MICROPROCESSEURS
ET
MICROCONTROLEURS

Docteur- Ingénieur

Alphonse AZOMA

SOMMAIRE

PREMIERE PARTIE : LES MICROPROCESSEURS

CHAPITRE 1 : Généralités sur les microprocesseurs

1.1 Introduction

1.2 Architecture de base d'un microprocesseur

1.3 Cycle d'exécution d'une instruction

1.4 Jeu d'instructions

1.5 Langage de programmation

1.6 Exercices

CHAPITRE 2 : Etude du microprocesseur 8086

2.1 Architecture externe du 8086

2.2 Architecture interne du 8086

2.3 Gestion de la mémoire par le 8086

2.4 Le microprocesseur 8088

2.5 Programmation en assembleur du microprocesseur 8086

2.6 Jeu d'instructions du 8086

2.7 Méthodes de programmation

2.8 Interfaces d'entrées-sorties

2.9 Les interruptions

DEUXIEME PARTIE : LES MICROCONTROLEURS

CHAPITRE 3 : Les microcontrôleurs

3.1 Du microprocesseur au microcontrôleur

3.2 Généralités sur le microcontrôleur PIC

3.3 Etude du microcontrôleur PIC 16F877

3.4 Programmation en langage C des PIC

3.5 Les interruptions

3.6 Le CAN

3.7 Les Timers

Première partie : les microprocesseurs

CHAPITRE 1 : Généralités sur les microprocesseurs

1. Introduction aux systèmes micro-programmés

Le développement de l'électronique numérique a suscité l'apparition de plusieurs types de composants très puissants en particulier les systèmes micro-programmés.

Leur aptitude à s'adapter aux contraintes technologiques de plus en plus complexes, leur capacité de gérer un grand nombre de fonctionnalités variées et leur coût de revient faible ont encouragé leur utilisation dans plusieurs applications tant domestiques qu'industrielles.

Le développement de ces composants programmables remplacent de plus en plus l'électronique classique vu que les circuits intégrés analogiques ou logiques ne peuvent plus résoudre des fonctions de plus en plus complexes.

Historiquement, les constructeurs développèrent d'abord les systèmes micro-programmés intégrés dans les calculateurs de bureau ou de poche, avec des codes d'ordre orientés vers le calcul numérique. Puis maitrisant cette technique ils offrirent des circuits d'usage généraux : les microprocesseurs.

Les systèmes micro-programmés ont vu leur importance progresser au rythme de l'importance prise par les microprocesseurs. C'est d'ailleurs l'élément de base pour de tels systèmes.

En 1971 est apparu le premier microprocesseur 4 bits 4004 d'Intel .

La miniaturisation des transistors a permis d'augmenter considérablement la capacité d'intégration sur silicium. On est passé rapidement du processeur 4 bits au :

• processeur 8 bits.
• processeurs 16 bits.
• processeurs 32 bits.
• processeurs 64 bits

Cette miniaturisation a offert des possibilités de réaliser des systèmes embarquées.

2. Définition d'un microprocesseur

Un microprocesseur ou processeur ou encore CPU (Central Processing Unit) est l'unité intelligente de traitement des informations. c'est un circuit intégré à très grande échelle d'intégration (VLSI) chargé d'organiser les tâches précisées par le programme, de les décoder et d'assurer leur exécution. Il doit aussi prendre en compte les informations extérieures au système et assurer leur traitement. Ils présentent trois avantages principaux : ils sont économiques et offre une souplesse d'emploi inhérente à la programmation.

3. Modèle de base d'un microprocesseur

Dans un microprocesseur, on retrouve :

- une Unité Arithmétique et Logique (UAL)
- une Unité de Contrôle (UC)
- des registres
- des bus ou chemins de données

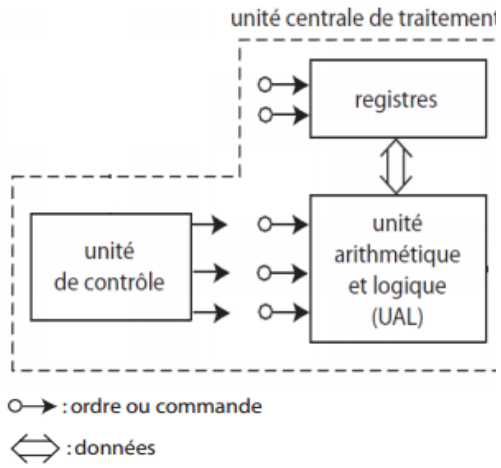


Fig. 1.1 : Structure de base d'un microprocesseur

3.1 Unité Arithmétique et Logique

Elle dispose de circuits réalisant des opérations des fonctions logiques (ET, OU, comparaison, décalage,...) ou arithmétique (addition, soustraction,...).

En entrée de l'UAL, on a des commandes permettant d'activer les opérations, venant de l'unité de contrôle En sortie, on a les résultats des opérations et les conditions qui sont en fait les entrées de l'unité de contrôle.

L'UAL est composé de : Les accumulateurs : Ce sont des registres de travail qui servent à stocker une

opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.

- L'Unité Arithmétique et Logique: C'est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage, etc...) ou arithmétique (Addition, soustraction...)
- Le registre d'état : Il est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle indicateur d'état ou flag

ou drapeaux (Retenue, débordement, zéro, ...).

3.2 Unité de contrôle ou séquenceur

L'unité de contrôle est une circuit logique séquentiel chargée de séquencer l'algorithme et de générer les signaux de contrôle pour piloter les éléments du chemin de données.
Elle envoie des commandes à l'unité de traitement qui va exécuter les traitements.

L'unité de contrôle contient:

- Le compteur de programme (PC : Programme Counter) appelé aussi Compteur Ordinal (CO) : Il est constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de la prochaine instruction à exécuter
- Le registre d'instruction et le décodeur d'instruction : Chacune des instructions à exécuter est transféré depuis la mémoire dans le registre instruction puis est décodée par le décodeur d'instruction.
- Bloc logique de commande (ou séquenceur) : IL organise l'exécution des instructions au rythme d'une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction de l'instruction qu'il a à exécuter. Il s'agit d'un automate réalisé de façon microprogrammée

3.3 Les bus

Un bus est un ensemble de lignes de communications groupés par fonction. Il permet de faire transiter (liaison série/parallèle) des informations codées en binaire entre deux points.
Il est caractérisé par le nombre de lignes et la fréquence de transfert.
Il existe 3 Types de bus :

- **Bus de données (bi-directionnel):** permet de transférer entre composants des données ex. : *résultat d'une opération, valeur d'une variable, etc.*

Le nombre de lignes du bus de données définie la capacité de traitement du microprocesseur ; selon le microprocesseur la largeur du bus peut être de 8 bits, 16 bits, 32 bits, 64 bits.

- **Bus d'adresses (uni-directionnel):** permet de transférer entre composants des adresses, ex. : *adresse d'une case mémoire, etc.*

L'espace adressable peut avoir 2n emplacements, avec n est le nombre de lignes du bus d'adresses.

- **Bus de contrôle (bi-directionnel):** permet l'échange entre les composants

d'informations de contrôle [bus rarement représenté sur les schémas].
ex. : *périphérique prêt/occupé, erreur/exécution réussie, etc.*

3.4 Les registres

C'est un espace mémoire interne au processeur. On distingue deux types : à usage général qui permettent à l'UAL de manipuler des données et les registres d'adresses qui sont connectés au bus d'adresse.

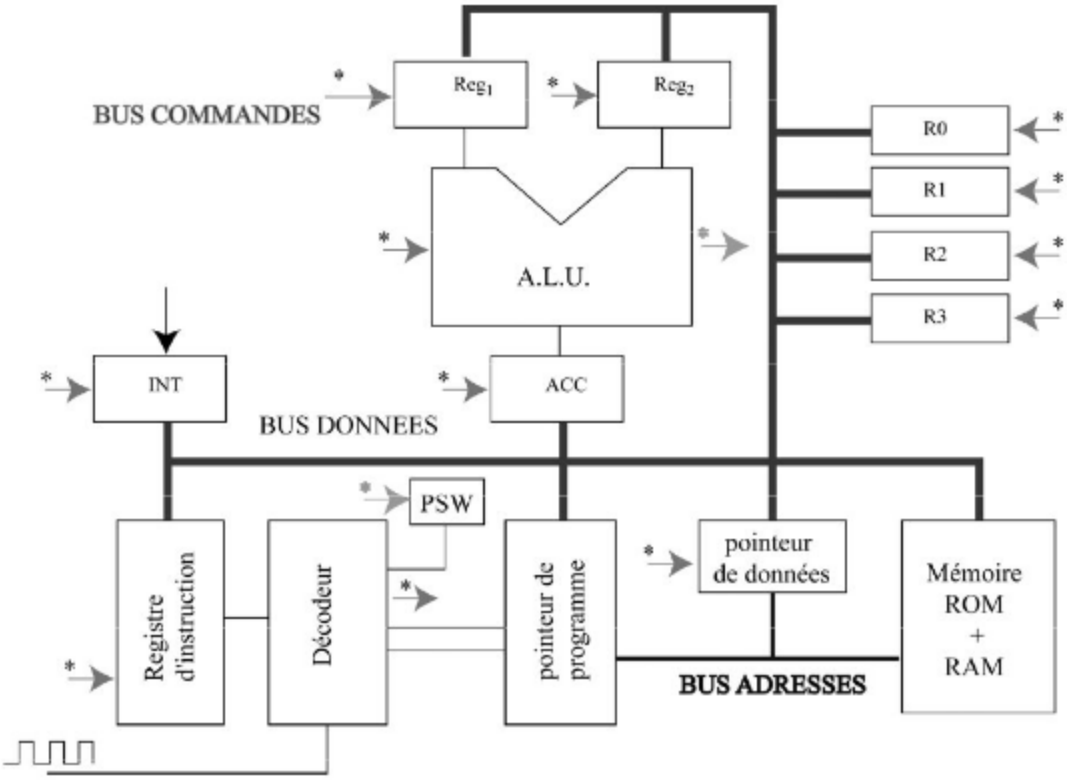


Fig. 1.2 : Structure interne d'un microprocesseur

4. Les mémoires

Le processeur exécute les instructions machines présente dans la mémoire et traite les données qu'elle contient : le fonctionnement du microprocesseur est entièrement conditionné par le contenu de celles-ci.

La mémoire peut être vue comme un ensemble de cellules ou cases contenant chacune une information : une instruction ou une donnée. Chaque case mémoire est repérée par un numéro d'ordre unique : son adresse.

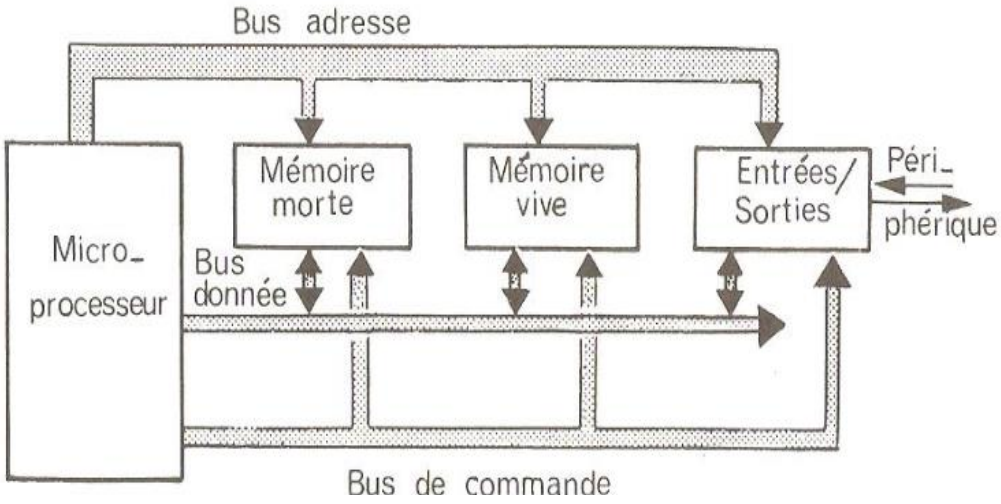
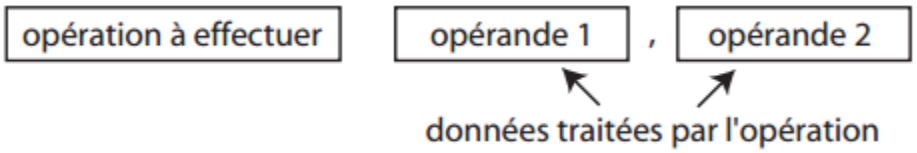


Fig. 1.3 : Structure générale d'un système piloté par un microprocesseur

Une case mémoire peut être lue ou écrite par le microprocesseur (cas des mémoires vives) ou bien seulement lue (cas des mémoires mortes).

Format d'une instruction



Les opérandes sont stockés dans des mémoires RAM.

Un jeu d'instruction est l'ensemble d'opérations élémentaires effectué par le microprocesseur

5. Fonctionnement d'un microprocesseur

Le traitement d'une instruction peut être décomposé en plusieurs phases. Celles-ci sont au nombre de trois :

- Recherche de l'instruction
- Décodage (decode)
- Exécution (execute)

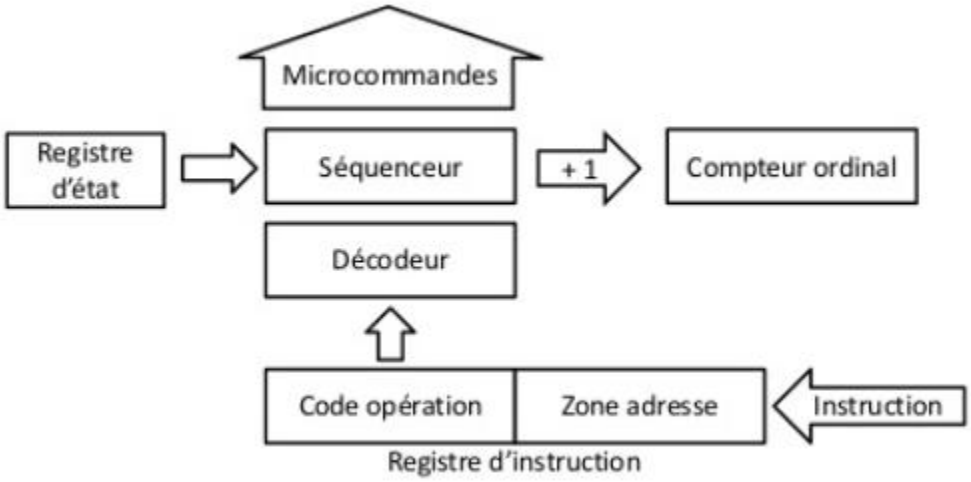


Fig. 1.4 : Fonctionnement d'un microprocesseur

5.1 Recherche de l'instruction

Pour exécuter les instructions dans l'ordre établi par le programme, le microprocesseur doit savoir à chaque instant l'adresse de la prochaine instruction à exécuter. Le microprocesseur utilise un registre contenant cette information : Le pointeur d'instruction

Le contenu de PC est placé sur le bus des adresses. L'unité de contrôle (UC) émet un ordre de lecture, au bout d'un certain temps (temps d'accès à la mémoire) le contenu de la case mémoire sélectionné est disponible sur le bus des données. L'unité de contrôle charge la donnée dans le registre d'instruction pour décodage.

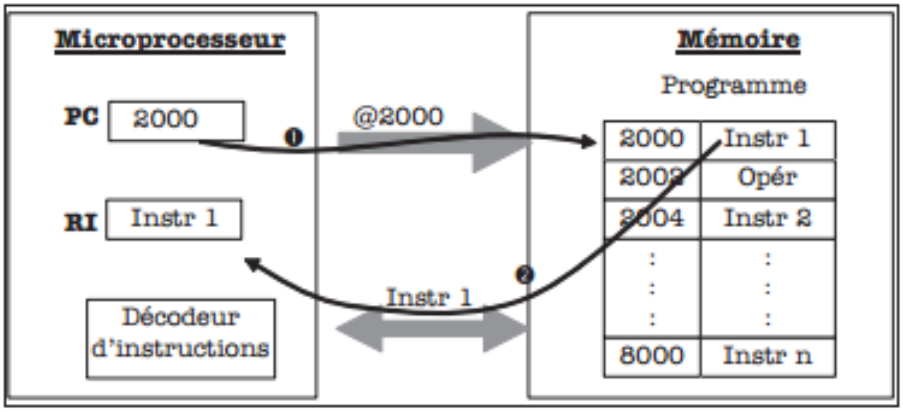


Fig. 1.5 : Recherche d'instruction

5.2 Décodage

Pour savoir quel type d'opération doit être exécuté (addition, soustraction, ...), le microprocesseur lit le premier octet de l'instruction pointée par le pointeur d'instruction (code opératoire) et le range dans le registre d'instruction. Le code opératoire est décodé par des circuits de décodage contenus dans le microprocesseur. Des signaux de commande pour l'UAL sont produits en fonction de l'opération demandée qui est alors exécutée.

Pendant que l'instruction est décodée, le pointeur d'instruction est incrémenté de façon à pointer vers l'instruction suivante. Puis, le processus de lecture et de décodage des instructions recommence.

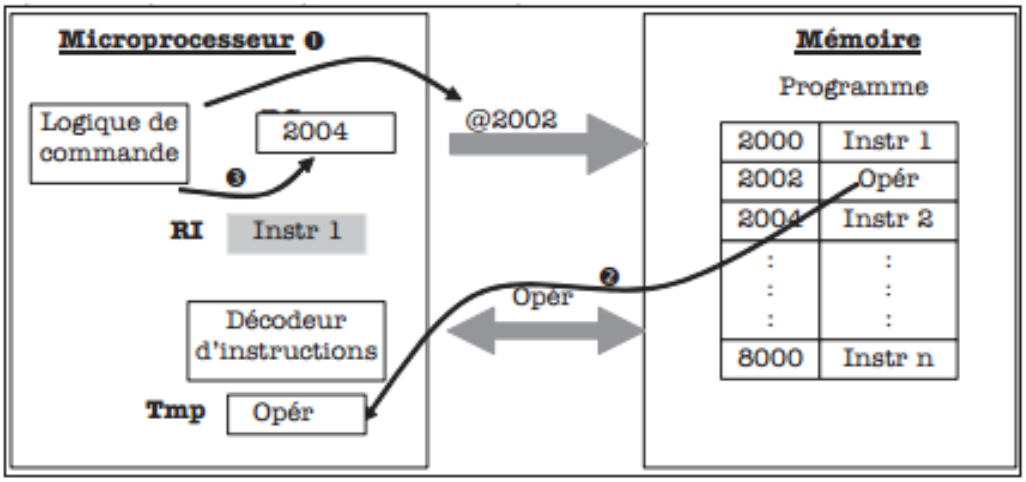


Fig. 1.6 : Récupération de l'opérande

5.3 Exécution

A la suite de chaque instruction, le registre d'état du microprocesseur est actualisé en fonction du dernier résultat.

6. Architecture d'un microprocesseur

Pour l'organisation des différentes unités, il existe deux architectures possibles:

6.1 Architecture de Von Neuman

La mémoire programme, la mémoire données et les périphériques d'entrées/sorties partagent le même bus d'adresses et de données.

Inconvénient: L'exécution d'une instruction nécessite plusieurs échanges de données sur le seul et unique bus dévolu à cet usage puisqu'il faut tout d'abord aller chercher le code de l'instruction puis le ou les données qu'elle doit manipuler.

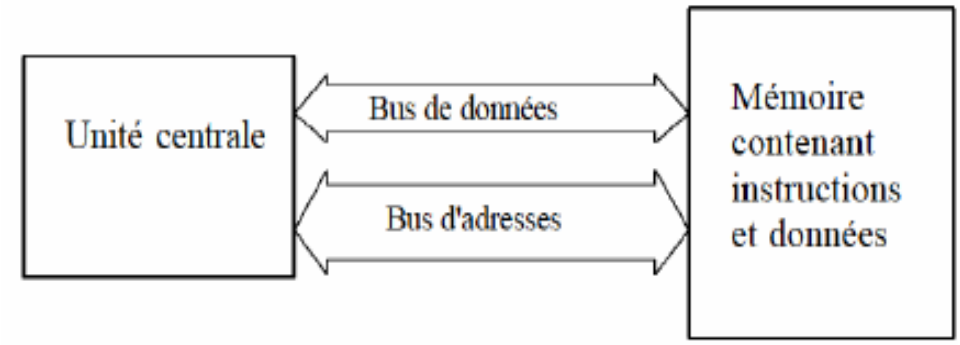


Fig. 1.7 : Architecture de Von Neuman

6.2 Architecture de Harvard

Cette architecture sépare systématiquement la mémoire de programme de la mémoire des données : l'adressage de ces mémoires est indépendant. Ce type d'architecture est utilisé sur des microcontrôleurs qui ont connu un développement important ces dernières années.

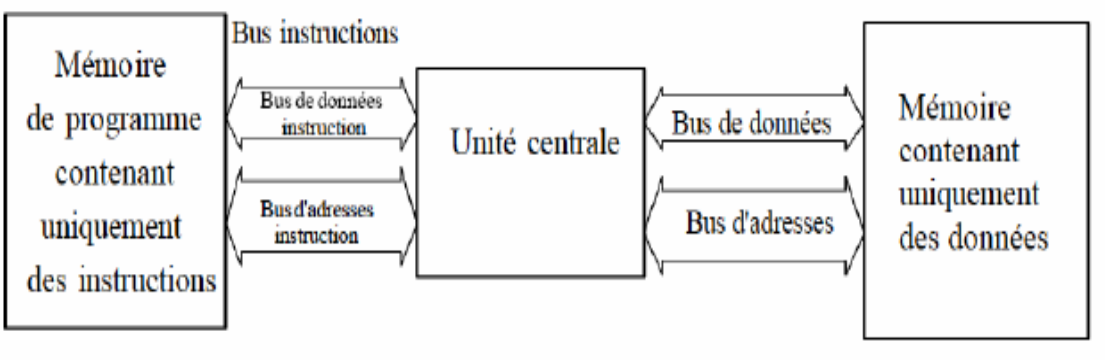


Fig. 1.8 : Architecture de Harvard

Quoique cette architecture puisse être complexe mais elle est performante: Gain en terme de vitesse d'exécution des programmes :

L'exécution d'une instruction ne fait plus appel qu'à un seul cycle machine puisque l'on peut simultanément, grâce au deux bus, rechercher le code de l'instruction et la ou les données qu'elle manipule

Travaux dirigés N°1

Exercice 1 :

On considère le système de la figure 1.

1. Que représente ce schéma ?
2. Expliquer le rôle de chaque partie.
3. Que peut-on conclure quant à la taille du bus de données ?
4. Que peut-on conclure quant à la taille du bus d'adresses ?

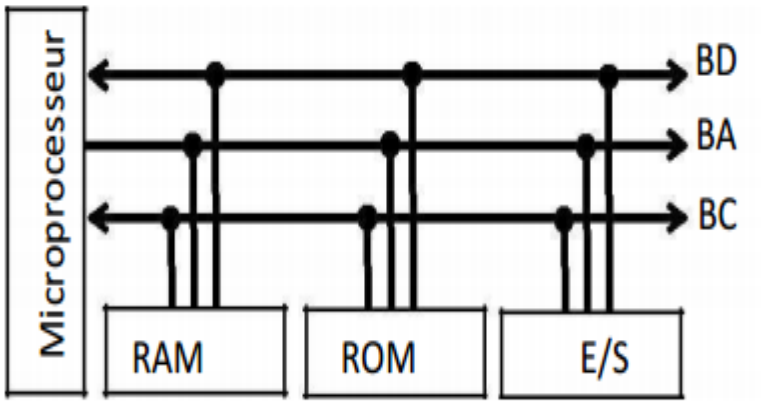


Figure 1.

