexées avec les entrées du convertisseur analogique numérique (AN0…AN4). RA4 est multiplexée avec le timer0 (T0CKI).
- Port B : 8 pins I/O numérotées de RB0 à RB7. Possibilité de déclenchement d'interruptions.

A chaque port correspondent 2 registres :

- Un registre direction pour programmer les lignes en entrée ou en sortie : TRISA, TRISB,
- Un registre de données pour lire ou modifier l'état des broches : PORTA, PORTB,

Pour déterminer les modes des ports (I/O), il faut sélectionner leurs registres TRISx :

- Le positionnement d'un bit à "1" programme la broche correspondante en entrée.
- Le positionnement d'un bit à "0" programme la broche correspondante en sortie.

Remarque : Toutes les lignes des PORTS peuvent fournir un courant de 25 mA par ligne de PORT. Une limite de 40 mA par PORT doit être respectée pour des questions de dissipation (échauffement).

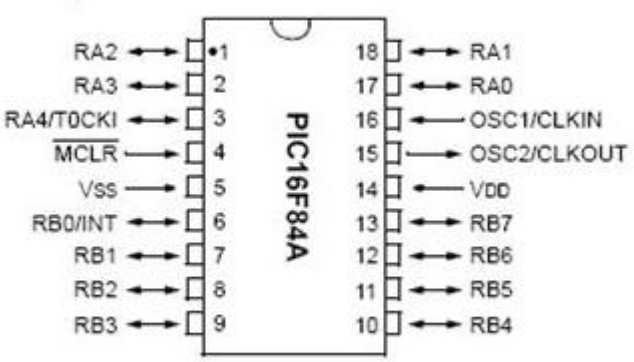


Figure 2. Brochage de Pic 16F84A

Le PIC 16F877A, dispose une zone de mémoire flash pour stocker le programme (1024 mots), une zone de ram (68 octets) pour les variables et une zone d'EEprom (64 octets) pour stocker des données non volatiles qui seront donc conservées après une coupure d'alimentation.

1.1. Plan de la mémoire ou mapping

Toutes les adresses ont leur « coordonnées » écrites en hexadécimal. Nous ne décrirons pas en détails tous les registres dans cette partie qui n'est qu'une présentation du 16F84A.

Nous remarquons que la mémoire est organisée en 2 banques. La RAM commence à partir de l'adresse 0Ch (12 en décimal) sur 68 octets. Les registres de configuration sont quant à eux placés entre l'adresse 00h et 0Bh inclus. Nous retrouvons le registre de statut en 03h.

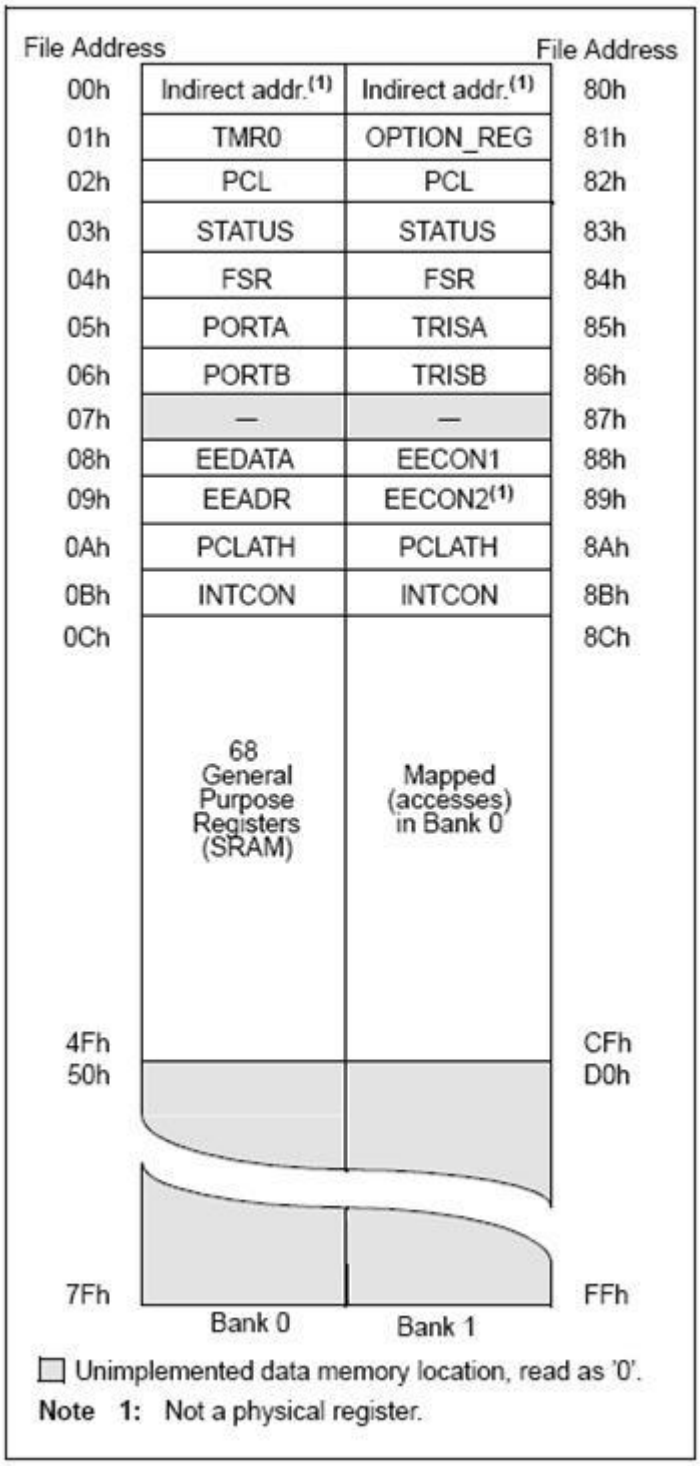


Figure 3. Plan de la mémoire ou mapping du PIC16F84A

2. Outils et technique de Programmation des PIC

Les microcontrôleurs PIC peuvent être programmés dans des langages de haut niveau ou dans leur langage machine natif. La programmation en langage machine est facilitée par l'utilisation d'un programme

