Chap 1

SYSTEMES A BASE DE MICROPROCESSEUR

I / Architecture d'un système à base de microprocesseur :

Un système à base de microprocesseur est formé des trois éléments :

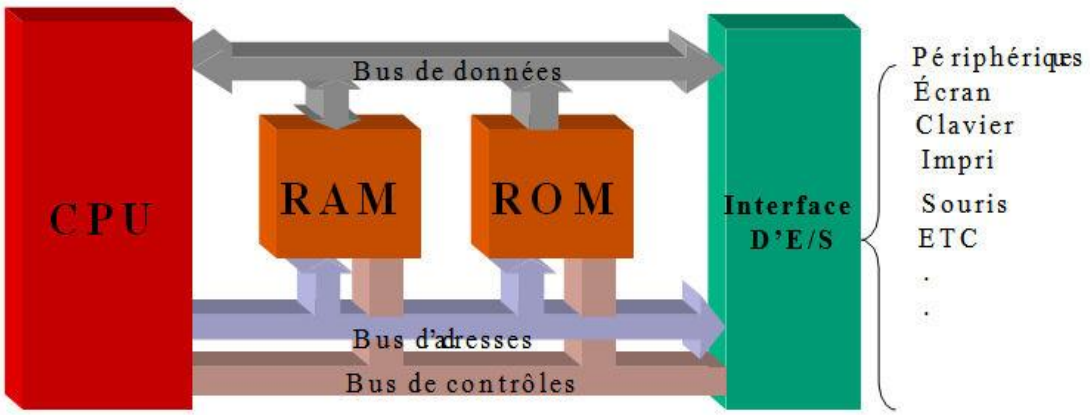
- Une unité CPU (central processing unit)

- Une mémoire (ROM et RAM)

- Des ports d'entrées/sorties.

Les trois modules sont interconnectés comme le montre la figure suivante autour de trois bus : bus de données, bus d'adresses et bus de contrôles et commandes.

Bus : Il s'agit de plusieurs pistes électroniques qui sont reliées au microprocesseur. Ces bus assurent la communication interne et externe du microprocesseur.



- Le bus de données : c'est un ensemble de fils bidirectionnels qui va permettre le transfert de données entre les différents éléments du système. Autrement dit, toutes les données entrantes et sortantes du microprocesseur sont véhiculées par le bus de données qui fixe la longueur du mot échangé avec la mémoire.

- Le bus d'adresses : il permet d'adresser un élément par le microprocesseur .il est unidirectionnel .il détermine la capacité maximale d'adressage du système, c'est à dire le nombre maximum de mots de la mémoire associée (ex : 16 bits « adressent » 64 Kmots).

- Le bus de commandes et de contrôle : c'est un bus qui permet de véhiculer les signaux de contrôles et de commandes tels que l'horloge les signaux Rd/Wr etc

-La mémoire sert au rangement de deux types d'informations :

-Des données : les informations traitée par le microprocesseur.

-Des instructions : ensemble d'informations codées qui gère l'activité du microprocesseur.

Remarque :

La mémoire morte (ROM : Read Only Memory) range en général le programme d'initialisation du système (exemple dans le PC elle range le BIOS : Basic Input Ouput systeme) .

La mémoire vive (RAM : Random Axes Memory) sert au rangement des programmes utilisateurs c'est une mémoire volatile.

- Les interfaces d'entrées sorties vont permettre au microprocesseur de communiquer avec le monde extérieur. Donc le microprocesseur peut lire des données à partir d'une interface d'entrée (exemple souris, clavier disque dur, Etc. …) de même il peut restituer le résultat de son traitement au monde extérieur en adressent des interfaces de sortie (tel que les imprimantes le clavier etc. …) donc les interfaces d'entrées / sorties vont soulager le microprocesseur pour la communication avec le monde extérieur.

Le microprocesseur doit Donc contrôler les fonctions effectuées par les autres modules, il doit chercher ainsi que décoder des instructions ranger en mémoire, et il doit adresser des interfaces d'entrées/sorties pour lire des données du monde extérieur, et restituer le résultat de son traitement.

II / Architecture d'un CPU :

Une CPU est formée par les trois éléments fonctionnels interconnectés suivants :

► Registres.

► UAL : Unité arithmétique et logique.

► Circuit de contrôle.

II-1 / Les registres :

II-1-1 / L'accumulateur :

Il s'agit d'un registre d'usage général recevant des opérandes, des résultats intermédiaires ou des résultats provenant de l'unité arithmétique et logique. Ils évitent des appels fréquents à la mémoire, réduisant ainsi les temps de calcul.

II-1-2 / Le compteur de programme :

Le compteur de programme contient l'adresse de l'instruction suivante en mémoire qui doit être exécutée. Le registre compteur de programme est constamment

modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

II-1-3 / Registre d'instruction et décodeur d'instruction :

II-1-3-1 / Le registre d'instruction :

Chaque opération que le microprocesseur va effectuer est codée « instruction code» ou « opération code » , Le registre d'instructions contient le code de la prochaine instruction à être exécutée par le processeur. Cette instruction sera acheminée (par un bus de données) au décodeur d'instructions qui sera chargé de l'interpréter.

II-1-3-2 / Le décodeur d'instruction :

C'est lui qui va interpréter l'instruction contenue dans le registre d'instruction (RI). C'est-à-dire qu'elle est l'opération à effectuer (Addition, branchement etc...) Et comment aller chercher les opérandes requises pour cette opération (par exemple, les nombres à additionner). Le décodeur d'instructions communique alors avec l'unité de commandes et de contrôles qui pourra déclencher les événements en conséquence.

II-1-4/ Registres d'adresses :

Ces registres servent à gérer l'adressage de la mémoire, et puisque les registres peuvent être incrémenter ou décrémenter donc on peut accéder facilement à des données qui se trouvent en mémoire.

II-1-5 Registre d'état (FLAGS) :

Le registre d'état FLAG sert à contenir l'état de certaines opérations effectuées par le processeur.

II-2 / L'unité arithmétique et logique (UAL) :

Comme son nom l'indique, cette unité peut exécuter deux types d'opérations.

- Opérations arithmétiques

Elles incluent l'addition et la soustraction qui sont des opérations de base (une soustraction est une addition avec le complément à deux), la multiplication et la division. Les données traitées sont considérées dans des représentations entières.

- Opérations logiques

Ces opérations sont effectuées bit à bit sur les bits de même poids de deux mots, tel que ET, OU, NOT OU EXCLUSIF, de même les opérations de rotation et de décalage (arithmétique et logique)

A la fin d'une opération, l'UAL peut aller modifier certains bits du registre d'état (FLAG).

II – 3 / Unité de contrôle et commande :

Synchronisée par le signal de l'horloge, c'est elle qui déclenche les événements dans le microprocesseur. Par exemple, quand une information passe dans un bus, cette information est destinée à un seul endroit (par exemple, un registre). C'est donc l'unité de commande et de contrôle qui va « déverrouiller » l'entrée de cette destination pour que l'information qui circule sur le bus puisse y entrer

III / Fonctionnement d'un système à base de microprocesseur :

III-1 / Les interruptions :

Les interruptions permettent au matériel (périphérique) de communiquer avec le processeur. Dans certains cas, on désire que le processeur réagisse rapidement à un évènement extérieur, mais la majorité de ces périphériques n'ont besoin du microprocesseur qu'à certains moments, alors ils génèrent une demande d'interruption.

Une interruption est signalée au processeur par un signal électrique sur une borne spéciale. Lors de la réception de ce signal, le processeur "traite" l'interruption dès la fin de l'instruction qu'il était en train d'exécuter. Le traitement de l'interruption consiste soit :

– à l'ignorer et passer normalement à l'instruction suivante : *interruptions masquables*.

– à exécuter un *traitant d'interruption* (interrupt handler). *Interruptions non masquables*.

Remarque :

Parfois le microprocesseur est sollicité par plusieurs interruptions en même temps, pour répondre à ces appels un ordre de priorité est souvent pris en compte pour leurs traitements.

Les interruptions augmentent considérablement l'efficacité du processeur.

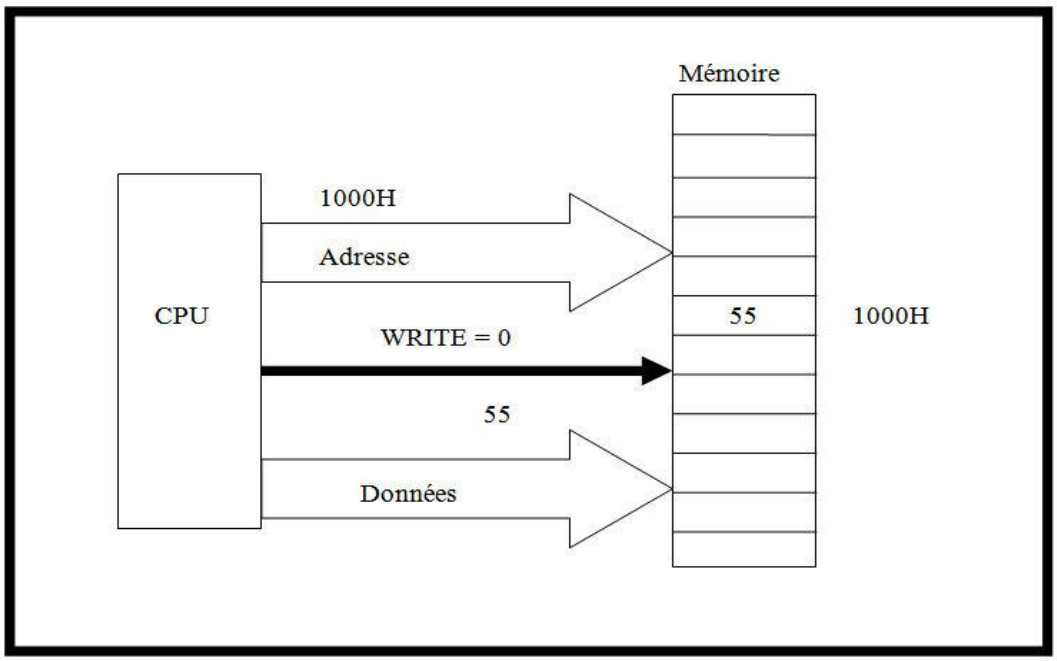
Les interruptions sont de deux types :

- Interruption matérielle.
- Interruption logicielle.

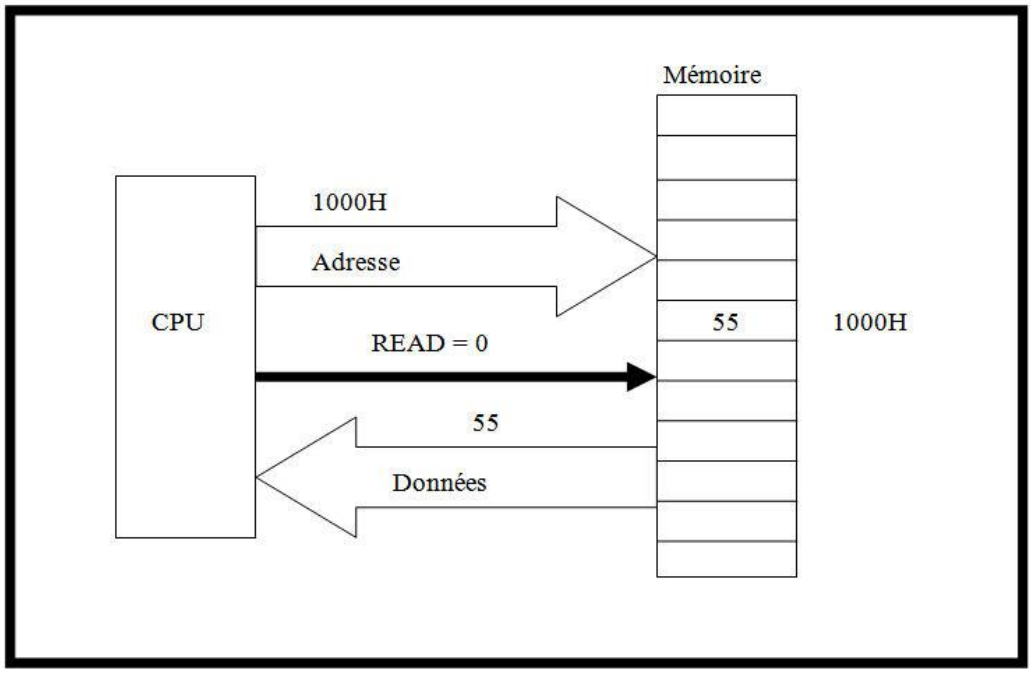
III-2 /L'écriture en mémoire (WRITE) :

Pour écrire une donnée dans la mémoire le microprocesseur doit placer l'adresse de la donnée sur le bus d'adresses, puis il place la donnée sur le bus de données et enfin génère le signal WRITE.

III-3 / La lecture de la mémoire (READ) :



Pour lire une donnée de la mémoire le microprocesseur dépose son adresse sur le bus d'adresses puis génère le signal READ, alors la donnée sera acheminée vers le microprocesseur à travers le bus de données. La donnée sera stockée dans un registre dans le microprocesseur.



Remarque :

Si la donnée est un code opératoire d'une instruction alors elle sera logée dans le registre d'instructions sinon elle sera logée dans un registre de données (en général l'accumulateur)

III- 4 /Communication avec les entrées/sorties :

Les données échangées entre un périphérique et le processeur transitent par l'interface associé à ce périphérique .L'interface possède de la mémoire tampon pour stocker les données échangées et les informations pour gérer la communication avec le périphérique :

– des *informations de command*e, pour définir le mode de fonctionnement de l'interface : sens de transfert (entrée ou sortie), mode de transfert des données (par scrutation ou interruption), etc. Ces informations de commandes sont communiquées à l'interface lors de la phase d'*initialisation* de celle-ci, avant le début du transfert.

– des *informations d'éta*t, qui mémorisent la manière dont le transfert c'est effectué (erreur de transmission, réception d'informations, etc). Ces informations sont destinées au processeur.

Lors de l'exécution des instructions d'entrées/sorties, le processeur met à 1 sa borne IO/M et présente l'adresse E/S sur le bus d'adresse. Le signal IO/M indique aux circuits de décodage d'adresses qu'il ne s'agit pas d'une adresse en mémoire principale, mais de l'adresse d'une interface d'entrées/sorties.

III -5 / Accès direct à la mémoire (DMA)

Lorsqu'un transfert en mémoire est nécessaire de la mémoire RAM à un port d'E/S, la CPU lit le premier octet en mémoire et le charge dans l'un des registres du microprocesseur. La CPU écrit ensuite l'octet rangé précédemment sur le port d'E/S approprié.

Il en résulte que le microprocesseur effectue des opérations de lecture et d'écriture répétées. Ainsi un certain temps est perdu entre le traitement de chaque octet. Pour remédier à ce problème, une procédure est mise au point pour l'accès direct à la mémoire (Direct Memory Acess), qui permet de transférer des données de la mémoire RAM au port d'E/S sans passer par le microprocesseur. Pour cela, un contrôleur DMA, qui reprend le rôle de la CPU, c'est à dire qu'il gère les transferts de la RAM aux ports d'E/S.

Chap 2

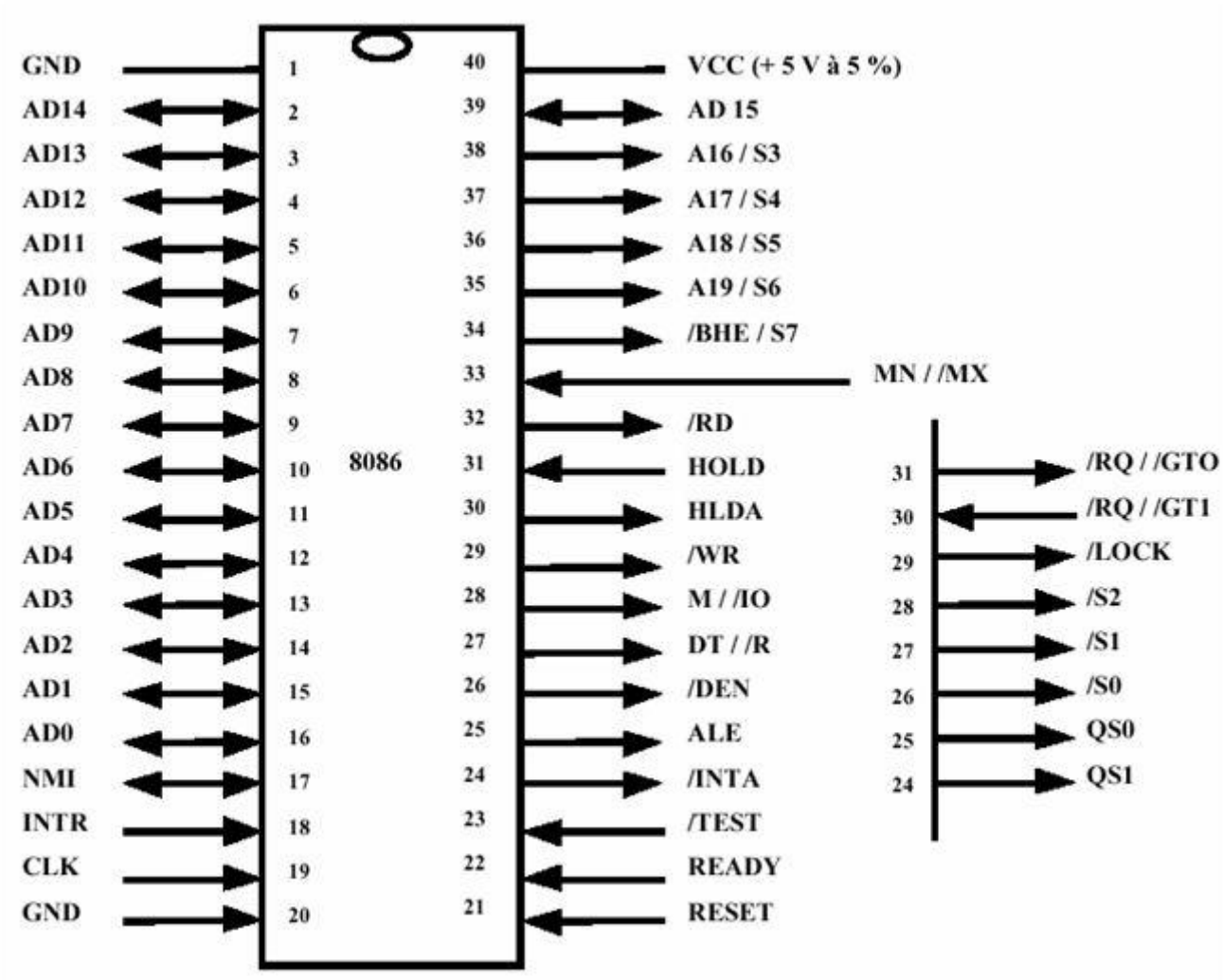
LE MICROPROCESSEUR 8086 / 8088

I / Introduction :

Le processeur 8086 d'Intel est à la base des processeurs Pentium actuels. Les processeurs successifs (de PC) se sont en effet construits petit à petit en ajoutant à chaque processeurs des instructions et des fonctionnalités supplémentaires, mais en conservant à chaque fois les spécificités du processeur précédent.

II / Architecture externe du 8086 :

Le 8086 est un circuit intégré de forme DIL de 40 pattes comme le montre la figure suivante :

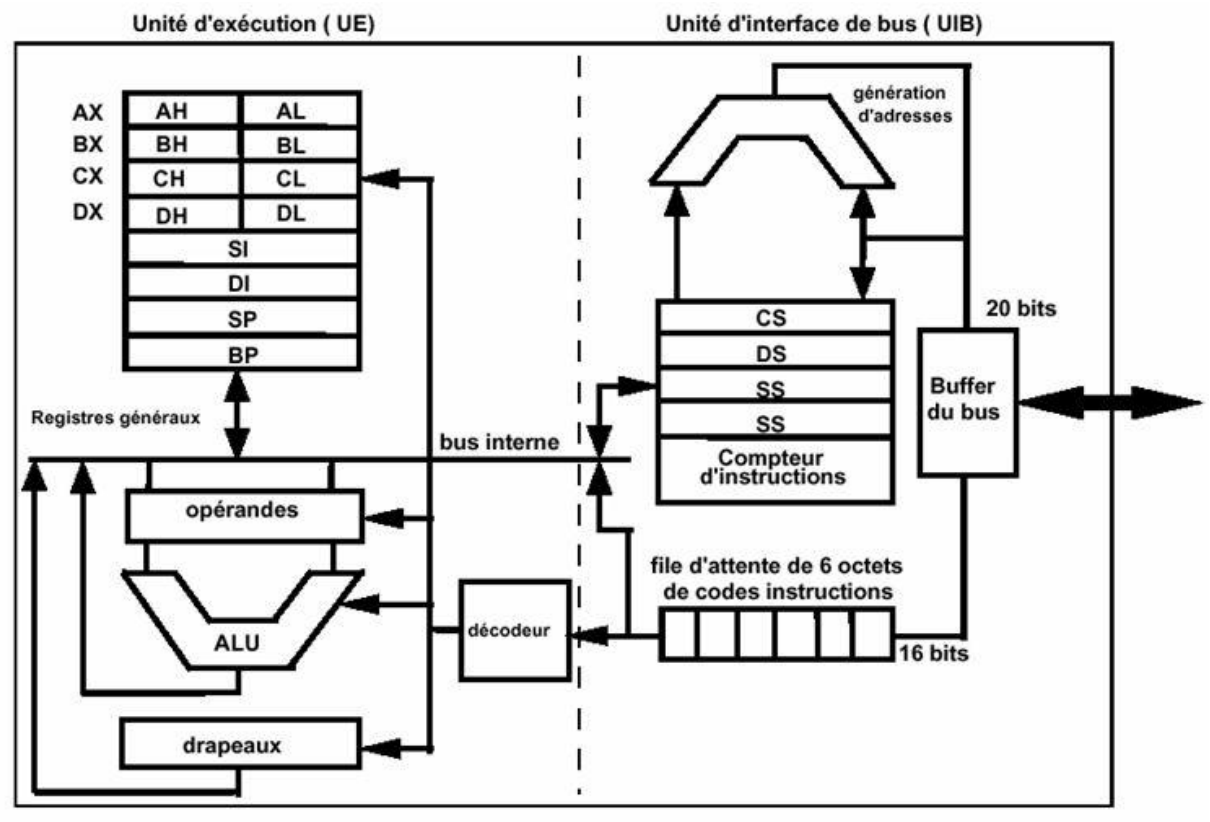


Le 8086 (développé en 1978) est le premier microprocesseur de type x86.Il est équipé d'un bus de données de 16 bits et un bus d'adresses de 20 bits et fonctionne à des fréquences diverses selon plusieurs variantes: 5, 8 ou 10 MHz.

III / Architecture interne du 8086 :

Il existe deux unités internes distinctes: l'UE (Unité d'Exécution) et l'UIB (Unité d'Interfaçage avec le Bus). Le rôle de l'UIB est de récupérer et stocker les informations à traiter, et d'établir les transmissions avec les bus du système. L'UE exécute les instructions qui lui sont transmises par l'UIB.

Pendant que l'UE exécute les informations qui lui sont transmises, l'instruction suivante est chargée dans l'UIB. Les instructions qui suivront sont placées dans une file d'attente. Lorsque l'UE a fini de traiter une instruction l'UIB lui transmet instantanément l'instruction suivante, et charge la troisième instruction en vue de la transmettre à l'UE. De cette façon, l'UE est continuellement en activité.



Donc en conclusion on peut dire que le 8086/8088 se compose essentiellement de deux unités : la BIU qui fournit l'interface physique entre le microprocesseur et le monde extérieur et l'EU qui comporte essentiellement l'UAL de 16 bits qui manipule les registre généraux de 16 bits aussi .

Remarque :

La file d'attente d'instructions contient des informations qui attendent d'être traitées par l'UE. elle est parfois appelée *capacité de traitement.* Le 8086 est capable de mémoriser jusqu'à six octets.

IV / Les registres du 8086/8088 :

IV-1 / Introduction :

Un registre est une petite partie de mémoire intégrée au microprocesseur, dans le but de recevoir des informations spécifiques, notamment des adresses et des données stockées durant l'exécution d'un programme. Il existe plusieurs types de registres. Certains d'entre eux sont affectés à des opérations d'ordre général et sont accessibles au programmeur à tout moment. Nous disons alors qu'il s'agit de registres généraux. D'autres registres ont des rôles bien plus spécifiques et ne peuvent pas servir à un usage non spécialisé.

IV-2 / Les registres généraux :

Les registres généraux peuvent être utilisés dans toutes les opérations arithmétiques et logiques. Chaque registre est une grandeur de 16 bits mais en réalité divisé en deux registres distincts de 8 bits.

Le programmeur dispose de 8 registres internes de 16 bits qu'on peut diviser en deux groupes :

- *registres de données* : formé par 4 registres de 16 bits (AX,BX,CX,et DX) chaque registre peut être divisé en deux registres de 8 bits (AH,AL,BH,BL,CH,CL,DH et DL)

- *groupe de pointeur et indexe* : formé de 4 registres de 16 bits (SI, DI, SP, BP) et font généralement référence à un emplacement en mémoire.

Registres de données :

	15 — 8	7 — 0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Registres de pointeur et indexe :

	15 — 0
Stack pointer	SP
Base pointer	BP
Source index	SI
Destination index	DI

IV – 2 – 1 / registres de données :

Registre AX : (Accumulateur)

Toutes les opérations de transferts de données avec les entrées – sorties ainsi que le traitement des chaînes de caractères se font dans ce registre, de même les opérations arithmétiques et logiques.

Les conversions en BCD du résultat d'une opération arithmétique (addition, soustraction, multiplication et la division) se font dans ce registre.

Registre BX : (registre de base)

Il est utilisé pour l'adressage de données dans une zone mémoire différente de la zone code : en général il contient une adresse de décalage par rapport à une adresse de référence.). (Par exemple, l'adresse de début d'un tableau). De plus il peut servir pour la conversion d'un code à un autre.

Registre CX : (Le compteur)

Lors de l'exécution d'une boucle on a souvent recours à un compteur de boucles pour compter le nombre d'itérations, le registre CX a été fait pour servir comme compteur lors des instructions de boucle.

Remarque :

Le registre CL sert en tant que compteur pour les opérations de décalage et de rotation, dans ce cas il va compter le nombre de décalages (rotation) de bits à droite ou à gauche.

Registre DX :

On utilise le registre DX pour les opérations de multiplication et de division mais surtout pour contenir le numéro d'un port d'entrée/sortie pour adresser les interfaces d'E/S.

IV – 2 – 2 / Groupe de pointeur et indexe :

Ces registres sont plus spécialement adaptés au traitement des éléments dans la mémoire. Ils sont en général munis de propriétés d'incrémentation et de décrémentation.

Un cas particulier de pointeur est le pointeur de pile (Stack Pointer : SP). Ce registre permet de pointer la pile pour stocker des données ou des adresses selon le principe du "Dernier Entré Premier Sorti" ou "LIFO" (Last In First Out).

L'idexe SI : (source indexe) :

Il permet de pointer la mémoire il forme en général un décalage (un offset) par rapport à une base fixe (le registre DS), il sert aussi pour les instructions de chaîne de caractères, en effet il pointe sur le caractère source

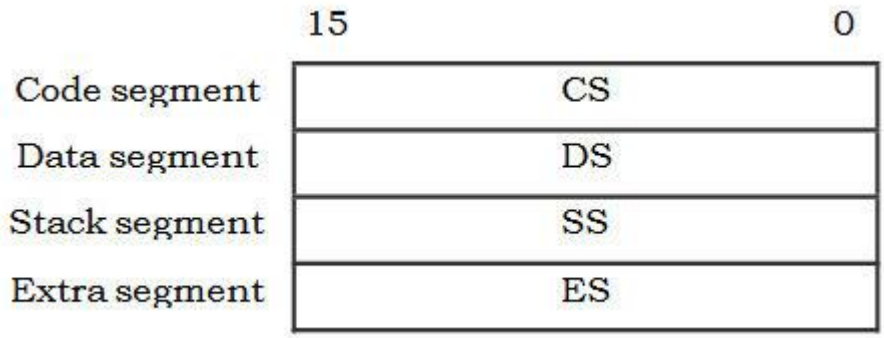
L'indexe DI : (Destination indexe) :

Il permet aussi de pointer la mémoire il présente un décalage par rapport à une base fixe (DS ou ES), il sert aussi pour les instructions de chaîne de caractères, il pointe alors sur la destination

Les pointeurs SP et BP : (Stack pointer et base pointer)

Ils pointent sur la zone pile (une zone mémoire qui stocke l'information avec le principe filo), ils présentent un décalage par rapport à la base (le registre SS). Pour le registre BP il a un rôle proche de celui de BX, mais il est généralement utilisé avec le segment de pile.

IV –2- 3 / Les registres segment:



Le 8086 a quatre registres segments de 16 bits chacun : CS (code segment, DS (Data segment), ES (Extra segment) et SS (stack segment), ces registres sont chargés de sélectionner les différents segments de la mémoire en pointant sur le début de chacun d'entre eux. Chaque segment de mémoire ne peut excéder les 65535 octets.

Le registre CS (code segment) :

Il pointe sur le segment qui contient les codes des instructions du programme en cours.

Remarque :

Si la taille du programme dépasse les 65535 octets alors on peut diviser le code sur plusieurs segments (chacun ne dépasse pas les 65535 octets) et pour basculer d'une partie à une autre du programme il suffit de changer la valeur du registre CS et

de cette manière on résout le problème des programmes qui ont une taille supérieure à 65535 octets.

Le registre DS (Data segment) :

Le registre segment de données pointe sur le segment des variables globales du programme, bien évidemment la taille ne peut excéder 65535 octets (si on a des données qui dépassent cette limite, on utilise la même astuce citée dans la remarque précédente mais dans ce cas on change la valeur de DS).

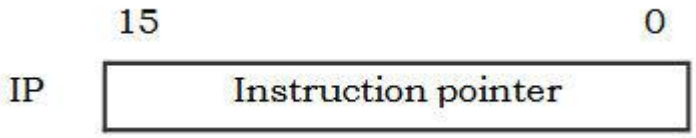
Le registre ES (Extra segment) :

Le registre de données supplémentaires ES est utilisé par le microprocesseur lorsque l'accès aux autres registres est devenu difficile ou impossible pour modifier des données, de même ce segment est utilisé pour le stockage des chaînes de caractères.

Le segment SS (Stack segment) :

Le registre SS pointe sur la pile : la pile est une zone mémoire ou on peut sauvegarder les registres ou les adresses ou les données pour les récupérer après l'exécution d'un sous programme ou l'exécution d'un programme d'interruption, en général il est conseillée de ne pas changer le contenu de ce registre car on risque de perdre des informations très importantes (exemple les passages d'arguments entre le programme principal et le sous programme)

IV-2-4 / Le registre IP : (Le compteur de programme) :



Instruction Pointer ou Compteur de Programme, contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Autrement dit, il doit indiquer au processeur la prochaine instruction à exécuter. Le registre IP est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

II-2-5 : Le registre d'état (Flag) :

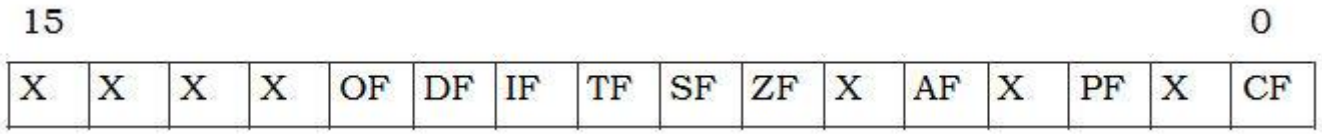
Le registre d'état FLAG sert à contenir l'état de certaines opérations effectuées par le processeur. Par exemple, quand le résultat d'une opération est trop grand pour être

contenu dans le registre cible (celui qui doit contenir le résultat de l'opération), un bit spécifique du registre d'état (le bit OF) est mis à 1 pour indiquer le débordement.

Remarque : Drapeaux (flags)

Les drapeaux sont des indicateurs qui annoncent une condition particulière suite à une opération arithmétique ou logique.

Le registre d'état du 8086 est formé par les bits suivants :



Remarque :

X : bit non utilise.

CF (*Carry Flag*) :

Retenue : cet indicateur et mis à 1 lorsque il y a une retenue du résultat à 8 ou 16 bits. Il intervient dans les opérations d'additions (retenue) et de soustractions (borrow) sur des entiers naturels. Il est positionné en particulier par les instructions ADD, SUB et CMP (Comparaison entre deux valeurs).

CF = 1 s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes. Exemples (sur 8 bits pour simplifier) :

```
    10010110          11011001
+   01010100      +   01010010
CF=0 11101010     CF=1 00101011
```

PF (*Parity Flag*) :

Parité : si le résultat de l'opération contient un nombre pair de 1 cet indicateur est mis à 1, sinon zéro.

AF (*Auxiliary Carry*) :

Demie retenue : Ce bit est égal à 1 si on a une retenue du quarter de poids faible dans le quarter de poids plus fort.

ZF (*Zero Flag*) :

Zéro : Cet indicateur est mis à 1 quand le résultat d'une opération est égal à zéro. Lorsque l'on vient d'effectuer une soustraction (ou une comparaison), ZF=1 indique que les deux opérandes étaient égaux. Sinon, ZF est positionné à 0.

SF (*Sign Flag*) :

SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1 ; sinon SF=0. SF est utile lorsque l'on manipule des entiers signés, car le bit de poids fort donne alors le signe du résultat. Exemples (sur 8 bits) :

```
    10010110          11011001
+   01010100      +   01010010
SF=1 11101010     SF=0 00101011
```

OF (*Overflow Flag*) :

Débordement : si on a un débordement arithmétique ce bit est mis à 1.c a d le résultat d'une opération excède la capacité de l'opérande (registre ou case mémoire), sinon il est à 0.

DF (*Direction Flag*) :

Auto Incrémentation/Décrémentation : utilisée pendant les instructions de chaîne de caractères pour auto incrémenté ou auto décrémenter le SI et le DI.

IF(*Interrupt Flag*) :

Masque d'interruption : pour masquer les interruptions venant de l'extérieur ce bit est mis à 0, dans le cas contraire le microprocesseur reconnaît l'interruption de l'extérieur.

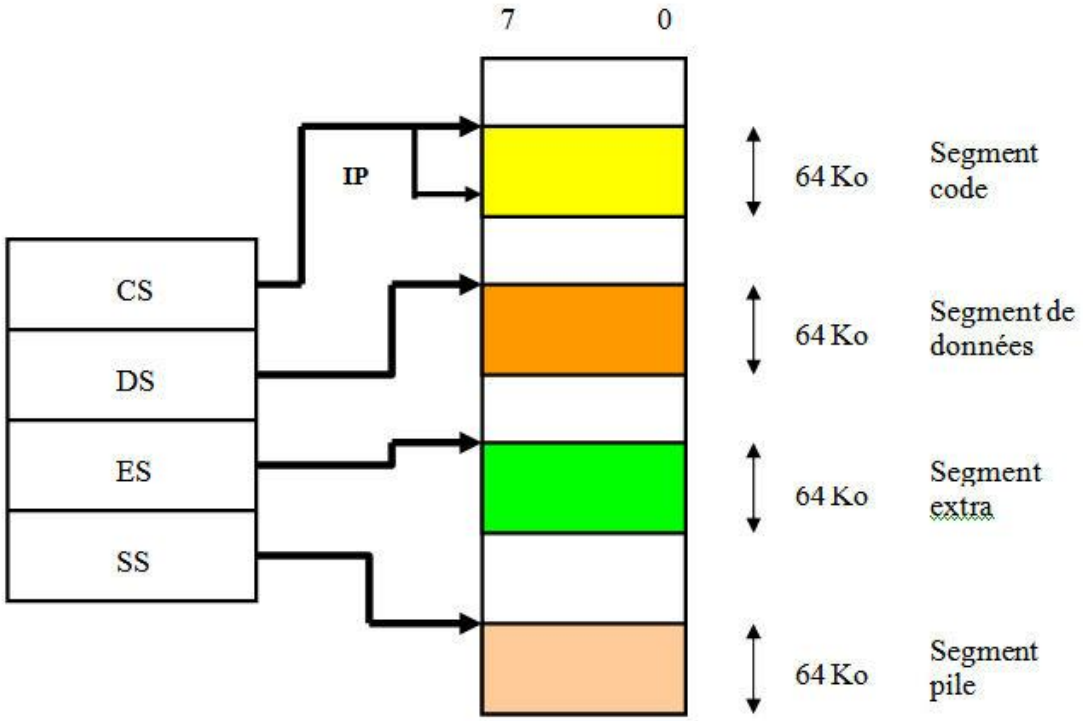
TF (*Trap Flag*) :

Piége : pour que le microprocesseur exécute le programme en mode pas à pas.

