

Institut Supérieur des Études Technologiques de Radès
Département de Génie Électrique

MICRO PROCESSEUR

Support de cours

4^{ème} niveau Génie Électrique

Dr J.Y. Haggège
Ingénieur ENIT

Aggrégé de Génie Électrique
Technologue à l'ISSET de Radès

2003

Table des matières

1	Historique et évolution des ordinateurs	1
1.1	« Préhistoire » des ordinateurs.....	1
1.2	Machines électromécaniques	1
1.3	Machines électroniques.....	1
1.4	Machines actuelles	2
2	Architecture et fonctionnement d'un microprocesseur	3
2.1	Structure d'un ordinateur	3
2.2	Organisation de la mémoire centrale.....	4
2.3	Circulation de l'information dans un ordinateur	5
2.4	Description matérielle d'un microprocesseur	6
2.5	Fonctionnement d'un microprocesseur.....	6
3	Les mémoires	11
3.1	Mémoires ROM et RAM.....	11
3.2	Schéma fonctionnel d'une mémoire	11
3.3	Interface microprocesseur/mémoire	12
3.4	Chronogrammes de lecture/écriture en mémoire	13
3.5	Connexion de plusieurs bits mémoire.....	14
3.6	Décodage d'adresses	16
3.7	Classification des mémoires.....	17
4	Le microprocesseur Intel 8086	19
4.1	Description physique du 8086	19
4.2	Schéma fonctionnel du 8086	20
4.3	Description et utilisation des signaux du 8086	20
4.4	Organisation interne du 8086.	26
4.5	Gestion de la mémoire par le 8086.	29
4.6	Le microprocesseur 8088	32
5	La programmation en assembleur du microprocesseur 8086	33
5.1	Généralités	33
5.2	Les instructions de transfert	33
5.3	Les instructions arithmétiques	37
5.4	Les instructions logiques	38

5.5	Les instructions de branchement.....	42
5.6	Méthodes de programmation.....	48
6	Les interfaces d'entrées/sorties	51
6.1	Définitions.....	51
6.2	Adressage des ports d'E/S	52
6.3	Gestion des ports d'E/S par le 8086.	53
6.4	L'interface parallèle 8255.	55
6.5	L'interface série 8250.	60
7	Les interruptions	71
7.1	Définition d'une interruption.....	71
7.2	Prise en charge d'une interruption par le microprocesseur	72
7.3	Adresses de sous-programmes d'interruptions.	73
7.4	Les interruptions du 8086	74
7.5	Le contrôleur programmable d'interruptions 8259	75
	Annexe: Jeu d'instructions du 8086	77
	Bibliographie	81

Chapitre 1

Historique et évolution des ordinateurs

1.1 « Préhistoire » des ordinateurs

Les premières machines à calculer étaient purement mécaniques: bouliers, abaques, ... (antiquité).

Première vraie machine à calculer : Pascal, 1642, machine à additionner.

Machine à multiplier: Leibniz, 1694, basée sur les travaux de John Napier (1617, logarithmes).

Première machine programmable: métier à tisser, Jacquard, XVIII^{ème} siècle, machine à cartes perforées.

Machine programmable universelle : Babbage, XIX^{ème} siècle, non réalisable avec les technologies de l'époque (machines à vapeur), principe des machines actuelles.

1.2 Machines électromécaniques

Machine à calculer à cartes perforées: Hermann Hollerith, 1885, facilite le recensement américain.

Machines industrielles pour la comptabilité et les statistiques. Ces machines sont à base de relais électromécaniques (Aiken et Stibitz, 1936-1939).

1.3 Machines électroniques

Première machine à calculer électronique : ENIAC, 1944, Eckert et Mauchly, 18000 tubes électroniques, machine à programme câblé.

Machine à programme enregistré : John Von Neumann, 1946, les instructions sont enregistrées dans la mémoire du calculateur : ordinateur.

Premier ordinateur commercialisé : SSEC d'IBM, 1948.

Ordinateur à transistors : 1963, PDP5 de Digital Equipment Corporation (DEC), introduction des mémoires à ferrites : mini-ordinateurs.

Micro-ordinateurs : 1969-70, utilisation des circuits intégrés LSI.

Premier microprocesseur: Intel, 1971, microprocesseur 4004 puis 8008, premier micro-ordinateur : le Micral, 1973, France, puis l'Altair, 1975, Etats-Unis.

Autres microprocesseurs 8080 et 8085 d'Intel, 6800 de Motorola, Z80 de Zilog: microprocesseurs 8 bits, début des années 1980.

Microprocesseurs 16 bits : 8086/8088 d'Intel, 68000 de Motorola.

Microprocesseurs 32 bits en 1986 : 80386 d'Intel et 68020 de Motorola.

Fabrication en grandes séries des micro-ordinateurs: 1977, Apple, Commodore, Tandy.

IBM PC + MS-DOS (Microsoft) en 1981.

1.4 Machines actuelles

Ordinateurs de plus en plus puissants, basés sur des microprocesseurs performants : Pentium, Power PC, ...

Nouvelles architectures de microprocesseurs : RISC.

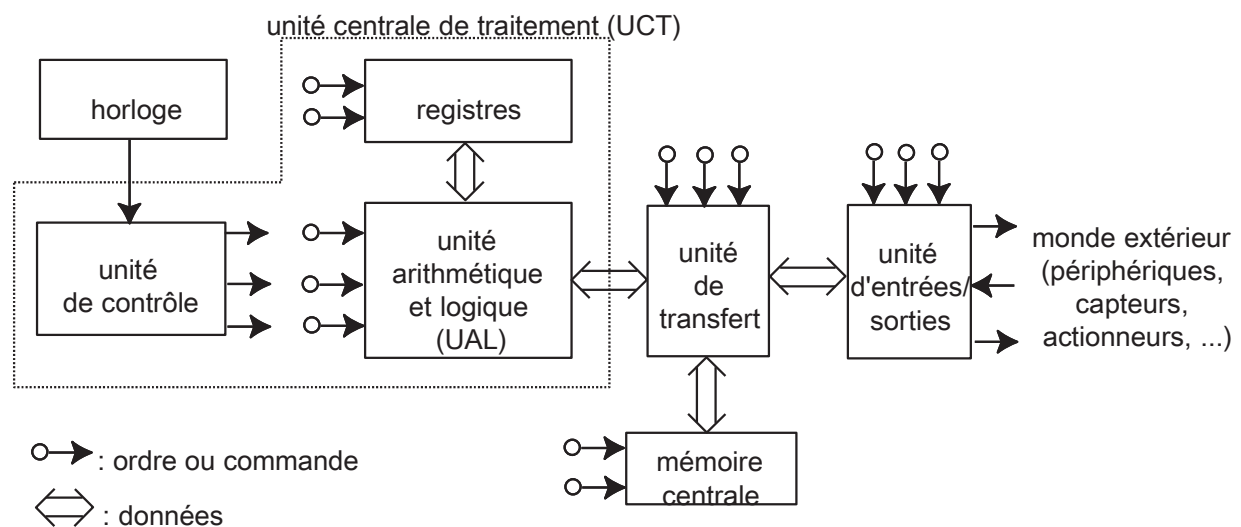
Applications multimédia, réseaux, ...

Systèmes embarqués : microcontrôleurs, processeurs de traitement de signal (DSP), ...

Chapitre 2

Architecture et fonctionnement d'un microprocesseur

2.1 Structure d'un calculateur



L'élément de base d'un calculateur est constitué par l'unité centrale de traitement (UCT, CPU: Central Processing Unit). L'UCT est constituée de :

- d'une unité arithmétique et logique (UAL, ALU: Arithmetic and Logic Unit): c'est l'organe de calcul du calculateur;
- de registres: zones de stockage des données de travail de l'UAL (opérations, résultats intermédiaires);
- d'une unité de contrôle (UC, CU: Control Unit): elle envoie les ordres (ou commandes) à tous les autres éléments du calculateur afin d'exécuter un programme.

La mémoire centrale contient :

- le programme à exécuter: suite d'instructions élémentaires;
- les données à traiter.

L'unité d'entrées/sorties (E/S) est un intermédiaire entre le calculateur et le monde extérieur.

L'unité de transfert des données support matériel de la circulation des données.

Les échanges d'ordres et de données dans le calculateur sont synchronisés par une horloge qui délivre des impulsions (signal d'horloge) à des intervalles de temps fixes.

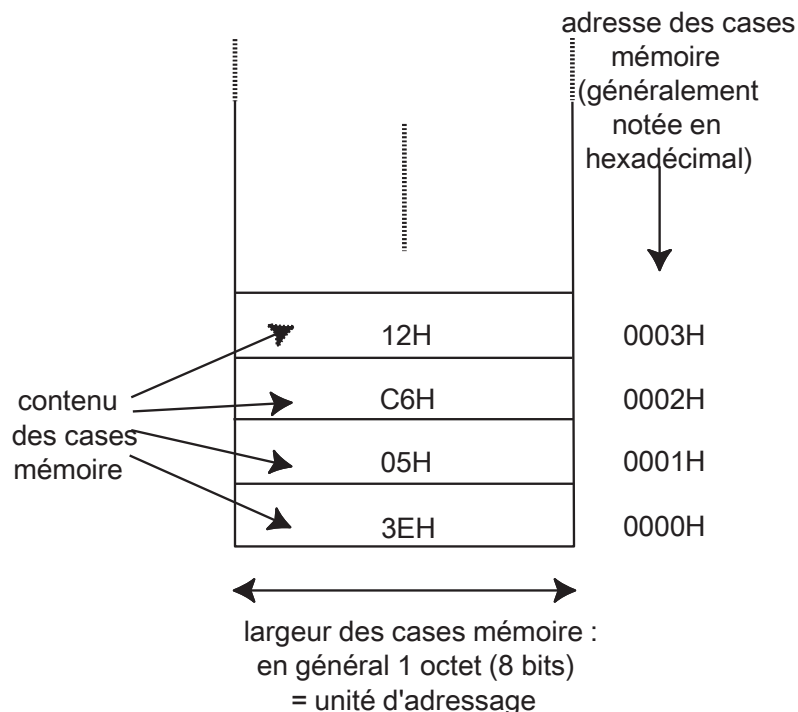
Définition: un microprocesseur consiste en une unité centrale de traitement (UAL + registres + unité de contrôle) entièrement contenue dans un seul circuit intégré. Un calculateur construit autour d'un microprocesseur est un microcalculateur ou un microordinateur.

Remarque: un circuit intégré qui inclut une UCT, de la mémoire et des périphériques est un microcontrôleur.

2.2 Organisation de la mémoire centrale

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

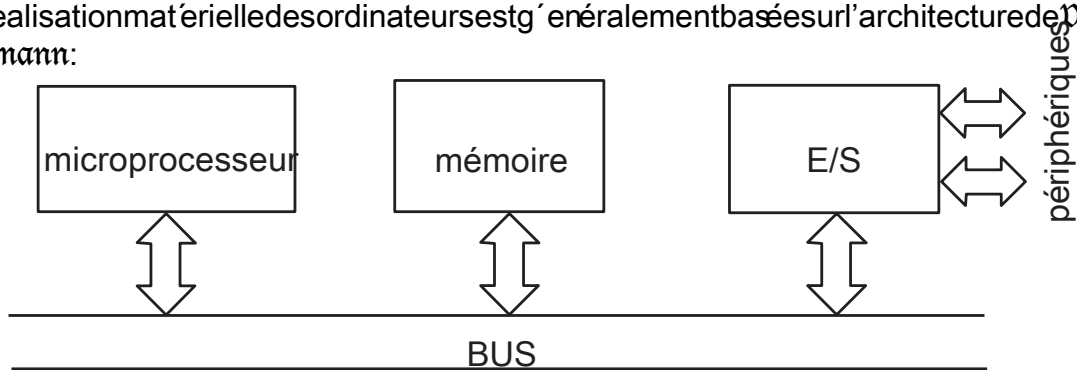
Représentation:



Une case mémoire peut être lue ou écrite par le microprocesseur (cases mémoires vivantes) ou bien seulement lue (cases mémoires mortes).

2.3 Circulation de l'information dans un ordinateur

L'architecture matérielle des ordinateurs est généralement basée sur l'architecture de Von Neumann :

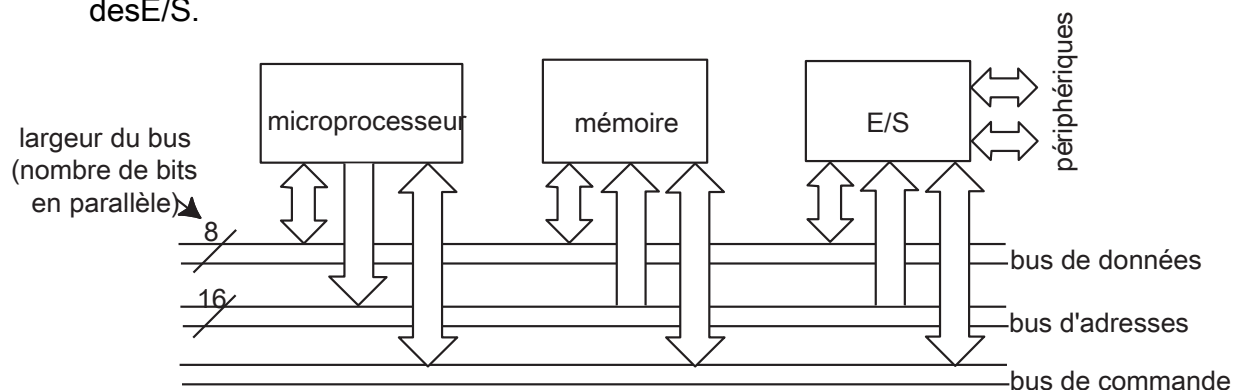


Le microprocesseur échange des informations avec la mémoire et l'unité E/S, sous forme de mots binaires, au moyen d'un ensemble de connexions appelé bus. Un bus permet de transférer des données sous forme parallèle, c'est-à-dire en faisant circuler n bits simultanément.

Les microprocesseurs peuvent être classés selon la longueur maximale des mots binaires qu'ils peuvent échanger avec la mémoire et les E/S : microprocesseurs 8 bits, 16 bits, 32 bits, ...

Le bus peut être composé de trois bus distincts :

- le bus d'adresses permet au microprocesseur de spécifier l'adresse de la case mémoire à lire ou à écrire ;
- le bus de données permet les transferts entre le microprocesseur et la mémoire ou les E/S ;
- le bus de commande transmet les ordres de lecture et d'écriture de la mémoire et des E/S.



Remarque : les bus de données et de commande sont bidirectionnels, le bus d'adresse est unidirectionnel : seul le microprocesseur peut délivrer des adresses (il existe une dérogation pour les circuits d'accès direct à la mémoire, DMA).

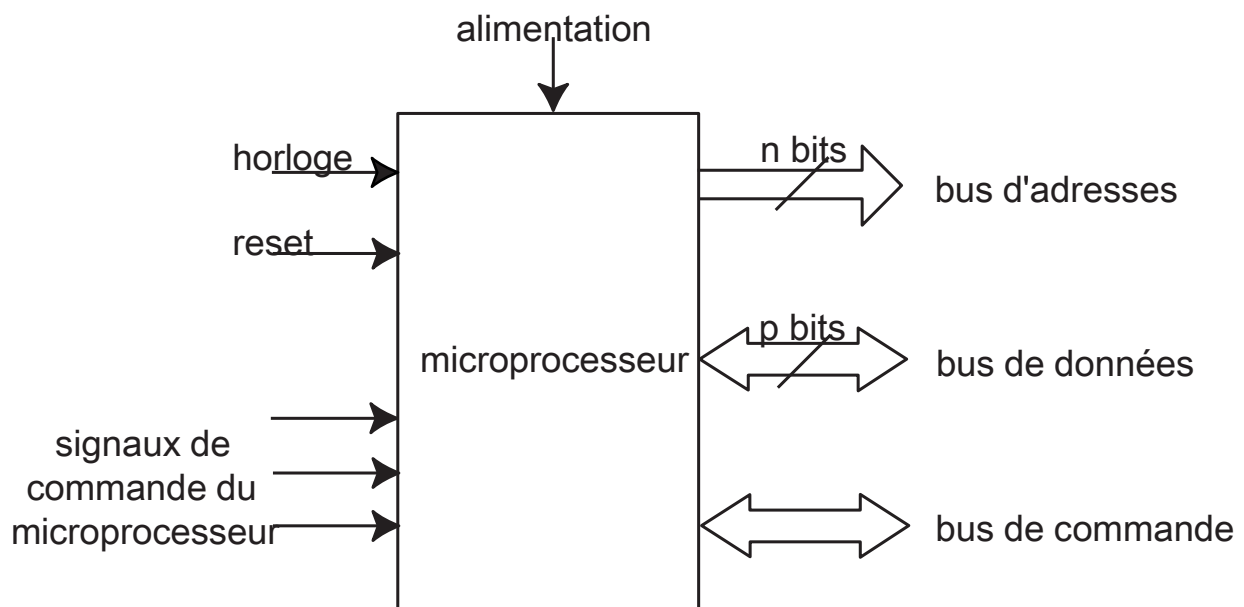
2.4 Description matérielle d'un microprocesseur

Un microprocesseur se présente sous la forme d'un circuit intégré équipé d'un nombre généralement important de broches. Exemples :

- Intel 8085, 8086, Zilog Z80 : 40 broches, DIP (Dual In-line Package);
- Motorola 68000 : 64 broches, DIP;
- Intel 80386 : 196 broches, PGA (Pin Grid Array).

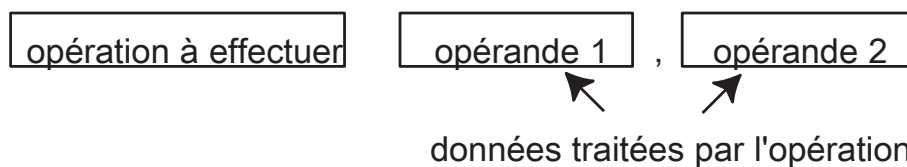
Technologies de fabrication : NMOS, PMOS, CMOS.

On peut représenter un microprocesseur par son schéma fonctionnel :



2.5 Fonctionnement d'un microprocesseur

Un microprocesseur exécute un programme. Le programme est une suite d'instructions stockées dans la mémoire. Une instruction peut être codée sur un ou plusieurs octets. Format d'une instruction :

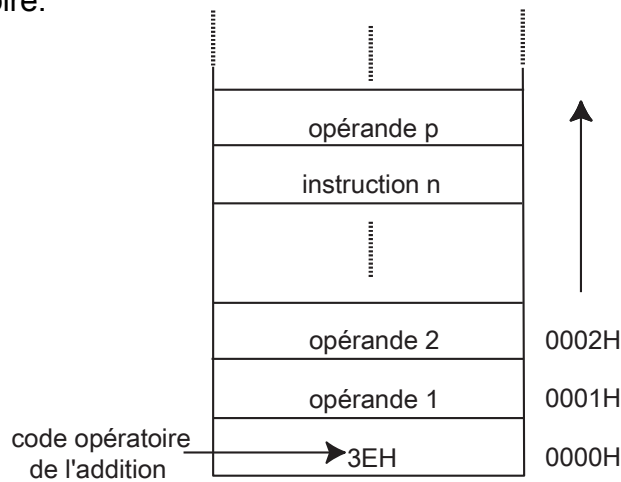


Exemple :

ADDITIONNER casemémoire1, casemémoire2

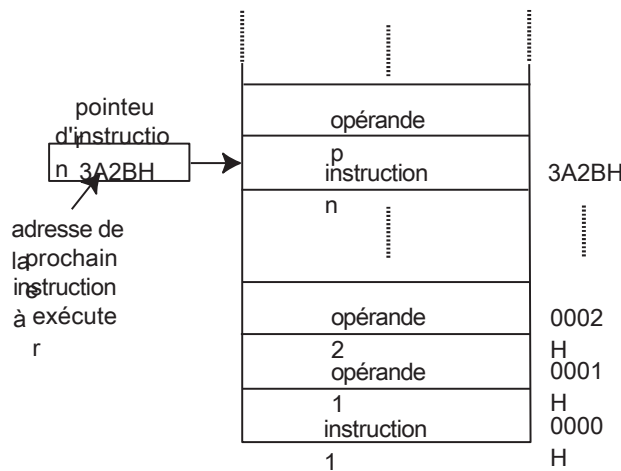
opération opérandes

Rangement en mémoire:



Pour exécuter les instructions dans l'ordre établi par le programme, le microprocesseur doit savoir à chaque instant l'adresse de la prochaine instruction à exécuter. Le microprocesseur utilise un registre contenant cette information. Ce registre est appelé **pointeur d'instruction (IP: Instruction Pointer)** ou **compteur d'instructions** ou **compteur ordinal**.

Exemple:

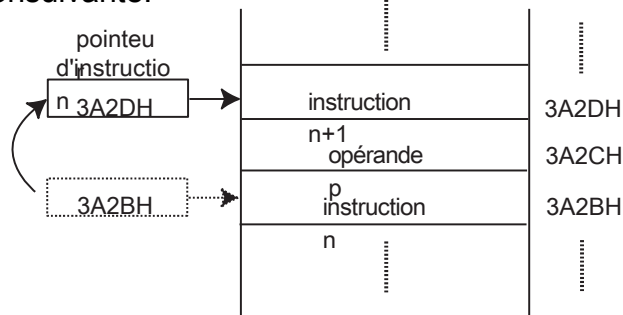


Remarque: la valeur initiale du pointeur d'instruction est fixée par le constructeur du microprocesseur. Elle vaut une valeur bien définie à chaque mise en tension du microprocesseur ou bien lors d'une remise à zéro (reset).

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

Remarque: pour exécuter une instruction, l'UAL utilise des **registres de travail**, exemple: l'**accumulateur**, registre temporaire recevant des données intermédiaires.

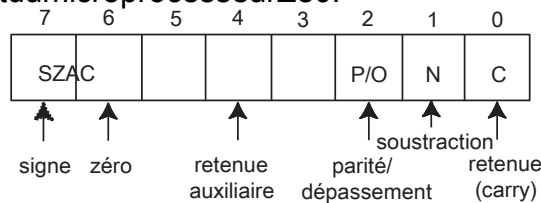
Pendant que l'instruction est décodée, le pointeur d'instruction est incrémenté de façon à pointer vers l'instruction suivante :



puis le processus de lecture et de décodage des instructions recommence.

À la suite de chaque instruction, un registre du microprocesseur est actualisé en fonction du dernier résultat : c'est le registre d'état du microprocesseur. Chacun des bits du registre d'état est un indicateur d'état ou *flag* (drapeau).

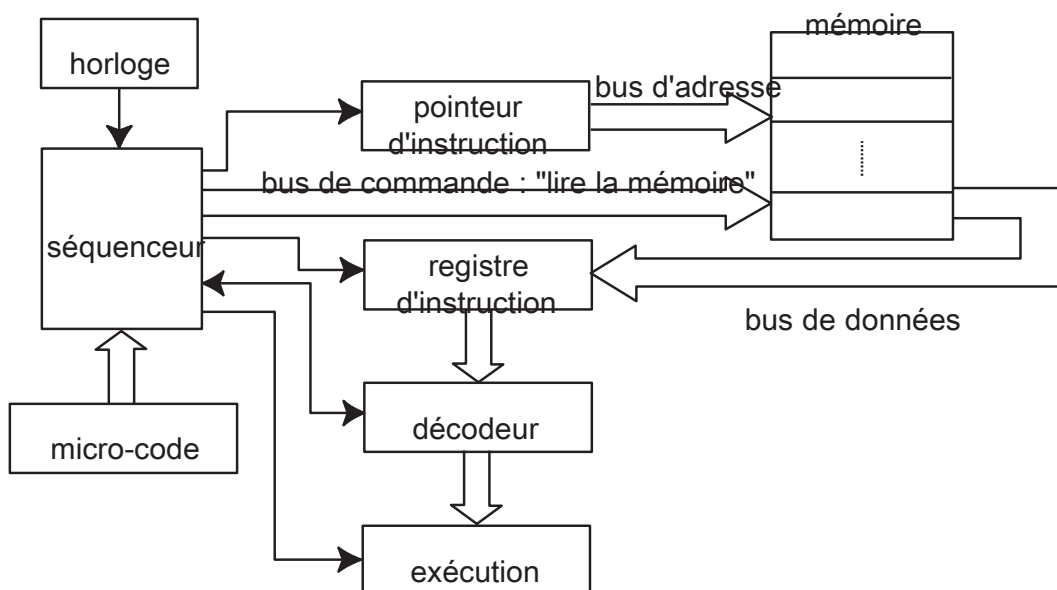
Exemple : registre d'état du microprocesseur Z80 :



Les indicateurs d'état sont activés lorsqu'une certaine condition est remplie, exemple : le flag Z est mis à 1 lorsqu'une opération donne un résultat nul, le flag C est mis à 1 lorsqu'un résultat d'une addition possède une retenue, ...

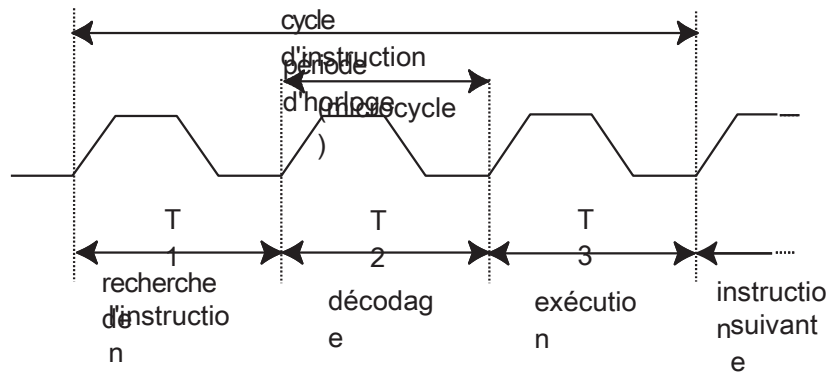
Les indicateurs d'état sont utilisés par les instructions de saut conditionnels : en fonction de l'état d'un (ou plusieurs) flags, le programme se poursuit de manière différente.

