ESERCITAZIONE USART

Nella cartella degli esercizi troviamo 4 esercizi, abbiamo S1 e S2 che comunicano con periferica seriale. In un primo esercizio in ricezione c'è l'interruzione, in trasmissione solo un ciclo, in un secondo esercizio abbiamo le interruzioni anche in trasmissione. C'è poi un esercizio per il comportamento sincrono e uno per utilizzare il terminale, anche se dal punto di vista delle interruzioni non c'è grande differenza.

Comunicazione seriale asincrona con interruzione solo in ricezione

Sistema di trasmissione - S1: nell'area dati c'è il messaggio che vogliamo trasmettere, abbiamo usato configurazioni binarie che fossero facili da vedere in simulazione, dato che si manda un bit alla volta. Mettiamo anche la dimensione del messaggio e il contatore dei dati inviati.

```
**********************************************************
**********AREA DATI*****************************************
    ORG $8000
MSG DC.B    170,204,153,129,195,6
DIM DC.B    6
COUNTINV    DC.B    0 ;contatore caratteri inviati
```

Nell'area main abbiamo gli indirizzi della periferica seriale, che espone solo 2 locazioni di memoria, abbiamo l'indirizzo pari 2004 e dispari 2005, usato per il registro di controllo/stato e di modo. *L'indirizzo pari è sempre associato al registro dati della periferica.* La prima cosa che facciamo nell'area main è sistemare nei parametri USARTD e USARTC gli indirizzi di partenza della periferica. Poi, la prima operazione da fare è **inizializzare la periferica**, andiamo a settare il modo di funzionamento e diamo una serie di segnali di controllo iniziali che servono a far lavorare correttamente la periferica.

```
USARTD  EQU     $2004       ;registro dato USART
USARTC  EQU     $2005       ;registro di controllo USART

MAIN    JSR     INITUSART   ;inizializza USART
```

```
PRIMO ACCESSO IN SCRITTURA INDIRIZZO DISPARI => REGISTRO MODE

MODE |  0  |  1  |  0  |  1  |  1  |  1  |  0  |  1  |
                                                  |_____b0=0 Trasmissione Sincrona, b0=1 Asincrona
                                                  |_____b1 non utilizzato
                                                  |_____b3b2="00"(5bit/dato), "01"(6bit/dato), "10"(7bit/dato), "11"(8bit/dato)
                                                  |_____b4=0: non trasmette bit di parità, b4=1: lo trasmette
                                                  |_____b5=0: parità pari, b5=1: parità dispari
                                                  |_____b6=0: 1 bit di stop, b6=1: 2 bit di stop
                                                  |_____b7=0: 1 carattere di sync, b7=1: 2 caratteri di sync (in trasmissione sincrona)
```

Per il modo (figura sopra), da destra a sinistra vogliamo settare la **trasmissione asincrona**, poi settiamo informazioni legate a **quanti bit di dato trasmettiamo** alla volta (8,7,6 o 5 bit, 4 configurazioni possibili), abbiamo il bit del registro di modo che indica **se trasmettere il bit di parità, che tipo di parità e quanti bit di stop** servono per il carattere stesso. Possiamo scegliere tra 1 bit di stop o 2. Nel registro modo possiamo anche configurare le informazioni che andiamo a usare per il sincrono, il bit 7 non lo usiamo perché siamo in modalità asincrona. Nella nostra configurazione usiamo 8 bit di informazione, per cui in b3b2 mettiamo 11, poi vogliamo la parità e la vogliamo dispari, abbiamo 2 bit di stop. Anche lato ricevente faremo la stessa configurazione. In simulazione vedremo come vengono passati nello shift-register i bit di dato vero ma poi **c'è un'ulteriore latenza necessaria a trasmettere i bit di parità e di stop**. ***Facciamo l'accesso all'indirizzo dispari, dopo il reset, quindi accediamo nel registro mode, i successivi accessi a indirizzo dispari vanno all'indirizzo di controllo***, dove possiamo abilitare e disabilitare trasmissione e ricezione, perché la periferica e full duplex.

```
SECONDO ACCESSO IN SCRITTURA INDIRIZZO DISPARI => REGISTRO CNTRL

MODE |  0  |  0  |  1  |  0  |  0  | 0  |  1  |  1  |
                                                  |_____b0=1: abilita trasmettitore, b0=0: disabilita trasmettitrore
                                                  |_____b1=1: attiva DTR, b1=0: disattiva DTR
                                                  |_____b2=1: abitilita ricevitore, b0=0: disabilita ricevitore
                                                  |_____b3 non utilizzato
                                                  |_____b4=1: cancella bit di errore, b4=0: non cancella
                                                  |_____b5=1: attiva RTS, b6=0: disattiva RTS
                                                  |_____b6=1: resetta l'interfaccia seriale
                                                  |_____b7=1: porta il ricevitore nello stato HUNT in cui cerca i caratteri di sincronismo sync
                                                        (in comunicazione sincrona)
```

Questo è il codice del trasmettitore per cui **abilitiamo la trasmissione e disabilitiamo la ricezione**. Un'altra cosa che si fa è settare i segnali per l'handshaking, ovvero b3, DTR e RTS. Questi sono stati concepiti per la comunicazione con il modem, che aveva gli stessi segnali, la coppia DTR DSR serviva per stabilire la connessione sulla linea, invece la coppia ready to send e clear to send (RTS e CTS) serviva per fare il trasferimento e si può usare anche un controllo di flusso. Per Asim si usa una configurazione particolare, detta *no modem*(?), in cui il modem non viene usato, ***le due periferiche seriale sono direttamente connesse e il DTR di una finisce nel DSR dell'altra, e viceversa, RTS di una va nel CTS dell'altra***. Sono collegate in modo incrociato. **Nel registro di controllo porremo direttamente a 1 il bit DTR e a 1 RTS, per cui stabiliamo in modo fisso l'handshaking**, le due periferiche sono disponibili a cambiare messaggi. La configurazione no modem, la connessione fisica dei fili è fatta da Asim, dove diamo dei dettagli in più sulla connessione fisica. Dobbiamo abilitare il trasmettitore, attivare il DTR, disabilitare il ricevitore, poi possiamo settare b4 per cancellare l'errore ma ha senso in ricezione, poi attiviamo RTS, poi c'è un altro bit che si usa in ricezione per dire che il dispositivo si mette in attesa per ricevere i caratteri di sync, 1 o 2, configurati opportunamente, per cui il ricevitore si aspetta di ricevere prima i sync e poi lo stream di dati, poi di nuovo i sync dopo un certo numero di caratteri ricevuti.

