Module : Microprocesseurs et API

Partie 1 :

Microprocesseur

1. Architecture de base d'un microprocesseur

1.1 définition d'un µP

Le microprocesseur (processeur) est le composant hardware le plus connu d'un système micro-programmé, noté aussi M.P.U. (Microprocessor unit) ou encore C.P.U. (Central Processing Unit). C'est l'unité intelligente de traitement des informations. Son travail consiste à lire des programmes (des suites d'instructions), à les décoder et à les exécuter. Les années 80 voyaient l'émergence de ces circuits avec les Zylog Z80, 6800 de Motorola, le 8085 de Intel qui est souvent utilisé en tant que microcontrôleur.

Il doit aussi prendre en compte les informations extérieures au système et assurer leur traitement.

1.2 Composition d'un µP

Un microprocesseur est constitué de :

- une unité de commande qui lit les instructions et les décode;
- une unité de traitement (UAL - unité arithmétique et logique) qui exécute les instructions;
- d'un ensemble de mémoire appelés registres;
- d'un bus de données externe;
- d'un bus d'adresse externe;
- d'un bus de commande externe;
- d'un bus de données interne reliant l'unité de commande l'UAL et les registres.

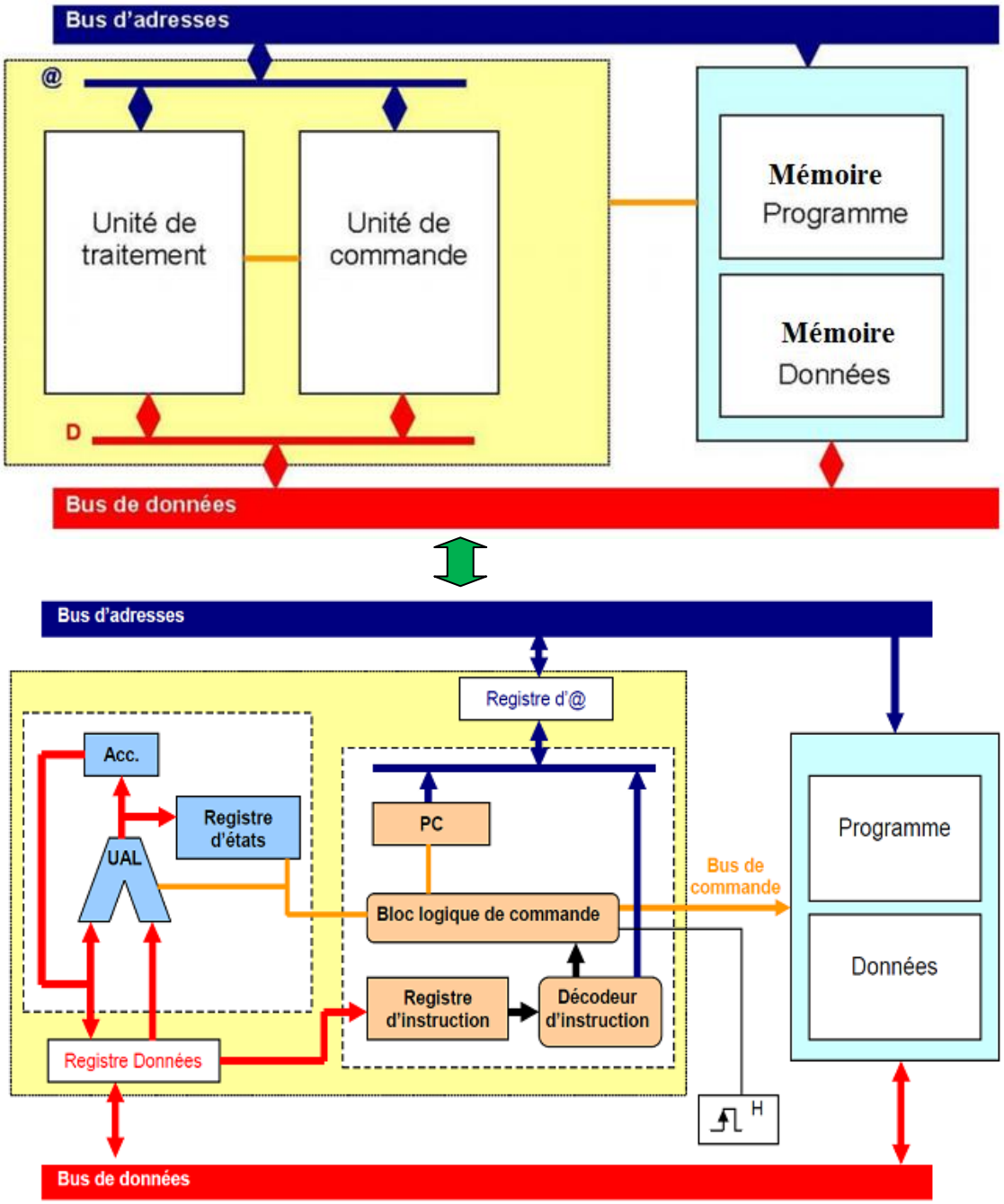
Principe d'exécution d'une instruction

Dans une architecture standard, l'exécution d'une instruction se fait en trois étapes:

-Recherche de l'instruction (Fetch) ;

-Décodage (décode) ;

-Exécution (exécute).



Lorsque tous ces éléments sont assemblés sur une même puce, on parle alors de microprocesseur.

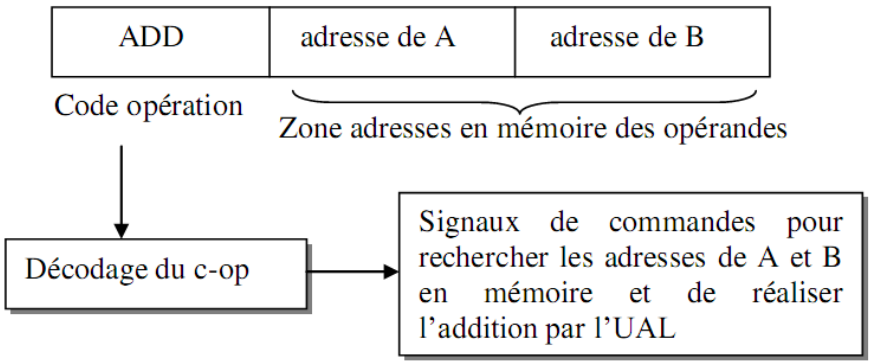
1.2.1 Unité de commande

Elle permet de "séquencer" le déroulement des instructions. Elle effectue la recherche en mémoire du programme, le décodage, le pilotage de l'exécution des instructions via les signaux de l'horloge ainsi que la préparation de l'instruction suivante. L'unité de commande élabore tous les signaux de synchronisation internes ou externes (bus des commandes) au microprocesseur.

Elle est chargée également à travers le décodeur d'instruction de décomposer et analyser l'instruction se trouvant dans le registre d'instructions, selon le code opération, elle envoie la nature des opérations à effectuer à l'unité de commande et précisément le séquenceur qui va ensuite générer les microcommandes nécessaires aux différents composants participant à l'exécution de l'instruction en cours.

