Soluzioni del Capitolo 5

Questo documento contiene le soluzioni ad un numero selezionato di esercizi del Capitolo 5 del libro "Calcolatori Elettronici - Architettura e organizzazione", Mc-Graw Hill 2017.
Sarò grato a coloro che mi segnaleranno errori di qualunque genere. Sarò altrettanto grato a chi mi segnalerà errori nel libro.

Vorrei invitare coloro che avessero sviluppato soluzioni alternative a quelle da me proposte, o soluzioni a esercizi non compresi tra quelli qui trattati, a trasmettermele, in modo da migliorare i contenuti di questo sito.
Inviare le segnalazioni a: `giacomo.bucci@unifi.it`

Aggiornato il 18 aprile 2017

Avvertenza

Per quanto possibile, nello stendere i tratti di codice in linguaggio Assembler 8086 si cerca di evitare gli aspetti relativi alla segmentazione, al fine di evitare le complicazioni che essa comporta.

I numeri che esprimono indirizzi o configurazione di bit, se non indicato diversamente, sono da intendersi in notazione esadecimale.

5.3 L'interfaccia richiesta non differisce molto da quella di Figura 5.8 del testo. Si differenzia solo per il fatto che c'è una esplicita modalità di funzionamento data dal valore della coppia di bit [1,0] del registro CREG. Lo schema dell'interfaccia è in Figura 5.1.

5.4 Poiché a fronte di un bus dati a 8 bit l'interfaccia deve presentare 64 bit verso l'esterno, la cosa più naturale è utilizzare 8 *latch* a 8 bit, secondo lo schema di Figura 5.2. Essendo richiesto che la routine `SETBIT` aggiorni uno solo dei 64 bit di uscita, ogni scrittura sull'interfaccia dovrà avvenire in modo tale che:

a) ogni scrittura avvenga sul latch contenente il bit selezionato;
b) venga modificato il solo bit di interesse lasciando immutati i rimanenti 7.

Per rendere verificata la condizione del punto b) si può procedere in questo modo:

1) leggere lo stato del latch su cui avviene la scrittura;
2) portare a zero il bit nella posizione corrispondente a quella da aggiornare, mantenendo invariati gli altri bit. Questo risultato si ottiene con una semplice operazione di AND con una maschera che ha "0" nella posizione da aggiornare e "1" in tutte le altre;
3) portare il bit di interesse al valore voluto. Questo risultato si ottiene effettuando una semplice operazione di OR.

Resta ora da stabilire come si individua il latch di destinazione e la posizione del bit di interesse entro il medesimo. Il numero d'ordine del latch è dato dal quoziente

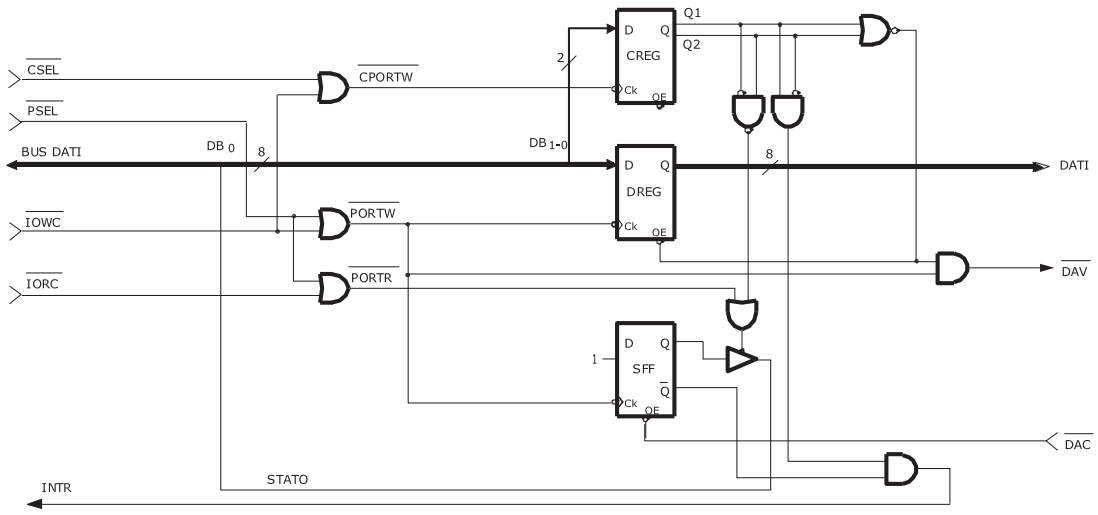

Figura 5.1 Esercizio 5.3. Rispetto all'interfaccia di Figura 5.8 del testo sono state aggiunte le porte che decodificano la modalità di funzionamento dell'interfaccia e che ne condiziono il funzionamento. Lo schema non mostra la decodifica degli indirizzi; si noti che di DREG e SFF sono allo stesso indirizzo, mentre l'indirizzo di CREG è necessariamente diverso.

$Q = x/8$ (parte intera), mentre la posizione entro il latch è data dal resto R della medesima divisione. In termini di programmazione assembler non conviene calcolare Q e R con operazioni di divisione[1]. Conviene piuttosto ragionare come segue. Il numero x si trova sicuramente nei 6 bit meno significativi di CX all'atto della chiamata di `SETBIT`. E' facile verificare che il valore di Q è contenuto nel campo di bit [5..3] di CX, mentre il campo [2..0] contiene R. Il primo si isola scorrendo di 3 bit verso destra, il secondo con un'operazione di AND.