V / Gestion de la mémoire :

V – 1 / Introduction :

L'espace mémoire adressable (1 méga = 2^{20} bits du bus d'adresse) du 8086/8088 est divisé en quatre segment logiques allant jusqu'à 64 KOctets chacun. L'accès à ces espaces est direct et simultané, or Le compteur de programme est de 16 bits donc la possibilité d'adressage est de 2^{16} = 64 Ko (Ce qui ne couvre pas la totalité de la mémoire), alors on utilise deux registres pour indiquer une adresse au processeur, Chaque segment débute à l'endroit spécifié par le registre segment. Le déplacement (offset) à l'intérieur de chaque segment se fait par un registre de décalage qui permet de trouver une information à l'intérieur du segment. Exemple la paire de registre CS:IP : pointe sur le code d'une instruction (CS registre segment et IP Déplacement)



V – 2 / Adresse physique (Segmentation de la mémoire) :

Les données sont regroupées dans une zone mémoire nommée *segment de donnée*s, tandis que les instructions sont placées dans un *segment d'instruction*s (de même pour le segment pile et segment de données supplémentaires). Ce partage se fonde sur la notion plus générale de *segment de mémoir*e, qui est à la base du mécanisme de gestion des adresses par les processeurs 80x86. On a vu aussi que le registre IP, qui stocke l'adresse d'une instruction, fait lui aussi 16 bits. Or, avec 16 bits il n'est possible d'adresser que 2^{16} = 64 Kilo octets. Le bus d'adresses du 8086 possède 20 bits. Cette adresse de 20 bits est formée par la juxtaposition d'un registre segment (16 bits de poids fort) et d'un déplacement *(offse*t, 16 bits de poids faible). Adresse physique= Base * 16+ offset

Le schéma de la figure suivante illustre la formation d'une adresse 20 bits à partir du segment et du déplacement sur 16 bits :

Bits	20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3 2 1 0
	Base	0 0 0 0
+	0 0 0 0	Offset
=	Adresse physique	

Remarque :

On appellera *segment de mémoire* une zone mémoire adressable avec une valeur fixée du segment (les 16 bit de poids fort). Un segment a donc une taille maximale de 64 Ko.

Exemple :

Pour CS=1000 et IP = 2006

Adresse physique =

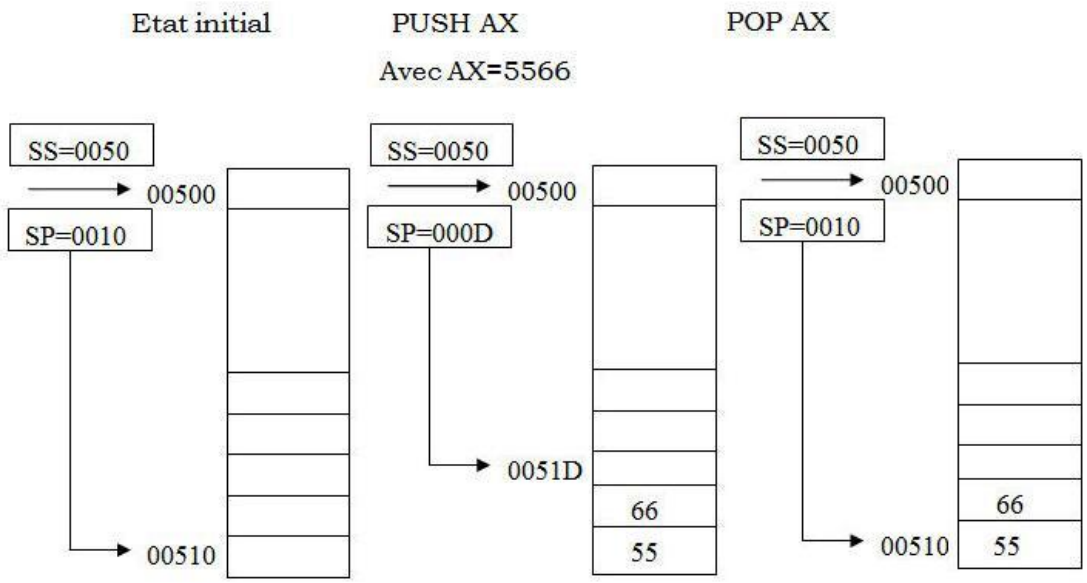
```
   10000
+  2006
=  12006
```

V-3 / Implémentation de la pile :

Le pointeur de pile (en combinaison avec le segment de pile SS) pointe vers le dessus de la pile (TOS : top of stack) en mémoire. Une pile est un ensemble de données placées en mémoire de manière à ce que seulement la donnée du "dessus" soit disponible à un instant donné. Quand un processeur exécute une instruction, il est possible qu'il soit interrompu par une "Interruption" (c'est-à-dire un appel au processeur qui est prioritaire aux instructions du programme qu'il traite). Il doit alors arrêter de s'occuper de l'instruction qu'il traite présentement pour s'occuper de l'interruption. Quand l'interruption sera traitée, il retournera à l'instruction qu'il traitait quand il a été interrompu. Mais pour cela, il doit se rappeler de cette instruction ainsi que de l'état de certains registres au moment où il traitait l'instruction. Donc pour ne pas les perdre, il les placera temporairement dans une pile (à l'intérieur de la mémoire RAM par exemple) et pourra les récupérer une fois l'interruption traitée. Le pointeur de pile (SP) donne donc l'adresse en mémoire de cette pile temporaire.

Les *piles* offrent un nouveau moyen d'accéder à des données en mémoire principale, qui est très utilisé pour stocker temporairement des valeurs.

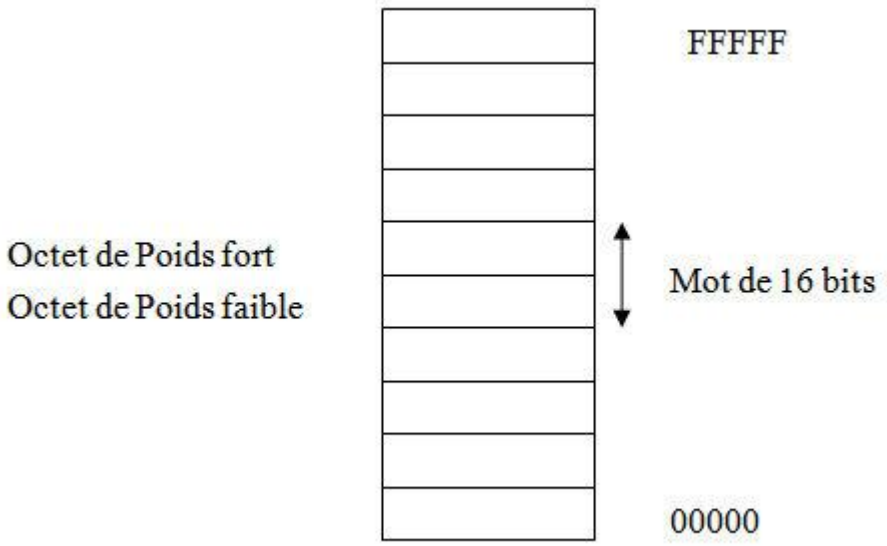
Le schéma suivant montre comment une valeur est stocker dans la pile (pushed) et comment elle est récupérée (poped) :



V – 4 / Organisation de la mémoire :

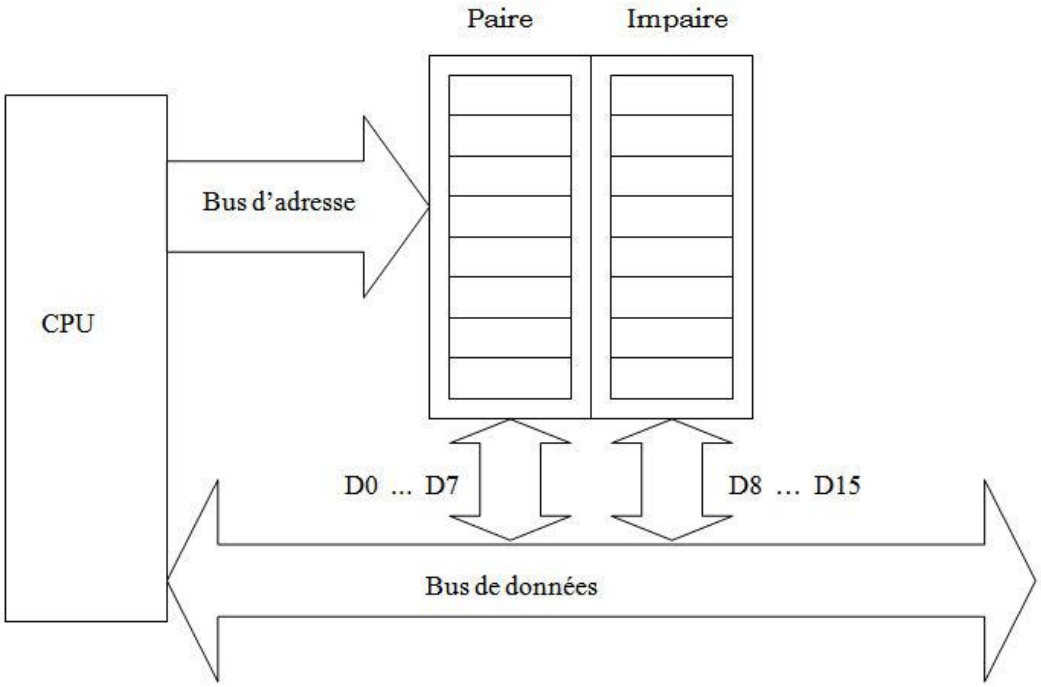
V – 4 – 1 / Organisation logique :

Logiquement la mémoire est organisée de cette manière :



V – 4 – 2 / Organisation physique :

Le microprocesseur 8086 est un processeur 16 bits (bus de données de 16 bits), ce qui lui donne la possibilité d'accéder en même temps à deux cases mémoires de 8 bits. En effet pour le 8086 la mémoire est organisée en deux Banks (un bank pair et un bank impair chacun de 512 Koctet) comme le montre la figure suivante :



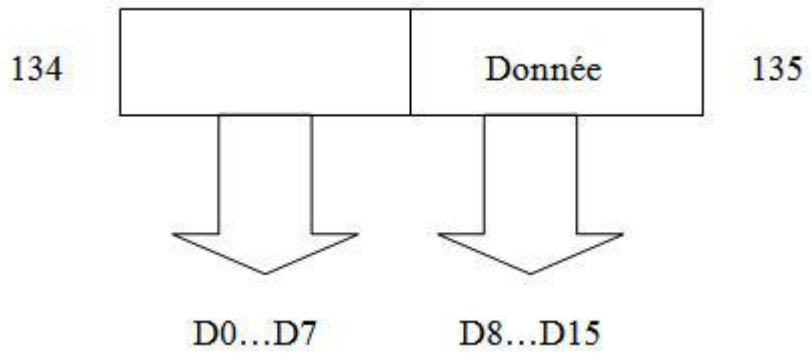
Les bits D0..D7 sont appelé partie base alors que les bits D8..D15 sont appelé partie haute. Le 8086 peut charger un octet (8 bits) ou un mot (16 bits) ou un double mot (32 bits) de la mémoire, en effet pour l'octet il suffit de donner l'adresse de ce dernier pour être chargé dans la CPU, pour le mot il suffit de donner l'adresse le 8086 cherche l'octet du poids faible à l'adresse donnée et l'octet du poids le plus fort à l'adresse qui suit , mais un problème apparaît lorsque on veut accéder à une case mémoire impaire tel que 135 par exemple , en effet :

La figure suivante montre comment les cases sont rangées dans les deux banks :

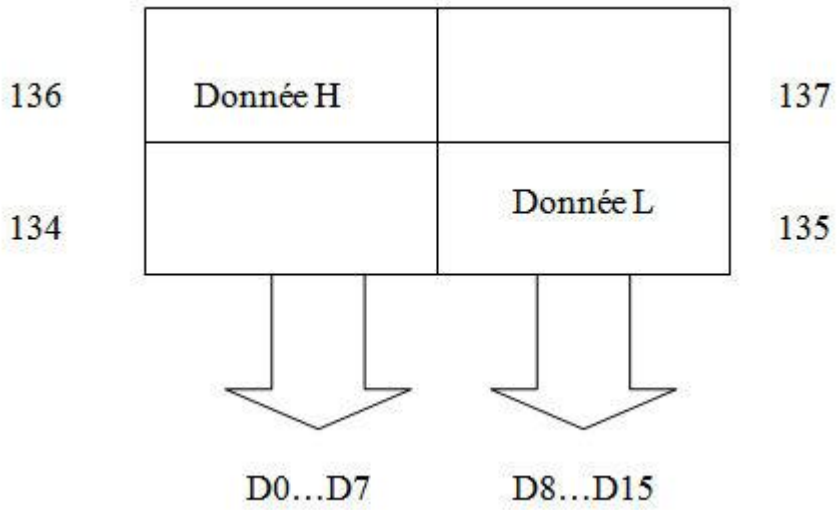
Paire	Impaire
6	7
4	5
2	3
0	1

- Si on veut accéder par exemple à l'octet (8 bits) d'une adresse paire celle ci sera directement transmise sur les lignes D0…D7 mais si on veut accéder à une adresse impaire tel que 135 par exemple, donc il faut aussi que la donnée serait charger sur

les lignes D0.. D7, or en réalité et en regardant l'organisation de la mémoire par la figure précédente on constate que la donnée sera transmise sur les lignes D8..D15 (adresse impaire) ce qui va obliger le microprocesseur de changer cette octet du poids haut au poids faible d'une manière automatique.



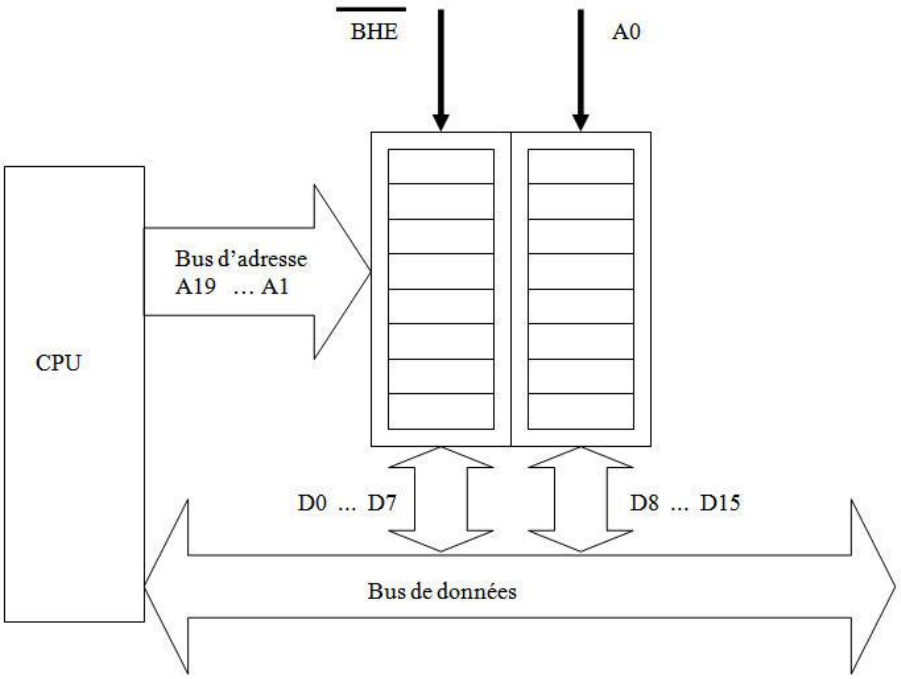
- Si on veut accéder à un mot : si l'adresse est paire on n'aura pas de problème le pois faible sera chargé sur les lignes D0…D7 et le pois fort sera chargé sur les lignes D8…D15 donc l'accès à la mémoire se fait avec un seul cycle .mais si on veut accéder à un mot logé dans une case impaire tel que 135 par exemple alors il nous faut deux cycles pour charger la donnée en effet l'organisation de la mémoire nous donne la disposition suivante :



Donc le microprocesseur doit accéder à la mémoire en deux temps le premier pour chercher l'octet 135 et le deuxième pour chercher l'octet haut à partir de l'adresse 136.de plus il doit permuter ces deux octet pour avoir le poids faible sur les lignes D0…D7 et le poids fort sur les lignes D8…D15.

Remarque :

Pour sélectionner les banks pair et impair le microprocesseur utilise deux signaux (BHE et A0 : le premier bit du bus d'adresse) comme le montre la figure suivante :



Pour sélectionner le bank pair A0=0

Pour sélectionner le bank impair BHE = 0

Chap 3

PROGRAMMATION du 80806

I / Introduction :

Lorsque l'on doit lire ou écrire un programme en langage machine, il est difficile d'utiliser la notation hexadécimale. On écrit les programmes à l'aide des instructions en mnémonique comme MOV, ADD, etc. Les concepteurs de processeurs, comme Intel, fournissent toujours une documentation avec les codes des instructions de leurs processeurs, et les symboles correspondantes.

L'opération d'assemblage traduit chaque instruction du programme source en une instruction machine.

Remarque 3 :

L'assembleur est utilisé pour être plus prés de la machine, pour savoir exactement les instructions générées (pour contrôler ou optimiser une opération) On retrouve l'assembleur dans :

- la programmation des systèmes de base des machines (le pilotage du clavier, de l'écran, etc...),

- certaines parties du système d'exploitation,

- le pilotage de nouveaux périphériques (imprimantes, scanners, etc..

- l'accès aux ressources du système,

L'avantage donc de l'assembleur est de générer des programmes efficaces et rapides (à l'exécution) par contre ses inconvénients : développement et mise au point long.

II / Le fichier source (code source) :

Comme tout programme, un programme écrit en assembleur (programme source) comprend des définitions, des données et des instructions, qui s'écrivent chacune sur une ligne de texte.

Les données sont déclarées par des *directive*s, mots clefs spéciaux que comprend l'assembleur (donc ils sont destinés à l'assembleur. Les instructions (sont destinées au microprocesseur)

II – 1 / Les instructions :

La syntaxe des instruction est comme suit :

{Label :} Mnémonique {opérande} { ; commentaire}

- Le champ opérande est un champ optionnel selon l'instruction (parfois l'instruction nécessite une opérande et parfois non).

- Le champ commentaire : champ sans signification syntaxique et sémantique pour l'assembleur , il est optionnel mais très intéressant lorsque on programme en assembleur, en effet les instructions en assembleur sont des instructions élémentaires donc dans un programme le nombre d'instructions et assez élevé (par exemple pour utiliser des fonctions tels que COS ou SIN il faut réaliser ça en utilisant des opérations arithmétiques et logiques de base) donc contrairement au langage évolué de programmation dans les programmes source on va trouver plus d'instructions. Ce qui va rendre la compréhension des programmes assez délicate et difficile. Pour cette raison lorsqu'on programme en assembleur il vaut mieux mettre des commentaires pour que le programme soit lisible pour les utilisateurs.

- Le champ Label (étiquette) est destiné pour marquer une ligne qui sera la cible d'une instruction de saut ou de branchement. Une label peut être formée par 31 caractère alphanumérique ({A.. Z} {a.. z} {0.. 9} {.?@_$}) au maximum .Les noms des registres ainsi que la représentation mnémonique des instructions et les directives ne peuvent être utilisées comme Label. Le champ Label doit se terminer par ' : ' et ne peut commencer par un chiffre. De même il n'y a pas de distinction entre minuscules et majuscules.

Exemple :

```
ET1 : MOV  AX , 500H   ; mettre la valeur 500 dans le registre AX
```

II-2 / Les directives :

Une directive est une information que le programmeur fournit au compilateur. *Elle n'est pas transformée en une instruction en langage machine. Les directives* sont des déclarations qui vont guider l'assembleur.

Une directive est utilisée par exemple pour créer de l'espace mémoire pour des variables, pour définir des constantes, etc…

Pour déclarer une directive il faut utiliser la syntaxe suivante :

{Nom} Directive {opérande} { ; commentaire}

- Le champ opérande dépend de la directive
- Le champ commentaire a les mêmes propriétés vues dans le paragraphe précèdent.
- Le champ Nom indique le nom des variables : c'est un champ optionnel (selon la directive).

II – 3 /Exemple de directives :

III- 3 – 1 / Les directives de données :

III – 3- 1 – 1 / EQU :

Syntaxe : Nom EQU Expression

Exemples :

```
VAL  EQU   50    ; assigne la valeur 50 au nom VAL
ET1   EQU   VAL* 5 + 1 ; assigne une expression calculer a VAL
```

III – 3- 1 – 2 / DB/DW/DD/DF/DP/DQ/DT:

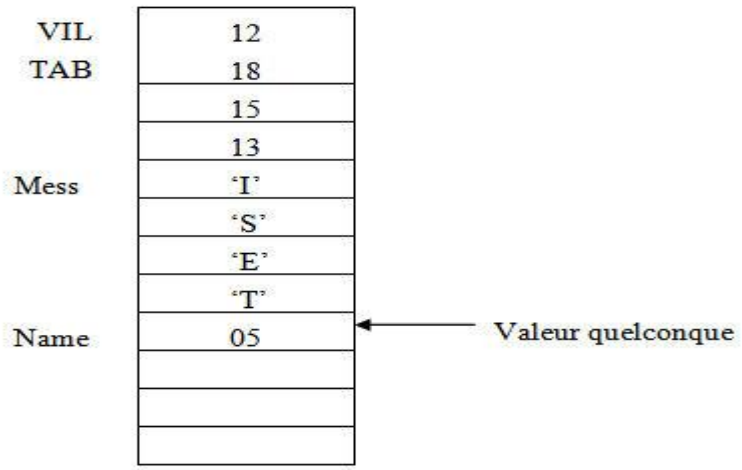
Ces directives sont utilisées pour déclarer les variables : L'assembleur attribue à chaque variable une adresse. Dans le programme, on repère les variables grâce à leurs noms. Les noms des variables sont composés d'une suite de 31 caractères au maximum, commençant obligatoirement par une lettre. Le nom peut comporter des majuscules, des minuscules, des chiffres, plus les caractères @, et _. Lors de la déclaration d'une variable, on peut lui affecter une valeur initiale.

a° / **DB (Define byte):** définit une variable de 8 bits : c a d elle réserve un espace mémoire d'un octet : donc les valeurs qu'on peut stocker pour cette directive sont comprises entre 0 et 255 (pour les nombres non signés) et de -128 jusqu'à 127 pour les nombres signés .

Syntaxe : Nom DB Expression

Exemple :

```
Vil  DB  12H ; Définit une variable (un octet) de valeur Initiale 12.
Tab  DB 18H, 15H, 13H ; définit un tableau de 3 cases
                      ;(3 octet) Qui démarre à partir de l'adresse TAB.
Mess DB 'ISET' ; définit aussi un tableau mais les valeurs de chaque case
               ; n'est autre que le code ascii de chaque lettre.
Name DB  ?  ; définit une variable 8 bits de valeur initiale quelconque.
```

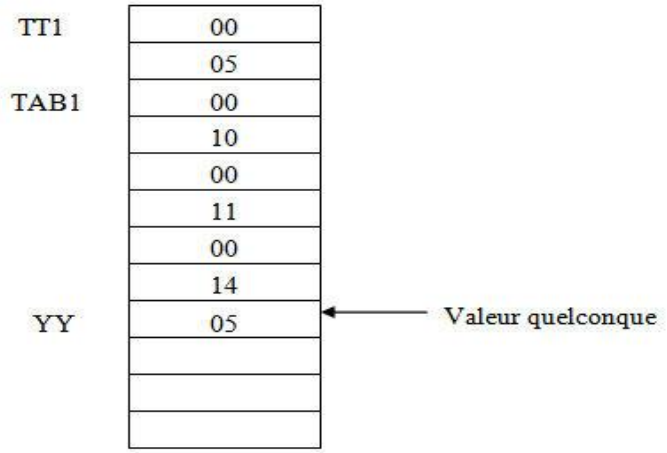


b° / DW (define word) : définit une variable de 16 bits : c a d elle réserve un espace mémoire d'un mot : donc les valeurs qu'on peut stocker pour cette directive sont comprises entre 0 et 65535 (pour les nombres non signés) et de -32768 jusqu'à 32767 pour les nombres signés .

Syntaxe : Nom DW Expression

Exemple :

```
TT1  DW     500H ; réserve deux cases mémoire (un mot) a partir de
l'adresse TT1.
TAB1 DW    10H,11H,14H ; réserve u  tableau de 6 cases chaque valeur sera
mise sur deux cases
YY     DW    ?        ; réserve un mot dans la mémoire de valeur initial
quelconque.
```



c° / **DD : (Define Double)** : réserve un espace mémoire de 32 bits (4 cases mémoire ou 4 octets) :

Syntaxe : nom DD expression

Exemple :

```
 ff DD   15500000H
```

e°/ Directive **dup**

Lorsque l'on veut déclarer un tableau de n cases, toutes initialisées à la même valeur, on utilise la directive *du*p:

```
tab DB 100 dup (15) ; 100 octets valant 15
y DW 10 dup (?) ; 10 mots de 16 bits non initialises
```

f°/ Les direcitive **Word PTR et Byte PTR** :

Dans certains cas, l'adressage indirect est ambigu. Par exemple, si l'on

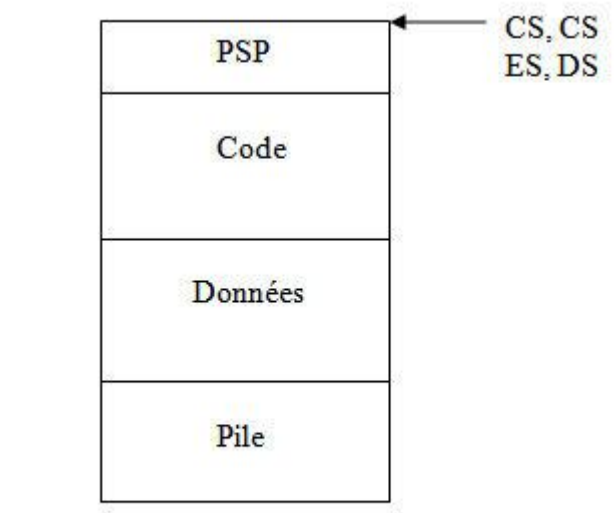
écrit :

```
MOV [BX], 0 ;          range 0 a l'adresse spécifiée par BX
```

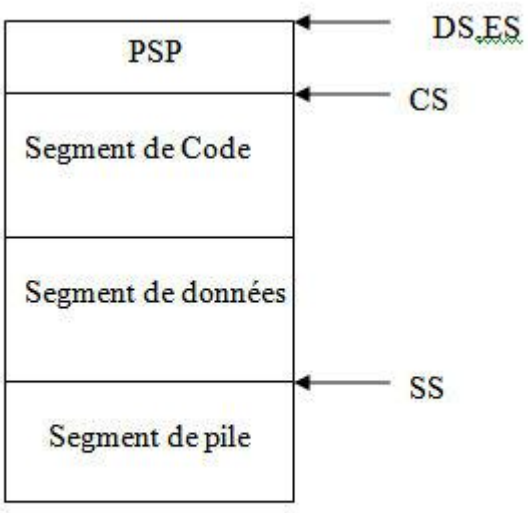
L'assembleur ne sait pas si l'instruction concerne 1, 2 ou 4 octets Consécutifs. Afin de lever l'ambiguïté, on doit utiliser une directive spécifiant la taille de la donnée à transférer :

```
MOV byte ptr [BX], val ; concerne 1 octet
MOV word ptr [BX], val ; concerne 1 mot de 2 octets
```

IV / Structure d'un programme source :



Structure d'un programme .COM



Structure d'un programme .EXE

Les directives de procédure :

Déclaration d'une procédure

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

```
Calcul     PROC near                    ; procedure nommé Calcul
                ...                              ; instructions
              RET                              ; dernière instruction
Calcul      ENDP                          ; fin de la procédure
```

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

L'appel s'écrit simplement : CALL Calcul

VI / 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 8086possède 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 indexe.

- Mode d'adressage indexée.

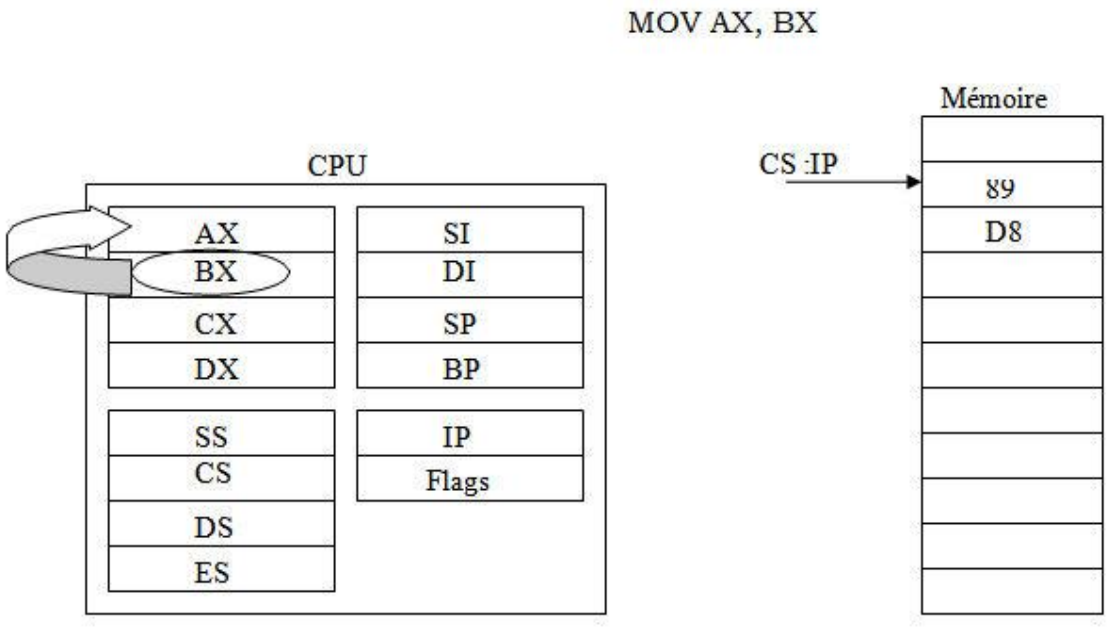
VI -1 / Mode d'adressage registre :

Ce mode d'adressage concerne tout transfert ou toute opération, entre deux registres de même taille.

Dans ce mode l'opérande sera stockée dans un registre interne au microprocesseur.

Exemple :

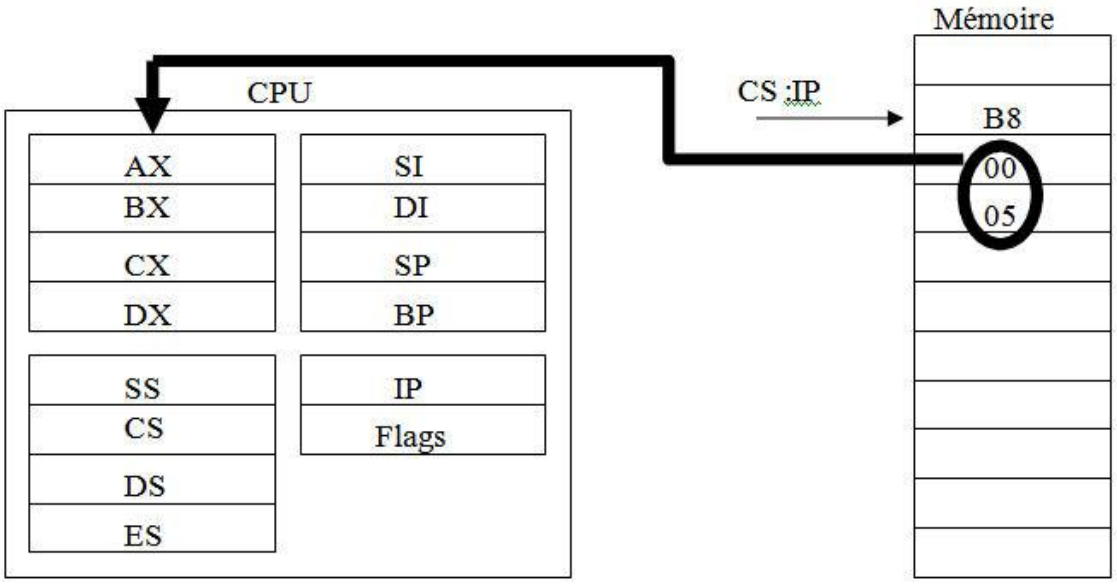
Mov AX, BX ; cela signifie que l'opérande stocker dans le registre BX sera transféré vers le registre AX. Quand on utilise l'adressage registre, le microprocesseur effectue toutes les opérations d'une façon interne. Donc dans ce mode il n'y a pas d'échange avec la mémoire, ce qui augmente la vitesse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



IV -2 / 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.
```

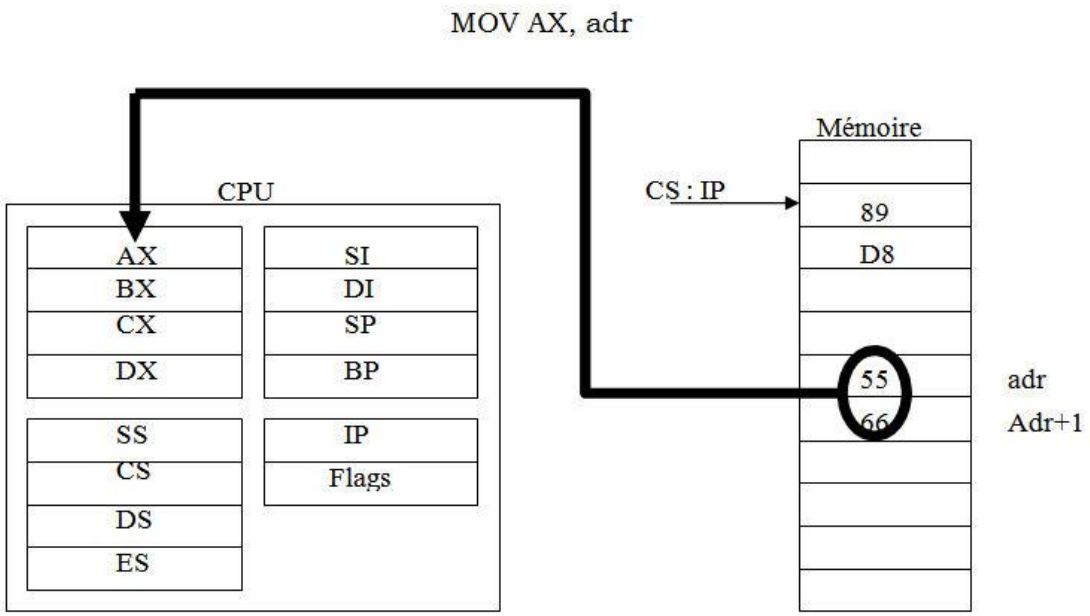
Exemple dans notre cas MOV AX,-500H donne AX =1111101100000000B MOV BL,-20H donne BL = 11100000B

VI -3 / 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
```

VI -4 / 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 indexe (SI ou DI).

