I Manuali di Giobe2000

PORTA SERIALE

HW Terza parte

Studio Tecnico ing. Giorgio OBER eurosito@giobe2000.it

La Monografia sulla **PORTA SERIALE** può differire in parte dalla versione on-line
soggetta a costanti aggiornamenti e integrazioni
Verifica le eventuali novità direttamente sul Sito

Caratteristiche UART — Porta Seriale

1 - Caratteristiche UART - UART: Caratteristiche generali

1.1 I compiti dell'UART

- Nella prima parte della trattazione ho parlato della ***porta seriale*** come oggetto capace di ***ricevere*** e ***trasmettere*** dati inserito in un contesto di *regole* e di *tipologie* codificato ormai da tanti anni; per rinfrescare questi concetti te li riassumo sinteticamente:
  - organizzazioni che hanno fornito le **regole** della comunicazione seriale
  - sintesi delle **caratteristiche** di un'***interfaccia seriale***
  - **tipologie** della *comunicazione seriale* **DTE-DCE**.
  - **specifiche** dettagliate dello standard **RS232** nella *comunicazione seriale* con protocollo di tipo *asincrono* (generalità, struttura dei dati, forma e temporizzazioni dei segnali, specifiche elettriche, velocità seriale (differenza bps/ baud e valori raggiungibili), descrizione dei segnali e loro funzione nell'ambito del protocollo (handshake) hardware, connettori, cavi)
- Questa seconda parte è decisamente orientata alla conoscenza profonda del dispositivo **UART**, chiamato a sintetizzare tutti i concetti descritti in precedenza, con lo scopo dichiarato di arrivare alla tanto desiderata *programmazione* della ***porta seriale***.
- Prima di iniziare desidero ricordarti i compiti che l'***interfaccia seriale*** (**UART**) deve assolvere:
  - **converte** ciascun dato da trasmettere, fornito in ***parallelo*** dal processore, in un **flusso *seriale*** di bit da porre sulla linea di trasmissione; oppure **converte** in ***parallelo***, per il processore, il **flusso *seriale*** di bit ricevuto dalla linea di ricezione
  - **formatta** il dato seriale *asincrono*, aggiungendo *bit di inizio e fine dato* ed eventuali *bit di controllo*, adatti a rilevare eventuali *errori di ricetrasmissione*
  - **organizza** le temporizzazioni necessarie per sincronizzare la *collocazione* di ciascun bit sulla linea dati, durante la trasmissione, e la loro *lettura*, durante la ricezione
  - in aggiunta alle *linee di dato* deve **mettere a disposizione** altre *linee*, previste dallo standard **RS232** *per il controllo del flusso* dati (*hardware handshaking*) o *per indicare lo stato* del mezzo di trasmissione o del modem che lo controlla.

1 - Caratteristiche UART - UART: Caratteristiche generali

1.2 La necessità dei Buffer di memoria

- Fin dalla sua prima comparsa sul *PC IBM originale* l'***interfaccia seriale*** ha affidato tutti questi compiti ad un componente appositamente inventato da *National Semiconductor*, l'**Universal Asynchronous Receiver/Transmitter**, sinteticamente noto come **UART** e completamente programmabile; solo per completezza va ricordata anche la disponibilità di un'interfaccia di tipo sincrono detta **USART,** Universal Syncronous Asynchronous Receiver/Transmitter, più complessa sia nei circuiti che nei protocolli di gestione.
- Nei moderni computer il componente **UART** (talvolta noto come **SCC**, **S**erial **C**ommunications **C**hip) è incluso in un chip della *scheda madre* chiamato a fare anche altre cose; solo molti anni or sono il suo chip, spesso su zoccolo, era ben visibile (per la sua grossa dimensione) sulle vecchie *motherboard* oppure su apposite *scheda di I/O*; anche le *schede modem* hanno il loro chip **UART**.
- Questo componente ha subìto una naturale *evoluzione nel tempo*, non solo logisticamente; le elevate velocità di ricetrasmissione, soprattutto nei confronti di un modem, hanno reso necessaria una sostanziale revisione dei suoi **buffer** interni.

> Un **buffer** è una piccola quantità di *memoria ram* destinata ad immagazzinare *informazioni di passaggio*; fin delle prime versioni i dispositivi **UART** ne hanno fatto uso, per memorizzare *temporaneamente* i *dati in arrivo* e *quelli in partenza*.

- Sebbene i *compiti funzionali* siano rimasti inalterati negli anni la dimensione originale dei **buffer** degli **UART** si è mostrata ben presto un *insopportabile* ***limite***, tale da ***rendere inaffidabile*** la stessa ricetrasmissione seriale, specialmente all'aumentare della velocità.
- Per comprenderne le cause bisogna sapere che l'**UART**, quando *riceve* o *trasmette* un byte, lo colloca in un suo **buffer** interno (di ricezione o di trasmissione) e poi, con un *interrupt hardware*, avvisa il processore della necessità di ***leggerlo*** dal buffer o di ***predisporne*** un altro, se lo ritiene necessario.

1 - Caratteristiche UART - UART: Caratteristiche generali

1.3 Effetto di Buffer ad una sola locazione

- Il problema è dunque tanto più probabile quanto *più piccolo* è il buffer e tanto *più grande* è la velocità di trasferimento:
  - se i **buffer** sono costituiti da ***una sola*** *locazione* (come negli **UART** della prima generazione) si ha una *richiesta di servizio* ***ogni volta*** che un byte è ricevuto o trasmesso
  - in presenza della *richiesta di servizio* di una *interruzione* il processore, ***prima o poi***, esegue il compito richiesto
  - se ogni byte ricevuto o trasmesso ne ha bisogno, il *tempo a sua disposizione* è facilmente calcolabile a partire dalla *velocità* della comunicazione; nell'ipotesi più semplice (formattazione con **solo** *bit di start* e *di stop*) per ogni byte d'informazione servono 10 bit, per cui:

velocità bit/sec	**caratteri e interrupt** al sec	**tempo a disposizione** per ogni interrupt	
1200	120	**8,33** ms	[1000/120]
2400	240	**4,12** ms	[1000/240]
4800	480	**2,08** ms	[1000/480]
9600	960	**1,04** ms	[1000/960]
19200	1920	**521** µs	[1000/1920]
38400	3840	**260** µs	[1000/3840]
57600	5760	**137** µs	[1000/5760]
115200	11520	**87** µs	[1000/11520]

  - con *bassa velocità* di trasferimento (massimo **2400** baud, con gli **UART** della prima generazione) riesce a soddisfare le *richieste* senza compromettere la sua normale attività; il numero di *caratteri al secondo* è in sostanza il numero di *richiesta di interruzione al secondo*
  - la cosa diventa critica all'*aumentare* della *velocità*: il processore può essere occupato a servire altri dispositivi (dischi, tastiera, monitor, ..) per cui, se le richieste diventano troppo frequenti, può succedere che non riesca (per esempio) a *scaricare* in tempo il byte dal **buffer** di ricezione, ***prima*** dell'arrivo del nuovo byte, con evidente ***perdita d'informazione***
  - per esempio, a **9600** baud, per fare ogni cosa ha a disposizione poco più di **1** millisecondo..
  - anche se (con speciali *modem a correzione d'errore*) in presenza di *perdita di bytes* può essere richiesta la loro ritrasmissione è evidente che anche questo comporta un inutile aumento della durata del collegamento
  - in aggiunta, la CPU sarà certamente *troppo impegnata* ed avrà poco tempo per gli altri compiti, riducendo le *prestazioni di sistema* a valori inaccettabili

1 - Caratteristiche UART - UART: Caratteristiche generali

1.4 Effetto di Buffer a 16 locazioni

- Non potendo certo porre limiti *alla massima velocità possibile*, per permettere una *corretta* ricetrasmissione si è scelto di intervenire sulla dimensione dei **buffer**, andata crescendo in ogni **UART** della nuova generazione, da **16** a **128** bytes.
- Anche la struttura di questa *memoria temporanea* ha contribuito ad ottimizzare la gestione dei dati, essendo del tipo **FIFO** (**F**irst **I**nput **F**irst **O**utput); come dice il suo nome i bytes saranno *estratti* nell'***esatto ordine*** con cui sono stati *immessi*, cioè il primo inserito è anche il primo ad essere tolto.
- Con la presenza di un buffer **FIFO** (per esempio di **16** bytes):
  - la ricetrasmissione viene gestita dall'**UART** senza l'intervento del processore, che nell'attesa è libero di fare altre cose
  - l'intervento del processore è richiesto dall'**UART** (attivando da *hardware* una linea di *interruzione*) ***solo quando*** sono stati *inviati* tutti i 16 bytes predisposti nel buffer o ne sono stati *ricevuti fino a 14*, ***riducendone*** quindi ***drasticamente*** il numero
  - il valore massimo di bytes (**14** su **16**, detto *trigger level*) da *ricevere* prima di far partire una *richiesta di interrupt*, si può predisporre da programma anche ai valori **8**, **4** e **1**
  - il processore ***può trasferire*** (o ***leggere***) ***in blocco*** *tutti i bytes* da trasmettere (o ricevuti), con ulteriore ottimizzazione dei tempi

- se le *richiesta di interruzione* sono fatte (mediamente) ogni 16 bytes ricevuti o trasmessi, i tempi a disposizione per il *servizio* diventano ovviamente 16 volte più grandi; la tabella mostra una stima dei tempi nell'ipotesi precedente, formattazione con **solo** *bit di start* e *di stop* cioè 10 bit per ogni byte d'informazione:

velocità bit/sec	caratteri al sec	interrupt al sec		tempo a disposizione per ogni interrupt	
1200	120	7,5	[120/16]	**133** ms	[1000/7,5]
2400	240	15	[240/16]	**66,6** ms	[1000/15]
4800	480	30	[480/16]	**33,3** ms	[1000/30]
9600	960	60	[960/16]	**16,6** ms	[1000/60]
19200	1920	120	[1920/16]	**8,33** ms	[1000/120]
38400	3840	240	[3840/16]	**4,16** ms	[1000/240]
57600	5760	360	[5760/16]	**2,77** ms	[1000/360]
115200	11520	720	[11520/16]	**1,38** ms	[1000/720]

- la velocità di comunicazione può raggiungere tranquillamente **115200** baud, senza *perdita d'informazione*

- Certamente ti sarai chiesto perchè, in *ricezione*, la *richiesta* di *interruzione* viene operata dall'**UART** un po' prima che il buffer **FIFO** sia riempito del tutto, per esempio con soli **14** bytes su **16**; lo scopo è quello di garantire un po' di spazio ad eventuali dati in arrivo mentre il processore è chiamato a soddisfare la *richiesta* stessa, cosa possibile dato che, come già detto, la ricetrasmissione viene gestita dall'**UART** senza l'intervento del processore.
- Un'altra curiosità è legata ad una possibile situazione critica: in accordo con il valore programmato del *trigger level*, se la ricezione non è fluida potrebbe succedere che il buffer **FIFO** non riesca a riempirsi del tutto, impedendo la *richiesta* di *interruzione* fino a quando è ricevuto il 14.mo byte e bloccando di fatto gli applicativi che fruiscono dei dati in arrivo.
- In realtà i progettisti dell'**UART** hanno previsto questa situazione, facendo intervenire comunque una *richiesta* di *interruzione* ***dopo*** *un tempo prestabilito* (***timeout***) anche se il buffer **FIFO** è ancora parzialmente vuoto; in questo modo potrebbe partire un servizio CPU anche con la presenza di un solo byte ma, nonostante la similitudine, la situazione è assolutamente diversa da quella tipica dei **buffer** da **1** byte degli **UART** della prima generazione.
- Sebbene meno pressante anche la *trasmissione dati* gode delle medesime attenzioni.

2 - Caratteristiche UART - Storia degli UART

2.1 Premessa

- Non rimane che fare la storia di questo importante componente; nel tempo i dispositivi seriali dei computer *IBM compatibili* furono dotati di **UART** (**Universal Asynchronous Receiver/Transmitter**) logicamente compatibili con il vecchio precursore, ma con architetture sempre più sofisticate e ottimizzate; vediamo tutte le ***versioni***, con le loro caratteristiche principali.

2 - Caratteristiche UART - Storia degli UART

2.2 UART della **prima generazione** - buffer di comunicazione ad **un solo byte**

- I componenti di questa categoria sono ormai retaggio storico; la loro sicura *obsolescenza* è legata ad una architettura *poco veloce* e *inadeguata* ai compiti richiesti: ciascuno di essi dispone infatti di **buffer** di trasmissione e di ricezione, entrambi **ad 1 solo byte**.
- Sono ospitati in *contenitori* (*chips*) *Dual-In-Line* (*DIL*) a 40 piedini, di solito *pin-out* compatibili e montati su *zoccolo* (*sockets*), per facilitarne la sostituzione con versioni più aggiornate.

8250

- Questo componente, prodotto dalla *National Semiconductor* con la sigla INS**8250** su commissione della *IBM*, fu installato nel 1981 sulle schede di ***interfaccia seriale*** dei suoi *PC originale* e *PC/XT* (8088 e 8086).
- Noto con il nome originale ***ACE*** (**A**synchronous **C**ommunications ***E***lement) era fatto in tecnologia NMOS; disponeva di 8 registri.
- La sua velocità massima era limitata a **9600** baud, del tutto inadeguata per dialogare con *modem* ad alta velocità.
- Poichè soffriva di qualche imperfezione funzionale la IBM pensò bene di porvi rimedio intervenendo sulle procedure di gestione della sua BIOS, rendendo impossibile l'uso delle *compatibili* e *migliori* versioni UART successive sui suoi *PC originale* e *PC/XT*, ma questa è solo storia.

8250A

- Pur provvedendo a correggere le imperfezioni funzionali della versione precedente non potè essere impiegato sui *PC IBM originale* e *PC/XT*, per il descritto, sconsiderato intervento della stessa *IBM* sulle rispettive BIOS, rese *8250 dipendenti*; per questo fu impiegato su *computer IBM compatibili* dotati di BIOS ripulite dalle (ora) inutili modifiche.
- Fatto in tecnologia XMOS, disponeva di un registro supplementare in grado di consentire al software di verificare se era veramente un **8250**.
- Era disponibile una versione CMOS a basso consumo, con la sigla **82c50A**.

16450

- Questo componente fu installato, a partire dal 1984, sulle ***interfaccie seriali*** dei *computer IBM AT* (286, 386 e 486); la stessa *IBM* si premurò di aggiornare anche la sua BIOS, ripristinandola alle sue funzioni originali (cioè eliminando le devastanti modifiche introdotte per rendere possibile il funzionamento del difettoso modello precedente).
- Funzionalmente identico al **8250**, disponeva di miglioramenti tali da consentirgli una convivenza ottimale con i bus a 8 Mhz dei processori più veloci.
- La sua velocità massima era leggermente superiore ai **9600** baud, meglio adatta a dialogare con *modem* ad alta velocità.
- Era disponibile una versione CMOS a basso consumo, con la sigla **16c450**.

16550 (prima versione)

- Nato nel 1987 per risolvere il problema della dimensione dell'area di memoria di transito fu il primo ad introdurre i **buffer FIFO** a **16** bytes ma, per un triste destino, proprio questa grande novità manifestò difetti di progettazione che la resero inutile, facendolo lavorare bene solo come 16450 (cioè con buffer *non FIFO*, a 1 solo byte).
- Pin-out compatibile con i predecessori, evidenziavano una differenza funzionalmente non rilevante dei segnali Trasmit Ready (**TXRDY**) e il Receive Ready (**RXRDY**), disponibili sui pin24 e pin29, garantendo comunque la compatibilità software.

2 - Caratteristiche UART - Storia degli UART

2.3 UART della seconda generazione - buffer di comunicazione a 16*n byte

- Dalla fine degli anni '80 gli **UART** sono stati migliorati sostanzialmente, rendendo possibile la memorizzazione locale temporanea (**buffer**) di **numerosi bytes**.
- Per garantire la compatibilità funzionale con i predecessori, devono ignorare (per default) la presenza del loro potente **buffer FIFO** interno: per poterne disporre è necessario *attivarli da software*, compito di solito assolto automaticamente dai ***drivers*** (programmi di gestione) e dai *programmi di comunicazione* seriali.
- Per la presenza del nuovo buffer questi componenti sono da tempo il punto di riferimento nelle ***interfaccie seriali***, per la loro capacità di gestire i moderni *modem* ad alta velocità nell'ambiente multitasking degli ultimi sistemi operativi, *senza perdita d'informazione*, in virtù della drastica riduzione di *richieste* di *interruzione* operate dall'**UART** in occasione della *ricetrasmissione* di bytes.
- Oltre che dalla *National Semiconductor*, detentrice di molti brevetti, i componenti di questa categoria sono ora prodotti su licenza anche da numerosi altri costruttori; sulle recenti *schede madri* non sono più visibili nello splendore degli storici 40 pin, essendo intergrati nel *chipset*, con molti altri vitali dispositivi.
- Alcune versioni *non originali* possono mostrare *differenze tali* da richiedere l'installazione di speciali ***drivers***, *per essere riconosciute* dai sistemi operativi più noti (come Windows o Linux); la cosa è più frequente per gli **UART** impiegati sulle schede *modem* interne o nei dispositivi *modem* esterni, dove possono essere addirittura ***simulati*** da un processore dedicato, in grado tra l'altro di gestire **buffer** di ricetrasmissione anche dell'ordine dei *Kbytes*.