Indicando con `PORT` l'indirizzo di base dell'interfaccia, l'indirizzamento dei singoli latch si ottiene in questo modo[2]:

```
MOV DX, CX      ; DX<- x
SHR DX, 3       ; DX<- parte intera di = x/8
                ; ..ovvero indice del latch indirizzato
ADD DX, PORT    ; DX<- indirizzo del latch di destinazione
```

Conviene assegnare all'indirizzo di base dell'interfaccia (`PORT`) un valore multiplo di 8 in modo che esso venga decodificato attraverso i bit $A_7 - A_3$, mentre la decodifica del contenuto di $A_2 - A_0$ fornisca il selettore dello specifico latch indirizzato. In conclusione si ottiene lo schema di Figura 5.3.

Resta infine da stabilire come si effettua la sequenza di operazioni dei precedenti punti 1), 2) e 3). Per quanto si riferisce alla determinazione dello stato del latch ci sono

[1] Il lettore verifichi questa affermazione esaminando il repertorio di istruzioni di un processore di sua scelta, analizzando come si effettua la divisione tra interi e che risultati determina.

[2] La porta viene indirizzata tramite il contenuto di un registro. Avendo scelto di riferirci, almeno come sintassi assemler dell'8086, useremo il registro DX, che può svolgere la funzione di contenitore di indirizzo di I/O.

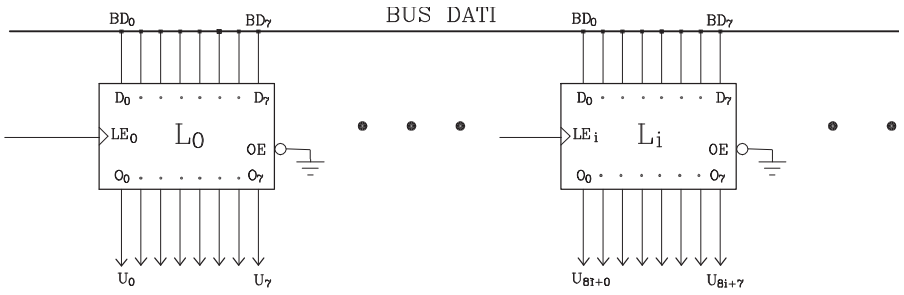

Figura 5.2 Esercizio 5.4. Struttura dell'interfaccia con 8 latch da 8 bit. Il generico latch viene indicato con L_i ($i = 0, 1, ..7$) mentre LE_i rappresenta il relativo segnale di *Latch Enable* che determina la memorizzazione su Li degli 8 bit in ingresso. L'uscita del singolo latch è sempre abilitata. Il latch i-mo fornisce le 8 uscite da U_{8i+0} a U_{8i+7}.

due alternative. La prima consiste nel fare tutto per via software, ovvero prevedere un numero adeguato di variabili (di stato) interne alla routine `SETBIT` che tenga traccia dello stato dei latch; la seconda consiste nel prevedere sull'interfaccia la logica che permette si leggere lo stato dei latch stessi. Ovviamente la seconda soluzione è la più "costosa" in quanto richiede della logica aggiuntiva.

Incidentalmente osserviamo che nella progettazione si presenta non di rado la necessità di decidere, per alcune delle funzionalità che un sistema deve svolgere, la loro ripartizione tra hardware e software. Il criterio generale è quello di cercare il miglior rapporto costo/prestazioni, che, di solito, si traduce nel fare eseguire al software quanto più possibile. Nel caso specifico si tratta di tenere traccia in memoria dello stato dei latch, anziché prevedere la loro lettura. In generale, le soluzioni in software hanno anche il pregio della flessibilità, mentre, al contrario, le soluzioni hardware hanno lo svantaggio della rigidità. L'allocazione di funzionalità a componenti hardware ha senso solo quando il loro svolgimento software è sconsigliato per motivi di assoluta inefficienza o perché la complessità che si introduce nel software è tale da farle preferire l'aggiunta di una parte hardware specializzata.

In base ai precedenti ragionamenti occorre prevedere un vettore di 8 byte per tener

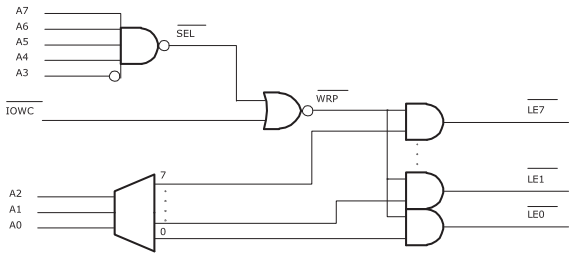

Figura 5.3 Esercizio 5.4. Schema di decodifica degli indirizzi. Per semplificare, si suppone che gli indirizzi delle porte di I/O siano su 8 bit (anche se trasmessi attraverso DX). Si è fatta l'ipotesi che l'interfaccia sia all'indirizzo di base F0.

traccia dello stato degli 8 latch. Chiameremo tale vettore `COPIA`. Il tratto di codice commentato che segue illustra come si effettua la sequenza di operazioni corrispondenti ai punti 1), 2) e 3) e gli altri dettagli di programmazione di `SETBIT`.
Il vettore `COPIA` sarà cosi dichiarato[3].

```
PORT    EQU        FOH         ; Indirizzo della porta
COPIA   DB         8 DUP (0)   ; 8 byte di copia inizializ. a ``0''
```