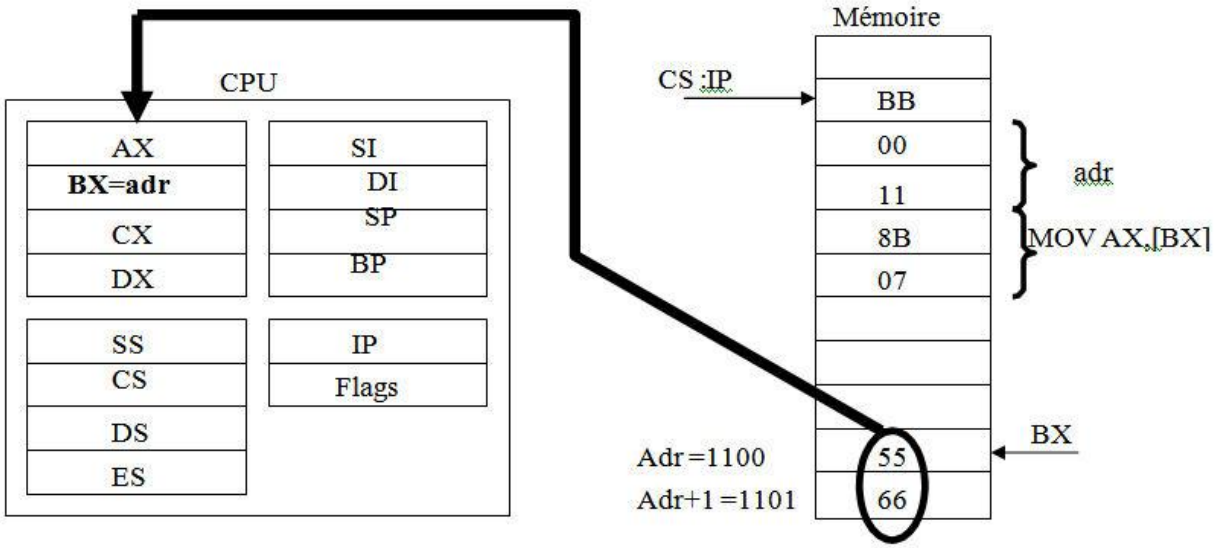
Exemple :

```
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.



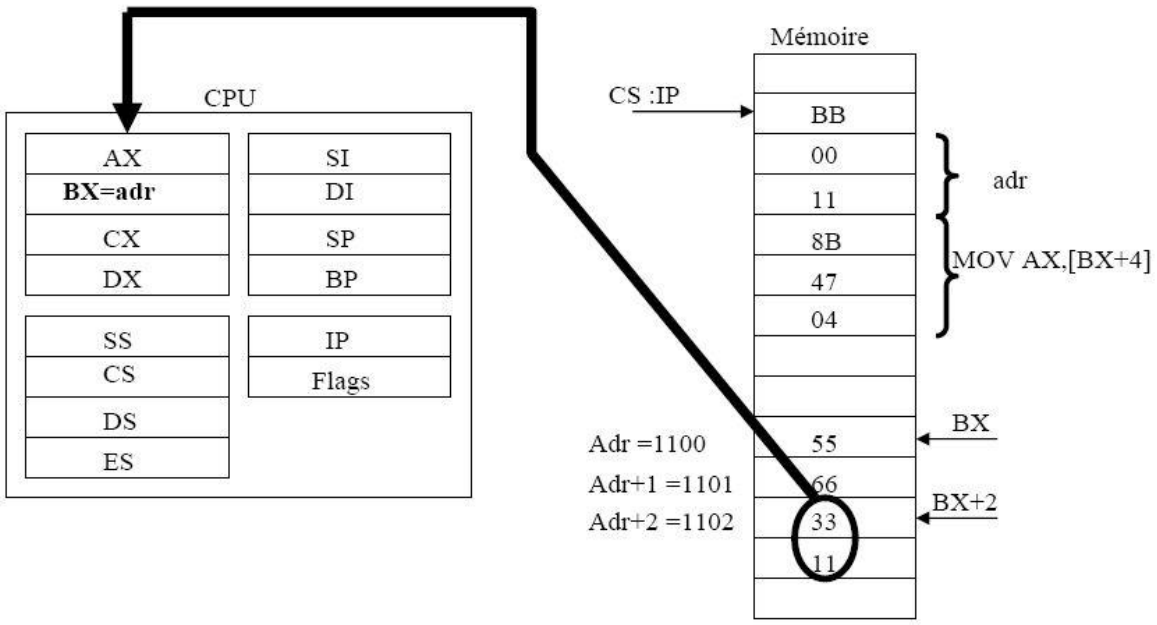
VI - 5 / Mode d'adressage relatif à une base :

Dans ce mode d'adressage Le déplacement est déterminé par soi, 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



Remarque :

Les syntaxes suivantes sont identiques :

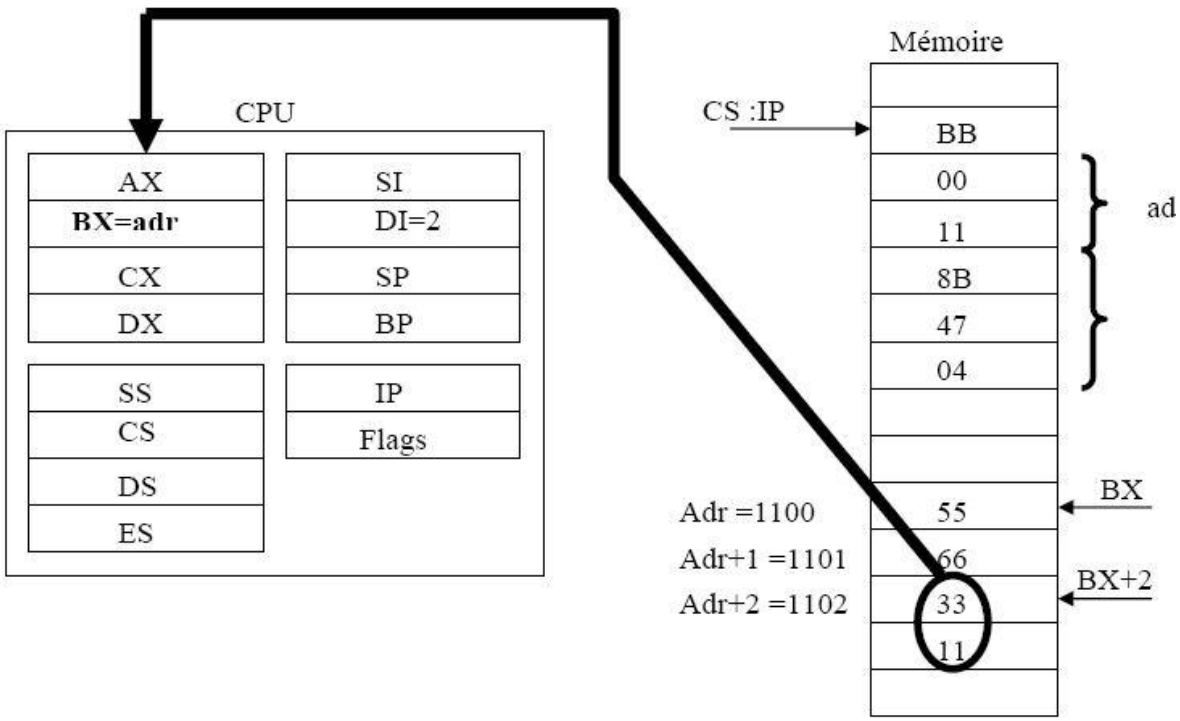
```
MOV AX,[BX+2] MOV AX,[BX]+2
MOV AX,2[BX]
```

VI - 6 / Mode d'adressage direct indexe :

Dans ce mode d'adressage Le déplacement est déterminé par soi, le contenu de SI, soit le contenu de DI, auquel est éventuellement ajouté un décalage sur 8 ou 16 bits signé. DS est pris par défaut.

Exemple :

```
MOV DI,2
MOV AL,adr[DI]
```



VI - 7 / Mode d'adressage base indexe avec déplacement :

Dans ce mode d'adressage le déplacement est déterminé par la somme du contenu du registre BX ou BP et d'un des registres d'index SI ou DI, auquel est éventuellement ajouté un décalage sur 8 ou 16 bits signé. DS et SS sont pris par défaut.

Exemple :

```
MOV AX,[BX][SI]
```

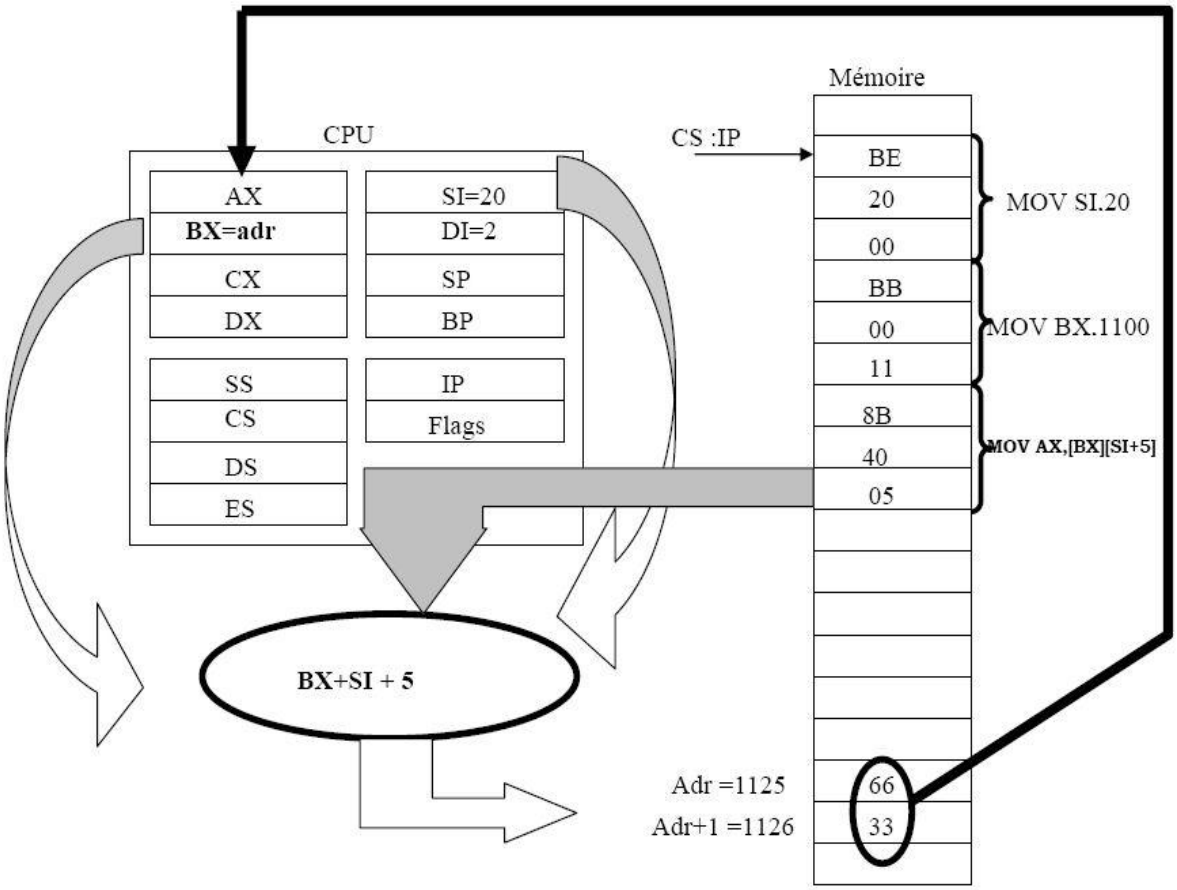
Cela signifie que dans le registre AX on va charger le contenu de la case mémoire pointer par BX+SI

De même dans ce mode on trouve aussi :

```
MOV AX,[BX][SI+5]
```

Cela signifie que dans le registre AX on va mètre le contenu de la case mémoire pointer BX+SI+5

```
MOV SI,20
MOV BX,1100
MOV AX,[BX][SI+5]
```



Remarque :

Les syntaxes suivantes ont la même équivalence :

MOV AX,[BX][DI+4] MOV AX,[BX+DI+4] MOV AX,[BX+4+DI] MOV AX,[BX+4][DI]

Résumé:

MODE d'adressage	**Format de l'opérande**	**Registre Segment**
Registre	Registre	Aucun
Immédiat	Donnée	Aucun
Direct	Déplacement ou label	DS
Registre indirect	[BX]	DS
	[BP]	SS
	[SI]	DS
	[DI]	DS
Relatif a une base	[BX] +déplacement	DS
	[BP]+déplacement	SS
Direct indexe	Label+ [SI]	DS
	Label+ [DI]	DS
Base indexée	Label+[BX] [SI]	DS
	Label+ [BX] [DI]	DS
	Label+ [BP] [SI]	SS
	Label+[BP] [DI]	SS

LE JEU D'INSTRUCTIONS DU 8086/8088

I / introduction :

On peut diviser les instructions du 8086/88 en 6 groupes comme suit :

- Instructions de transfert de données.
- Instructions arithmétiques.
- Instructions de bits (logiques).
- Instructions de sauts de programme.
- Instructions de chaîne de caractères.
- Instructions de contrôle de processus.
- Instructions d'interruptions.

II / Les instructions de transfert de données :

Ils sont divisés en 4 sous- groupes comme le montre le tableau suivant :

Usage	Nom	Fonction
Général	MOV	Transfert d'octets ou de mots
	PUSH	Chargement de la pile
	POP	Déchargement de la pile
	PUSHA	Chargement de tous les registres dans la
	POPA	pile Déchargement de tous les registres
	XCHG	dans la pile Echange d'octet ou de mot
	XLAT	Translation d'octet
Entrées-sorties	IN	Entrée de mot ou d'octet
	OUT	Sortie de mot ou d'octet
Adresses	LEA	Chargement de l'adresse effective
	LDS	Chargement du pointeur avec DS
	LES	Chargement du pointeur avec ES
Indicateurs	LAHF	Transfert des indicateurs dans AH
	SAHF	Rangement de AH dans les indicateurs
	PUSHP	Chargement des indicateurs dans la pile
	POPF	Déchargement des indicateurs de la pile.

II – 1 / Les instructions d'usage général :

II – 1 – 1 / MOV :

Elle permet de transférer les données (un octet ou un mot) d'un registre à un autre registre ou d'un registre à une case mémoire, sa syntaxe est comme suit :

Exemples :

```
MOV destination, source
MOV   AX, BX     ;  Transfert d'un registre de 16 bits vers un registre de
16 Bits
MOV   AH, CL     ;  Transfert d'un registre de 8 bits vers un registre de 8
bits
MOV   AX, Val1   ; Transfert du contenu d'une case mémoire 16 bits vers AX
MOV   Val2, AL   ; Transfert du contenu du AL vers une case mémoire
D'adresse Val2
```

Remarques :

- Il est strictement interdit de transférer le contenu d'une case mémoire vers une autre case mémoire comme suit

```
 MOV    Val1, Val2
```

Pour remédier à se problème on va effectuer cette opération sur deux étapes :

```
MOV AL, Val2
MOV Val1, AL
```

- On n'a pas le droit aussi de transférer un registre segment vers un autre registre segment sans passer par un autre registre :

```
MOV   DS, ES
```

On va passer comme la première instruction :

```
MOV   AX, ES
MOV   DS, DS
```

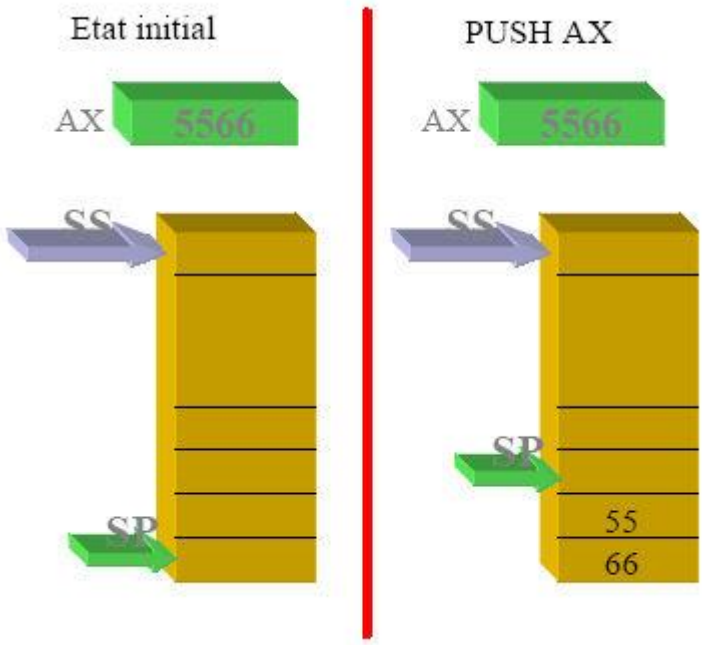
- Le CS n'est jamais utilisé comme registre destination.

II – 1 – 2 / PUSH :

Elle permet d'empiler les registres du CPU sur le haut de la pile

```
Syntaxe : PUSH  SOURCE
```

Exemple :

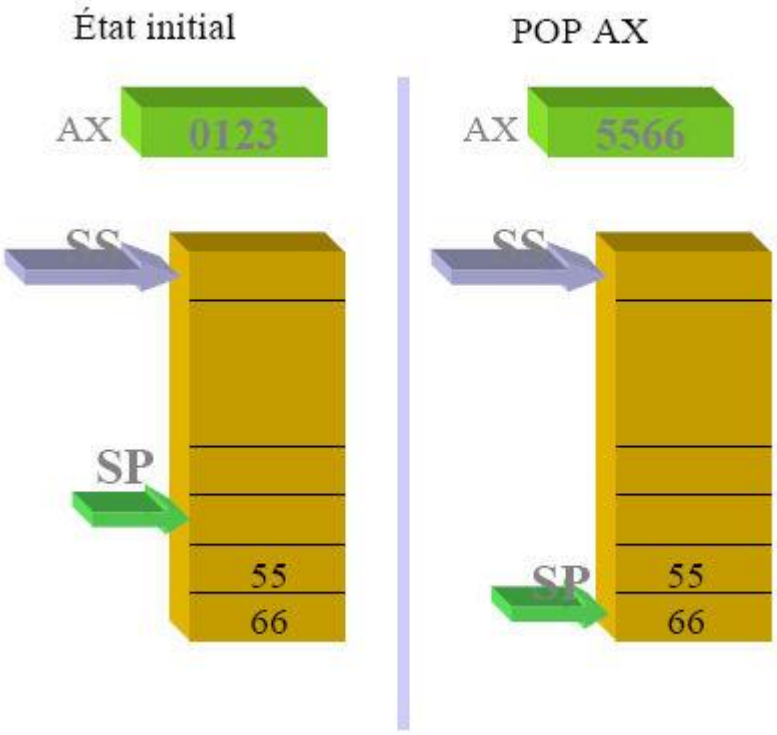


II – 1 – 3 / POP :

Elle permet de dépiler les registres du CPU sur le haut de la pile

```
Syntaxe : POP  destination
```

Exemple :



II – 1 – 4 / PUSHA :

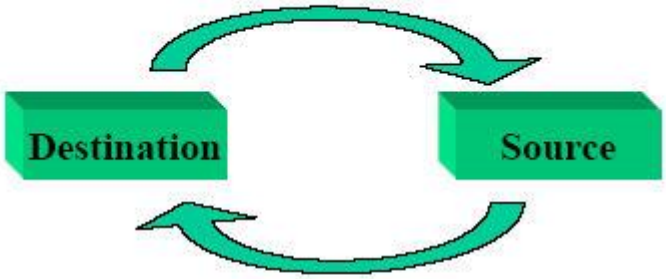
Cette instruction permet d'empiler la totalité des registres internes du microprocesseur sur la pile.

II – 1 – 5 / POPA :

Cette instruction permet de dépiler la totalité des registres internes du microprocesseur sur la pile.

II – 1 – 6 / XCHG :

Elle permet de commuter la source avec la destination comme suit :



XCHG AX,BX ; elle permute entre AX et BX
Exemple AX=0123 et BX = 5678
Après l'exécution de l'instruction on aura :
AX=5678 et BX = 0123

II – 1 – 7 / XLAT :

Cette instruction est utilisée pour convertir des données d'un code à un autre, en effet elle permet de placer dans l'accumulateur AL le contenu de la case mémoire adressée en adressage base+décalage (8 bits), la base étant le registre BX et le décalage étant AL lui même dans le segment DS

```
(AL)<--------[ (BX) + (AL) ]
```

Syntaxe : XLAT tab_source

Exemple :

conversion du code binaire 4 bits en un digit hexa codé en ASCII

```
Tab  db     '0123456789ABCDEF'
MOV AL,1110B        ; chargement de la valeur à convertir (07) MOV BX,
OFFSET TAB
                    ; pointé sur le tableau
XLAT                ; Al est chargé par le code ASCII de 'E'
```

II – 2 / les instructions d'entrées – sorties :

II – 2 – 1 / IN / OUT:

Elle permet de récupérer des données d'un port (donc de la périphérie) ou restituer des données à un port, dans les deux cas s'il s'agit d'envoyer ou de recevoir un octet on utilise l'accumulateur AL, s'il s'agit d'envoyer ou de recevoir un mot on utilise l'accumulateur AX.

Syntaxe :

```
IN ACCUMULATEUR, DX
OUT DX, ACCUMULATEUR
```

Remarque :

DX : contient l'adresse du port.

ACCUMULATEUR : contient la donnée (à recevoir ou à emmètre).

II – 3 / Les instructions de transfert d'adresses :

II – 3 – 1 / LEA (Load Effective Address):

Elle transfert l'adresse offset (décalage) d'une opérande mémoire dans un registre de 16 bits (pointeur ou index). Cette commande a le même rôle que l'instruction MOV avec offset mais elle est plus puissante car on peut utiliser avec elle toute technique d'adressage.

Exemple :

LEA BX, TAB_VAL (c'est équivalent à MOV BX, offset TAB_VAL)

II – 3 – 2 / LDS / LES :

Cette instruction permet de charger le segment et l'offset d'une adresse

Exemple :

Au lieu de faire :

```
MOV   BX, offset tab_val
MOV   AX , Seg tab_val
MOV  DS , AX
```

On remplace ces trois instructions par une seule :

```
LDS BX , tab_val ;elle   charge   automatiquement l'offset de tab_val dans
le registre BX
                 ;et le segment dans le registre DS .
```

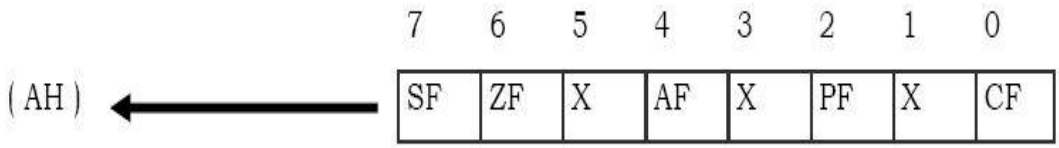
Remarque :

Pour l'instruction LES : le segment est ES.

II – 4 / Les instructions d'indicateur :

II – 4 – 1 / LAHF / SAHF :

LAHF : Load AH from Flags : place l'octet de poids faible du registre d'état (FLAGS) dans le registre AH comme suit :



SAHF : Store AH into Flags : Place le contenu de AH dans l'octet de poids faible du registre d'état (FLAGS).

II – 4 – 2 / PUSHF / POPF:

PUSHF : Permet d'empiler la totalité du registre d'état (FLAGS)

POPF : Permet de dépiler le registre d'état (FLAGS).

III / Instructions arithmétiques :

Les instructions arithmétiques peuvent manipuler quatre types de nombres :

- Les nombres binaires non signés
- Les nombres binaires signés.
- Les nombres décimaux codés binaires (DCB), non signés.
- Les nombres DCB non condensés, non signés.

Les instructions arithmétiques sont divisées en quatre sous-groupes comme le montre le tableau suivant :

Usage	Nom	Fonction
Addition	ADD	Addition sur un octet ou un mot
	ADC	Addition sur un octet ou un mot avec retenue
	INC	Incrémentation de 1
	AAA	Ajustement ASCII
	DAA	Ajustement décimal
Soustraction	SUB	Soustraction sur un octet ou un mot
	SBB	Soustraction sur un octet (mot) avec retenue
	DEC	Décrémentation de 1
	NEG	Mètre un octet ou un mot en négatif
	CMP	Comparaison d'octet ou mot
	AAS	Ajustement ASCII
	DAS	Ajustement décimal
Multiplication	MUL	Multiplication d'octet ou de mot **non signée**
	IMUL	Multiplication d'octet ou de mot **signée**
	AAM	Ajustement ASCII
Division	DIV	Division d'octet ou de mot **non signée**
	IDIV	Division d'octet ou de mot **signée**
	AAD	Ajustement ASCII
	CBW	Conversion d'un octet en un mot
	CWD	Conversion d'un mot en double mots

III – 1 / Addition :

III – 1 – 1 / ADD: (Addition)

```
Syntaxe :    ADD   Destination, source
```

Elle permet d'additionner le contenu de la source (octet ou un mot) avec celui de la destination le résultat est mis dans la destination

```
  Destination <---------- Destination  +  source
```

Exemples :

```
ADD AX, BX  ; AX = AX + BX (addition sur 16 bits) ADD  AL,BH
            ; AL = AL + BH (addition sur 8 bits )
ADD AL, [SI]  ; AL = AL + le contenu de la case mémoire
              ; pointé par SI
ADD [DI], AL ; le contenu de la case mémoire pointé par DI
             ; Est additionnée avec AL, le résultat est mis
```

```
            ; dans la case mémoire pointé par DI
```

Remarques :

- Ici on a presque les mêmes restrictions de l'instruction MOV c.a.d on n'a pas le droit d'additionner deux cases mémoires sans utiliser un registre de données.

Exemples :

```
ADD Tab1 , Tab2
```

sera remplacé par :

```
MOV AX , Tab2
ADD Tab1 , AX
```

- De même une valeur immédiate ne peut être une destination.

III – 1 – 2 / ADC : (Addition avec retenue)

```
Syntaxe :    ADC   Destination, source
```

Elle permet d'additionner le contenu de la source (octet ou un mot) avec celui de la destination et la retenue (CF) le résultat est mis dans la destination

```
  Destination  <------------   Destination  +  source + retenue
```

Exemples :

```
ADC AX,BX   ; AX = AX + BX + CF(addition sur 16 bits )
ADC AL,BH   ; AL = AL + BH + CF(addition sur 8 bits )
ADC AL,[SI] ; AL = AL + le contenu de la case mémoire pointé par SI + CF
ADC [DI],AL ; le contenu de la case mémoire pointé par DI
            ; est additionné avec AL + CF , le résultat est
            ; mis dans  la case mémoire pointé par DI
```

Remarque :

Les restrictions de l'instruction ADD sont valables pour l'instruction ADC.

III – 1 – 3 / INC : (Incrémentation)

```
Syntaxe :    INC    Destination
```

Elle permet d'incrémenter le contenu de la destination

```
  Destination <----------  Destination  +  1
```

Exemples :

```
INC AX     ; AX = AX +  1 (incrémentation sur 16  bits).
INC AL     ; AL = AL +1 (incrémentation sur 8 bits).
INC [SI]   ; [SI] = [SI] + 1 le contenu de la case mémoire pointé  par SI
sera incrémenter
```

Remarque :

On ne peut pas incrémenter une valeur immédiate.

III – 1 – 4 / AAA / DAA : (ASCII / DECIMAL Adjust for Addition)

L'addition de deux nombres BCD génére parfois un résultat qui n'est pas un nombre en BCD d'ou il faut faire des corrections sur ces nombres pour avoir un résultat cohérent. Cette instruction examine le quarte bas de AL et vérifie s'il est conforme ou non :

- Si oui (elle met AF et CF à zéro pour information) efface la quarter haut de AL
- Si non :
    - Elle ajoute 6 à AL
    - Ajoute 1 à AH
    - Efface le quarter haut de AL
    - Met AF et CF à 1 (pour information)

Exemples : on veut faire l'addition en BCD de 73 + 88

73		0111 0011
	+	
88		1000 1000
163		1111 1011

On remarque que ni 1111 ni 1011 est un nombre BCD donc on va ajouter 6

au premier quarte d'ou l'opération devient :

	1111 1011		
+	0110		
=	10000 0001		
+	0110		
=	0110 0001	(63)	

L'octet le plus haut :

	00000001	
+	00000000	
=	00000001	(01)

d'ou les résultat 163

III – 2 / Soustraction :

III – 2 – 1 / SUB : (Soustraction)

```
Syntaxe :    SUB   Destination, source
```

Elle permet de soustraire la destination de la source (octet ou un mot) le résultat est mis dans la destination

```
  Destination <---------  Destination  --  source
```

Exemples :

```
SUB AX,BX    ; AX = AX - BX (Soustraction sur 16 bits )
SUB  AL,BH   ; AL = AL - BH ( Soustraction sur 8 bits )
SUB AL,[SI]  ; AL = AL - le contenu de la case mémoire pointé par SI
SUB [DI],AL  ; le contenu de la case mémoire pointé par DI
             ; est soustraite de AL , le résultat est mis
             ; dans la case mémoire pointé par DI
```

Remarques :

- On a les mêmes restrictions de l'instruction ADD.

III – 2 – 2 / SBB : (Soustraction avec retenue)

```
Syntaxe :    SBB   Destination, source
```

Elle permet de soustraire la destination de la source et la retenue (octet ou un mot) le résultat est mis dans la destination

```
 Destination <--------   Destination  --  source -- retenue
```

Exemples :

```
SBB AX,BX    ; AX = AX – BX - CF (Soustraction sur 16 bits )
SBB  AL,BH   ; AL = AL - BH - CF( Soustraction sur 8 bits )
SBB AL,[SI]  ; AL = AL - le contenu de la case mémoire
             ; pointé  par SI - CF
SBB [DI],AL  ; le contenu de la case mémoire pointé par DI
             ; est  soustraite avec AL - CF, le résultat est
             ; mis dans  la case mémoire pointé par DI
```

Remarques :

- On a les mêmes restrictions de l'instruction ADD.

III – 2 – 3 / DEC : (Décrémentation)

```
Syntaxe :    DEC   Destination
```

Elle permet de décrémenter le contenu de la destination

```
  Destination   <----------- Destination  -  1
```

Exemples :

```
DEC AX    ; AX = AX -  1 (décrémentation sur 16  bits).
DEC  AL   ; AL = AL -1 (décrémentation sur 8 bits).
DEC [SI]  ; [SI] = [SI] - 1 le contenu de la case mémoire
          ; pointé  par SI sera décrémenter
```

Remarque :

On ne peut pas décrémenter une valeur immédiate.

III – 2 – 4 / NEG : (Négatif)

```
Syntaxe :    NEG   Destination
```

Elle soustrait l'opérande destination (octet ou mot) de 0 le résultat est stocker dans la destination, donc avec cette opération on réalise le complément à deux d'un nombre

```
  Destination  <---------- 0 - Destination
```

Exemples :

```
NEC AX           ; AX = 0 - AX
NEC AL           ; AL = 0 - AL
NEC [SI]         ; [SI] = 0 - [SI]
```

Remarque :

Les indicateurs affectés par cette opération sont : AF, CF, OF, PF, SF, ZF

III – 2 – 5 / CMP : (Comparaison)

```
Syntaxe :    CMP  Destination   , Source
```

Elle soustrait la source de la destination , qui peut être un octet ou un mot , le résultat n'est pas mis dans la destination , en effet cette instruction touche uniquement les indicateurs pour être tester avec une autre instruction ultérieure de saut conditionnel

Les indicateurs susceptibles d'être touché sont : AF, CF, OF, PF, SF, ZF

Donc cette instruction va nous permettre de comparer deux nombres comme le montre le tableau suivant :

	Opérande non signé				Opérande signé			
	OF	SF	ZF	CF	OF	SF	ZF	CF
Source < destination	-	-	0	0	0/1	0	0	-
Source = destination	-	-	1	0	0	0	1	-
Source > destination	-	-	0	1	0/1	1	0	-

III – 2 – 6 / AAS / DAS: (ASCII / DECIMAL Adjust for Substraction)

Elle est identique à l'instruction AAA/DAA mais l'ajustement se fait en BCD pour la soustraction.

III – 3 / La multiplication :

III – 3 – 1 / MUL : (Multiplication pour les nombres non signés)

MUL effectue une multiplication non signée de l 'opérande source avec l'accumulateur :

```
Syntaxe : MUL   Source
```

- -Si la source est un octet alors elle sera multipliée par l'accumulateur AL le résultat sur 16 bits sera stocké dans le registre AX.
- Si la source est un mot alors elle sera multipliée avec l'accumulateur AX le résultat de 32 bits sera stocké dans la paire des registres AX et DX

Remarque :

Cette multiplication traite les données en tant que nombres non signés

Donc on aura :

```
(AX)       <-------------   (AL) X  Source (octet)
(AX)(DX)   <------------    (AX) X Source (mots)
```

En conclusion :

Multiplication	Opérande 1	Opérande 2	Résultat
Octet x Octet	AL	Registre ou memoire	AX
Mots x Mots	AX	Registre ou memoire	DX AX
Mots x Octet	AL= Octet, AH=0	Registre ou memoire	DX AX

III – 3 – 2 / IMUL : (Multiplication pour les nombres signés)

MUL effectue une multiplication signée de l 'opérande source avec l'accumulateur :

```
Syntaxe :    IMUL   Source
```

- Si la source est un octet alors elle sera multiplieéepar l'accumulateur
- AL le résultat sur 16 bits sera stocké dans le registre AX .
- Si la source est un mot alors elle sera multiplié avec l'accumulateur AX le résultat de 32 bits sera stocké dans la paire des registres AX et DX

Remarque :

Cette multiplication traite les données en tant que nombres signés

III – 3 – 3 / AAM: (ASCII Adjust for Multiplication)

Comme AAA et AAS cette instruction va nous permettre de corriger le résultat d'une multiplication de deux nombres en BCD, pour corriger le résultat de l'instruction AAM divise AL par 10.

Exemple :

```
MOV AL , 6
MOV DL , 8
MUL DL
AAM
```

Si on décortique cette instruction on aura :

```
      AL        0110
X     DL        1000
-------------------------
=     AX    01100000      cela implique que AL= 01101000
```

si on divise AL par 10 = 1010

```
01100000 | 1010
- 1010   | ----
-------  |  100
    1000 |
         |
```

III – 4 / La division :

III – 4 – 1 / DIV : (Division des nombres non signés)

```
Syntaxe :   DIV  Source
```

Elle effectue une division non signée de l'accumulateur par l'opérande source :

Exemples :

- Si l'opérande est un octet : alors on récupère le quotient dans le registre AL et le reste dans le registre AH.
- Si l'opérande est un mot : alors on récupère le quotient dans le registre AX et le reste dans le registre DX

A/

```
MOV AH,00h
MOV AL,33H
MOV DL,25H
DIV DL    ; Cela implique que AH=    et   AL =
```

B/

```
MOV AX,500H
MOV CX,200H
DIV CX     ;Cela implique que    AX=         et    AL =
```

III – 4 – 2 / IDIV : (Division des nombres signés)

```
Syntaxe :    IDIV  Source
```

Elle effectue une division signée de l'accumulateur par l'opérande source :

- Si l'opérande est un octet : alors on récupère le quotient dans le registre AL et le reste dans le registre AH.
- Si l'opérande est un mot : alors on récupère le quotient dans le registre AX et le reste dans le registre DX

Exemples:

A/

```
MOV  AH,00h
MOV  AL,-33H
MOV DL,25H
IDIV  DL  ;   Cela implique que AH=   et   AL =
```

B/