Exercice 2 :

On considère le schéma synoptique ci-dessous :

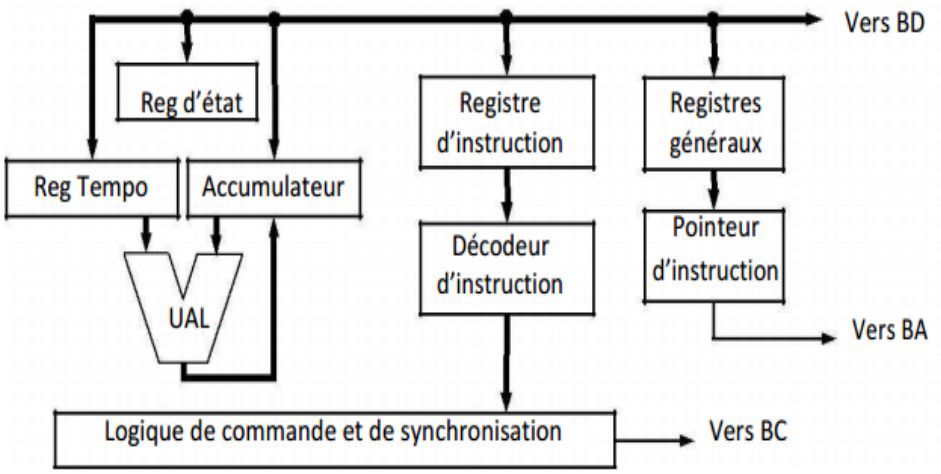


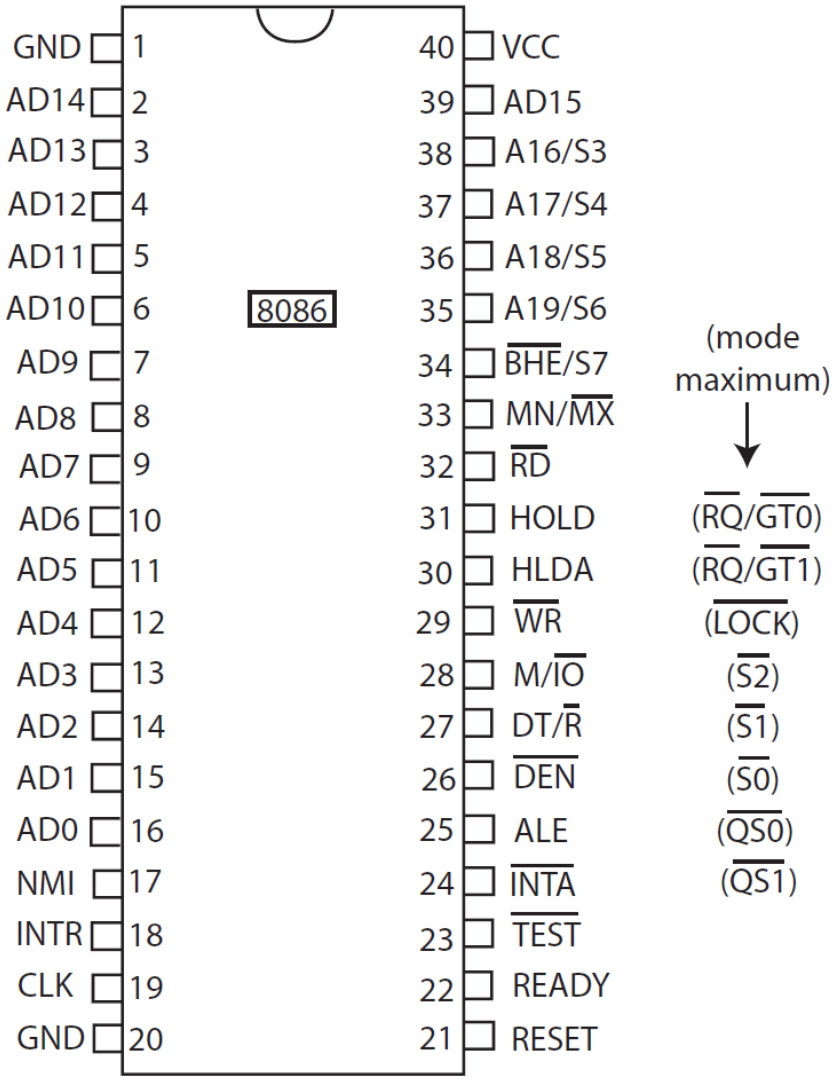
Figure 2.

1. Quel est le rôle de chaque unité ?
2. Donner les différentes étapes par lesquelles passe l'exécution d'une instruction.

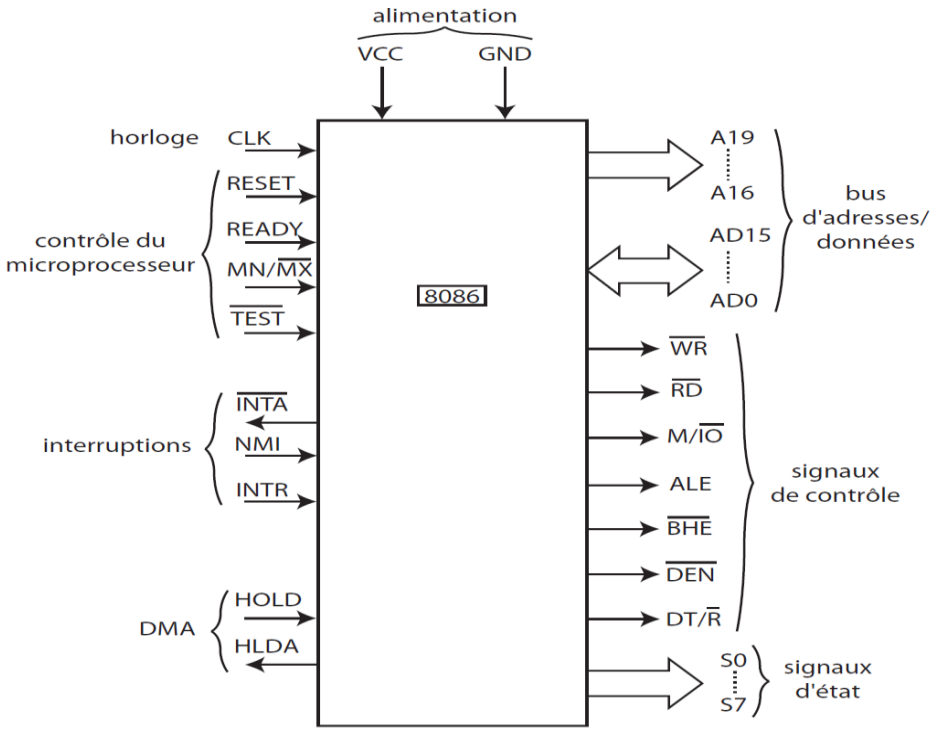
CHAPITRE 2 : Etude du microprocesseur 8086

2.1 Description physique du 8086

Le microprocesseur Intel 8086 est un microprocesseur 16 bits, apparu en 1978. C'est le premier microprocesseur de la famille Intel 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...). Il se présente sous la forme d'un boîtier DIP (Dual In-line Package) `a 40 broches :

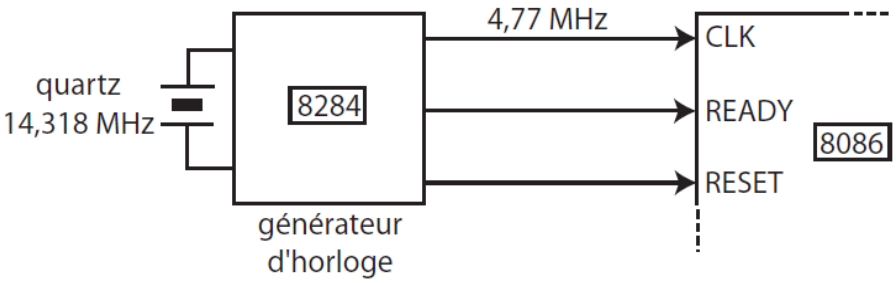


2.2 Schéma fonctionnel du 8086



2.3 Description et utilisation des signaux du 8086

CLK : entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur. Ce signal provient d'un **générateur d'horloge** : le 8284.



RESET : entrée de remise à zéro du microprocesseur. Lorsque cette entrée est mise `a l'état haut pendant au moins 4 périodes d'horloge, le microprocesseur est réinitialisé : il va exécuter

l'instruction se trouvant `a l'adresse FFFF0H (adresse de bootstrap). Le signal de RESET est fourni par le générateur d'horloge.

READY : entrée de synchronisation avec la mémoire. Ce signal provient également du générateur d'horloge.

TEST : entrée de mise en attente du microprocesseur d'un événement extérieur.

MN/MX : entrée de choix du mode de fonctionnement du microprocesseur :

• mode minimum (MN/MX = 1) : le 8086 fonctionne de manière autonome, il génère lui-même le bus de commande (RD, WR, ...) ;

• mode maximum (MN/MX = 0) : ces signaux de commande sont produits par un **contrôleur de bus**, le 8288. Ce mode permet de réaliser des systèmes multiprocesseurs.

NMI et **INTR** : entrées de demande d'interruption. INTR : interruption normale, NMI (Non Maskable Interrupt) : interruption prioritaire.

INTA : Interrupt Acknowledge, indique que le microprocesseur accepte l'interruption.

HOLD et **HLDA** : signaux de demande d'accord d'accès direct `a la mémoire (DMA).

S0 `a **S7** : signaux d'´etat indiquant le type d'opération en cours sur le bus.

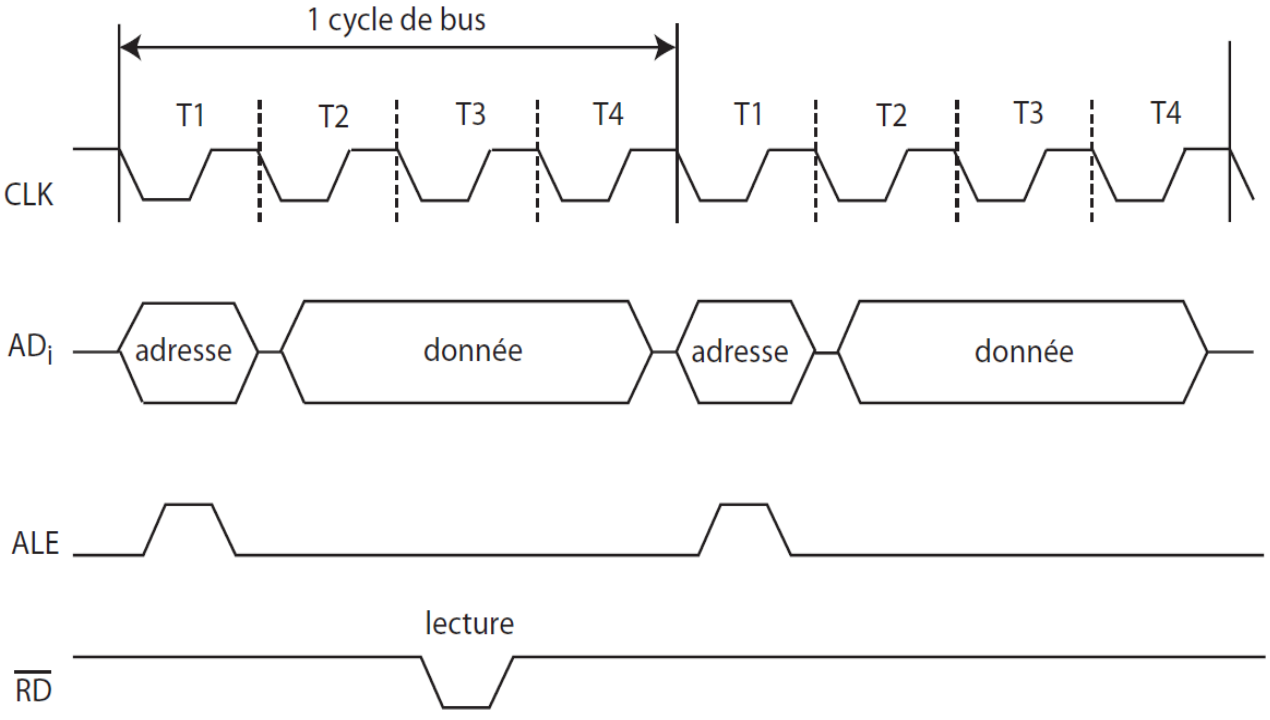
A16/S3 `a **A19/S6** : 4 bits de poids fort du bus d'adresses, **multiplex´es** avec 4 bits d'´etat.

AD0 `a **AD15** : 16 bits de poids faible du bus d'adresses, **multiplex´es** avec 16 bits de données. Le bus A/D est multiplex´e (multiplexage temporel) d'o`u la nécessité d'un **démultiplexage** pour obtenir séparément les bus d'adresses et de données :

• 16 bits de données (microprocesseur 16 bits) ;

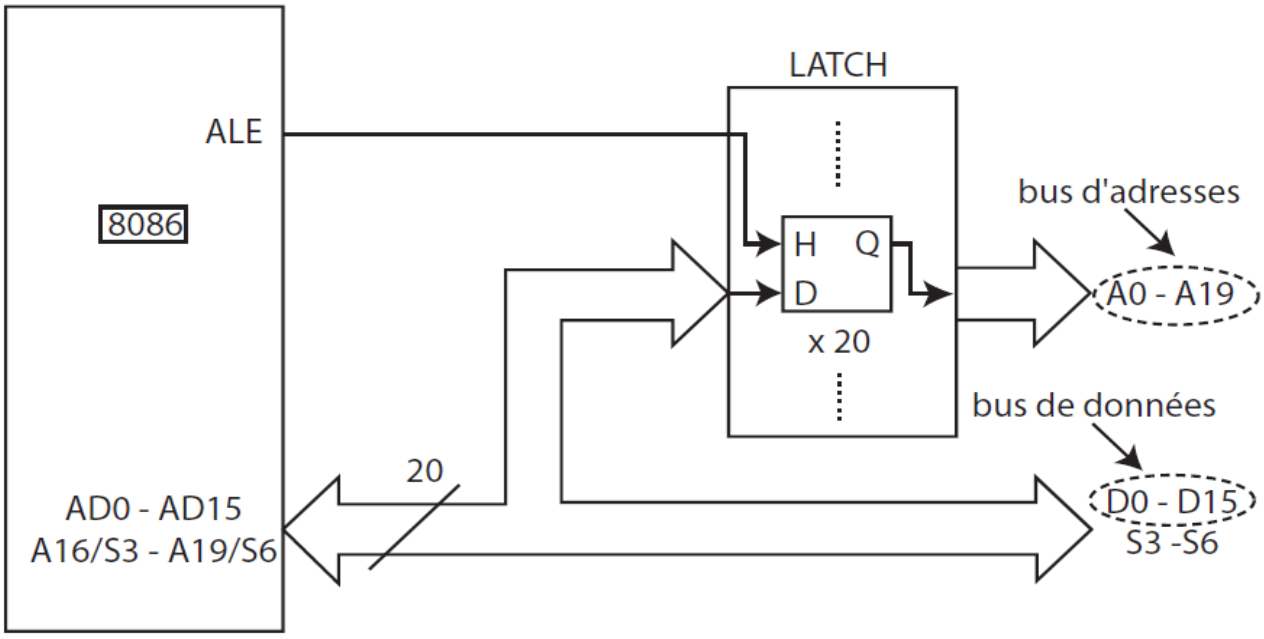
• 20 bits d'adresses, d'o`u 220 = 1 Mo d'espace mémoire adressable par le 8086.

Chronogramme du bus A/D :



Le démultiplexage des signaux AD0 `a AD15 (ou A16/S3 `a A19/S6) se fait en mémorisant l'adresse lorsque celle-ci est présente sur le bus A/D, `a l'aide d'un **verrou** (latch), ensemble de bascules D. La commande de mémorisation de l'adresse est générée par le microprocesseur : c'est le signal **ALE**, Address Latch Enable.

Circuit de démultiplexage A/D :



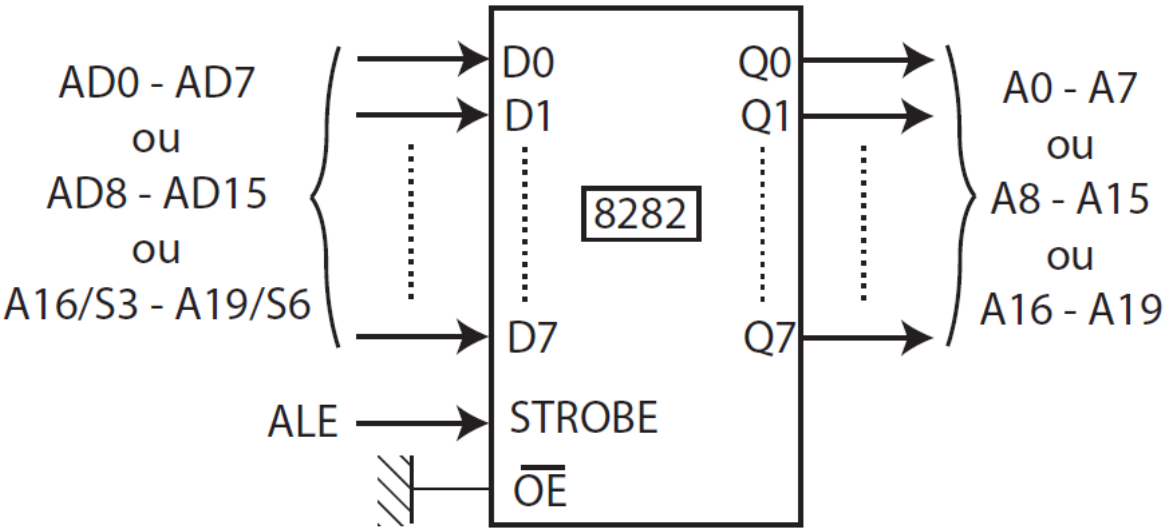
Fonctionnement :

• si ALE = 1, le verrou est transparent (Q = D) ;

• si ALE = 0, mémorisation de la dernière valeur de D sur les sorties Q;

• les signaux de lecture (RD) ou d'´ecriture (WR) ne sont g´en´er´es par le microprocesseur que lorsque les données sont présentes sur le bus A/D.

Exemples de bascules D : circuits 8282, 74373, 74573.



RD : Read, signal de lecture d'une donnée.

WR : Write, signal d'écriture d'une donnée.

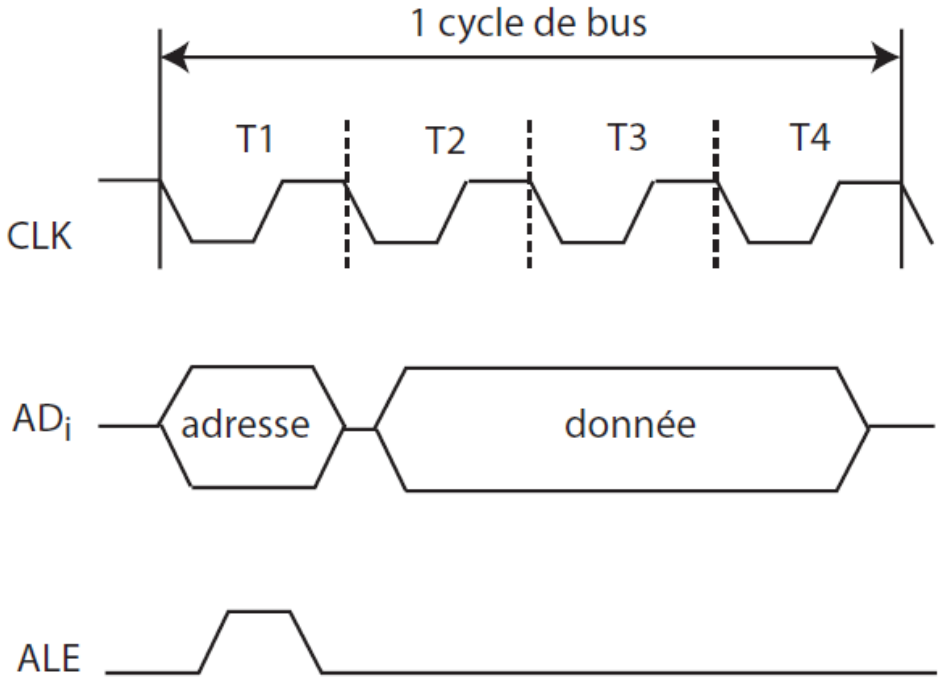
M/IO : Memory/Input-Output, indique si le 8086 adresse la mémoire (M/IO = 1) ou les entrées/sorties (M/IO = 0).

DEN : Data Enable, indique que des données sont en train de circuler sur le bus A/D (´équivalent de ALE pour les données).

DT/R : Data Transmit/Receive, indique le sens de transfert des données :

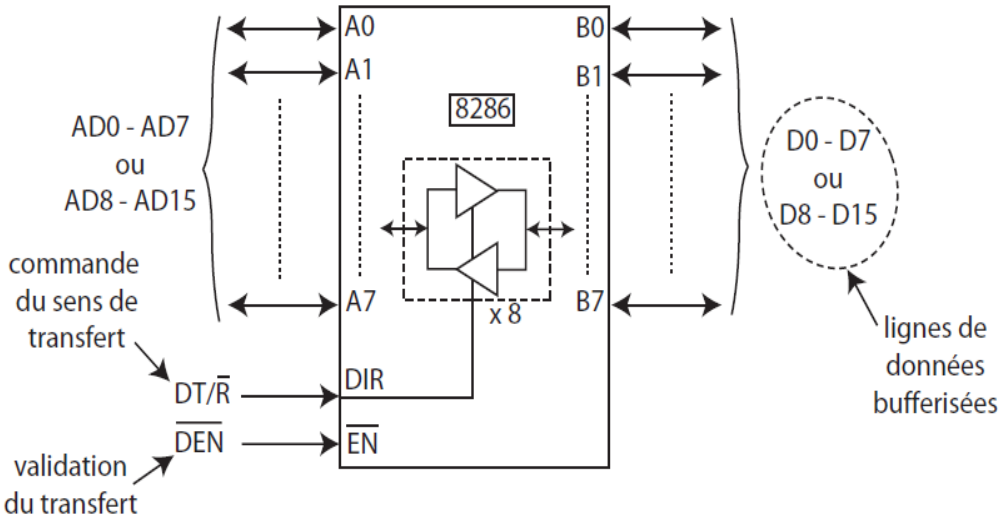
• DT/R = 1 : données émises par le microprocesseur (écriture) ;

• DT/R = 0 : données reçues par le microprocesseur (lecture).



Les signaux DEN et DT/R sont utilisés pour la commande de **tampons de bus** (buffers) permettant d'amplifier le courant fourni par le microprocesseur sur le bus de données.

Exemples de tampons de bus : circuits transmetteurs bidirectionnels 8286 ou 74245.