Toutes ces étapes (lecture de l'instruction, décodage, exécution) sont synchronisées par une séquenceur qui assure le bon déroulement des opérations :



Pour exécuter le programme contenu dans la mémoire centrale, le séquenceur du microprocesseur exécute lui-même un programme appelé microcode, contenu dans une mémoire morte à l'intérieur du microprocesseur.

Le séquenceur est dirigé par une horloge qui délivre un signal de fréquence donnée permettant d'enchaîner les différentes étapes de l'exécution d'une instruction :

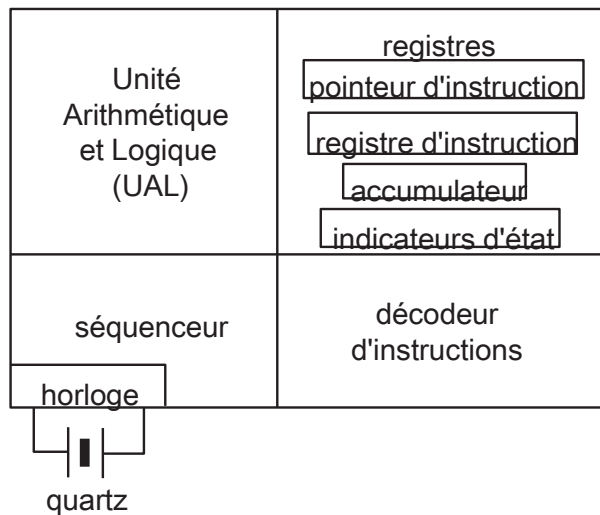


Chaque instruction est caractérisée par le nombre de périodes d'horloge (ou microcycles) qu'elle utilise (donnée fournie par le fabricant du microprocesseur).

Exemple: horloge à 5MHz, période $T = 1/f = 0,2 \mu s$. Si l'instruction s'exécute en 3 microcycles, la durée d'exécution de l'instruction est: $3 \times 0,2 = 0,6 \mu s$.

L'horloge est constituée par un oscillateur à quartz dont les circuits peuvent être internes ou externes au microprocesseur.

Structure complète d'un microprocesseur simple: pour fonctionner, un microprocesseur nécessite donc au minimum les éléments suivants:



Chapitre 3

Les mémoires

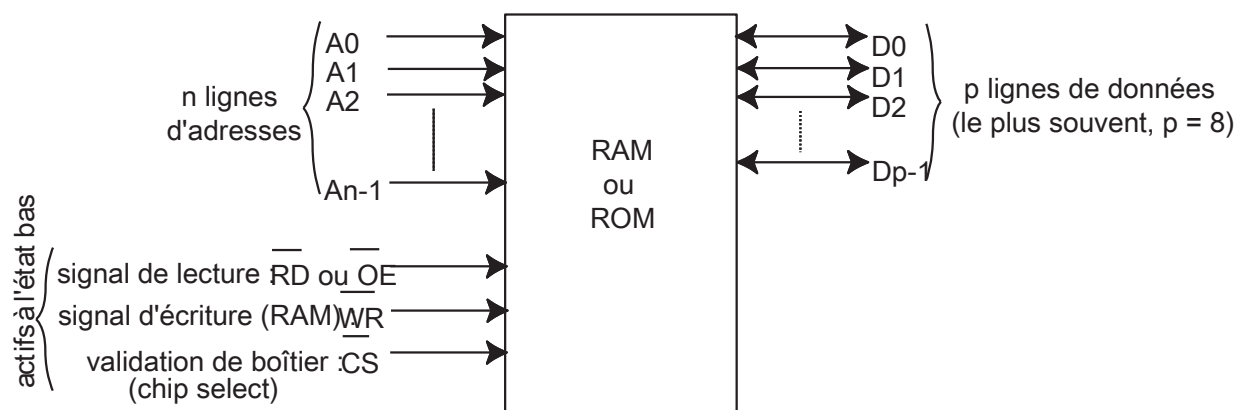
3.1 Mémoires ROM et RAM

On distingue deux types de mémoires :

- les **mémoires vivres** (RAM: Random Access Memory) ou **mémoires volatiles**. Elles perdent leur contenu en cas de coupure d'alimentation. Elles sont utilisées pour stocker temporairement des données et des programmes. Elles peuvent être lues et écrites par le microprocesseur;
- les **mémoires mortes** (ROM: Read Only Memory) ou **mémoires non volatiles**. Elles conservent leur contenu en cas de coupure d'alimentation. Elles ne peuvent être lues par le microprocesseur (pas de possibilité d'écriture). On les utilise pour stocker des données et des programmes de manière définitive.

Les mémoires sont caractérisées par leur **capacité** : nombre total de cases mémoire contenues dans un même boîtier.

3.2 Schéma fonctionnel d'une mémoire

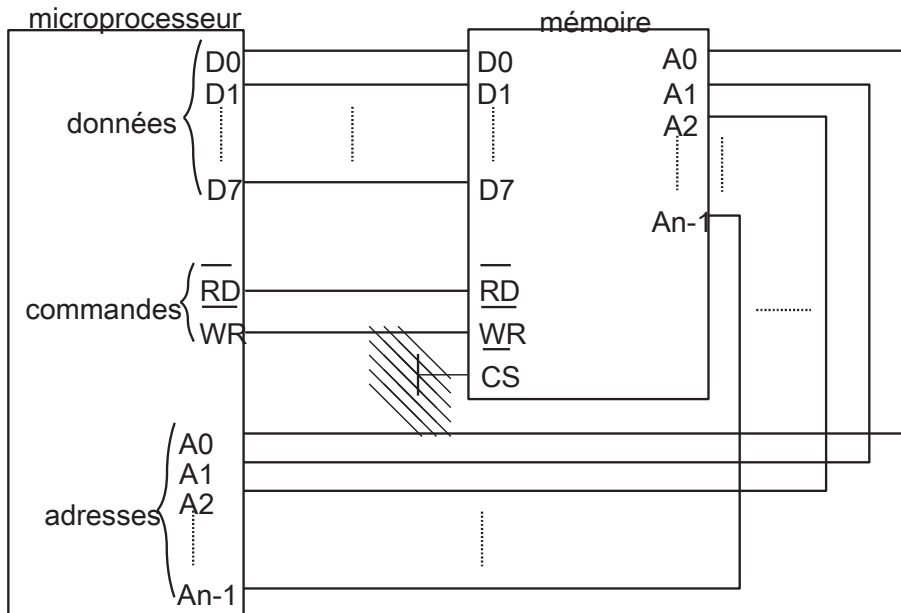


Le nombre de lignes d'adresses dépend de la capacité de la mémoire : n lignes d'adresses permettent d'adresser 2^n cases mémoire. 8 bits d'adresses permettent d'adresser 256 oc-

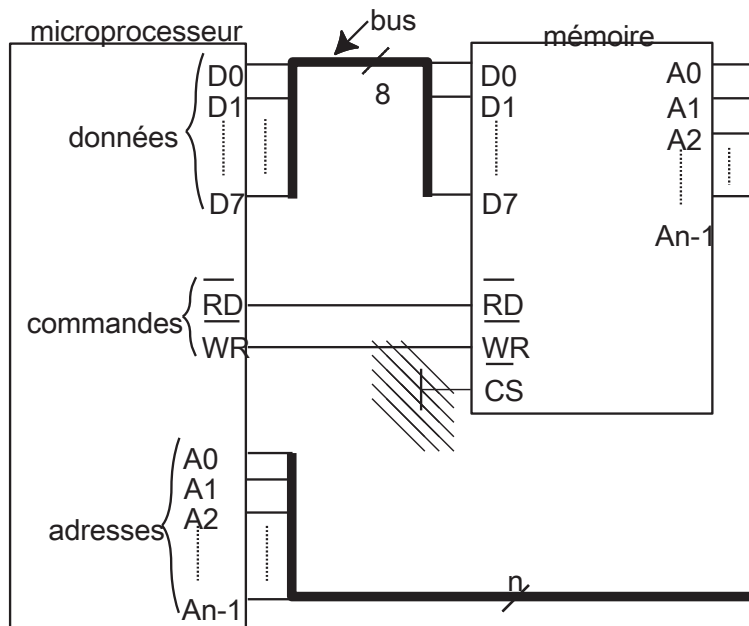
tets, 16 bits d'adresses permettent d'adresser 65536 octets (=64Ko),...

Exemple: mémoire RAM 6264, capacité = 8K × 8 bits: 13 broches d'adresses A0` aA12, 2¹³ = 8192 = 8Ko.

3.3 Interface microprocesseur/mémoire



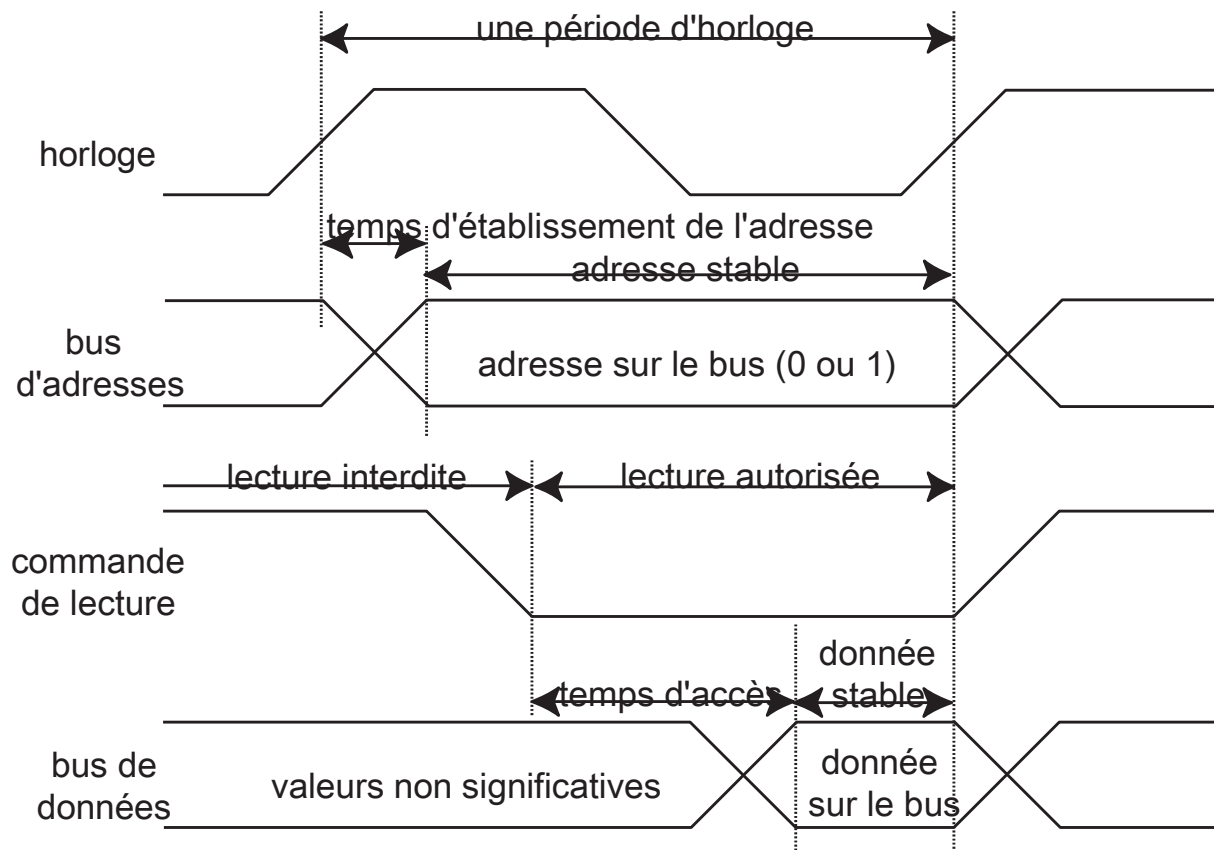
Représentation condensée (plus pratique):



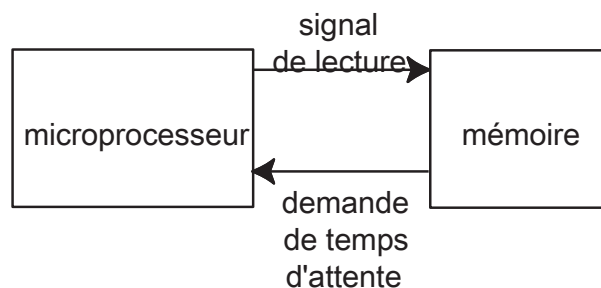
3.4 Chronogrammes de lecture / écriture en mémoire

Une caractéristique importante des mémoires est leur temps d'accès : c'est le temps qui s'écoule entre l'instant où l'adresse de la case mémoire est présentée sur le bus d'adresses et celui où la mémoire place la donnée demandée sur le bus de données. Ce temps varie entre 50 ns (mémoires rapides) et 300 ns (mémoires lentes).

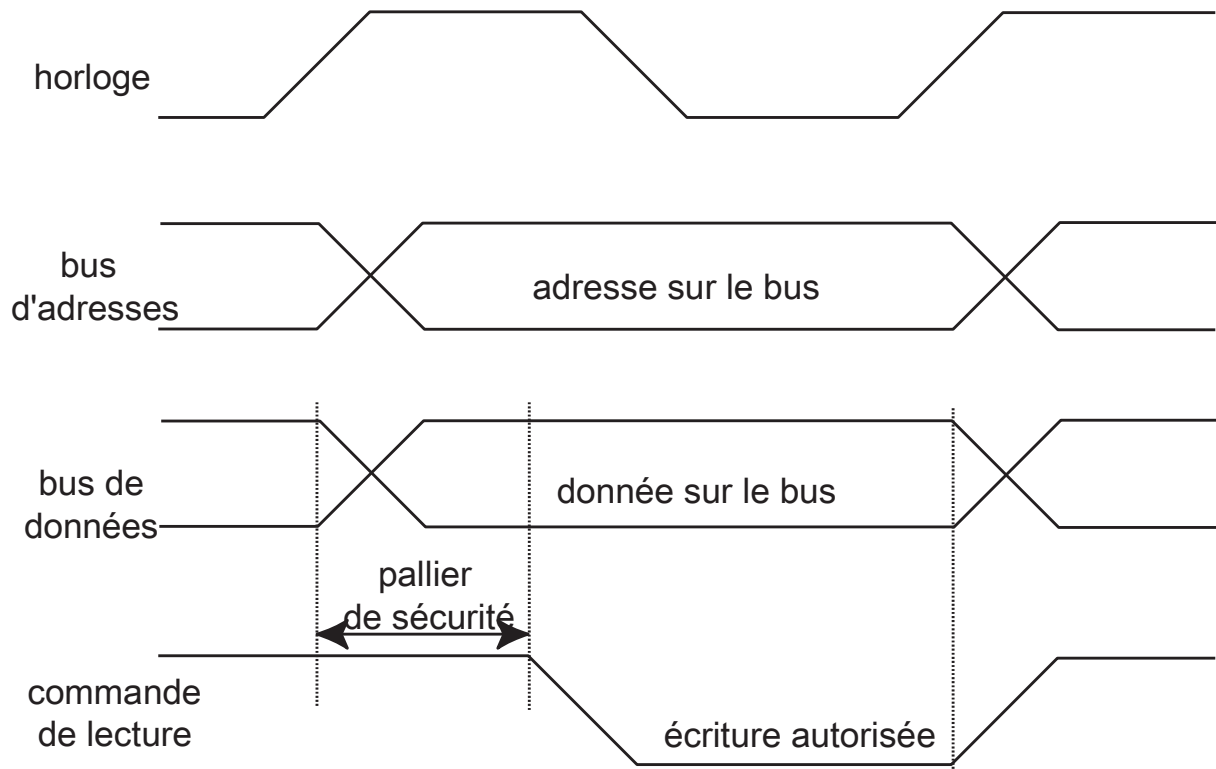
Chronogramme de lecture en mémoire :



Remarque : si le temps d'accès d'une mémoire est supérieur à une période d'horloge (mémoire lente), le microprocesseur peut accorder à la mémoire un temps supplémentaire (une ou plusieurs périodes d'horloge), à la demande de celle-ci. Ce temps supplémentaire est appelé temps d'attente (waittime: T_w):



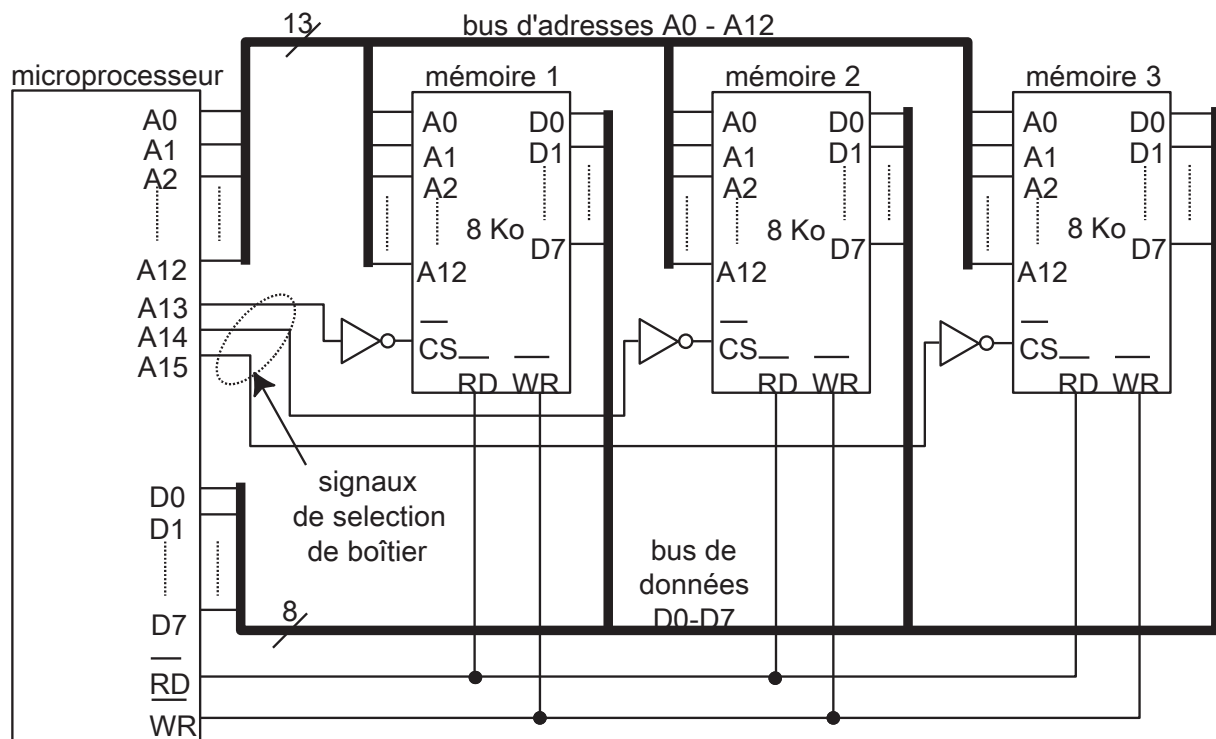
Chronogramme d'écriture en mémoire:



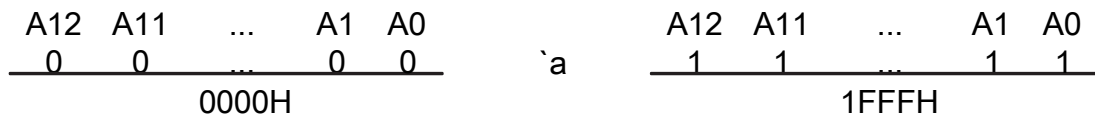
3.5 Connexion de plusieurs boîtiers mémoire sur le bus d'un microprocesseur

Les boîtiers mémoire possèdent une broche notée CS: Chip Select. Lorsque cette broche est active (état bas), le circuit peut être lu ou écrit. Lorsqu'elle est inactive (état haut), le circuit est exclu du service: ses broches de données D0 à D7 passent à l'état de haute impédance: tout se passe comme si la mémoire était déconnectée du bus de données du microprocesseur, d'où la possibilité de connecter plusieurs boîtiers mémoire sur un même bus: un seul signal CS doit être actif à un instant donné pour éviter les conflits entre les différents boîtiers.

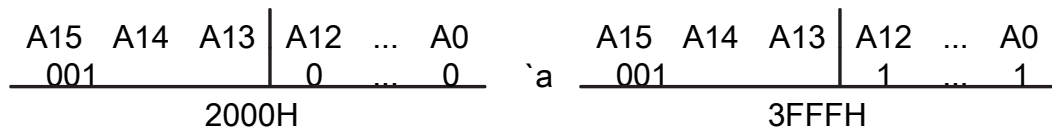
Exemple: connexion de trois boîtiers mémoire d'une capacité de 8 Ko chacun (13 lignes d'adresses) sur un bus d'adresse de 16 bits:



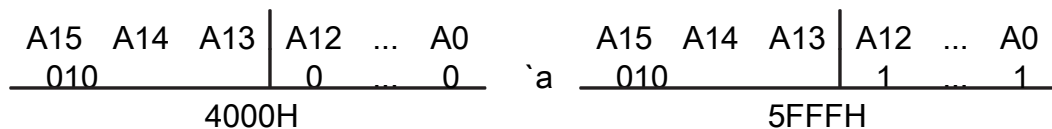
Dans un même boîtier, une case mémoire est désignée par les bits d'adresses A0 à A12:



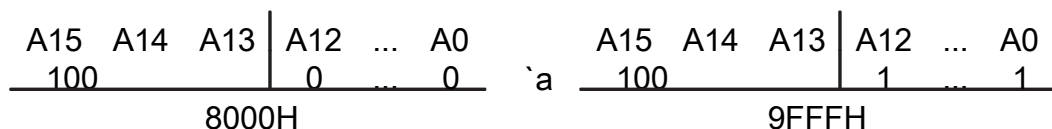
Pour atteindre la mémoire n°1, il faut mettre à 1 le bit A13 et à 0 les bits A14 et A15. Le plage d'adresses occupée par cette mémoire est donc:



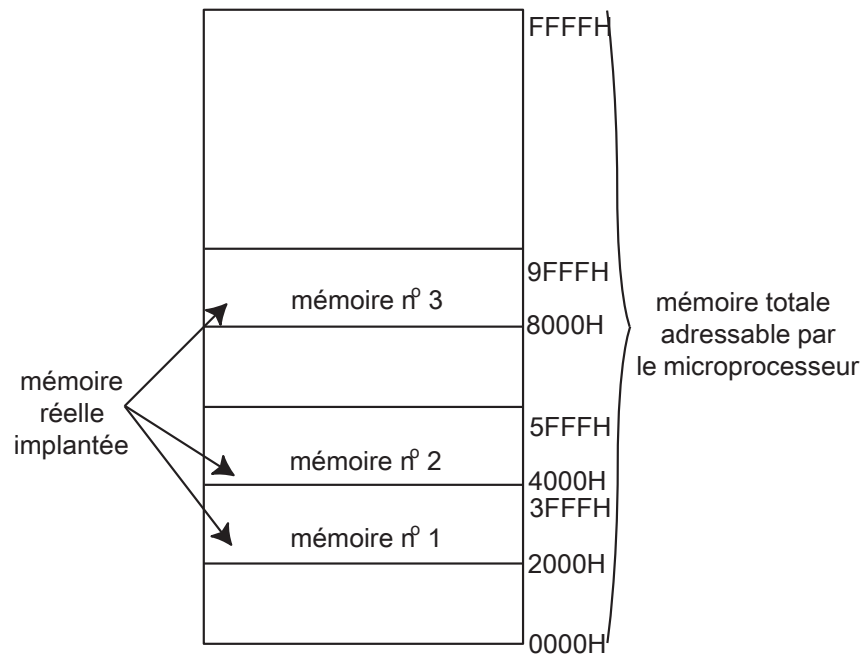
De même, pour la mémoire n°2, on doit avoir A13=0, A14=1 et A15=0 d'où la plage d'adresses occupée cette mémoire:



Pour la mémoire n°3, on doit avoir A13=0, A14=0 et A15=1 d'où la plage d'adresses occupée cette mémoire:



On en déduit la cartographie ou mapping de la mémoire visible par le microprocesseur:



3.6 Décodage d'adresses

Les trois bits A13, A14 et A15 utilisés précédemment fournissent en fait 8 combinaisons, de 000 à 111, d'où la possibilité de connecter jusqu'à 8 bits mémoire de 8 Ko sur le bus. La mémoire totale implantée devient donc de $8 \times 8 \text{Ko} = 64 \text{Ko}$: valeur maximale possible avec 16 bits d'adresses.

Pour cela, il faut utiliser un circuit de décodage d'adresses, dans ce cas: un décodeur 3 vers 8.

