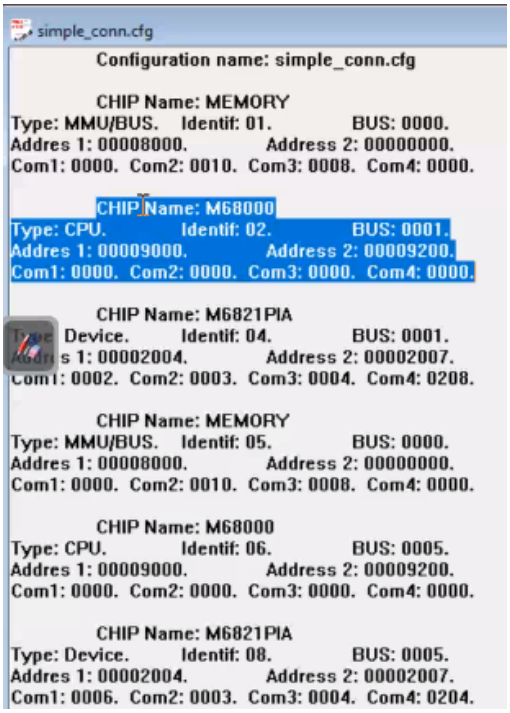
ESERCITAZIONE PIA

Vediamo come programmare il driver per la **ricezione di un vettore di caratteri**, tramite la periferica e usando le interruzioni. La prima cosa che dobbiamo ricordare è che nella PIA abbiamo due porti uguali, A e B, che internamente al dispositivo PIA sono assegnati 6 registri interni, 3 per ogni porto. Ogni porto ha un registro per i dati, uno direzione e uno di controllo. Dal punto di vista esterno, il device viene mappato in memoria con uno spazio di 4 registri, soli 2 bit di indirizzamento per 6 registri. C'è una ***tecnica di indirizzamento interno*** che consente di accedere, con un'operazione leggermente più complessa, al registro dati in modo differente a seconda di un bit particolare nel registro di controllo. Questa cosa nel driver si concretizza *accedendo prima al registro di controllo, dove si posiziona il bit, poi si fa l'accesso all'indirizzo pari, condiviso da direzione e dati, per settare prima la direzione del porto, poi gli altri accessi all'indirizzo pari saranno fatti al registro dati.* La seconda cosa da ricordare è come interconnettere due PIA per far parlare due sistemi differenti. Qui abbiamo due sistemi, s1 e s2, che si scambiano informazioni. S1 invia un messaggio di n caratteri a s2 tramite una periferica parallela. **Entrambi i sistemi hanno una periferica parallela all'interno, collegate in handshaking.** La modalità vista prevede l'incrocio dei fili CA1, CA2, CB1 e CB2 per realizzare l'handshaking tra i due sistemi per essere certi che ogni carattere inviato da s1 sia ricevuto e letto da s2. Usiamo il codice assembly del 68k, simulato in ambiente Asim, anche per le periferiche. Noi abbiamo simulato solo programmi con processore e memoria, **per poter simulare un sistema per prima cosa dobbiamo caricare una configurazione in Asim**, questa specifica i componenti del sistema da simulare, che nel caso di CE1 erano processore e memoria, e ***la memoria in Asim ingloba anche il concetto di bus***. Dovevamo dire al simulatore quali erano i parametri fondamentali dei dispositivi, per il processore dove si trovava RAM e ROM, e le grandezze, poi definivamo lo SP, nelle porte com.



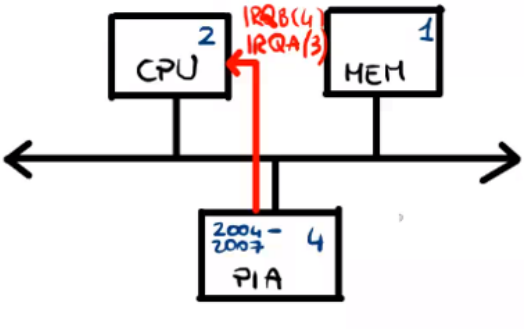
1. **Nel processore** dobbiamo inserire un *tipo*, ovvero CPU, un *identificativo*, nel nostro caso 2, **l'indicazione del bus memoria in cui è connesso** e negli indirizzi address 1 e 2 dovevamo inserire SP utente e supervisore. **I 4 parametri aggiuntivi com sono per le periferiche**, erano lasciati vuoti.
2. **Per la memoria**, la tipologia era mmu/bus, l'identificativo dato era 1, *in address 1 si mette l'indirizzo base della RAM, in 2 della ROM, nella com2 e com3 si metteva la grandezza della RAM e ROM, in esadecimale.*

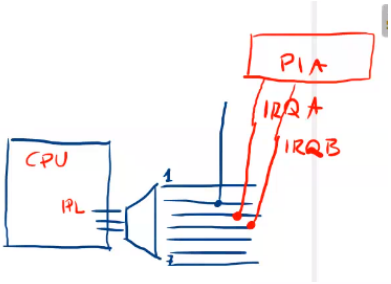
Noi dobbiamo fare un sistema più complesso, abbiamo processore e memoria e poi aggiungiamo una periferica parallela, sia per s1 sia per s2. *La periferica parallela si configura usando i campi address e com*. Per prima cosa, il tipo si setta a **device**, identif è 4, **indichiamo il bus a cui è connesso**, per il primo sistema è quello con identificativo 1, **negli indirizzi address1 e 2 si mettono il primo e l'ultimo indirizzo dello spazio di indirizzamento assegnato al dispositivo, sono 4 locazioni consecutive,** quindi, sono ad esempio 2004 e 2007. Poi, abbiamo i 4 com che sono usati per definire parametri aggiuntivi, se servono. **Nella com1 abbiamo l'identificativo del gestore delle interruzioni** per il dispositivo, queste per la periferica parallela *sono gestite dal processore*, con identificativo 2, ovvero mettiamo 2. **In com2 e com3 mettiamo le linee di interruzione associate ai due porti** del dispositivo. Abbiamo 4 cifre per cui possiamo definire anche il vettore nel caso vettorizzato, vettore a priorità e linea di interruzione. *Delle 4 cifre, i primi due simboli sono per il vettore, il terzo per la priorità e l'ultima per la linea di interruzione (per gli autovettori).* La versione corrente di Asim, per l'utilizzo della PIA e dei dispositivi seriale usa solo gli autovettori, allora ***nell'ultima cifra specifichiamo la linea di interruzione su cui colleghiamo la linea di interruzione su cui colleghiamo IRQA in COM2 e IRQB in COM3.***

Se una certa periferica è collegata alla linea 3 di priorità del processore, vuol dire che le interruzioni sono gestite dall'ISR, il cui indirizzo si trova nel vettore delle interruzioni nell'autovettore 27. Questo perché **gli autovettori sono identificati dai vettori della tabella delle istruzioni che vanno dal 25° in poi**, fino a 31°. Il 3° vettore allora è il 27°. *Se prendiamo dalla ROM il 27° vettore, quindi facciamo 27x4 perché ogni indirizzo è da 4 byte, otteniamo il numero 108, in esadecimale è 6c*, nella ROM in 6c dovremo mettere l'indirizzo di partenza dell'ISR, che gestisce l'interruzione sulla linea IRQA (27) e IRQB (28).

