

2.1.11 Gestione di una linea seriale

Un altro tipo di periferica molto importante è quella che consente la trasmissione a distanza su una linea seriale. Nella comunicazione tra due calcolatori, o in generale tra qualunque due unità intelligenti, posti a distanza superiore al metro è conveniente ricorrere alla **trasmissione seriale**. Con ciò si vuole intendere che gli otto bit di un carattere, invece che venir trasmessi contemporaneamente su otto fili diversi (**trasmissione parallela**), vengono trasmessi uno alla volta, in successione temporale, su un unico filo. Dato che nel bus di un calcolatore i dati vengono trasferiti in parallelo tra i vari chips (tipicamente tra **CPU** e **Memoria**), è necessario predisporre un adeguato dispositivo di interfaccia in grado di ricevere (o trasmettere) dal **Data Bus** tutti i bits contemporaneamente, e di trasmetterli (riceverli), uno alla volta, alla (dalla) linea periferica di comunicazione (**linea seriale**). La **Fig.2.1.11.1** illustra questo sistema di comunicazione seriale.

Ogni bit della trasmissione seriale deve permanere sulla linea per un tempo, detto **bit time**, legato alle caratteristiche di velocità del ricevitore.

Questo segnale di tempo è prodotto da un **generatore di clock** la cui frequenza, detta **baud rate** (o **velocità di segnalazione**), determina il numero di cambiamenti di livello del segnale trasmessi ad ogni secondo. Se si usa un segnale a due livelli (binario) ogni livello codifica un bit, per cui questa velocità coincide con il numero di bit al secondo trasmessi. Trasmettitore e ricevitore devono quindi sincronizzarsi sulla stessa velocità se si vuole che ogni bit trasmesso sia anche correttamente ricevuto.

Per mantenere il sincronismo, il trasmettitore deve informare il ricevitore dell'istante preciso nel quale pone sulla linea un nuovo bit del dato, ed il ricevitore, prima che venga cambiato questo bit con quello successivo, deve leggerlo. Ciò si può ottenere se il trasmettitore invia, con un filo di linea aggiuntivo, al ricevitore il suo segnale clock (**trasmissione sincrona**), oppure se il ricevitore, equipaggiato con un generatore di baud rate uguale a quello del trasmettitore si sincronizza sulla trasmissione di ogni singolo carattere con l'uso di due bit aggiuntivi (**trasmissione asincrona**). Il primo di questi bit, detto **bit di start**, precede l'invio dei bit del carattere, e avvisa il ricevitore che è in arrivo un nuovo carattere. Il secondo, detto **bit di stop**, chiude la trasmissione e lascia tempo al ricevitore che si predisponga per la successiva acquisizione. Più precisamente il **protocollo di comunicazione** (a livello fisico) è composto dalle seguenti regole:

- 1) il livello logico 1 indica nessuna comunicazione (stato riposo della linea);
- 2) la transizione 1→0 indica l'inizio della trasmissione di un carattere con l'inizio del bit di start, che permane in linea per un bit time, e da tempo al ricevitore di sincronizzare il suo generatore di baud rate. A metà di questo intervallo il ricevitore produce il suo primo impulso di clock;
- 3) per altri otto bit time, della stessa durata, il trasmettitore invia sulla linea gli otto bit del dato. Il ricevitore acquisisce ogni bit, al centro di ogni bit time, utilizzando il suo generatore di baud rate;
- 4) conclude un ultimo bit time a livello logico 1, bit di stop, che lascia il tempo al ricevitore di leggere il carattere ricevuto prima di predisporre alla ricezione del successivo;
- 5) terminato il bit di stop il trasmettitore può trasmettere un nuovo dato, ripetendo il processo, o può lasciare la linea nella condizione di riposo fino a nuova trasmissione.

Si comprende che con questo protocollo anche se le frequenze dei due baud rate, in trasmissione e ricezione, non sono rigorosamente uguali gli otto impulsi del ricevitore, pur subendo slittamenti rispetto alla posizione centrale del bit time, sono sempre all'interno del loro intervallo di competenza e, quindi garantiscono letture corrette. Nella **Fig.2.1.11.1** sono illustrate queste regole di trasmissione seriale con riferimento ad una velocità di 9600 bit/sec. Il bit time risulta di circa 0,1 msec, per cui il tempo richiesto per la trasmissione di un carattere, composto da 10 bit (8 bit di dato, 1 bit di start, 1 bit di stop) è:

$$\text{Tempo}_{\text{trasmissione}_{\text{carattere}}} = 10\text{bit} \bullet 0,1 \frac{\text{msec}}{\text{bit}} = 1\text{msec}$$

Il componente, all'interno del dispositivo di interfaccia, che realizza la trasformazione parallelo-serie, e viceversa, è un registro a scorrimento (**shift register**), nel quale gli scorrimenti si succedono alla cadenza imposta dal generatore di baud-rate del trasmettitore, sul quale il ricevitore dovrà sincronizzarsi. La CPU è chiamata ad effettuare la lettura (ricevitore) o la scrittura (trasmettitore) nello shift register, ogni otto impulsi di clock, su segnalazione prodotta da un contatore modulo dieci, che agisce attivando il solito bit di stato ad ogni passaggio per lo zero. Questo segnale, come per le periferiche descritte, può produrre una interruzione se viene abilitata questa funzione da un adeguato bit di mascheratura posto sul dispositivo di interfaccia.

