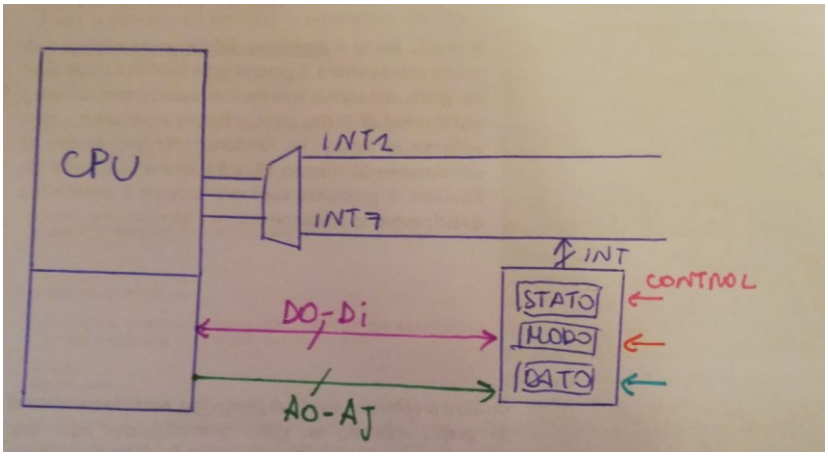
PIA – PERIPHERAL INTERFACE ADAPTER

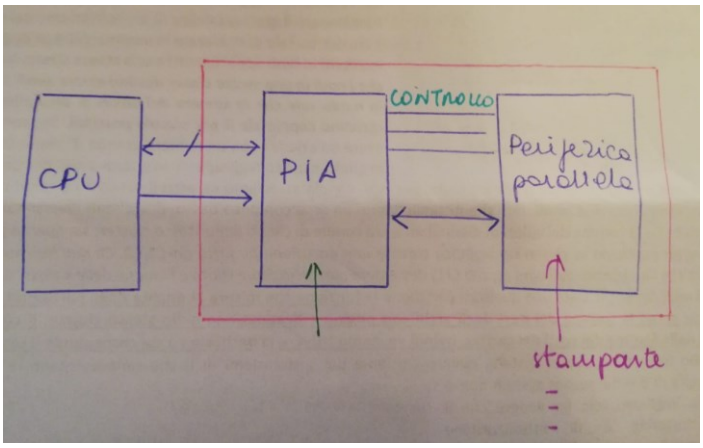
Vediamo *come possiamo usare le periferiche e come programmare i driver per interfacciare diversi dispositivi di I/O con il processore*. Abbiamo fatto un percorso per capire come è possibile migliorare le prestazioni di un processore, lavorando prevalentemente sul parallelismo interno. Per migliorare le prestazioni di un processore abbiamo differenti componenti, *di solito possiamo agire sul singolo componente sia tecnologicamente sia aumentando il parallelismo interno*; questo è vero sia per il processore sia per la memoria (gerarchia di memoria) al fine di garantire tempi di accesso del processore alla memoria abbastanza veloci in modo da avere il dato in un ciclo di clock. Fino ad adesso non abbiamo mai posto il problema di **come il processore possa comunicare con elementi esterni al binomio processore-memoria**. Il processore non è un monolite, comunica con l'esterno sia per ricevere dati in ingresso sia per fornire dati in uscita. Abbiamo molte periferiche di I/O che permettono di accettare dati dall'esterno e permettono di mandare dati all'esterno, come unità video o stampante. **Le periferiche, per poter funzionare con il processore, dispongono di due meccanismi differenti**. **Dobbiamo implementare meccanismi perché le periferiche sono asincrone rispetto al flusso di esecuzione del processore**: abbiamo un meccanismo di tipo **polling**, per cui il processore è in attesa attiva, costantemente interroga lo stato di una periferica per vedere se è pronta a comunicare con lui; oppure può comunicare con il **meccanismo delle interrupt** che possono essere gestite e servite nel M68K in diverse modalità, vettorizzata e auto-vettorizzata. *L'interrupt viene verificata in un momento specifico del ciclo del processore e comporta una serie di azioni* che portano ad individuare la periferica che ha interrotto, capire se la periferica ha la giusta priorità per poter essere eseguita, capire l'indirizzo della ISR cui "saltare" (nella pratica dobbiamo aggiornare il PC a quell'indirizzo) per gestire l'interrupt e gestire il cambio di contesto, il che vuol dire salvare il contesto corrente e switchare, caricare il contesto nuovo. Alla fine della procedura si ripristina il contesto della procedura chiamante per riprendere l'esecuzione del programma principale esattamente dal punto in cui avevamo interrotto.

Come è collegata la periferica al processore?

