Quanto segue non è rilevante per il superamento dell'esame di Reti logiche.
Consideratelo un supporto per Calcolatori elettronici.

Interruzioni

Negli episodi precedenti

Riprendiamo un problema affrontato durante il corso di Reti logiche.

- ***Problema**: sincronizzazione tra processore e dispositivo. Il processore non ha modo di sincronizzarsi con i dispositivi (anche perché li vede solo come memorie)*
  - *Supponiamo che un programma contenga le seguenti istruzioni*

    ```
    IN offset_RBR, %AL
    …
    IN offset_RBR, %AL
    ```

    Nessuno può garantirmi che tra le due IN il dispositivo sia stato in grado di produrre un dato nuovo. Il secondo dato potrebbe non essere significativo.
  - *Dualmente…*

    ```
    OUT %AL, offset_TBR
    …
    OUT %AL, offset_TBR
    ```

    Nessuno può garantire che tra le due OUT il dispositivo abbia processato il dato.
- Abbiamo risolto questo problema introducendo dei flag (FI e FO con appositi registri RSR e TSR) e dei meccanismi di handshake (gestiti da RSS).
- Tuttavia una questione è rimasta in sospeso…

  ***Ma tutto questo è efficiente?** Per nulla: l'idea è che il processore rimanga in attesa, cioè che cicli finchè il dispositivo esterno non sarà pronto. Il processore perde tempo, considerando la lentezza delle reti che comunicano con lui.*

  ***Molto meglio** se il processore può andare avanti per conto proprio, con l'interfaccia che segnala al processore quando è pronto. A quel punto il processore **interrompe** il lavoro che sta facendo.*

Introduzione alle interruzioni

Invece di tenere in attesa il processore permetteremo a questo di continuare a lavorare. È necessario potenziare l'interfaccia e il processore stesso in modo che:

- L'interfaccia sia in grado di segnalare al processore quando essa è pronta al trasferimento
- Il processore reagisca a questa segnalazione interrompendo temporaneamente l'esecuzione del programma in corso, passando a un sottoprogramma di servizio predisposto per compiere l'operazione di ingresso/uscita del dato.

Non ci limiteremo a parlare di interruzioni solo per operazioni di I/O, ma lo faremo anche

- al verificarsi di particolari condizioni all'interno del processore, e
- al lancio di specifiche istruzioni Assembler (INT in testa, ne parleremo a breve).

Classi di interruzioni

Le interruzioni si dividono in tre classi:

- **interruzioni esterne**, richieste dalle interfacce mediante apposite variabili di collegamento;
- **interruzioni software**, richieste nella fase di esecuzione dell'istruzione INT;
- **interruzioni interne** (o *eccezioni*), richieste da circuiterie del processore in situazioni anomale.

Alcune interruzioni esterne sono dette *mascherabili*, poiché prese in considerazione dal processore solo in presenza di un certo flag uguale ad 1 (di cui parleremo più avanti).

Tipo dell'interruzione

Una interruzione è sempre accompagnata da un identificatore che permette di individuare il sottoprogramma di servizio da utilizzare.

Questo identificatore è detto *tipo dell'interruzione* ed è un numero naturale ad 8 bit: seguono 256 tipi di interruzioni possibili! Il modo in cui viene indicato il tipo dell'interruzione dipende dalla classe relativa:

- nelle interruzioni esterne si utilizzano i fili di dati d7_d0;
- nelle interruzioni software il tipo è l'unico operando esplicito dell'istruzione INT;
- nelle interruzioni interne il tipo è implicito e legato alla causa che comporta l'interruzione.

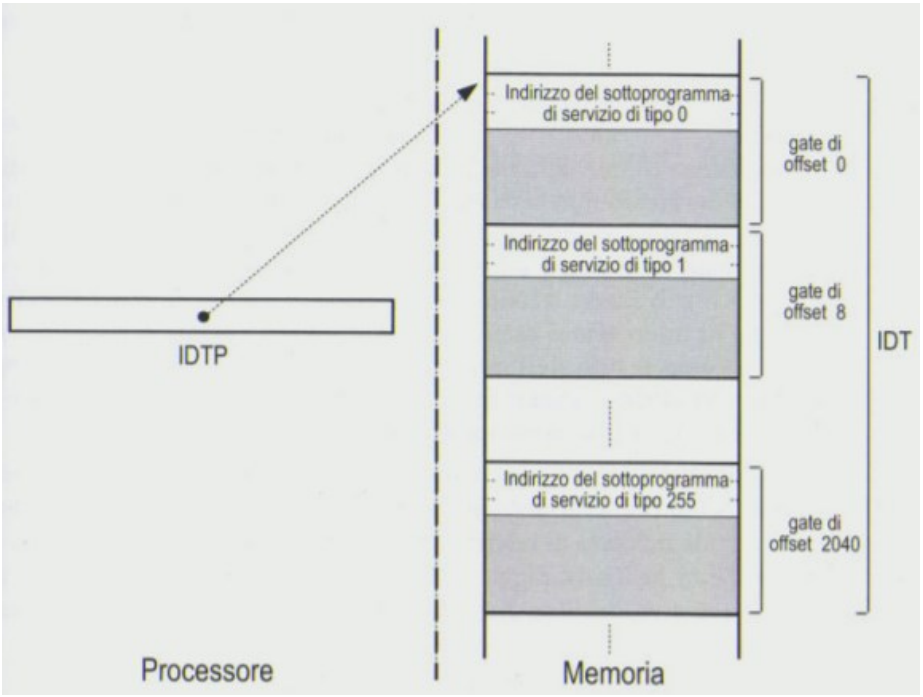
Legame tra tipo dell'interruzione e sottoprogramma da eseguire