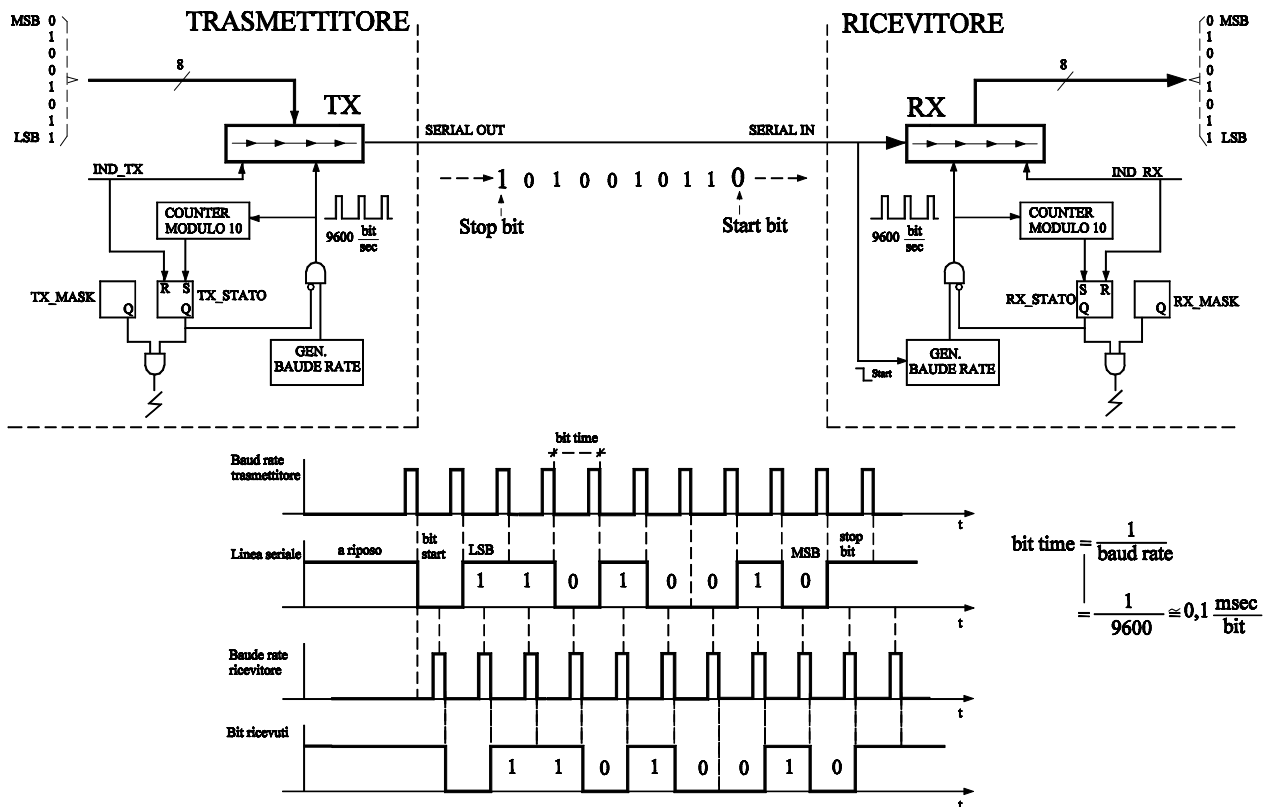


Fig.2.1.11.1 Trasmissione seriale asincrona sincronizzata su ogni carattere

La **Fig.2.1.11.1** mette in luce solo parzialmente i componenti che si trovano all'interno di un dispositivo di interfaccia. In realtà, la necessità di definire la velocità di trasmissione, il numero dei bit del dato (7 o 8 bit) il numero dei bit di stop (in condizioni più gravose per il ricevitore 1,5 o 2 bit di stop), la presenza o meno di un eventuale bit di parità, rende necessario predisporre nel dispositivo di interfaccia un adeguato numero di registri che consentano di programmare la trasmissione con valori appropriati dei suddetti parametri.

Uno di questi dispositivi è l'integrato denominato **UART** (Universal Asynchronous Receiver Transmitter). Una sua versione commerciale, che comprende due sezioni equivalenti per la comunicazione seriale, è il **DUART** (Double UART) contraddistinto dalla sigla **MC68681**.

La **Fig.2.1.11.2** mostra i componenti interni direttamente programmabili da software.

Questo dispositivo di interfaccia è multifunzione in quanto, oltre a gestire due linee seriali gemelle (Linea A e Linea B), prevede due porte parallele, non indicate in figura, di ingresso a 6 bit e di uscita a 8 bit. Contiene infine un Timer a 16 bit per generare un real time clock. Tutti questi componenti interni possono essere programmati usando registri interni specializzati per determinare diverse modalità di funzionamento. Vista la complessità del dispositivo sono implementati 32 registri interni da 8 bit. La **Fig. 2.1.11.2** mette in luce due gruppi di 7 registri per ogni linea seriale e quattro registri, di uso generale, tra i quali si riconoscono il registro **ISR** che contiene i bit di stato delle linee seriali, ed il registro **IMR** che contiene i corrispondenti bit di mascheratura di interrupt. I registri **MR1** **MR2** e **CSR** consentono di programmare i parametri che caratterizzano il protocollo di comunicazione (baud rate, tipo di parità, bit del dato, bit di stop).

