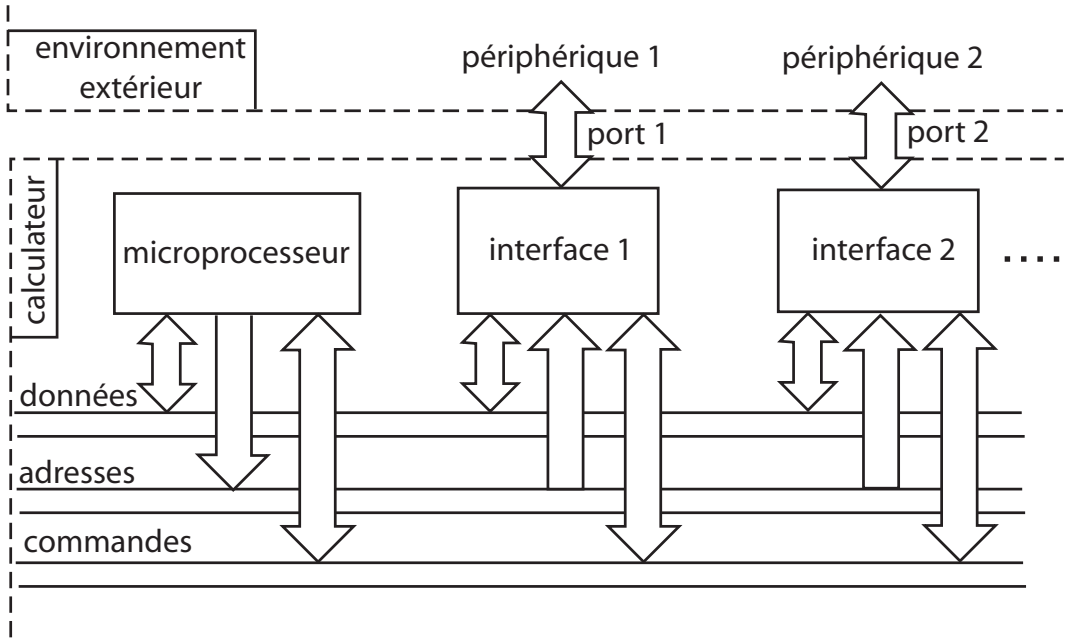
Chapitre 6

Les interfaces d'entrées/sorties

6.1 Définitions

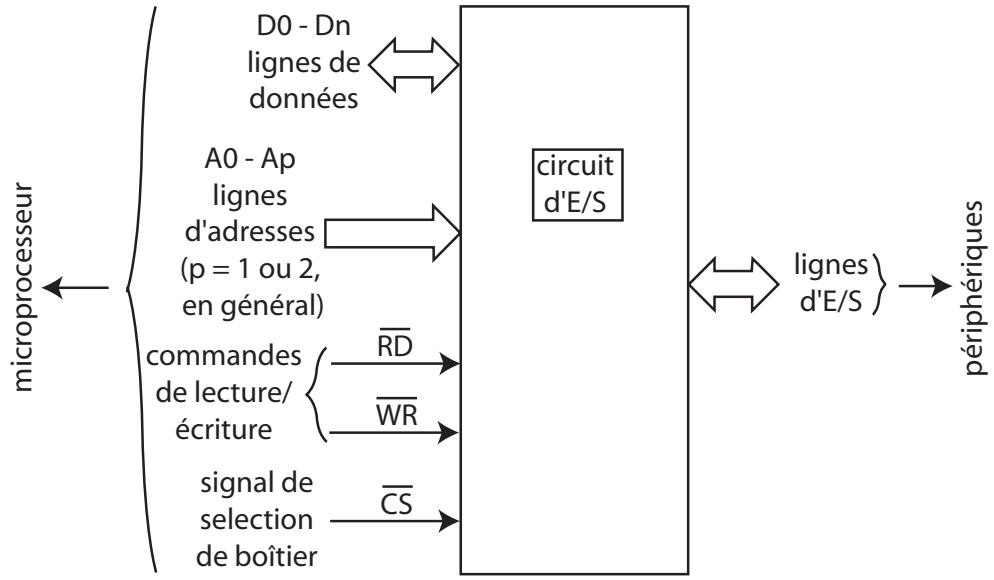
Une **interface d'entrées/sorties** est un circuit intégré permettant au microprocesseur de communiquer avec l'environnement extérieur (périphériques) : clavier, écran, imprimante, modem, disques, processus industriel, ...
Les interfaces d'E/S sont connectées au microprocesseur à travers les bus d'adresses,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

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



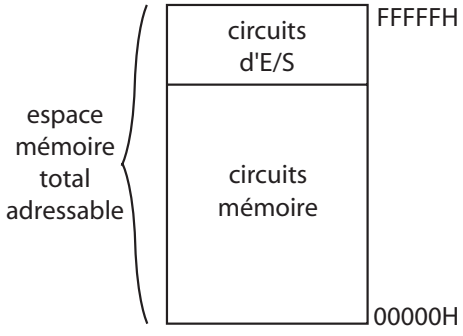
6.2 Adressage des ports d'E/S

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 assigné une adresse : le microprocesseur accède à un port d'E/S en spécifiant l'adresse de l'un de ses registres.
Le microprocesseur peut voir les adresses des ports d'E/S de deux manières :

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

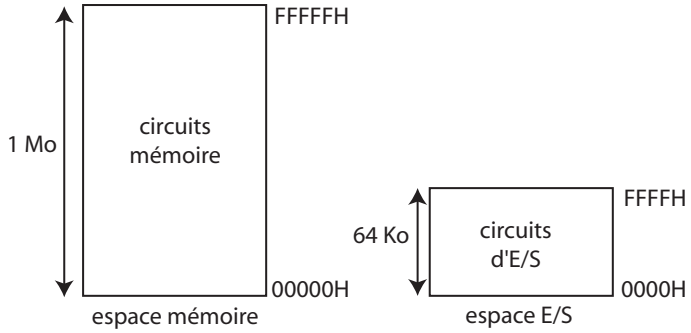


  Conséquences :
  - l'espace d'adressage des mémoires diminue ;
  - l'adressage des ports d'E/S se fait avec une adresse de même longueur (même nombre de bits) que pour les cases mémoires ;

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

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

  C'est le cas du microprocesseur 8086 :



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

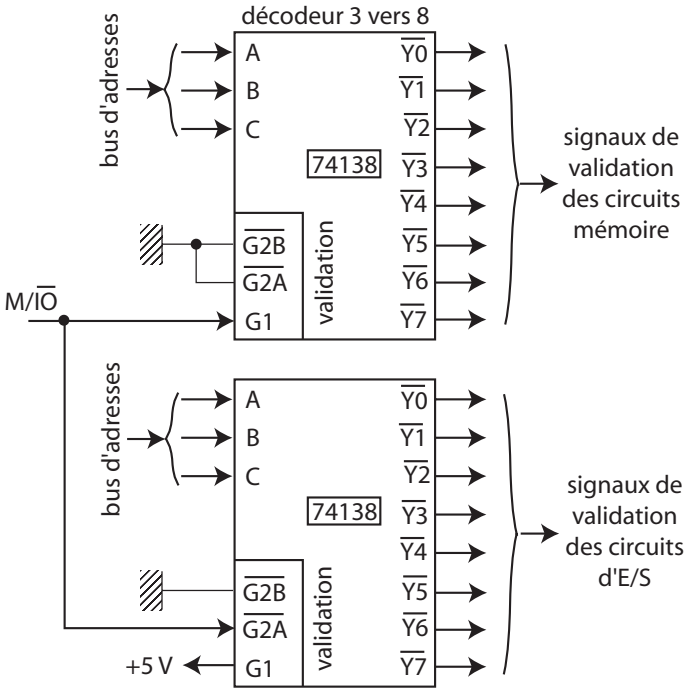
Remarque : l'adressage indépendant des ports d'E/S n'est possible que pour les microprocesseurs possédant un signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S ainsi que les instructions spécifiques pour l'accès aux ports d'E/S. Par contre, l'adressage cartographique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