Exemple :

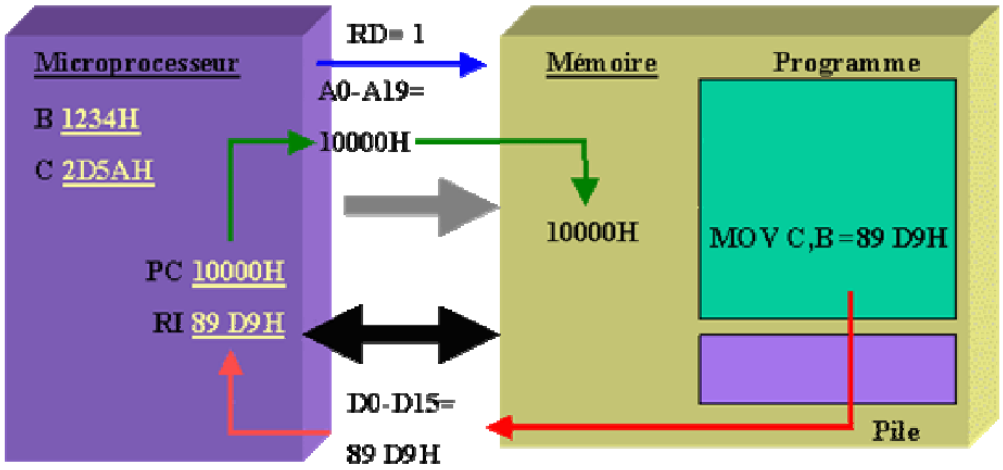
Pour un microprocesseur ayant un format d'instructions à deux adresses, l'instruction d'addition de deux variables en mémoire A et B est représenté dans le registre d'instructions comme suit :



Recherche de l'instruction

Le contenu de PC (compteur ordinal) est placé sur le bus d'adresse (c'est l'unité de commande qui établit la connexion). L'unité de commande (UC) émet un ordre de lecture (READ=RD=1). Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données. L'unité de commande charge la donnée dans le registre d'instruction pour décodage.

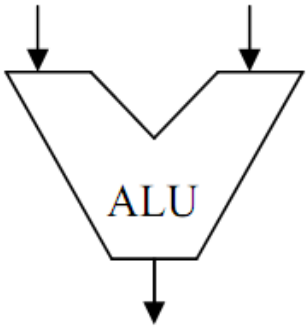
Exemple : Le microprocesseur place le contenu de PC (10000H) sur le bus d'adresse et met RD à 1 (cycle de lecture). La mémoire met sur le bus de données le contenu de sa mémoire n° 10000H (ici 89D9H qui est le code de MOV C,B). Le microprocesseur place dans son registre d'instruction le contenu du bus de données (89D9H). L'unité de commande décode et prépare l'exécution de l'instruction MOV C,B.



1.2.2 Unité de traitement

C'est le cœur du microprocesseur. Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions.

- **L'unité arithmétique et logique (UAL) :** Elle assure les fonctions logiques (ET, OU, Comparaison, Décalage, etc…) ou arithmétique (Addition, soustraction…). Toute instruction qui modifie une donnée fait toujours appel à l'ALU.



-

Symbole de l'Unité arithmétique et logique

- **L'accumulateur :** C'est un registre nommé A il représente l'une des deux entrées de l'UAL. Il est impliqué dans presque toutes les opérations réalisées par l'UAL. Certains constructeurs ont des microprocesseurs à deux accumulateurs (Motorola : 6800).
  Exemple: A étant l'accumulateur et B un registre, on peut avoir : A+B (ADD A,B : addition du contenu du registre A avec celui du registre B, le résultat étant mis dans A)
- **Le registre d'état** : à chaque opération, le microprocesseur positionne un certain nombre de bascules d'état. Ces bascules sont appelées aussi indicateurs d'état ou drapeaux (status, flags). Par exemple, si une soustraction donne un résultat nul, l'indicateur de zéro (Z) sera mis à 1. Ces bascules sont regroupées dans le registre d'état.On peut citer comme indicateurs:

  - retenue (carry : C)
  - retenue intermédiaire (Auxiliary-Carry : AC)
  - signe (Sign : S)
  - débordement (Overflow : O)
  - zéro (Z)
  - parité (Parity : P)

➢ **Le Retenue : (carry : C)**

Flag indiquant si une retenue a eu lieu d'une addition ou d'une soustraction. Si une retenue a été générée, ce bit passe à 1.

Exemple :

		11111100			**FCH**
	+	**10000010**		+	**82H**
carry : **1**	=	**01111110**	carry : **1**	=	**7EH**

De la même manière lors d'une opération de décalage ou de rotation ce bit peut être positionné en cas de débordement. Par exemple, soit le décalage à gauche sur d'un bit de cet octet : 10010110, après le décalage nous avons 00101100. En ce qui concerne le carry, il va être positionné à 1, puisque le bit expulsé est égal à 1.

➢ **Retenue intermédiaire : (Auxiliary Carry : AC)**

Flag fonctionnant comme le bit de Carry, sauf qu'ici la surveillance de la retenue s'exerce non pas sur l'octet entier, mais sur le premier demi-octet. Ce flag se positionne à 1 si une retenue est générée

du bit 3 (bit de poids fort du quartet inférieur) vers le bit 0 du quartet supérieur. Il est utile pour corriger le résultat d'opérations effectuées en code BCD.

- **Signe: (S)**

Ce bit est mise à **1** lorsque le résultat de l'opération est négatif (MSB: bit de plus fort poids du résultat: à 1).

- **Débordement : (overflow : O)**

Cet indicateur est mis **1**, lorsqu'il y a un dépassement de capacité pour les opérations arithmétiques en complément à 2. Sur 8 bits, on peut coder de -128 (1000 0000) à +127 (0111

1111).

104	**0110 1000**	**- 18**	**1110 1110**	
+ 26	**+ 0001 1010**	**- 118**	**1000 1010**	
=130	**= 1000 0010 (-126)**	**-136**	**0111 1000 (120)**	**avec C=1**

- **Le bit Zéro : (Zéro : Z)**

Ce bit est mis à **1** lorsque le résultat de l'opération est nul.

- **Le bit de parité : (P)**

Ce bit est mis à **1** lorsque le nombre de 1 de l'accumulateur est pair. Remarque : La plupart des instructions modifient le registre d'état.

1.3 Les registres

Il y'a deux type de registres : les registres d'usage général, et les registres d'adresses (pointeurs).

- **Les registres d'usage général**

Ce sont des mémoires rapides, à l'intérieur du microprocesseur, qui permettent à l'UAL de manipuler des données à vitesse élevée. Ils sont connectés au bus de données interne au microprocesseur. L'adresse d'un registre est associée à son nom (on donne généralement comme nom une lettre) A, B,C...

Exemple : MOV C,B: transfert du contenu du registre B dans le registre C.

- **Les registres d'adresses (pointeurs)**

Ce sont des registres connectés sur le bus d'adresses. On peut citer comme registre:

- Le compteur ordinal (pointeur de programme PC : Programme Counter) ;

- Le pointeur de pile (stack pointer SP) ;

- Les registres d'index.

1.4 Les BUS

On appelle Bus, en informatique, un ensemble de liaisons physiques (câbles, pistes de circuits imprimés, ...) pouvant être exploitées en commun par plusieurs éléments matériels afin de communiquer. ans le cas

où la ligne sert uniquement à la communication de deux composants matériels, on parle parfois de port (port série, port parallèle, ...). Un Bus est caractérisé par le volume d'informations transmises simultanément (exprimé en bits), correspondant au nombre de lignes sur lesquelles les données sont envoyées de manière simultané. Une nappe de 32 fils permet ainsi de transmettre 32 bits en parallèle. On parle ainsi de "largeur de bus" pour désigner le nombre de bits qu'il peut transmettre simultanément. D'autre part, la vitesse du bus est également définie par sa fréquence (exprimée en Hertz), c'est-à-dire le nombre de paquets de données envoyés ou reçus par seconde. On parle de cycle pour désigner chaque envoi ou réception de données. De cette façon, il est possible de connaître la bande passante d'un bus, c'est-à-dire le débit de données qu'il peut transporter, en multipliant sa largeur par sa fréquence. Un Bus d'une largeur de 16 bits, cadencé à une fréquence de 133 Mhz possède donc une bande passante égale à :