Abbiamo il processore 68K, che ha 3 linee di interruzioni collegate ad un PIC o decoder che codifica 7 diverse linee da int0 a int7. Su una certa linea abbiamo una periferica che ha i **tre registri, stato – modo – dato, che rappresentano l'interfaccia di una periferica**. Questa è una linea di interruzione, tipicamente di uscita, dalla *periferica la quale è collegata al processore sia con le linee indirizzo che con le linee dati*. **Le line D0-Di sono bidirezionali mentre quelle indirizzo A0-Aj sono unidirezionali**. Ci sono anche altre linee di controllo oltre la linea di interruzione che servono per comunicare. **Dobbiamo collegare la periferica** e abbiamo a disposizione il *simulatore ASIM del M68K;* poi andrà programmata la periferica che comporta due aspetti. Il primo è **l'inizializzazione**, il secondo è la **scrittura della ISR o delle ISR** perché ce ne possono essere più di una (dipende dalla periferica) che permette lo scambio dei dati, quindi se è di input permetterà di leggere i dati, se è di output permetterà di scrivere i dati. **Nell'inizializzazione settiamo il modo operativo della periferica**, scriviamo nel registro modo della periferica. **Nella ISR per la parte di scambio dati usiamo il registro dato, mentre per il protocollo di scambio dati vero e proprio usiamo sia il registro di stato che il controllo**. Capiremo periferica per periferica sia la parte di collegamento fisico e anche di protocollo, sia la parte di inizializzazione che le ISR corrispondenti al fine di realizzare collegamenti più complessi tra nodi di processori. Fino ad ora abbiamo lavorato su sistemi monoprocessore, dovremo vedere le architetture multiprocessore e multicomputer che sono nodi elaborativi distribuiti che devono comunicare tra loro, per cui dobbiamo aver ben presente cosa significa scambio di informazioni per poter eseguire un obiettivo comune.



La PIA è un dispositivo periferico parallelo con parallelismo ad 8 bit. Il bus di collegamento tra il processore e l'adattatore e l'adattatore e la periferica è di 8 bit.



Noi usiamo gli adattatori, per cui se la PIA si collega al processore con il parallelismo ad 8, la PIA poi deve essere fisicamente collegata ad una periferica. Quella a destra è la periferica parallela, la centrale è la PIA e il processore a sinistra, perciò **la PIA è tra periferica e processore per gestire protocollo di sincronizzazione e di interruzione**. Tra PIA e periferica abbiamo un altro bus dati e una serie di segnali per il controllo. Questa nel riquadro rosa è tutta l'architettura della periferica, la PIA è l'interfaccia parallela che è sempre la stessa, mentre la periferica può essere qualunque, per cui possiamo avere in quel blocco la stampante o un'altra periferica parallela che però è collegata con la PIA allo stesso modo. Vediamo l'interfacciamento della PIA verso il processore e verso un qualsiasi dispositivo parallelo.

La PIA è costituita da due porti separati, quindi *è possibile anche collegare due dispositivi differenti da 8 bit*. Questi dispositivi possono essere anche configurati separatamente, parliamo del porto A e porto B che configuriamo singolarmente. **Ogni porto ha un parallelismo di 8 bit, per cui abbiamo due registri dati nei porti che permettono di comunicare verso le periferiche con due bus separati di 8 bit ciascuno**. **Per ogni porto abbiamo due coppie di linee differenti che sono CA per il porto A e CB per il porto B**, che chiamiamo CA1 e CB1 e CA2 e CB2. **Queste sono le linee di controllo e sono caratterizzate dal fatto che CA1 e CB1 sono sempre in ingresso al dispositivo, mentre CA2 e CB2 sono programmabili**, per cui in funzione del protocollo che vogliamo realizzare possiamo avere l'implementazione, per esempio di handshaking o un'altra modalità di trasferimento a seconda di come CA2 e CB2 siano programmati, sia in ingresso che in uscita. I segnali del 68k prevedono ***un insieme di segnali verso il processore e un insieme verso la periferica***. Verso il processore abbiamo un bus dati indicato con i pin Di di 8 bit, abbiamo altri segnali ricavati dalle linee indirizzo che permettono di selezionare il chip, quindi con 3 segnali differenti di chip-select abilitiamo o meno la periferica. Abbiamo un unico segnale di lettura/scrittura, un segnale generale di enable ed un segnale di reset. **Dal lato processore avendo a disposizione 6 registri interni, 3 per porto, abbiamo bisogno di altre linee di indirizzo provenienti dal processore per indirizzare questi registri**.

- Data Bus bidirezionale verso la CPU (D0..D7)
- Segnali di selezione del Chipset (CS0, CS1, CS2)
- Segnale di Lettura\Scrittura (R\W)
- Segnale di Enable e Reset
- 2 Data Bus verso le Periferiche ad 8 bit (PA0..PA7, PB0..PB7)
- 2 Linee di interruzione esterne (CA1, CB1)
- 2 Linee di controllo periferico (CA2, CB2) programmate per funzionare come IN\OUT
- 6 Registri Interni (2 segnali RS0, RS1 per indirizzarli)

