IUT Lyon A Informatique - 2gr4 année 1997

Projet de fin d'année

Communication série entre 2 PC par câble NULL-MODEM

Auteurs :

Brice Copy
Jean-Yves Dorelle
Jérôme Duraffourg
Arnaud Gaillard
Pierre Goiffon
Hatem Oueslati

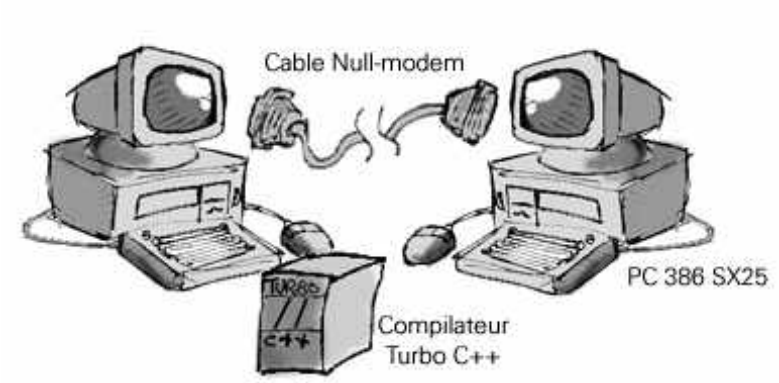
Table des matières

1. Introduction

1.1 But du projet

Le but de notre projet était de relier deux PC entre eux à l'aide d'une liaison Null-modem (un câblage permettant de simuler l'existence d'un modem entre deux PC et de communiquer par l'intermédiaire d'un port série), puis, une fois cette liaison établie, d'élaborer un protocole d'échange basique permettant la gestion des transferts de données.

1.2 Le matériel utilisé



Nous avons eu à notre disposition :

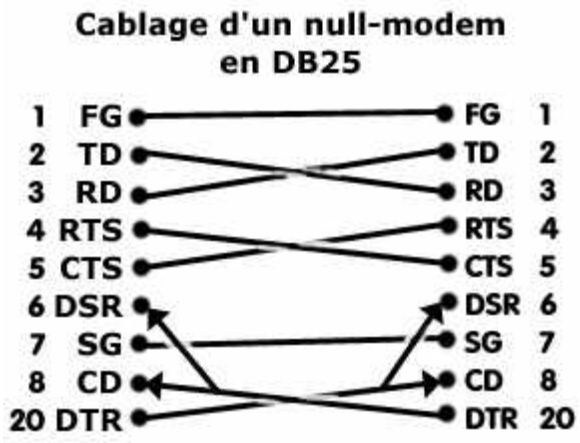
- Deux PC équipés chacun d'un processeur Intel 80386 cadencé à 25 Mhz
- Un compilateur Borland Turbo C++ 3 pour Windows, avec sa documentation complète
- Un câble Null-Modem DB25 Femelle-Femelle
- De la documentation sur la gestion des interruptions et la programmation de l'UART
- Un énoncé de TP comportant des sources non complètes d'un programme sommaire de transmission - nous nous sommes efforcés de faire que ce programme marche. En l'état actuel des choses, il compile, sans pour autant fonctionner.

2. L'interface série

Pour communiquer ou échanger des données, les ordinateurs emploient généralement le RTC (Réseau Téléphonique Commuté) via deux ETCD (Equipement Terminal de Communication de Données) : des modems le plus souvent. Dans ce cas, les données sont transférées de façon série, c'est à dire bit à bit. Cependant, dans le cas où les deux machines sont très proches (une dizaine de mètres au maximum), l'usage du RTC devient disproportionné, on préférera employer un câble simulant l'existence des deux modems - un câble "null-modem".

2.1 La liaison Null-modem

L'usage d'un modem est simulé en câblant simplement les broches d'émission d'un des deux ports série sur les broches réception de l'autre port série disponible. Ce câblage peut être schématisé de la manière suivante :



En face de chaque numéro de broche figure l'abrégé usuel de celle-ci. Voici la signification de ces abrégés :

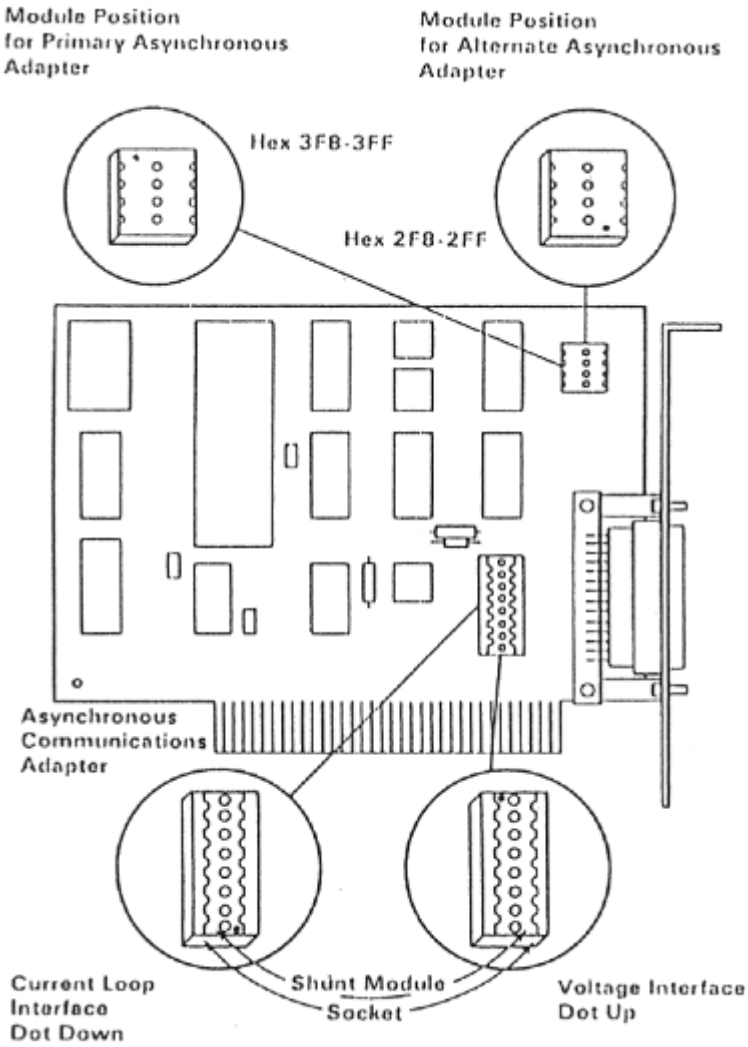
Abrégé	Signification détaillée
FG	Terre
TD	Emission des données
RD	Réception des données
RTS	Demande d'émission
CTS	Préparation d'émission
DSR	Données prêtes
SG	Masse des signaux (broche 9 sur DB9)
CD	Détection porteuse
DTR	Terminal prêt (broche 7 sur DB9)

Il nous suffit donc de brancher l'**émission de données TD** sur la **réception de données RD**, la **demande d'émission RTS** sur la **préparation d'émission CTS** et de rediriger le signal **terminal prêt DTR** sur la **détection de porteuse CD** et la broche **données prêtes DSR**.

Nous simulons ainsi simplement l'existence d'un modem entre deux PC, pour le coût d'un simple câblage. Il ne faudra tout de même pas, comme on le verra plus tard, être attaché aux performances !

2.2 La carte série

Les premiers PC ne permettaient pas ce type de liaison, il fallait leur ajouter une carte spécifique appelée "RS232". Maintenant, cette carte tend de plus en plus à être intégrée directement dans la carte mère. Cette carte est aussi adaptateur pour la transmission asynchrone de données. C'est par cette interface que seront réalisés les échanges de données. La figure suivante montre un exemple de carte RS-232 pour PC :



Outre les composantes physiques, les échanges de données nécessitent un logiciel approprié, qui prenne en compte le travail complexe de gestion de la carte RS232. Le BIOS remplit ce rôle en fournissant quatre fonctions, expliquées plus bas, gérées à travers l'interruption 14h. Mais pour répondre à certains besoins, l'utilisateur doit écrire des fonctions qui s'adressent directement à la carte. Il faut savoir que la carte RS232 est gérée par un processeur appelé **UART**, dont nous allons détailler le fonctionnement, et les moyens de le programmer.