La routine `SETBIT` assume questo aspetto.

```
SETBIT:
        MOV DX, CX          ; DX<- x
        SHR DX, 3           ; DX<- indice del latch indirizzato
        ADD DX, PORT        ; DX<- indirizzo del latch di destinazione
 ;
 ;Costruzione dell'indice in COPIA (l'indice è uguale a numero d'ordine del
 ;latch)
 ;
        MOV     BX, CX          ;
        SHR     BX, 3           ; BX<-  indice del latch
        AND     CL, 111B        ;CL<- R    (posizione entro il latch)
        SHL     AL, CL          ;Posizionamento del bit da scrivere
 ;
        MOV     AH, 1           ;Preparazione
        SHL     AH, CL          ; della maschera di "1" eccetto
        XOR     AH, OFFH        ;  lo "0" nella posizione di interesse
        AND     AH, COPIA[BX]   ;AH<- copia latch con "0" in posizione
 ;
        OR      AL, AH          ;Aggiunta del bit dato
        MOV     COPIA[BX], AL   ;Aggiornamento della copia
        OUT     [DX], AL        ;Scrittura sulla porta
 ;
        RET
```

Una variazione sul tema. Si provi a progettare l'interfaccia che opera un modo duale rispetto a quella vista, nel senso che essa deve leggere un bit selezionato da un insieme di 64 ingressi digitali. La relativa routine di gestione `GETBIT` viene chiamata esattamente come `SETBIT`, dove ora x rappresenta il numero d'ordine del bit da leggere; la routine `GETBIT` restituisce nel bit 0 di AL il valore dell'ingresso digitale letto.

Si verifichi che, in linea di principio, questa interfaccia si riduce a un "grosso" selettore (*multiplexer*) da 64 in 1. Se si pensa di realizzare l'interfaccia con componenti discreti non programmabili, la funzionalità di un tale selettore deve essere costruita a partire, per esempio, da 8 selettori 8 in 1. Si può tuttavia scoprire che la soluzione più conveniente consiste invece nel prevedere 8 semplici buffer da 8 bit, lasciando al programma il compito di isolare il bit di interesse tra gli otto letti.

5.5 Se l'Interrupt Handler deve poter gestire anche interruzioni che sopravvengono, lo schema del libro deve essere modificato in modo che, servita ogni richiesta, ISR venga letto nuovamente. Ciò può comportare che lo stesso periferico può essere servito più volte

[3]Si osservi che nell'assembler 8086 i dati andrebbero dichiarati nel segmento dei dati e il codice nel segmento di codice. Per non annoiare il lettore evitiamo questi dettagli, rimandando all'appendice sul linguaggio assembler.

se fa a tempo a riasserire la propria richiesta prima dell'uscita da IH. L'uscita ci sarà solo quando tutti i bit di ISR saranno a zero, ovvero quando tutte le richieste saranno state esaudite. Lo schema di IH si modifica in questo modo

```
IH:       <Salvataggio del contesto>
IN_ISR    IN      AL,ISR            ; Lettura di ISR
          AND     AL,FFh            ; per la condizione di test
          JZ      ESCI              ; si esce se tutto è a 0
          AND     AL,1              ; Isolamento Bit 0
          JZ      TESTB_1           ; se 0 avanti col prossimo
          CALL    RSI_0             ; AL[0]=1 : servizio periferico 0
          JMP     IN_ISR            ; torna a leggere ISR
TESTB_1:  MOV     AL,COPIA
          AND     AL,2              ; Isolamento Bit 1
          JZ      TESTB_2           ; se 0 avanti col prossimo
          CALL    RSI_1             ; AL[1]=1 : servizio periferico 1
          JMP     IN_ISR            ; torna a leggere ISR
          :: Come sopra
          :::::
ESCI:     <Ripristino del contesto>
          IRET
```

Si noti che se a causa di un malfunzionamento all'entrata in IH ISR fosse tutto a zero, ovvero l'interruzione generata da un disturbo, questo IH farebbe ritornare immediatamente al programma interrotto, mentre lo IH del testo esaminerebbe tutti i bit prima di concludere.

5.6 Con il nostro schema di riferimento (Figura 5.1) ogni interfaccia è costruita per funzionare autonomamente. La richiesta di interruzione richiede una sola linea sul bus, come da schema di Figura 5.9. Se si vuole costruire un registro come ISR è necessario prelevare da ogni interfaccia il bit di stato e portarlo a un buffer da selezionare con un esplicito comando di lettura. Ma come? Bisognerebbe:

- predisporre per ogni interfaccia una linea dedicata di bus, cui collegare l'uscita del FF che rappresenta il bit di stato;
- portare le corrispondenti linee di bus in ingresso a una interfaccia aggiuntiva (ovvero al buffer usato per creare ISR su questa interfaccia aggiuntiva);
- aggiungere su questa interfaccia la la logica di decodifica dell'indirizzo.

Tutto sommato, se non ci sono particolari esigenze di rapidità di risposta, meglio la soluzione in polling (ovvero interrogare direttamente le singole interfacce)!

Si deve aggiungere che con le tecniche di integrazione tipo ASIC o FPGA, con le quali le interfacce vengono integrate sullo stesso chip, la costituzione di un ISR potrebbe essere tenuta in considerazione.