16550A

- Poco dopo la sua presentazione il **16550**, gravato dai noti malfunzionamenti, fu *sostituito* da questa versione; installato sui computer 486, Pentium e successivi, il nuovo chip (per la discutibile scelta di taluni costruttori) è spesso presente con la vecchia sigla (**16550** invece di **16550A**).
- Sono ancora pin-out compatibili con i predecessori, per cui è possibile aggiornarli semplicemente sostituendoli nel loro zoccolo.
- La loro velocità (**DTE-DCE**, cioè *tra computer e modem*, *velocità di terminale*) può raggiungere i **115** Kbaud, ideale per dialogare con *modem* ad alta velocità, garantendo trasferimenti (tra **DCE-DCE**, cioè *tra modem e modem*) alle *velocità di linea* (o *velocità di dato*) di **14,4K**bps, **28,8K**bps, **33,6K**bps, **56K**bps.
- Nel tempo la *National Semiconductor* ha proposto 4 revisioni del suo componente, l'ultima delle quali è disponibile con la sigla **PC16550D**.
- E' disponibile anche in versione veloce, con la sigla **16550AF** e **16550AFN** (con chip in ceramica), e in versione CMOS, con la sigla **16c550A** e **16c552** (due **16c550A** in un singolo chip).

16650

- Questo componente, in dotazione sulle *schede madri* (spesso intergrati nel *chipset*) a partire dal 2000, è dotato di un **buffer FIFO** di **32** *bytes*. e di altre sofisticate caratteristiche, come la gestione avanzata dell'energia (Energy Star).
- La loro *velocità di terminale* (quella **DTE-DCE**, *tra computer e modem*) può raggiungere i **460.8** Kbaud, potenzialmente utile con *modem* ad alta velocità ed alto rapporto di compressione.

16C750

- Questi componenti sono dotati di un **buffer FIFO** di **64** *bytes*, sia in trasmissione che in ricezione
- La loro *velocità di terminale* (quella **DTE-DCE**, *tra computer e modem*) può raggiungere i **921.6** Kbaud, potenzialmente utile con *modem* ad alta velocità ed alto rapporto di compressione

16C950

- Questi componenti sono dotati di un **buffer FIFO** di **128** *bytes*, sia in trasmissione che in ricezione
- La loro *velocità di terminale* (quella **DTE-DCE**, *tra computer e modem*) può raggiungere i **15** Mbaud in modo normale e i **60** Mbaud in external clock mode (modo isocrono)

3 - Caratteristiche UART - UART: Caratteristiche tecniche

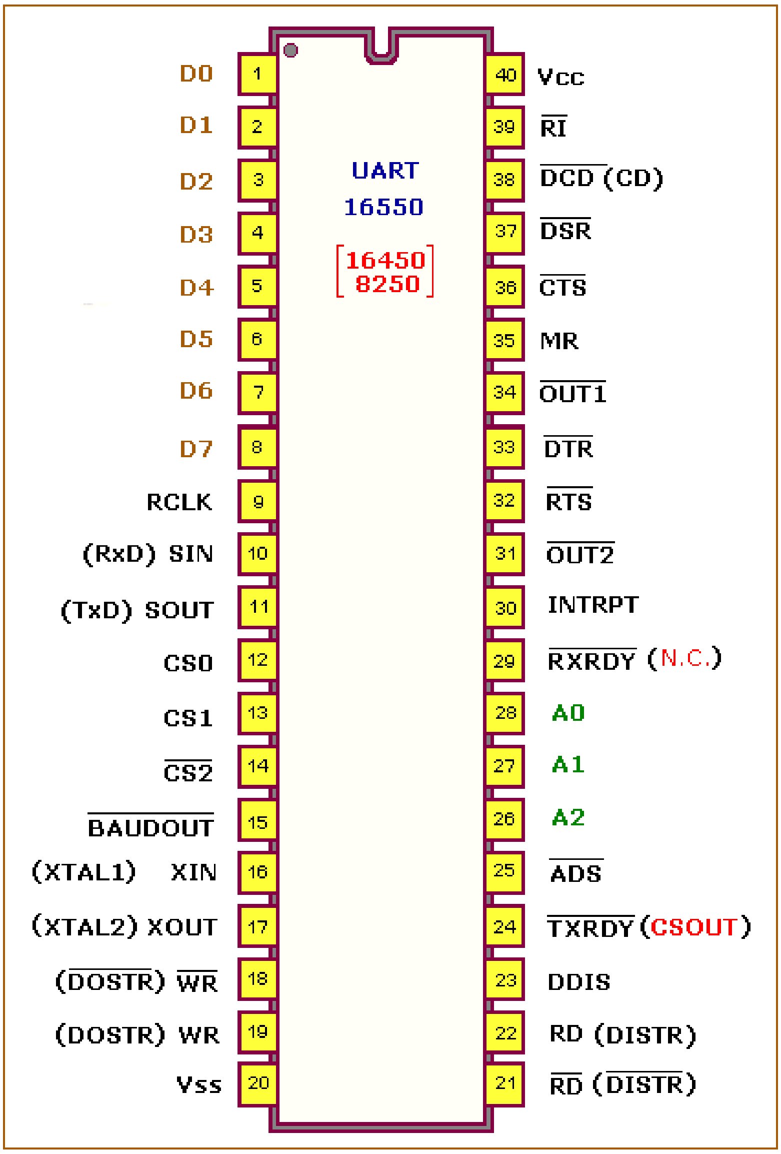
3.1 Componente **UART**: chip package **dual-in-line**

- Le caratteristiche tecniche di un **UART** (Universal Asynchronous Receiver/Transmitter) non sono particolarmente necessarie nell'uso di una porta seriale, se non per la parte dedicata ai suoi preziosi registri; tuttavia una panoramica su di esso può essere di qualche interesse...

- Il capostipite **8250** non è ovviamente più in commercio da tempo e la sua documentazione è piuttosto rara, ma tutte le moderne versioni successive sono comunque compatibili con **8250**/**16450**, anche a livello pin-out (con le eccezioni descritte tra breve...).

- La disposizione dei pins del componente in contenitore ***dual-in-line*** è visibile nello schema a lato.

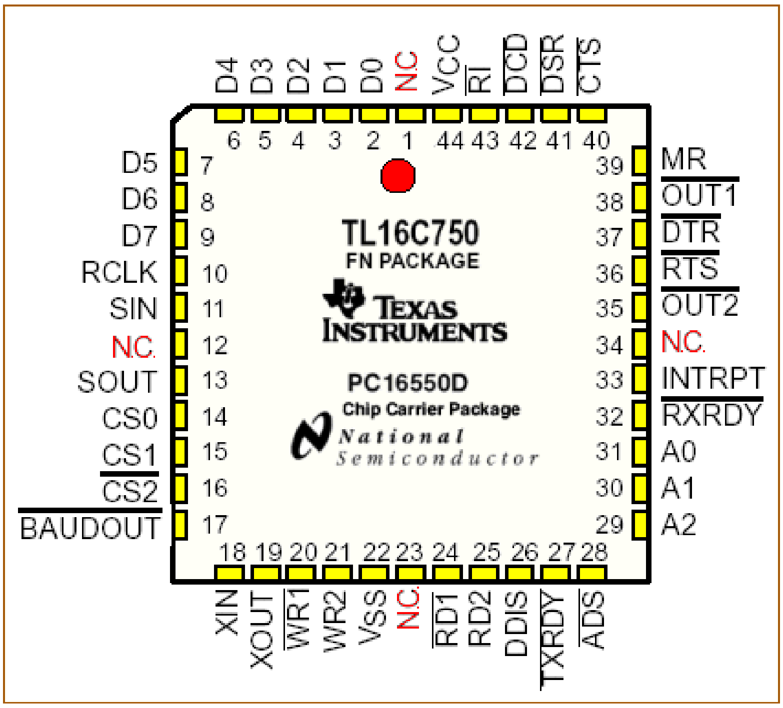
- Le uniche due differenze tra i primi **UART** e quelli attuali sono marcate in rosso nello schema:
  - nel **8250** il pin**24** serviva per selezionare il dispositivo (**C**hip **S**elect) mentre il pin**29** non era utilizzato
  - nei moderni **UART**, dal **16550** in poi, a questi pins è stato affidato rispettivamente il compito di *Trasmit Ready* (**TxRdy**) e di *Receive Ready* (**RxRdy**) per permettere l'uso del DMA (*Direct Memory Access*)

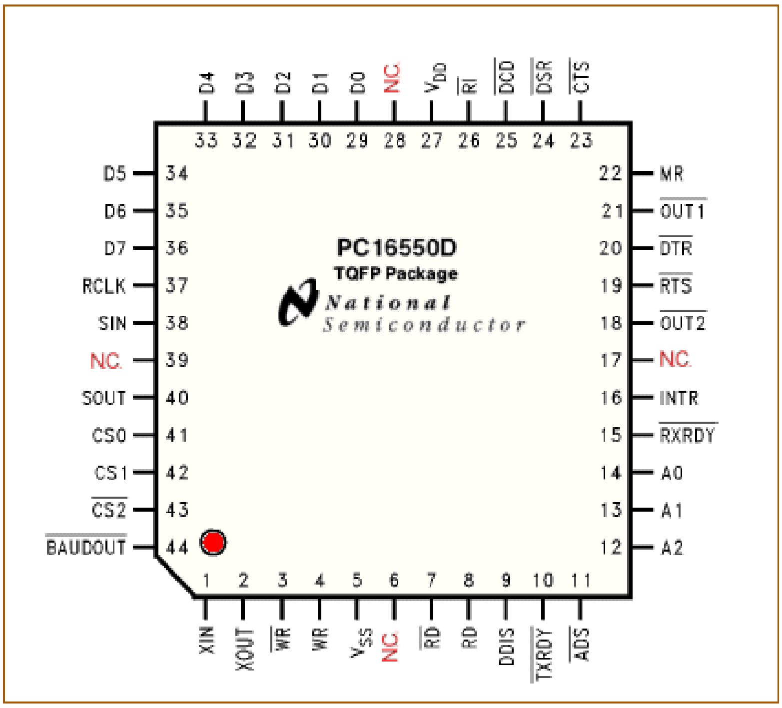


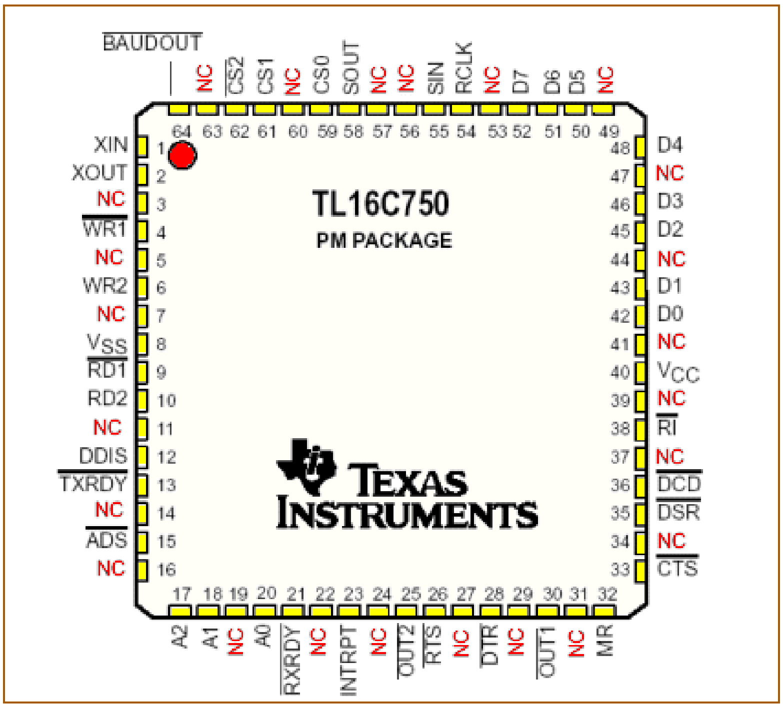
3 - Caratteristiche UART - UART: Caratteristiche tecniche

3.2 Componente **UART**: chip package **quadrangolare**

- Nelle moderne schede di I/O la disposizione *dual-in-line* è sempre meno frequente.
- Le figure seguenti mostrano l'aspetto di **UART 16550** e successivi in diversi contenitori *quadrangolari*.
- La maggiore disponibilità di pins (**44** per i *package* **FN** o **TQFP**, e ben **64** per quelli **PM**) assicura comunque la disponibilità delle **40** funzioni presenti sul contenitore *dual-in-line*; i rimanenti **4** e **24** pins sono dichiarati NC, cioè non collegati internamente.







3 - Caratteristiche UART - UART: Caratteristiche tecniche

3.3 Descrizione piedini [*prima parte*] - Segnali **UART**: linee di **selezione** e **controllo logico**

- Le *funzioni* associate a ciascun pin di un **UART** sono legate al suo *modo di operare* e, probabilmente, la loro conoscenza non è strettamente necessaria per il corretto uso e per la programmazione di questo componente; la lettura di queste pagine può essere però accattivante per farti capire in che modo verranno gestite le risorse a disposizione.
- Durante la loro descrizione può essere utile far riferimento allo schema funzionale, aperto in una nuova pagina.
- Tutti i piedini di un **UART** sono **TTL compatibili**, cioè presentano tensioni che vanno da **0,2**V (**0** logico) a **2,4**V (tipico per l'**1** logico); alcuni di essi sono *attivi bassi*, cioè a riposo sono a **1** logico ed esercitano la loro azione passando a **0**.
  - **Vcc** (pin**40**, **+5**V) e **Vss** (GND, pin**20**, **0**V): sono i riferimenti per l'**alimentazione** del componente
  - da **D0** (pin**1**) a **D7** (pin**8**): sono le 8 linee (consecutive) del **Bus Dati**, tutte *bidirezionali* di tipo *tri-state*; lo scambio di dati tra processore e UART passa attraverso di esse, ma anche il passaggio di comandi (da CPU a UART) e di informazioni di stato (da UART a CPU)
  - **A2** (pin**26**), **A1** (pin**27**) e **A0** (pin**28**): sono 3 linee di indirizzo (**Bus Address**) con le quali il processore può accedere alla scrittura/lettura di uno dei 12 registri interni dell'UART [come vedremo il codice binario da scrivere su queste linee (solo 8 combinazioni, da 000 a 111) è sufficiente per puntare tutti e 12 i registri, dato che alcuni di essi sono *alternativi*, cioè hanno lo stesso indirizzo ma possono essere discriminati sulla base di un particolare bit del Registro di Controllo della Linea].
  - **CS2** (pin**14**), **CS1** (pin**13**) e **CS0** (pin**12**): sono 3 linee di selezione (**Chip Select**) del dispositivo; in altre parole il processore per accedere all'UART dovrà ***attivare*** (*abilitare*) ***tutte e 3*** queste linee; l'operazione sarà esecutiva sul *fronte di salita* della linea **ADS**, dopo aver predisposto **CS2** (*attiva bassa*) a **0** e **CS1** e **CS0** (*attive alte*) a **1** logico.
  - **ADS** (pin**25**): questa linea (*attiva bassa*, **Address Strobe**) memorizza sul *fronte di salita* lo stato delle linee **A2**, **A1**, **A0**, **CS2**, **CS1** e **CS0** e ne mantiene il valore inalterato per tutto il tempo che rimane a **1** logico); solo quando è a **0** logico le linee d'*indirizzamento* dei registri interni e quelle di *abilitazione* possono essere impostate e consentire (quando sono *stabilizzate*) le operazioni di scrittura o di lettura dell'UART
  - **INTRPT** (pin**30**): questa linea (*attiva alta*, **Interrupt Output**) è posta a **1** dall'UART per avvisare il processore che uno degli eventi autorizzati dai bit del **Registro** di Abilitazione delle Interruzioni ha generato la richiesta di servizio d'Interruzione; in ordine di priorità essi sono: errore in linea o segnale di break (Receiver Line Status *Interrupt*), dato ricevuto pronto (Received Data Available *Interrupt*), dato trasmesso (Transmit Holding Register Empty *Interrupt*) e variazione segnali **CD**, **RI**, **DSR** o **CTS** (Modem Status *Interrupt*). La linea è riportata a **0** non appena la richiesta è riconosciuta (o in occasione di un reset (**MR**)

