Esercizio in Motorola 68k

Esame di

Architettura e progettazione dei calcolatori

Agostino D'Amora

M63001784

Indice

Introduzione

In tale documento è mostrata la soluzione della prova intracorso tenutasi il 22 Maggio 2025. Essa è commentata da vari paragrafi e considerazioni sui vari conflitti mitigati e le varie scelte progettuali effettuate.

Traccia

Un sistema è composto da 3 unità, A, B e C, tra loro collegate mediante due periferiche parallele che interconnettono A con B e A con C rispettivamente. I messaggi hanno un primo carattere identificativo che può essere pari a 0 a un valore diverso da 0. Il sistema opera in due fasi successive come descritto di seguito:

1. A riceve K messaggi di N caratteri da B e da C in modo alternato. In ordine non prefissato, quindi si pare da B o da C, e non ci sono sovrapposizioni fra i messaggi ricevuti da B e da C

2. Al termine della fase1, il nodo A continua nella stessa modalità alternata e termina la ricezione dei messaggi se due messaggi ricevuti (dalle due diverse periferiche) hanno il carattere identificativo del messaggio pari a 0

Capitolo 1

Architettura del sistema

In tale capitolo è mostrata la strutturazione dell'architettura del sistema richieso con commenti vari al file di configurazione per la sua simulazione in ambiente ASIM

1.1 Architettura complessiva

L'architettura complessiva del sistema è data dalla connessione di tre sistemi mediante periferiche parallele (PIA). Tali periferiche sono collegate opportunamente al sistema come mostrato in figura [1.1]

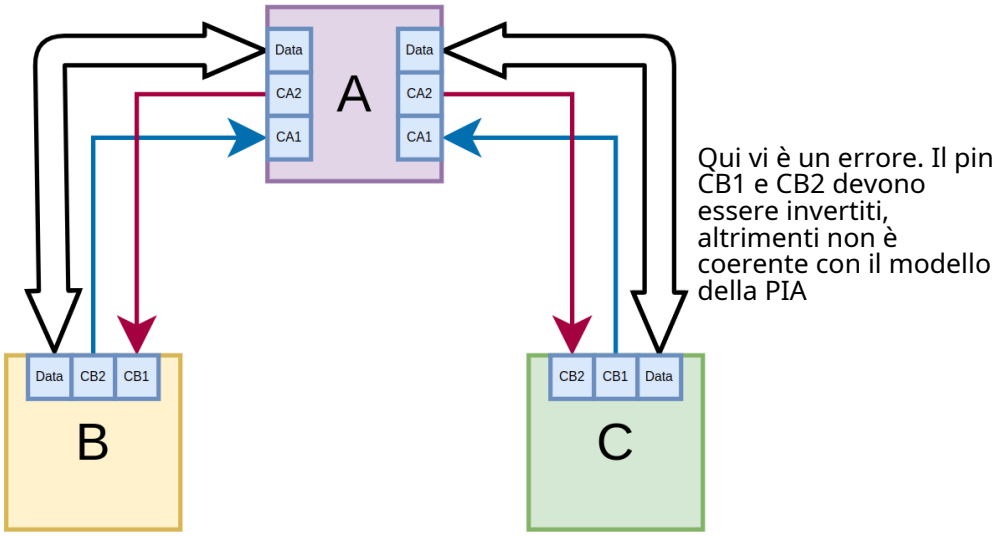

Figura 1.1: Architettura del sistema completo (con evidenziati i collegamenti tra le varie PIA)

1.2 Architettura del singolo sistema

Le architetture interne dei sistemi sono classiche architetture di Von-Neumann, caratterizzate da:

- **Processore**
- **memoria**
- **Periferiche I/O**

1.2.1 Sistema A

Il sistema A, in particolare, ha un architettura che presenta 2 differenti PIA, il che rende l'architettura più "pesante". Il sistema A è configurato come un sistema autovettorizzato, quindi le periferiche PIA saranno collegate opportunamente al processore, che provvederà poi esso stesso a prelevare, dalla tabella delle ISR autovettorizzare, l'indirizzo di memoria associato. Il sistema A, come collegamento delle componenti interne, rispetta l'architettura mostrata nella figura [1.2]