3. L'UART

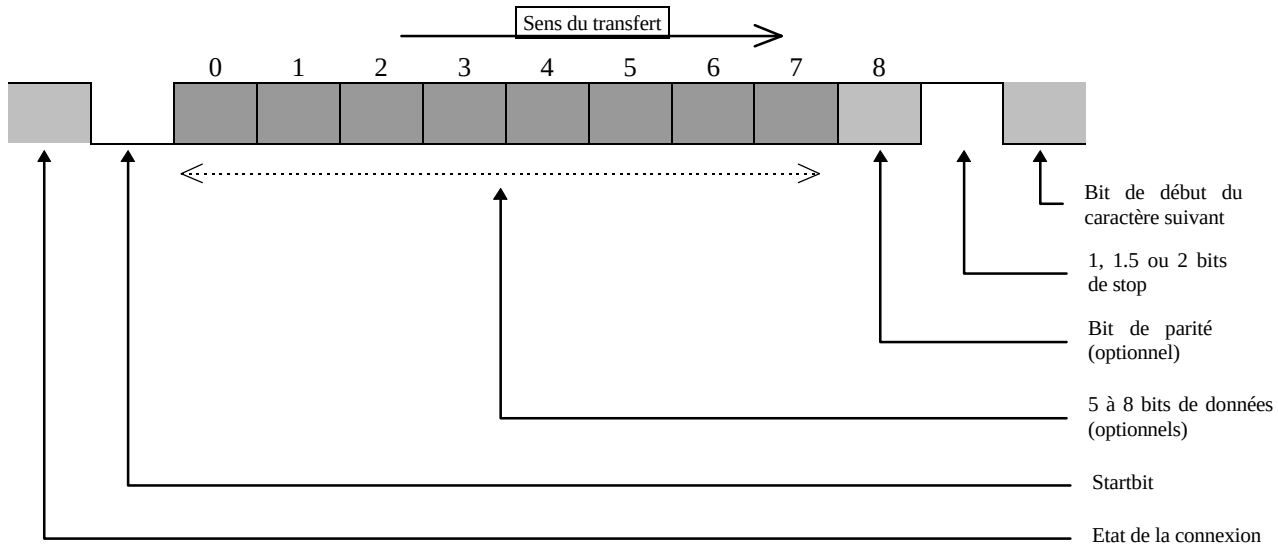
"UART" signifie "Universal Asynchronous Reciever Transmitter", en français Emetteur-Récepteur Asynchrone Universel. Cette appellation désigne une puce très importante sur la carte RS-232.

En effet, Ce processeur est le cerveau d'une carte RS232. Grâce à ses registres, on peut contrôler totalement une liaison série, sans avoir à passer par les quatre fonctions du BIOS, explicitées en fin de chapitre. Nous nous limiterons dans ce dossier à l'UART de modèle INS8250, qui n'est pas la puce la plus moderne et la plus performante, mais qui est compatible avec les nouvelles UART 16450, 16550 et 16650.

3.1 Définitions

Avant d'étudier plus avant le fonctionnement des registres et le moyen de contrôler l'UART, il est nécessaire de définir quelques éléments :

3.1.1 Longueur de mot



Comme le montre la figure, le protocole de base employé pour la transmission de données ne prévoit que deux états significatifs : 0 (ou low), et 1 (ou high). Quand on ne transmet rien, la ligne est à l'état *high*. Le "Startbit", d'état *low*, permettra au récepteur de savoir que des données vont être transmises. On pourra régler l'UART pour envoyer de 5 à 8 bits de données : ainsi ne parlera-t-on pas d'octets transmis, mais de **caractères transmis**.

On peut émettre deux remarques d'ordre pratique :

- **Les quatre fonctions du BIOS ne permettent que l'emploi de 7 ou 8 bits de données...**
- **Ce sera toujours le bit de poids le plus faible de chaque caractère qui sera envoyé en premier, le bit de plus fort poids en dernier.**

3.1.2 Parité

Le bit de parité est optionnel. Il permet une meilleure qualité de la transmission, car il permet de détecter certaines erreurs. Le bit de parité est un complément : si c'est une parité paire on le complétera afin que le nombre de "1" dans le mot de données soit pair, et inversement si c'est une parité impaire, il faudra que le nombre de "1" dans le mot de données soit impair. Les deux méthodes - parité paire et impaire - garantissent de toute façon la même sécurité.

3.1.3 Bits de stop

Ces bits (1, 1.5 ou 2 bits, à définir) signalent la fin du mot de données - ils valent toujours 1. Attribuer 1.5 bits de stop signifiera simplement que la valeur *high* (puisque le bit de stop est toujours à l'état 1) sera transmise sur la ligne pendant 1.5 quantum de temps. Ce quantum de temps, correspondant à l'émission d'un bit, a une valeur de 1/300 secondes ; cela vient des anciens standards qui privilégiaient une vitesse fixée à 300 bauds (avec un bit par symbole, donc 300 bps). Plus le nombre de bits de stop est grand, plus la transmission est sûre, mais plus le caractère suivant met du temps à être envoyé...

3.1.4 Les signaux du modem

L'UART est conçue pour gérer les signaux en provenance d'un modem. Aussi est-il nécessaire de donner quelles broches sont utilisées par l'ETCD.

Nom générique	Signification
DCD	Détection de porteuse
RD	Réception de données
TD	Envoi de données
DTR	Equipement terminal prêt
GND	Masse
DSR	Equipement de communication prêt
RTS	Demande à émettre
CTS	Prêt à émettre
RI	Indicateur d'appel (Ring Indicator)

3.2 Les registres de contrôle de protocole

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

3.2.1 LCR - Line Control Register (registre de contrôle de la ligne)

7	6	5	4	3	2	1 0
DLAB	BCB	SPB	EPS	PEN	STB	WLS0-WLS1

- **Word Length Select (WLS0 et WLS1) :** spécifie le nombre de bits de données

Bit 1	Bit 0	Nb bits de données
0	0	5
0	1	6
1	0	7
1	1	8

- **Stop Transmission Bit (STB) :**
  0 ➔ 1 bit de de stop (ou bit d'arrêt)
  1 ➔ 1.5 bit de de stop (si 5 bits de données), 2 bits de stop sinon
- **Parity Enable (PEN) :**
  0 ➔ pas de contrôle de parité
  1 ➔ contrôle de parité activé
- **Even Parity Select (EPS) :**
  0 ➔ parité impaire
  1 ➔ parité paire
- **Stick Parity Bit (SPB)**
- **Break Control Bit (BCB) :** quant ce bit est à l'état logique 1, la sortie série est forcée à 0 (permet au processeur d'alerter un terminal)
- **Divisor Latch Access Bit (DLAB) :**
  0 ➔ ce bit doit être à cette valeur pour qu'on puisse accéder aux registres d'émission THR et de réception RBR, et à celui d'autorisation d'interruption IER
  1 ➔ règle l'UART à la vitesse de transmission indiquée dans le registre DLR

3.2.2 DLR - Divisor Latch Register (registre de sélection de la vitesse de transfert)

Notons tout d'abord que ce registre est codé sur 16 bits : il aura donc deux adresses. Pour pouvoir utiliser ce registre, qui permet de sélectionner facilement une vitesse de transfert de 50 à 9600 bauds (c'est à dire un maximum de 19200 bps), il ne faut pas oublier auparavant de **forcer le bit DLAB du registre LCR à l'état 1**. La vitesse n'est pas programmée directement dans l'UART - c'est un circuit comparable à une horloge (baud rate generator) qui se charge de réguler celle-ci.

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

EXEMPLE : pour régler l'UART à une vitesse de 9600 bauds, il faudra copier 0x0C vers DLR (se référer au chapitre 5, "bases de programmation en C").

3.3 Les registres de contrôle de la ligne

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

3.3.1 LSR - Line Status Register (registre d'état de la ligne)

7	6	5	4	3	2	1	0
0	TSR	THRE	BE	FE	PE	OR	DR

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

3.3.2 MSR - Modem Status Register (registre d'état du modem)

7	6	5	4	3	2	1	0
RLSD	RI	DSR	CTS	DRLSD	TERI	DDSR	DCTS

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

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

3.3.3 MCR - Modem Control Register (registre de contrôle du modem)

7	6	5	4	3	2	1	0
0	0	0	Loop	Out2	Out1	RTS	DTR

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

EXEMPLE : Si l'on force à 1 le bit DTR, la broche DTR prendra la valeur 0.

3.4 Les registres de contrôle d'interruptions

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

3.4.1 IER - Interrupt Enable Register (registre d'autorisations d'interruptions)

7	6	5	4	3	2	1	0
0	0	0	0	EMSI	ERLSI	ETHREI	EDAI

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

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

3.4.2 IIR - Interrupt Identification Register (registre de cause d'interruption)

7	6	5	4	3	2	1	0
0	0	0	0	0	IID2	IID1	IID0

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

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

3.5 Les registres de transfert

3.5.1 Les registres d'émission

Lorsqu'un caractère doit être transmis, il doit d'abord être transféré dans le registre **THR** (Transmission Holding Register = registre d'attente de l'émetteur). Il y restera tant que le caractère précédent ne sera pas acquitté par la machine distante. Une fois l'acquittement reçu, le caractère sera transféré dans un autre registre : **TSR** (Transmission Shift Register = Registre de décalage de l'émetteur). L'UART se chargera alors de transmettre le caractère bit à bit sur la ligne, et il pourra y insérer, selon les réglages effectué dans LCR (contrôle de ligne), un bit de parité et un nombre fixé de bits de stop.

