REPUBLIQUE TUNISIENNE
Ministère de l'enseignement supérieur et de la recherche scientifique
Direction générale des études technologiques
ISET Rades

Support de cours

Microprocesseurs et microcontrôleurs

Niveau : deuxième année Licence appliquée génie électrique
parcours Automatismes et Informatique Industrielle

UE : 4.4 Circuits programmables	Code ECUE : 4.4.1

Réalisé par : MABROUK Issam

Année universitaire 2013-2014

Avant propos

Ce cours est dédié aux étudiants de l'ISET inscrits en deuxième année licence appliquée en génie électrique (parcours Automatismes et Informatique Industrielle). Il est enseigné en tant que cours intégré en semestre 4 pour une charge horaire totale de 22,5 heures soit une séance d'une heure et demie par semaine. Il porte sur des notions avancées de l'étude et de la programmation des microprocesseurs, et en particuliers le 8086 d'Intel, ainsi que sur des notions de bases sur les microcontrôleurs.

Nous commençons le cours avec une présentation générale des microprocesseurs ancêtres des microprocesseurs actuels, dans un second lieu, nous traitons les architectures et les fonctions de base des microprocesseurs, passant par une explication des différents types de mémoires. Une fois les connaissances de bases assimilées, nous continuant avec l'étude et la programmation du 8086 d'Intel ainsi que la manière dont ce dernier s'interface avec les modules externes tels que l'interface parallèle 8255 et l'interface série 8250. Nous terminons ce cours avec l'étude et la programmation des microcontrôleurs en prenant comme exemple le PIC 16F 877 de Microchip.

Ce support est réalisé en collaboration avec d'autres enseignants qui n'ont pas hésité à porter leur soutien et leur savoir faire à commencer par Mme CHAOUCH Khadouja enseignant technologue à l'ISET de Rades ainsi que Mr RHAIEM Ramzi ancien assistant technologue à l'ISET de Rades.

Enfin, ce cours est entièrement disponible sur le Web, à l'adresse suivante : http://issammabrouk.weebly.com/ ou vous trouverez également d'autres cours et documents utiles.

Sommaire

Liste des figures

Liste des tableaux

Généralités sur les microprocesseurs

1.1. Définition d'un microprocesseur

On peut donner du terme microprocesseur la définition suivante: "Composant renfermant dans un seul boîtier l'unité de contrôle et l'unité de traitement d'une machine informatique de type VON NEUMANN" (Figure 1)

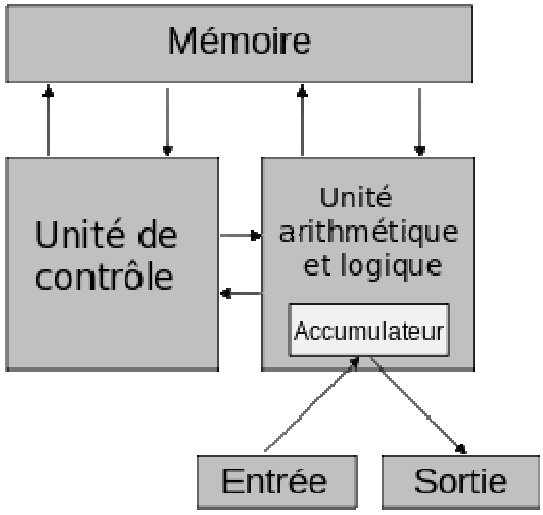


Figure 1: Architecture Von Neumann

Cette définition exclut volontairement :

- Les architectures non VON NEUMANN
- Les microcontrôleurs qui incluent d'autres composantes des systèmes informatiques (mémoire, entrées/sorties)

Ces composants constituent l'unité centrale des ordinateurs, des serveurs, des téléphones GSM etc.

1.2. Historique

L'histoire des microprocesseurs est intimement liée à celle de la technologie des semi-conducteurs dont voici les principales étapes :

- 1947 Invention du transistor
- 1958 TEXAS INSTRUMENTS produit le 1er circuit intégré (CI)
- 1961 Mise au point des technologies bipolaires TTL et ECL
- 1964 Intégration à petite échelle (SSI de 1 à 10 transistors)
- 1965 Intégration à moyenne échelle (MSI de 10 à 500 transistors)
- 1970 Mise au point de la technologie MOS
- 1971 Intégration à grande échelle (LSI de 500 à 20 000 transistors)
- 1985 Intégration à très grande échelle (VLSI plus de 20000 transistors)

C’est l’apparition de la technologie CMOS permettant un grand degré d’intégration qui a ouvert la voie à la fabrication de composants capables de contenir l’unité de contrôle et l’unité de traitement d’un ordinateur sur une seule puce.

Le premier microprocesseur a été fabriqué par INTEL en 1971. C'était un 4 bits baptisé 4004 destiné à équiper des calculatrices de bureau. En 1972 INTEL produit le premier microprocesseur 8 bits baptisé 8008 par référence au précédent. Ce microprocesseur était destiné à répondre à un contrat de fabrication d'un terminal. En réalité le 8008 s'est révélé trop lent pour satisfaire le cahier des charges du terminal et INTEL a décidé de tenter le lancement de ce produit sur le marché grand public.

L'énorme succès de cette initiative fut à l'origine de la fabrication massive des microprocesseurs. Le 8008 est constitué de 3300 transistors et effectue 60000 instructions par seconde grâce à une horloge à 300KHz.

A la suite du succès du 8008, INTEL produisit, dès 1974, le 8080 qui constituera le premier élément de la future famille de microprocesseurs de ce fabriquant. Fort de ses 6000 transistors, le 8080, doté d'une horloge à 2MHz effectue 640000 instructions par seconde. En 1974, MOTOROLA, autre fondeur de silicium, décide de lancer le 6800 qui constituera lui aussi le début d'une grande famille.

Les années 70 voient alors apparaître de petites entreprises de fabrication de microprocesseurs parfois constituées par des transfuges des deux grandes compagnies. On peut notamment citer MOS Technologies avec son 6502 très inspiré du 6800 mais vendu seulement 25$ et ZILOG avec son Z80 qui constitue une amélioration technique du 8080 (augmentation du nombre de registres, simplification de l'alimentation...). Les autres grands constructeurs (TEXAS INSTRUMENT, FAIRCHILD, RCA, SIGNETICS etc.) se lanceront aussi dans ce marché.

Ces composants seront à la base des premiers micro-ordinateurs tant personnels (ALTAIR 8800 de MITS avec un 8080, TRS80 de TANDY avec un Z80 et le tout premier APPLE avec un 6502) que professionnels (EXORCISER de MOTOROLA et multiples constructeurs développant autour du format de carte S100 pour 8080 et Z80).

On peut remarquer que la conception même du composant avait été chez certains constructeurs (MOTOROLA, MOS Technologies …) plus guidée par des considérations informatiques qu'électroniques (possibilité de tests selon la représentation des nombres, sauvegarde automatique de contexte, distinction entre interruption matérielle et logicielle...). Tandis que d'autres (INTEL, ZILOG …) avaient opté pour une approche plus matérielle (test de parité, pas de sauvegarde automatique de contexte pour prendre en compte les interruptions plus rapidement, rafraîchissement de mémoires dynamiques).

1.3. LES ANCÊTRES

Les microprocesseurs 4 et 8 et 16 bits font déjà partie du passé, néanmoins une connaissance de ces microprocesseurs nous permet d'avoir une idée sur l'évolution qu'a connue la technologie et son impact sur ces derniers qui ont permis à ceux d'aujourd'hui d'exister.

1.3.1. Les 4 bits :

Le Tableau 1 illustre les principaux microprocesseurs 4 bits :

Constructeur Référence	INTEL 4004	INTEL 4040	ROCKWELL PPS4	FAIRCHILD PPS25
Nombre d'instructions	46	60	50	95
Temps d'addition entre registres (en ias)	8	8	5	3
Espace mémoire	4K	8K	4K	6,5K
Registres d'usage général	16	24	4	1

Tableau 1: les principaux microprocesseurs 4 bits

Le 4004, apparu en Mars 1971, intègre 2250 transistors et est doté d'une horloge à 740KHz. Il traite des données sur 4 bits bien que la mémoire soit organisée en mots de 8 bits. Le jeu d'instructions comporte 46 instructions codées sur 8 bits.

La mémoire maximale adressable est de 1Ko pour les données (RAM) et 4Ko pour le code (ROM). Le 4004 possède 16 registres de 4 bits utilisables aussi comme 8 registres de 8 bits. Il gère les appels de sous programmes par une pile interne à 4 niveaux.

1.3.2. Les 8 bits :

Le Tableau 2 présente les principaux microprocesseurs 8 bits.

L'architecture interne de ces microprocesseurs est très simple et directement calquée sur l'architecture de VON NEUMANN.

Constructeur Référence	INTEL 8008	INTEL 8080	INTEL 8085	MOTOROLA 6800	ZILOG Z80	MOS Technologies 6502	ROCKWELL PPS8	NATIONAL SC/MP
Nombre d'instructions	48	69	71	71	69	71	90	50
Temps d'addition entre registres	12,5 à 20	1,3 à 2	1,3	2	1,6	2	4	5 à 25
Espace mémoire	16K	64K	64K	64K	64K	64K	32K	64K
Registres d'usage général	7	7	7	3	17	3	3	6
Nombre de transistors	3300	4000	6200	4000				
Horloge (en MHz)	0,3	2 ou 2,67 ou	3,5 ou 6	1 ou 1,5 ou 2				
Année	1972	1974	1976	1974	1976	1975	1976	1976

Tableau 2: les principaux microprocesseurs 8 bits

1.3.3. Les 16 bits

L'Intel 8086 (également appelé iAPX 86) est un microprocesseur 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.

Il fut lancé en mai 1978 au prix de 360 dollars. Il est basé sur des registres 16 bits, et dispose d'un bus externe de données de 16 bits et d'un bus d'adresse de 20 bits, qui lui permet d'adresser 1 Mio. Il contient 29 000 transistors gravés en 3 µm. Sa puissance de calcul varie de 0,33 MIPS (lorsqu'il est cadencé à 4.77 MHz comme dans l'IBM PC) jusqu'à 0,75 MIPS pour la version 10 MHz.

Le 8086 a été conçu et commercialisé comme étant compatible et permettant l'utilisation du même langage assembleur que pour le 8008,8080, ou 8085 de manière à ce que l'exécution des codes sources ne demande que peu voire aucune modifications pour le 8086, Cependant, la conception du 8086 a été conçu de manière a supporté un traitement complet du 16 bits, au lieu des capacités 16 bits assez basique du 8080/8085.

Architecture et fonctions de base des microprocesseurs

Nous allons au cours de ce chapitre nous pencher sur l'architecture des microprocesseurs ainsi que leurs fonctions de base.

2.1. Architecture de base d'un ordinateur

L'architecture, dite architecture de Von Neumann décompose l'ordinateur en quatre parties distinctes : (Figure 2)

- Le processeur est composé d'une unité arithmétique et logique (UAL ou ALU en anglais) ou unité de traitement : son rôle est d'effectuer les opérations de base et d'une unité de contrôle, chargée du séquençage des opérations ;
- Les registres qui sont des petites mémoires de travail.
- La mémoire principale qui contient à la fois les données et le programme exécuté par l'unité de contrôle.
- Les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.

Les différents composants sont reliés par des bus.

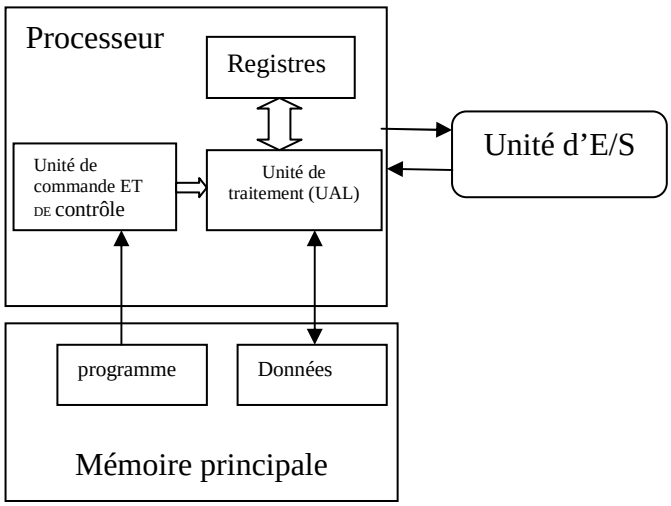


Figure 2: Architecture de base d'un ordinateur

2.2. Structure de la mémoire principale (MP)

La mémoire est divisée en emplacements (des cases mémoires contiguës) de taille fixe (par exemple huit bits) utilisés pour stocker instructions et données. En principe, la taille d'un emplacement mémoire pourrait être quelconque ; en fait, la plupart des ordinateurs en service aujourd'hui utilisent des emplacements mémoire d'un octet (" byte " en anglais, soit huit bits, unité pratique pour coder un caractère par exemple).

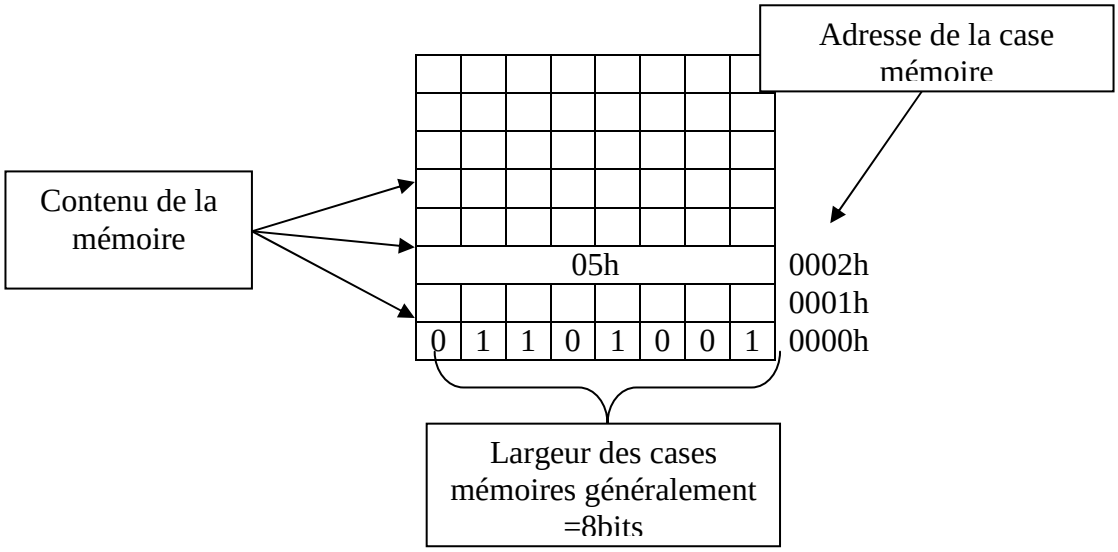


Figure 3: structure de la mémoire principale

Seul le processeur peut modifier l'état de la mémoire. Chaque emplacement mémoire conserve les informations que le processeur y écrit jusqu'à coupure de l'alimentation électrique, où tout le contenu est perdu (contrairement au contenu des mémoires externes comme les disquettes et disques durs).

Les seules opérations possibles sur la mémoire sont :

- *écriture* d'un emplacement : le processeur donne une valeur et une adresse, et la mémoire range la valeur à l'emplacement indiqué par l'adresse;
- *lecture* d'un emplacement : le processeur demande à la mémoire la valeur contenue à l'emplacement dont il indique l'adresse. Le contenu de l'emplacement lu reste inchangé.

2.3. L'unité arithmétique et logique

L'unité arithmétique et logique est une unité combinatoire permettant de réaliser plusieurs fonctions sur un couple d'entrée. Elle permet d'exécuter plusieurs fonctions de type arithmétique (addition, soustraction, ...). Elle peut aussi exécuter plusieurs fonctions logiques (et, ou, ...). Figure 4 montre les différents éléments fournis à l'unité arithmétique et logique, et les différents éléments fournis par l'unité arithmétique et logique.