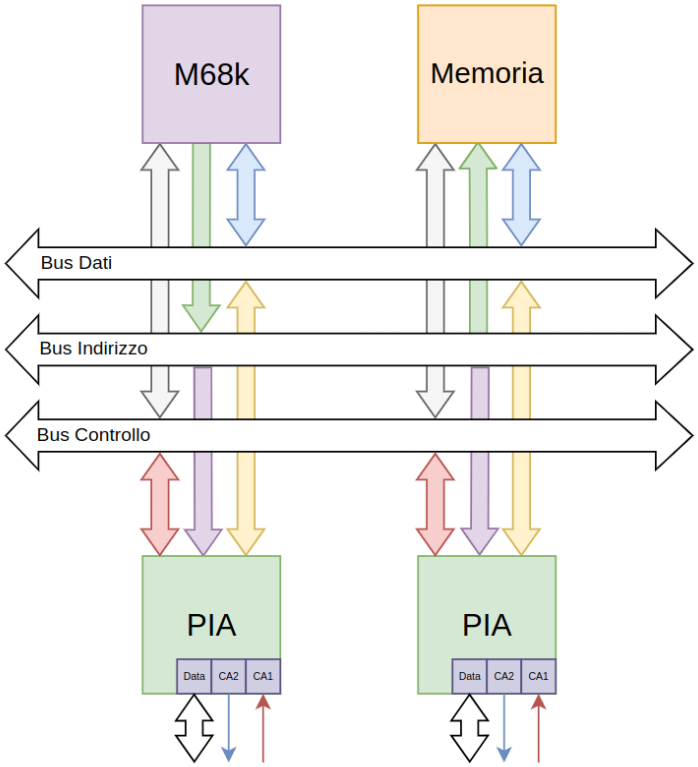

Figura 1.2: Architettura interna del sistema A

1.2.2 Sistemi B e C

I sistemi B e C a differenza del sistema A, risultano più semplice nella loro costruzione, poichè comprendono una singola PIA, quindi conservano la struttura originale ma leggermente variata. Tale struttura è visualizzabile alla figura [1.3]

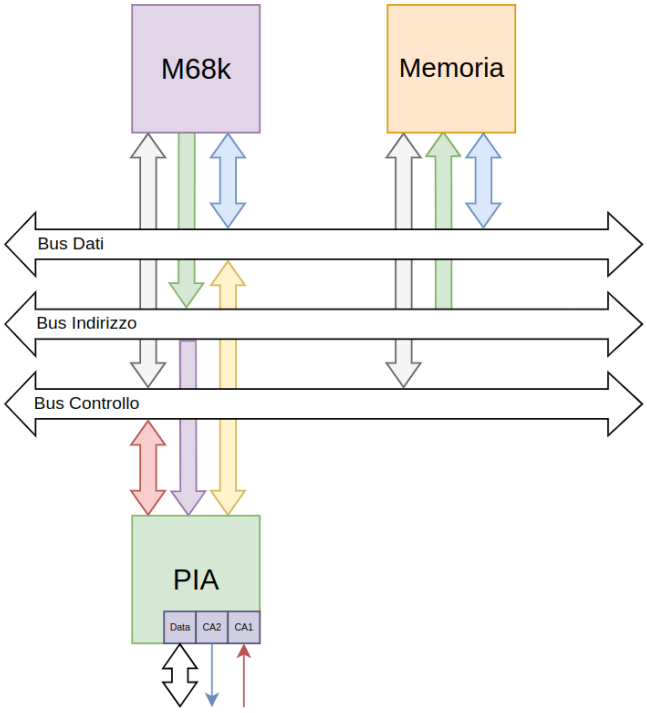

Figura 1.3: Architettura interna dei sistemi B e C

Capitolo 2

Diagramma della memoria

In tale capitolo è mostrata la strutturazione effettiva della memoria per il solo sistema A. Sarà fatto un riferimento particolare all'indirizzamento secondo ASIM (Ambiente in cui il sistema sarà effettivamente simulato)