- **XIn** (**External Crystal Input**, pin**16**) e **XOut** (**External Crystal Output**, pin**17**): su queste linee è collegato il (*cristallo di*) ***quarzo*** necessario all'UART per sincronizzare il suo funzionamento e per generare le frequenze di ricetrasmissione dei dati seriali; per rendere ottimale il *segnale di **clock***, il quarzo oscilla alla frequenza di **1,84320** MHz; se viene usato un generatore di ***clock esterno*** esso sarà collegato alla linea **XIn** mentre la linea **XOut** rimarrà inutilizzata
- **BaudOut** (pin**15**): su questa linea (**Baud Output**) è disponibile il segnale di ***clock*** generato internamente dal *Programmable Baud Rate Generator* per i circuiti di temporizzazione del ***trasmettitore*** (Transmitter Clock), ma non per quelli del ricevitore, per i quali è necessario un collegamento esterno, vedi linea **RClk**); la frequenza di questo segnale è ottenuta a partire da quella d'oscillazione del *quarzo* divisa per la costante programmata nei 2 **Registri** Divisori di velocità; la velocità di trasmissione effettiva (*Baud Rate*) è poi ottenuta dividendo questa *per 16* , con l'aiuto di un *prescaler* (*divisore*) interno [se la costante fosse programmata a 1 si potrebbe disporre della *velocità massima* per questo tipo di **UART**, pari a 1843200/**16** = **115200** Hz]
- **RClk** (pin**9**): questa linea (**Receiver Clock Input**) è normalmente collegata alla linea **BaudOut** per fornire il segnale di ***clock*** ai circuiti di temporizzazione del ***ricevitore*** (Receiver Clock); la frequenza del segnale in ingresso è dunque *16 volte più grande* della velocità di ricezione effettiva
- **MR** (pin**35**): questa linea (*attiva alta*, **Master Reset**) è posta a **1** per ***reinizializzare*** la ***logica di controllo***: i segnali d'uscita vengono resi *non attivi* cioè **INTRPT** a **0** e **OUT1**, **OUT2**, **RTS** e **DTR** a **1** come **SOut**, per ribadire che non c'è dato sulla linea); inoltre sono ***azzerati*** entrambi i ***buffer FIFO*** e molti ***registri*** dell'UART, ad eccezione di quelli di dato (*Receiver Buffer* e *Transmitter Holding*) e divisori di frequenza (*Divisor Latches*) che mantengono inalterato il valore presente prima dell'attivazione di questo segnale. Questo ingresso è bufferizzato da una logica *TTL trigger di Schmitt*
- **WR** (**DOStr**, pin**18**) e **WR** (**DOStr**, pin**19**): questo segnale (**Write**, **Data Output Strobe**), disponibile in forma *attiva bassa* e *attiva alta*, abilita il processore a **scrivere** dati o parole di controllo nei registri interni dell'UART, se esso è abilitato; poichè i segnali *sono alternativi* (non possono essere attivi contemporaneamente) uno dei 2 deve essere *non attivo*
- **RD** (**DIStr**, pin**21**) e **RD** (**DIStr**, pin**22**): questo segnale (**Read**, **Data Input Strobe**), disponibile in forma *attiva bassa* e *attiva alta*, abilita il processore a **leggere** dati o parole di stato dai registri interni dell'UART, se esso è abilitato; poichè i segnali *sono alternativi* (non possono essere attivi contemporaneamente) uno dei 2 deve essere *non attivo*
- **DDis** (pin**23**): questa linea (*attiva alta*, **Driver Disable**) è posta a **1** quando il processore **legge** dati o parole di stato dall'UART; può essere usata per controllare la *direzione* del flusso dei dati sul *data bus transceiver*
- **TxRdy** (**Transmit Ready**, pin**24**) e **RxRdy** (**Receive Ready**, pin**29**): queste linee, come anticipato, sono le uniche non disponibili sull'UART **8250** originale; nelle versioni successive ad esse è affidato il compito di permettere l'uso del DMA (*Direct Memory Access*) e possono operare in 2 modi:
  - il **Mode0** (detto ***16450 mode***) è selezionato quando i buffer FIFO sono disabilitati con l'aiuto di bit**0** del **Registro** di Controllo FIFO o quando i buffer FIFO sono abilitati ma il bit**3** (*DMA Mode Select*) dello stesso **Registro** di Controllo FIFO è forzato a **0**; il **Mode0** supporta il DMA a trasferimento singolo (*single transfer DMA*), operato tra i cicli di bus del processore; il segnale **RxRdy** andrà basso (*attivo*) quando è presente almeno un byte nel Registro di Ricezione e tornerà alto (*inattivo*) quando il medesimo Registro risulta azzerato; il segnale **TxRdy** andrà basso (*attivo*) quando il Registro di Trasmissione non contiene dati e tornerà alto (*inattivo*) quando nel medesimo Registro è presente almeno un byte
  - il **Mode1** (detto ***FIFO mode***) è selezionato quando i buffer FIFO e il *DMA Mode* sono attivi, cioè quando i bit**0** e bit**3** del **Registro** di Controllo FIFO sono entrambi a **1**; il **Mode1** supporta il DMA a trasferimento multiplo (*Multi-transfer DMA*), operato continuamente fino a quando i FIFO sono rispettivamente vuoto (Ricevitore) o pieno (Trasmettitore) ; il segnale **RxRdy** andrà basso (*attivo*) quando si raggiunge il *trigger level* o quando si ha un *TimeOut* e tornerà alto (*inattivo*) quando il Buffer FIFO del Ricevitore è completamente vuoto; il segnale **TxRdy** andrà basso (*attivo*) quando il Buffer FIFO del Trasmettitore non contiene dati e tornerà alto (*inattivo*) quando esso è completamente pieno

3 - Caratteristiche UART - UART: Caratteristiche tecniche

3.4 Descrizione piedini [*seconda parte*] - Segnali **UART**: linee di **dato** e di **controllo modem**

- In particolare i segnali coinvolti nella gestione della Comunicazione Seriale RS232 sono affidati ai seguenti piedini.
  - **SIn** (**RxD**, pin**10**): questa linea (**Serial Input**, **Receive Data**) riceve il dato direttamente dal dispositivo remoto (*modem* o altro) collegato alla linea di comunicazione; in *LoopBack Mode* questa linea viene *scollegata*
  - **SOut** (**TxD**, pin**11**): questa linea (**Serial Output**, **Transmit Data**) trasmette il dato direttamente verso il dispositivo (*modem* o altro) collegato alla linea di comunicazione; in assenza di trasmissione (o dopo un reset, **MR**, o in *LoopBack Mode*) questa linea è forzata a **1** logico

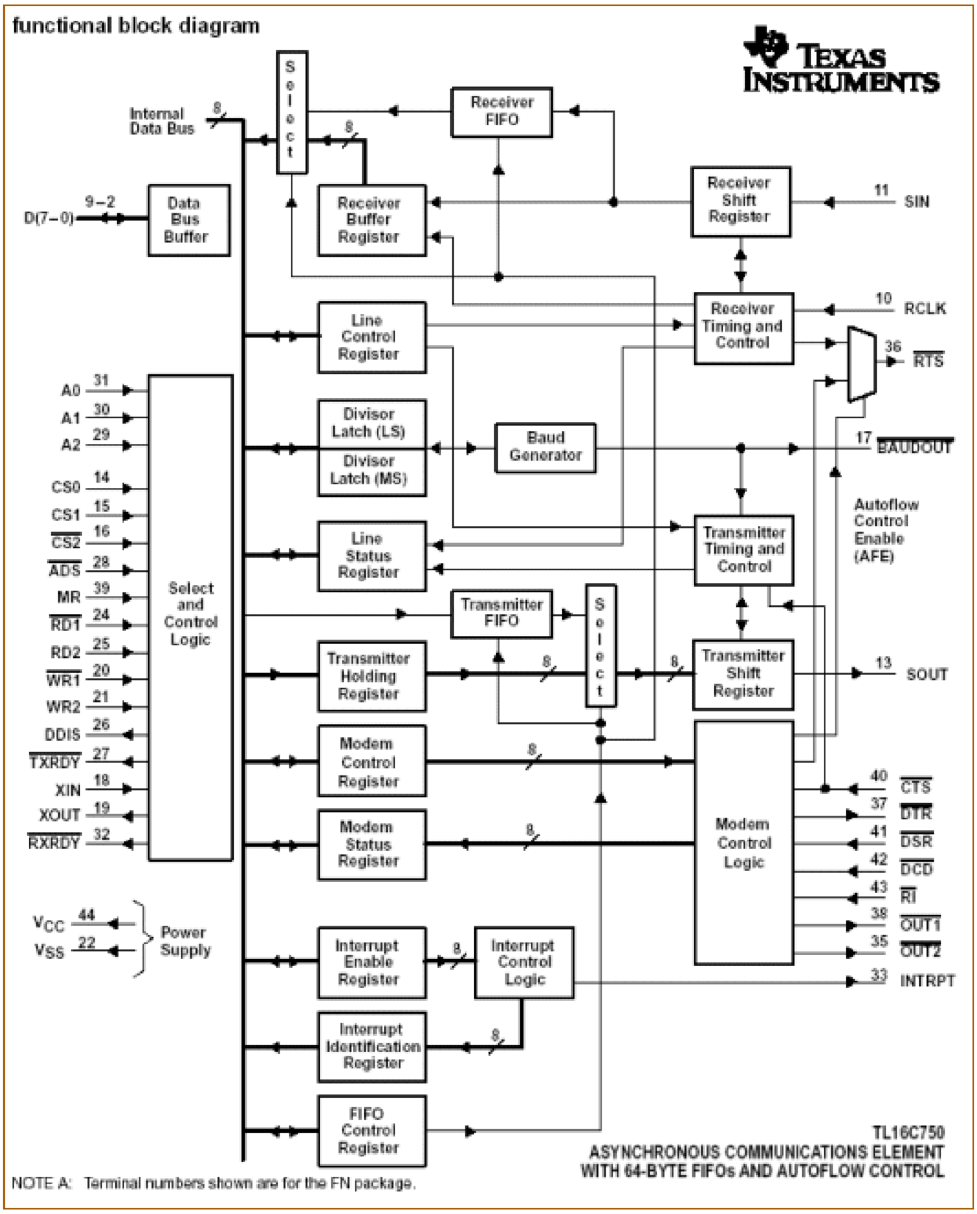
- **OUT1** (pin**34**): questa linea (*uscita ausiliaria* di *uso generale*, *attiva bassa*, **User Output 1**) è una linea programmabile dall'utente (non utilizzata nei PC IBM e compatibili) che può servire per controllare eventuali azioni non correlate con la comunicazione seriale; può essere *attivata* forzando a **1** il bit**2** del **Registro di Controllo Modem**; dopo un reset (**MR**) e in *LoopBack Mode* questa linea è forzata a livello logico **1** (*non attiva*)
- **OUT2** (pin**31**): questa linea (*uscita ausiliaria* di *uso generale*, *attiva bassa*, **User Output 2**) è una linea programmabile dall'utente; può essere *attivata* forzando a **1** il bit**3** del **Registro** di Controllo Modem; nell'ambito dei PC IBM e compatibili è usata per controllare (*enable*) l'uscita di un *buffer tri-state* che collega la linea **INTRPT** alla linea **IRQ3**/**IRQ4** del *controller delle interruzioni*; dopo un reset (**MR**) e in *LoopBack Mode* questa linea è forzata a livello logico **1** (*non attiva*)
- **RTS** (pin**32**): questa linea (*uscita*, *attiva bassa*, **Request To Send**) è posta a **0** dall'UART per *avvisare* il dispositivo remoto (*modem* o altro) che *dispone di dati* ed è *pronto a trasmetterglieli* (letteralmente, per *chiedergli di collegarsi*); in risposta il dispositivo ***attiverà*** il segnale **CTS** per segnalare all'UART la sua disponibilità a ricevere; questa linea può essere *attivata* forzando a **1** il bit**1** del **Registro** di Controllo Modem; dopo un reset (**MR**) e in *LoopBack Mode* questa linea è forzata a livello logico **1** (*non attiva*)
- **DTR** (pin**33**): questa linea (*uscita*, *attiva bassa*, **Data Terminal Ready**) è posta a **0** dall'UART per *avvisare* il dispositivo remoto (*modem* o altro) che è *regolarmente collegato alla linea di comunicazione* ed è *pronto a trasmettere o ricevere dati*; in risposta il dispositivo ***attiverà*** il segnale **DSR** per segnalare all'UART la sua presenza sulla *linea di comunicazione*; questa linea può essere *attivata* forzando a **1** il bit**0** del **Registro di Controllo Modem**; dopo un reset (**MR**) e in *LoopBack Mode* questa linea è forzata a livello logico **1** (*non attiva*)
- **CTS** (pin**36**): se questa linea (*ingresso*, *attiva bassa*, **Clear To Send**) è trovata a **0** il dispositivo remoto (*modem* o altro) *avvisa* l'UART che è *pronto a ricevere* i suoi *dati*, cioè per informarlo che *la trasmissione può cominciare* (o *continuare*); questa linea può essere *testata* leggendo i bit**4** (*valore*) e bit**0** (*variazione* dopo l'ultima lettura) del **Registro di Stato del Modem**; in *LoopBack Mode* questa linea è *scollegata*
- **DSR** (pin**37**): se questa linea (*ingresso*, *attiva bassa*, **Data Set Ready**) è trovata a **0** il dispositivo remoto (*modem* o altro) *avvisa* l'UART che è *regolarmente collegato alla linea di comunicazione* ed è *pronto a concorrere sul canale di comunicazione* (cioè a trasmettere o ricevere dati); questa linea può essere *testata* leggendo i bit**5** (*valore*) e bit**1** (*variazione* dopo l'ultima lettura) del **Registro** di Stato del Modem; in *LoopBack Mode* questa linea è *scollegata*
- **RI** (pin**39**): se questa linea (*ingresso*, *attiva bassa*, **Ring Indicator**) è trovata a **0** il dispositivo remoto (*modem* o altro) *avvisa* l'UART che sta ricevendo un *segnale acustico* di chiamata; questa linea può essere *testata* leggendo i bit**6** (*valore*) e bit**2** (*variazione* dopo l'ultima lettura) del **Registro** di Stato del Modem; in *LoopBack Mode* questa linea è *scollegata*
- **DCD** (pin**38**): se questa linea (*ingresso*, *attiva bassa*, **Data Carrier Detect**) è trovata a **0** il dispositivo remoto (*modem* o altro) *avvisa* l'UART che ha rilevato la presenza in linea della *portante* (*carrier*) sulla quale l'UART ha *modulato* i suoi dati, sinonimo di *segnale di buona qualità* e di canale di comunicazione *pulito, poco rumoroso*; questa linea può essere *testata* leggendo i bit**7** (*valore*) e bit**3** (*variazione* dopo l'ultima lettura) del **Registro** di Stato del Modem; in *LoopBack Mode* questa linea è *scollegata*

3 - Caratteristiche UART - UART: Caratteristiche tecniche

3.5 Schema a blocchi e Link a data Sheet

- La proposta dello **schema a blocchi** di un **UART** è nella logica delle pagine precedenti: probabilmente non è strettamente necessaria per saper usare correttamente questo componente, ma dopo la *meticolosa descrizione* dei pin del componente può aiutarti a capire la sua filosofia...