Nella parola 23 in esadecimale mettiamo i segnali visti, dunque con questo comando di move stiamo settando il registro di controllo. Dopo aver inizializzato la periferica, è pronta a trasmettere.

```
MOVE.B      #$23,USARTC abilita trasmettitore
                    e attiva i segnali di handshaking.

RTS
```

Nel codice vengono fatte una serie di operazioni, la prima **azzera il registro di stato**, perché Asim parte con la maschera di abilitazione 111, per cui le interruzioni non sarebbero viste. Lato trasmittente ora non abbiamo interruzioni ma la seriale ha una particolarità: con la PIA tramite un apposito bit del registro di controllo disabilitavamo le interruzioni irqb, ora **abbiamo due interruzioni, associate a *trasmissione_ready (TxRDY)*, per cui il carattere è stato inviato e siamo pronti a darne un altro, invece *ricezione_ready (RxRDY)* indica che il dato è andare nel registro parallelo, pronto per essere letto dal processore**. ***Per questo dispositivo non possiamo disabilitare l'interruzione però comunque possiamo non farle fare niente, è vuota***, ma non si può disabilitare. Per prima cosa azzeriamo il registro di stato così da lavorare con le interruzioni di ogni livello. Successivamente, **mettiamo gli indirizzi che ci interessano nei registri interni**. Poi ci sono due cicli, checkdsr in realtà è superfluo in questo caso, è un ciclo che *controlla se il segnale DSR che viene dall'altra USART è attivo*, ovvero il collegamento sulla linea è attivo. **Da che usiamo una configurazione in cui, tramite il registro di controllo, forziamo ad entrambi gli estremi DSR sempre alto, la condizione è sempre vera.** Questo codice lo mettiamo per capire che in teoria prima di trasmettere dovremmo verificare che la connessione sia correttamente stabilita. **Non abbiamo un controllo anche di RTS e CTS ma anche qui non ci serve perché tramite il registro di controllo pure RTS è sempre alto, l'handshaking è stabilito**. Il secondo loop invece serve perchè il concetto è che *prima di poter trasmettere un nuovo carattere, quindi mandare un nuovo dato nel registro dato della seriale, dobbiamo assicurarci che il precedente sia correttamente inviato*. Nella PIA andavamo a guardare crb7, quando era alto aveva letto, c'era un handshaking, invece nella seriale non è un handshaking di questo tipo perché **l'assunto che facciamo per far funzionare la periferiche e che lavorino in modo sincrono, legato alla detection dei bit di sync nel singolo carattere** ma di fatto c'è una base dei tempi comune. *Come fa il trasmittente a capire che il ricevitore ha ricevuto il carattere?* **S1 non ha informazioni da s2 ma sa quanto tempo ci mette a mandare fuori i singoli bit del carattere che sta trasmettendo**. Il carattere va nel registro dato della seriale, viene messo nello shift-register, viene mandato un bit alla volta con la frequenza della seriale, oltre agli 8 bit del carattere mandiamo anche lo start bit all'inizio e poi bit di parità e di stop configurato nel registro di modo. ***Sapendo quanto tempo ci vuole, perché è un dispositivo clockato, il tx_ready si alza quando il carattere precedente è stato completamente trasferito all'esterno***, allora vediamo costa accade a txrdy. Allora *il ciclo checktxrdy prende di volta in volta il registro di stato, messo in D0 e facciamo la and con 01, che individua txrdy, quando è alto usciamo dal ciclo e facciamo l'invio.* All'inizio troviamo il bit tale da poter inviare, per cui il ciclo vale negli altri invii. *Per fare l'invio (figura a sinistra) prendiamo il dato, o lo mettiamo nel registro dato della seriale, incrementiamo il contatore dei dati trasmessi, vediamo se abbiamo finito e se non abbiamo finito torniamo a txrdy, aspettiamo che torni alto per fare un nuovo invio.* Questo è tutto quello che dobbiamo fare per cui ci possiamo mettere in loop jmp loop. Alla fine troviamo 8800 che è l'interruzione associata a txrdy, che dobbiamo mettere per forza ma non la usiamo, infatti il corpo è vuoto (figura a destra).

```
MOVE.W  SR,D0   ;legge il registro di stato
ANDI.W  #$D8FF,D0 ;maschera per reg stato (stato utente, int abilitati)
MOVE.W  D0,SR   ;pone liv int a 000
```

```
MOVEA.L #USARTD,A1  ;indirizzo registro dato
MOVEA.L #USARTC,A2  ;indirizzo registro controllo/stato

MOVEA.L #MSG,A0 ;indirizzo area messaggio
MOVE.B  DIM,D3
CLR D1  ;appoggio
CLR D2  ;contatore elementi trasmessi
```

```
CHECKDSR    MOVE.B  (A2),D0
    ANDI.B      #$80,D0
*
*
*
    BEQ     CHECKDSR
```

```
CHECKTxRDY  MOVE.B  (A2),D0
    ANDI.B      #$01,D0
    BEQ     CHECKTxRDY
```

```
INVIO   MOVE.B  (A0)+,D1
    MOVE.B  D1,(A1)
    ADD.B       #1,D2

    CMP D2,D3
    BNE CHECKTxRDY

LOOP    JMP LOOP
```

```
    ORG $8800

    RTE


    END MAIN
```

[Sistema di ricezione - S2](): predisponiamo l'area di memoria per mettere i caratteri, la dimensione e mettiamo il contatore **come variabile globale in memoria perché di volta in volta l'interruzione userà il contatore come aiuto per capire la posizione corretta in cui inserire il carattere ricevuto**. L'area main (a sinistra) è simile a quella vista dall'altro lato, prendiamo gli indirizzi della periferica e inizializziamo.

```
**********AREA DATI********
    ORG $8000
MSG DS.B    6
DIM DC.B    6
COUNTRIC    DC.B    0
```

```
USARTD  EQU     $2004
USARTC  EQU     $2005

MAIN    JSR     INITUSART
```

Per il registro di modo dobbiamo fare la stessa cosa fatta in trasmissione, **per il controllo** (figura in basso) **invece ora abilitiamo in ricezione, il primo bit allora è 0 (trasmissione disabilitata) mentre il 3° bit è alto**. Invece DTR e RTS sono posti a 1, stabiliamo la connessione fisica con i segnali alti, in entrambi i casi.

```
SECONDO ACCESSO IN SCRITTURA INDIRIZZO DISPARI => REGISTRO CNTRL

MODE |  0  |  0  |  1  |  1  |  0  | 1  |  1  |  0  |
                |     |     |     |     |     |     |     |____b0=1: abilita trasmettitore, b0=0: disabilita trasmettitrore
                |     |     |     |     |     |     |_____ b1=1: attiva DTR, b1=0: disattiva DTR
                |     |     |     |     |     |___________b2=1: abitilita ricevitore, b0=0: disabilita ricevitore
                |     |     |     |     |_____________b3 non utilizzato
                |     |     |     |_________________b4=1: cancella bit di errore, b4=0: non cancella
                |     |     |_____________________b5=1: attiva RTS, b6=0: disattiva RTS
                |     |_______________________b6=1: resetta l'interfaccia seriale
                |_________________________b7=1: porta il ricevitore nello stato HUNT in cui cerca i caratteri di sincronismo sync
                          (in comunicazione sincrona)
```