- Per ogni tipo è necessario indicare l'indirizzo della prima istruzione del relativo sottoprogramma: ciò avviene attraverso un'area di memoria detta *Interrupt Descriptor Table* (IDT).
- Per raggiungere quando necessario quest'area di memoria è necessario introdurre un nuovo registro all'interno del processore: l'*Interrupt Descriptor Table Pointer* (IDTP)

- L'area di memoria consiste in $256 \cdot 8 = 2048$ locazioni di memoria. Ad ogni tipo di interruzioni sono associate 8 locazioni contigue, che formano l'*interrupt gate*. In questo:
  - le prime tre contengono l'indirizzo del sottoprogramma di servizio relativo all'interruzione;
  - le successive cinque locazioni contengono altre informazioni per noi irrilevanti.

- L'indirizzo dell'interrupt gate relativo a un certo tipo di interruzione può essere ottenuto con la seguente formula

$$\text{indirizzo\_del\_gate} = \text{IDTP} + \text{tipo} \cdot 8 = \text{IDTP} + \{\text{SOURCE}, 3'B000\}$$



si tenga conto che l'identificativo del tipo viene memorizzato nel registro SOURCE ogni volta.

- Durante l'esecuzione del programma bootstrap:
  - viene scelta un'area di memoria per la tabella IDT;
  - vengono scritti nella tabella IDT i relativi indirizzi;
  - viene impostato il contenuto del registro IDTP con l'istruzione LIDTP.

Istruzioni Assembler per gestire il meccanismo di interruzione

- `LIDTP $operando`
  L'istruzione inizializza il contenuto del registro IDTP ponendo come contenuto l'operando indicato: esso consiste in un indirizzo di memoria a 24 bit.
  - **Formato**: F1, varie ed eventuali visto che l'operando è a 24bit.

```
//----------- istruzione LIDTP $operando -----------------
ldIDTP:  begin
              A23_A0 <= IP;
              IP <= IP + 3;
              MJR <= ldIDTP1; STAR <= readM;
         end
ldIDTP1: begin
              IDTP <= {APP2, APP1, APP0};
              STAR <= fetch0;
         end
```

- `INT $operando`

  L'istruzione:
  1. memorizza nella pila il contenuto del registro dei flag e l'indirizzo dell'istruzione di rientro;
  2. azzera il contenuto del registro dei flag (mascherando così le richieste di interruzioni esterne);
  3. interpreta l'operando immediato come il tipo dell'interruzione e si procura nella IDT l'indirizzo della prima istruzione del sottoprogramma di servizio;
  4. immette tale indirizzo nel registro IP.

  - **Formato**: F4, visto che abbiamo come operando source una costante ad 8bit.

```
//----------- istruzione INT $operando -----------------
int:  begin
              A23_A0 <= SP-4;
              SP <= SP – 4;
              {APP3, APP2, APP1, APP0} <= {F, IP};
              F <= 'H00;
              MJR <= int1;
              STAR <= writeM;
      end
int1: begin
              A23_A0 <= IDTP + {SOURCE, 3'B000};
              MJR <= int2;
              STAR <= readM;
      end
int2: begin
              IP <= {APP2, APP1, APP0};
              STAR <= fetch0;
      end
```

- `IRET`

  equivalente della RET per la conclusione di sottoprogrammi di servizio relativi a interruzioni. Fa le stesse cose della RET, ma in più rinnova il contenuto dei registri IP e flag recuperando il loro valore dalla pila (quindi si prelevano 4 byte dalla pila).
  - **Formato**: F0, non c'è da fare nulla per quanto riguarda gli operandi.

```
//----------- istruzione IRET -----------------
iret:  begin
              A23_A0 <= SP;
              SP <= SP+4;
              MJR <= iret1;
              STAR <= readL;
       end
iret1: begin
              {F,IP} <= {APP3, APP2, APP1, APP0};
              STAR <= fetch0;
       end
```

Modifiche da apportare al processore sEP8 per implementare il meccanismo delle interruzioni interne: esempio

- Abbiamo già detto che le interruzioni interne, ossia le eccezioni, vengono lanciate in caso di situazioni anomale.
- Un esempio di situazione anomala è un OPCODE invalido. Abbiamo già visto che nella fase di fetch si verifica la validità dell'OPCODE mediante la function `valid_fetch`

```
function valid_fetch;
input [7:0] opcode;
...  ...
endfunction
```