```
MOV AX,-500H
MOV CX,200H
IDIV  CX   ;  Cela implique que AX=   et   AL =
```

III – 4 – 3 / AAD : (ASCII Adjust for Division)

Pour corriger le résultat elle va multiplier le contenu de AH par 10 et l'ajoute à celui de AL

Remarque :

Pour cette instruction il faut faire l'ajustement avant l'instruction de division.

III – 4 – 4 / CBW (convert byte to word)

Cette instruction permet de doubler la taille de l'opérande signé

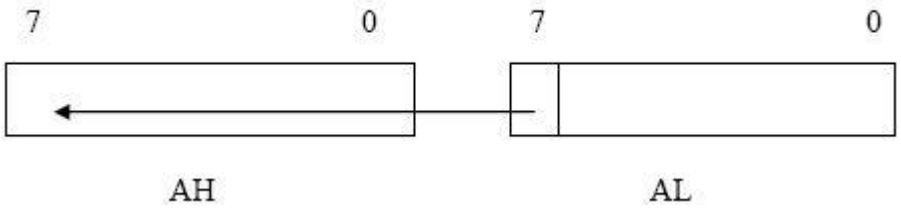
```
  Octet ------------>  Mots
```

Remarque :

CBW reproduit le bit 7 (bits de signe) de AL dans AH jusqu'à remplissage de ce dernier.

Exemple :

```
MOV AL, +96       ; AL=0110 0000
CBW               ; AH=0000 0000 et AL=0110 0000
```



III – 4 – 5 / CWD (convert Word to Double)

Cette instruction permet de doubler la taille de l'opérande signé

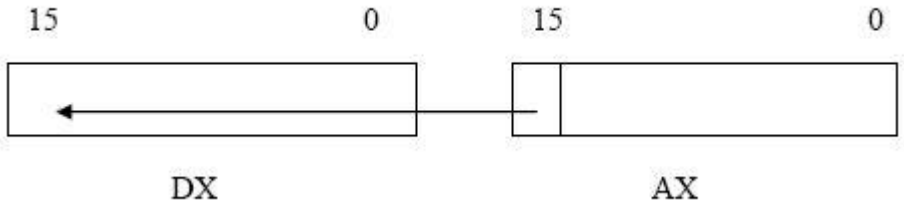
```
Word    -------- >    Double
```

Remarque :

CWD reproduit le bit 15 (bits de signe) de AX dans DX jusqu'à remplissage de ce dernier.

Exemple :

```
MOV  AX, +260     ; AX=0000 0001 0000 0100
CWD               ; DX=0000H, AX=0104H
```



IV / Les instructions logiques (de bits) :

Ils sont divisés en trois sous-groupes comme le montre le tableau suivant :

Usage	Nom	Fonction
Logique	NOT	Inversion logique sur un octet ou un mot
	AND	Et logique
	OR	Ou logique
	XOR	Ou exclusif
	TEST	Et logique sans résultat, affecte uniquement les indicateurs du registre des flags.
Décalages	SHL	Décalage logique à gauche
	SAL	Décalage arithmétique à gauche
	SHR	Décalage logique à droite
	SAR	Décalage arithmétique à droite
Rotation	ROL	Rotation à gauche
	ROR	Rotation à droite
	RCL	Rotation à gauche à travers le bit de retenue
	RCR	Rotation à droite à travers le bit de retenue

IV – 1 / Les instructions logiques :

IV – 1 – 1 / NOT : (Négation)

Elle réalise la complémentation à 1 d'un nombre

```
Syntaxe :       NOT   Destination
```

Exemple :

```
MOV AX, 500        ; AX = 0000 0101 0000 0000
NOT AX             ; AX = 1111 1010 1111 1111
```

IV – 1 – 2 / AND : (Et logique)

```
Syntaxe :    AND   Destination, source
```

Elle permet de faire un ET logique entre la destination et la source (octet ou un mot) le résultat est mis dans la destination

```
  Destination    <----------   Destination  .  source
```

Exemples :

```
MOV AX , 503H       ; AX = 0000 0101 0000 0011
AND AX , 0201H      ;       0000 0101 0000 0011
                    ;  AND  0000 0010 0000 0001
                    ;     = 0000 0000 0000 0001
AND AX,BX           ; AX = AX . BX (Et logique entre AX et BX)
AND  AL,BH          ; AL = AL . BH (ET logique sur 8 bits)
AND AL,[SI]         ; AL = AL AND le contenu de la case mémoire
                    ; pointé  par SI
AND [DI],AL         ; ET logique entre la case mémoire pointé par
                    ; DI et  AL , le résultat est mis dans la case
                    ; mémoire pointé par DI
```

IV – 1 – 3 / OR : (OU logique)

```
Syntaxe :    OR     Destination, source
```

Elle permet de faire un OU logique entre la destination et la source (octet ou un mot) le résultat est mis dans la destination

```
   Destination   <----------- Destination  +  source
```

Exemples :

```
MOV AX , 503H        ; AX = 0000 0101 0000 0011
OR AX , 0201H        ;          0000 0101 0000 0011
                     ;    OR    0000 0010 0000 0001
                     ;    =     0000 0111 0000 0011
OR AX,BX             ; AX = AX + BX ( OU logique entre AX et BX )
OR  AL,BH            ; AL = AL + BH ( OU logique sur 8 bits )
OR AL,[SI]           ; AL = AL OU le contenu de la case mémoire
                     ; pointé  par SI
OR [DI],AL           ; OR logique entre la case mémoire pointé par
                     ; DI et  AL, le résultat est mis dans la case
                     ; mémoire pointé par DI
```

IV – 1 – 4 / XOR : (OU exclusif)

```
Syntaxe :    XOR  Destination, source
```

Elle permet de faire un OU exclusif logique entre la destination et la source (octet ou un mot) le résultat est mis dans la destination

```
  Destination   <------------ Destination  +  source
```

Exemples :

```
MOV AX , 503H   ; AX = 0000 0101 0000 0011
XOR AX , 0201H  ;          0000 0101 0000 0011
                ;   XOR    0000 0010 0000 0001
                ;    =     0000 0011 0000 0010
XOR AX,BX       ; AX = AX + BX (OU exclusif  entre AX et BX )
XOR  AL,BH      ; AL = AL + BH ( OU exclusif sur 8 bits )
XOR AL,[SI]     ; AL = AL OU exclusif le contenu de la case
                ; Mémoire pointé  par SI
XOR [DI],AL     ; XOR logique entre la case mémoire pointé par
                ; DI et  AL, le résultat est mis dans la case
```

```
                ; mémoire pointé par DI
```

IV – 1 – 5 / TEST :

```
Syntaxe :    TEST   Destination, source
```

Elle permet de faire un ET logique entre la destination et la source (octet ou un mot) mais la destination ne sera pas touchée en effet cette instruction ne touche que les indicateurs.

Exemples :

```
MOV AX, 503H    ; AX = 0000 0101 0000 0011
TEST AX, 0201H  ;  0000 0101 0000 0011
```

Elle va effectuer un ET logique entre le premier nombre et le second sans toucher les deux mais elle va affecter uniquement les indicateurs (Flags)

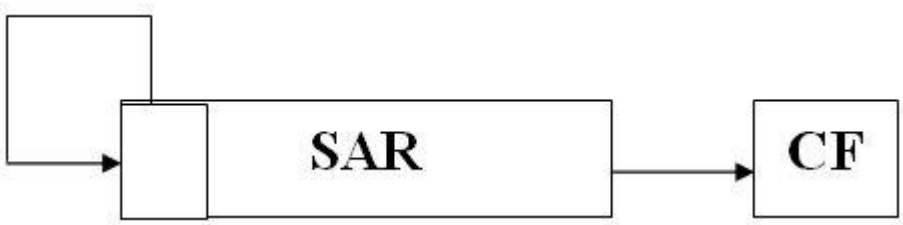
IV – 2 / Les instructions de décalages :

SHL : décalage logique à gauche :

SHR : décalage logique à droite :

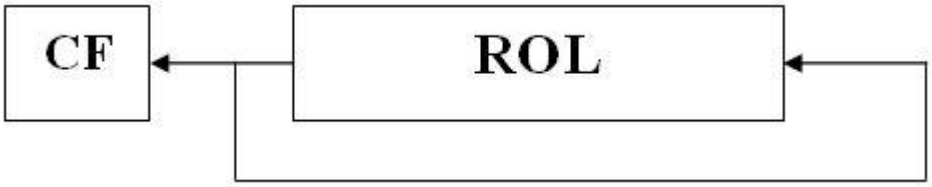
SAL : décalage arithmétique à gauche :

SAR : décalage arithmétique à gauche :

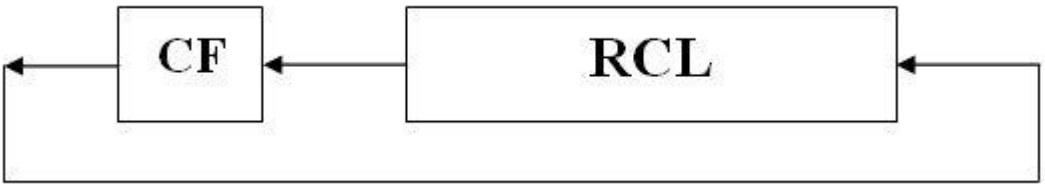


IV – 3 / Les instructions de rotations :

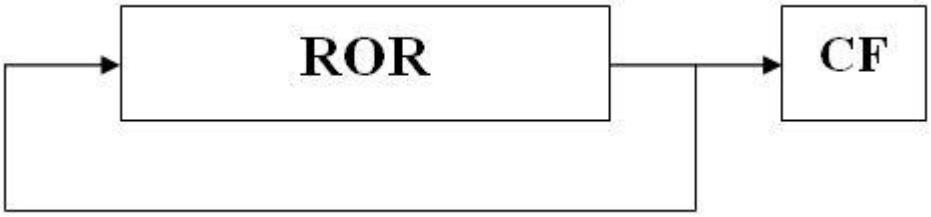
ROL : Rotation à gauche :



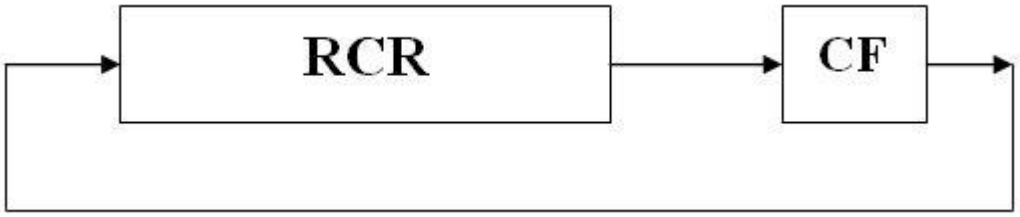
RCL : Rotation a travers la retenue à gauche :



ROR : Rotation à droite :



RCR : Rotation à travers la retenue à droite :



Syntaxe des instructions de rotation et de décalage :

```
ROR  destination, compteur
```

Exemple :

```
        ROR AX,1
        ROL AL,1
```

Si on veut faire quatre rotations de suite on a deux solutions :

```
ROR AL,1
```

```
ROR AL,1
ROR AL,1
ROR AL,1
```

Ou encore :

```
        MOV CL,4
        ROR AL,CL
```

Remarque :

Les instructions de rotations et de décalages logiques ne tiennent pas compte du bit de signe donc elles travaillent avec les nombres non signés.

Les instructions de rotations et de décalages arithmétiques préservent le bit de signe donc elles sont réservées aux nombres signés.

V / Instructions de sauts de programme :

Elles permettent de faire des sauts dans l'exécution d'un programme (rupture de séquence)

Remarque :

Ces instructions n'affectent pas les Flags. Dans cette catégorie on trouve toutes les instructions de branchement, de boucle et d'interruption après un branchement, le tableau suivant donne ces instructions :

Type	Nom	Fonction
Branchements inconditionnels	CALL	Appel à un sous programme
	RET	Retour d'un sous programme
	JMP	Saut
Branchements conditionnels (arithmétique non signée)	JA/JNBE	Si supérieur / Si non inférieur ou non égal
	JAE/JNB	Si supérieur ou égal/ Si non inférieur
	JB/JNAE	Si inférieur/si non supérieur ni égal
	JBE/JNA	Si inférieur ou égal/si non supérieur.
Branchements conditionnels (arithmétique signée)	JG/JNLE	Si plus grand/si pas inférieur ni égal
	JGE/JNL	Si plus grand ou égal/Si pas inférieur
	JL/JNGE	Si moins que/Si pas plus grand ni égal
	JLE/JNG	Si moins que ou égal/Si pas plus grand
Branchement conditionnels (flags)	JC	Si retenue
	JE/JZ	Si égal/Si zéro
	JNC	Si pas de retenue
	JNE/JNZ	Si non égal / Non zéro
	JNO	Si pas de débordement
	JNP/JPO	Si pas de parité/ Si parité impaire
	JNS	Si pas de signe
	JO	Si débordement
	JP/JPE	Si parité / Si parité paire
	JS	Si signe (négatif)
Boucles	LOOP	Boucle
	LOOPE/LOOPZ	Boucle si égal/Si zéro
	LOOPNE/LOOPNZ	Boucle si différent/si diff 0
	JCXZ	Branchement si CX=0
Interruptions	INT	Interruption
	INTO	Interruption si débordement
	IRET	Retour d'interruption.

V-1 / Branchement inconditionnel

V-1-1 / CALL : notion de procédure :

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

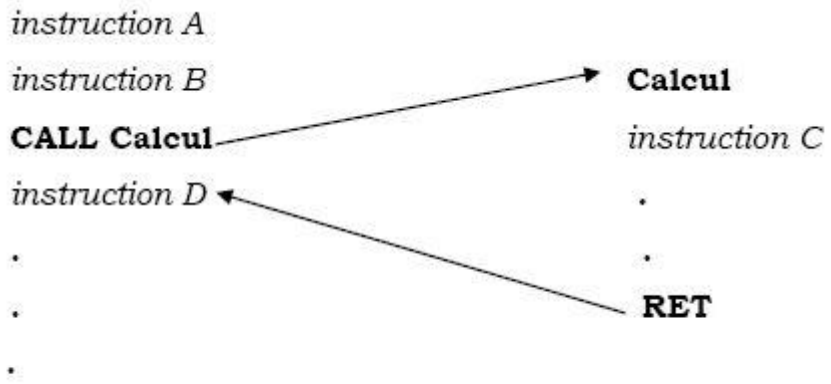


FIG. – Appel d'une procédure.

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

Une procédure est une suite d'instructions effectuant une action précise, qui sont regroupées par commodité et pour éviter d'avoir à les écrire à plusieurs reprises dans le programme.

Les procédures sont repérées par l'adresse de leur première instruction, à laquelle on associe une étiquette en assembleur.

L'exécution d'une procédure est déclenchée par un programme *appelant*. Une procédure peut elle-même appeler une autre procédure, et ainsi de suite.

Instructions CALL et RET

L'appel d'une procédure est effectué par l'instruction CALL.

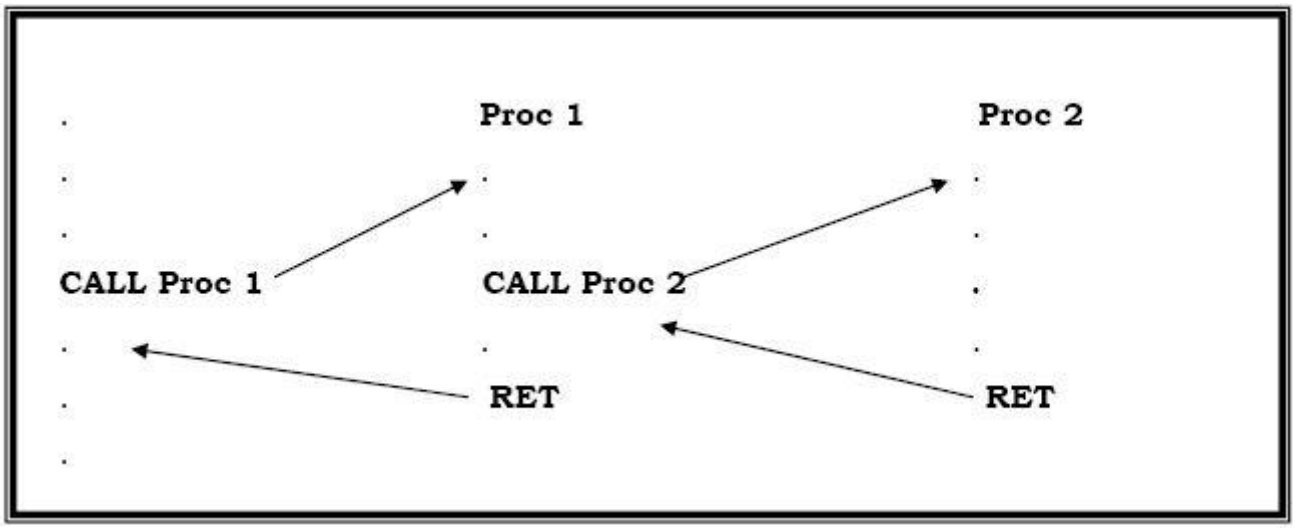
CALL *adresse_debut_procedure*

L'adresse est sur 16 bits, la procédure est donc dans le même segment d'instructions. CALL est une nouvelle instruction de branchement inconditionnel. La fin d'une procédure est marquée par l'instruction RET :

V-1-2 / RET :

RET ne prend pas d'argument ; le processeur passe à l'instruction placée immédiatement après le CALL.

RET est aussi une instruction de branchement : le registre IP est modifié pour revenir à la valeur qu'il avait avant l'appel par CALL. Comment le processeur retrouve-t-il cette valeur ? Le problème est compliqué par le fait que l'on peut avoir un nombre quelconque d'appels imbriqués, comme sur la figure suivante :



L'*adresse de retour*, utilisée par RET, est en fait sauvegardée sur la pile par l'instruction CALL. Lorsque le processeur exécute l'instruction RET, il dépile l'adresse sur la pile (comme POP), et la range dans IP.

L'instruction CALL effectue donc les opérations :

- Empiler la valeur de IP. A ce moment, IP pointe sur l'instruction qui suit le CALL.
- Placer dans IP l'adresse de la première instruction de la procédure (donnée en argument).

Et l'instruction RET :

- Dépiler une valeur et la ranger dans IP.

Remarque 1 :

Si la procédure appartient au même segment que le programme principal elle est dite de type NEAR sinon elle est dite de type FAR, la différence entre eux c'est que dans le premier cas le processeur doit empiler une seule valeur dans la pile c'est le registre IP mais dans le deuxième cas il faut empiler le registre IP ainsi que le registre segment CS et bien sur il les dépiler pendant le retour de la procédure.

Remarque 2 : Passage de paramètres

En général, une procédure effectue un traitement sur des données

(*paramètres*) qui sont fournies par le programme appelant, et produit un résultat qui est transmis à ce programme. Plusieurs stratégies peuvent être employées :

1. *Passage par registre* : les valeurs des paramètres sont contenues dans des registres du processeur. C'est une méthode simple, mais qui ne convient que si le nombre de paramètres est petit (il y a peu de registres).

2. *Passage par la pile* : les valeurs des paramètres sont empilées. La procédure lit la pile.

Exemple avec passage par registre

On va écrire une procédure "SOMME" qui calcule la somme de 2 nombres naturels de 16 bits.

Convenons que les entiers sont passés par les registres AX et BX, et que le résultat sera placé dans le registre AX.

La procédure s'écrit alors très simplement : SOMME PROC NEAR

```
ADD AX, BX      ; AX <- AX + BX
RET SOMME ENDP
```

et son appel, par exemple pour ajouter 6 à la variable Truc :

```
MOV AX, 6
MOV BX, Truc
CALL SOMME
MOV Truc, AX
```

Exemple avec passage par la pile

Cette technique met en œuvre un nouveau registre, BP (*Base Pointer*), qui permet de lire des valeurs sur la pile sans les dépiler ni modifier SP. Le registre BP permet un mode d'adressage indirect spécial, de la forme :

```
MOV AX, [BP+6]
```

Cette instruction charge le contenu du mot mémoire d'adresse BP+6 dans

AX. Ainsi, on lira le sommet de la pile avec :

```
MOV BP, SP          ; BP pointe sur le sommet
MOV AX, [BP]        ; lit sans dépiler
```

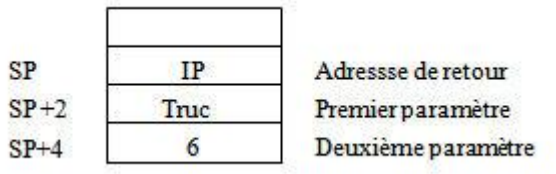
Et le mot suivant avec :

```
MOV AX, [BP+2]      ; 2 car 2 octets par mot de pile.
```

L'appel de la procédure "SOMME2" avec passage par la pile est :

```
PUSH 6
PUSH Truc
CALL SOMME2
```

La procédure SOMME2 va lire la pile pour obtenir la valeur des paramètres. Pour cela, il faut bien comprendre quel est le contenu de la pile après le CALL :



Le sommet de la pile contient l'adresse de retour (ancienne valeur de IP

empilée par CALL). Chaque élément de la pile occupe deux octets. La procédure SOMME2 s'écrit donc :

```
SOMME2 PROC near              ; AX <- arg1 + arg2
MOV BP, SP                    ; adresse sommet pile
MOV AX, [BP+2]                ; charge argument 1
ADD AX, [BP+4]                ; ajoute argument 2
RET
SOMME2 ENDP
```

La valeur de retour est laissée dans AX.

La solution avec passage par la pile parait plus lourde sur cet exemple simple. Cependant, elle est beaucoup plus souple dans le cas général que le passage par registre. Il est très facile par exemple d'ajouter deux paramètres supplémentaires sur la pile. Une procédure bien écrite modifie le moins de registres possible. En général, l'accumulateur est utilisé pour transmettre le résultat et est donc modifié. Les autres registres utilisés par la procédure seront normalement sauvegardés sur la pile. Voici une autre version de SOMME2 qui ne modifie pas la valeur contenue par BP avant l'appel :

```
SOMME2 PROC near                  ; AX <- arg1 + arg2
PUSH BP                           ; sauvegarde BP
MOV BP, SP                        ; adresse sommet pile
MOV AX, [BP+4]                    ; charge argument 1
ADD AX, [BP+6]                    ; ajoute argument 2
POP BP                            ; restaure ancien BP
RET
SOMME2 ENDP
```

Noter que les index des arguments (BP+4 et BP+6) sont modifiés car on a ajouté une valeur au sommet de la pile.

V-1-3 / JMP : (Saut inconditionnel)

Syntaxe :

```
JMP cible
```

Si le JMP est de type NEAR alors IP = IP + Déplacement

Si le JMP est de type FAR alors CS et IP sont remplacé par les nouvelles valeurs obtenues à partir de l'instruction.

JMP transfert, sans condition, la commande à l'emplacement de destination. L'opérande Cible peut être obtenu à partir de l'instruction elle- même (JMP direct) ou à partir de la mémoire ou à partir d'un registre indiqué par l'instruction.

V-2 saut conditionnel :

V-2-1 / JC : (Si retenue)

Si CF=1 alors IP = IP + déplacement

V-2-2 / JE/JZ :(Si égal/Si zéro)

Si ZF=1 alors IP = IP + déplacement

V-2-3 / JNC :(Si pas de retenue)

Si CF=0 alors IP = IP + déplacement

V-2-4 / JNE/JNZ :(Si non égal / Non zéro)

Si ZF=0 alors IP = IP + déplacement

V-2-5 / JNO :(Si pas de débordement)

Si OF=0 alors IP = IP + déplacement

V-2-6 / JNP/JPO :(Si pas de parité/ Si parité impaire)

Si PF=0 alors IP = IP + déplacement

V-2-7 / JNS :(Si pas de signe)

Si SF=0 alors IP = IP + déplacement

V-2-8 / JO :(Si débordement)

Si OF=0 alors IP = IP + déplacement

V-2-9 / JP/JPE:(Si parité / Si parité paire)

Si PF=1 alors IP = IP + déplacement

V-2-10 / JS :(Si signe (négatif))

Si SF=1 alors IP = IP + déplacement

V-3 / Les instructions de boucle :

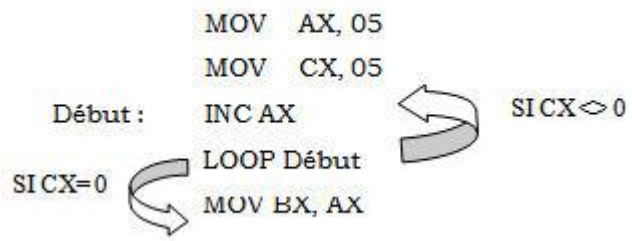
V-3-1 /LOOP : (boucle) :

Elle décrémente le contenu de CX de 1.

Si CX est différente de zéro alors IP = IP + déplacement

Si CX = 0 l'instruction suivante est exécutée.

Exemple :

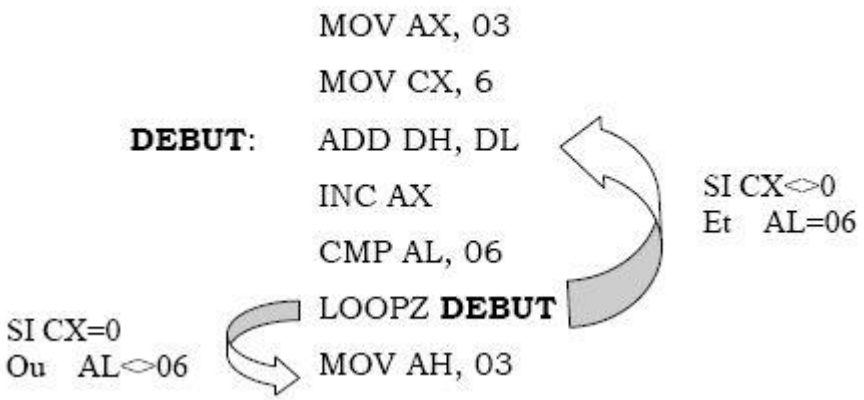


L'exécution de l'instruction MOV BX, AX sera faite après l'exécution de la boucle 5 fois.

V-3-2 / LOOPE / LOOPZ : (boucle si égale ou si égale à zéro) : Le registre CX est décrémenter de 1 automatiquement

Si CX est différent de zéro et ZF=1 alors IP = IP + déplacement

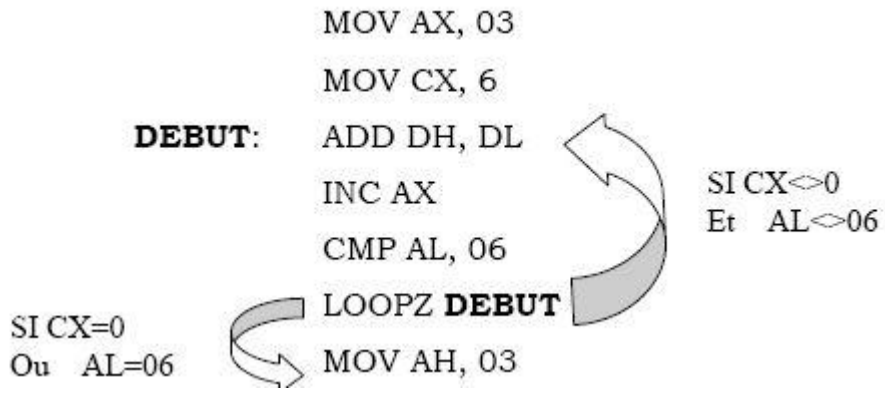
Exemple :



V-3-3 / LOOPNE / LOOPNZ : (boucle si égale ou si égale à zéro) : Le registre CX est décrémenter de 1 automatiquement

Si CX est différent de zéro et ZF=0 alors IP = IP + déplacement

Exemple :



VI / Les instructions de chaînes de caractères :

Les instructions de chaînes des caractères sont au nombre de 14 comme le montre le tableau suivant :

Nom	Fonction
REP	Préfixe de répétition
REPE/REPZ	Répétition tant qu'égal à zéro
REPNE/REPNZ	Répétition tant que différent de zéro
MOVS	Déplacement de chaîne
MOVSB/MOVSW	Déplacement de chaîne
CMPS	Comparaison de chaînes
INS	Entrée (de port d'E/S)
OUTS	Sortie (vers un port d'E/S)
SCAS	Balayage d'une chaîne
LODS	Chargement de chaîne
STOS	Rangement de chaînes

Elles permettent de travailler sur des blocs d'octets ou de mots allant jusqu'à 64 Koctet.

Remarque :

Ces blocs peuvent être des valeurs numériques ou alphanumériques.

VI -1 / Les préfixes de répétitions :

VI-1-1 / REP :

Ces instructions sont utilisées avec les instructions de chaînes de caractères pour assurer la répétition de l'instruction si on veut appliquer l'instruction sur un ensemble d'informations.

REP décrément automatiquement CX est test est ce qu'il est égal à zéro ou non. Si CX = 0 REP s'arrêt

VI-1-2 / REPE / REPZ :

Pour REPE/REPZ : c'est la même chose que REP c'est-à-dire elle décrément automatiquement le registre CX mais elle peut sortir de la boucle si ZF<>0

VI-1-2 / REPNE / REPNZ :

Pour REPNE/REPNZ : c'est la même chose que REP c'est-à-dire elle décrément automatiquement le registre CX mais elle peut sortir de la boucle si ZF=0

VI-2/ Les instructions MOVE-STRING :

Elle déplace un élément du segment de données pointé par

DS : SI vers le segment Extra pointé par ES : DI

Remarque :

Si l'élément à transférer est un octet on utilise : MOVB Si l'élément à transférer est un Mot on utilise : MOVW

Mais dans les deux cas on n'utilise que d'opérande.

SI et DI sont ensuite incrémentés de 1 (si DF=0) ou décrémentés de 1 (si DF=1) d'une manière automatique.

Exemple :

```
Donnee SEGMENT
Mess_Sour db 'bonjour iset de nabeul'     ; message source
Donnee ENDS
Extra SEGMENT
Mes_Des db 22 dup (0)                    ; message destination
Extra ENDS
Code SEGMENT
Assume CS : Code, DS : Donnee, ES : extra
PROG PROC
MOV AX,Donnee MOV DS, AX MOV AX,Extra MOV ES, AX
LEA SI, Mess_Sour                ; pointé le message source
LEA DI, Mess_Des                 ; pointé le message destination
MOV CX, 22                       ; nombre de caractère à transfèrer
CLD                              ; incrémentation automatique du SI et DI
```

```
REP MOVSB                          ; transfert avec le préfixe REP MOV AX,
4C00H
                                   ; Retour au DOS
INT 21H
PROG ENDP
CODE ENDS
END PROG
```

VI-3/ Les instructions COMPARE-STRING :

Comparaison de chaîne : elle soustrait l'octet ou mot de destination (pointé par DI) de l'octet ou mot source (pointé par SI).CMPS affecte les indicateurs mais ne change pas les opérandes.

Si CMPS est utilisé avec le préfixe de répétition REPE/REPZ, elle est interprétée comme « comparer tant que la chaîne n'est pas finie (CX <>0) et que les éléments à comparer ne sont pas égaux (ZF=1)

Si CMPS est utilisé avec le préfixe de répétition REPNE/REPNZ, elle est interprétée comme « comparer tant que la chaîne n'est pas finie (CX <>0) et que les éléments à comparer ne sont pas égaux (ZF=0)

Remarque :

On ne peut pas utilisé le préfixe REP avec l'instruction CMPS car cela revient à comparer uniquement les deux derniers éléments des deux chaînes.

Exemple :

trouver le premier caractère différent entre les deux chaînes.

```
Donnee SEGMENT
Mess_1 db 'bonjour iset de nabeul' Mess_2 db 'bonsoir iset de nabeul'
Donnee ENDS
Code SEGMENT
Assume CS : Code, DS : Donnee
PROG PROC
MOV AX,Donnee
MOV DS, AX
LEA SI, Mess_Sour                  ; pointé le message source
LEA DI, Mess_Des                   ; pointé le message destination
MO V CX, 22                        ; nombre de caractère àcomparer
CLD                                ; incrémentation automatique du SI et DI
REP CMPSB                          ; transfert avec le préfixe REP
JCXZ                                Tito  ; les deux chaînes sont identiques
MOV al, [SI-1]          ; on met dans AL le caractère
; Différent
Tito : MOV AX, 4C00H               ; Retour au DOS INT 21H
PROG ENDP
CODE ENDS
END PROG
```

VI–4 / Les instructions SCAN STRING :

Syntaxe :

```
SCAS chaine_destination
SCASB
SCASW
```

SCAS soustrait l'élément de la chaîne de destination (octet ou mot) adressé par DI dans le segment extra du contenu de AL (un octet) ou de AX (un mot) et agit sur les indicateurs. Ni la chaîne destination ni l'accumulateur ne change de valeur.

Exemple :

recherche de la lettre 'a' dans une chaîne. EXTRA SEGMENT

```
Mess_Des db 'bonjour iset de nabeul' EXTRA ENDS
Code SEGMENT
Assume CS : Code, ES : EXTRA
PROG PROC MOV AX, EXTRA MOV ES, AX
LEA DI, Mess_Des                    ; pointé le message destination
MOV CX, 22                          ; nombre de caractère à comparer
CLD                                 ; incrémentation automatique du DI
MOV AL,'a'
REPNZ SCASB                         ; transfert avec le préfixe REP
JCXZ                                 Tito  ; les deux chaînes sont identiques
Call oui_le_a_existe          ; on met dans AL le caractère
; Différent
Tito : MOV AX, 4C00H                ; Retour au DOS INT 21H
PROG ENDP
CODE ENDS
END PROG
```

VI-5 / Les instructions LOAD STRING (LODS) et STORE STRING (STOS) :

VI-4-1 / LODS :

Syntaxe :

```
LODS chaine_source
LODSB
LODSW
```

LODS transfert l'élément de chaîne (octet ou mot) adressé par SI au registre AL ou AX et remet à jour SI pour qu'il pointe vers l'élément suivant de la chaîne.

VI-4-2 / STOS :

Syntaxe :

```
STOS chaine_destination
STOSB
STOSW
```

STOS transfert un octet ou un mot du registre AL ou AX vers l'élément de chaîne adressé par DI et modifie DI pour qu'il pointe vers l'emplacement suivant de la chaîne.

Exemple :

transfert d'une chaîne source vers une chaîne destination en utilisant LODS et STOS.