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

Le 8086 dispose d'un espace mémoire de 1 Mo (adresse d'une case mémoire sur 20 bits) et d'un espace d'E/S de 64 Ko (adresse d'un port d'E/S sur 16 bits).
Le signal permettant de différencier l'adressage de la mémoire de l'adressage des ports d'E/S est la ligne $\mathbf{M/\overline{IO}}$:

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

Ce signal est utilisé pour valider le décodage d'adresse dans les deux espaces :



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

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

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

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

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

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

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

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

Ecriture d'un port d'E/S :

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

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

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

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

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

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

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

Exemples :

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

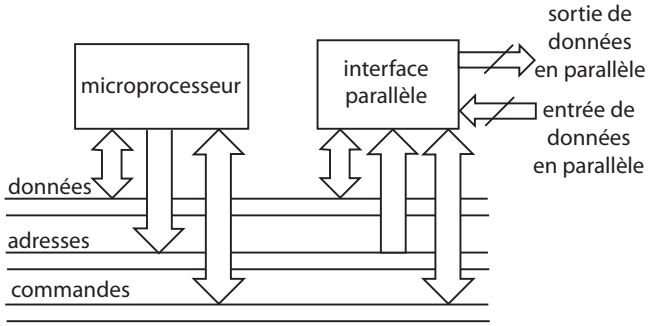
```
mov dx,300H
in al,dx
```

- 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 8255

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



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

Brochage du 8255 :

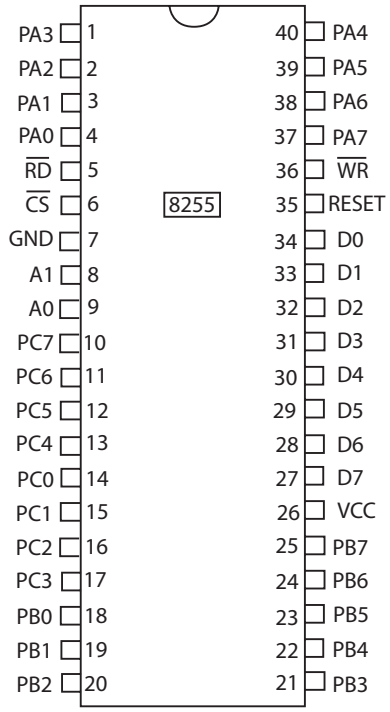
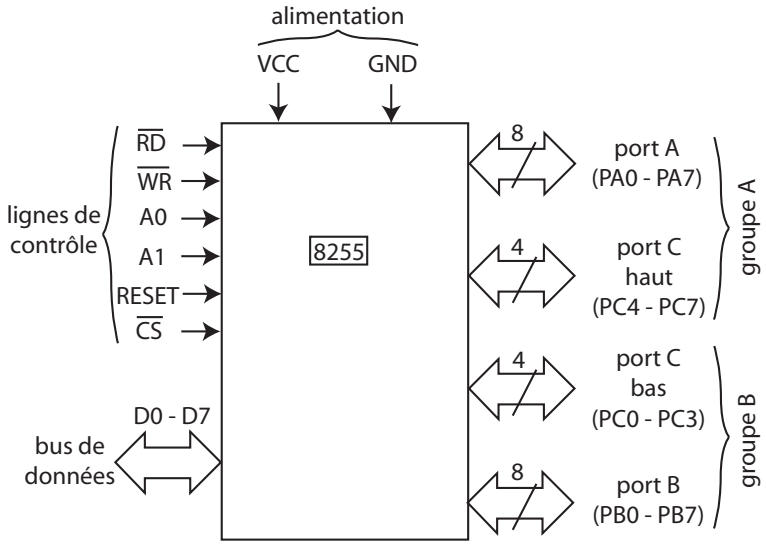


Schéma fonctionnel :



Le 8255 contient 4 registres :

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

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

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

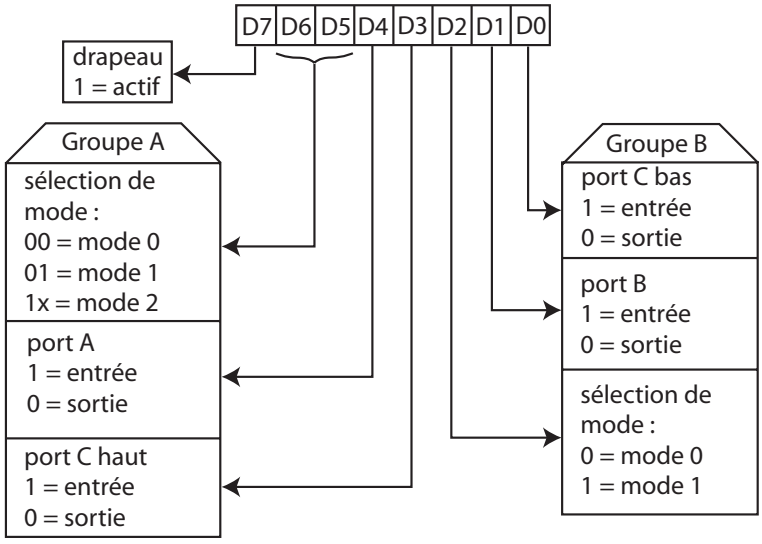
Remarque : le registre de commande est accessible uniquement en écriture, la lecture de ce registre n'est pas autorisée.

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

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

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

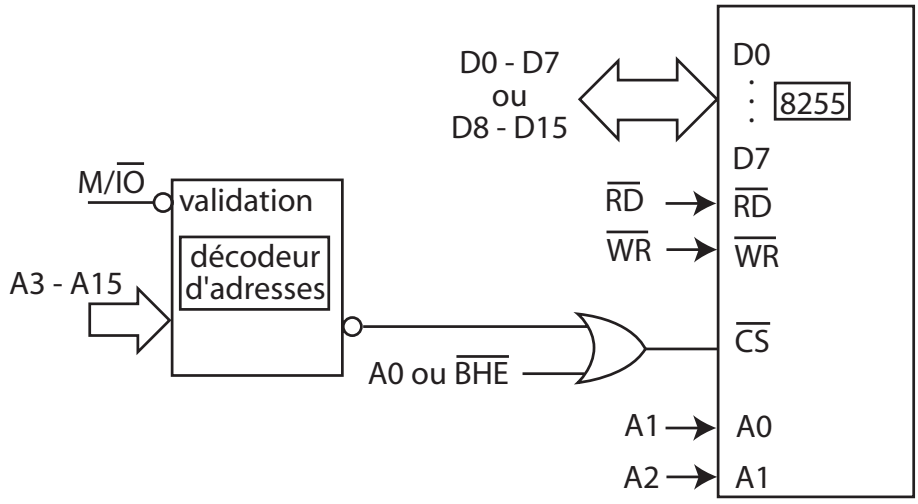
Structure du registre de commande :