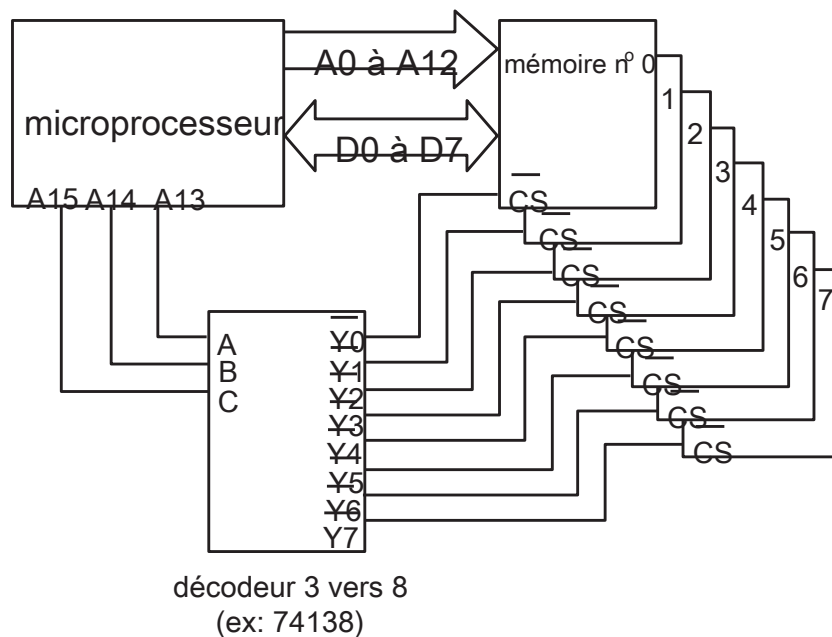


Table de vérité de l'encodeur d'adresses:

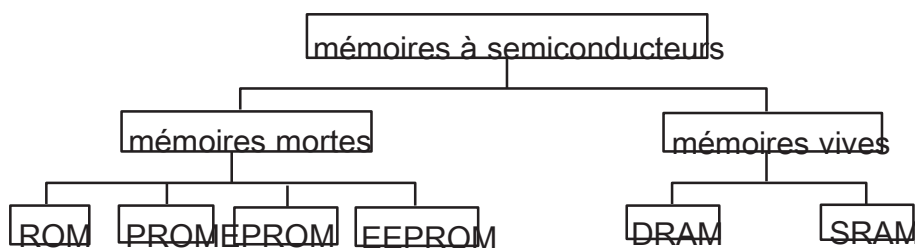
CBAY0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0000	1	1	1	1	1	1	1
0011	0	1	1	1	1	1	1
0101	1	0	1	1	1	1	1
0111	1	1	0	1	1	1	1
1001	1	1	1	0	1	1	1
1011	1	1	1	1	0	1	1
1101	1	1	1	1	1	0	1
1111	1	1	1	1	1	1	0

Le mapping de la mémoire devient ainsi:

mémoire n° 7	FFFFH E000H DFFFH
mémoire n° 6	C000H BFFFH
mémoire n° 5	A000H 9FFFH
mémoire n° 4	8000H 7FFFH
mémoire n° 3	6000H 5FFFH
mémoire n° 2	4000H 3FFFH
mémoire n° 1	2000H 1FFFH
mémoire n° 0	0000H

3.7 Classification des mémoires

Jusqu'à la fin des années 1970, on utilisait des mémoires à aires magnétiques, lentes et de faibles capacités. Actuellement, on n'utilise plus que des mémoires à semi-conducteurs.



Mémoires mortes:

- **ROM**: Read Only Memory. Mémoire à lecture seule, sans écriture. Son contenu est programmé une fois pour toutes par le constructeur. Avantage: faible coût. Inconvénient: nécessite une production très grande quantité.
- **PROM**: Programmable Read Only Memory. ROM programmable une seule fois par l'utilisateur (ROM OTP: One Time Programming) en faisant sauter des fusibles. Nécessite un programmeur spécialisé: application d'une tension de programmation (21 ou 25V) pendant 20ms.
- **EPROM**: Erasable PROM, appelé aussi UV PROM. ROM programmable électriquement avec un programmeur et effacable par exposition à un rayonnement ultraviolet pendant 30 minutes. Famille 27xxx, exemple: 2764 (8Ko), 27256 (32Ko). Avantage: reprogrammable par l'utilisateur.
- **EEPROM**: Electrically Erasable PROM. ROM programmable et effacable électriquement. Lecture à vitesse normale (≤ 100 ns). Écriture (= effacement) très lente (≈ 10 ms). Application: les EEPROM contiennent des données qui peuvent être modifiées de temps en temps, exemple: paramètres de configuration des ordinateurs. Avantage: programmations sans extraction de la carte et sans programmeur. Inconvénient: coût élevé.

Mémoires vives:

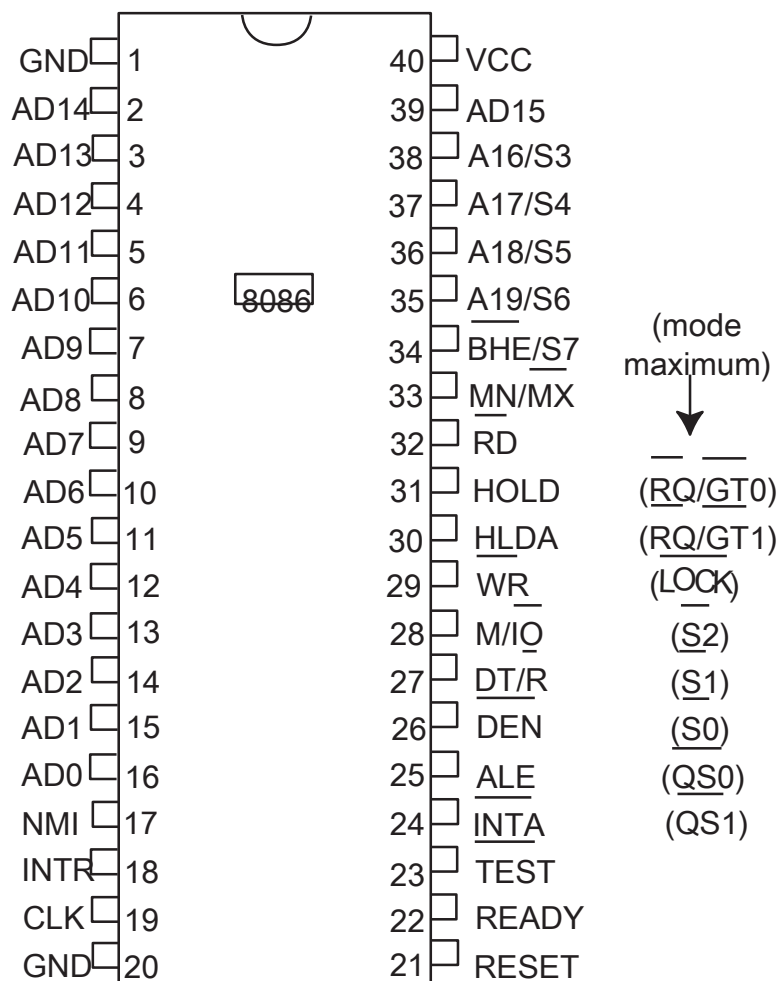
- **SRAM**: Static Random Access Memory. Mémoire statique à accès aléatoire, à base de bascules à semi-conducteurs à deux états (bascules RS). Famille 62xxx, exemple: 62128 (16Ko). Avantage: très rapide, simple utilisation. Inconvénient: compliqué à réaliser.
- **DRAM**: Dynamic RAM. Basées sur la charge de condensateurs: condensateur chargé = 1, condensateur déchargé = 0. Avantage: intégration élevée, faible coût. Inconvénient: nécessite un rafraîchissement périodique à cause du courant de fuite des condensateurs. Application: réalisation de la mémoire vive des ordinateurs (barettes mémoire SIMM: Single In-line Memory module).

Chapitre 4

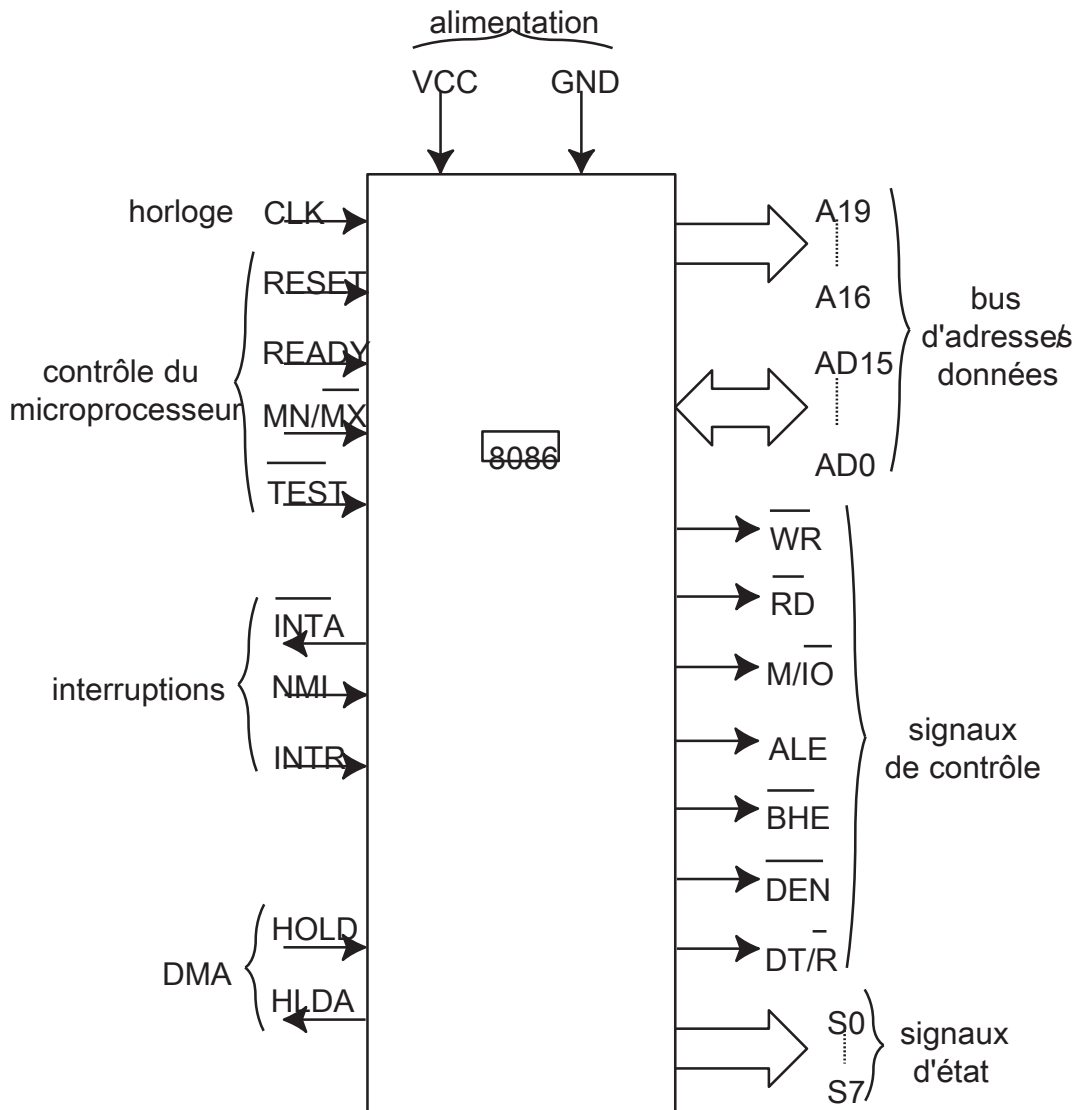
Le microprocesseur Intel 8086

4.1 Description physique du 8086

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

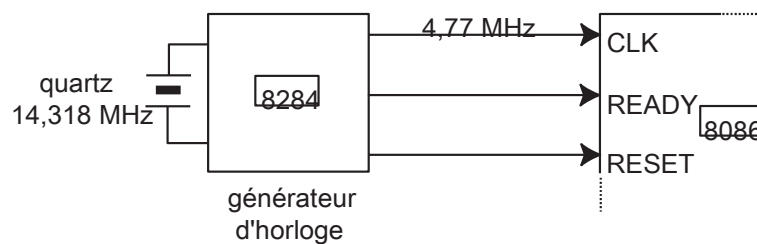


4.2 Schéma fonctionnel du 8086



4.3 Description et utilisation des signaux du 8086

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



RESET: entrée de remise à zéro du microprocesseur. Lorsque cette entrée est mise à l'état haut pendant au moins 4 périodes d'horloge, le microprocesseur est réinitialisé: il va exécuter l'instruction se trouvant à l'adresse FFFF0H (adresse de bootstrap). Le signal de RESET est fourni par le générateur d'horloge.

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

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

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

- mode minimum ($\overline{MN}/\overline{MX}=1$): le 8086 fonctionne de manière autonome, il génère lui-même le bus de commande (RD, WR, ...);
- mode maximum ($\overline{MN}/\overline{MX}=0$): les signaux de commande sont produits par un contrôleur de bus, le 8288. Ce mode permet de réaliser des systèmes multiprocesseurs.

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

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

HO/D et HDA: signaux de demande d'accord d'accès direct à la mémoire (DMA).

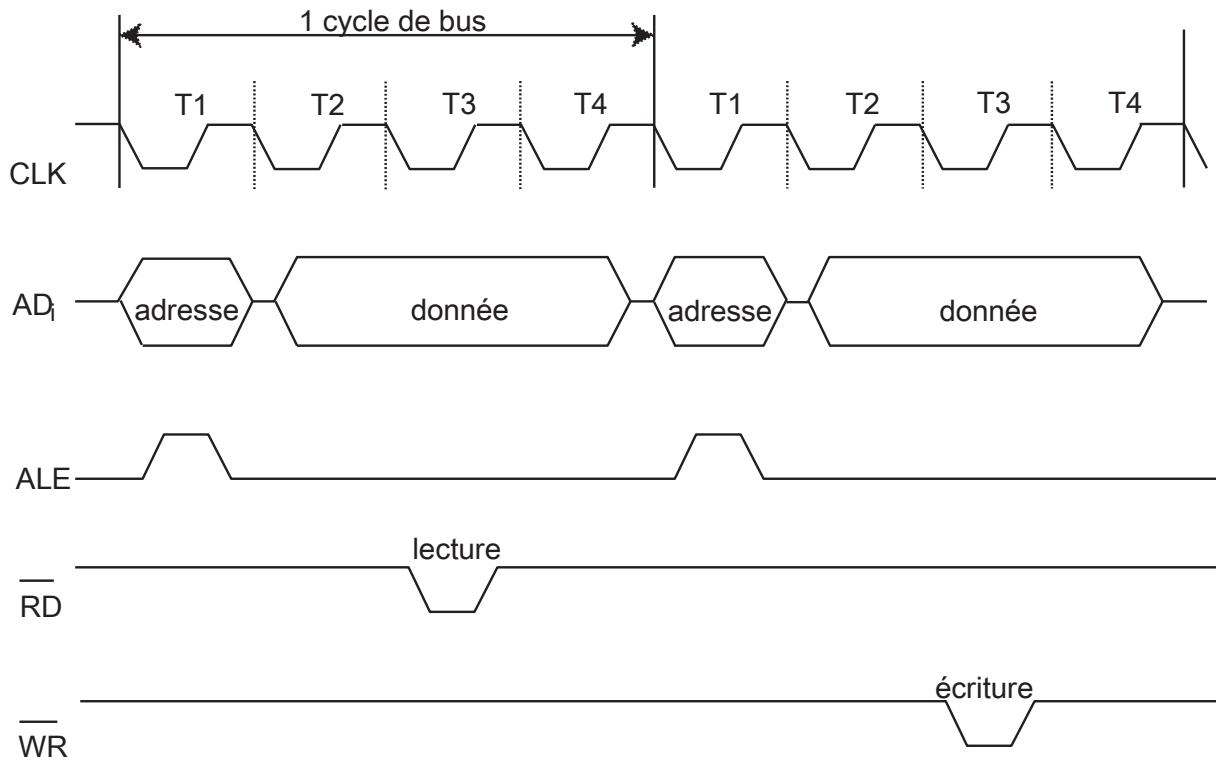
S0 à S7: signaux d'état indiquant le type d'opération en cours sur le bus.

A16/S3 à A19/S6: 4 bits de poids fort du bus d'adresses, multiplexés avec 4 bits d'état.

A0 à A15: 16 bits de poids faible du bus d'adresses, multiplexés avec 16 bits de données. Le bus A/D est multiplexé (multiplexage temporel) d'où l'absence d'un démultiplexage pour obtenir séparément les bus d'adresses et de données:

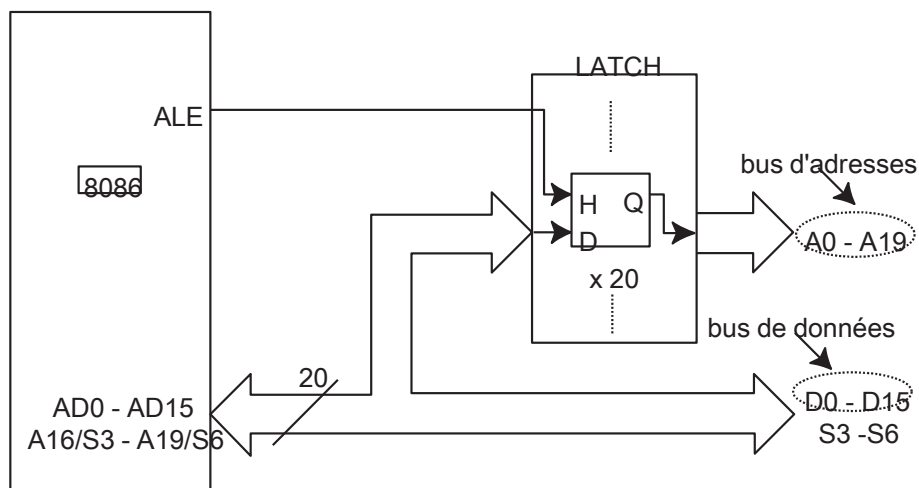
- 16 bits de données (microprocesseur 16 bits);
- 20 bits d'adresses, d'où $2^{20} = 1 \text{ Mo}$ d'espace mémoire adressable par le 8086.

Chronogramme du bus A/D:



L'émultiplexage des signaux AD_0 à AD_{15} (ou A_{16}/S_3 à A_{19}/S_6) se fait en mémorisant l'adresse lors que celle-ci est présente sur le bus A/D, à l'aide d'un verrou (latch), ensemble de bascules D. La commande de mémorisation de l'adresse est générée par le microprocesseur : c'est le signal ALE , Address Latch Enable.

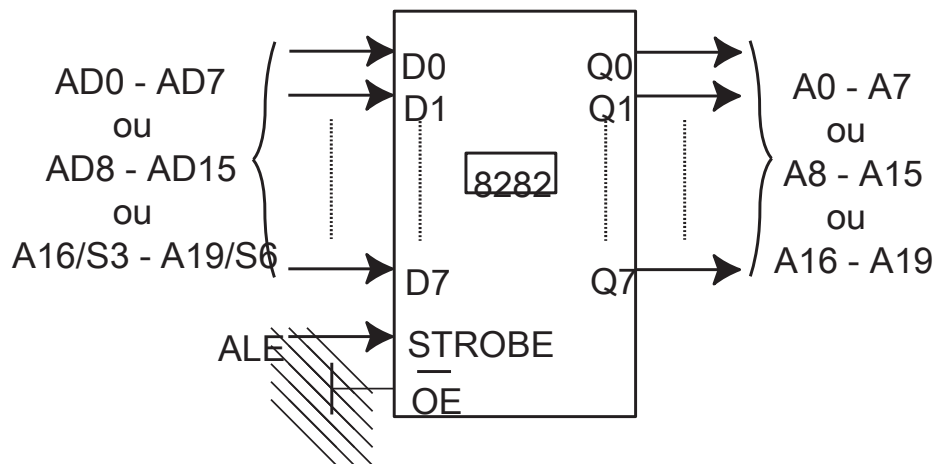
Circuit de l'émultiplexage A/D :



Fonctionnement :

- si $ALE=1$, le verrou est transparent ($Q=D$);
- si $ALE=0$, mémorisation de la dernière valeur de D sur les sorties Q;
- les signaux de lecture (\overline{RD}) ou d'écriture (\overline{WR}) ne sont générés par le microprocesseur que lorsque les données sont présentes sur le bus A/D.

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



\overline{RD} : Read, signal de lecture d'une donnée.

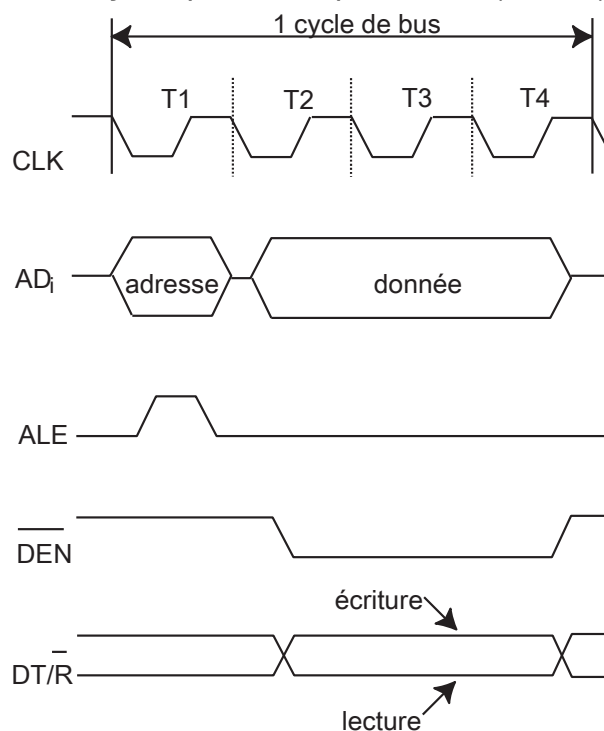
\overline{WR} : Write, signal d'écriture d'une donnée.

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

\overline{DEN} : Data Enable, indique que des données sont entrain de circuler sur le bus A/D (équivalent de ALE pour les données).

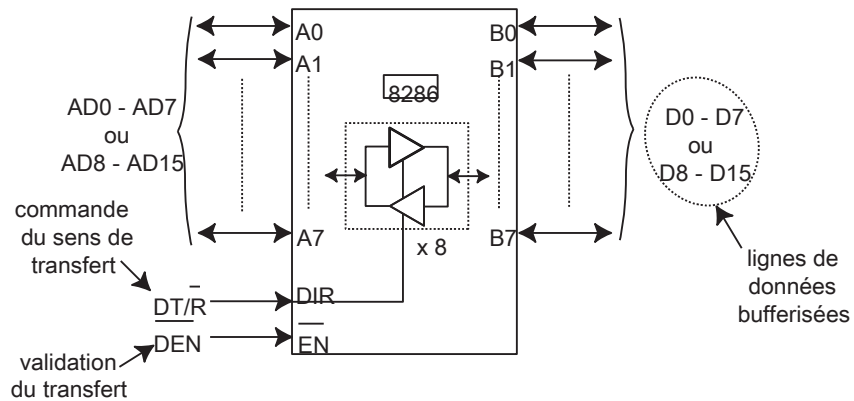
$\overline{DT}/\overline{R}$: Data Transmit/Receive, indique le sens de transfert des données:

- $\overline{DT}/\overline{R}=1$: données émises par le microprocesseur (écriture);
- $\overline{DT}/\overline{R}=0$: données reçues par le microprocesseur (lecture).



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

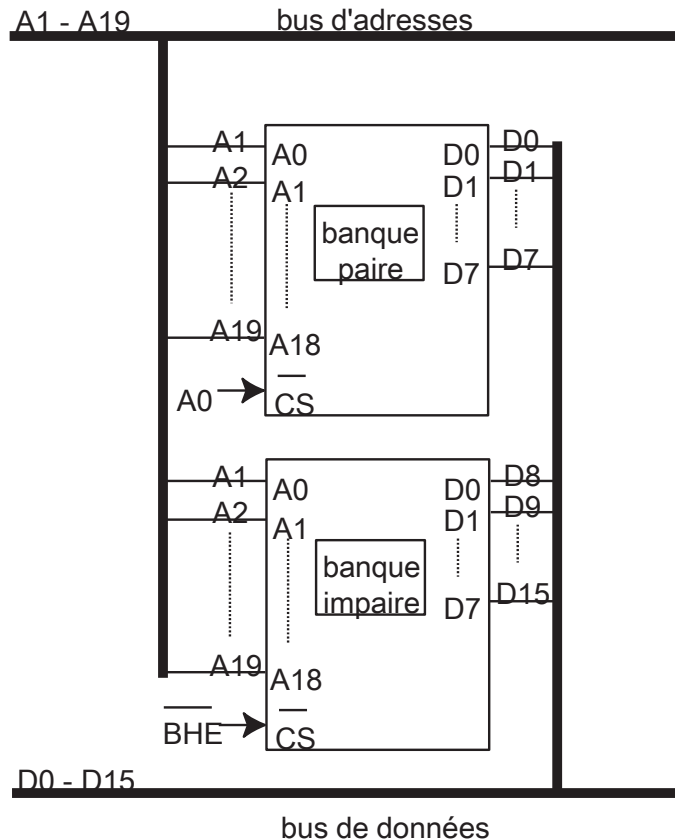
Exemples de tampons de bus : circuits transmetteurs bidirectionnels 8286 ou 74245.



BHE: BusHighEnable, signal de lecture de l'octet de poids fort du bus de données. Le 8086 possède un bus d'adresse sur 20 bits, d'où une capacité d'adressage de 1 Mo ou 512 K mots de 16 bits (bus de données sur 16 bits).

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

- A0 pour la banque paire qui contient les octets de poids faible ;
- BHE pour la banque impaire qui contient les octets de poids fort.