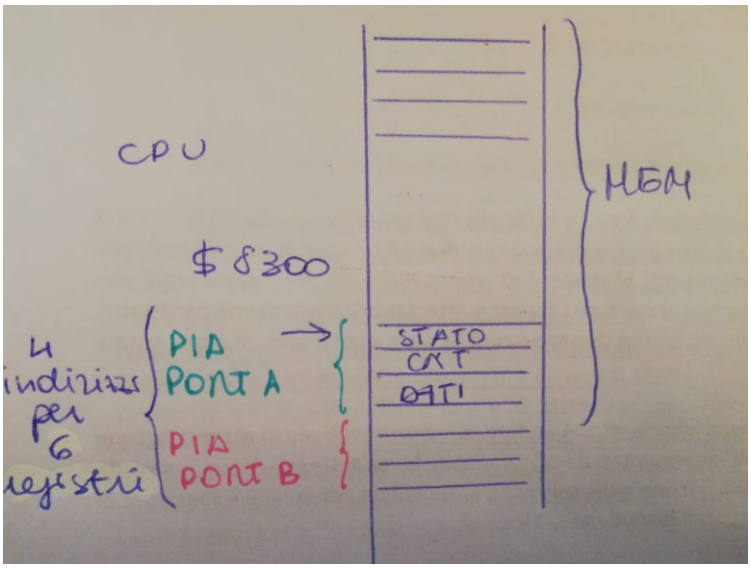
Ai registri visti prima, presenti nella periferica, stato modo dato, come si accede? Il 68k ha 8 registri per i dati interni al processore e 8 per i dati esterni. Ci sono due filosofie per accedere ai dati ed ai registri di I/O. La prima politica prevede di ***avere istruzioni dedicate e porti dedicati per l'IO, il che significa associare nella memoria indirizzi specifici con cui fare operazioni di read and write e questo si chiama memory I/O (NON è la tecnica usata dal 68000)***. La famiglia Intel usa istruzioni dedicate per accedere a questi determinati indirizzi.

Viceversa, il 68k usa tecnica differente, per cui questi registri sono mappati nella memoria principale dell'architettura, non abbiamo bisogno di istruzioni particolari. **L'accesso in memoria avviene sempre con l'istruzione di MOVE, cioè possiamo accedere ai registri interni della periferica esattamente allo stesso modo con cui accediamo ad un indirizzo di memoria**, dichiariamo 3 variabili stato-modo-dato mappate su tre indirizzi con tre etichette all'interno del programma e con questi tre indirizzi possiamo accedere direttamente ai registri. Questo si chiama **memory mapped**, non abbiamo bisogno di istruzioni aggiuntive per fare I/O.

Avendo a disposizione 6 registri interni, 3 per porto, servirebbero in teoria 3 linee indirizzo per accedere a 6 indirizzi. **Il 68k ha una tecnica di indirizzamento interna per cui basteranno 2 segnali RS0/RS1** per accedere agli indirizzi al sistema.

Dal lato periferica abbiamo due data bus, due linee di interruzioni esterne CA1/CB1 e due linee di controllo CA2/CB2 programmate per funzionare sia in input sia in output. **Non è vero che i data bus associati ai due porti sono bidirezionali, ma sono unidirezionali, solo che la direzione è programmabile**. *In fase di inizializzazione della periferica possiamo decidere*

che un porto è di ingresso e uno di uscita oppure entrambi di ingresso o entrambi di uscita. I due porti e quindi i dati ed i segnali associati ad i registri dati PA0-PA7/ PB0-PB7. Possono essere sia ingresso che uscita, **la direzione dei dati può essere programmata settando i bit nel registro Data Direction Register - DDR**, per cui se vogliamo settare tutte le linee del porto A in uscita metteremo tutti 1 in DDRA. In linea teorica è possibile programmare metà in una direzione e metà in un'altra, però nella pratica non si fa mai perché complica il protocollo di comunicazione con le periferiche. *Le linee D0-D7 sono di interfacciamento con il processore e sono di tipo tristate perché sono collegate al bus del processore e sono 8*. **La direzione dei dati è bidirezionale e dipende dall'operazione richiesta dal processore**, per cui se abbiamo chiesto scrittura da processore a periferica abbiamo abilitato un segnale di scrittura, quindi la direzione è processore->periferica, se abbiamo abilitato un segnale di lettura, la direzione è periferica->processore. Le operazioni in entrambi i casi sono di MOVE, spostamento di dati, ma cambia la direzione a seconda del segnale abilitato.



Il processore ha la sua memoria, con il suo programma. **A determinati indirizzi mettiamo nella memoria il banco di 6 registri, 3 per porto, che sono i registri della PIA**. Mappiamo questi registri in memoria ed abbiamo i registri di stato/modo o controllo/dati. Questa è l'interfaccia della PIA mappata in memoria. Se il programma si fermava per esempio ad $8300, dall'indirizzo successivo possiamo trovare i registri interni della PIA.

Proprio perché la PIA ha due porti potrà essere collegata al processore con due linee di interruzione differenti, una linea di interruzione che si chiama IRQA che è associata al porto A e quella che si chiama IRQB associata al porto B. **Queste linee sono direttamente collegate al processore con tecnica di elettronica che si chiama open drain per realizzare una wired or**. Si possono realizzare funzioni logiche di OR, pur potendo collegare le uscite di questi dispositivi direttamente sulla stessa linea, quindi **non mettiamo una OR esplicita tra tutte le linee di interruzioni per generare l'interrupt i-esima sul processore ma la colleghiamo direttamente perché la tecnologia con cui sono implementate le porte logiche fisiche che generano l'interruzione di per sé realizzano una OR**. Stiamo mettendo in OR le varie interruzioni e **dopo la richiesta di un'interruzione ci sarà una routine software, una ISR,** *che leggerà i bit presenti all'interno del registro di stato per capire che tipo di interruzione è stata richiesta e qual è la ISR corrispondente che dobbiamo eseguire*. Se c'è una interruzione e viene abilitata per la corretta priorità vengono disabilitate eventuali altre richieste di interruzione.