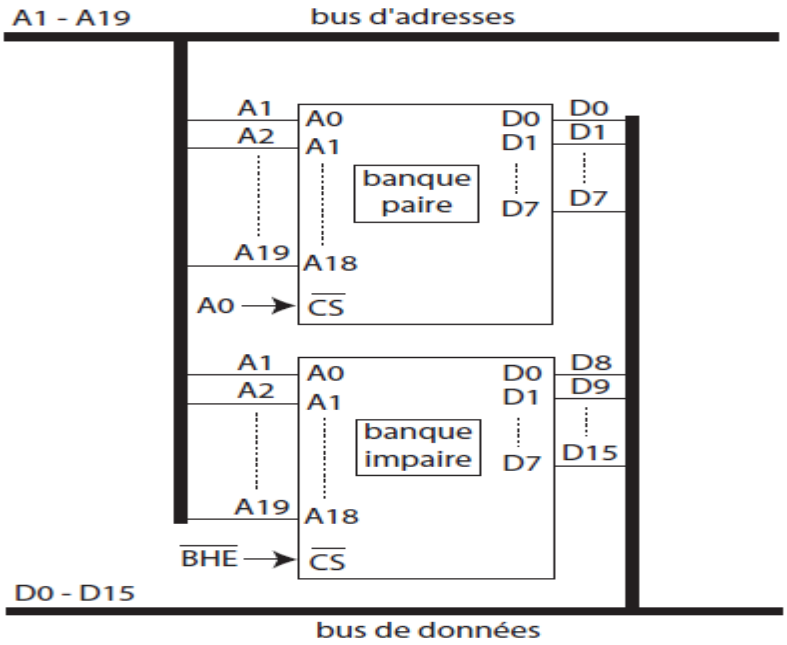
BHE : Bus High Enable, signal de lecture de l'octet de poids fort du bus de données.

Le 8086 possède un bus d'adresses sur 20 bits, d'où la capacité d'adressage de 1 Mo ou 512 Kmots de 16 bits (bus de données sur 16 bits).

Le méga-octet adressable est divis´e en deux **banques** de 512 Ko chacune : la banque **inférieure** (ou **paire**) et la banque **supérieure** (ou **impaire**). Ces deux banques sont s´sélectionnées par :

• A0 pour la banque paire qui contient les octets de poids faible ;

• BHE pour la banque impaire qui contient les octets de poids fort

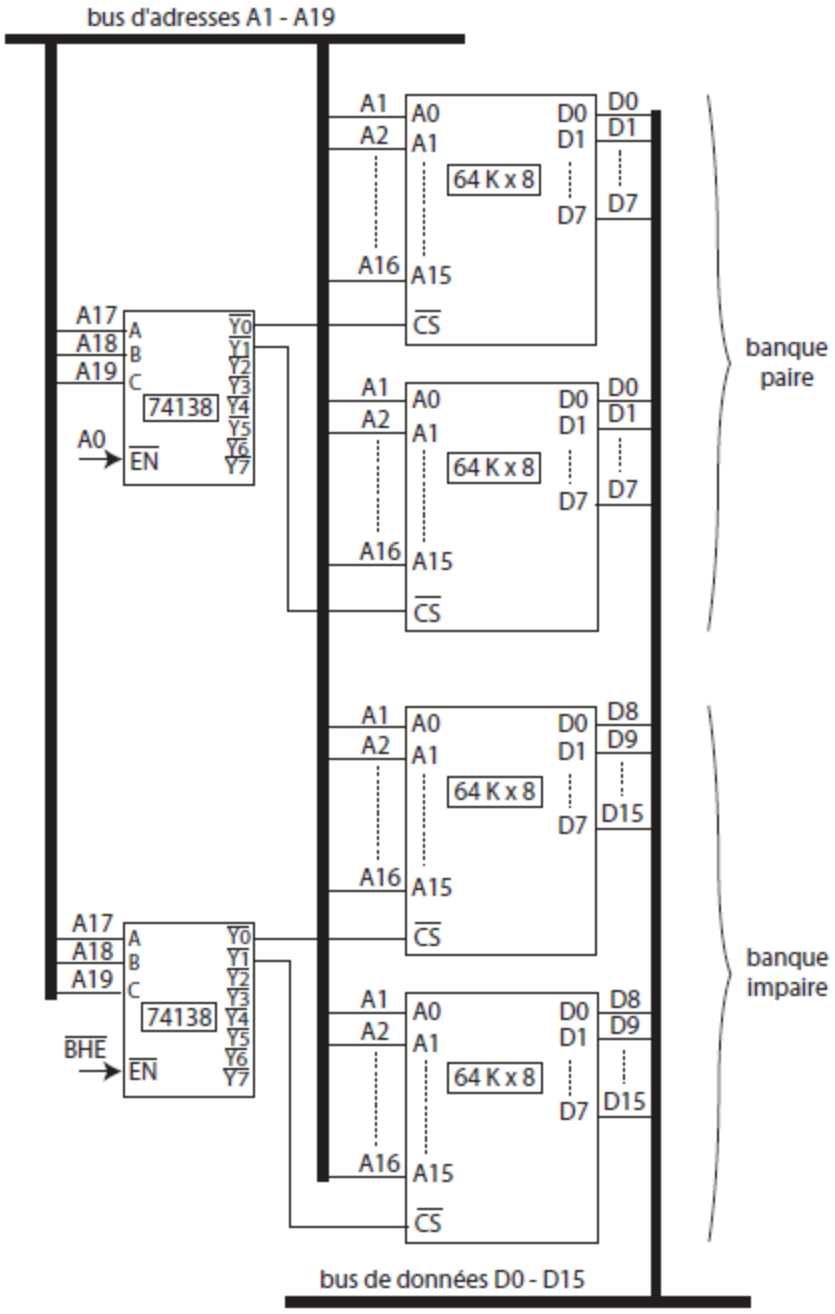


Seuls les bits A1 `a A19 servent à désigner une case mémoire dans chaque banque de 512 Ko.
Le microprocesseur peut ainsi lire et écrire des données sur 8 bits ou sur 16 bits :

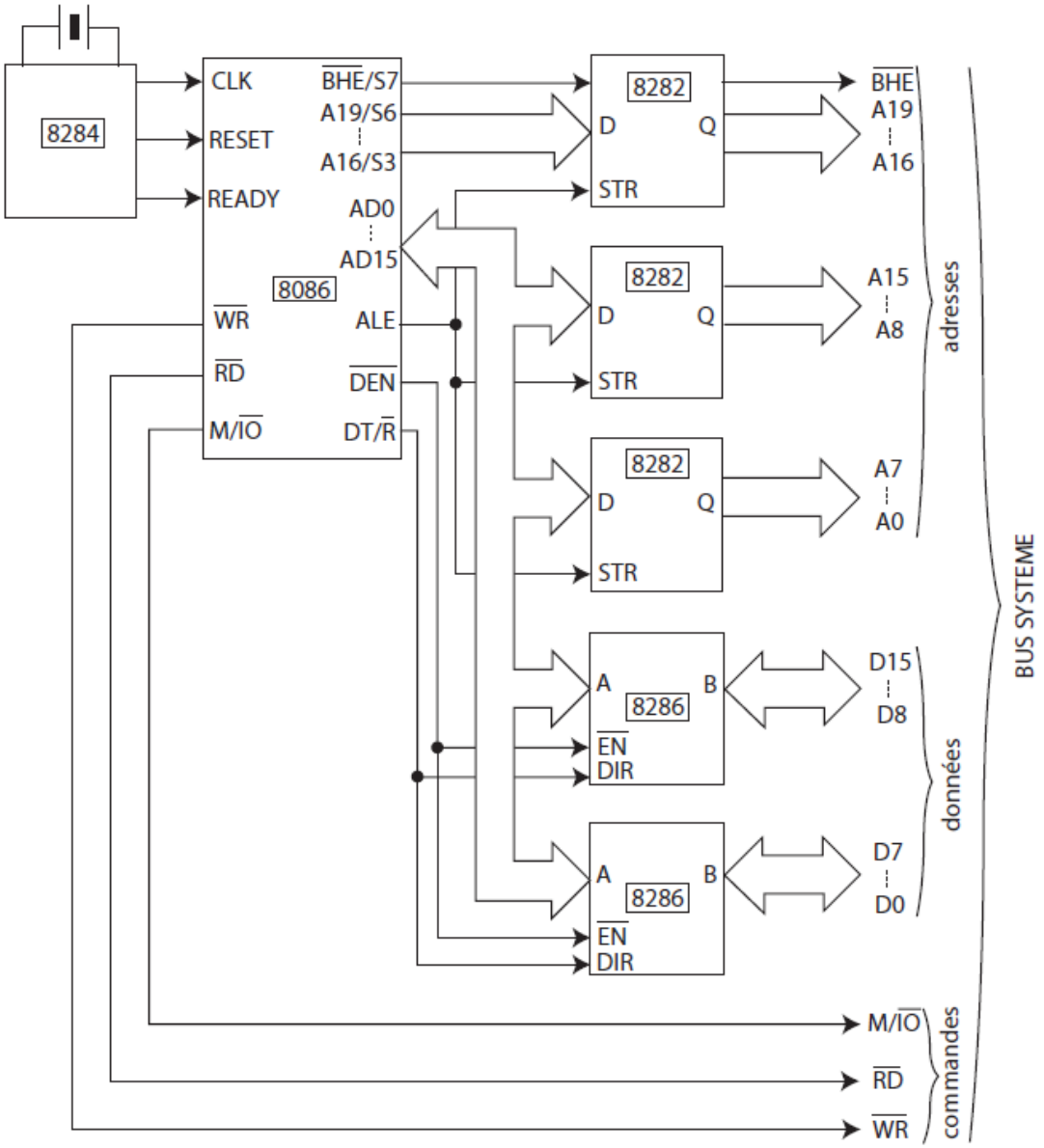
$\overline{\text{BHE}}$	A0	octets transférés
0	0	les deux octets (mot complet)
0	1	octet fort (adresse impaire)
1	0	octet faible (adresse paire)
1	1	aucun octet

Remarque : le 8086 ne peut lire une donnée sur 16 bits en une seule fois, uniquement si l'octet de poids fort de cette donnée est rang´e `a une adresse impaire et l'octet de poids faible `a une adresse paire (alignement sur les adresses paires), sinon la lecture de cette donnée doit se faire en deux opérations successives, d'où une augmentation du temps d'exécution du transfert dû à un mauvais alignement des données.

Réalisation des deux banques avec plusieurs boitiers mémoire :



Création du bus système du 8086 :

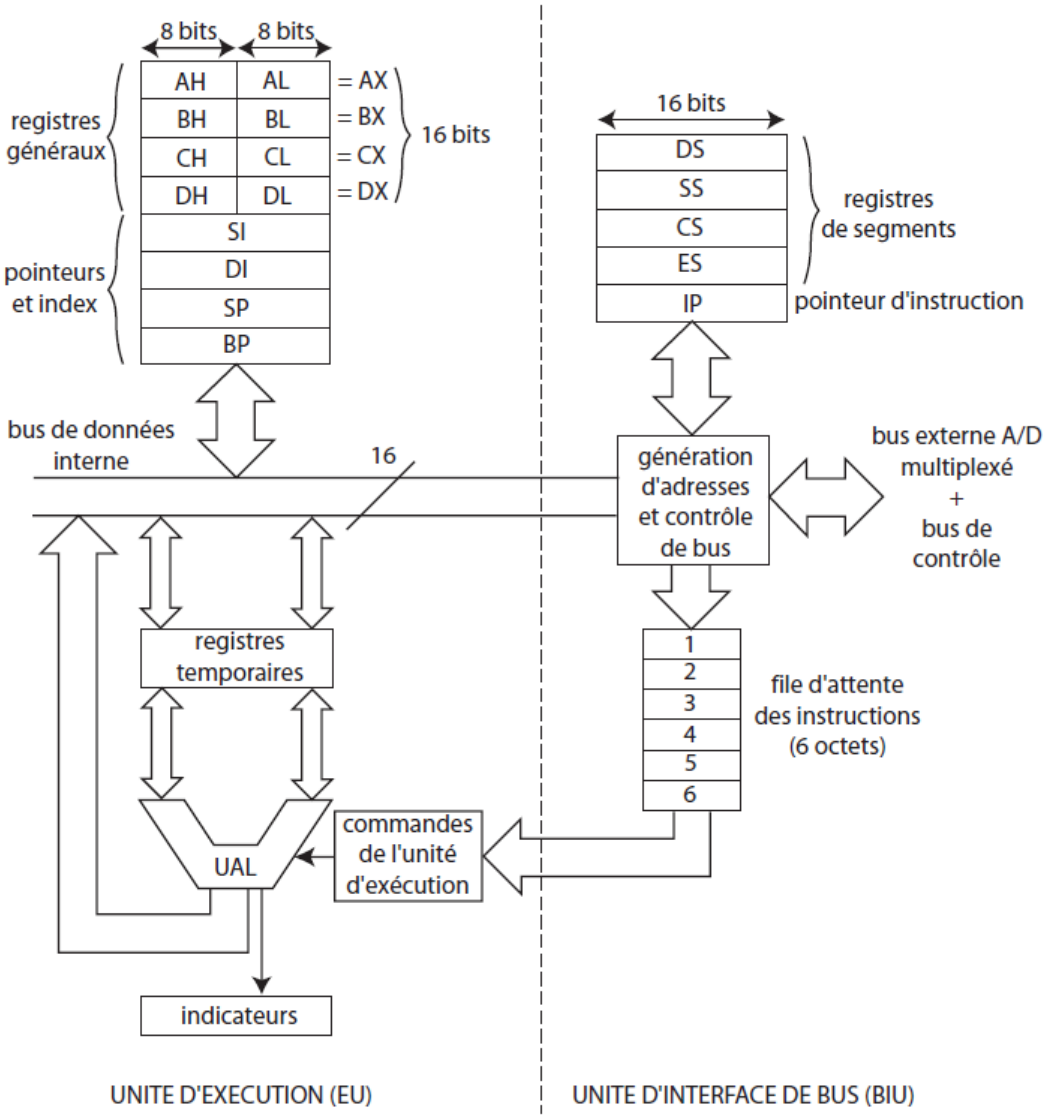


2.4 Organisation interne du 8086

Le 8086 est constitué de deux unit´es fonctionnant en parallèle :

• l'**unit´e d'exécution** (EU : Execution Unit) ;

• l'**unit´e d'interface de bus** (BIU : Bus Interface Unit).



Rôle des deux unités :

• l'unité d'interface de bus (BIU) recherche les instructions en mémoire et les range dans une **file d'attente** ;

• l'unité d'exécution (EU) exécute les instructions contenues dans la file d'attente.

Les deux unités fonctionnent simultanément, d'o`u une accélération du processus d'exécution d'un programme (fonctionnement selon le principe du **pipe-line**).

Le microprocesseur 8086 contient 14 registres répartis en 4 groupes :

• **Registres généraux** : 4 registres sur 16 bits.

AX = (**AH**,**AL**) ;

BX = (**BH**,**BL**) ;

CX = (**CH**,**CL**) ;

DX = (**DH**,**DL**).

Ils peuvent être également considérés comme 8 registres sur 8 bits. Ils servent `a contenir temporairement des données. Ce sont des registres généraux mais ils peuvent être utilisés pour des opérations particulières. Exemple : AX = accumulateur, CX = compteur.

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

Pointeurs :

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

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

Index :

SI : Source Index ;

DI : Destination Index.

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

Les pointeurs et les index contiennent des adresses de cases mémoire.

• **Pointeur d'instruction et indicateurs (flags)** : 2 registres sur 16 bits.

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

Flags :

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

CF : indicateur de retenue (carry) ;

PF : indicateur de parité ;

AF : indicateur de retenue auxiliaire ;

ZF : indicateur de zéro ;

SF : indicateur de signe ;

TF : indicateur d'exécution pas `a pas (trap) ;

IF : indicateur d'autorisation d'interruption ;

DF : indicateur de décrémentation ;

OF : indicateur de dépassement (overflow).

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

CS : Code Segment, registre de segment de code ;

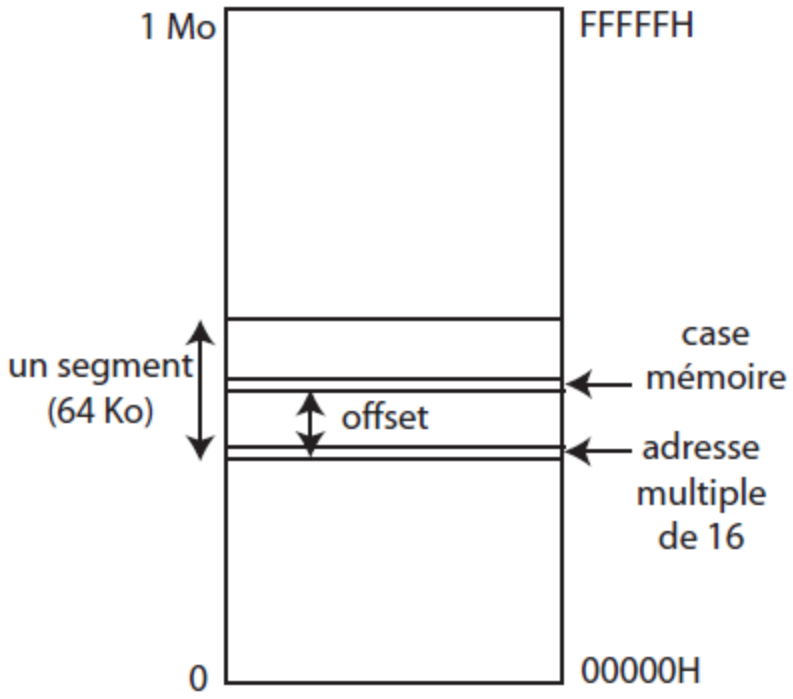
DS : Data Segment, registre de segment de données ;

SS : Stack Segment, registre de segment de pile ;

ES : Extra Segment, registre de segment supplémentaire pour les données ;
Les registres de segments, associés aux pointeurs et aux index, permettent au microprocesseur 8086 d'adresser l'ensemble de la mémoire.

2.5 Gestion de la mémoire par le 8086

L'espace mémoire adressable par le 8086 est de 2^{20} = 1 048 576 octets = 1 Mo (20 bits d'adresses). Cet espace est divis´e en **segments**. Un segment est une zone mémoire de 64 Ko (65 536 octets) définie par son adresse de départ qui doit être un multiple de 16.
Dans une telle adresse, les 4 bits de poids faible sont à zéro. On peut donc représenter l'adresse d'un segment avec seulement ses 16 bits de poids fort, les 4 bits de poids faible étant implicitement `a 0.
Pour désigner une case mémoire parmi les 216 = 65 536 contenues dans un segment, il suffit d'une valeur sur 16 bits.
Ainsi, une case mémoire est repérée par le 8086 au moyen de deux quantités sur 16 bits :
• l'adresse d'un segment ;
• un déplacement ou **offset** (appel´e aussi **adresse effective**) dans ce segment.
Cette méthode de gestion de la mémoire est appelée **segmentation de la mémoire**.

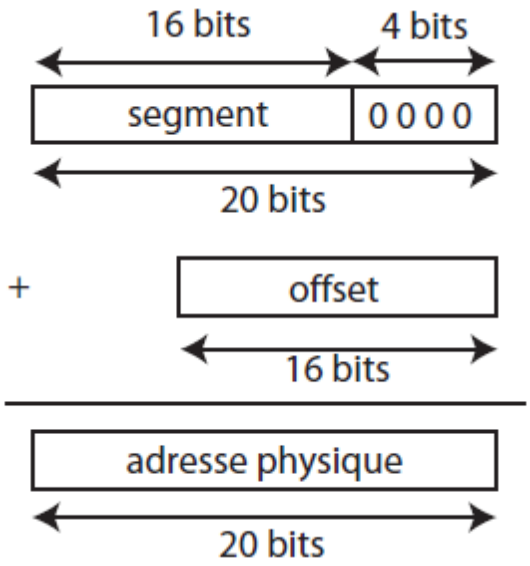


La donnée d'un couple (segment, offset) définit une **adresse logique**, notée sous la forme

segment : offset.

L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexa) est appelée **adresse physique** car elle correspond `a la valeur envoyée réellement sur le bus d'adresses A0 - A19.

Correspondance entre adresse logique et adresse physique :



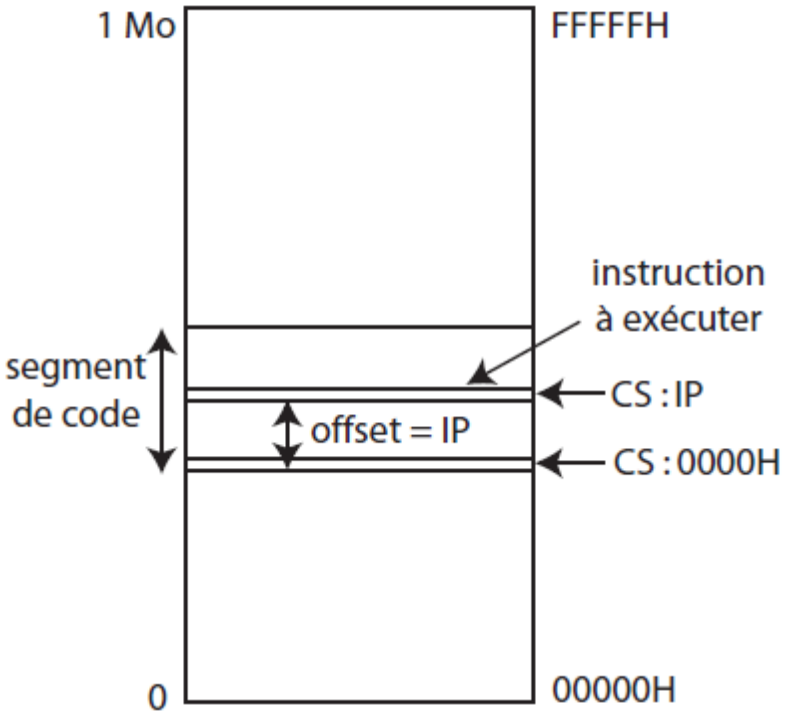
Ainsi, l'adresse physique se calcule par l'expression :

adresse physique = **16** × **segment** + **offset**

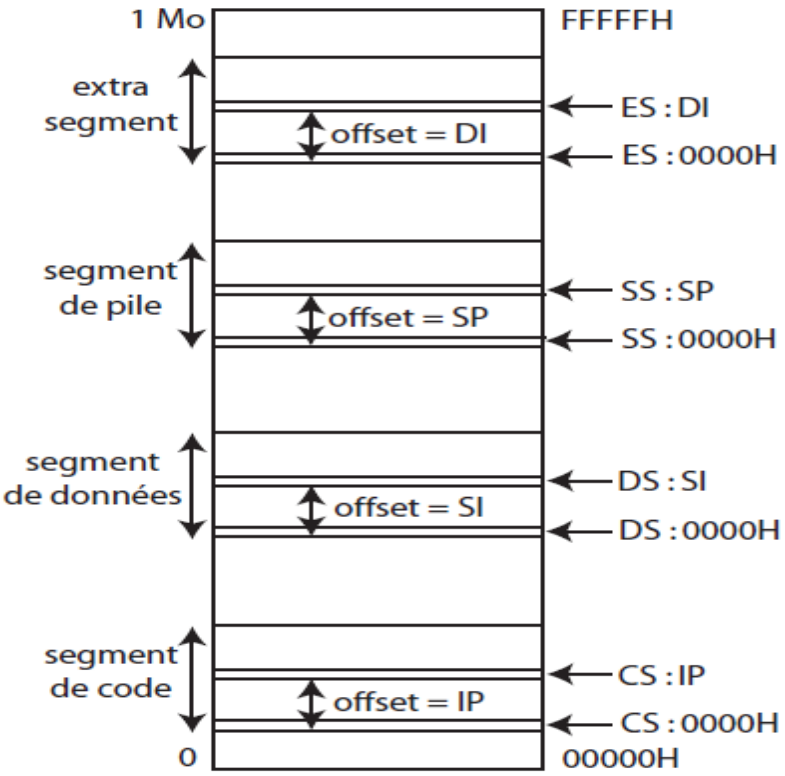
car le fait d'injecter 4 zéros en poids faible du segment revient `a effectuer un décalage de 4 positions vers la gauche, c'est `a dire une multiplication par 24 = 16.