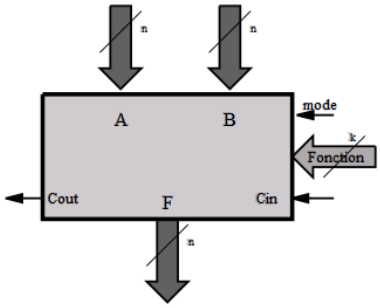


Figure 4: Unité arithmétique et logique

Une commande **mode** permet de spécifier le type de calcul (arithmétique ou logique). La commande **fonction** sur k bits, permet d'indiquer l'une parmi les 2^k fonctions possibles dans le mode considéré. La fonction sélectionnée s'appliquera sur les données **A** et **B** pour fournir le résultat **F**. L'entrée Cin sera utilisée dans le mode arithmétique comme retenue entrante, tandis que la sortie Cout sera la retenue sortante dans ce même mode arithmétique. Dans le cas du mode logique, étant donné que la retenue entrante ne sert pas, nous pourrons utiliser cette entrée comme un bit de commande supplémentaire, de manière à obtenir 2^{k+1} fonctions logiques différentes.

2.4. L'unité de commande et de contrôle

C'est l'unité de control qui supervise le déroulement de toutes les opérations au sein du Processeur. Elle est constituée principalement de :

- **l'horloge :** C'est l'horloge qui génère les signaux qui permettent le cadencement et la synchronisation de toutes les opérations. Attention, l'horloge n'est pas une montre au sens commun du terme, c'est juste un signal carré qui a une fréquence fixe (3 Ghz par exemple), a chaque coup (front) d'horloge, le microprocesseur (qui ne l'oublions pas n'est qu'un circuit électronique) réalise une tache élémentaire. L'exécution d'une instruction nécessite plusieurs coups d'horloges.

- **Le compteur programme PC :** Le compteur programme (PC : program counter) est un registre (pointeur) qui contient l'adresse de la case mémoire où est stockée le prochain élément d'instruction qui devra être chargé dans le processeur pour être analysé et exécuté. Au début de l'exécution d'un programme, le PC est initialisé par le système d'exploitation à l'adresse mémoire où est stockée la première instruction du programme. Le compteur programme est incrémenté automatiquement chaque fois qu'un élément d'instruction est chargée dans le processeur

- **Le registre d'instruction RI :** C'est là où le CPU stocke l'instruction en cours d'exécution.

- **Le décodeur :** C'est lui qui va "décoder" l'instruction contenue dans RI et générer les signaux logiques correspondant et les communiquer au séquenceur.

- **Le séquenceur :** Il gère le séquencement des opérations et génère les signaux de commande qui vont activer tous les éléments qui participeront à l'exécution de l'instruction et spécialement l'ALU.

- **Le registre d'état :** Le registre d'état est formé de plusieurs bits appelés drapeaux ou indicateurs (Flags) qui sont positionnés par l'ALU après chaque opération. Par exemple l'indicateur Z indique quand il est positionné que le résultat de l'opération est égal à Zéro. L'indicateur C indique que l'opération a généré une retenue. Le bit N indique que le résultat est négatif …

On dispose d'un jeu d'instructions conditionnées par l'état de différents drapeaux

2.5. Les registres et l'accumulateur

Le processeur utilise toujours des *registres*, qui sont des petites mémoires internes très rapides d'accès utilisées pour stocker temporairement une donnée, une instruction ou une adresse. Chaque registre stocke 8, 16 ou 32 bits.

Le nombre exact de registres dépend du type de processeur et varie typiquement entre une dizaine et une centaine.

Parmi les registres, le plus important est le registre *accumulateur*, qui est utilisé pour stocker les résultats des opérations arithmétiques et logiques.

L'accumulateur intervient dans une proportion importante des instructions.

Par exemple, examinons ce qu'il se passe lorsque le processeur exécute une instruction comme ``*Ajouter 5 au contenu de la case mémoire d'adresse 180*" :

- Le processeur lit et décode l'instruction;
- le processeur demande à la mémoire la contenu de l'emplacement 180;
- la valeur lue est rangée dans l'accumulateur;
- l'unité de traitement (UAL) ajoute 5 au contenu de l'accumulateur;
- le contenu de l'accumulateur est écris en mémoire à l'adresse 180.
- C'est l'unité de commande qui déclenche chacune de ces actions dans l'ordre.
- L'addition proprement dite est effectuée par l'UAL.

2.6. Exécution d'un programme :

Le travail d'un processeur est d'exécuter des programmes. Un programme est une suite d'instructions écrites une par ligne. Une instruction peut être plus ou moins sophistiquée selon le langage utilisé. Pour un langage de bas niveau comme l'assembleur, une instruction réalise une tache élémentaire comme une addition par exemple. Avec un langage de haut niveau comme le C++, une instruction peut réaliser un ensemble de taches qui nécessiterait plusieurs instructions en Assembleur.

Un processeur quel qu'il soit sait exécuter un ensemble bien défini de codes machines (jeux d'instructions). Chaque code machine est un nombre binaire de quelques octets, il correspond à une instruction élémentaire bien définie. Sur papier, on a pris l'habitude de les représenter en hexadécimal pour faciliter.

exécutable ↓	↓	Source ↓
Codes machine tels qu'ils seront présentés au processeur	Codes machine comme on a pris l'habitude de les représenter sur papier	Ecriture plus lisible dite Mnémonique ou Assembleur
10111101 01000001 00000000	BD 41 00	MOV BP,41h
10111110 01100101 01000001	BE 65 41	MOV SI,4165h
00000001 11011000	01 D8	ADD AX,BX
00111000 01000111 00000011	38 47 03	CMP [BX+3],AL

Figure 5: exemples d'instructions dans leurs différentes formes

Par exemple, l'instruction (assembleur) MOV BP,41h qui signifie : placer le nombre 41h dans le registre BP est codée (en hexadécimal) par les trois octets BD 41 00 , Dans la suite de ce cours, nous désignerons ces octets par : éléments d'instruction.

Quand on veut écrire un programme, on dispose d'un choix très important de langages de programmation différents les uns des autres : Assembleur, Basic/Qbasic, Pascal, C/C++, Visual Basic, Visual C++, Delphi, Java, …

En fait, les lignes de programme que nous écrivons constituent ce qu'on appelle un programme source qui sera stocké dans un fichier texte dont l'extension dépend du langage choisi (test.pas pour le pascal, test.cpp pour le c++ etc.)

Ces programmes sources sont compréhensibles par nous mais pas par le processeur. Pour que le processeur puisse les comprendre il faut les traduire (compiler) en langage machine qui est une suite de codes machine. Sur les PCs, se sont les fichiers avec l'extension .exe (test.exe). Chaque langage de programmation a son compilateur qui permet de transformer le programme source en un programme exécutable compréhensible par le processeur. Tous les exécutables se ressemblent et le processeur ne sait pas avec quel langage ils ont été écrits.

Avec un langage de haut niveau comme le C++, une instruction que nous écrivons peut être très sophistiquée. C'est le compilateur C++ qui la traduit en un ensemble d'instructions élémentaires compréhensible par le processeur.

L'intérêt du langage assembleur est que chaque instruction que nous écrivons correspond à une instruction élémentaire du processeur. C'est comme si on travaillait directement en langage machine. Ainsi, on sait exactement tout ce que fait le processeur lors de l'exécution d'un programme.

Quand on demande l'exécution d'un programme, celui-ci est chargé par le système d'exploitation (à partir du disque dur) dans une zone de la mémoire RAM. Celle-ci étant organisée en octets, chaque élément d'instruction est stocké dans une position mémoire. L'adresse (le numéro) de la case mémoire de début est communiquée au processeur pour qu'il commence l'exécution au bon endroit.

Les mémoires

Une mémoire est un circuit à semi-conducteur permettant d'enregistrer, de conserver et de restituer des informations (instructions et variables). C'est cette capacité de mémorisation qui explique la polyvalence des systèmes numériques et leur adaptabilité à de nombreuses situations. Les informations peuvent être écrites ou lues. Il y a écriture lorsqu'on enregistre des informations en mémoire, lecture lorsqu'on récupère des informations précédemment enregistrées.

3.1. Organisation d'une mémoire

Avec une adresse de n bits il est possible de référencer au plus 2_n cases mémoire. Chaque case est remplie par un mot de données (sa longueur m est toujours une puissance de 2). Le nombre de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.

En plus du bus d'adresses et du bus de données, un boîtier mémoire comprend une entrée de commande qui permet de définir le type d'action que l'on effectue avec la mémoire (lecture/écriture) et une entrée de sélection qui permet de mettre les entrées/sorties du boîtier en haute impédance.

On peut donc schématiser un circuit mémoire par la Figure 6 où l'on peut distinguer :

- les entrées d'adresses
- les entrées de données
- les sorties de données
- les entrées de commandes :
  - une entrée de sélection de lecture ou d'écriture. (R/$\overline{W}$)
  - une entrée de sélection du circuit. ($\overline{CS}$)

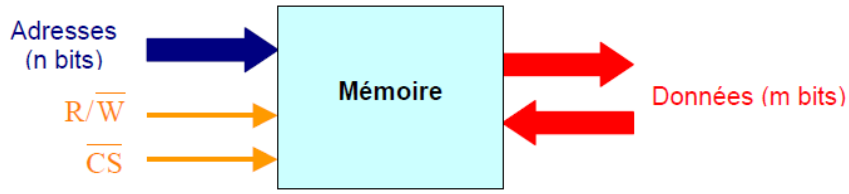


Figure 6: schéma d'une mémoire

Une opération de lecture ou d'écriture de la mémoire suit toujours le même cycle (Figure 7) :

1. sélection de l'adresse
2. choix de l'opération à effectuer (R/W)
3. sélection de la mémoire (CS = 0)
4. lecture ou écriture la donnée

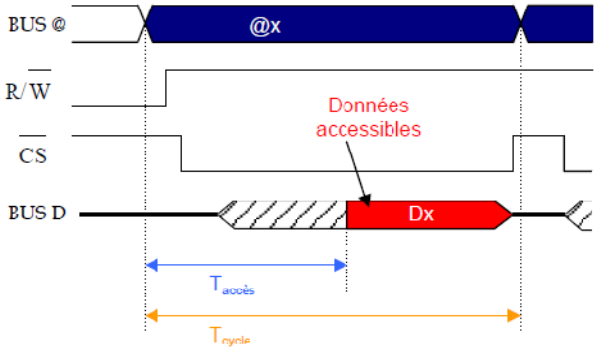


Figure 7: Chronogramme d'un cycle de lecture

3.2. Caractéristiques d'une mémoire

- La capacité : c'est le nombre total de bits que contient la mémoire. Elle s'exprime aussi souvent en octet.

- Le format des données : c'est le nombre de bits que l'on peut mémoriser par case mémoire. On dit aussi que c'est la largeur du mot mémorisable.

- Le temps d'accès : c'est le temps qui s'écoule entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.

- Le temps de cycle : il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.

- Le débit : c'est le nombre maximum d'informations lues ou écrites par seconde.

- Volatilité : elle caractérise la permanence des informations dans la mémoire. L'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.

Les mémoires utilisées pour réaliser la mémoire principale d'un système à microprocesseur sont des mémoires à semi-conducteur. On a vu que dans ce type de mémoire, on accède directement à n'importe quelle information dont on connaît l'adresse et que le temps mis pour obtenir cette information ne dépend pas de l'adresse. On dira que l'accès à une telle mémoire est aléatoire ou direct.

A l'inverse, pour accéder à une information sur bande magnétique, il faut dérouler la bande en repérant tous les enregistrements jusqu'à ce que l'on trouve celui que l'on désire. On dit alors que l'accès à l'information est séquentiel. Le temps d'accès est variable selon la position de l'information recherchée. L'accès peut encore être semi-séquentiel : combinaison des accès direct et séquentiel.

Pour un disque magnétique par exemple l'accès à la piste est direct, puis l'accès au secteur est séquentiel.

3.3. Différents types de mémoires

3.3.1. *Les ROMs :* Mémoire Morte, la ROM (Read Only Memory)

C'est une mémoire à lecture seule (Read Only Memory), l'écriture nécessite un programmateur ou une procédure plus longue que pour les RAMs. Les informations qu'elle contient sont conservées en permanence, même lors d'une coupure d'alimentation. Nous vous proposons à la suite les différents types de mémoires ROM (Figure 8)

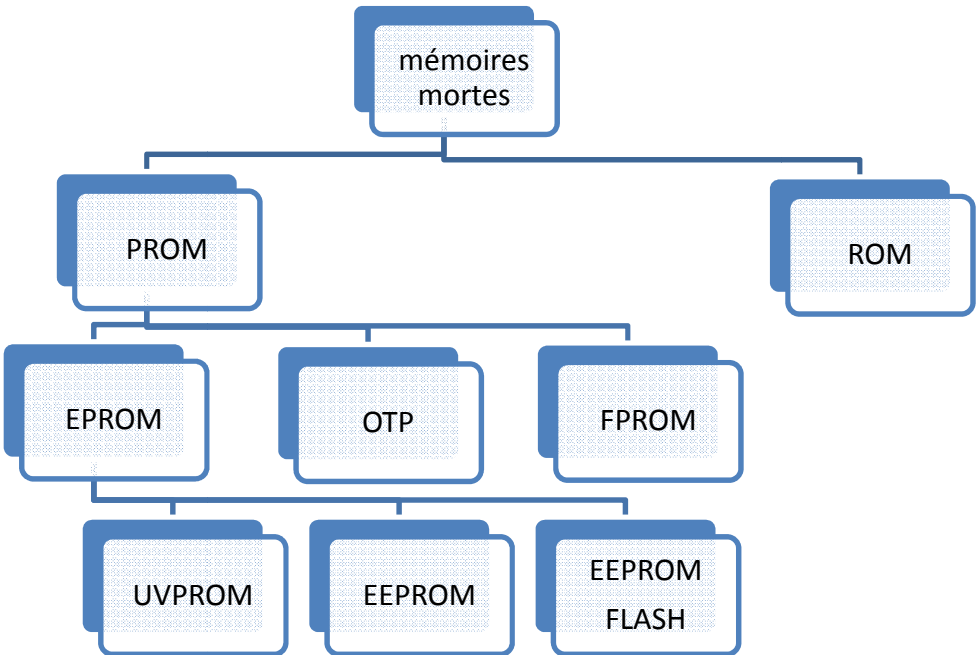


Figure 8: les mémoires ROM

- ROM (Read Only Memory)
  Masque de fabrication à:
  - diodes disposées sur un réseau de lignes et de colonnes.
  - ou à transistors dont sont effectuées des coupures à leurs bases.

- PROM à fusibles ou FPROM (Programmable Read Only Memory ou Fuse PROM) Réalisée à partir de transistors bipolaires dont leurs liaisons entre l'émetteur et la colonne sont effectuées par l'intermédiaire d'un fusible.

- EPROM (Erasable Programmable Read Only Memory): Elles possèdent les avantages de la PROM avec un plus, qui est l'effacement des données par l'utilisateur.
  Le terme EPROM correspond à un ensemble de composants. Abusivement on utilise ce terme pour les UVPROM.

- OTP MEMORY (One Time Programmable MEMORY): programmable une seule fois à l'aide d'un programmateur (Rem : On utilise souvent le terme PROM

- UVPROM (EPROM effaçable par une exposition aux ultraviolet U.V.)
  Ce type de mémoire est placé dans un boîtier céramique avec fenêtre de quartz effaçable aux U.V.

- EEPROM ou E2PROM (Erasable Electricaly PROM) : effaçable électriquement.

  Il existe des mémoires EEPROM à accès parallèle et des mémoires EEPROM à accès série. À Pour celle à accès parallèle, les données entrent et sortent sous la forme d'un octet.
  Pour celles à accès série, les données entrent et sortent en série en commençant par le bit de poids fort des octets. La liaison série utilisée est de type synchrone (SPI ou I2C). Dans ce cas, ces mémoires sont intéressantes par leur faible encombrement (boîtier DIP8), puisque l'adresse et la donnée sont transmis sous forme série. Toutefois le temps d'accès aux données est alors relativement long.