5.8 Per il buon funzionamento del driver si presuppone che la sezione di inizializzazione venga chiamata solo e sempre quando non è in corso un trasferimento, cosa di cui il programmatore può accertarsi testando lo stato della variabile `BUSY`.

Se assumiamo che il driver venga chiamato mentre è in corso un trasferimento e facciamo l'ipotesi che durante l'esecuzione del tratto di codice della sezione di inizializzazione non si manifesti l'interruzione, l'effetto della routine di inizializzazione è riconfermare `BUSY` asserita ma anche quello di aggiornare la variabile che tiene traccia dell'indirizzo

del prossimo byte da trasmettere e della variabile di conteggio; inoltre viene trasmesso un carattere al dispositivo esterno.

La trasmissione di questo carattere può determinare un effetto impredicibile, nel senso che esso dipende dalla natura del periferico e da quanto è progredito il trattamento del carattere precedente. In ogni caso, al manifestarsi dell'interruzione, essendo cambiati l'indirizzo di prelievo e il contatore, la trasmissione dei caratteri verso l'esterno procederà sulla base dei nuovi valori assunti dalle rispettive variabili. Se, per esempio, si ipotizza di avere a che fare con una stampante, l'effetto macroscopico sarebbe l'abbandono della stampa in corso e il passaggio, senza soluzione di continuità, alla stampa del contenuto dell'area di memoria specificata con la chiamata al driver.

L'interruzione può anche pervenire mentre è in corso di esecuzione proprio la sezione di inizializzazione. Si lascia al lettore l'analisi dei possibili casi che si possono determinare a seconda del punto in cui si manifesta l'interruzione.

Per quanto si riferisce alla possibilità che il driver rifiuti la chiamata se già occupato, basta che all'ingresso venga effettuato il test dello stato della variabile `BUSY` e che, in caso sia diversa da zero, il controllo ritorni senza procedere col trasferimento, riportando, per esempio, in AL un valore convenzionale (per esempio -1) a indicare che la chiamata non ha avuto effetto.

5.9 Sull'interfaccia è necessario un contatore modulo 4, che venga caricato con il valore contenuto in AL all'atto dell'operazione di `OUT` e il cui stato fornisca il selettore del byte da leggere. Inoltre ad ogni lettura il contatore deve avanzare. Queste considerazioni portano allo schema di Figura 5.4. Il contatore modulo 4, indicato come C/4, ha una coppia di

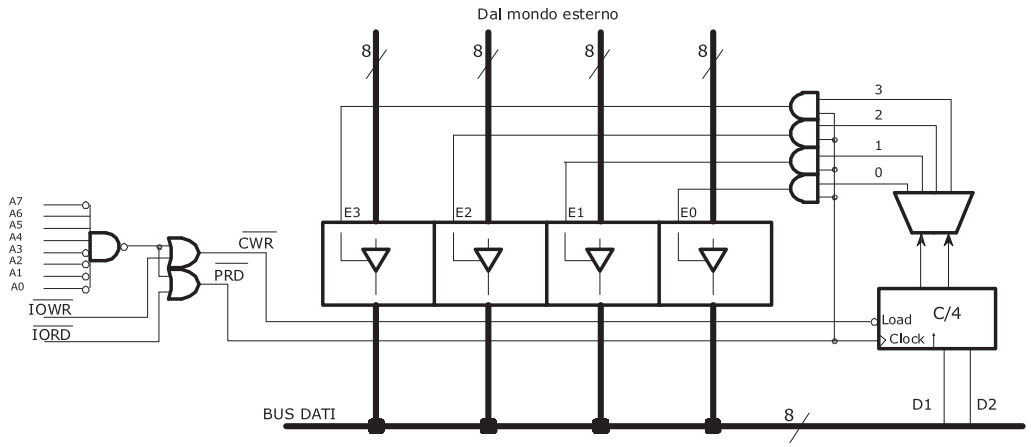

Figura 5.4 Schema dell'interfaccia per l'Esercizio 5.9.

ingressi che servono a precaricare lo stato di partenza. Il caricamento ha effetto quando viene asserito l'ingresso "Load", ovvero quando la CPU effettua una scrittura all'indirizzo dell'interfaccia (70). Le letture determinano l'abilitazione dell'uscita del buffer selezionato attraverso il contatore. Sul fronte di salita del segnale $\overline{\text{PRD}}$ il contatore avanza.

5.11 L'esercizio non presenta particolari difficoltà dal punto di vista della logica dell'interfaccia. Uno schema possibile è in Figura 5.5.

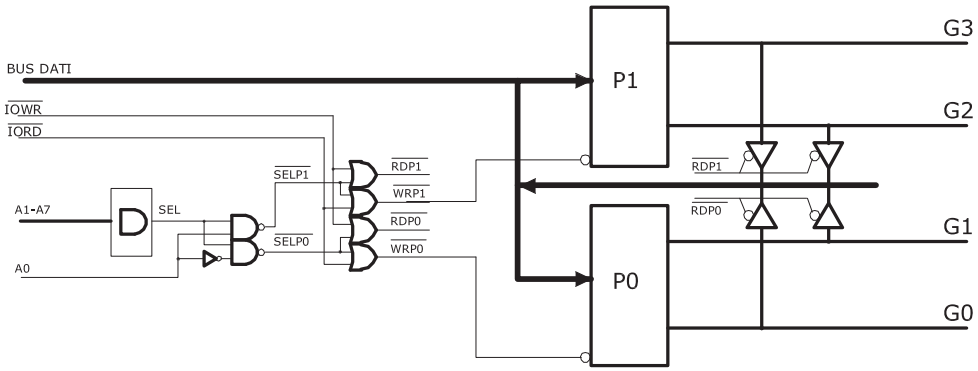

Figura 5.5 Schema dell'interfaccia per l'Esercizio 5.11.