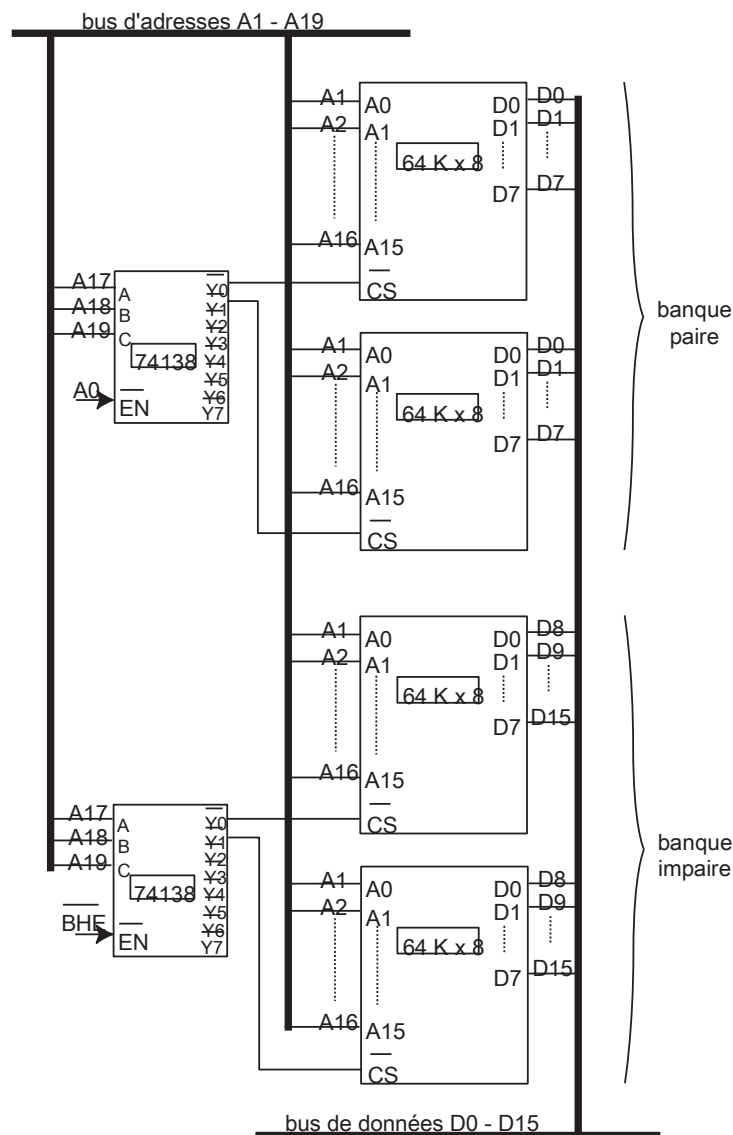


Seuls les bits A1 à A19 servent à désigner une case mémoire dans chaque banque de 512 Ko. Le microprocesseur peut ainsi lire et écrire des données sur 8 bits ou sur 16 bits :

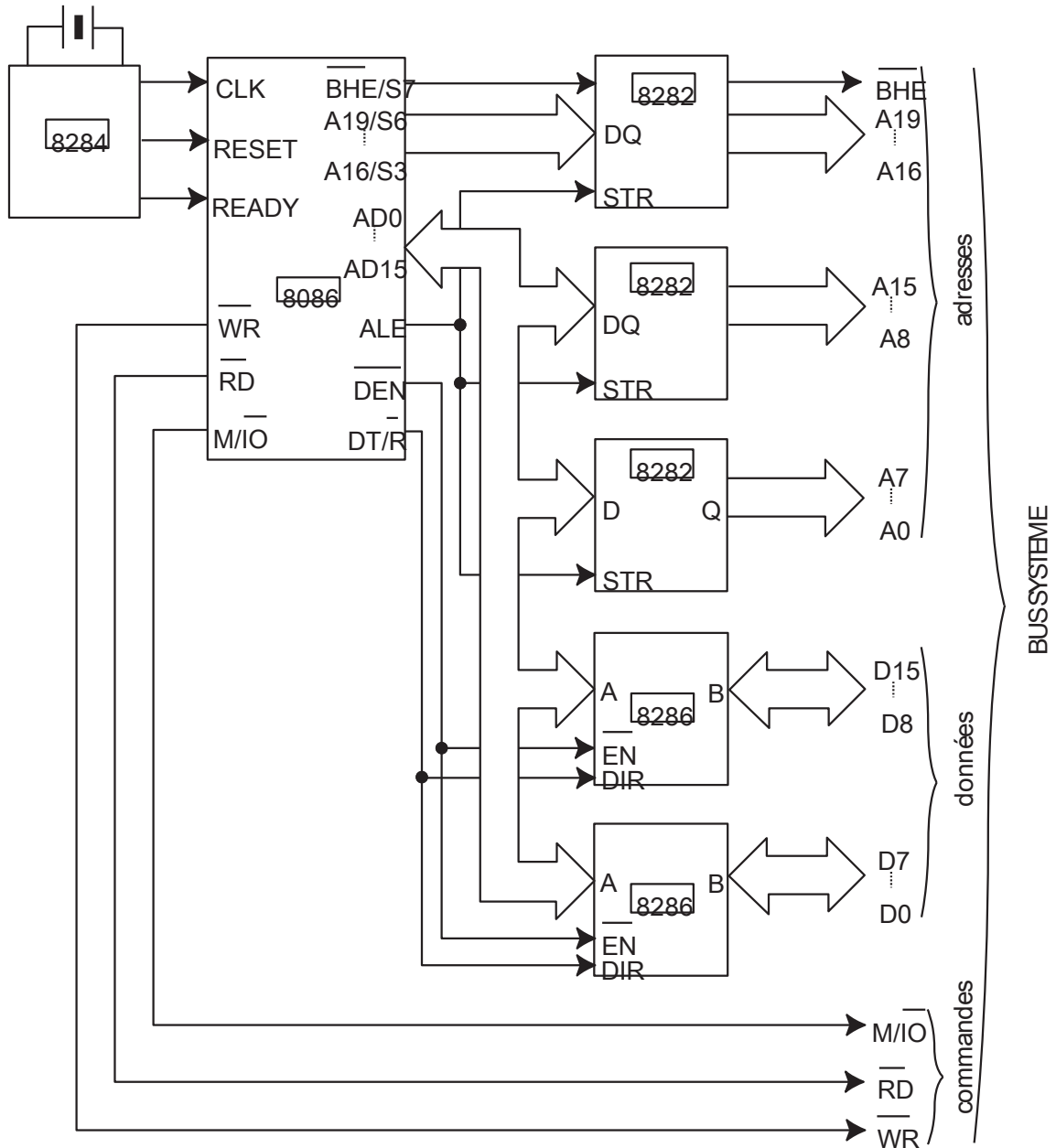
BHE	A0	octet transférés
00		les deux octets (mot complet)
01		octet fort (adresse impaire)
10		octet faible (adresse paire)
11		aucun octet

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

Réalisation des deux banques avec plusieurs boîtiers mémoire :



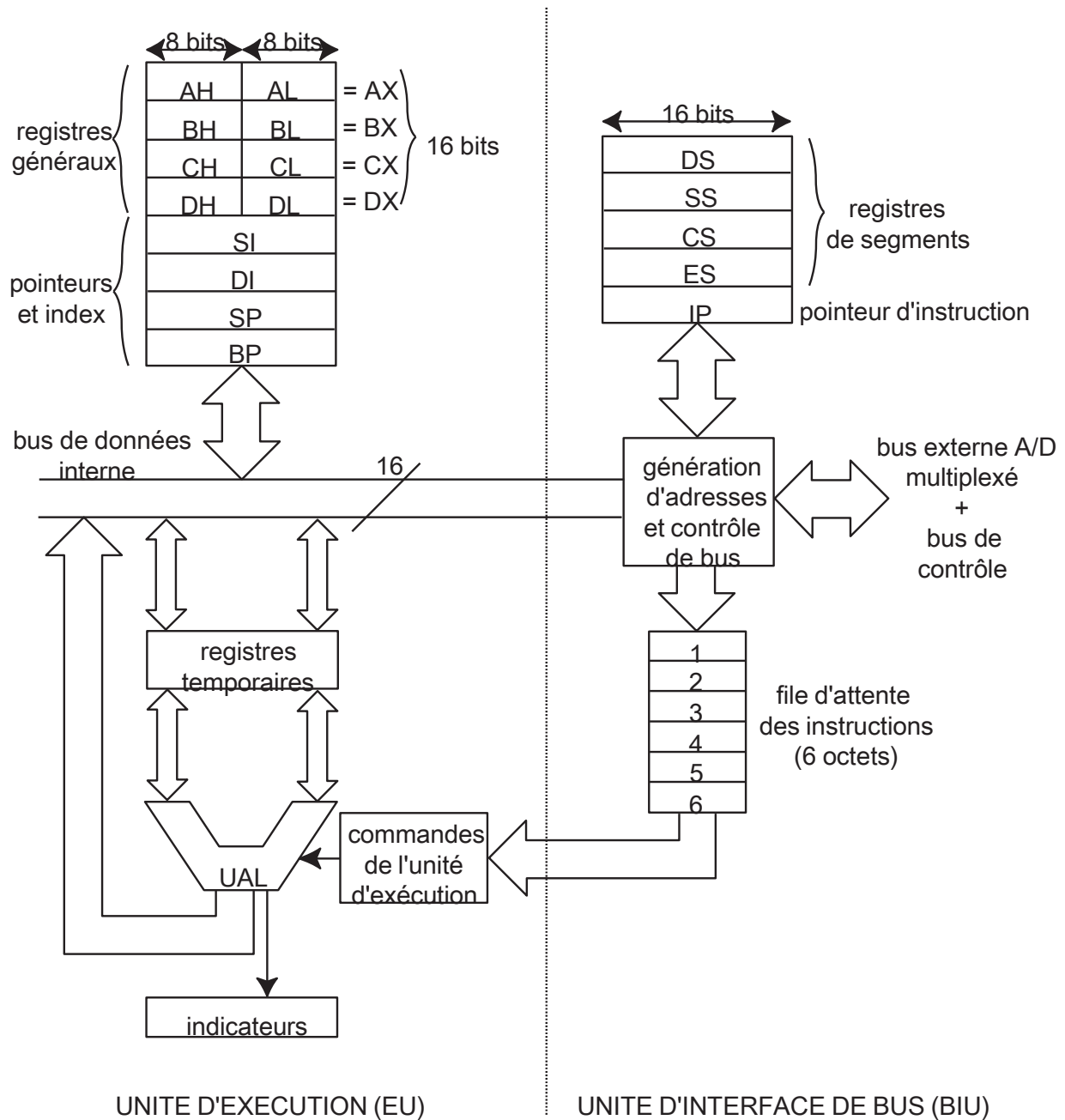
Création du bus système de l'8086:



4.4 Organisation interne de l'8086

L'8086 est constitué de deux unités fonctionnant en parallèle :

- l'unité d'exécution (EU: Execution Unit);
- l'unité d'interface de bus (BIU: Bus Interface Unit).



Rôle des deux unités :

- l'unité d'interface de bus (BIU) recherche les instructions en mémoire et les range dans une file d'attente;
- l'unité d'exécution (EU) exécute les instructions contenues dans la file d'attente.

Les deux unités fonctionnent simultanément, d'où une accélération du processus d'exécution d'un programme (fonctionnement selon le principe du pipeline).

Le microprocesseur 8086 contient 14 registres répartis en 4 groupes :

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

$AX = (AH, AL);$

$BX = (BH, BL);$

$CX = (CH, CL);$

$DX = (DH, DL).$

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

- Registres de pointeur et d'index: 4 registres sur 16 bits.

Pointeurs:

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

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

Index:

SI : Source Index;

DI : Destination Index.

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

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

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

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

Flags:

				OF	DF		IF		ZF		AF		CF		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CF : indicateur de retenue (carry);

PF : indicateur de parité;

AF : indicateur de retenue auxiliaire;

ZF : indicateur de zéro;

SF : indicateur de signe;

TF : indicateur d'exécution pas à pas (trap);

IF : indicateur d'autorisation d'interruption;

DF : indicateur de débordement;

OF : indicateur de débordement (overflow).

- Registres de segments: 4 registres sur 16 bits.

CS : Code Segment, registre de segment de code;

DS: Data Segment, registre de segment de données;

SS: Stack Segment, registre de segment de pile;

ES: Extra Segment, registre de segments supplémentaires pour les données;

Les registres de segments, associés aux pointeurs et aux index, permettent au microprocesseur 8086 d'adresser l'ensemble de la mémoire.

4.5 Gestion de la mémoire par le 8086

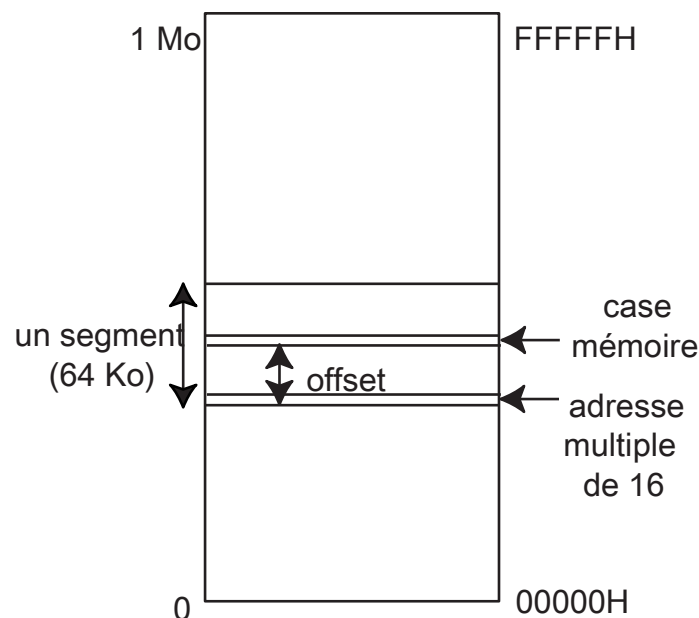
L'espace mémoire adressable par le 8086 est de $2^{20} = 1048576$ octets = 1 Mo (20 bits d'adresses). Cet espace est divisé en segments. Un segment est une zone mémoire de 64 Ko (65536 octets) définie par son adresse de départ qui doit être un multiple de 16. Dans une telle adresse, les 4 bits de poids faible sont à zéro. On peut donc représenter l'adresse d'un segment avec seulement ses 16 bits de poids fort, les 4 bits de poids faible étant implicitement à 0.

Pour désigner une case mémoire parmi les $2^{16} = 65536$ contenues dans un segment, il suffit d'une valeur sur 16 bits.

Ainsi, une case mémoire est représentée par le 8086 au moyen de deux quantités sur 16 bits:

- l'adresse d'un segment;
- un déplacement ou *offset* (appelé aussi *adresse effective*) dans ce segment.

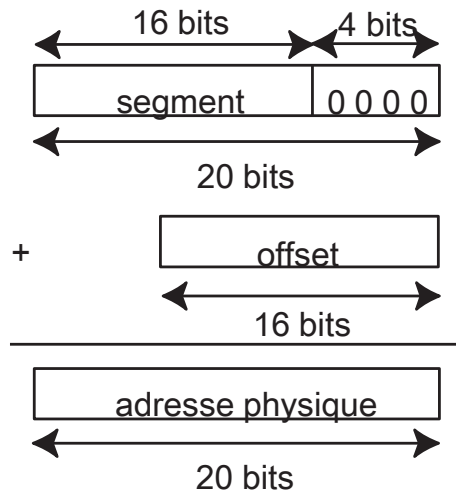
Cette méthode de gestion de la mémoire est appelée *segmentation de la mémoire*.



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

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

Correspondance entre adresse logique et adresse physique:



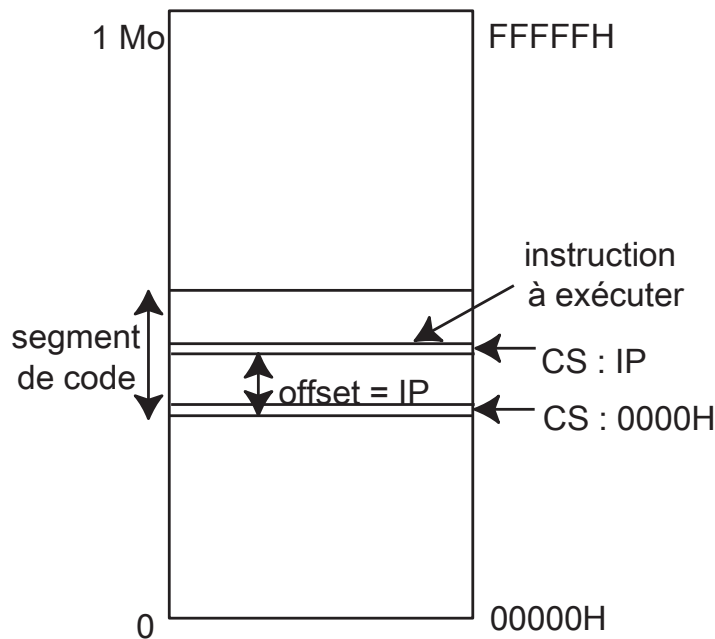
Ainsi, l'adresse physique se calcule par l'expression:

$$\text{adresse physique} = 16 \times \text{segment} + \text{offset}$$

car le fait d'injecter 4 zéros en poids faibles du segment revient à affecter un décalage de 4 positions vers la gauche, c'est à dire une multiplication par $2^4 = 16$.

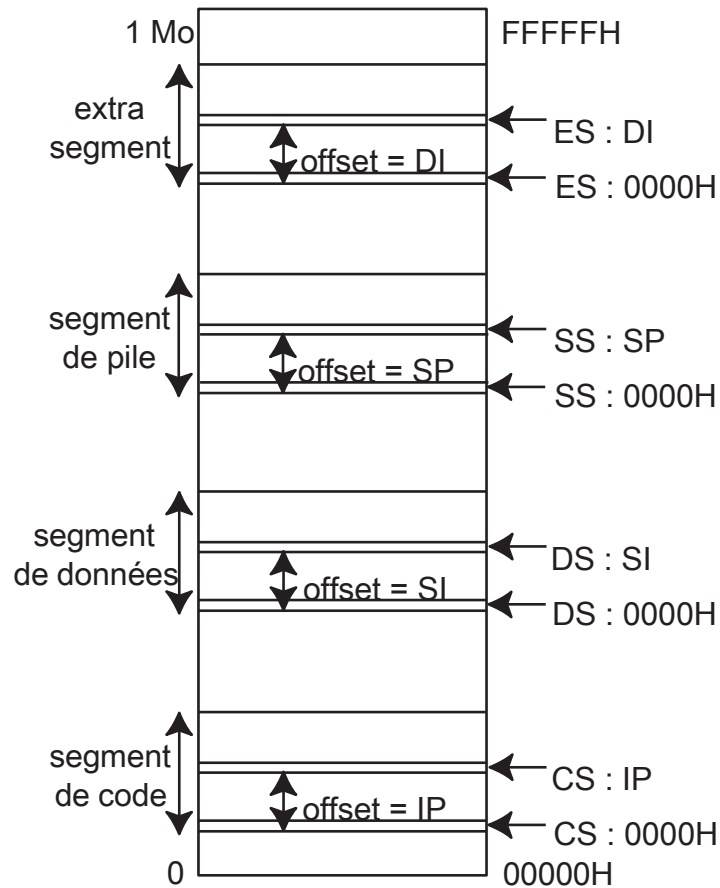
A un instant donné, le 8086 accède à 4 segments dont les adresses se trouvent dans les registres de segment CS, DS, SS et ES. Le segment de code contient les instructions du programme, le segment de données contient les données manipulées par le programme, le segment de pile contient la pile de sauvegarde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

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

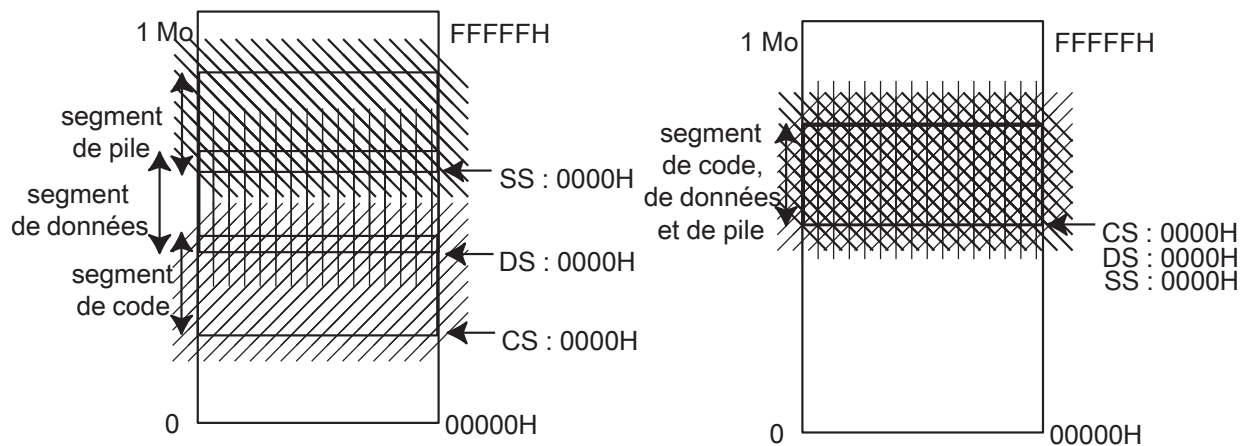


De même, les registres de segments DS et ES peuvent être associés à un registre d'index. Exemple: DS:SI, ES:DI. Le registre de segment de pile peut être associé aux registres de pointeurs: SS:SP ou SS:BP.

Mémoire accessible par le 8086 à un instant donné:



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



Le nombre de segments utilisés est défini dans le mode d'emploi de la mémoire du programme.

Contenu des registres après un RESET du microprocesseur:

IP=0000H
 CS=FFFFH
 DS=0000H
 ES=0000H
 SS=0000H

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

4.6 Le microprocesseur 8088

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

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

Différences avec le 8086:

- les broches $\overline{AD}8$ à $\overline{AD}15$ deviennent $\overline{A}8$ à $\overline{A}15$ (bus de données sur 8 bits);
- la broche \overline{BHE} n'existe pas dans le 8088 car il n'y a pas d'octet de poids fort sur le bus de données;
- la broche $\overline{M}/\overline{IO}$ devient $\overline{IO}/\overline{M}$ pour la compatibilité avec d'anciens circuits d'E/S.

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

Chapitre 5

La programmation en assembleur du microprocesseur 8086

5.1 Généralités

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

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

Les instructions peuvent être classées en groupes:

- instructions de transfert de données;
- instructions arithmétiques;
- instructions logiques;
- instructions de branchement...

5.2 Les instructions de transfert

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

- registre vers mémoire;
- registre vers registre;
- mémoire vers registre.

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

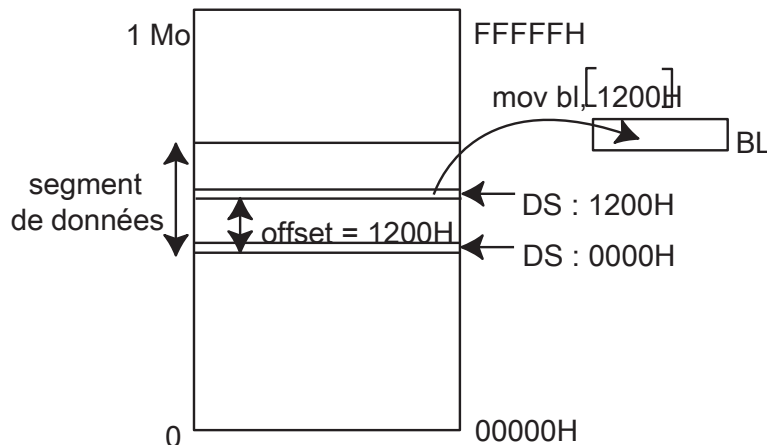
Syntaxe: MOV destination, source

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

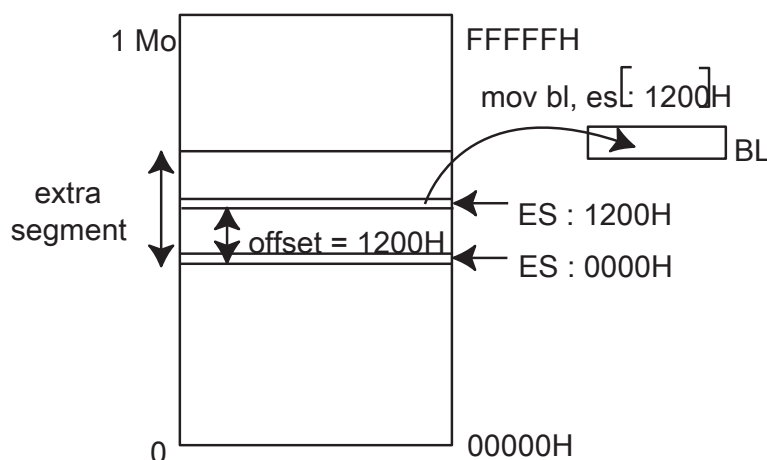
Il existe différentes facons de spécifier l'adresse d'une case mémoire dans une instruction: ce sont les modes d'adressage.

Exemples de modes d'adressage simples:

- `mov ax, bx` : charge le contenu du registre BX dans le registre AX. Dans ce cas, le transfert se fait de registre à registre: adressage par registre;
- `mov al, 12H` : charge le registre AL avec la valeur 12H. Ladonnée est fournie immédiatement avec l'instruction: adressage immédiat.
- `mov bl, [1200H]` : transfère le contenu de la case mémoire d'adresse effective (offset) 1200H vers le registre BL. L'instruction comporte l'adresse de la case mémoire où se trouve ladonnée: adressage direct. L'adresse effective représente l'offset de la case mémoire dans le segment donné (segment dont l'adresse est contenue dans le registre DS): segment par défaut.

