

2.1.8 Gestione di un dispositivo di ingresso (o uscita) con la tecnica dell'interrupt

La tecnica di trasferimento dei dati tra CPU e periferiche, esaminata precedentemente, prevede che il processore attenda passivamente, mediante ripetute riletture del bit di stato, che la periferica completi la transazione con il buffer di interfaccia, prima di avere la disponibilità del dato. Questo protocollo di trasmissione delle informazioni, detto di **busy waiting** (attesa passiva) comporta un notevole spreco del tempo di CPU, che, in sistemi di una certa complessità, può essere più utilmente utilizzato per compiere altre elaborazioni.

Per superare questo problema, si arricchisce l'hardware del calcolatore con un meccanismo che concede alla periferica la possibilità di inviare alla CPU un **segnale di interruzione** del programma in esecuzione, detto **segnale di interrupt**, per richiedere, alla CPU, l'esecuzione di una **routine specializzata** adatta a rispondere alle esigenze della periferica che ha interrotto. Questa routine è detta routine di **risposta all'interrupt**. La Fig.2.1.8.1 mostra una semplice architettura hardware che consente questo tipo di gestione di una periferica di input.

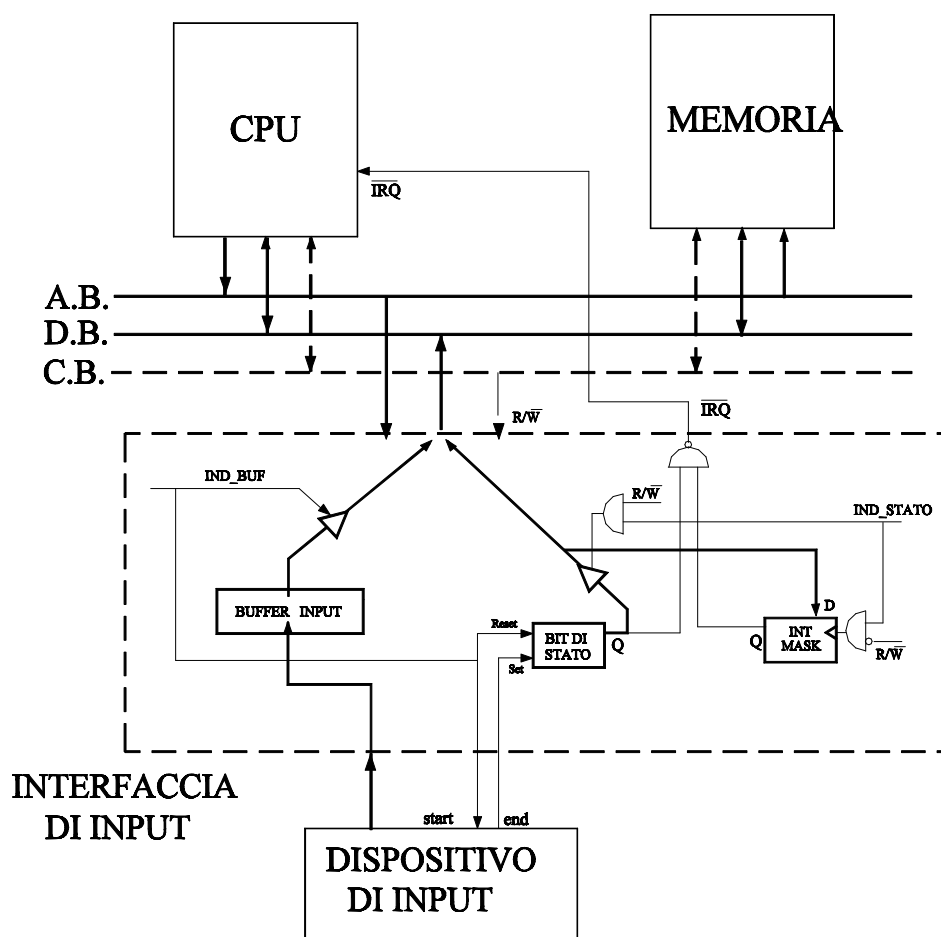


Fig.2.1.8.1 Sistema a microprocessore con semplice hardware per la gestione di periferica di input con la tecnica dell'**interrupt**.