Facciamo l'inizializzazione e poi azzeriamo il registro di stato e **andiamo a fare il controllo della connessione, poi ci mettiamo in attesa attiva, saremo svegliati dall'interruzione in ricezione**. Vediamo com'è fatta l'interruzione (codice a sinistra), dobbiamo prendere quello che arriva dal registro dato e lo mettiamo in memoria nella locazione giusta, *il dato in ingresso è quello che prendiamo dal registro dato della seriale, il cui indirizzo è messo in A1 e per la destinazione usiamo il modo di indirizzamento indiretto con registro indice, in a0 aggiungiamo il contenuto di d0 (contatore dei messaggi) e andiamo in memoria* a questo indirizzo calcolato come somma. Incrementiamo il contatore dei caratteri ricevuti e rimettiamo a posto i registri usati sullo stack, ritornando all'esecuzione.

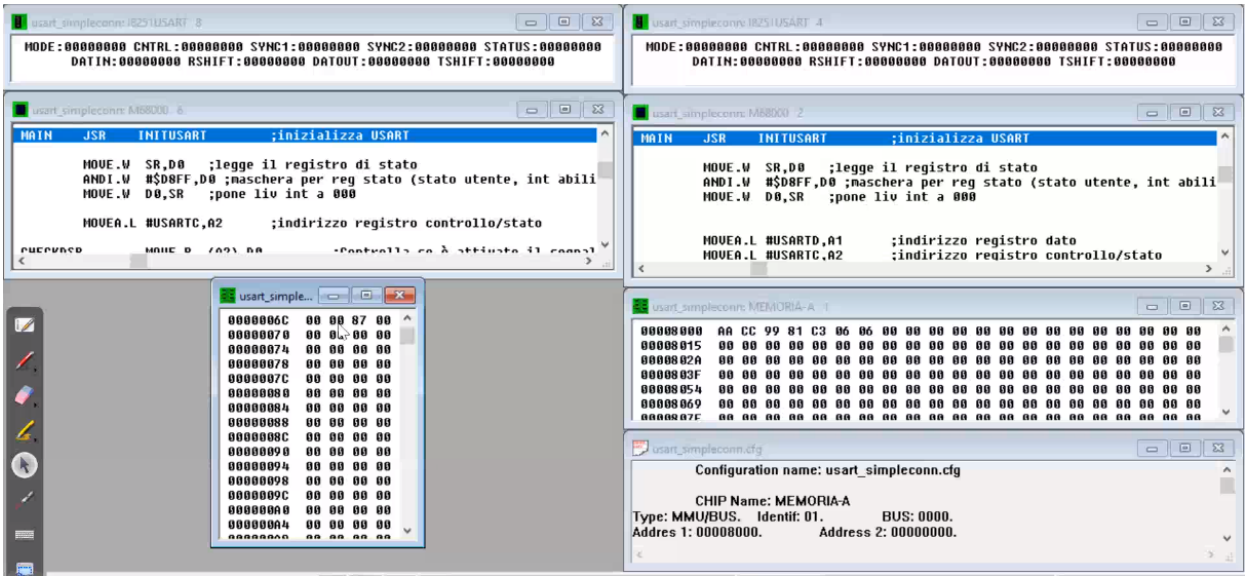
```
    ORG $8700

INT3        MOVE.L  A1,-(A7)        ;salvataggio registri
    MOVE.L  A0,-(A7)
    MOVE.L  D0,-(A7)

    MOVEA.L #USARTD,A1
    MOVEA.L #MSG,A0     ;indirizzo area di salvataggio
    MOVE.B  COUNTRIC,D0 ;contatore corrente degli elementi ricevuti

RICEZ   MOVE.B  (A1),(A0,D0)    ;riceve un carattere e lo memorizza

    ADD.B   #1,D0
    MOVE.B  D0,COUNTRIC ;aggiorna contatore caratteri ricevuti

FINE    MOVE.L  (A7)+,D0    ;ripristino registri
    MOVE.L  (A7)+,A0
    MOVE.L  (A7)+,A1

    RTE
```

```
    MOVE.W  SR,D0
    ANDI.W  #$D8FF,D0
    MOVE.W  D0,SR

    MOVEA.L #USARTC,A2

CHECKDSR    MOVE.B  (A2),D0
            ANDI.B      #$80,D0
*
*
*
            BEQ     CHECKDSR

LOOP    JMP LOOP
```

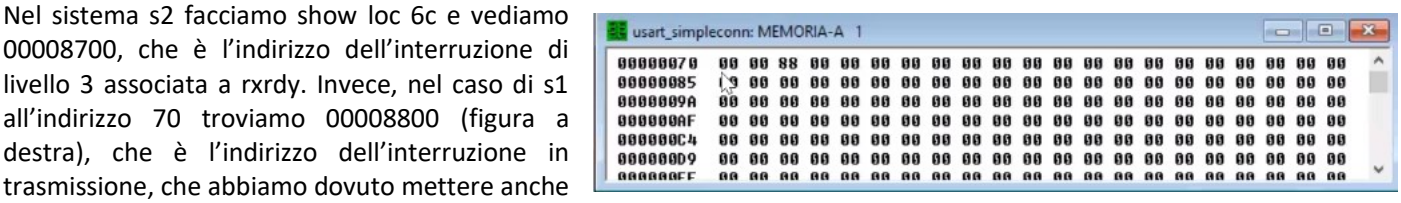
[Esecuzione in Asim:](#) come visto per la PIA, **ci sono due sistemi con gli stessi componenti**, il primo ha memoria con identificativo 1, il processore con identificato 2 connesso al bus 1 e la USART ha identificativo 4 ed è connesso al bus 1. **Nel dispositivo seriale in address1 e address2 mettiamo i due indirizzi assegnati alla periferica**, 2004 e 2005, *gli altri 3 dispositivi sono un altro sistema per cui possiamo usare gli stessi indirizzi.* **Nella com1 abbiamo l'identificativo del gestore delle interruzioni, per la prima USART è il processore 2**, invece **nella com2 e com3 abbiamo la linea di interruzione a cui il dispositivo è attaccato per rxrdy e txrdy**, rxrdy è attaccata alla linea di priorità 3 del processore, invece la linea txrdy è alla linea di priorità 4 del processore. ***Anche la USART lavora con gli autovettori***, mettiamo l'indirizzo della ISR dell'rxrdy nel 3° autovettore, ovvero il 27°, quindi all'indirizzo 6c. Invece, txrdy si trova in 70, autovettore 28. **In com4 invece troviamo l'identificativo dell'altra USART, con cui parla, negli altri due bit troviamo il tipo di connessione**, *lo 0 a destra indica la configurazione precedente, ovvero il DSR di s1 finisce nel DTR di s2, e viceversa, lo stesso anche per RTS e CTS. Infine, il segnale trasmissione del dato seriale in uscita di s1 va in ricezione del secondo, e viceversa, questa **configurazione "completo"** è identificato con lo 0 davanti*. La configurazione fisica dei fili va detta ad Asim che altrimenti non saprebbe come lavorare.