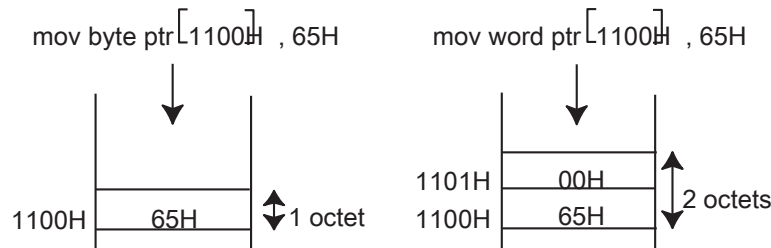


On peut changer le segment lors d'un adressage direct en ajoutant un préfixe fixe de segment, exemple: `mov bl, es:[1200H]`. On parle alors de forçage de segment.



Remarque: dans le cas de l'adressage immédiat de la mémoire, il faut indiquer le format de la donnée: octet ou mot (2 octets) car le microprocesseur 8086 peut manipuler des données sur 8 bits ou 16 bits. Pour cela, on doit utiliser un *spécificateur de format*:

- `mov byte ptr [1100H], 65H` : transfère la valeur 65H (sur 1 octet) dans la case mémoire d'offset 1100H;
- `mov word ptr [1100H], 65H` : transfère la valeur 0065H (sur 2 octets) dans les cases mémoire d'offset 1100H et 1101H.



Remarque: les microprocesseurs Intel rangent l'octet de poids faible d'une donnée sur plusieurs octets à l'adresse la plus basse (format *Little Endian*).

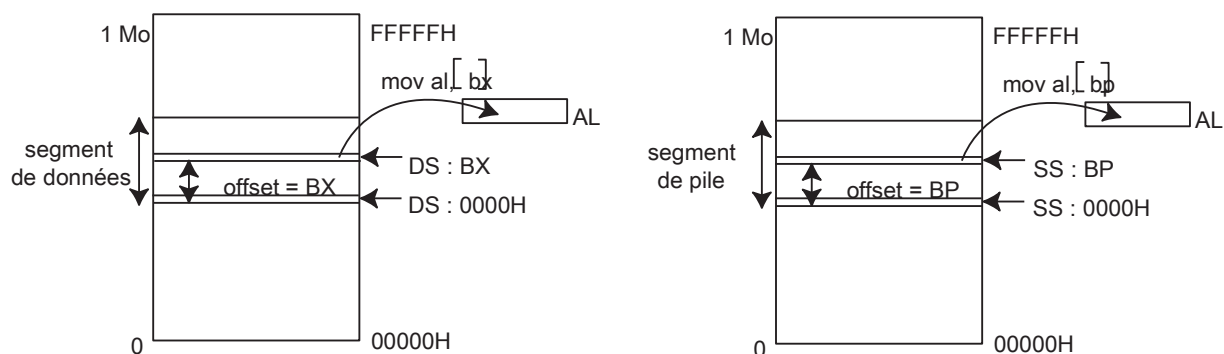
Modes d'adressage évolués:

- **adressage basé:** l'offset est contenu dans un registre de base BX ou BP.

Exemples:

`mov al, [bx]` : transfère la donnée dont l'offset est contenu dans le registre de base BX vers le registre AL. Le segment associé par défaut est le segment de données: on dit que l'adressage est basé sur DS;

`mov al, [bp]` : le segment par défaut est associé au registre de base BP et le segment de pile. Dans ce cas, l'adressage est basé sur SS.



- **adressage indexé:** semblable à l'adressage basé, sauf que l'offset est contenu dans un registre d'index SI ou DI, associé par défaut au segment de données.

Exemples:

`mov al, [si]` : charge le registre AL avec le contenu de la case mémoire dont l'offset est contenu dans SI;

```
mov[di],bx :charge les cases m'emoire d'offset DI et DI+1 avec le contenu
du registre BX.
```

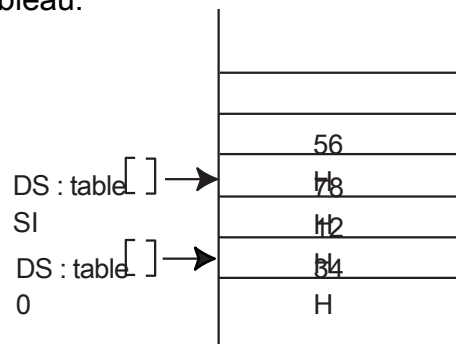
Remarque: une valeur constante peut éventuellement être ajoutée aux registres de base ou d'index pour obtenir l'offset. Exemple:

```
mov[si+100H],ax
qui peut aussi s'écrire
mov[si][100H],ax
ou encore
mov100H[si],ax
```

Les modes d'adressage basés sur l'index permettent la manipulation de tableaux rangés en mémoire. Exemple:

```
movsi,0
movwordptrtable[si],1234H
movsi,2
movwordptrtable[si],5678H
```

Dans cet exemple, table représente l'offset du premier élément du tableau et le registre SI joue le rôle d'index de tableau:



- adressage basé sur l'index : l'offset est obtenu en faisant la somme d'un registre de base, d'un registre d'index et d'une valeur constante. Exemple:

```
movah,[bx+si+100H]
```

Ce mode d'adressage permet l'adressage de structures de données complexes: matrices, enregistrements, ... Exemple:

```
movbx,10
movsi,15
movbyteptrmatrice[bx][si],12H
```

Dans cet exemple, BX et SI jouent respectivement le rôle d'indices de ligne et de colonne dans la matrice.

5.3 Les instructions arithmétiques

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

Addition: ADD opérande1, opérande2

L'opération effectuée est: $\text{opérande1} \leftarrow \text{opérande1} + \text{opérande2}$.

Exemples:

- addah,[1100H] :ajoute le contenu de la case mémoire d'offset 1100H à l'accumulateur AH (adressage direct);
- addah,[bx] :ajoute le contenu de la case mémoire pointée par BX à l'accumulateur AH (adressage basé);
- addbyteptr[1200H],05H :ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H (adressage immédiat).

Soustraction: SUB opérande1, opérande2

L'opération effectuée est: $\text{opérande1} \leftarrow \text{opérande1} - \text{opérande2}$.

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

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

Exemples:

- mov al,51
mov bl,32
mul bl
→ AX=51 × 32
- mov ax,4253
mov bx,1689
mul bx
→ (DX, AX)=4253 × 1689
- mov al,43
mov byteptr[1200H],28
mul byteptr[1200H]
→ AX=43 × 28
- mov ax,1234
mov wordptr[1200H],5678
mul wordptr[1200H]
→ (DX, AX)=1234 × 5678

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

Cette instruction effectue la division du contenu de AX par un opérande sur 1 octet ou le contenu de (DX, AX) par un opérande sur 2 octets. Résultat: si l'opérande est sur 1 octet,

alors AL=quotient et AH=reste; si l'opérande est sur 2 octets, alors AX=quotient et DX=reste.

Exemples:

- mov ax, 35
mov bl, 10
div bl
→ AL=3(quotient) et AH=5(reste)
- mov dx, 0
mov ax, 1234
mov bx, 10
div bx
→ AX=123(quotient) et DX=4(reste)

Autres instructions arithmétiques:

- ADC: addition avec retenue;
- SBB: soustraction avec retenue;
- INC: incrément d'une unité;
- DEC: décrément d'une unité;
- IMUL: multiplication signée;
- IDIV: division signée.

5.4 Les instructions logiques

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

- ET;
- OU;
- OU exclusif;
- complément à 1;
- complément à 2;
- décalages et rotations.

Les différents modes d'adressage sont disponibles.

ET logique: AND opérande1, opérande2

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

Exemple:

mov al, 10010110B		AL=10010110
mov bl, 11001101B	→	BL=11001101
and al, bl		AL=10000100

Application: masquage de bits pour mettre à zéro certains bits dans un mot.

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

```

76543210
01010111
00111100 ← masque
00010100

```

Oplogique: OR opérande1, opérande2

L'opération effectuée est: opérande1 \leftarrow opérande1 OU opérande2.

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

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

```

76543210
10110      0 0 1
00001010 ← masque
10111      0 1 1

```

Les instructions correspondantes peuvent s'écrire:

```
movah,10110001B
```

```
orah,00001010B
```

Complément à 1: NOT opérande

L'opération effectuée est: opérande \leftarrow opérande.

Exemple:

```

movah,10010001B → AL = 01101110B
notah

```

Complément à 2: NEG opérande

L'opération effectuée est: opérande \leftarrow opérande + 1.

Exemple:

```

movah,25
movbl,12 → AL=25+( -12)=13
negbl
addah,bl

```

Opérateur exclusif: XOR opérande1, opérande2

L'opération effectuée est: opérande1 \leftarrow opérande1 \oplus opérande2.

Exemple: mise à zéro d'un registre:

```

movah,25 → AL=0
xorah,al

```

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

Dans les décalages, les bits qui sont déplacés sont remplacés par des zéros. Il y a les décalages logiques (opérations non signées) et les décalages arithmétiques (opérations signées).

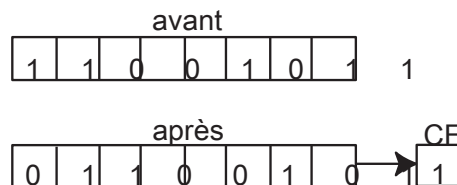
Dans les rotations, les bits d'entrée placés dans un sens sont injectés de l'autre côté du mot.

Décalage logique vers la droite (Shift Right): SHR Opérande, n

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

Exemple:

```
mov al, 11001011B
shrl, 1
```



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

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

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

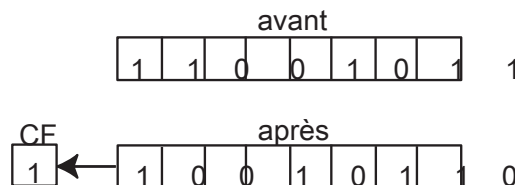
```
mov cl, 3
shrl, cl
```

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

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

Exemple:

```
mov al, 11001011B
shll, 1
```



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

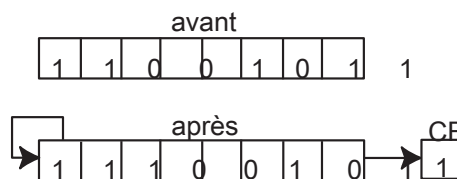
Même remarque que précédemment si le nombre de positions à décaler est supérieur à 1.

Décalage arithmétique vers la droite: SAR Opérande, n

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

Exemple:

```
mov al, 11001011B
sarl, 1
```



→ le bit de signe est injecté.

Décalage arithmétique vers la gauche: SAR Opérande, n
Identique au décalage logique vers la gauche.

Applications des instructions de décalage:

- cadrage d'un groupe de bits.
Exemple: on veut avoir la valeur du quart de poids fort du registre AL:

```
mov al, 11001011B
mov cl, 4          →    AL = 00001100B
shra l, cl
```

- tester l'état d'un bit dans un mot.
Exemple: on veut déterminer l'état du bit 5 de AL:

```
mov cl, 6          ou          mov cl, 3
shra l, cl         ou          shla l, cl
```

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

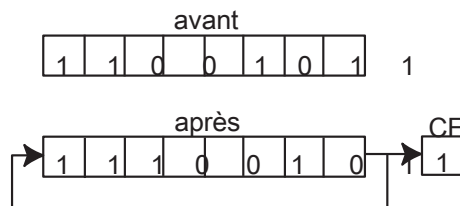
- multiplication ou division par une puissance de 2: un décalage à droite revient à faire une division par 2 et un décalage à gauche, une multiplication par 2.
Exemple:

```
mov al, 48
mov cl, 3          →    AL = 48 / 23 = 6
shra l, cl
```

Rotation à droite (Rotate Right): ROR Opérande, n
Cette instruction décale l'opérande de n positions vers la droite et réinjecte par la gauche les bits sortants.

Exemple:

```
mov al, 11001011B
ror al, 1
```

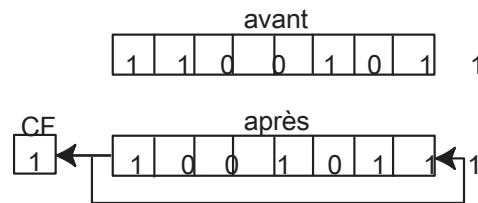


→ réinjection des bits sortants qui est copiée dans l'indicateur de retenue CF.

Rotation à gauche (Rotate Left): ROL Opérande, n
Cette instruction décale l'opérande de n positions vers la gauche et réinjecte par la droite les bits sortants.

Exemple:

```
mov al, 11001011B
rol al, 1
```

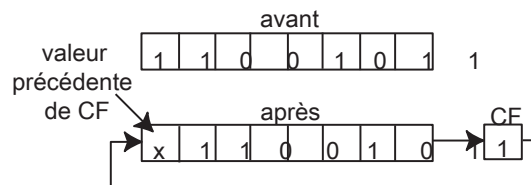


→ réinjection des bits sortant qui est copiée dans l'indicateur de retenue CF.
 Notation : à droite avec passage par l'indicateur de retenue (Rotate Right through Carry) : RCR opérande, n

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

Exemple :

```
mov al, 11001011B
rcr al, 1
```



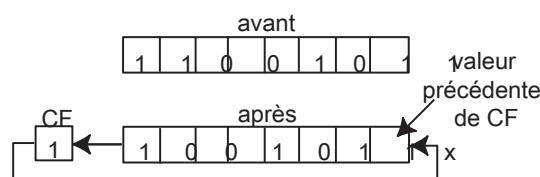
→ les bits sortant par la droite est copiée dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la gauche.

Notation : à gauche avec passage par l'indicateur de retenue (Rotate Left through Carry) : RCL opérande, n

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

Exemple :

```
mov al, 11001011B
rcl al, 1
```



→ les bits sortant par la gauche est copiée dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la droite.

5.5 Les instructions de branchement

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

- saut inconditionnel ;
- sauts conditionnels ;
- appel des sous-programmes.

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

```

      : ← instructions précédant lesaut
      jmp suite
      : ← instructions suivant lesaut (jamais exécutées)
suite: ... ← instruction exécutée après lesaut
  
```

Exemple:

```

boucle: inc ax
      dec bx      →      boucle infinie
      jmp boucle
  
```

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

Instructions desauts conditionnels: `Jcondition label`
 Un saut conditionnel n'est exécuté que si une certaine condition est satisfaite, sinon l'exécution se poursuit séquentiellement à l'instruction suivante.

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

instruction	nom	condition
<code>JZ label</code>	JumpifZero	saut si ZF=1
<code>JNZ label</code>	JumpifNotZero	saut si ZF=0
<code>JE label</code>	JumpifEqual	saut si ZF=1
<code>JNE label</code>	JumpifNotEqual	saut si ZF=0
<code>JC label</code>	JumpifCarry	saut si CF=1
<code>JNC label</code>	JumpifNotCarry	saut si CF=0
<code>JS label</code>	JumpifSign	saut si SF=1
<code>JNS label</code>	JumpifNotSign	saut si SF=0
<code>JO label</code>	JumpifOverflow	saut si OF=1
<code>JNO label</code>	JumpifNotOverflow	saut si OF=0
<code>JP label</code>	JumpifParity	saut si PF=1
<code>JNP label</code>	JumpifNotParity	saut si PF=0

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

Exemple:

```

      : ← instructions précédant lesaut conditionnel
      jnz suite
      : ← instruction exécutée si la condition ZF=0 est vérifiée
suite: ... ← instruction exécutée à la suite desaut
  
```

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

condition	nombres signés	nombres non signés
=	JEQlabel	JEQlabel
>	JGlabel	JAlabel
<	JLlabel	JBlabel
=	JNElabel	JNElabel

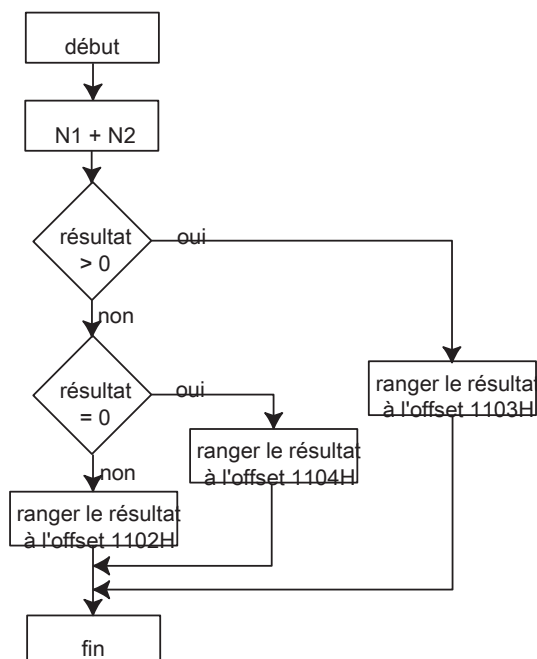
Exemple:

```

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

Exemple d'application des instructions des sauts conditionnels: on veut additionner deux nombres signés N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H. Le résultat est rangé à l'offset 1102H s'il est positif, à l'offset 1103H s'il est négatif et à l'offset 1104H s'il est nul:

Organigramme:



Programme:

```

mov al, [1100H]
add al, [1101H]
js  negatif
jz  nul
mov [1102H], al
jmp fin
negatif:
mov [1103H], al
jmp fin
nul:
mov [1104H], al
fin:
hlt
    
```

Appel des sous-programmes: pour éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme, on crée une séquence unique et on lui attribue un nom (au choix) et on l'appelle lorsqu'on en a besoin. Le programme ap-

pendant le programme principal. La séquence appelée est un sous-programme ou procédure.

Écriture d'un sous-programme:

```

nomsp PROC
      : ← instructions du sous-programme
      ret ← instruction de retour au programme principal
nomsp ENDP
  
```

Remarque: une procédure peut être de type NEAR si elle se trouve dans le même segment ou de type FAR si elle se trouve dans un autre segment.

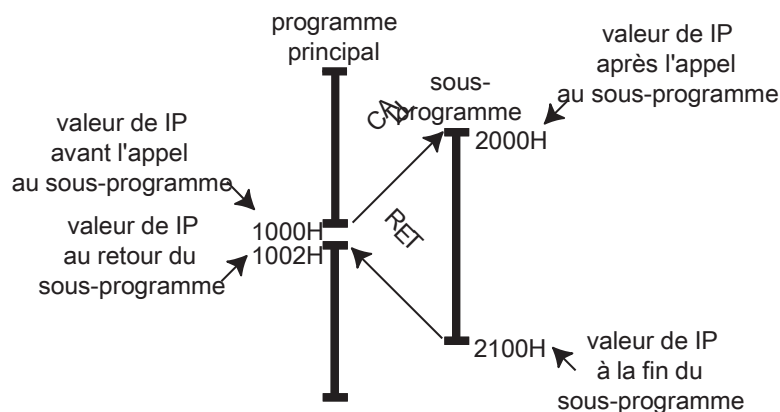
Exemple: ss-prog1 PROC NEAR
 ss-prog2 PROC FAR

Appel d'un sous-programme par le programme principal: CALL procédure

```

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

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



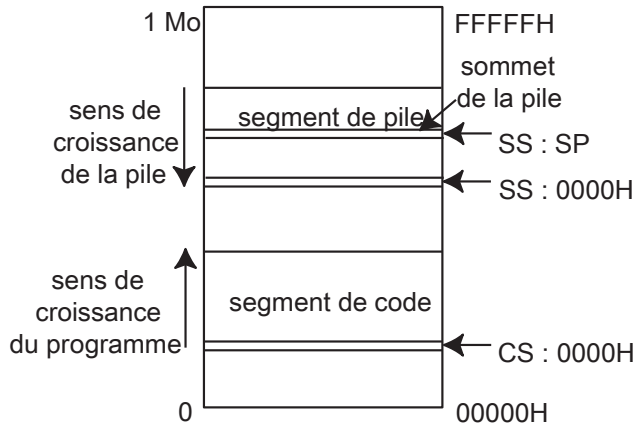
Avant de charger l'IP avec l'adresse du sous-programme, l'adresse de retour au programme principal, c'est-à-dire le contenu de l'IP, est sauvegardé dans une zone mémoire particulière appelée épipile. Lors de l'exécution de l'instruction RET, cette adresse est récupérée à partir de la pile et rechargée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

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

- empiler un donné: placer le donné au sommet de la pile;
- dépiler un donné: lire le donné se trouvant au sommet de la pile.

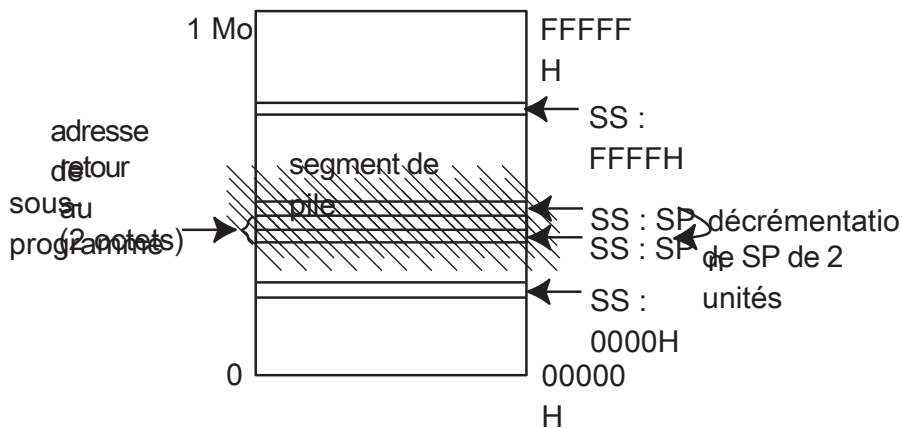
Le sommet de la pile est représenté par un registre appelé pointeur de pile (SP: Stack Pointer) qui contient l'adresse de la dernière donnée empilée.

La pile est définie dans le segment de pile dont l'adresse de départ est contenue dans le registre SS.



Remarque: la pile et le programme croissent en sens inverse pour diminuer le risque de collision entre le code et la pile dans le cas où celle-ci est placée dans le même segment que le code (SS=CS).

Lors de l'appel à un sous-programme, l'adresse de retour au programme appelant (contenu de IP) est empilée et le pointeur de pile SP est automatiquement décrémenté. À la fin du sous-programme, le pointeur d'instruction IP est rechargé avec la valeur contenue au sommet de la pile et SP est incremented.



La pile peut également servir à sauvegarder le contenu de registres qui ne sont pas automatiquement sauvegardés lors de l'appel à un sous-programme:

- instruction d'empilage: PUSH opérande
- instruction de dépileage: POP opérande

où l'opérande est un registre ou une donnée sur 2 octets (on ne peut empiler que des mots de 16 bits).

Exemple:

```

pushax      ;empilage du registre AX...
pushbx      ;... du registre BX...
push[1100H] ;... et de la case m'emoire 1100H-1101H
:
pop [1100H] ;d'epilage dans l'ordre inverse de l'empilage
pop bx
pop ax

```

Remarque: la valeur de SP doit être initialisée par le programme principal avant de pouvoir utiliser la pile.

Utilisation de la pile pour le passage de paramètres: pour transmettre des paramètres à une procédure, on peut les placer sur la pile avant l'appel de la procédure, puis celle-ci les récupère en effectuant un adressage basé de la pile en utilisant le registre BP.

Exemple: soit une procédure effectuant la somme de deux nombres et retournant le résultat dans le registre AX:

- programme principal:

```

mov ax,200
pushax      ;empilage du premier paramètre
mov ax,300
pushax      ;empilage du deuxième paramètre
callsomme   ;appel de la procédure

```
- procédure somme

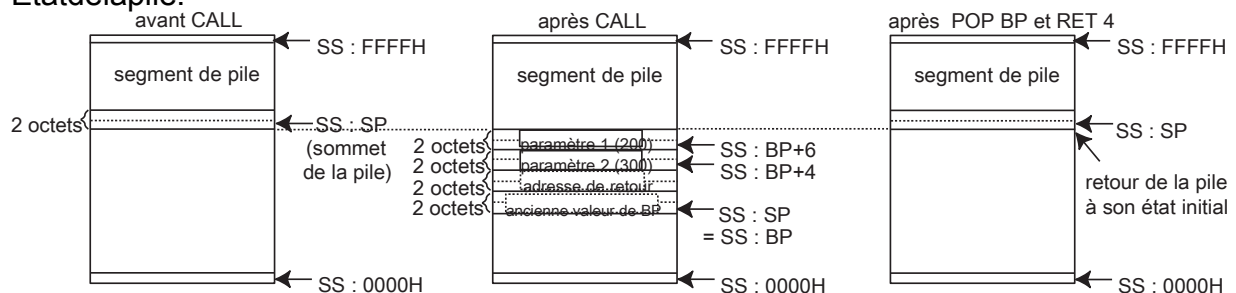
```

somme proc
pushbp      ;sauvegarde de BP
mov bp,sp   ;faire pointer BP sur le sommet de la pile
mov ax,[bp+4];récupération du deuxième paramètre
add ax,[bp+6];addition au premier paramètre
pop bp      ;restauration de l'ancienne valeur de BP
ret 4       ;retour de 4 octets de la pile
somme endp

```

L'instruction `ret 4` permet de retourner au programme principal et d'incrémenter le pointeur de pile de 4 unités pour dépiler les paramètres et se faire remettre la pile dans son état initial.

Etat de la pile:



5.6 Méthodes de programmation

Étapes de la réalisation d'un programme :

- Définir le problème à résoudre : que faut-il faire exactement ?
- Déterminer des algorithmes, des organigrammes : comment faire ? Par quoi commencer, puis poursuivre ?
- Rédiger le programme (code source) :
 - utilisation du jeu d'instructions (mnémoniques) ;
 - création de documents explicatifs (documentation).
- Tester le programme en réel ;
- Corriger les erreurs (bugs) éventuelles : déboguer le programme puis refaire des tests jusqu'à obtention d'un programme fonctionnant de manière satisfaisante.