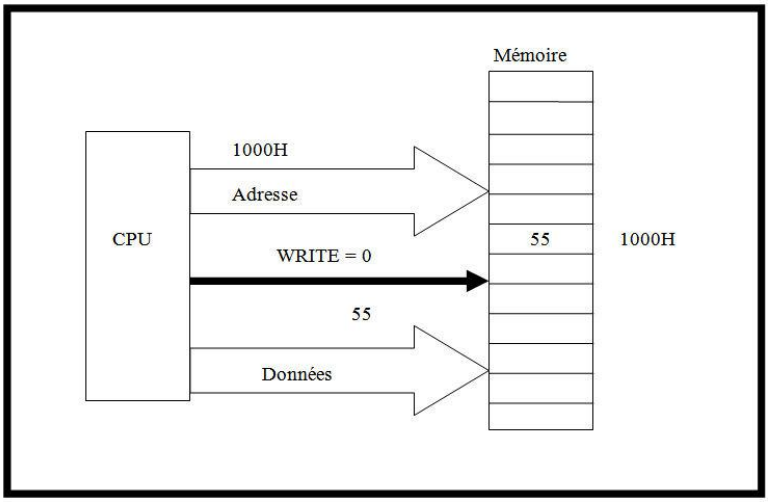
$$16 * 133.10^6 = 2128*10^6 \text{ bit/s,}$$

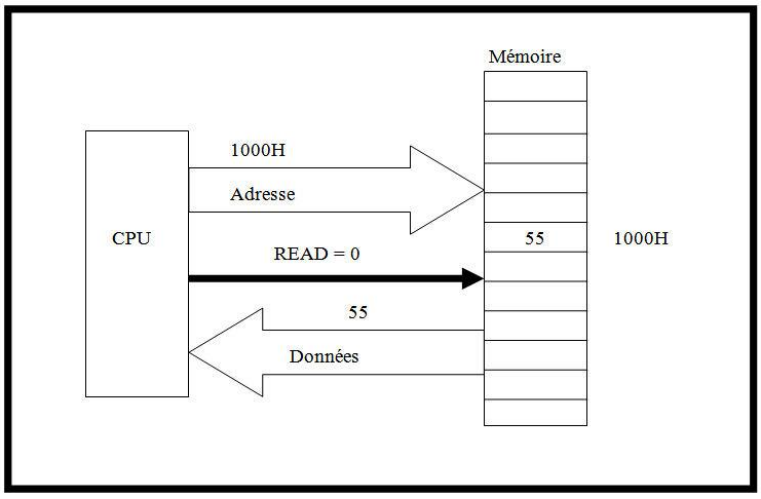
$$\text{soit } 2128*10^6/8 = 266*10^6 \text{ octets/s}$$

$$\text{soit } 266*10^6 /1024 = 259.7*10^3 \text{ Ko/s}$$

$$\text{soit } 259.7*10^3 /1024 = 253.7 \text{ Mo/s}$$

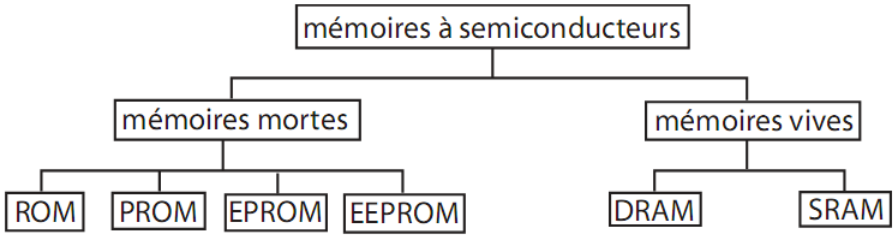
Pour la communication, un microprocesseur a besoin en général de trois Bus. Un Bus de données, un Bus d'adresse et un Bus de commande. Sur la figure suivante nous présentons un système à base de microprocesseur classique avec ces trois Bus.

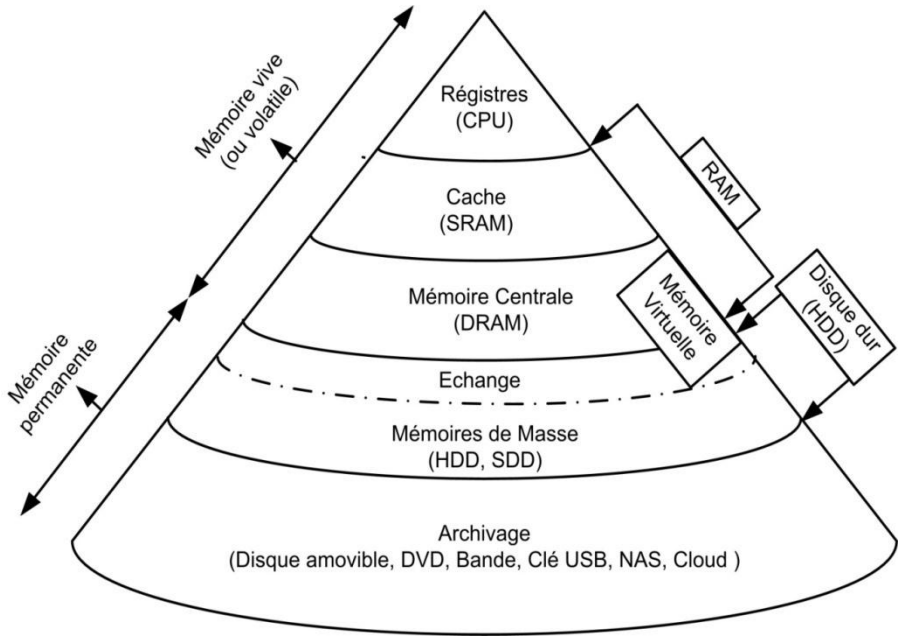




1.5 les mémoires

Les mémoires sont connectées à un bus d'adresse de **n** bits, un bus de données de p (p=1 c'est-à-dire 8bits) bits et des lignes de commandes. Pour pouvoir communiquer avec le microprocesseur, on va relier leurs bus ensembles. Pour cela, il est nécessaire d'avoir adéquation entre le nombre de bits des bus de données et d'adresse de la mémoire et du microprocesseur.





1.5.1 les mémoires ROM : Read Only Memory. Mémoire à lecture seule, sans écriture.

Types de ROM : PROM (Mémoire programmable une seule fois), UVPROM (ROM programmable électriquement et effaçable par exposition au rayons ultraviolet), EEPROM (ROM programmable et effaçable électriquement)

1.5.2 les mémoires RAM : La mémoire vive, généralement appelée RAM (Random Access Memory, mémoire à accès aléatoire), est la mémoire principale du système, c'est-à-dire qu'il s'agit d'un espace permettant de stocker de manière temporaire des données lors de l'exécution d'un programme. En effet le stockage de données dans la mémoire vive est temporaire, contrairement au stockage de données sur une mémoire de masse telle que le disque dur, car elle permet uniquement de stocker des données tant

qu'elle est alimentée électriquement. Ainsi, à chaque fois que l'ordinateur est éteint, toutes les données présentes en mémoire sont irrémédiablement effacées.

Types de RAM

Les mémoires statiques SRAM (Static RAM) : rapides mais couteuses, elles sont utilisées comme mémoires caches pour les processeurs.

La mémoire cache a pour fonction principale d'optimiser les accès aux différentes instructions et données que le microprocesseur a besoin lors de l'exécution des programmes. Elle permet de mettre dans la mémoire cache qui est beaucoup plus rapide que la mémoire centrale, les informations les plus utilisées et que le microprocesseur a besoin fréquemment.

Les mémoires dynamiques DRAM : peu couteuses avec une grande capacité de stockage par rapport aux SRAM mais moins rapides, elles sont utilisées principalement pour la mémoire centrale des ordinateurs. Plusieurs types de mémoires vives dynamiques existes : SDRAM qui fonctionnent à 100MHz, les RDRAM, les DDR-SDRAM, DDR2 ou DDR3.

2. Le microprocesseur 8086

Du fait de la compatibilité, ascendante des microprocesseurs de INTEL, il est possible de programmer un i7 avec les instructions du 8086 que nous allons étudier dans cette partie.