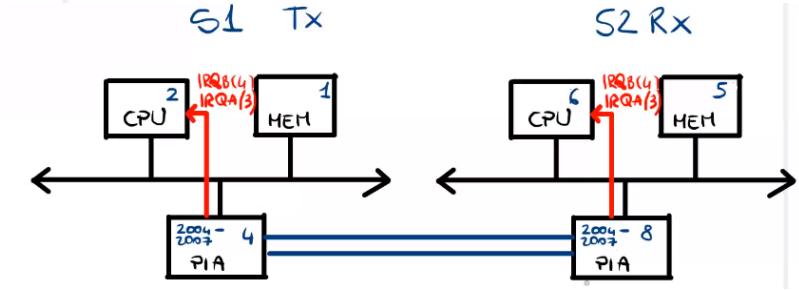
Ritorniamo allo screen precedente, il primo sistema è fatto dai primi 3 componenti, la memoria ha identificativo 1, la prima PIA è collegata a 1 (memoria) come il primo processore. Sul bus 1 sono attaccati il processore, con identificativo 2, e la PIA, con identificativo 4. Sotto vediamo un altro sistema, fatto da altri 3 componenti, il bus memoria ha identificativo 5, il processore sta su 5, come anche la PIA, sono due sistemi separati.

Vediamo lo schema. **Su Asim memoria e bus sono simulati come un solo dispositivo**, questo che vediamo a destra (senza considerare la linea rossa) è un solo sistema, la CPU ha identificativo 2, la memoria 1 e la PIA 4. La PIA è mappata dagli indirizzi 2004 a 2007. Il **processore ha, verso l'esterno, un encoder di priorità, su ogni filo di priorità è associata una priorità per le interruzioni**, come vediamo a sinistra (disegno in blu). Materialmente possiamo collegare la periferica a un certo filo.

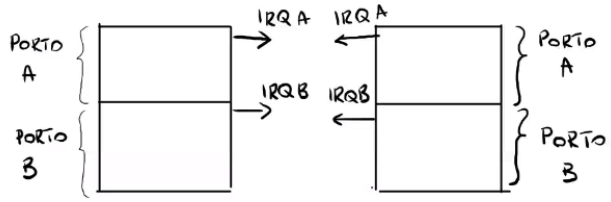




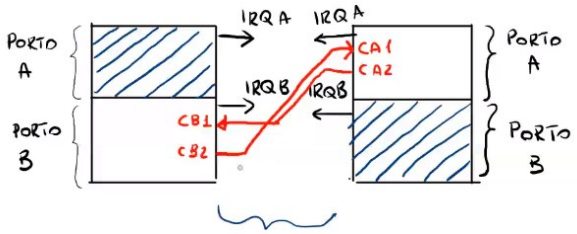
Questa freccia rossa che vediamo a destra nella realtà vuol dire che la CPU ha un encoder di priorità, **dove abbiamo le linee IPL che si settano a seconda della priorità dell'interruzione che stiamo servendo**. Ogni dispositivo che colleghiamo tramite le linee di interruzione, avrà le sue linee collegate. Nell'esempio della PIA, IRQA è collegata al 3° filo, IRQB al 4° filo. In Asim abbiamo questi due sistemi, identici, che comunicano tra loro tramite il collegamento tra le



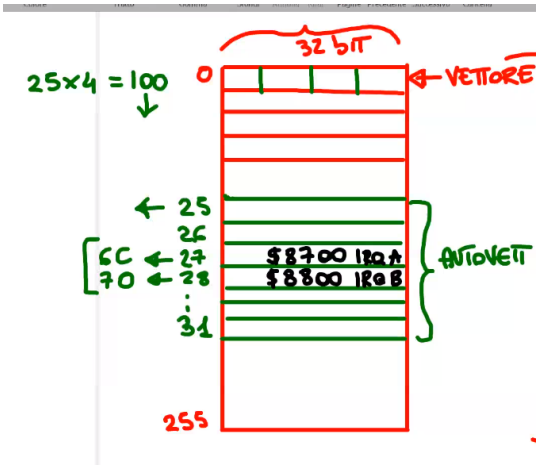
due PIA. *Sono due sistemi separati collegati tramite la PIA.* Vediamo un dettaglio del collegamento, nella figura a sinistra. L'altra volta abbiamo visto che **la PIA è un adattatore, sulla sinistra ha il processore e sulla destra una periferica parallela.** Dobbiamo scegliere un componente da attaccare alla PIA, abbiamo delle configurazioni base dove mettiamo un terminale



parallelo, poco utile ai nostri fini, nell'esercitazione considerata *l'altra periferica parallela è un computer, connesso tramite la PIA con handshaking*. Invece di avere una stampante ora abbiamo connesso un'altra PIA che può mandare e ricevere caratteri, per cui dobbiamo distinguere i driver di ricezione su un porto, di trasmissione sull'altro. **Dobbiamo prima configurare i due oggetti, una parte per ricevere e una per trasmettere, poi dobbiamo collegarli opportunamente per gestire l'handshaking tra le due parti.** Entrambi i porti si possono configurare in modo indipendente, noi considereremo *il sistema s1, in questo esercizio, come trasmettitore, s2 come ricevitore*. **Per far funzionare s1 solo da trasmettitore, scegliamo uno dei due porti per fare la parte di output, prendiamo B. *Il porto A della prima PIA allora possiamo non considerarlo,*** lo abbiamo perché la PIA lo da ma non ci serve perché questa PIA ci serve solo in output per mandare i dati verso la seconda PIA, che lavora come input. Lato trasmittente usiamo solo B, lato ricevente solo A, analogamente. *Come sono collegati i due?* Abbiamo a disposizione per ogni porto due segnali, in rosso, ovvero per B CB1 e CB2, per A abbiamo CA1 e CA2. **Questi segnai vengono programmati e la modalità che usiamo per l'handshaking fa sì che i segnali CA2 e CB2 siano sempre in uscita, invece CA1 e CB1 sono in ingresso**. In più, la regola che dobbiamo seguire dice che ***il segnale CB2 che esce dal porto B della prima PIA, va a finire in CA1 della seconda PIA. Analogamente, CA2 del porto A della seconda PIA, finisce in ingresso al segnale CB1 della prima PIA***. Con questa configurazione siamo in grado di fare l'handshaking. Questo ***serve per essere certi che ogni carattere che s1 manda a s2 sia ricevuto e letto.*** Tramite questi segnali di ingresso e uscita abbiamo il modo per assicurarci dell'avvenuta lettura di ogni carattere. In Asim dobbiamo pure poter dire come colleghiamo i segnali in rosso. **Dato che possiamo avere diverse modalità, dobbiamo specificare tale modalità e lo facciamo con la 4° com**. Da un punto di vista fisico, se dobbiamo disegnare com'è fatto il sistema che realizza la specifica di progetto è fatto in tal modo, due sistemi identici collegati tramite la PIA. I fili in blu (del disegno precedente, con S1 Tx e S2 Rx) diventano la configurazione di dettaglio che vediamo al di sotto.