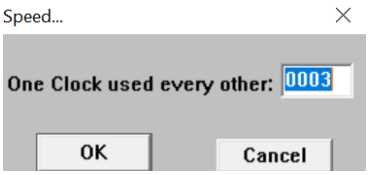
```
Configuration name: usart_simpleconn.cfg

CHIP Name: MEMORIA-A
Type: MMU/BUS.   Identif: 01.        BUS: 0000.
Addres 1: 00008000.          Address 2: 00000000.
Com1: 0000.  Com2: 0010.  Com3: 0008.  Com4: 0000.

CHIP Name: M68000
Type: CPU.       Identif: 02.        BUS: 0001.
Addres 1: 00009000.          Address 2: 00009200.
Com1: 0000.  Com2: 0000.  Com3: 0000.  Com4: 0000.

CHIP Name: I8251USART
Type: Device.    Identif: 04.        BUS: 0001.
Addres 1: 00002004.          Address 2: 00002005.
Com1: 0002.  Com2: 0003.  Com3: 0004.  Com4: 0008.

CHIP Name: MEMORIA-B
Type: MMU/BUS.   Identif: 05.        BUS: 0000.
Addres 1: 00008000.          Address 2: 00000000.
Com1: 0000.  Com2: 0010.  Com3: 0008.  Com4: 0000.

CHIP Name: M68000
Type: CPU.       Identif: 06.        BUS: 0005.
Addres 1: 00009000.          Address 2: 00009200.
Com1: 0000.  Com2: 0000.  Com3: 0000.  Com4: 0000.

CHIP Name: I8251USART
Type: Device.    Identif: 08.        BUS: 0005.
Addres 1: 00002004.          Address 2: 00002005.
Com1: 0006.  Com2: 0003.  Com3: 0004.  Com4: 0004.
```

Esplodiamo la configurazione, generando i chip e per ogni USART vediamo tutto il modello di programmazione, troviamo il registro modo, di controllo, i s2 sync e il registro di stato. Dato che ogni USART lavora sia in ricezione che in trasmissione troviamo datin rshift, datout tshift in entrambe le USART. Carichiamo i file dei due sistemi e poi carichiamo le memorie.

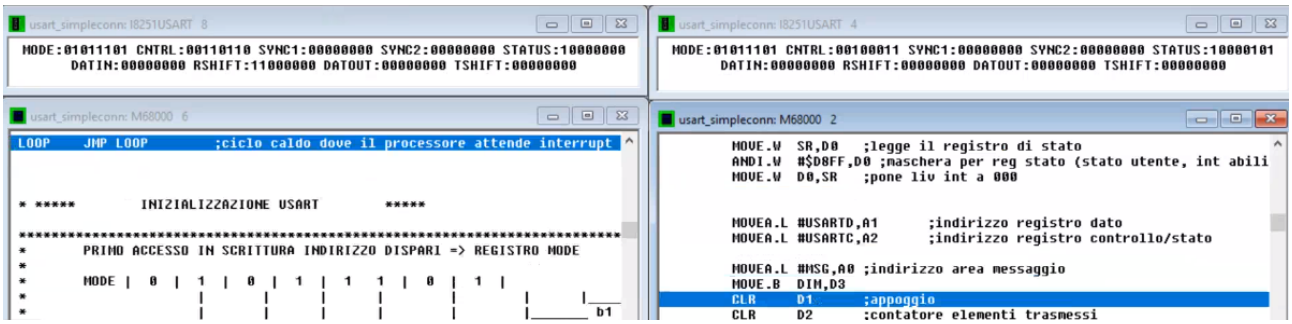


Nel sistema s2 facciamo show loc 6c e vediamo 00008700, che è l'indirizzo dell'interruzione di livello 3 associata a rxrdy. Invece, nel caso di s1 all'indirizzo 70 troviamo 00008800 (figura a destra), che è l'indirizzo dell'interruzione in trasmissione, che abbiamo dovuto mettere anche se non la utilizziamo. Per prima cosa, dobbiamo inizializzare le periferiche, **a destra dobbiamo vedere cosa accade in dataout e tshift, invece a sinistra datain e rshift**. Quando clicchiamo su device, abbiamo un menu *speed* (figura a sinistra), il valore 0003 vuol dire che ogni tic che facciamo con lo step di simulazione è associato a come lavora il processore, le periferiche lavorano a una velocità molto più bassa. Come simuliamo questa differenza di velocità? Il 3 vuol dire che ***per fare una singola operazione con la USART dobbiamo fare 3 tic del processore, in realtà dovrebbe essere circa 30 però in tal modo riusciamo a vedere la differenza senza avere una simulazione troppo lunga.***

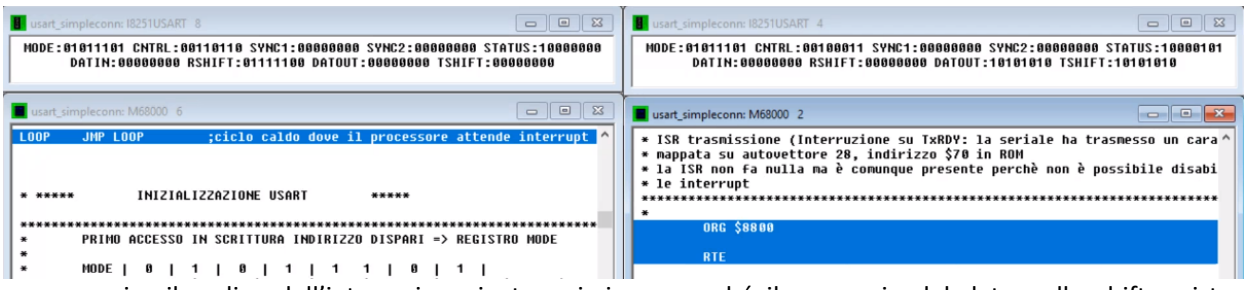




Facciamo partire la simulazione, abbiamo configurato modo e controllo, torniamo dalla init e andiamo avanti nell'esecuzione.