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

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

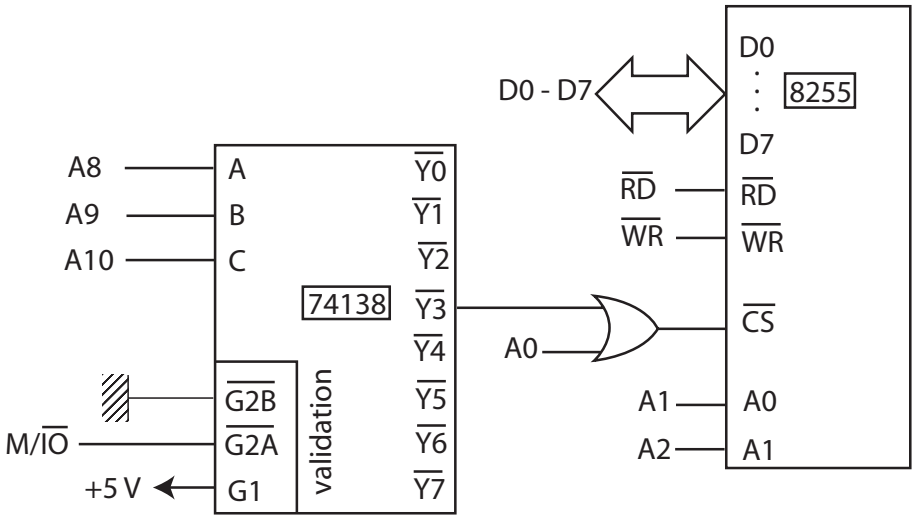
Ainsi l'un de ces deux signaux A0 ou $\overline{\text{BHE}}$ doit être utilisé pour sélectionner le 8255 :



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) :



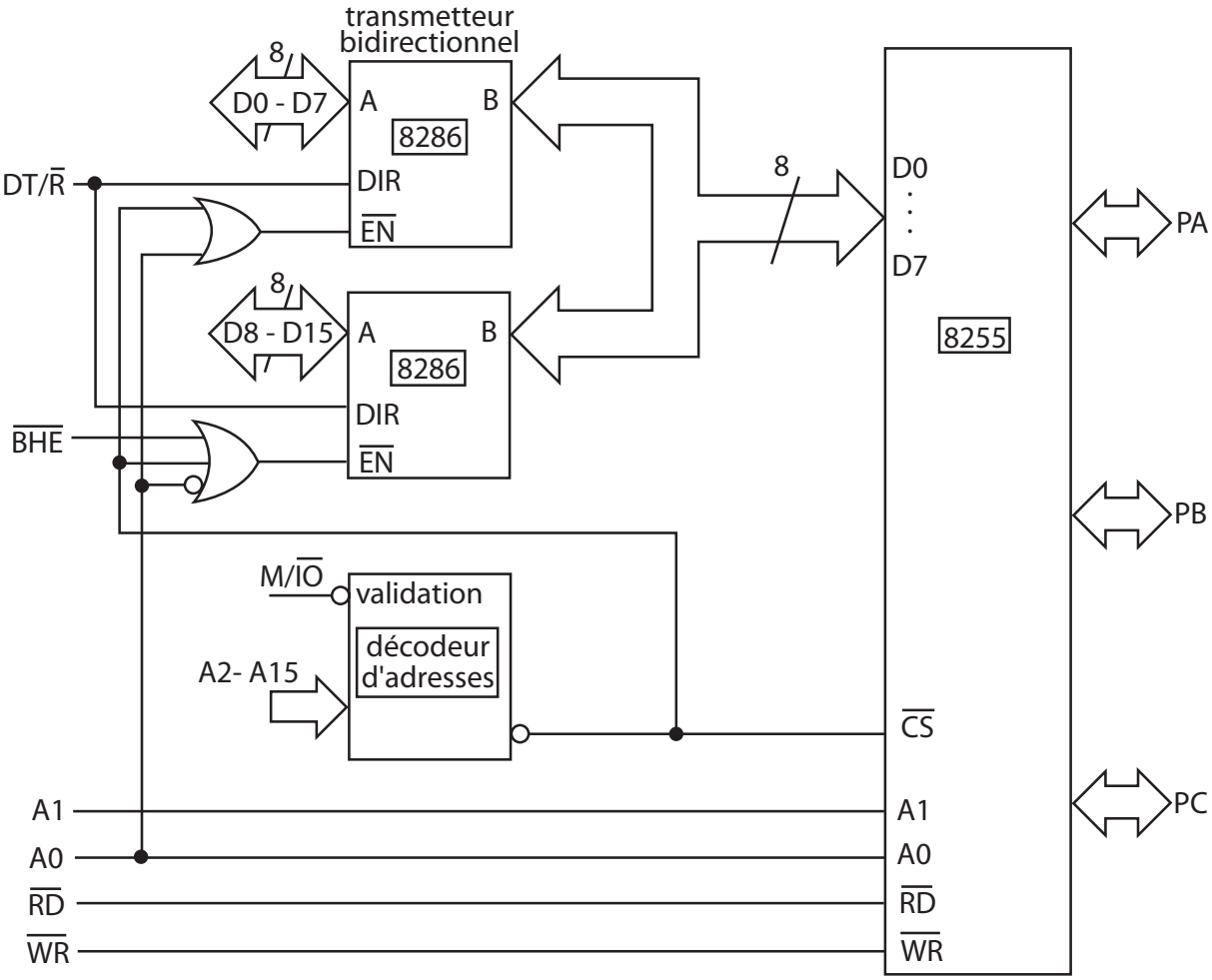
Détermination de l'adresse du 8255 :

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

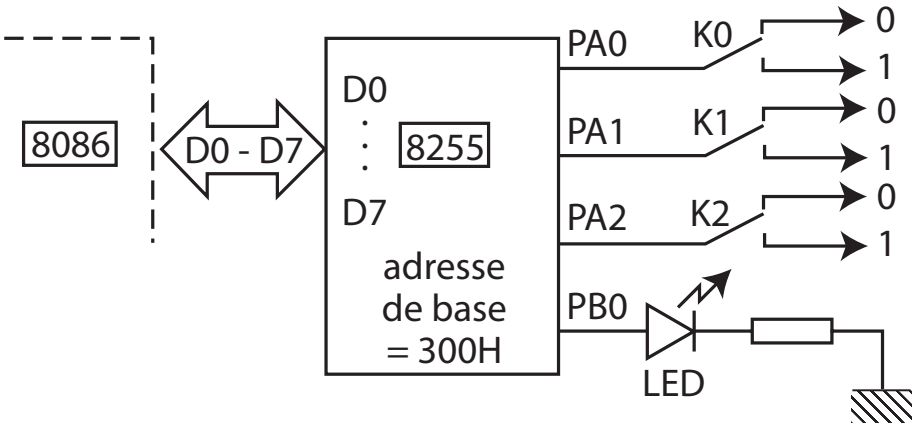
- A2 = 0 et A1 = 0 : adresse du port A = adresse de base + 0 = 300H ;
- A2 = 0 et A1 = 1 : adresse du port B = adresse de base + 2 = 302H ;
- A2 = 1 et A1 = 0 : adresse du port C = adresse de base + 4 = 304H ;
- A2 = 1 et A1 = 1 : adresse du registre de commande = adresse de base + 6 = 306H.