Prende in ingresso un byte e restituisce 1 se quel byte è l'OPCODE di un'istruzione nota, 0 altrimenti

- Ricordiamo la fase di fetch

```
fetch0: begin A23_A0<=IP; IP<=IP+1; MJR<=fetch1; STAR<=readB; end
fetch1: begin OPCODE<=APP0; STAR<=fetch2; end
fetch2: begin
MJR<=(OPCODE[7:5]==F0)? fetchEnd:
(OPCODE[7:5]==F1)? fetchEnd:
(OPCODE[7:5]==F2)? fetchF2_0:
(OPCODE[7:5]==F3)? fetchF3_0:
(OPCODE[7:5]==F4)? fetchF4_0:
(OPCODE[7:5]==F5)? fetchF5_0:
(OPCODE[7:5]==F6)? fetchF6_0:
/* default */ fetchF7_0;
STAR<=(valid_fetch(OPCODE)==1)? fetch3 : nvi;
end
```

Istruzione di ritorno. Passiamo a fetch1 dopo la lettura.

Porto il byte letto in OPCODE e passo allo stato successivo

Prendo le tre cifre più significative dell'OPCODE e verifico a quale formato corrispondono.

Se l'OPCODE è valido passo a fetch3, altrimenti ad nvi

Nessun problema a eseguirle nello stesso stato, valid_fetch non dipende da MJR.

```
// SALTO AL PRIMO PASSO SPECIFICO DELLA FASE DI FETCH (Lo abbiamo determinato prima)
fetch3: begin STAR<=MJR; end

// Eseguo se l'OPCODE non è valido (rivedere fetch2)
nvi: begin STAR<=nvi; end
```

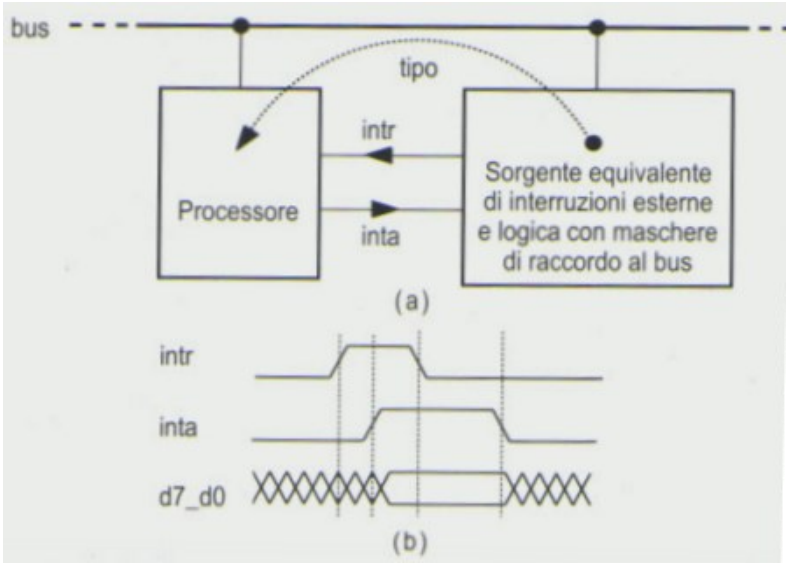
- Per introdurre la gestione dell'interruzione interna, relativa all'OPCODE non valido, dobbiamo modificare quanto avviene nello stato interno `nvi`

  ```
  nvi: begin SOURCE <= 'H06; STAR <= int; end
  ```

  Si tenga conto che il tipo dell'interruzione è implicito e identificato dal numero naturale 6 (*eccezione per codice operativo non valido*).

Modifiche da apportare al processore sEP8 per implementare il meccanismo delle interruzioni esterne

Per poter permettere al processore di gestire le interruzioni esterne è necessario potenziarlo fornendolo di variabili di ingresso dedicate alla ricezione di tali richieste di interruzione.



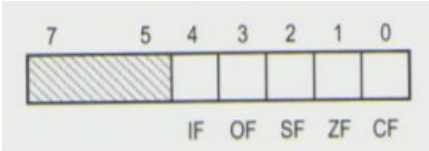
Connessioni tra processore e sorgente equivalente di interruzioni esterne (a); handshake tra processore e sorgente equivalente (b).

- Si suppone che ci sia una sola variabile di ingresso, quindi che tutte le richieste di eccezione passino da un'unità di raccordo (un controllore delle interruzioni). Questa unita presenta una alla volta le richieste provenienti dalle varie sorgenti (il tutto appare come una sorgente equivalente).

- Le variabili che introduciamo sono le seguenti:
  - *intr* (variabile di ingresso per il processore e di uscita per la sorgente equivalente)
  - *inta* (variabile di uscita per il processore, segue relativo registro a sostegno, e di ingresso per la sorgente equivalente)

- **Step presenti**:
  1. Inizialmente i due valori sono uguali a zero;
  2. La sorgente equivalente compie il primo step alzando ad 1 intr.
  3. Quando il processore è pronto ad accettare la richiesta di interruzione esterna, risponde portando inta ad 1.