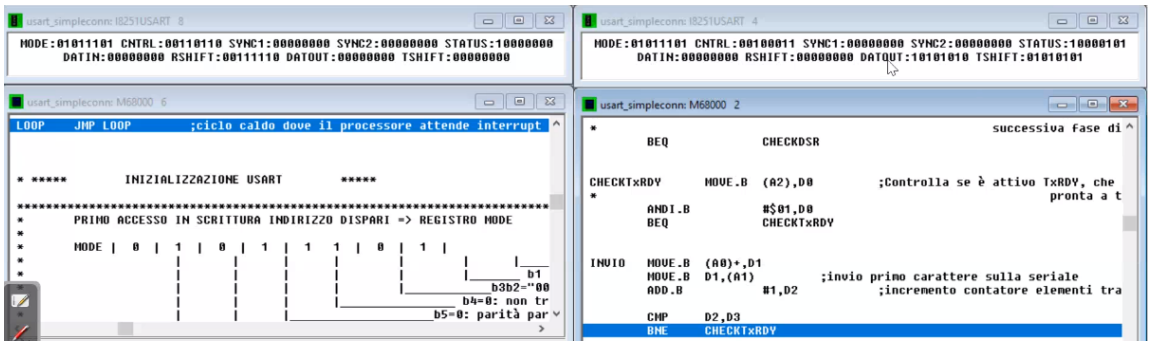


Azzeriamo il registro di stato di entrambi, mettiamo gli indirizzi che ci servono, a sinistra facciamo chackdsr invece a destra mettiamo un check di valori. A sinistra andiamo su loop jmp loop, a destra superiamo il controllo, facciamo checktxrdy e vediamo che *il primo bit a destra dello status, è 1, per cui il controllo va bene e possiamo fare l'invio vero e proprio*. Ora il dato in D1 viene inserito nella seriale e lo troviamo in dataout, **il dato verrà passato automaticamente da dataout a tshift**. **All'inizio si può fare in modo automatico perché la periferica è resettata, ed <u>è l'operazione che fa alzare il txrdy</u>**.

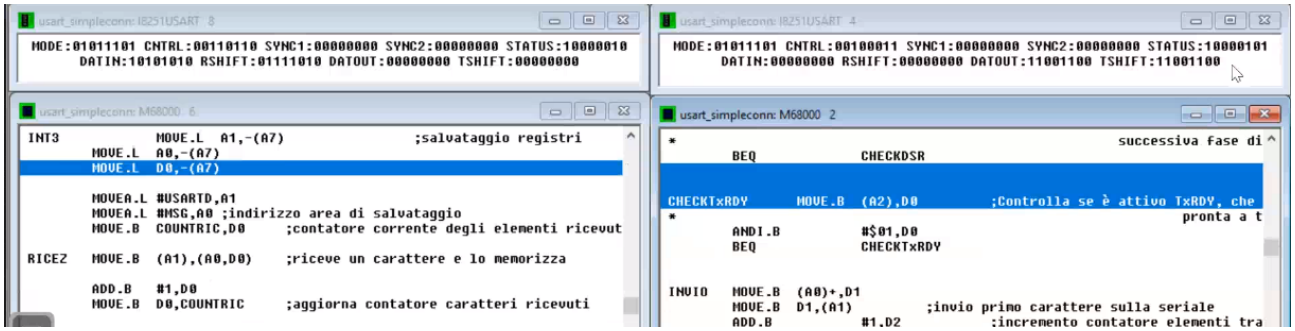


Si passa a eseguire il codice dell'interruzione in trasmissione, perché il passaggio del dato nello shift-register causa l'alzamento di txrdy, *possiamo inserire un nuovo dato in dataout perché nel frattempo lo shift-register lavora*. <u>**Nella realtà sono due eventi diversi ma per il primo carattere sono contemporanei**</u>, per questo si alza l'interruzione ma è perché stiamo

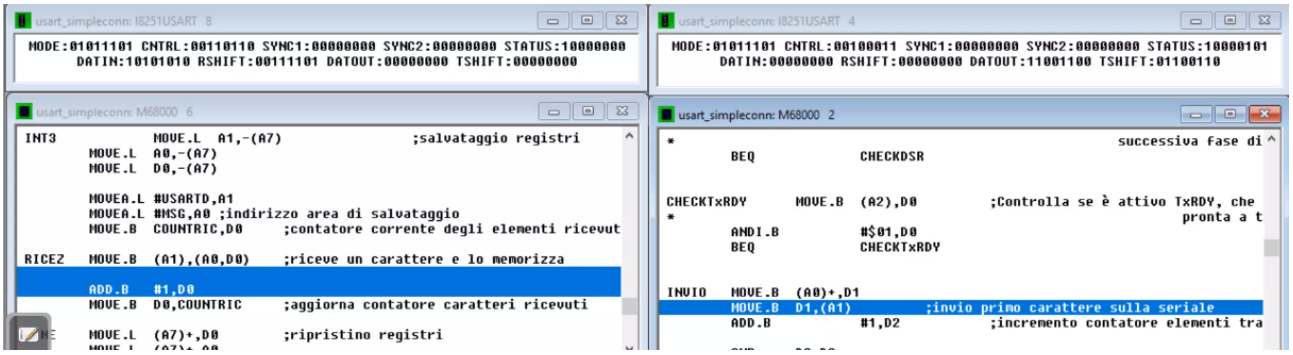
inviando il primo carattere, l'interruzione non fa niente e quindi torniamo all'esecuzione. Nel dataout abbiamo messo AA, ovvero 10101010, nel tshift abbiamo già shiftato di 1, è andato via il primo 0.



Per ogni click che facciamo tshift non si muove sempre, ma ogni 3. Questo simula la velocità diversa della seriale rispetto al processore. Andiamo avanti e poi dobbiamo trasmettere parità e stop. Abbiamo finito di trasmettere il carattere di prima e bit di start, 2 di stop e di parità (4 bit in più), **poi quando finiamo si alza di nuovo il bit 1 del registro di stato, che segnala il fato che txrdy è alto, possiamo trasmettere**. In S2 il dato seriale è arrivato in rshift, in datain abbiamo caricato l'intero valore da leggere, ora dobbiamo leggerlo.



Siamo nel codice dell'interruzione, facciamo la lettura per cui **il secondo bit dello status** (figura in basso)**, legato a rxrdy, si è azzerato, perché abbiamo fatto la lettura. Si alza quando finiamo di ricevere, si alza quando leggiamo**. A destra, nel frattempo, sta inviando il secondo dato 11001100. In Rshift, a sinistra, vediamo che arrivano nuovi dati, ogni 3 tic. Alla fine si alza rxrdy. Entrambi i sistemi lavorano, con i loro tempi, più lenti del processore, trasferiscono i bit di dati più quelli necessari alla sincronizzazione.