A un instant donné, le 8086 a accès `a 4 segments dont les adresses se trouvent dans les registres de segment CS, DS, SS et ES. Le segment de code contient les instructions du programme, le segment de données contient les données manipulées par le programme, le segment de pile contient la pile de sauvegarde et le segment supplémentaire peut aussi contenir des données.

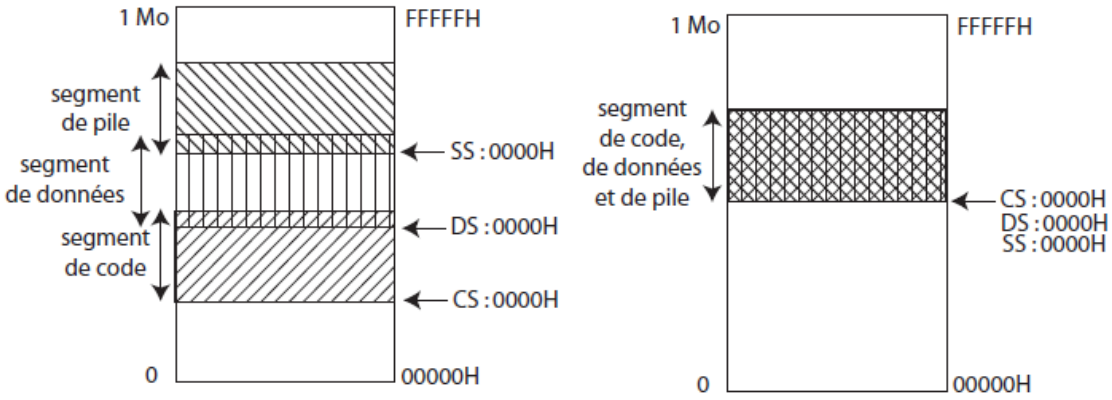
Le registre CS est associé au pointeur d'instruction IP, ainsi la prochaine instruction à éxécuter se trouve `a l'adresse logique CS : IP.



De même, les registres de segments DS et ES peuvent être associés `a un registre d'index. Exemple : DS : SI, ES : DI. Le registre de segment de pile peut être associé aux registres de pointeurs : SS : SP ou SS : BP.
Mémoire accessible par le 8086 `a un instant donné :



Remarque : les segments ne sont pas nécessairement distincts les uns des autres, ils peuvent se chevaucher ou se recouvrir complètement.



Le nombre de segments utilisé définit le **modèle mémoire** du programme.

Contenu des registres après un RESET du microprocesseur :

IP = 0000H

CS = FFFFH

DS = 0000H

ES = 0000H

SS = 0000H

Puisque CS contient la valeur FFFFH et IP la valeur 0000H, la première instruction exécutée par le 8086 se trouve donc `a l'adresse logique FFFFH : 0000H, correspondant `a l'adresse physique FFFF0H (bootstrap). Cette instruction est généralement un saut vers le programme principal qui initialise ensuite les autres registres de segment.

2.6 Le microprocesseur 8088

Le microprocesseur 8088 est identique au 8086 sauf que son bus de données externe est sur 8 bits au lieu de 16 bits, le bus de données interne restant sur 16 bits.

Le 8088 a ´et´e produit par Intel après le 8086 pour assurer la compatibilité avec des circuits périphériques déjà existant, fabriqués pour les microprocesseurs 8 bits 8080 et 8085.

Différences avec le 8086 :

• les broches AD8 `a AD15 deviennent A8 `a A15 (bus de données sur 8 bits) ;

• la broche BHE n'existe pas dans le 8088 car il n'y a pas d'octet de poids fort sur le bus de données ;

• la broche M/IO devient IO/M pour la compatibilité´e avec d'anciens circuits d'E/S.

Au niveau de l'architecture interne, pas de différences avec le 8086 sauf que la file d'attente des instructions passe de 6 `a 4 octets.

Chapitre 3 : Programmation en assembleur du microprocesseur 8086

3.1 Généralités

Chaque microprocesseur reconnait un ensemble d'instructions appelé **jeu d'instructions** (Instruction Set) fixé par le constructeur. Pour les microprocesseurs classiques, le nombre d'instructions reconnues varie entre 75 et 150 (microprocesseurs **CISC** : Complex Instruction Set Computer). Il existe aussi des microprocesseurs dont le nombre d'instructions est très réduit (microprocesseurs **RISC** : Reduced Instruction Set Computer) : entre 10 et 30 instructions, permettant d'améliorer le temps d'exécution des programmes.

Une instruction est définie par son code opératoire, valeur numérique binaire difficile à manipuler par l'être humain. On utilise donc une **notation symbolique** pour représenter les instructions : les **mnémoniques**. Un programme constitué de mnémoniques est appel´e **programme en assembleur**.

Les instructions peuvent être classées en groupes :

• instructions de transfert de données ;

• instructions arithmétiques ;

• instructions logiques ;

• instructions de branchement ...

3.2 Les instructions de transfert

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

• registre vers mémoire ;

• registre vers registre ;

• mémoire vers registre.

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

Syntaxe : MOV destination, source

Remarque : MOV est l'abréviation du verbe « to move » : déplacer.

Il existe différentes façons de spécifier l'adresse d'une case mémoire dans une instruction : ce sont les **modes d'adressage**.

3.3 Les modes d'adressage du 8086:

Les instructions et leurs opérandes (paramètres) sont stockées en mémoire principale. La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire)

dépend du type d'instruction et aussi du type d'opérande. Chaque instruction est toujours codée sur un nombre entier d'octets, afin de faciliter son décodage par le processeur.

Une instruction est composée de deux champs :

- le code opération, qui indique au processeur quelle instruction réaliser.

- le champ opérande qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

Les façons de désigner les opérandes constituent les "modes d'adressage". Selon la manière dont l'opérande (la donnée) est spécifié, c'est à dire selon le mode d'adressage de la donnée, une instruction sera codée par 1, 2, 3 ou 4 octets.

Le microprocesseur 8086 possède 7 modes d'adressage :

- Mode d'adressage registre.

- Mode d'adressage immédiat.

- Mode d'adressage direct.

- Mode d'adressage registre indirect.

- Mode d'adressage relatif à une base.

- Mode d'adressage direct indexé.

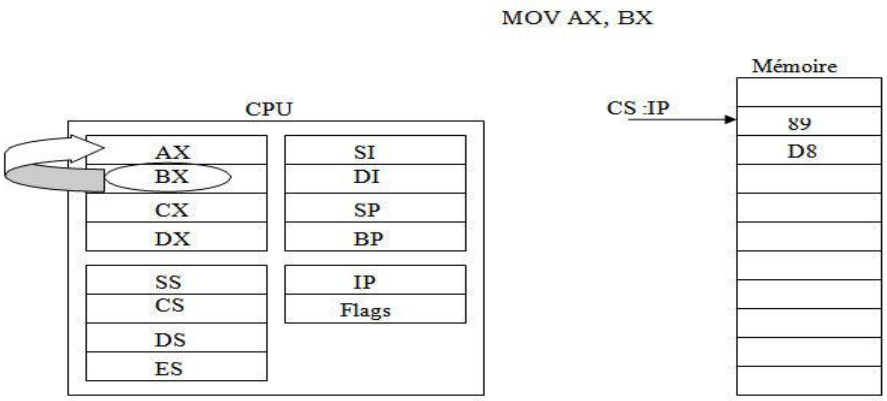
- Mode d'adressage indexé.

3.4 Mode d'adressage registre :

Ce mode d'adressage concerne tout transfert ou toute opération, entre deux registres de même taille. Dans ce mode l'opérande sera stocké dans un registre interne au microprocesseur.

Exemple :

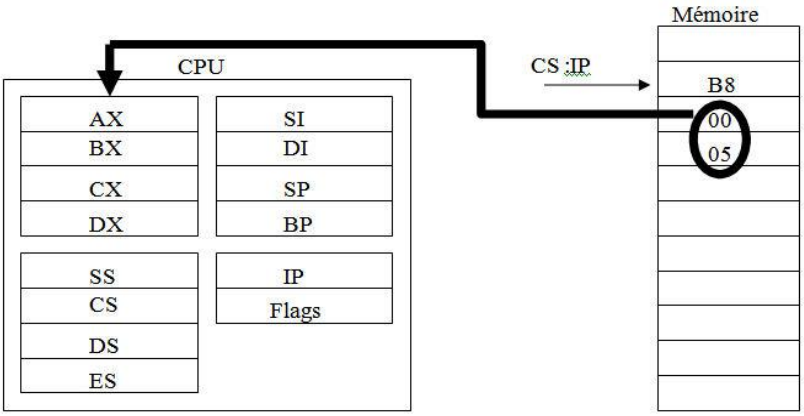
Mov AX, BX ; cela signifie que l'opérande stocker dans le registre BX sera transféré vers le registre AX. Quand on utilise l'adressage registre, le microprocesseur effectue toutes les opérations d'une façon interne. Donc dans ce mode il n'y a pas d'échange avec la mémoire, ce qui augmente la vitesse de traitement de l'opérande.



3.5 Mode d'adressage immédiat :

Dans ce mode d'adressage l'opérande apparaît dans l'instruction elle- même, exemple :

MOV AX, 500H ; cela signifie que la valeur 500H sera stockée immédiatement dans le registre AX



Remarque :

Pour les instructions telles que :

MOV AX,-500H ; le signe - sera propager dans le registre jusqu'à remplissage de ce dernier.

Exemple dans notre cas MOV AX,-500H donne AX =1111110100000000B

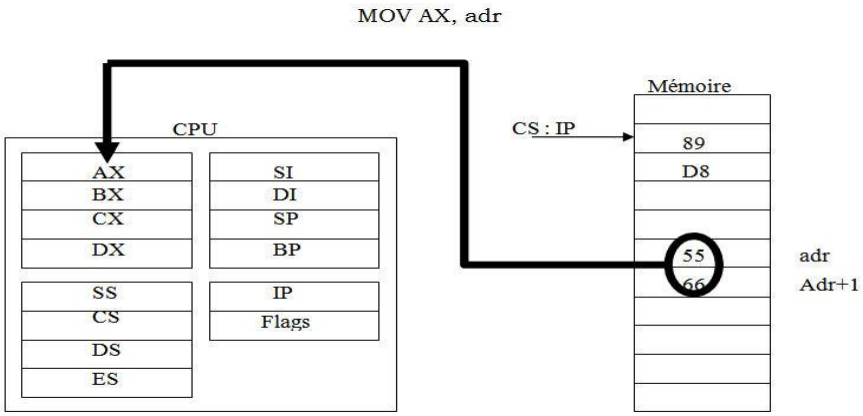
MOV BL,-20H donne BL = 11100000B

3.6 Mode d'adressage direct :

Dans ce mode on spécifie directement l'adresse de l'opérande dans l'instruction .exemple :

MOV AX,adr

La valeur adr est une constante (un déplacement) qui doit être ajouté au contenu du registre DS pour former l'adresse physique de 20 bits.



Remarque :

En général le déplacement est ajouté par défaut avec le registre segment DS pour former l'adresse physique de 20 bits, mais il faut signaler qu'on peut utiliser ce mode d'adressage avec d'autres registres segment tel que ES par exemple, seule la syntaxe en mnémonique de l'instruction change et devient :

MOV AX, ES : adr

3.7 Mode d'adressage registre indirect :

Dans ce mode d'adressage l'adresse de l'opérande est stockée dans un registre qu'il faut bien évidemment le charger au préalable par la bonne adresse. L'adresse de l'opérande sera stockée dans un registre de base (BX ou BP) ou un index (SI ou DI).

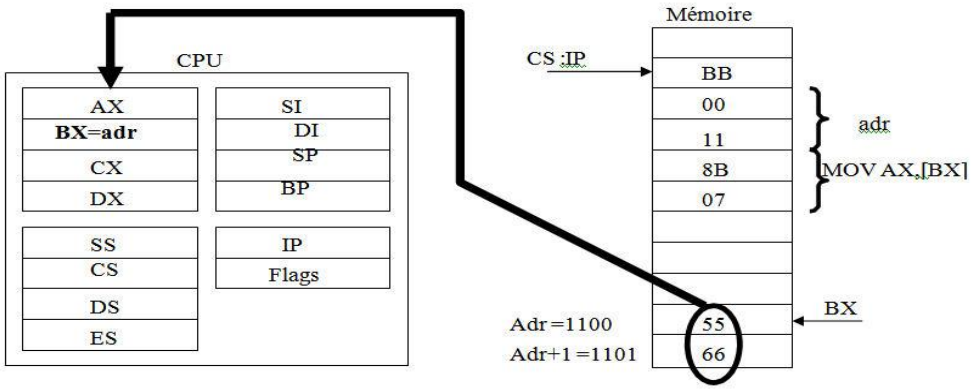
Exemple :

MOV BX, offset adr
MOV AX, [BX]

Le contenu de la case mémoire dont l'adresse se trouve dans le registre BX (c.a.d : Adr) est mis dans le registre AX

Remarque:

Le symbole [ ] design l'adressage indirect.



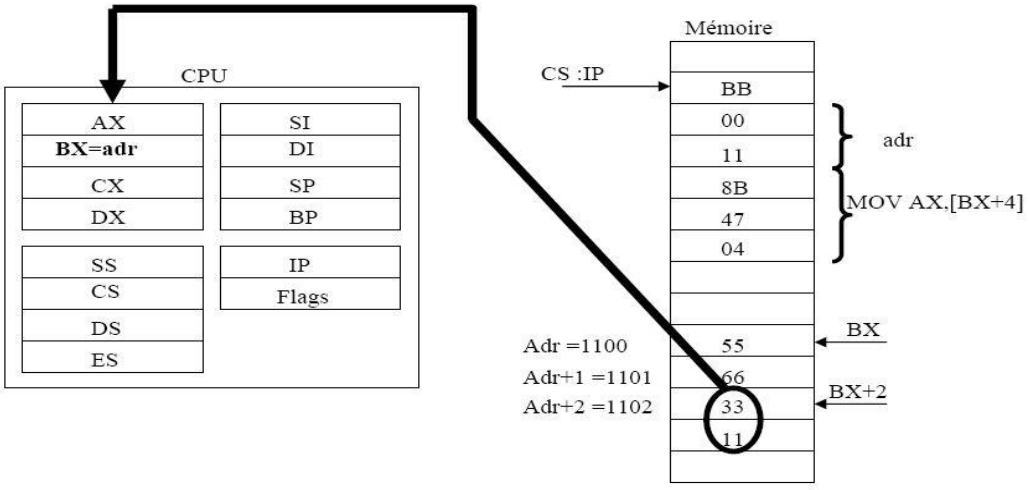
3.8 Mode d'adressage relatif à une base :

Dans ce mode d'adressage Le déplacement est déterminé par soit, le contenu de BX, soit le contenu de BP, auquel est éventuellement ajouté un décalage sur 8 ou 16 bits signé. DS et SS sont pris par défaut.

Exemple :

MOV AX,[BX]+2

Cela signifie que dans le registre AX on va mettre le contenu de la case mémoire pointe par BX+2



Les syntaxes suivantes sont identiques :

MOV AX, [BX+2]

MOV AX, [BX]+2

MOV AX, 2[BX]

Adressage direct indexé : Dans ce mode d'adressage, la source est calculée en ajoutant un offset à un registre <DI> ou <SI>. Ex : MOV [2000H+DI], AL ;

Adressage indexé

Ex : MOV [SI][BX], AX ;

3.9 Les instructions arithmétiques

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

Addition : ADD opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 + opérande2.

Exemples :

• add ah,[1100H] : ajoute le contenu de la case mémoire d'offset 1100H `a l'accumulateur AH (adressage direct) ;

• add ah,[bx] : ajoute le contenu de la case mémoire pointée par BX `a l'accumulateur AH (adressage bas´e) ;

• add byte ptr [1200H],05H : ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H (adressage immédiat).

Soustraction : SUB opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 − opérande2.

Multiplication : MUL opérande, où opérande est un registre ou une case mémoire.

Cette instruction effectue la multiplication du contenu de AL par un opérande sur 1 octet ou du contenu de AX par un opérande sur 2 octets. Le résultat est placé dans AX si les données `a multiplier sont sur 1 octet (résultat sur 16 bits), dans (DX,AX) si elles sont sur 2 octets (résultat sur 32 bits).

Exemples :

• mov al,51

mov bl,32

mul bl

→ AX = 51 × 32

• mov ax,4253

mov bx,1689

mul bx

→ (DX, AX) = 4253 × 1689

• mov al,43

mov byte ptr [1200H],28

mul byte ptr [1200H]

→ AX = 43 × 28

• mov ax,1234

mov word ptr [1200H],5678

mul word ptr [1200H]

→ (DX, AX) = 1234 × 5678

Division : DIV opérande, où opérande est un registre ou une case mémoire.

Cette instruction effectue la division du contenu de AX par un opérande sur 1 octet ou le contenu de (DX,AX) par un opérande sur 2 octets. Résultat : si l'opérande est sur 1 octet, alors AL = quotient et AH = reste ; si l'opérande est sur 2 octets, alors AX = quotient et DX = reste.

Exemples :

• mov ax,35

mov bl,10

div bl

→ AL = 3 (quotient) et AH = 5 (reste)

• mov dx,0

mov ax,1234

mov bx,10

div bx

→ AX = 123 (quotient) et DX = 4 (reste)

Autres instructions arithmétiques :

• ADC : addition avec retenue ;

• SBB : soustraction avec retenue ;

• INC : incrémentation d'une unit´e ;

• DEC : décrémentation d'une unit´e ;

• IMUL : multiplication signée ;

• IDIV : division signée.

3.10 Les instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont :

• ET;

• OU;

• OU exclusif ;

• complément `a 1;

• complément `a 2;

• décalages et rotations.

Les différents modes d'adressage sont disponibles.

ET logique : AND opérande1,opérande2

L'opération effectuée est : opérande1 ← opérande1 ET op´erande2.

Exemple :

```
mov al,10010110B
mov bl,11001101B
and al, bl
```

→

$$\begin{array}{ll} \text{AL} = & 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\ \text{BL} = & 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1 \\ \hline \text{AL} = & 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \end{array}$$

Application : **masquage** de bits pour mettre `a zéro certains bits dans un mot.

Exemple : masquage des bits 0, 1, 6 et 7 dans un octet :

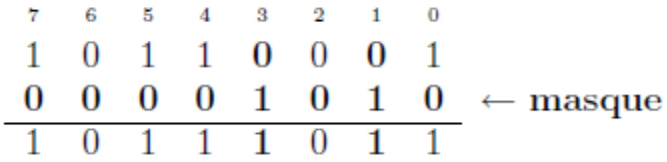
$$\begin{array}{cccccccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline \mathbf{0} & \mathbf{1} & 0 & 1 & 0 & 1 & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & 0 & 1 & 0 & 1 & \mathbf{0} & \mathbf{0} \end{array} \leftarrow \textbf{masque}$$

OU logique : OR opérande1,opérande2

L'opération effectuée est : opérande1 ← opérande1 OU opérande2.

Application : mise à 1 d'un ou plusieurs bits dans un mot.

Exemple : dans le mot 10110001B on veut mettre `a 1 les bits 1 et 3 sans modifier les autres bits.



Les instructions correspondantes peuvent s'´ecrire :

mov ah,10110001B

or ah,00001010B

Complément à 1 : NOT opérande

L'opération effectuée est : opérande ← opérande.

Exemple:

mov al,10010001B

not al

→ AL = 10010001B = 01101110B

Complément `a 2 : NEG opérande

L'opération effectuée est : opérande ← opérande + 1.

Exemple :

mov al,25

mov bl,12

neg bl

add al,bl

→ AL = 25 + (−12) = 13

OU exclusif : XOR opérande1, opérande2

L'opération effectuée est : opérande1 ← opérande1 $\oplus$ opérande2.

Exemple : mise `a zéro d'un registre :

mov al,25

xor al,al

→ AL = 0

Instructions de décalages et de rotations : ces instructions déplacent d'un certain nombre de positions les bits d'un mot vers la gauche ou vers la droite.

Dans 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 rotations, les bits déplacés dans un sens sont réinjectés de l'autre côté du mot.

Décalage logique vers la droite (Shift Right) : SHR opérande

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

Exemple :

mov al,11001011B

shr al,1

→ entrée d'un 0 `a la place du bit de poids fort ; le bit sortant passe `a travers l'indicateur de retenue CF.

Remarque : si le nombre de bits `a décaler est supérieur `a 1, ce nombre doit être placé dans le registre CL ou CX.

Exemple : décalage de AL de trois positions vers la droite :

mov cl,3

shr al,cl

Décalage logique vers la gauche (Shift Left) : SHL opérande,n

Cette instruction décale l'opérande de n positions vers la gauche.

Exemple :

mov al,11001011B

shl al,1

→ entrée d'un 0 `a la place du bit de poids faible ; le bit sortant passe `a travers l'indicateur de retenue CF.

Même remarque que précédemment si le nombre de positions `a décaler est supérieur `a 1.

Décalage arithmétique vers la droite : SAR opérande,n

Ce décalage conserve le bit de signe bien que celui-ci soit décalé.

Exemple :

mov al,11001011B

sar al,1

→ le bit de signe est **réinjecté**.

Décalage arithmétique vers la gauche : SAL opérande,n

Identique au décalage logique vers la gauche.

Applications des instructions de décalage :

• cadrage `a droite d'un groupe de bits.

Exemple : on veut avoir la valeur du quartet de poids fort du registre AL :

mov al,11001011B

mov cl,4

shr al,cl

→ AL = 0000**1100**B

• test de l'´etat d'un bit dans un mot.

Exemple : on veut déterminer l'´etat du bit 5 de AL :

mov cl,6

shr al,cl

ou

mov cl,3

shl al,cl

→ avec un décalage de 6 positions vers la droite ou 4 positions vers la gauche, le bit 5 de AL est transféré dans l'indicateur de retenue CF. Il suffit donc de tester cet indicateur.

• multiplication ou division par une puissance de 2 : un décalage `a droite revient `a faire une division par 2 et un décalage `a gauche, une multiplication par 2.

Exemple :

mov al,48

mov cl,3

shr al,cl

→ AL = 48/23 = 6

Rotation à droite (Rotate Right) : ROR opérande,n

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

Exemple :

mov al,11001011B

ror al,1

→ réinjection du bit sortant qui est copié dans l'indicateur de retenue CF.

Rotation à gauche (Rotate Left) : ROL opérande,n

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

Exemple :

mov al,11001011B

rol al,1

→ réinjection du bit sortant qui est copié dans l'indicateur de retenue CF.

Rotation à droite avec passage par l'indicateur de retenue (Rotate Right through Carry) : RCR opérande,n

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

Exemple :

mov al,11001011B

rcr al,1

→ le bit sortant par la droite est copié dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la gauche.

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

RCL opérande,n

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

Exemple :

mov al,11001011B

rcl al,1

→ le bit sortant par la gauche est copié dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la droite.