Più interessante è invece la ricerca di un algoritmo per il `DRIVER`. Quello che segue opera in questo modo:

a) viene prima individuata la porta, ovvero la coppia di gruppi, su di cui deve avvenire la scrittura;
b) si stabilisce se i 4 bit da modificare devono andare nella parte alta o bassa della porta;
c) in base alla risultanza del punto b) si forma il byte da trasmettere.

Ricordiamo che in entrata al driver il registro AH contiene il numero di gruppo mentre AL contiene, posizionato a destra, il dato da presentare sul gruppo. Il driver può essere scritto nel modo seguente.

```
DRIVER:     MOV     DX,0F0H     ;Base dell'interfaccia
            MOV     CL,AH       ;AH = numero del gruppo
            AND     CL,010B     ;Si isola il bit 1 del selettore di gruppo
            JZ      AVANTI      ;Se "0" Sono stati scelti G1-G0
            INC     DX          ;Sono stati scelti G3-G2: adeguamento DX
AVANTI:     IN      BL,[DX]     ;Lettura della porta
            AND     AH,01B      ;Si isola il bit 0 del selettore di gruppo
            JZ      PARI        ;Vale "0" per G0 e G2
;
;Il controllo arriva alla parte che segue se si tratta di trasmettere
;i 4 bit contenuti in AL sulla parte alta (G1 o G3) della porta .
;
            SHL     AL,4        ;Posizionamento nuovo dato da trasmettere
            AND     BL,0FH      ;Isolamento parte invariata (di G0 o G2)
            OR      AL,BL       ;Formazione byte
EXIT:       OUT     [DX],AL     ; e sua trasmissione
            RET
;
;Il controllo arriva alla parte che segue se i 4 bit contenuti in AL
;devono essere trasmessi nella parte bassa (G0 o G2) della porta.
;
```

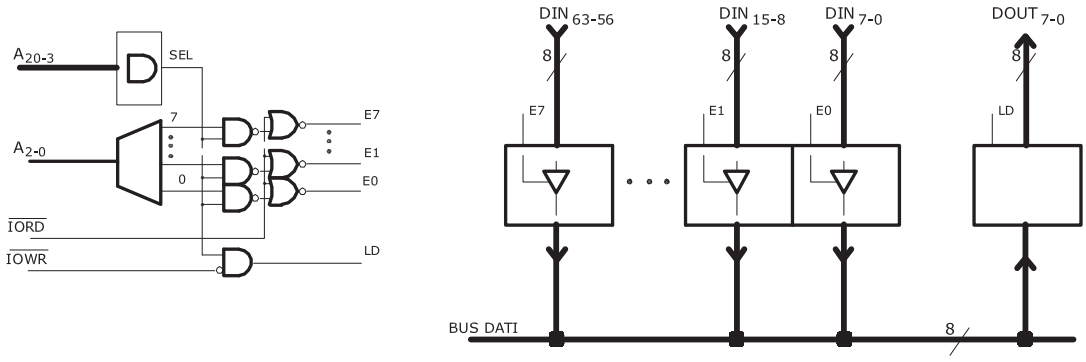

Figura 5.6 Schema dell'interfaccia per l'Esercizio 5.12. Il blocco di destra è un latch le cui uscite sono tenute sempre abilitate. Poiché in lettura viene usato il registro DX, si è ipotizzato di decodificare 20 bit di indirizzo.

```
PARI:       AND     BL,0F0H     ;Isolamento parte invariata (di G1 o G3)
            OR      AL,BL       ;Formazione byte
            JMP     EXIT
```

Si noti che l'algoritmo precedente si basa in modo rilevante sul fatto che le porte sono 2. Il lettore provi a generalizzare il problema, assumendo un numero qualsivoglia di porte.

5.12 Questo esercizio, almeno per quanto riguarda la logica, ha molte similitudini con l'esercizio 5.4. La routine LOOKUP risulta invece di una certa complessità. Circa la modalità di enumerare i bit si faccia riferimento alla soluzione dell'Esercizio 5.4.

Saltiamo la stesura del diagramma di flusso, che lasciamo come compito non risolto al lettore, e passiamo direttamente a costruire la routine INITLKUP. Essa deve solo leggere ordinatamente le porte di ingresso memorizzare i valori letti in un vettore di appoggio, di 8 posizioni, che chiameremo `COPIA`.

```
INITLKUP:   MOV     CX,8            ;CX: contatore
            MOV     DI,0            ;DI: indice
            MOV     DX,PORT         ;DX: indirizzo base dell'interfaccia
;
READ0:      IN      AL,[DX]         ;Lettura
            MOV     COPIA[DI],AL    ; memorizzazione
            INC     DX              ;Puntamento alla porta successiva
            INC     DI              ; e incremento indice
            LOOP    READ0           ;equivale a decremento di CX e JNZ
            MOV     DX,PORT         ;DX: indirizzo porta di uscita
            MOV     AL,0FFH         ;Inizializzata dicendo:
            OUT     [DX],AL         ; "non ci sono cambiamenti"
            RET
```

Più difficile è scrivere LOOKUP. Al fine di evitare di costruire un algoritmo troppo intricato, conviene dividere LOOKUP in due parti: la prima dedicata a leggere gli ingressi, aggiornare il vettore `COPIA` e costruire il vettore `CAMBIATI`; la seconda a determinare il numero d'ordine richiesto.