Per una periferica di uscita cambia solo la direzione dei trasferimenti con il buffer di interfaccia (*buffer output* vedi Fig.2.1.5.1).

La logica descritta in questa figura mostra come il dispositivo di ingresso, ponendo a 1 il bit di stato dopo aver posto il dato nel buffer, produce l'attivazione del segnale di uscita **IRQ**, a livello logico zero, se è stato preventivamente posto a 1 il contenuto del flip flop **INT MASK** (Interrupt mask). Scrivendo il valore zero nel flip flop **INT MASK** dell'interfaccia, l'attivazione del bit di stato non provoca alcun segnale d' interrupt e la gestione della periferica può essere fatta ancora con la tecnica del *busy waiting*, descritta nel paragrafo 2.1.4. In questo caso si dice che le interruzioni sono *disabilitate* a livello della periferica.

Il segnale elettrico **IRQ** (*segnale di interrupt*) è un nuovo ingresso alla **CPU** che agisce direttamente sulla sua unità di controllo.

Per consentire la gestione di questo segnale la struttura hardware interna della **CPU**, studiata precedentemente, deve essere modificata nel modo rappresentato in Fig.1.8.1.2 a). Per consentire una adeguata estensione del set di istruzioni si suppone che il codice operativo e l'operando passino, entrambi, dalla lunghezza di cinque bit a quella di otto bit. Ciò non altera il funzionamento e l'architettura descritta precedentemente.

Deve essere tenuto presente il fatto che l'istante in cui arriva il segnale di interrupt è del tutto imprevedibile da parte della **CPU** (*modo asincrono*) che è impegnata ad eseguire alla rigida cadenza del suo veloce clock il programma assegnatole, memorizzato nella **RAM**. Nella Fig.2.1.8.2b) questo programma, individuato dal nome **MAIN**, è memorizzato dall'indirizzo 50 all'indirizzo 80.

Quello che ci si aspetta da questo sistema è che la **CPU**, terminata l'esecuzione dell'istruzione, durante la quale è avvenuta la segnalazione di interrupt, acquisisca il dato prodotto dal dispositivo di ingresso e riprenda l'esecuzione del programma **MAIN**, momentaneamente sospesa, proseguendo correttamente le elaborazioni, *come se l'interruzione non fosse mai avvenuta*.

Per rendere possibile questo funzionamento è necessario rispettare le seguenti regole:

- lasciare che la CPU termini l'istruzione in corso, durante la quale è stato rivelato dall'hardware l'attivazione del segnale di interrupt **IRQ**;
- salvare in una zona riservata di memoria (solitamente lo **stack**) il contenuto del Program Counter e del registro di stato. Il primo contiene l'indirizzo dell'istruzione successiva all'ultima eseguita prima dell'interruzione, che dovrà essere utilizzato successivamente per riprendere il programma interrotto dal punto giusto. Il registro di stato contiene sia i segnalatori aritmetici, che rappresentano i risultati delle elaborazioni effettuate fino al momento dell'interruzione, sia le modalità operative del programma interrotto. Si effettua in questo modo l'operazione di **salvataggio del contesto**.
- porre nel Program Counter l'indirizzo della prima istruzione della routine di risposta del segnale di interruzione, raccolto in memoria in base al numero assegnato alla periferica interrompente (vedi Fig.2.1.8.3);
- eseguire la routine di risposta relativa alla periferica che ha interrotto nella quale deve essere eseguita un'istruzione che annulla il bit di stato che ha provocato l'interrupt, con la quale si comunica alla periferica che l'interrupt è stato regolarmente **servito**. È compito di questa subroutine salvare inizialmente nello stack e ripristinare, al suo termine, i contenuti di tutti i registri che deve utilizzare per le sue elaborazioni per far in modo che il ritorno al programma **MAIN** venga effettuato con lo stesso ambiente elaborativo presente al momento dell'interruzione. Si completa in questo modo il salvataggio del contesto;
- eseguire l'istruzione di ritorno da interrupt, ultima istruzione della routine che ha servito l'interrupt, capace di ripristinare il Program Counter ed il registro di stato con i valori salvati nello stack all'inizio di questo meccanismo (vedi secondo punto).