- EPROM FLASH : effaçable électriquement, écriture plus rapide que les EEPROM mais effacement de toute la capacité de la mémoire en un coup.

3.3.2. *Les RAMs* : Mémoire Vive, la RAM (Random Access Memory)

Les mémoires RAM sont volatiles et à accès direct (Accès aléatoire en lecture ou écriture). Dans cette catégorie de mémoires on trouve :

- Les mémoires RAM statiques (SRAM) dans lesquelles les informations sont mémorisées par une bascule de type D et conservées tant que l'alimentation est présente (mémoire volatile), elles sont réalisées en technologie MOS ou bipolaire.

- Les mémoires RAM dynamiques (DRAM) qui utilisent un condensateur comme cellule mémoire (un bit mémorisé) de l'information. Cette information tend à se dégrader à cause des courants de fuites, ce qui nécessite un rafraîchissement périodique.

C'est l'espace de travail pour les ordinateurs. C'est là que se recopient le système d'exploitation et les applications (programmes), stockés sur le disque dur, et ou sont transformés vos documents avant sauvegarde sur une mémoire de masse.

Pour les systèmes micro industriels c'est l'emplacement des données appelées variables. Ces données peuvent correspondre à des variables globales ou locales du programme, ou à des données de transmission ou de traitement (acquisition, valeurs de sortie).

- NOVRAM = (Non volatile RAM) : Mémoire de type SRAM associée à une pile de sauvegarde, ou système de transfert vers une mémoire morte de type FLASH. Leur coût à capacité égale est supérieur aux SRAMs. Elles sont donc limitées aux applications micro contrôleurs industriels ou il est indispensable de ne pas perdre les données.

Etude et programmation d'un microprocesseur (8086)

Les premiers pc commercialisés au début des années 1980 utilisaient le 8086, qui est un microprocesseur 16bits. La gamme de microprocesseurs sui équipe les micro-ordinateurs de type PC et compatibles sont les 80x86.

Chacun de ces processeurs, Le 80286, 80386, 80486 et Pentium (ou 80586) est plus puissant que les précédents : horloge plus rapide, bus de données plus large, de nouvelles instructions sont ajoutées comme le calcul sur les réels et ajout de registres.

Chacun de ces microprocesseurs est compatible avec les modèles précédents. On parle de compatibilité ascendante car un programme écrit sur un 286 fonctionne sur un 386 mais pas l'inverse.

Du fait de cette compatibilité, nous allons étudier et programmer le 8086, utilisé dans nos salles de Travaux Pratiques.

4.1. LE MICROPROCESSEUR 8086

Il se présente sous forme d'un boîtier de 40 broches alimenté par une alimentation unique de 5V. (Figure 9: brochage du 8086)

- Il possède un bus multiplexé adresse/donnée de 20 bits.
- Le bus de donnée occupe 16 bits ce qui permet d'échanger des mots de 2 octets
- Le bus d'adresse occupe 20 bits ce qui permet d'adresser 1 Mo (2^{20})
- Il est entièrement compatible avec le 8088, le jeu d'instruction est identique. La seule différence réside dans la taille du bus de données, celui du 8088 fait seulement 8 bits. Les programmes tourneront donc un peu plus lentement sur ce dernier puisqu'il doit échanger les mots de 16 bits en deux étapes.
- Tous les registres sont de 16 bits, mais pour garder la compatibilité avec le 8085/8088, certains registres sont découpés en deux et on peut accéder séparément à la partie haute et à la partie basse.

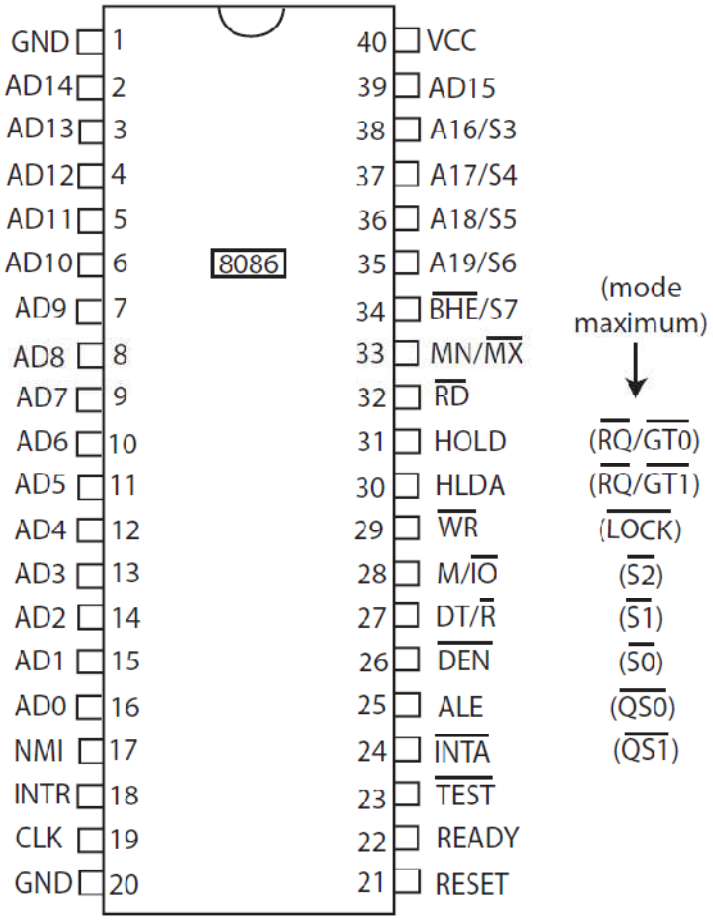


Figure 9: brochage du 8086

Le tableau suivant donne la description des différentes broches de 8086 :

CLK	Entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur.
RESET	Entrée de remise à zéro du microprocesseur.
READY	Entrée de synchronisation avec la mémoire.
TEST\	Entrée de synchronisation du microprocesseur d'un événement extérieur.
MN/MX\	Entrée de choix du mode de fonctionnement du microprocesseur : MN : Mode minimum. MX : Mode maximum.
NMI	Entrée de demande d'interruption non masquée.
INTR	Entrée de demande d'interruption masquée.
INTA\	Sortie indique la réponse à une demande d'interruption.
HOLD et HLDA	Signaux de l'accès direct mémoire par le circuit DMA.
S0….S7	Signaux d'état du µp en mode STEP BY STEP (pas à pas).
A0 ... A19	Signaux de bus d'adresse de 20 bits (1Mo espace adressable).
D0 ... D15	Signaux de bus de données de 16 bits.
RD\	Signal de demande de lecture.
WR\	Signal de demande d'écriture.

M/IO\	Signal de séparation d'accès mémoire ou port : M/IO\ = 1 : accès mémoire. M/IO\ = 0 : accès port d'E/S
DEN	Sortie indique que l'information qui circule dans bus AD est une donnée.
DT/R\	Sortie indique le sens de transfert des données sur la bus de données: DT/R\ = 1 : le bus de donnée en sortie. DT/R\ = 0 : le bus de donnée en entrée.
BHE\	Signal d'accès de l'octet du poids fort sur la bus (D8 / D15).
ALE	Sortie indique que l'information qui circule dans bus AD est une adresse.

Tableau 3:description des broches du 8086

4.2. La segmentation de la mémoire

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

L'adresse de la première case mémoire est 0000 0000 0000 0000 0000 celle de la dernière casse est 1111 1111 1111 1111 1111 1111. Nous allons représenter les adresses en hexadécimal, et notre 8086 peut donc adresser 1 Mo allant de 00000 à FFFFF. Le problème qui se pose est comment représenter ces adresses au sein du µP puisque les registres ne font que 16 bits soit 4 digits au maximum en hexadécimal. La solution adoptée par Intel a été la suivante : Puisque avec 16 bits en peut adresser 2^{16} octets = 65535 octets = 64 ko, La mémoire totale adressable de 1 Mo est fractionnée en pages de 64 ko appelés segments. On utilise alors deux registres pour adresser une case mémoire donnée, 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**

A titre d'exemple, procédons au découpage de la mémoire en 16 segments qui ne se chevauche pas.

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

Tableau 4: segmentation de la mémoire

Considérons la case mémoire d'adresse 20350, appelée adresse absolue ou adresse linéaire. Cette case mémoire se situe dans le segment 2, son adresse relative à ce segment est 350, on peut donc la référencer par le couple segment: offset = 20000:350, Se pose maintenant le problème de la représentation de cette adresse au sein du CPU car les registres de 16 bits ne peuvent contenir que 4 digits. S'il n'y a

aucun problème pour représenter 350 dans un registre d'offset, on ne peut pas représenter 20000 dans un registre segment. La solution adoptée par Intel est la suivante :

- Dans le registre segment, on écrit l'adresse segment sans le chiffre de faible poids Dans le registre d'adressage (d'offset) on écrit l'adresse relative dans le segment
- Pour calculer l'adresse absolue qui sera envoyée sur le bus d'adresse de 20 bits, le CPU procède à l'addition des deux registres après avoir décalé le registre segment d'un chiffre à gauche :

	x	x	x	x	0	Segment
+		x	x	x	x	Offset
	x	x	x	x	x	Adresse absolue

Dans notre exemple, l'adresse de la case mémoire considérée devient 2000:350 soit :

Segment = 2000
Offset = 350

L'adresse absolue est calculée ainsi :

	2	0	0	0	0	Segment
+		0	3	5	0	Offset
	2	0	3	5	0	Adresse absolue

Remarque : Les zones réservées aux segments ne sont pas exclusives, elles peuvent se chevaucher. La seule règle à respecter lors du choix d'un segment est que le digit de plus faible poids soit nul. Nous pouvons donc commencer un segment tous les 16 octets.

4.3. Les registres du 8086

Registres généraux		Registres d'adressage	Registres de segment	Registre de commande
AX	AH AL	SP	CS	IP
BX	BH BL	BP	DS	FLAGS
CX	CH CL	SI	SS	
DX	DH DL	DI	ES	

Tous les registres et le bus interne du 8086 sont structurés en 16 bits. Vu de l'utilisateur, le 8086 comprend 3 groupes de 4 registres de 16 bits, un registre d'état de 9 bits et un compteur programme de 16 bits non accessible par l'utilisateur.

4.3.1. Les registres généraux

Les registres généraux participent aux opérations arithmétiques et logiques ainsi qu'à l'adressage. Chaque demi-registre est accessible comme registre de 8 bits, le 8086 peut donc effectuer des opérations 8 bits d'une façon compatible avec le 8080.

- **AX** : Accumulateur : Usage général, Obligatoire pour la multiplication et la division, Ne peut pas servir pour l'adressage
- **BX** : Base : Usage général, Adressage, (Par défaut, son offset est relatif au segment DS)
- **CX** : Comptage et calcul : Usage général, Utilisé par certaines instruction comme compteur de répétition. Ne peut pas servir pour l'adressage
- **DX** : Data : Usage général, Dans la multiplication et la division 16 bits, il sert comme extension au registre AX pour contenir un nombre 32 bits, Ne peut pas servir pour l'adressage.

4.3.2. Les registres d'adressage (offset)

Ces registres de 16 bits permettent l'adressage d'un opérande à l'intérieur d'un segment de 64 ko (216 positions mémoires)

- **SP** : Pointeur de Pile : Utilisé pour l'accès à la pile. Pointe sur la tête de la pile. Par défaut, son offset est relatif à SS
- **BP** : Pointeur de Base : Adressage comme registre de base, (Par défaut, son offset est relatif à SS), Usage général
- **SI** : Registre d'index (source) : Adressage comme registre d'index, (Par défaut, son offset est relatif à DS), Certaines instruction de déplacement de donnés l'utilise comme index de l'opérande source. L'opérande destination étant indexé par DI, Usage général
- **DI** : Registre d'index (destination) : Adressage comme registre d'index, (par défaut, son offset est relatif à DS), Certaines instruction de déplacement de donnés l'utilise comme index de l'opérande destination, l'opérande destination étant indexé par SI.

4.3.3. Les registres de segments

Ces registrent sont combiné avec les registres d'offset pour former les adresses. Une case mémoire est repérée par une adresse de la forme RS:RO. On place le registre segment au début d'une zone mémoire de 64Ko, ensuite on fait varier le registre d'offset qui précise l'adresse relative par rapport à cette position.

- **CS** : **C**ode **S**egment : Définit le début de la mémoire programme. Les adresses des différentes instructions du programme sont relatives à CS
- **DS** : **D**ata **S**egment : Début de la mémoire de données dans laquelle sont stockées toutes les données traitées par le programme

- **SS** : **S**tack **S**egment : Début de la pile (voir chapitre V)
- **ES** : **E**xtra **S**egment Début d'un segment auxiliaire pour données

Une adresse doit avoir la forme [Rs : Ro]

Si le registre segment n'est pas spécifié (cas rien), alors le processeur l'ajoute par défaut selon l'offset choisit :

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

Tableau 5: segments par défaut

4.3.4. Le registre d'état (flags)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					O	I	T	S	Z		A		P		C

Tableau 6: registres d'états

Six bits reflètent les résultats d'une opération arithmétique ou logique et 3 participent au control du processeur.

- **C** : (Carry) indique le dépassement de capacité de 1 sur une opération 8 bits ou 16 bits. Ce flag peut être utilisé par des instructions de saut conditionnel, des calculs arithmétiques en chaîne ou dans des opérations de rotation.
- **P** : (Parité) indique que le nombre de 1 est un nombre pair. Ce flag est utilisé avec certains sauts conditionnels.
- **A** : (retenue **A**rithmétique) indique une retenue sur les 4 bits (digit) de poids faible. Par exemple quand la somme des 2 digits de poids faible dépasse F (15)
- **Z** : (Zéro) Indique que le résultat d'une opération arithmétique ou logique est nul. Il est utilisé dans plusieurs instructions de sauts conditionnels.
- **S** : (Signe) reproduit le bit de poids fort d'une quantité signée sur 8 bits ou sur 16 bits. L'arithmétique signée fonctionne en complément à 2. S=0 : positif, S=1 : négatif. Ce flag sert lors de sauts conditionnels.
- **T** : (Trap) met le CPU en mode pas à pas pour faciliter la recherche des défauts d'exécution.
- **I** : (Interruption) autorise ou non la reconnaissance des interruptions : I = 0 alors Interruptions autorisées. I = 1 alors Interruptions non autorisées
- **D** : (Direction) fixe la direction de l'auto-inc/décrémentation de SI et DI lors des instructions de manipulation de chaînes. D = 0 alors incrémentation des index. D = 1 alors décrémentation des index
- **O** : (Overflow) indique un dépassement de capacité quand on travaille avec des nombres signés. Comme par exemple si la somme de 2 nombres positifs donne un nombre négatif ou inversement. (40h + 40h = 80h et O=1)

4.4. Les modes d'adressage

Dans la suite on utilisera les abréviations suivantes :

INST	instruction
R	Registre quelconque
Rseg	Registre Segment
Roff	Registre d'offset
Adr	Adresse
[adr]	contenu Mémoire
Off	Offset de l'adresse
Im	donnée (constante)
Dep	déplacement (constante)
Op	Opérande
Os	Opérande source
Od	opérande destination

Tableau 7: les abréviations utilisées

La structure la plus générale d'une instruction est la suivante : INST op1, op2

L'opération est réalisée entre les 2 opérandes et le résultat est toujours récupéré dans l'opérande de gauche. Il y a aussi des instructions qui agissent sur un seul opérande Les opérandes peuvent être des registres, des constantes ou le contenu de cases mémoire, on appelle ça le mode d'adressage :

- Adressage registre (R) : L'opération se fait sur un ou 2 registres (INST R, R ; INST R)

Exemples : INC AX : incrémenter le registre AX
MOV AX, BX : Copier le contenu de BX dans AX

- Adressage Immédiat (IM) : Un des opérandes est une constante (valeur) :

INST R , im
INST taille [adr] , im

Exemples :
MOV AX, 243 : charger le registre AX par le nombre décimal 243
ADD AX, 243h : additionner le registre AX avec le nombre hexadécimal 243
MOV AX, 0xA243 : Quand le chiffre de gauche du nombre hexadécimal est une lettre, il est préférable d'utiliser le préfix 0x pour l'hexadécimal
MOV AL, 'a' : Charger le registre AL par le code ASCII du caractère 'a'
MOV AX, 'a' : Charger le registre AH par 00 et le registre AL par le code ASCII du caractère 'a'
MOV AX,'ab' : Charger AH par 'a' et AL par 'b'

- Adressage direct (DA) : Un des deux opérandes se trouve en mémoire. L'adresse de la case mémoire ou plus précisément son Offset est précisé directement dans l'instruction. L'adresse Rseg:Off doit être placée entre [ ], si le segment n'est pas précisé, DS est pris par défaut

INST R , [adr]
INST [adr] , R
INST taille [adr] , im

Exemples :

MOV AX, [243] : Copier le contenu de la mémoire d'adresse DS:243 dans AX
MOV [123], AX : Copier le contenu de AX dan la mémoire d'adresse DS:123
MOV AX, [SS:243] : Copier le contenu de la mémoire SS:243 dans AX

- Adressage indirect (IR) : Un des deux opérandes se trouve en mémoire. L'offset de l'adresse n'est pas précisé directement dans l'instruction, il se trouve dans l'un des 4 registres d'offset BX, BP, SI ou DI et c'est le registre qui sera précisé dans l'instruction : [Rseg : Roff].