Rimane da capire la com4, la com1 dice chi è il gestore delle interruzioni, per la prima PIA è il processore di s1, per la seconda PIA è il processore del 2° sistema, ovvero 6. Poi, **abbiamo lo stesso contenuto di com2 e com3 perché sono due sistemi diversi, hanno replicato tutto, le tabelle delle eccezioni che possono essere gestite in modo indipendente dall'altro**. Per la gestione delle interruzioni, all'indirizzo 0 in memoria per un certo numero di locazioni, 255, da 32 bit, questa è la tabella



delle eccezioni (in rosso). Al suo interno ogni locazione è detta vettore, sappiamo che **dal 25° abbiamo gli autovettori (in verde), che servono per gestire in modo semplificato le interruzioni che vogliamo gestire solo per priorità**. Nell'encoder avevamo 7 fili in uscita, a seconda del filo fisico in cui attacchiamo l'indirizzo di interruzione, andiamo a scatenare l'ISR il cui indirizzo si trova, ad esempio, nel 3° autovettore, 27, perché il dispositivo si trova nella 3° linea, quindi nel nostro caso la IRQA. Ora serve l'indirizzo, ***27 è un indice, dato che ogni locazione è da 32 bit, ovvero 4 byte, dobbiamo fare 25x4, che è un numero decimale, lo dobbiamo trasformare in esadecimale perché in Asim gli indirizzi sono esadecimali***. Il 27 in esadecimale viene 6c, 28 viene 70. Nel codice cerchiamo direttamente queste locazioni, in cui troveremo un indirizzo, ad esempio $8700, $8800, *gli indirizzi di partenza delle ISR in grado di gestire IRQA e IRQB*.

Infine, vediamo la **com4**, vediamo che per la PIA2 è 0204 e per la PIA1 è 0208. ***Le due cifre meno significative sono l'identificatore dell'altra PIA con cui parliamo***, ovvero la 04 con la 08 e viceversa. **Le cifre a sinistra, a seconda di come sono scritte, indicano come sono connessi CA1 CA2 CB1 e CB2**. ***Con 02 realizziamo la connessione per l'handshaking***, vista in rosso nel disegno precedente. Negli esercizi useremo sempre questa nella com4, per cui scriveremo nei primi due simboli sempre 02. Con la configurazione Asim stiamo realizzando il sistema in disegno.

```
CHIP Name: M6821PIA
Type: Device.      Identif: 08.        BUS: 0005.
Addres 1: 00002004.          Address 2: 00002007.
Com1: 0006.  Com2: 0003.  Com3: 0004.  Com4: 0204.
```

Dobbiamo avere un pezzo di codice per il sistema s1, un pezzo di codice per s2 perché per simulare l'interconnessione tra i sistemi dobbiamo scrivere i driver per s1 e per s2. Dal punto di vista di quali pezzi software ci sono nel driver di s1 e di s2, possiamo dire che nel primo esercizio che vediamo, con l'architettura vista, la modalità di funzionamento è la seguente: **s1 invia in un ciclo n caratteri, vuol dire che usa il porto B ma, *dal punto di vista della potenziale interruzione che B potrebbe sollevare, IRQB, questa è completamente disabilitata***. L'invio vero e proprio si fa solo scrivendo di volta in volta il carattere che vogliamo mandare nel registro dato, questo viene fatto come se fosse in un ciclo for. Non si usa in alcun modo l'interruzione sebbene il porto B abbia la possibilità di generarla, come ogni porto. **S2 invece riceve con interruzione**, ovvero sappiamo che *l'alternativa a ricevere con interruzione sarebbe continuare a guardare il registro di stato per vedere se è arrivato un carattere*, questo è poco efficiente per cui nell'esercizio usiamo le interruzioni in assembly ma limitate al lato ricezione. **Sul lato s2 usiamo il porto A, con IRQA abilitata**. *A quale evento è legata l'interruzione?* ***Si ha un'interruzione in s2 ogni volta che arriva un nuovo carattere, quando arriva un nuovo carattere viene automaticamente sollevata l'interruzione***, dato che è collegata a una linea fissa di priorità, viene automaticamente invocato il pezzo di codice all'indirizzo scritto nella tabella delle eccezione, dato che la linea era 3, nell'autovettore 27 con dento $8700, passeremo a eseguire le istruzioni da 8700 per servire l'interruzione. L'interruzione nasce quando arriva il carattere e poi muore, ogni volta che arriva il carattere. Nell'interruzione non ha senso gestire l'interruzione dell'intero messaggio perché **l'interruzione della PIA è associata al singolo byte**, dobbiamo gestirlo opportunamente, ovvero preleviamo il carattere dal registro dato e lo mettiamo in un'area di memoria preposta a gestire i dati in arrivo. Sono tutti eventi separati, per ogni byte. Questo è il modo in cui usiamo le interruzioni in ricezione nel primo esercizio. Poi, complicheremo un poco, dal punto di vista di s2 non cambia mentre useremo le interruzioni anche per s1. Usiamo le interruzioni anche in trasmissione, quindi IRQB è abilitata. Il concetto di interruzione in trasmissione va approfondito, ***cosa vuol dire che c'è un'interruzione in trasmissione?*** L'evento scatenante è sempre l'invio del singolo carattere, quando inviamo il primo carattere, a un certo punto s2 lo legge e per questo ci saranno degli eventi che interessano CA1, CB2, etc., che collegano le periferiche, per cui in risposta all'evento di lettura di s2, in s1 si scatena l'interruzione che significa che il carattere inviato è stato letto. ***L'interruzione in trasmissione indica che il carattere precedente è stato consumato***, possiamo inviarne un altro, la chiamiamo in trasmissione perché è dal lato di chi invia. In ricezione dice invece che è arrivato un carattere da consumare. Se su s1 dobbiamo fare un invio semplice come un ciclo, possiamo fare tutto nel programma principale main, invece in s2 abbiamo anche un'interruzione, allora c'è un main e un pezzo di codice dell'ISR.