assembleur, et devient ainsi une programmation en langage assembleur. Bien que le langage d'assemblage soit le moyen le plus utilisé et le plus populaire de programmation PIC.

2.1. MPLAB IDE

Le système de développement MPLAB consiste en un système de programmes qui s'exécutent sur un PC. Ce *progiciel* est conçu pour aider à développer, éditer, tester et déboguer le code PIC. le logiciel est exécuté en cliquant sur l'icône MPLAB IDE, La figure montre l'écran de commande

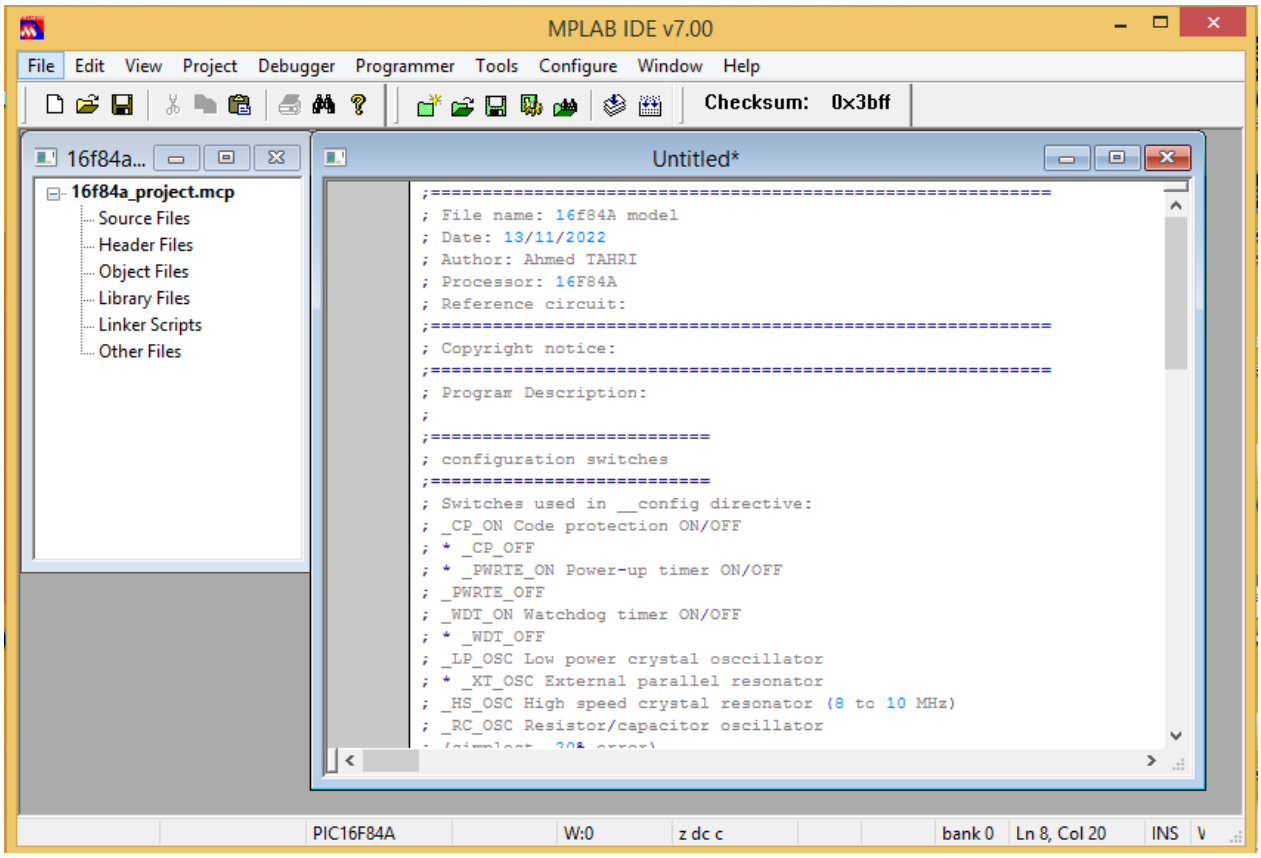


Figure 1 Image d'écran de l'IDE MPLAB

2.2. Création d'un projet

Les projets peuvent être créés à l'aide de la commande <Nouveau> du menu Projet. Le programmeur procède ensuite à la configuration manuelle du projet et y ajoute les fichiers requis. Une option alternative, à privilégier lors de l'apprentissage de l'environnement, consiste à utiliser la commande <ProjectWizard> du menu Projet. L'assistant vous demande toutes les décisions et options requises, comme suit :

1. Sélection de l'appareil. Ici, le programmeur sélectionne le matériel PIC pour le projet, par exemple 16F84A.

2. Sélectionnez la suite d'outils linguistiques. Son but est de s'assurer que les bons outils et chemins de développement sont actifs.