Il est important de noter ces quelques remarques :

- ce registre est à **écriture seule** (on le verra plus bas, il possède la même adresse que RBR, le registre de réception : un ordre de lecture à l'adresse de THR renverra donc le contenu de RBR)
- le bit DLAB du registre LCR (contrôle de la ligne) doit être forcé à 0 pour pouvoir effectuer une écriture dans ce registre.
- ce registre est sur 8 bits. On pourra y envoyer de 5 à 8 bits, suivant les réglages du registre de contrôle de ligne LCR (bits WLS), sachant que ce sera le bit 0 (poids le plus faible) qui sera le premier transmis.
- il sera judicieux d'effectuer une lecture de LSR (registre d'état de la ligne) - bits 5 et 6 - pour savoir quand le registre THR devient libre, et quand le caractère présent dans TDR a été transmit totalement sur la ligne - se reporter à la partie traitant de ce registre.

3.5.2 Les registres de réception

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

Quelques remarques importantes :

- le registre RBR est à **lecture seule**. Comme cité plus haut, les registres d'émission THR et de réception RBR possèdent une adresse commune. Ainsi une écriture sur l'adresse de RBR provoquera un remplissage du registre THR, et n'aura aucun effet sur le registre de réception.
- le bit DLAB du registre LCR (contrôle de la ligne) doit être forcé à 0 pour pouvoir effectuer une lecture dans ce registre.
- ce registre est sur 8 bits : y seront copiés les 5 à 8 bits de données, sans bits de contrôle. Le bit 0, de poids le plus faible, est le premier a avoir été reçu.
- il sera judicieux d'effectuer une lecture de LSR (registre d'état de la ligne) - bits 0 à 4 - pour détecter les éventuelles erreurs de transmission, ou tout simplement pour savoir si un caractère a été reçu - se reporter à la partie traitant de ce registre.

3.6 Synthèse et Compléments sur les registres

Voici un petit tableau rappelant les différents registres (dénomination, fonction), les restrictions d'accès ("lecture" signale une MAJ automatique), et aussi pour quatre d'entre eux, l'état du bit DLAB (registre LCR de contrôle de ligne) qui permet de les utiliser.

Dénomination	Etat de DLAB (registre LCR)	Restrictions d'accès	Adresse COM1	Adresse COM2
Contrôle de protocole				
LCR (contrôle de ligne)			3FB	2FB
DLR (sélection vitesse de transfert - 16 bits)	1		3F8 3F9	2F8 2F9
Contrôle de ligne				
LSR (état de ligne)		lecture	3FD	2FD
MSR (état modem)		lecture	3FE	2FE
MCR (contrôle modem)			3FC	2FC
Contrôle des interruptions				
IER (autorisations d'interruptions)	0		3F9	2F9
IIR (cause d'interruption)		lecture	3FA	2FA
Registres de transfert				
THR (émission) [+ TSR (transit)]	0	écriture seule	3F8 -	2F8 -
RBR (réception) [+ RDR (transit)]	0	lecture seule	3F8 -	2F8 -

On notera que les deux registres TSR et RDR ne sont cités qu'à titre informel. Il n'ont en effet pas vocation à être utilisés ; il est toutefois utile de les connaître pour gérer les erreurs de transmission.

3.7 Les fonctions de gestion du BIOS

Le BIOS fournit quatre fonctions de gestion, que l'on peut appeler via l'interruption 14h. Ces fonctions ont pour avantage... la simplicité !

3.7.1 Fonction 0 : réglage du protocole

Par cette fonction, on décrit le nombre de bits de stop, l'existence ou non de contrôle de parité, la vitesse de transmission. Elle sera appelée par l'interruption 14h, fonction 00h, avec 0 dans le registre AH et les valeurs définissant le protocole dans AL :

7	6	5	4	3	2	1	0
Vitesse de transmission 000 ⇔ 110 bauds 001 ⇔ 150 bauds 011 ⇔ 600 bauds 100 ⇔ 1200 bauds 101 ⇔ 2400 bauds 110 ⇔ 4800 bauds 111 ⇔ 9600 bauds			Test de parité 00 ⇔ aucun 01 ⇔ impaire 10 ⇔ paire		Nb bits de stop 0 ⇔ 1 1 ⇔ 2 ou 1.5	Longueur de données 10 ⇔ 7 bits 11 ⇔ 8 bits	

3.7.2 Fonction 1 : envoi de caractère

C'est la fonction 01h. Lors de son appel, le registre AH doit contenir 01h et le registre AL le caractère à transmettre. La transmission réussie du caractère sera signalée par le passage à 0 du bit 7 du registre AH - un 1 dans le même bit signale une erreur de transmission. Les autres bits de ce registre indiquent quant à eux l'état du canal.

3.7.3 Fonction 2 : Réception de caractère

Cette fonction est identifiée 02h. Si un caractère a été reçu, le registre AL contiendra le caractère reçu après appel de cette fonction. Si AH contient la valeur 0, c'est qu'aucune erreur n'est à signaler, sinon il contient la valeur de l'état du canal.

3.7.4 Fonction 3 : état de la ligne/modem

Cette fonction est dénommée 03h. Elle permet de connaître l'état du canal, mais aussi celui du modem : ils sont renvoyés respectivement dans les registres AH et AL. Ces résultats sont en fait une copie des registres LSR et MSR de l'UART, détaillés plus haut - les informations sont donc codées de la même manière dans les deux cas.

3.8 Choix entre programmation de l'UART par registres et fonctions du BIOS

Il est évident que la programmation bas niveaux, directement sur les registres de l'UART, permet une souplesse immense, que ne procure pas l'emploi des quatre fonctions du BIOS. Ainsi avons-nous travaillé sur la première option, à travers le programme non achevé que nous devions compléter.

Dans tous les cas, la première solution permet une liberté de programmation presque totale. En conséquence, il faut construire les fonctions qui permettront de gérer le protocole, qu'on aura pris soin d'établir. Mais, après quelques efforts, il est possible d'écrire une bibliothèque qui gère un, voire des protocoles plus ou moins complexes : contrôle de flux, etc...

La programmation par les fonctions du BIOS est quant à elle très simple : elle ne nécessite pas de se documenter sur le fonctionnement précis de l'UART. Mais les quatre fonctions, bien évidemment, sont très limitées. Elles conviennent toutefois très bien à l'écriture de programmes simples de transmission entre deux PC, sur de courtes distances (par câble Null-modem : quasiment aucune erreur de transmission), comme un programme de dialogue. Mais dès que l'on s'intéresse aux applications qui nécessitent une utilisation intensive de la ligne - transfert de fichiers par exemple -, il devient nécessaire de définir un protocole sûr, permettant une gestion plus souple par le programme principal des erreurs (le protocole définissant quoi faire en cas d'erreur).

4. Les interruptions

4.1 Principes et raisons du choix de la programmation par interruptions

Dans le cas qui nous intéresse, la transmission d'informations entre deux PC via l'interface série, nous sommes amenés dans les programmes à être attentifs aux changements d'état de différentes choses (modem, ligne, tampons de réception/émission, ...). Pour surveiller ces valeurs et leur changements, nous avons à notre dispositions deux techniques : la scrutation, et la programmation par interruptions.

4.1.1 La scrutation (polling)

Cette méthode consiste à interroger périodiquement un périphérique. Elle s'appuie donc sur des **boucles**, dont on sort dès que la valeur lue à l'instant *t* est différente de la valeur lue à l'instant *t-1*. Cette méthode a pour inconvénient majeur son fondement : les boucles. Celles-ci, outre de nécessiter une écriture particulière des programmes, prennent un temps de traitement sur le microprocesseur non négligeable... C'est donc une solution qui présente un premier abord simple, mais qui ne peut en fait être implémentée que sur des programmes d'une extrême simplicité.