Possiamo vedere l'architettura interna schematizzata in figura. A sinistra è il porto A e a destra il porto B. Il porto A ha il registro PDR, il registro DDR e il registro CR di controllo. Tutti questi sono relativi al porto A, quindi sono **PDRA-DDRA-CRA**, *poi abbiamo gli stessi relativi al porto B*. Vediamo il collegamento esterno. Il registro PDR con parallelismo 8 è collegato con le **linee bidirezionali** verso una **periferica parallela** e lo stesso vale per il porto B. Potrebbe essere anche la stessa periferica che usa entrambi i porti, un'unica periferica a 16 bit. C'è poi la linea **CA1** in ingresso, mentre **CA2** la programmiamo noi. **Nella parte sotto c'è l'interfacciamento con il processore**. *C'è la linea di interrupt **IRQ** generata dal porto A e dal porto B che sono in uscita*. Abbiamo il segnale di reset e l'altro di enable, che sono i due segnali di controllo. Un unico segnale per una read/write, le 8 linee dati verso il processore bidirezionali, 3 linee per abilitare la periferica dalle linee indirizzo (chip select) e poi abbiamo ulteriori linee **RS1 e RS0** che permettono di selezionare dalla linea indirizzo uno dei 6 registri.

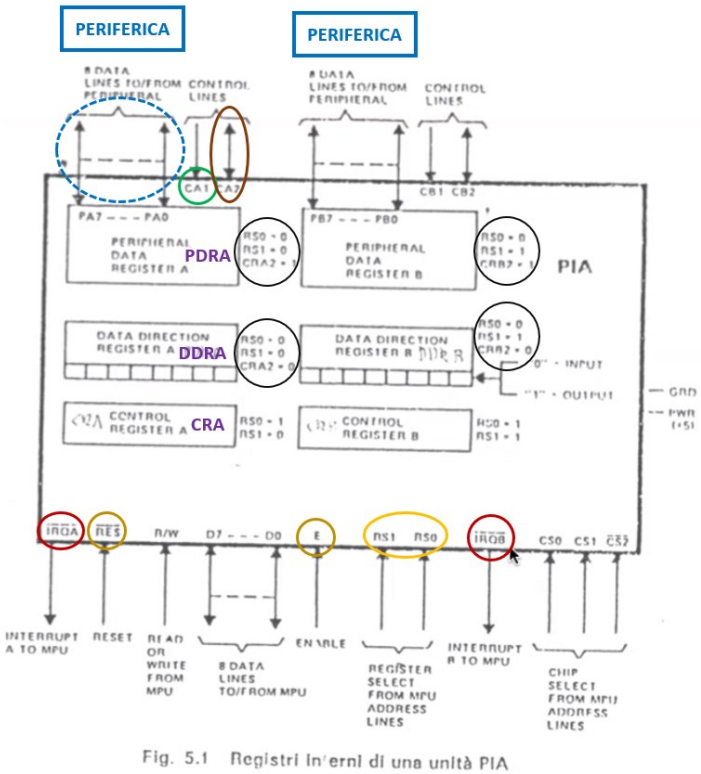


Fig. 5.1 Registri interni di una unità PIA

Con due linee abbiamo 4 indirizzi possibili, eppure con solo 4 indirizzi riusciamo ad indirizzare 6 registri interni. **Questo è possibile grazie ad una tecnica di indirizzamento interno che controlliamo nel CR**: **il secondo bit del CR (CRA2 e CRB2), in funzione del valore, che può essere 0 o 1, permetterà di accedere al registro**

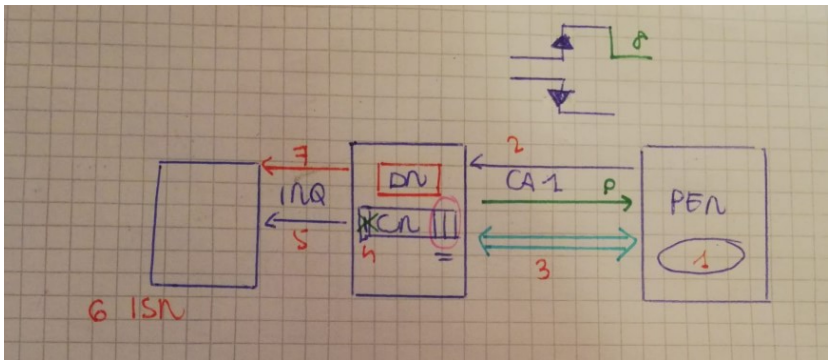
PDR o DDR. Se il processore indica l'indirizzo 10 stiamo accedendo al CRA, se indica 11 stiamo accendendo a CRB, 10 e 11 sono i valori assunti dai bit RS1 e RS0.
Da programma dobbiamo dire se il prossimo indirizzo generato sarà verso un DDR o PDR, allora viene implementato un indirizzamento interno tale che nel registro possiamo mettere 0 o 1 e a seconda di questo valore possiamo indirizzare 4 registri differenti. La combinazione 000 (le combinazioni sono cerchiate in nero in figura) mi fa accedere al DDRA, la combinazione 001 fa accedere al PDRA. La combinazione 010 farà accedere a DDRB, 011 a PDRB. **Nella memoria del processore, ci saranno solo 4 indirizzi perché il registro interno DD si può pilotare da programma per indicare queste linee come funzioneranno**.
Abbiamo 6 registri che riusciamo a gestire con 4 indirizzi con la tecnica di indirizzamento interno, che ritroveremo in più casi. *Ognuna delle periferiche mappa diversamente in memoria i registri interni con tecniche differenti*. **In funzione delle due linee indirizzi RS1 e RS0 e del valore del bit presente in CRA2 e CRB2 avremo un registro selezionato differente**.