- L'accettazione della richiesta avviene dopo aver concluso l'esecuzione di un'istruzione e prima di iniziare la fase di fetch per una nuova istruzione.
- Il processore, inoltre, accetta nuova richieste solo se un particolare flag nel registro dei flag è uguale ad 1: il flag in questione (che introduciamo ora) è detto *Interrupt flag* (IF) ed è posto nella posizione 4 del registro.



Registro F col nuovo flag IF

    - **Perché non accettiamo quando il flag è a zero?** Abbiamo già detto che una delle cose fatte nella gestione delle interruzioni è l'azzeramento del contenuto del registro dei flag[1]. Segue che il flag è a zero quando stiamo eseguendo un sottoprogramma di servizio relativo a un'interruzione.
    - Non si ha annidamento salvo uso della STI nel sottoprogramma di servizio stesso.
4. Successivamente la sorgente equivalente invia, mediante fili di dati, il tipo dell'interruzione: notifica ciò ponendo intr a 0.
5. A questo punto eseguiamo azioni identiche a quelle compiute con l'esecuzione dell'istruzione INT.

- **Nuovi stati interni**: per gestire queste nuove variabili è necessario introdurre nuovi stati interni

```
//------------ Verifica della presenza di richieste di interruzioni -----
------------- esterne non mascherate ed eventuale prelievo del tipo ----
test_intr:  begin
                  STAR <= ((intr&F[4]) == 0) ? fetch0 : pre_tipo;
            end
pre_tipo0:  begin
                  INTA <= 1;
                  STAR <= (intr == 1) ? pre_tipo0 : pre_tipo1;
            end
pre_tipo1:  begin
                  SOURCE <= d7_d0;
                  INTA <= 0;
                  STAR <= int;
            end
```

Il meccanismo che vediamo è chiaramente un handshake. Abbiamo già detto che il controllo avviene tra la fine della fase di esecuzione di un'istruzione e l'inizio della fase di fetch dell'istruzione successiva: segue che andremo ad effettuare le seguenti sostituzioni in tutta la descrizione Verilog del calcolatore

```
STAR <= fetch0;          ==========>          STAR <= test_intr;
MJR  <= fetch0;          ==========>           MJR <= test_intr;
```

- **Istruzioni Assembler per gestire il flag IF**:
    1. `CLI`
       Il processore mette a 0 il contenuto del flag IF, lasciando inalterato il contenuto di tutti gli altri flag (*CLear Interrupt flag*)
        - **Formato**: F0 (non abbiamo operandi)

    2. `STI`
       Il processore mette a 1 il contenuto del flag IF, lasciando inalterato il contenuto di tutti gli altri flag (*Set Interrupt flag*)
        - **Formato**: F0 (non abbiamo operandi)

[1] Presumo che si esegua la STI nel programma bootstrap.

Controllore delle interruzioni

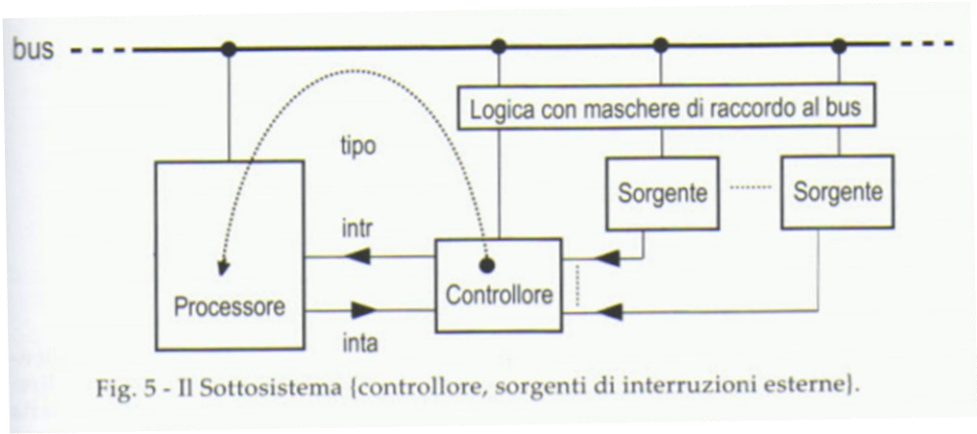


Fig. 5 - Il Sottosistema {controllore, sorgenti di interruzioni esterne}.