4.1.2 La programmation par interruptions

L'utilisation de cette technique apporte une souplesse incomparable au programmeur. Celui-ci n'a qu'à définir un "**gestionnaire d'interruption**", une fonction particulière, puis de l'affecter à une certain événement matériel, et enfin de déclarer l'interruption à un circuit programmable. Dès que l'événement choisi se produira, le processeur suspendra son activité, pour déclencher le gestionnaire d'interruption... Une fois le traitement effectué, il reprendra son activité dans l'état où il l'avait laissé avant de lancer le gestionnaire. Cette méthode à l'avantage d'être transparente, et de ne nécessiter que très peu de ressources : les interruptions sont gérées par des circuits spécialisés - les contrôleurs d'interruptions, tel le PIC sur PC. Vous pourrez retrouver des détails sur l'installation de gestionnaire d'interruptions à la fin de ce chapitre, à la suite de la description du fonctionnement du PIC.

On privilégiera bien sûr cette technique - l'écriture du programme s'en trouve en effet extrêmement simplifiée ! Le programme principal n'aura pas à se préoccuper des erreurs qui peuvent survenir, celles-ci seront traitées par le ou les gestionnaire(s)... qu'il faudra par contre prendre grand soin à programmer ! Il faudra aussi faire attention aux priorités à attribuer, et aux filtres à appliquer - tant au niveau de l'UART (registre IER, déjà vu dans le chapitre sur l'UART) que du PIC.

4.2 Le contrôleur d'interruption (PIC)

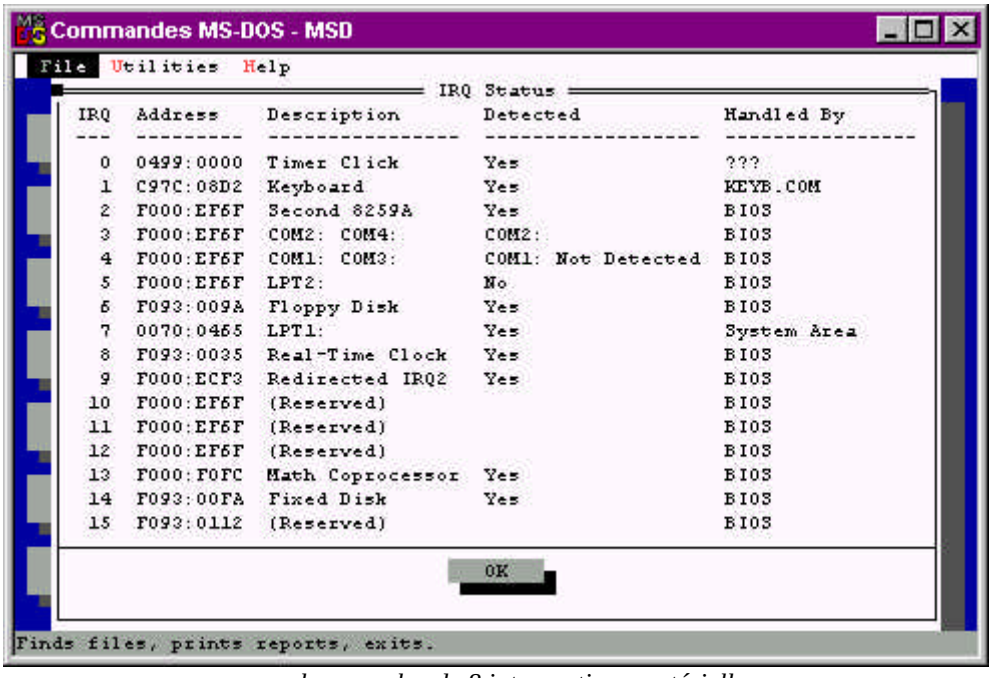
En 1985, INTEL proposa un composant portant le nom de PIC (Programmable Interrupt Controller - ou contrôleur d'interruption - 8259A) dont le but est de prendre en considération les demandes d'interruption émanant de l'électronique, et de les transmettre au CPU pour qu'il active le gestionnaire d'interruption correspondant dans la mémoire RAM. Cependant, ceci est très complexe car le PIC doit gérer simultanément de nombreuses interruptions matérielles en provenance du clavier, des disques durs, de l'horloge, ... périphériques qui nécessitent d'être utilisés en même temps alors que le processeur ne peut exécuter qu'une seule tâche à un instant donné (concept du pseudo-parallélisme sur un système monoprocesseur). Le contrôleur d'interruption aura donc pour tâches :

- de "sérialiser" les appels d'interruption pour les acheminer les uns après les autres vers le processeur
- de suivre un ordre précis dans l'exécution des gestionnaires d'interruptions, fonction de la priorité de l'interruption correspondante

Le PIC gère huit interruptions numérotées de 0 à 8. L'IRQ (interruption) 0 possède la plus forte priorité et l'IRQ 7 la plus faible. On affecte donc les IRQ (et la priorité correspondante) en fonction des besoins.

Interruption	Périphérique
IRQ 0	horloge
IRQ 1	clavier
IRQ 2	*libre*
IRQ 3	COM 2
IRQ 4	COM 1
IRQ 5	disque dur
IRQ 6	lecteur disquettes
IRQ 7	imprimante

exemple d'interruptions matérielles sur un PC

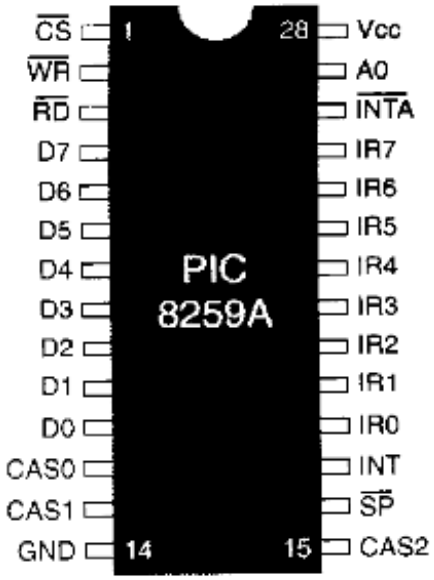


exemple avec plus de 8 interruptions matérielles

Si on est amené à utiliser plus de huit sources d'interruption différentes, on active tout simplement plusieurs PIC. Sur les PC actuels, deux PIC offrent leurs services de façon à obtenir 16 interruptions matérielles différentes. Dans le cas de l'activation simultanée de plusieurs PIC, l'un d'eux agit en tant que maître, les autres en tant qu'esclaves. Seul de maître est relié au CPU, alors que les esclaves transmettent les demandes d'interruption via leur maître - ce concept sera détaillé à la suite de la description du circuit PIC.

4.3 Le PIC 8259A

4.3.1 Description technique du circuit PIC



PIC 8259A d'INTEL et ses 28 lignes de contrôle

Ligne	Broche	Nom	Fonction
CS	1	Chip-select	Requis par les signaux RD et WR pour communiquer entre le PIC et CPU via le bus de données
WR	2	Write	Règle ce signal sur low, le CPU peut écrire des données dans le registre interne via le bus de données
RD	3	Read	Règle ce signal sur low, le CPU peut lire des données depuis le registre interne via le bus de données
D0 à D7	4-11	Bus de données	Ces lignes permettent de transférer des octets entre PIC et CPU
CAS0 à CAS2	12,13,15	Cascade	Un PIC maître communique avec ses esclaves à travers ces lignes
GND	14	Ground	Masse
SP	16	Slave Program	Indique si le PIC agit en tant que maître ou esclave
INT	17	Interrupt	Cette ligne est associée au CPU avec l'entrée INTR. Elle indique au CPU le moment où il faut déclencher une interruption électronique
IR0 à IR7	18-25	Interrupt-Request	Chacune de ces lignes est liée à une source d'interruption pouvant adresser une interruption via sa ligne. IR0 est associée à IRQ 0 et possède la plus forte priorité, IR7 la plus faible
INTA	26	Interrupt Acknowledge	A travers cette ligne, le CPU montre qu'il est prêt à déclencher une interruption demandée via sa ligne INT. A l'étape suivante, le CPU se sert de cette ligne pour signaler qu'il attend le numéro d'interruption
A0	27	Adress	Utilisé lors des accès en lecture et écriture pour indiquer le numéro du registre à lire ou à écrire
Vcc	28	Alim	Alimentation : +3,3 V ou +5V selon le système

4.3.2 Déroulement d'un appel d'interruption

Le déroulement d'un appel d'interruption se fait à l'aide des ports du PIC. Chaque source d'interruption est associée à l'une des lignes IR0 à IR7. Dès qu'un périphérique déclenche une interruption, il émet de ce fait un signal (une impulsion électrique) sur le port correspondant (IR0 à IR7). Le PIC envoie alors sa demande d'interruption au CPU en tenant compte des priorités et en vérifiant si une réponse n'a pas déjà été donnée à une autre interruption. Si tel est le cas, il émet un signal sur la ligne INT qui est reliée au port INTR du CPU et qui a pour but de lui indiquer si une interruption doit être déclenchée.