3. Ensuite, l'assistant demande à l'utilisateur un nom de projet et un répertoire. Il est possible de créer un nouveau répertoire à ce moment.

4. À l'étape suivante, l'utilisateur a la possibilité d'ajouter des fichiers existants au projet et de renommer ces fichiers si nécessaire. Cela peut être une option utile, car la plupart des projets réutilisent un modèle, un fichier inclus ou d'autres ressources préexistantes.

5. Enfin, l'assistant affiche un récapitulatif des paramètres du projet. Lorsque l'utilisateur clique sur le bouton <Terminer>, le projet est créé et la programmation peut commencer.

2.3. Construire « Build » le projet

Le processus de building est lancé en sélectionnant la commande <Build All> dans le menu Project. Une fois la construction terminée, un écran intitulé Sortie s'affiche et affiche les résultats de l'opération de construction. Si la construction a réussi, la dernière ligne de l'écran de sortie affiche ce résultat. La figure 2 montre l'écran de sortie après une génération réussie. Après la réussite du building, l'assembleur génère plusieurs fichiers, dont un fichier en code machine « .Hex », c'est ce dernier qui sera transférer vers le PIC à travers un programmateur.

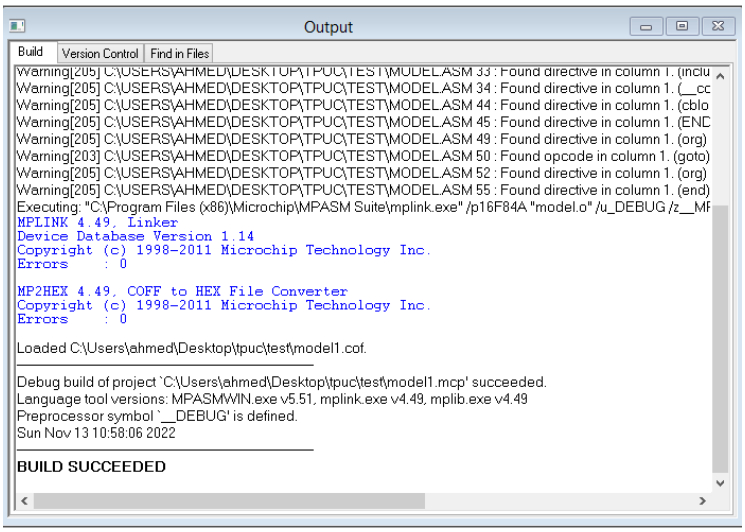


Figure 2 Le résultat du Building

2.4. MPLAB SIM

La documentation de Microchip décrit le programme SIM comme un simulateur d'événements discrets.

SIM fait partie de l'IDE MPLAB et est sélectionné en cliquant sur la commande <Select Tool> dans le menu Debugger. La commande propose plusieurs options, l'une d'entre elles étant MPLAB SIM. Une fois

le programme SIM sélectionné, une barre d'outils de débogage spéciale s'affiche. La barre d'outils et ses fonctions sont illustrées à la Figure 3.

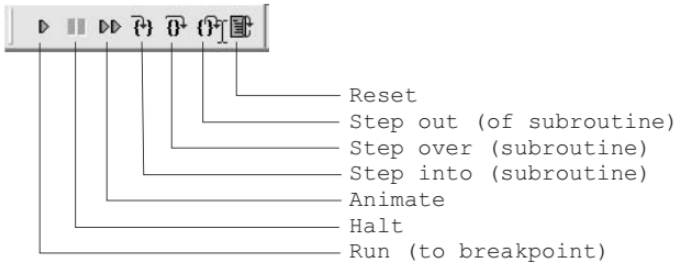


Figure 3 Le menu MPLAB SIM

Le menu « Watch » dans l'anglait « View » contient plusieurs commandes qui fournissent des fonctionnalités utiles lors de la simulation et du débogage du programme. Ceux-ci incluent des commandes pour programmer la mémoire, les registres de fichiers (FSR), l'EEPROM et les registres de fonctions spéciales. Une commande en particulier, nommée « Watch », permet d'inspecter le contenu des FSR et des GPR sur le même écran. La commande « Watch » affiche une fenêtre de programme qui contient des références à tous les registres de fichiers utilisés par le programme. L'utilisateur sélectionne ensuite les registres à afficher et ceux-ci sont affichés dans la fenêtre Watch. La fenêtre Watch est illustrée sur la Figure.4

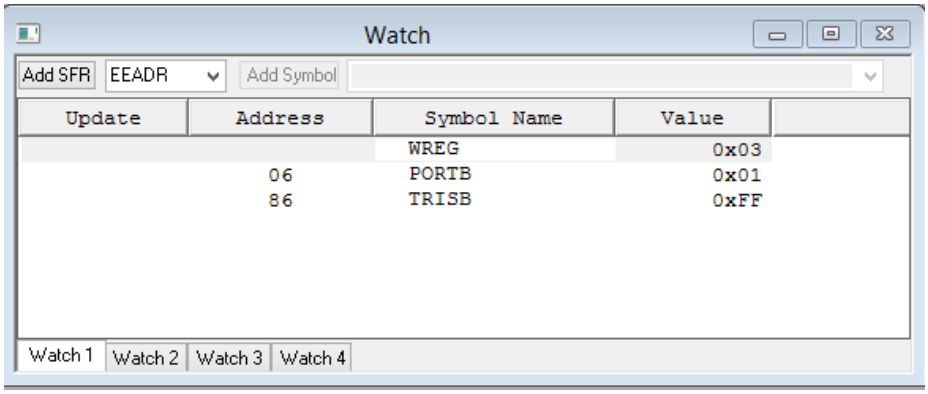


Figure 4 Le menu Watch de MPLAB

Lorsque le programme est en mode pas à pas ou en mode point d'arrêt, le contenu des différents registres peut être observé dans la fenêtre « Watch » , et les changements sont colorés en rouge.