3.11 Les instructions de branchement

Les instructions de branchement (ou **saut**) permettent de modifier l'ordre d'exécution des instructions du programme en fonction de certaines conditions. Il existe 3 types de saut :

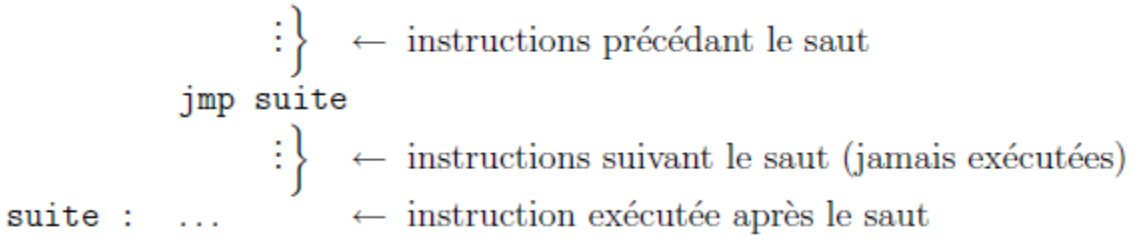
• saut inconditionnel ;

• sauts conditionnels ;

• appel de sous-programmes.

Instruction de saut inconditionnel : JMP label

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



Exemple :

boucle : inc ax

dec bx

jmp boucle

→ boucle infinie

Remarque : l'instruction JMP ajoute au registre IP (pointeur d'instruction) le nombre d'octets (distance) qui sépare l'instruction de saut de sa destination. Pour un saut en arrière, la distance est négative (codée en complément `a 2).

Instructions de sauts conditionnels : J*condition* label

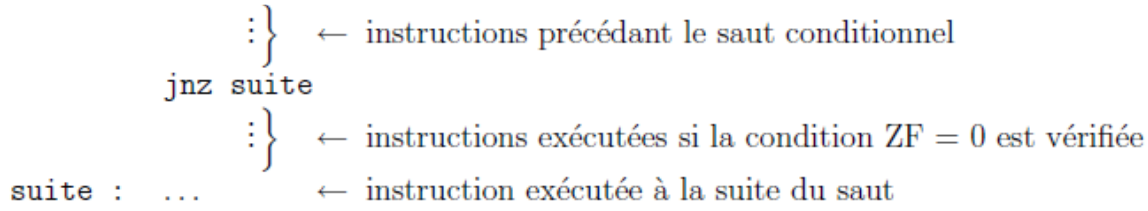
Un saut conditionnel n'est exécuté que si une certaine condition est satisfaite, sinon l'exécution se poursuit séquentiellement `a l'instruction suivante.

La condition du saut porte sur l'´etat de l'un (ou plusieurs) des indicateurs d'´etat (flags) du microprocesseur :

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

Remarque : les indicateurs sont positionnés en fonction du résultat de la dernière opération.

Exemple :



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

condition	nombres signés	nombres non signés
$=$	JEQ label	JEQ label
$>$	JG label	JA label
$<$	JL label	JB label
$\neq$	JNE label	JNE label

Exemple :

```
cmp ax,bx
jg superieur
jl inferieur
superieur : . . .
...
inferieur : . . .
```

Exercice d'application :

On veut additionner deux nombres signés N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H. Le résultat est rangé à l'offset 1102H s'il est positif, à l'offset 1103H s'il est négatif et `a l'offset 1104H s'il est nul

Appel de sous-programmes : pour éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme, on rédige la séquence une seule fois en lui attribuant un

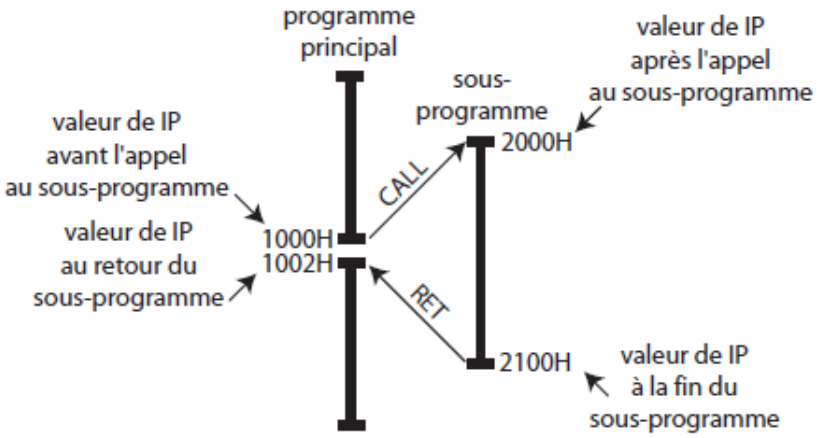
nom (au choix) et on l'appelle lorsqu'on en a besoin. Le programme appelant est le **programme principal**. La séquence appelée est un **sous-programme** ou **procédure**.
Ecriture d'un sous-programme :

```
nom_sp    PROC
            ⋮   ← instructions du sous-programme
          ret   ← instruction de retour au programme principal
nom_sp    ENDP
```

Remarque : une procédure peut être de type NEAR si elle se trouve dans le même segment ou de type FAR si elle se trouve dans un autre segment.
Exemple : ss prog1 PROC NEAR
ss prog2 PROC FAR
Appel d'un sous-programme par le programme principal : CALL procédure

```
      ⋮   ← instructions précédant l'appel au sous-programme
call nom_sp   ← appel au sous-programme
      ⋮   ← instructions exécutées après le retour au programme principal
```

Lors de l'exécution de l'instruction CALL, le pointeur d'instruction IP est chargé avec l'adresse de la première instruction du sous-programme. Lors du retour au programme appelant, l'instruction suivant le CALL doit être exécutée, c'est-`a-dire que IP doit être rechargé avec l'adresse de cette instruction.



Avant de charger IP avec l'adresse du sous-programme, l'adresse de retour au programme principal, c'est-`a-dire le contenu de IP, est sauvegardée dans une zone mémoire particulière appelée **pile**. Lors de l'exécution de l'instruction RET, cette adresse est récupérée `a partir de la pile et rechargée dans IP, ainsi le programme appelant peut se poursuivre.

Fonctionnement de la pile : la pile est une zone mémoire fonctionnant en mode LIFO (Last In First Out : dernier entré, premier sorti). Deux opérations sont possibles sur la pile :

- **empiler** une donnée : placer la donnée au sommet de la pile ;
- **dépiler** une donnée : lire la donnée se trouvant au sommet de la pile.

3.12 Méthodes de programmation

Etapes de la réalisation d'un programme :

• Définir le problème à résoudre : que faut-il faire exactement ?

• Déterminer des algorithmes, des organigrammes : comment faire ? Par quoi commencer, puis poursuivre ?

• Rédiger le programme (**code source**) :

– utilisation du jeu d'instructions (mnémoniques) ;

– création de documents explicatifs (documentation).

• Tester le programme en réel ;

• Corriger les erreurs (**bugs**) éventuelles : **déboguer** le programme puis refaire des tests jusqu''a obtention d'un programme fonctionnant de manière satisfaisante.

Langage machine et assembleur :

• Langage machine : codes binaires correspondant aux instructions ;

• Assembleur : logiciel de traduction du code source ´ecrit en langage assembleur (mnémoniques).

Réalisation pratique d'un programme :

• Rédaction du code source en assembleur `a l'aide d'un éditeur (logiciel de traitement de texte ASCII) :

– edit sous MS-DOS,

– notepad (bloc-note) sous Windows,

• Assemblage du code source (traduction des instructions en codes binaires) avec un assembleur :

– MASM de Microsoft,

– TASM de Borland,

– A86 disponible en shareware sur Internet, ...

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

• Chargement en mémoire centrale et exécution : rôle du système d'exploitation (ordinateur) ou d'un moniteur (carte de développement `a base de microprocesseur).

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

• l'exécution pas `a pas;

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

• la pose de points d'arrêt ...

Structure d'un fichier source en assembleur : pour faciliter la lisibilité du code source en assembleur, on le rédige sous la forme suivante :

labels instructions commentaires

label1 : mov ax,60H ; ceci est un commentaire ...

...

...

...

sous prog1 proc near ; sous-programme

...

...

...

sous prog1 endp

...

...

...

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**, reservation d'un octet ;

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

Exemples :

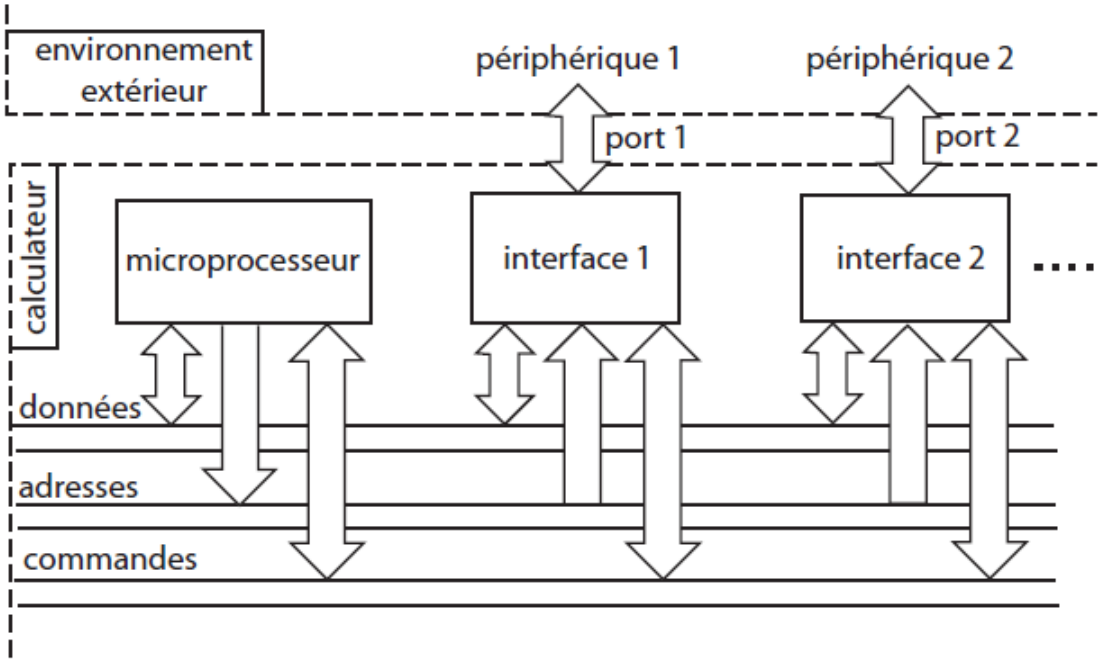
```
nombre1 db 25
nombre2 dw ? ; pas de valeur initiale
buffer db 100 dup ( ?) ; réservation d'une zone mémoire
; de 100 octets
```

Chapitre 4 : Les interfaces d'entrées/sorties

4.1 Définitions

Une **interface d'entrées/sorties** est un circuit intégré permettant au microprocesseur de communiquer avec l'environnement extérieur (périphériques) : clavier, écran, imprimante, modem, disques, processus industriel, ...

Les interfaces d'E/S sont connectées au microprocesseur `a travers les bus d'adresses, de données et de commandes.

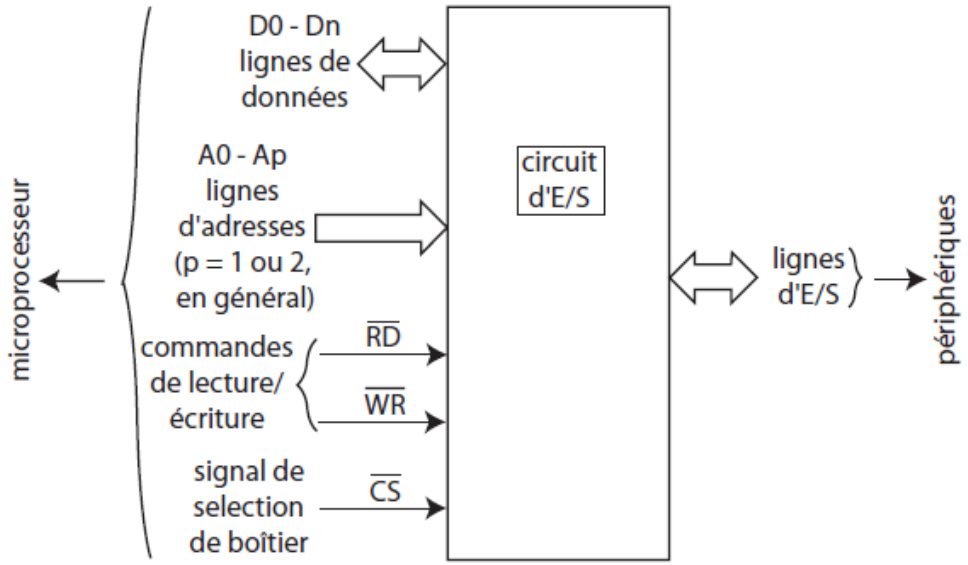


Les points d'accès aux interfaces sont appelés **ports**.

Exemples :

interface	port	exemple de périphérique
interface parallèle	port parallèle	imprimante
interface série	port série	modem

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



4.2 Adressage des ports d'E/S

Un circuit d'E/S possède des registres pour gérer les échanges avec les périphériques :

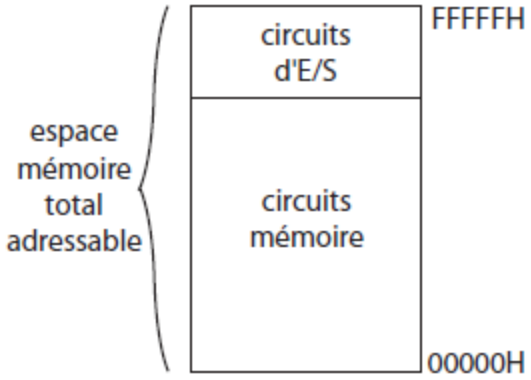
• registres de configuration ;

• registres de données.

A chaque registre est assigné une adresse : le microprocesseur accède à un port d'E/S en spécifiant l'adresse de l'un de ses registres.

Le microprocesseur peut voir les adresses des ports d'E/S de deux manières :

• **adressage cartographique** : les adresses des ports d'E/S appartiennent au même espace mémoire que les circuits mémoire (on dit que les E/S sont **mappées en mémoire**) :



Conséquences :

– l'espace d'adressage des mémoires diminue ;

– l'adressage des ports d'E/S se fait avec une adresse de même longueur (même nombre de bits) que pour les cases mémoires ;

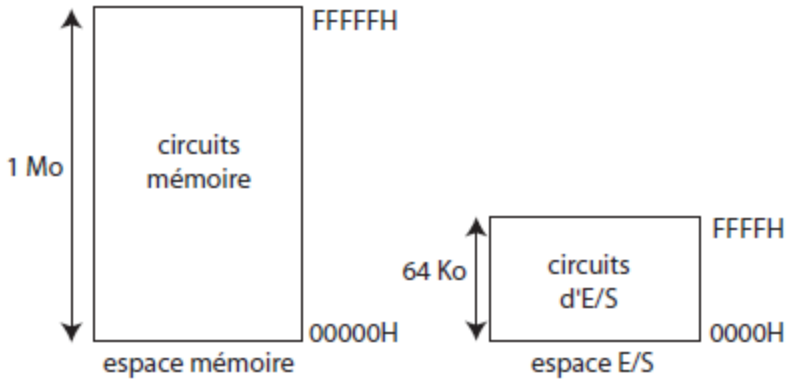
– toutes les instructions employées avec des cases mémoires peuvent être appliquées aux ports d'E/S : les mêmes instructions permettent de lire et écrire dans la mémoire et les ports d'E/S, tous les modes d'adressage étant valables pour les E/S.

• **adressage indépendant** : le microprocesseur considère deux espaces distincts :

– l'espace d'adressage des mémoires ;

– l'espace d'adressage des ports d'E/S.

C'est le cas du microprocesseur 8086 :



Conséquences :

– contrairement à l'adressage cartographique, l'espace mémoire total adressable n'est pas diminué ;

– l'adressage des ports d'E/S peut se faire avec une adresse plus courte (nombre de bits inférieur) que pour les circuits mémoires ;

– les instructions utilisées pour l'accès `a la mémoire ne sont plus utilisables pour l'accès aux ports d'E/S : ceux-ci disposent d'instructions spécifiques ;

– une même adresse peut désigner soit une case mémoire, soit un port d'E/S : le microprocesseur doit donc fournir un signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S.

Remarque : l'adressage indépendant des ports d'E/S n'est possible que pour les microprocesseurs possédant un signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S ainsi que les instructions spécifiques pour l'accès aux ports d'E/S. Par contre, l'adressage cartographique est possible pour tous les microprocesseurs.

4.3 Gestion des ports d'E/S par le 8086

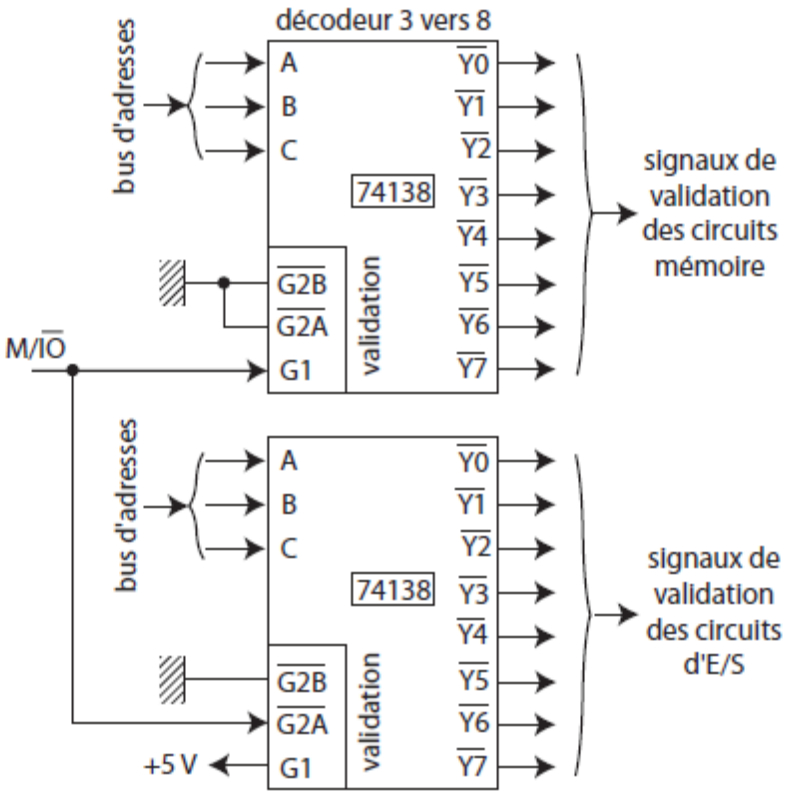
Le 8086 dispose d'un espace mémoire de 1 Mo (adresse d'une case mémoire sur 20 bits) et d'un espace d'E/S de 64 Ko (adresse d'un port d'E/S sur 16 bits).

Le signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S est la ligne **M/IO** :

• pour un accès `a la mémoire, M/IO = 1 ;

• pour un accès aux ports d'E/S, M/IO = 0.

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 `a 1.

Lecture d'un port d'E/S :

• si l'adresse du port d'E/S est sur un octet :

IN AL, adresse : lecture d'un port sur 8 bits ;

IN AX, adresse : lecture d'un port sur 16 bits.

• si l'adresse du port d'E/S est sur deux octets :

IN AL,DX : lecture d'un port sur 8 bits ;

IN AX,DX : lecture d'un port sur 16 bits.

Où le registre DX contient l'adresse du port d'E/S à lire.

Ecriture d'un port d'E/S :

• si l'adresse du port d'E/S est sur un octet :

OUT adresse, AL : écriture d'un port sur 8 bits ;

OUT adresse, AX : écriture d'un port sur 16 bits.

• si l'adresse du port d'E/S est sur deux octets :

OUT DX, AL : écriture d'un port sur 8 bits ;

OUT DX, AX : écriture d'un port sur 16 bits.

Où le registre DX contient l'adresse du port d'E/S à écrire.

Exemples :

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

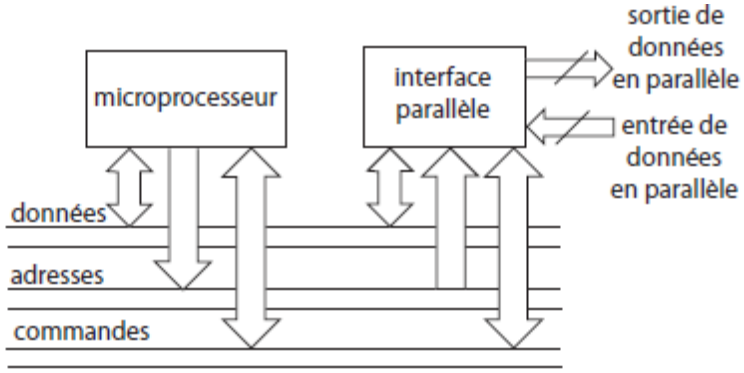
```
mov dx,300H
in al, dx
```

• écriture de la valeur 1234H dans le port d'E/S sur 16 bits `a l'adresse 49H :

```
mov ax,1234H
out 49H,ax
```

4.4 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.

Brochage du 8255 :

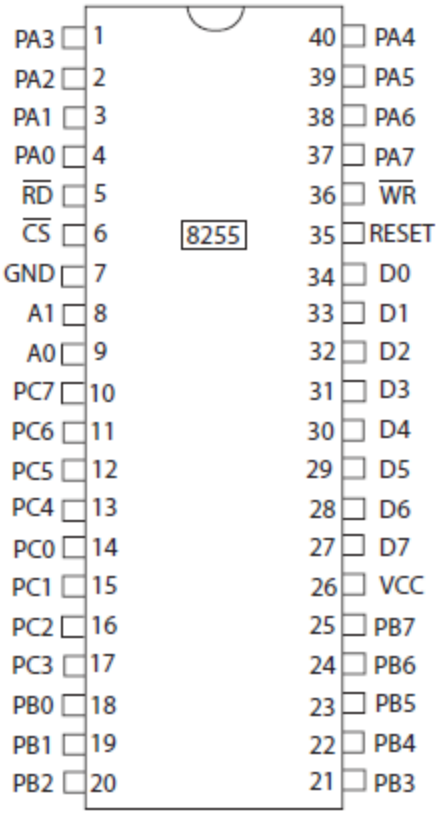
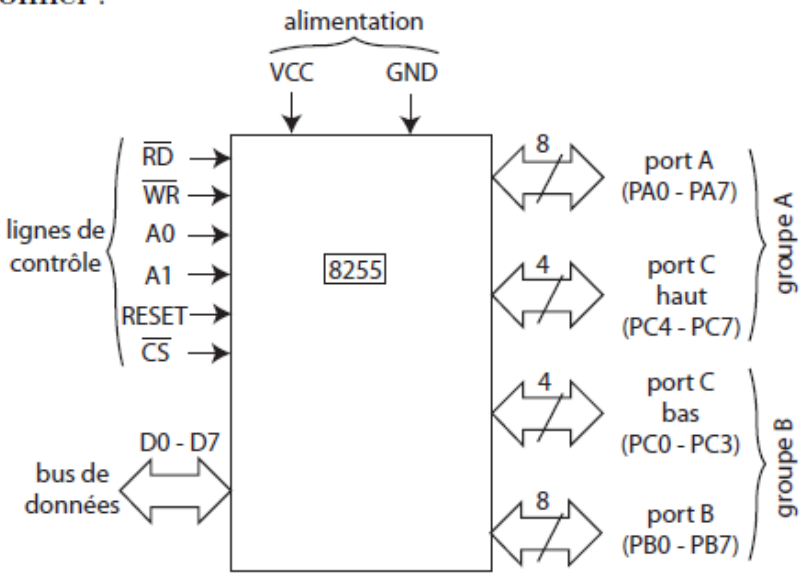


Schéma fonctionnel :



Le 8255 contient 4 registres :

• trois registres contenant les donn´ees pr´esentes sur les ports A, B et C;

• un registre de commande pour la configuration des port A, B et C en entrées et/ou en sorties.