Dès que le CPU est de nouveau prêt à recevoir une interruption matérielle et à appeler le gestionnaire correspondant, il en informe le PIC via la ligne INTA. Lorsque le CPU est prêt à recevoir le numéro de l'interruption à déclencher, il informe le PIC par une impulsion sur la ligne INTA. Le PIC émet alors le numéro d'IRQ sur les lignes D0 à D7 qui jouent temporairement le rôle de bus de données pour transmettre des octets entre le CPU et le PIC. Le CPU peut alors appeler le gestionnaire d'interruption adéquate en lisant son adresse dans la table des vecteurs d'interruption, le numéro du PIC étant alors utilisé comme index dans cette table. La communication "électronique" est terminée - mais il faut encore que le gestionnaire d'interruption rentre en contact avec le PIC, dès lors qu'il aura terminé son travail et qu'il aura rendu le contrôle au CPU. Le but de cette liaison est d'informer le PIC que l'appel d'interruption s'est achevé et qu'il peut donc autoriser d'autres appels de l'interruption concernée.

4.4 Les Registres internes du PIC et l'ordre des priorités

4.4.1 Les registres du PIC

Le PIC est doté de trois registres permettant ainsi de connaître l'interruption en cours et celles en attente. Le registre de requête d'interruption (IRR = Interrupt Request Register), le registre en service (ISR = In Service Register) et le registre de masque (IMR = Interrupt Mask Register). Ces trois registres sur 8 bits correspondent à l'une des 8 interruptions matérielles.

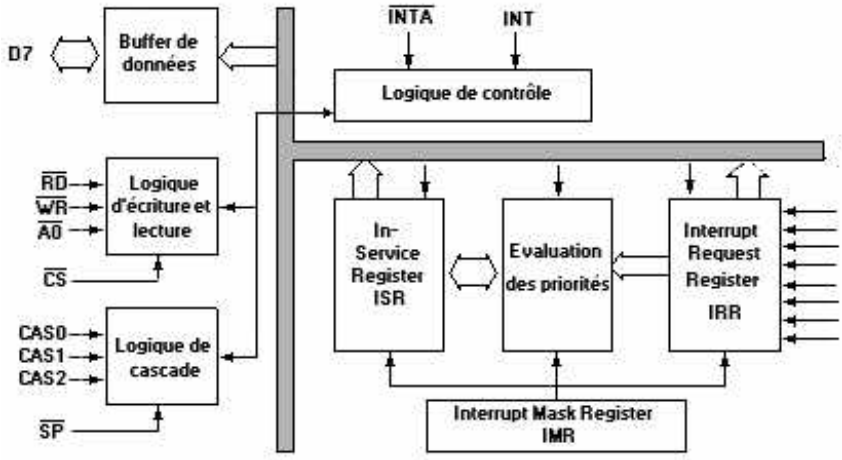


Schéma du PIC avec ses registres internes IRR, ISR et IMR

4.4.2 Le registre IRR (état des interruptions)

L'IRR est directement lié aux huit lignes d'interruptions IR0 à IR7. Dès qu'une source d'interruption annonce sa volonté, le bit correspondant est mis à 1 dans l'IRR

7	6	5	4	3	2	1	0
IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0

Registre IRR

Cependant, le bit concerné ne doit pas être actif dans l'IMR car c'est dans ce registre que sont désactivées les différentes interruptions... Si tel n'est pas le cas, le PIC doit prendre l'initiative d'envoyer ou non l'interruption au CPU. Sachant que d'autres interruptions risquent d'ores et déjà d'attendre qu'elles soient appelées, le PIC décide d'extraire tout d'abord le bit de poids faible du registre IRR. Il compare ce dernier avec le registre ISR qui renferme la liste des interruptions en cours.

4.4.3 Priorité des interruptions

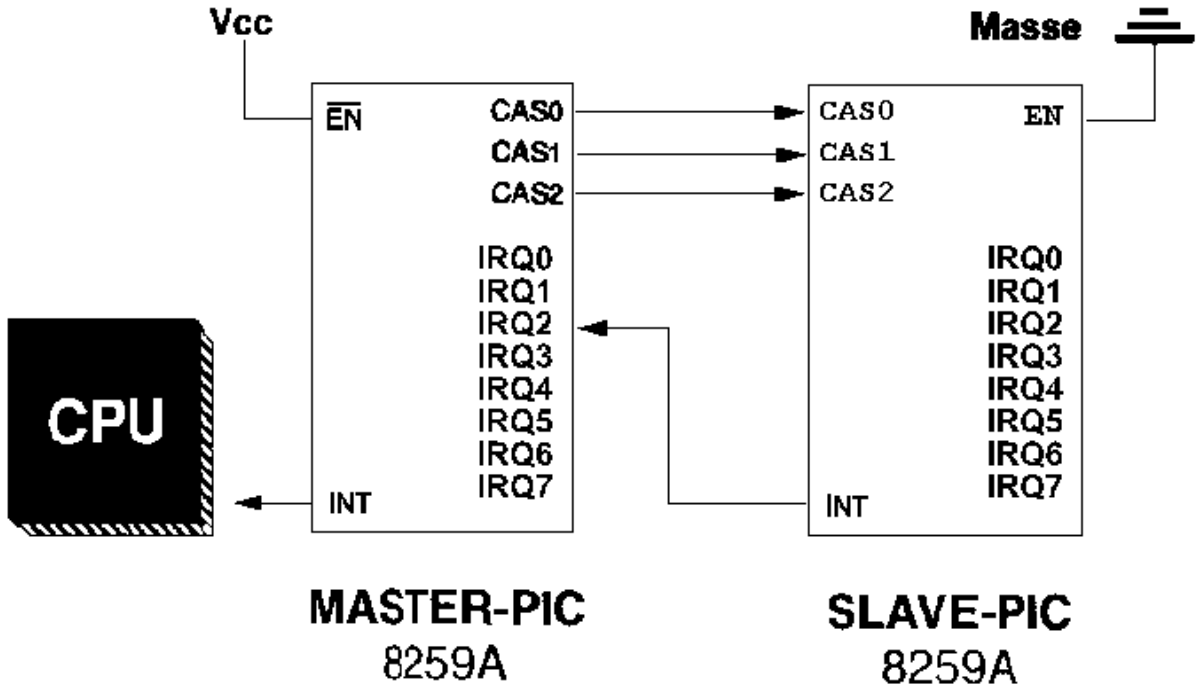
Si une interruption de forte priorité est en train d'être exécutée, le PIC renvoie le nouvel appel d'interruption tant que les interruptions ayant une forte priorité ne sont pas traitées. S'il arrive entre-temps une interruption de plus faible priorité, ou si aucune interruption n'est reçue, l'interruption est alors envoyée au CPU. Il se peut donc qu'une interruption de forte priorité provoque la suspension d'une interruption en cours d'exécution (ou de son gestionnaire d'interruption). Pour ce faire, le flag d'interruption du processeur doit être actif.

Normalement, l'interruption ayant une plus faible priorité ne subit pas de suspension, car le CPU efface automatiquement le flag d'interruption lors de l'appel. Le gestionnaire d'IRQ doit donc être protégé contre une éventuelle annulation provoquée par une interruption de plus forte priorité. Cependant, si un gestionnaire autorise explicitement cette action, il peut définir le flag d'interruption du processeur grâce à la commande machine STI et libérer le CPU pour qu'il puisse recevoir d'autres appels. Le flag sera systématiquement réactivé avec la commande machine IRET lors de la fin du gestionnaire correspondant, dans le but de permettre de nouveaux appels.

Dès lors que le PIC a découvert la prochaine interruption à exécuter et qu'il a reçu le feu vert de la part du CPU pour la déclencher, celui-ci active le bit correspondant dans le registre ISR et efface simultanément ce dernier dans le registre IRR. D'une part, il sait que l'interruption concernée est en train de s'exécuter et d'autre part, il n'essaye pas de déclencher cette même interruption une deuxième fois.

4.5 Une architecture en cascade

Sur un PC, deux PIC sont utilisés pour créer ainsi 16 interruptions. Pour cela, deux PIC, le maître et l'esclave, sont reliés entre eux . La ligne EN permet à chaque PIC de savoir s'il doit agir en tant que maître ou esclave. Chaque esclave est attaché au maître à travers quatre lignes : CAS0, CAS1, CAS2 et INT (qui dépend de la ligne IRx du maître à travers laquelle s'effectue la cascade : sur un PC, il s'agit de la ligne IR2 qui correspond à l'IRQ2).