Comunicazione seriale asincrona con interruzione in trasmissione e ricezione

Sistema S1: Vediamo il caso in cui usiamo le interruzioni anche in trasmissione, **in S1 ora il meccanismo di interruzione dev'essere innescato, dobbiamo inviare almeno il primo bit, poi ci mettiamo in loop**. L'interruzione in trasmissione di volta in volta prende il carattere da inviare, usando il registro A0 in cui ha messo l'indirizzo di base dell'area di memoria, e andando a sommare il registro indice, per cui ***non abbiamo usato il modo di indirizzamento indiretto con indice ma proprio una somma,*** ma non cambia. L'unica cosa che osserviamo è che prima l'interruzione in trasmissione non faceva niente, ora invece prende il carattere successivo. (figura pagina successiva)

```
PRIMO   MOVE.B  (A0),D1
    MOVE.B  D1,(A1)      ;invio primo carattere sulla seriale
    ADD.B       #1,D2        ;incremento contatore elementi trasmessi
    MOVE.B  D2,COUNTINV


LOOP    JMP LOOP    ;ciclo caldo dove il processore attende interrupt
```

```
        ORG $8800

INT4        MOVE.L  A1,-(A7)        ;salvataggio registri
        MOVE.L  A0,-(A7)
        MOVE.L  D0,-(A7)
        MOVE.L  D1,-(A7)

        MOVEA.L #USARTD,A1
        MOVEA.L #MSG,A0 ;indirizzo area di salvataggio
        MOVE.B  COUNTINV,D0 ;contatore corrente degli elementi ricevuti
        MOVE.B  DIM,D1

        CMP.B   D0,D1           ;controlla se devo trasmettere altri caratteri
        BEQ FINE
INVIO   ADDA.L  D0,A0
        MOVE.B  (A0),D1
        MOVE.B  D1,(A1)     ;trasmette il carattere sulla seriale

        ADD.B   #1,D0
        MOVE.B  D0,COUNTINV ;aggiorna contatore caratteri inviati

FINE    MOVE.L  (A7)+,D1        ;ripristino registri
        MOVE.L  (A7)+,D0
        MOVE.L  (A7)+,A0
        MOVE.L  (A7)+,A1

        RTE
```

Comunicazione seriale sincrona

Sistema S1: vediamo il sistema sincrono. È tutto simile tranne la sequenza di inizializzazione del sistema, **nel registro di modo dobbiamo dire che vogliamo trasmissione sincrona, e quindi dobbiamo anche settare quanti byte di sync vogliamo usare**. Nel registro di modo diciamo che vogliamo la trasmissione sincrona e un carattere di sincronismo. Gli altri bit sono gli stessi, possiamo sempre dire la parità, la dimensione del dato (sempre 8 bit).

```
INITUSART   MOVE.B      #%10001100,USARTC
```

```
PRIMO ACCESSO IN SCRITTURA INDIRIZZO DISPARI => REGISTRO MODE

MODE |  1  |  0  |  0  |  0  |  1  |  1  |  0  |  0  |
     |     |     |     |     |     |     |     |___b0=0 Trasmissione Sincrona, b0=1 Asincrona
     |     |     |     |     |     |     |_____ b1 non utilizzato
     |     |     |     |     |     |___________b3b2="00"(5bit/dato), "01"(6bit/dato), "10"(7bit/dat
     |     |     |     |     |_________________ b4=0: non trasmette bit di parità, b4=1: lo trasmette
     |     |     |     |_______________________b5=0: parità pari, b5=1: parità dispari
     |     |     |_____________________________b6=0: 1 bit di stop, b6=1: 2 bit di stop
     |     |___________________________________b7=0: 2 caratteri di sync, b7=1: 1 carattere di sync (in trasm
```

Dopo aver specificato il modo, dato che abbiamo chiesto la trasmissione sincrona **dobbiamo definire i caratteri sync, dall'esterno**. Il modello di programmazione ci dice che quando facciamo un primo accesso, dopo il reset, all'indirizzo dispari lo facciamo all'indirizzo di modo, ***se il modo è sincrono il secondo accesso in scrittura a indirizzo dispari sarà al registro sync1, se abbiamo definito 2 caratteri di sync anche il secondo accesso sarà per il carattere di sync e scriverà in sync2***.

Quindi, nel nostro caso abbiamo dato un solo carattere di sync per cui definiamo del registro sync1 il carattere che vogliamo, nel nostro caso FF.

```
SECONDO ACCESSO IN SCRITTURA INDIRIZZO DISPARI => REGISTRO SYNC1

MOVE.B      #$FF,USARTC setta il segnale di sync1 a $FF
```

Dato che abbiamo definito che vogliamo solo un sync, dal terzo accesso in poi accediamo al registro di controllo, da ora in poi diamo la configurazione: abilitiamo il trasmettitore, abilitiamo DTR e RTS, non abilitiamo il ricevitore e anche il b7 non viene toccato (perché lo utilizza il ricevitore).*L'unica cosa che cambia, nel codice della periferica sincrona, è la parte di inizializzazione, dobbiamo dire che vogliamo la comunicazione sincrona e dobbiamo dargli il sync*.

```
MODE |  1  |  0  |  1  |  0  |  0  |  1  |  1  |  0  |
     |     |     |     |     |     |     |     |___b0=1: abilita trasmettitore, b0=0: disabilita t
     |     |     |     |     |     |     |_____ b1=1: attiva DTR, b1=0: disattiva DTR
     |     |     |     |     |     |___________b2=1: abitilita ricevitore, b0=0: disabilita ricevi
     |     |     |     |     |_________________ b3 non utilizzato
     |     |     |     |_______________________b4=1: cancella bit di errore, b4=0: non cancella
     |     |     |_____________________________b5=1: attiva RTS, b6=0: disattiva RTS
     |     |___________________________________b6=1: resetta l'interfaccia seriale
     |_________________________________________b7=1: porta il ricevitore nello stato HUNT in cui cerca i carat
                                               (in comunicazione sincrona)
```

Sistema S2: abbiamo lo stesso (figura sopra), nel registro modo diciamo che vogliamo sincrono un solo carattere sync, gli diciamo qual è (lo stesso del trasmettitore) e ***nel registro di controllo abilitiamo il ricevitore e lo mettiamo nello stato hunt***, perché deve aspettare il carattere di sync, ***è come se modificassimo l'automa***, *ora non è in attesa dello start bit (asincrona) ma dell'intero carattere che riceve, lo decodifica per capire se è il sync, poi gli altri caratteri sono quelli che riceve normalmente.*