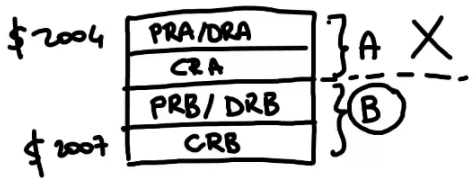
Codice S1_TX senza interruzioni

Ogni programma ha area dati e codice, i dati del programma chi sono? S1 vuole inviare n caratteri a s2, **per prima cosa definiamo un'area di memoria message in cui inseriamo 6 valori byte, poi mettiamo la dimensione del messaggio, ovvero** 6. Dc.B è una pseudoistruzione, *define constant DC, che consente di inizializzare un'area di memoria a cui diamo un'etichetta, un sinonimo di un indirizzo* per cui dalla locazione 8000 stiamo allocando l'area di memoria con significato message, fatta da tanti byte con significato individuale di 1,2,3,4,5

```
**********AREA DATI********
    ORG $8000
MSG DC.B    1,2,3,4,5,6
DIM DC.B    6
```

e 6. Dopo i primi 6 byte troviamo, in 8006, la dimensione, inizializzata a 6. L'area dati è solo questa. Invece, nell'area codice non troviamo niente di complicato, è un programma che, **tramite un ciclo for, deve prendere il singolo byte del messaggio e lo trasferisce in uscita tramite la porta parallela**. Come si scrive un dato sulla parallela? Si copia nel registro dati. ***Per scrivere si fa la MOVE sul registro dato, per leggere si fa la MOVE dal registro dato***.

Per lavorare sul driver **per prima cosa dobbiamo definire dove sono mappati in memoria i dispositivi** (tecnica di IO memory mapped) e questi indirizzi devono coincidere con quelli della configurazione Asim, avevamo detto che era da 2004 a 2007.



La PIA è mappata su 4 locazioni consecutive, abbiamo usato 2004 fino a 2007, in particolare abbiamo PRA/DRA del registro A e CRA del porto A, che è nella prime 2 locazioni, il porto B nelle 2 seguenti, ovvero PRB/DRB e CRB. In Asim abbiamo detto solo che occupavano le locazioni da 2004 a 2007, nel codice conosciamo questo ordine **e lato trasmittente il porto A non lo consideriamo, ma solo il B, è inutile segnare gli indirizzi di A che non useremo mai.** Nella configurazione iniziale ***con degli EQU scriviamo dei parametri, gli indirizzi di partenza dei vari registri, così da usarli nel codice***. Non partiamo da 2004 ma da 2006 perché 2004 e 2005 sono del porto A, che non usciamo in Tx. La PIA dato del porto B è in 2006, la PIA controllo del porto B è in 2007. Ora abbiamo gli indirizzi che ci servono, per prima cosa ***dobbiamo configurare la periferica***. Con i registri di controllo possiamo settare i bit per definire la modalità. *Il porto B dev'essere configurato in output, allora accediamo al registro direzione, mettiamo tutti 1 (uscita), quindi B è in uscita, poi configuriamo la modalità di funzionamento, ad esempio dicendo che interruzioni sono abilitate, su quali fronte attivo lavora CB1 e CA1, qual è la modalità di funzionamento (handshaking) e così via*.

```
PIADB   EQU   $2006    ;indirizzo di PIA-B dato, usato in output
PIACB   EQU   $2007    ;indirizzo di PIA-B controllo
```

```
*****************************************************************************************
*INIZIALIZZAZIONE DELLA PIA: PORTO B
*    CRB |0 |0 |1 0 0 |1 |0 0|
*                |       |    |____Controllo CB1: interruzioni disabilitate su IRQB, CB1 sensibile
*                |       |
*                |       |________Accesso a DRB: il prossimo accesso ad indirizzo pari è per PRB
*                |________________Controllo CB2: linea di uscita, con gli altri due bit a 00 diver
*                                                attivo di CB1 e si riabbassa a seguito a scrittur
*
*    DRB=1 => PORTO B PORTO DI USCITA
```

Per fare ciò usiamo una subroutine, chiamata DVBOUT. Non fa altro che inizializzare la periferica, il porto B e il modo di funzionamento generico della periferica. Nel driver abbiamo il dettaglio del significato dei vari bit del registro di controllo (figura sopra), ricordiamo la questione dell'indirizzamento interno. Dobbiamo settare B in uscita**, dato che il registro direzione e dati del porto B hanno lo stesso indirizzo, ne dobbiamo disciplinare l'accesso, vedendo il 3° bit del registro di controllo**. ***Se è 0 allora quando accediamo all'indirizzo pari (2006 PIADB) abbiamo l'accesso al registro direzione***, finchè il 3° bit di controllo è basso accediamo al registro direzione, poi ***quando abbiamo settato la direzione, rimettiamo a 1 il bit del registro di controllo, così ora ogni volta che accediamo a PIADB non accediamo al registro direzione ma dati***. Per prima cosa ci assicuriamo che ci siano tutti 0 nel registro di controllo (MOVE #0,PIACB), accediamo al registro direzione e scriviamo FF, tutti 1, per cui *negli 8 bit del registro direzione mettiamo tutti 1. La MOVE usa come primo operando un immediato (#) con tutti 1, il secondo operando è un'etichetta, è il* ***modo di indirizzamento assoluto*** *che dice che l'operando dell'istruzione è una locazione di memoria che si trova all'indirizzo PIADB, ovvero 2006*: accede all'indirizzo 2006, per com'è configurato il registro di controllo (tutti 0) andremo nel registro direzione e non dati, scriveremo tutti 1. Abbiamo configurato in output il porto B e **possiamo definire la modalità di configurazione della periferica, nel registro di controllo**. *Nei primi due bit significativi possiamo scrivere cosa vogliamo, sono dei bit di stato non di controllo*, sono settati da noi, ci interessano i restanti 6 che vediamo da destra a sinistra. **I primi due bit indicano rispettivamente se l'interruzione sul porto è abilitata o meno e su quale fronte attivo lavora il segnale del porto con pedice 1, CB1**. 00 vuol dire che IRQB è disabilitata e siamo sensibili al fronte di discesa dei segnali. **Il 3° bit dice da questo momento in poi ogni volta che diciamo PIADB è il registro dati**, **gli altri 3 bit definiscono la modalità di funzionamento, 100 è l'handshaking**. In questa modalità abbiamo i segnali con il pedice 2, CB2, in uscita, con 1, CB1, sempre in ingresso. Abbiamo che lavorano in modo intrecciato per realizzare l'handshaking e far capire quando un dato è correttamente letto dal sistema ricevuto. Nei driver abbiamo sempre una prima parte con cui inizializziamo le periferiche, nella PIA scegliamo le direzioni, inizializziamo i porti e le configurazioni per lavorare. La modalità è sempre 100 (ai nostri scopi), mentre potremmo abilitare o meno le interruzioni. Nel bit più a destra con 0 le interruzioni sono disabilitate, con 1 sono abilitate. Il terzo bit da destra va messo alto dopo aver settato la direzione, così accediamo sempre al registro dati.

```
DVBOUT  MOVE.B  #0,PIACB
    MOVE.B  #$FF,PIADB
    MOVE.B  #%00100100,PIACB
*
    RTS
```