[vedi *pagina seguente*]



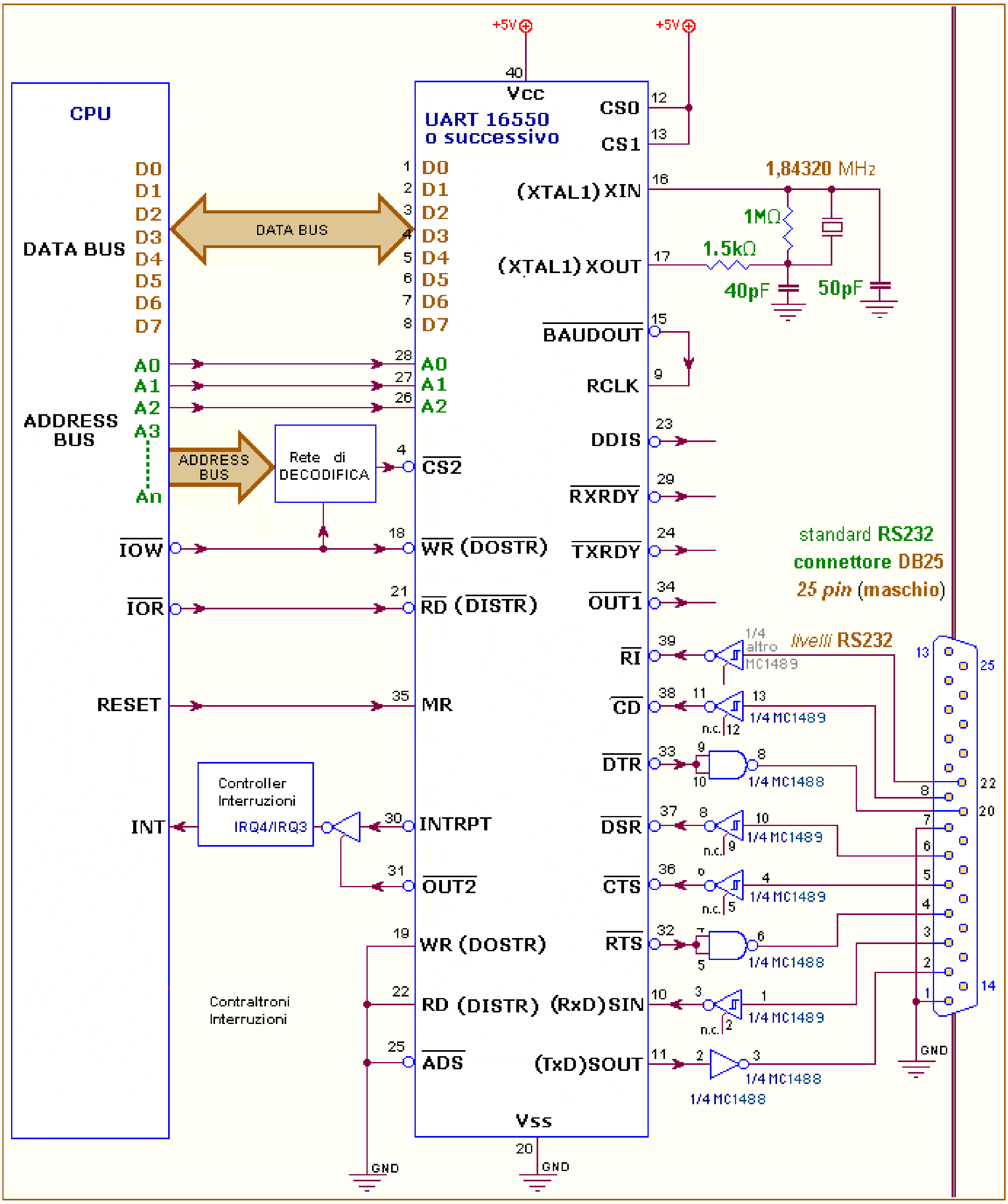
I seguenti **link** rendono disponibili i **Data Sheet originali**, in **formato PDF** dei moderni **UART**, dal **16550** in poi; la loro consultazione può fornirti ulteriori informazioni sulle ***temporizzazioni*** e i ***ritardi tipici*** di questo componente

16550 / 16650	
Texas Instruments	http://dec.bournemouth.ac.uk/staff/awatson/micro/datasheets/TL16C650Bpdf.pdf
National Semiconductor	http://www.national.com/ds/PC/PC16550D.pdf
CAST	http://www.altera.com/products/ip/ampp/cast/documents/m-cas-h16550.pdf
Synopsys	http://www.synopsys.com/products/designware/docs/doc/dwf/datasheets/dw_16550.pdf
16750	
Texas Instruments	http://manuales.elo.utfsm.cl/datasheet/texas_instruments/analog_design/data/pdf/slls191c.pdf http://dec.bournemouth.ac.uk/staff/awatson/micro/datasheets/TL16C750.pdf
Philips	http://www.semiconductors.philips.com/acrobat/literature/9397/75010337.pdf
16950	
Oxford Semiconductor	http://gaia.ecs.csus.edu/~ghansahi/classes/notes/185notes/NS16950UART.pdf http://www.oxsemi.co.uk/download/standard/dsheets/ox16c950bds.pdf http://www.oxsemi.com/products/uarts/ox16c954/ox16c954ds.pdf

3 - Caratteristiche UART - UART: Caratteristiche tecniche

3.6 Schema d'applicazione tra processore e UART

- Per finire desidero proporti lo schema di una **probabile interfaccia** tra un generico processore e un **UART**, con i dettagli sui segnali coinvolti da una parte e dall'altra.



Dentro il Sistema	Porta Seriale

1 - Dentro il Sistema - Indirizzi Base

1.1 Collezione di Indirizzi Base per le Porte Seriali

- Di solito sulla scheda madre sono presenti 2 *porte seriali*, ma con poca spesa è possibile acquistare delle ***schede di espansione*** da inserire sugli *slot* liberi del computer.
- Il *Sistema Operativo* e i *nostri programmi* sono sempre in grado di conoscere il numero di *porta seriali* presenti nel sistema, consultando le *variabili di sistema* caricate dal BIOS nelle fasi preliminari che seguono l'accensione del computer.

> Non appena il nostro computer viene acceso il processore è obbligato a *saltare alla locazione* **FFFF0H** posto nella memoria **BIOS** ospitata dalla scheda madre; da questo indirizzo mette in esecuzione la procedura di **POST** (**P**ower-**O**n-**S**elf-**T**est). Il compito di questa procedura è quello di *verificare lo stato del computer*, ispezionando con meticolosità ogni dispositivo disponibile (tra cui le eventuali *porte seriali*) e annotando tutto in una **zona RAM riservata** di grande importanza, detta **Area di Comunicazione BIOS**, di solito allocata nelle prime 260 (circa) locazioni della Ram convenzionale, certamente *a partire dall'indirizzo* **00400H** .
> Tra l'altro la procedura **POST** provvede a scrivere il valore 1234H nella locazione 00472H (detta **Reset State Flag**) per evitare di ripetere l'indagine in caso di *ripartenza a caldo* (***warm boot***), cioè dopo un *reset software* prodotto dalla pressione simultanea dei tasti **Ctrl-Alt-Del**

- Naturalmente non c'è niente di magico: la **procedura POST** riconosce la presenza dei circuiti di una *porta seriale* semplicemente cercando di scrivere nei suoi **Registri**; è quindi necessario conoscere l'*indirizzo di I/O* di ciascuna di esse.
- Il Sistema riserva, per le *porte seriali*, **4** intervalli di 8 **indirizzi di I/O** ciascuno:

03F8H ÷ 03FFH	Porta Seriale n°1
02F8H ÷ 02FFH	Porta Seriale n°2
0378H ÷ 037FH	Porta Seriale n°3
02E8H ÷ 02EFH	Porta Seriale n°4

- La **procedura POST** verifica la presenza delle *porte seriali* applicando *in stretta sequenza* i 4 intervalli di **indirizzi di I/O** della tabella; al primo gruppo trovato viene associato il numero **0**, al secondo il numero **1**, e così via.
- La prova consiste nello ***scrivere*** il byte 0AAH nel **primo indirizzo** di ciascuna serie, ***leggendolo*** *di ritorno* dal medesimo indirizzo; naturalmente solo se il dato ricevuto è ancora 0AAH la porta è presente nel sistema.
- Da notare che la particolare scelta rende minime le probabilità d'errore: 0AAH è infatti **10101010** in binario, una alternanza di bit altamente improbabile.
- In seguito il DOS riconosce la porta 0 come **COM1**, la 1 come **COM2**, e così via...
- Oggi la *porta seriale* è, di norma, integrata nella scheda madre; poiché di solito sono ne presenti almeno 2, l'indagine sul primo gruppo di indirizzi (**03F8H ÷ 03FFH**) darà certamente esito positivo e sarà associato alla porta **0** (per il BIOS) o **COM1** (per il DOS).
- In conclusione possiamo ritenere che l'**indirizzo Base** (il primo della serie) sia **03F8H** per la porta**0** (**COM1**), **02F8H** per la porta**1** (**COM12**), **03E8H** per la porta**2** (**COM3**) e **02E8H** per la porta**3** (**COM14**).

Messa a Punto	Porta Seriale

1 - Messa a Punto - Descrizione dei Registri UART

1.1 Premesse e Generalità

- Questa parte completa le ***caratteristiche tecniche*** di un **UART**, descrivendo i suoi **Registri interni** in modo assolutamente unico.
- Gli **UART** originali **8250** avevano **10 Registri** associati a 7 indirizzi; la cosa era possibile per la *multifunzione* dei primi 2 registri, subordinata (come vedremo) al valore di un bit (detto DLAB) del **Registro** di Controllo Linea [**port_B**].

- Alla seconda serie degli **UART** (8250A) e a quelli compatibili (come il 16450 e il 16550) fu aggiunto un **undicesimo** Registro, caratterizzato da un ottavo indirizzo, Registro di Scratch [**port_F**].

Per evitare di appesantire le descrizioni, nelle pagine seguenti ***eviterò*** di citare *per esteso* gli **indirizzi possibili** per ciascun possibile gruppo di **Registri**, sostituendo espressioni del tipo **03F8H/02F8H/03E8H/02E8H** (riferite per esempio al **Registro Base** di ogni gruppo) con il riferimento sintetico [**port_8**].

- Le *porte seriali* più moderne (16550A e le successive) hanno **12 Registri**, per esercitare il controllo sul **buffer FIFO**, la speciale novità che *ha fatto la differenza*; eccoli in dettaglio:

03F8H / 02F8H / 03E8H / 02E8H	**out**	Registro di ***Uscita*** Dati	Transmitter Holding Buffer
	in	Registro di ***Ingresso*** Dati	Receiver Buffer
	in/out	Programmazione Divisore	Divisor Latch Low Byte
03F9H / 02F9H / 03E9H / 02E9H	**in/out**	Registro di ***Abilitazione Interruzioni***	Interrupt Enable Register
	in/out	Programmazione Divisore	Divisor Latch High Byte
03FAH / 02FAH / 03EAH / 02EAH	**in**	Registro di ***Riconoscimento Interruzioni***	Interrupt Identification Register
	out	Registro di ***Controllo*** del Buffer FIFO	FIFO Control Register
03FBH / 02FBH / 03EBH / 02EBH	**in/out**	Registro di ***Controllo*** della Linea	Line Control Register
03FCH / 02FCH / 03ECH / 02ECH	**in/out**	Registro di ***Controllo*** del Modem	Modem Control Register
03FDH / 02FDH / 03EDH / 02EDH	**in**	Registro di ***Stato*** della Linea	Line Status Register
03FEH / 02FEH / 03EEH / 02EEH	**in**	Registro di ***Stato*** del Modem	Modem Status Register
03FFH / 02FFH / 03EFH / 02EFH	**in/out**	Registro di scratch	Scratch Register

- E' facile rilevare che la *porta seriale* presenta molti più **Registri** di una porta *parallela standard* **SPP** e che il loro impiego non è da intendere nel senso di poter disporre liberamente di ciascun loro bit per l'Input o per l'Output diretto da o verso una periferica.

- Ciascun **Registro** della *porta seriale* è strettamente legato alle specifiche dello standard **RS232** nella *comunicazione seriale* con protocollo di tipo *asincrono* ed è pensato per la ***messa a punto*** (struttura dei dati, forma e temporizzazioni dei segnali, velocità seriale, ecc), il ***controllo*** (in/out) e la verifica dello ***stato*** (in) di questi dispositivi.
- Sebbene possa sembrare un compito complesso, scrivendo alcune mirate *procedure di servizio* potremo disporre di un potente mezzo di comunicazione con estrema facilità.
- In aggiunta, alcuni modelli di **UART**, detti di **Tipo 3** e destinati ai modelli *IBM PS/2*, dispongono di un ulteriore gruppo di registri destinati alla lettura/scrittura diretta dei *circuiti* ***DMA*** del computer.

1 - Messa a Punto - Descrizione dei Registri UART

1.2 03F8H / 02F8H / 03E8H / 02E8H - OUT / IN - Registro di **Uscita** Dati / Registro di **Ingresso** Dati

- Il primo **Registro** dell'**UART** della *porta seriale* (certamente **indirizzo Base** della serie) è utilizzato, in condizioni di normale esercizio, come **Registro Dati**; in realtà costituito da 2 **porte a 8 bit *separate***:
  - il **Registro** di Trasmissione (Transmitter Holding Register), direttamente accessibile *scrivendo* verso l'indirizzo
  - il **Registro** di Ricezione (Receive Buffer Register), direttamente accessibile *leggendo* dal medesimo indirizzo

- Per l'accesso alla **scrittura/lettura** dei **Dati** è necessario che il bit**7** del Registro di Controllo Linea [**port_B**] sia a **0** logico; in caso contrario l'indirizzo punta un terzo registro interno (Registro divisore di Baud Rate), destinato alla predisposizione della *velocità di terminale* da imporre sulla tratta **DTE-DCE**, descritto in una prossima pagina.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_8	03F8H - 02F8H - 03E8H - 02E8H
								THR	Transmitter Holding Register (scrittura)
								RBR	Receive Buffer Register (lettura)
							x		bit0 in uscita o in ingresso
						x			bit1 in uscita o in ingresso
					x				bit2 in uscita o in ingresso
				x					bit3 in uscita o in ingresso
			x						bit4 in uscita o in ingresso
		x							bit5 in uscita o in ingresso
	x								bit6 in uscita o in ingresso
x									bit7 in uscita o in ingresso

- La **Trasmissione Dati** verso la *porta seriale* non è cosa limitata a *questo* **Registro**; sebbene l'*interfaccia **di scrittura** a 8 bit* con il processore sia il suo indirizzo [**port_8**], coinvolto dall'istruzione **OUT**, questo servizio richiede una discreta attività da parte dell'**UART**:
  - il byte viene in realtà trasmesso da un **buffer** non accessibile da programma, un *registro a scorrimento* detto **Transmit Shift Register**, **TSR**, dal quale viene fatto *scorrere fuori* (***serializzato***) direttamente sulla linea di trasmissione **TD**, subito dopo essere stato ***formattato*** in accordo alle *scelte* **RS232** predisposte ***prima*** della trasmissione, inserendogli davanti un bit di start e facendogli seguire l'eventuale bit di parità e il numero desiderato di bit di stop
  - il byte da trasmettere viene automaticamente trasferito dalla locazione corrente del **buffer FIFO** in Trasmissione (con **UART 16550A**) o dal nostro **Transmit Holding Register**, **THR** (con **UART 8250/16450** o con **UART 16550A** se il **FIFO** è disabilitato)
  - nessun nuovo dato può essere accettato (cioè *scritto* nel **THR**) prima che *il precedente* sia stato spedito sulla linea (cioè prima che il **TSR** sia *vuoto*, con **UART 8250/16450**); con **UART 16550A** o successivi la scrittura dati può continuare *in blocco* fino a quando il **buffer FIFO** in Trasmissione è *pieno*, senza attendere il loro effettivo trasferimento in linea
  - per stabilire l'esatto momento per fornire nuovi dati ogni programma può consultare 2 bit del **Registro di Stato della Linea** [**port_D**] (o affidarsi alle tecniche d'interruzione, discusse in altra parte):
    - il **bit5** (detto Transmitter Holding Register Empty, THRE) a **1** logico indica che il **THR** [**port_8**] (o che il **buffer FIFO** in Trasmissione) è *vuoto* cioè che il dato in esso *scritto* in precedenza dal processore è stato trasferito al **TSR**; il Trasmettitore è pronto ad accettare nuovi caratteri da trasmettere e, non appena uno di essi entra nel **THR** (o nel **FIFO**), riporta a **0** logico questo bit
    - il **bit6** (detto Transmitter Shift Register Empty, TSRE) a **1** logico indica che il **TSR** [*interno*] è *vuoto*: il Trasmettitore è inoperoso, in attesa di dati da mettere in linea; in queste condizioni anche il **THR** [**port_8**] (o il **buffer FIFO** in Trasmissione) è ovviamente *vuoto* e, non appena un dato vi entrerà, questo bit torna a **0** logico

- La **Ricezione Dati** dalla *porta seriale* non è cosa limitata a *questo* **Registro**; sebbene l'*interfaccia **di lettura** a 8 bit* con il processore sia il suo indirizzo [**port_8**], coinvolto dall'istruzione **IN**, questo servizio richiede una discreta attività da parte dell'**UART**:
  - il byte viene in realtà ricevuto in un **buffer** non accessibile da programma, un *registro a scorrimento* detto **Receive Shift Register**, **RSR**, nel quale viene fatto *scorrere dentro* (***in serie***, bit dopo bit) direttamente dalla linea di ricezione **RD** e, quando è stato completamente ricevuto, viene *pulito* dai bit di start e di stop e sottoposto a controllo di parità (se è prevista la presenza del bit a ciò delegato)
  - il byte così ottenuto viene automaticamente trasferito nella locazione corrente del **buffer FIFO** in Ricezione (con **UART 16550A**) o nel nostro **Receive Buffer Register**, **RBR** (con **UART 8250/16450** o con **UART 16550A** se il **FIFO** è disabilitato)
  - nessun nuovo dato può essere accettato dalla linea prima che *il precedente* sia stato *letto* dal processore dal **RBR** (cioè prima che il **RSR** sia *vuoto*, con **UART 8250/16450**); con **UART 16550A** o successivi la lettura dati può continuare *in blocco* fino a quando il **buffer FIFO** in Ricezione è *vuoto*, senza attendere la loro effettiva assunzione dalla linea