Tant que le maître ne reçoit pas de demande d'interruption sur la ligne en cascade, tout se passe normalement. Lorsque l'esclave reçoit sur l'une de ses lignes IRx le signal de déclencher une interruption, il vérifie son registre interne pour savoir s'il peut satisfaire la demande. Si oui, il émet un signal sur la ligne INT qui est reçu par le maître et non par le CPU. Le PIC maître traite l'interruption comme si elle émanait d'un composant électronique, et après modification du contenu des registres, il transmet l'appel d'IRQ au CPU par sa ligne INT.

4.6 Mise en place d'un gestionnaire d'interruption

On l'a vu, un contrôleur d'interruption est activé entre le CPU et les lignes d'interruption des périphériques, afin de séparer davantage le CPU de l'électronique. Ce contrôleur est le PIC (Programmable Interrupt Controller) sur PC. Ce circuit dévolu aux compatibles est programmable. Le BIOS programme au démarrage certaines interruptions qui deviendront réservées, comme indiqué dans les schémas de la page 17. Ce paragraphe explicite le principe à suivre pour programmer ses propres interruptions.

4.6.1 Programmer une interruption

Pour programmer une interruption, il faut associer la fonction qui va être exécutée lors de son déclenchement (le gestionnaire d'interruption) à un vecteur d'interruption.

Rappel : Qu'est ce qu'une interruption ?

> Une interruption est un mécanisme permettant à un périphérique d'interrompre brièvement le processeur dans l'exécution courante, pour l'obliger à se brancher sur un sous-programme, appelé gestionnaire d'interruption. Ces interruptions constituent aussi le

moyen de communication central entre un programme et les fonctions du système d'exploitation ou du BIOS.

Les périphériques, ou plutôt les événements se produisant sur ces périphériques, sont référencés par le BIOS, dans une table appelée "vecteur d'interruption". Une fois son gestionnaire d'interruption écris, il faudra donc l'associer à l'un des vecteurs de cette table, c'est à dire à un certain événement (division par 0, pression de Ctrl-Alt*Suppr, etc...). Inversement, le PIC se servira cette table pour déterminer la cause des interruptions qu'il a à gérer.

4.6.2 La table des vecteurs d'interruption

La table des vecteurs d'interruption est en quelques sorte la clé d'accès aux interruptions dans laquelle se trouve l'adresse de la fonction recherchée. Chaque élément de cette table occupe deux mots successifs car il s'agit, concrètement, d'un pointeur FAR qui donne le segment et l'offset du gestionnaire associé. La table s'étend donc sur 1024 Octets (1 Ko).

Le numéro de l'interruption constitue l'indice d'entrée dans la table, qui est initialisée au démarrage du système, de façon à ce que les différents vecteurs d'interruption pointent sur les fonctions du BIOS. Cependant, certaines entrées dans cette table sont vides et par conséquent utilisable par un programmeur. Voici cette table :

(a été rajouté à la table des vecteurs d'interruption le libellé de l'événement correspondant à l'interruption)

N°	Adresse	Fonction
00	000 - 03	CPU : Division par 0
01	004 - 007	CPU : Pas à pas
02	008 - 00B	CPU : NMI (Défaut dans RAM)
03	00C - 00F	CPU : Point d'arrêt atteint
04	010 - 013	CPU : Débordement numérique
05	014 - 017	Copie écran
06	018 - 01B	Instruction inconnue (80286 seul)
07	01D - 01F	Réservé
08	020 - 023	IRQ0 : Timer (appel 18,2 fois/sec)
09	024 - 027	IRQ1 : Clavier
0A	028 - 02B	IRQ2 : 2eme 8259 (Correspond au PIC)
0B	02C - 02F	IRQ3 : Interface série 2
0C	030 - 033	IRQ4 : Interface série 1
0D	034 - 037	IRQ5 : Disque dur
0 E	038 - 03B	IRQ6 : Disquette
0F	03D - 03F	IRQ7 : Imprimante
10	040 - 043	BIOS : fonction vidéo
11	044 - 047	BIOS : Détermination configuration
12	048 - 04B	BIOS : Détermination taille RAM
13	04C - 04F	BIOS : Fonction disquette/Disque dur
14	050 - 053	BIOS : Accès à interface série
15	054 - 057	BIOS : Fonction cassette ou étendue
16	058 - 05B	BIOS : Interrogation clavier
17	05D - 05F	BIOS : Accès à imprimante parallèle
18	060 - 063	Appel du BASIC en ROM
19	064 - 067	BIOS : Démarrage à chaud (CTRL + ALT + DEL)
1A	068 - 06B	BIOS : Lecture date et heure
1B	06C - 06F	Touche Break actionnée
1C	070 - 073	Appelé après chaque INT 08
1D	074 - 077	Adresse de la table des paramètres vidéo

1 E	078 - 07B	Adresse de la table des paramètres disquette
1F	07C - 07F	Adresse des caractères graphiques
20	080 - 083	DOS : Terminaison su programme
21	084 - 087	DOS : Fonction de DOS
22	088 - 08B	Adresse de routine DOS fin du programme
23	08C - 08F	Adresse de routine CTRL-BREAK du DOS
24	090 - 093	Adresse de routine d'erreur du DOS
25	094 - 097	DOS : Lecture disquette/disque dur
26	098 - 09B	DOS : Ecriture sur disquette/disque dur
27	09C - 09F	DOS : Fin de programme, laisser résident
28- 3F	0A0 - - 0FF	Réservé pour fonction du DOS
40	100 - 103	BIOS : Fonction disquette
41	104 - 107	Adresse table des paramètres disque dur 1
42 - 45	108 - - 117	Réservé
46	118 - 11B	Adresse table des paramètres disque dur 2
47 - 49	11C - - 127	Librement définissable par le programme utilisateur
4A	128 - 12B	Heure alarme atteinte (AT)
4B - 67	12C - - 19F	Librement définissable par le programme utilisateur
68 - 6F	1A0 - - 1BF	Inutilisé
70	1C0 - 1C3	IRQ08 : Horloge temps réel (AT)
71	1C4 - 1C7	IRQ09 (AT seulement)
72	1C8 - 1CB	IRQ10 (AT seulement)
73	1CC - 1CF	IRQ11 (AT seulement)
74	1D0 - 1D3	IRQ12 (AT seulement)
75	1D4 - 1D7	IRQ13 : 80287 NMI (AT)
76	1D8 - 1DB	IRQ14 : Disque dur (AT)
77	1DC -1DF	IRQ15 (AT seulement)
78 - 7F	1 E0 - - 1FF	Inutilisé
80 - F0	200 - - 3C3	Utilisé à l'intérieur de l'interpréteur BASIC
F1 - FF	3C4 - - 3CF	Inutilisé

Table des vecteurs d'interruption (toutes les adresses sont en hexadécimal)

Après avoir déclaré la fonction dans la table des vecteurs d'interruption, il faut attribuer au gestionnaire d'interruption une IRQ. L'interruption sera alors installée. On peut résumer cette procédure par le scéma :

Installation du gestionnaire d'interruption ➔ Affectation à un vecteur d'interruption ➔ Affectation d'un n° d'IRQ

Vous trouverez des détails sur les commandes C permettant de réaliser ces opérations à la page 27 ("Manipulation des interruptions").

5. Bases de programmation en C

La plus importante révélation pour nous durant ce projet fut la découverte de la programmation à bas niveau. Nous avons ainsi dut programmer avec des méthodes et des instructions particulières, qu'un élève moyen de DUT ne maîtrise pour ainsi dire pas. Voici un petit récapitulatif des notions importantes pour une programmation efficace des registres de l'UART.

5.1 Utilisation de la notation hexadécimale

Une courte remarque concernant la notation hexadécimale : elle permet de simplifier l'écriture d'une longue suite de bits, plus difficile à déduire, en les regroupant par quatre. Elle se signale en C par le signe "0x" puis un nombre exprimé en hexadécimal.

Exemple d'utilisation de cette notation :

	Premier chiffre				**Deuxième chiffre**			
Héxa	C				2			
Décimal	12				2			
Binaire	1	1	0	0	0	0	1	0
Valeur du Digit	$2^3 + 2^2 = 8 + 4 = 12$				$2^5 = 32$			

Ainsi, le nombre 44 (32 + 12) est retranscrit dans le programme C sous la forme **0x2C**

5.2 Les opérateurs binaires

Il existe différents opérateurs en C qui permettent de manipuler les bits. Certains d'entre eux sont utilisés dans le programme UART.C (voir le chapitre "Programme d'application" ci-dessous) et leur fonctionnement doit donc être détaillé.