Ora, la periferica è configurata, possiamo lavorare come dobbiamo. Dobbiamo solo fare il ciclo per l'invio dei caratteri. Avremo un loop. **Prima di fare il ciclo abbiamo messo gli indirizzi che ci servono nei registri del processore, ovvero PIACB in AI, PIADB in A2, il MSG in A0 e la dimensione in D0, perché serve per il ciclo**.

```
MAIN    JSR    DVBOUT   ;inizializza PIA porto B in output

        MOVEA.L #PIACB,A1   ;indirizzo registro di controllo CRB
        MOVEA.L #PIADB,A2   ;indirizzo registro PRB
        MOVEA.L #MSG,A0 ;indirizzo area messaggio
        MOVE.B  DIM,D0  ;dim del messaggio

        CLR D1  ;appoggio
        CLR D2  ;contatore elementi trasmessi
```

Nel ciclo utilizziamo l'indirizzamento indiretto con post incremento, accediamo, copiamo e incrementiamo il puntatore così da avanzare nel vettore. Usiamo **(A2)** che contiene l'indirizzo del registro dato, quindi **scriviamo nella locazione di memoria che coincide col porto del dato della periferica,** scriviamo nella periferica per andare verso l'esterno. Dunque, rendiamo il dato corrente, lo mettiamo in dati e **incrementiamo il contatore dei messaggi inviati**, poi come ci accorgiamo che s2 ha consumato il carattere? Dobbiamo vedere l'handshaking che nasce proprio per questo motivo, consente di capire se il carattere inviato è stato correttamente consumato.

```
INVIO   MOVE.B  (A2),D1
        MOVE.B  (A0)+,(A2)  ;
*
*
        ADD.B       #1,D2
```

Caratteristiche dell'Handshaking

Per capire l'handshaking dobbiamo ricordare che:

- **CA2 e CB2 sono sempre in uscita;**
- **CB2 diventa basso quando abbiamo un'operazione di scrittura**, perché il fronte attivo è quello di discesa, per come abbiamo definito nel registro di controllo;
- **CA2 diventa basso quando abbiamo un'operazione di lettura**, vuol dire che B è fatto per scrivere e A per leggere, per questo li usiamo sempre in questo modo: per il porto B l'evento importante è la scrittura mentre per il porto A è la scrittura;
- **CB1 e CA1 sono sempre di ingresso** e sono collegati correttamente con il concetto di interruzione. **Quando CB1 passa da 1 a 0 si scatena l'interruzione sul porto B, quando CA1 va da 1 a 0 si scatena l'interruzione sul porto A**. Questo scatenarsi dell'interruzione per il porto B **fa sì che il bit 7 del registro di controllo, CRB7, diventa** alto, infatti i primi due bit sono di stato; analogamente **per il porto A fa in modo che CRA7 diventi alto.**
- **CRA7 e CRB7 tornano bassi solo con lettura da PRA e PRB.**

Con l'interruzione qualcuno sveglia il driver per dire che è arrivato il dato, senza dovrebbe fare sempre il polling sul registro di stato.

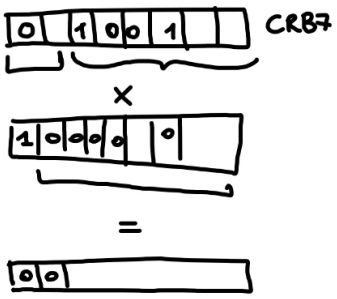
Vediamo cosa accade passo passo nei due sistemi quando scriviamo per la prima volta un dato. S1 scrive il dato, abbiamo fatto una MOVE da (A0)+ in (A2), ovvero PIADB, il registro per i dati. Facciamo la scrittura, abbiamo visto **che l'operazione di scrittura sul porto B fa abbassare CB2**. Le nostre due PIA sono collegate come visto prima, CB2 con CA1 e CB1 con CA2. **Se CB2 si abbassa, automaticamente si abbassa dopo un po' di tempo CA1**, che è collegato con un filo, avremo CA1 che si abbassa. *La transizione alto basso di CA1 è direttamente collegata all'interruzione, collegata al bit flag del registro di stato.* **Lato sistema s2 abbiamo l'interruzione IRQA e il CRA7 che si alza**: quando dal primo sistema scriviamo si scatenano degli eventi per cui nel secondo sistema ce ne accorgiamo in due modi, ovvero si alza l'interruzione e il registro di stato CRA7. Dal punto di vista del sistema che trasmetteva non possiamo fare niente, sappiamo solo che abbiamo inviato e siamo in attesa. In S2 invece ci siamo accorti che è successo qualcosa perché abbiamo appena avuto l'interruzione, allora dobbiamo leggere, perché sappiamo di aver ricevuto. **Si passa a servire l'interruzione, in base alla priorità del processore**, nel momento in cui serviamo l'interruzione dobbiamo leggere il valore ricevuto. **Sul porto A quando leggiamo abbiamo l'abbassamento di CA2**, quindi la lettura scatena l'abbassamento. Abbiamo detto anche che *l'evento di lettura fa abbassare il CRA7*. **CA2 è connesso a CB1, vuol dire che dopo un poco che si è abbassato CA2 si deve abbassare pure CB1**. Se si abbassa CB1 cosa accade? Avremo sempre il doppio evento, **si scatena l'interruzione e si alzerebbe CRB7**, avremmo IRQB e CRB7 alto. *In questo esempio IRQB è disabilitata, però CRB7 si è alzato*. Allora, ***per vedere se s2 ha letto dobbiamo aspettare che CRB7 si alzi, quando si alza l'handshaking è completo. Se avessimo avuto anche le interruzioni, non avremmo dovuto guardare in continuazione CRB7, avremmo avuto l'interruzione***. Abbiamo completato l'handshaking e lato s2 è come se non fosse successo niente, CA2 è tornato basso, poi come torna alto è legato alla modalità 100. CRA7 è tornato basso. **Abbiamo solo un problema lato trasmittente perché l'handshaking si è completato ma CRB7 è alto e se lo lasciassimo così non riconosceremmo il secondo carattere**, banalmente, se lo lasciamo alto non possiamo più controllare il prossimo handshaking, per cui prima di inviare un secondo carattere dobbiamo abbassare CRB7. Il bit 7 e 6 sono dei bit di stato, la loro variazione non la decidiamo da programma. Abbiamo però visto che **l'operazione che fa abbassare CRB7 è la lettura**, quindi anche se stiamo trasmettendo su B, ***prima di fare la successiva scrittura siamo costretti a fare una lettura per far abbassare CRB7, è la lettura fittizia.*** Se vediamo il codice, nell'invio, la prima operazione che facciamo è proprio una lettura dal registro dati. Dato che dobbiamo inviare più caratteri dobbiamo assicurarci che dopo l'handshaking tutto ritorni al punto di partenza, così possiamo farne

```
INVIO   MOVE.B  (A2),D1
```