INST R , [Rseg : Roff]
INST [Rseg : Roff] , R
INST taille [Rseg : Roff] , im

Exemples :
MOV AX, [BX] ; Charger AX par le contenu de la mémoire d'adresse DS:BX
MOV AX, [BP] ; Charger AX par le contenu de la mémoire d'adresse SS:BP
MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI
MOV AX, [DI] ; Charger AX par le contenu de la mémoire d'adresse DS:DI
MOV AX, [ES:BP] ; Charger AX par le contenu de la mémoire d'adresse ES:BP

L'adressage indirect est divisé en 3 catégories selon le registre d'offset utilisé. On distingue ainsi, l'adressage Basé, l'adressage indexé et l'adressage basé indexé,

- Adressage Basé (BA) : L'offset se trouve dans l'un des deux registres de base BX ou BP. On peut préciser un déplacement qui sera ajouté au contenu de Roff pour déterminer l'offset

INST R , [Rseg : Rb+dep]
INST [Rseg : Rb+dep] , R
INST taille [Rseg : Rb+dep] , im

Exemples :
MOV AX, [BX] : Charger AX par le contenu de la mémoire d'adresse DS:BX
MOV AX, [BX+5] : Charger AX par le contenu de la mémoire d'adresse DS:BX+5
MOV AX, [BP-200] : Charger AX par le contenu de la mémoire d'adresse SS:BX-200
MOV AX, [ES:BP] : Charger AX par le contenu de la mémoire d'adresse ES:BP

- Adressage Indexé (X) : L'offset se trouve dans l'un des deux registres d'index SI ou DI. On peut préciser un déplacement qui sera ajouté au contenu de Ri pour déterminer l'offset

INST R , [Rseg : Ri+dep]
INST [Rseg : Ri+dep] , R
INST taille [Rseg : Ri+dep] , im
Exemples :
MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI
MOV AX, [SI+500] ; Charger AX par la mémoire d'adresse DS:SI+500
MOV AX, [DI-8] ; Charger AX par la mémoire d'adresse DS:DI-8
MOV AX, [ES:SI+4] ; Charger AX par la mémoire d'adresse ES:SI+4

- Adressage Basé Indexé (BXI) : l'offset de l'adresse de l'opérande est la somme d'un registre de base, d'un registre d'index et d'un déplacement optionnel. Si Rseg n'est pas spécifié, le segment par défaut du registre de base est utilisé :

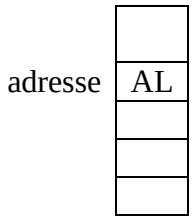
INST R , [Rseg : Rb+Ri+dep]
INST [Rseg : Rb+Ri+dep] , R
INST taille [Rseg : Rb+Ri+dep] , im

Exemples :
MOV AX,[BX+SI] ; AX est chargé par la mémoire d'adresse DS:BX+SI
MOV AX,[BX+DI+5] ; AX est chargé par la mémoire d'adresse DS:BX+DI+5
MOV AX,[BP+SI-8] ; AX est chargé par la mémoire d'adresse SS:BP+SI-8
MOV AX,[BP+DI] ; AX est chargé par la mémoire d'adresse SS:BP+DI

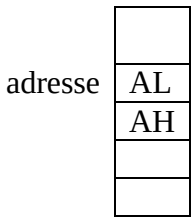
4.5. Taille des échanges avec la mémoire

La mémoire est organisée en octets. Quand on fait une instruction entre un registre et une donnée qui se trouve en mémoire, c'est le registre qui détermine la taille de l'opération. Si le registre est un registre simple (8 bits), l'opération se fera avec une seule case mémoire.

MOV [adresse], AL donne



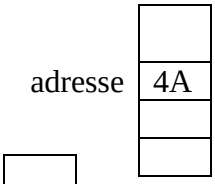
Si le registre est un registre double (2 octets), l'opération se fera avec deux cases mémoires
MOV [adresse], AX donne



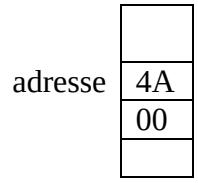
Nous remarquons que c'est la partie basse du registre qui est traitée en premier, et ceci dans les deux sens.

Quand on fait une opération entre une constante et une case mémoire, il y a ambiguïté, le processeur ne sait pas s'il faut considérer la constante sur 8 bits ou sur 16 bits. Il faut utiliser les préfixes BYTE et WORD pour préciser le nombre d'octets á écrire :

MOV BYTE ptr [adresse],4Ah ; On écrit 4A dans la position adresse



MOV WORD ptr [adresse],4Ah ; On écrit 004A, donc 4A dans la position adresse, et 00 dans la position adresse+1 :



4.6. La pile :

4.6.1. Notion de pile

Les *piles* offrent un nouveau moyen d'accéder à des données en mémoire principale, qui est très utilisé pour stocker temporairement des valeurs. Une pile est une zone de mémoire et un pointeur qui conserve l'adresse du *sommet* de la pile.

4.6.2. Instructions PUSH et POP

Deux nouvelles instructions, PUSH et POP, permettent de manipuler la pile.

PUSH *registre* empile le contenu du registre sur la pile.

POP *registre* retire la valeur en haut de la pile et la place dans le registre spécifié.

Exemple : transfert de AX vers BX en passant par la pile.

```
PUSH AX ; Pile <- AX
POP BX ; BX <- Pile
```

(Note : cet exemple n'est pas très utile, il vaut mieux employer MOV AX, BX.). La pile est souvent utilisée pour sauvegarder temporairement le contenu des registres :

AX et BX contiennent des données à conserver

```
PUSH AX
PUSH BX
MOV AX, valeur ; // on utilise AX
ADD AX, BX ; // et BX
MOV truc, BX
POP BX ; récupère l'ancien BX
POP AX ; et l'ancien AX
```

On voit que la pile peut conserver plusieurs valeurs. La valeur dépilée par POP est la *dernière* valeur empilée ; c'est pourquoi on parle ici de pile LIFO (*Last In First Out*, Premier Entré Dernier Sorti).

4.6.3. Registres SS et SP

La pile est stockée dans un segment séparé de la mémoire principale. Le processeur possède deux registres dédiés à la gestion de la pile, SS et SP.

Le registre SS (*Stack Segment*) est un registre segment qui contient l'adresse du segment de pile courant (16 bits de poids fort de l'adresse). Il est normalement initialisé au début du programme et reste fixé par la suite.

Le registre SP (*Stack Pointer*) contient le déplacement du sommet de la pile (16 bits de poids faible de son adresse).

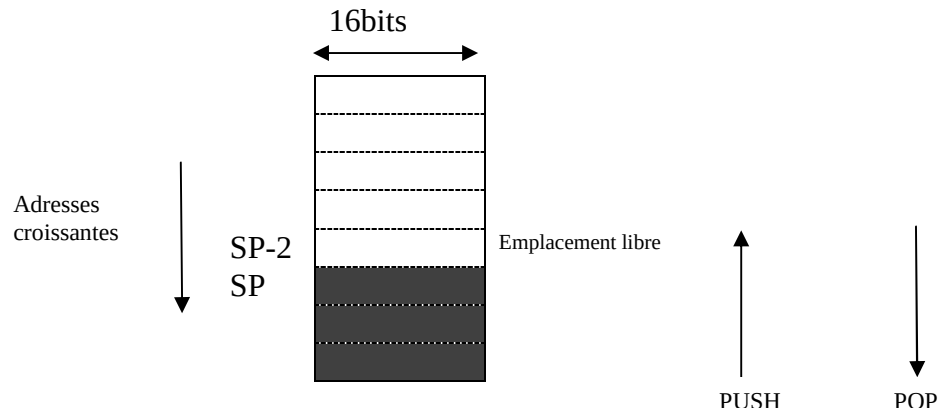


Figure 10: fonctionnement d'une pile

4.7. Les instructions du 8086

4.7.1. Les instructions de transfert

- MOV Od , Os

  Copie l'opérande Source dans l'opérande Destination

  MOV R1 , R2 copier un registre dans un autre
  MOV R , M copier le contenu d'une case mémoire dans un registre
  MOV M , R copier un registre dans une case mémoire
  MOV R , im copier une constante dans un registre
  MOV taille M , im copier une constante dans une case mémoire (taille = BYTE ou WORD)

- PUSH Op
  Empiler l'opérande Op (Op doit être un opérande 16 bits)
  - Décrémente SP de 2
  - Copie Op dans la mémoire pontée par SP
  PUSH R16
  PUSH word [adr]

- POP Op
  Dépiler dans l'opérande Op (Op doit être un opérande 16 bits)

- Copie les deux cases mémoire pointée par SP dans l'opérande Op
- Incrémente SP de 2

POP R16
POP word M

4.7.2. Les instructions Arithmétiques

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 signés ou non signés. Les nombres signés sont représentés en complément à 2. Des instructions d'ajustement décimal permette de faire des calculs en décimal (BCD).

- Addition :

ADD Od , Os Additionne l'opérande source et l'opérande destination avec résultat dans l'opérande destination, Od + Os → Od
ADD AX,123
ADD AX,BX
ADD [123],AX
ADD BX,[SI]

- ADC Od , Os Additionne l'opérande source, l'opérande destination et le curry avec résultat dans l'opérande destination :Od + Os + C → Od

- INC Op Incrémente l'opérande Op Op + 1 →Op

Attention, l'indicateur C n'est pas positionné quand il y a débordement, C'est l'indicateur Z qui permet de détecter le débordement Pour incrémenter une case mémoire, il faut préciser la taille :

INC byte [ ]
INC word [ ]

- Soustraction : SUB Od , Os Soustrait l'opérande source et l'opérande destination avec résultat dans l'opérande destination. Od - Os → Od

- SBB Od , Os Soustrait l'opérande source et le curry de l'opérande destination avec résultat dans l'opérande destination. Od - Os - C → Od

- DEC Op Décrémente l'opérande Op ; Op - 1 →Op

- NEG Op Donne le complément à 2 de l'opérande Op : remplace Op par son négatif. Cà2(Op) → Op

- CMP Od , Os Compare (soustrait) les opérandes Os et Od et positionne les drapeaux en fonction du résultat. L'opérande Od n'est pas modifié.

- Multiplication : MUL Op instruction à un seul opérande. Elle effectue une multiplication non signée entre l'accumulateur (AL ou AX) et l'opérande Op. Le résultat de taille double est stocké dans l'accumulateur et son extension (AH:AL ou DX:AX).

MUL Op8 alors AL x Op8 → AX
MUL Op16 alors AX x Op16 → DX:AX

L'opérande Op ne peut pas être une donnée, c'est soit un registre soit une position mémoire, dans ce dernier cas, il faut préciser la taille (byte ou word)

MUL BL ; AL x BL → AX
MUL CX ; AX x CX → DX:AX
MUL byte [BX] ; AL x (octet pointé par BX) → AX
MUL word [BX] ; AX x (word pointé par BX) → DX :AX

- IMUL Op (Integer Multiply) Identique à MUL excepté qu'une multiplication signée est effectuée.

- Division : DIV Op Effectue la division AX/Op8 ou (DX|AX)/Op16 selon la taille de Op qui doit être soit un registre soit une mémoire. Dans le dernier cas il faut préciser la taille de l'opérande, exemple : DIV byte [adresse] ou DIV word [adresse].

DIV Op8 ;AX / Op8 , Quotient → AL , Reste → AH
DIV S16 ;DX:AX / S16 , Quotient → AX , Reste → DX

S ne peut pas être une donnée (immédiat). Après la division L'état des indicateurs est indéfini. La division par 0 déclenche une erreur

- IDIV Op Identique à DIV mais effectue une division signée

- CBW (Convert Byte to Word) Effectue une extension de AL dans AH. On écrit le contenu de AL dans AX en respectant le signe

Si AL contient un nombre positif, On complète par des 0 pour obtenir la représentation sur 16 bits.
Si AL contient un nombre négatif, On complète par des 1 pour obtenir la représentation sur 16 bits.
+5 = 0000 0101 alors AX devient 0000 0000 0000 0101
5 = 1111 1011 alors AX devient 1111 1111 1111 1011

- CWD (Convert Word to Double Word) effectue une extension de AX dans DX en respectant le signe. On écrit AX dans le registre 32 bits obtenu en collant DX et AX

4.7.3. Les instructions logiques

- NOT Op :Complément à 1 de l'opérande Op

- AND Od , Os : ET logique

Od ET Os → Od

- OR Od , Os : OU logique

Od OU Os → Od

- XOR Od , Os OU exclusif logique

Od OUX Os → Od

- TEST Od , Os Similaire à AND mais ne retourne pas de résultat dans Od, seuls les indicateurs sont positionnés

4.7.4. Les masques logiques :

Le 8086 ne possède pas d'instructions permettant d'agir sur un seul bit. Les masques logiques sont des astuces qui permettent d'utiliser les instructions logiques vues ci-dessus pour agir sur un bit spécifique d'un octet ou d'un mot.

- Forcer un bit à 0 : Pour forcer un bit à 0 sans modifier les autres bits, on utilise l'opérateur logique AND et ces propriétés :
  - ✓ x AND 0 = 0 (0 = élément absorbant de AND)
  - ✓ x AND 1 = x (1 = élément neutre de AND)

On fait un AND avec une valeur contenant des 0 en face des bits qu'il faut forcer à 0 et des 1 en face des bits qu'il ne faut pas changer.

	x	x	x	x		x	x	x	X
and	1	1	1	0		1	1	0	1
	x	x	x	**0**		x	x	**0**	x

- Forcer un bit à 1 : Pour forcer un bit à 1 sans modifier les autres bits, on utilise l'opérateur logique OR et ces propriétés :

  - ✓ x OR 1 = 1 (1 = élément absorbant de OR)
  - ✓ x OR 0 = x (0 = élément neutre de OR)

On fait un OR avec une valeur contenant des 1 en face des bits qu'il faut forcer à 1 et des 0 en face des bits qu'il ne faut pas changer

	x	x	x	x		x	x	x	x
OR	0	0	1	0		0	0	0	0
	x	x	**1**	x		x	x	x	x

- Inverser un bit : Pour inverser la valeur d'un bit sans modifier les autres bits, on utilise l'opérateur logique XOR et ces propriétés :

  - ✓ X XOR 1 = $\bar{X}$

✓ X XOR 0 = X (0 = élément neutre de XOR)

Donc, on fait un XOR avec une valeur contenant des 1 en face des bits qu'il faut inverser et des 0 en face des bits qu'il ne faut pas changer

	x	x	x	x		x	x	x	x
XOR	0	0	1	0		0	0	0	0
	x	x	$\overline{X}$	x		x	x	x	x

4.7.5. Les instructions de décalage

Ces instructions déplacent d'un certain nombre de positions les bits d'un mot vers la gauche ou vers la droite. Dans les décalages, 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 les instructions de décalage, l'opérande k peut être soit une constante (immédiat) soit le registre CL :

INST AX,1 ; décaler AX de 1 bit
INST BL,4 ; décaler BL de 4 bits
INST BX,CL ; décaler BX de CL bits

On peut aussi décaler le contenu d'une case mémoire mais il faut préciser la taille
INST byte [BX],1 ; décaler une fois le contenu de la case mémoire d'adresse BX.