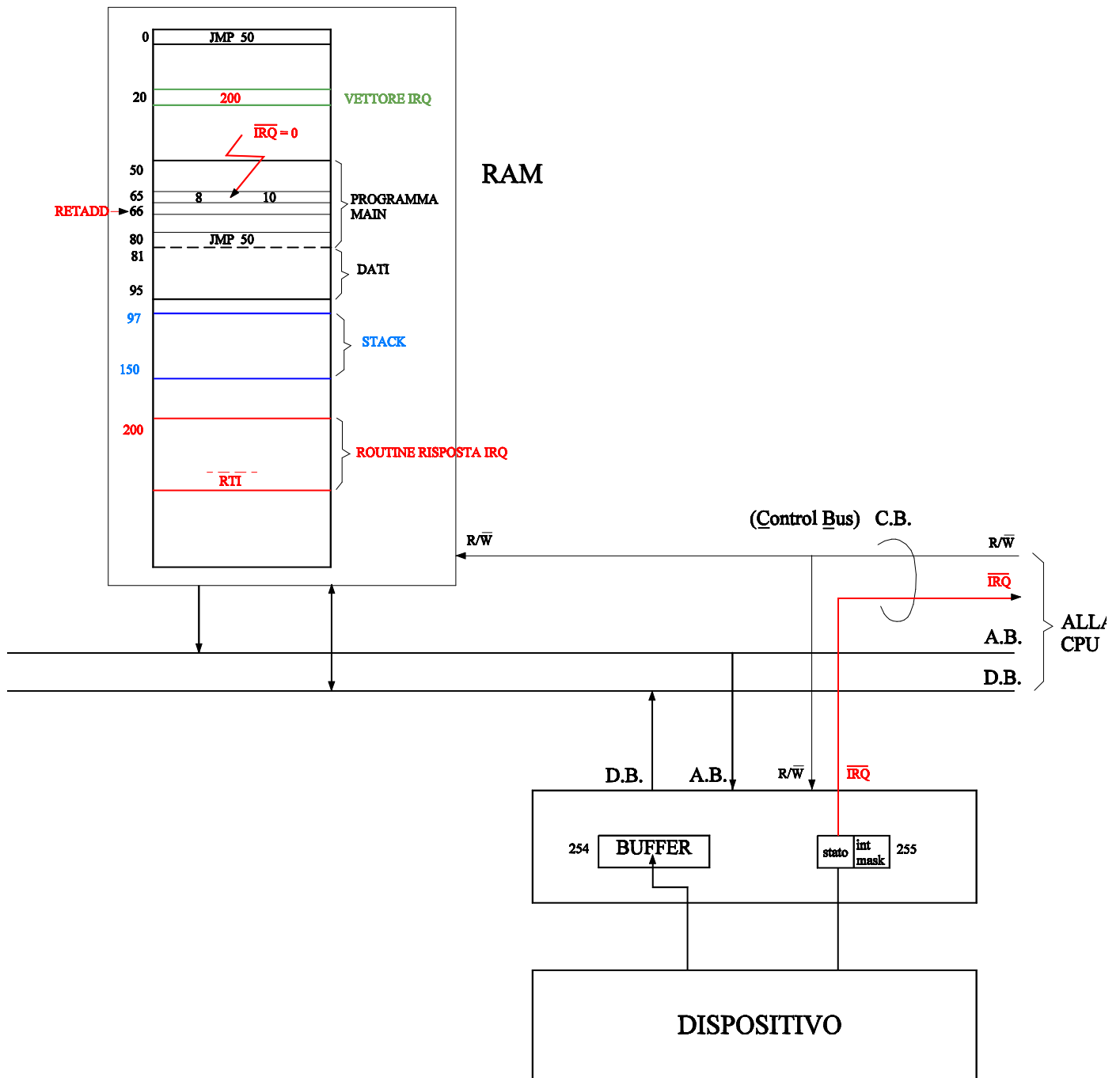


Fig.2.1.8.2 b) Mappa della **RAM**, dell'interfaccia e **Dispositivo** di input connessi alla **CPU** di **Fir.2.1.8.2 a)**