Sistema con USART e terminale

```
Configuration name: usart.cfg

CHIP Name: MEMORIA-A
Type: MMU/BUS.   Identif: 01.          BUS: 0000.
Addres 1: 00008000.          Address 2: 00000000.
Com1: 0000.  Com2: 0010.  Com3: 0008.  Com4: 0000.

CHIP Name: M68000
Type: CPU.       Identif: 02.          BUS: 0001.
Addres 1: 00009000.          Address 2: 00009200.
Com1: 0000.  Com2: 0000.  Com3: 0000.  Com4: 0000.

CHIP Name: TERMINAL
Type: Device.    Identif: 03.          BUS: 0001.
Addres 1: 00002000.          Address 2: 00002001.
Com1: 0002.  Com2: 0001.  Com3: 0002.  Com4: 0000.

CHIP Name: I8251USART
Type: Device.    Identif: 04.          BUS: 0001.
Addres 1: 00002004.          Address 2: 00002005.
Com1: 0002.  Com2: 0003.  Com3: 0004.  Com4: 0008.

CHIP Name: MEMORIA-B
Type: MMU/BUS.   Identif: 05.          BUS: 0000.
Addres 1: 00008000.          Address 2: 00000000.
Com1: 0000.  Com2: 0010.  Com3: 0008.  Com4: 0000.

CHIP Name: M68000
Type: CPU.       Identif: 06.          BUS: 0005.
Addres 1: 00009000.          Address 2: 00009200.
Com1: 0000.  Com2: 0000.  Com3: 0000.  Com4: 0000.

CHIP Name: TERMINAL
Type: Device.    Identif: 07.          BUS: 0005.
Addres 1: 00002000.          Address 2: 00002001.
Com1: 0006.  Com2: 0001.  Com3: 0002.  Com4: 0000.

CHIP Name: I8251USART
Type: Device.    Identif: 08.          BUS: 0005.
Addres 1: 00002004.          Address 2: 00002005.
Com1: 0006.  Com2: 0003.  Com3: 0004.  Com4: 0004.
```

Vediamo un sistema con memoria bus, processore, terminale e USART, anche il secondo è uguale. La USART 8 è collegata con la 4, con la stessa modalità di prima. Il terminale si usa come un altro dispositivo per inserire dati da tastiera, nel processore, oppure per visualizzarli. **Il terminale consente sia di inserire dei dati sia di mostrarli a video, è un tastiera/terminale**. Questo esercizio carica lo stesso programma su entrambi i sistemi, che ***possono lavorare contemporaneamente da trasmettitore o ricevitore,*** *come fanno?* **Lo decide il tipo di operazione che fa il processore, se fa move verso il registro dato fa la trasmissione, se riceve qualcosa (dato che ha l'interruzione) fa la ricezione**. Nelle periferiche vediamo che sono abilitati sia ricevitore sia trasmettitore (ultimi due bit a destra).

```
SECONDO ACCESSO IN SCRITTURA ALLA SERIALE => REGISTRO CNTRL *
INDIRIZZO DISPARI                          *
CNTRL | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |                  *
        |   |   |   |   |   |   |   |__Abilita trasmettitore          *
        |   |   |   |   |   |   |______Attiva DTR                *
        |   |   |   |   |   |__________Attiva ricevitore           *
        |   |   |   |   |______________Non utilizzato              *
        |   |   |   |__________________Azzera bits di errore in STATUS  *
        |   |   |______________________Attiva RTS                *
        |   |__________________________Non resetta la periferica      *
        |______________________________Non va in 'hunt'          *
```

Vediamo cosa fa il sistema, associa 2004 alla USART, il terminale è mappato in 2000.

```
usart   EQU     $2004       Interfaccia seriale.
ter     EQU     $2000       Terminale video.

*
    ORG $8200       Indirizzo di partenza del main program.
*
START   MOVEA.W #usart,A0       Inizializza l'interfaccia seriale,
```

La prima cosa da fare è inizializzare la USART, scrittura dei registri modo e controllo. La comunicazione è asincrona, con 8 bit di dati, parità dispari e due bit di stop (come nel 10 esercizio).

```
PRIMO ACCESSO IN SCRITTURA ALLA SERIALE => REGISTRO MODE    *
INDIRIZZO DISPARI                                  *
MODE | 0 | 1 | 0 | 1 | 1   1 | - | 1 |                        *
       |   |   |   |     |     |   |____Trasmissione Asincrona        *
       |   |   |   |     |     |________Non utilizzato                *
       |   |   |   |     |______________8 bit per dato                *
       |   |   |   |____________________trasmette bit di parità       *
       |   |   |________________________tipo di parità dispari        *
       |   |____________________________2 bit di stop                 *
       |________________________________#bit di sync in trasmissione asincrona  *
```

Il secondo accesso in scrittura in controllo abilita sia ricevitore sia trasmettitore (visto nella figura precedente). Poi, dobbiamo inizializzare il terminale, su cui possiamo scrivere una parola di controllo, nel registro di controllo *un bit ci dice se abbiamo abilitato le interruzioni su buffer full e un bit l'interruzione sul tasto enter*. **Un'interruzione si ha quando premiamo il tasto *enter* sulla tastiera, si invia l'interruzione al processore**. Scriviamo quello che vogliamo nel terminale e **quando premiamo enter, quello che scriviamo nel terminale diventano dei caratteri che inviamo tramite la seriale a s2**. All'interno all'interruzione del tasto enter facciamo delle scritture nella USART. **Poi se scriviamo più di 256 caratteri nella tastiera si genera l'interruzione *buffer full***. Le altre configurazioni del terminale ci dicono che vogliamo pulire lo schermo, il buffer di tastiera, quindi *c'è un buffer nel dispostivo che tiene dentro tutti gli elementi che abbiamo inserito finchè non premiamo enter*. Poi possiamo abilitare la tastiera, l'eco, e poi due bit di stato che ci dicono se abbiamo la condizione di buffer full e di enter. Con 3f settiamo il funzionamento del terminale.

```
INIZIALIZZAZIONE DEL TERMINALE:

  CNTRL |0 |0 |1 |1 |1 |1 |1 |1 |
        |   |   |   |   |   |   |   |__Abilita interruzioni su Buffer full
        |   |   |   |   |   |   |______Abilita interruzioni su Enter
        |   |   |   |   |   |__________Pulisci schermo
        |   |   |   |   |______________Pulisci buffer tastiera
        |   |   |   |__________________Abilita tastiera
        |   |   |______________________Abilita echo
        |   |__________________________Stato di buffer full
        |______________________________Stato di Enter inviato
```