Langage machine et assembleur :

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

Réalisation pratique d'un programme :

- Rédaction du code source en assembleur à l'aide d'un éditeur (logiciel de traitement de texte ASCII) :
 - edit sous MS-DOS,
 - notepad (bloc-note) sous Windows,
- Assemblage du code source (traduction des instructions en codes binaires) avec un assembleur :
 - MASM de Microsoft,
 - TASM de Borland,
 - A86 disponible en shareware sur Internet, ...
 pour obtenir le code objet : code machine exécutable par le microprocesseur ;
- Chargement en mémoire centrale et exécution : rôle du système d'exploitation (ordinateur) ou d'un moniteur (carte de développement à base de microprocesseur).

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

- l'exécution pas à pas ;
- la visualisation du contenu des registres et de la mémoire ;
- la pose de points d'arrêt et...

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

labels	instructions	commentaires
label1:	movax,60H	;ceciestuncommentaire...
:	:	:
sous-prog1	procnear	;sous-programme
:	:	:
sous-prog1	endp	
:	:	:

Directives pour l'assembleur:

- Origine du programme en mémoire: ORG offset
Exemple: org 1000H
- Définitions de constantes: nom constante EQU valeur
Exemple: escape equ 1BH
- Réserve de cases mémoire:

nom variable DB valeur initiale

nom variable DW valeur initiale

DB: *Define Byte*, r' eservation d'un octet;

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

Exemples:

nombre1 db 25

nombre2 dw ? ; pas de valeur initiale

buffer db 100 dup(?) ; r' eservation d'une zone mémoire de 100 octets

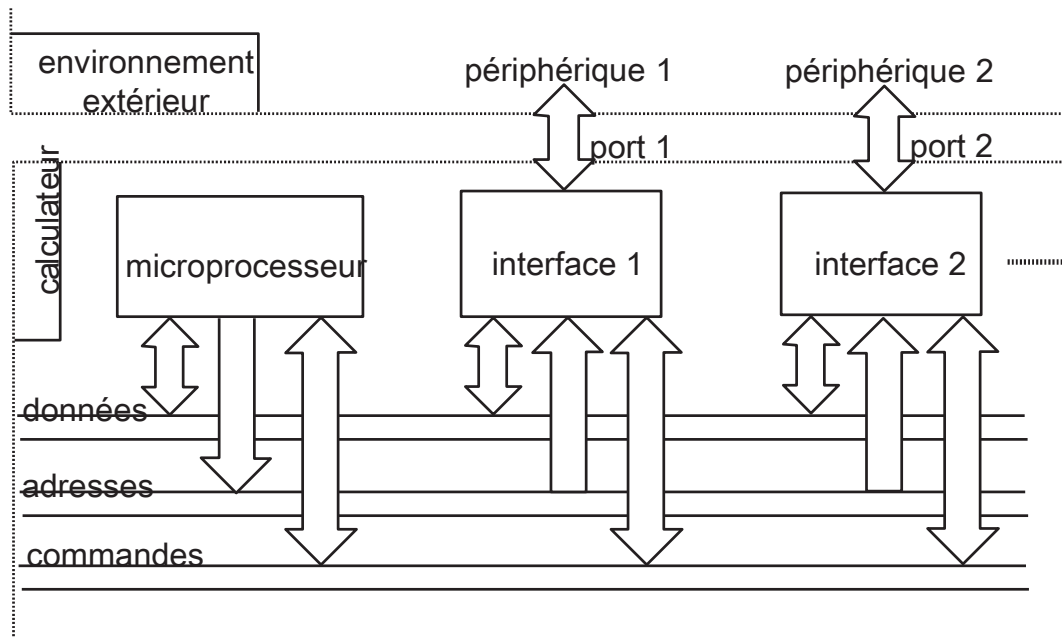
Chapitre 6

Les interfaces d'entrées/sorties

6.1 Définitions

Une interface d'entrées/sorties est un circuit intégré qui permet à un 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 données et de commandes.

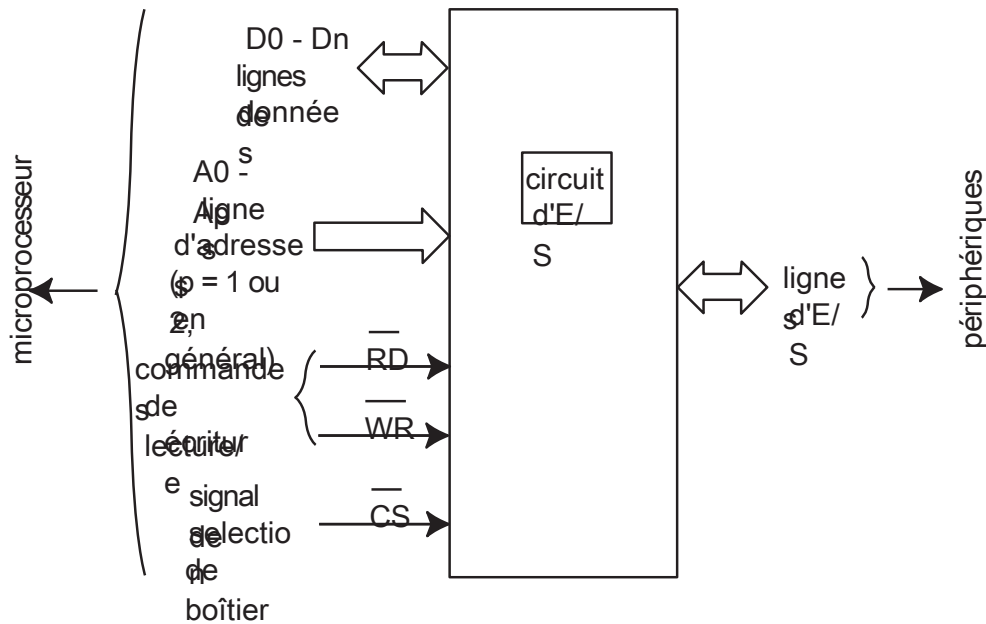


Les points d'accès aux interfaces sont appelés ports.

Exemples:

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

Schémasynoptique d'un circuit d'E/S:



6.2 Adressage des ports d'E/S

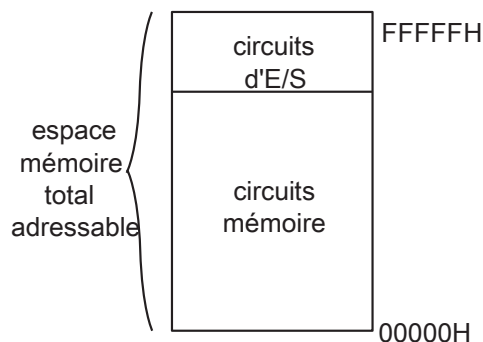
Un circuit d'E/S possède des registres pour gérer les échanges avec les périphériques:

- registres de configuration;
- registres de données.

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

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

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



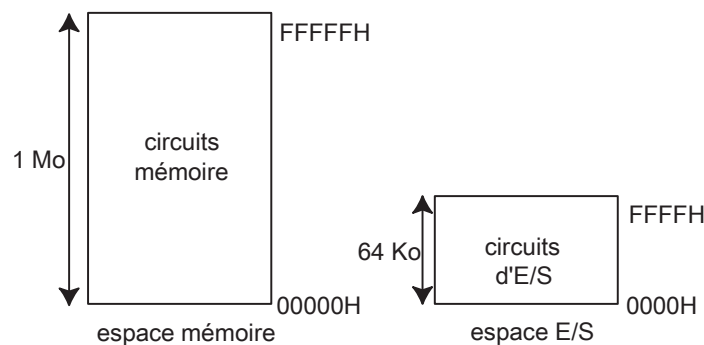
Conséquences:

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

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

- adressage indépendant: le microprocesseur considère deux espaces distincts:
 - l'espace d'adressage des mémoires;
 - l'espace d'adressage des ports d'E/S.

C'est le cas du microprocesseur 8086:



Conséquences:

- contrairement à l'adressage cartographique, l'espace mémoire total adressable n'est pas diminué;
- l'adressage des ports d'E/S peut se faire avec une adresse plus courte (nombre de bits inférieur) que pour les circuits mémoire;
- les instructions utilisées pour l'accès à la mémoire sont plus utilisables pour l'accès aux ports d'E/S: ceux-ci disposent d'instructions spécifiques;
- une même adresse peut désigner soit une case mémoire, soit un port d'E/S: le microprocesseur doit donc fournir un signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S.

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

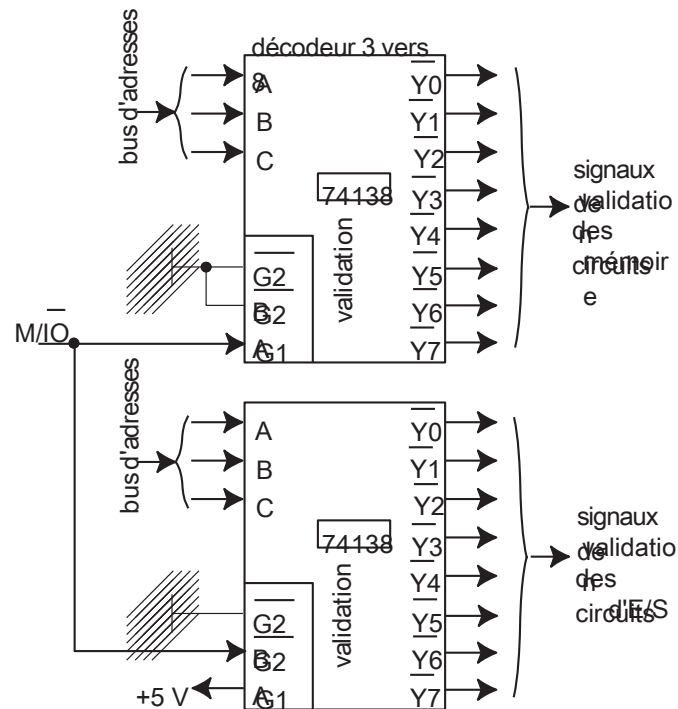
6.3 Gestion des ports d'E/S par le 8086

Le 8086 dispose d'un espace mémoire de 1 Mo (adressé à une case mémoire sur 20 bits) et d'un espace d'E/S de 64 Ko (adressé à 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 $\overline{IO}/\overline{IO}$:

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

Ces signaux sont utilisés pour valider le décodage d'adresse dans les deux espaces :



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

Lecture d'un port d'E/S :

- si l'adresse du port d'E/S est sur un octet :
 $INAL, \text{adresse}$: lecture d'un port sur 8 bits ;
 $INAX, \text{adresse}$: lecture d'un port sur 16 bits.
- si l'adresse du port d'E/S est sur deux octets :
 $INAL, DX$: lecture d'un port sur 8 bits ;
 $INAX, DX$: lecture d'un port sur 16 bits.
 où le registre DX contient l'adresse du port d'E/S à lire.

Écriture d'un port d'E/S :

- si l'adresse du port d'E/S est sur un octet :
 $OUT \text{adresse}, AL$: écriture d'un port sur 8 bits ;
 $OUT \text{adresse}, AX$: écriture d'un port sur 16 bits.
- si l'adresse du port d'E/S est sur deux octets :
 $OUTDX, AL$: écriture d'un port sur 8 bits ;
 $OUTDX, AX$: écriture d'un port sur 16 bits.
 où le registre DX contient l'adresse du port d'E/S à écrire.

Exemples:

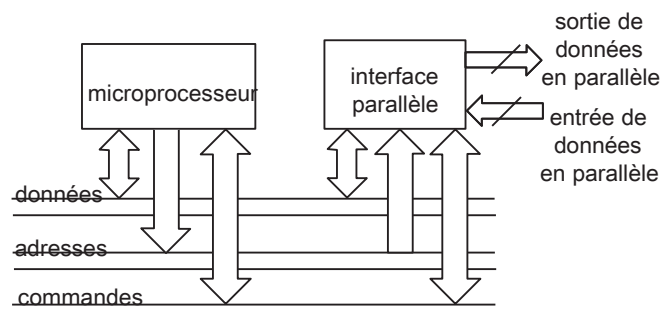
- lecture d'un port d'E/S sur 8 bits à l'adresse 300H:

```
mov dx, 300H
in al, dx
```
- écriture de la valeur 1234H dans le port d'E/S sur 16 bits à l'adresse 49H:

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

6.4 L'interface parallèle de l'8255

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



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

Brochage de l'8255:

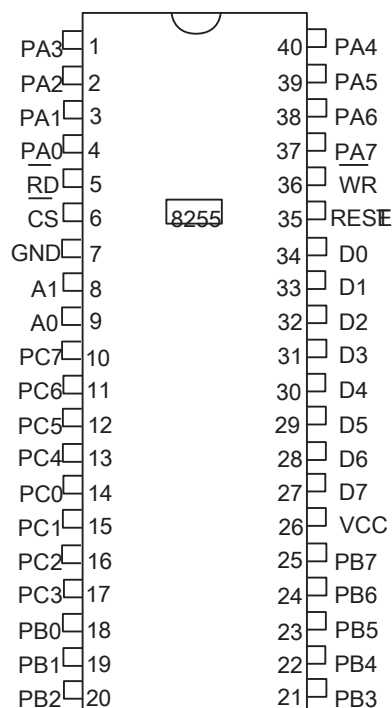
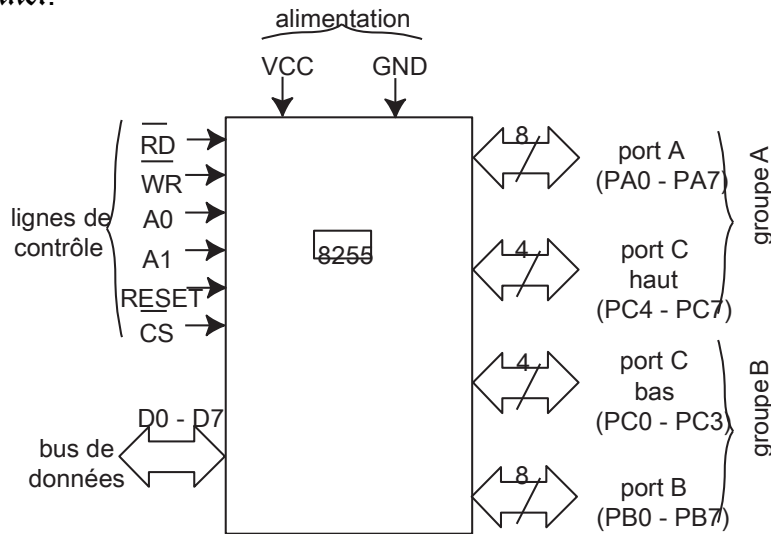


Schéma fonctionnel :



Le 8255 contient 4 registres :

- trois registres contenant les données présentes sur les ports A, B et C ;
- un registre de commande pour la configuration des ports A, B et C en entrées/sorties.

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

A ₁	A ₀	ADWRCS	opération
0	0	0	lecture du port A
0	1	0	lecture du port B
1	0	0	lecture du port C
0	0	1	écriture du port A
0	1	1	écriture du port B
1	0	1	écriture du port C
1	1	1	écriture du registre de commande
X	X	1	pas de transaction
1	1	0	illégal
X	X	1	pas de transaction

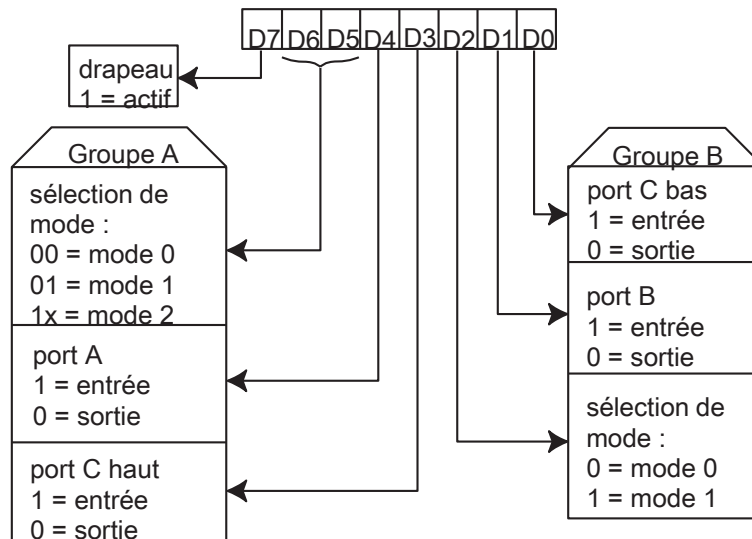
Remarque : le registre de commande est accessible uniquement en écriture, la lecture de ce registre n'est pas autorisée.

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

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

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

Structure du registre de commande:

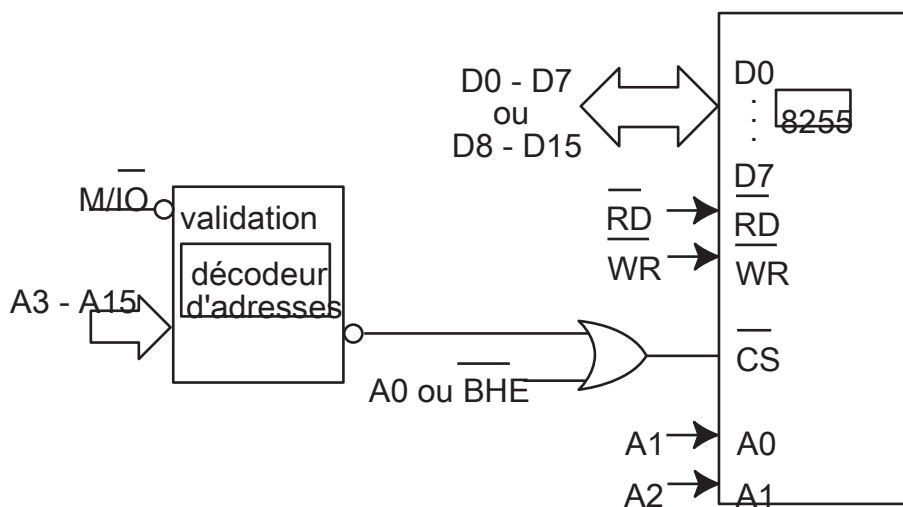


Connexion du 8255 sur le bus du 8086: le bus de données du 8255 est sur 8 bits alors que celui du microprocesseur 8086 est sur 16 bits. On peut donc connecter le bus de données du 8255 sur les lignes de données faibles du 8086 (D0-D7) ou sur celles de données fortes (D8-D15).

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

- sur la partie faible du bus de données lorsqu'une adresse à écrire (ou à lire) est paire: validation par A0;
- sur la partie haute lorsqu'une adresse est impaire: validation par $\overline{\text{BHE}}$.

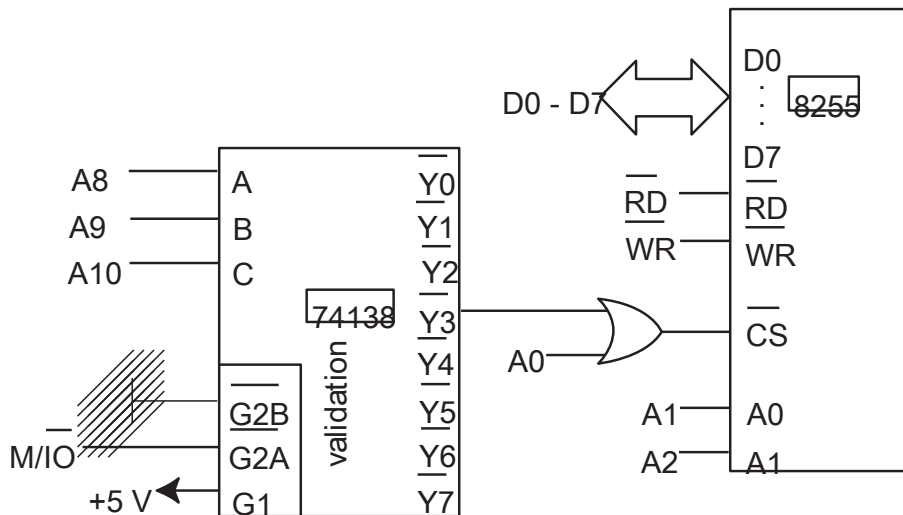
Ainsi, un des deux signaux A0 ou $\overline{\text{BHE}}$ doit être utilisé pour sélectionner le 8255:



Conséquence: les adresses des registres du 8255 se trouvent à des adresses paires (validation par A0) ou impaires (validation par $\overline{\text{BHE}}$).

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

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



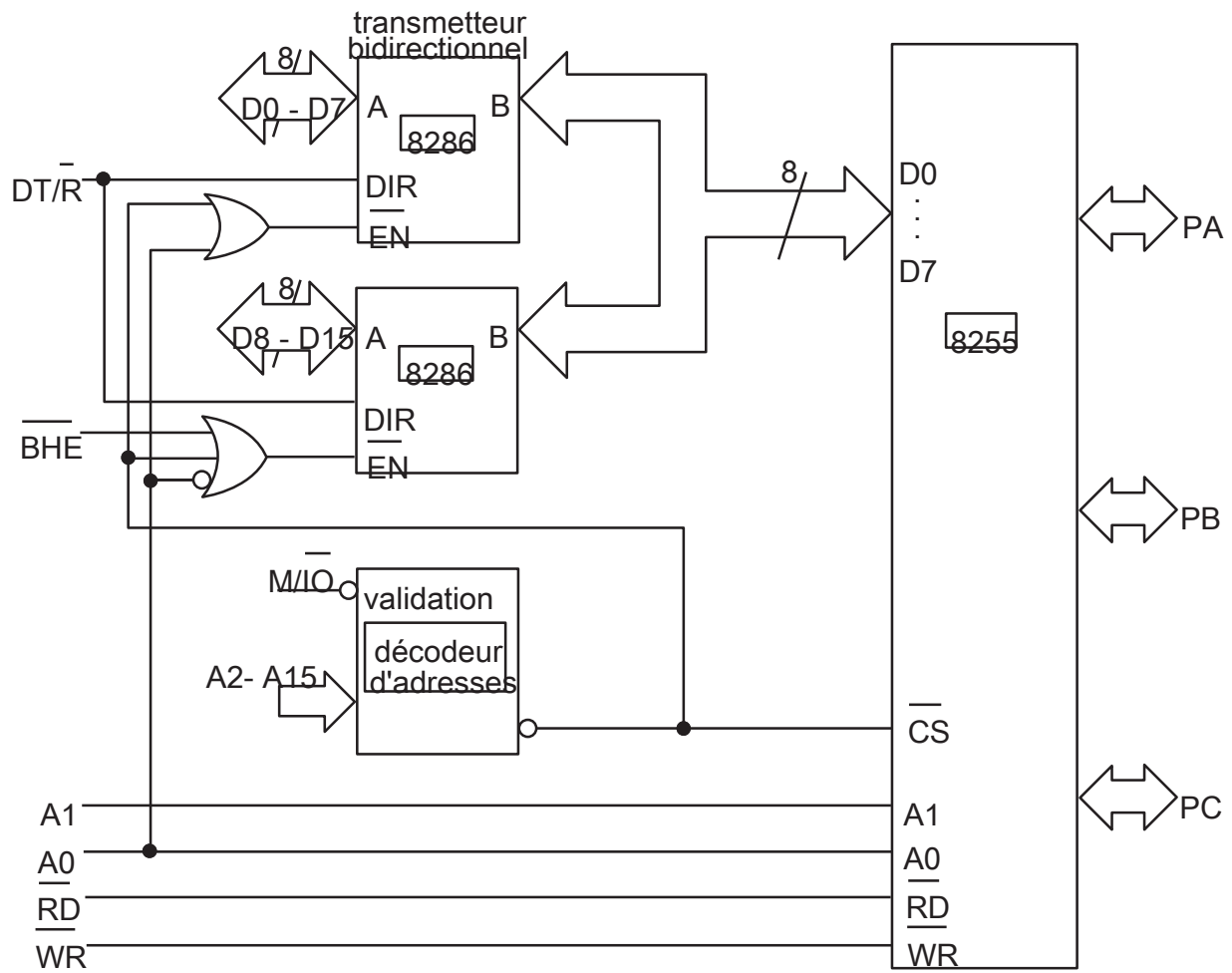
Détermination de l'adresse du 8255:

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

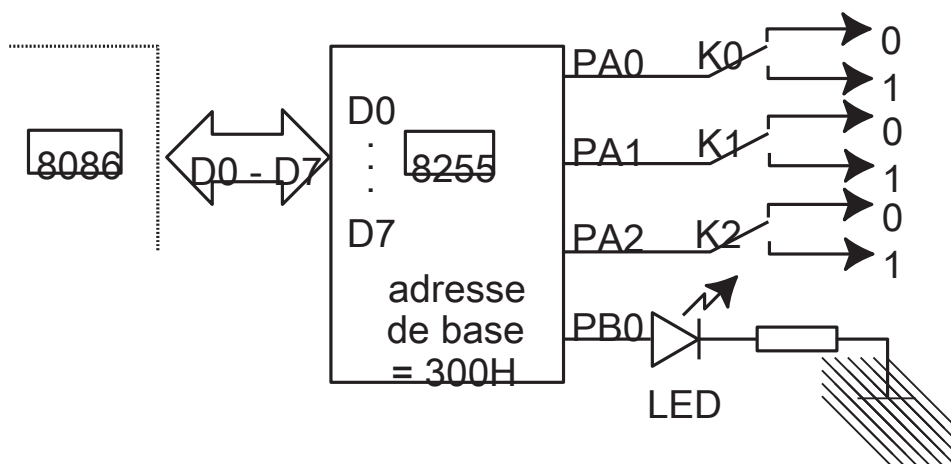
- A2=0 et A1=0: adresse du port A = adresse de base + 0 = 300H;
- A2=0 et A1=1: adresse du port B = adresse de base + 2 = 302H;
- A2=1 et A1=0: adresse du port C = adresse de base + 4 = 304H;
- A2=1 et A1=1: adresse du registre de commande = adresse de base + 6 = 306H.