- SHL R/M,k (SHift Logical Left) décalage logique à gauche de k bits

- SHR R/M,k (SHift Logical right) décalage logique à droite

- SAR R/M,k (SHift Arithmetic right) décalage arithmétique à droite

- SAL R/M,k (SHift Arithmétic Left) décalage arithmétique à gauche

Les décalages arithmétiques permettent de conserver le signe. Ils sont utilisés pour effectuer des opérations arithmétiques comme des multiplications et des divisions par 2.

- ROL R/M,k (Rotate Left) Rotation à gauche

- ROR R/M,k (Rotate Right) Rotation à droite

- RCL R/M,k (Rotate Through CF Left) Rotation à gauche à travers le Carry

- RCR R/M,k (Rotate Through CF Right) Rotation à droite à travers le Carry

4.7.6. Instructions agissant sur les indicateurs

- CLC (CLear Carry) positionne le drapeau C à 0
- STC (Set Carry) positionne le drapeau C à 1
- CMC Complémente le drapeau C
- CLD Positionne le Drapeau D à 0

- STD Positionne le Drapeau D à
- CLI Positionne le Drapeau I à 0
- STI Positionne le Drapeau I à 1
- LAHF : Copier l'octet bas du registre d'état dans AH
- SAHF : Opération inverse de LAHF : Transfert AH dans l'octet bas du registre d'état
- PUSHF : Empile le registre d'état,
- POPF : Dépile le registre d'état,

4.7.7. Les instructions de contrôle de boucle

- **LOOP xyz** L'instruction **loop** fonctionne automatiquement avec le registre CX (compteur). Quant le processeur rencontre une instruction loop, il décrémente le registre CX. Si le résultat n'est pas encore nul, il reboucle à la ligne portant l'étiquette xyz, sinon il continue le programme à la ligne suivante ;

L'étiquette est une chaîne quelconque qui permet de repérer une ligne. Le caractère ':' à la fin de l'étiquette n'est obligatoire que si l'étiquette est seule sur la ligne

- **LOOPZ xyz** (Loop While Zero) Décrémente le registre CX (aucun flag n'est positionné) on reboucle vers la ligne xyz tant que CX est différent de zéro et le flag Z est égal à 1. La condition supplémentaire sur Z, donne la possibilité de quitter une boucle avant que CX ne soit égal à zéro.

- **LOOPNZ xyz** Décrémente le registre CX et reboucle vers la ligne xyz tant que CX est différent de zéro et le flag Z est égal à 0. Fonctionne de la même façon que loopz,

4.7.8. Les instructions de branchement

3 types de branchement sont possibles :

- Branchements inconditionnels
- Branchements conditionnels
- Appel de fonction ou d'interruptions

Tous ces transferts provoquent la poursuite de l'exécution du programme à partir d'une nouvelle position du code. Les transferts conditionnels se font dans une marge de -128 à +127 octets à partir de la position de transfert.

4.7.8.1. Branchements inconditionnels

- **JMP xyz** Provoque un saut sans condition à la ligne portant l'étiquette xyz.

- **CALL xyz** Appel d'une procédure (sous programme) qui commence à la ligne xyz. La position de l'instruction suivant le CALL est empilée pour assurer une poursuite correcte après l'exécution du sous programme.

- **RET** Retour de sous programme. L'exécution du programme continue à la position récupérée dans la pile.

- **INT** n appel à l'interruption logicielle n° n

4.7.8.2. Branchements conditionnels

Les branchements conditionnels sont conditionnés par l'état des indicateurs (drapeaux) qui sont eux même positionnés par les instructions précédentes.

Dans la suite nous allons utiliser la terminologie :

- **supérieur** ou **inférieur** pour les nombres non signés
- **plus petit** ou **plus grand** pour les nombres signés
- + pour l'opérateur logique OU

- **JE/JZ xyz** (Jump if Equal or Zero) Aller à la ligne xyz si résultat nul ou si égalité. C'est-à-dire si Z=1

- **JNE/JNZ xyz** (Jump if Not Equal or Not Zero) Aller à la ligne xyz si résultat non nul ou si différent. C'est-à-dire si Z=0

- **JA xyz** (Jump if Above) aller à la ligne xyz si supérieur (non signé). C'est-à-dire si C + Z = 0

- **JAE xyz** (Jump if Above or Equal) aller à la ligne xyz si supérieur ou égal (non signé). C'est-à-dire si C = 0

- **JB xyz** (Jump if Bellow) Branche si inférieur (non signé). C'est-à-dire si C = 1

- **JBE xyz** (Jump if Bellow or Equal) aller à la ligne xyz si inférieur ou égal (non signé). C'est-à-dire si C + Z = 1

- **JC xyz** (Jump if CArry) aller à la ligne xyz s'il y a retenu. C'est-à-dire si C = 1

- **JNC xyz** (Jump if No CArry) aller à la ligne xyz s'il n'y a pas de retenu. C'est-à-dire si C = 0

- **JG xyz** (Jump if Grater) aller à la ligne xyz si plus grand (signé). C'est-à-dire si (S ^ O) + Z = 1

- **JGE xyz** (Jump if Grater or Equal) aller à la ligne xyz si plus grand ou égal (signé). C'est-à-dire si S ^ O = 0

- **JL xyz** (Jump if Less) aller à la ligne xyz si plus petit (signé). C'est-à-dire si S ^ O = 1
- **JLE xyz** (Jump if Less or Equal) aller à la ligne xyz si plus petit ou égal (signé). C'est-à-dire si (S ^ O) + Z = 1
- **JO xyz** (Jump if Overflow) aller à la ligne xyz si dépassement. C'est-à-dire si O = 1
- **JNO xyz** (Jump if No Overflow) aller à la ligne xyz s'il n'y a pas de dépassement O = 0
- **JP/JPE xyz** (Jump if Parity or Parity Even) aller à la ligne xyz si parité paire. C'est-à dire si P = 1
- **JNP/JPO xyz** (Jump if No Parity or if Parity Odd) aller à la ligne xyz si parité impaire. C'est-à-dire si P = 0
- **JS xyz** (Jump if Sign) aller à la ligne xyz si signe négatif. C'est-à-dire si S = 1
- **JNS xyz** (Jump if No Sign) aller à la ligne xyz si signe positif. C'est-à-dire si S= 0

4.8. Procédures

4.8.1. Notion de procédure

La notion de procédure en assembleur correspond à celle de fonction en langage C, ou de sous-programme dans d'autres langages.

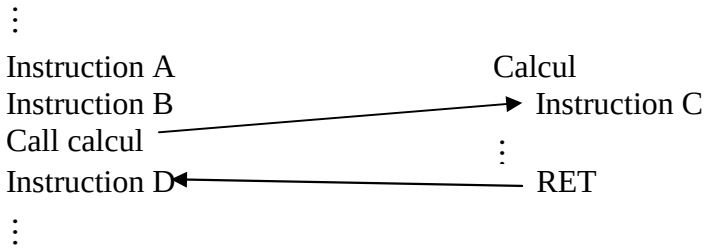


Figure 11: fonctionnement d'une procédure

Dans l'exemple de la Figure 11: fonctionnement d'une procédure

La procédure est nommée **calcul**. Après l'instruction B, le processeur passe à l'instruction C de la procédure, puis continue jusqu'à rencontrer RET et revient à l'instruction D.

4.8.2. Déclaration d'une procédure

L'assembleur possède quelques directives facilitant la déclaration de procédures. On déclare une procédure dans le segment d'instruction comme suit :

```
Calcul    Proc near    ; procédure nommée
                       calcul
          …            ; instructions


          RET          ; dernière instruction
Calcul    ENDP         ;fin de la procédure
```

Le mot clef PROC commence la définition d'une procédure, near indiquant qu'il s'agit d'une procédure située dans le même segment d'instructions que le programme appelant. L'appel s'écrit simplement : CALL Calcul

4.9. Méthodes de programmation

4.9.1. Les étapes de réalisation

Les étapes de réalisation d'un programme sont les suivantes:

- Définir la problématique, sous forme de schémas, d'organigrammes, de grafcet etc.
- Déterminer des algorithmes : comment faire, par quoi commencer etc.
- 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'`a obtention d'un programme fonctionnant de manière satisfaisante.

4.9.2. Langage machine et assembleur :

Le langage machine est le langage utilisant des codes binaires correspondant aux instructions ;

L'assembleur est un logiciel de traduction du code source écrit en langage assembleur (mnémoniques) vers le langage machine.

4.9.3. Réalisation pratique d'un programme :

On commence tout d'abord par la rédaction du code source en assembleur à l'aide d'un éditeur (logiciel de traitement de texte ASCII): (exemples edit sous MS-

DOS, notepad (bloc-note) sous Windows), ensuite on procède à l'assemblage du code source (traduction des instructions en codes binaires) avec un assembleur :(exemples : MASM de Microsoft, TASM de Borland, A86 disponible en shareware sur Internet, etc.) et ce pour obtenir le code objet : code machine exécutable par le microprocesseur.

Par la suite système d'exploitation ou à l'aide du moniteur d'une carte de développement, le code est chargé en mémoire centrale et le programme est exécuté.

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) qui permet d'exécuter pas à pas le programme, de visualiser le contenu des registres et de la mémoire et de poser des points d'arrêt etc.

4.9.4. Structure d'un fichier source en assembleur :

Afin de facilité la clarté de notre programme il est préférable d'organiser le code source de la manière suivante :

étiquettes	instructions	commentaires
Etiquette 1 :	Mov ax,05h	; ceci est un commentaire
⋮	⋮	⋮
Sous_prog1	Proc near	; sous programme
⋮	⋮	⋮
Sous_prog1	endp	

4.9.5. Directives pour l'assembleur :

- 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).

  Exemples :

```
nombre1 db 25
nombre2 dw ? ; pas de valeur initiale
buffer db 100 dup ( ?) ; réservation d'une zone mémoire de 100 octets
```

4.10. Les interruptions

Une interruption est une requête spéciale au microprocesseur pour lui indiquer un événement particulier nécessitant l'attention du microprocesseur.

Il existe trois types d'interruption :

- Interruption matérielle : causé par un des composants de l'unité centrale (souris, clavier, imprimante, etc.)
- Interruption logicielle : causé par un programme en cours d'exécution
- Exception : du à une erreur dans le programme en cours.

NB : Une interruption peut être initiée par le microprocesseur lui-même en cas de problèmes (exemple : division par zéro, mémoire défectueuse, etc.)

Une interruption est dite non masquée si elle est reconnue par le microprocesseur dés que le signal électrique a été déclenché. Elle est dite masquée sinon.

4.10.1. Les interruptions matérielles (externes)

Une interruption matérielle est un arrêt de l'exécution d'un programme suite à un **événement matériel.** Les demandes d'interruptions matérielles sont effectuées par les périphériques Signaux d'interruptions matérielles.

Les périphériques utilisent les **signaux d'interruption** du microprocesseur (RESET, NMI, et INTR) pour signaler un évènement.

Le µp 8086 contient 3 sources d'interruptions externes :

- Reset : réinitialisation du µp :

FLAG : 0000h
CS : FFFFh
IP : 0000h
DS : 0000h
ES : 0000h
SS : 0000h
L'adresse de démarrage est : FFFF0H

- NMI : (NO MASKABLE INTERUPT) : Cette interruption interrompe directement le CPU.

- INTR : (INTERUPT REQUASTE) : C'est une interruption autorisée si IF=1 sinon elle est masquée

4.10.2. Les interruptions logicielles

Une interruption logicielle est un arrêt de l'exécution d'un programme pour exécuter une routine d'interruption du **DOS** ou **BIOS**

Pour le 8086, les interruptions logicielles sont provoquées par l'instruction **INT** suivie du numéro d'interruption (exemple : INT 21h : pour les interruptions du DOS ; INT 14h : pour les interruptions du BIOS) ; L'instruction INTO est équivalente à INT 4 si overflow=1

Le déroulement d'une interruption logicielle est le suivant :

- Le μP reçoit l'instruction INT suivie du numéro d'interruption
- Le μP utilise le numéro d'interruption pour trouver le vecteur d'interruption
- Le μP met le flag IF=0 (IF est le flag d'interruption), sauvegarde dans la pile le registre d'état, et les registres CS et IP, et charge le vecteur d'interruption dans CS:IP
- Le μP exécute la routine d'interruption qui se termine par l'instruction IRET
- Le μP restaure le registres d'état et les registres CS et IP, et reprend l'exécution du programme en cours

Exemples d'interruptions logicielles :

Saisie avec écho d'un caractère au clavier :

```
MOV AH, 1
INT 21
--> Met dans le registre AL, le caractère lu au clavier.
```

Saisie sans écho d'un caractère au clavier

```
MOV AH, 8
INT 21
--> Met dans le registre AL, le caractère lu au clavier.
```

Affichage d'un caractère à l'écran

```
MOV AH, 2
INT 21
--> Affiche à l'écran le caractère contenu dans le registre DL.
```

4.10.3. Les exceptions

Une exception est la réponse normale du microprocesseur à une situation détectée lors de l'exécution d'une instruction et qui nécessite une gestion spéciale. Les exceptions sont internes au microprocesseur et ne peuvent pas être masquées.

Les exceptions ont une priorité plus élevée que les interruptions provenant des signaux d'interruption.

Interfaçage des microprocesseurs

Afin de pouvoir communiquer avec l'environnement extérieur (écran, souris, clavier, etc.), le microprocesseur utilise les interfaces d'entrées/sorties. Cette connexion se fait à travers les bus de données, d'adresses et de commande (voir Figure 12)

Dans ce chapitre, nous allons présenter quelques interfaces d'E/S à savoir l'interface parallèle 8255, l'interface série 8250,

Les accès à ces périphériques sont appelés les ports (exemple : port série, port parallèle, etc.)

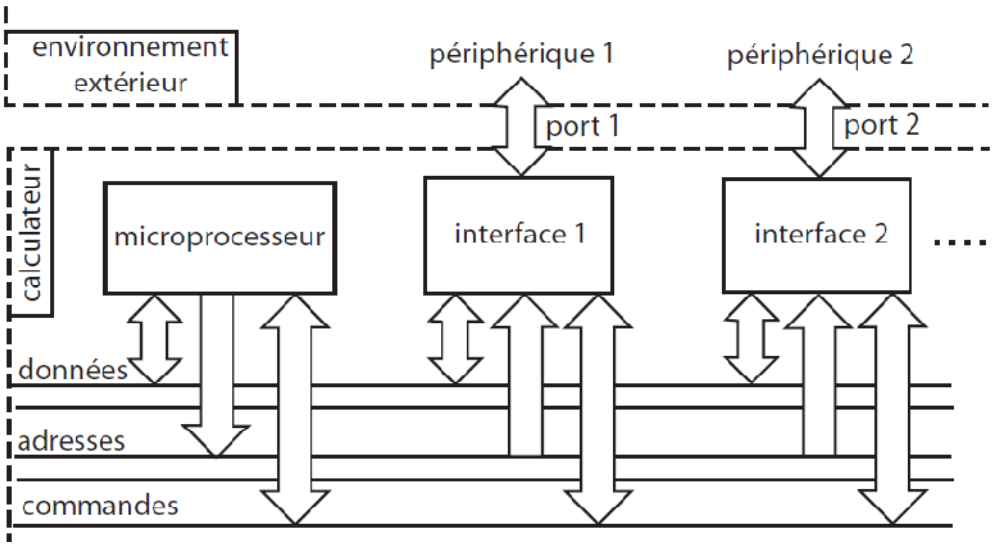


Figure 12: les interfaces d'E/S

Un circuit d'E/S est schématisé par la Figure 13.