La **Fir.2.1.8.2 a)** fotografa la **CPU** nel momento in cui si verifica la chiamata con interrupt del dispositivo periferico. Si fa l'ipotesi che sia stata raccolta l'istruzione "8 10" all'indirizzo 65 del MAIN e che la stessa sia in fase di esecuzione (**MPC=8**). Si noti che il **PC** contiene il valore 66, indirizzo dell'istruzione successiva, che dovrà essere raccolta al termine di quella attuale.

Il segnale $\overline{\text{IRQ}}=0$ indica che il Dispositivo ha chiamato per interrompere la **CPU** durante l'esecuzione del codice operativo 8. La **CPU** deve terminare regolarmente l'esecuzione di questa istruzione, come di ogni altra istruzione che fosse stata in fase di esecuzione. È predisposto allo scopo il bit di controllo **S0** che assume valore 0 per tutti i microprogrammi di esecuzione di tutti i codici operativi.

Quando l'unità di controllo si appresta alla successiva fase di fetch (**MPC=0**), per raccogliere dalla **RAM** l'istruzione di indirizzo 66, il valore 1 di **S0** ed il valore $\overline{\text{IRQ}}=0$ producono una commutazione nel **MUX** che sostituisce il contenuto del registro **OP CODE** con quello predefinito del registro **microIRQ**. Quest'ultimo registro, di tipo **ROM**, contiene il numero 25 che indirizza la prima microistruzione del microprogramma di risposta all'attivazione del segnale di interrupt. Questo microprogramma si compone di 5 microistruzioni dalla 25 alla 29.

Le microistruzioni 25 e 26:

$\text{SP} - 1 \rightarrow \text{SP}$
 $\text{PC} \rightarrow \text{M}[\text{SP}]$

decrementano di 1 il contenuto del registro **SP** ($150-1=149$) e copiano a questo indirizzo di memoria l'indirizzo 66 (**RETADD**) contenuto del registro **PC**, utile a raccogliere l'istruzione successiva del programma MAIN a quella appena eseguita. In questo modo la raccolta di questa istruzione viene momentaneamente sospesa.

Le successive due microistruzioni 27 e 28:

$\text{SP} - 1 \rightarrow \text{SP}$
 $\text{P} \rightarrow \text{M}[\text{SP}]$

decrementano ancora di 1 il registro **SP** ($149-1=148$) e copiano a questo indirizzo il contenuto del registro **P**. Questo registro contiene i valori dei segnalatori aritmetici determinati dalle elaborazioni del programma MAIN che sta per essere momentaneamente sospeso. Alla ripresa del programma bisognerà ripristinare questi valori per poter continuare le elaborazioni come se l'interruzione non fosse avvenuta.

Queste prime quattro microistruzioni del microprogramma di risposta al segnale di interrupt salvano nella **RAM** il program counter ed il registro di stato a indirizzi decrescenti consecutivi:

148	(N, Z, V, C)	TOP
149	66	
150		BOTTON

La zona di **RAM** nella quale i dati vengono memorizzati a indirizzi decrescenti "uno sopra l'altro" costituisce lo **stack** (pila).

La successiva microistruzione 29:

M[vectADDR] → PC

impone nel registro PC l'indirizzo raccolto dalla RAM usando il contenuto del registro interno **vectADDR**. Quest'ultimo, di tipo ROM, contiene il valore 20, per cui si ha **M[20] → PC**. All'indirizzo 20 della RAM, il programmatore deve porre l'indirizzo iniziale del programma di risposta alla chiamata di interrupt. Nell'esempio di **Fig.2.1.8.2 b)** il valore memorizzato a questo indirizzo è **M[20]=200**.