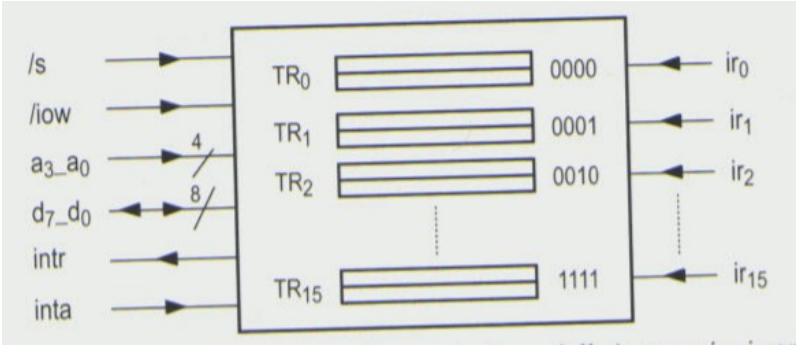
- In un qualunque calcolatore sono presenti più sorgenti di interruzioni esterne: segue che l'unica sorgente equivalente (quella con cui il processore colloquia) sia ottenuta interponendo fra le sorgenti e il processore una unità di raccordo nota come *controllore delle interruzioni*.

- Il compito del controllore è associare un tipo ad ogni sorgente di interruzioni
    - nel raccogliere le richieste che provengono dalle varie sorgenti,
    - nel sequenzializzare le richieste su base prioritaria,
    - nel presentare le richieste una alla volta al processore (unitamente al loro tipo).
- **Osservazione**: ogni sorgente invia un segnale al controllore, ma non è presente un meccanismo di handshake che informa il sorgente dell'avvenuta ricezione del segnale da parte del controllore.

- **Definizione di sorgente di interruzioni esterne**: una qualunque unità dotata di una variabile di uscita *ir* e di un registro OK. L'unità opera in accordo alle seguenti specifiche:
    - Partendo da una condizione iniziale in cui *ir* è a 0, la sorgente mette *ir* a 1 quando nella sorgente stessa nasce l'esigenza di inviare al processore una richiesta di interruzione.
    - Il registro OK è atto a dar corpo a una porta dello spazio di I/O.
    - Un accesso del processore al registro OK ha l'effetto di informare la sorgente che la richiesta di interruzione è stata accolta, e che la variabile ir va riportata a 0.
    - L'istruzione di IN/OUT che svolge questo accesso deve trovarsi nel sottoprogramma di servizio dell'interruzione stessa.

- **Esempi**:
    - **Interfaccia parallela di uscita con handshake**: utilizzare la variabile fo come variabile ir e il registro TBR come registro OK.
    - **Interfaccia parallela di ingresso con handshake**: utilizzare la variabile fi come variabile ir e il registro RBR come registro OK.

Visione funzionale del controllore delle interruzioni



- Il controllore da noi esaminato supporta 16 sorgenti di interruzioni. Osserviamo la presenza di
  - 16 variabili *ir* provenienti dalle sorgenti esterne, e di
  - 16 registri TR dove poniamo l'identificativo numerico del tipo dell'interruzione.

  Si consideri come **legge di associazione** la seguente: *alla k-esima sorgente, cioè alla sorgente connessa al controllore tramite le variabile ir k-esima, il controllore assegna come tipo il contenuto del registro TR k-esimo*.

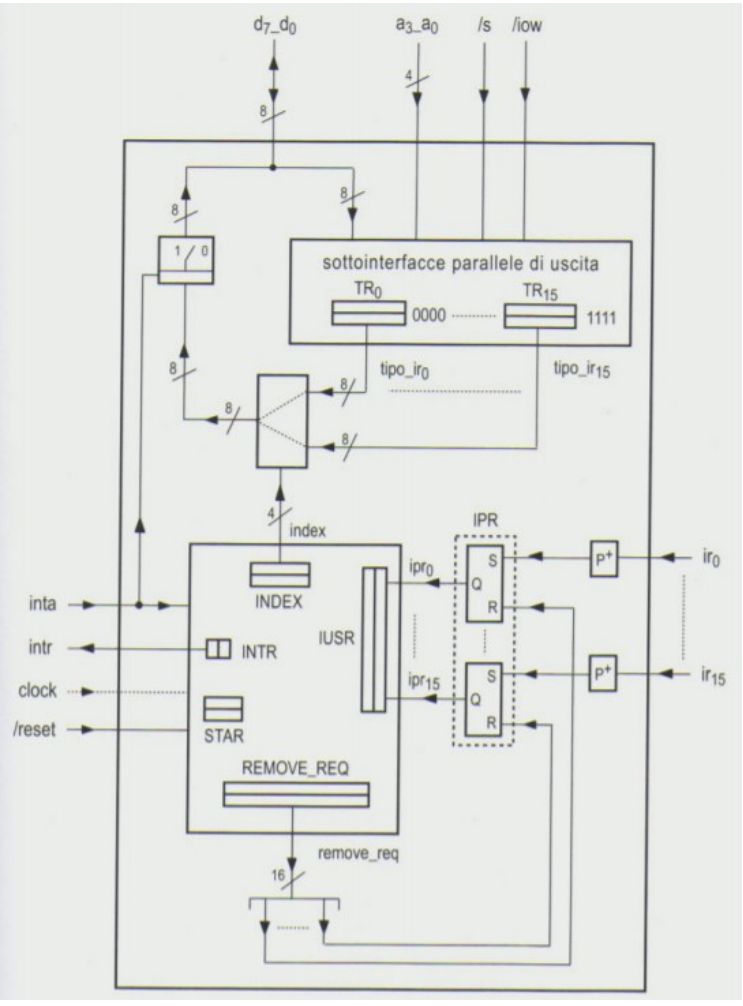
- I registri TR, ricordiamolo, danno luogo a 16 porte dello spazio di I/O. Si consideri la presenza, in ingresso, dei fili di pilotaggio tipici di una interfaccia (piedino di select, di iow, fili di indirizzo e fili di dati).