2.1 ROM

Nella memoria ROM sono memorizzate tutte le informazioni che durante il flusso di esecuzione non cambiano mai. Ad esempio, per il nostro particolare esempio, nella memoria ROM, è memorizzata la tabella per la gestione delle periferiche in modalità autovettorizzata. Andando a visionare la divisione della memoria dal file di configurazione ASIM, notiamo che alcuni indirizzi di memoria vengono dedicati alla ROM. È pertanto importante, all'atto dell'inserimento del sistema, essere accorti alla corretta configurazione della memoria. In particolare per la memoria ROM si avrà la struttura presentanta nella figura 2.1. Tale memoria contiene tutti i primi indirizzi. I primi 24 sono dedicati alle eccezioni mentre quelli successivi (dal 24 esimo al 31 esimo), sono dedicati all'inserimento degli indirizzi di memoria delle ISR di gestione delle periferiche in modalità autovettorizzata. In questo caso stiamo considerando il sistema C collegato alla PIA con priorità 4, ed il sistema B collegato alla PIA con priorità 5. Tale differenza di proprietà, però, non sarà usata in alcun modo in fase di definizione della specifica ISR

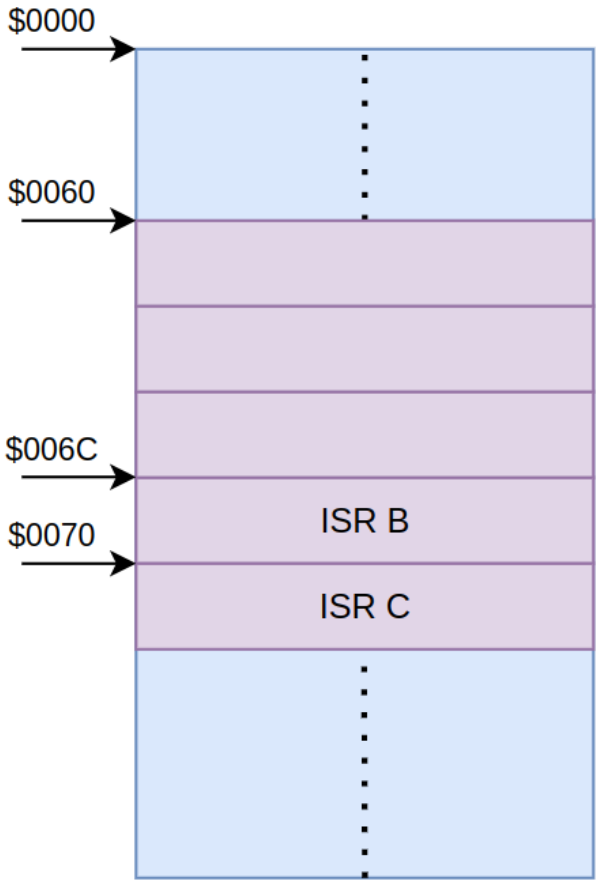

Figura 2.1: Strutturazione della memoria ROM in ASIM

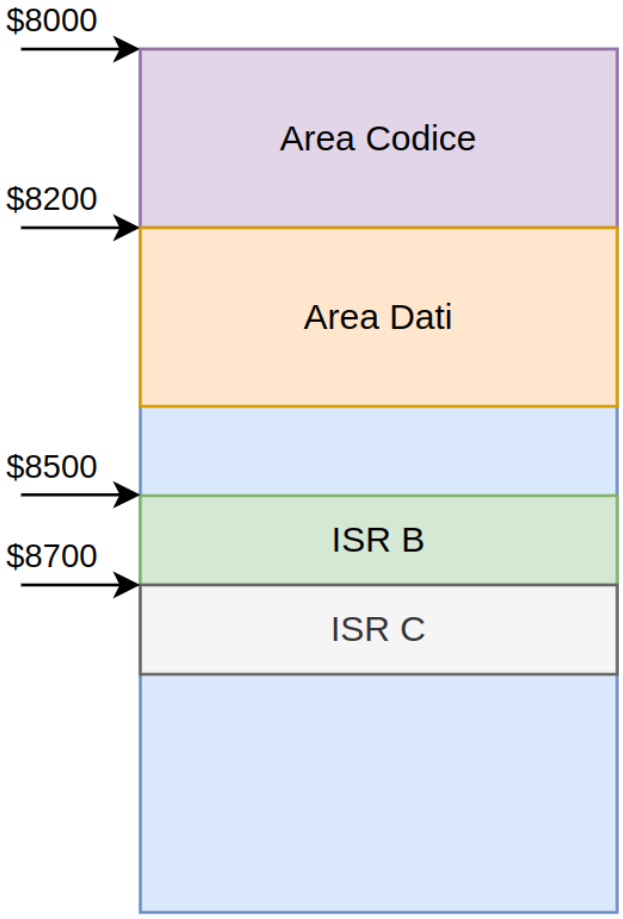

Figura 2.2: Struttura interna della memoria RAM

2.2 RAM