Accès aux registres du 8255 : les lignes d'adresses A0 et A1 définissent les adresses des registres du 8255 :

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	opération
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	illégal
X	X	1	1	0	pas de transaction

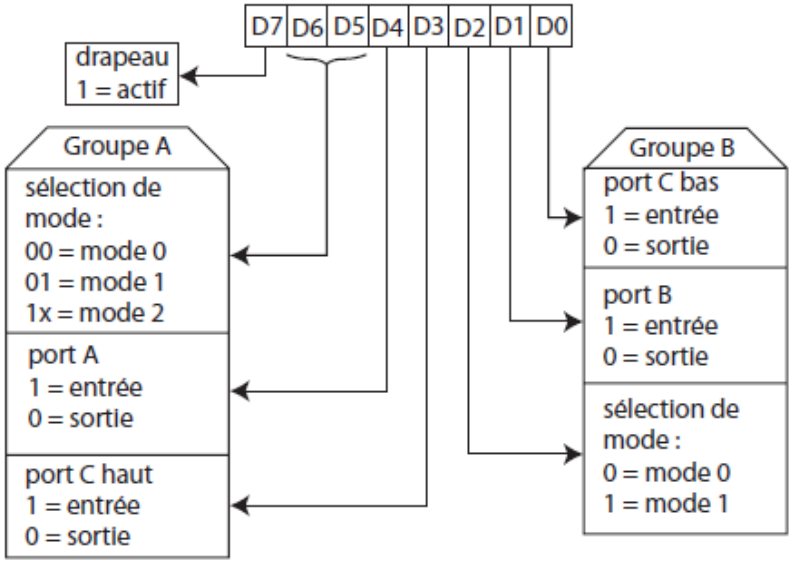
Remarque : le registre de commande est accessible uniquement en écriture, la lecture de ce registre n'est pas autorisée.

Configuration du 8255 : les ports peuvent être configurés en entrées ou en sorties selon le contenu du registre de commande. De plus le 8255 peut fonctionner selon 3 modes : **mode 0**, **mode 1** ou **mode 2**.

Le mode 0 est le plus simple : les ports sont configurés en entrées/sorties de base. Les données écrites dans les registres correspondants sont mémorisées sur les lignes de sorties ; l'´etat des lignes d'entrées est recopié dans les registres correspondants et n'est pas mémorisé.

Les modes 1 et 2 sont plus complexes. Ils sont utilisés pour le dialogue avec des périphériques nécessitant un asservissement.

Structure du registre de commande :



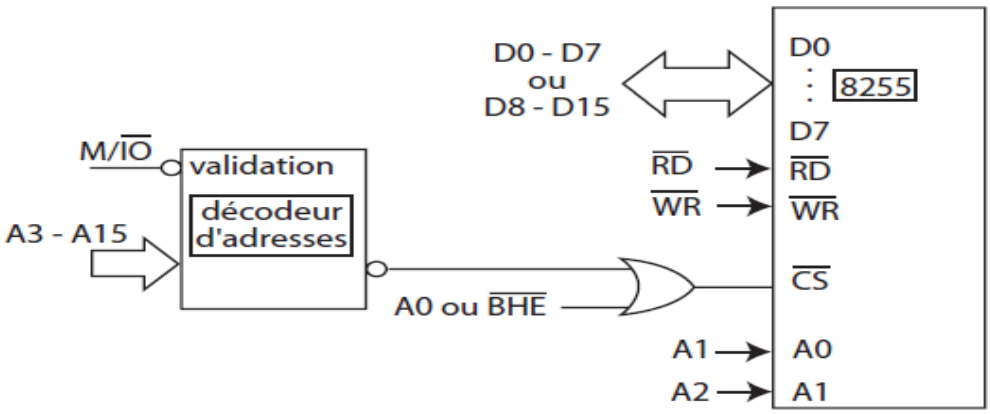
Connexion du 8255 sur les bus du 8086 : le bus de données du 8255 est sur 8 bits alors que celui du microprocesseur 8086 est sur 16 bits. On peut donc connecter le bus de données du 8255 sur les lignes de données de poids faible du 8086 (D0 - D7) ou sur celles de poids fort (D8 - D15).

Une donnée est envoyée (ou reçue) par le microprocesseur 8086 :

• sur la partie faible du bus de données lorsque l'adresse à écrire (ou `a lire) est paire : validation par A0 ;

• sur la partie haute lorsque l'adresse est impaire : validation par BHE.

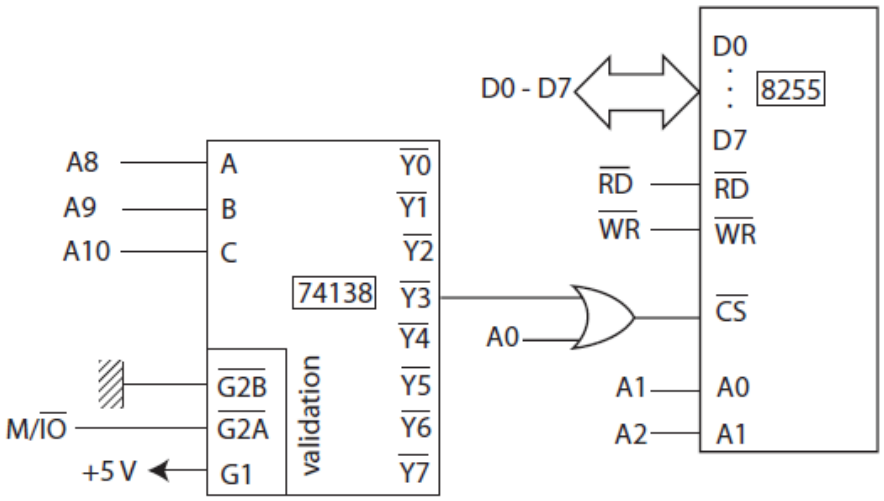
Ainsi l'un de ces deux signaux A0 ou BHE doit être utilisé pour sélectionner le 8255 :



Conséquence : les adresses des registres du 8255 se trouvent `a des adresses paires (validation par A0) ou impaires (validation par BHE).

Le décodeur d'adresses détermine l'**adresse de base** du 8255 ; les lignes A1 et A2 déterminent les adresses des registres du 8255.

Exemple : connexion du 8255 sur la partie faible du bus de données du 8086, avec décodage d'adresses incomplet (les lignes d'adresses A3 - A15 ne sont pas toutes utilisées) :



Détermination de l'adresse du 8255 :

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
X	X	X	X	X	0	1	1	X	X	X	X	X	A1	A0	0
adresse de base = 300H													sélection de registre		$\overline{CS}$ = 0

• A2 = 0 et A1 = 0 : adresse du port A = adresse de base + 0 = 300H ;

• A2 = 0 et A1 = 1 : adresse du port B = adresse de base + 2 = 302H ;

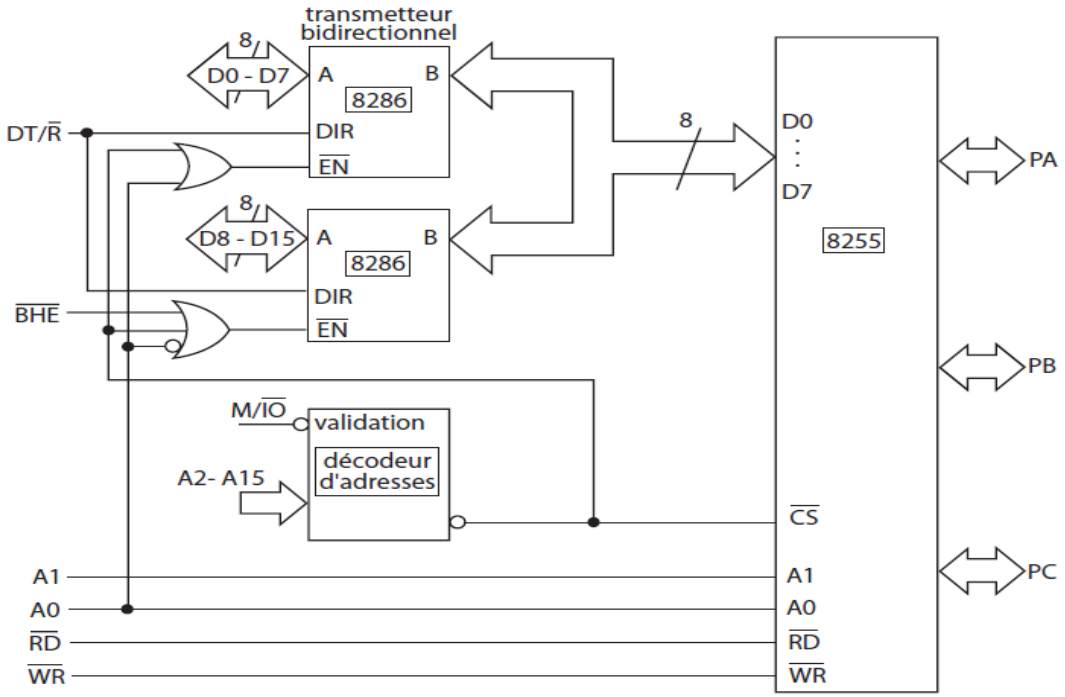
• A2 = 1 et A1 = 0 : adresse du port C = adresse de base + 4 = 304H ;

• A2 = 1 et A1 = 1 : adresse du registre de commande = adresse de base + 6 = 306H.

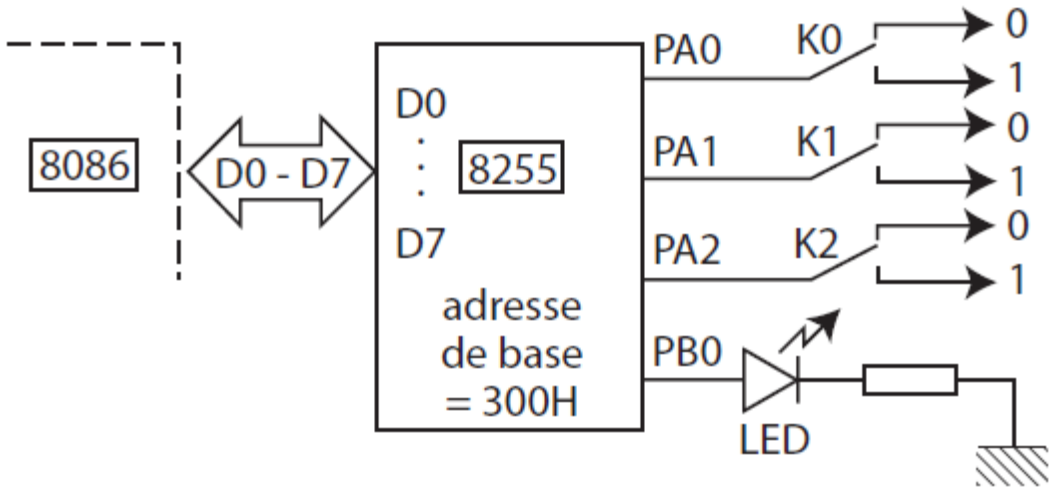
Remarque : le décodage d'adresses étant incomplet, le 8255 apparaît dans plusieurs plages d'adresses selon les valeurs des bits d'adresses non décodés (A7 - A13 et A12 -A15). Dans cet exemple, l'adresse de base 300H correspond `a la première adresse possible (bits d'adresses non décodés tous égaux `a 0).

Remarque : si on veut que le 8255 possède des adresses consécutives (par exemple 300H,

301H, 302H et 303H), on peut utiliser le schéma suivant qui exploite tout le bus de données (D0 - D15) du 8086 :



Exercice d'application



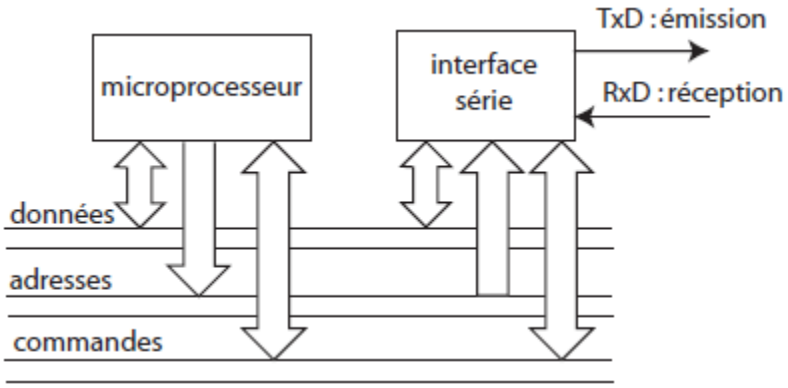
Soit le montage ci-dessus

On veut que la led s'allume lorsqu'on a la combinaison : **K0 = 1** et **K1 = 0** et **K2 = 1**.

Ecrire le programme correspondant.

4.5 L'interface série 8250

Une interface série permet d'échanger des données entre le microprocesseur et un périphérique **bit par bit**.



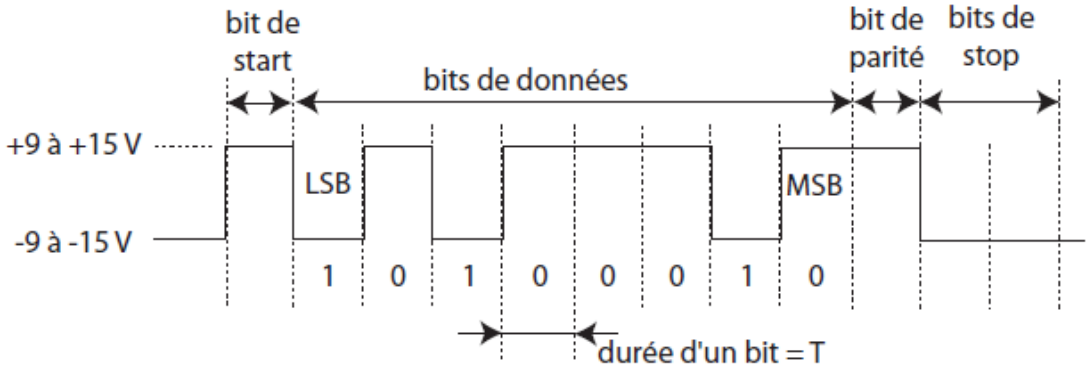
Avantage : diminution du nombre de connexions (1 fil pour l'´emission, 1 fil pour la réception).
Inconvénient : vitesse de transmission plus faible que pour une interface parallèle.
Il existe deux types de transmissions séries :

• **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 :



• l’´etat 1 correspond `a une tension **négative** comprise entre −9 et −15 V, l’´etat 0 à une tension **positive** comprise entre +9 et +15 V. Au repos, la ligne est `a l’état 1 (tension négative) ;

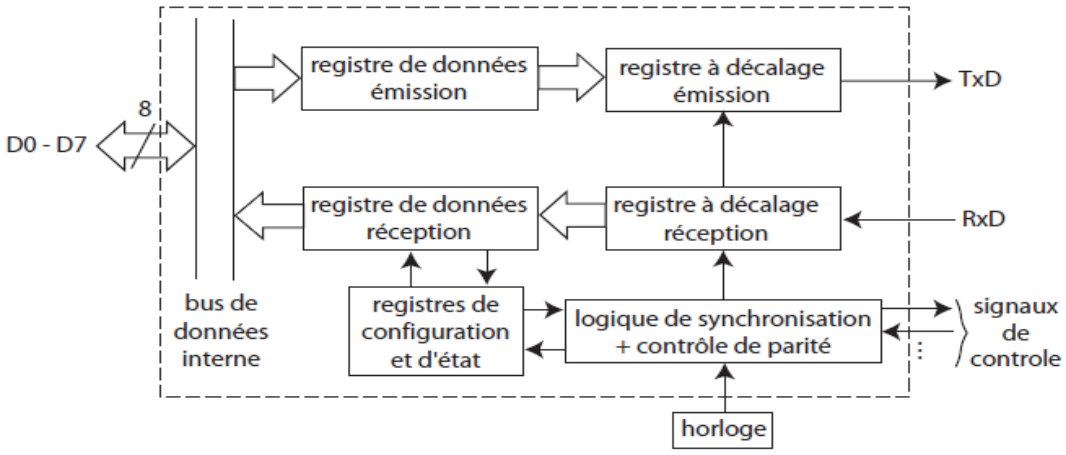
• le **bit de start** marque le début de la transmission du caractère ;

• 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 `a 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.

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) ;

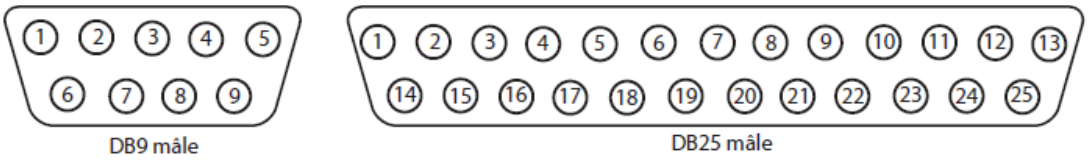
• 6850 (Motorola).

Connexion de deux équipements par une liaison série RS232 : les équipements qui peuvent être connectés `a travers une liaison série RS232 sont de deux types :

• les **équipements terminaux de données** (DTE : Data Terminal Equipment) qui génèrent les données à transmettre, exemple : un ordinateur ;

• les **équipements de communication de données** (DCE : Data Communication Equipment) qui transmettent les données sur les lignes de communication, exemple : un modem.

Pour connecter ces équipements, on utilise des connecteurs normalisés **DB9** ou **DB25** :



DB9 mâle DB25 mâle

Différents signaux sont transportés par ces connecteurs :

signal	n° broche DB9	n° broche DB25	description	sens	
				DTE	DCE
TxD	3	2	Transmit Data	sortie	entrée
RxD	2	3	Receive Data	entrée	sortie
RTS	7	4	Request To Send	sortie	entrée
CTS	8	5	Clear To Send	entrée	sortie
DTR	4	20	Data Terminal Ready	sortie	entrée
DSR	6	6	Data Set Ready	entrée	sortie
DCD	1	8	Data Carrier Detect	entrée	sortie
RI	9	22	Ring Indicator	entrée	sortie
GND	5	7	Ground	—	—

Seuls les 2 signaux TxD et RxD servent à transmettre les données. Les autres signaux sont des signaux de contrôle de l'échange de données.

Mise en œuvre d'une interface série, l'UART 8250 :

Brochage du 8250 :

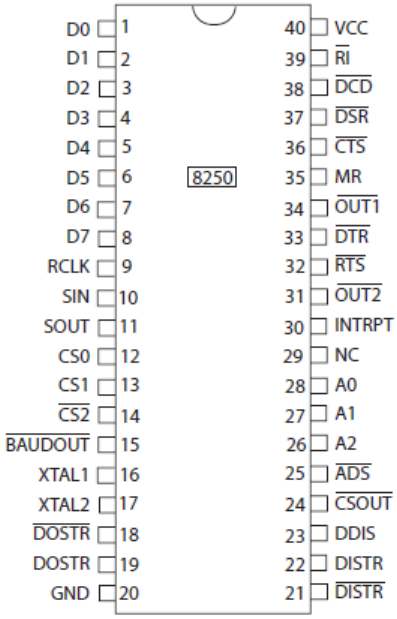
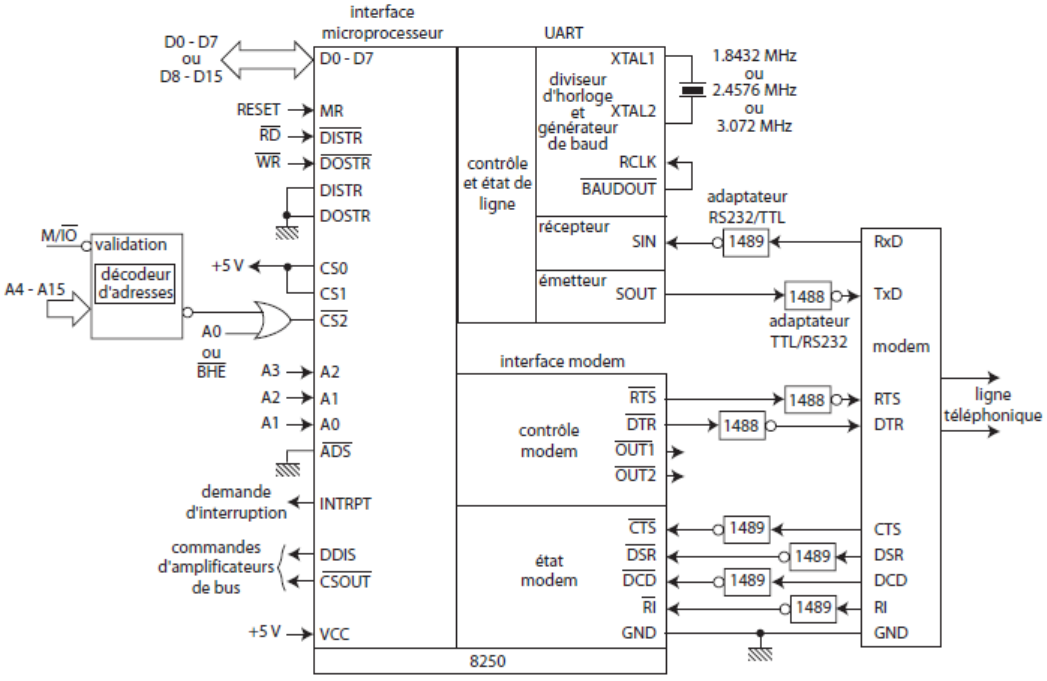


Schéma fonctionnel :



Accès aux registres du 8250 : le 8250 possède 11 registres. Comme il n'y a que 3 bits d'adresses (A0, A1 et A2), plusieurs registres doivent se partager la même adresse :

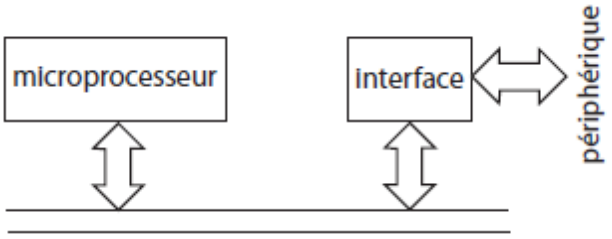
DLAB	A2	A1	A0	registre
0	0	0	0	**RBR** : Receiver Buffer Register, registre de réception (accessible seulement en lecture)
0	0	0	0	**THR** : Transmitter Holding Register, registre d'émission (accessible seulement en écriture)
1	0	0	0	**DLL** : Divisor Latch LSB, octet de poids faible du diviseur d'horloge
1	0	0	1	**DLM** : Divisor Latch MSB, octet de poids fort du diviseur d'horloge
0	0	0	1	**IER** : Interrupt Enable Register, registre d'autorisation des interruptions
X	0	1	0	**IIR** : Interrupt Identification Register, registre d'identification des interruptions
X	0	1	1	**LCR** : Line Control Register, registre de contrôle de ligne
X	1	0	0	**MCR** : Modem Control Register, registre de contrôle modem
X	1	0	1	**LSR** : Line Status Register, registre d'état de la ligne
X	1	1	0	**MSR** : Modem Status Register, registre d'état du modem
X	1	1	1	**SCR** : Scratch Register, registre à usage général

En fonction de l'é tat de DLAB (Divisor Latch Access Bit = bit de poids fort du registre LCR), on a accès soit au registre d'´emission/r´réception, soit au diviseur d'horloge, soit au masque d'interruptions.