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 le schéma suivant qui exploite tout le bus de données (D0 - D15) du 8086 :



Exemple de programmation : soit le montage suivant :



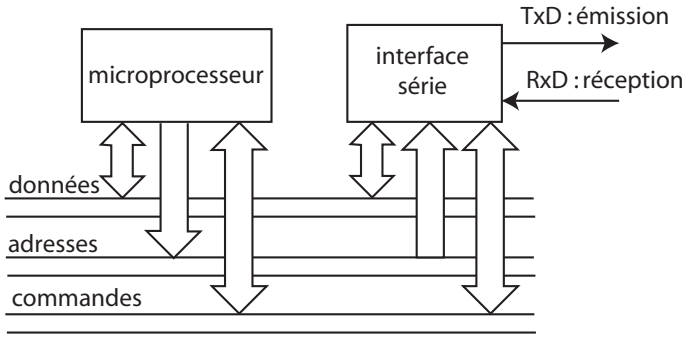
On veut que la led s'allume lorsqu'on a la combinaison : $\mathbf{K0 = 1}$ et $\mathbf{K1 = 0}$ et $\mathbf{K2 = 1}$.

Programme :

```
portA       equ   300H          ; adresses des registres du 8255
portB       equ   302H
portC       equ   304H
controle    equ   306H

            mov   dx,controle   ; initialisation du port A en entrée
            mov   al,90H        ; et du port B en sortie (mode 0) :
            out   dx,al         ; controle = 10010000B = 90H

boucle :    mov   dx,portA      ; lecture du port A
            in    al,dx
            and   al,00000111B  ; masquage PA0, PA1 et PA2
            cmp   al,00000101B  ; test PA0 = 1, PA1 = 0 et PA2 = 1
            jne   faux          ; non -> aller au label ''faux'' ...
            mov   al,00000001B  ; oui -> mettre PB0 à 1
            jmp   suite         ; et continuer au label ''suite''
faux :      mov   al,00000000B  ; ... mettre PB0 à 0
suite :     mov   dx,portB      ; écriture du port B
            out   dx,al
            jmp   boucle        ; retourner lire le port A
```

6.5 L'interface série 8250

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

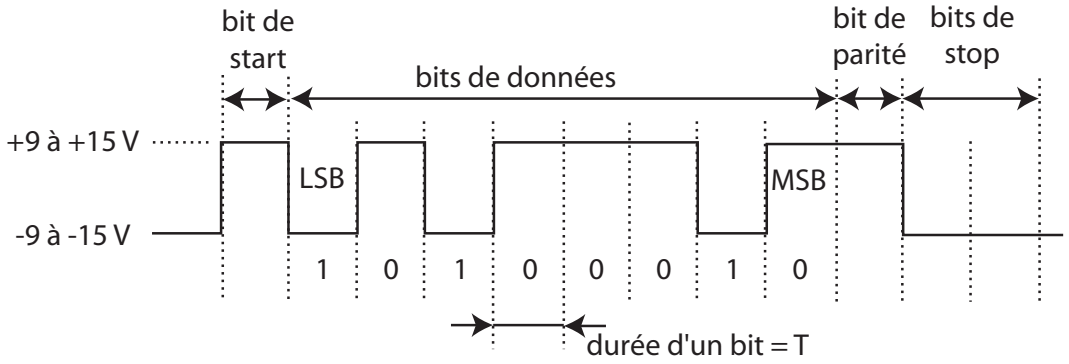


Avantage : diminution du nombre de connexions (1 fil pour l'émission, 1 fil pour la réception).
Inconvénient : vitesse de transmission plus faible que pour une interface parallèle.
Il existe deux types de transmissions séries :

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

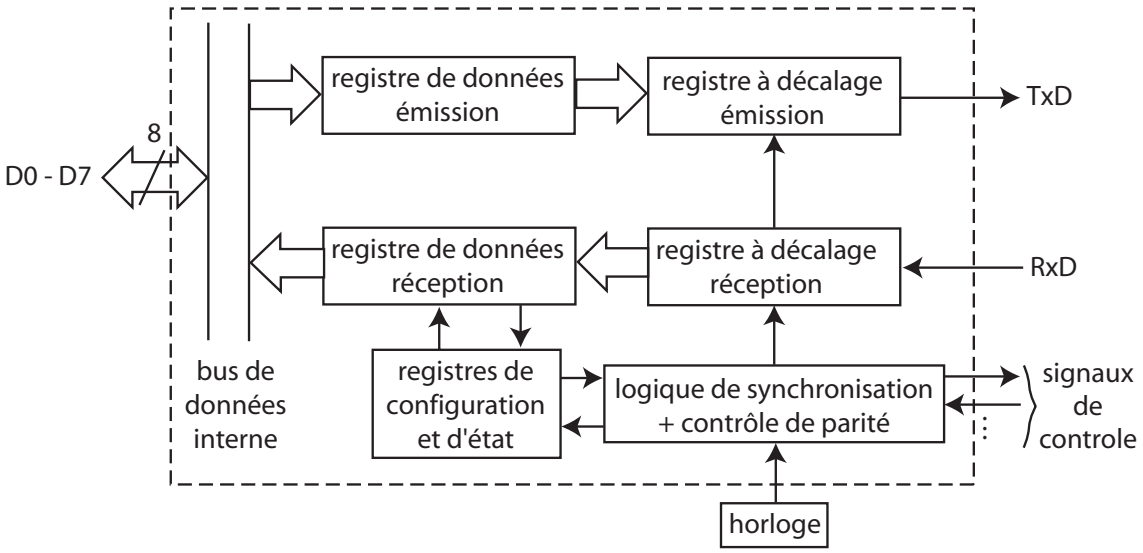
La transmission asynchrone la plus utilisée est celle qui est définie par la norme **RS232**.

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



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

Principe d'une interface série :



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

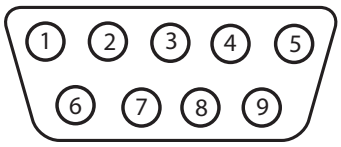
Exemples d'interfaces séries :

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

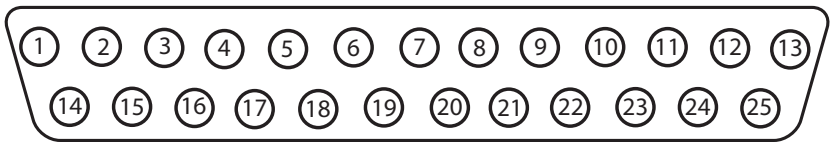
Connexion de deux équipements par une liaison série RS232 : les équipements qui peuvent être connectés à travers une liaison série RS232 sont de deux types :

- 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** :