La memoria RAM è quella che conterrà i dati ed il codice che dovranno essere utilizzati al fine che il sistema funzioni correttamente. Difatti saranno identificate 4 macro-aree:

- **Area Dati**
- **Area Codice**
- **ISR-B**
- **ISR-C**

Più precisamente, la struttura della memoria RAM, è quella mostrata nella figura [2.2]

Capitolo 3

Diagrammi temporali

In tale sezione saranno strutturati i diagrammi temporali riguardanti il protocollo utilizzato dalla PIA in base alla sua specifica configurazione

3.1 Handshacking

Nel caso della PIA, una delle sue modalità di funzionamento, gestisce il trasferimento dei dati mediante dei meccanismi appositi di handshacking, tali meccanismi, particolarmente, sono descritti dagli specifici settaggi che vengono fatti riguardo la gestione dei pin CA2 e CA1. Il più classico funzionamento della PIA è quello mediante la configurazione della seguente modalità:

- **CA1** configurato con la modalità 01, il che attiva CRA7 appena rileva un fronte di discesa su CA1. Oltre a questo, essendo che il bit meno significativo è 1, sarà anche attivato il segnale di interruzione
- **CA2** configurato con la modalità 100, il che imposta il pin CA2 come uscita e sarà impostato come:
  - **basso**: Quando sarà effettuata un operazione di lettura su PRA
  - **alto**: Quando CRA7 diventa alto per via di una variazione (qualunque) di CA1

Date le seguenti specifiche di funzionamento possiamo tirare furi i diagrammi temporali visualizzabili alla figura [3.1].