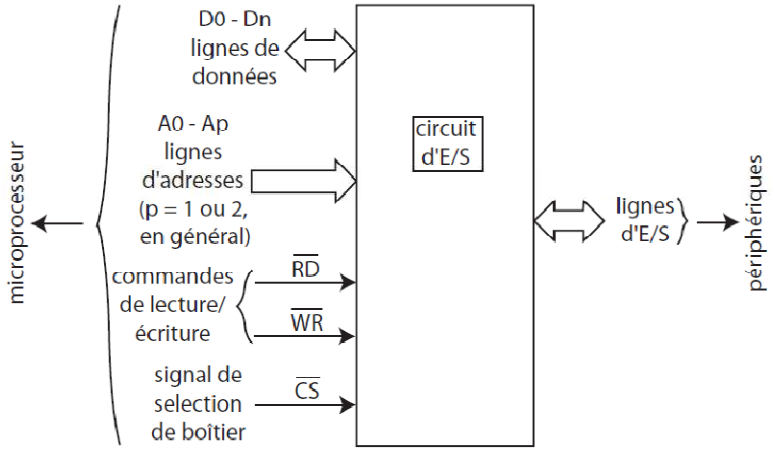


Figure 13: Schéma synoptique d'un circuit d'E/S

5.1. Adressage des ports d'E/S

Afin de gérer les échanges avec les périphériques, un circuit d'E/S possède des registres : les registres de données et les registres de configuration.

Une adresse est assignée à chacun de ces registres et le microprocesseur accède à un port en spécifiant l'un de ces registres.

Les adresses des ports d'E/S peuvent être vues par les microprocesseurs de deux manières :

- Les E/S sont mappées en mémoire : les adresses des ports d'E/S appartiennent au même espace mémoire que les circuits mémoire ; on parle alors d'adressage cartographique.

  Dans ce cas l'espace d'adressage des mémoires diminue, et l'adressage des ports se fait avec le même nombre de bits que pour les mémoires. Donc toutes les instructions possibles sur les mémoires peuvent être utilisées par les E/S.

- Le microprocesseur considere deux espaces distincts: l'espace d'adressage des mémoires et l'espace d'adressage des ports d'E/S. C'est le cas du microprocesseur 8086. On parle adressage indépendant.

  Dans ce cas l'espace mémoire adressable ne diminue pas et l'adressage des E/S peut se faire avec un nombre de bits moins important. Les instructions utilisées par les E/S sont quant à elles spécifiques.

  Une même adresse peut désigner une case mémoire ou un port d'E/S, le microprocesseur doit donc fournir un signale qui permet de faire la différence entre les deux.

Le 8086 dispose d'un espace mémoire de 1 Mo (adresse d'une case mémoire sur 20 bits) et d'un espace d'E/S de 64 Ko (adresse d'un port d'E/S sur 16 bits).

Le signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S est la ligne M/$\overline{IO}$: M/$\overline{IO}$ = 1 alors accès à la mémoire et M/IO = 0

pour un accès aux E/S. Ce signal est utilisé pour valider le décodage d'adresse dans les deux espaces :

Les instructions de lecture et d'écriture d'un port d'E/S sont respectivement les instructions IN et OUT. Elles placent la ligne M/IO à 0 alors que l'instruction MOV place celle-ci à 1.

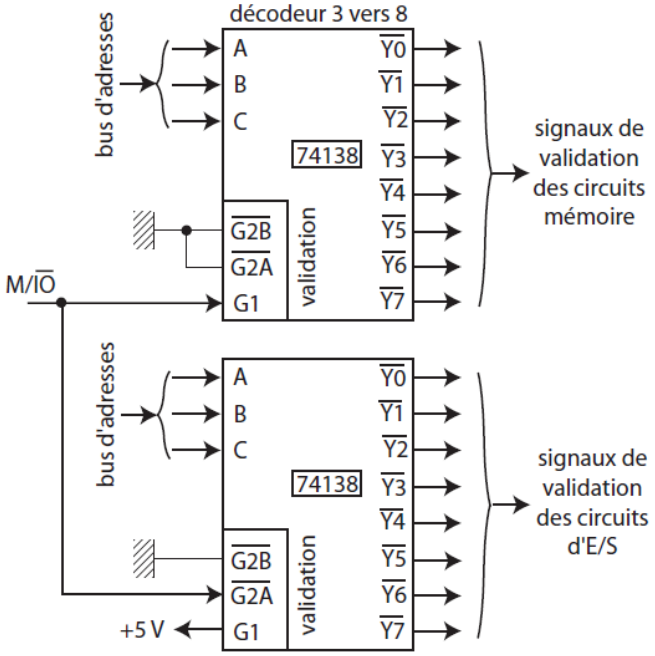


Figure 14: adressage des E/S par le 8086

Lecture d'un port d'E/S		
IN AL,adresse	lecture d'un port sur 8 bits	si l'adresse du port d'E/S est sur un octet
IN AX,adresse	lecture d'un port sur 16 bits	
IN AL,DX	lecture d'un port sur 8 bits	si l'adresse du port d'E/S est sur deux octets (DX contient l'adresse du port d'E/S
IN AX,DX	lecture d'un port sur 16 bits	
Ecriture d'un port d'E/S		
OUT adresse,AL	écriture d'un port sur 8 bits	si l'adresse du port d'E/S est sur un octet
OUT adresse,AX	écriture d'un port sur 16 bits	
OUT DX,AL	écriture d'un port sur 8 bits	si l'adresse du port d'E/S est sur deux octets (DX contient l'adresse du port d'E/S)
OUT DX,AX	écriture d'un port sur 16 bits	

Tableau 8: lecture et écriture d'un port d'E/S

Exemples :

1/ lecture d'un port d'E/S sur 8 bits `a l'adresse 300H

```
mov dx,300H        ; le registre dx contient la valeur 300
in al,dx           ; lecture du contenu du registre dx
```

2/ écriture de la valeur 1234H dans le port d'E/S sur 16 bits à l'adresse 49H

```
mov ax,1234H       ; le registre ax contient la valeur 1234h
out 49H,ax         ; écriture dans le port d'E/S à l'adresse 49h
```

5.2. L'interface parallèle 8255

Le rôle d'une interface parallèle est de transférer des données du microprocesseur vers des périphériques et inversement, tous les bits de données étant envoyés ou reçus simultanément.

Le 8255 est une interface parallèle programmable : elle peut être configurée en entrée et/ou en sortie par programme.

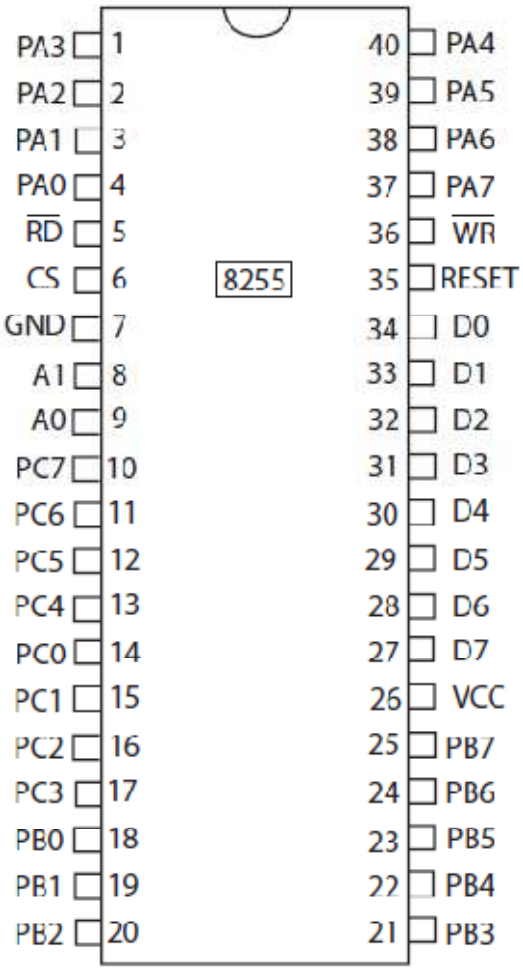


Figure 15: Brochage du 8255

La présente Figure 16 le schéma synoptique du 8255 :

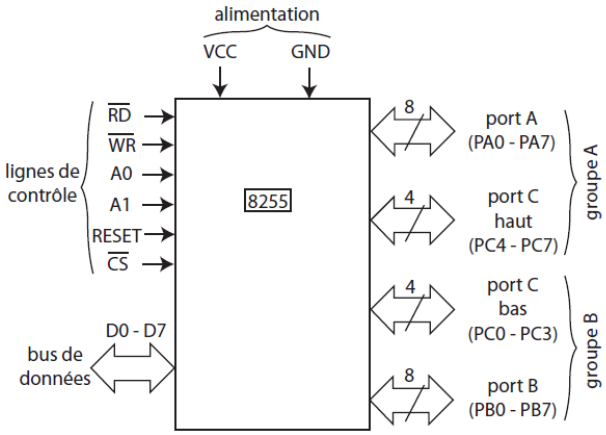


Figure 16: schéma synoptique du 8255

Le 8255 contient 4 registres : trois registres contenant les données présentes sur les ports A, B et C et un registre de commande pour la configuration des ports A, B et C en entrées et/ou en sorties.

Pour accéder aux registres du 8255, nous utilisons les lignes d'adresses A0 et A1 :

A1	**A0**	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	**operation**
0	0	0	1	0	lecture du port A
0	1	0	1	0	lecture du port B
1	0	0	1	0	lecture du port C
0	0	1	0	0	écriture du port A
0	1	1	0	0	écriture du port B
1	0	1	0	0	écriture du port C
1	1	1	0	0	écriture du registre de commande
X	X	X	X	1	pas de transaction
1	1	0	1	0	illegal
X	X	1	1	0	pas de transaction

Tableau 9: Configuration des registres du 8255

5.3. L'interface série 8250

Une interface série permet d'échanger des données entre le microprocesseur et un périphérique bit par bit. Ce qui permet de diminuer le nombre de connexions (1 fil pour l'émission, 1 fil pour la réception). Mais la vitesse de transmission devient plus faible que celle d'une interface parallèle.

Deux types de transmissions séries existent:

- asynchrone : chaque octet peut être émis ou reçu sans durée déterminée entre un octet et le suivant ;
- synchrone : les octets successifs sont transmis par blocs séparés par des octets de synchronisation.

La transmission asynchrone la plus utilisée est celle qui est définie par la norme RS232.

Exemple : transmission du caractère 'E' (code ASCII 45H = 01000101B) sous forme série selon la norme RS232 :

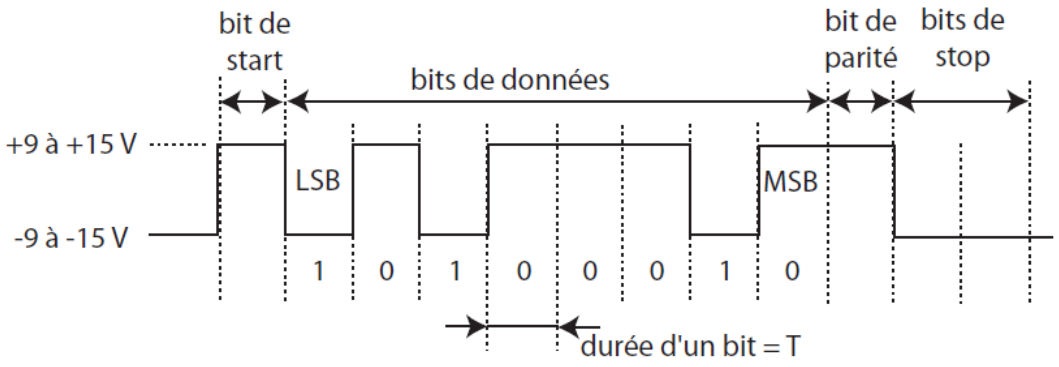


Figure 17: exemple de transmission série

- L'état 1 correspond à une tension négative comprise entre −9 et −15 V, l'état 0 à une tension positive comprise entre +9 et +15 V. Au repos, la ligne est à l'état 1 (tension négative) ;

- Le bit de start marque le début de la transmission du caractère ;

- Les bits de données sont transmis l'un après l'autre en commençant par le bit de poids faible. Ils peuvent être au nombre de 5, 6, 7 ou 8. Chaque bit est maintenu sur la ligne pendant une durée déterminée T. L'inverse de cette durée définit la fréquence de bit = nombre de bits par secondes = vitesse de transmission. Les vitesses normalisées sont : 50, 75, 110, 134.5, 150, 300, 600, 1200, 2400, 4800, 9600 bits/s ;

- Le bit de parité (facultatif) est un bit supplémentaire dont la valeur dépend du nombre de bits de données égaux à 1. Il est utilisé pour la détection d'erreurs de transmission ;

- Les bits de stop (1, 1.5 ou 2) marquent la fin de la transmission du caractère.

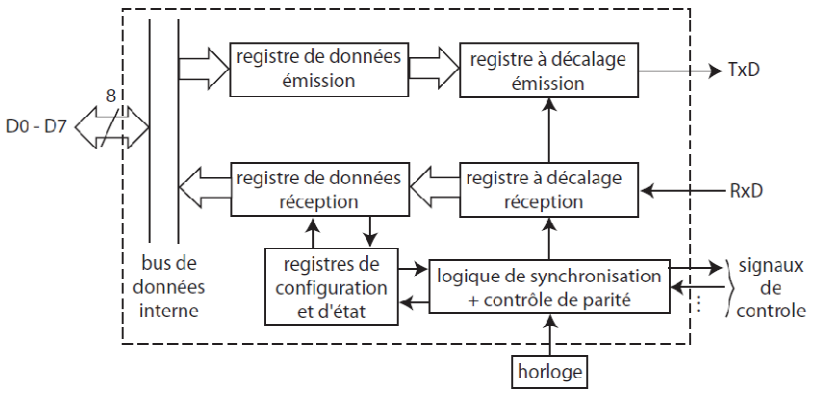


Figure 18: Principe d'une interface série

Un circuit intégré d'interface série asynchrone s'appelle un UART : Universal Asynchronous Receiver Transmitter) ; une interface série synchrone/asynchrone est un USART.

Exemples d'interfaces séries : 8251 (Intel) ; 8250 (National Semiconductor) ; 6850 (Motorola).

Etude et programmation des microcontrôleurs

Nous allons au cours de ce chapitre définir le microcontrôleur ainsi que les éléments contenus dans ce dernier et qui n'ont pas été abordé au cours des chapitres précédents afin de comprendre l'architecture d'un système à microcontrôleurs, le pic 16f877 sera utilisé comme exemple dans la suite du chapitre. Nous allons aussi voir la manière de programmer un microcontrôleur avec un langage évolué.

6.1. Définition

Un microcontrôleur se présente sous la forme d'un circuit intégré réunissant tous les éléments d'une structure à base de microprocesseur. Voici généralement ce que l'on trouve à l'intérieur d'un tel composant :

- Un microprocesseur (C.P.U.),
- De la mémoire de donnée (RAM et EEPROM),
- De la mémoire programme (ROM, OTPROM, UVPROM ou EEPROM),
- Des interfaces parallèles pour la connexion des entrées / sorties,
- Des interfaces séries (synchrone ou asynchrone) pour le dialogue avec d'autres unités,
- Des timers pour générer ou mesurer des signaux avec une grande précision temporelle,
- Des convertisseurs analogique / numérique pour le traitement de signaux analogiques.