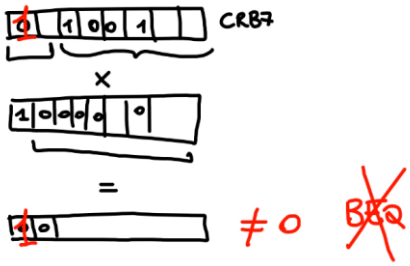
altre, allora dopo dobbiamo fare la lettura fittizia. Facciamo la lettura, scriviamo il dato, incrementiamo il contatore, poi **aspettiamo che CRB7 diventi alto**, solo ora vuol dire che S2 ha ricevuto correttamente.

```
ciclo2  MOVE.B  (A1),D1      ;In attesa di DATA ACKNOWLEDGE
    ANDI.B  #$80,D1              ;aspetta che CRB7 divenga 1
    BEQ ciclo2

    CMP D2,D0    ;controllo se ho finito di trasmettere
    BNE INVIO


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

Come facciamo ad aspettare? *Ciclicamente leggiamo (A1), ovvero PIACB, il registro di controllo, lo mettiamo in registro dati D1 e applichiamo la mascheratura con la stringa 80 che in binario ha un 1 in testa. Solo quando il valore di CRB7 è proprio 1, allora la maschera ci da un numero diverso da 0*. **Finchè quel numero è 0 continuiamo ad aspettare, torniamo sopra (BEQ ciclo2), solo quando la maschera è 0 vuol dire che CRB7 si è alzato**, da ora in poi possiamo continuare a salire sopra nell'invio, prima di farlo dobbiamo verificare di non essere arrivati a 6 caratteri, quindi controlliamo, con CMP D2,D0.



Perché c'è BEQ? Abbiamo il registro di 8 bit, sui primi due non possiamo fare niente, facciamo la AND bit a bit con 1000000 (figura a sinistra), se ci troviamo nell'ultimo bit 0 otteniamo uno 0 completo, finchè otteniamo 0 dobbiamo sempre aspettare. CRB7 si è alzato quando al posto dello 0 va 1, in tal modo troviamo un valore diverso da 0 e il BEQ non viene preso (figura a destra).



Codice S2_Rx con interruzione

Lato ricezione non abbiamo solo il main ma anche un codice a parte per l'ISR, che deve ogni volta leggere il dato e lo mette in memoria, non fa altro. **Lato ricezione è la lettura stessa a fare abbassare CRA7 e tutto ricomincia da capo**. Non ci sono controlli da fare perché non c'è polling ma interruzioni. Vediamo il codice. Nell'area dati abbiamo un'area messaggio in cui mettiamo i messaggi, *è un Define Storage DS.B perché creiamo e allochiamo l'area, senza riempirla perché lo faremo con i dati ricevuti*. **Creiamo un contatore da usare come contatore dei dati che riceviamo, non possiamo usare un registro interno perché non lo possiamo usare anche per le invocazioni delle interruzioni, usiamo queste aree di memoria che l'interruzione ogni volta va ad aggiornarle.**

```
    ORG $8000
MSG DS.B    6
DIM DC.B    6
COUNT       DC.B    0
```

```
    ORG     $8200

PIADA   EQU     $2004    ;indirizzo di PIA-A dato, usato in input
PIACA   EQU     $2005    ;indirizzo di PIA-A stato/controllo
```

Ora segniamo gli indirizzi solo del porto A, PIADA e PIACA, ovvero 2004 e 2005. La prima operazione da fare è l'inizializzazione, **mettiamo in direzione tutti 0 perché il porto è in ingresso, poi settiamo il modo di controllo, solo il bit a destra è diverso da prima perché ora abilitiamo le interruzioni**. La subroutine DVAIN fa questo: *inizialmente mettiamo tutti 0 nel registro di controllo, così entriamo nell'indirizzo direzione, poi mettiamo tutti 0 in PIADA, registro direzione, poi settiamo il modo di funzionamento nel registro di controllo, con la differenza che mettiamo un 1 a destra perché vogliamo abilitare IRQA*.

```
DVAIN   MOVE.B  #0,PIACA
*
    MOVE.B  #$00,PIADA
    MOVE.B  #%0010010I,PIACA
*
    RTS
```

In entrami i codici abbiamo il pezzo nel main che **modifica il valore contenuto nello SR**, questo ***non serve al driver ma alla sua simulazione in Asim, che parte in modalità supervisore con la maschera delle interruzioni settata a tutti 1***. La maschera delle interruzione da un codice che indica il livello di priorità che se è 5 vuol dire che se arriva un'interruzione di livello fino a 5, compreso, il processore non la vede, se arriva un'interruzione di livello superiore passa a eseguirla. Se Asim parte con la maschera delle interruzioni 111 vuol dire che solo un'interruzione di livello 7 (che non sono mascherabili) potrebbe essere servita, **noi abbiamo l'IRQA di livello 3, per cui se lasciassimo le cose come sono Asim non farebbe passare le interruzioni**. *Usiamo queste 3 istruzioni per azzerare la maschera delle interruzioni e mettiamo la modalità utente mettendo 0 nel registro di stato, nel bit della modalità supervisore.*