- per stabilire l'esatto momento in cui il Ricevitore ha nuovi dati da leggere ogni programma può consultare un bit del **Registro** di Stato della Linea [**port_D**] (o affidarsi alle tecniche d'interruzione, discusse in altra parte):
  - il bit**0** (detto Received Data Ready, RDR) a **1** logico indica che un dato è stato completamente ricostruito dal **RSR** e da esso è stato trasferito nel **RBR** [**port_8**] (o nel **buffer FIFO** in Ricezione); questo bit è riportato a **0** logico non appena il processore estrae il dato dal **RBR** (o dal **FIFO**) oppure se il processore azzera il contenuto del **FIFO** in Ricezione.
  - altri bit dello stesso registro possono essere consultati da programma per stabilire se i dati ricevuti sono attendibili o gravati da **errori** *di sovrapposizione* (bit**1**), *di parità* (bit**2**), *di composizione* (bit**3**) o *di break* (bit**4**), tutti descritti nella recensione dedicata al **Registro** di Stato della Linea

1 - Messa a Punto - Descrizione dei Registri UART

1.3 03F8H / 02F8H / 03E8H / 02E8H - OUT / IN - Programmazione Divisore (**Divisor Latch**)
03F9H / 02F9H / 03E9H / 02E9H

- Il primo e il secondo indirizzo della serie prevista per la *porta seriale* sono utilizzati, almeno una volta, per predisporre la *velocità di terminale* da imporre sul collegamento tra essa stessa (*computer* **DTE**) e il *modem* **DCE**.
- Per esercitare questo servizio è necessario che il bit**7** del **Registro** di Controllo Linea [**port_B**] sia a **1** logico; dopo la messa a punto della velocità il valore di questo bit va riportato a **0**, per consentire il regolare accesso ai **Registri** di Ricetrasmissione (*Transmitter Holding Register/Receive Buffer Register*) e di Abilitazione/Disabilitazione Interruzioni (*Interrupt Enable Register*), puntati dagli stessi indirizzi.
- La *velocità massima* per questo tipo di **UART** è dunque di **115,2**kbaud, ideale per *dispositivi* o *modem* ***veloci***, ma non sempre adatta alle effettive necessità; per questa ragione è prevista la possibilità di ridurne il valore con l'aiuto di un *generatore programmabile di velocità* (Programmable Baud Rate Generator) che utilizza il numero (fattore di divisione) memorizzato nei 2 **Registri** (Divisor Latch) che stiamo descrivendo per ***dividere ulteriormente*** la *frequenza base* ed ottenere la desiderata velocità (Baud Rate) per il *flusso di bit* in ingresso e in uscita dall'**UART**.
- Poichè il numero è a **16** bit esso è lasciato in 2 metà in questa coppia di **Registri** a **8** bit; in accordo con la ricorrente *logica* **Lo-Hi** la *parte bassa* del divisore sarà salvata in [**port_8**] e la *parte alta* del divisore sarà salvata in [**port_9**].
- La Tabella mostra i valori consigliati per questo divisore per ottenere diverse *velocità di terminale*:

Baud Rate Divisor Latch				(lettura/scrittura)
Fattore di Divisione		High Byte port_9	Low Byte port_8	Baud Rate
decimale [n]	esadecimale [n]	03F9H - 02F9H 03E9H -02E9H	03F8H - 02F8H 03E8H -02E8H	115200 / n
2304	0900H	09H	00H	50
1536	0600H	06H	00H	75
1047	0417H	04H	17H	110
768	0300H	03H	00H	150
384	0180H	01H	80H	300
192	00C0H	00H	C0H	600
96	0060H	00H	60H	1200
64	0040H	00H	40H	1800
58	003AH	00H	3AH	2000
48	0030H	00H	30H	2400
32	0020H	00H	20H	3600
24	0018H	00H	18H	4800
16	0010H	00H	10H	7200
12	000CH	00H	0CH	9600
6	0006H	00H	06H	19200
3	0003H	00H	03H	38400
2	0002H	00H	02H	57600
1	0001H	00H	01H	115200

- In talune interfacce seriali le *velocità di terminale*, previste dallo standard **RS232** possono essere ottenute anche con (cristalli di) **quarzo** oscillanti a **3,072** MHz, per i quali la *frequenza base di lavoro* dell'**UART** è ora di 3072000/**16** = **192000** Hz (per la presenza del ***prescaler*** interno *divisiore per* **16**); in questo caso i valori da lasciare nei **Registri** [**port_8**] e [**port_9**] sono:

Baud Rate Divisor Latch				(lettura/scrittura)
Fattore di Divisione		High Byte port_9 03F9H - 02F9H 03E9H -02E9H	Low Byte port_8 03F8H - 02F8H 03E8H -02E8H	Baud Rate 192000 / n
decimale [n]	esadecimale [n]			
3840	0F00H	0FH	00H	50
2560	0A00H	0AH	00H	75
1745	06D1H	06H	D1H	110
1280	0500H	05H	00H	150
640	0280H	02H	80H	300
320	0140H	01H	40H	600
160	00A0H	00H	A0H	1200
107	006BH	00H	6BH	1800
96	0060H	00H	60H	2000
80	0050H	00H	50H	2400
53	0035H	00H	35H	3600
40	0028H	00H	28H	4800
27	001BH	00H	1BH	7200
20	0014H	00H	14H	9600
10	000AH	00H	0AH	19200
5	0005H	00H	05H	38400
3	0003H	00H	03H	64000
2	0002H	00H	02H	96000
1	0001H	00H	01H	192000

- La *velocità massima* per questo tipo di **UART** è dunque di **192**kbaud, e non consente la disponibilità delle velocità di **57,6 kbs** e **115,2kbs** assicurate con il quarzo precedente.

- La mancanza di un **buffer FIFO** in Ricezione ha limitato la *velocità massima* ai valori **9,6**kbps (per i primi **UART 8250**) e **19,2**kbps (per gli **UART 16450**), rendendo impossibile l'uso di *modem* ***veloci***.

- L'avvento dei moderni **UART** (**16550A** o successivi) ci consente invece di *programmare* la loro *velocità* per ottenere le massime prestazioni dai vari tipi di *modem*:
  - la *velocità di terminale* **115,2**kbaud è ideale per i *modem* definiti dallo *standard* **V.90** (*velocità di dati* massima di **56**Kbps) o per quelli definiti dallo *standard* **V.34** (*velocità di dati* massima di a **28,8**Kbps)
  - anche con *modem* meno veloci, combinando la loro *velocità dati* massima con la massima *compressione* (**4:1**) prevista dallo *standard* **V.42bis** è possibile programmare elevate *velocità di terminale*: fino a **57,6**kbaud per dei *modem standard* **V.32bis** (a **14,4**Kbps) e fino a **38,4**kbaud per dei *modem standard* **V.32** (a **9,6**Kbps)

- In aggiunta va segnalata la disponibilità di speciali schede con *porte seriali* sulle quali è possibile selezionare quarzi in grado di oscillare a frequenza **4** o **8** volte più veloce di quella standard, di **1,84320** MHz, sempre ammesso di disporre di periferiche in grado di poterne fruire:
  - con frequenza **4** volte più veloce (**7.3728**MHz) è garantita una *velocità base di terminale* fino a **460,8**kbaud
  - con frequenza **8** volte più veloce (**14.7456**MHz) è garantita una *velocità base di terminale* fino a **921,6**kbaud

- Per finire voglio mostrare un breve codice adatto a predisporre la porta **COM1** per funzionare alla velocità di **9600 baud**; ricordo che la tabella prevede per essa un fattore di divisione pari a **12** (=**000C**H):

```
                           ;┐ ┌[port_B]──────────────────────────────────┐
                           ;│ │ Registro di controllo della LINEA        │
                           ;│ └──────────────────────────────────────────┘
  MOV     DX,CS:[port_B];│Inizia la programmazione dell'8250:  per poter
  IN      AL,DX          ;│accedere ai  registri  divisori  di BaudRate è
  OR      AL,10000000B   ;│necessario  anzitutto  porre  a  "1"  il  bit7
  OUT     DX,AL          ;│di [port_B]:  con questa predisposizione  pos-
                           ;│sono ora essere introdotti i  corretti  valori
                           ;│per il divisore di  baud  rate, "Baud Rate-lo"
                           ;┘(su [port_8])  e "Baud Rate-hi"  (su [port_9])

                           ;┐ ┌──────────────────────────────────────────┐
                           ;│ │ AX=costante a partire da [115200/costante]│
                           ;│ └──────────────────────────────────────────┘
                           ;│Prepara in AX (AL=LSB, AH=MSD) la costante  di
                           ;│divisione necessaria per dividere il clock  ad
                           ;│alta frequenza (115200Hz) fornito  all'UART (a
                           ;│partire dalla frequenza di norma applicata, di
                           ;│1.8432 MHz, LOCALMENTE  pre-divisa per 16)  al
                           ;│fine di garantire la  desiderata  velocità  di
                           ;│RICE-TRASMISSIONE in baud (es:  9600  bit/sec)
  MOV     AX,CS:[DepVEL];│Si ricorda che l'UART contiene  un  generatore
                           ;│programmabile di BAUD RATE in grado di tratta-
                           ;│re qualunque frequenza da 0 a 3.1 MHz; normal-
                           ;│mente la  frequenza applicata è di  1.8432 MHz
                           ;│o, talvolta,  quella  due  terzi  più  grande,
                           ;│3.072 MHz), su cui viene operata immediatamen-
                           ;│te una predivisione per 16,  cosicché  la  co-
                           ;│stante descritta andrà a  dividere  effettiva-
                           ;│mente la frequenza di 115200 Hz, generando  la
                           ;┘desiderata frequenza di ricetrasmissione
                           ;┐ ┌[port_8]──────────────────────────────────┐
                           ;│ │ Registro divisore Baud RATE (parte bassa) │
  MOV     DX,CS:[port_8];│ └──────────────────────────────────────────┘
  OUT     DX,AL          ;│Se su [port_B] il bit7="1"  la  porta [port_8]
                           ;┘serve come Registro divisore Baud RATE (low)
                           ;┐ ┌[port_9]──────────────────────────────────┐
  MOV     AL,AH          ;│ │ Registro divisore Baud RATE (parte alta)  │
  MOV     DX,CS:[port_9];│ └──────────────────────────────────────────┘
  OUT     DX,AL          ;│Se su [port_B] il bit7="1"  la  porta [port_9]
                           ;┘serve come Registro divisore  Baud RATE (high)

                           ;┐ ┌[port_B]──────────────────────────────────┐
                           ;│ │ Registro di controllo della LINEA        │
                           ;│ └──────────────────────────────────────────┘
  MOV     DX,CS:[port_B];│Dopo la  programmazione  della  BaudRate  del-
  IN      AL,DX          ;│l'UART per poter accedere ai registri di Input
  AND     AL,01111111B   ;│e Output  (che  hanno  gli  stessi  indirizzi,
  OUT     DX,AL          ;│[port_8] e [port_9]) è necessario  riportare a
                           ;┘0 il bit7 del Registro controllo LINEA[port_B]
```

- Messa a Punto - Descrizione dei Registri UART

1.4 03F9H / 02F9H / 03E9H / 02E9H - OUT / IN - Registro di **Abilitazione Interruzioni**

- Come tutti i dispositivi importanti anche la *porta seriale* può essere controllata con la tecnica delle *Interruzioni hardware*: quando si presenta un particolare evento che richiede l'attenzione del processore l'**UART** provvede a segnalarlo attivando la sua linea d'uscita **INTRPT** (pin**30** dell'**UART**).
- Il secondo **Registro** dell'**UART** è utilizzato, in condizioni di normale esercizio, per **abilitare** o **disabilitare** *individualmente* le possibili *fonti di* ***interrupt***.
- Per modificare il contenuto di questo registro è necessario che il bit**7** del **Registro** di Controllo Linea [**port_B**] sia a **0** logico; in caso contrario l'indirizzo punta un registro interno alternativo (**Registro** divisore di Baud Rate), destinato alla predisposizione della *velocità di terminale* da imporre sulla tratta **DTE-DCE**, descritto nella prossima pagina.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_9	03F9H - 02F9H - 03E9H - 02E9H
								IER	Interrupt Enable Register (lettura/scrittura)
							1		0 = interruzione disabilitata 1 = interruzione abilitata per Dato Ricevuto (Received Data Available) o per FIFO timeout [16550] [interrompe se il Receive Buffer Register, RBR, è *pieno*]
						1			0 = interruzione disabilitata 1 = interruzione abilitata per Dato Trasmesso (Transmit Data Empty) [interrompe se il Transmit Holding Register, THR, è *vuoto*]
					1				0 = interruzione disabilitata 1 = interruzione abilitata per Variazioni dello Stato della Linea (Receiver Line Status) [interrompe se è stato rilevato un errore nel flusso dati o un Break]
				1					0 = interruzione disabilitata 1 = interruzione abilitata per Variazioni dello Stato del Modem (Modem Status) [interrompe se stato rilevato un cambiamento nel handshake]
			0						riservato, non utilizzato, sempre a 0 logico [16750 - interruzione abilitata per Sleep Mode]
		0							riservato, non utilizzato, sempre a 0 logico [16750 - interruzione abilitata per Low Power Mode]
	0								riservato, non utilizzato, sempre a 0 logico
0									riservato, non utilizzato, sempre a 0 logico

- Il valore dei bit riportato in tabella è quello **attivo** (l'interruzione desiderata è abilitata con **1** logico e disabilitata con lo **0**); ufficialmente l'**UART** può richiedere servizio d'interruzione per 4 tipi di eventi; in dettaglio:
  - **Received Data Available**: se il bit**0** è **1** l'**UART** genera *richiesta di interruzione* quando il numero di bytes ricevuti ha superato quello massimo (*trigger level*) previsto per il **buffer FIFO** in Ricezione (si aspetta che il **FIFO** sia *pieno* di dati, per farli leggere *in blocco* dal processore), oppure quando il **FIFO** contiene bytes in misura inferiore al massimo ma il tempo concesso a nuovi arrivi è terminato, *timeout*) oppure quando nel **Receive Buffer Register**, **RBR**, è pronto un singolo dato (con **UART 8250**/**16450** o con **UART 16550A**, se il **FIFO** è disabilitato)
  - **Transmit Data Empty**: se il bit**1** è **1** l'**UART** genera *richiesta di interruzione* quando il **Transmit Holding Register**, **THR**, è *vuoto* (con **UART 8250**/**16450** o con **UART 16550A**, se il **FIFO** in Trasmissione è disabilitato) oppure quando il **FIFO** ha posti liberi per uno o più bytes
  - **Receiver Line Status**: se il bit**2** è **1** l'**UART** genera *richiesta di interruzione* quando, durante la ricezione di dati, ha rilevato la presenza di **errori** (*di sovrapposizione*, *di parità* o *di composizione*) o la presenza di un **segnale** *di break*, tutti eventi segnalati da bit del **Registro di Stato della Linea** [**port_D**]
  - **Modem Status**: se il bit**3** è a **1** l'**UART** genera *richiesta di interruzione* quando, durante la comunicazione con il *Modem*, ha rilevato la variazione dei segnali in arrivo, direzione **DCE>DTE**, come Data Carrier Detect (**CD**, rilevato modem remoto/possibile comunicare), Ring Indicator (**RI**, ricevuto segnale acustico sul canale), Data Set Ready (**DSR**, DCE connesso e pronto a comunicare) e Clear To Send (**CTS**, DCE Pronto a ricevere), tutti eventi segnalati da bit del **Registro di Stato del Modem** [**port_E**]
- Non va dimenticato che la ***tecnica corretta*** per gestire questo registro è quella di modificare **solo** sui bit desiderati, evitando di influenzare gli altri; per questa ragione: ***prima*** è necessario **leggerlo**, intervenendo sul valore binario ottenuto con delle maschere **OR** (per mettere a **1** i bit, cioè *abilitare* la relativa interruzione) o **AND** (per mettere a **0** i bit, cioè *disabilitare* la relativa interruzione); **solo *dopo*** questa operazione il registro dovrà essere **scritto**. Un esempio di corretta gestione può essere il seguente:

```
                                ;┐ ┌─[port_9]──────────────────────────┐
                                ;│ │ Registro di abilitazione delle INTERRUZIONI│
                                ;│ └──────────────────────────────────┘
                                ;│Vengono specificati solo gli eventi che si de-
                                ;│sidera utilizzare per innescare il  meccanismo
                                ;│dell'interruzione, in grado di attivare la li-
        MOV     DX,CS:[port_9]; │nea IRQ3/IRQ4 dedicata  al  nostro  UART 8250,
        IN      AL,DX           ;│avvisando così il controllore  delle  interru-
        OR      AL,00000011B    ;│zioni 8259 che farà partire la relativa proce-
        OUT     DX,AL           ;│dura di servizio BIOS, INT 0BH o  INT 0CH;  se
                                ;│NON SI DESIDERA far riferimento alle  tecniche
                                ;│d'interruzione tutti i bit di questo  registro
                                ;│sono lasciati a 0; nell'esempio  si abilitano:
                                ;│1) bit0="1", abilita  l'interrupt  per  DATO
                                ;│             PRONTO in RICEZIONE (DATO PRONTO)
                                ;│2) bit1="1", abilita l'interrupt per  registro
                                ;┘           TRASMISSIONE VUOTO. DATO TRASMESSO
```