Le fonctionnement d'un microcontrôleur est cadencé par une horloge généralement à base de quartz ou de résonateur céramique.

Il peut aussi contenir les composants suivants :

- Un Watchdog : (surveillance du programme)
- Une sortie PWM (modulation d'impulsion)
- Une interface I²C.

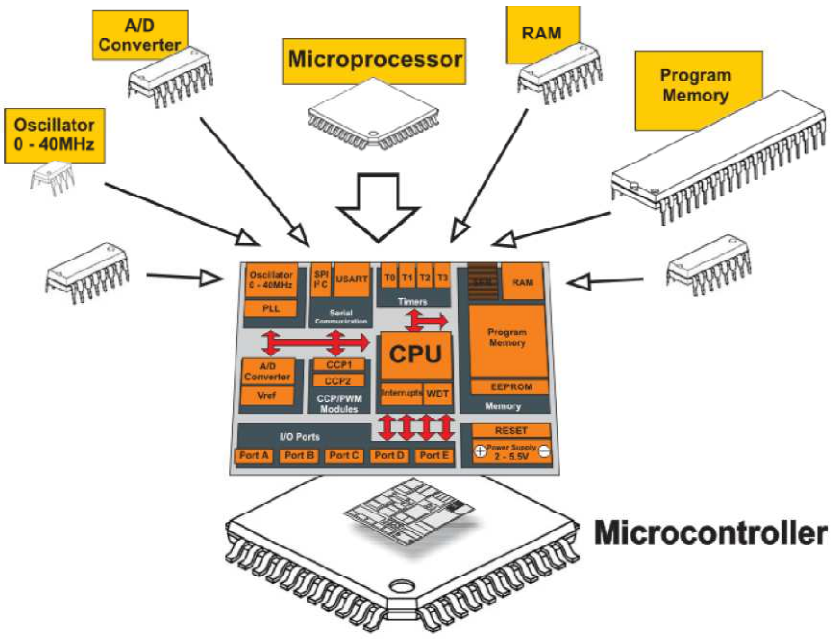


Figure 19: les composants d'un microcontrôleur

Les microcontrôleurs présentent donc les avantages suivants :

- Encombrement réduit,
- Faible consommation,
- Coût réduit
- Fiabilité
- Mise en œuvre plus simple

6.2. Les composants d'un microcontrôleur : (PIC 16F877)

6.2.1. Présentation du PIC 16F877

La famille des PICs est subdivisée en 3 grandes familles : La famille **Base-Line**, qui utilise des mots d'instructions de 12 bits, la famille **Mid-Range**, qui utilise des mots de 14 bits (et dont font partie le 16F877), et la famille **High-End**, qui utilise des mots de 16 bits.

Les éléments essentiels du PIC 16F877 sont :

- Consommation : moins de 2mA sous 5V à 4 MHz.
- Architecture RISC : 35 instructions de durée 1 ou 2 cycles.

- Durée du cycle : Période de l'oscillateur quartz divisée par 4 soit 200 ns pour un quartz de 20 MHz.
- Deux bus distincts pour le code programme et les data.
- Code instruction : mot de 14 bits et compteur programme (PC) sur 13 bits, ce
- qui permet d'adresser 8 K mots (de h'0000' à h'1FFF')
- Bus DATA sur 8 bits.
- 33 Ports Entrée-Sortie bidirectionnels pouvant produire 25 mA par sortie.
- PORTA = 6 bits et PORTB PORTC et PORTD = 8bits PORTE = 3 bits pour le
- 16F877 et 22 I/O seulement pour le 16F876.
- 4 sources d'interruption :

    - ✓ Externe par la broche partagée avec le Port B : PB0
    - ✓ Par changement d'état des bits du Port B: PB4 PB5 PB6 ou PB7
    - ✓ Par un périphérique intégré dans le chip: écriture de Data en EEPROM terminée, conversion analogique terminée, réception USART ou I2C.
    - ✓ Par débordement du Timer.

- 2 Compteurs 8 bits et 1 compteur 16 bits avec pré diviseur programmable.
- Convertisseur analogique 10 bits à 8 entrées.
- UART pour transmission série synchrone ou asynchrone.
- Interface I2C.
- 2 modules pour PWM avec une résolution de 10 bits.
- Interface avec un autre micro: 8 bits + 3 bits de contrôle pour R/W et CS.
- 368 Octets de RAM
- 256 Octets d'EEPROM Data.
- 8K mots de 14 bits en EEPROM Flash pour le programme (h'000' à h'1FFF').
- 1 registre de travail : W et un registre fichier : F permettant d'accéder à la RAM ou aux registres internes du PIC. Tous les deux sont des registres 8 bits.

- **PORTA :** 6 entrées -sorties. 5 entrées du CAN. Entrée CLK du Timer 0.

- **PORTB :** 8 entrées-sorties. 1 entrée interruption ext. Clk et Data pour prog.

- **PORTC :** 8 entrées-sorties. Clk Timer1 et PWM1. USART. I2C.

- **PORTD :** 8 entrées-sorties. Port interface micro processeur (8 bits data).

- **PORTE :** 3 entrées-sorties. 3 bits de contrôle. 3 entrées du CAN.

6.2.2. Organisation de la mémoire du 16F877

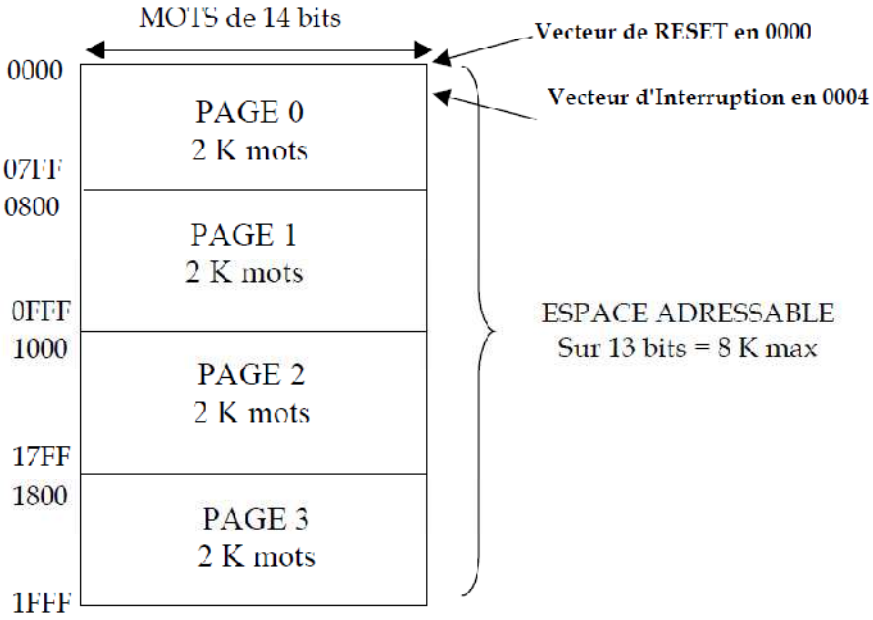


Figure 20: organisation de l'espace mémoire du 16F877

Les 2 bits MSB des 13 bits d'adresse (bits 11 et 12), viennent du registre PCLATH (bits 3 et 4) qui est à l'adresse : h'0A'. Il faut impérativement les positionner pour la bonne page, avant d'utiliser les instructions: CALL et GOTO.

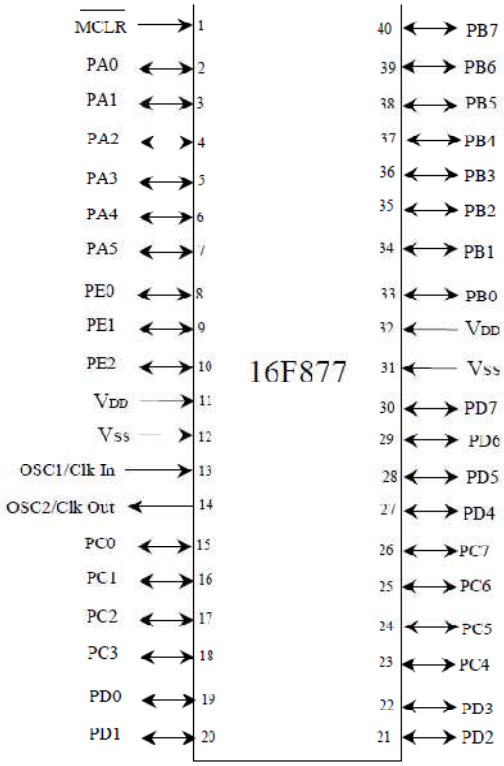


Figure 21: Brochage du 16F877

Dans la suite nous allons présenter quelques éléments de ce microcontrôleur, néanmoins pour plus de détail il faut se référer au datasheet.

6.2.3. Présentation de quelques registres internes

Un microcontrôleur présente plusieurs registres internes permettant la configuration de ce dernier, nous allons au cours de ce paragraphe présenter quelques un de ces registre afin d'avoir une idée sur leur utilité.

6.2.3.1. Le registre OPTION : (h'81' ou h'181').

Ce registre en lecture écriture permet de configurer les prédiviseurs du Timer et du Watchdog, la source du Timer, le front des interruptions et le choix du Pull up sur le Port B

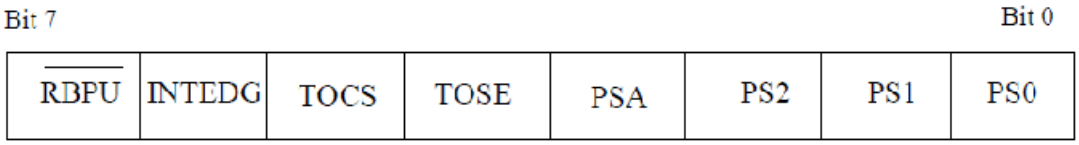


Figure 22: le registre OPTION

Au reset : OPTION = 11111111

Bit 7 : $\overline{\boldsymbol{RBPU}}$ = Pull up Enable bit on Port B.
1 = Pull up désactivé sur le Port B.
0 = Pull up activé.

Bit 6 : **INTEDG** = Interrupt Edge select bit.
1 = Interruption si front montant sur la broche PB0/IRQ (pin 6).
0 = Interruption si front descendant sur PB0/IRQ.

Bit 7 Bit 0
RBPU INTEDG TOCS TOSE PSA PS2 PS1 PS0
DOC PIC 16F876 et 16F877 D. 8 *MENESPLIER ENAC/ELE 2001*

Bit 5 : **TOCS** = Timer TMR0 Clock Source select bit.
1 = L'horloge du Timer est l'entrée PA4/Clk (pin 3).
0 = Le Timer utilise l'horloge interne du PIC.

Bit 4 : **TOSE** = Timer TMR0 Source Edge select bit.
1 = Le Timer s'incrémente à chaque front montant de la broche PA4/Clk.
0 = Le Timer s'incrémente à chaque front descendant de la broche PA4/Clk.

Bit 3 : **PSA** = Prescaler Assignement bit.
1 = Le prédiviseur est affecté au watchdog..
0 = Le prédiviseur est affecté au Timer TMR0.

Bits 2 à 0 : **PS2 PS1 PS0** = Prescaler Rate Select bits.

Quand le prédiviseur est affecté au Watchdog (PSA=1), TMR0 est prédivisé par 1.

6.2.3.2. Le registre INTCON : (h'0B' ou h'8B' ou h'10B' ou h'18B').

Ce registre en lecture écriture permet de configurer les différentes sources d'interruption.

Au reset : INTCON = 0000000X

Bit 7 : **GIE** = Global Interrup Enable bit
1 = Autorise toutes les interruptions non masquées.

0 = Désactive toutes les interruptions.

Bit 6 : **PEIE** = Peripheral Interrupt Enable bit.

1 = Autorise les interruptions causées par les périphériques.

0 = Désactive les interruptions causées par le périphériques.

Bit 5 : **TOIE** = Timer TMR0 Overflow Interrup Enable bit.

1 = Autorise les interruptions du Timer TMR0.

0 = Désactive les interruptions du Timer TMR0.

Bit 4 : **INTE** = RB0/Int Interrup Enable bit.

1 = Autorise les interruptions sur la broche : PB0/IRQ (pin6).

0 = Désactive les interruptions sur la broche : PB0/IRQ (pin6).

Bit 3 : **RBIE** = RB Port Change Interrup Enable bit.

1 = Autorise les interruptions par changement d'état du Port B (PB4 à PB7).

0 = Désactive les interruptions par changement d'état du Port B (PB4 à PB7).

Bit 2 : **TOIF** = Timer TMR0 Overflow Interrup Flag bit.

1 = Le Timer à débordé. Ce flag doit être remis à zéro par programme.

0 = Le Timer n'a pas débordé.

Bit 1 : **INTF** = RB0/Int Interrup Flag bit.

1 = Une interruption sur la broche PB0/IRQ (pin 6) est survenue.

0 = Pas d' interruption sur la broche PB0/IRQ (pin 6).

Bit 0 : **RBIF** = RB Port Change Interrup Flag bit. Ce flag doit être remis à zéro par programme.

1 = Quand au moins une entrée du port B (de PB4 à PB7) a changé d'état.

0 = Aucune entrée de PB4 à PB7 n'a changé d'état.

6.2.4. LES PORTS ENTREE / SORTIE

Les ports d'entrée / sortie numériques peuvent être considérés comme les périphériques les plus simples du microcontrôleur. Pour le PIC, on contrôle leur fonctionnement à l'aide de registres spéciaux (deux registres par port). Par exemple, pour le port A, on a le registre PORTA et le registre TRISA.

Les registres « TRISx » contrôlent le mode de fonctionnement des entrées / sorties : selon la valeur de chaque bit, 0 ou 1, la pin correspondante du port fonctionnera soit en sortie, soit en entrée. Les registres « PORTx » contiennent les données, lues, ou à écrire sur le port. Certains ports possèdent en plus des fonctionnalités spécifiques : le PORTB est équipé de résistances de tirage (pull-up) internes qui peuvent être activées ou non. La pin RB0 du PORTB sert d'entrée d'interruption externe, et les pins RB4 à RB7 peuvent générer une interruption lorsqu'un changement d'état est détecté à leurs bornes. Toutes ces particularités sont bien entendu décrites dans les documentations Microchip spécifiques à chaque composant.

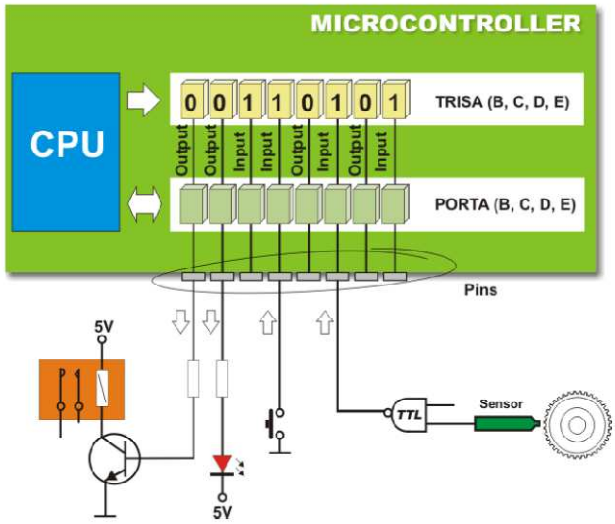


Figure 23: configuration des ports d'E/S

6.2.5. CONVERTISSEUR A/D

Il est constitué d'un module convertisseur à 8 entrées. Les 5 premières entrées sont sur le Port A en PA0, PA1, PA2, PA3 et PA5, les 3 autres sont en PE0, PE1 et PE2.

Le résultat de la conversion est codé sur 10 bits. C'est une valeur comprise entre h'000' et h'3FF'.