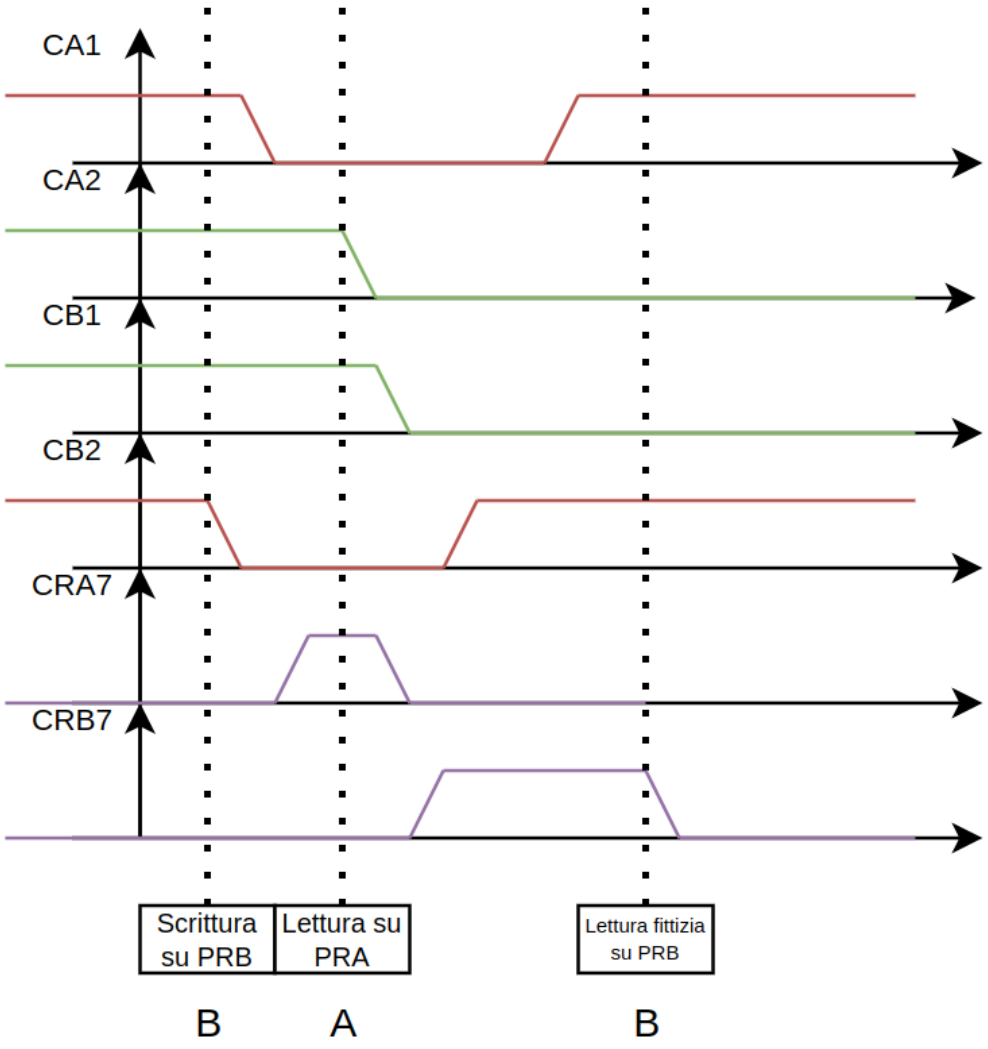

Figura 3.1: Diagrammi temporali del processo di handshacking della PIA descritto precedentemente

Capitolo 4

Pseudocodice

In tale sezione sarà commentato lo pseudocodice per le varie implementazioni ristrette al solo sistema A. Quindi saranno escluse eventuali implementazioni di alto livello dei sistemi B e C.

4.1 Area Dati

Nell'area dati sono specificate tutte le variabili e tutte le macro a cui si fa riferimento all'interno dell'area codice

```
// -----> [DATA] <----- //
// Dichiarazione della variabile del TAS
byte TAS; // Semaforo

byte possesso = 0

// Buffer dato
byte buff[] = {0}

// Variabili di conteggio
byte ch_msg = 0
byte ch_tot = 0
byte msg_b = 0
byte msg_c = 0

// Variabili per la terminazione
byte end_b = 0
byte end_c = 0

byte N = 3 // Lunghezza del messaggio

byte K = 5 // Quantita' di messaggi prima della fase 2
```

Listing 4.1: Area Dati

4.2 Area Codice

Nell'area dati è specificato tutto il flusso di funzionamento del codice, con particolati dettagli dati alla giusta implementazione delle ISR, che andranno a gestire l'apposita comunicazione con le opportune periferiche

4.2.1 Main

Nel main si vanno a:

- Configurare i dispositivi PIA
- Attivare le interruzioni
- Nel caso, qualche inizializzazione

```
void Main(){
    // Avvio delle subroutine di configurazione delle PIA
    PIAB.configurePIA()
    PIAC.configurePIA()

    // Abilitazione delle interruzioni
    SR = SR & $D8FF

    while(true){
        // Ciclo caldo
    }
}
```