```
Donnee SEGMENT
Mess_Sour db 'bonjour iset de nabeul'      ; message source
Donnee ENDS
Extra SEGMENT
Mes_Des db 22 dup (0)                      ; message destination
Extra ENDS
Code SEGMENT
Assume CS : Code, DS : Donnee, ES : extra
PROG PROC
MOV AX,Donnee
MOV DS, AX
MOV AX,Extra
MOV ES, AX
LEA SI, Mess_Sour                  ; pointé le message source
LEA DI, Mess_Des                   ; pointé le message destination
MOV CX, 22                         ; nombre de caractère à transfèrer
CLD                                ; incrémentation automatique du SI et DI
DEBUT :     LODSB                  ; transfert avec le préfixe REP STOSB
LOOP DEBUT
MOV AX, 4C00H                      ; Retour au DOS INT 21H
PROG ENDP
CODE ENDS
END PROG
```

VII / Les instructions de commande du processeur :

Ces instructions agissent sur le processeur et ses indicateurs (Flags) ils sont en nombre de 12 comme le montre le tableau suivant

Type	Nom	Fonction
Indicateur (FLAGS)	STC	Met à 1 la retenue CF
	CLC	MET à 0 la retenue CF
	CMC	Complémente la retenue
	STD	Met à 1 la direction DF
	CLD	Met à 0 la direction DF
	STI	Met à 1 l'autorisation d'interruption
	CLI	Met à 0 l'autorisation d'interruption
Synchronisation	HLT	Halte jusqu'à interruption ou RESET
	WAIT	Attente jusqu'à broche TEST passe à 0
	ESC	Pour un coprocesseur
	LOCK	Verrouillage des bus pendant la prochaine instructions
Sans opération	NOP	Pas d'opération

VII-1 / Indicateurs :

VII-1-1/ STD :

Met CF à 1 ; les registres d'indexation SI et/ou DI sont alors automatiquement décrémenter par les instructions de chaîne de caractère.

VII-1-2 / STI :

Met IF à 1, permettant ainsi au CPU de reconnaître des demandes d'interruption masquables apparaissant sur la ligne d'entrée INTR.

VII-2 / Synchronisation :

VII-2-1 / HALT :

Maintient le processeur dans un état d'attente d'un RESET ou d'une interruption externe non masquable ou masquable (avec IF=1).

VII-2-2 / WAIT :

Met le CPU en état d'attente tant que sa ligne de TEST n'est pas active. En effet toutes les cinq périodes d'horloge le CPU vérifie est ce que cette entrée est active ou non, si elle est active le processus exécute l'instruction suivante à WAIT.

VII-2-3 / ESC :

L'instruction Escape fournit un mécanisme par lequel des coprocesseurs peuvent recevoir leurs instructions à partir de la suite d'instructions du 8086.

VII-2-4 / LOCK :

Elle utilise dans les systèmes Multiprocesseur en effet elle permet le verrouillage du bus vis-à-vis des autres processeurs.

VII-3 Sans opération :

VII-3-1 / NOP (No operation) :

Le CPU ne fait rien on peut s'en servir pour créer des temporisations.

Exemple :

```
Tempo : MOV CX, 7FFFH            ; Effectuer une temporisation
Temp1: PUSH CX                       ; avec deux boucles imbriqués
MOV CX,7FFFH
Temp2: NOP NOP NOP NOP
LOOP Temp2
POP CX
LOOP Temp1
RET
```

LES INTERRUPTIONS

I / Introduction :

Un système à base de microprocesseur peut comporter plusieurs périphériques tels que : un Ecran, une souris, une imprimante, un disque dure, un CNA (convertisseur numérique analogique), un CAN etc ..

Pour dialoguer avec ces périphériques le microprocesseur a trois façons de communiquer avec ces derniers :

- En questionnant de façon continue le périphérique pour vérifier que des données peuvent être lues ou écrites (Polling).

- En l'interrompant lorsqu'un périphérique est prêt à lire ou écrire des données (interruption)

- En établissant une communication directe entre deux périphériques (DMA : Direct memory acces).

I-1 / Le polling :

Bien que le polling soit une façon simple d'accéder à un périphérique , le débit des données dans un périphérique est parfois beaucoup plus lent que la vitesse de fonctionnement d'un processeur , et le polling peut donc être très inefficace . On lui préfère en général la méthode par interruptions.

I-2 / Le DMA :

En cas de transfert de données de grande quantité entre deux périphériques, il peut être plus efficace de laisser communiquer entre eux les deux périphériques plutôt que de solliciter le processeur. Ce type de communication avec les entrées/sorties s'appelle gestion de transfert par DMA, Le processeur indique au contrôleur de DMA quels sont les périphériques qui doivent communiquer, le nombre et éventuellement l'adresse des données à transférer, puis il initie le transfert. Le processeur n'est pas sollicité durant le transfert, et une interruption signale au processeur la fin du transfert.

I-3 / L'interruption :

Il arrive fréquemment qu'un périphérique ou un programme nécessite une intervention du microprocesseur à cet effet il est possible de l'interrompre quelques instants, celui-ci va effectuer l'instruction demandée, est exécuter, une fois qu'elle rend le travail initial.

Ce mode d'échange est principalement basé sur l'initiative du périphérique. Grâce à ce mécanisme d'interruption, le dispositif périphérique prend l'initiative de l'échange.

Une interruption est un évènement qui provoque l'arrêt du programme en cours et provoque le branchement du microprocesseur à un sous- programme particulier dit de "traitement de l'interruption".

Remarque 1 : (ordre de priorité)

Lorsque un périphérique reçoit des données pour le processeur, il envoie au processeur un signal d'interruption. Si celui-ci peut être interrompu (à condition qu'il ne soit pas en train de communiquer avec un périphérique de plus haute priorité) , il stoppe la tâche en cours , sauve son état (ces registres) en mémoire (la zone pile) et appelle la routine correspondant au numéro d'interruption .

Remarque 2 : (plusieurs interruptions):

Le fonctionnement des périphériques se fait en général d'une manière asynchrone, donc plusieurs interruptions peuvent être déclenchées en même temps, il existe souvent un contrôleur d'interruptions destiné à gérer les conflits entre interruptions

II / Interruption matériel et logiciel :

II – 1 / Interruption Matériel :

II-1-1 / Introduction

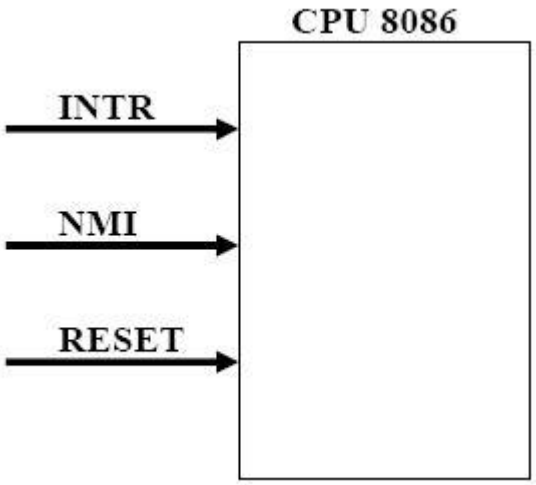
Une interruption est signalée au processeur par un signal électrique sur une borne spéciale. Lors de la réception de ce signal, le processeur "traite" l'interruption dès la fin de l'instruction qu'il était en train d'exécuter. Le traitement de l'interruption consiste soit :

– à l'ignorer et passer normalement à l'instruction suivante : c'est possible uniquement pour certaines interruptions, nommées *interruptions masquables*. Il est en effet parfois nécessaire de pouvoir ignorer les interruptions pendant un certains temps, pour effectuer des traitements très urgents par exemple. Lorsque le traitement est terminé, le processeur *démasque* les interruptions et les prend alors en compte.

- à exécuter un *traitant d'interruption* (interrupt handler). Un traitant d'interruption est un programme qui est appelé automatiquement lorsqu'une interruption survient. L'adresse de début du traitant est donnée par la table des *vecteurs d'interruptions (voir paragraphe suivant)* . Lorsque le (programme d'interruption) traitant a effectué son travail, il exécute l'instruction spéciale IRET qui permet de reprendre l'exécution à l'endroit où elle avait été interrompue.

II-1-2 / Cas du processeur 8086 :

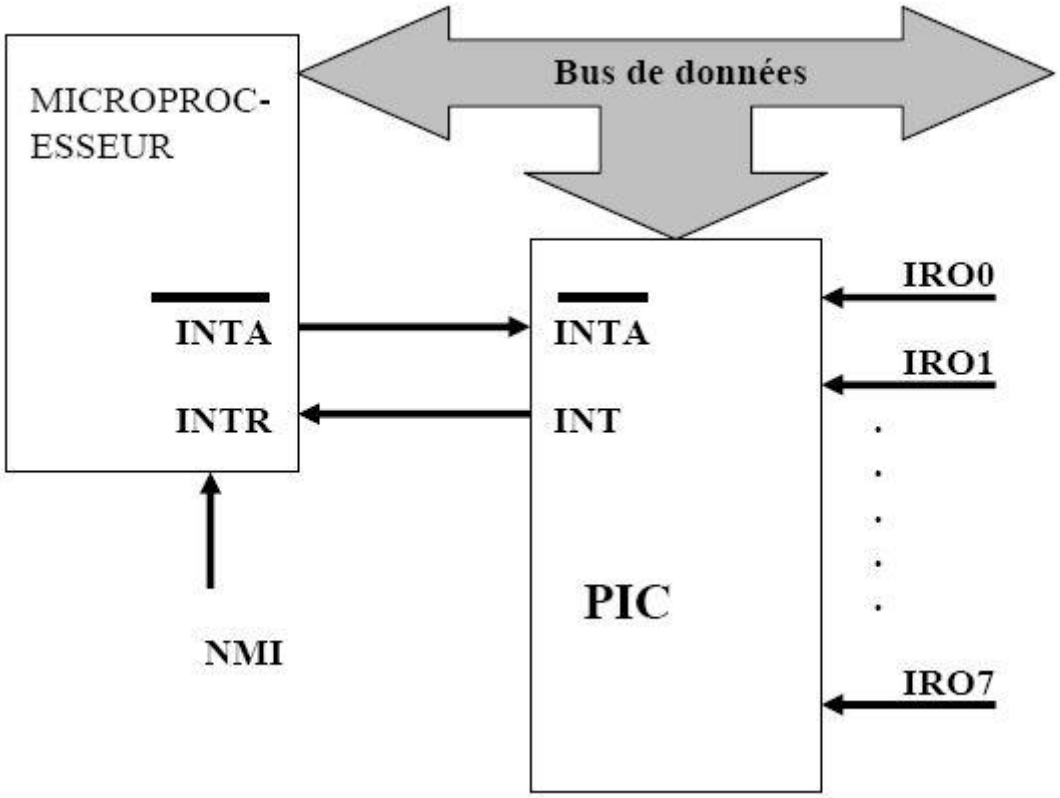
Le microprocesseur 8086 possède trois lignes principales d'interruption : INTR, NMI, et RESET



Ces trois entrées permettent l'arrivée d'une demande (interruption externe) extérieur.

Remarque : Contrôleur d'interruptions dans un PC

L'ordinateur est relié à plusieurs périphériques, mais nous venons de voir qu'il n'y avait qu'un seul signal de demande d'interruption, INTR. Le *contrôleur d'interruptions* est un circuit spécial, extérieur au processeur, dont le rôle est de distribuer et de mettre en attente les demandes d'interruptions provenant des différents périphériques.



Le contrôleur d'interruptions (PIC : Programmable Interruption Controler)

La figure précédente indique les connexions entre le microprocesseur et le contrôleur d'interruptions.

Le contrôleur est relié aux interfaces gérant les périphériques par les bornes IRQ (*InteRrupt reQuest*). Il gère les demandes d'interruption envoyées par les périphériques, de façon à les envoyer une par une au processeur (via INTR). Il est possible de programmer le contrôleur pour affecter des priorités différentes à chaque périphérique. Avant d'envoyer l'interruption suivante, le contrôleur attend d'avoir reçu le signal INTA, indiquant que le processeur a bien traité l'interruption en cours.

A / NMI : (No masquable interrupt)

Une interruption est dite non masquable signifie qu'elle doit toujours être reconnue par le microprocesseur dés que le signal électrique a été déclenché.

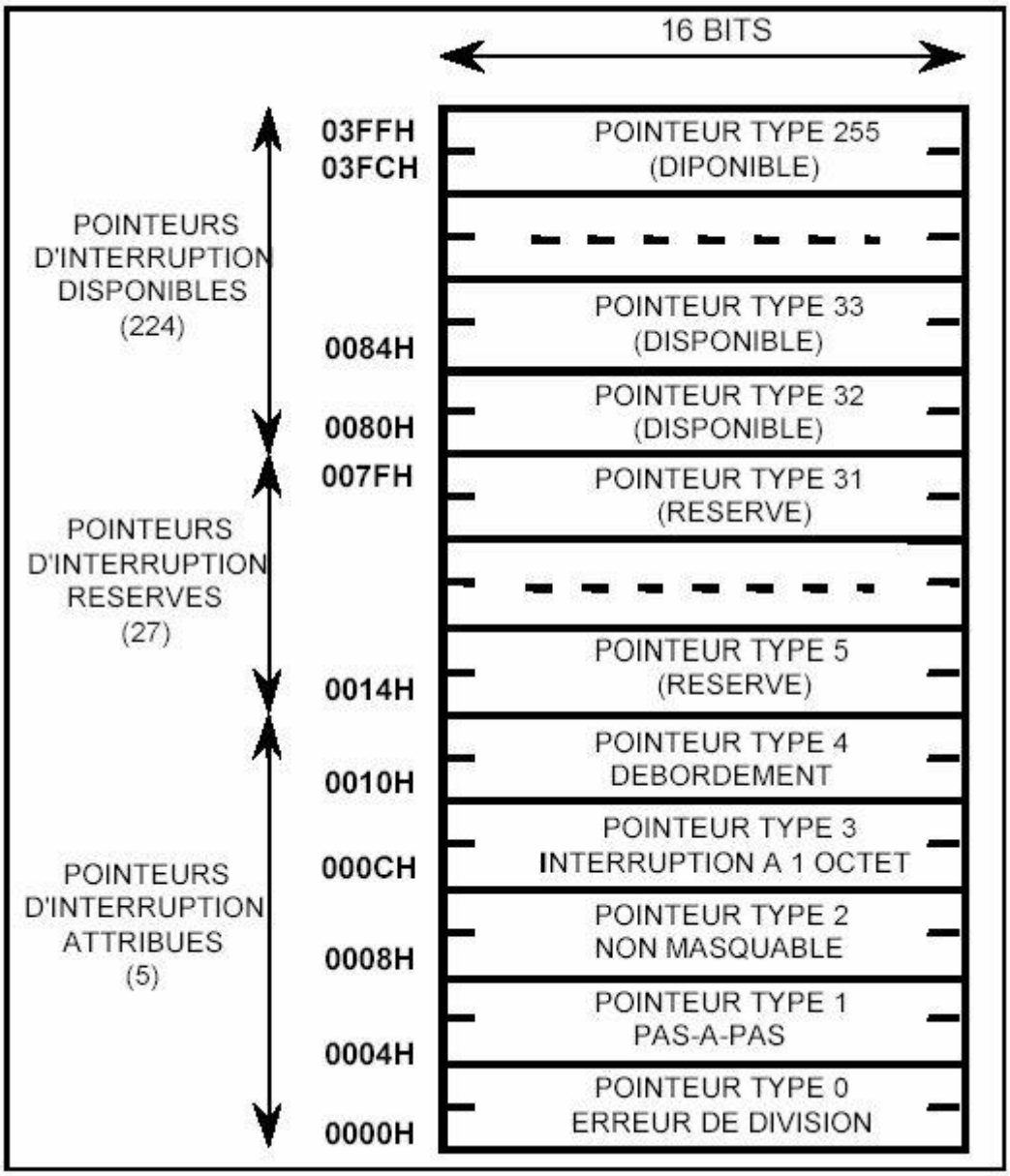
Remarque :

La reconnaissance de l'interruption NMI est généralement à la fin du cycle de l'instruction en cours, or les instructions du 8086 peuvent exiger un temps d'exécution plus ou moins long selon le nombre de cycles machines (comme l'instruction de multiplication) donc le temps de réponse va dépendre de l'instruction en cours. Ce temps est appelé temps de prise en charge de l'interruption.

Pendant la demande d'une interruption NMI tous les indicateurs sont empilés pour sauvegarder leurs valeurs en cours, puis l'indicateur IF (indicateur d'autorisation d'interruption : Interruption Flag) est mis à zéro (ce qui interdit toute arrivée de demande d'interruption sur la ligne INTR. Ensuite l'indicateur TF est mis à zéro (Trap Flag : indicateur pas à pas), puis le microprocesseur empile les registres CS et IP dans la pile. Enfin le microprocesseur charge le registre IP avec la valeur à 16 bits qui se trouve à l'adresse mémoire 00008H dans le vecteur d'interruption (voir figure suivante), et le registre CS sera ensuite chargé avec la valeur de 16 bits située à l'adresse 0000AH, donc le microprocesseur maintenant pointe sur le programme d'interruption qu'il va exécuter puis revenir à son programme principal.

Remarque : (Le vecteur d'interruption)

Nommer aussi table d'interruption, il contient toujours les adresses des programmes d'interruptions aux quels le microprocesseur doit se brancher. Le microprocesseur se branche toujours à l'adresse 4 * n et 4 * n +1 pour chercher le contenu sur 16 bits du CS Si on demande l'interruption INT n.



Déroulement d'une interruption externe masquable (exemple dans un PC) :

Reprenons les différents évènements liés à la réception d'une interruption masquable :

1. Un signal INT est émis par un périphérique (ou par l'interface gérant celui- ci).

2. Le contrôleur d'interruptions reçoit ce signal sur une de ses bornes IRQi. Dès que cela est possible (suivant les autres interruptions en attente de traitement), le contrôleur envoie un signal sur sa borne INT.

3. Le microprocesseur prend en compte le signal sur sa borne INTR après avoir achevé l'exécution de l'instruction en cours (ce qui peut prendre quelques cycles d'horloge). Si l'indicateur IF=0, le signal est ignoré, sinon, la demande d'interruption est acceptée.

4. Si la demande est acceptée, le microprocesseur met sa sortie INTA au niveau 0 pendant 2 cycles d'horloge, pour indiquer au contrôleur qu'il prend en compte sa demande.

5. En réponse, le contrôleur d'interruption place le numéro de l'interruption associé à la borne IRQi sur le bus de données.

6. Le processeur lit le numéro de l'interruption sur le bus de données et l'utilise pour trouver le vecteur d'interruption. Ensuite, tout se passe comme pour un appel système c'est à dire que le processeur :

(a) sauvegarde les indicateurs du registre d'état sur la pile ;

(b) met l'indicateur IF à 0 (masque les interruptions suivantes) ;

(c) sauvegarde CS et IP sur la pile ;

(d) cherche dans la table des vecteurs d'interruptions l'adresse du traitant d'interruption, qu'il charge dans CS:IP.

7. La procédure traitant l'interruption se déroule. Pendant ce temps, les interruptions sont masquées (IF=0). Si le traitement est long, on peut dans certains cas ré-autoriser les interruptions avec l'instruction STI.

8. La procédure se termine par l'instruction IRET, qui restaure CS, IP et les indicateurs à partir de la pile, ce qui permet de reprendre le programme qui avait été interrompu.

II – 2 / Interruption logiciel :

Les interruptions logicielles sont semblables aux interruptions matérielles. L'unique différence réside dans le fait que les interruptions logicielles sont émises par des programmes. Les cinq premières interruptions sont définies par Intel. Les autres interruptions sont définies par le DOS et le BIOS. Ces interruptions ont une fonction définie, par exemple la lecture et l'écriture sur le disque, l'écriture des données à l'écran, etc. Contrairement à l'entrée INTR du microprocesseur, l'interruption logicielle ne peut être ni invalidé ni masquée.

Exemple d'interruption software :

Fonctions BIOS :

Int 1Ah, Fct 02h Date et heure : Lecture de l'horloge temps réel BIOS (> AT) : Cette fonction permet de lire l'heure de l'horloge temps réel alimentée par batterie. Comme ce type d'horloge n'existe que sur les AT, seul ce modèle de PC soutient cette fonction.

Entrée :

```
AH    = 02h
```

Sortie :

```
Flag Carry =0 : Tout va bien, dans ce cas
                  CH = Heures
                  CL = Minutes
                  DH = Secondes
Flag Carry =1 : La batterie de l'horloge est déchargée
```

Remarques :

- Toutes les indications sont fournies en format BCD.

- Le contenu des registres BX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Fonction DOS :*Int 21h, Fct 06h Entrée/sortie directe de caractère :*

Cette fonction permet de sortir des caractères sur le périphérique de sortie standard ou de les lire sur le périphérique d'entrée standard. Le caractère reçu ou écrit chaque fois n'est pas examiné par le système d'exploitation. Rien de particulier ne se produit donc lorsqu'un caractère Ctrl-C apparaît. Comme l'entrée et la sortie standard peuvent être redirigées sur d'autres périphériques ou vers un fichier, les caractères sortis ne doivent pas nécessairement apparaître sur l'écran ni les caractères lus provenir obligatoirement du clavier. Toutefois, lorsque l'accès se fait sur un fichier, il est impossible pour le programme d'appel de détecter si tous les caractères de ce fichier ont déjà été lus ou bien si le support (disquette, disque dur) sur lequel figure ce fichier est déjà plein. Pour l'entrée d'un caractère, la fonction n'attend pas qu'un caractère soit prêt mais revient immédiatement au programme d'appel dans tous les cas.

Entrée :

```
AH  = 06h
DL  = 0 - 254 : Sortir ce caractère
DL  = 255 : Lire un caractère Sortie :
Pour la sortie de caractères : aucune
Pour l'entrée de caractères :
Flag Zéro =1 : aucun caractère n'est prêt
Flag Zéro =0 : Le caractère entré figure dans le registre AL
```

Remarques :

Lorsque des codes étendus sont lus, le code 0 est tout d'abord renvoyé dans le registre AL. La fonction doit être appelée à nouveau pour lire le code étendu lui-même.

- Le caractère de code ASCII 255 ne peut être sorti à l'aide de cette fonction puisqu'il est interprété comme d'entrée d'un caractère.

- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Exemple de programme : lecture et affichage d'un caractère sur

l'écran :

```
CODE SEGMENT ASSSUME CS :CODE PROG PROC
DEBUT :
MOV  AH,06H             ; préparer l'interruption de lecture d'un
MOV  DL,255              ; caractère
INT 21H
JZ DEBUT                   ; si le bit ZF = 0 => aucune touche n'est
appuyée
MOV DL,AL                ; afficher le caractère appuyer .
INT 21H
MOV AX,4C00H          ; retour au DOS
INT 21H PROG ENDP
CODE ENDS
END PROG
```

Int 21h, Fct 09h Sortie d'une chaîne de caractères

Cette fonction permet de sortir une chaîne de caractères sur le périphérique de sortie standard. Comme ce périphérique standard peut être redirigé sur un autre périphérique ou vers un fichier, il n'y a aucune garantie que la chaîne de caractères apparaisse sur l'écran. Si la sortie est redirigée sur un fichier, le programme d'appel n'a aucune possibilité de détecter si le support (disquette, disque dur) sur lequel figure le fichier est déjà plein, autrement dit s'il est encore possible d'écrire la chaîne de caractères dans le fichier.

Entrée :

AH = 09h

DS = Adresse de segment de la chaîne de caractères DX = Adresse d'offset de la chaîne de caractères

Sortie : Aucune

Exemple : affichage d'un message sur l'écran :

```
DONNEE SEGEMENT
MESSAGE   DB  ' BONJOUR ISET DE NABEUL $',13,10
DONNEE ENDS
CODE SEGMENT
ASSUME   CS : CODE , DS : DONNEE
PROG PROC
MOV AX , DONNEE MOV DS , AX
LEA DX , MESSAGE   ; Pointé le message par DX MOV AH , 09H
INT 21H                      ; afficher le message
MOV AX,4C00H          ; Retour au DOS
INT 21H
PROG ENDP CODE ENDS END PROG
```

Remarques :

La chaîne de caractères doit être stockée dans la mémoire sous forme d'une séquence d'octets correspondant aux codes ASCII des caractères composant la chaîne. La fin de la chaîne de caractères doit être signalée au DOS à l'aide d'un caractère "$" (code ASCII 36).

- Si la chaîne de caractères contient des codes de commande comme Backspace, Carriage Return ou Line Feed, ceux-ci seront traités comme tels.

- Seul le contenu du registre AL est modifié par l'appel de cette fonction. Exemple de programme utilisons les interruptions : affichage de l'heure d'un PC :

```
;*************************************************************************
******
;*************************************************************************
******
;****************** Programme affichage de l'heur du PC
************************
;*************************************************************************
******
;*************************************************************************
******
code segment assume cs:code
;*********************** programme principal
************************************
prog proc debut: mov ah,02
int 1aH                 ; interruption qui permet de lire l'heur du PC
mov bl,ch              ; ch = heure
call affiche         ; afficher les heurs
 mov bl,cl             ; cl=minute
call affiche          ; afficher les minutes
mov bl,dh               ; dh=seconde
call affiche             ; afficher les secondes
prog endp              ; fin de la procédure principale
mov dl,13              ; retour chariot
mov ah,06
int 21h
jmp debut
affiche  proc        ; procédure d'affichage d'un nombre BCD
mov dl,bl
and dl,0f0h        ; masquer pour avoir les dizaines
ror dl,1             ; faire une rotation
ror dl,1
ror dl,1
ror dl,1
add dl,30h          ; ajouter 30H pour avoir le code ASCII correct
mov ah,06           ; du chiffre a afficher
int 21h
mov dl,bl
and dl,0fh
add dl,30h
mov ah,06
int 21h
mov dl,':'
mov ah,06
int 21h ret
affiche endp
```

```
code ends
end prog
```

Fonctions Gestionnaire de souris

Int 33h, Fct 00h Réinitialisation du driver de souris Initialise le driver de souris et enclenche ainsi le test de la souris. Entrée :

AX = 0000h

Sortie :

AX = FFFFh : dans ce cas, un driver de souris est installé

BX = Nombre de boutons de la Souris (à partir de 1.0)

AX = 0000h : erreur, aucun driver de souris n'est installé

Remarques :

L'appel de cette fonction agit sur toute une série de paramètres de la souris qui peuvent être fixés à l'aide des différentes fonctions de la souris :

- Le curseur de la souris est placé au centre de l'écran puis caché. Il apparaîtra désormais en mode graphique sous forme d'une flèche, alors qu'il sera représenté en mode de texte sous forme d'une case de texte en inversion vidéo. Le curseur de la souris est systématiquement affiché dans la page 0 de l'écran, indépendamment du mode. La zone de déplacement sélectionnée est la totalité de l'écran.

- Les gestionnaires d'événements installés par un programme. Ils sont désactivés.

Int 33h, Fct 01h Afficher le curseur de la souris sur l'écran

Après appel de cette fonction, le curseur de la souris devient visible sur l'écran et suit désormais les déplacements de la souris sur l'écran.

Entrée :

AX = 0001h

Sortie :

Aucune

Remarques :

L'appel de cette fonction entraîne l'incrémentation d'un compteur interne, en fonction duquel le curseur de la souris est ou non affiché sur l'écran. Si ce compteur contient la valeur 0, le curseur de la souris sera affiché sur l'écran, alors

que la valeur -1 a pour effet de le faire disparaître de l'écran. Lorsque la fonction 00h (Reset) est appelée, ce compteur est tout d'abord fixé sur -1, pour prendre ensuite la valeur 0 lors du premier appel de cette fonction, ce qui se traduit par la réapparition du curseur sur l'écran.

Int 33h, Fct 03h Lire la position de la souris et l'état des boutons de la souris :

L'appel de cette fonction fournit la position actuelle de la souris et l'état des différents boutons de la souris.

Entrée :

AX = 0003h

Sortie :

BX = Etat des boutons de la Souris (à partir de 1.0) Bit Signification

0 1 = bouton gauche de la souris appuyé

1 1 = bouton droit de la souris appuyé

2 1 = bouton central de la souris appuyé

3-15 Aucune (0)

CX = Position horizontale de la Souris (à partir de 1.0) DX = Position verticale de la souris

Remarques :

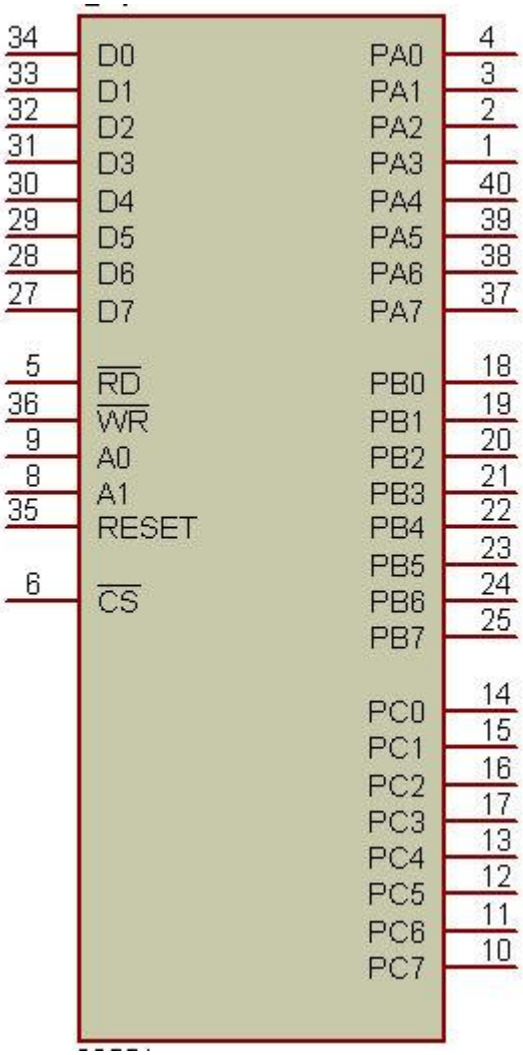
Les ordonnées renvoyées dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.

Si la souris ne possède que deux boutons, les informations fournies sur le bouton central sont dépourvues de signification

L'INTERFACE PARALLELE LE 8255A

I / Introduction :

Le 8255 est un circuit programmable de 40 broches (voir figure 1) d'interface d'entrée/sortie parallèle qui a été conçu pour travailler avec les microprocesseurs de la famille INTEL. Il est formé par trois ports d'entrées/sorties, chaque port est de 8 bits qui peuvent être programmés en entrée ou en sortie avec trois modes différents (mode zéro, mode un et mode deux).



Remarque :

Le port C est divisé en deux port C haut et port C bas

Le schéma bloque du 8255A est donné par la figure suivante :

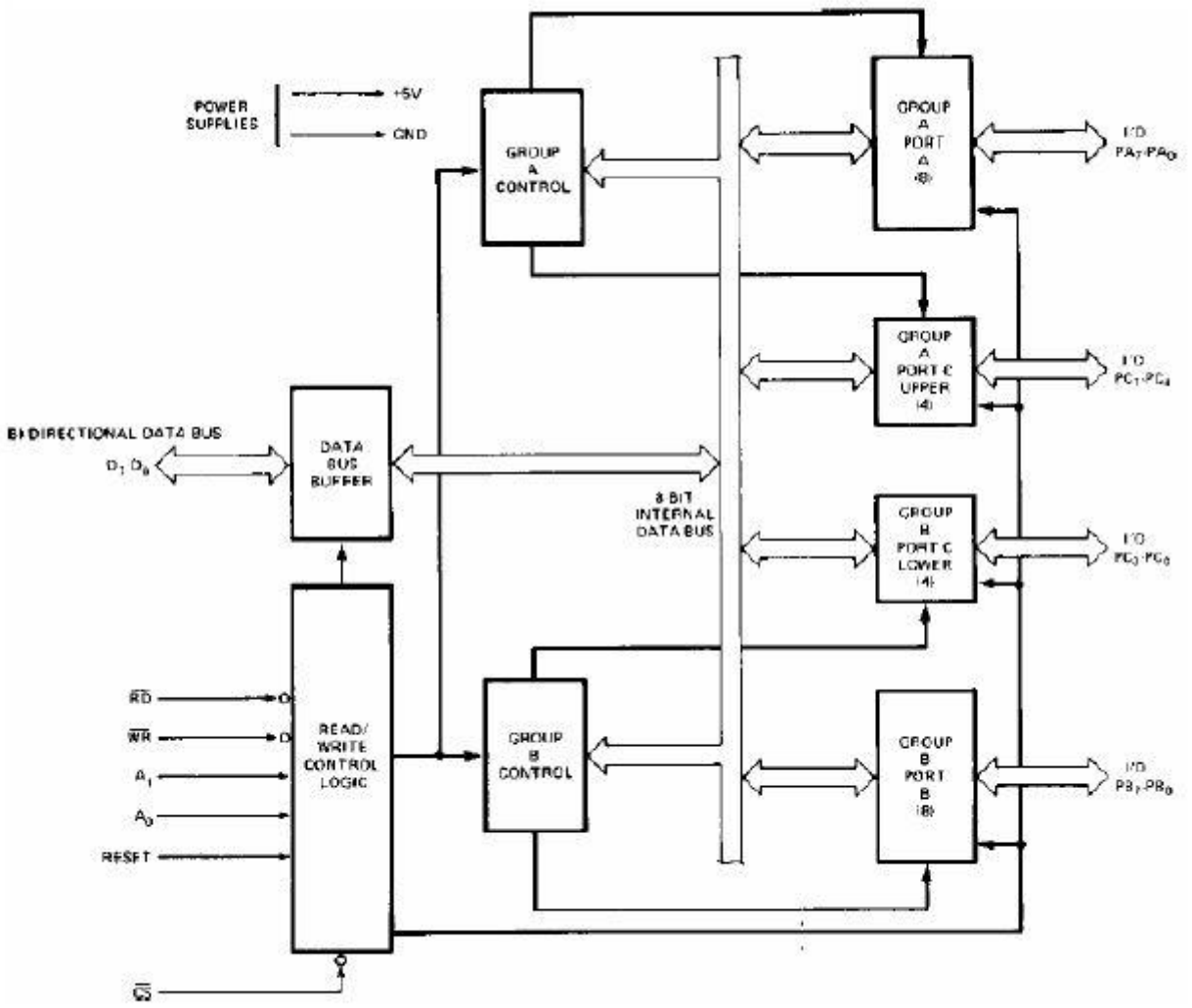


Schéma bloc du 8255A

Le schéma bloc du 8255 montre bien qu'il est divisé en deux groupes :

- Groupe A : formé par le port A et le port C haut.
- Groupe B : formé par le port B et le port C bas.

Le registre de données (Data buffer Bus) assure la liaison entre le bus de données extérieur et le registre de controle ainsi que les ports d'entrées/sorties.

La sélection du 8255A se fait par l'intermédiaire de la pine CS (qui est en général fournie par une logique de décodage qui permet la sélection du 8255A : voir plus loin les exemples d'applications)

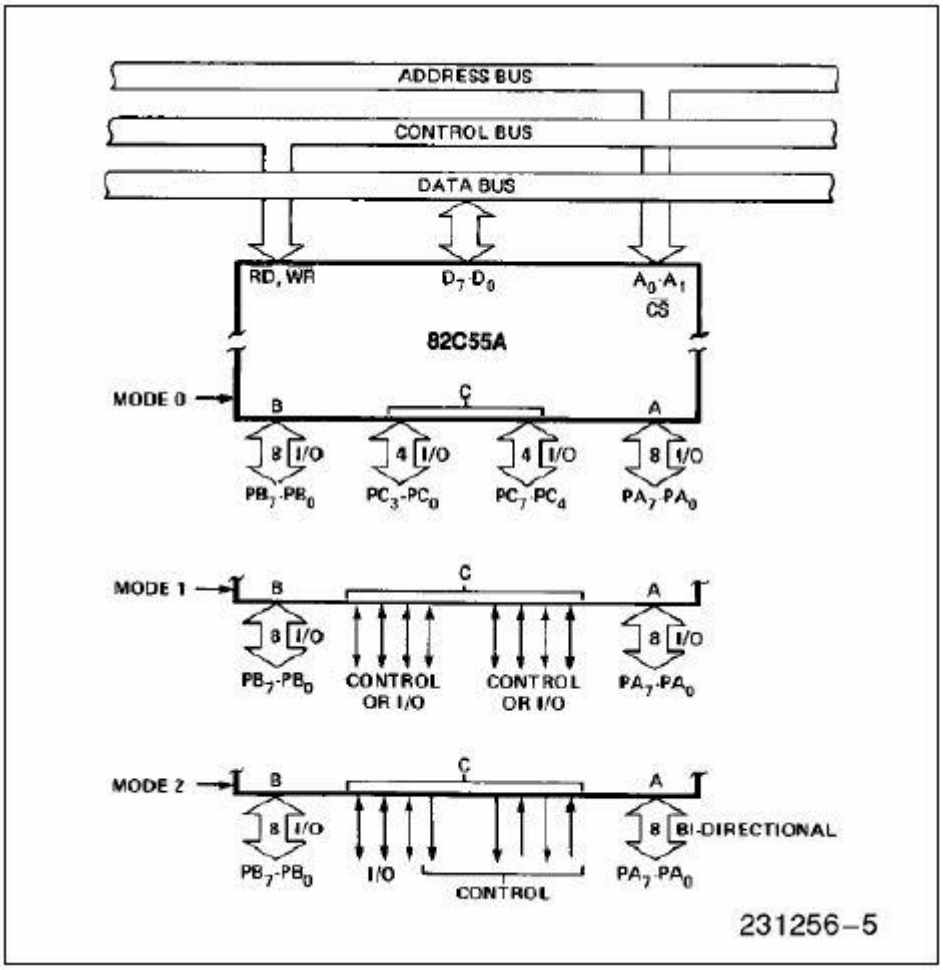
Le bus d'adresse du 8255A est formé essentiellement par deux pines (A0et A1) qui permettent de sélectionner les ports ainsi que le registre de contrôle comme le montre le tableau suivant :

A_1	A_0	Registre
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Registre de commande

Plus en détail l'adressage des différents ports en entrée sortie se fait selon la table de vérité suivante :

A_1	A_0	RD	WR	CS	Opération de lecture (Read)
0	0	0	1	0	Le bus de données = Port A
0	1	0	1	0	Le bus de données = Port B
1	0	0	1	0	Le bus de données = Port C
					Opération de écriture (Write)
0	0	1	0	0	Le bus de données = Port A
0	1	1	0	0	Le bus de données = Port B
1	0	1	0	0	Le bus de données = Port C
1	1	1	0	0	Le bus de données = reg_com
					Le 8255 hors fonctionnement
X	X	X	X	1	Bus de donnée troisième état
X	X	1	1	0	Bus de donnée troisième état

Le 8255 est connecté avec le système à base de microprocesseur comme le montre la figure suivante :



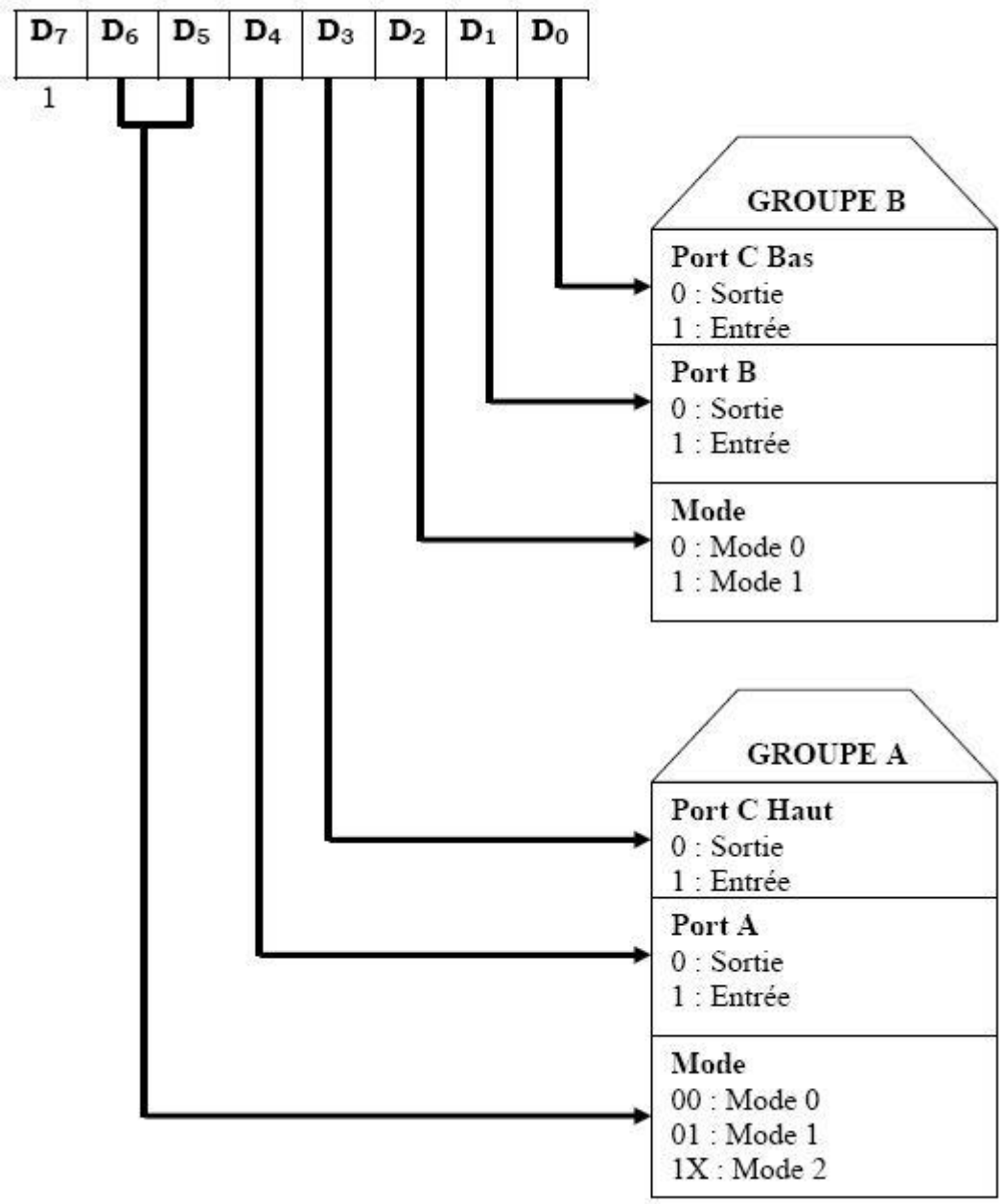
Interfaçage avec le bus et mode de fonctionnement du 8255A

II / Programmation du 8255A :

On peut programmer le 8255A selon trois modes :

- Mode 0 : Entrée/sortie de base.
- Mode 1 : Entrée sortie échantillonnée.
- Mode 2 : Bus bidirectionnel.

Le format ainsi que le choix des modes se fait à partir du mot de contrôle suivant :



II – 1 / Le mode 0 du 8255 :

En mode zéro les ports du 8255A peuvent être programmés en entrée ou en sortie : 8 bits pour le port A , 8 bits pour le port B et le port C est formé de deux quartes (un quarte haut et un quarte bas) , donc il y a 16 combinaisons possibles :

Mot de commande	PORT A	PORT B	PORT C	
			PC_4 à PC_7	PC_0 à PC_3
10000000	S	S	S	S
10000001	S	S	S	E
10000010	S	E	S	S
10000011	S	E	S	E
10001000	S	S	E	S
10001001	S	S	E	E
10011010	S	E	E	S
10001011	S	E	E	E
10010000	E	S	S	S
10010001	E	S	S	E
10010010	E	E	S	S
10010011	E	E	S	E
10011000	E	S	E	S
10011001	E	S	E	E
10011010	E	E	E	S
10011011	E	E	E	E

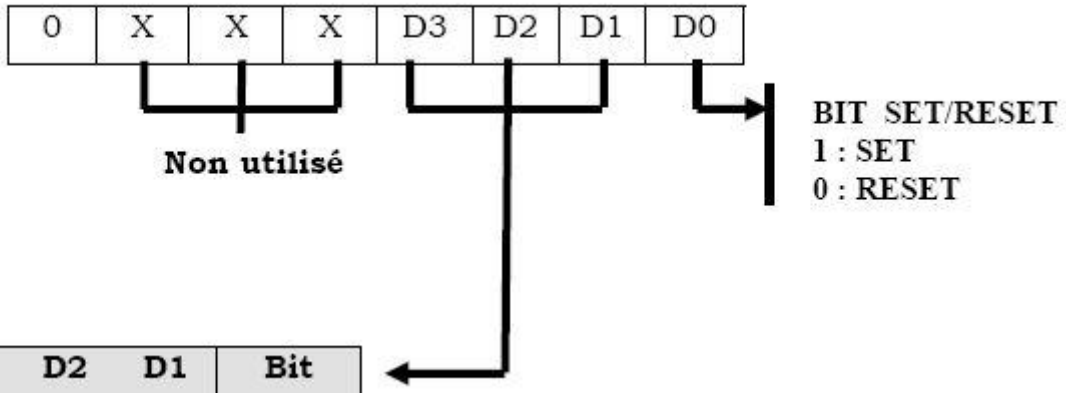
Exemple si on veut configurer le port A en sortie , port B en entrée , portc haut entrée et port C bas sortie le mot de commande est 93H

Le programme :