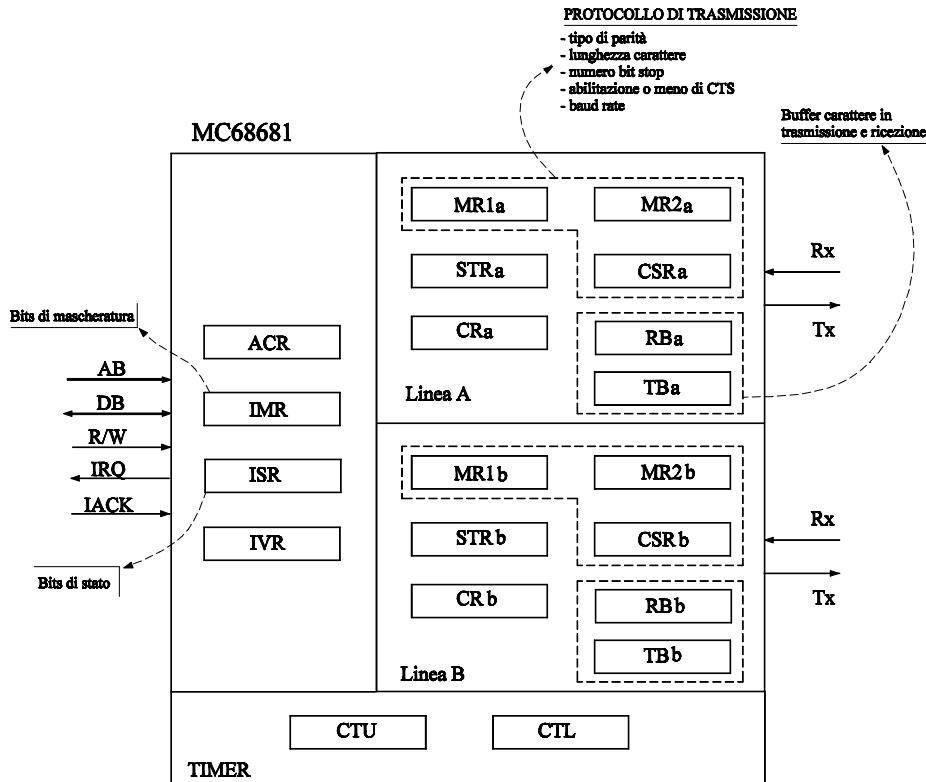


Fig2.1.11.2 Dispositivo di interfaccia per comunicazione seriale **MC68681**.

L'esemplificazione dettagliata sull'uso di questo integrato per la gestione di una linea seriale va oltre gli scopi di questa trattazione, per cui nel seguito ci si limiterà all'uso dello schema semplificato di **Fig2.1.11.1** rispetto al quale verrà illustrata l'organizzazione del programma per la gestione della linea seriale, in modo da produrre la ricezione e la trasmissione sulle linee **Rx** e **Tx** (*full duplex*) con tecnica dell'interrupt.

La **Fig. 2.1.11.3** schematizza i ruoli svolte dalle singole subroutine che compongono questo software. In questo schema le due parti comunicanti sono un **PC** ed una **UNITA' PERIFERICA**, che può essere un altro **PC**, qualunque tipo di microcontrollore o, più in generale una rete di comunicazione. Ovviamente, anche il **PC** possiede al suo interno una struttura simile a quella evidenziata nell'**UNITA' PERIFERICA**, ed ogni unità comunicante possiede sia la sezione trasmittente sia quella ricevente descritte in **Fig. 2.1.11.1**.

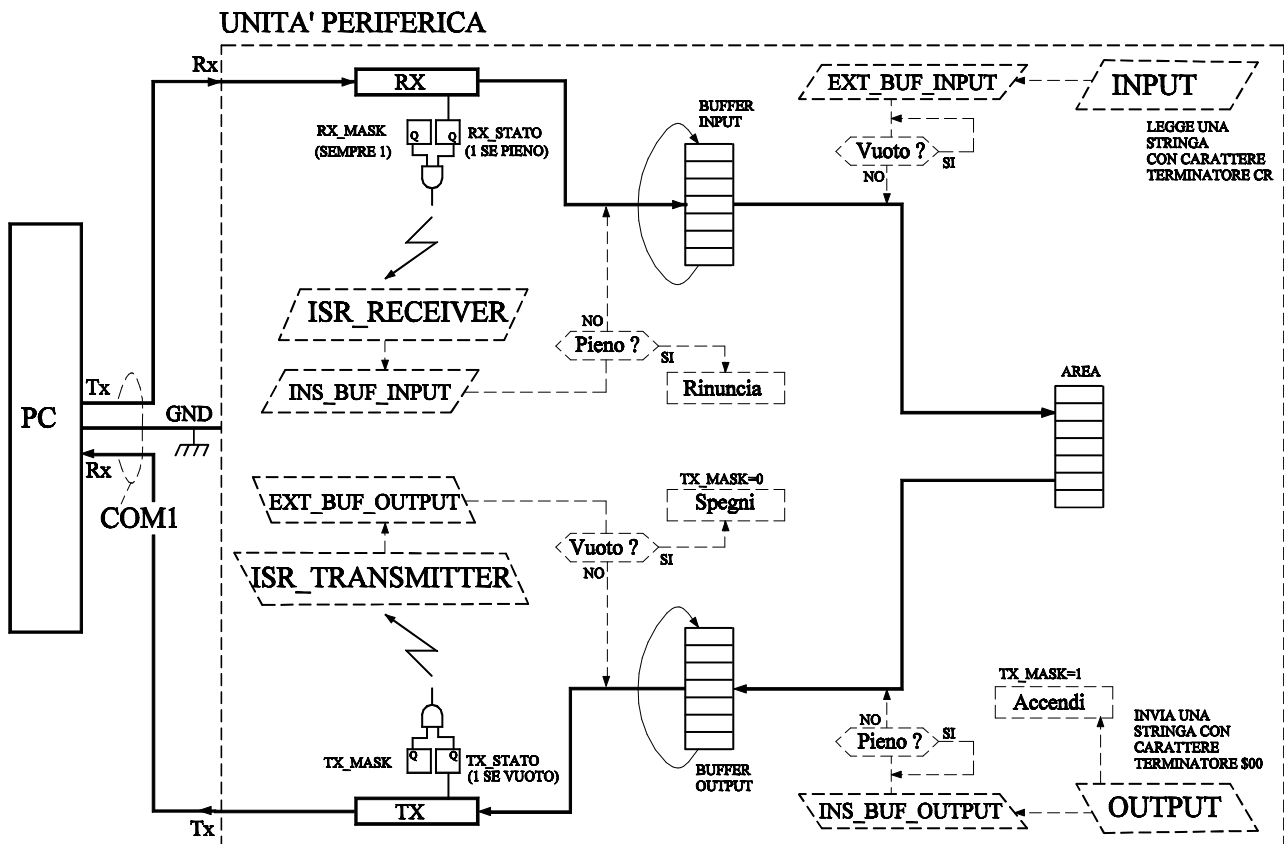


Fig.2.1.11.3 Mappa che visualizza i legami tra software e hardware per la gestione di linea seriale che pone in comunicazione due entità: un PC e una UNITA' PERIFERICA. Le parti dell'unità periferica indicate con tratto pieno rappresentano sinteticamente la parte circuitale (hardware), quelle indicate con linee tratteggiate rappresentano, in modo astratto, le routine (software), con riferimenti (freccie) alle connessioni hardware sulle quali svolgono la loro azione.