DB9 mâle



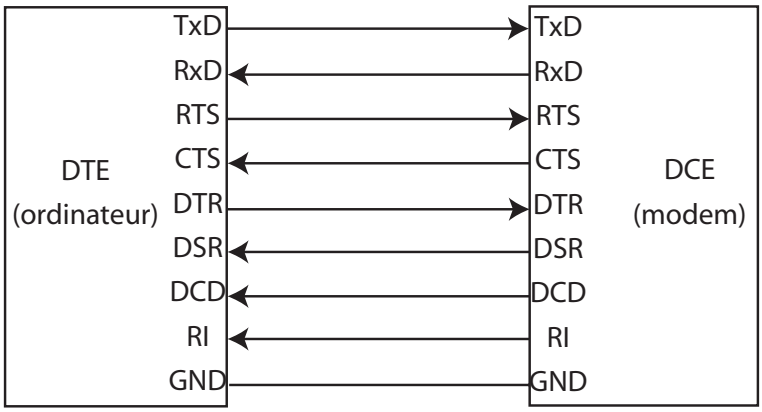
DB25 mâle

Différents signaux sont transportés par ces connecteurs :

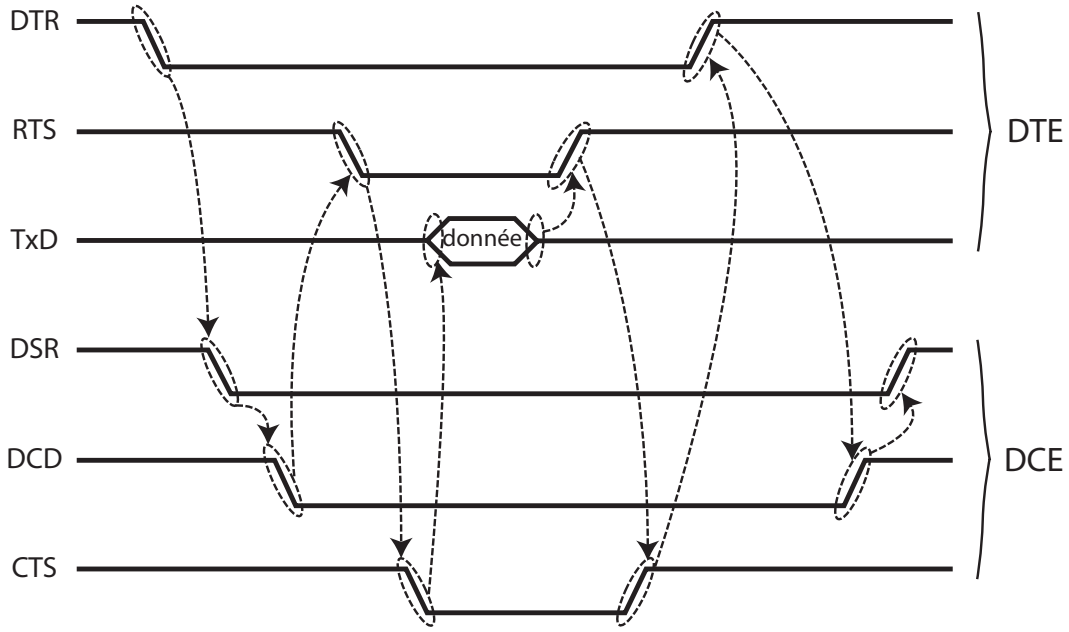
signal	n° broche DB9	n° broche DB25	description	sens	
				DTE	DCE
TxD	3	2	Transmit Data	sortie	entrée
RxD	2	3	Receive Data	entrée	sortie
RTS	7	4	Request To Send	sortie	entrée
CTS	8	5	Clear To Send	entrée	sortie
DTR	4	20	Data Terminal Ready	sortie	entrée
DSR	6	6	Data Set Ready	entrée	sortie
DCD	1	8	Data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

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

Connexion entre DTE et DCE :



Dialogue entre DTE et DCE :

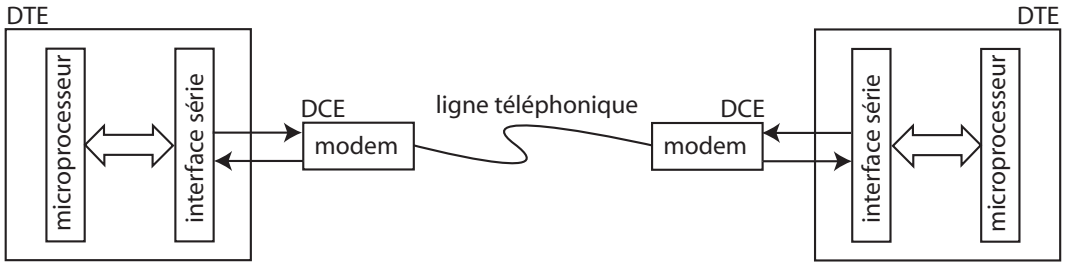


(les signaux de contrôle sont actifs à l'état bas = tension positive)

- 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 CTS : le DTE envoie la donnée sur la ligne TxD ;
- si le DCE veut demander une pause dans la transmission, il désactive 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.

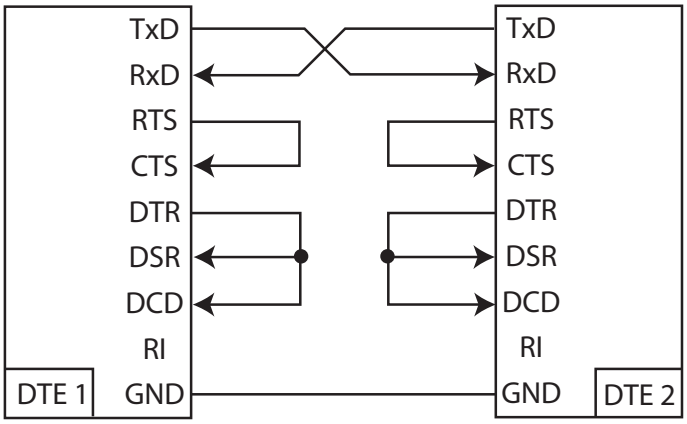
Applications des liaisons séries :

- transmission de données à travers une ligne téléphonique :



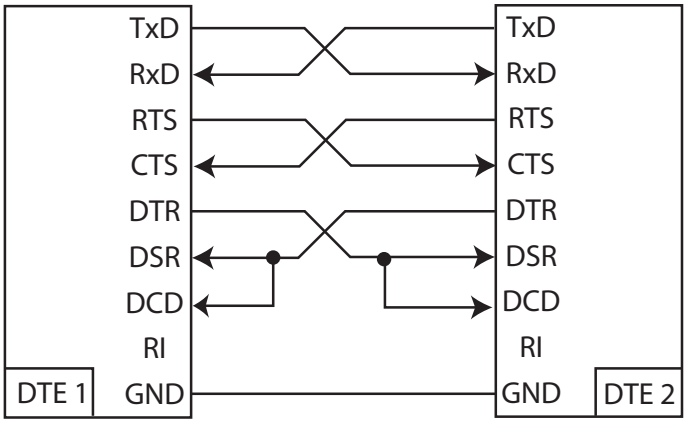
Le modem transforme les signaux numériques produits par l'interface série en signaux analogiques acceptés par la ligne téléphonique et inversement (modulations numériques FSK, PSK, ...)

- liaison série directe entre deux DTE :
  - liaison simple à 3 fils : rebouclage (strapping) des signaux de contrôle :



Ce câblage ne permet pas le contrôle matériel du flux entre les deux DTE.