```
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
```

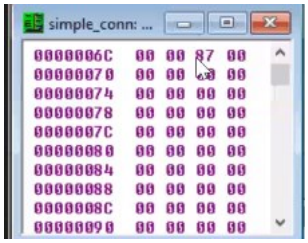
Il main poi è banale inizializza solo la periferica, il resto, la ricezione di un dato è qualcosa che deve fare l'interruzione. Vediamo il codice delle interruzione. Abbiamo allocato il codice dall'indirizzo 8700, per l'interruzione della PIA collegata al livello 3 di priorità abbiamo scelto come indirizzo di partenza 8700, allora nel 27° vettore dobbiamo scrivere 8700 altrimenti non sapremo mai come invocare questa interruzione**. Con Asim dobbiamo attaccare fisicamente IRQA sulla terza linea di priorità del processore ma non basta, dobbiamo anche scrivere l'indirizzo giusto di partenza dell'ISR dentro al vettore delle interruzioni che sta nella ROM**. Altrimenti il meccanismo di scatena male, il processore salta all'indirizzo che si trova in 6c che sarà tutti 0. **L'interruzione in una fase iniziale prende tutti i dati che servono e li mette nei registri interni, poi fa la lettura**. Questo modo di indirizzamento per il secondo operando è ***based indexed,*** *prende l'indirizzo in a0, base dell'area di memoria in cui mettiamo i caratteri, aggiunge il contenuto di D0, variabile di conteggio*, all'inizio è 0 quindi fa 9000+0, con l'indirizzamento indiretto va in memoria con quell'indirizzo. Poi aggiorna il conteggio degli elementi ricevuti, allora la volta successiva in count non c'è 0 ma 1, fa 9000+1 e mette il valore in 9001.

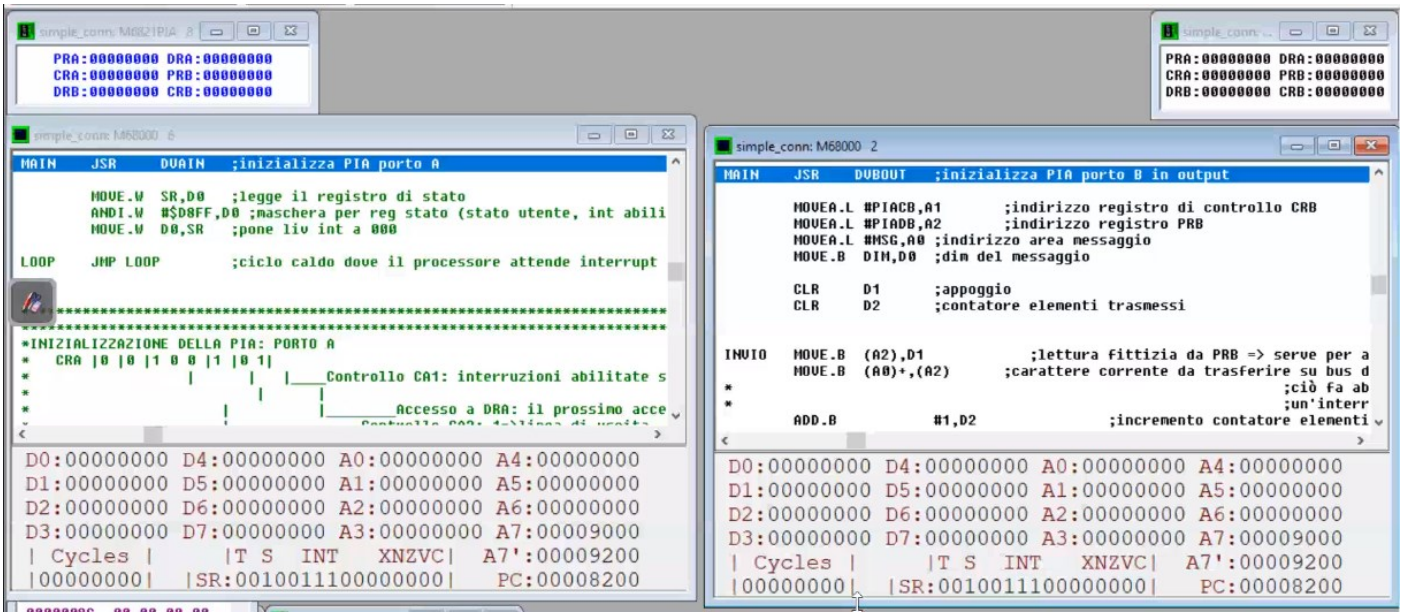
```
INT3        MOVE.L  A1,-
            MOVE.L  A0,-(A7)
            MOVE.L  D0,-(A7)