In **Fig.2.1.11.3**, sulla base di quanto illustrato in **Fig.2.1.11.3**, non sono indicati i componenti hardware e software che inizializzano il dispositivo di interfaccia. La descrizione che segue presume che, in qualche modo, siano fissati i parametri della trasmissione (baud rate, tipo di parità, bit del dato, bit di stop).

Le parti hardware mostrate in questo schema sono:

- i registri (buffer) di ingresso **RX** e di uscita **TX**, ad ognuno dei quali è associata la logica che mostra il bit di stato e quello di mascheratura necessari a generare e controllare gli interrupt;
- due **buffer circolari** in memoria, uno associato alla linea e al registro di ingresso **RX** e l'altro associato alla linea e al registro di uscita **TX**;
- un'area di memoria (**AREA**) dalla quale vengono prelevate le stringhe di caratteri da inviare in uscita e nella quale memorizzare le stringhe che arrivano all'ingresso.

Le parti software realizzano quattro processi praticamente indipendenti. I primi due costituiti dalle due routine di risposta agli interrupt prodotti dal trasmettitore e dal ricevitore, rispettivamente **ISR_TRANSMITTER** e **ISR_RECEIVER**. La

prima è la risposta al segnale di interrupt provocato dallo svuotamento del registro **TX**, quando è completata la trasmissione di un carattere (**TX** vuoto). La seconda è la risposta al segnale di interrupt provocato dal riempimento del registro **RX**, quando si è completata la ricezione di un nuovo carattere (**RX** pieno). La **ISR_TRANSMITTER** preleva ogni carattere da inviare all'uscita seriale dal buffer circolare **BUFFER OUTPUT**, mentre la **ISR_RECEIVER** memorizza nel buffer circolare **BUFFER INPUT** ogni carattere ricevuto dall'ingresso seriale.

Gli altri due processi sono costituiti da altre due routine **INPUT** e **OUTPUT** utilizzabili, rispettivamente, da qualunque programma per leggere eventuali stringhe presenti sul **BUFFER INPUT** e inviare stringhe all'uscita seriale previa memorizzazione nel **BUFFER OUTPUT**.

Ad una routine, non mostrata in figura (il cui nome potrebbe essere ad esempio **OPEN_COM**), potrebbe essere assegnato il compito di inizializzare la trasmissione programmando adeguatamente dei registri (non indicati) in modo da rispettare il corretto protocollo di comunicazione (baud rate, tipo di parità, bit del dato, bit di stop).

La logica dell'intero sistema si può così riassumere:

ricevitore

gli interrupt del ricevitore sono sempre abilitati (**RX_MASK=1**). Al completamento della ricezione di tutti i bit del carattere viene posto **RX_STATO=1** che produce un interrupt al quale risponde la routine **ISR_RECEIVER**. Questa routine si serve, a sua volta, della subroutine **INS_BUF_INPUT**, che gestisce il buffer circolare **BUFFER INPUT**, per inserire il carattere purché il buffer non sia pieno. Nel caso il buffer circolare del ricevitore pieno, di caratteri non ancora letti, la subroutine **INS_BUF_INPUT** rinuncia all'inserimento ed il carattere acquisito dalla seriale viene perso. La routine **INPUT** può essere utilizzata da qualunque programma che voglia leggere il buffer circolare per vedere se ci sono caratteri da leggere. In questo modo il buffer circolare **BUFFER INPUT** è una risorsa utilizzata, indipendentemente, dalla routine **ISR_RECEIVER** che vi inserisce caratteri (*processo produttore*) e dalla routine **INPUT**, che legge caratteri (*processo consumatore*).

trasmettitore

gli interrupt del trasmettitore sono abilitati solo dalla routine **OUTPUT** che pone nel **BUFFER OUTPUT**, con l'uso della subroutine **INS_BUF_OUTPUT**, i caratteri da trasferire, e successivamente abilita gli interrupt del trasmettitore ponendo **TX_MASK=1** (in senso figurato si può dire che “accende” il trasmettitore). La routine **ISR_TRANSMITTER**, richiamata ad ogni svuotamento di **TX**, che provoca **TX_STATO=1**, preleva dal buffer circolare del trasmettitore, servendosi della subroutine **EXT_BUF_OUTPUT**, il carattere da inviare successivamente. Questo processo continua fino allo svuotamento completo di **BUFFER OUTPUT**, dopo il quale la routine **ISR_TRANSMITTER** spegne il trasmettitore, ponendo **TX_MASK=0**. In questo caso si hanno ancora due processi indipendenti: la routine **OUTPUT**, che funziona da processo produttore in quanto invia caratteri al buffer circolare **BUFFER OUTPUT**, e la routine **ISR_TRANSMITTER** che funziona da processo consumatore, in quanto legge caratteri dal buffer circolare e li invia al trasmettitore.

Da quanto descritto sia il canale seriale di ricezione (**RX** → **BUFFER INPUT** → AREA) sia quello di trasmissione (AREA → **BUFFER OUTPUT** → **TX**) utilizzano ognuno un buffer circolare per la comunicazione tra due processi indipendenti, rispettivamente **ISR_RECEIVER** → **BUFFER INPUT** → **INPUT** e **OUTPUT** → **BUFFER OUTPUT** → **ISR_TRANSMITTER**.

Le modalità con cui agisce questo sistema di comunicazione possono far pensare ad una analogia con il sistema di comunicazione postale. Il **BUFFER INPUT**, nel lato ricezione, funziona come la cassetta postale disponibile in una qualunque abitazione domestica. La routine **ISR_TRANSMITTER**, in funzione di portalettere, deposita nella “cassetta **BUFFER INPUT**” ogni carattere che arriva in **RX**. Il proprietario della cassetta postale, la routine **INPUT**, con tutto comodo e solo quando lo decide indipendentemente dal portallettere, preleva dalla “cassetta **BUFFER INPUT**” la posta arrivata. Si possono, comunque, rilevare, rispetto al sistema postale due anomalie: primo il portallettere di questo sistema elimina la posta quando trova la cassetta piena, secondo, il “proprietario” che trova la cassetta vuota aspetta finché il postino non la riempie. Questo certamente non è il modo con cui vogliamo che funzionino le nostre cassette postali !!