2.5. Modèle de circuit pour 16F84A

Comme le programmeur utilise un modèle de programmation pour développer le code 16F84A, le concepteur de circuit utilise un circuit modèle. Ce circuit contient les composants qui la plupart des cartes 16F84A nécessitent. Les éléments incluent un diagramme du PIC lui-même avec le brochage, ainsi que le

câblage des composants standards, y compris la source d'alimentation, masse, la broche de réinitialisation (MCLR) et l'oscillateur le plus couramment utilisé. Illustration 5 montre un modèle de circuit pour le 16F84A.

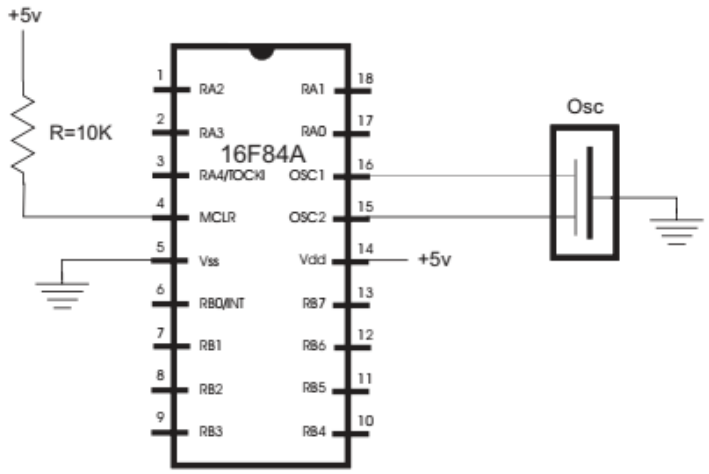


Figure 5 Modèle de circuit à base de PIC 16F84A

3. Programmation de microcontrôleur PIC

3.1. Jeu d'instructions de moyenne gamme

Le jeu d'instructions PIC de moyenne gamme se compose de 35 instructions, divisées en trois groupes généraux :

1. Opérations de registre de fichiers orientées octet et octet par octet

2. Opérations de registre de fichiers orientées bit et bit à bit

3. Instructions littérales et de contrôle

Le tableau suivant répertorie et décrit brièvement chaque instruction de l'ensemble de moyenne gamme.

Mid-range PIC Instruction Set

MNEMONIC	OPERAND	DESCRIPTION	CYCLES	BITS AFFECTED
		BYTE-ORIENTED OPERATIONS:		
ADDWF	f,d	Add w and f	1	C,DC,Z
ANDWF	f,d	AND w with f	1	Z
CLRF	f	Clear f	1	Z
CLRW	-	Clear w	1	Z
COMF	f,d	Complement f	1	Z
DECF	f,d	Decrement f	1	Z

(continues)

Mid-range PIC Instruction Set (continued)

MNEMONIC	OPERAND	DESCRIPTION	CYCLES	BITS AFFECTED
		BYTE-ORIENTED OPERATIONS		
DECFSZ	f,d	Decrement, skip if 0	1(2)	-
INCF	f,d	Increment f	1	Z
INCFSZ	f,d	Increment, skip if 0	1(2)	-
IORWF	f,d	Inclusive OR w and f	1	Z
MOVF	f,d	Move f	1	Z
MOVWF	f	Move w to f	1	-
NOP	-	No operation	1	-
RLF	f,d	Rotate left through carry	1	C
RRF	f,d	Rotate right through carry	1	C
SUBWF	f,d	Subtract w from f	1	C,DC,Z
SWAPF	f,d	Swap nibbles in f	1	-
XORWF				
		BIT-ORIENTED OPERATIONS		
BCF	f,b	Bit clear in f	1	-
BSF	f,b	Bit set in f	1	-
BTFSC	f,b	Bit test, skip if clear	1	-
BTFSS	f,b	Bit test, skip if set	1	-
		LITERAL AND CONTROL OPERATIONS		
ADDLW	k	Add literal and w	1	C,DC,Z
ANDLW	k	AND literal and w	1	Z
CALL	k	Call procedure	2	-
CLRWDT	-	Clear watchdog timer	1	TO,PD
GOTO	k	Go to address	2	-
IORLW	k	Inclusive OR literal with w	1	Z
MOVLW	k	Move literal to w	1	-
RETFIE	-	Return from interrupt	2	-
RETLWk	-	Return literal in w	2	-
RETURN	-	Return from procedure	2	-
SLEEP	-	Go into SLEEP mode	1	TO,PD
SUBLW	k	Subtract literal and w	1	C,DC,Z
XORLW	k	Exclusive OR literal with w	1	Z

Legend:
f = file register
d = destination: 0 = w register, 1 = file register
b = bit position
k = 8-bit constant

Avec

f : Address de Registre(0x00 to 0x7F)

w : Registre de travail « working register » (accumulateur)

3.1.1. Example

```
MOVLW 0x5a ;  Charger le registre W par 0x5A;
MOVWF reg ; Déplacer le contenu de W (0x5A) vers le registre d'adresse reg
BSF      f,b ;  Le bit 'b' dans le registre 'f' est activé '1'.
```

3.2. Ports Entrée/Sortie

Les broches de port peuvent être configurées en entrée ou en sortie, c'est-à-dire que les ports généraux sont bidirectionnels. Chaque port a un registre **TRIS** correspondant qui détermine si un port est désigné comme entrée ou sortie. Une valeur de 1 dans le registre TRIS du port fait du port une entrée et une valeur de 0 fait du port mappé une sortie.