Spogliamo il controllore delle interruzioni

- Ogni variabile *ir* k-esima viene gestita mediante un Latch SR, che memorizza lo stato della richiesta di interruzione. In ciascun Latch SR:
  - s consiste nella variabile ir passata da un formatore di impulsi (mi basta un semplice impulso per settare il valore del latch sr)
  - r consiste in un bit proveniente dal registro REMOVE_REQ.

- Lo stato delle richieste viene memorizzato nel registro IUSR (*Interrupt Under Service Requests*). Il registro avrà tanti bit quante le sorgenti esterne supportate.

- Il controllore seleziona una delle richieste di interruzione presenti e pone l'indice della variabile di ingresso ir nel registro INDEX. Il controllore supporta 16 interruzioni, quindi il registro avrà 4 bit.
- Il contenuto del registro è una variabile di pilotaggio di un multiplexer che restituisce il tipo dell'interruzione (memorizzato nel relativo registro TR).



- **Ricordiamo l'handshake inta-intr**:
  1. Inizialmente i due valori sono uguali a zero;
  2. Il controllore alza intr ad 1 quando ha delle richieste.
  3. Il processore indica che è disponibile a gestire l'interruzione alzando inta ad 1.
  4. Il controllore risponde ponendo sui fili di dati il tipo dell'interruzione. Notifica ciò abbassando intr a 0.

- Si osservi la presenza di una porta tristate che è in conduzione quando *inta* è uguale ad 1, cioè quando il processore indica la sua disponibilità a gestire l'interruzione esterna.

- **Come gestiamo la priorità?** Assegnamo maggiore priorità alle interruzioni pervenute da variabili *ir* con indice più basso. La cosa è evidente analizzando la function `select_index`: essa restituisce l'indice della richiesta da gestire dato in ingresso il contenuto del registro IUSR. Il valore restituito dalla function viene posto nel registro INDEX introdotto nella pagina precedente.

- **Come resettiamo i bit del registro IUSR?**
  - Resettiamo in IUSR il bit della relativa richiesta di interruzione non appena la prendiamo in carico.
  - Poniamo come contenuto del registro REMOVE_REQ il risultato della function `decode`. Data una particolare richiesta di interruzione (posta in INDEX) ottengo una serie di bit a zero, tutti tranne uno. Questi bit consistono nelle variabili di reset dei Latch SR: segue che andremo a resettare il contenuto di un Latch SR (quello associato alla relativa variabile ir), quindi il contenuto del registro IUSR sarà aggiornato.

- **Descrizione Verilog della sottorete che gestisce le richieste di interruzione e la loro priorità**:

```
module Sottorete_Interna_al_Controllore(
      inta, ipr15_ipr0,
      intr, index, remove_req,
      clock, reset_
);
      input clock, reset_;

      input inta;
      output intr;
      input [15:0] ipr15_ipr0;
      output [3:0] index;
      output [15:0] remove_req;

      reg INTR; assign intr = INTR;
      reg [3:0] INDEX; assign index = INDEX;
      reg [15:0] REMOVE_REQ; assign remove_req = REMOVE_REQ;
      reg [15:0] IUSR;

      reg [1:0] STAR;
      localparam S0 = 0, S1 = 1, S2 = 2;

      always @(reset_ == 0) #1 begin
                                  INTR <= 0;
                                  REMOVE_REQ <= 'HFFFF;
                                  IUSR <= 'H0000;
                                  STAR <= S0;
                               end

      always @(posedge clock) if(reset_ == 1) #3
            casex(STAR)
              S0: begin
                    REMOVE_REQ <= 'H0000;
                    IUSR <= ipr15_ipr0;
                    STAR <= S1;
                  end
              S1: begin
                    INTR <= (IUSR == 'H0000) ? 0 : 1;
                    INDEX <= selected_index(IUSR);
                    STAR <= (inta == 0) ? S0 : S2;
                  end
              S2: begin
                    INTR <= 0;
                    REMOVE_REQ <= decode(INDEX);
                    STAR <= (inta == 1) ? S2 : S0;
                  end
            endcase
```