Buffer Circolare

E' necessario chiarire cosa si deve intendere per buffer circolare e cosa si intende per buffer pieno e buffer vuoto per comprendere il meccanismo di comunicazione descritto.

Per buffer circolare, vedi **Fig.2.1.11.4**, si intende un'area di memoria, delimitata da due locazioni di indirizzo **MIN_BUF** e **MAX_BUF**, formata da un certo numero di byte per la quale sono definite le seguenti regole di gestione:

scrittura nel buffer

- se il buffer non è pieno, scrittura nel buffer con l'uso di un puntatore **PIN_BUF** e aggiornamento del puntatore per la prossima scrittura nella locazione consecutiva;
- se l'aggiornamento del puntatore alla locazione da scrivere successivamente supera il limite superiore del buffer, **MAX_BUF**, allora il puntatore viene posto uguale al valore **MIN_BUF** in modo che la locazione successiva all'ultima scritta sia la prima del buffer. Per questo fatto l'area di memoria è come se si richiudesse su se stessa: circolare appunto!

lettura dal buffer

- se il buffer contiene almeno un carattere (non è vuoto), lettura del buffer con l'uso di un puntatore **POUT_BUF** e aggiornamento del puntatore per la prossima lettura nella locazione consecutiva;

- se l'aggiornamento del puntatore alla locazione da leggere successivamente supera il limite superiore del buffer, **MAX_BUF**, allora il puntatore viene posto uguale al valore **MIN_BUF** in modo che la locazione successiva all'ultima letta sia la prima del buffer;

buffer vuoto

- il buffer si ritiene vuoto quando, prima della lettura di un carattere, risulta **PIN_BUF=POUT_BUF**. In queste condizioni la prima operazione che si può fare è quella di inserimento, utilizzando **PIN_BUF**. In caso di inserimento di nuovo carattere, la diversità dei due puntatori, dovuta al successivo incremento del puntatore **PIN_BUF** codifica la condizione di buffer non più vuoto;

buffer pieno

- il buffer si ritiene pieno quando, dopo l'inserimento di un carattere, risulta **PIN_BUF=POUT_BUF**. In queste condizioni l'inserimento è annullato e **POUT_BUF** punta alla locazione immediatamente precedente (ultimo carattere inserito) a quella a cui punta **PIN_BUF** (primo carattere inserito). In queste condizioni la prima operazione che si può fare è quella di estrazione (lettura) di un carattere, utilizzando **POUT_BUF**. In caso di estrazione di nuovo carattere, la diversità dei due puntatori, dovuta al successivo incremento del puntatore **POUT_BUF** codifica la condizione di buffer non più pieno.

La **Fig.2.1.11.4** mostra la gestione del buffer circolare del ricevitore, **BUFFER INPUT**, risorsa comune delle routine **ISR_RECEIVER** e **INPUT**. Nella parte a) della figura è mostrata la situazione iniziale di buffer vuoto in quanto i due puntatori coincidono. Si suppone poi che il trasmettitore voglia trasmettere la frase **"Silvy tu sei bella alta"** e che la trasmissione avvenga trasmettendo le singole parole (stringhe) terminate dal carattere di controllo **carriage return (CR)**, perché la routine **INPUT** è stata scritta per acquisire singole parole.

Fissate queste condizioni nella figura è descritta la seguente sequenza:

- a) stato iniziale buffer vuoto
- b) il ricevitore (**ISR_RECEIVER**) ha ricevuto le prime due stringhe della frase **"Silvy"** e **"tu"** che con l'uso del puntatore **PIN_BUF** vengono poste nel buffer
- c) il software del ricevitore manda in esecuzione la routine **INPUT** che, servendosi del puntatore **POUT_BUF**, legge la stringa **"Silvy"** dal buffer circolare e la pone in un'area di memoria (AREA) per eventuali elaborazioni (variabile stringa).
- d) il ricevitore ha ricevuto la terza parola **"sei"**, che viene posta nel buffer. In questa circostanza si osserva come la locazione del buffer successiva all'ultima (**MAX_BUF**) sia la prima locazione (**MIN_BUF**)
- e) il ricevitore riceve la parola **"bella"**. Ciò causa il riempimento del buffer dopo l'introduzione dei primi due caratteri **"be"**, per cui al tentativo di inserire il terzo carattere **"l"** la subroutine di inserimento si accorge che il buffer è pieno (coincidenza tra i due puntatori) e rinuncia a questa operazione (malissimo se ciò avvenisse con la nostra cassetta postale !!), lasciando il puntatore **PIN_BUF** alla locazione precedente. La condizione di buffer pieno si ha, quindi, quando sono state scritte tutte le locazioni del buffer meno una.
- f) viene mandata in esecuzione la routine **INPUT** che legge la parola **"tu"**
- g) viene mandata in esecuzione ancora la routine **INPUT** che legge la parola **"sei"**.
- h) viene invocata ancora la **INPUT**, che legge gli ultimi due caratteri inseriti **"be"** nel punto e). Questa operazione svuota completamente il buffer.
- i) il ricevitore seriale inserisce l'ultima parola, **"alta"**, della frase da trasmettere. La successiva chiamata della routine **IN_COM** legge l'ultima stringa arrivata. La frase ricevuta a questo punto è **"Silvy tu sei bealta"**.

L'errore in ricezione può essere eliminato prendendo vari provvedimenti. Il più semplice è quello di prevedere un buffer sufficientemente ampio in modo che non si possa mai riempire completamente. Con un colloquio più articolato il ricevitore potrebbe richiedere la ritrasmissione della frase.