Anzitutto saranno definiti questi dati[4]:

```
        ...........
COPIA       DB      8 DUP (?)       ;64 bit copia ultima lettura
CAMBIATI    DB      8 DUP (?)       ;64 bit per tener traccia dei cambiati
        ...........
```

La prima parte di LOOKUP corrisponde al seguente codice:

```
LOOKUP:     MOV     CX,8                ;CX: Contatore
            MOV     DI,0                ;DI: indice
            MOV     DX,PORT             ;DX<- indirizzo base dell'interf.
READ:       IN      AL,[DX]             ;Lettura
            XOR     AL,COPIA[DI]        ;Calcola i cambiamenti
            MOV     CAMBIATI[DI],AL     ; e li memorizza in CAMBIATI
            XOR     AL,COPIA[DI]        ;Ripristina in AL la lettura
            MOV     COPIA[DI],AL        ; e memorizza in COPIA
            INC     DX                  ;Puntamento alla porta successiva
            INC     DI                  ; e incremento indice
            LOOP    READ                ;
;Al termine del ciclo COPIA contiene l'ultima lettura dei 64 ingressi
        ....segue.......
```

La seconda parte di LOOKUP ha lo scopo di determinare il numero d'ordine del più basso ingresso modificato. Si tratta di effettuare la scansione del vettore `CAMBIATI`, arrestandosi al riconoscimento del primo bit a "1". A scopo didattico, nello stendere questa parte, anziché nominare direttamente il vettore `CAMBIATI`, lo si indirizza attraverso BX in modo da avere pronto il puntatore all'uscita di `LOOKUP`.

```
        ....segue.......
            MOV     CX,8                ;Contatore
            MOV     DI,0                ;Indice
            MOV     BX,OFFSET CAMBIATI  ;Scostamento di CAMBIATI
            MOV     AH,0                ;AH dirà quale byte
TESTBYTE:   MOV     AL,[BX][DI]         ;Salto a THISBYTE se....
            SUB     AL,0                ;
            JNZ     THISBYTE            ;....c'è almeno una variazione
            INC     AH                  ;Forse nel prossimo byte
            INC     DI
            LOOP    TESTBYTE
;
;Se il controllo arriva qui non c'è stato alcun cambiamento
;
            MOV     AH,0FFH             ;Segnala nessun cambiamento
EXIT:       OUT     PORT,AH
            RET
;
THISBYTE:
;Il numero d'ordine del primo byte che presenta un cambiamento è in AH
;lo si moltiplica per 8
;
            SHL     AH,3                ;La parte alta del numero d'ordine
;
```

[4]Nel segmento DATA se si tratta effettivamante di assembler 8086.

```
;Resta da ora trovare entro questo byte il bit variato di ordine minore
;
           MOV     CX,8
           MOV     DL,0                ;DL: posiz. del bit entro il byte
TESTBIT:   MOV     DH,AL               ;DH copia di appoggio
           AND     AL,1                ;Isola il bit in posizione 0
           JNZ     THISBIT             ;Salto se cambiato
           INC     DL                  ;Conteggia
           MOV     AL,DH               ;Si riprende il byte
           SHR     AL,DL               ;Avanti col prossimo bit
           LOOP    TESTBIT
;
;Se il controllo arriva qui c'è un errore: un byte indicava cambiamento
;ma nessun bit sembra modificato. Un buon programma dovrebbe prevedere
;una qualche azione di recupero. Si "finge" di saltare a un analizzatore
;di disastro
;
           JMP     DISASTRO
;
;Il numero d'ordine del bit è in DL
;
THISBIT:   OR      AH,DL               ;Unione parte alta e bassa
           JMP     EXIT
```

La scrittura di LOOKUP è conclusa. La routine è stata divisa in due parti. E' evidente che il numero da presentare sulla porta di uscita poteva essere determinato anche all'interno del ciclo di lettura delle porte di ingresso. Il lettore è invitato a riscrivere LOOKUP in modo che operi in tal senso.

Ulteriori miglioramenti, sia estetici che di efficienza, possono essere apportati evitando i doppi contatori (CX e AH, oppure CX e DL) e utilizzando istruzioni che effettuano direttamente il test dei bit.

5.15 Poiché in presenza di più interruzioni il controllore programmabile che stiamo progettando deve presentare il VT relativo a quella più prioritaria, si rende necessario un codificatore di priorità (*priority encoder*, PE). PE è rappresentato come un blocco a 8 ingressi e 3 uscite, sulle quali è codificato il numero d'ordine dell'ingresso più prioritario asserito[5]; l'uscita di PE, abilitata dal segnale INTA, viene usata per formare il *vector type* e piloterà i 3 bit meno significativi del bus dati. Da ciò deriva uno schema del nostro controllore di interruzione come quello di Figura 5.7.