  - liaison complète : câble Null Modem :



Ce câblage simule la présence d'un modem (DCE) en croisant les signaux de contrôle et permet le contrôle matériel du flux.

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

Brochage du 8250 :

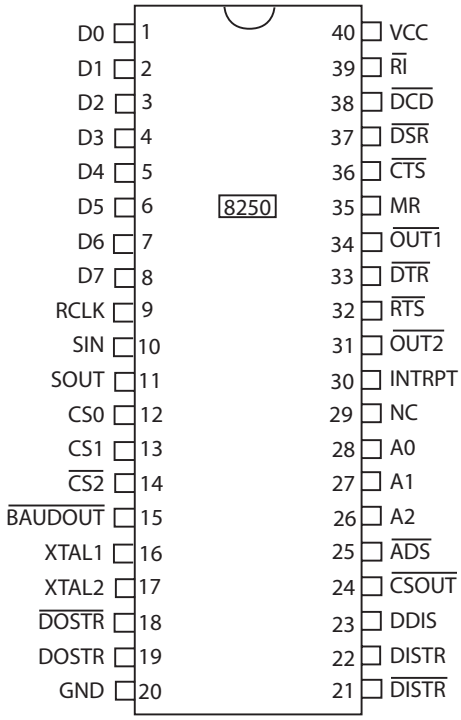
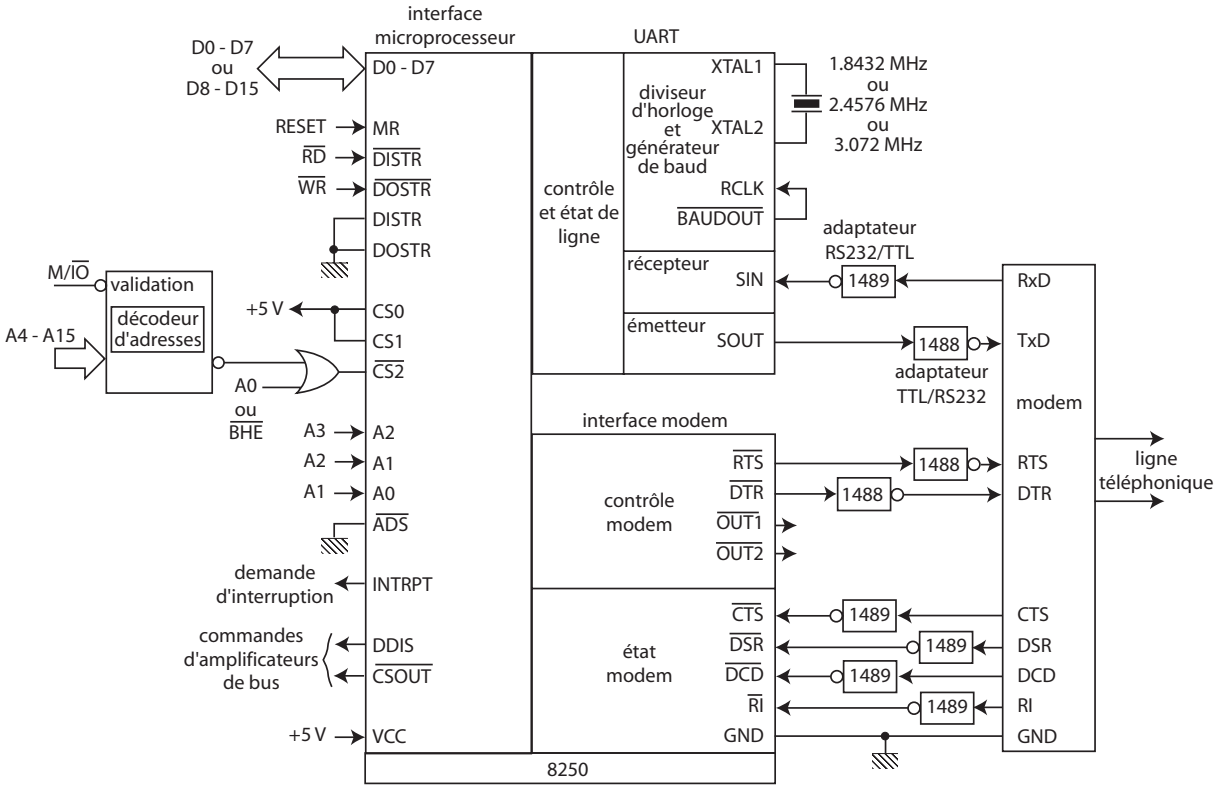


Schéma fonctionnel :



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	**DLL** : Divisor Latch LSB, octet de poids faible du diviseur d'horloge
1	0	0	1	**DLM** : Divisor Latch MSB, octet de poids fort du diviseur d'horloge
0	0	0	1	**IER** : Interrupt Enable Register, registre d'autorisation des interruptions
X	0	1	0	**IIR** : Interrupt Identification Register, registre d'identification des interruptions
X	0	1	1	**LCR** : Line Control Register, registre de contrôle de ligne
X	1	0	0	**MCR** : Modem Control Register, registre de contrôle modem
X	1	0	1	**LSR** : Line Status Register, registre d'état de la ligne
X	1	1	0	**MSR** : Modem Status Register, registre d'état du modem
X	1	1	1	**SCR** : Scratch Register, registre à usage général

En fonction de l'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,

bit 1	bit 0	
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 ;

bit 6 : contrôle de l'état de la ligne TxD, 0 → ligne en fonctionnement normal,
1 → forçage de TxD à l'état 0 (break) ;
bit 7 : 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)** :
  bit 0 : 1 → donnée reçue ;
  bit 1 : 1 → erreur d'écrasement de caractère ;
  bit 2 : 1 → erreur de parité ;
  bit 3 : 1 → erreur de cadrage (bit de stop non valide) ;
  bit 4 : 1 → détection d'un état logique 0 sur RxD pendant une durée supérieure à la durée d'un mot ;
  bit 5 : 1 → registre de transmission vide ;
  bit 6 : 1 → registre à décalage vide ;
  bit 7 : non utilisé, toujours à 0.
- **Modem Control Register (MCR)** :
  bit 0 : $\overline{\text{DTR}}$, bit 1 : $\overline{\text{RTS}}$, bit 2 : $\overline{\text{OUT1}}$, bit 3 : $\overline{\text{OUT2}}$ } activation (mise à 0) des lignes correspondantes en mettant à 1 ces bits ;
  bit 4 : 1 → fonctionnement en boucle : TxD connectée à RxD (mode test) ;
  bit 5, bit 6, bit 7 : inutilisés, toujours à 0.
- **Modem Status Register (MSR)** :
  bit 0 : 1 → changement de CTS depuis la dernière lecture : delta CTS ;
  bit 1 : 1 → delta DSR ;
  bit 2 : 1 → delta RI (uniquement front montant sur $\overline{\text{RI}}$) ;
  bit 3 : 1 → delta DCD ;
  bit 4 : $\overline{\text{CTS}}$, bit 5 : $\overline{\text{DSR}}$, bit 6 : $\overline{\text{RI}}$, bit 7 : $\overline{\text{DCD}}$ } ces bits indiquent l'état des lignes correspondantes.
- **Diviseur d'horloge (DLM,DLL)** : la vitesse de transmission est fixée par la valeur du diviseur d'horloge :