Concludendo, l'azione svolta dal microprogramma di risposta all'interrupt è quella di produrre un **salto**, dall'istruzione del MAIN colpita dall'interruzione prodotta dal dispositivo periferico, **all'istruzione che comincia all'indirizzo 200** (previo salvataggio nello stack del contenuto della CPU al momento dell'interruzione). Da questo indirizzo a quelli consecutivi il programmatore dovrà memorizzare le istruzioni che vuole vengano eseguite ad ogni interruzione del dispositivo. Queste istruzioni costituiscono la **routine di risposta all'interrupt** prodotto dal dispositivo. In questo esempio la routine di risposta può essere la seguente:

```

IND_BUF      = 254
IND_STATO    = 255
200 MOVE     IND_BUF      ,D0
201 MOVE     D0           ,81
202 RTI

```

Le prime due istruzioni leggono il dato posto dal dispositivo nel buffer dell'interfaccia e lo memorizzano all'indirizzo 81, prima locazione dell'area di RAM destinata a contenere i dati (vedi **Fig.2.1.8.2 b)**). È importante rilevare che la prima delle due, oltre a copiare in **D0** il dato posto nel buffer, annulla il bit di stato dell'interfaccia, disattivando in questo modo il segnale di interrupt.

L'ultima istruzione, che si aggiunge al set di istruzioni previste per questa macchina, ha il codice operativo 35 e non richiede alcun operando. Dopo che è stata raccolta, viene eseguito il microprogramma che è rappresentato nella memoria di controllo di **Fig.2.1.8.2 b)** e che ha 35 come indirizzo iniziale.

La prima microistruzione copia dal TOP dello stack il valore del registro di stato (**M[SP]→P;M[148]→P**) mentre la seconda incrementa il registro **SP** (**SP+1→SP; 148+1→SP**) eliminando dallo stack il valore di **P** appena ripristinato. Le due rimanenti ricopiano nel **PC** il valore di **RETADDR** (**M[SP]→PC; M[149]=66→PC**) e rispettivamente incrementano ancora **SP** che viene riportato al valore che aveva prima dell'interruzione. In pratica l'istruzione **RTI**, svuotando lo stack, produce un salto dalla routine di risposta all'interrupt all'istruzione di indirizzo 66 del MAIN.

La **CPU** continua ad eseguire le istruzioni di questo programma come se l'interruzione non fosse avvenuta, dato che anche i segnalatori aritmetici assumono gli stessi valori presenti in quel momento operativo.

In realtà la routine di risposta all'interrupt ha cambiato anche il valore del registro **D0**, per cui sarebbe stato opportuno che, questa routine, avesse garantito il mantenimento di questo dato al ritorno al MAIN. Il modo utilizzato più di frequente per svolgere questo compito è quello di prevedere una ulteriore coppia di istruzioni capaci da salvare e di ripristinare il valore di **D0** nello stack. La prima

MOVE -(SP), D0

(con codice operativo, ad esempio, 40) è eseguita dal seguente microprogramma:

```
SP - 1 → SP
D0      → M[SP]
```

che pone in cima allo stack il valore di **D0**, previo decremento dello stack pointer. Questa operazione è anche detta di **push** sullo stack.
La seconda

```
MOVE D0, (SP)+
```

(con codice operativo, ad esempio, 41) è eseguita dal seguente microprogramma:

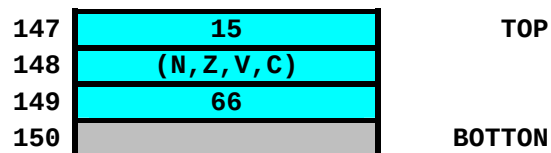
```
M[SP] → D0
SP + 1 → SP
```

che preleva dalla cima dello stack il valore di **D0**, con successivo incremento dello stack pointer. Questa operazione è anche detta di **pop** dallo stack.
Per completare il salvataggio del contesto sullo stack, la routine di risposta all'interrupt deve essere modificata nel seguente modo

```

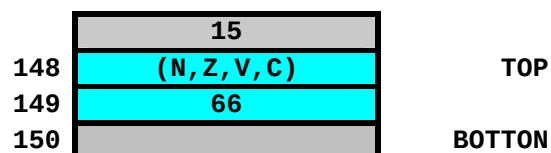
IND_BUF    = 254
IND_STATO  = 255
200 MOVE    D0      , -(SP)
201 MOVE    IND_BUF , D0
202 MOVE    D0      , 97
203 MOVE    (SP)+   , D0
204 RTI
```