RS0	RS1	
0	0	CRA2 0 – *Accediamo al DDRA* 1 – *Accediamo al PDRA*
0	1	CRB2 0 – *Accediamo al DDRB* 1 – *Accediamo al PDRB*
1	0	*Accediamo al registro CRA*
1	1	*Accediamo al registro CRB*

I segnali di controllo CA e CB funzionano differentemente per permettere l'implementazione di un protocollo tra l'interfaccia periferica e la periferica vera e propria***, CA1 e CB1 sono sempre in ingresso***. Se la periferica ha il dato pronto alzerà questo segnale per dire che il dato è pronto; mentre ***CA2 e CB2 possono essere implementati e programmati in funzione del protocollo che implementiamo***. Il protocollo più semplice è l'handshake per cui immaginiamo una linea che indichi il dato è pronto e l'altro che indichi che è stata effettuata l'operazione. **Gli altri tre bit del registro controllo, 3 4 5, permettono di pilotare fisicamente queste linee**.

Il registro di controllo dovrà essere programmato affinché possiamo fare operazioni specifiche e possiamo pilotare l'interfaccia opportunamente con le interruzioni. Il bit 2 serve per pilotare il DDR, i bit 5-4-3 per pilotare CA2 (o CB2 nel caso del CRB), **i bit 6 e 7 servono per abilitare una richiesta di interruzione IRQA1 ed IRQA2**. Quando vogliamo alzare un interruzione verso il processore, possiamo mettere un bit alto in questo registro per dire che c'è un interruzione. Viceversa, **i bit 0 ed 1 permettono di capire se la periferica deve essere sensibile al segnale che riceve in ingresso da CA1 e su quale fronte prima di poter generare un'interruzione.**

Al centro abbiamo la periferica, che ha il registro di controllo CR, poi *abbiamo l'adattatore che sarà collegato ad una periferica con la linea CA1 per il porto 1*. Arriva su CA1 una richiesta da parte della periferica, **in CR dobbiamo dire se vogliamo essere sensibile al segnale e su quale fronte, il fronte di salita o il fronte di discesa**. Questo viene indicato nei primi due bit di CR. A seconda di come li pilotiamo, possiamo abilitare sui bit a sinistra l'interruzione. Se indichiamo che la periferica è sensibile al segnale ed al fronte di quel segnale, appena varia il fronte deve essere alzata la linea interruzione e viene generata l'interrupt request verso il processore.



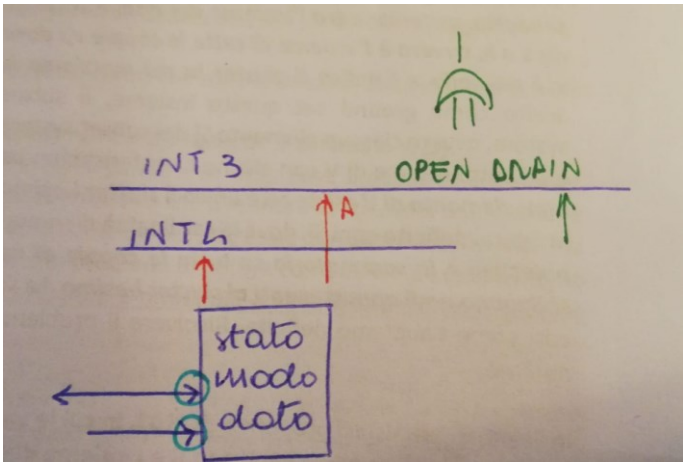
Per esempio, la periferica vuole mandare un dato. Indica che il dato è pronto (1), diciamo alla periferica che il dato è pronto (2), mettiamo il dato sul bus (3), la periferica alza il bit (4) ed alza una linea di interruzione (5). Qui partirà la ISR (6) che va a prendere il dato (7) dall'interfaccia. La periferica appena ha un dato pronto alza e butta sul bus in modo che sia immediatamente disponibile sul DR; ora è l'interfaccia che deve comunicare alla CPU che deve essere stabilito un protocollo di comunicazione basato sulle interruzioni per poter prelevare questo dato. Appena il dato viene mandato (7), si abbassa la linea di interruzione (cioè i bit a sinistra di CR) , viene mandato un segnale alla periferica (P verde) e viene abbassato il segnale (8 verde) per indicare che adesso può riceverne un altro.