Listing 4.2: Main di partenza del sistema A

4.2.2 PIA Configuration

L'inizializzazione della PIA avviene mediante una apposita subroutine che viene chiamata ed utilizzata durante la fase di esecuzione del programma. Nel caso dello pseudocodice è stata immaginata come la funzione di una specifica classe, che non sarà realmente implementata ma solo utilizzata per rendere lo pseudocodice più pulito.

```
1  void configurePIA(){
2          // Configuro i collegamenti data tra le due pia come
               ingressi (ponendo tutti i bit a zero in DRA)
3          this.PIA["DRA"] = %00000000
4
5          // Configurazione del registro di controllo effettivo
               della PIA
6          // CRA7-6 = 00 -> Bit di sola lettura
7          // CRA 5-4-3 = 100 -> modalita' di funzionamento del
               pin CA2
8          // CRA 2 = 1 -> Al prossimo accesso ad indirizzo pari
               (DRA)
9          // si accedera' all'indirizzo PRA (Registro dato per
               la comunicazione)
10         // CRA 1-0 = 01 -> modalita' di funzionamento del pin
               CA1 (precisamente la sua gestione)
11         this.PIA["CRA"] = %00100101
12 }
```

Listing 4.3: Subroutine di configurazione dei dispositivi PIA

4.2.3 ISR B e C

La ISR di gestione dei sistemi B e C è esattamente speculare, pertanto se ne evince l'implementazione di una univoca (ISR di B). Per comprendere al meglio tale codice si ricordano le seguenti presupposizioni:

- **Possesso**: La variabile assume valore 1 quando il turno è di B, valore -1 quando invece è di C. Mentre 0 viene associato solo in una fase iniziale (indecisione su chi deve iniziare)

- **Semaforo**: Semaforo all'interno dello pseudocodice viene utilizzato come una normale variabile, ma il suo utilizzo passa per funzioni atomiche (TAS) che permettono sia di controllare che di settare (eventualmente) il registro target; tale operazione atomica garantisce la mutua esclusione su determinati dati

```
/*
*   Nella traccia si fa intendere che l'ordine con cui
   vengono ricevuti i messaggi
*   non e' prefissato fino a che non vi e' la prima ISR (o
   prima iterazione), che una volta svolta
*   dettera' l'ordine effettivo degli andamenti.
*/
function ISR_B() {
    if (possesso == 0){
        if(TAS == Verde){
            TAS = Rosso

            if (possesso == 0){
                possesso = 1
            }

            TAS = Verde
        }else{
            RTE
        }
    }
```

Listing 4.4: Componente per la gestione della mutua esclusione in fase iniziale: ISR_B

```
// Arrivato a questo punto sono sicuro di avere un ordine prefissato, quindi la variabile
  // possesso potra' essere solo o 1 o -1
  if (possesso == 1 && end_b == 0){
      buff[ch_tot] = PIAB.readPRA()
      ch_tot++
      ch_msg++

      if (ch_msg == N){
          ch_msg = 0
          msg_b++

          if (msg_b >= K){
              // Se entro qui vuol dire che sono nella fase 2

              if (buff[ch_tot-N] == 0){
                  if (msg_b == msg_c){
                      if (end_c == 1){
                          // Caso fine, sono nella stessa iterazione con entrambi i valori iniziali a 0
                          end_b = 1
                          RTE
                      }
                  }else{
                      // Caso in cui devo aspettare ancora un messaggio, imposto l'end_b
                      end_b = 1
                  }
              }else{
                  // Caso in cui il primo elemento e' diverso da 0 e sono a fine iterazione
                  if (msg_b == msg_c){
                      // Indipendentemente dal valore di end_c lo vado ad impostare a 0
                      end_c = 0
                  }
              }
          }

          possesso = -1

          // Caso in cui C si e' sospeso mentre eseguiva B
          // 1. Verifico se si e' effettivamente sospeso
          if (c_suspended == 1){

              // 2. Controllo che da quando ho settato possesso non abbia proseguito e che non
```

```
                    abbia finito
                if (possesso == -1 && end_c == 0){
                    ch_tot++
                    ch_msg++
                    buff[ch_tot-1] = PIAC.readPRA()
                }
            }
        }
    }
}
```