Ipotizzando che il registro **D0** contenga al momento dell'interruzione il valore 15, lo stack, dopo l'esecuzione della istruzione di indirizzo 200 assume la seguente configurazione:



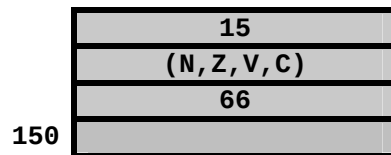
Lo stack occupa tre locazioni.

Dopo l'esecuzione dell'istruzione di indirizzo 203 la configurazione diviene:



Lo stack è ridotto a due locazioni.

Ed infine dopo l'esecuzione dell'istruzione RTI:



Lo stack risulta completamente svuotato.

2.1.9 Gestione di più dispositivi di ingresso e uscita con la tecnica dell'interrupt

Solitamente nei sistemi di calcolatori le periferiche gestite con la tecnica dell'interrupt sono più di una. Quando ciò si verifica, ad ogni periferica che interrompe la CPU deve rispondere con appropriata routine di risposta, diversa da periferica a periferica. Per questo motivo il sistema deve essere attrezzato in modo da **identificare la periferica interrompente** prima di poter eseguire il programma che le compete.

Il sistema più semplice che si può attuare è quello di mantenere la struttura hardware descritta precedentemente e collegare tutte le uscite $\overline{IRQ}$ di ogni dispositivo all'unico ingresso $\overline{IRQ}$ della CPU, nel modo rappresentato in Fig.2.1.9.1, per la gestione di due dispositivi di ingresso e uno di uscita.