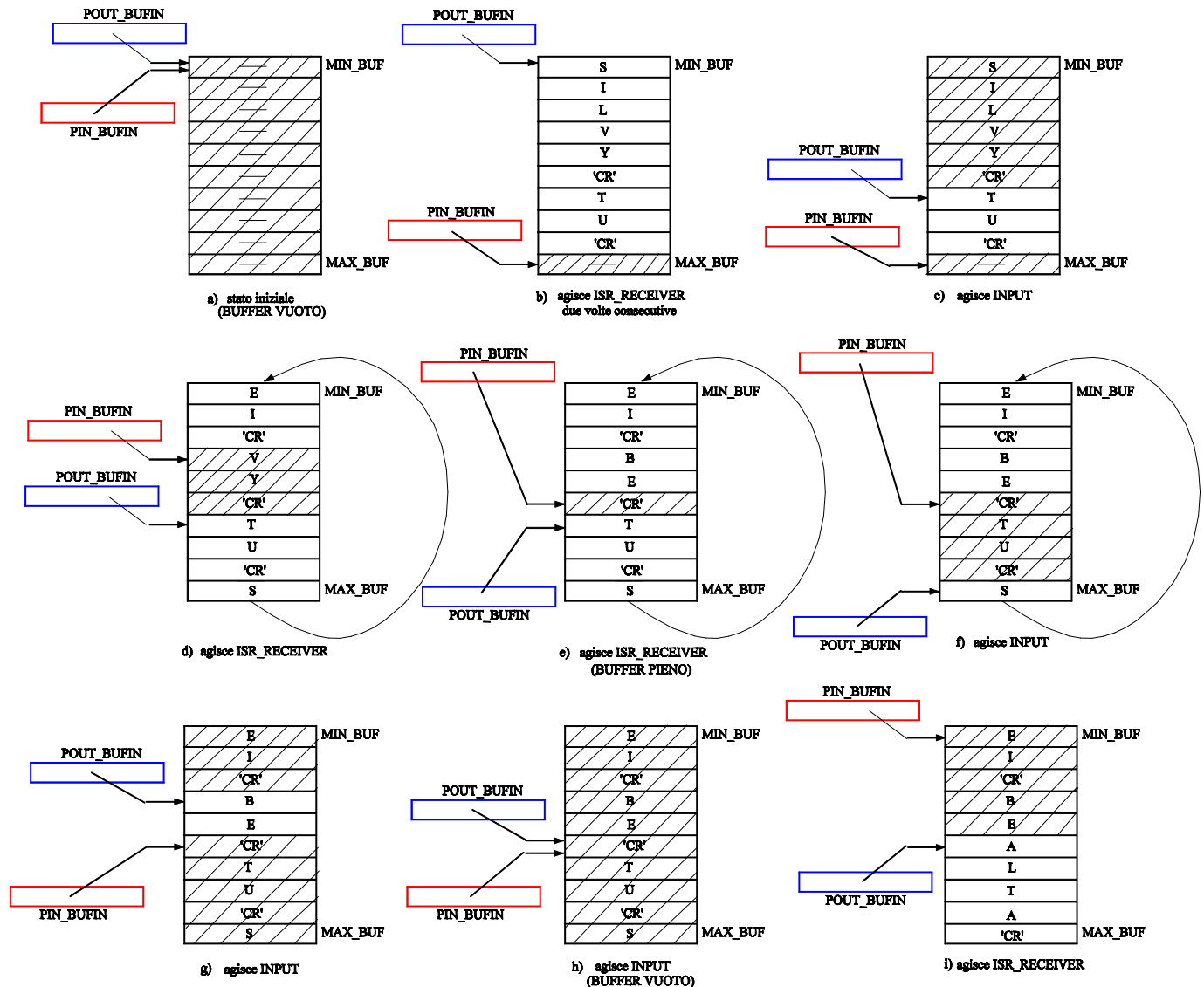


Fig.2.1.11.4 Immagini che illustrano il buffer circolare del ricevitore seriale di Fig.2.1.11.3 utilizzato dalle routine **ISR_RECEIVER** (*produttore*) che vi scrive caratteri e dalla **INPUT** che li legge (*consumatore*).

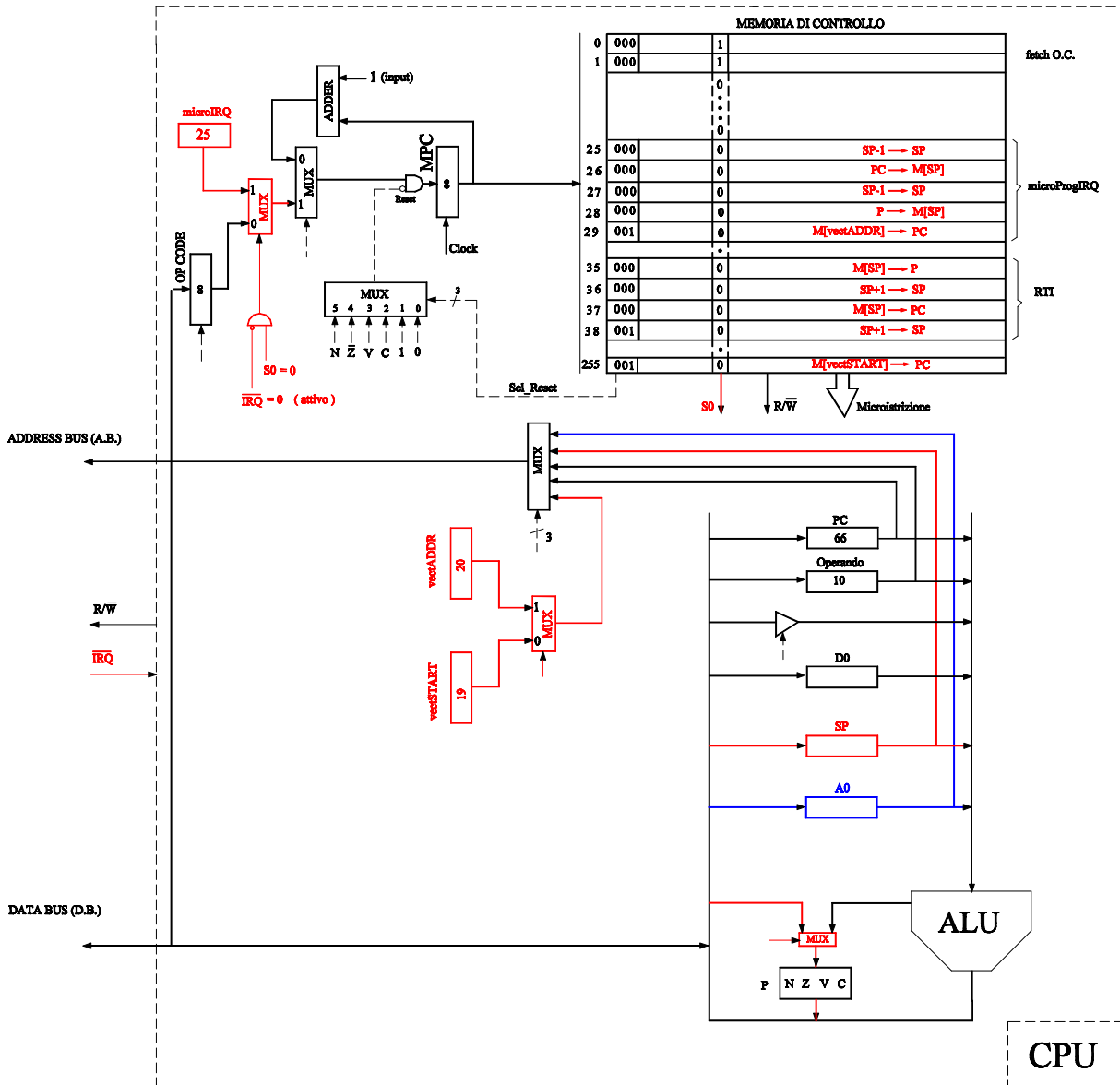
La gestione del buffer di trasmissione **BUFFER OUTPUT** è, ovviamente, fatta con le stesse regole viste per quello di ricezione.