```
MOV  AL,  39H       ; Mot de commande
MOV DX , adresse_registre_de_commande
OUT  DX,AL   ; envoi du mot vers le registre de commande
```

Remarque :

- Les sorties sont mémorisées dans des bascules D.
- Les entrées ne sont pas mémorisées.
- Sur un RESET, tous les ports sont positionnés en mode entrées et il faut les reprogrammer, si nécessaire.
- Les bascules du port C peuvent être mises à 1 ou à 0 individuellement, lorsque le bit 7 du mot de commande est égal à 0 . Le mot de commande devient alors :



D3	D2	D1	Bit
0	0	0	PC_0
0	0	1	PC_1
0	1	0	PC_2
0	1	1	PC_3
1	0	0	PC_4
1	0	1	PC_5
1	1	0	PC_6
1	1	1	PC_7

- Pour mettre à 1 le PC3 par exemple le mot de commande est : 00000111
- Pour mettre à 0 le PC3 par exemple le mot de commande est : 00000110

II – 1 / Exemple d'application en mode 0 :

Exemple 1 :

On donne le schéma de la figure 1,

A/ on veut écrire un programme qui permet de faire clignoter les diodes Led . Jusqu'à l'appui sur SW0

B / On veut écrire un programme qui affiche les chiffres de 0 à 15 sur les 7 segments.

C / On veut écrire un programme qui permet de faire clignoter les diodes paires si on appuie sur SW0 et les diodes impaires si on appuie sur SW1

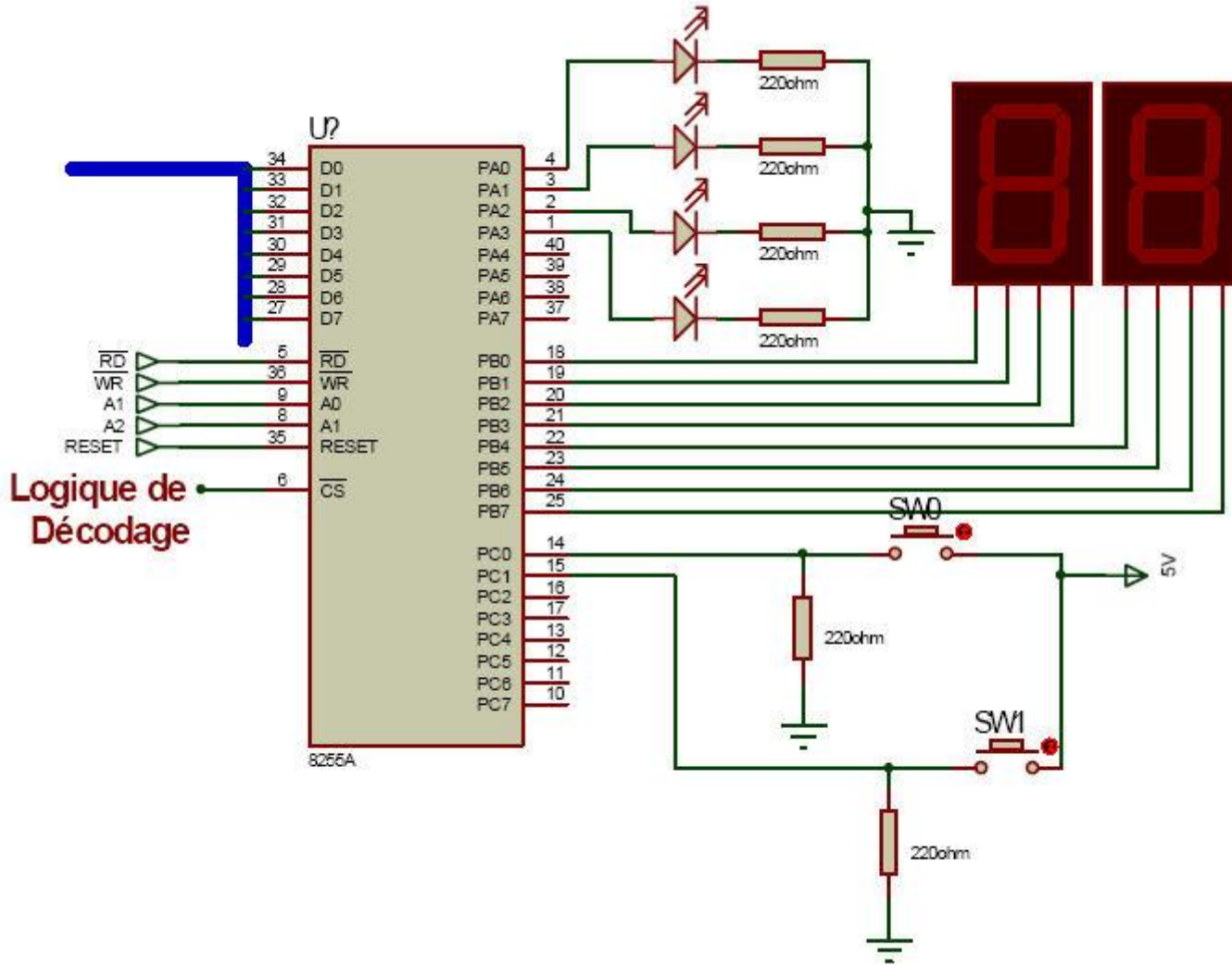


Figure 1

On suppose que les adresses des ports est comme suit :

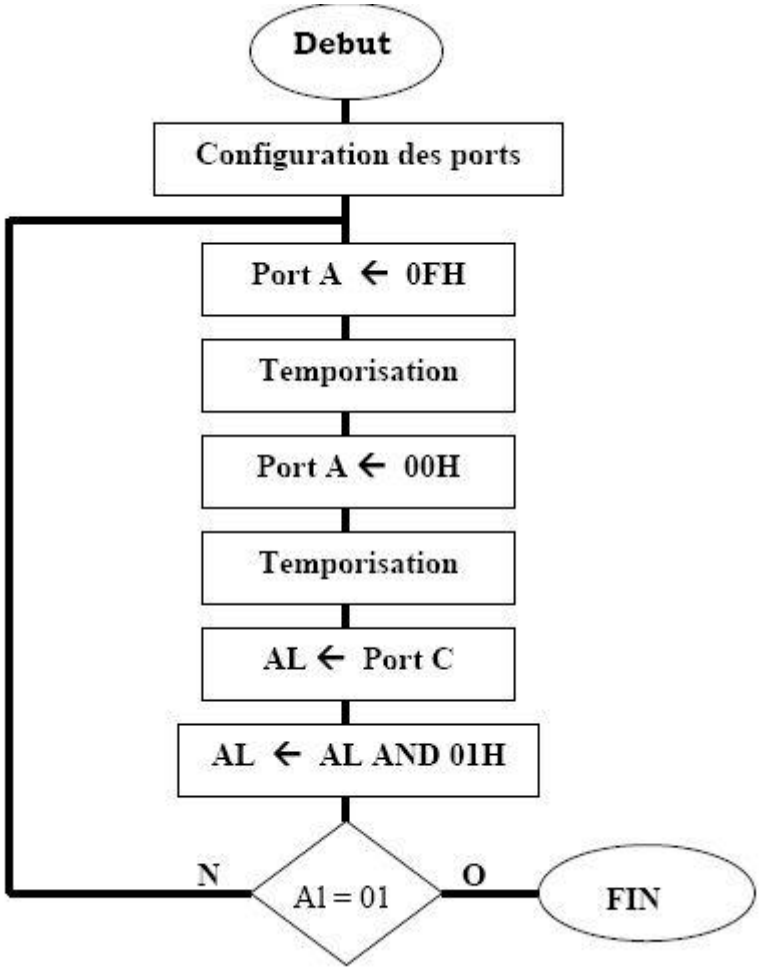
Port A : 300H

Port B : 302H

Port C : 304H

Registre de commande : 306H

A /Pour faire clignoter les diodes led il faut envoyer une fois 0FH puis effectuer une temporisation puis envoyer 00H et effectuer une temporisation voici l'organigramme qui assure ce fonctionnement :

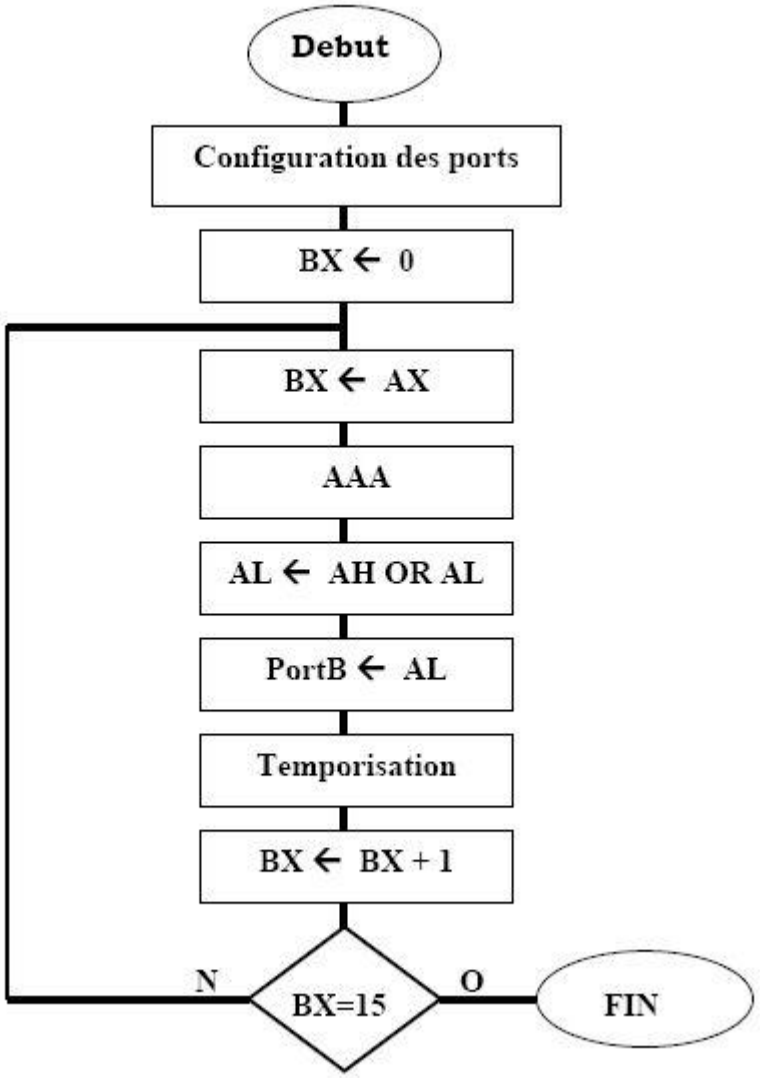


Le programme est le suivant :

```
Donnee SEGMENT
PortA  EQU   300H
PortC  EQU   300H
Reg_com  EQU   306H
Mot_com EQU   91H
Masque_SW0  EQU  01H
Diode_allume   EQU  0FH
Diode_etainte EQU   00H
Donnee ENDS
Code SEGMENT
Assume CS :code , DS :donnee
Prog Proc
       MOV AX,donnee       ; pointer le data segment
       MOV DS,AX
       MOV AL,Mot_com        ; configurer  les ports en sorties
       OUT  Reg_com,AL
Debut: MOV AL,Diode_allume   ; D3D2D1D0 = FH led allumé
       OUT PortA,AL
       CALL Tempo         ; temporisation
       MOV AL,Diode_etainte    ; D3D2D1D0 = 0H led etainte
       OUT PortA,AL
       CALL Tempo            ; temporisation
       IN AL,PortC                ; Lecture du portC
       AND AL,01H            ; verifier s'il a appui sur SW0
       CMP AL,01H
```

```
        JNZ Debut
        MOV AX,4C00H
        INT 21H
        Prog endp
Tempo: MOV CX,7FFFH         ; Effectuer une temporisation
Temp1: PUSh CX               ; avec deux boucles imbriqués
        MOV CX,7FFFH
Temp2: NOP
        NOP
        NOP
        NOP
        LOOP Temp2
        POP CX
        LOOP Temp1
        RET Code
        ends
End prog
```

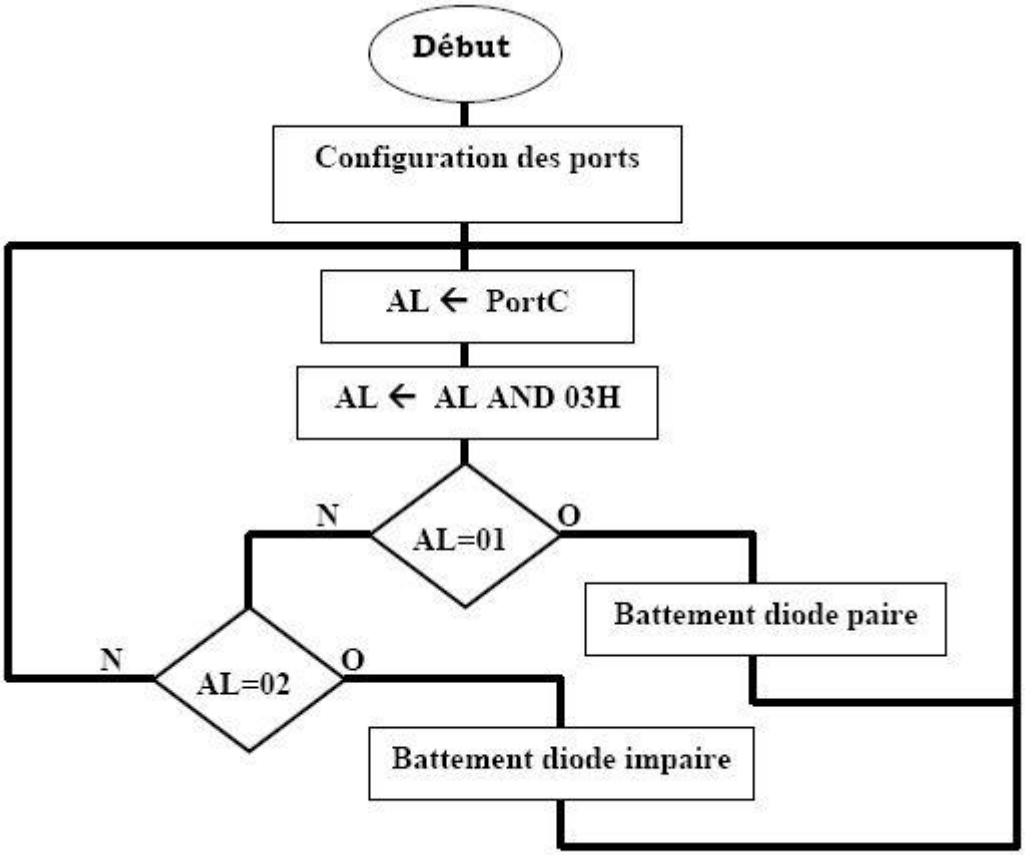
B / Programme qui affiche les chiffres de 0 à 15 sur le 7 segment : L'organigramme du programme est donné par la figure suivante :



Le programme qui réalise cette opération :

```
Donnee SEGMENT
PortA EQU  300H
PortB EQU  302H
Reg_com EQU 306H
Mot_com EQU 91H
Masque_SW0 EQU  01H
Diode_allume  EQU  0FH
Diode_eteinte EQU  00H
Donnee ENDS
Code SEGMENT
Assume CS :code , DS :donnee
Prog Proc
MOV AX,donnee       ; pointer le data segment
MOV DS,AX
MOV AL,Mot_com       ; configurer  les ports (A :S )
OUT  Reg_com,AL
XOR BX,BX
DEBUT: MOV AX,BX
AAA                  ; séparer les unités et les dizaines
OR AL,AH
OUT portB,al
CALL Tempo          ; effectuer une temporisation
INC BX
CMP BX,15           ; voir si BX = 15
JNZ DEBUT
MOV AX,4C00H
INT 21H
Prog endp
Tempo : MOV CX,7FFFH           ; Effectuer une temporisation
Temp1: PUSh CX              ; avec deux boucles imbriqués
MOV CX,7FFFH
Temp2: NOP NOP NOP NOP
LOOP Temp2
POP CX LOOP Temp1
RET
Code ends
End prog
```

C / L'organigramme du programme est donné par la figure suivante :



Remarque :

Le battement signifie qu'on allume la diode puis on l'éteint une seule fois car dans notre cas il faut toujours lire les switches car l'utilisateur peut changer l'état des switches à tout moment.

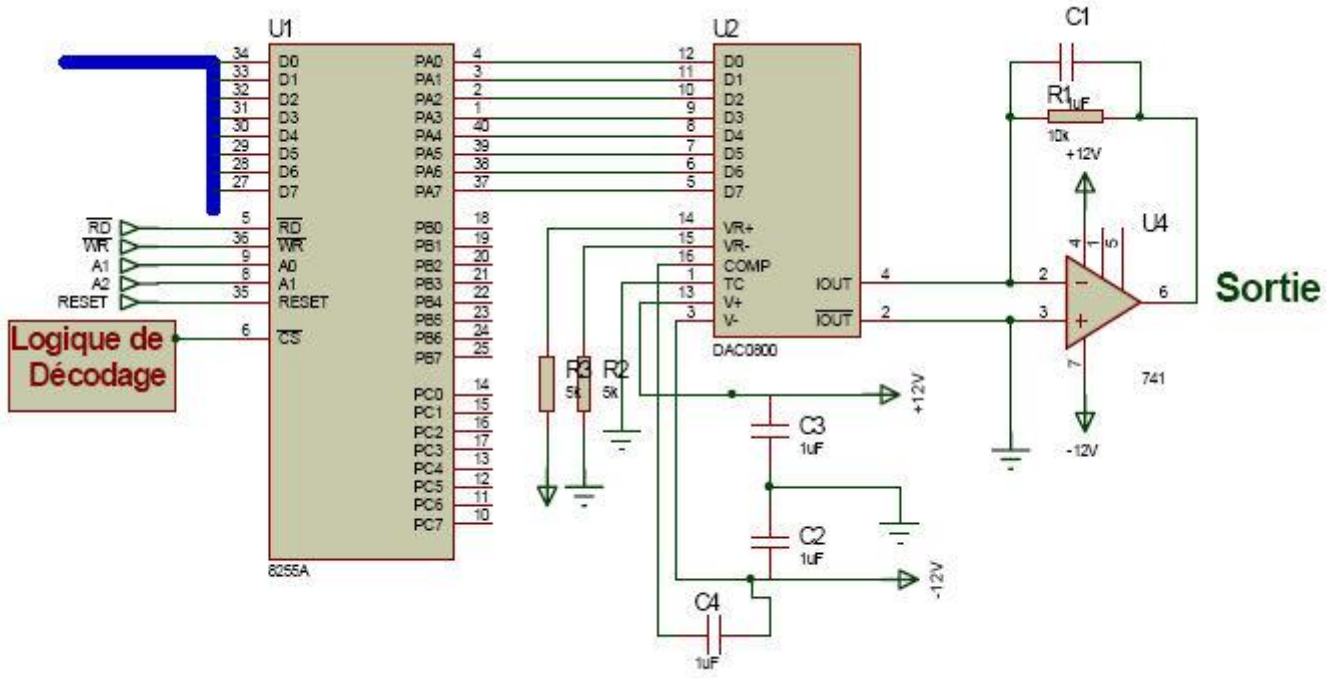
Le programme de l'application est le suivant :

```
Donnee SEGMENT
PortA  EQU 300H
PortC  EQU 302H
Reg_com EQU 306H
Mot_com EQU 91H
Masque_SW0 EQU  01H
Diode_allume  EQU  0FH
Diode_etainte EQU 00H
Donnee ENDS
Code SEGMENT
Assume CS :code , DS :donnee
Prog Proc
MOV AX,donnee          ; pointer le data segment
MOV DS,AX
MOV AL,Mot_com        ; configurer  les ports en sorties
OUT  Reg_com,AL
DEBUT: IN AL,PortC
AND AL,03H
CMP AL,01
JZ Diode_paire          ; si SW=1 alors battement paire
```

```
CMP AL,02
JZ Diode_impaire       ; si SW=1 alors battement impaire
JMP DEBUT
Diode_paire : MOV AL,05H          ; Battement des diodes paire
OUT PortA,AL
CALL Tempo                ; Temporisation
MOV AL,00H
OUT PortA,AL
CALL Tempo         ; Temporisation
JMP DEBUT
Diode_impaire : MOV AL,0AH         ; Battement des diodes impaire
OUT PortA,AL
CALL Tempo             ; Temporisation
MOV AL,00H
OUT PortA,AL
CALL Tempo                ; Temporisation
JMP DEBUT
MOV AX,4C00H
INT 21H
Prog endp
Tempo : MOV CX,7FFFH         ; Effectuer une temporisation
Temp1: PUSh CX           ; avec deux boucles imbriqués
MOV CX,7FFFH
Temp2: NOP
NOP
NOP
NOP
LOOP Temp2
POP CX
LOOP Temp1
RET
Code ends
End prog
```

Exemples 2 :

On donne le schéma de la figure suivante :



On suppose que les adresses des ports est donner comme suit :

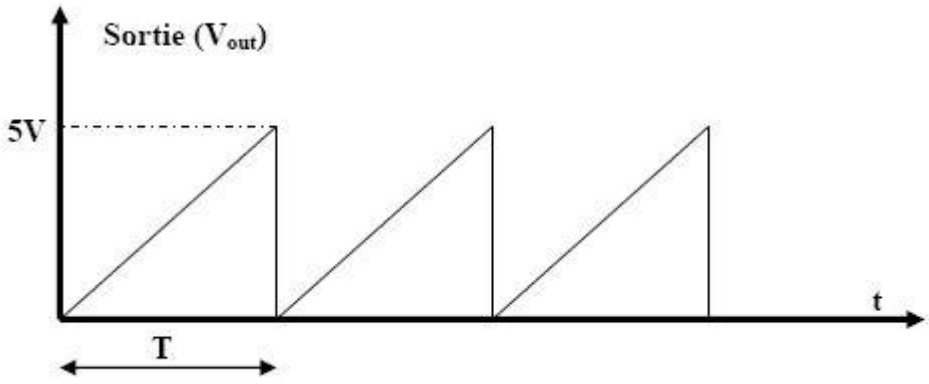
Port A : 300H

Port B : 302H

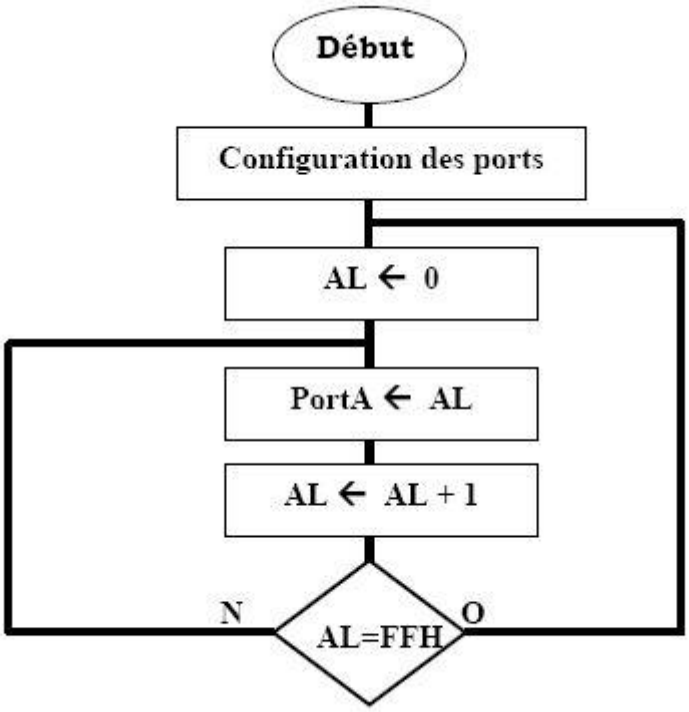
Port C : 304H

Registre de commande : 306

Donner l'organigramme ainsi que le programme qui permet de générer le signal suivant :



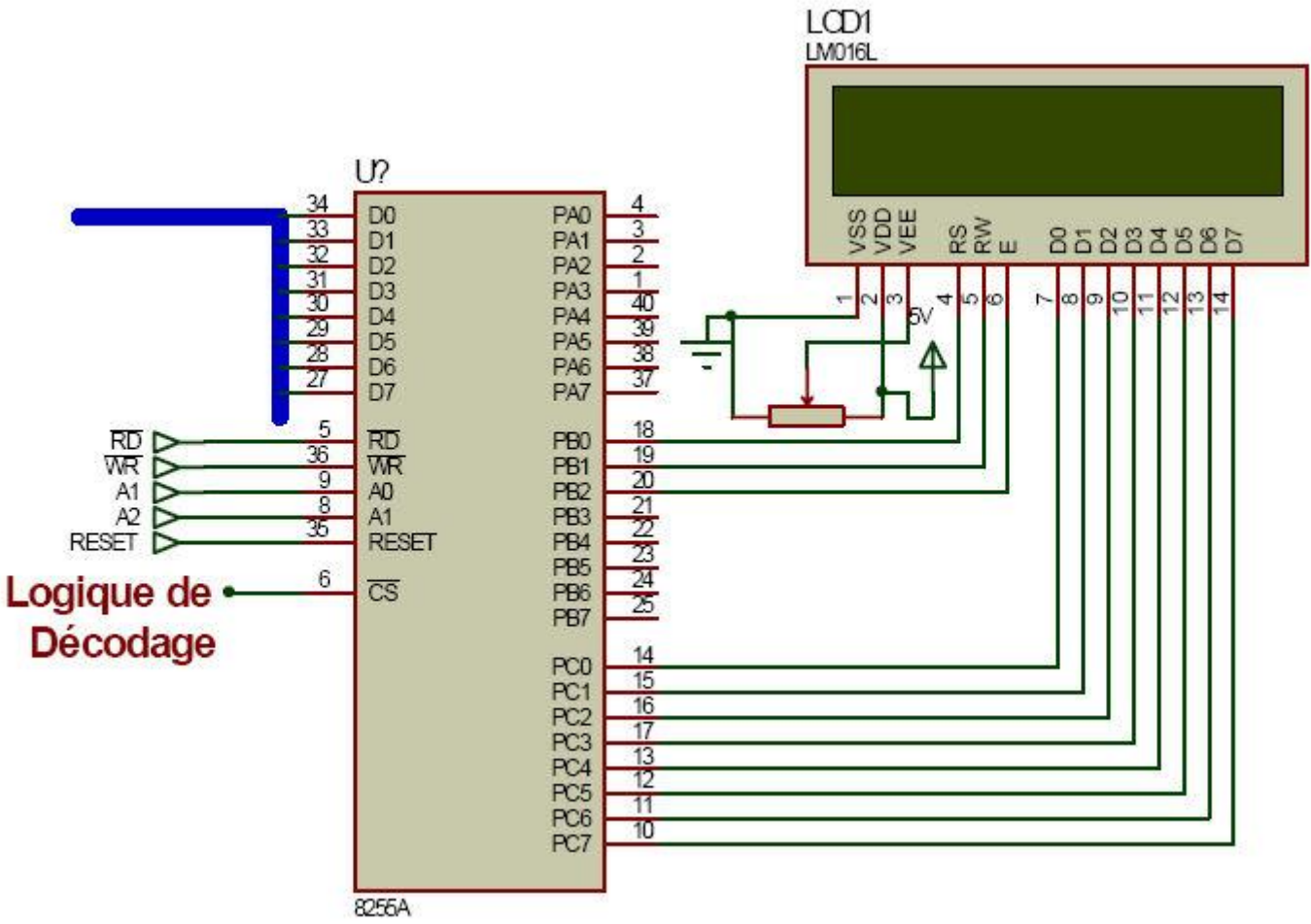
L'organigramme qui permet de générer le signal précédant est comme suit :