Example : Pour configurer le pin 0 du Port A comme entrée est les restes pins comme des sorties on utilise :

```
MOVLW B'00000001'
MOVWF TRISA
```

3.3. Modèle de programmation 16F84A

Nous avons constaté que le développement de programmes peut être considérablement simplifié en utilisant des modèles de code. Un modèle de code est un programme dépourvu de fonctionnalité qui sert à implémenter les fonctionnalités les plus courantes et typiques d'une application. Le modèle suivant est pour le PIC 16F84A :

```
;=============================================================
; File name:
; Date:
; Author:
; Processor:
; Reference circuit:
;=============================================================
; Copyright notice:
;=============================================================
; Program Description:
;
;===========================
; configuration switches
;===========================
; Switches used in __config directive:
; _CP_ON Code protection ON/OFF
```

```
; * _CP_OFF
; * _PWRTE_ON Power-up timer ON/OFF
; _PWRTE_OFF
; _WDT_ON Watchdog timer ON/OFF
; * _WDT_OFF
; _LP_OSC Low power crystal osccillator
; * _XT_OSC External parallel resonator
; _HS_OSC High speed crystal resonator (8 to 10 MHz)
; _RC_OSC Resistor/capacitor oscillator
; (simplest, 20% error)
; |
; |_____ * indicates setup values
;=========================
; setup and configuration
;=========================
processor 16f84A
include <p16f84A.inc>
__config _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF
;=====================================================
; constant definitions
;=====================================================
;=====================================================
; PIC register equates
;=====================================================
;=====================================================
; variables in PIC RAM
;=====================================================
cblock 0x0c
endc
;=============================================================
; program
;=============================================================
org 0 ; start at address
goto main
; Space for interrupt handlers
org 0x08
main:
;=============================================================
end ; END OF PROGRAM
;=============================================================
```

Travaux pratiques N°06 : Programmation du µ-contrôleur PIC (I)

1. Objectifs :

- Premier pas en programmation en assembleur du PIC 16f84A.
- Se familiariser avec les outils employés pour la programmation microcontrôleur PIC, MPLAB et MPLAB SIM

2. Application :

2.1. Circuit à un seul LED

L'un des circuits les plus simples consiste en une seule lampe LED câblée au Port-B, ligne 0, d'un PIC 16F84A, comme illustré à la Figure 1.

Un programme pour allumer la LED sur le Port-B, ligne 0, nécessite quelques opérations de traitement essentielles. Le code doit effectuer les opérations suivantes :

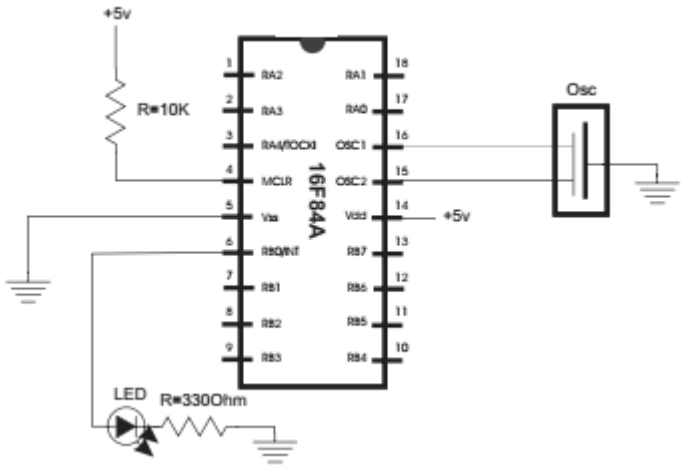


Figure 1 Circuit à un seul LED

1. Définissez et sélectionnez le processeur (dans ce cas 16F84A).

2. Liez le fichier inclus correspondant (p16f84A.inc).

3. Sélectionnez le type d'oscillateur (ici résonateur externe, type _XT).

4. Exécution directe sur l'étiquette principale.

5. Initialisez le Port-B pour la sortie.

6. Réglez la ligne 0 dans Port-B High

7. Vérifier le déroulement de programme à l'aide de MPLAB SIM.

Remarque : utiliser le modèle de programme donnée dans (2.3).

2.2. Programme de clignotant LED

Ecrire un programme qui fait clignoter la LED dans le circuit de la Figure 1.

Tout ce qui est nécessaire est une boucle de retard utilisant un compteur de registre. La logique allume la LED et décrémente le compteur jusqu'à zéro. Ensuite, il éteint la LED et compte à nouveau.