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

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

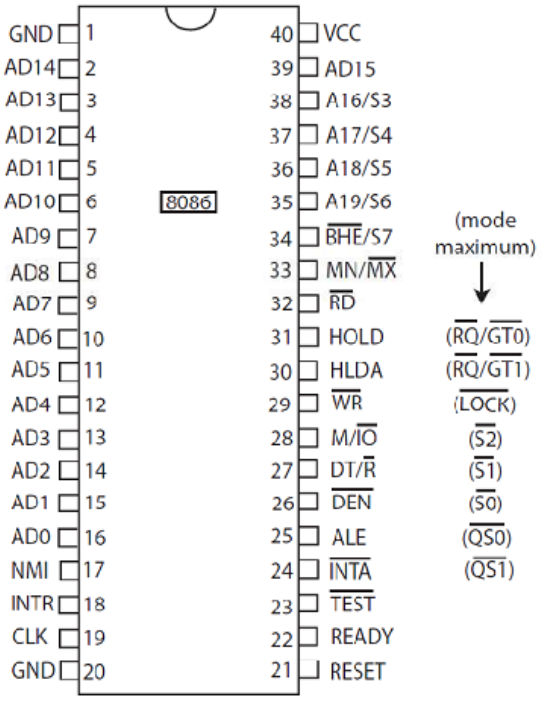
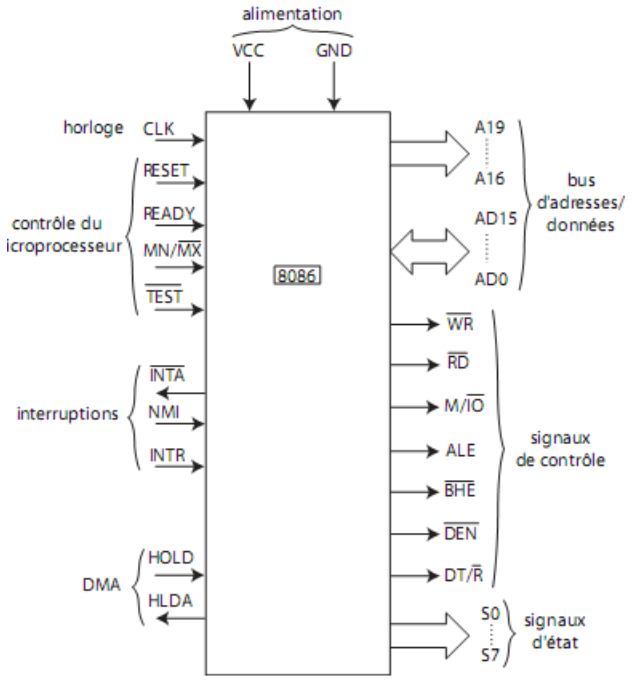


Figure : brochage 8086



Le tableau ci-dessous illustre la description des différentes broches

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

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

2.1 les registres du 8086

Les registres du microprocesseur 8086 sont tous des registres 16 bits. Ils sont répartis en 5 catégories: registres de travail, registres de segment, registres pointeurs, registres index et registres spéciaux.

2.1.1 les registres de travail

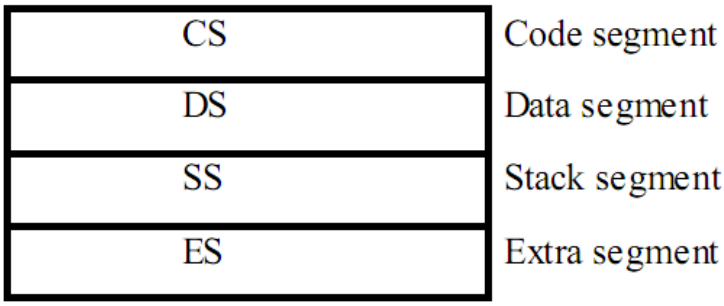
Le registre AX peut être décomposé en AH (H pour High, les 8 bits de fort poids de AX) et AL (L pour Low, les 8 bits de faible poids de AX). De la même façon BX peut être décomposé en BH et BL, CX en CH et CL, DX en DH et DL.

Dans beaucoup d'instructions, ces registres sont banalisés et peuvent être utilisés indifféremment. Toutefois, dans certaines instructions, ils ont un rôle bien précis:

- le registre AX est utilisé implicitement comme accumulateur (dans les instructions de multiplication),
- le registre BX comme registre de base (pour des données dans le segment pointé des adresses dans le segment de données),
- CX comme compteur d'itérations (dans les instructions de boucle),
- DX contient l'adresse du port d'E/S dans les instructions d'E/S IN et OUT et sert, également, d'extension à AX (données dur 32 bits).

2.1.2 les registres de segment

L'espace adressable de la mémoire de 8086 est un espace de 1 Mo divisé en segments logiques ayant des tailles pouvant aller jusqu'à 64K octets.



- CS : Code Segment : sert à l'adressage des octets du programme (code)
- DS: Data Segment : sert à l'adressage de la mémoire de données dans laquelle sont stockées toutes les données traitées par le programme.
- SS : Stack Segment : gère la pile.
- ES : Extra Segment :sert à l'adressage d'une mémoire de données supplémentaires.

2.1.3 les registres pointeurs ou d'adressage (offset)

Ces registres (SP, BP, SI et DI) de 16 bits permettent l'adressage d'un opérande à l'intérieur d'un segment de 64 ko (216 positions mémoires).

Ils sont au nombre de 2 et notés SP et BP. Ils sont dédiés à l'utilisation de la pile. Le registre SP (Stack Pointer) pointe sur le sommet de la pile et il est mis à jour automatiquement par les instructions d'empilement et de dépilement; BP (Base Pointer) pointe la base de la région de la pile contenant les données accessibles (variables locales, paramètres,...) à l'intérieur d'une procédure. Il doit être mis à jour par le programmeur.

2.1.4 les registres index

Ils sont au nombre de deux et ils sont notés SI et DI. Les registres d'index peuvent être utilisés comme des registres généraux pour sauvegarder et pour compter. De plus, chacun de ces 2 registres peut être utilisé pour indexer les éléments d'un tableau.

2.1.5 Les registres spéciaux

Ils sont au nombre de 2 notés IP (Instruction Pointer) et SR (Status Register) et qui ne sont pas modifiable par l'utilisateur.

IP joue le rôle de compteur ordinal et contient le déplacement dans le segment de code de la prochaine instruction à exécuter. Le couple CS:IP donne donc l'adresse physique sur 20 bits. À chaque type d'accès en mémoire correspond par défaut un registre de segment et parfois un registre général. Par exemple, CS et IP sont combinés pour accéder à la prochaine instruction à exécuter sous la forme CS :IP. SS et SP sont combinés en SS:SP pour toutes les opérations concernant la pile. II est parfois possible d'utiliser un registre de segment autre que celui utilisé par défaut en le spécifiant explicitement.

Le registre SR contient l'état du microprocesseur matérialisé par les indicateurs (Flags).

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

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

• C : (Carry) indique le dépassement de capacité de 1 sur une opération 8 bits ou 16 bits. Ce flag peut être utilisé par des instructions de saut conditionnel, des calculs arithmétiques en chaîne ou dans des opérations de rotation.

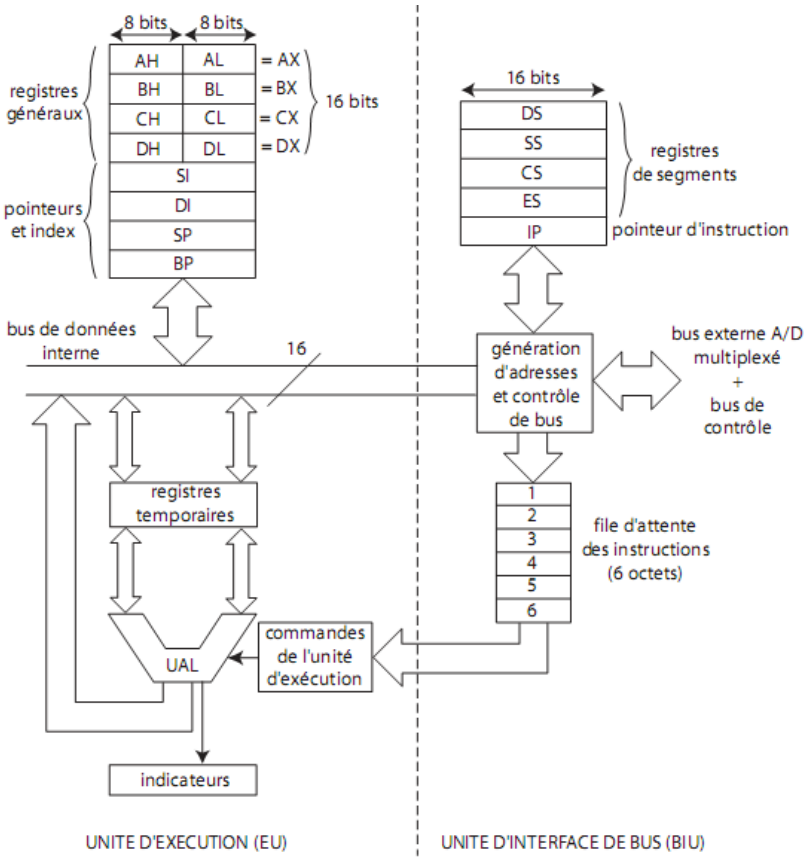
• P : (Parité) indique que le nombre de 1 est un nombre pair. Ce flag est utilisé avec certains sauts conditionnels.