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

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

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

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

  bits 1 et 2 : source d'interruption,

bit 2	bit 1	
1	1	→ erreur
1	0	→ donnée reçue
0	1	→ registre d'émission vide
0	0	→ changement d'état modem

  (ordre de priorité décroissant) ;

  bit 3, bit 4, bit 5, bit 6, bit 7 : inutilisés, toujours à 0.
- **Interrupt Enable Register (IER)** : autorisation des interruptions
  bit 0 : 1 → donnée reçue ;
  bit 1 : 1 → registre d'émission vide ;
  bit 2 : 1 → erreur ;
  bit 3 : 1 → changement d'état modem ;
  bit 4, bit 5, bit 6, bit 7 : inutilisés, toujours à 0.
- **Scratch Register (SCR)** : registre à 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              ; adresses des registres du 8250
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               ; caractère End Of Text
ACK     equ   06H               ; caractère Acknowledge
NAK     equ   15H               ; caractère No Acknowledge
LIGNE   db    80 dup(?)         ; zone de rangement des caractères reçus
MSG     db    'Test 8250', EOT  ; message à envoyer

INIT    PROC  NEAR              ; procédure d'initialisation du 8250
        mov   dx,LCR            ; DLAB = 1 pour accéder au diviseur
        mov   al,80H            ; d'horloge
        out   dx,al
        mov   dx,DLL            ; vitesse de transmission = 2400 bit/s
        mov   al,48             ; => DLL = 48 ...
        out   dx,al
        mov   dx,DLM            ; ... et DLM = 0
        mov   al,0
        out   dx,al
        mov   dx,LCR            ; DLAB = 0 , 8 bits de données,
        mov   al,00011011B      ; parité paire, 1 bit de stop
        out   dx,al
        ret
INIT    ENDP

ENVOI_CARACTERE PROC NEAR       ; procédure d'émission du contenu de AH
        mov   dx,LSR            ; lecture du registre d'état de la ligne
attente_envoi :                 ; attente registre de transmission vide
        in    al,dx
        and   al,20H            ; masquage bit 5 de LSR
        jz    attente_envoi     ; si bit 5 de LSR = 0 => attente ...
        mov   dx,THR            ; ... sinon envoyer le caractère
        mov   al,ah             ; contenu dans le registre AH
        out   dx,al
        ret
ENVOI_CARACTERE ENDP
```

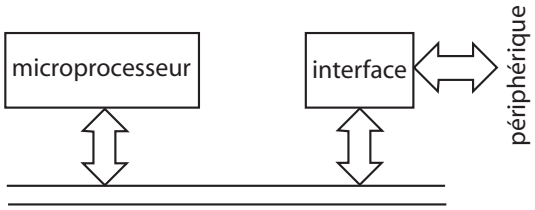
```
ENVOI   PROC  NEAR              ; procédure d'émission du message
        mov   si,offset MSG     ; pointer vers le début du message
boucle :mov   ah,[si]           ; AH <- caractère à envoyer
        cmp   AH,EOT            ; fin du message ?
        jz    fin_envoi         ; oui => fin procédure
        call  ENVOI_CARACTERE   ; non => envoyer caractère ...
        inc   si                ; ... et passer au caractère suivant
        jmp   boucle
fin_envoi :
        ret
ENVOI   ENDP