```
Donnee SEGMENT
PortA EQU  300H
Reg_com EQU 306H
Mot_com EQU 91H
Donnee ENDS
Code SEGMENT
Assume CS :code , DS :donnee
Prog Proc
MOV AX,donnee        ; pointer le data segment
MOV DS,AX
MOV AL,Mot_com      ; configurer  les ports en sorties
OUT  Reg_com,AL
DEBUT1: XOR AL,AL
DEBUT2:OUT PORTA,AL
CALL Tempo          ; Si on augmente la temporisation
INC AL         ; on augmente la pente du signal
CMP AL,0FFH         ; et bien sur la période T
JNZ DEBUT2
JMP DEBUT1
Prog endp
Tempo : MOV CX,7FFFH     ; Effectuer une temporisation
Temp1: PUSh CX         ; avec deux boucles imbriqués
MOV CX,7FFFH
Temp2: NOP
NOP
NOP
NOP
LOOP Temp2
POP CX
LOOP Temp1
RET
Code ends
End prog
```

Exemple 3 :

Pour commander un afficheur LCD 16*2 (16 caractères , 2 lignes) on propose le schéma du montage suivant :



Un LCD est formé essentiellement par un bus de donnée de 8 bits et un bus de commande et contrôle formé par trois pines(E, Rd/Rw, RS) :

E : entrée de validation un front descendant sur cette pine provoque la validation de la donnée ou de la commande.

RS : elle permet de distinguer les commandes et les données. RS = 0 le bus D0-D7 accepte des commandes RS = 1 le bus D0-D7 accepte des données

Enfin Rd/Rw : c'est pour donner l'ordre de lecture ou écriture sur LCD. Parmis les commandes on trouve :

Commande	Code hexa
Effacement LCD	01H
Home	0EH
Direction vers la droite	06H

Exemple de programme qui affiche le message 'bonjour iset n' sur LCD

Avant de commencer le programme il faut déterminer les mots qu'il faut envoyer au portB pour valider une donnée ou valider une commande d'où le tableau suivant :

	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0	HEXA
	X	X	X	X	X	E	R/W	RS	
Validation d'une commande	0	0	0	0	0	1	0	0	04H
	0	0	0	0	0	0	0	0	00H
Validation d'une donnée	0	0	0	0	0	1	0	1	05H
	0	0	0	0	0	0	0	1	01H

On suppose que les adresses des ports est donner comme suit :

Port A : 300H

Port B : 302H

Port C : 304H

Registre de commande : 306H

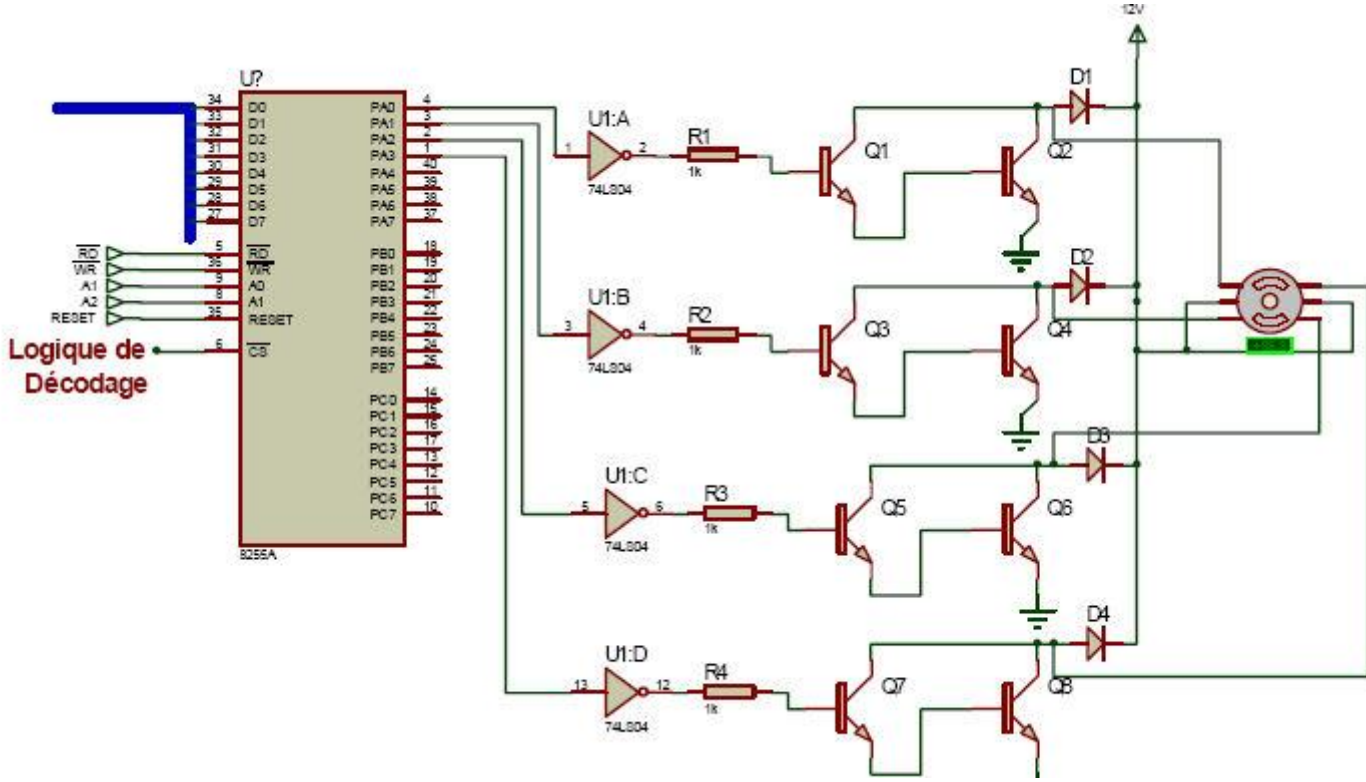
Le programme est comme suit :

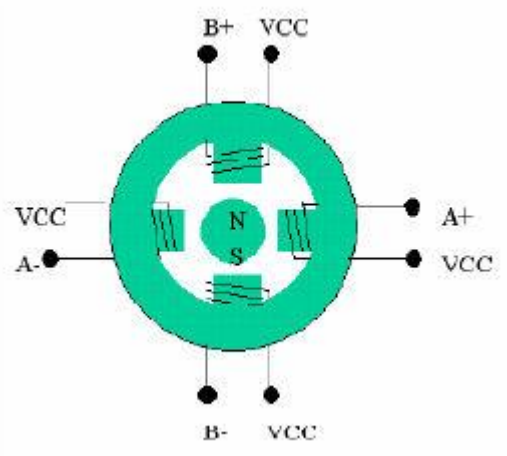
```
Donnee SEGMENT
Message db 'bonjour iset n'
PortB EQU  302H
PortC EQU  304H
Reg_com  EQU  306H
Mot_com  EQU   80H
Donnee ENDS
Code SEGMENT
Assume CS :code , DS :donnee
Prog Proc
MOV AX,donnee                  ; pointer le data segment
MOV DS,AX
MOV AL,Mot_com                ; configurer  les ports en sorties
OUT  Reg_com,AL
```

```
LEA SI,message                  ; Pointe le message à afficher
MOV CX,14           ; on 14 caractère à afficher
CALL Init_LCD     ; initialisation de l'LCD
DEBUT :
MOV AL,[SI]                           ; programme principal
OUT PortC,AL     ; envoie de la donnée
CALL Vali_donnée          ; validation de la donnée
INC SI
LOOP DEBUT
Init_LCD :  MOV AL,01    ; effacer LCD
OUT PortC,AL
CALL Vali_commande      ; Validation de l'effacement
MOV AL,0EH                        ; effectuer un Home pour LCD
OUT PortC,AL
CALL Vali_commande      ; Valide le HOME
MOV AL,06                           ; Ecriture vers la droite
OUT PortC,AL
CALL Vali_commande      ; valide le mode d'écriture.
RET
Vali_commande :
MOV AL,04                  ; Sous programme validation
OUT PortB,AL                      ; de la commande
MOV AL,00
OUT PortB,AL
RET
Vali_donnée :
MOV AL,05                        ; Sous programme validation
OUT PortB,AL                      ; de la donnée
MOV AL,01
OUT PortB,AL
RET
MOV AX,4C00H
INT 21H
Prog endp
```

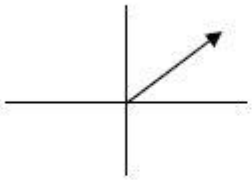
Exemple 4 : Commande d'un moteur Pas à Pas

On donne le schéma de la figure suivante :

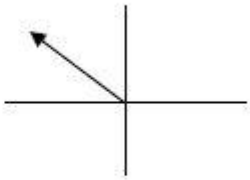




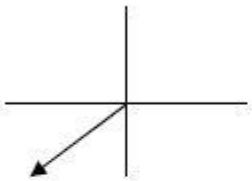
Les différentes phases sont les suivantes (en mode avancement par un pas):



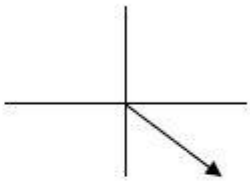
Phase 1



Phase 2

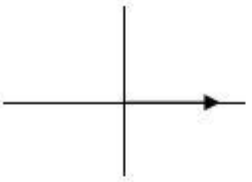


Phase 3

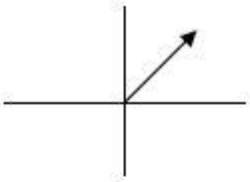


Phase 4

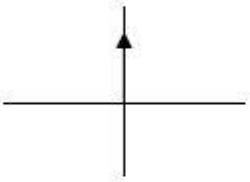
Les différentes phases sont les suivantes (en mode avancement par demi pas):



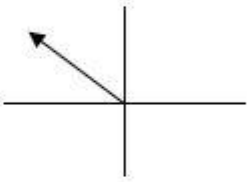
Phase 1



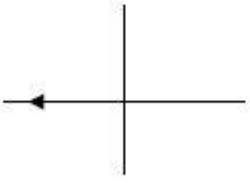
Phase 2



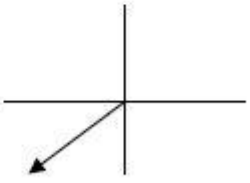
Phase 3



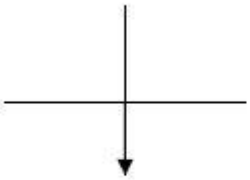
Phase 4



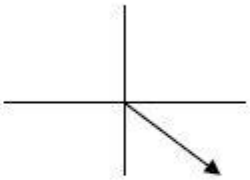
Phase 5



Phase 6



Phase 7



Phase 8

La commande des phases du moteur pas à pas bipolaire a aimant permanent est donnée par le tableau suivant :

- pas complet

Phase	1	2	3	4	5	6	7	8
Phase 1 (A+)	1	0	0	1	1	0	0	1
Phase 2 (B+)	1	1	0	0	1	1	0	0
Phase 3 (A-)	0	1	1	0	0	1	1	0
Phase 4 (B-)	0	0	1	1	0	0	1	1

- Demi pas

Phase	1	2	3	4	5	6	7	8
Phase 1 (A+)	1	0	0	0	1	0	0	0
Phase 2 (B+)	0	1	0	0	0	1	0	0
Phase 3 (A-)	0	0	1	0	0	0	1	0
Phase 4 (B-)	0	0	0	1	0	0	0	1

D’où le programme par exemple qui fait tourner le moteur pas à pas de 90° : On suppose que les adresses des ports est donner comme suit :

Port A : 300H

Port B : 302H

Port C : 304H

Registre de commande : 306H

1 pas = 0.9°

A+ est connecté avec PA0

B+ est connecté avec PA1

A- est connecté avec PA2

B- est connecté avec PA3

Le programme est comme suit :