Vediamo *il significato dei bit del processore.* **I primi due bit, CRA0 e CRA1, determinano se e su quale fronte la PIA è sensibile o meno alle richieste della periferica**. **Il primo bit CRA0 determina se la richiesta di interrupt deve essere inoltrata al processore attraverso la linea IRQA**. *È come se fosse una maschera, se CRA1 è 0 tutte le richieste che arrivano dalle periferiche non sono inoltrate al processore, sono mascherate; viceversa se è 1, sono abilitati, per cui ogni volta che arriva CA1 in ingresso viene generata automaticamente un interruzione al processore*. Viceversa, **CRA1 permette di pilotare su quale fronte dell'interruzione è riconoscibile l'ingresso CA1.** Dobbiamo indicare sia se è abilitato o non è abilitato e sia se è abilitato sul fronte di salita o di discesa**. CRA2 serve per selezionare il registro dati o il registro direzione**. **Ci sono i 3 bit centrali che permettono di pilotare la linea CA2**. **Il bit 5 stabilisce se la linea CA2 funziona da ingresso o uscita**: *se è 0, CA2 viene programmata in ingresso e funziona come CA1*. **Se abbiamo CA2 in ingresso, dobbiamo pilotarla come CA1, per cui CRA4 e CRA3 funzioneranno come i bit 0 ed 1 per decidere se è abilitato e su quale fronte è abilitato**. Se invece CRA5 è 1, **CA2 è pilotata per funzionare in uscita allora gli altri due bit determinano un modo di funzionamento, qual è il protocollo di trasferimento dei dati, in che modo i dati della periferica vengono sincronizzati con l'interfaccia**. *Il modo più semplice che utilizziamo è l'hanshake in cui CA1 e CA2 serviranno da ready-hack*, ci sono anche altri modi che danno un controllo maggiore al programmatore. Se i bit sono posti ad 11 permette di indicare da programma cosa fare con le linee e cosa fare per il trasferimento dati. **In funzione di quello che scriviamo in CRA3, pilotando il segnale a mano, implementiamo il protocollo che preferiamo**. Un'altra modalità è quella impulsiva che permette di abilitare CA2 a seguire il profilo del segnale impulsivo del clock.

- **CRA3, CRA4, CRA5** individuano uno dei possibili modi di funzionamento della linea CA2:
  - CRA5 stabilisce se la linea CA2 funziona da INGRESSO di interruzione o da USCITA.
  - Se **CRA5= 0** CA2 è programmata per funzionare come linea di ingresso, funziona come CA1 per cui:
    - CRA4 determina il tipo di fronte di interruzione riconoscibile sull' ingresso CA2
    - CRA3 viene utilizzato per mascherare le richieste di interrupt provenienti da CA2

- Se **CRA5= 1** CA2 è programmata per funzionare da USCITA, in questo caso i bit CRA4 e CRA3 sono utilizzati per stabilire uno dei seguenti modi di funzionamento:
  - (100) modo Handshake
  - (101) modo Impulsivo
  - (11x) modo dipendente da CRA3

Osservazione: supponiamo che la linea di interruzione INTA sia collegata con INT3 e poi abbiamo INTB (l'altra freccia) collegata con INT4. Se abbiamo altri dispositivi collegati sulla linea INT3 potremmo avere un conflitto, ma questa linea (tra la freccia A e la verde) è in open drain, ovvero più linee collegate fisicamente sulla stessa linea dovrebbero essere separate logicamente da una OR, ma non serve fisicamente la OR perché per la tecnologia utilizzata è direttamente realizzata. Tramite questa tecnologia anche se le linee di interruzione sono solo 7 possiamo avere più periferiche collegate. Logicamente non ha nessun impatto, è una scelta tecnologica per collegare fisicamente più linee sulla stessa linea. Infatti, nel processore abbiamo tutte porte tristate quando dobbiamo collegare i dati sul bus, quindi anche qui (cerchi blu) avremo porte tristate per abilitare queste scritture. **Nel caso delle interruzioni, sulle linee di controllo non abbiamo nessun tristate per abilitarlo, ma abbiamo un collegamento diretto a patto che la tecnologia implementi un OR logico**.



Esempio: modo handshake (100)

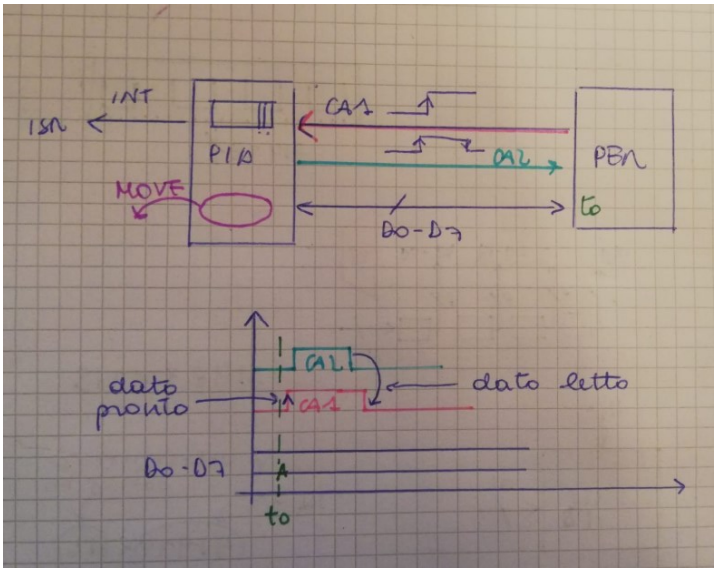
Abbiamo visto come definire il comportamento delle linee di controllo tra interfaccia periferica e la linea stessa, in particolare come pilotare la linea CA2 che può avere comportamenti differenti: se la programmiamo in ingresso si comporta come la linea CA1 e gli altri bit permettono di capire se abilitata e su quale fronte; se la programmiamo per funzionare in uscita possiamo implementare diversi protocolli, tra cui quello dell'handshake, dove *10 indica il protocollo e 0 indica che è in uscita*. ***Quali dati abbiamo tra periferica ed interfaccia?*** CA1 di uscita, CA2 di ingresso ed il bus dati bidirezionale. Supponiamo che la periferica voglia mandare all'interfaccia un dato. Innanzitutto, la prima cosa che fa è **mettere sul bus dati il dato che vuole mandare verso il registro dati della periferica**. Per comunicare alla PIA che il dato è pronto per essere letto dal bus, dovrà **alzare la linea CA1** il che significa "***dato pronto per la lettura***". **Su questa transazione basso-alto, automaticamente si alza la linea CA2**. Quindi, inviamo all'adattatore il segnale di dato pronto, su questa transazione l'adattatore alza CA2, *internamente la PIA, se abbiamo abilitato l'interruput, cosa fa?* **Verrà mandata l'interruzione al processore che causa lo**