Chapitre 5: Les interruptions

5.1 Définition d'une interruption

Soit un microprocesseur qui doit ´échanger des informations avec un périphérique :



Il y a deux méthodes possibles pour recevoir les données provenant des périphériques :

• **scrutation périodique** (ou **polling**) : le programme principal contient des instructions qui lisent cycliquement l'´etat des ports d'E/S.

Avantage : facilité de programmation.

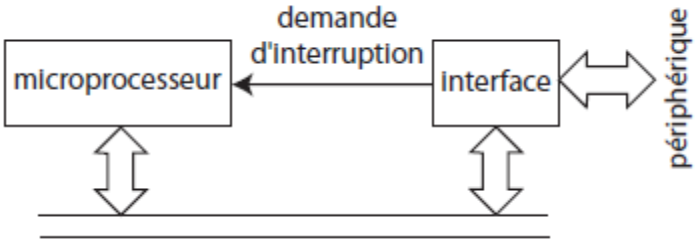
Inconvénients :

– perte de temps s'il y a de nombreux périphériques `a interroger ;

– de nouvelles données ne sont pas toujours présentes ;

– des données peuvent être perdues si elles changent rapidement.

• **interruption** : lorsqu'une donnée apparaît sur un périphérique, le circuit d'E/S le signale au microprocesseur pour que celui-ci effectue la lecture de la donnée : c'est une **demande d'interruption** (IRQ : Interrupt Request) :



Avantage : le microprocesseur effectue une lecture des ports d'E/S seulement lorsqu'une donnée est disponible, ce qui permet de gagner du temps et d'´eviter de perdre des données.

Exemples de périphériques utilisant les interruptions :

• clavier : demande d'interruption lorsqu'une touche est enfoncée ;

• port série : demande d'interruption lors de l'arrivée d'un caractère sur la ligne de transmission.

Remarque : les interruptions peuvent être générées par le microprocesseur lui-même en cas de problèmes tels qu'une erreur d'alimentation, une division par zéro ou un circuit mémoire défectueux (erreurs fatales). Dans ce cas, la demande d'interruption conduit à l'arrêt du microprocesseur.

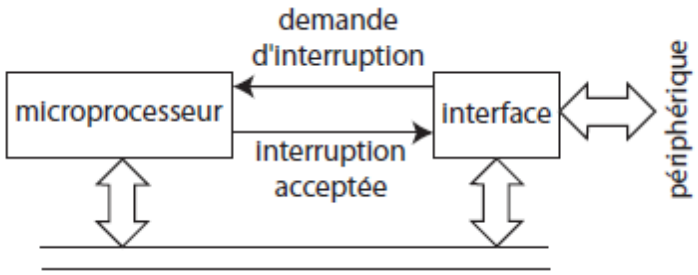
5.2 Prise en charge d'une interruption par le microprocesseur

A la suite d'une demande d'interruption par un périphérique :

• le microprocesseur termine l'exécution de l'instruction en cours ;

• il range le contenu des principaux registres sur la pile de sauvegarde : pointeur d'instruction, flags, ...

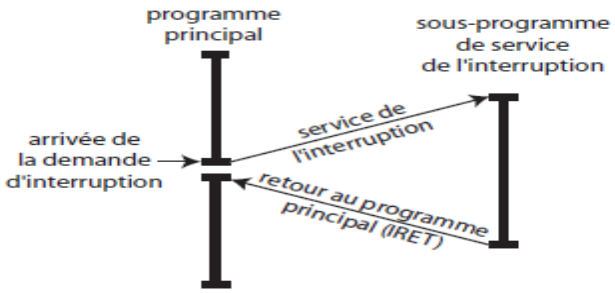
• il émet un **accusé de réception** de demande d'interruption (Interrupt Acknowledge) indiquant au circuit d'E/S que la demande d'interruption est acceptée :



Remarque : le microprocesseur peut refuser la demande d'interruption : celle-ci est alors **masquée**. Le masquage d'une interruption se fait généralement en positionnant un flag dans le registre des indicateurs d'´etat. Il existe cependant des interruptions **non masquables** qui sont toujours prises en compte par le microprocesseur.

• il abandonne l'exécution du programme en cours et va exécuter un **sous-programme de service de l'interruption** (ISR : Interrupt Service Routine) ;

• après l'exécution de l'ISR, les registres sont restaurés `a partir de la pile et le microproceseur reprend l'exécution du programme qu'il avait abandonné :



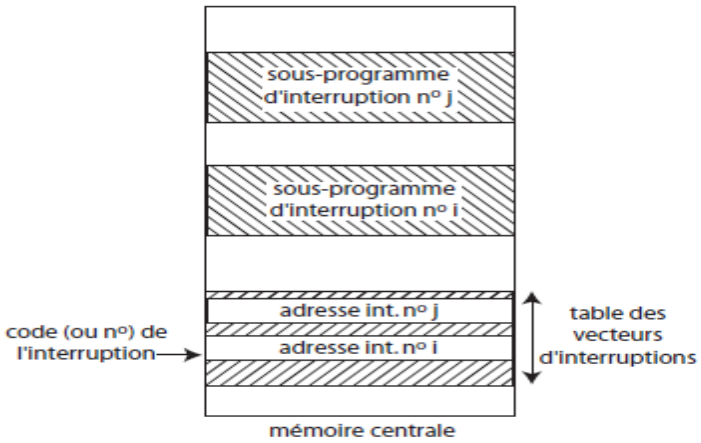
Remarque : la dernière instruction d'un sous-programme de service d'interruption doit être l'instruction IRET : retour d'interruption.

Si plusieurs interruptions peuvent se produire en même temps, on doit leur affecter une **priorité** pour que le microprocesseur sache dans quel ordre il doit servir chacune d'entre elle.

5.3 Adresses des sous-programmes d'interruptions

Lorsqu'une interruption survient, le microprocesseur a besoin de connaître l'adresse du sous-programme de service de cette interruption. Pour cela, la source d'interruption place sur le bus de données un code numérique indiquant la nature de l'interruption. Le microprocesseur

utilise ce code pour rechercher dans une table en mémoire centrale l'adresse du sous-programme d'interruption à exécuter. Chaque élément de cette table s'appelle un **vecteur d'interruption** :



Lorsque les adresses des sous-programmes d'interruptions sont gérées de cette manière, on dit que les interruptions sont **vectorisées**.

Avantage de la vectorisation des interruptions : l'emplacement d'une ISR peut être n'importe où dans la mémoire, il suffit de spécifier le vecteur d'interruption correspondant.

5.4 Les interruptions du 8086

Le microprocesseur 8086 peut gérer jusqu''a 256 interruptions. Chaque interruption reçoit un numéro compris entre 0 et 255 appel´e **type** de l'interruption.

Trois sortes d'interruptions sont reconnues par le 8086 :

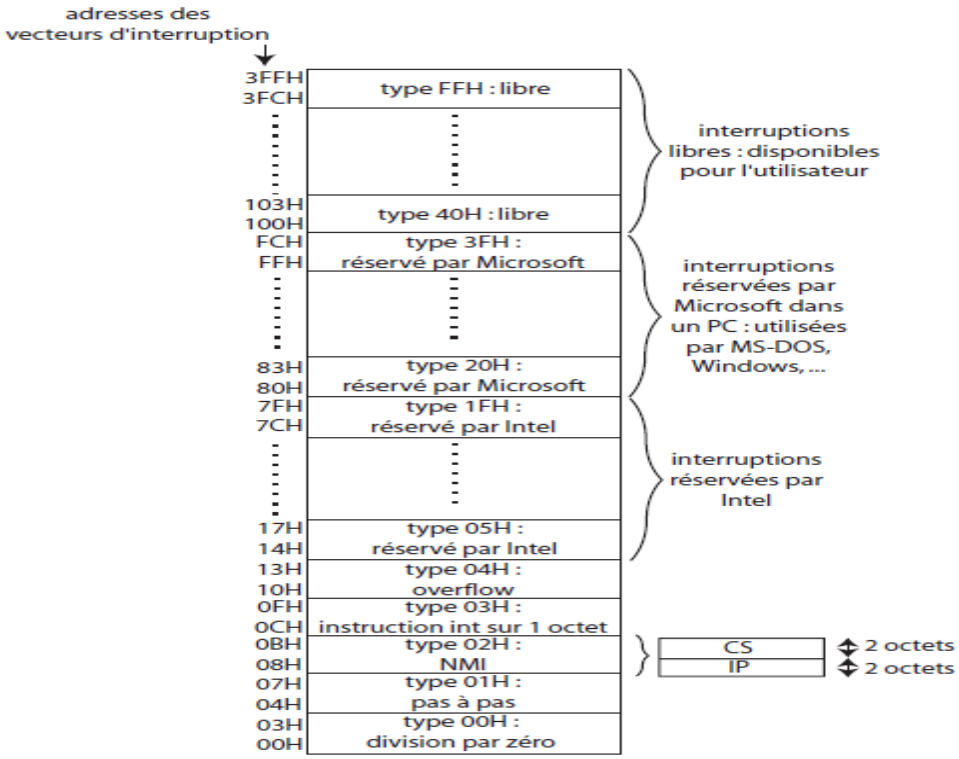
• interruptions **matérielles** produites par l'activation des lignes INTR et NMI du microprocesseur ;

• interruptions **logicielles** produites par l'instruction INT n, o`u n est le type de l'interruption

;• interruptions **processeur** générées par le microprocesseur en cas de dépassement, de division par zéro ou lors de l'exécution pas `a pas d'un programme.

Les interruptions du 8086 sont vectorisées. La table des vecteurs d'interruptions doit obligatoirement commencer `a l'adresse physique 00000H dans la mémoire centrale.

Chaque vecteur d'interruption est constitué de 4 octets représentant une adresse logique du type **CS : IP**.



Remarque : correspondance entre le type de l'interruption et l'adresse du vecteur correspondant:

adresse vecteur d'interruption = 4 × type de l'interruption

Exemple : interruption 20H, adresse du vecteur = 4 × 20H = 80H.

La table des vecteurs d'interruptions est chargée par le programme principal (carte `a microprocesseur)
ou par le système d'exploitation (ordinateur) au démarrage du système.
Elle peut être modifiée en cours de fonctionnement (détournement des vecteurs d'interruptions).

5.5 Le contrôleur programmable d'interruptions 8259

Le microprocesseur 8086 ne dispose que de deux lignes de demandes d'interruptions matérielles (NMI et INTR). Pour pouvoir connecter plusieurs périphériques utilisant des interruptions, on peut utiliser le contrôleur programmable d'interruptions 8259 dont le rôle est de :

• recevoir des demandes d'interruptions des périphériques ;
• résoudre les priorités des interruptions ;
• générer le signal INTR pour le 8086 ;
• émettre le numéro de l'interruption sur le bus de données.

Un 8259 peut gérer jusqu''a 8 demandes d'interruptions matérielles.

Brochage du 8259 :

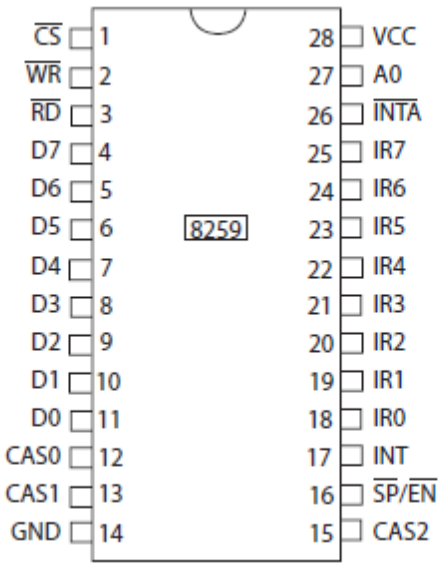
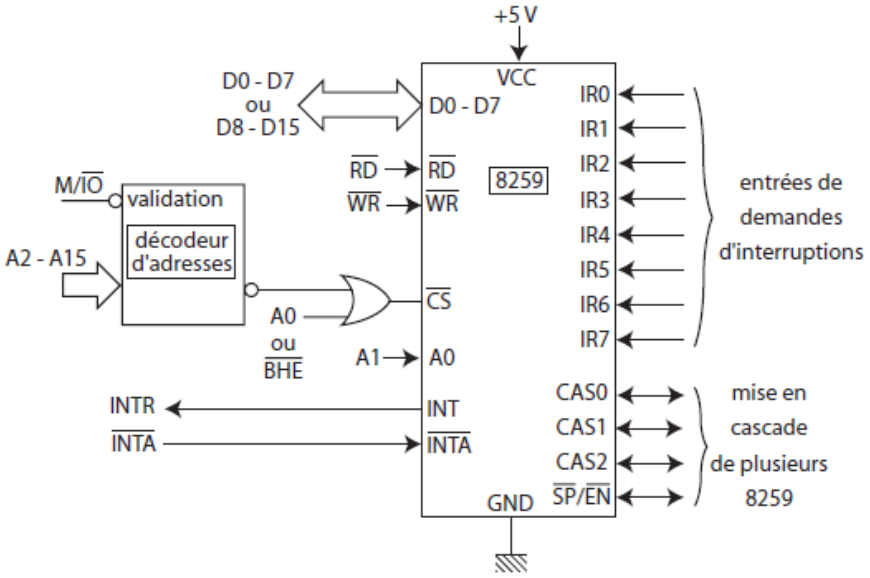
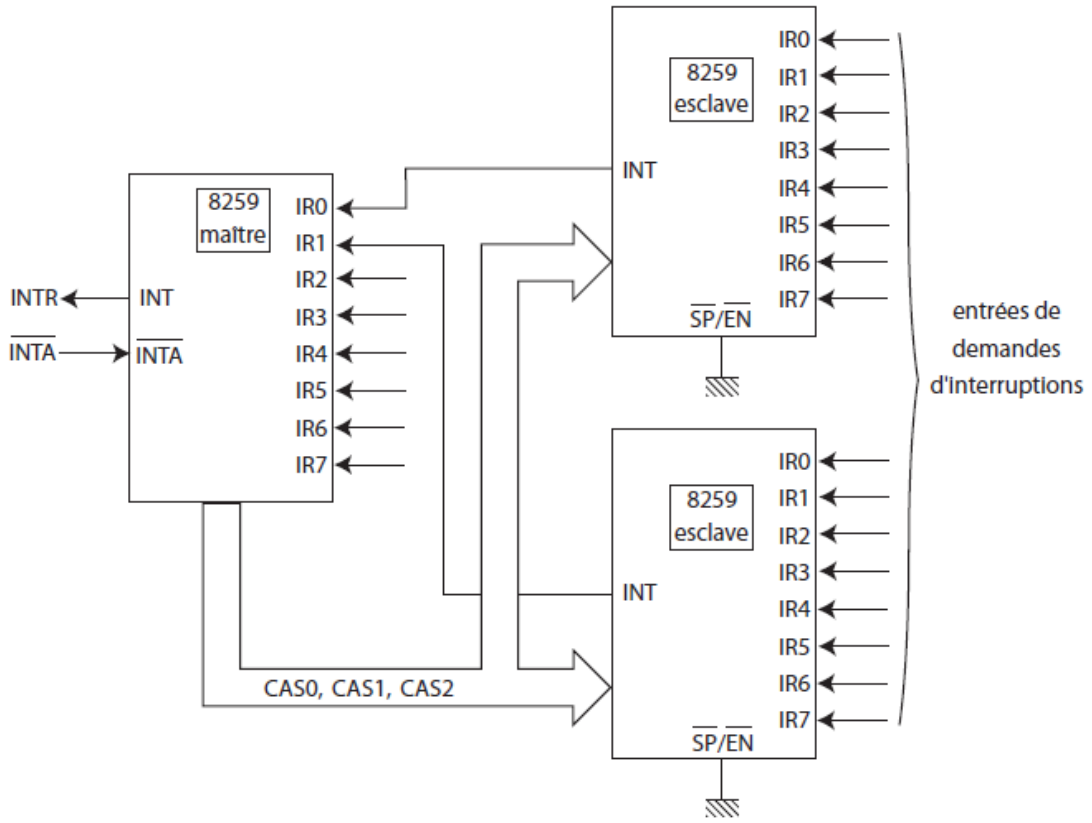


Schéma fonctionnel :



Remarque : si le nombre de demandes d'interruptions est supérieur à 8, on peut placer plusieurs 8259 en cascade :



Annexe

Jeu d'instructions du 8086

Transfert de données :

Général	
MOV	Déplacement d'un octet ou d'un mot
PUSH	Ecriture d'un mot au sommet de la pile
POP	Lecture d'un mot au sommet de la pile
XCHG	Echange d'octets ou de mots
XLAT ou XLATB	Traduction d'un octet à l'aide d'une table
Entrées/Sorties	
IN	Lecture d'un port d'E/S
OUT	Ecriture d'un port d'E/S
Transfert d'adresses	
LEA	Chargement d'une adresse effective
LDS	Chargement d'un pointeur utilisant DS
LES	Chargement d'un pointeur utilisant ES
Transfert des flags	
LAHF	Transfert des 5 flags bas dans AH
SAHF	Transfert de AH dans les 5 flags bas
PUSHF	Sauvegarde des flags sur la pile
POPF	Restauration des flags à partir de la pile

Instructions arithmétiques :

Addition	
ADD	Addition d'octets ou de mots
ADC	Addition d'octets ou de mots avec retenue
INC	Incrémentation de 1 d'un octet ou d'un mot
AAA	Ajustement ASCII de l'addition
DAA	Ajustement décimal de l'addition

Soustraction	
SUB	Soustraction d'octets ou de mots
SBB	Soustraction d'octets ou de mots avec retenue
DEC	Décrémentation de 1 d'un octet ou d'un mot
NEG	Complémentation à 2 d'un octet ou d'un mot (changement de signe)
CMP	Comparaison d'octets ou de mots
AAS	Ajustement ASCII de la soustraction
DAS	Ajustement décimal de la soustraction
Multiplication	
MUL	Multiplication non signée d'octets ou de mots
IMUL	Multiplication signée d'octets ou de mots
AAM	Ajustement ASCII de la multiplication
Division	
DIV	Division non signée d'octets ou de mots
IDIV	Division signée d'octets ou de mots
AAD	Ajustement ASCII de la division
CBW	Conversion d'octet en mot
CWD	Conversion de mot en double mot

Instructions logiques :

Logique	
NOT	Complément à 1 d'un octet ou d'un mot
AND	ET logique de deux octets ou de deux mots
OR	OU logique de deux octets ou de deux mots
XOR	OU exclusif logique de deux octets ou de deux mots
TEST	Comparaison, à l'aide d'un ET, d'octets ou de mots
Décalages	
SHL/SAL	Décalage à gauche arithmétique ou logique (octet ou mot)
SHR	Décalage logique à droite d'un octet ou d'un mot
SAR	Décalage arithmétique à droite d'un octet ou d'un mot
Rotations	
ROL	Rotation à gauche d'un octet ou d'un mot)
ROR	Rotation à droite d'un octet ou d'un mot
RCL	Rotation à gauche incluant CF (octet ou mot)
RCR	Rotation à droite incluant CF (octet ou mot)

Instructions sur les chaînes de caractères :

Préfixes	
`REP`	Répétition tant que CX n'est pas nul
`REPE` ou `REPZ`	Répétition tant qu'il y a égalité et que CX n'est pas nul
`REPNE` ou `REPNZ`	Répétition tant qu'il n'y a pas égalité et que CX n'est pas nul
Instructions	
`MOVS` ou `MOVSB/MOVSW`	Déplacement de blocs d'octets ou de mots
`CMPS` ou `CMPSB/CMPSW`	Comparaison de blocs d'octets ou de mots
`SCAS` ou `SCASB/SCASW`	Exploration d'un bloc d'octets ou de mots
`LODS` ou `LODSB/LODSW`	Tranfert d'un octet ou d'un mot dans AL ou AX
`STOS` ou `STOSB/STOSW`	Chargement d'un bloc d'octets ou de mots par AL ou AX

Instructions de branchements :

Branchements inconditionnels	
`CALL`	Appel de procédure
`RET`	Retour d'une procédure
`JMP`	Saut inconditionnel
Contrôles d'itérations	
`LOOP`	Bouclage tant que $CX \neq 0$
`LOOPE` ou `LOOPZ`	Bouclage tant que $CX \neq 0$ et $ZF = 1$ (égalité)
`LOOPNE` ou `LOOPNZ`	Bouclage tant que $CX \neq 0$ et $ZF = 0$ (inégalité)
`JCXZ`	Saut si CX est nul
Interruptions	
`INT`	Interruption logicielle
`INTO`	Interruption si $OF = 1$ (overflow)
`IRET`	Retour d'une interruption

Instructions de branchements conditionnels :

Sauts conditionnels	
JA ou JNBE [1]	Saut si « supérieur » (si CF + ZF = 0)
JAE ou JNB [1]	Saut si « supérieur ou égal » (si CF = 0)
JB ou JNAE [1]	Saut si « inférieur » (si CF = 1)
JBE ou JNA [1]	Saut si « inférieur ou égal » (si CF + ZF = 1)
JC	Saut en cas de retenue (si CF = 1)
JE ou JZ	Saut si « égal » ou « nul » (si ZF = 1)
JG ou JNLE [2]	Saut si « plus grand » (si (SF $\oplus$ OF) + ZF = 0)
JGE ou JNL [2]	Saut si « plus grand ou égal » (si SF $\oplus$ OF = 0)
JL ou JNGE [2]	Saut si « plus petit » (si SF $\oplus$ OF = 1)
JLE ou JNG [2]	Saut si « plus petit ou égal » (si (SF $\oplus$ OF) + ZF = 1)
JNC	Saut si « pas de retenue » (si CF = 0)
JNE ou JNZ	Saut si « non égal » ou « non nul » (si (ZF = 0)
JNO	Saut si « pas de dépassement » (si OF = 0)
JNP ou JPO	Saut si « parité impaire » (si PF = 0)
JNS	Saut si « signe positif » (si SF = 0)
JO	Saut si « dépassement » (si OF = 1)
JP ou JPE	Saut si « parité paire » (si PF = 1)
JS	Saut si « signe négatif » (si SF = 1)

[1] concerne des nombres non signés.

[2] concerne des nombres signés.

Instructions de contrôle du 8086 :

Opérations sur les flags	
STC	Met le flag de retenue à 1
CLC	Efface le flag de retenue
CMC	Inverse l'état du flag de retenue
STD	Met le flag de direction à 1 (décrémentation)
CLD	Met le flag de direction à 0 (incrémentation)
STI	Autorise les interruptions sur INTR
CLI	Interdit les interruptions sur INTR
Synchronisation avec l'extérieur	
HLT	Arrêt du microprocesseur (sortie de cet état par interruption ou reset)
WAIT	Attente tant que $\overline{\text{TEST}}$ n'est pas à 0
ESC	Préfixe = instruction destinée à un coprocesseur
LOCK	Préfixe = réservation du bus pour l'instruction
Pas d'opération	
NOP	Pas d'opération

Deuxième partie : les microcontrôleurs

CHAPITRE 1 MICROCONTROLEUR PIC 16F877

1.1 Du microprocesseur au microcontrôleur

Le microcontrôleur est un dérivé du microprocesseur. Sa structure est celle des systèmes à base de microprocesseurs. Il est donc composé en plus de l'unité centrale de traitement, d'une mémoire (mémoire vive RAM et mémoire morte ROM), une (ou plusieurs) interface de communication avec l'extérieur matérialisé par les ports d'entrée/sortie.