            MOVEA.L #PIADA,A1
            MOVEA.L #MSG,A0 ;ind
            MOVE.B  COUNT,D0


            MOVE.B  (A1),(A0,D0)
*
*


            ADD.B   #1,D0
            MOVE.B  D0,COUNT

            MOVE.L  (A7)+,D0
            MOVE.L  (A7)+,A0
            MOVE.L  (A7)+,A1

            RTE
```

Non possiamo scrivere (A0)+ perché non possiamo pensare che l'indirizzo di memoria rimanga invariato dentro A0, tra un'esecuzione e l'altra dell'interruzione, per questo usiamo la variabile globale count, che aggiorniamo ogni volta che la leggiamo. L'interruzione ha un codice semplice perché l'unica vera operazione che fa è ricezione e incrementare il contatore. Si occupa di un singolo carattere perché viene scatenata per il singolo carattere.

Simulazione Asim

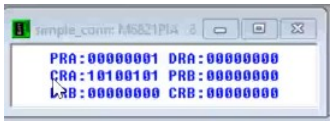
Vediamo cosa accade su Asim. Nel tool il cerchio apre tutto i chip e la I li inizializza, vediamo che per le periferiche abbiamo una finestra a parte. A destra vediamo il sistema s1, a sinistra s2. Clicchiamo sulla finestra del primo processore (identificativo 2) e carichiamo l'assembly s1, andiamo nel secondo e carichiamo il sistema s2. Facciamo proc-unit->show register per vedere cos'è scritto nei registri. **L'unica cosa che non abbiamo messo è il contenuto della tabella dei vettori**, da un punto di vista dell'esercizio s1, a destra, con memory 1, non lavora con le interruzioni per cui possiamo anche non farci niente. Il problema è s2 che usa le interruzioni, IRQA è collegata al livello 3, autovettore 27, ovvero *se andiamo in memoria e facciamo show ROM e vediamo cos'è scritto in 6c, vediamo tutti 0.* Se facessimo girare il programma non funzionerebbe. Allora, **facciamo load memory e prendiamo s2, ora vediamo che in 6c abbiamo 00008700, dove 8700 è l'indirizzo di partenza dell'ISR, messo nel codice**. Ora abbiamo settato tutto quello che serve per far funzionare il dispositivo. Facciamo un'istruzione alla volta. Il sistema di destra è Tx, a sinistra Rx.





Cominciano con l'inizializzazione della periferica, solo i registri B vengono modificati a destra, a sinistra solo A. A destra vedremo che il registro direzione diventa tutti 1 e il registro di controllo ha la configurazione vista. L'ultimo bit di controllo è 0 perché non usiamo le interruzioni, a sinistra CRA avrà il primo bit alto.

Eseguiamo, vediamo che accade proprio questo, ci sono le configurazioni messe. Ora a sinistra si azzera solo lo SR che è partito con S alto e al centro tutti 1, a destra vediamo cosa accade quando facciamo la scrittura. Inizialmente mettiamo nei registri interni cosa utilizziamo, a sinistra si ferma perché va sul loop. Partiamo con invio, fittizio, non fa niente, CRB7 è già basso, possiamo eseguire, ora **dobbiamo fare la vera scrittura, si abbassa CB2, a sinistra CA1, per cui a sinistra si scatena l'interruzione: se vediamo CRA a sinistra il bit più significativo si è alzato perché CRA7 va di pari passo con l'interruzione**, ora parte l'interruzione a sinistra, i due sistemi sono in parallelo per cui ad ogni click avanzano entrambi, a destra continuiamo a fare l'invio e a sinistra la ricezione. A sinistra stiamo nell'interruzione, per prima cosa **ci sono le operazioni di copie del contenuto dei registri dello stack, cosa da fare di buona norma per salvare il contenuto dei registri sporcati dall'interruzione**. Ora a destra dobbiamo aspettare sempre finchè CRB7 non torna alto, ovvero quando il sistema sinistra legge. **A sinistra poi facciamo la lettura, vuol dire che il CRA7 si abbassa ma la lettura stessa fa abbassare CA2, che fa abbassare CB1, che non fa alzare l'interruzione ma comunque fa alzare CRB7**, ora a sinistra facciamo il ciclo di nuovo, prende il valore ed esce dal ciclo, l'handshaking è completato ma dobbiamo riabbassare il ciclo, ritorniamo a invio e con la lettura fittizia si abbassa CRB7. Siamo tornati alla posizione di partenza, il sistema che riceve è tornato in loop jmp loop, la ricezione al punto di partenza e possiamo inviare il secondo carattere.



> Se volessimo inventare un protocollo dovremmo usare cosa abbiamo a disposizione del modello di programmazione, ovvero solo i registri nel caso di Asim, dovremmo inventare un protocollo di comunicazione a livello applicativo, scriviamo determinati dati che hanno un significato. Questa è una limitazione del nostro simulatore, **il modello di programmazione è quello che ci mette a disposizione Asim con le sue risorse**. La PIA è programmabile, per cui possiamo realizzare un dispositivo fisico, con una board, su cui mettiamo gli interruttori che ci pilotano CA1 e CA2, così da definire il protocollo. Anche in configurazione con Asim possiamo configurare i due bit che abbiamo messo a 02 diversamente, se usiamo il dispositivo fisico possiamo configurarlo diversamente. Abbiamo anche detto che il pedice 2 sia sempre in uscita e pedice 1 sempre in ingresso, ma possiamo configurare differentemente, nei vari modi visti.

Codice S1_Rx con interruzioni

Vediamo il caso in cui anche in trasmissione sono abilitate le interruzioni, **per abilitare le interruzioni anche sul porto B basta mettere un 1 nella posizione meno significativa**, come nel sistema Rx s2 precedentemente. Come facciamo a far funzionare il sistema con le interruzioni anche sul porto B? L'interruzione in trasmissione non dice trasmetti ma solo che il carattere è stato consumato per cui può trasmettere. *L'evento che scatena l'interruzione in trasmissione è "il carattere che hai mandato prima è stato ricevuto, consumato".* Questa cosa deve subito fare capire che il meccanismo delle interruzioni va scatenato in qualche modo. **Almeno il primo carattere non lo invia l'interruzione** perché si scatena da sola, in base all'evento. **Abbiamo lasciato la lettura fittizia** perché se assumiamo che quando accendiamo la macchina non siamo sicuri dello stato dei bit, per cui potrebbe essere alto, facciamo la lettura fittizia. *Anche lato trasmittente usiamo la variabile globale per il conteggio*, altrimenti abbiamo lo stesso problema di prima per mantenere il conteggio tra esecuzioni successive dell'ISR. **Il main si prende gli indirizzi, azzera il registro di stato e invia il primo carattere, incrementando il conteggio**. La scrittura fa abbassare CB2, quindi dall'altro lato CA1, in s2 c'è l'interruzione, CRA7 si alza**, il sistema ricevente legge e la lettura fa riabbassare CRA7, contemporaneamente CA2, quindi CB1 si abbassa e si attiva IRQB**. Mettiamo l'ISR nella locazione 8800, in Asim abbiamo detto che le interruzioni di IRQB devono essere gestite dagli autovettori di livello 4, ovvero il 28°, l'indirizzo della tabella delle eccezioni allora non è 6c ma 70, in cui scriviamo 8800**. L'interruzione viene svegliata perché il carattere è stato ricevuto, per cui invia il nuovo carattere**. ***La questione della lettura fittizia è attuale, CRB7 non si abbassa se non leggiamo per cui la dobbiamo mantenere altrimenti non si scatenerebbe tutto il meccanismo***, abbassiamo scriviamo, usiamo sempre il modo di indirizzamento con base più spiazzamento, incrementiamo il conteggio e usciamo, perché *lo scopo è solo inviare e incrementare i.* La prossima volta ISR verrò invocata per il terzo carattere, poi per il quarto e così via. è estremamente semplice, la configurazione da usare in Asim è la stessa, carichiamo la memoria e lo eseguiamo.

```
MAIN    JSR    DVBOUT    ;

    MOVEA.L #PIACB,A1    ;
    MOVEA.L #PIADB,A2    ;
    MOVEA.L #MSG,A0 ;indi

* invio primo carattere:
INVIO1  MOVE.B  (A2),D1
    MOVE.B  (A0),D1      ;
    MOVE.B  D1,(A2)      ;
*                  ;ciò
*                  ;un'i

    MOVE.B  #1,COUNT

    MOVE.W  SR,D0    ;legg
    ANDI.W  #$D8FF,D0 ;ma
    MOVE.W  D0,SR    ;pone


LOOP    JMP LOOP     ;cicl
```

```
DVBOUT  MOVE.B  #0,PIACB
    MOVE.B  #$FF,PIADB
    MOVE.B  #%00100101,PIACB
*
    RTS
```

```
INVIO   MOVE.B  (A1),D2
    MOVE.B  (A0,D1),(A1)
*                  ;datc

    ADD.B   #1,D1
    MOVE.B  D1,COUNT
```

Per far girare i file, per ogni periferica troviamo anche il software Asim da usare perché esistono due versioni differenti, fatte in tempi differenti, una non va bene per la usart, per risolvere i problemi di incompatibilità in ogni cartella abbiamo l'eseguibile da usare per quell'esercizio. Abbiamo 3 sottocartelle con gli esempio, TxRx è il primo esercizio. Communic collega due sistemi con pia e terminale, in modo più complesso non per il driver ma per i dispositivi. Per il driver di funzionamento di una periferica, per le interruzioni, è tutto fatto nei due esercizi di base precedenti.