La periferica vuole inviare un dato alla PIA sul latoA (settato da Ingresso):
- La periferica mette i dati sul BUS-DATI
- Attraverso la linea CA1 invia alla PIA il segnale "Dato pronto per la lettura", sulla transizione di CA1 anche CA2 si alza;
- Internamente alla PIA il flag IRQA1 (CRA7) si alza e viene inviata al processore una richiesta di interruzione che causerà lo svolgimento di un' apposita routine di interruzione

13

svolgimento di una routine apposita, per esempio permetterà il trasferimento dei dati dall'interfaccia verso il processore. **A trasferimento completato, il registro dati è libero, la periferica abbassa il segnale CA2 che torna al valore basso il che indica alla periferica che l'operazione di lettura è terminata, se vuole può iniziare un altro ciclo**.

Abbiamo a destra la periferica che vuole mandare un dato al processore, in questo istante t0 la periferica mette dei dati sulla PIA, cosa che corrisponde alla freccia D0-D7. Dopo questa transazione alza contemporaneamente il segnale di ingresso, cioè CA1 da periferica a PIA (rosa). Avendo programmato di essere sensibile al fronte di salita negli ultimi due bit del registro, verrà generata un interrupt verso il processore che farà partire la ISR corrispondente. **Mentre facciamo partire il processore, questo contemporaneamente alza anche la linea CA2**. Viene eseguita tutta la ISR che comporterà tra le varie cose il *prelievo del dato da PIA presso un registro interno con una MOVE.* Finita questa operazione, la ISR termina e la periferica controlla il segnale CA2. **Il processore abbassa CA2, come conseguenza di un abbassamento di CA2 la periferica abbasserà CA1**. Il fronte di salita significa dato pronto, *il fronte basso di CA2 fa abbassare automaticamente il CA1 e significa dato letto e quindi si può iniziare un nuovo invio dati*. Questo è quello che succede come protocollo di handshake.



Esempio: uscita dipendente da CRA3

Un altro meccanismo è quello che prevede che il valore del segnale CA2 dipenda dal valore del bit CRA3. Mentre nel caso precedente CA2 era pilotato dal processore perché quando sia alza CA1, CA2 si alza automaticamente e quando termina la ISR si abbassa automaticamente (è il processore che per implementare l'handhsake gestisce CA2), in questo caso è da programma che si può cambiare il valore dei bit mettendo 0 o 1 in funzione se si vuole alzare o abbassare il segnale CA2. L'utilità di questo sta nel fatto che possiamo decidere di implementare il protocollo a scelta perché abbiamo una linea di controllo che pilotiamo e possiamo assegnare un significato.

- Se CRA5 = CRA4 =1
  - all' uscita di CA2 si presenta un livello logico coincidente con quello presente nel bit CRA3

Guardiamo a sinistra i bit 0 e 1 che ***controllano CA1***. Abbiamo la sequenza 00 che indica che l'interruzione è disabilitata; 01 indica che l'interruzione è abilitata sul fronte alto basso (di discesa) e l'interruzione viene inviata quando CRA7 diventa alto. Viceversa, con 11 abbiamo l'abilitazione sul fronte di salita e viene abilitata quando cra7 viene posto ad 1. Per il ***controllo di CA2*** (parte a destra), se CRA5 è 0 significa che è in ingresso e quindi è esattamente la stessa cosa, solo che è CRA6 che avrà questo secondo bit di interruzione, dato che avremo due bit di interruzione associata al porto A che permettono di abilitare due richieste separatamente, mentre la linea è sempre la stessa. Se CRA5 è 1 possiamo avere le diverse modalità di protocollo, se CRA4=CRA3=0 abbiamo l'handhsake per cui si abbassa in seguito ad un'operazione di lettura sul registro dati, si alza quando CRA7 va ad 1 per una variazione di CA1; se CRA4CRA3=01 la modalità impulsiva, mentre CRA4CRA3=10/11 la modalità di uscita dipendente da CRA3. Per CB2 funziona lo stesso ma lavora sul porto B.