Descritta la logica delle routine di gestione della linea seriale, si lascia come esercizio, per la macchina sin qui descritta, la scrittura in linguaggio assembly delle routine di risposta agli interrupt sia del ricevitore **ISR_RECEIVER** che del trasmettitore **ISR_TRANSMITTER**. Questo linguaggio consente di ottimizzare i tempi di esecuzione, prerogativa fondamentale delle risposte ai segnali di interruzione che hanno esigenze di essere serviti in tempo reale.

Le rimanenti routine caratterizzate da logiche più elaborate, potrebbero essere scritte in linguaggio C. Successivamente, dove ciò fosse necessario, per rispettare più stretti requisiti di risposta in tempo reale, si potrebbe effettuare la compilazione di queste routine in linguaggio assembly.

Routine di gestione della comunicazione seriale

Per rendere più agevole la scrittura del software si arricchisce l'hardware della CPU aggiungendo il registro **A0** che è utilizzabile per contenere l'indirizzo di una locazione di memoria (come **PC**, **operando**, **SP**, **vectStart** e **vectADDR**). La Fig.2.1.11.5 mostra questa variante rispetto alla Fig.2.1.8.2.



E' usata inoltre l'istruzione "**CMP #MAX_BAFIN, A0**" (e altre simili) il cui codice operativo (mnemonico **CMP** : **CoMPare**) che comanda di fare una sottrazione tra il contenuto del registro **A0** e l'operando **MAX_BUFIN** con il solo scopo di attivare il segnalatori aritmetici senza memorizzare il risultato in nessun registro.

routine del canale di ricezione seriale

; Risposta agli interrupt del ricevitore

```
ISR_RECEIVER:  MOVE    D0, -(SP)           ; Salva il contesto
                MOVE    IND_RX, D0         ; Leggi dato. Reset RX_STATO
                JSR      INS_BUF_INPUT     ; Chiama subroutine di inserimento nel buffer. Se pieno pazienza
                MOVE    (SP)+, D0         ; Ripristina contesto
                RTI
```

; Inserisce un carattere nel **BUFFER INPUT**.

; se il buffer non è pieno, altrimenti rinuncia

```
INS_BUF_INPUT: MOVE    A0, -(SP)           ; Salva contesto
                MOVE    PIN_BUFIN, A0      ; Poni in A0 puntatore per inserzione
                MOVE    D0, (A0)+          ; Inserisci carattere e avanza di una posizione
                CMP      #MAX_BAFIN, A0    ; Oltrepassato il fondo ?
                BMI      INS_Q_IN
                BEQ      INS_Q_N
                MOVE    #MIN_BUFIN, A0     ; Riporta puntatore in cima
INS_Q_IN:      CMP      POUT_BUFIN, A0     ; La coda è piena
                BEQ      INS_EXIT_IN       ; Salta se si
                MOVE    A0, PIN_BUFIN      ; La coda non era piena. Accetta inserzione e aggiorna puntatore
                MOVE    (SP)+, A0          ; Ripristina contesto
                RTS
INS_EXIT_IN:   MOVE    (SP)+, A0           ; Ripristina contesto
                RTS
```

; Routine di acquisizione di una stringa dall'ingresso seriale

; La stringa inviata deve essere terminata con 'CR'.

; Il programma chiamante "passa" in A0 il puntatore

; all'area di memoria in cui porre la stringa.

```
INPUT:        MOVE    D0, -(SP)           ; Salva contesto
                MOVE    A0, -(SP)
GET_CAR:      JSR      EXT_BUF_INPUT       ; Chiama subroutine per estrarre un carattere dal BUFFER INPUT
                MOVE    FLAG, D0
                BNE      GET_CAR           ; Se FLAG=1 buffer vuoto, ritenta
                MOVE    oneCar, D0
                CMP      #13, D0           ; Se CR la stringa è finita
                BEQ      IN_EXIT
                MOVE    D0, (A0)+          ; Non è CR. Memorizza dato e aggiorna puntatore
                JMP      GET_CAR
IN_EXIT:      MOVE    #0, (A0)            ; Tutta la stringa è arrivata. Poni terminatore NULL=$00
                MOVE    (SP)+, A0          ; Ripristina contesto
                MOVE    (SP)+, D0
                RTS
```