Le 3 righe di codice seguente predispongono la base dei vettori all'indirizzo `C0`. Si noti che in questo caso, essendo la base minore di 256 si è caricato il solo registro AL; se la base fosse stata compresa tra 256 e 1024 occorreva caricare il registro AX. In ogni caso la divisione per 4 (l'istruzione `SHR`) riporta entro il byte e quindi l'istruzione `OUT` resterebbe invariata.

```
; TPORT è l'indirizzo del registro T
;
            MOV     AL, 0C0H        ;Base dei vett. di interr.
            SHR     AL,2            ; divisa per 4
```

[5]Il lettore è invitato esaminare i dati di catalogo dei prodotti commerciali, apportando allo schema eventuali modifiche dettate dal loro reale modo di funzionare.

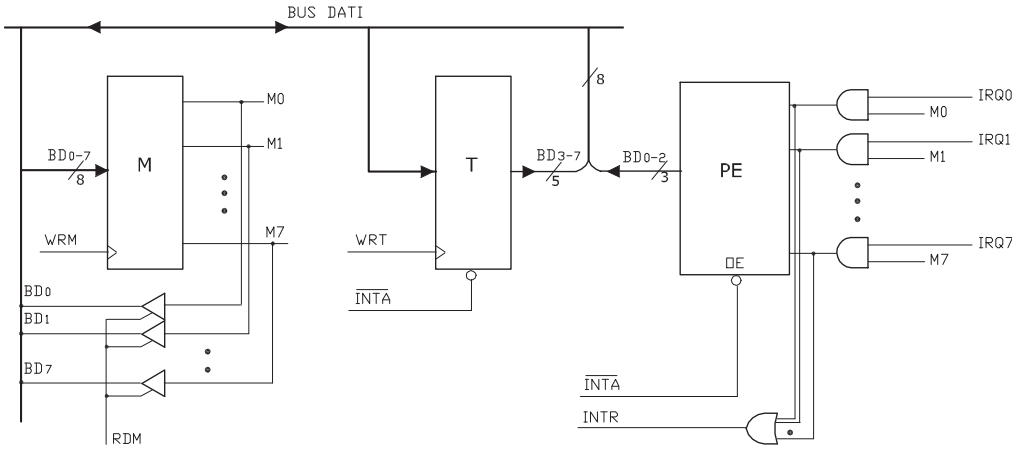

Figura 5.7 Logica per l'Esercizio 5.15. Il registro di mascheramento M, come il registro T (per il VT) sono programmabili. Il contenuto del registro M è leggibile direttamente da programma. Non è stato tracciata la logica che genera i comandi RDM, WRM e WRT. Si deve assumere che al dispositivo corrispondano almeno due indirizzi, relativi al registro M e al registro T.

```
            OUT     TPORT,AL        ;
```

Il seguente tratto di codice porta a 1 il bit 5 di M, lasciando immodificati gli altri bit.

```
;
; MPORT è l'indirizzo del registro M
;
            IN      AL,MPORT
            OR     AL,00100000B    ; Bit 5 a 1; gli altri invariati
            MOV     MPORT,AL
```

5.16 Tralasciamo il disegno dell'interfaccia che richiede un semplice latch e dedichiamoci al testo di INTPW

Per tenere traccia del fatto che un periodo corrisponde a 10 interruzioni useremo il contatore modulo 10 `CONT10`. Il suo contenuto va da 0 a 9, incrementandosi ad ogni interruzione.
Per tenere traccia delle interruzioni che devono dare uscita alta useremo il contatore `CONTH`.
Infine, useremo la variabile (logica) `A_B`, per tener traccia se si è nella parte in cui l'uscita va tenuta a "1" o in quella in cui deve essere "0". INTPW prende la forma che segue.

```
INTPW:      PUSH    AX              ; salvataggio
            PUSH    CX              ; registri

            MOV     CH,CONT10       ;A che punto è il
            DEC     CH              ; contatore?
            JZ      NUOVOPER        ;  se 0 è la fine di un periodo
```

```
;
;Non è l'inizio di un periodo: c'è da decidere come pilotare
;il bit 0 di PORT
;
            MOV     CONT10,CH       ;Aggiornamento contatore in memoria
            MOV     AH,A_B
            JZ      LOW             ;Se 0 siamo nella seconda fase
            MOV     CL,CONTH        ;Siamo nella prima fase
            DEC     CL              ;
            JZ      ABBASSA         ;Se 0 è la fine della prima fase
            MOV     AL,1            ;Trasmette 1 su B0
EXIT:       OUT     PORT,AL
            MOV     CONTH,CL        ;Aggiornamento CONTH

            POP     CX              ;Ripristino
            POP     AX              ; registri salvati
            IRET
;
ABBASSA:    XOR     AH,AH           ;Si passa alla seconda fase
            MOV     A_B,AH          ;A_B<- 0
LOW:        MOV     AL,0            ;Trasmette 0 su B0
            JMP     EXIT
;
NUOVOPER:   MOV     CH,10           ;Inizializzazione contatore
            MOV     CONT10,CH       ; di periodo
            MOV     AH,1            ;Inizializzazione variabile
            MOV     A_B,AH          ; di stato
            MOV     CL,N            ;CL<- quante volte alto
            MOV     AL,1            ;Trasmette 1 su B0
            JMP     EXIT
```

5.18 La rete richiede:

a) un codificatore a 8 ingressi (ai quali vengono portati IR7-IR0) e 3 uscite (sulle quali appare codificato il numero d'ordine della richiesta di interruzione più prioritaria asserita);
b) un decodificatore 1/8, al quale si portano in ingresso le 3 uscite del codificatore.