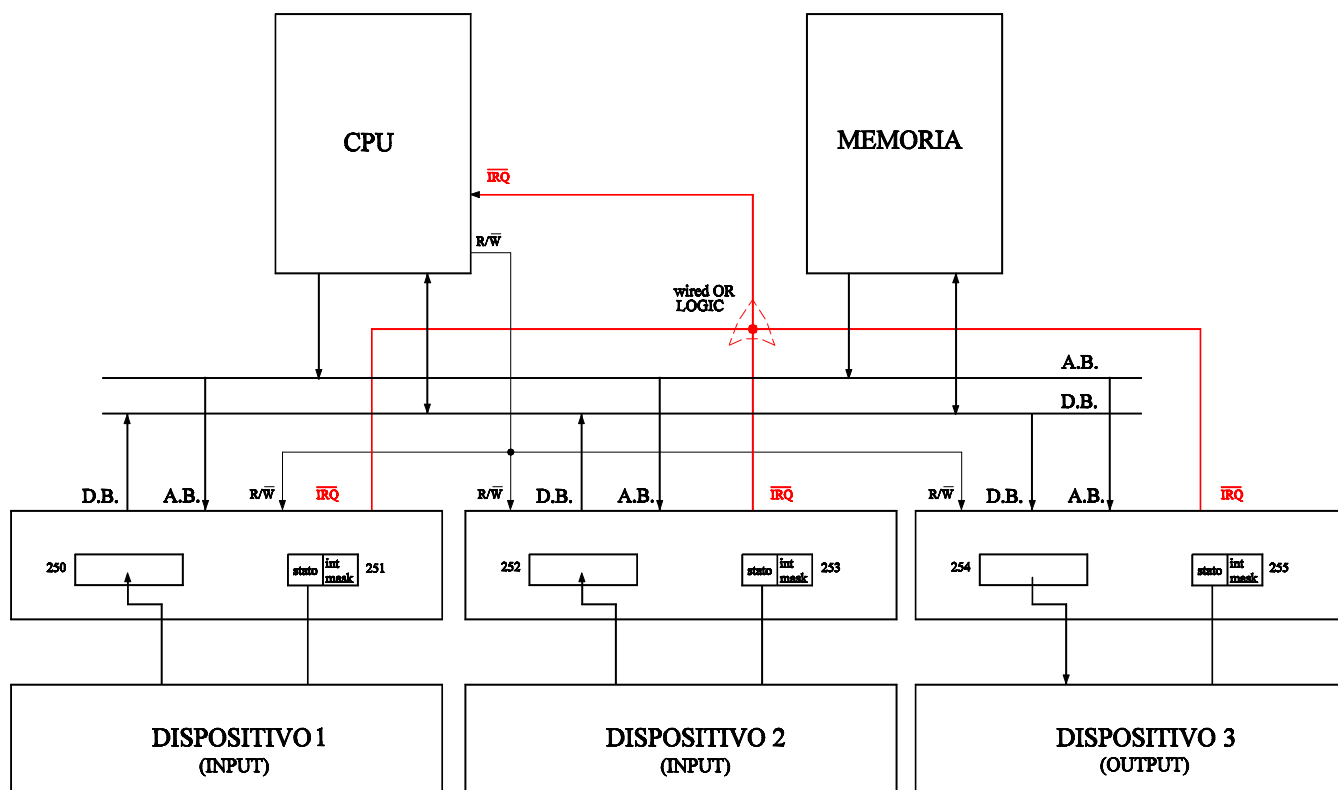


Fig.2.1.9.1. Hardware necessario alla gestione di tre dispositivi di I/O.

La connessione di tutte le uscite $\overline{\text{IRQ}}$ delle interfacce garantisce che se una, o più di una, attiva questo segnale a livello logico zero (collegamento a massa) l'ingresso $\overline{\text{IRQ}}$ della CPU è a livello logico zero (richiesta di interruzione). Viceversa, si può affermare che l'ingresso $\overline{\text{IRQ}}$ della CPU è a livello logico 1 (non attivo) solo se tutte le uscite $\overline{\text{IRQ}}$ delle interfacce sono a livello logico 1 (nessun dispositivo richiede interruzione). Questa descrizione mostra che la connessione in questione si comporta come una porta OR rispetto agli ingressi a valore 0 (l'uscita è 0 se almeno un ingresso è zero). Si parla di **wired OR**.

Dato che la CPU è quella già analizzata, la risposta all'interrupt lancia sempre la routine di indirizzo 200 (nell'ipotesi che nella RAM sia mantenuto $M[20]=200$), qualunque sia la periferica, o le periferiche che richiedono l'interruzione. Deve essere quindi questa routine a identificare la periferica che ha chiamato, esaminando ordinatamente i **bit di stato** nelle interfacce dei dispositivi.

Un esempio di routine di risposta per il sistema in esame è il seguente:

```

IND_BUF1      = 250
IND_STAT01    = 251
IND_BUF2      = 252
IND_STAT02    = 253
IND_BUF3      = 254
IND_STAT03    = 255
200 MOVE      D0          , -(SP)
201 MOVE      IND_STAT01  , D0
202 BNE       209
203 MOVE      IND_STAT02  , D0
204 BNE       213
205 MOVE      IND_STAT03  , D0
206 BNE       217
207 MOVE      (SP)+       , D0
208 RTI
209 MOVE      IND_BUF1    , D0
210 MOVE      D0          , 97
211 MOVE      (SP)+       , D0
212 RTI
213 MOVE      IND_BUF2    , D0
214 MOVE      D0          , 98
215 MOVE      (SP)+       , D0
216 RTI
217 MOVE      99          , D0
218 MOVE      D0          , IND_BUF3
219 MOVE      (SP)+       , D0
220 RTI