La routine de compteur illustre la création d'une procédure dans la programmation PIC. En fait, une procédure n'est rien de plus qu'une routine appelée par une étiquette à son point d'entrée et terminée par une instruction de retour. La procédure est exécutée par une instruction d'appel à son étiquette initiale, comme suit :

```
call delay ; Call to procedure
. . .
. . .
; Elsewhere in the program
delay:
; procedure instructions go here
return ; End of procedure
```

La boucle de retard la plus simple consiste à perdre du temps processeur. Étant donné que chaque instruction prend quatre cycles d'horloge, le retard peut être calculé en multipliant le nombre d'instructions dans la boucle par la vitesse d'horloge de l'appareil divisée par quatre. Ici, nous présentons simplement une boucle à double compteur sans entrer dans les détails de synchronisation.

```
delay:
        movlw .200 ; w = 200 decimal
        movwf j ; j = w
jloop:
        movwf k ; k = w
kloop:
        decfsz k,f ; k = k-1, skip next if zero
        goto kloop
        decfsz j,f ; j = j-1, skip next if zero
        goto jloop
return
```

Travail demandé

- Répétez les mêmes étapes que l'application 2.1.

3. Solution :

3.1. Circuit à un seul LED

```
;=============================
; File: LEDOn.asm
; Date:
; Author:
; Processor: 16F84A
;
; Description:
; Turn on LED wired to Port-B, line 0
;=============================
; switches
;=============================
; Switches used in __config directive:
; _CP_ON Code protection ON/OFF
; * _CP_OFF
; * _PWRTE_ON Power-up timer ON/OFF
; _PWRTE_OFF
; _WDT_ON Watchdog timer ON/OFF
; * _WDT_OFF
; _LP_OSC Low power crystal occilator
; * _XT_OSC External parallel resonator/crystal oscillator
; _HS_OSC High speed crystal resonator (8 to 10 MHz)
; Resonator: Murate Erie CSA8.00MG = 8 MHz
; _RC_OSC Resistor/capacitor oscillator
; |
; |_____ * indicates setup values
processor 16f84A
include <p16f84A.inc>
__config _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF
;=============================================================
; variables in PIC RAM
;=============================================================
; None used
;=============================================================
; m a i n p r o g r a m
;=============================================================
org 0 ; start at address 0
goto main
;=============================
; space for interrupt handler
;=============================
org 0x04
;=============================
; main program
;=============================
main:
; Initialize all line in Port-B for output
movlw B'00000000' ; w = 00000000 binary
tris PORTB ; Set up Port-B for output
; Turn on line 0 in Port-B. All others remain off
movlw B'00000001'
        ; ---|
           ; | |____ Line 0 ON
     ; |________ All others off
```

```
movwf PORTB
; Endless loop intentionally hangs up program
wait:
goto wait
end
;============================
```

3.2. Programme de clignotant LED

```
; File: LEDFlash.asm
; Date: June 2, 2006
; Author: Julio Sanchez
; Processor: 16F84A
;
; Description:
; Turn on and off LED wired to Port-B, line 0
;==========================
; switches
;==========================
; Switches used in __config directive:
; _CP_ON Code protection ON/OFF
; * _CP_OFF
; * _PWRTE_ON Power-up timer ON/OFF
; _PWRTE_OFF
; _WDT_ON Watchdog timer ON/OFF
; * _WDT_OFF
; _LP_OSC Low power crystal occilator
; * _XT_OSC External parallel resonator/crystal oscillator
; _HS_OSC High speed crystal resonator (8 to 10 MHz)
; Resonator: Murate Erie CSA8.00MG = 8 MHz
; _RC_OSC Resistor/capacitor oscillator
; |
; |_____ * indicates setup values
processor 16f84A
include <p16f84A.inc>
__config _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF
;=====================================================
; variables in PIC RAM
;=====================================================
; Declare variables at 2 memory locations
j equ 0x0c
k equ 0x0d
;=========================================================
; m a i n p r o g r a m
;=========================================================
org 0 ; start at address 0
goto main
;============================
; space for interrupt handler
;============================
org 0x04
;============================
; main program
;============================
main:
; Initialize all line in Port-B for output
movlw B'00000000' ; w = 00000000 binary
```

```
tris PORTB ; Set up Port-B for output
;
; Program loop to turn LED on and off
LEDonoff:
; Turn on line 0 in Port-B. All others remain off
movlw B'00000001' ; LED ON
movwf PORTB
call delay ; Local delay routine
; Turn off line 0 in Port-B.
movlw B'00000000' ; LED OFF
movwf PORTB
call delay
goto LEDonoff
;================================
; delay subroutine
;================================
delay:
   movlw .200 ; w = 200 decimal
   movwf j ; j = w
jloop:
   movwf k ; k = w
   kloop:
   decfsz k,f ; k = k-1, skip next if zero
   goto kloop
   decfsz j,f ; j = j-1, skip next if zero
   goto jloop
return
End
```

Travaux pratiques N°07 : Programmation du µ-microcontrôleur PIC (II)

Objectifs :

- Simulation des applications développés en MPLAB avec logiciel Proteus

1. Proteus ISIS et vérification de l'application par simulation

C'est un très bon logiciel de simulation en électronique. C'est un éditeur de schémas qui intègre un simulateur analogique, logique ou mixte. Proteus ***Schematic*** est capable de vérifier et simuler le comportement du PIC 16F84A.

En effet, la simulation permet d'ajuster et de modifier les paramètres du circuit étudié comme si on manipulait un montage électronique réel. Ceci permettra d'accélérer le prototypage et de réduire le coût de réalisation.