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

Remarque: si on veut que le 8255 possède des adresses consécutives (par exemple 300H, 301H, 302H et 303H), on peut utiliser les schémas suivants qui exploitent tout le bus de données (D0-D15) du 8086:



Exemple de programmation: soit le montage suivant:



On veut que les leds s'allument lorsqu'on a la combinaison: $K_0=1$ et $K_1=0$ et $K_2=1$.

Programme:

```

portA      equ 300H      ;adressesdesregistresdu8255
portB      equ 302H
portC      equ 304H
controle   equ 306H

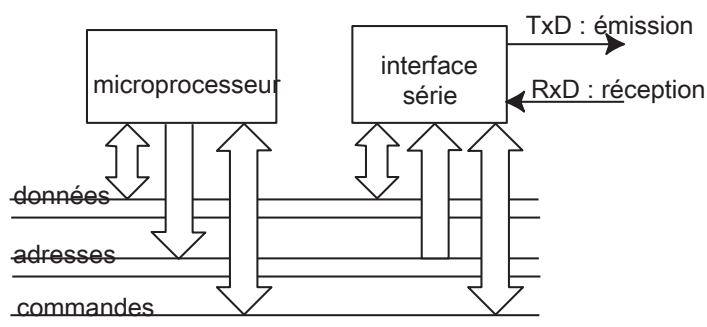
          mov dx,controle ;initialisationduportAenentrées/sorties
          mov al,90H      ;etduportBensortie(mode0):
          out dx,al       ;controle=10010000B=90H

boucle:    mov dx,portA   ;lectureduportA
          in  al,dx
          and al,00000111B ;masquagePA0,PA1etPA2
          cmp al,00000101B ;testPA0=1,PA1=0etPA2=1
          jne faux        ;non->alleraulabel"faux"...
          mov al,00000001B ;oui->mettrePB0`a1
          jmp suite       ;etcontinueraulabel"suite"
faux:     mov al,00000000B ;...mettrePB0`a0
suite:    mov dx,portB   ;écritureduportB
          out dx,al
          jmp boucle     ;retournerlireleportA

```

6.5 L'interface série 8250

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



Avantage: diminution du nombre de connexions (1 fil pour l'émission, 1 fil pour la réception).

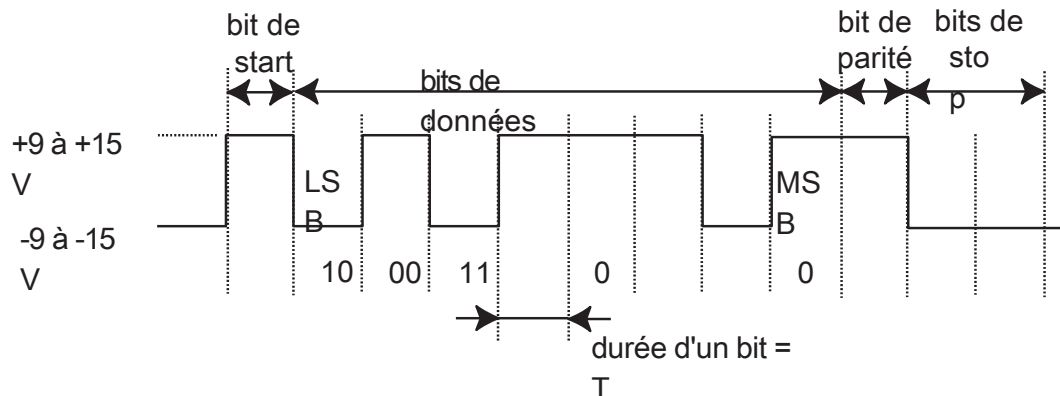
Inconvénient: vitesse de transmission plus faible que pour une interface parallèle.

Il existe deux types de transmissions série :

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

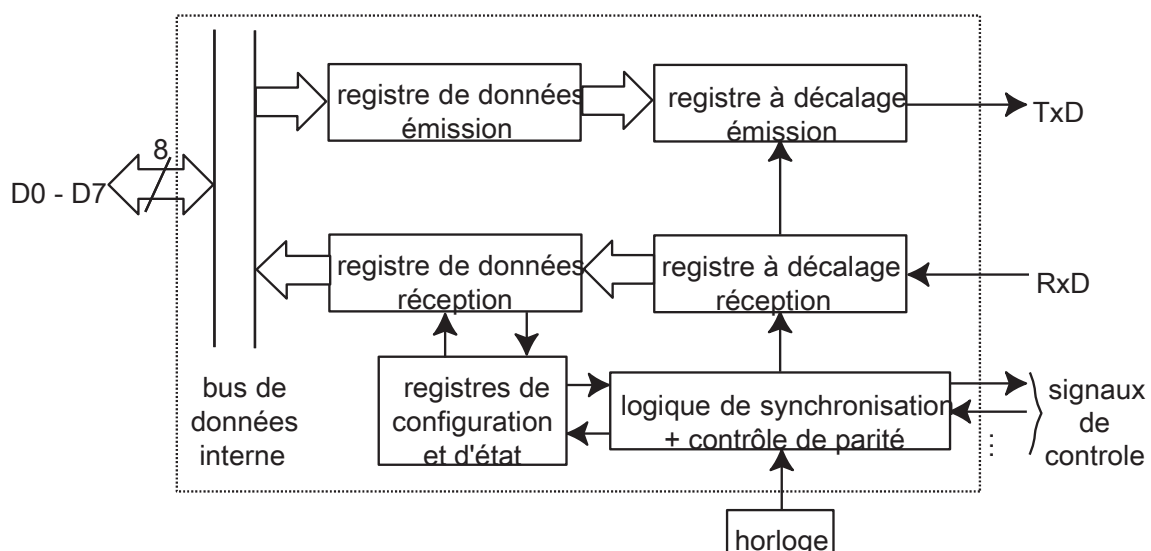
La transmission asynchrone la plus utilisée est celle qui est définie par la norme RS232.

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



- l'état 1 correspond à une tension négative comprise entre -9 et -15 V, l'état 0 à une tension positive comprise entre $+9$ et $+15$ V. Au repos, la ligne est à l'état 1 (tension négative);
- le bit de start marque le début de la transmission du caractère;
- les bits de données sont transmis un après l'autre en commençant par le bit de poids faible. Ils peuvent être au nombre de 5, 6, 7 ou 8. Chaque bit est maintenu sur la ligne pendant une durée déterminée T . L'inverse de cette durée définit la fréquence de débit = nombre de bits par seconde = vitesse de transmission. Les vitesses normalisées sont: 50, 75, 110, 134.5, 150, 300, 600, 1200, 2400, 4800, 9600 bits/s;
- le bit de parité (facultatif) est un bit supplémentaire dont la valeur dépend du nombre de bits de données égaux à 1. Il est utilisé pour la détection d'erreurs de transmission;
- les bits de stop (1, 1.5 ou 2) marquent la fin de la transmission du caractère.

Principe d'une interface série:



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

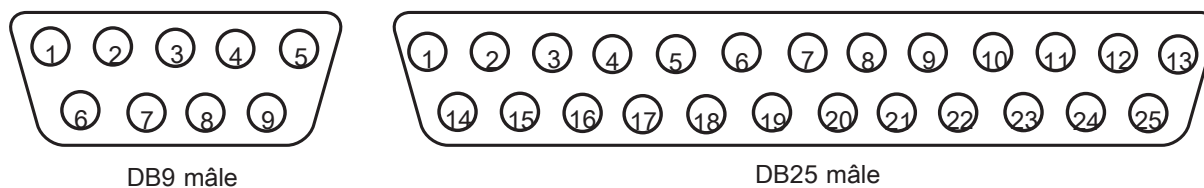
Exemples d'interfaces séries :

- 8251 (Intel);
- 8250 (National Semiconductor);
- 6850 (Motorola).

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

- les équipements terminaux de données (DTE : Data Terminal Equipment) qui génèrent les données à transmettre, exemple : un ordinateur;
- les équipements de communication de données (DCE : Data Communication Equipment) qui transmettent les données sur les lignes de communication, exemple : un modem.

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

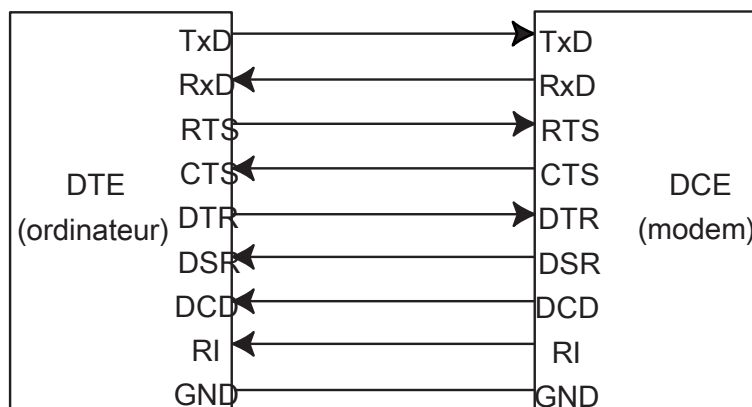


Différents signaux sont transportés par ces connecteurs :

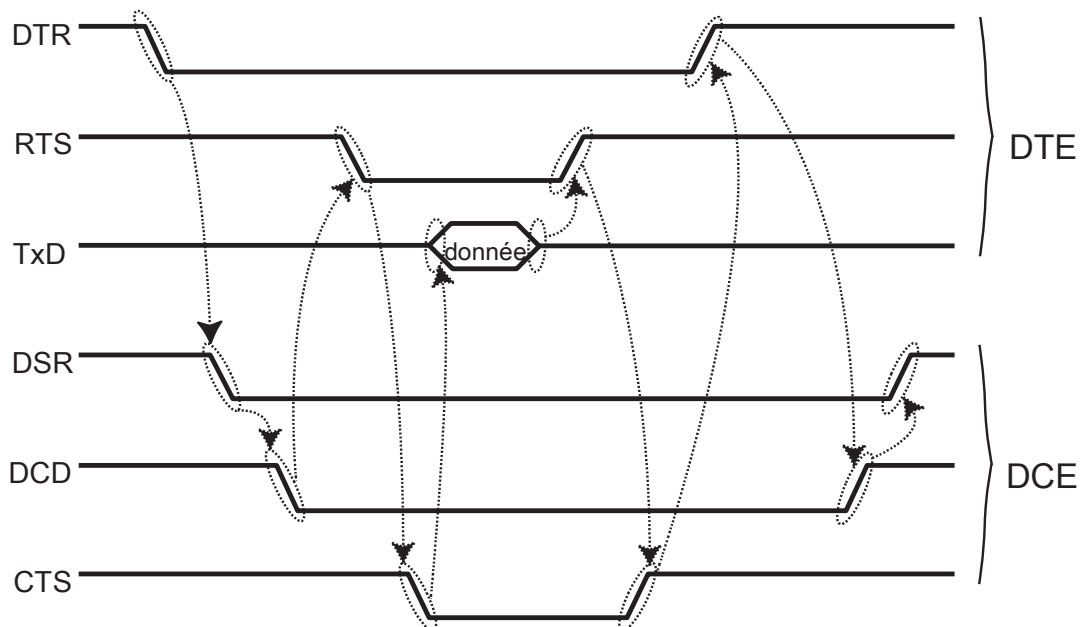
signal	n° broche DB9	n° broche DB25	Description	sens	
				DTE	DCE
TxD	3	2	TransmitData	sortie	entrée
RxD	2	3	ReceiveData	entrée	sortie
RTS	7	4	RequestToSend	sortie	entrée
CTS	8	5	ClearToSend	entrée	sortie
DTR	4	20	DataTerminalReady	sortie	entrée
DSR	6	6	DataSetReady	entrée	sortie
DCD	1	8	DataCarrierDetect	entrée	sortie
RI	9	22	RingIndicator	entrée	sortie
GND	5	7	Ground	—	—

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

Connexion entre DTE et DCE :



Dialogue entre DTE et DCE :

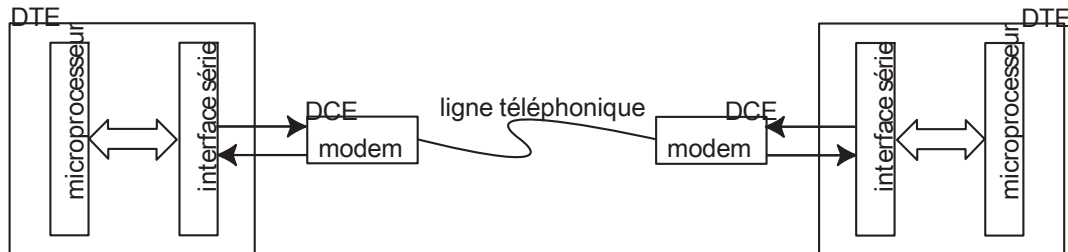


(les signaux de contrôles actifs à l'état bas = tension positive)

- quand le DTE veut transmettre des données, il active le signal DTR. Si le DCE est prêt à recevoir les données, il active le signal DSR puis le signal DCD : la communication peut débuter ;
- lorsque le DTE a une donnée à émettre, il active le signal RTS. Si le DCE peut recevoir la donnée, il active le signal CTS : le DTE envoie la donnée sur la ligne TxD ;
- si le DCE veut demander une pause dans la transmission, il désactive le signal CTS : le DTE arrête la transmission jusqu'à ce que CTS soit réactivé. C'est un contrôle matériel du flux de données ;
- Lorsque la transmission est terminée, les signaux RTS, CTS, DTR, DCD et DSR sont successivement désactivés.

Applications des liaisons s eries:

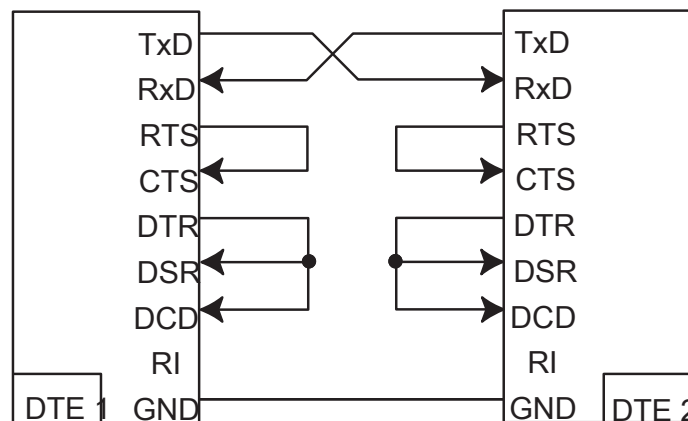
- transmission de donn ees  travers un ligne t el phonique:



Le modem transforme les signaux num eriques produits par l'interface s erie en signaux analogiques accept es par la ligne t el phonique et inversement (modulations num eriques FSK, PSK, ...)

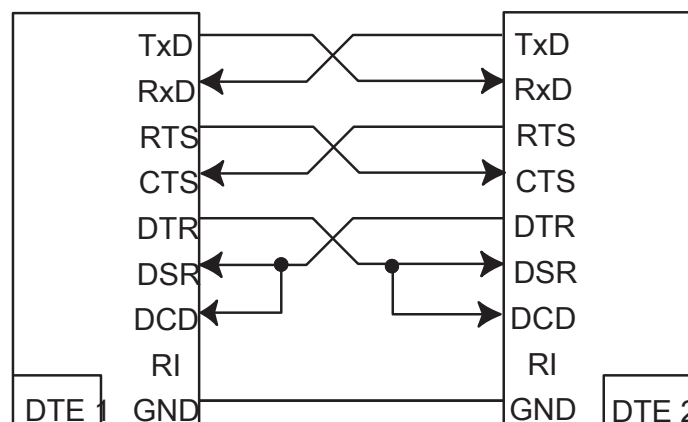
- liaison s erie directe entre deux DTE:

– liaison simple   3 fils: rebouclage (strapping) des signaux de contr ole:



Ce c ablage ne permet pas le contr ole mat eriel du flux entre les deux DTE.

– liaison compl ete: c able Null Modem:



Ce c ablage simule la pr esence d'un modem (DCE) en croisant les signaux de contr ole et permet le contr ole mat eriel du flux.

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

Brochage du 8250:

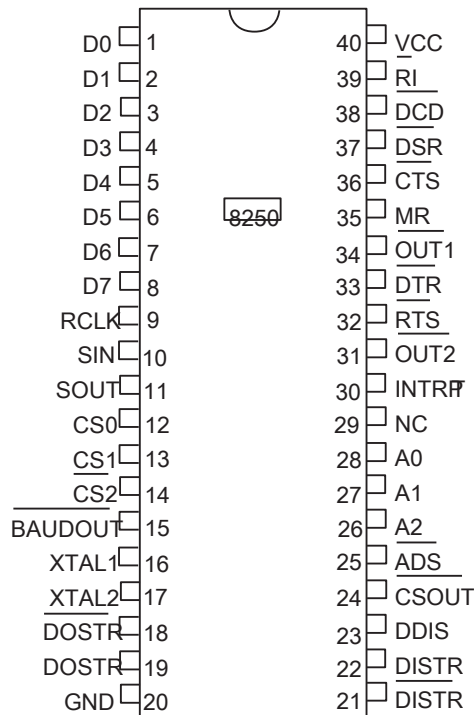
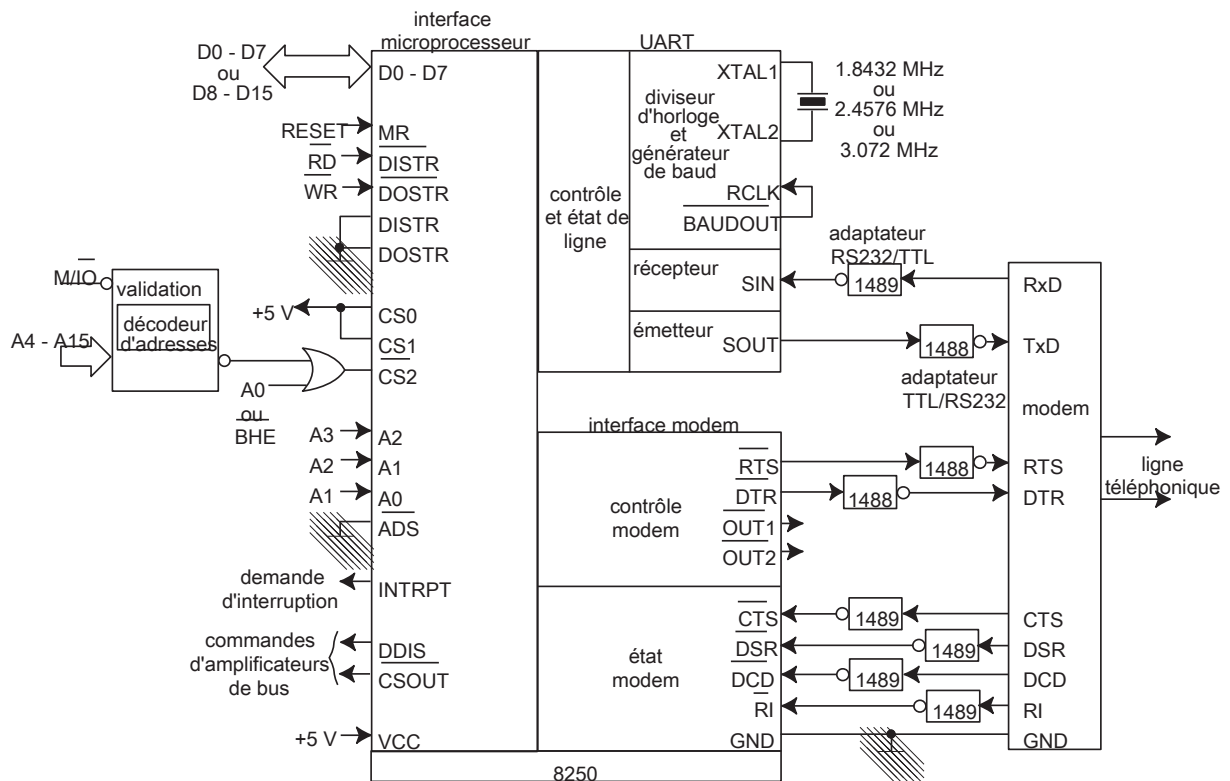


Schéma fonctionnel:



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

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

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

Structure des registres :

- **Line Control Register (LCR) :**

bits 0 et 1 : longueur du mot transmis,

bit1	bit0	
0	0	→ 5 bits
0	1	→ 6 bits
1	0	→ 7 bits
1	1	→ 8 bits

bit 2 : nombre de bits de stop, 0

→ 1 bit de stop,
1 → 1.5 bits de stop si 5 bits sont transmis, 2 bits de stop sinon;

bit 3 : autorisation de parité, 0

→ pas de parité, 1 → parité générée et vérifiée;

bit 4 : sélection de parité, 0

→ parité impaire, 1 → parité paire;

bit 5 : forçage de parité, 0

→ parité non forcée, 1 → parité fixe;

- bit6**: contrôle de l'état de la ligne Tx, 0 → ligne en fonctionnement normal, 1 → forçage de Tx à l'état 0 (break);
bit7: DLAB (Divisor Latch Access bit), 0 → accès aux registres d'émission, de réception et IER, 1 → accès au diviseur d'horloge.
- Line Status Register (LSR):**
 - bit0**: 1 → donnée reçue;
 - bit1**: 1 → erreur d'écrasement de caractère;
 - bit2**: 1 → erreur de parité;
 - bit3**: 1 → erreur de cadrage (bit de stop non valide);
 - bit4**: 1 → détection d'un état logique 0 sur Rx pendant une durée supérieure à la durée d'un mot;
 - bit5**: 1 → registre de transmission vide;
 - bit6**: 1 → registre d'adécalage vide;
 - bit7**: non utilisé, toujours à 0.
 - Modem Control Register (MCR):**
 - bit0**: DTR
 - bit1**: RTS
 - bit2**: OUT1
 - bit3**: OUT2
 - bit4**: 1 → fonctionnement en boucle: Tx connecté à Rx (modetest);
 - bit5**:
 - bit6**: inutilisés, toujours à 0.
 - bit7**:
 - Modem Status Register (MSR):**
 - bit0**: 1 → changement de CTS depuis la dernière lecture: delta CTS;
 - bit1**: 1 → delta DSR;
 - bit2**: 1 → delta RI (uniquement front montant sur RI);
 - bit3**: 1 → delta DCD;
 - bit4**: CTS
 - bit5**: DSR
 - bit6**: RI
 - bit7**: DCD
 - Diviseur d'horloge (DLM, DLL):** la vitesse de transmission est fixée par la valeur du diviseur d'horloge:

$$\text{vitesse (bit/s)} = \frac{\text{fréquence horloge (quartz)}}{16^{\times} (\text{DLM}, \text{DLL})}$$

Exemple de calcul: vitesse de transmission désirée = 1200 bit/s, fréquence d'horloge = 1.8432 MHz, détermination de la valeur du diviseur d'horloge:

$$\text{diviseur} = \frac{\text{fréquence horloge}}{16^{\times} \text{vitesse}} = \frac{1.8432 \times 10^6}{16^{\times} 1200} = 96 \Rightarrow \text{DLM} = 0 \text{ et } \text{DLL} = 96.$$

- **Receiver Buffer Register (RBR)**: contient la donnée reçue, cue.
- **Transmitter Holding Register (THR)**: contient la donnée à transmettre.
- **Interrupt Identification Register (IIR)**:
 - ~~bit0~~ : 0 → interruption en cours, 1 → pas d'interruption en cours;

~~bits 1 et 2~~ : source d'interruption,

bit2	bit1	
1	1	→ erreur
1	0	→ donnée reçue, cue
0	1	→ registre d'émission vide
0	0	→ changement d'état modem

(ordre de priorité d'écroissant);

~~bit3~~
~~bit4~~ }
~~bit5~~ } : inutilisées, toujours à 0.
~~bit6~~ }
~~bit7~~ }

- **Interrupt Enable Register (IER)**: autorisation des interruptions

~~bit0~~ : 1 → donnée reçue, cue;
~~bit1~~ : 1 → registre d'émission vide;
~~bit2~~ : 1 → erreur;
~~bit3~~ : 1 → changement d'état modem;
~~bit4~~ }
~~bit5~~ } : inutilisées, toujours à 0.
~~bit6~~ }
~~bit7~~ }

- **Scratch Register (SCR)**: registre d'usage général pouvant contenir des données temporaires.

Exemple de programmation: soit un UART 8250 dont le bus de données est connecté sur la partie faible du bus de données du microprocesseur 8086. L'adresse de base du 8250 est fixée à la valeur 200H par un décodeur d'adresses. La fréquence d'horloge du 8250 est de 1.8432 MHz. On veut:

- écrire une procédure `init` qui initialise le 8250 avec les paramètres suivants: 2400 bits/s, 8 bits par caractère, parité paire, 1 bit de stop (2400, 8, P, 1);
- écrire une procédure `envoi` qui émet un message contenu dans la zone de données `msg`. L'émission s'arrête lorsque le caractère EOT (End Of Text, code ASCII=03H) est rencontré;
- écrire une procédure `reception` qui reçoit une ligne de 80 caractères et la range dans une zone de données appelée `ligne`. En cas d'erreur de réception, envoyer le caractère NAK (No Acknowledge, code ASCII=15H) sinon envoyer le caractère ACK (Acknowledge, code ASCII=06H).