RECEPTION PROC NEAR             ; procédure de réception d'une ligne
        mov   di,offset LIGNE   ; pointer vers début zone de réception
        mov   cx,80             ; compteur de caractères reçus
attente_reception :
        mov   dx,LSR            ; lecture du registre d'état de la ligne
        in    al,dx
        test  al,01H            ; test de l'état du bit 0 de LSR
        jz    attente_reception ; pas de caractère reçu => attente
        test  al,00001110B      ; sinon test erreurs : bits 1,2,3 de LSR
        jz    suite             ; pas d'erreurs => continuer
        mov   ah,NAK            ; erreurs => envoyer NAK ...
        call  ENVOI_CARACTERE
        jmp   attente_reception ; ... et retourner attendre un caractère
suite : mov   dx,RBR            ; lire caractère reçu ...
        in    al,dx
        mov   [di],al           ; ... et le ranger dans LIGNE
        mov   ah,ACK            ; puis envoyer ACK
        call  ENVOI_CARACTERE
        dec   cx                ; décrémenter compteur de caractères
        jz    fin_reception     ; si compteur = 0 => fin réception
        inc   di                ; sinon incrémenter DI
        jmp   attente_reception ; et aller attendre caractère suivant
fin_reception :
        ret
RECEPTION ENDP
```

Chapitre 7

Les interruptions

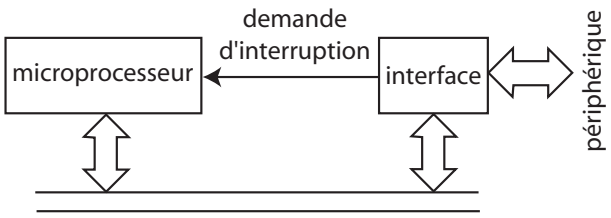
7.1 Définition d'une interruption

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 le signale au microprocesseur pour que celui-ci effectue la lecture de la donnée : c'est une **demande d'interruption** (IRQ : Interrupt Request) :



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

Exemples de périphériques utilisant les interruptions :

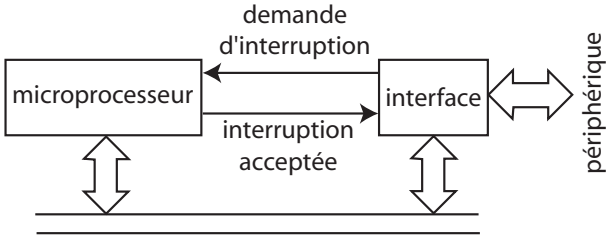
- clavier : demande d'interruption lorsqu'une touche est enfoncée ;
- port série : demande d'interruption lors de l'arrivée d'un caractère sur la ligne de transmission.

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

7.2 Prise en charge d'une interruption par le microprocesseur

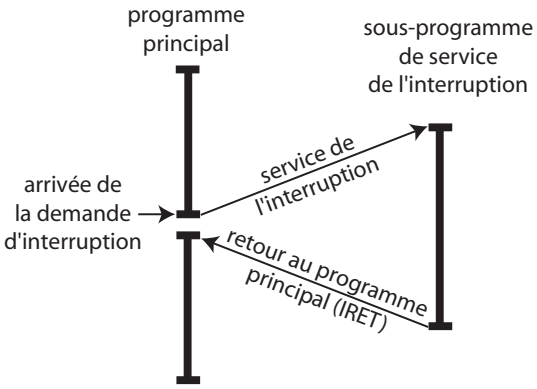
A la suite d'une demande d'interruption par un périphérique :

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



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

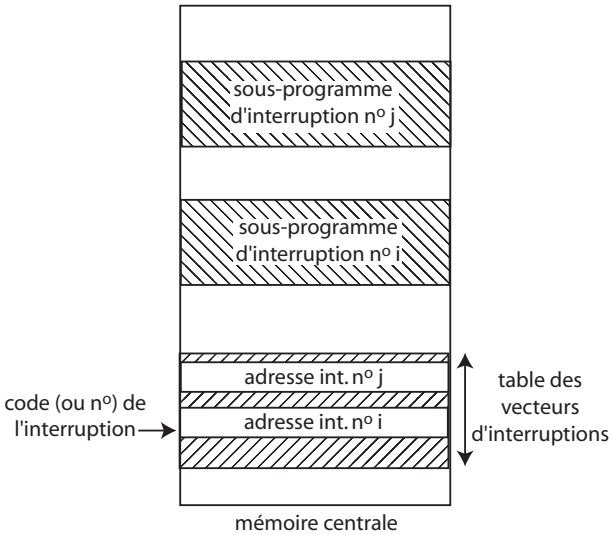
- il abandonne l'exécution du programme en cours et va exécuter un **sous-programme de service de l'interruption** (ISR : Interrupt Service Routine) ;
- après l'exécution de l'ISR, les registres sont restaurés à partir de la pile et le microproceseur reprend l'exécution du programme qu'il avait abandonné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



Remarque : la dernière instruction d'un sous-programme de service d'interruption doit être l'instruction `IRET` : retour d'interruption.
Si plusieurs interruptions peuvent se produire en même temps, on doit leur affecter une **priorité** pour que le microprocesseur sache dans quel ordre il doit servir chacune d'entre elle.

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** :



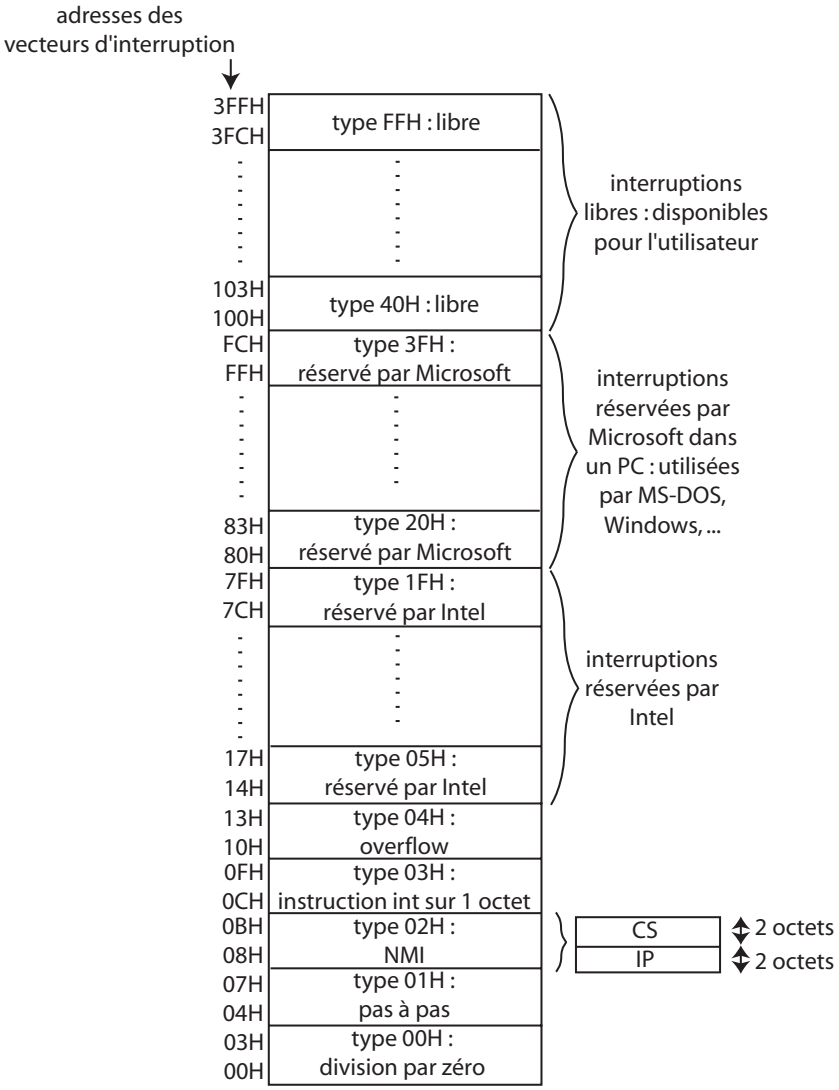
Lorsque les adresses des sous-programmes d'interruptions sont gérées de cette manière, on dit que les interruptions sont **vectorisées**.
Avantage de la vectorisation des interruptions : l'emplacement d'une ISR peut être n'importe où dans la mémoire, il suffit de spécifier le vecteur d'interruption correspondant.

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 `INT n`, où n est le type de l'interruption ;
- interruptions **processeur** générées par le microprocesseur en cas de dépassement, de division par zéro ou lors de l'exécution pas 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 :

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

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

La table des vecteurs d'interruptions est chargée par le programme principal (carte 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 péripériques utilisant des interruptions, on peut utiliser le contrôleur programmable d'interruptions 8259 dont le rôle est de :

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

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

Brochage du 8259 :

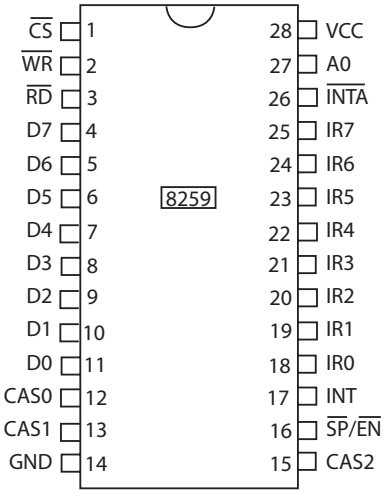
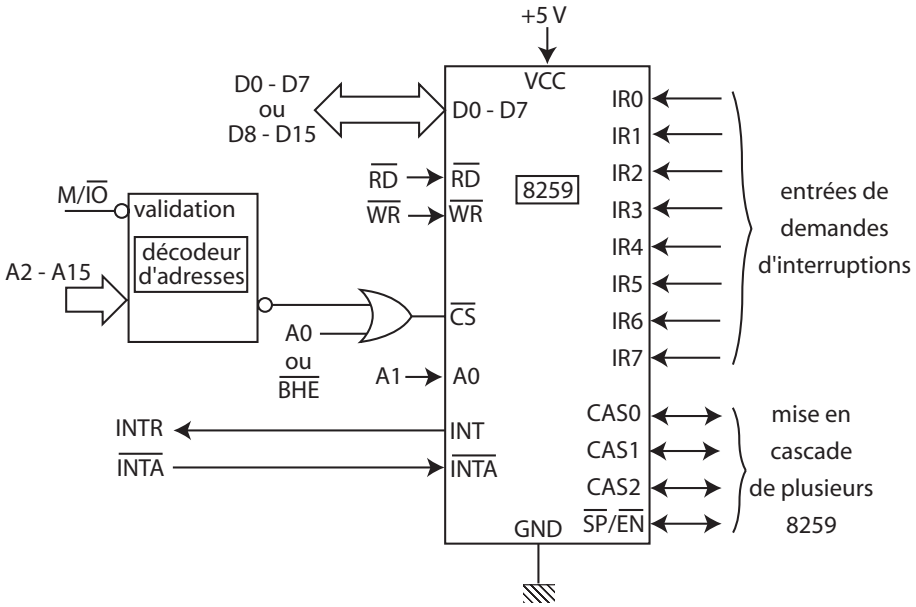


Schéma fonctionnel :



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