Figure 1. Page d'accueil de Proteus et l'accès au Schematic

2. Conception du circuit à un seul LED dans Schematic :

2.1. Le schéma de circuit

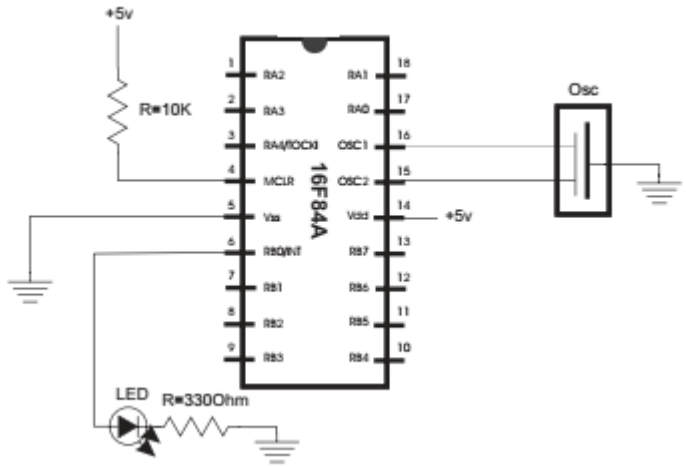


Figure 2. Circuit à un seul LED à base de PIC 16F84A

2.2. Les composants nécessaires :

-1 microcontrôleur : PIC16F84A

-Des Résistances :Res 330, 10K

-1 Oscillateur : Crystal 16Mhz

-1 LED

-Alimentation : Power(+5V), GND

2.3. Conception du circuit sous shematic

1. Parcourir les composants

Après avoir accéder à Shématic de Proteus, figure 1, commencer à insérer les composant nécessaires, pour cela appuyez sur le bouton « P » dans « Devices » ou allez dans l'anglait « Library » puis « Pick part » une bibliothèques des composants s'affiche, pour trouver le composant demander, vous avez qu'a introduire son nom de référence dans la zone de recherche comme il est illustrer sur la figure 3.

2. Insérer les composant dans la plateforme de schématique, l'emplacement des composant doit respecter le schéma de la fugure 2.
3. Lier les composants.

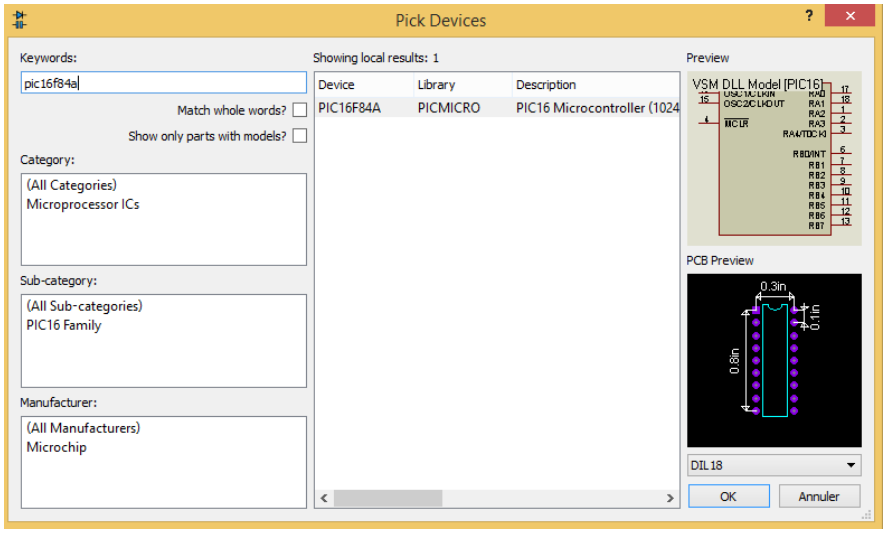


Figure 3. Bibliothéque des composants

2.4. Simulation de circuit

1. Charger le microcontrôleur PIC 16F84A par le fichier Hex de programme générer par MPLAB, pour cela , double clique sur le PIC16F84A , dans la fenêtre qui s'affiche (figue 4), parcourir le fichier Hex dans la partie « program file », puis appuyer sur OK.
2. Lancer la simulation cliquant sur le bouton Run 
3. Après l'exécution de la simulation, la LED doit s'allumer.

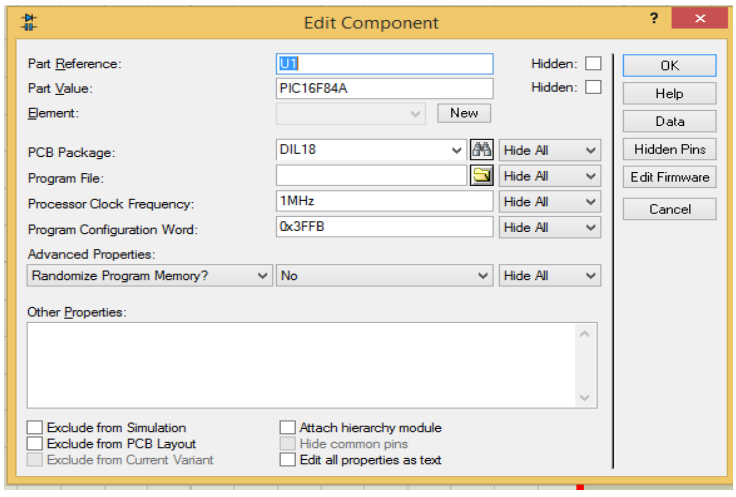


Figure 4. Menu des propriétés de PIC

3. Conception du circuit clignotant de LED dans Schématique :

Pour cette partie, veillez modifier le programme dans le PIC par le programme clignotant de LED. Puis lancer la simulation pour vérifier le fonctionnement de circuit. La LED doit clignoter.