En plus de cette configuration minimale, les microcontrôleurs sont dotés d'autres circuits d'interface qui vont dépendre du microcontrôleur choisi à savoir les systèmes de comptage (TIMER), les convertisseurs analogique/numérique (CAN) intégré, gestion d'une liaison série ou parallèle, un Watchdog (surveillance du programme), une sortie PWM (modulation d'impulsion),…

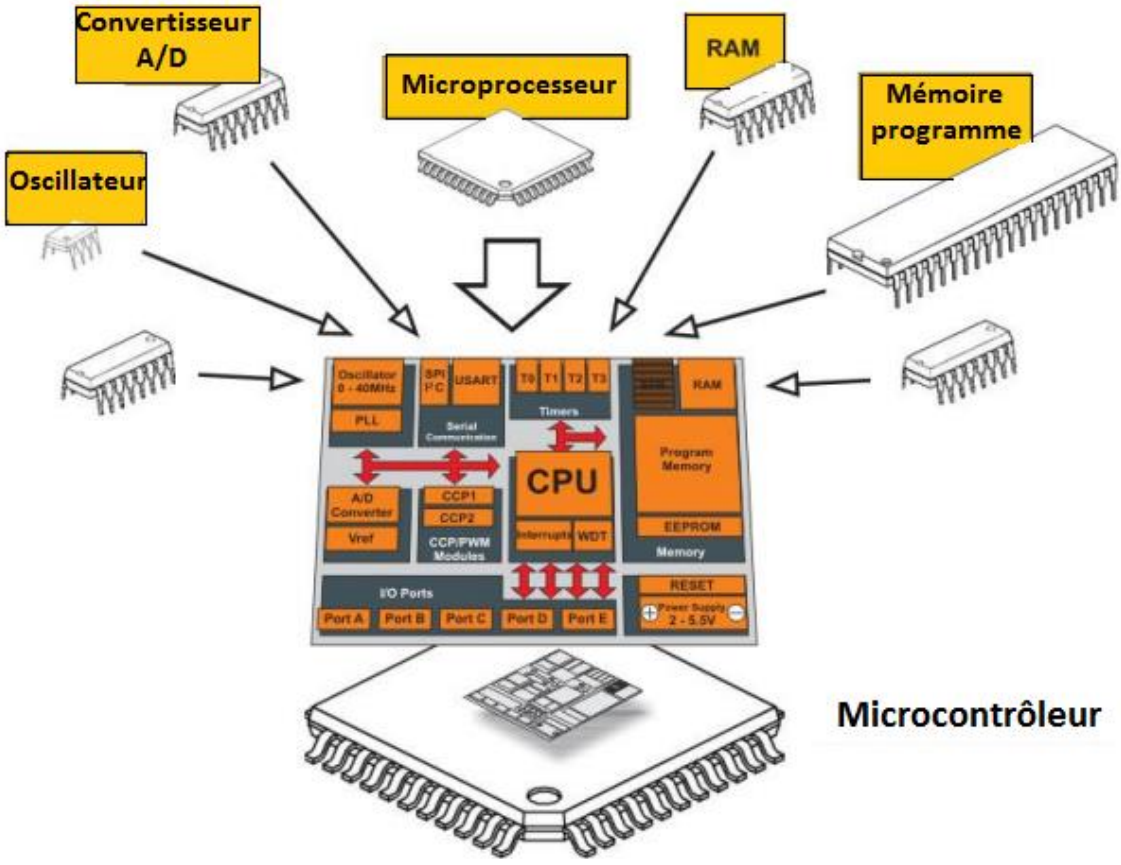


Fig 2.1 : Contenu type d'un microcontrôleur

Les microcontrôleurs améliorent l'intégration et le coût (lié à la conception et à la réalisation)

d'un système à base de microprocesseur en rassemblant ces éléments essentiels dans un seul circuit intégré. On parle alors de "système sur une puce" (en anglais : "System On chip").
Les microcontrôleurs sont plutôt dédiés aux applications qui ne nécessitent pas une grande quantité de calculs complexes, mais qui demandent beaucoup de manipulations d'entrées/sorties. C'est le cas de contrôle de processus.
Les systèmes à microprocesseur sont plutôt réservés pour les applications demandant beaucoup de traitement de l'information et assez peu de gestion d'entrées / sorties. Les ordinateurs sont réalisés avec des systèmes à microprocesseur.
Il existe plusieurs famille de microcontrôleurs dont les plus connues sont :
Atmel : AT; familles AT89Sxxxx, AT90xxxx, …
Motorolla : famille 68HCxxx, …
Microship : PIC ; familles 12Cxxx, 16Cxxx, 16Fxxx, 18Fxxx, …
Intel : famille 80C186XX
STMicroelectronics : famille STX
Analog Devices : famille ADuC
Nous allons nous intéresser dans le cadre de ce cours à la famille Microchip PIC (Programmable Integrated Circuit) de moyenne gamme (MIDRANGE).

1.2 Présentation d'un microcontrôleur PIC

Ils sont des composants dits RISC (Réduced Instructions Construction Set), ou encore composant à jeu d'instructions réduit. Chaque instruction complexe peut être programmée par plusieurs instructions simples. Sachant que plus on réduit le nombre d'instructions, plus facile et plus rapide qu'en est le décodage, et plus vite le composant fonctionne.
La famille des PIC à processeur 8 bits est subdivisée à l'heure actuelle en 3 grandes catégories:

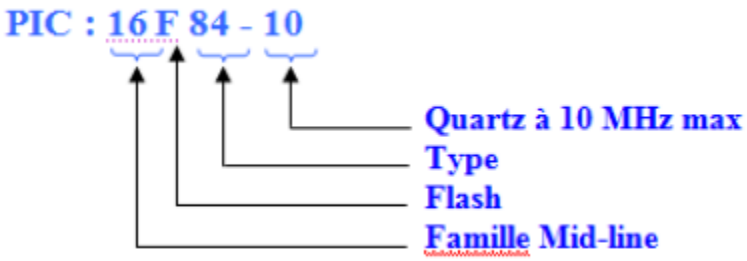
- ✓ Base-Line : ils utilisent des mots d'instruction de 12 bits.
- ✓ Mid-Range : ils utilisent des mots d'instruction de 14 bits.
- ✓ High-End : ils utilisent des mots d'instruction de 16 bits.

Il existe aussi des PIC à processeur 16 bits (PIC24F/PIC24H) et 32 bits (PIC32M) aussi.
Toutes les PICs Mid-Range ont un jeu de 35 instructions, stockent chaque instruction dans un seul mot de programme, et exécutent chaque instruction (sauf les sauts) en un cycle machine.

On atteint donc de très grandes vitesses, et les instructions sont de plus très rapidement

assimilées. L'horloge fournie au PIC est divisée par 4.C'est cette base de temps qui donne le temps d'un cycle. Si on utilise par exemple un quartz de 4MHz, on obtient donc 1000000 de cycles/seconde ; or, comme le PIC exécute pratiquement une instruction par cycle, hormis les sauts, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d'Instructions Par Seconde).
Pour identifier un PIC, on utilise simplement son appellation du type : wwlxxyyy-zz

- WW: Représente la catégorie du composant (12, 14, 16, 17, 18),
- L: Tolérance plus importante de la plage de tension.
- XX: Type de mémoire de programme:
  - ✓ C: EPROM ou EEPROM.
  - ✓ CR: PROM.
  - ✓ F: FLASH.
- YYY: Identification.
- ZZ: Vitesse maximum tolérable.



Les PICs sont des composants STATIQUES, c'est à dire que la fréquence d'horloge peut être abaissée jusqu'à l'arrêt complet sans perte de données et sans dysfonctionnement.
Ceci par opposition aux composants DYNAMIQUE, donc la fréquence d'horloge doit rester dans des limites précises.
Les microcontrôleurs PIC sont présentés en boîtier DIL (Dual In Line).Un point ou une encoche donne un repérage de la broche 1, ensuite il faut ce déplacer vers la droite pour avoir les autres broches.

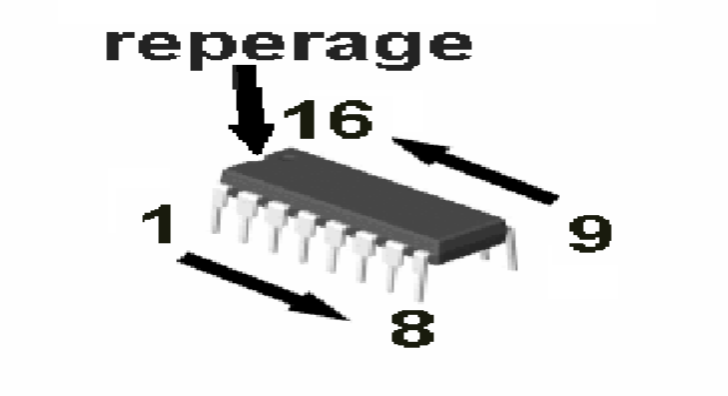


Fig. 2.2 : Repérage des broches

1.3 Microcontrôleur PIC 16F877

Dans la suite du chapitre, on va prendre comme exemple le PIC 16F877 et présenter sa structure interne et externe. Les éléments essentiels du PIC 16F877 sont :

1.3.1 Structure interne

- **Caractéristiques du CPU**

- CPU à architecture RISC (8 bits)
- Mémoire programme de 8 Kmots de 14 bits (Flash),
- Mémoire donnée de 368 Octets,
- EEPROM donnée de 256 Octets,
- 14 sources interruptions
- Générateur d'horloge de type RC ou quartz (jusqu‟ à 20 MHz)
- 05 ports d'entrée sortie
- Fonctionnement en mode sleep pour réduction de la consommation,
- Programmation par mode ICSP (In Circuit Serial Programming) 12V ou 5V,
- Possibilité aux applications utilisateur d’accéder à la mémoire programme

- **Caractéristiques des périphériques**

- Timer0 : Timer/Compteur 8 bitsavec un prédiviseur 8 bits
- Timer1 : Timer/Compteur 16 bits avec un prédivision de 1, 2, 4, ou 8 ; il peut être incrémenté en mode veille (Sleep), via une horloge externe,
- Timer2 : Timer 8 bits avec deux diviseurs (pré et post diviseur)
- Deux modules « Capture, Compare et PWM » :

Module capture 16 bits avec une résolution max. 12,5 ns,

Module Compare 16 bits avec une résolution max. 200 ns,

Module PWM avec une résolution max. 10 bits,

- Convertisseur Analogiques numériques multi-canal (8 voies) avec une conversion sur 10 bits,

Synchronous Serial Port (SSP) SSP, Port série synchrone en mode I2C (mode maitre/escalve),

- Universel Synchronous Asynchronous Receiver Transmitter (USART) : Port série universel, mode asynchrone (RS232) et mode synchrone

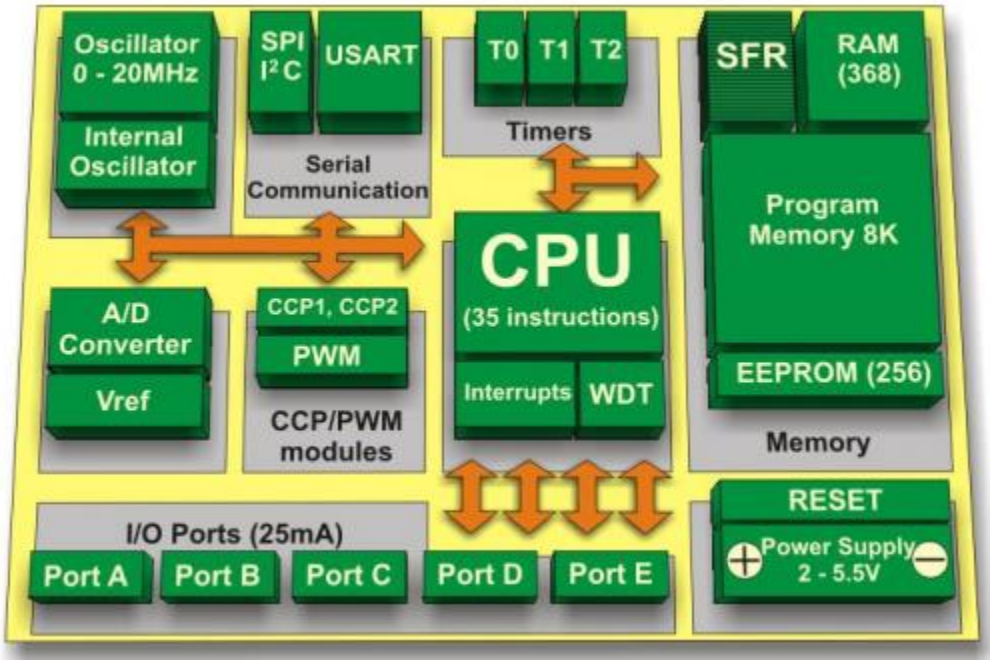


Fig. 2.3 : Architecture interne du PIC 16F877

1.3.2 Structure externe

Le PIC16F877 est un circuit intégré de 40 broches :

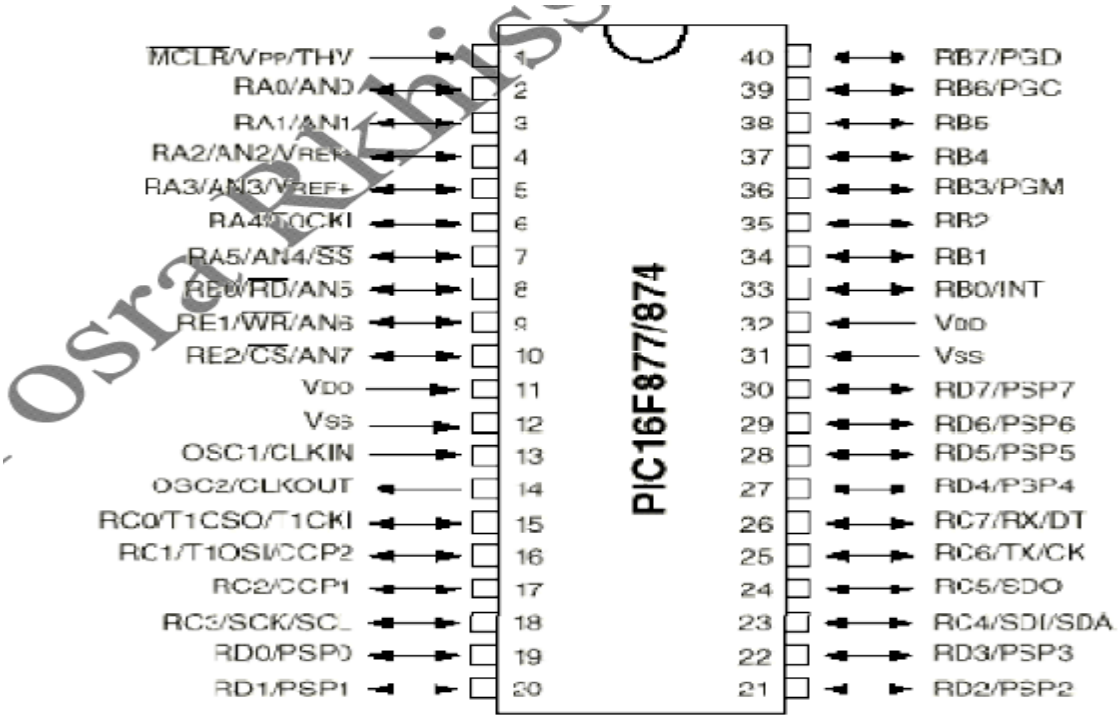


Fig. 2.4 : Brochage du PIC 16F877

Certaines pattes ont plusieurs fonctions :On dit que les fonctions sont multiplexée

1.3.2.1 L'alimentation

L'alimentation du circuit est assurée par les pattes VDD et VSS. Elles permettent à l'ensemble des composants électroniques du PIC de fonctionner. Pour cela on relie VSS (patte 5) à la masse (0 Volt) et VDD (patte 14) à la borne positive de l'alimentation qui doit délivrer une tension continue comprise entre 3 et 6 Volts.

1.3.2.2 Cadencement du PIC

Le PIC 16F877A peut fonctionner en 4 modes d'oscillateur.

- ✓ LP : Low Power crystal : quartz à faible puissance.
- ✓ XT : Crystal/Resonator : quartz/résonateur en céramique.
- ✓ HS : High Speed crystal/resonator : quartz à haute fréquence/résonateur en céramique HF.
- ✓ RC : Circuit RC (oscillateur externe).

Dans le cas du 16F877, on peut utiliser un quartz allant jusqu'à 20Mhz relié avec deux condensateurs de découplage, du fait de la fréquence importante du quartz utilisé.
Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4. Avec un quartz de 4 MHz, on obtient une horloge instruction

de 1 MHz, soit le temps pour exécuter une instruction de 1µs

1.3.2.3 Circuit Reset MCLR

La broche MCLR (Master Clear) a pour effet de provoquer la réinitialisation du microprocesseur lorsqu'elle est connectée à 0.

Lorsque le signal de "RESET" est activé, tous les registres sont initialisé et le compteur programme se place à une adresse spécifique appelée "Vecteur de RESET".

1.3.2.4 Ports d'entrées/sortie

Le PIC 16F877 dispose de 5 ports :

- ✓ Port A : 6 pins I/O numérotées de RA0 à RA5.
- ✓ Port B : 8 pins I/O numérotées de RB0 à RB7.
- ✓ Port C : 8 pins I/O numérotées de RC0 à RC7.
- ✓ Port D : 8 pins I/O numérotées de RD0 à RD7.
- ✓ Port E : 3 pins I/O numérotées de RE0 à RE2.

A chaque port correspondent deux registres :

- ✓ Un registre direction pour programmer les lignes soit en entrée, soit en sortie *TRISA, TRISB, TRISC, TRISD* et *TRISE.*
- ✓ Un registre de données pour lire ou modifier l'état des broches. *PORTA, PORTB, PORTC, PORTD* et *PORTE*

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

- ✓ Le positionnement d'un bit à « 1 » place le pin en entrée.
- ✓ Le positionnement de ce bit à « 0 » place le pin en sortie.

La plupart des broches des PORTs sont partagées avec des périphériques. En général si un périphérique est utilisé, les broches correspondantes ne peuvent pas être utilisées comme broches d'entrée/sortie.

Au reset, les lignes des ports A et E sont configurées en entrées analogiques, les autres lignes sont configurées en entrées digitales.

Le courant absorbé ou fourni peut atteindre 25 mA.

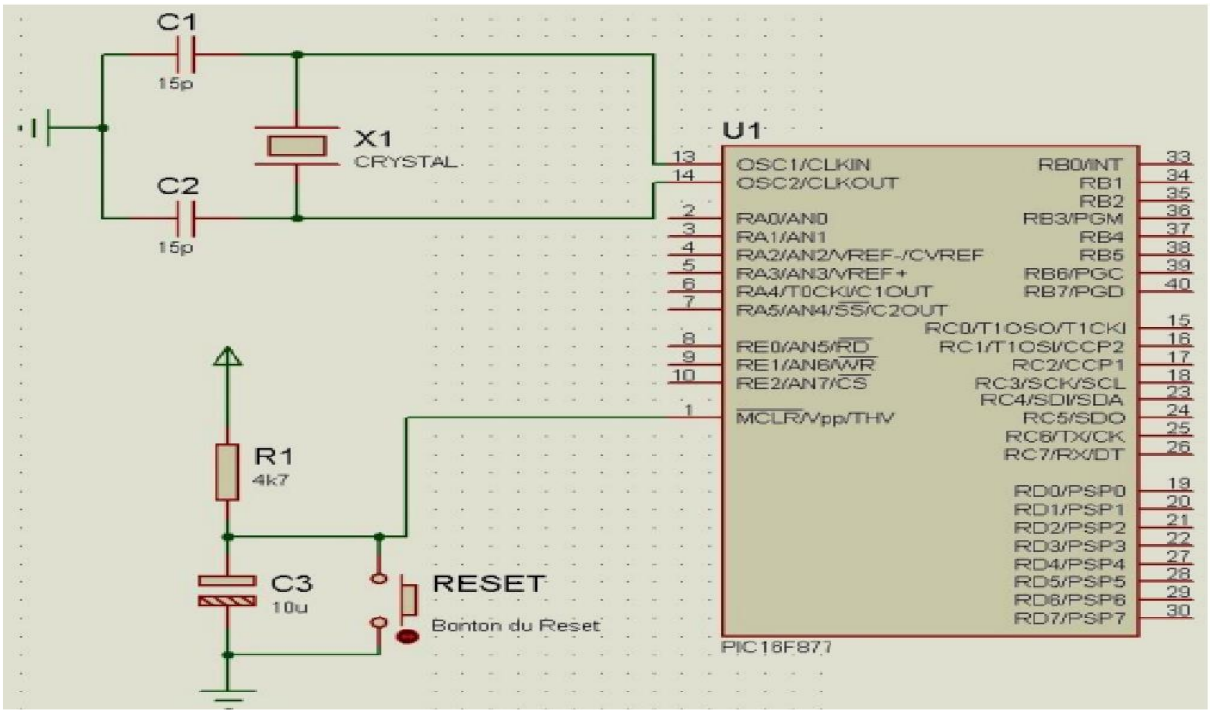


Fig. 2.5 : Circuit RESET et oscillateur d'un PIC 16F877

- Port A

Les broches port A, excepté RA4, sont multiplexées, avec les entrées du convertisseur analogique numérique (AN0 .. AN4) .

La broche RA4 est multiplexé avec l'entrée d'horloge externe du timer0 (RA4/T0CKI).

- Port B

Le port B peut être programmé pour un tirage à 5V (*pull up*) de toutes ses lignes que l'on peut mettre ou non en service en mode entrée uniquement. Elles sont automatiquement désactivées quand le port est configuré en sortie.

En mode entrée, chaque broche du PORTB doit être maintenue à un niveau haut par l'intermédiaire de résistances de 10 k pour ne pas déclencher d'interruptions imprévues.

Cette possibilité d'interruption sur un changement d'état associé à la fonction de tirage configurable sur ces 4 broches, permet l'interfaçage facile avec un clavier. Cela rend possible le réveil du PIC en mode SLEEP par un appui sur une touche du clavier.

- Port C

Le port C est partagé avec liaisons, les timers 1 et 2 et les modules CCP.

- Port D et E

En plus de leur utilisation comme PORTS E/S; les ports D et E, permettent au microcontrôleur de travailler en mode PSP (Parallel Slave Port) c'est-à-dire, qu'il peut être interfacé avec un autre

microprocesseur. Dans ce cas le PORTD représente le bus de données et le PORTE les signaux de contrôle (RD\, WR\ et CS\).
Le PORTE peut être aussi, configuré en mode analogique pour former avec le PORTA les 8 entrées du convertisseur analogique numérique. Par défaut, le PORTE est configuré comme port analogique, et donc, comme pour le PORTA,

1.3.2.5 Chien de garde

Un chien de garde est un circuit électronique ou un logiciel utilisé en électronique numérique pour s'assurer qu'un automate ou un ordinateur ne reste pas bloqué à une étape particulière du traitement qu'il effectue. C'est une protection destinée généralement à redémarrer le système, si une action définie n'est pas exécutée dans un délai imparti ;
Dans le PIC, il s'agit 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, deux situations sont possibles :
- Si le µC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester planté en cas de blocage du microcontrôleur par un processus indésirable non contrôlé
- Si le µC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode SLEEP. Cette situation est souvent exploitée pour réaliser des temporisations.

;