; Estrae un carattere dal **BUFFER INPUT**.
 ; Il carattere è passato al chiamante nella locazione **oneCAR**
 ; Se l'estrazione ha successo restituisce **FLAG=0**, altrimenti **FLAG=1**

```

EXT_BUF_INPUT: MOVE    A0, -(SP)          ; Salva contesto
                MOVE    POUT_BUFIN, A0    ; Poni in A0 puntatore per estrazione
                CMP      PIN_BUFIN, A0     ; La coda è vuota ?
                BEQ      EXT_EXIT          ; Salta se sì
                MOVE     (A0)+, D0         ; Estrai il carattere e avanza di una posizione
                MOVE     D0, oneCAR
                CMP      #MAX_BUFIN, A0    ; Oltrepasato il fondo ?
                BMI      EXT_Q             ; Salta se no
                BEQ      EXT_Q
EXT_Q:          MOVE     #MIN_BUFIN, A     ; Riporta puntatore in cima
                MOVE     A0, POUT_BUFIN    ; Aggiorna puntatore per estrazione avvenuta
                MOVE     (SP)+, A0         ; Ripristina contesto
                MOVE     #0, D0            ; Poni FLAG a zero
                MOVE     D0, FLAG
                RTS
EXT_EXIT:       MOVE     (SP)+, A0         ; Ripristina contesto
                MOVE     #1, D0            ; Poni FLAG a uno
                MOVE     D0, FLAG
                RTS

```

routine del canale di trasmissione seriale

; Risposta agli interrupt del trasmettitore.

```

ISR_TRANSMITTER: MOVE    D0, -(SP)        ; Salva contesto
                  JSR      EXT_BUF_OUTPUT  ; Chiama subroutine estrazione carattere
                  MOVE     FLAG, D0
                  BNE      BUF_VUOTO       ; Salta se buffer vuoto. Non vi è più niente da trasmettere
                  MOVE     D0, IND_TX      ; Poni carattere su TX. Reset TX_STATO
                  JMP      ISR_TR_OUT
BUF_VUOTO:       MOVE     #0, D0          ; Non ci sono più caratteri:
                  MOVE     D0, IND_MASK    ; .....spegni trasmettitore: TX_MASK=bit3=0
ISR_TR_OUT:      MOVE     (SP)+, D0
                  RTI

```

; Estrae un carattere dal **BUFFER OUTPUT**.
 ; Il carattere è passato al chiamante nella locazione **oneCAR**
 ; Se l'inserimento ha successo restituisce **FLAG=0**, altrimenti **FLAG=1**

```
EXT_BUF_OUTPUT: MOVE    A0, -(SP)           ; Salva contesto
                  MOVE    POUT_BUFOUT, A0   ; Poni in A0 puntatore per estrazione
                  CMP     PIN_BUFOUT, A0     ; La coda è vuota ?
                  BEQ     EXT_EXIT_OUT      ; Salta se sì
                  MOVE     (A0)+, D0         ; Estrai il carattere e avanza di una posizione
                  MOVE     D0, oneCAR
                  CMP     #MAX_BUFOUT, A0   ; Oltrepasato il fondo ?
                  BMI     EXT_Q_OUT         ; Salta se no
                  BEQ     EXT_Q_OUT
EXT_Q_OUT:        MOVE     #MIN_BUFOUT, A0   ; Riporta puntatore in cima
                  MOVE     A0, POUT_BUFOUT   ; Aggiorna puntatore per estrazione avvenuta
                  MOVE     (SP)+, A0         ; Ripristina contesto
                  MOVE     #0, D0           ; Poni FLAG a zero
                  MOVE     D0, FLAG
EXT_EXIT_OUT:     RTS
                  MOVE     (SP)+, A0         ; Ripristina contesto
                  MOVE     #1, D0           ; Poni FLAG a uno
                  MOVE     D0, FLAG
                  RTS
```

; Routine di trasmissione di una stringa dall'uscita seriale
 ; La stringa raccolta dalla zona di memoria AREA è terminata con 'CR'.
 ; Il programma chiamante "passa" in **A0** il puntatore
 ; all'area di memoria AREA da cui prelevare la stringa.

```
OUTPUT:  MOVE     A0, -(SP)           ; Salva contesto
OUT_CM1: MOVE     (A0)+, D0           ; Leggi carattere e inseriscilo in BUFFER OUTPUT
          CMP     #0, D0              ; Se NULL inserisci CR
          BNE     OUT_CAR
          MOVE     #13, D0            ; Spedisci CR al posto di NULL
OUT_CAR: JSR      INS_BUF_OUTPUT      ; Poni carattere nel buffer circolare
          MOVE     FLAG, D0
          BNE     OUT_CAR             ; Se pieno ritenta
          CMP     #13, D0             ; E' terminata la stringa ?
          BNE     OUT_CM1
          MOVE     #1, D0             ; Inserimento completato:
          MOVE     D0, IND_MASK       ; ...accendi trasmettitore ponendo TX_MASK=1
          MOVE     (SP)+, A0         ; Ripristina contesto
          RTS
```

; Inserisce un carattere nel BUFFER OUTPUT.
 ; Il carattere è passato dal chiamante nel registro D0
 ; Se l'inserimento ha successo restituisce FLAG=0, altrimenti FLAG=1

```

INS_BUF_OUTPUT: MOVE    A0, -(SP)           ; Salva contesto
                  MOVE    PIN_BUFOUT, A0    ; Poni in A0 puntatore per inserzione
                  MOVE    D0, (A0)+         ; Inserisci carattere e avanza di una posizione
                  CMP     #MAX_BAFOUT, A0    ; Oltrepasato il fondo ?
                  BMI      INS_Q_OUT        ; Salta se no
                  BEQ      INS_Q_OUT
INS_Q_OUT:      MOVE     #MIN_BUFOUT, A0    ; Riporta puntatore in cima
                  CMP     POUT_BUFOUT, A0   ; La coda è piena
                  BEQ      INS_EXIT_OUT     ; Salta se si
                  MOVE     A0, PIN_BUFOUT    ; La coda non era piena. Accetta inserzione e aggiorna puntatore
                  MOVE     (SP)+, A0         ; Ripristina contesto
                  MOVE     #0, D0           ; Poni FLAG a zero
                  MOVE     D0, FLAG
                  RTS
INS_EXIT_OUT:  MOVE     (SP)+, A0         ; Ripristina contesto
                  MOVE     #1, D0           ; Poni FLAG a uno
                  MOVE     D0, FLAG
                  RTS

```