```

Le istruzioni dalla 200 alla 208 sono il prologo necessario ad identificare la periferica che ha interrotto. Il Dispositivo 1 è quello "indagato" per primo dalle istruzioni 201 e 202. Per questo motivo, in caso di chiamate contemporanee, il Dispositivo 1 è quello al quale è assegnata la **massima priorità**. Se il suo bit di stato è attivo (valore logico 1) si produce il salto al segmento di programma dalla istruzione 209 alla 212, risposta software a questo dispositivo di input già descritta nel paragrafo precedente (l'istruzione 209 disattiva anche il bit di stato del dispositivo 1). L'istruzione RTI all'indirizzo 212 fa ritornare l'esecuzione al programma interrotto (MAIN), anche se vi sono ancora in sospeso chiamate di interruzione contemporanee prodotte dai dispositivi 2 e/o 3.

Il ritorno al programma principale "lascerebbe", in questa circostanza, attive le uscite **IRQ** dei dispositivi 2 e/o 3 che produrrebbero immediatamente un'altra richiesta di interruzione.

Nel caso il Dispositivo 1 non sia responsabile della chiamata di interruzione (il suo bit di stato è 0), l'istruzione 202 non produce salto e le successioni istruzioni 203 e 204 operano nello stesso modo sul Dispositivo 2.

Se anche questo non è il chiamante, appare evidente che l'interruzione è da attribuire al Dispositivo 3. Per essere garantiti che la chiamata non è dovuta ad un disturbo sulla linea **IRQ** è riservato anche all'ultimo dispositivo, a più **bassa priorità**, lo stesso trattamento di indagine del bit di stato. Le istruzioni 205 e 206 svolgono questa funzione.

Se si arriva all'esecuzione delle istruzioni 207 e 208 significa che l'attivazione di **IRQ** è stata provocata da un disturbo. Si potrebbero prevedere a questo livello istruzioni che segnalano l'errore.

L'uso ripetuto del registro **D0** impone il suo salvataggio preliminare sullo stack, con la prima istruzione di indirizzo 200, ed il suo ripristino, prima di ogni istruzione **RTI**.

Nella stesura di questo programma gli operandi delle istruzioni di salto in avanti di indirizzi 202, 204, 206 sono stati scritti solo alla fine del programma, quando si conoscevano i valori numerici delle istruzioni di salto. In pratica la traduzione di questo programma in linguaggio macchina viene affidata al programma assemblatore, al quale è anche lasciato l'incarico di fare questi calcoli. In questo modo è consentito al programmatore l'uso di nomi simbolici per indicare gli indirizzi delle istruzioni, detti **label**, e marcando con questi nomi solo gli indirizzi ai quali le istruzioni di salto fanno specifico riferimento.

La prima stesura del programma in assembly è riportata di seguito. In questo ultimo listato anche agli indirizzi di memoria, dove porre il dato raccolto dai dispositivi di input, e dal quale prelevare il dato, da inviare al dispositivo di output, rispettivamente i valori 97, 98, 99, sono stati sostituiti con dei nomi appropriati. In questo modo anche gli operandi delle istruzioni hanno dei nomi simboli, inventati ad hoc, che rendono più leggibile il programma stesso.

```

IND_BUF1      = 250
IND_STAT01    = 251
IND_BUF2      = 252
IND_STAT02    = 253
IND_BUF3      = 254
IND_STAT03    = 255
MEM_DISP1     = 97
MEM_DISP2     = 98
MEM_DISP3     = 99
MOVE          D0      , -(SP)
MOVE          IND_STAT01 , D0
BNE           IRQ1
MOVE          IND_STAT02 , D0
BNE           IRQ2
MOVE          IND_STAT03 , D0
BNE           IRQ3
MOVE          (SP)+     , D0
RTI
IRQ1 MOVE      IND_BUF1  , D0
      MOVE      D0      , MEM_DISP1
      MOVE      (SP)+   , D0
      RTI
IRQ2 MOVE      IND_BUF2  , D0
      MOVE      D0      , MEM_DISP2
      MOVE      (SP)+   , D0
      RTI
IRQ3 MOVE      MEM_DISP3 , D0
      MOVE      D0      , IND_BUF3
      MOVE      (SP)+   , D0
      RTI

```