CRA1 (CRB1)	CRA0 (CRB0)	Flag Interruzione CRA7 (CRB7)	Richiesta Interr IRQA (IRQB)
0	0	Alto su high/low di CA1 (CB1)	Disabilitata
0	1	Alto su high/low di CA1 (CB1)	Inviata quando CRA7 (CRB7) diventa alto
1	0	Alto su low/high di CA1 (CB1)	Disabilitata
1	1	Alto su low/high di CA1 (CB1)	Inviata quando CRA7 (CRB7) diventa alto

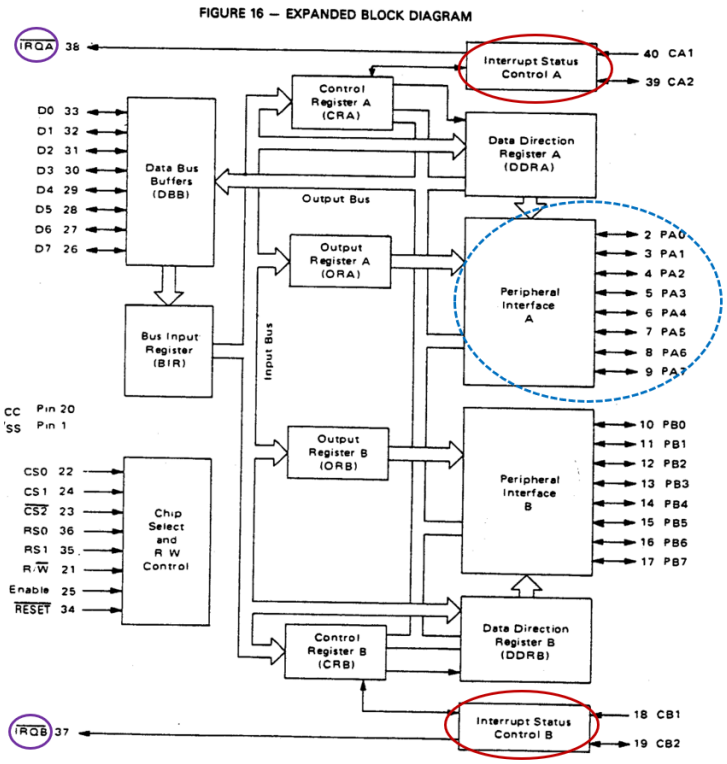
Il flag di interruzione CRA7 (CRB7) torna al valore basso in seguito ad un' operazione di lettura su PRA (PRB).

CRA5 (CRB5)	CRA4 (CRB4)	CRA3 (CRB3)	Flag Interruzione CRA6 (CRB6)	Richiesta Interr IRQA (IRQB)
0	0	0	Alto su high/low di CA2 (CB2)	Disabilitata
0	0	1	Alto su high/low di CA2 (CB2)	Inviata quando CRA6 (CRB6) diventa alto
0	1	0	Alto su low/high di CA2 (CB2)	Disabilitata
0	1	1	Alto su low/high di CA2 (CB2)	Inviata quando CRA6 (CRB6) diventa alto

Il flag di interruzione CRA6 (CRB6) torna al valore basso in seguito ad un' operazione di lettura su PRA (PRB).

Tabella 6 : Controllo della linea di uscita CA2

CRA5	CRA4	CRA3	CA2 BASSO	CA2 ALTO
1	0	0	Basso in seguito ad un' operazione di lettura su PRA	Alto quando CRA7 va ad 1 per una variazione h/l o l/h di CA1
1	0	1	Basso in seguito ad un' operazione di lettura su PRA	Alto al primo colpo di clock successivo alla lettura su PRA
1	1	0	Basso quando CRA3 diviene basso in seguito ad un' operazione di scrittura su CRA	Sempre basso finche CRA3 è basso. Diviene alto seCRA3 va ad 1 per un' op. di scritt. su CRA
1	1	1	Sempre alto finchè CRA3 è alto Diviene basso se CRA3 va a 0 per un' op. di scrittura su CRA	Alto quando CRA3 diviene alto in seguito ad un' operazione di scrittura su CRA



Architettura della PIA

Vediamo il datasheet del componente. Guardiamo il lato periferica a destra. Abbiamo il *registro dati dell'interfaccia A collegato direttamente sul bus dati A* verso la periferica, il registro dati del porto B collegato al bus dati del porto B. Poi abbiamo due blocchi ***controllore dello stato interruzioni***, in alto per A e in basso per B. hanno lato periferica l'accesso ai segnali CA1 CA2 e CB1 e CB2. Questo blocco può generare, in collaborazione con il registro di controllo, direttamente il segnale di interruzione sul porto A e sul porto B (**IRQ**). Abbiamo il registro di controllo, CRA e CRB. Poi c'è un buffer di collegamento con il processore verso i registri D0-d7 e poi abbiamo un blocco di selezione che permette di abilitare sia tutto il chip, sia RS0 e RS1 per i vari registri interni e così via. I *due registri di DRA e DRB sono interni e sono pilotati dal buffer per poter poi decidere se il dato deve essere di ingresso o uscita*.

Per ogni periferica analizzata, realizziamo l'architettura interna che dobbiamo programmare e per la programmazione usiamo il simulatore del Motorola 68k che è ASIM che permette sia di configurare opportunamente un'architettura con determinate caratteristiche sia di abilitare il meccanismo delle interruzioni autovettorizzate e vettorizzate.