4 - Messa a Punto - Descrizione dei Registri UART

1.5 03FAH / 02FAH / 03EAH / 02EAH - IN - Registro di Riconoscimento Interruzioni

- Il terzo **Registro** dell'**UART** della *porta seriale* è utilizzato in tutte le versioni come **Registro** di **Identificazione** delle **Interruzioni** [i moderni **UART 16550A** (e successivi) utilizzano lo stesso indirizzo (*in scrittura*) anche per il controllo del **buffer FIFO**, ma di questo parleremo nella pagina successiva].
- Si tratta di un registro *a sola lettura* indispensabile per sapere se il processore ha attivato una *procedura di servizio*, in risposta alla *richiesta di interruzione* generata dalla *porta seriale* (**UART**), specificando contemporaneamente la causa che l'ha causata, certamente una delle 4 già coinvolte nella descrizione del **Registro** di Abilitazione delle Interruzioni [**port_9**].
- Tutte le versioni utilizzano per questo scopo *solo* i 3 bit meno significativi; gli **UART 16550A** (e successivi) utilizzano *anche* i 3 bit più significativi, per il controllo del loro prezioso **buffer FIFO**.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_A	03FAH - 02FAH - 03EAH - 02EAH
								IIR	**Interrupt Identification Register (lettura)**
							0		0 = interruzione pendente: un evento ha richiesto servizio d'interruzione 1 = nessuna interruzione pendente
					x	x			Indica (Interrupt ID) quale degli eventi interrompenti è attualmente in attesa di servizio: 11 = Receiver Line Status *Interrupt*, priorità 1, massima, errore in linea o break 10 = Received Data Available *Interrupt*, priorità 2, dato ricevuto pronto 01 = Transmit Holding Register Empty *Interrupt*, priorità 3, dato trasmesso 00 = Modem Status *Interrupt*, priorità 4, minima, variazione segnali CD, RI, DSR o CTS
				0/1					0 = riservato, non utilizzato [8250/16450] 1 = TimeOut *Interrupt* Pending [16550A e successivi], insieme a bit2/bit1=10=priorità 2 1 = pochi bytes nel buffer FIFO e tempo scaduto per nuovi arrivi (*timeout*)
			0						riservato, non utilizzato, sempre a 0 logico
		0/1							0 = riservato, non utilizzato [8250/16450/16550A] 0 = buffer FIFO non abilitato = simulazione 8250 [solo 16750] 1 = buffer FIFO (a 64 bit) abilitato al funzionamento [solo 16750]
	0/1								0 = riservato, non utilizzato [8250/16450] 0 = buffer FIFO inutilizzabile, se bit7=1=FIFO abilitato [16550A e successivi] 1 = buffer FIFO utilizzabile, se bit7=1=FIFO abilitato [16550A e successivi]
0/1									0 = riservato, non utilizzato [8250/16450] 0 = buffer FIFO non abilitato = simulazione 8250 [16550A e successivi] 1 = buffer FIFO abilitato al funzionamento [16550A e successivi]

- Come anticipato sono 2 le zone significative della tabella; quella comune a tutti le versioni di **UART** (i **3** bit *meno significativi*, l'unica presente nelle prime versioni **8250**/**16450**) aiuta a documentare la presenza di potenziali *richieste di interruzione*.
- Poichè gli eventi che le possono generare sono **4** e poichè la richiesta può pervenire contemporaneamente anche da **più di una**, è prevista una codifica che associa loro una **priorità**, cioè un numero che stabilisce quale di esse sarà servita per prima.

  - **Interrupt Pending**: se il bit**0** è **0** significa che un'*interruzione* è ***pendente***, cioè uno o più eventi hanno indotto l'**UART** ha generare una *richiesta* che non è stata ancora stata *servita* (o *servita del tutto*) dal processore (per sapere quale evento è attualmente sottoposto all'attenzione della CPU basta interpretare i 2 bit successivi); questo bit può essere interrogato (***polling***) da software per scoprire se tutte le potenziali richieste sono state soddisfatte

  - **Interrupt ID**: il compito di stabilire quale dei possibili 4 eventi è ***in attesa*** *di essere servito* è affidato ai bit**2**/bit**1**: è prevista una codifica per ciascuno di essi che consente anche di dirimere l'*ordine* con cui dovranno essere serviti, nel caso di 2 o più *richieste contemporanee*: la coppia di bit presente di volta in volta nel registro è quella dell'evento a più alta priorità e, dopo il suo completo servizio, sarà sostituita (dall'**UART**) con quella dell'evento con la successiva più alta prioritaria. Ecco le codifiche, in dettaglio:

    - bit**2**/bit**1** = **11**: durante la ricezione di dati è stata rilevata la presenza di **errori** (*di sovrapposizione*, *di parità* o *di composizione*) o la presenza di un **segnale** *di break*; l'**UART** ha generato una *richiesta di* **Receiver Line Status** *Interrupt*, di priorità massima (prima), ed è in attesa di essere servito; la lettura del **Registro** di Stato della Linea [**port_D**] resetta questa segnalazione

- bit2/bit1 = 10: il numero di bytes ricevuti ha superato quello massimo (*trigger level*) previsto per il FIFO in Ricezione oppure il FIFO contiene bytes in misura inferiore al massimo ma il tempo concesso a nuovi arrivi è terminato (*timeout*) oppure nel Receive Buffer Register è pronto un singolo dato; l'UART ha generato una *richiesta di* **Received Data Available** *Interrupt*, di seconda priorità, ed è in attesa di essere servito; la lettura del Registro di Ricezione Dati [port_8] resetta questa segnalazione
- bit2/bit1 = 01: il Transmit Holding Register è *vuoto* (con UART 8250/16450 o con UART 16550A, se il FIFO in Trasmissione è disabilitato) oppure il FIFO ha posti liberi per uno o più bytes; l'UART ha generato una *richiesta di* **Transmit Holding Register Empty** *Interrupt*, di terza priorità, ed è in attesa di essere servito; la scrittura del Registro di Trasmissione Dati [port_8] resetta questa segnalazione
- bit2/bit1 = 00: durante la comunicazione con il *Modem* è stata rilevata la variazione dei segnali in arrivo, direzione DCE>DTE, come Data Carrier Detect (CD, rilevato modem remoto/possibile comunicare), Ring Indicator (RI, ricevuto segnale acustico sul canale), Data Set Ready (DSR, DCE connesso e pronto a comunicare) e Clear To Send (CTS, DCE Pronto a ricevere); l'UART ha generato una *richiesta di* **Modem Status** *Interrupt*, di priorità più bassa (quarta), ed è in attesa di essere servito; la lettura del il Registro di Stato del Modem [port_E] resetta questa segnalazione

- La parte rimanente del Registro (i 5 bit *più significativi*) non era *non utilizzata* nelle versioni 8250/16450; come spesso accade i progettisti si sono riservati l'uso di questi bit per il futuro, consentendo agli UART 16550A (e successivi) di utilizzarli per segnalare eventi legati alla presenza del loro buffer FIFO:
  - **TimeOut Interrupt Pending**: il bit3 è posto a 1 dagli UART 16550A (e successivi) per segnalare che il loro FIFO contiene bytes in misura inferiore al massimo (*trigger level*) previsto e che ***il tempo concesso*** per riempirlo completamente con nuovi arrivi ***è terminato*** (*timeout*); questo evento pone anche a 10 la coppia bit2/bit1, per segnalare che è in attesa di essere servita una particolare *richiesta di* **Received Data Available** *Interrupt*
  - Reserved: il bit4 è sempre a 0 (non è usato da nessuna versione di UART)
  - **FIFO enabled**: il bit5 è posto a 1 (solo) dagli UART 16750 se il loro FIFO è abilitato al funzionamento; altrimenti, con bit5 a 0, segnalano di essere in simulazione 8250
  - **FIFO available**: il bit6 è posto a 1 dagli UART 16550A (e successivi) per segnalare che il loro FIFO è utilizzabile (altrimenti bit6=0); naturalmente è significativo solo se il bit7=1=FIFO abilitato
  - **FIFO enabled**: il bit7 è posto a 1 dagli UART 16550A (e successivi) per segnalare che il loro FIFO è abilitato al funzionamento; altrimenti, con bit7 a 0, segnalano di essere in simulazione 8250

- Il breve codice seguente suggerisce un modo intelligente per sfruttare il contenuto di questo registro per aiutare la procedura di servizio dell'*interruzione hardware* IRQ4 (generata dall'UART della *porta seriale* COM1) a servire automaticamente l'evento che l'ha provocata

- La tecnica si basa sul fatto che il contenuto, se è ***pendente*** un'*interruzione* (bit0=0) e se i 5 bit *più significativi* sono tutti a 0, è un numero pari da 0 (00000000 binario) a 6 (00000110 binario), condizione ideale per scorrere una Tabella (TabServ) contenente la sequenza degli indirizzi (*word*) della parte dedicata (dalla procedura di servizio) specificatamente ad ogni evento:

  - TabServ+0 = *servizio* per **Modem Status** *Interrupt*, variazione segnali CD, RI, DSR o CTS
  - TabServ+2 = *servizio* per **Transmit Holding Register Empty** *Interrupt*, dato trasmesso
  - TabServ+4 = *servizio* per **Received Data Available** *Interrupt*, dato ricevuto pronto
  - TabServ+6 = *servizio* per **Receiver Line Status** *Interrupt*, errore in linea o segnale di break

```
                                   ;┌──────────────────────────────────────────┐
                                   ;│Tabella Puntatori alle Procedure di Servizio│
                                   ;└──────────────────────────────────────────┘
TabServ DW   SrvModem              ;Modem Status Interrupt, priorità 4, minima
                                   ;[variazione segnali CD, RI, DSR o CTS]
        DW   SrvTxD                ;Transmit Register Empty Interrupt, priorità 3
                                   ;[dato trasmesso, pronto per il prossimo]
        DW   SrvRxD                ;Received Data Available Interrupt, priorità 2
                                   ;[dato ricevuto, pronto da leggere]
        DW   SrvStato              ;Receiver Line Status Interrupt, priorità 1
                                   ;[errore in linea o presenza di segnale break]
;-----------------------------------------------------------------------------
port_A  DW   03FAH                 ;Registro di Riconoscimento Interruzioni, COM1
;-----------------------------------------------------------------------------
```

```
                                   ;┐ ┌─[port_A]──────────────────────────────┐
                                   ;│ │ Registro di Riconoscimento Interruzioni │
                                   ;│ └─────────────────────────────────────────┘
        MOV     DX,CS:[port_A];│Se l'UART  ha  attivato la  linea di interrupt
        IN      AL,DX          ;│IRQ4 (tipica per COM1, coinvolta nell'esempio)
        AND     AL,00000111B   ;│certamente il bit0 di [port_A] è a "0" e i bit
        MOV     AH,00H         ;│2  e  1 portano il codice dell'evento che deve
                               ;│essere servito per primo, tra quelli che gene-
                               ;│rato la  richiesta  di  interruzione:
                               ;│AL=00000000 = offset x Modem Status  Interrupt
                               ;│AL=00000010 = offset x Transmit Hold.Reg.Empty
                               ;│AL=00000100 = offset x Received Data Available
                               ;│AL=00000110 = offset x Receiver Line Status
                               ;┘In AX è dunque pronto  l'offset per la tabella
                               ;┐Il numero AX dell'offset aggiunto all'indiriz-
        LEA     SI,TabServ     ;│zo Base della Tabella, essendo pari,  consente
        ADD     AX,SI          ;│di scorrerla per localizzare il puntatore alla
        JMP     AX             ;│procedura di servizio associata all'evento che
                               ;┘ha richiesto l'interruzione... e di "saltarci"
;-----------------------------------------------------------------------------
SrvModem: ...........          ;Procedura servizio Modem Status Interrupt
SrvTxD:   ...........          ;Procedura servizio Transmit Register Empty Int
SrvRxD:   ...........          ;Procedura servizio Received Data Available Int
SrvStato: ...........          ;Procedura servizio Receiver Line Status Intrupt
```

1 - Messa a Punto - Descrizione dei Registri UART

1.6 03FAH / 02FAH / 03EAH / 02EAH - OUT - Registro di Controllo del Buffer FIFO

- I moderni **UART 16550A** (e successivi) utilizzano lo stesso indirizzo del **Registro** di Identificazione delle Interruzioni [**port_A**] per il controllo del loro **buffer FIFO**.
- Si tratta di un registro *a sola scrittura*.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_A / FCR	03FAH - 02FAH - 03EAH - 02EAH / FIFO Control Register (scrittura)
							1		0 = azzera entrambi i buffer FIFO e i *registri a scorrimento* TSR e RSR 1 = FIFO Enable: abilita e azzera entrambi i buffer FIFO
						1			1 = Receiver FIFO Reset: azzera il contenuto del FIFO in Ricezione e forza bit1=0 [il *registro a scorrimento* RSR termina la ricezione del dato in arrivo]
					1				1 = Transmit FIFO Reset: azzera il contenuto del FIFO in Trasmissione e forza bit2=0 [il *registro a scorrimento* TSR termina la trasmissione del dato in uscita]
				1					1 = DMA Mode Select: cambia l'attività dei segnali RXRDY e TXRDY da Mode1 a Mode2 (se bit0=1)
		0	0						riservati, non utilizzati, sempre a 0 logico
x	x								Receiver Trigger: numero di bytes (*trigger level*) che devono essere presenti nel FIFO in Ricezione per generare la *richiesta di* Received Data Available *Interrupt*: 11 = 14 bytes; 10 = 8 bytes; 01 = 4 bytes; 00 = 1 byte

- L'azione associata a ciascun bit di questo registro è ora descritta in dettaglio:
  - **FIFO Enable**: se il bit**0** è posto a **1** è possibile l'uso sia del **buffer FIFO** in Trasmissione che del **buffer FIFO** in Ricezione; ogni byte contenuto in entrambi i buffer e nei relativi *registri a scorrimento* (Transmit Shift Register, **TSR** e Receive Shift Register, **RSR**, descritti nella trattazione del **Registro** di Ricetrasmissione Dati [**port_8**]) viene ***azzerato*** forzando questo bit a **0**; per programmare gli altri bit del registro bit**0** deve essere a **1**
  - **Receiver FIFO Reset**: forzando il bit**1** a **1** il contenuto del **FIFO** in Ricezione viene ***azzerato***; anche il suo contatore logico e il bit**1** stesso sono forzati automaticamente a **0**, mentre il relativo *registro a scorrimento* (Receive Shift Register, **RSR**, porterà a termine la ricezione del dato in arrivo, lasciandolo a disposizione
  - **Transmit FIFO Reset**: forzando il bit**2** a **1** il contenuto del **FIFO** in Trasmissione viene ***azzerato***; anche il suo contatore logico e il bit**2** stesso sono forzati automaticamente a **0**, mentre il il dato in uscita dal relativo *registro a scorrimento* (Transmit Shift Register, **TSR**), sarà spedito regolarmente
  - **DMA Mode Select**: se il bit**3** è posto a **1** (e anche bit**0**=**1**, cioè i FIFO sono abilitati) i segnali presenti sui pin **RXRDY** e **TXRDY** dell'**UART** cambiano modo di funzionare, da Mode0 a Mode1
  - Reserved: i bit**4** e bit**5** sono sempre a **0** (non usati da nessuna versione di **UART**)
  - **Receiver Trigger**: i bit**6** e bit**7** forniscono un codice binario associato al numero di bytes (detto *trigger level*) che devono essere presenti nel **FIFO** in Ricezione per generare la *richiesta di* **Received Data Available** *Interrupt*; in dettaglio: **00**=1 byte, **01**=4 bytes, **10**=8 bytes e **11**=14 bytes