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

ax	ah	al
bx	bh	bl
cx	ch	cl
dx	dh	dl
si		
di		
sp	offset du sommet de pile	
bp		
ip	offset de la prochaine instruction à exécuter	
cs	numéro du segment d'instructions	
ds	numéro du segment de données	
es		
ss	numéro du segment de pile	
flags		

2.2 L'unité d'interface avec le bus (BUS)

Le BIU effectue toutes les opérations de bus sur l'unité d'exécution EU (ALU, SR et les registres généraux). Les transferts de données entre le CPU d'une part et la mémoire ou les E /S d' autre par se font sur demande de L'EU. Elle recherche les instructions en mémoire et les rangs dans une file d'attente ;



2.3 Le fonctionnement interne d'un 8086

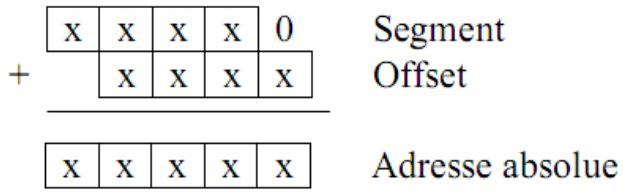
Le fonctionnement avec la mémoire

Pour le 8086/88 le bits couvrant 1Mo (de 00000H à FFFFFH)

La solution adoptée par Intel a été la suivante : Puisque avec 16 bits en peut adresser 2^{16} octets = 65535 octets = 64 ko, La mémoire totale adressable de 1 Mo est fractionnée en pages de 64 ko appelés segments. On utilise alors deux registres pour adresser une case mémoire donnée. Un registre pour adresser le segment qu'on appelle registre segment et un registre pour adresser à l'intérieur du segment qu'on désignera par registre d'adressage ou offset. Une adresse se présente toujours sous la forme segment: offset

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

Pour calculer l'adresse physique ou l'adresse absolue on procède à cette opération : on décale le registre segment vers la gauche et on donne la valeur 0 en hexa au bit le mois significatif puis on effectue une addition entre le registre segment et l'offset :

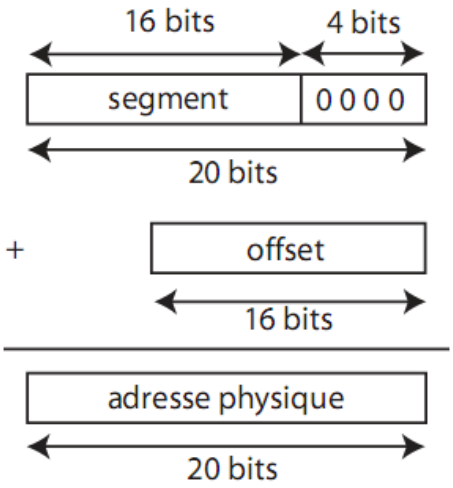


Dans notre exemple, l'adresse de la case mémoire considérée devient 2000:350 soit :

Segment = 2000

Offset = 350

L'adresse absolue ou adresse physique est calculée ainsi :



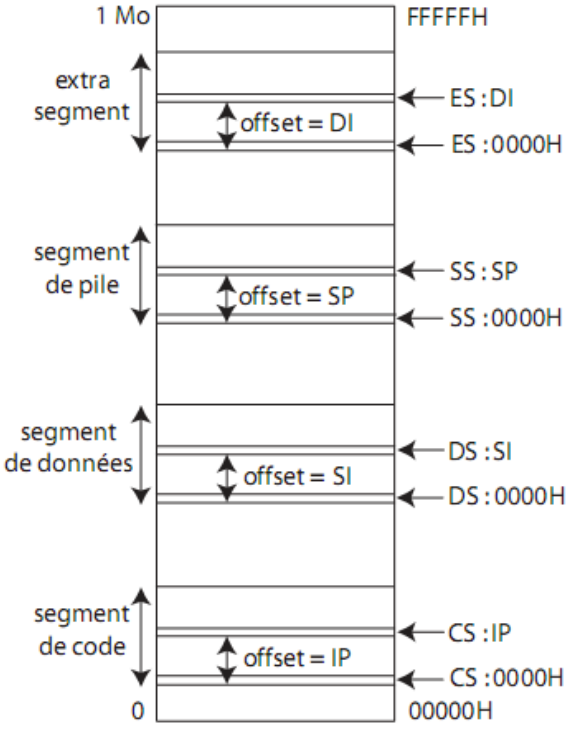
A un instant donné, le 8086 a accès à 4 segments dont les adresses se trouvent dans les registres de segment CS, DS, SS et ES. Le segment de code contient les instructions du programme, le segment de données contient les données manipulées par le programme, le segment de pile contient la pile de sauvegarde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

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

De même, les registres de segments DS et ES peuvent être associés à un registre d'index.

Exemple : DS : SI, ES : DI. Le registre de segment de pile peut être associé aux registres de pointeurs : SS : SP ou SS : BP.

Mémoire accessible par le 8086 à un instant donné:



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

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

Segment par défaut

3. LES MODES D'ADRESSAGE DU 8086

3.1 Mode d'adressage immédiat

Dans le mode d'adressage immédiat, l'opérande source est une donnée sur 8 ou 16 bits. L'exécution d'une telle instruction est très rapide.

MOV CX,500H ; dans CX=0000 0101 0000 0000 =0500H

MOV CX, -40H ; dans CX=1111 1111 1100 0000 =FFC0H

3.2 Mode d'adressage registre

Dans le mode d'adressage registre, l'opérande à utiliser est contenu dans un des registres généraux du CPU (8 bits ou 16 bits). La longueur de l'opérande source et l'opérande destination doivent être identiques.

Exemples :MOV AX,BX

MOV AL,BL

3.3 Mode d'adressage direct

Le mode d'adressage direct spécifie complètement dans l'instruction l'emplacement mémoire qui contient l'opérande. L'adresse Registre_seg:Offset doit être placée entre [ ], si le segment n'est pas précisé, DS est pris par défaut.

Exemple : MOV AX,COMPTE ;COMPTE est une variable déjà déclarée

Dans cette instruction, le contenu des cases mémoire dont l'adresse est pointée par

DS :COMPTE et DS :COMPTE+1 sera transféré dans AX.

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

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

MOV AX, SS:[243] : Copier le contenu de la mémoire SS:243 dans AX

3.4 Mode d'adressage indirect

Dans le mode d 'adressage registre indirect, l'adresse n'est pas donnée ou plus précisement l'offset n'est pas précisé directement dans l'instruction mais il se trouve dans l'un des 4 registres d'offset BX, BP, SI

ou DI et c'est le registre qui sera précisé dans l'instruction : [Registre segment :Registre offset] et qu'il faudrait évidemment charger au préalable par la bonne adresse.

Exemple : MOV BX, offset COMPTE

MOV AX,[BX]

La première instruction signifie qu'on a mis dans le registre BX, l'offset de la variable COMPTE.

La deuxième instruction signifie qu'on à charger le registre AX par le contenu de la case mémoire dont l'adresse se trouve pointée par le registre BX, (qui est l'offset de COMPTE).