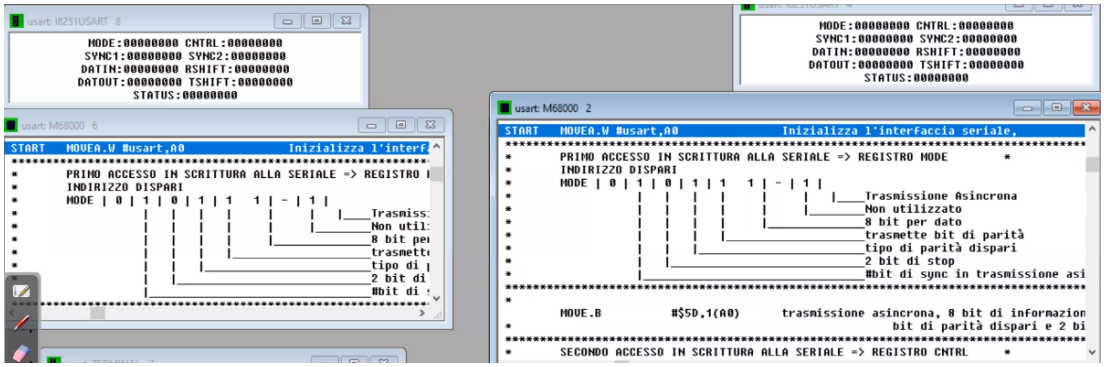
Poi, tramite la semplice AND azzeriamo lo statusregister e aspettiamo le interruzioni. Non c'è un sistema che trasmette o riceve nel main, la trasmissione viene fatta quando premiamo enter. **Nel codice dobbiamo trovare l'interruzione di enter, buffer full, trasmissione della seriale e ricezione della seriale, quindi in tutto 4 interruzioni, 2 della seriale e 2 della USART.** Prima di tutto, come fatto nell'altro esercizio, andiamo a controllare DSR (potremmo

```
    MOVE.B      #$3f,1(A0)  abilita tastiera, eco, interruzioni enter e buffer
*                   full, cancella video e pulisce buffer di tastiera.
*
    ANDI        #$F8FF,SR   Azzera l'interrupt mask.
*
main    NOP             Processore è in attesa di qualche interruzione.
    JMP     main
```

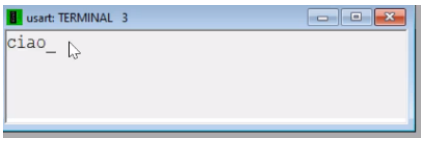
eliminarlo anche) poi vediamo se siamo pronti a trasmettere, vediamo se txrdy è alto, bit 1 del registro di stato. ***Se il bit è alto siamo pronti a trasmettere un nuovo dato***, allora prendiamo un dato dal buffer di tastiera, ***a0 lo abbiamo associato al terminale, prendiamo un dato dal terminale e lo spariamo su a1, associato al registro dato della USART***. Scriviamo, di fatto, i singoli caratteri che abbiamo nel buffer di tastiera sulla seriale. *Facciamo questo finchè non riconosciamo il tasto enter.* Abbiamo finito di trasmettere se arriva anche il tasto enter. Finchè scriviamo caratteri si riempie il buffer, quando leggiamo viene svuotato. Fatto questo, riabilitiamo tastiera, cancelliamo video e cancelliamo il buffer.

```
*******************************************************************************
wait11  MOVE.B      1(A1),D0        Controlla se è attivato il segnale DSR della
    ANDI.B      #$80,D0             USART ed in caso affermativo trasmette,
    BEQ     wait11                  altrimenti attende.
*
wait12  MOVE.B      1(A1),D0        Se l'interfaccia seriale è pronta a trasmetter
    ANDI.B      #$01,D0             continua altrimenti attende.
    BEQ     wait12
*
    MOVE.B      (A0),D0     Preleva uno alla volta i caratteri presenti nel
    MOVE.B      D0,(A1)     buffer di tastiera e li trasmette all'interfaccia
    CMPI.B      #13,D0      seriale finchè non riconosce il tasto enter che
    BNE     wait12          ha causato l'interruzione.

*
end MOVE.B      #$3f,1(A0)  Riabilita la tastiera, cancella il video
*                       e pulisce il buffer di tastiera.
*
    MOVE.L      (A7)+,D0        Ripristino dei registri
    MOVE.L      (A7)+,A1        precedentemente salvati.
    MOVE.L      (A7)+,A0
    RTE             Ritorno alla routine interrotta.
*******************************************************************************
```

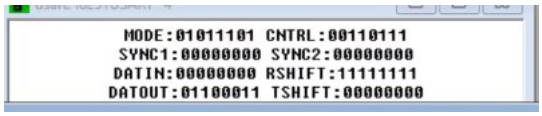
[Esecuzione in Asim](): mettiamo lo stesso programma su s1 e s2, poi mettiamo la stessa memoria su entrambi i sistemi. Il primo che scrive fa funzionare il sistema, per prima cosa dobbiamo fare inizializzare i due sistemi. Arriviamo a main jmp main.

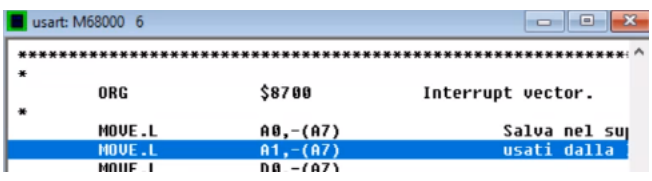


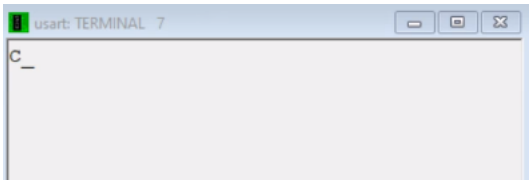
Andiamo nel terminale del primo e scriviamo una parola, ciao, **man mano che scriviamo il buffer della tastiera è riempito, finchè non premiamo enter o scriviamo 256 caratteri scriviamo**. Premiamo enter e vediamo che si attiva l'interruzione di livello 1, ovvero la pressione di enter, a destra, ora vediamo DSR che è attivo, vediamo txrdy che è attivo per cui possiamo fare la prima scrittura.



Ora, dentro il dataout va il codice della c, ovvero 01100011, codice binario della c, questa lettera viene mandata dall'altro lato, a sinistra nel frattempo rshift lavora, ogni 3 tic**. Dopo un po' si riempie datain e a sinistra parte l'interruzione della ricezione su rxrdy, per cui legge il carattere, lo mette nel registro e stampa il carattere che ha ricevuto sul terminale numero 7**, vediamo c.







Faremo questo processo per ogni carattere scritto, mandandoli fuori. Se scriviamo a sinistra ale, dopo un po' vediamo che arriva a destra. Tutto quello che dobbiamo fare è predisporre le interruzione, quindi scrivere le ISR degli autovettori e poi settare la connessione fisica con Asim.