Le tableau ci-dessous dresse une liste des opérateurs les plus utiles, avec une explication de leur utilité, plus un exemple (en italiques) pour chacun d'entre eux :

Opérateur	Utilisation
V1 << V2	Cet opérateur renvoie le contenu de la variable V1 mais dont les bits sont décalé de V2 positions vers la gauche, avec insertion de zéros. *Ainsi l'expression "1110 0111 << 2" renvoie "1001 1100"*
V1 >> V2	Cet opérateur renvoie le contenu de la variable V1, mais dont les bits sont décalés de V2 positions vers la droite, avec insertion de zéros. *Ainsi l'expression "1110 0111 >> 2" renvoie "0011 1001"*
V1~	Permet d'appliquer l'opérateur logique NON à la variable V1. *Ainsi l'expression "1001~" renvoie "0110"*
V1 **&** V2	Permet de combiner les bits de deux variables uns à uns par la fonction ET logique. *Ainsi l'expression "1011 **&** 0111" renvoie "0011"*
V1 ^ V2	Permet de combiner les bits de deux variables uns à uns par la fonction logique OU EXCLUSIF. *Ainsi l'expression "1010 ^ 0110" renvoie "1100"*
V1 | V2	Permet de combiner les bits de deux variables uns à uns par la fonction OU logique. *Ainsi l'expression "1010 | 0110" renvoie "1110"*

Enfin, tous les opérateurs binaires (donc ceux mettant en oeuvre deux termes) peuvent être combinés avec le signe "**=**", pour devenir un opérateur d'affectation.

*Ainsi, l'expression "V1 **&**= V2" est équivalente à l'expression "V1=V1**&**V2" - Cette opérateur sera d'ailleurs courrament employé quand il y a nécessité d'appliquer des masques (faire disparaître certains bits)*

5.3 Commandes pour accéder à la mémoire

5.3.1 **unsigned char** *inportb(***int** *portid)*

Cette fonction renvoie le contenu du registre mémoire identifié par l'adresse **portid** sous la forme d'un caractère non signé. **Inportb** est en réalité une macro qui peut être compilée comme une fonction inline et qui est équivalente à l'instruction **80x86 In**.

Exemple d'utilisation de cette instruction extraite du programme UART.C

```
#define LCR 0x2FB
(...)
```

etat_ligne.lcr=**inportb**(**LCR**);

L'instruction **inportb** permet de placer le contenu du registre de contrôle de ligne de l'UART dans la variable **etat_ligne.lcr**.

5.3.2 **void** *outportb(***int** *portid,* **unsigned char** *value)*

Cette fonction permet d'assigner le contenu du registre mémoire identifié par l'adresse **portid** à la valeur **value** (de type caractère non signé, donc exprimé sur 8 bits). **Outportb** est en réalité une macro qui, à l'instar de **inportb**, peut être compilée comme une fonction inline et qui est équivalente à l'instruction **80x86 out**.

Exemple d'utilisation de cette instruction extraite du programme UART.C

```
#define MCR 0x2FC
(...)
outportb(MCR,0x08);
```

L'instruction **outportb** permet d'assigner le contenu du registre de contrôle de l'état du modem à la valeur 0x08 (ou en suite de bits : 0000 1000).

5.4 Manipulation des interruptions

5.4.1 **void interrupt(*** *isr)()*

Cette fonction permet de définir une fonction comme étant un gestionnaire d'interruption.

Exemple d'utilisation de cette instruction extraite des programmes UART.H et UART.C

```
#define COM2 0x0B
(…)
void interrupt (*old_f_int)();
(...)
old_f_int=getvect(COM2);
```

Ainsi, l'adresse de l'ancien gestionnaire d'interruption assigné au port COM2 est stocké dans le pointeur sur fonction *old_f_int*.

5.4.2 ***** *getvect(***int** *interruptno)*

Cette fonction renvoie l'adresse en mémoire du gestionnaire d'interruption assigné au vecteur d'interruption numéro *interruptno*.

Exemple d'utilisation de cette instruction extraite du programme UART.C

```
#define COM2 0x0B
(...)
old_f_int=getvect(COM2);
```

Ainsi le pointeur sur fonction old_f_int reçoit pour valeur l'adresse du gestionnaire d'interruption assigné au vecteur d'interruption du port **COM2**.

5.4.3 **void** *setvect(***int** *interruptno,* **void** *interrupt (*isr))*

Cette fonction place dans le vecteur *interruptno* l'adresse d'une nouvelle fonction d'interruption présente à l'adresse contenue dans le pointeur *isr*.

Exemple d'utilisation de cette instruction extraite du programme UART.C

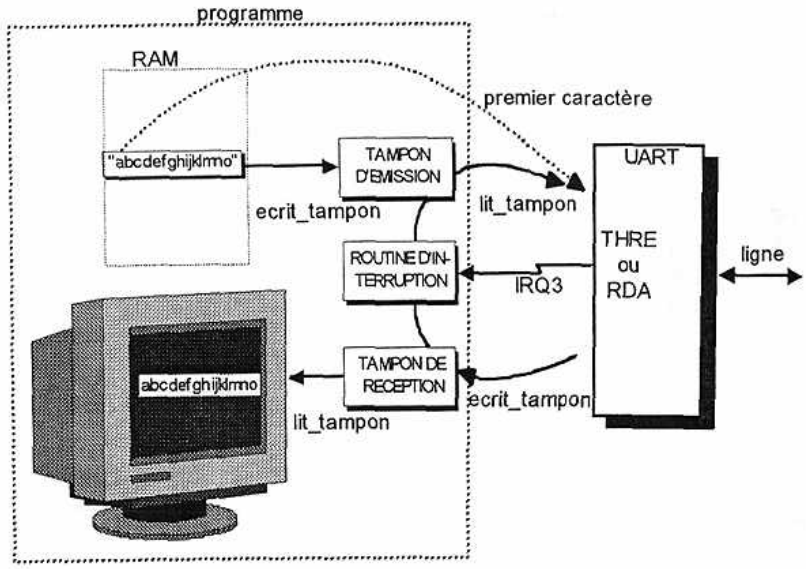
```
#define COM2 0x0B
(...)
setvect(COM2,V24_int);
```

L'instruction *setvect()* assigne un nouveau gestionnaire d'interruption (la fonction *V24_int( )* déclarée au préalable comme gestionnaire d'interruption) au vecteur d'interruption du port **COM2**.

6. Programme d'application

6.1 Principe et fonctionnement

Le fonctionnement du programme d'application peut être résumé de manière simple par le schéma figurant ci dessous :



Il s'agit de simuler un transfert de caractères en full duplex entre deux PC, et de mettre en évidence les problèmes de saturation du tampon de réception.

Une chaîne de caractères est transmise par l'ordinateur au port série depuis la RAM. La chaîne est émise puis réceptionné par ce même port série et affichée avec un délai permettant de simuler une cadence de traitement différée.

Pour l'émission comme pour la réception, deux tampons circulaires de type FIFO (First In First Out) pouvant recevoir ***taille*** caractères sont utilisés. Le nombre courant de caractères est noté ***nbre_car*** et les pointeurs de début et de fin ***debut*** et ***fin***. Le pointeur ***RD*** contient l'adresse du prochain caractère à lire et le pointeur ***WR*** l'adresse du prochain caractère à écrire. L'entier ***full*** permet de signaler si le tampon est saturé ou non. L'ensemble de ces informations est regroupé dans une structure de données intitulée ***TAMPON***. Enfin on définit une structure intitulée ***BUFFERS*** qui contient deux pointeurs sur les deux tampons (un tampon d'émission et un tampon de réception).

Les sources de la bibliothèque gérant la laison avec l'UART (UART.H) et du programme principal (SATUR.C) sont fournies ci-dessous.