```
Donnee SEGMENT
Phase db 03H, 06H, 0CH, 09H, 03H, 06H, 0CH, 09H
PortA   EQU   300H
Reg_com EQU   306H
Mot_com EQU   80H
Donnee ENDS
Code SEGMENT
Assume CS : code, DS :donnee
Prog Proc
MOV AX,donnee                  ; pointer le data segment
MOV DS,AX
MOV AL,Mot_com                ; configurer  les ports en sorties
OUT  Reg_com, AL
MOV CX, 100                         ; 100 * 0.9° = 90°
DEBUT1 :   LEA SI, phase            ; Pointer sur le tableau des phases
DEBUT2 :   MOV AL,[SI]
OUT portA, AL                       ; Envoie des différentes phases
INC SI
CMP SI,7
JNZ DEBUT2
LOOP DEBUT1
MOV AX, 4C00H                  ; Retour au DOS INT 21H
Prog endp
Code ENDS
END PROG
```

II – 2 / Le Mode 1 du 8255A :

En mode 1 , les ports A et B sont utilisés en entrée ou en sortie gérées par les accès du portC .

Mode 1 : en entrée :

En entrée les ports A et B sont configurés en entrée alors que le PC0, PC1 et Pc2 assure le handshake pour B et PC3, PC4 et Pc5 pour le A (PC6 et PC7 peuvent être programmés en entrée ou en sortie). STB : (Strobe input) : Au niveau bas , charge

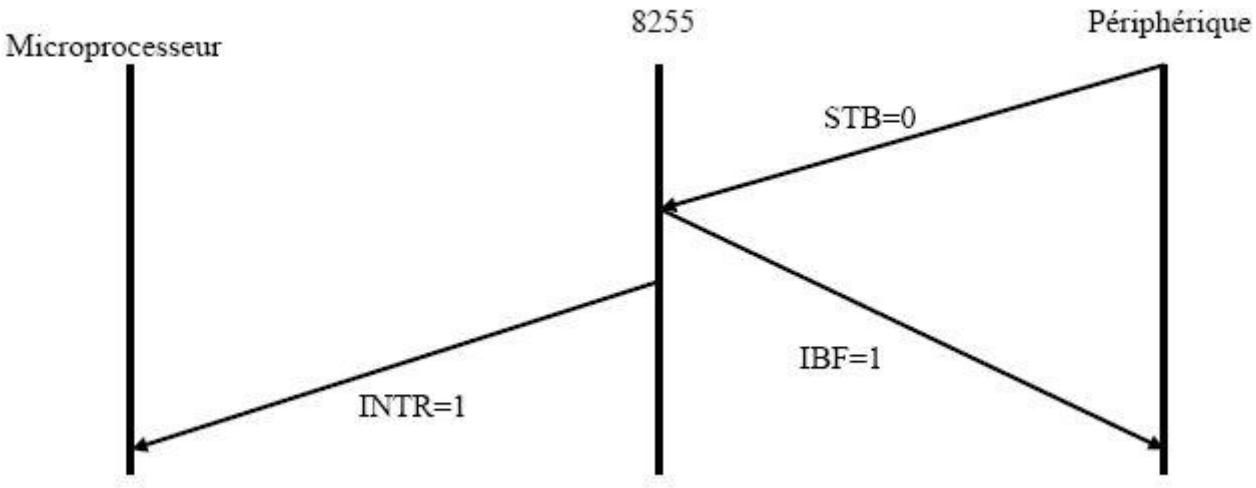
les données dans le verrou d'entrée . IBF : (Input Buffer Full) : Au niveau haut les données ont été verrouillées :

- IBF = 1 par STB au niveau 0
- IBF = 0 par le front montant d'un RD (read) INTR : (Interrupt Request) : Au niveau haut : demande d'interruption vers le microprocesseur :
- INTR = 1 par STB=1, IBF=1 et INTE=1
- INTR =0 par le front descendant d'un RD (read)

Remarque :

Le signal INTE est contrôlé par le mot de commande du portC (PC4 pour le Port A et PC2 pour le Port B)

Le handshak en entrée est résumé par cette figure :

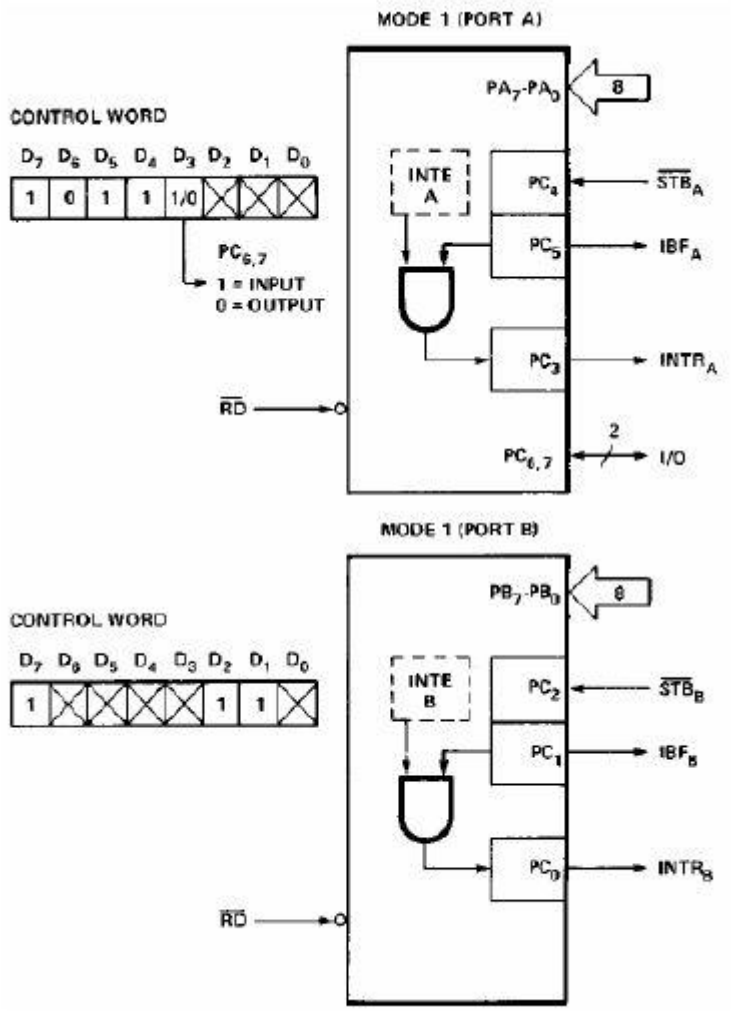


Au départ la périphérie charge les données dans les verrous d'entrée du 8255 (STB=0) puis le 8255 répond au périphérique par : chargement fait (IBF=1). Enfin le 8255 peut interrompre le fonctionnement du microprocesseur pour lui signaler qu'il a des données à prendre (INTR=1)

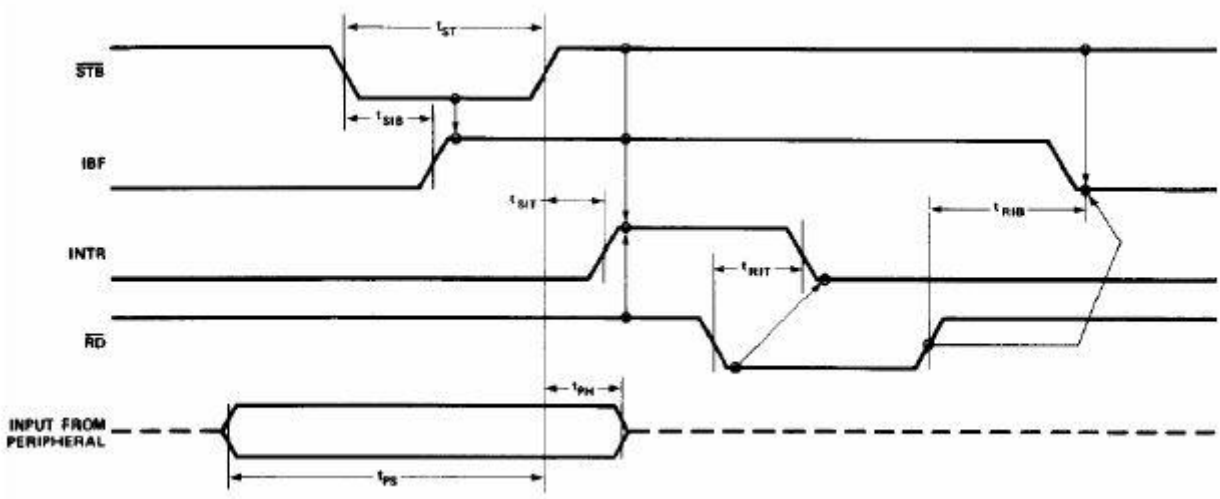
La figure suivante illustre ce mode de fonctionnement pour les portA et portB avec leur mot de commande respectif.

Remarque :

On ne trouve plus dans ce cas l'organisation symétrique du portc (c,a,d portc haut et portc bas)



Le chronogramme de fonctionnement dans ce mode est donné par la figure suivante :



Mode 1 : en sortie :

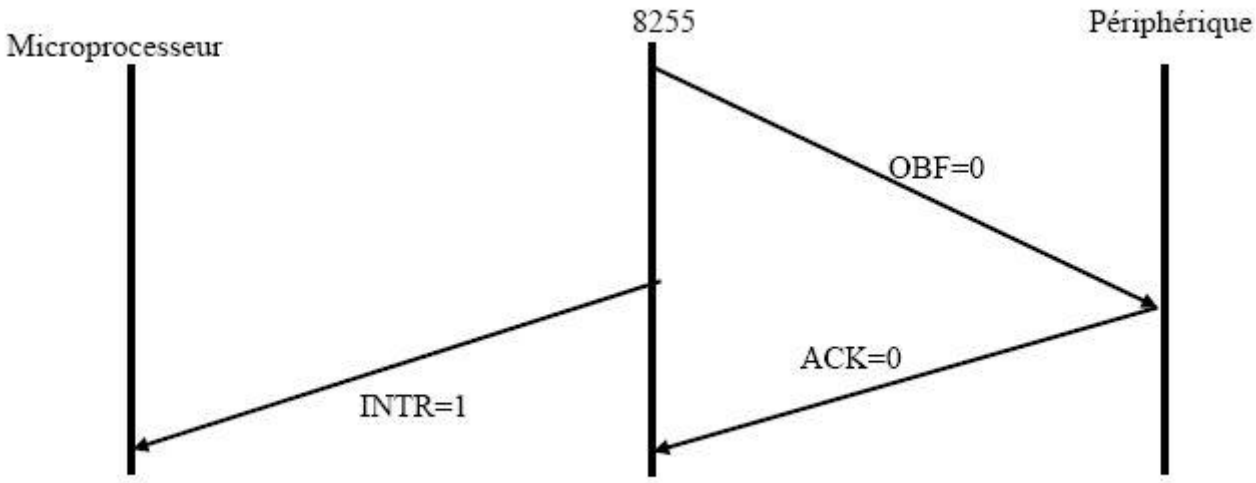
OBF : (Output Buffer Full) : Devient égale à 0 quand le microprocesseur a écrit des données dans le port, et elle revient à 1 sur le front montant de WR et à 0 sur ACK à zéro.

ACK : (Acknowledge Input) : = 0 signifie que les données ont été acquises.

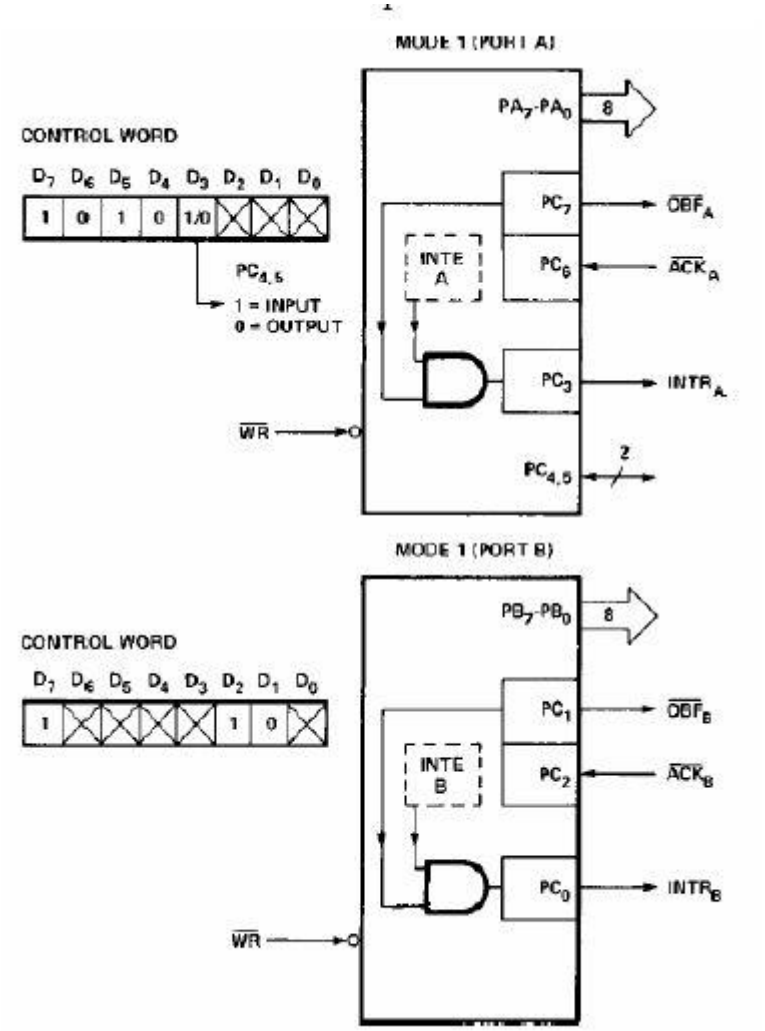
INTR : (Interrupt Request) : =1 demande d'interruption vers le microprocesseur :

- Mise à 1 par ACK, avec OBF=1 et INTE=1
- Mise à zéro par le front descendant de WR

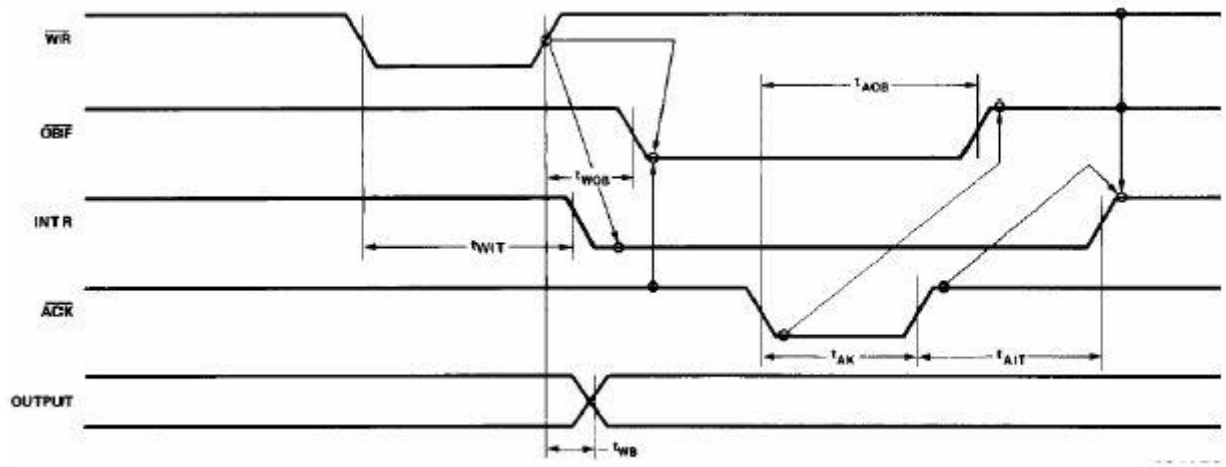
Donc on peut schématisé le transfert par la figure suivante :



La figure suivante illustre ce mode de fonctionnement pour les portA et portB avec leur mot de commande respectif.

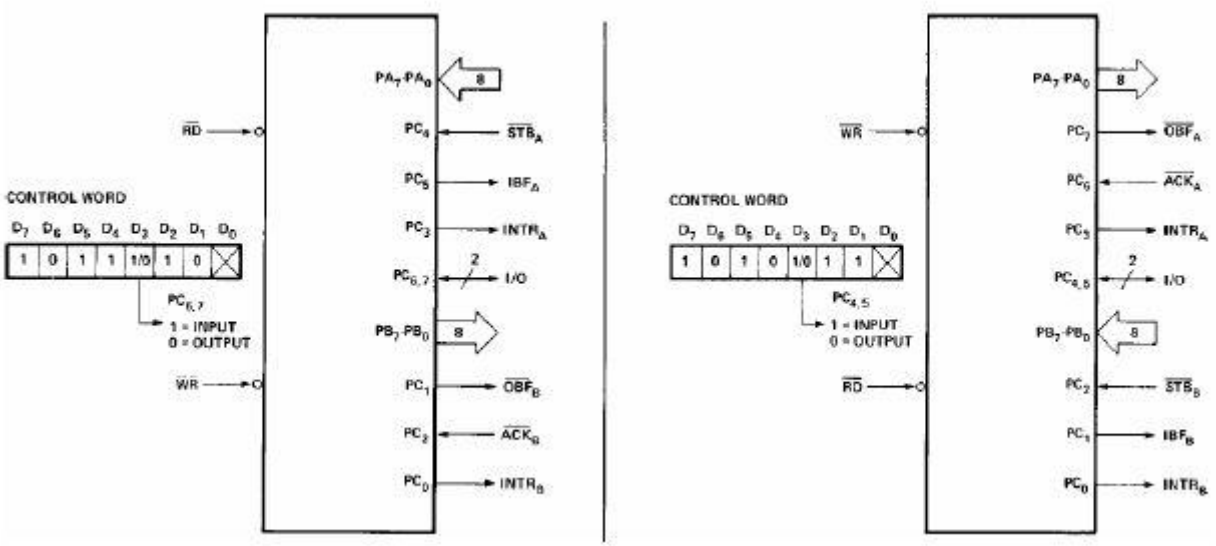


Le chronogramme de fonctionnement dans ce mode est donné par la figure suivante :



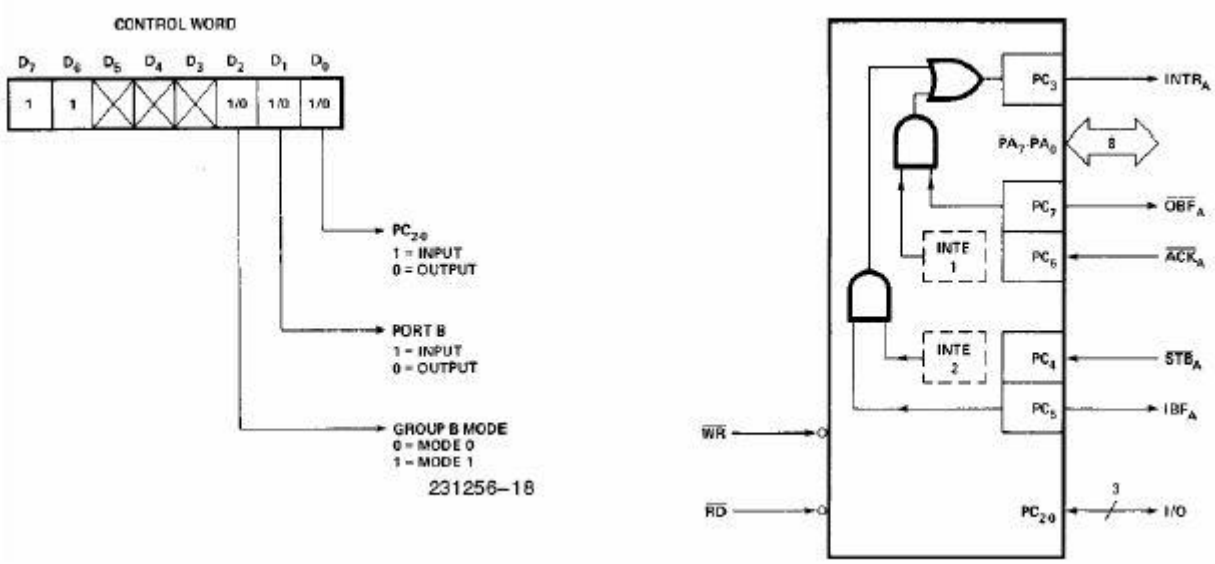
Remarque :

Le port A ainsi que le Port B peuvent être mis en entrée ou en sortie individuellement dans le mode 1 : c'est le mode mixte



II – 3 / Le Mode 2 du 8255A :

Le mode 2 ne s'applique qu'au port A. il permet de crée un bus bidirectionnel sur le port A sur 8 bits, 5 bits du portC sont utilisés pour le statut et le contrôle du portA (permette un handshak similaire au mode 1).



Commande du Bus bidirectionnel :

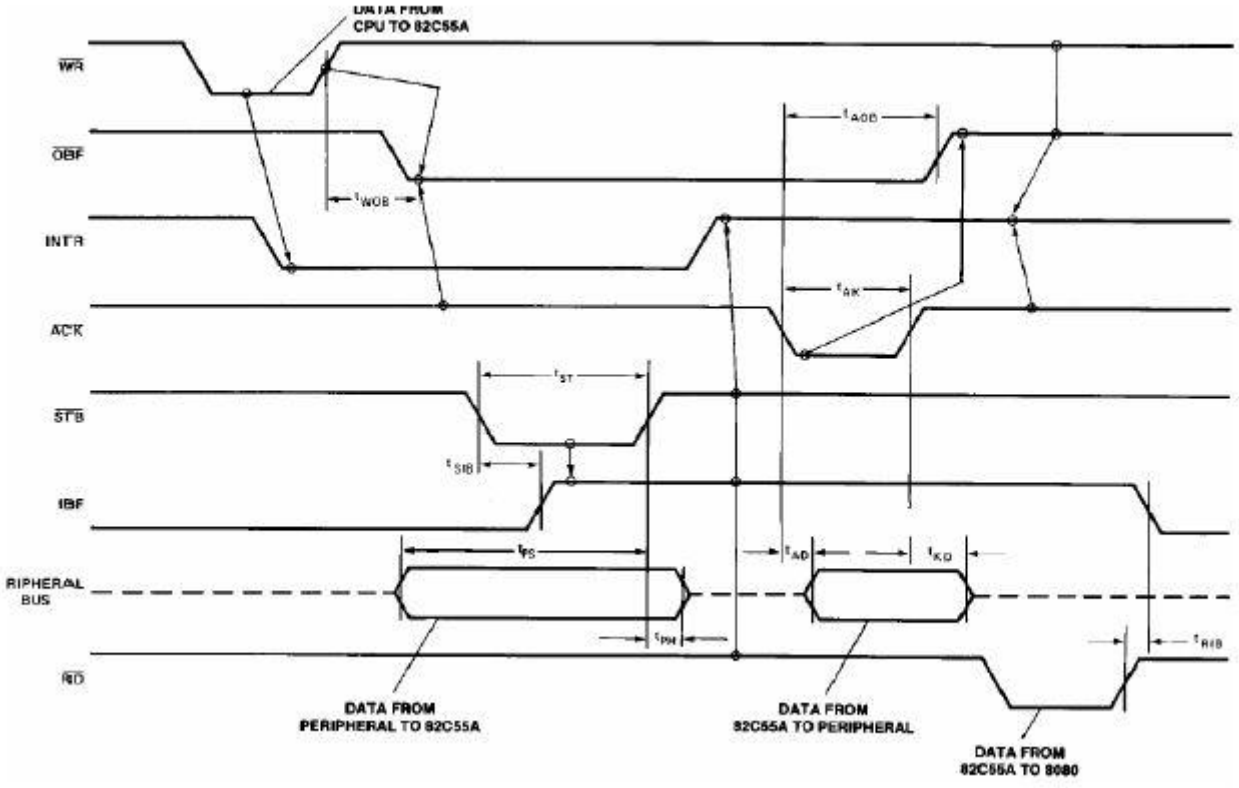
- INTR : (Interrupt Request : E/S) : Demande d'interruption (au niveau haut).
- OBF : (Output Buffer Full : S) :

i. Niveau bas : le buffer de sortie est autorisé à émettre la donnée.

ii. Niveau haut : le buffer de sortie est au 3eme état

- STB : (Strobe Input : E) : Niveau bas : les données sont chargées dans le verrou.
- IBF : (Input Buffer Full : E) : Haut : les données ont été chargées.
- INTE 1 : (Interrupt 1 : S) : Le flip-flop associé à OBF est contrôlé par PC6.
- INTE 2 : (Interrupt 2 : E) : Le flip-flop associé à IBF, contrôlé par

PC4.

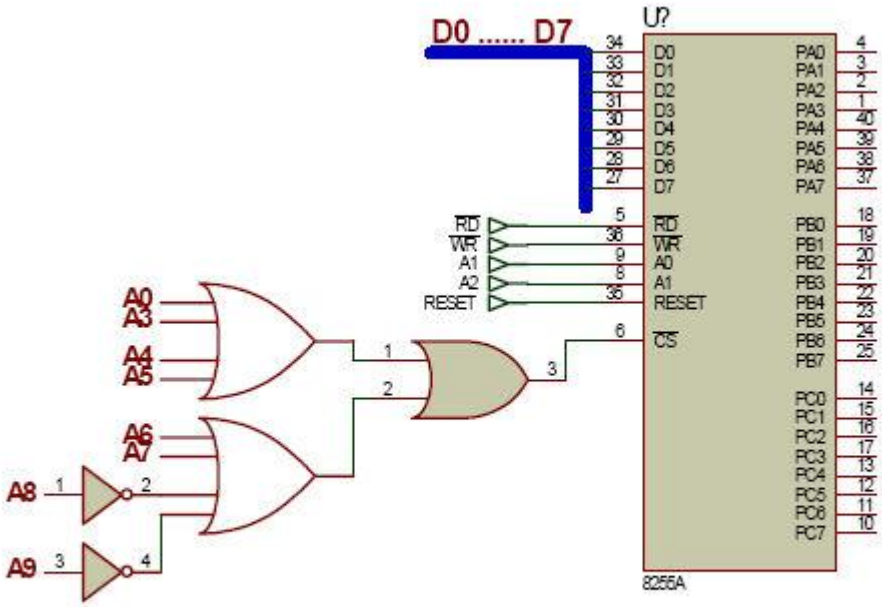


III / Connections du 8255 avec le microprocesseur 8086 :

Pour connecter une interface de 8 bits (le 8255A) avec un bus de données de 16 bits (Le 8086) il faut prendre des précautions en effet on a vu dans le paragraphe « organisation physique de la mémoire » qu'avec le microprocesseur 8086 si on accède à une adresse paire les données serons transmises sur le bus de données D0 – D7 mais si on accède à une adresse impaire les données seront transmises sur D8 – D15, Or avec la mémoire on veut que le Bank 0 est connecte au signal BHE et le bank1 est lié avec A0 (pine d'adresse). Dans le cas du 8255 on a le même problème qu'il faut résoudre lorsque on veut connecter cette interface avec un microprocesseur 8086.on prenons en considération le tableau suivant :

$\overline{BHE}$	A_0	Données
1	0	Octet d'adresse paire sur $D_7 - D_0$
0	1	Octet d'adresse paire sur $D_{15} - D_8$

- 1er solution : utilisation des adresses paires : c'est-à-dire que les pines A1 et A2 du microprocesseur seront connecté respectivement au pine A0 et A1 du 8255. Le schéma de la solution est donné comme suit :

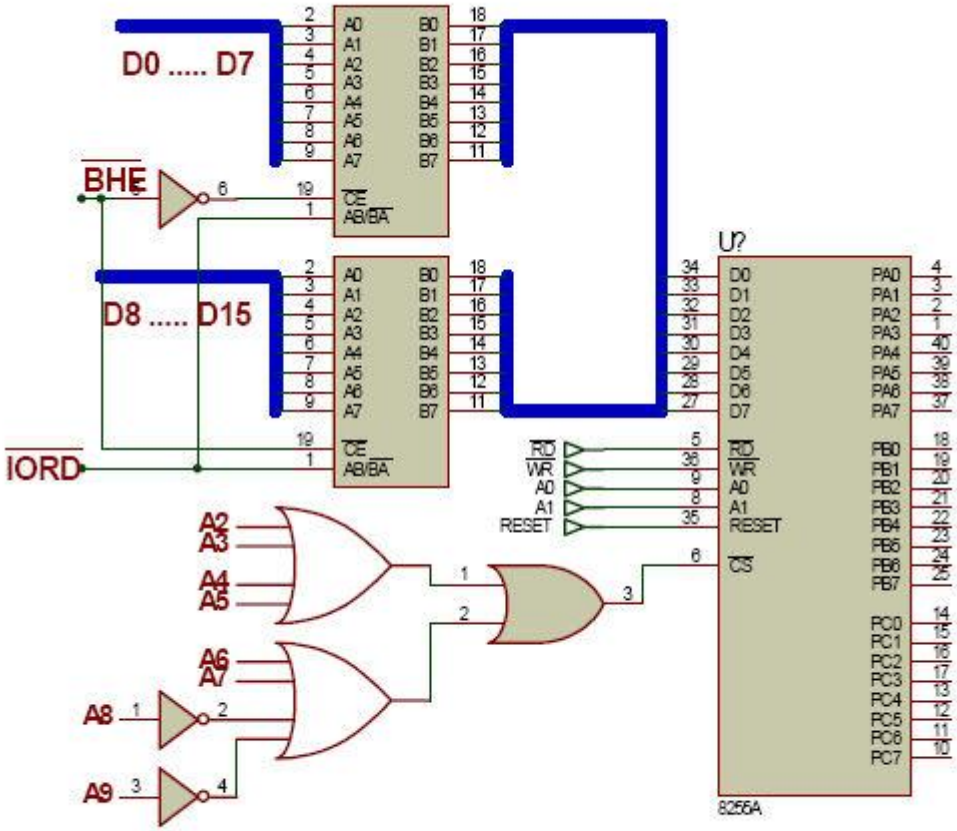


Donc les adresses des ports :

PORT	$A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$	HEXA
PORTA	1 1 0 0 0 0 0 0 0 0	300H
PORTB	1 1 0 0 0 0 0 0 0 0	302H
PORTC	1 1 0 0 0 0 0 0 0 0	304H
Reg_Com	1 1 0 0 0 0 0 0 0 0	206H

- 2eme solution : utilisation des adresses paires et impaires

On utilise les adresses paires et impaires mais à ce moment il faut mettre deux buffer bidirectionnels tels que le 8286 ou encore 74245. Le schéma est donné alors comme suit :



Donc les adresses des ports :

PORT	$A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$	HEXA
PORTA	1 1 0 0 0 0 0 0 0 0	300H
PORTB	1 1 0 0 0 0 0 0 0 1	301H
PORTC	1 1 0 0 0 0 0 0 2 0	302H
Reg_Com	1 1 0 0 0 0 0 0 1 1	203H

L'interface série le 8250/16550

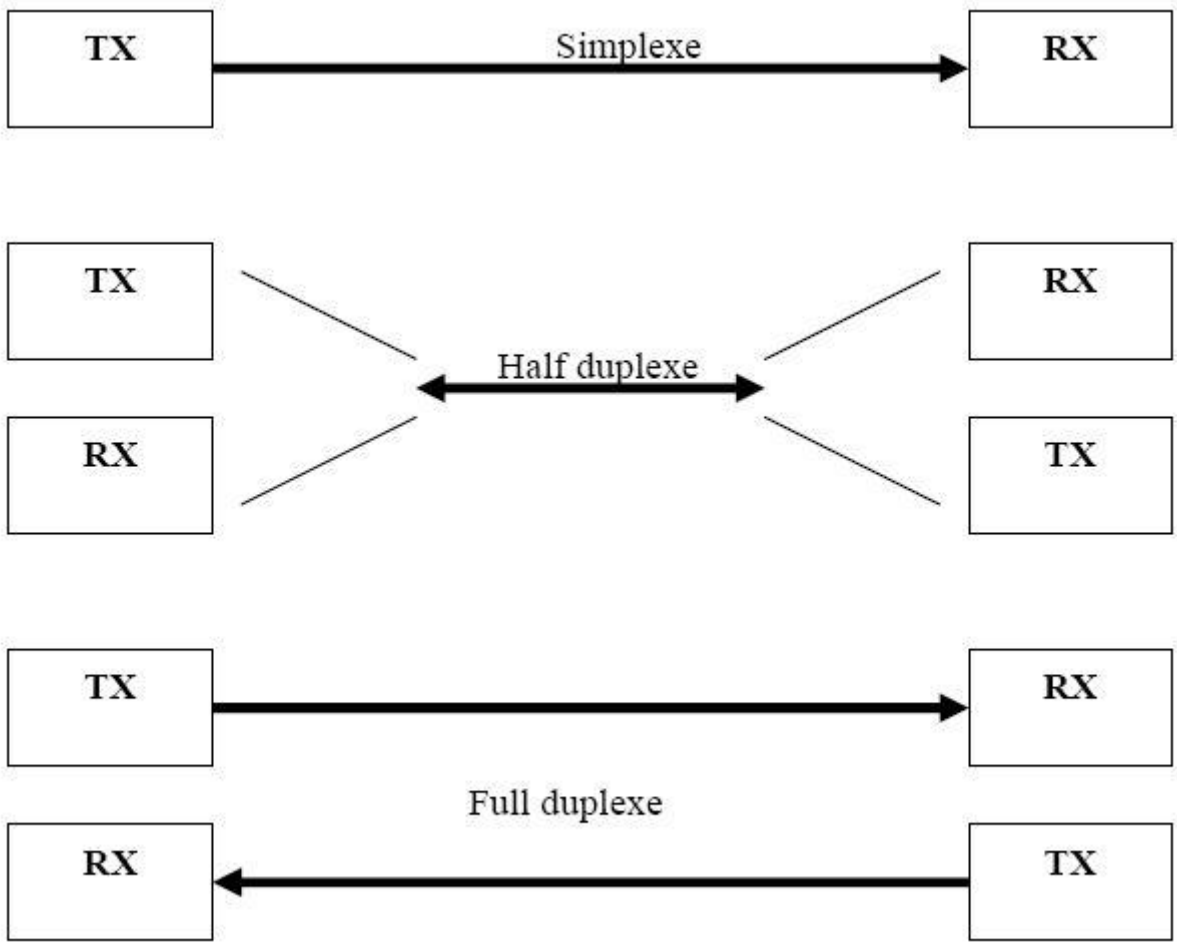
I / Introduction :

Malgré sa rapidité la liaison parallèle n'est pas adapter à quelque application lors de la communication entre le microprocesseur et ses périphériques, en effet ce type de liaison a des faiblesses lorsqu'il s'agit de communiquer avec un organe pour une longue distance, à cet effet on préfère utiliser la liaison série malgré son faible débit contre la liaison parallèle.

La liaison entre l'émetteur et le récepteur en mode série peut être :

- Simplexe.
- Half duplexe
- Full duplexe

La figure suivante montre la différence entre les trois modes :



TX : ligne d'émission. RX : ligne de réception.

Remarque :

La vitesse de transmission est mesurée en bps (bits par seconde), mais aussi BAUD (nombre des changements du signal par Seconde). Il existe 2 types de Communications Série: Synchrone et Asynchrone :

I-1 / Communications asynchrones

Ici la transmission s'effectue caractère par caractère.

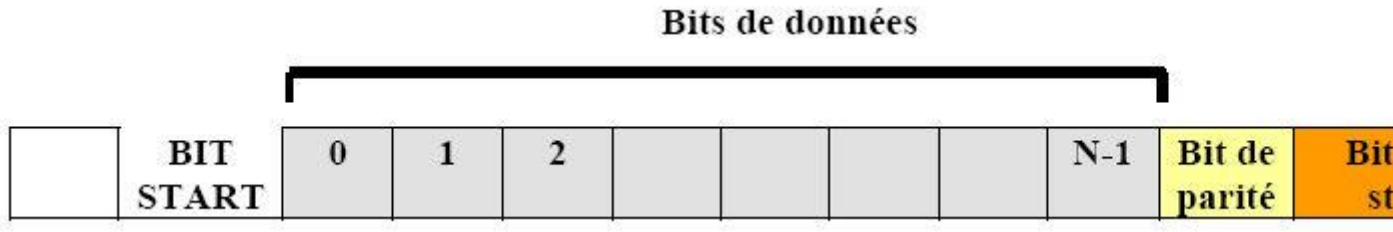
Un caractère comprend :

- un bit de départ (START),

- des bits de donnée selon un format variable (5, 6, 7,8 bits),

- un bit de parité (optionnel),

- un ou plusieurs bits d'arrêt (Stop).



La cadence de transmission est fixée par une horloge dont la période donne le temps de transmission de 1 bit.

Au repos, la ligne de transmission est à l'état 1. Le début de transmission est signalé par l'apparition du bit de départ qui est dans l'état logique 0 (START). Les bits de donnée sont ensuite transmis séquentiellement. Un bit de parité est ajouté éventuellement à la suite des bits de donnée pour vérifier la bonne transmission des données en effet il permet de détecter des erreurs de transmission. La fin de transmission du caractère est signalée par un ou plusieurs bits d'arrêt à l'état 1 (bits STOP). Le bit de départ permet au récepteur de détecter l'arrivée d'un caractère et de recaler la phase de l'horloge de réception afin d'échantillonner correctement les bits suivants (au milieu de chacun des bits). Ainsi, comme les fréquences d'émission et de réception ne sont pas strictement identiques, il suffit de garantir une précision de quelques pour-cent pour assurer une réception valide. Les vitesses normalisées sont : 50, 75, 110, 150, 300, 600, 1200, 2400,4800, 9600, 19200, 38400 bauds.

I-2 / Communication synchrone :

Avec la communication synchrone les données sont transmissent d'une manière continue. Il est donc nécessaire d'effectuer la synchronisation des

caractères au début du bloc de données. Cette synchronisation peut être interne ou externe :

- La synchronisation interne : le récepteur détecte un ou plusieurs caractères de synchronisation en tête de message, en cherchant une correspondance bit à bit entre un caractère de synchronisation préétabli et le train de bits arrivant.
- La synchronisation externe : nécessite une ligne spécialisée qui fournit un top d'horloge à chaque début de transmission de bloc.

Remarque 1 :

Dans tous les cas, une horloge cadence la transmission des bits qui ne sont donc émis qu'à des instants déterminés.

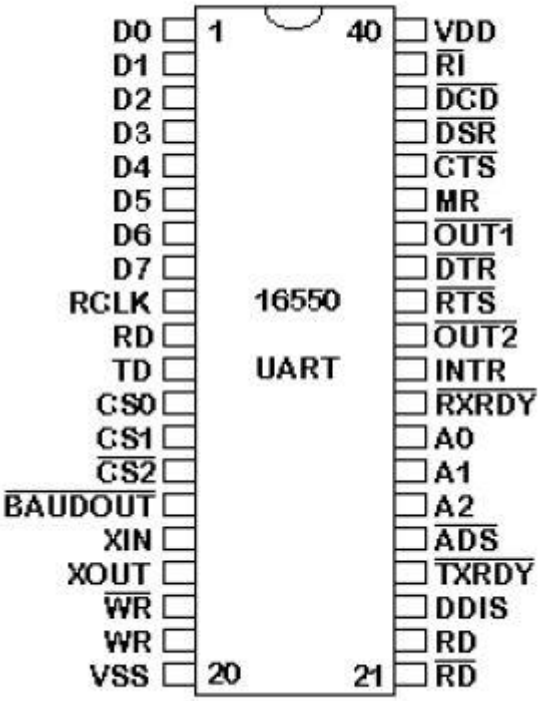
Remarque 2 :

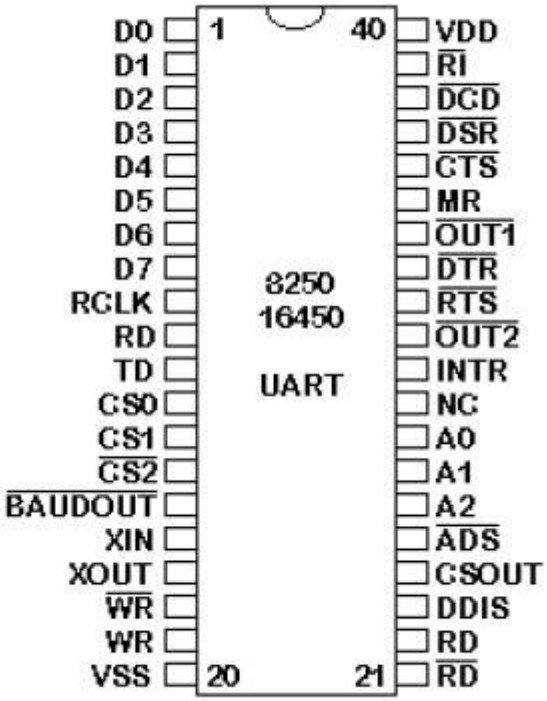
Il existe deux circuits pour les Communications Série: UART (Universal Asynchrenous Receiver-Transmitter, (les Ports COM d'IBM PC utilisent l'UART8050/16550 de NS) et USART (Universal Synchroneous- Asynchroneous Receiver-Transmitter (Le 8251 de Intel).

II / L'interface série le 8250/16550 :

II-1 / Introduction :

Le composant électronique chargé de la gestion des transmissions séries asynchrones dans les PC est appelé UART (*Universal Asynchronous Receiver Transmitter*).le circuit qui gère cette communication est le 8250 de National Semiconductor ou son équivalent le 16550 dont le brochage est comme suit :


D0
D1
D2
D3
D4
D5
D6
D7
RCLK
RD
TD
CS0
CS1
CS2
BAUDOUT
XIN
XOUT
WR
WR
VSS
1
40
20
21
16550
UART
VDD
RI
DCD
DSR
CTS
MR
OUT1
DTR
RTS
OUT2
INTR
RXRDY
A0
A1
A2
ADS
TXRDY
DDIS
RD
RD


D0
D1
D2
D3
D4
D5
D6
D7
RCLK
RD
TD
CS0
CS1
CS2
BAUDOUT
XIN
XOUT
WR
WR
VSS
1
40
20
21
8250
16450
UART
VDD
RI
DCD
DSR
CTS
MR
OUT1
DTR
RTS
OUT2
INTR
NC
A0
A1
A2
ADS
CSOUT
DDIS
RD
RD

II-2 / Description des pines :

N° pine	Nom	Notes
Pin 1:8	D0:D7	Bus de données
Pin 9	RCLK	Receiver Clock Input. La fréquence de cette entrée doit être égale au baud du réception * 16.
Pin 10	RD	Ligne de réception de données
Pin 11	TD	Ligne de transmission de données
Pin 12	CS0	Chip Select 0 - Active haut
Pin 13	CS1	Chip Select 1 - Active Haut
Pin 14	CS2	Chip Select 2 - Active bas
Pin 15	nBAUDOUT	Baud Output = (Baud Rate x 16)
Pin 16	XIN	External Crystal Input
Pin 17	XOUT	External Crystal Output
Pin 18	nWR	Write Line - Inverted
Pin 19	WR	Write Line - Not Inverted
Pin 20	VSS	GND
Pin 21	RD	Read Line - Inverted

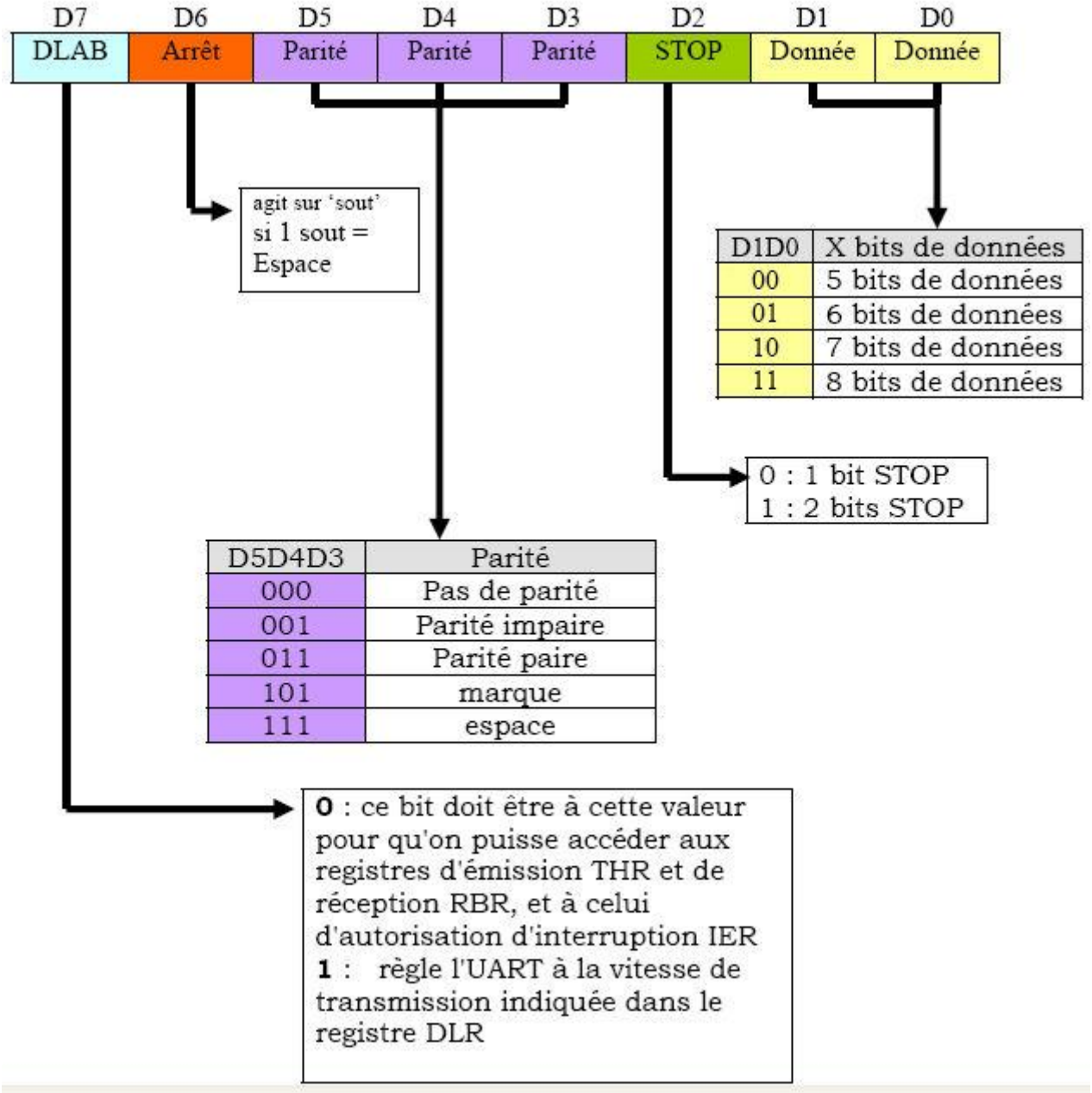
Pin 22	nRD	Read Line -
Pin 23	DDIS	= 0 lorsque le microprocesseur lit les données à partir du 8250, = 1 pour inhiber un transceiver se trouvent sur le bus de données.
Pin 24	nTXRDY	Transmit Ready
Pin 25	nADS	Address Strobe. Utiliser lorsque les signaux ne son pas stables pendant le cycle d'écriture ou de lecture.
Pin 26	A2	Address Bit 2
Pin 27	A1	Address Bit 1
Pin 28	A0	Address Bit 0
Pin 29	nRXRDY	Receive Ready
Pin 30	INTR	Interrupt Output
Pin 31	nOUT2	User Output 2
Pin 32	nRTS	Request to Send
Pin 33	nDTR	Data Terminal Ready
Pin 34	nOUT1	User Output 1
Pin 35	MR	Master Reset
Pin 36	nCTS	Clear To Send
Pin 37	nDSR	Data Set Ready
Pin 38	nDCD	Data Carrier Detect
Pin 39	nRI	Ring Indicator
Pin 40	VDD	+ 5 Volts

III / Les registres du 8250 :

III-1 /Les registres de contrôle de protocole

Ces registres permettent de construire un protocole sommaire (réglages des bits de parités et de stop, sélection de la vitesse de transmission) :

III-1-1 / LCR - Line Control Register (registre de contrôle de la ligne)



III- 1- 2 / DLR - Divisor Latch Register (registre de sélection de la vitesse de transfert)

ce registre est codé sur 16 bits : Pour pouvoir utiliser ce registre on a deux adresses, qui permetent de sélectionner facilement une vitesse de transfert de 50 à 9600

bauds (c'est à dire un maximum de 19200 bps), mais il ne faut pas oublier auparavant de forcer le bit DLAB du registre LCR à l'état 1.

Remarque :

La vitesse n'est pas programmée directement dans l'UART - c'est un circuit comparable à une horloge (baud rate generator) qui se charge de réguler celle-ci.

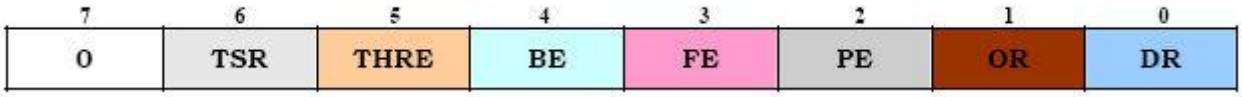
Vitesse en bauds	Contenu du registre (valeur hexa)
50	0900
75	0600
1200	0060
2400	0030
3600	0020
4800	0018
7200	0010
9600	000C

EXEMPLE : pour régler l'UART à une vitesse de 9600 bauds, il faudra copier 0x0C vers DLR.

III - 2 / Les registres de contrôle de la ligne

Ils sont au nombre de trois, et permettent de connaître l'état de la ligne ou du modem, et de réinitialiser celui-ci :

III - 2 - 1 / LSR - Line Status Register (registre d'état de la ligne)



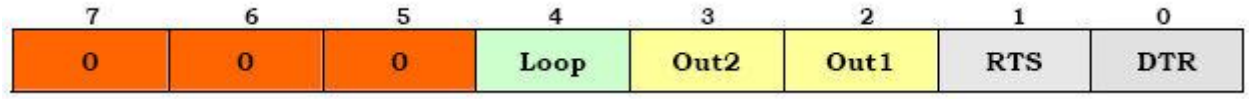
- DR (Data Received) : Si DR =1, ce bit indique qu'un caractère complètement transmis se trouve dans le registre de réception RBR. La lecture de celui-ci réinitialisera automatiquement ce bit DR à 0.
- OR (OverRun) : ce bit est mis à 1 quant le registre de réception RBR n'est pas vide et qu'une donnée reçue risque d'écraser son contenu. Une simple lecture du registre de LSR réinitialisera ce bit à 0.
- PE (Parity Error) : mis à 1 sur une erreur de parité. Réinitialisé à la lecture de LSR.
- FE (Frame Error) : bits de stop incorrects. Reste à 1 tant que le premier bit de stop (celui suivant le dernier bit de données ou le bit de parité) est à 0.
- BE (Break Interrupt) : erreur de type break sur la ligne
- THR (Transmission Holding Register Empty) : ce bit prend la valeur de 1 quand l'UART a fini de transmettre le dernier caractère du registre de transmission THR dans le registre TDR.
- TSRE (Transmitter Shift Register Empty) : ce bit est en lecture seule. Il indique (passage à l'état 1) que le caractère du registre TDR a été envoyé sur la ligne, et donc que celui-ci est vide. Dès qu'une donnée arrive de THR dans TDR, la valeur repasse à 0.

III-2-2 / MSR - Modem Status Register (registre d'état du modem)

Les valeurs de ce registre correspondent aux broches de l'interface. Les bits 0 à 3 indiquent les variations depuis la dernière lecture du processeur du registre - ils seront forcés à 1 si il y a eu changement. Toutes leurs valeurs sont réinitialisées, s'il y a lieu, à un 0 logique dès que le processeur a achevé une lecture du registre. Les bits 4 à 7 contiennent, quant à eux, l'état actuel de diverses données de la ligne.

- DCTS (Delta Clear To Send) : variation d'état de la broche CTS (préparation d'émission)
- DDSR (Delta Data Set Ready) : variation d'état de la broche DSR (données prêtes)
- TERI (Trailing Edge of Ring Indicator) : entrée RI de l'UART (Ring Indicator). Ce bit passe à 1 si RI passe de ON (0) à OFF (1). Un appel a été détecté depuis la dernière lecture.
- DRLSD (Delta Received Line Signal Detector) : variation d'état de l'entrée RLSD de l'UART. Indique si la liaison avec le modem distant a été établie.
- CTS (Clear To Send) : état de la broche CTS.
- DSR (Data Set Ready) : état de la broche DSR.
- RI (Ring Indicator) : signale un appel sur la ligne.
- RLSD (Receive Line Signal Detect) : indique si la liaison avec le modem distant a été établie.

III-2-3 / MCR - Modem Control Register (registre de contrôle du modem)

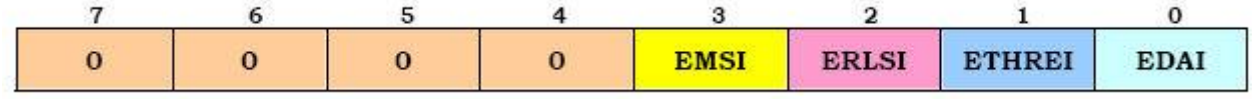


Les bits OUT1, OUT2, RTS et DTR permettent, en forçant leur valeur, de contrôler la valeur barre des signaux correspondants (DTR et RTS correspondent à des broches de l'interface, OUT1 et OUT2 sont des sorties auxiliaires).

III- 3 / Les registres de contrôle d'interruptions

On pourra par le biais de ces registres déclarer quels événements déclencheront une interruption. Une fois l'interruption détectée dans le programme, on pourra définir sa cause, et y répondre.

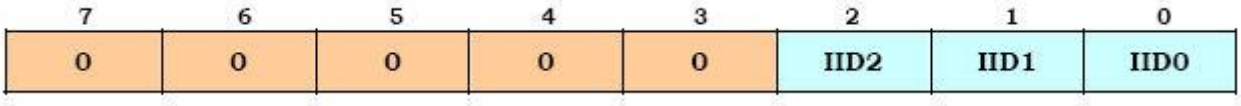
III - 3 - 1 / IER - Interrupt Enable Register (registre d'autorisations d'interruptions) :



Le bit DLAB du registre LCR (registre de contrôle de ligne) doit être forcé à 0 pour pouvoir accéder à ce registre. Les quatre bits de poids faible (bits 0 à 3) permettent, lorsqu'ils sont forcés à 1, de générer une interruptions lors de l'apparition de l'événement associé - on pourra retrouver la cause de cette interruption dans le registre IIR. La liste ci-dessous définit quels événements associés avec les bits de ce registre :

- EDAI (Enable Data Available Interrupt) : arrivée d'un caractère
- ETHREI (Enable Tx Holding Register Empty Interrupt) : registre THR vide
- ERLSI (Enable Receive Line Status Interrupt) : modification de l'état de la ligne
- EMSI (Enable Modem Status Interrupt) : modification de l'état du modem

III – 3 – 2 / IIR - Interrupt Identification Register (registre de cause d'interruption)



Les trois bits IID (Interrupt ID) permettent de connaître la cause de l'interruption, sa priorité, et le moyen de désactiver l'interruption. Le tableau suivant fournit la liste des combinaisons de ces trois bits et leur signification :

IID2	IID1	IID0	Niveau de priorité	Type d'interruption	Source de l'interruption	Méthode de réinitialisation
0	0	1	-	aucune	aucune	-
1	1	0	Maximale	Modification de LSR (état de la ligne) : bit ERLSI de IER	Surcharge (une donnée arrive alors qu'on n'a pas encore lu la précédente) OU Erreur de parité OU Frame Error (bit de stop) OU Break	Lecture de LSR
1	0	0	2ème ordre	Réception d'un caractère : bit EDAI de IER	Données transmises disponibles	Lecture de RBR (registre de réception)
0	1	0	3ème ordre	Registre THR (registre d'envoi) vide : bit ETHREI de IER	Registre THR vide	Lecture de IIR OU écriture dans THR
0	0	0	4ème ordre	Modification de l'état du modem : bit EMSI de IER	Modification broche Clear To Send OU Ring Indicator OU Received Line Signal Direct	Lecture de MSR (état du modem)

III – 4 / Les registres de transfert

III – 4 – 1 / Les registres d'émission

Lorsqu'un caractère doit être transmis, il doit d'abord être transféré dans le registre THR (Transmission Holding Register = registre d'attente de l'émetteur). Il y restera tant que le caractère précédent ne sera pas acquitté par la machine distante. Une fois l'acquittement reçu, le caractère sera transféré dans un autre registre :

TSR (Transmission Shift Register = Registre de décalage de l'émetteur). L'UART se chargera alors de transmettre le caractère bit à bit sur la ligne, et il pourra y insérer, selon les réglages effectués dans LCR (contrôle de ligne), un bit de parité et un nombre fixé de bits de stop. Il est important de noter ces quelques remarques :

· Ce registre est à écriture seule (il possède la même adresse que RBR, le registre de réception : un ordre de lecture à l'adresse de THR renverra donc le contenu de RBR)

· Le bit DLAB du registre LCR (contrôle de la ligne) doit être forcé à 0 pour pouvoir effectuer une écriture dans ce registre.

· Ce registre est sur 8 bits. On pourra y envoyer de 5 à 8 bits, suivant les réglages du registre de contrôle de ligne LCR (bits WLS), sachant que ce sera le bit 0 (poids le plus faible) qui sera le premier transmis.

· il sera judicieux d'effectuer une lecture de LSR (registre d'état de la ligne) - bits 5 et 6 - pour savoir quand le registre THR devient libre, et quand le caractère présent dans TDR a été transmi totalement sur la ligne .

III – 4 - 2 / Les registres de réception

L'octet reçu sera transféré (sans ses bits de parité et de stop) dans le registre RBR (Receiver Buffer Register = registre de réception). De la même manière qu'avec les registres d'émission, il existe un registre de transit - appelé RDR - par lequel le bit reçu passera en premier, avant d'être débarrassé de ses bits de contrôle.

Remarque :

· Le registre RBR est à lecture seule. les registres d'émission THR et de réception RBR possèdent une adresse commune. Ainsi une écriture sur l'adresse de RBR provoquera un remplissage du registre THR, et n'aura aucun effet sur le registre de réception.

· Le bit DLAB du registre LCR (contrôle de la ligne) doit être forcé à 0 pour pouvoir effectuer une lecture dans ce registre.

· Ce registre est sur 8 bits : y seront copiés les 5 à 8 bits de données, sans bits de contrôle. Le bit 0, de poids le plus faible, est le premier à avoir été reçu.

· il sera judicieux d'effectuer une lecture de LSR (registre d'état de la ligne) - bits 0 à 4 - pour détecter les éventuelles erreurs de transmission, ou tout simplement pour savoir si un caractère a été reçu – se reporter à la partie traitant de ce registre.

IV Exemple de programmation du 8250/16550 :

IV – 1 / Programmation directe :

Exemple de programme qui permet de lire un caractère du port COM1 par polling :

```
LSR EQU 3FDH
TXRX EQU 3F8H
RDA EQU 01H
TBE EQU 20H
DEBUT : IN AL, LSR   ; Lecture de LSR
TEST AL, RDA ; tester RDA
JZ DEBUT                ; if not RDA
IN AL, TXRX             ; lecture de la donnée
```

EXAMPLE 2: Ecriture d'un caractère sur le port COM1 :

```
DEBUT :
IN AL, LSR   ; Lecture LSR
TEST AL, TBE          ; tester TBE
JZ DEBUT
MOV AL, 'a'         ; caractère à transmettre
OUT TXRX, AL         ; envoie caractère sur le port
```

IV – 2 / Programmation en utilisant les interruptions BIOS/ DOS :

L'interruption 21H (DOS):

Fonction (AH)	opération
03H	Lecture d'un caractère
04H	Envoie d'un caractère

L'interruption BIOS 14H :

Fonction (AH)	Opération	Entrée	Sortie
00	initialisation	AL = paramètre DX = Port	AH : Etat du port AL : Etat du modem
01	Ecriture d'un caractère	AL = caractère DX=port	AH : bit7=0 succé AH : bit7=1 échec
02	Lecture d'un caractère	DX=port	AH : bit7=0 succé AH : bit7=1 échec
03	Lecture du Status	DX=port	AH : état du port AL : Etat du modem

Exemple de programme Ecriture d'un mot :

```
COM1 EQU  0
COM2 EQU  1
TBE  EQU  20H
RDA  EQU  01H
DEBUT :MOV DX, COM1
MOV AH, 03H
INT 14H
TEST AH, TBE   ; test la valeur de TBE JZ DEBUT
MOV DX, COM1
MOV AH, 01H
MOV AL, 'a'    ; Ecriture du caractère
INT 14H
```

Exemple de programme de lecture d'un caractère :

```
COM1 EQU  0
COM2 EQU  1
TBE  EQU   20H
RDA  EQU   01H
DEBUT:
MOV DX, COM1
MOV AH, 03H
INT 14H
TEST AH, RDA
JZ DEBUT
MOV DX, COM1
MOV AH, 02H
INT 14H
```