Listing 4.5: Componente che gestisce l'alternanza fissa: ISR_B

Appendice A

Appendice

A.1 Sistemi B e C

In tale sezione andremo ad approfondire la strutturazione dei due sistemi periferici B e C. In particolare la loro gestione mediante software (pseudocodice del sender). Tali sistemi sono identici, il che quindi ne richiede una singola implementazione ed una replica (nel caso della simulazione in ASIM)

A.1.1 Pseudocodice

Per quanto riguarda i sistemi B e C, essi sono esattamente identici. La loro implementazione, a differenza del sistema A, richiede una logica molto più semplice, attua solo all'invio degli specifici messaggi

A.1.2 Main

Il main di tali sistemi si occupa di tutta la gestione delle varie periferiche. Si può scegliere se gestire l'invio dei messaggi mediante l'utilizzo di Interrupt (richiede che il main invii solo il primo carattere). Oppure mediante il polling sul bit più significativo del registro di controllo. In questo caso si utilizzerà il caso con interrupt.

```
1 void Main(){
2     // Configuro la PIA
3     PIA.configurePIA_TX()
4
5     // Abilito le interruzioni
6     SR = SR & $D8FF
7
8     // Incremento il contatore per i caratteri presenti in
        memoria
```

```
9       // (scritti in fase di definizione del codice M68k)
10      byte_send ++
11
12      // Invio i dati (lo faccio post-incremento per evitare
           conflitti dovuti all'interrupt)
13      PIA["PRB"] = buffer[byte_send-1]
14
15      while(true){
16          // Ciclo caldo
17      }
18  }
```

Listing A.1: Main dei sistemi B e C

A.1.3 Configurazione PIA

Per la configurazione della PIA si prosegue allo stesso modo del sistema A, stando attenti a configurare il porto B. Ciò è dovuto principalmente al principio di funzionamento del pin CB2, che si abbasserà, data la modalità 100, solo all'atto della scrittura sul registro PRB

```
1   void configurePIA_TX(){
2       // Configuro i collegamenti data tra le due pia come
           uscite (ponendo tutti i bit a uno in DRA)
3       this.PIA["DRB"] = %11111111
4
5       // Configurazione del registro di controllo effettivo
           della PIA
6       // CRA7-6 = 00 -> Bit di sola lettura
7       // CRA 5-4-3 = 100 -> modalita' di funzionamento del pin
           CA2
8       // CRA 2 = 1 -> Al prossimo accesso ad indirizzo pari (
           DRA)
9       // si accedera' all'indirizzo PRB (Registro dato per la
           comunicazione)
10      // CRA 1-0 = 01 -> modalita' di funzionamento del pin CA1
            (precisamente la sua gestione)
11      this.PIA["CRB"] = %00100101
12  }
```

Listing A.2: Configurazione della PIA nel caso dei sistemi trasmittenti

A.1.4 ISR Invio

La ISR di gestione dell'invio gestisce il caso in cui sarà scatenata l'interrupt al trasmettitore (fine della fase di handshacking (quando viene letto il valore nel caso 100)). Tale ISR quindi dovrà gestire, quindi, il solo invio

del prossimo carattere. Nel mio caso l'ho pensato come un trasmettitore "Ignorante" che non avrà controllo su quanti messaggi sta inviando.

```
void ISR_TX(){
    // Lettura fittizia
    a = PIA["PRB"]

    // Invio il prossimo carattere
    PIA["PRB"] = buff[byte_send]
    byte_send++
}
```

Listing A.3: ISR di gestione dell'invio dei caratteri nei sistemi B e C