6.2 Sources

6.2.1 Bibliothèque de gestion de la liaison : UART.H et UART.C

```
/*********************************************/
/* Fichier : UART.H                          */
/*********************************************/




#include<dos.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

#define TAILLE 16  /* taille du tampon de reception
*/
#define ESC 0x1B
#define LF 0x0D
#define COM1 0x0C
#define COM2 0x0B

#define OUT 1 /* moitie sup de l'ecran */
#define IN 0 /* moitie inf de l'ecran */

/* controleur d'it 8259 (PIC) */
#define ICR 0x20 /* ctrl des IT */
#define EOI 0x20 /* acquittement des IT */
#define _8259 0x21 /* masquage des IT */

/* controleur de liaison serie 8250 (UART) */
#define IER 0x2F9
#define IIR 0x2FA
#define LSR 0x2FD
#define THR 0x2F8
#define RBR 0x2F8
#define DLR 0x2FB
#define DRL 0x2F8
#define DRH 0x2F9
#define MCR 0x2FC
#define LCR 0x2FB

/* a controler !!!!!! */
#define IER_RDA 0x01    /*autorisation si recept
plein */
#define IER_THRE 0x02   /*autorisation si reg
transmission vide */
#define IER_RLS 0x04    /*autorisation si erreur en
recept */
#define IIR_RDA 0x04    /* identification IT ->
recept plein */
#define IIR_THRE 0x02   /* identification IT -> reg
transmission vide */
#define IIR_RLS 0x06    /* identification IT ->
erreur en reception */
#define IT_MSK 0x00
#define DTR_1 0x1
#define RTS_1 0x2

typedef struct {
                  unsigned taille;
                  unsigned nbre_car;
                  unsigned full;
                  char *debut;
                  char *fin;
                  char *WR; /* & du caractere en
ecriture*/
                  char *RD; /* & du caractere en
lecture */
                } tampon;

typedef struct {
                  tampon *tampon_recept;
                  tampon *tampon_emiss;
                  int etat_IT;
                } BUFFERS;
typedef struct {
         unsigned char imr,lcr,drl,drh,mcr,lsr;
       } etat_ligne;

static BUFFERS *pt_it;

tampon *ouvre_tamp(unsigned);
void init_ligne(void);
int ecrit_tampon(tampon*,char);
char lit_tampon(tampon *);
void interrupt V24_int(BUFFERS *);
void sortie(void);
int out_ctrlbrk(void);
void affiche(int,char);
void scroll(int);
void restaure_ligne(void);

/* void interrupt (inrterrupt* old_f_int) ();
pointeur sur ancienne fonction d'IT */

void delay(int);



-----------------------------------------------




/*********************************************/
/* Fichier : UART.C                          */
/*********************************************/



#include "uart.h"

etat_ligne etatl; /* Variable globale etat ligne */

void init_ligne()
{
 char c;

 /* sauvegarde ancienne fonction d'IT */
 /* interrupt old_f_int;
 old_f_int=getvect(COM2); */

 /* sauvegarde des anciennes valeurs de l'UART */
 etatl.imr=inportb(_8259);
 etatl.lcr=inportb(LCR);
 etatl.drl=inportb(DRL);
 etatl.drh=inportb(MCR);
 etatl.lsr=inportb(LSR);

 disable(); /* masque les interruptions pendant les
modifs */

/* nouvelle config de la ligne */
/* outportb(LCR,????); */
 outportb(DLR,0x0C);
 outportb(DRH,0x00);
 outportb(LCR,0x03 | 0x04 ); /* 8 bits, 1 arret, pas
de parite */
 outportb(MCR,0x08 | DTR_1 | RTS_1);
 outportb(IER,IER_RDA); /* genere une IT si
caractere present */
 c=inportb(IIR); /* lecture d'etat du registre IIR
(cause IT) */
 c &=0x06; /* Masque sur les IT port COM2 */
 outportb(ICR,0x08); /* modification du controleur
d'IT */
 setvect(COM2,V24_int); /* installation nouveau
                                        gestionnaire
d'IT */
 enable();
}

/* ouverture buffer ciculaire et initialisation
parametres */
tampon *ouvre_tamp(unsigned taille)
{
  tampon *t;
  t=(tampon*) malloc (sizeof(tampon));
  disable();
```

```
  t->taille=taille;
  t->nbre_car=0; /* Initialisation */
  t->full=0;
  t->debut=NULL;
  t->fin=NULL;
  t->WR=NULL;
  t->RD=NULL;
  enable();
  return (t);
}

/* ecriture d'un caractere c dans le buffer
circulaire et mise
      a jour des parametres */
int ecrit_tampon(tampon *t,char c)
{
    disable();
  if (t->full !=1)
      {
        t->nbre_car++;
        t->WR=&c;
         t->debut=&c;
        t->fin=&c;
        enable();
        return 0;
      }
else {enable();return -1;}/* Si erreur return -1 */
}

/* lecture d'un caractere c dans le buffer
circulaire et mise
      a jour des parametres */
char lit_tampon(tampon *t)
{
  disable();
  if (t->nbre_car==0){enable();return -1;}

  else
      {enable();return *(t->WR);}
}

/* Gestionnaire d'interruption
lecture du registre de cause d'IT
       si reg emiss vide -> lecture tampon et
emission
       si reg recept plein -> lecture reg et mise a
jour tampon */
void interrupt V24_int(BUFFERS *b)
{
char s;
char c;/* a modif */
int i;
b->etat_IT=c; /* inportb(IIR) */
if(b->etat_IT==0x02) { s=lit_tampon(b-
>tampon_emiss); /* lecture tampon et emission */
                     }
if(b->etat_IT==0x04) { s=inportb(RDR);
                       i=ecrit_tampon(b-
>tampon_recept,s); /* lecture reg et mise a jour
tampon */
                     }
}

void affiche(int level,char c)
{
  static int col[2]={1,1};
  if ((col[level]>=79)||(c==LF))
     {
      col[level]=1;
      scroll(level);
     }
  if (c>=0x20) /* caractere affichable ? */
     {
      gotoxy(col[level],level?24:12);
      putch(c);
      col[level]++;
     }
  gotoxy(col[OUT],24);
}

void scroll(int level)
{
 union REGS regs;
 regs.h.al=1; /* defilement des lignes */
 regs.h.ch=level?13:0;
 regs.h.cl=0;
 regs.h.dh=level?24:12;
 regs.h.dl=79;
 regs.h.bh=7;
 regs.h.ah=6;
 int86(0x10,&regs,&regs);
}

void sortie() /* fonction executee automatiquement a
la fin du prog */
{
  restaure_ligne();
}

int out_ctrlbrk() /* fonction executee en cas de
ctrl-brk */
{
  restaure_ligne();
  return 0;
}

void restaure_ligne()
{
 outportb(_8259,etatl.imr);
 outportb(LCR,etatl.lcr);
 outportb(DRL,etatl.drl);
 outportb(MCR,etatl.drh);
 outportb(LSR,etatl.lsr);
}

void delay(){}
```

6.2.2 Programme principal : SATUR.C

```
/*********************************************/
/* Fichier : SATUR.C                         */
/*********************************************/



#include "c:\projet\uart.h"

char ch[16]="abcdefghijklmno"; /* chaine test */

void main()
{
 char caract,c;
 int i;
 BUFFERS *b;

 pt_it=b; /* type static "BUFFERS*" */

 b->tampon_recept=ouvre_tamp(100); /* taille du
tampon de recept voulu ? */
 b->tampon_emiss=ouvre_tamp(100);
 b->etat_IT=0;

 atexit(sortie); /* exe la fct sortie() en quittant
*/
 ctrlbrk(out_ctrlbrk); /* exe la fct out_ctrlbrk()
si util de ^C */

 clrscr();
 gotoxy(1,24);
 init_ligne();

 do
   {
    if (kbhit()) /* si touche enfoncee */
         {
          c=getch();

          if (c==ESC)
                break;
          else
              if(c==' ') /* caractere espace */
                for (i=0;i<sizeof(ch);i++)
```

```
                  {
                   caract=ch[i];
                   disable();
                   if (i==0)
                    {
                     enable();
                     outportb(TDR,caract);
                     outportb(IER,IER_RDA |
IER_THRE); /* autorisation IT THRE */
                    }
                   else if (ecrit_tampon(b-
>tampon_emiss,caract)==EOF)
                    {
                     enable();
                     while(ecrit_tampon(b-
>tampon_emiss,caract)==EOF) {};
                    }
                   else enable();
                  }
                else printf(" caractere errone ...
");
        }
    caract=lit_tampon(b->tampon_recept);
    delay(10); /* pour simuler une cadence de
traitement differente */
    if (caract!=EOF) affiche(IN,caract);
   }
   while (c!=ESC);

   printf("bye bye !\n");
}
```

6.3 Conclusions sur le programme d'applications

Notre première tâche fut de compléter le programme. En effet, de nombreux passages avaient été laissés vides... Par exemple, il nous a fallu trouver les valeurs des #define de la bibliothèque. Ensuite, comme le programme ne compilait pas, nous avons remarqué les nombreuses erreurs de programmation... Ainsi nous a-t-il fallu les corriger unes à unes pour arriver à une compilation "sans fautes" du programme.

Bien malheureusement, notre équipe n'a pas eu le temps de continuer, et le programme ne fonctionne pas : rien ne s'affiche à son exécution - nous n'avons pas pu déterminer quel problème (pas de transmission, ou erreur de la procédure d'affichage) était la source de ce disfonctionnement. Notre but a donc été de réaliser une documentation la plus complète possible afin qu'un autre groupe d'étudiant puisse faire marcher ce programme, puis bien sûr reprendre la bibliothèque UART, pour la compléter et établir un véritable protocole.