D'autres exemples :

MOV AX, [BX] ; Charger AX par le contenu de la mémoire d'adresse DS:BX

MOV AX, [BP] ; Charger AX par le contenu de la mémoire d'adresse SS:BP

MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI

MOV AX, [DI] ; Charger AX par le contenu de la mémoire d'adresse DS:DI

MOV AX, ES:[BX] ; Charger AX par le contenu de la mémoire d'adresse ES:BX

3.4.1 Mode d'adressage relatif à une base (ou adressage basé)

L'offset se trouve dans l'un des deux registres de base BX ou BP. On peut préciser un déplacement qui sera ajouté au contenu de Roff pour déterminer l'offset,

Avec le registre BX on peut accéder à divers structure de données qui se trouvent en différents endroits de la mémoire. Pour ce la il suffit de mettre l'adresse de base dans le registre de base (BX ou BP) et on pointe les éléments de la structure par leurs déplacement par rapport à la base. Pour se déplacer vers une autre structure, il suffit de changer le registre de base.

Exemples :

MOV AX, [BX] : Charger AX par le contenu de la mémoire d'adresse DS:BX

MOV AX, [BX+5] : Charger AX par le contenu de la mémoire d'adresse DS:BX+5

MOV AX, [BP-200] : Charger AX par le contenu de la mémoire d'adresse SS:BP-200

AX←DS :BX+3	**AX←SS :BP+3**
MOV AX,[BX]+3	MOV AX,[BP]+3
MOV AX,[BX+3]	MOV AX,[BP+3]
MOV AX,3[BX]	MOV AX,3[BP]

3.4.2 Mode d'adressage direct indexé

Dans le mode d'adressage indexé, l'adresse effective est la somme d'un label ou d'un déplacement et d'un registre indexe SI ou DI.

Exemples :

MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI

MOV AX, [SI+500] ; Charger AX par la mémoire d'adresse DS:SI+500

MOV AX, [DI-8] ; Charger AX par la mémoire d'adresse DS:DI-8

MOV AX, ES:|[SI+4] ; Charger AX par la mémoire d'adresse ES:SI+4

- TAB est une table d'octets (TAB DB 100 DUP(?))

MOV DI,2

MOV AL,TAB[DI]

Cette instruction charge le 3ème octet du TAB dans le registre AL.

- TAB est une table de mots (TAB DW 100 DUP(?))

MOV DI,4

MOV AX,TAB[DI]

Cette instruction charge le 3 ème mot du TAB dans le registre AX.

3.5 Mode d'adressage basé indexé

L'offset de l'adresse de l'opérande est la somme d'un registre de base, d'un registre d'index et d'un déplacement optionnel.

Si Rseg n'est pas spécifié, le segment par défaut du registre de base est utilisé :.

Ce modes d'adressage est intéressant dans le cas ou on veut accéder a une structure bidimensionnelle comme le matrices. Dans ce cas le registre de base contient l'adresse de départ du tableau, le registre index contient le numéro de la ligne et le déplacement le numéro de la colonne.

Exemples :

MOV AX,[BX+SI] ; AX est chargé par la mémoire d'adresse DS:BX+SI

MOV AX,[BX+DI+5] ; AX est chargé par la mémoire d'adresse DS:BX+DI+5

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

MOV AX,[BP+DI] ; AX est chargé par la mémoire d'adresse SS:BP+DI

Résumé

Mode D'adressage	Format Opérande	Registre Segment	Exemple
Immédiat	Donnée	Aucun	MOV AX,5
Registre	Registre	Aucun	MOV AX,BX
Directe	Déplacement ou Label	DS	MOV AX,COMPTE
Registre Indirect	[BX]	DS	MOV AX,[BX]
	[BP]	SS	MOV AX,[BP]
	[SI]	DS	MOV AX,[SI]
	[DI]	DS	MOV AX,[DI]
Relatif à une Base	[BX]+Déplacement	DS	MOV AX,[BX]+3
	[BP]+Déplacement	SS	MOV AX,[BP]+3
Direct Indexé	Label + [SI] ou Label[SI]	DS	MOV AL,MSG[SI]
	Label + [DI] ou Label[DI]	DS	MOV BL, MSG+[DI]
Basé Indexé	Label+[BX][SI]	DS	
	Label+[BX][DI]	DS	MOV AX, MATR[BX][DI]
	Label+[BP][SI]	SS	MOV AX, MATR[BP][SI]
	Label+[BP][DI]	SS	

4. La Programmation en assembleur du microprocesseur 8086

Constituion d'un programme .

Un programme assembleur est un ensemble de déclarations constitué de directives et d'instructions

4.1 Les directives

4.1.1 Les directives de données : (EQU; DB; DW ; DD)

4.1.1.1 Déclaration d'une constante

- ***EQU :***

Nom EQU expression ; assigne le nom de l'expression au nom

Exemples :

Pi EQU 3.14 ; valeur constante

4.1.1.2 Déclaration des variables

- ***DB (define byte) :***

[nom] DB expression

Exemple :

- *Nb_max DB 123 ;* déclaration d'une variable initialisée à 123
- *Deux_octets DB 22,102* ; cette décalration réserve deux octets dont le premier octet est initialisé par 22 et le deuxième octet sera initialisé par la valeur 102

- Tab_oct DB 3 DUP(0), -5, -100, 2 DUP(30), ? ; cette table réserve des cases d'octets initialisées avec 0,0,0,-5,-100,30,30, et une case non initialisée.
- *LETTRE* DB 'e' ; l'octet sera initialiser avec le code en ASCII du caractère 'e'.
- Message DB 'Hello world !',13, 10,'$' ; réserve une chaîne de caractères composée des caractères 'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r','l', 'd', ' ', '!', des caractères de code 13 (retour en début de ligne) et 10 (aller à la ligne suivante) et enfin du caractère '$' qui indique la fin de la chaîne de caractères.

- ***DW (define word) :***

[nom] DW expression

DW réserve des mots à deux octets.

- ***DD (define double) :***

[nom] DD expression

DD réserve des mots à quatre octets.

- ***Dup :***

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

- tab DB 100 dup (15) ; 100 octets valant 15

4.1.2 Les directives Word PTR et Byte PTR

Dans certains cas, l'adressage indirect est indéterminé par rapport à la taille qui doit être pris en considération.

Exemple :

Mov [bx], 25h ; range 21 a l'adresse spécifiée par BX

L'assembleur ne sait pas si l'instruction concerne 1, 2 ou 4 octets

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

4.1.3 Les directives de segment et de procédure (Assume ; segment ; proc/endp)

- ***Segment*** : cette directive délimite un segment, elle permet de contrôler la relation entre la génération du code objet et la gestion des segments logiques ainsi générés.

Syntax de la directive

nom_seg SEGMENT [align][combine]['class']

……

……

……

nom_seg ENDS

ENDS marque la fin du segment.

Exemple :

```
DONNEE   SEGMENT
  Message  DB ‘ bonjours a tous$ ‘
DONNEE ENDS
```

- ***Assume :***
  La directive ASSUME permet d'indiquer à l'assembleur quel est le segment de données et celui de codes (etc...), afin qu'il génère des adresses correctes.