Travaux pratiques N°08 : Commande d'un moteur pas-à-pas par un microcontrôleur PIC

1. Objectifs :

- Programmation en assembleur du PIC 16F84A pour commander un moteur pas à pas bipolaire.

2. Applications

Cette application permet de commander un moteur pas à pas bipolaire à travers le port B du µ-contrôleur PIC16F84A. Le circuit de fonctionnement sous Proteus est illustré sur la figure 1.

On suppose qu'une durée de 500 ms pour chaque pas. Les impulsions à donner au moteur pour chaque pas sont : 0110 (1er pas), 0101 (2ème pas), 1001 (3ème pas), 1010 (4ème pas).

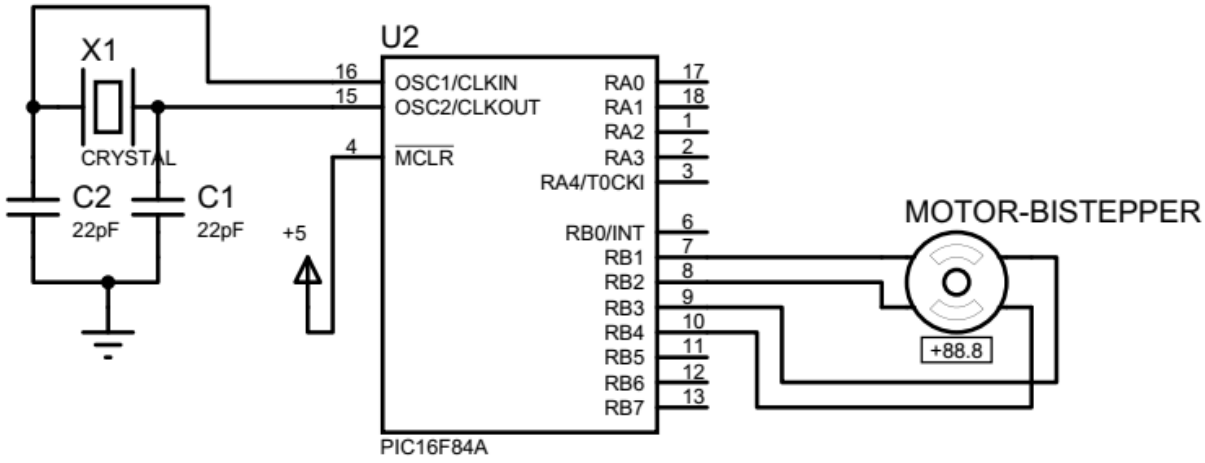


Figure 1. Circuit de fonctionnement de l'application

2.1. Composantes nécessaires :

-1 microcontrôleur : PIC16F84A

-1 Oscillateur : Crystal 8Mhz

-Moteur Pas-à-pas bipolaire « MOTOR-BISTEPPER »

-Alimentation : Power(+5V), GND

2.2. Travail demandé

1- Ecrire un programme en assembleur pour cette application en utilisant MPLAB.

2- Réaliser le circuit de la figure 1 sous Proteus

3- Vérifier le bon fonctionnement du circuit

3. Solution

Programme :

```
;=============================
; File: StepperMotor.asm
; Date:
; Author:
; Processor: 16F84A
;
; Description:
; This application allows you to control a bipolar stepper motor
; through port B of the PIC 16F84A µ-controller
;============================
; switches
;============================
; Switches used in __config directive:
; _CP_ON Code protection ON/OFF
; * _CP_OFF
; * _PWRTE_ON Power-up timer ON/OFF
; _PWRTE_OFF
; _WDT_ON Watchdog timer ON/OFF
; * _WDT_OFF
; _LP_OSC Low power crystal occilator
; * _XT_OSC External parallel resonator/crystal oscillator
; _HS_OSC High speed crystal resonator (8 to 10 MHz)
; Resonator: Murate Erie CSA8.00MG = 8 MHz
; _RC_OSC Resistor/capacitor oscillator
; |
; |_____ * indicates setup values
processor 16f84A
include <p16f84A.inc>
__config _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF
;=========================================================
; variables in PIC RAM
j equ 0x0c
k equ 0x0d
;=========================================================
; None used
;=========================================================
; m a i n p r o g r a m
;=========================================================
org 0 ; start at address 0
goto main
;=============================
; space for interrupt handler
;=============================
org 0x04
;=============================
; main program
;=============================
main:
; Initialize all line in Port-B for output
banksel TRISB ; Select the memory bank where the TRISB and PORTB reg
              ; is located
movlw B'00000000' ; w = 00000000 binary
movwf TRISB ; Set up Port-B for output
```

```
; Runnig the stepper motor
Round:
movlw B'00001100' ;0110 (step1)
movwf PORTB
call delay
movlw B'00001010' ;0101(step2)
movwf PORTB
call delay
movlw B'00010010' ;1001 (step3)
movwf PORTB
call delay
movlw B'00010100' ;1010(step4)
movwf PORTB
call delay
goto Round

;===Procedure Delay============================
delay:
movlw .200 ; w = 200 decimal
movwf j ; j = w
jloop:
movwf k ; k = w
kloop:
decfsz k,f ; k = k-1, skip next if zero
goto kloop
decfsz j,f ; j = j-1, skip next if zero
goto jloop
return

end
================================================
```