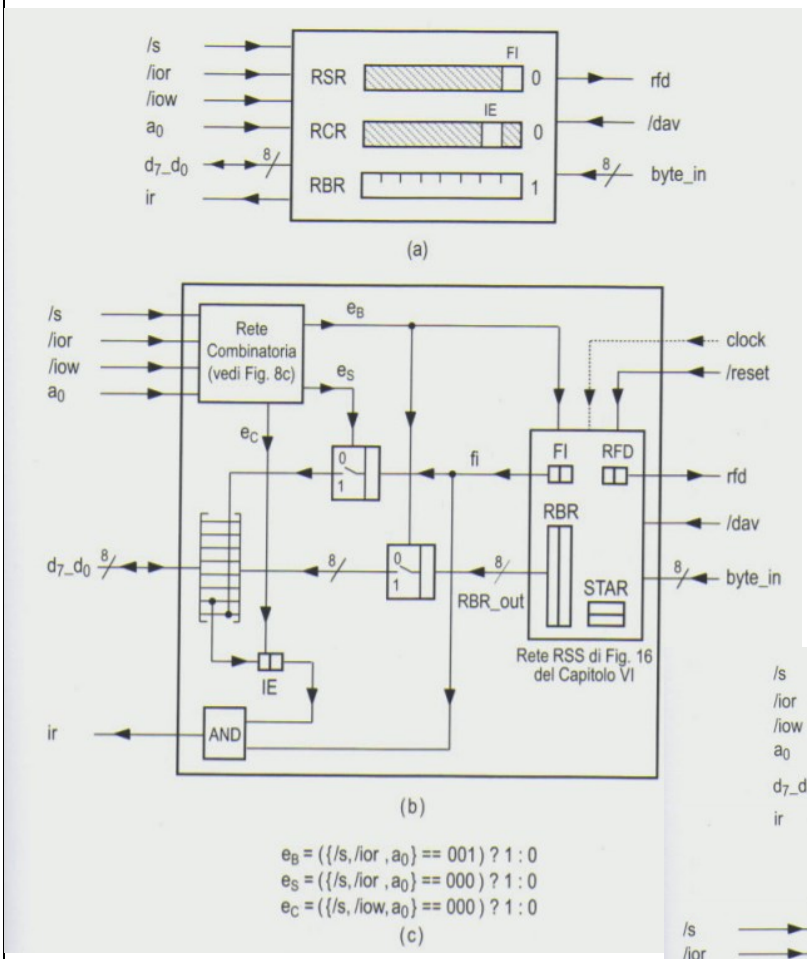
```
    function [3:0] selected_index;
        input [15:0] IUSR;
        casex(IUSR)
            'B???????????????1 : selected_index = 'B0000;
            'B??????????????10 : selected_index = 'B0001;
            'B?????????????100 : selected_index = 'B0010;
            'B????????????1000 : selected_index = 'B0011;
            'B???????????10000 : selected_index = 'B0100;
            'B??????????100000 : selected_index = 'B0101;
            'B?????????1000000 : selected_index = 'B0110;
            'B????????10000000 : selected_index = 'B0111;
            'B???????100000000 : selected_index = 'B1000;
            'B??????1000000000 : selected_index = 'B1001;
            'B?????10000000000 : selected_index = 'B1010;
            'B????100000000000 : selected_index = 'B1011;
            'B???1000000000000 : selected_index = 'B1100;
            'B??10000000000000 : selected_index = 'B1101;
            'B?100000000000000 : selected_index = 'B1110;
            'B1000000000000000 : selected_index = 'B1111;
        endcase
    endfunction

    function [15:0] decode;
        input [3:0] INDEX;
        casex(INDEX)
            'B0000: decode = 'B0000000000000001;
            'B0001: decode = 'B0000000000000010;
            'B0010: decode = 'B0000000000000100;
            'B0011: decode = 'B0000000000001000;
            'B0100: decode = 'B0000000000010000;
            'B0101: decode = 'B0000000000100000;
            'B0110: decode = 'B0000000001000000;
            'B0111: decode = 'B0000000010000000;
            'B1000: decode = 'B0000000100000000;
            'B1001: decode = 'B0000001000000000;
            'B1010: decode = 'B0000010000000000;
            'B1011: decode = 'B0000100000000000;
            'B1100: decode = 'B0001000000000000;
            'B1101: decode = 'B0010000000000000;
            'B1110: decode = 'B0100000000000000;
            'B1111: decode = 'B1000000000000000;
        endcase
    endfunction
endmodule
```