1 - Messa a Punto - Descrizione dei Registri UART

1.7 03FBH / 02FBH / 03EBH / 02EBH - OUT / IN - Registro di Controllo della Linea

- Il quarto **Registro** dell'**UART** della *porta seriale* è utilizzato in tutte le versioni come **Registro** di **Controllo Linea**; in esso sono inizializzati (*scritti*) i ***parametri*** previsti dallo standard **RS232** per il ***frame*** *seriale* nell'ambito della comunicazione seriale *asincrona* (cioè la dimensione dei dati, il numero di bit di stop, numero di bit di parità) e il controllo del bit di break.
- Come abbiamo sottolineato nelle pagine precedenti, il suo bit più significativo consente (a parità di indirizzo) il controllo di registri alternativi; in particolare quando il bit**7** è a **1** logico consente la messa a punto della velocità di terminale, cioè l'accesso al **Registro** divisore di Baud Rate.
- Questo registro può essere anche *letto*, facilitando la modifica dei suoi singoli bit.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_B: 03FBH - 02FBH - 03EBH - 02EBH LCR: Line Control Register (lettura/scrittura)
						x	x	stabiliscono (Word Length) il numero di bit previsti per ogni *carattere* trasmesso o ricevuto: 11 = 8 bit 10 = 7 bit 01 = 6 bit 00 = 5 bit
					0/1			stabilisce (Stop bits) il numero di stop previsti per completare il *frame* del *dato seriale*: 0 = 1 bit di stop 1 = 2 bit di stop (o 1½ bit di stop, con caratteri a 5 bit, cioè con i bit0/bit1=00)
				0/1				stabilisce (Parity Enable) se il *frame seriale asincrono* prevede la presenza del *bit di parità*: 0 = nessuna parità 1 = parità abilitata
			0/1					stabilisce (Parity Control) il criterio scelto per verificare la *parità* (solo se bit3=1=parità abilitata): 0 = si utilizza parità dispari (odd) 1 = si utilizza parità pari (even)
		0/1						introduce (Sticky Parity) una variante sul valore del bit di parità (solo se bit3=1=parità abilitata): 0 = valore dinamico, qualificato dal valore del bit4 1 = valore fisso (parità bloccata), sempre uguale a 1 (se bit4=0) o a 0 (se bit4=1)
	0/1							0 = in condizioni normali 1 = Transmit break: il trasmettitore pone in continuazione il livello logico 0 (SPACE) sulla linea
0/1								0 = in condizioni normali: l'indirizzo [port_8] punta il Registro di Ricetrasmissione Dati e l'indirizzo [port_9] punta il Registro di Abilitazione delle Interruzioni 1 = gli indirizzi [port_8] e [port_9] puntano i 2 Registri divisori di Baud Rate

- L'azione associata a ciascun bit di questo registro è ora descritta in dettaglio:
  - **Word Length**: i bit**0** e bit**1** stabiliscono il numero di bit previsti per ogni *carattere* trasmesso o ricevuto, associando loro la codifica binaria: bit**1**/bit**0**=**00** = **5** bit, bit**1**/bit**0**=**01** = **6** bit, bit**1**/bit**0**=**10** = **7** bit, bit**1**/bit**0**=**11** = **8** bit; la dimensione più diffusa è quella a **8** bit, ma in passato si è praticata anche quella a **7** bit, tipica dei caratteri ASCII standard (con 1 bit in meno su 10 la comunicazione seriale è più veloce del **10%**)
  - **Stop bits**: il bit**2** stabilisce il numero di stop previsti per completare il *frame* del *dato seriale*: **0** = **1** *bit di* ***stop***, **1** = **2** *bit di* ***stop*** (o **1½** *bit di* ***stop***, nel caso rarissimo di caratteri a **5** bit, cioè con i bit**0**/bit**1**=**00**); la presenza di (almeno) un *bit di* ***stop*** è stata pensata per dare all'**UART** il tempo di *processare il dato corrente*, prima di servire il successivo; con 2 bit di stop questo tempo raddoppia ma si tratta di un'esigenza sentita *solo* da dispositivi particolarmente lenti e quindi da evitare (con 1 bit in più su 10 la comunicazione seriale è più lenta del **10%**)
  - **Parity Enable**: se il bit**3** è **1** significa che il ***frame*** *seriale asincrono* prevede la presenza del *bit di* ***parità***, posizionato subito prima di quello di stop con lo scopo di *tentare di scoprire* se il dato *ricevuto* è uguale a quello *trasmesso*, a causa della possibile *rumorosità* della linea di comunicazione. Il suo valore è generato dall'**UART** in modo da rendere *pari* o *dispari* il numero di *bit a **1*** presenti **prima** del bit di stop (quindi quelli del *dato* ospitato e dello stesso *bit di* ***parità***); ovviamente con bit**3**=**0** (=*nessuna* ***parità***) la mancanza del bit aggiunto rende più veloce la comunicazione seriale
  - **Parity control**: il bit**4**, significativo solo se il controllo di ***parità*** è *abilitato* (bit**3**=**1**), stabilisce il criterio scelto per verificarla:
    - con bit**4**=**0** si utilizza ***parità*** *dispari* (*odd*): se il *dato seriale* contiene un numero dispari di *bit a **1*** (esempio: **10010010**) il *bit di* ***parità*** è lasciato a **0**, e viceversa
    - con bit**4**=**1** si utilizza ***parità*** *pari* (*even*): se il *dato seriale* contiene un numero pari di *bit a **1*** (esempio: **10110010**) il *bit di* ***parità*** è lasciato a **0**, e viceversa

- **Sticky Parity**: il bit5, significativo solo se il controllo di ***parità*** è *abilitato* (bit3=1), introduce una *variante* sui valori fatti assumere dall'**UART** al *bit di* ***parità***:
  - con bit5=0 è quello dinamico, legato al numero di *bit a* ***1*** presenti **prima** del bit di stop e qualificato dalla tipologia "*pari* o *dispari*" suggerita dal valore del bit4
  - con bit5=1 è invece fisso (*parità* ***bloccata***), cioè sempre uguale a **1** (se bit4=0=*parità dispari*) o a **0** (se bit4=1=*parità pari*); è evidente che, in questo caso il *controllo di parità* non c'entra per nulla; l'idea (particolarmente cervellotica) è quella di trasmettere **dati seriali** a **7** bit come fossero a **8** bit, aggiungendo l'ottavo con *questa tecnica*, rigorosamente a **1** o a **0**
- **Transmit break**: in condizioni normali il bit6 è **0**; quando è posto a **1** il trasmettitore porrà in continuazione sulla linea **SOUT** (pin11 dell'**UART**) il livello logico **0** (**SPACE**, **0V** in TTL, tra **+25V** e **+3V** in linea), ignorando ogni dato ancora da trasmettere fino a quando non sarà riportato a **0**; la presenza di questo *segnale di* ***Break*** può essere utilizzata per obbligare un *computer remoto* ad interrompere l'esecuzione di un suo programma o per richiedere la sua attenzione.
- **Baud Rate Divisor Latch**: con ogni versione di **UART** il valore logico assunto dal bit7 influenza il tipo di registro che può essere puntato con gli indirizzi [**port_8**] e [**port_9**]:
  - con bit7=1 quelli coinvolti sono i 2 **Registri** divisori di Baud Rate,
  - in condizioni normali il bit7 è **0**; l'indirizzo [**port_8**] punta il **Registro** di Ricetrasmissione Dati e l'indirizzo [**port_9**] punta il **Registro** di Abilitazione delle Interruzioni

- Il breve codice seguente suggerisce un modo corretto di programmare questo registro, nelle fasi iniziali della gestione seriale:

```
depTIP  DB    03H              ;┐COSTANTE  associata al PROTOCOLLO SERIALE
                               ;|          .... ..11 > 8 bit per il DATO
                               ;|          .... .0.. > 1 bit di stop
                               ;┘          ...x 0... > nessuna parità
;-----------------------------------------------------------------------
;                              ;┐ ┌─[port_B]──────────────────────────┐
                               ;| │ Registro di controllo della LINEA │
                               ;| └───────────────────────────────────┘
                               ;|Viene ricostruito il byte da spedire alla por-
                               ;|ta [port_B], che rappresenta le  caratteristi-
                               ;|che desiderate, cioè il Numero di bit per DATO
                               ;|il Numero di bit di STOP e la presenza del Bit
                               ;|di Parità; poiché la programmazione della  ve-
                               ;|locità di RICETRASMISSIONE è già stata esegui-
                               ;|ta è necessario poi riporre a "0" il  bit7  di
                               ;|tale byte; di norma si pone a "0" anche il bit
                               ;|6 (se ="1", impostazione di break, il dato  di
                               ;|output seriale è forzato  alla  condizione  di
                               ;|SPACE (0 logico) indipendentemente da ciò  che
                               ;|l'UART intende trasmettere) e il bit5 (se= "1"
                               ;|il bit di parità viene forzato  ad  un  valore
                               ;|costante, in funzione della programmazione dei
                               ;|bit4/bit3; altrimenti il bit di parità conser-
                               ;|va il suo significato)
                               ;| ┌────────────────────────────────────────┐
                               ;| |E' OVVIAMENTE necessario lasciare in  AL  la|
        MOV     DX,[port_B]    ;| |combinazione di BIT corrispondente alle  no-|
        MOV     AL,CS:[DepTIP] ;| |stre esigenze; la scelta qui fatta per esem-|
        OUT     DX,AL          ;| |pio si riferisce ad un protocollo con 8  bit|
                               ;| |di Dato, 1 bit di Stop e nessun controllo di|
                               ;| |parità:                                     |
                               ;| ├──[AL=0000 0011=03H]────────────────────────┤
                               ;| |      .... ..11 > 8 bit per il DATO        |
                               ;| |      .... .0.. > 1 bit di stop            |
                               ;┘ |      ...x 0... > nessuna parità           |
```

1 - Messa a Punto - Descrizione dei Registri UART

1.8 03FCH / 02FCH / 03ECH / 02ECH - OUT / IN - Registro di Controllo del Modem

- Il quinto **Registro** dell'**UART** della *porta seriale* è utilizzato in tutte le versioni come **Registro** di **Controllo Modem**, anche se l'azione di controllo nei confronti del modem sia limitata ai soli *2 bit meno significativi*; sarà possibile intervenire su questi bit per *forzare da software* le corrispondenti *linee hardware* dell'**UART** ai valori logici desiderati.
- Le linee controllate (**RTS** e **DTR**) sono entrambe in uscita dalla *porta seriale* (**UART**, cioè dal *computer*, **DTE**) e in ingresso al *modem* (**DCE**); altre linee (**CTS**, **DSR**, **CD** e **RI**), in ingresso all'UART (cioè con direzione **DCE>DTE**), possono essere controllate con il **Registro** di Stato del Modem [**port_E**].
- Un bit di questo Registro è particolarmente utile perchè consente di porre l'**UART** in condizioni di **autoverifica** (*loopback testing*).

- Questo registro può essere anche *letto*, facilitando la modifica dei suoi singoli bit.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_C	03FCH - 02FCH - 03ECH - 02ECH
								MCR	Modem Control Register (lettura/scrittura)
							1		1 = *attiva* (mette a 0 logico) la linea DTR (Data Terminal Ready, pin33 in uscita dall'UART) e quindi pone una *tensione positiva* (ON, SPACE, tra +25V e +3V) sul pin4/DB9 o sul pin20/DB25 del connettore; il segnale DTR è di solito attivato dal DTE per *avvisare* il DCE che è *regolamente collegato alla linea di comunicazione* ed è *pronto a trasmettere o ricevere dati*
						1			1 = *attiva* (mette a 0 logico) la linea RTS (Request To Send, pin32 in uscita dall'UART) e quindi pone una *tensione positiva* (ON, SPACE, tra +25V e +3V) sul pin7/DB9 o sul pin4/DB25 del connettore; il segnale RTS è di solito attivato dal DTE per *avvisare* il DCE che *dispone di dati* ed è *pronto a trasmetterglieli*
					0				0 = condizioni normali 1 = *attiva* (mette a 0 logico) la linea OUT1 (*uscita ausiliaria* di *uso generale*, *attiva bassa*, disponibile sul pin34 dell'UART); si tratta di una linea non utilizzata nei PC IBM e compatibili
				0					0 = condizioni normali (*interruzioni* da parte dell'UART *disabilitate*) 1 = *abilitare* le *interruzioni* da parte dell'UART cioè, *attivando* (0 logico) la linea OUT2 (pin31 in uscita dall'UART), collega la linea INTR (pin30 dell'UART) alla linea IRQ3/IRQ4 del *controller delle interruzioni*
			0						0 = condizioni normali 1 = pone la *porta seriale* in Lookback Mode cioè *collega tra loro* i *registri interni* di *Trasmissione* e di *Ricezione* e le *linee* di *controllo Modem*, RTS con CTS, DTR con DSR , OUT1 con RI e OUT2 con CD; ogni dato erogato può essere immediatamente letto
0	0	0							riservati, non utilizzati, sempre a 0 logico

- Come anticipato le azioni esercitate da questo registro (MCR) sono di 2 tipi; i *4 bit meno significativi* consentono di ***forzare*** a *livello* ***attivo*** (**0** logico) le linee **DTR**, **RTS**, **OUT1** e **OUT2** (*attive basse*) in uscita dall'**UART** (*computer* **DTE**); le prime 2, per intercessione di Driver di linea *invertenti* (come il DS1488) forniranno una *tensione* ***positiva*** (**ON**, **SPACE**, tra **+25V** e **+3V**) ai segnali **DTR** e **RTS**, sui rispettivi pin del connettore, a beneficio del *modem* **DCE**; in dettaglio:
  - **Data Terminal Ready**: se il bit**0** è posto a **1** la linea **DTR** (Data Terminal Ready, pin**33** UART) passa a **0** logico e forza una *tensione* ***positiva*** sul connettore (pin**4**/**DB9** o pin**20**/**DB25**); il segnale **DTR** è di solito attivato dal *computer* **DTE** (**UART**) per *avvisare* il *modem* **DCE** che è *regolarmente collegato alla linea di comunicazione* ed è *pronto a trasmettere o ricevere dati*; in risposta il **DCE** ***attiverà*** il segnale **DSR** (Data Set Ready) per segnalare al **DTE** la sua presenza sulla *linea di comunicazione*
  - **Request To Send**: se il bit**1** è posto a **1** la linea **RTS** (Request To Send, pin**32** UART) passa a **0** logico e forza una *tensione* ***positiva*** sul connettore (pin**7**/**DB9** o pin**4**/**DB25**); il segnale **RTS** è di solito attivato dal *computer* **DTE** (**UART**) per *avvisare* il *modem* **DCE** che *dispone di dati* ed è *pronto a trasmetterglieli* (letteralmente, per *chiedere* alla periferica *di collegarsi*); in risposta il **DCE** ***attiverà*** il segnale **CTS** (**Clear To Send**) per segnalare al **DTE** la sua disponibilità a ricevere
  - **OUT1**: se il bit**2** è posto a **1** la linea **OUT1** (*uscita ausiliaria* di *uso generale*, pin**34** UART) passa a *livello* ***attivo*** (**0** logico); si tratta di una linea programmabile dall'utente (non utilizzata nei PC IBM e compatibili) che può servire per controllare eventuali azioni non correlate con la comunicazione seriale
  - **Interrupt Enable - OUT2**: se il bit**3** è posto a **1** la linea **OUT2** (*uscita ausiliaria* di *uso generale*, pin**31** UART) passa a *livello* ***attivo*** (**0** logico); nell'ambito dei PC IBM e compatibili è usata per controllare (*enable*) l'uscita di un *buffer tri-state* che collega la linea **INTR** (pin**30** dell'UART) alla linea **IRQ3**/**IRQ4** del *controller delle interruzioni*: in questo modo il bit**3** va posto a **1** per ***abilitare*** le ***interruzioni*** da parte dell'**UART** e va posto a **0** se non si desidera utilizzare questa tecnica
- Dei rimanenti *4 bit più significativi* è importante solo il bit**4**:
  - **Lookback Mode Enable**: il bit**4** deve essere a **0** in condizioni normali; se è forzato a **1** la *porta seriale* (**UART**) si pone in uno stato molto particolare, straordinariamente utile per collaudare i nostri programmi, per sottoporre a verifica i nostri progetti seriali o semplicemente per rilevare la presenza dell'UART; ecco cosa succede:
    - la linea d'ingresso seriale (**SIN**, pin**10** dell'UART) viene *scollegata* e quella d'uscita seriale (**SOUT**, pin**11** dell'UART) è forzata a **1** logico (non attiva)
    - le 4 linee d'ingresso di *controllo Modem* [**CTS** (Clear To Send, pin**36**), **DSR** (Data Set Ready, pin**37**), **RI** (Ring Indicator, pin**39**) e **CD** (Carrier Detect, pin**38**)] sono *scollegate*
    - le 4 linee d'uscita di *controllo Modem* sono sono forzate a **1** logico (non attive) e *collegate* internamente ai rispettivi ingressi [**RTS** (Request To Send, pin**32**) con **CTS**, **DTR** (Data Terminal Ready, pin**33**) con **DSR**, **OUT1** (pin**34**) con **RI** e **OUT2** (pin**31**) con **CD**]
    - i *registri a scorrimento* interni di Trasmissione (Transmit Shift Register, **TSR**) e di Ricezione (Receive Shift Register, **RSR**) sono messi direttamente in comunicazione, per cui qualunque dato erogato con l'istruzione **OUT** può essere immediatamente letto con l'istruzione **IN**
  - Reserved: i bit**5**, bit**6** e bit**7** sono sempre a **0** (non usati da nessuna versione di **UART**)