4.2 Structure d'un programme source

Comme dans tout programme le fichier source doit être saisi de manière rigoureuse. Chaque définition et chaque instruction doivent ainsi s'écrire sur une nouvelle ligne (pour que l'assembleur puisse différencier les différentes instructions) Le fichier source contient:

1. Un nom du programme sous TITLE sui vie d’un nom. Cette partie est facultative.

2. Une partie pour déclarer une pile qui est définie dans le segment de pile délimité par les directives SEGMENT STACK et ENDS

3. Des définitions de données déclarées par des directives. Celles-ci sont regroupées dans le segment de données délimité par les directives SEGMENT et ENDS

4. Puis sont placées les instructions (qui sont en quelque sorte le coeur du programme), la première devant être précédée d'une étiquette, c'est-à-dire par un nom qu'on lui donne. Celles-ci sont regroupées dans le segment d'instructions délimité par les directives SEGMENT et ENDS

5. Enfin, le fichier doit être terminé par la directive END suivi du nom de l'étiquette de la première instruction (pour permettre au compilateur de connaître la première instruction à exécuter (Les points-virgules marquent le début des commentaires, c'est-à-dire que tous les caractères situés à droite d'un point virgule seront ignorés) Voici à quoi ressemble un fichier source (fichier .ASM):

```
Title  prog01.asm

Pile SEGMENT STACK ; On met les directives pour réserver de l'espace mémoire.

PILE ENDS

Data SEGMENT ; On met les directives de données pour réserver de la mémoire

        ; Pour les variables qui seront utilisées dans le programme.

Data ENDS

Extra SEGMENT ; On met les directives pour déclarer ;
```

```
        ;les variables (les chaînes de Caractères).
Extra ENDS
Code SEGMENT
ASSUME   cs : code, ds, data, es : pile :ss :pile
PROG:
        Mov AX,Data
        Mov DS,AX
        Mov AX,Extra
        Mov ES,AX
        Mov AX,pile
        Mov SS,AX

; mettre les instructions du programme
Code ENDS
END PROG
```

Comme nous l'avons indiquer au paragraphe précédent, c'est la directive ASSUME qui permet d'indiquer à l'assembleur où se situe le segment de données et le segment de code. Puis il s'agit d'initialiser le segment de données DS (même chose pour : ES et SS)

```
MOV AX, nom_du_segment_de_données;
MOV DS, AX
```

4.3 Les instructions

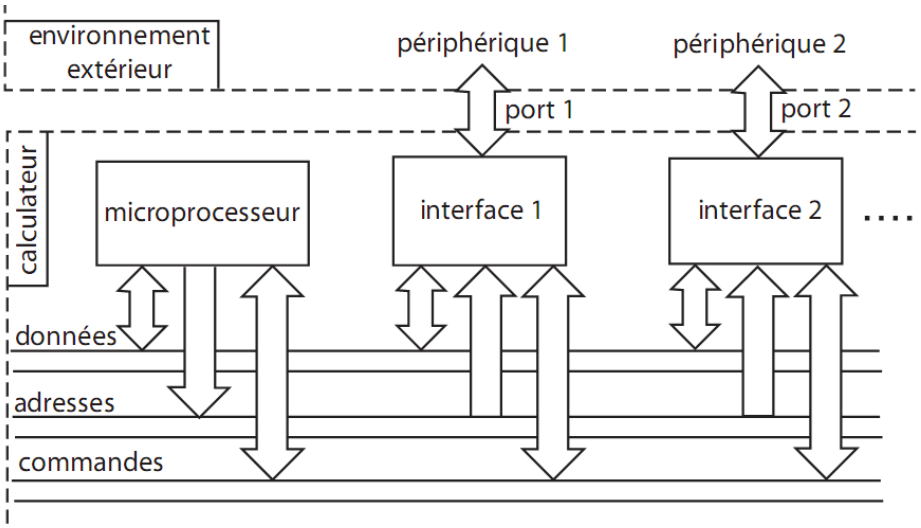
Chaque microprocesseur reconnait un ensemble d'instructions appelé jeu d'instructions (Instruction Set) fixé par le constructeur. Pour les microprocesseurs classiques, le nombre d'instructions reconnues varie entre 75 et 150 (microprocesseurs CISC : Complex Instruc-tion Set Computer). Il existe aussi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

Le 8086 possède 92 types d'instructions de base qu'on peut diviser comme suit :

Voir le fichier des instructions

5. Les entrée sorties

Une interface d'entrées/sorties est un circuit intégré permettant au microprocesseur de communiquer avec l'environnement extérieur (périphériques) : clavier, écran, imprimante,modem, disques, processus industriel, ...Les interfaces d'E/S sont connectées au microprocesseur à travers les bus d'adresses, de donnéees et de commandes. Les jonctions entre les interfaces et les périphériques sont appelés port.



Exemple :

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

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

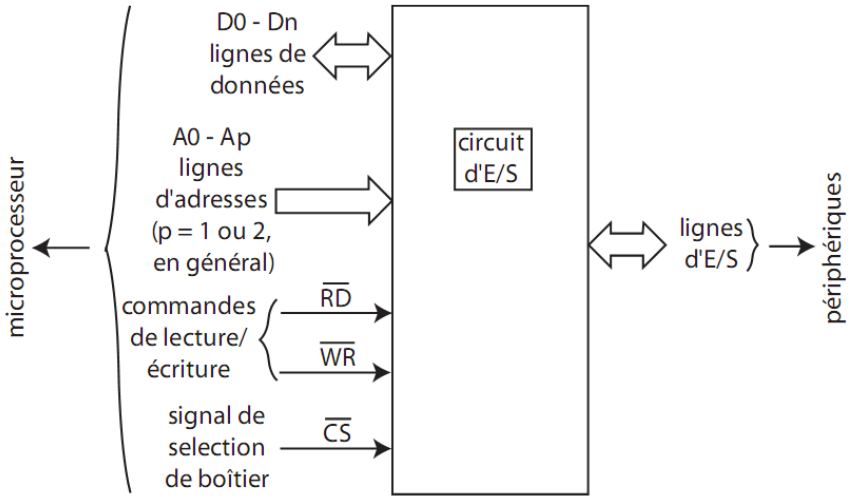


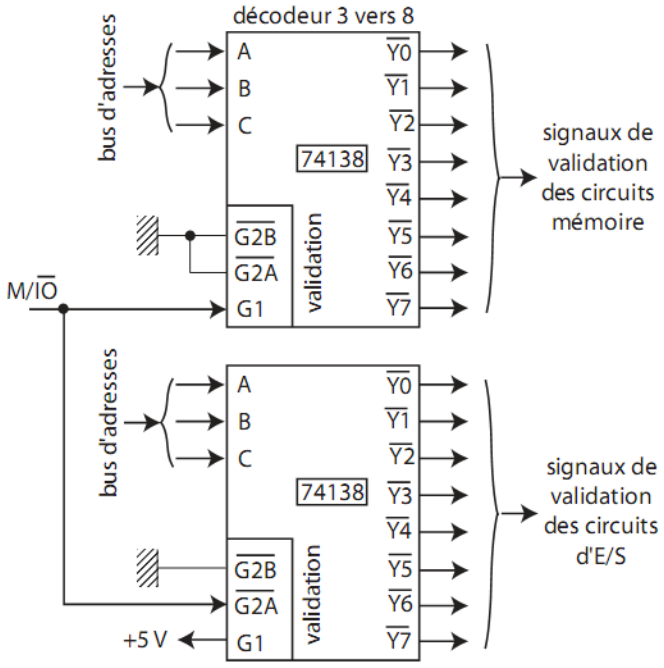
Figure : interface d'Entréé/Sortie

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

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

Le signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S est la ligne $\boldsymbol{M}$ / $\overline{\boldsymbol{IO}}$:

- pour un accès à la mémoire , $M\ /\overline{IO} = 1$:
- pour un accès aux port d'E/S, $M\ /\overline{IO} = 0$



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

Lecture d'un port d'E/S :

- si l'adresse du port d'E/S est sur un octet :
  IN al, adresse : lecture d'un port sur 8 bits
  IN ax, adresse : lecture d'un port sur 16 bits
- si l'adresse du port d'E/S est sur deux octets :
  IN al, dx: lecture d'un port sur 8 bits

  IN ax, dx: lecture d'un port sur 16 bits

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

Ecriture d'un port d'E/S :

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

  OUT adresse, al : lecture d'un port sur 8 bits

  OUT adresse, ax : lecture d'un port sur 16 bits

- si l'adresse du port d'E/S est sur deux octets :
  OUT dx,al: lecture d'un port sur 8 bits

  OUT dx,ax : lecture d'un port sur 16 bits

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

Exemples :

- lecture d'un port d'E/S sur 8 bits à l'adresse 4221h :

```
mov dx, 4221h
in al,dx
```

- Ecriture de la valeur AF37 H dans le port d'E/S sur 16 bits à l'adresse 52 h :

```
Mov ax, AF37H

OUT 52 h, ax
```

5.1 Les interfaces d'entrées/sorties

Afin de pouvoir piloter les périphériques avec un µP de la famille x86 Intel a conçu les principales interfaces suivantes :

- Interface parallèle programmable 8255
- Contrôleur d'interruptions programmable 8259
- Interface série programmable 8250
- Le controleur DMA 8237

6. Les Interruptions

Imaginer qu'un événement extérieur demande un traitement particulier du µp. Cet événement peut intervenir à n'importe quel instant. Le processeur doit être capable d'abandonner le programme en cours et servir la routine correspondant à cette demande d'intervention. Une fois cette demande traitée, le µp reprend le programme qui était en cours d'exécution sans aucune perte de donnée ou d'état. Sachant que la routine appelée va de toute façon, utiliser les mêmes registres et les mêmes drapeaux que le programme principal, il faut sauvegarder l'état complet du système avant de commencer la routine d'interruption et les récupérer juste avant de revenir au programme principal.