Programme:


```

RBR    equ    200H           ;adressesdesregistresdu8250
THR    equ    200H
DLL    equ    200H
DLM    equ    202H
IER    equ    202H
IIR    equ    204H
LCR    equ    206H
MCR    equ    208H
LSR    equ    20AH
MSR    equ    20CH
SCR    equ    20EH
EOT    equ    03H           ;caract`ereEndOfText
ACK    equ    06H           ;caract`ereAcknowledge
NAK    equ    15H           ;caract`ereNoAcknowledge
LIGNE  db    80dup(?)       ;zonederangementdescaract`eresre, cus
MSG    db    'Test8250',EOT ;message`aenvoyer

INIT   PROCNEAR             ;proc`edured'initialisationdu8250
      mov dx,LCR             ;DLAB=1pouracc`ederaudiviseur
      mov al,80H             ;d'horloge
      out dx,al
      mov dx,DLL             ;vitessede transmission=2400bit/s
      mov al,48              ;=>DLL=48...
      out dx,al
      mov dx,DLM             ;...etDLM=0
      mov al,0
      out dx,al
      mov dx,LCR             ;DLAB=0,8bitsdedonn`ees,
      mov al,00011011B       ;parit`epaire,1bitdestop
      out dx,al
      ret
INIT   ENDP

ENVOICARACTEREPROCNEAR    ;proc`edured'`emissionducontenudeAH
      mov dx,LSR             ;lectureduregistred'`etatdelaligne
attente `envoi:           ;attenteregistredetransmissionvide
      in  al,dx
      and al,20H             ;masquagebit5deLSR
      jz  attente `envoi     ;sibit5deLSR=0=>attente...
      mov dx,THR             ;...sinonenvoyerlecaract`ere
      mov al,ah              ;contenudansleregistreAH
      out dx,al
      ret
ENVOICARACTEREENDP

```

```

ENVOI PROCNEAR ;proc' edured'' emission dumessage
      mov si,offsetMSG ;point vers led' ebut dumessage
boucle:mov ah,[si] ;AH<-caract' ere` aenvoyer
      cmp AH,EOT ;findumessage?
      jz fin `envoi ;oui=>finproc' edure
      call ENVOICARACTERE ;non=>envoyer caract' ere...
      inc si ;...et passerau caract' eresuivant
      jmp boucle
fin `envoi:
      ret
ENVOI ENDP

RECEPTIONPROCNEAR ;proc' edureder' eception d'uneligne
      mov di,offsetLIGNE ;point vers d' ebut zoned' eception
      mov cx,80 ;compteur de caract' eresre, cus
attente `reception:
      mov dx,LSR ;lecture du registred'' etat de la ligne
      in al,dx
      test al,01H ;test del' etat du bit 0 de LSR
      jz attente `reception ;pas de caract' erere, cu=>attente
      test al,00001110B ;sinon tester reurs: bits 1,2,3 de LSR
      jz suite ;pas d'erreurs=>continuer
      mov ah,NAK ;erreurs=>envoyer NAK...
      call ENVOICARACTERE
      jmp attente `reception ;...et retourner attendre un caract' ere
suite:mov dx,RBR ;lire caract' erere, cu...
      in al,dx
      mov [di],al ;...et l' ranger dans LIGNE
      mov ah,ACK ;puis envoyer ACK
      call ENVOICARACTERE
      dec cx ;d' ecr' ementer compteur de caract' eres
      jz fin `reception ;si compteur=0=>finr' eception
      inc di ;sinon incr' ementer DI
      jmp attente `reception ;et aller attendre caract' eresuivant
fin `reception:
      ret
RECEPTIONENDP

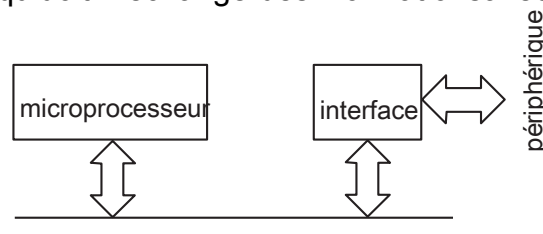
```

Chapitre 7

Les interruptions

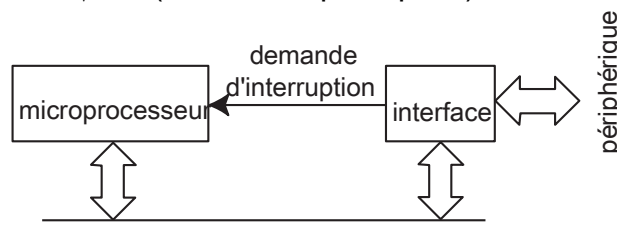
7.1 Définition d'une interruption

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



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

- **scrutation périodique (ou polling)** : le programme principal contient des instructions qui lisent cycliquement l'état des ports d'E/S.
Avantage : facilité de programmation.
Inconvénients :
 - perte de temps s'il y a de nombreux périphériques à interroger ;
 - de nouvelles données ne sont pas toujours présentes ;
 - des données peuvent être perdues si elles changent rapidement.
- **interruption** : lorsqu'une donnée apparaît sur un périphérique, le circuit d'E/S signale au microprocesseur pour que celui-ci effectue la lecture de la donnée : c'est une demande d'interruption (IRQ: Interrupt Request) :



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

Exemples de périphériques utilisant les interruptions:

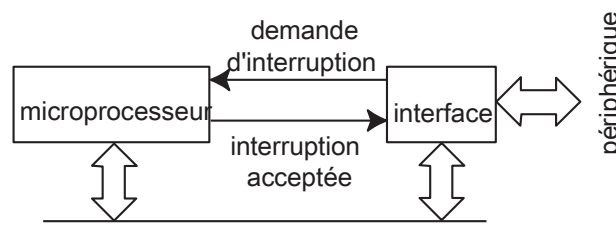
- clavier: demande d'interruption lorsqu'une touche est enfoncée;
- port série: demande d'interruption lors de l'arrivée d'un caractère sur la ligne de transmission.

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

7.2 Prise en charge d'une interruption par le microprocesseur

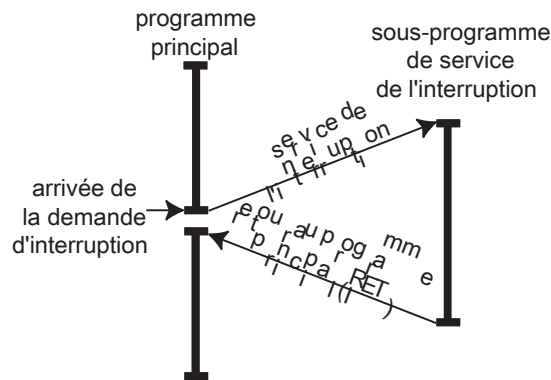
À la suite d'une demande d'interruption par un périphérique:

- le microprocesseur termine l'exécution de l'instruction en cours;
- il range le contenu des principaux registres sur la pile de sauvegarde: pointeur d'instruction, flags, ...
- il émet un accusé de réception de la demande d'interruption (Interrupt Acknowledge) indiquant au circuit d'E/S que la demande d'interruption est acceptée:



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

- il abandonne l'exécution du programme en cours et va exécuter un sous-programme de service de l'interruption (ISR: Interrupt Service Routine);
- après l'exécution de l'ISR, les registres sont restaurés à partir de la pile et le microprocesseur reprend l'exécution du programme qu'il avait abandonné:

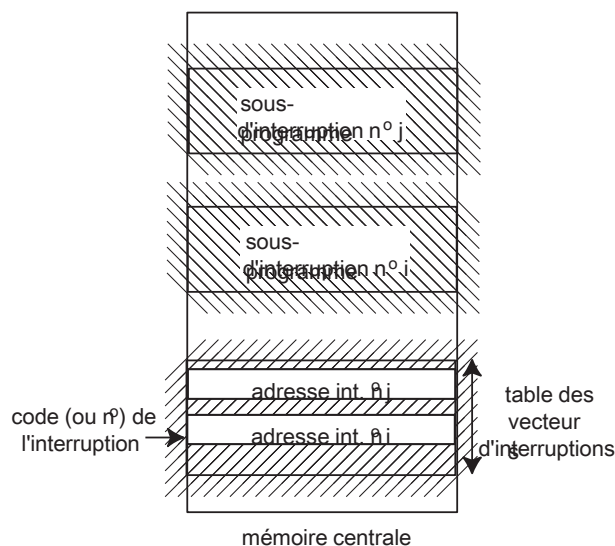


Remarque: la dernière instruction d'un sous-programme de service d'interruption doit être l'instruction IRET: retour d'interruption.

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

7.3 Adresses des sous-programmes d'interruptions

Lorsqu'une interruption survient, le microprocesseur a besoin de connaître l'adresse du sous-programme de service de cette interruption. Pour cela, la source d'interruption place sur le bus de données un code numérique indiquant la nature de l'interruption. Le microprocesseur utilise ce code pour rechercher dans une table en mémoire centrale l'adresse du sous-programme d'interruption à exécuter. Chaque élément de cette table s'appelle un vecteur d'interruption:



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

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

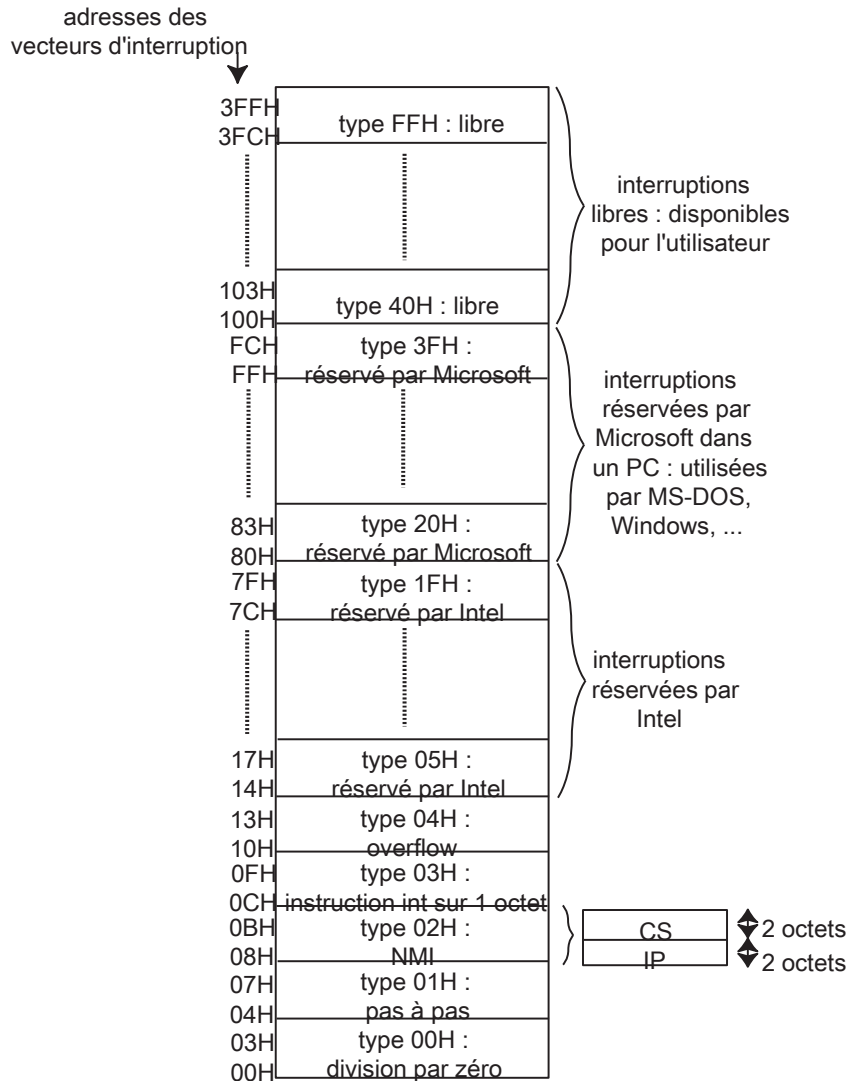
7.4 Les interruptions du 8086

Le microprocesseur 8086 peut gérer jusqu'à 256 interruptions. Chaque interruption reçoit un numéro compris entre 0 et 255 appelé "type" de l'interruption.

Trois sortes d'interruptions sont reconnues par le 8086 :

- interruptions matérielles produites par l'activation des lignes INTR et NMI du microprocesseur;
- interruptions logicielles produites par l'instruction INTn, où n est le type de l'interruption;
- interruptions processeur générées par le microprocesseur en cas de débordement, de division par zéro ou lors de l'exécution pas à pas d'un programme.

Les interruptions du 8086 sont vectorisées. La table des vecteurs d'interruptions doit obligatoirement commencer à l'adresse physique 00000H dans la mémoire centrale. Chaque vecteur d'interruption est constitué de 4 octets représentant une adresse logique du type CS:IP.



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

$$\text{adresse vecteur d'interruption} = 4 \times \text{type de l'interruption}$$

Exemple: interruption 20H, adresse du vecteur = $4 \times 20H = 80H$.

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

7.5 Le contrôleur programmable d'interruptions 8259

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

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

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

Brochage du 8259:

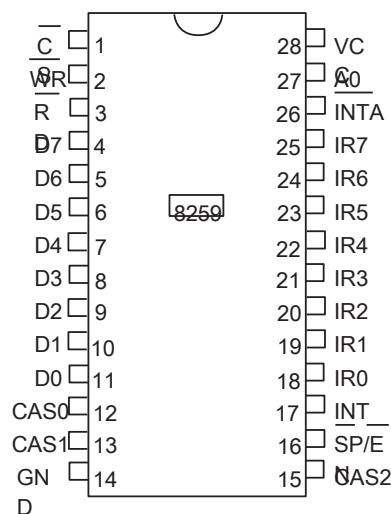
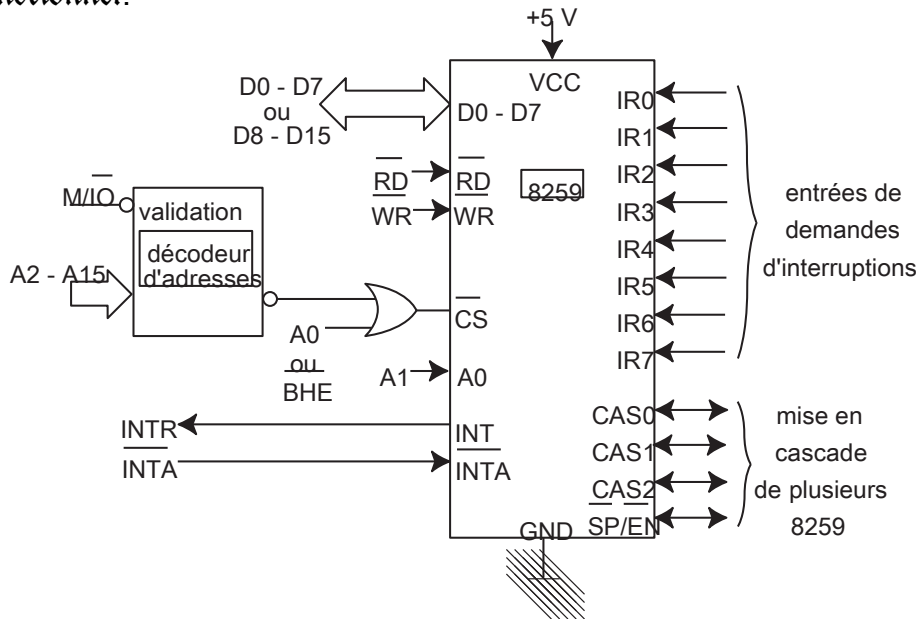
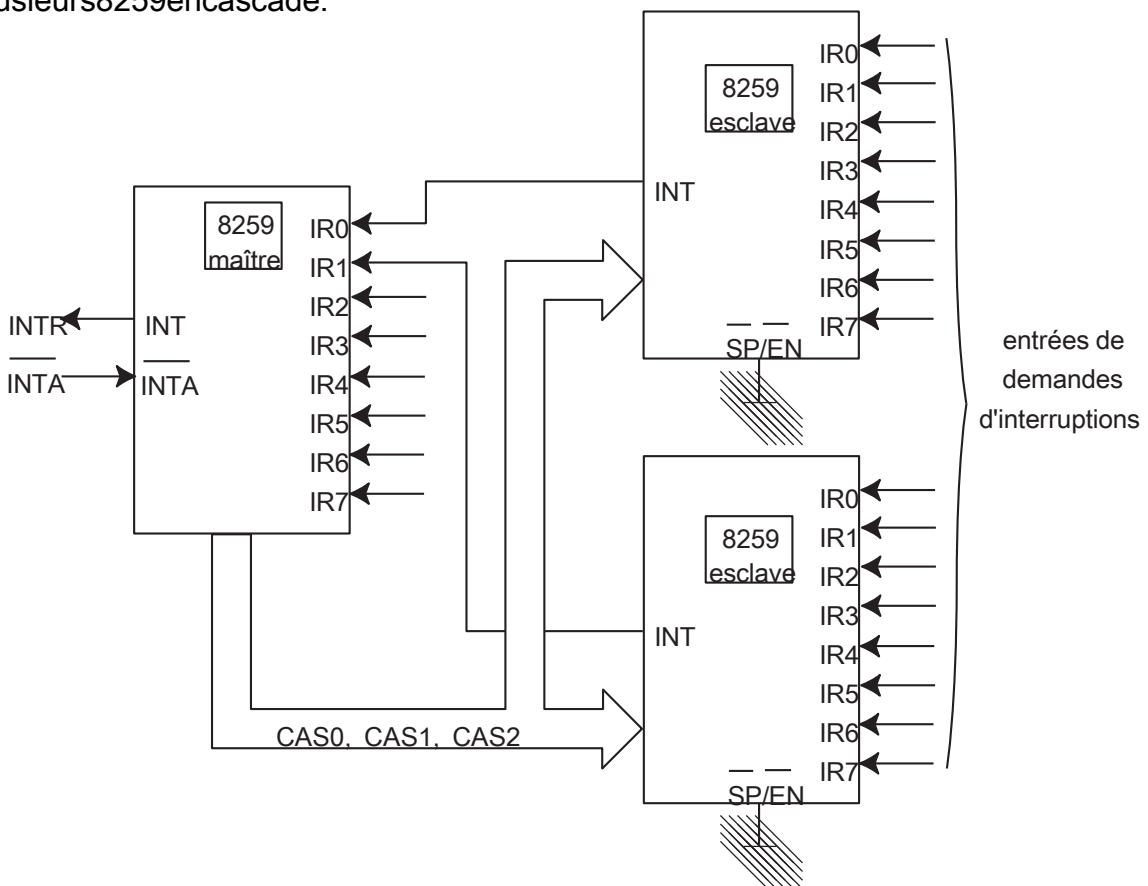


Schéma fonctionnel:



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



Annexe

Jeux d'instructions du 8086

Transfert de données :

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

Instructions arithmétiques :

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

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

Instructions logiques:

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

Instructions sur les caractéristiques :

Préfixes	
REP	Répétition tant que CX n'est pas nul
REP ou REPZ	Répétition tant qu'il y a égalité et que CX n'est pas nul
REPNE ou REP NZ	Répétition tant qu'il n'y a pas égalité et que CX n'est pas nul
Instructions	
MOV ou MOVB/MOVS/W	Déplacement de blocs d'octets ou de mots
CMP ou CMPB/CMP/W	Comparaison de blocs d'octets ou de mots
SCAS ou SCASB/SCAS/W	Exploration d'un bloc d'octets ou de mots
LODS ou LODSB/LODS/W	Transfert d'un octet ou d'un mot dans AL ou AX
STOS ou STOSB/STOS/W	Chargement d'un bloc d'octets ou de mots par AL ou AX

Instructions de branchements :

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

Instructions de branchements conditionnels:

Sauts conditionnels	
JA ou JNB ⁽¹⁾	Sautsi « supérieur » (siCF+ZF=0)
JAE ou JNB ⁽¹⁾	Sautsi « supérieur ou égal » (siCF=0)
JB ou JNAE ⁽¹⁾	Sautsi « inférieur » (siCF=1)
JBE ou JNA ⁽¹⁾	Sautsi « inférieur ou égal » (siCF+ZF=1)
JC	Sautencasderetenue (siCF=1)
JE ou JZ	Sautsi « égal » ou « nul » (siZF=1)
JG ou JNL ⁽²⁾	Sautsi « plus grand » (si(SF ⊕ OF)+ZF=0)
JGE ou JNL ⁽²⁾	Sautsi « plus grand ou égal » (siSF ⊕ OF=0)
JL ou JNG ⁽²⁾	Sautsi « plus petit » (siSF ⊕ OF=1)
JLE ou JNG ⁽²⁾	Sautsi « plus petit ou égal » (si(SF ⊕ OF)+ZF=1)
JNC	Sautsi « pasderetenue » (siCF=0)
JNE ou JNZ	Sautsi « non égal » ou « non nul » (si(ZF=0))
JNO	Sautsi « pasdedépassement » (siOF=0)
JNP ou JPO	Sautsi « parité impaire » (siPF=0)
JNS	Sautsi « signe positif » (siSF=0)
JO	Sautsi « dépassement » (siOF=1)
JP ou JPE	Sautsi « parité paire » (siPF=1)
JS	Sautsi « signe négatif » (siSF=1)

(1) concernedesnombresnonsignés.

(2) concernedesnombressignés.

Instructions de contrôle de du 8086:

Opérations sur les flags	
STC	Metleflagderetenue`a1
CLC	Effaceleflagderetenue
CMC	Inversel'étatduflagderetenue
STD	Metleflagdedirection`a1(d'incrémententation)
CLD	Metleflagdedirection`a0(incrémententation)
STI	AutoriselesinterruptionssurINTR
CLI	InterditlesinterruptionssurINTR
Synchronisation avec l'extérieur	
HLT	Arrêtdumicroprocesseur(sortiedecet'étatparinterruptionoureset)
WAIT	AttentetantqueTESTn'estpas`a0
ESC	Préfixe=instructiondestinée`auncoprocésseur
LOCK	Préfixe=réservationdubuspourl'instruction
Pas d'opération	
NOP	Pasd'opération

Bibliographie

- [1] P. André . *Liaison RS232*. Dunod, Paris, 1998.
- [2] T.C. Bartee . *Digital Computer Fundamentals*. McGraw-Hill, Tokyo, 1981.
- [3] J. Campbell . *L'interface RS-232*. Sybex, Paris, 1984.
- [4] B. Fabrot . *Assembleur pratique*. Marabout Informatique, Aleur, Belgique, 1996.
- [5] A.B. Fontaine . *Le microprocesseur 16 bits 8086/8088 – Matériel, logiciel, système d'exploitation*. Masson, Paris, 1988.
- [6] B. Geoffrion . *8086- 8088 – Programmation en langage assembleur*. Editions Radio, Paris, 1986.
- [7] J.P. Hayes . *Computer Architecture and Organization*. McGraw-Hill, Tokyo, 1982.
- [8] S. Leibson . *Manuel des interfaces*. McGraw-Hill, Paris, 1984.
- [9] H. Lilen . *Introduction à l'informatique – Du microprocesseur au micro-ordinateur*. Editions Radio, Paris, 1982.
- [10] H. Lilen . *8088 et ses périphériques – Les circuits clés des IBM PC et compatibles*. Editions Radio, Paris, 1985.
- [11] H. Lilen . *8088 – Assembleur IBM PC et compatibles*. Editions Radio, Paris, 1986.
- [12] H. Lilen . *Cours fondamental des microprocesseurs*. Editions Radio, Paris, 1987.
- [13] H. Lilen . *Microprocesseurs – Du CISC au RISC*. Dunod, Paris, 1995.
- [14] G.H. MacEwen . *Introduction to Computer Systems*. McGraw-Hill, Tokyo, 1981.
- [15] A. Mariatte . *PC, modems et serveurs*. P.S.I, Lagny, France, 1986.
- [16] P. Mercier . *Assembleur facile*. Marabout Informatique, Aleur, Belgique, 1989.
- [17] P. Mercier . *Le matériel du MS-DOS et du BIOS – Les interruptions – Organisation interne*. Marabout Informatique, Aleur, Belgique, 1989.
- [18] P. Mercier . *Les interruptions du MS-DOS*. Marabout Informatique, Aleur, Belgique, 1990.
- [19] A. Osborne . *Initiation aux micro-ordinateurs – Niveau 2*. Editions Radio, Paris, 1981.
- [20] J.B. Peatman . *Microcomputer Based Design*. McGraw-Hill, Tokyo, 1981.
- [21] E. Pissaloux . *Pratique de l'assembleur I80x86 – Cours et exercices*. Hermès, Paris, 1994.
- [22] H. Schakel . *Programmer en assembleur sur PC*. Micro Application, Paris, 1995.

- [23] C. Tavernier . *Modems*. ETSF, Paris, 1993.
- [24] M. Tischer et B. Jennrich . *La bible PC-Programmationsystème*. MicroApplication, Paris, 1997.
- [25] R. Tourki . *L'ordinateur PC-Architecture et programmation-Cours et exercices*. Centre de Publication Universitaire, Tunis, 2002.
- [26] J. Tracy Kidder . *The Soul of a New Machine*. Atlantic-Little Brown, U.S.A, 1981.
- [27] J.M. Trio . *Microprocesseurs 8086-8088-Architecture et programmation*. Eyrolles, Paris, 1984.
- [28] R. Zaks et A. Wolfe . *Du composant à système-Introduction aux microprocesseurs*. Sybex, Paris, 1988.