Les 4 registres utilisés par le module convertisseur A/D sont :
- ADRESH en h'1E' page 0 : MSB des 10 bits du résultat.
- ADRESL en h'9E' page 1 : LSB des 10 bits du résultat.
- ADCON0 en h'1F' page 0 : registre de contrôle n°0 du convertisseur.
- ADCON1 en h'9F' page 1 : registre de contrôle n°1 du convertisseur.

ADCON0 : (h'1F' : page 0)

Bit 7							Bit 0
ADSC1	ADSC0	CHS2	CHS1	CHS0	GO/$\overline{\text{Done}}$		ADON

Au reset : ADCON0 = 00000000
Bit 7 et bit 6 : **ADSC1 et ADSC0** = Clock Select bits.

Ces 2 bits permettent de choisir la vitesse de conversion :
00= Fosc/2.
01= Fosc/8.
10= Fosc/32.
11= Oscillateur RC interne.

Le temps de conversion d'un bit est TAD. Pour une conversion totale des 10 bits il faut : 12.TAD.

Bit 5 bit4 et bit 3 : **CHS2 CHS1 et CHS0** = Channel Select bits.
Ces 3 bits permettent de choisir l'entrée qui va être convertie.

Canal	CHS2	CHS1	CHS0	PORT
0	0	0	0	PA0
1	0	0	1	PA1
2	0	1	0	PA2
3	0	1	1	PA3
4	1	0	0	PA5
5	1	0	1	PE0
6	1	1	0	PE1
7	1	1	1	PE2

Bit 2 : **GO/DONE :** Status bit si ADON=1.
1 = Démarre la conversion A/D. Ce bit est remis à "0" par hard.
0 = La conversion A/D est terminée.
Bit 1 : **Bit non implanté.**
Bit 0 : **ADON :** A/D on bit.
1= Convertisseur A/D en service.
0 = Convertisseur A/D à l'arrêt.

ADCON1 : (h'9F' : page 1).
Au reset : ADCON1 = 00000000
Bit 7 : **ADFM** = A/D Result format.
1 = Justifié à droite. ADRESH ne contient que les 2 MSB du résultat. Les 6 MSB de ce registre sont lus comme des "0".
0 = Justifié à gauche. ADRESL ne contient que les 2 LSB du résultat. Les 6 LSB de ce registre sont lus comme des "0".

Bit 6 bit 5 et bit 4 : **Bits non implémentés.**
Bit 3 bit 2 bit 1 et bit 0 : **PCFG3 PCFG2 PCFG1 et PCFG0**
Bits de contrôle de la configuration des Ports.
Ces bits permettent de choisir le partage entre entrées analogiques et digitales sur les PORTS A et E.

A = Entrée Analogique.
D = I/O Digitale.

4 Bits PCFG	N°7 PE_2	N°6 PE_1	N°5 PE_0	N°4 PA_5	N°3 PA_3	N°2 PA_2	N°1 PA_1	N°0 PA_0	V_{REF^+}	V_{REF^-}
0 0 0 0	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}
0 0 0 1	A	A	A	A	V_{REF^+}	A	A	A	PA_3	V_{SS}
0 0 1 0	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}
0 0 1 1	D	D	D	A	V_{REF^+}	A	A	A	PA_3	V_{SS}
0 1 0 0	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}
0 1 0 1	D	D	D	D	V_{REF^+}	D	A	A	PA_3	V_{SS}
0 1 1 x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}
1 0 0 0	A	A	A	A	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 0 0 1	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}
1 0 1 0	D	D	A	A	V_{REF^+}	A	A	A	PA_3	V_{SS}
1 0 1 1	D	D	A	A	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 1 0 0	D	D	D	A	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 1 0 1	D	D	D	D	V_{REF^+}	V_{REF^-}	A	A	PA_3	PA_2
1 1 1 0	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}
1 1 1 1	D	D	D	D	V_{REF^+}	V_{REF^-}	D	A	PA_3	PA_2

Figure 24: configuration du registre ADCON1

Au reset le registre ADCON1 est initialisé à h'00'. Cela signifie que les 5 bits du Port A et les 3 bits du Port E sont configurés en entrées analogiques.

Pour récupérer le 5 bits du Port A et les 3 bits de Port E en tant que I/O digitales il faut écrire la valeur h'06' dans ADCON1.

6.2.6. Les timers

Un timer est le nom courant de compteur / temporisateur. Il sert à :

- Mesurer du temps (compter le nombre de coup d'horloge) > **Mode temporisateur**
- Compter le nombre d'évènement sur une broche (exemple : Nombre d'appuis sur un bouton poussoir > **Mode compteur**

Il arrive souvent qu'on désire introduire des temporisations pendant l'exécution d'un programme. Le PIC 16F877 dispose de 3 timers permettant de gérer le temps avec précision. L'idée est d'initialiser une variable à une valeur donnée et ensuite la décrémenter en boucle jusqu'a ce qu'elle atteint 0. Connaissant le temps d'exécution de chaque instruction, on peut calculer le temps que mettra le processeur à terminer la boucle de décrémentation.

En pratique, on visualise la valeur de départ, puis la valeur d'arrivée. La valeur de comptage est la différence des deux valeurs.

6.2.7. LE CHIEN DE GARDE (Le Watchdog Timer WDT)

Ce dispositif est un système anti-plantage du microcontrôleur. Il s'assure qu'il n'y ait pas d'exécution prolongé d'une même suite d'instruction.

Pour le 16F877 c'est un compteur 8 bits incrémenté en permanence (même si le μC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT TimeOut), deux situations sont possibles :

- Si le μC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester plante en cas de blocage du microcontrôleur par un processus indésirable non contrôle.
- Si le μC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement la ou elle s'est arrêtée avant de rentrer en mode SLEEP.

Cette situation est souvent exploitée pour réaliser des temporisations

6.3. Les étapes de réalisation d'une application à base de pic :

6.3.1. Les outils nécessaires

Pour développer une application fonctionnant à l'aide d'un microcontrôleur, il faut disposer d'un compilateur et d'un programmateur et éventuellement d'un logiciel de simulation.

Le compilateur est un logiciel traduisant un programme écrit dans un langage donné (C, basic, assembleur, etc.) en langage machine. Ce logiciel peut aussi comporter un « debugger » permettant la mise au point du programme, et un simulateur permettant de vérifier son fonctionnement, et le programmateur sert à transférer le programme du pc au pic. La simulation peut être effectuée par un logiciel indépendant du compilateur.

Donc afin de pouvoir réaliser une application à base de pic, nous devons tout d'abord commencer par écrire le programme dans un éditeur de texte indépendant ou dans l'éditeur du compilateur choisi et ce avec le langage évolué choisi (C, Pascal, basic, etc.), ensuite il faut utiliser le compilateur afin de générer le code machine, et c'est donc un fichier en .hex qui est généré.

Plusieurs compilateur existent, nous pouvons citer : mikroC, mikroBasic, mikropascal de mikroelektronika, MPLAB IDE de microchip ; *Pic* Basic *Compiler* de Selectronic etc.

Une fois nous avons notre code machine, nous avons le choix de le simuler sur un logiciel de simulation tel que ISIS ou de l'envoyer à l'aide du programmateur et de son logiciel au pic. Le programmateur permet de transférer le programme compilé (langage machine) dans la mémoire du microcontrôleur. Il est constitué d'un circuit branché sur le port COM du PC ou USB, sur lequel on implante le PIC, et d'un logiciel permettant d'assurer le transfert. Il existe différents logiciels, nous citons : Icprog, ICSP, picflash 2 usb etc.

Dans la suite nous allons utiliser le compilateur mikroC de mikroelektronika, le logiciel de simulation ISIS et le programmateur ICprog ;

6.3.2. Architecture d'un programme C pour mikroC

La saisie d'un programme en "C" répond pratiquement toujours à la même architecture. On peut noter que le symbole "#" est suivi d'une directive de compilation, le symbole "//" est suivi d'un commentaire.

```
#include<sdtio.h>          // Directive de compilation indiquant d'inclure la bibliothèque E/S standard //
#include <reg_uc.h>        //Directive de compilation indiquant d'inclure la bibliothèque spécifique au 0

#define clear=OxOO         // Directive de compilation indiquant des équivalences

char val1=0xA5;            // Déclaration d'une variable "caractère" avec valeur initiale
int val2;                  // Déclaration d'une variable "nombre entier"
.
.
.
void tempo(char temps) {   // Fonctions et procédures appelées plusieurs fois dans le programme
                           principal

}
int bintobcd(char bin) {
…

return ...;
}

void main (void)           // Programme principal
{
   DDRBA=0xFF              // initialisation et configuration

   while (1)               // Boucle principale
   {
    …
    tempo(100);
    …
   val2=bintobcd(val1);
  }
}

void nmi(void)interrupt 0
{                              // Sous programme d'interruption
```

Chaque ligne d'instruction se termine par un ";". Le début d'une séquence est précédé du symbole "{". La fin d'une séquence est suivie du symbole "}".

La notation des nombres peut se faire en décimal de façon normale ou en hexadécimal avec le préfixe "0x".

6.3.3. miKroC et exemple

Nous allons au cours de ce paragraphe présenter le compilateur mikroC, et réaliser un exemple afin de comprendre le fonctionnement.

Le compilateur *mikroC* pour *PIC* sauvegarde vos applications au sein de projets qui s'apparentent à un fichier 'projet' unique (avec l'extension .ppc) ainsi qu'à un ou plusieurs fichiers sources (avec l'extension .c). L'environnement *IDE* du compilateur *mikroC* pour PIC *ne* permet la gestion que d'un seul projet à la fois. Les fichiers sources ne peuvent être compilés que s'ils font partis d'un projet.

Un fichier projet renferme les informations suivantes:

- Nom du projet et description optionnelle;
- Composant cible;
- Options du composant (configuration word);
- Fréquence d'horloge du composant;
- Liste des fichiers sources du projet source;
- Fichiers binaires (*.mcl);
- Autres fichiers.

Dans ce paragraphe, nous allons apprendre à créer un nouveau projet, écrire un code, le compiler avec *mikroC* pour *PIC* et tester le résultat. Notre exemple permettra de faire clignoter des LEDs afin qu'il soit facile de le tester sur un microcontrôleur PIC.

Il faut donc installer le compilateur et créer un nouveau projet, le donner un nom ainsi que le chemin du dossier et choisir le pic à utiliser dans le champ device. Pour les autres réglages nous allons garder leurs valeurs par défaut. Une fenêtre vierge s'ouvre et c'est là ou nous allons écrire notre programme en C.

Le programme est le suivant :

```
void main() {
  PORTC = 0x00;              // initialisation du PORTC
  TRISC = 0;              // Configurer tout le PORTC comme sortie

  while(1) {
    PORTC =  ~PORTC;// changer la valeur du PORTC de 0 à 1 et inversement
    Delay_ms(1000);         // attendre pendant une seconde
  }
}
```

Une fois le programme écrit nous allons le compiler. Dans le dossier du projet nous allons retrouver notre code source en C dans un fichier dont l'extension est .ppc et notre fichier en code machine dont l'extension est .hex et c'est ce dernier que nous allons utiliser pour la simulation ou pour être transférer au pic via le programmateur.

Nous allons donc utiliser le logiciel PROTEUS ISIS pour simuler le programme.

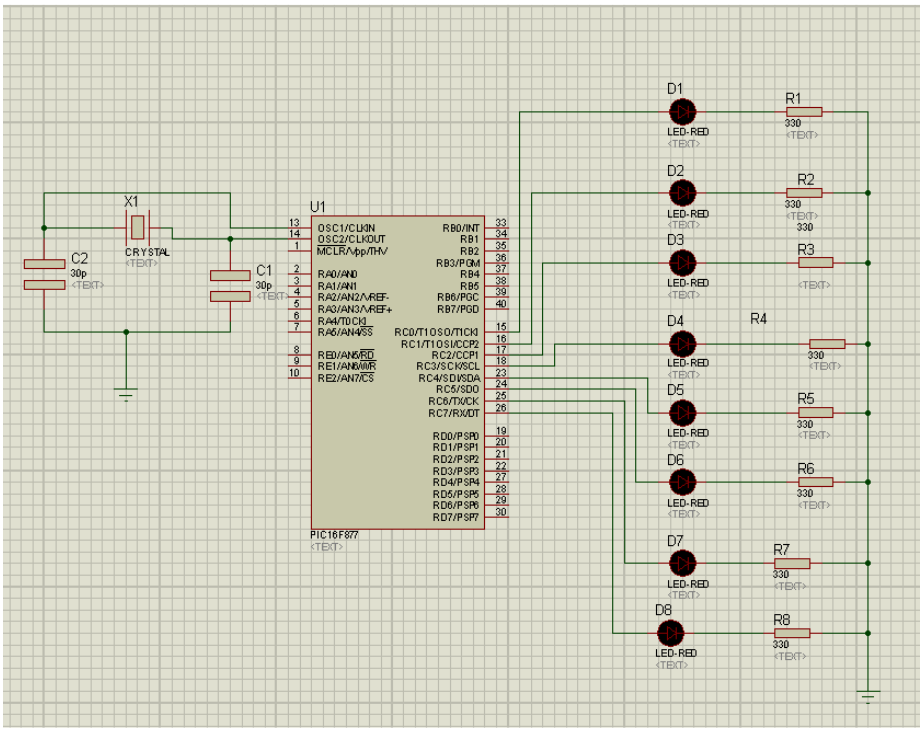


Figure 25: Schéma sous ISIS

Nous réalisons donc le schéma sous ISIS et nous éditons les propriétés du pic afin de l'indiquer le fichier .hex généré par le compilateur mikroC ainsi que la fréquence de l'horloge. Et en lançant la simulation, nous pourrons voir les LEDs s'allumer et s'éteindre toutes les secondes.

Nous pourrons par la suite transférer notre programme sur le pic via le programmateur qu'on peut aussi fabriquer nous même.

6.3.4. Quelques applications classiques :

Les microcontrôleurs peuvent être utilisés pour réaliser des applications diverses, nous pourrons trouver des applications basique dans le fichier d'aide de mikroC. Nous allons citer quelques uns dont la conception est assez simple :

- Afficher un texte sur un afficheur lcd
- Convertir un signal analogique en numérique et afficher le résultat sur des leds
- Convertir un signal analogique et afficher le résultat sur un afficheur lcd utilisant les commandes lcd.
- Envoyer un signal en utilisant la liaison série et les commandes USART

Bibliographie

http://www.lrde.epita.fr/~didier/lectures/os_00_intro.pdf
iutbayonne.univ-pau.fr/pub/perso/Info/dalmau/travail/textes/pedag/polycops/archi/microprocesseurs/lesmicro.doc.
http://www-gtr.iutv.univ-paris13.fr/Cours/Mat/Architecture/Cours/polyarch/chap-3_sec-3_sec-3.html
http://intranet-gei.insa-toulouse.fr:8181/Enseignements/SFO/Polycopie/index.html?part=ID_PubliTool_N12E6E
http://www-gtr.iutv.univ-paris13.fr/Cours/Mat/Architecture/Cours/polyarch.pdf
http://richard.grisel.free.fr/3-Cours_microprocesseur-16-bits.pdf
http://www.oumnad.123.fr/
http://www.sfa.univ-savoie.fr/formations/masters/electronique-telecoms/wp-content/files/ETRS-604/cours/Cours%20Microprocesseur-Microcontroleur.pdf

[Bey04] Beyondlogic _USB in a Nutshell_. www.beyondlogic.org, 2004.
[HWe98] Hirsch E., Wendling S. _Structure des ordinateurs_. Armand Colin, 1998.
[Pea97] Peacock C. _Using Interrupts, Interfacing the Serial and Parallel ports_. www.senet.com.au/ cpeacock, 1997.
[Tan03] Tanenbaum A. _Architecture de l'ordinateur_. 4ème édition. Dunod, 2003.
[TCGG02] Tichon J., Couwenbergh C., Giot R. et Garcia Acevedo S. _Communication avec les périphériques_. Techniques de l'Ingénieur, traité Informatique Industrielle, 2002.