6.1.1 L'interruption 21h du DOS

Normalement le DOS est relativement de haut niveau et ne dépend pas de la machine. Il fait souvent appel au bios qui fonctionne à un niveau plus proche de la machine. L'interruption 21h peut réaliser plusieurs fonctions différentes. Nous ne citerons ici que celles que nous utiliserons.

- Fonction 02

Cette fonction permet d'écrire un caractère. Le caractère est envoyé vers la sortie standard, l'écriture peut donc être redirigée dans un fichier.

Paramètres : AH = 02h

DL = Caractère à écrire

- Fonction 09

Cette fonction permet en un seul appel, d'écrire une suite de caractères.

Paramètres : AH = 09h

DX = Adresse de la chaîne de caractères

La chaîne doit être terminée par le caractère $

- Fonction 07

Cette fonction permet de lire un caractère du clavier sans qu'il n'y ait d'écho à l'écran.

Paramètre passé : AH = 07

Paramètre retourné : AL = caractère lu

Les touches fonction retourne 2 caractères, d'abord un octet nul, puis le code étendu

de la touche, il faut donc faire 2 appels consécutifs de la fonction 07.

- Fonction 0Ah

Permet de saisir une chaîne de caractère au clavier. La saisie s'arrête quand on tape la touche de retour , le caractère CR (13) est mémorisé avec la chaîne Paramètres :

DX : adresse du buffer (zone mémoire tampon) où seront stockés la longueur de la chaîne ainsi que la chaîne saisie

[DX] : longueur max. avant d'appeler la fonction, il faut placer dans le premier octet du buffer la longueur max à ne pas dépasser, il faut compter le CR.

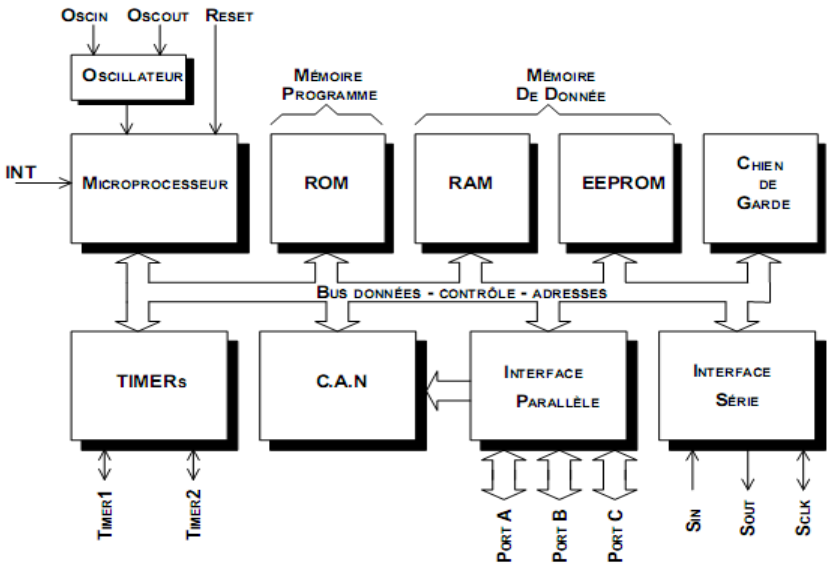
Une fois la saisie terminée, la fonction place dans le deuxième octet du buffer le nombre de caractère effectivement saisi. La chaîne saisie est placée tous de suite derrière.

Quelques interruptions									
		AH	AL	BH	BL	CX	DH	DL	commentaire
INT 10h	Mode écran	00	mode						
	Ecrit char	09	car	page	coul t/g	rep			!Curs !CarSpec
	Ecrt char	0A	car	page		rep			!Curs !CarSpec
	Ecrt char	0E	car		Coul g				Curs CarSpec
	Pos Curs	02		page			ligne	col	
	Choisir page	05	page						
	Allumer pixel	0Ch	couleur	page		x	y		
	50 lignes	11h	12h		30h				
INT 21h	Ecrt char	02						car	^C^B -> int 23h
	Ecrit chaîne	09					adresse		'$'= fin chaîne
	Lit car !écho	07	Car lu						
	Lit chaine	0Ah					adresse		Voir cours
	kbhit	0B	0 : non FF : oui						

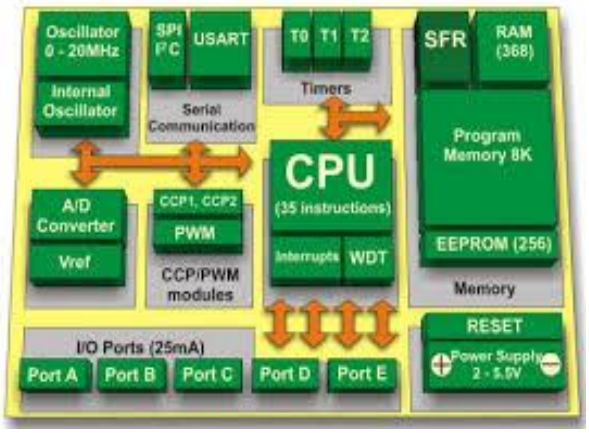
7. Les microcontrôleurs

Un microcontrôleur est un composant réunissant sur une seule puce un microprocesseur, divers dispositifs d'entrées/sorties et de contrôle d'interruptions ainsi que de la mémoire. Il intègre également un certain nombre de périphériques spécifiques des domaines ciblés (bus série, interface parallèle, convertisseur analogique numérique, ...). Les microcontrôleurs améliorent donc l'intégration et le coût (lié à la conception et à la réalisation) d'un système à base de microprocesseur en rassemblant les éléments essentiels d'un tel système dans un seul circuit intégré.

Un microcontrôleur possède généralement les éléments suivant :



- **Des convertisseurs analogiques-numériques (CAN) :** Ils permettent de codifier un signal analogique (constituer de grandeurs électriques) en donnant des paquets d'informations binaires.
- **Des convertisseurs numériques-analogiques (CNA)** : effectuent l'opération inverse d'un CAN c'est-à-dire, il permet de reconstituer un signal analogique à partir des donnés numériques.
- **Des générateurs de signaux à modulation de largeur d'impulsion (MLI, ou en anglais, PWM pour Pulse Width Modulation) :** il permet de modifier la largeur d'impulsion.
- **Des timers /compteurs :** compteurs d'impulsions d'horloge interne ou d'événements externes); son rôle est de permettre la synchronisation des opérations que microprocesseur (ou le microcontrôleur) est chargé d'effectuer.
- **Des chiens de garde (watchdog)** : Le watchdog, ou chien de garde est un mécanisme de protection de votre programme. Il sert à surveiller si le programme s'exécute toujours dans l'espace et dans le temps que vous lui avez attribués.
- **Des comparateurs comparent deux tensions électriques** : à titre d'exemple les modules CCP1 et CCP2 implantés dans un microcontrôleur 16F877.
- **Des contrôleurs de bus de communication** : UART, I2C, SSP, CAN, RS232, USB, Ethernet, etc.



µC :PIC 16F877