1 - Messa a Punto - Descrizione dei Registri UART

1.9 03FDH / 02FDH / 03EDH / 02EDH - IN - Registro di Stato della Linea

- Il sesto Registro dell'UART della *porta seriale* è utilizzato in tutte le versioni come **Registro** di **Stato Linea**, con il compito di fornire informazioni dettagliate sulla presenza di dati in linea e su eventuali errori riscontrati durante la ricetrasmissione.
- Si tratta di un Registro a *a sola lettura*.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_D / LSR	03FDH - 02FDH - 03EDH - 02EDH / Line Status Register (lettura)
							1		1 = Received Data Ready o Data Available: *nuovo dato* trasferito dal *registro a scorrimento* RSR al RBR [port_8] o nel FIFO in Ricezione (16550A) 0 = non appena il processore estrae il dato dal RBR (o dal FIFO) oppure se il processore azzera il contenuto del FIFO in Ricezione
						1			1 = ricezione gravata da errore *di sovrapposizione* (Overrun Error) 0 = non appena il registro LSR viene letto
					1				1 = ricezione gravata da errore *di parità* (Parity Error): prima del bit di stop l'UART ha rilevato nel dato ricevuto un numero di *bit a 1* diverso da quello (*pari* o *dispari*) previsto dalla programmazione del Registro di Controllo Linea [port_B] 0 = non appena il registro LSR viene letto
				1					1 = ricezione gravata da errori *di composizione* (Framing Error) 0 = non appena il registro LSR viene letto
			1						1 = è presente un *segnale di Break* (Break Interrupt) 0 = non appena il registro LSR viene letto
		1							1 = THR [port_8] o FIFO in Trasmissione (16550A) *vuoto* (Transmitter Holding Register Empty) 0 = non appena un nuovo carattere da trasmettere entra nel THR (o nel FIFO)
	1								1 = *non ci sono dati da trasmettere*, *registro* TSR e THR [port_8] o FIFO in Trasmissione (16550A) *vuoti* (Data Holding Register Empty) 0 = non appena un nuovo carattere da trasmettere entra nel TSR
0									riservati, non utilizzati, sempre a 0 logico (8250/16450) 1 = nel buffer FIFO in Ricezione sono presenti bytes gravati da errori (*di sovrapposizione*, *di parità* o *di composizione* o *di break*) (16550A) 0 = non appena il registro LSR viene letto, se non ci sono altri bytes non corretti

- L'azione associata a ciascun bit di questo registro è ora descritta in dettaglio:
  - **Received Data Ready** o **Data Available**: il bit**0** è trovato a **1** se ***un nuovo dato*** è stato completamente ricostruito dal *registro a scorrimento* interno (Receive Shift Register, **RSR**) e da esso è stato trasferito nel **Registro** di Ricezione [**port_8**] (Receive Buffer Register, **RBR**) (o nel **buffer FIFO** in Ricezione, con **UART 16550A**); questo bit è riportato a **0** logico non appena il processore estrae il dato dal **RBR** (o dal **FIFO**) oppure se il processore azzera il contenuto del **FIFO** in Ricezione; l'evento "***dato pronto***" genera una *richiesta di interruzione* di seconda priorità (se è abilitata, cioè se bit**0**=**1** nel **Registro** di Abilitazione delle Interruzioni [**port_9**]) segnalandola con il codice binario bit**2**/bit**1**/bit**0**=**100** nel **Registro** di Identificazione delle Interruzioni [**port_A**]
  - **Errori in Ricezione**: i bit**1**, bit**2**, bit**3** e bit**4** sono trovati a **1** se la ricezione è gravata da **errori** o se il *computer remoto* richiede attenzione; ciascuno di essi sarà riportato a **0** logico non appena questo registro (LSR) viene letto e l'evento seriale ad essi associato genererà la *richiesta di interruzione* della massima priorità (se è abilitata, cioè se bit**2**=**1** nel **Registro** di Abilitazione delle Interruzioni [**port_9**]) segnalandola con il codice binario bit**2**/bit**1**/bit**0**=**110** nel **Registro** di Identificazione delle Interruzioni [**port_A**]; in dettaglio:
    - **Overrun Error**: il bit**1** è trovato a **1** se la ricezione è gravata da **errore** *di sovrapposizione*: con **UART 8250**/**16450** il processore non ha fatto in tempo a leggere un dato dal **Registro** di Ricezione [**port_8**] (Receive Buffer Register, **RBR**, prima dell'arrivo del successivo, oppure (con **UART 16550**) il **buffer FIFO** in Ricezione è pieno e il dato ricostruito nel *registro a scorrimento* interno (Receive Shift Register, **RSR**) non può esservi inserito; il dato non letto viene perduto, sovrascritto da quello in arrivo
    - **Parity Error**: il bit**2** è trovato a **1** se la ricezione è gravata da **errore** *di parità*: prima del bit di stop l'**UART** ha rilevato nel dato ricevuto un numero di *bit a* ***1*** diverso da quello (*pari* o *dispari*) previsto dalla programmazione del **Registro** di Controllo Linea [**port_B**]
    - **Framing Error**: il bit**3** è trovato a **1** se la ricezione è gravata da **errori** *di composizione*: se, dopo l'ultimo bit di dato previsto dalla programmazione del **Registro** di Controllo Linea [**port_B**], viene rilevato uno **0** logico la struttura (*frame*) del dato ricevuto è scorretta perchè priva del bit di stop (notoriamente un bit a **1**)

- **Break Interrupt**: il bit4 è trovato a 1 se sulla linea d'ingresso seriale (**SIN**, pin10 dell'**UART**) è presente un *segnale di Break*, cioè se la linea è tenuta a livello logico **0** (**SPACE**) per un tempo maggiore a quello previsto per ricevere un dato formattato (cioè completo di bit di start, eventuale parità e di stop); nel **Registro** di Ricezione [**port_8**] (o nel **FIFO** in Ricezione, con **UART 16550**) è inserito un byte nullo. Questo *segnale* può essere generato dal *computer remoto* per richiedere attenzione e la ricezione riprenderà solo quando la linea **SIN** torna a livello logico **1**, in attesa del bit di start (notoriamente un bit a **0**) del nuovo dato in arrivo
- **Transmitter Holding Register Empty**: il bit5 è trovato a **1** se il **Registro** di Trasmissione [**port_8**] (Transmitter Holding Register, **THR**) (o il **buffer FIFO** in Trasmissione, con **UART 16550A**) è ***vuoto***: il dato in esso *scritto* in precedenza dal processore è stato trasferito al *registro a scorrimento* interno, **Transmit Shift Register**, **TSR**; l'**UART** è pronto ad accettare nuovi caratteri da trasmettere e, non appena uno di essi entra nel **THR** (o nel **FIFO**), riporta a **0** logico questo bit; l'evento "***trasmettitore vuoto***" genera una *richiesta di interruzione* di terza priorità (se è abilitata, cioè se bit**1**=**1** nel **Registro** di Abilitazione delle Interruzioni [**port_9**]) segnalandola con il codice binario bit2/bit1/bit0=**010** nel **Registro** di Identificazione delle Interruzioni [**port_A**]
- **Transmitter Shift Register Empty** (**Data Holding Register Empty**): il bit6 è trovato a **1** se ***non ci sono dati da trasmettere*** cioè se anche il *registro a scorrimento* interno, **Transmit Shift Register**, **TSR**, è *vuoto*: il trasmettitore è inoperoso, in attesa di dati da mettere in linea; in queste condizioni anche il **Registro** di Trasmissione [**port_8**] (Transmitter Holding Register, **THR**) (o il **buffer FIFO** in Trasmissione, con **UART 16550A**) è ovviamente *vuoto* e, non appena un dato vi entrerà, questo bit torna a **0** logico
- Reserved: il bit7 è sempre a **0**, con **UART 8250/16450**; nelle versioni più recenti (**16550** e successivi) il bit5 è trovato a **1** se, nel **buffer FIFO** in Ricezione sono presenti bytes gravati da **errori** (*di sovrapposizione*, *di parità* o *di composizione* o *di break*); sarà riportato a **0** logico non appena questo registro (LSR) viene letto, se non ci sono altri bytes non corretti

1 - Messa a Punto - Descrizione dei Registri UART

1.10 03FEH / 02FEH / 03EEH / 02EEH - IN - Registro di Stato del Modem

- Il settimo **Registro** dell'**UART** della *porta seriale* è utilizzato in tutte le versioni come **Registro** di **Stato del Modem**, con il compito di fornire informazioni utili al controllo del *flusso dati* durante la ricetrasmissione sul canale telefonico.
- Si tratta di un Registro a *a sola lettura* in grado di monitorare il segnale di *handshake* Clear To Send (**CTS**) e quelli di *stato del modem* Data Set Ready (**DSR**), Ring Indicator (**RI**) e Carrier Detect (**CD**): i *4 bit meno significativi* segnalano ogni *variazione* subita da questi segnali dopo l'ultima lettura di questo Registro (e per questo sono detti *delta*), mentre i *4 bit più significativi* assumono il loro *valore corrente*.
- Le linee controllate (**CTS**, **DSR**, **RI** e **CD**) sono tutte in ingresso alla *porta seriale* (**UART**, cioè al *computer*, **DTE**) e in uscita dal *modem* (**DCE**); altre linee (**RTS** e **DTR**), in uscita dall'UART (cioè con direzione **DTE>DCE**) possono essere controllate con il **Registro** di Controllo del Modem [**port_C**].

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_E MSR	03FEH - 02FEH - 03EEH - 02EEH **Modem Status Register (lettura)**
							1		1 = Delta Clear to Send o Clear to Send has changed: la linea CTS ha *cambiato livello logico* dopo l'ultima lettura 0 = non appena il registro MSR viene letto
						1			1 = Delta Data Set Ready o Data Set Ready has changed: la linea DSR ha *cambiato livello logico* dopo l'ultima lettura 0 = non appena il registro MSR viene letto
					1				1 = Trailing Edge of Ring Indicator: la linea RI ha *cambiato livello logico* dopo l'ultima lettura 0 = non appena il registro MSR viene letto
				1					1 = Delta Data Carrier Detect: la linea CD ha *cambiato livello logico* dopo l'ultima lettura 0 = non appena il registro MSR viene letto
			x						Clear to Send: posto al *livello logico* presente sulla linea CTS nel momento della lettura
		x							Data Set Ready: posto al *livello logico* presente sulla linea DSR nel momento della lettura
	x								Ring Indicato: posto al *livello logico* presente sulla linea RI nel momento della lettura
x									Data Carrier Detect: posto al *livello logico* presente sulla linea CD nel momento della lettura

- Come anticipato le informazioni fornite da questo registro (MSR) sono di 2 tipi; i *4 bit meno significativi* segnalano ogni ***variazione*** (***delta***) subita dai segnali **CTS**, **DSR**, **RI** e **CD** ***dopo*** l'*ultima lettura* di questo Registro da parte del processore; ciascuno di essi sarà riportato a **0** logico non appena il MSR viene letto e l'evento seriale ad essi associato (*cambiamento dello stato del Modem*) genererà la *richiesta di interruzione* di priorità più bassa (se è abilitata, cioè se bit**3**=**1** nel **Registro** di Abilitazione delle Interruzioni [**port_9**]) segnalandola con il codice binario bit**2**/bit**1**/bit**0**=**000** nel **Registro** di Identificazione delle Interruzioni [**port_A**]; in dettaglio:

- **Delta Clear to Send** (**Clear to Send has changed**): il bit**0** è posto a **1** se la linea $\overline{\text{CTS}}$ (Clear To Send, pin**36** UART) ha *cambiato livello logico* **dopo** l'ultima lettura del registro (vedi descrizione bit**4** per i dettagli sul segnale **CTS** *attivo*)
- **Delta Data Set Ready**: (**Data Set Ready has changed**): il bit**1** è posto a **1** se la linea $\overline{\text{DSR}}$ (Data Set Ready, pin**37** UART) ha *cambiato livello logico* **dopo** l'ultima lettura del registro (vedi descrizione bit**5** per i dettagli sul segnale **DSR** *attivo*)
- **Trailing Edge of Ring Indicator**: il bit**2** è posto a **1** se la linea $\overline{\text{RI}}$ (Ring Indicator, pin**39** UART) ha *cambiato livello logico* **dopo** l'ultima lettura del registro (vedi descrizione bit**6** per i dettagli sul segnale **RI** *attivo*)
- **Delta Data Carrier Detect**: il bit**3** è posto a **1** se la linea $\overline{\text{CD}}$ (Data Carrier Detect, pin**38** UART) ha *cambiato livello logico* **dopo** l'ultima lettura del registro (vedi descrizione bit**7** per i dettagli sul segnale **CD** *attivo*)

I *4 bit più significativi* del MSR riflettono il ***valore logico corrente*** delle linee $\overline{\text{CTS}}$, $\overline{\text{DSR}}$, $\overline{\text{RI}}$ e $\overline{\text{CD}}$ (*attive basse*) in ingresso all'**UART** (*computer* **DTE**), fornito dal Receiver di linea *invertente* (per esempio un DS1489) a partire dal livello dei segnali **CTS**, **DSR**, **RI** e **CD** imposti sul connettore dal *modem* **DCE**; esso *attiva* questi segnali mettendo una *tensione* ***positiva*** (**ON**, **SPACE**, tra **+25V** e **+3V**) sui rispettivi pin del connettore, trasformata in **0** logico dal Receiver di linea e fornita in ingresso all'UART; in dettaglio:

- **Clear to Send**: il bit**4** è posto al *livello logico* assunto dalla linea $\overline{\text{CTS}}$ (Clear To Send, pin**36** UART); il **DCE** *attiva* il segnale **CTS** [sul pin**8**/**DB9** o sul pin**5**/**DB25** del connettore] per *avvisare* il **DTE** che è *pronto a ricevere* i suoi *dati*, cioè per informarlo che *la trasmissione può cominciare* (o *continuare*)
- **Data Set Ready**: il bit**5** è posto al *livello logico* assunto dalla linea $\overline{\text{DSR}}$ (Data Set Ready, pin**37** UART); il **DCE** *attiva* il segnale **DSR** [sul pin**6**/**DB9** o sul pin**6**/**DB25** del connettore] per *avvisare* il **DTE** che è *regolarmente collegato alla linea di comunicazione* ed è *pronto a concorrere sul canale di comunicazione* (cioè a trasmettere o ricevere dati)
- **Ring Indicator**: il bit**6** è posto al *livello logico* assunto dalla linea $\overline{\text{RI}}$ (Ring Indicator, pin**39** UART); il **DCE** *attiva* il segnale **RI** [sul pin**9**/**DB9** o sul pin**22**/**DB25** del connettore] per *avvisare* il **DTE** che sul canale di comunicazione è in atto una chiamata entrante, cioè che sulla linea telefonica si sta ricevendo un *segnale acustico* a bassa frequenza e ad alta tensione (che fa suonare il campanello del telefono)
- **Carrier Detect**: il bit**7** è posto al *livello logico* assunto dalla linea $\overline{\text{CD}}$ (Data Carrier Detect, pin**38** UART); il **DCE** *attiva* il segnale **CD** [sul pin**1**/**DB9** o sul pin**8**/**DB25** del connettore] per *avvisare* il **DTE** che ha riconosciuto la presenza del modem (**DCE** remoto) *dall'altra parte* della linea telefonica, o meglio ha rilevato la presenza sulla linea della *portante* (*carrier*) sulla quale il **DCE** remoto ha *modulato* i suoi dati, sinonimo di *segnale di buona qualità* e di canale di comunicazione *pulito, poco rumoroso*

- Messa a Punto - Descrizione dei Registri UART

1.11 03FFH / 02FFH / 03EFH / 02EFH - OUT / IN - Registro di scratch

- L'ottavo **Registro** dell'**UART** della *porta seriale* è noto in tutte le versioni come **Registro** di **Scratch**; non viene coinvolto nella comunicazione seriale e può essere utilizzato dal processore (sia il *scrittura* che in *lettura*) come ***deposito temporaneo*** di dati, come fosse un una comune *locazione di memoria a 8 bit*.
- Poichè non era presente nella prima, antica, versione originale **UART 8250**/**8250B**, se il byte *letto* da questo registro è il medesimo *scritto* in esso in precedenza dalla *lettura*, è facile concludere che la *porta seriale* dispone di un **UART** di versioni più recente (**8250A**/**16450**/**16550** e successivi).

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	port_F / SR	03FFH - 02FFH - 03EFH - 02EFH / Scratch Register (lettura/scrittura)
							x		bit0 in uscita o in ingresso
						x			bit1 in uscita o in ingresso
					x				bit2 in uscita o in ingresso
				x					bit3 in uscita o in ingresso
			x						bit4 in uscita o in ingresso
		x							bit5 in uscita o in ingresso
	x								bit6 in uscita o in ingresso
x									bit7 in uscita o in ingresso

	Indice Analitico	Terza parte Porta Seriale