Le 8 uscite del decodificatore vengono portate ordinatamente agli 8 ingressi Set dei Flip-Flop ISR, ai cui ingressi di clock è collegato INTA, in modo che venga portato a 1 il solo ISR corrispondente all'interruzione identificata dal codificatore. Le 3 linee di uscita di quest'ultimo vengono impiegate per comporre il selettore di interruzione.

5.19 Il possibile malfunzionamento è questo. Si supponga che sia in corso un ciclo di INTA, in risposta a una richiesta di interruzione effettuata da un periferico meno prioritario di quello in questione. In tale ciclo la linea INTAO risulta asserita. Se nel corso del ciclo di INTA viene ora asserita la richiesta di interruzione IRQ, la linea INTAO passa allo stato "0". Ovviamente il tutto avviene in modo completamente casuale, in dipendenza dalla tempificazione dei segnali. É possibile che INTAO sia rimasta asserita quanto basta a selezionare l'interfaccia a valle. É però possibile che IRQ sopravvenga immediatamente

dopo il fronte di INTA. In tal caso, mentre non è detto quel che può accadere a valle, l'interfaccia in questione non è in grado di selezionarsi in quanto SFF non ha commutato sul fronte di INTA.

5.20 Se SFF passa a "0" con il sistema di interruzione abilitato e a valle è asserita una IRQ, questa determina immediatamente un'interruzione (sempre che IEI in ingresso alla relativa interfaccia sia a "1" e non sia disasserita per effetto di qualche interfaccia intermedia) che ha l'effetto di interrompere la routine in corso. In altre parole, una routine meno prioritaria interrompe una più prioritaria. Se si vuole evitare questa "inversione della priorità" occorre che la riabilitazione del sistema di interruzione (flip-flop IE della CPU) sia l'ultima azione effettuata dalla routine in corso di esecuzione.

In questo modo si perde però l'annidamento delle interruzioni (le più prioritarie che interrompono le meno). Il modo corretto di operare è un altro: riabilitare il sistema di interruzione durante la routine di servizio, ma riportare SFF a "0" solo al termine della routine medesima.

5.21 Per quanto concerne la domanda a) vale quanto detto per il precedente esercizio. Per quanto concerne la domanda b) l'interruzione si genera solo al momento in cui viene eseguita l'istruzione STI.

5.23 Il progetto dell'interfaccia verrà svolto in modo intuitivo e non formale. Per semplicità si fa l'assunzione che la richiesta di interruzione generata dal periferico si mantenga almeno fino al momento in cui non si ha la selezione. Si può ragionare in questo modo:

a) quando viene asserita la richiesta di interruzione IRQ del periferico, la linea INT viene asserita solo se è asserita IEI;
a) suk clock che campiona INTA asserito, se permangono le condizioni del punto precedente, si determina il passaggio allo stato "1" del flip-flop FF1, la cui uscita è in ingresso a un secondo flip-flop D (FF2); conseguentemente FF2 porta in stato "1" al clock successivo;
c) sull'ulteriore clock, il flip-flop SFF passa allo stato "1" se IEI è rimasto a "1", ovvero se le interfacce a monte, non hanno fatto richiesta di interruzione. Si noti che Sono passati due periodi di clock da quando la linea INTA è stata campionata asserita. L'interfaccia risulta selezionata, IEO è disasserito come pure IACK, che però resta asserito solo per un ciclo (il prossimo clock riporta il FF2 in stato "0");

Lo schema dell'interfaccia che opera come sopra è in Figura 5.8.

Si invita il lettore a effettuare il progetto in modo rigoroso, utilizzando il diagramma degli stati e provvedendo a sintetizzare la rete.

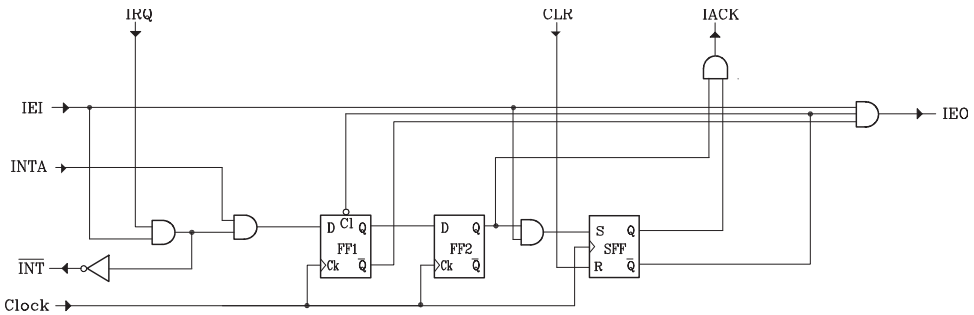

Figura 5.8 Schema della logica di daisy chain sincrona per l'Esercizio 5.23. Si è supposto che l'eventuale flip-flop di memorizzazione della richiesta di interruzione faccia parte delle logica del periferico. Si noti che INTA non esce verso valle, in quanto arriva in parallelo a tutte le interfacce. Il segnale IACK ha la durata di un ciclo di clock.