Modifiche alle interfacce parallele con handshake

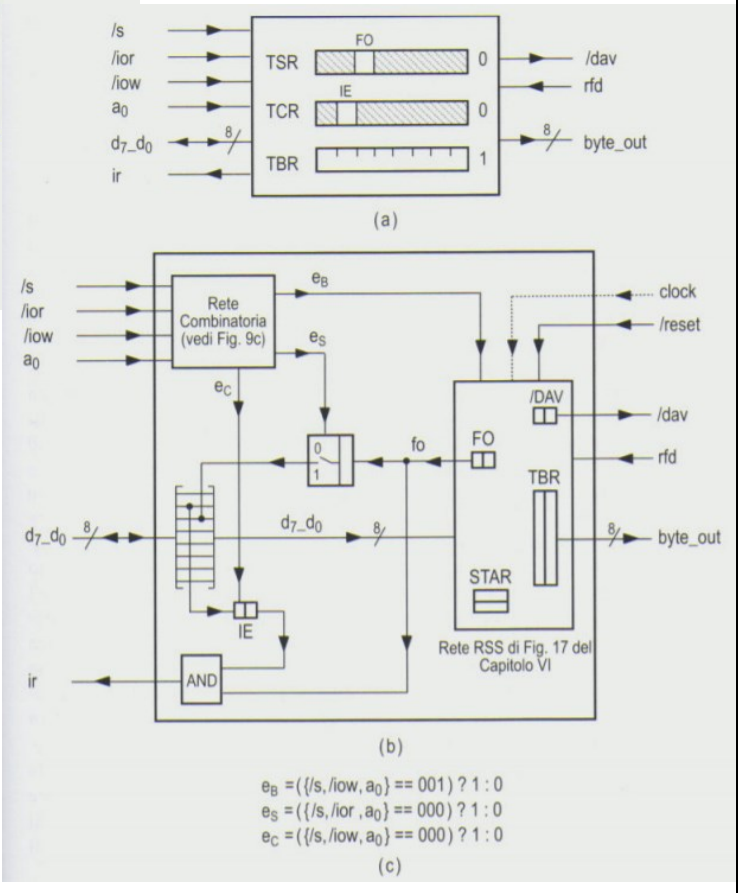
- Da pagina 270 del libro di Corsini è presente la spiegazione su come aggiornare le interfacce parallele con handshake per renderle gestibili a interruzione di programma.

- In entrambi i casi andremo a introdurre, come già anticipato, la variabile di uscita *ir*.
- Sia nell'interfaccia parallela di ingresso che in quella di uscita il valore di *ir* è il risultato di una porta AND avente in ingresso un flag (FI per l'ingresso, FO per l'uscita) e il valore di un nuovo registro: IE.
- Il registro IE (Interrupt Enable) permette di abilitare o disabilitare la gestione con interruzione di programma. Precisamente: se IE è uguale ad 1 l'uscita ir dipenderà dal flag relativo, se IE è uguale a 0 l'uscita ir sarà sempre uguale a zero indipendentemente dal flag relativo.
- Per evitare richieste di interruzione impreviste è necessario porre IE uguale a 0 come condizione di reset.
- Il clock del registro IE è la variabile e_C generata dalla rete combinatoria (si osservi nelle immagini quando e_C viene posto ad 1.

- **Ricapitoliamo i registri**:
  - Nell'interfaccia di ingresso abbiamo il RBR, accessibile in sola lettura
  - Nell'interfaccia di uscita abbiamo il TBR, accessibile in sola scrittura.
  - Il registro di stato (TSR nell'interfaccia di uscita, RSR nell'interfaccia di ingresso), accessibile in sola lettura.
  - In entrambi i casi un registro di comando (RCR – *Receive command register* - nell'interfaccia di ingresso, TCR – *Transmit command register* - nell'interfaccia di uscita), che è accessibile in sola scrittura e contiene il bit di IE. La posizione del bit varia: nell'interfaccia di ingresso è in posizione 1, mentre nell'interfaccia di uscita è in posizione 6.
    - **Osservazione**: RSR/TSR e RCR/TCR hanno lo stesso indirizzo interno. La cosa non genera ambiguita in quanto un registro è accessibile in sola lettura, e l'altro in sola scrittura.



Interfaccia parallela di ingresso compatibile col controllore delle interruzioni



Interfaccia parallela di uscita compatibile col controllore delle interruzioni

Protezione

Introduzione

- Nella guerra tra hacker e sviluppatori l'introduzione di sistemi di protezione hardware può essere un valido soccorso per i secondi (chiaramente non una soluzione definitiva).
- Distingueremo due modalità: la modalità sistema, in cui si possono eseguire tutte le istruzioni esistenti, e la modalità utente, in cui non è possibile eseguire le cosiddette *istruzioni privilegiate*.
- La tentata esecuzione di queste in modalità utente comporta una richiesta di interruzione interna.
- Il meccanismo di protezione che introdurremo prevede la possibilità di gestire due pile separate, di cui una accessibile esclusivamente in modalità sistema.

Quali sono le istruzioni privilegiate?

- La tentata esecuzione di una di queste istruzioni in modalità utente comporta un'interruzione interna di tipo 5 (*interruzione per tentata esecuzione di istruzione privilegiata*).

- Le istruzioni privilegiate sono le seguenti:
  - La HLT
  - Le istruzioni IN e OUT, la cui esecuzione può manomettere le interfacce
  - L'istruzione LIDTP, che agisce sul contenuto di un registro estremamente delicato
  - Le istruzioni CLI e STI, la cui esecuzione interagisce con il mascheramento/smascheramento delle richieste di interruzione esterne
  - L'istruzione IRET, per motivi che chiariremo più avanti.

- La INT non è istruzione privilegiata, ma è centrale nella gestione del meccanismo: la sua esecuzione, dopo alcune modifiche, permetterà il passaggio in modalità sistema per eseguire, in qualunque contesto, i sottoprogrammi di servizio.
- L'istruzione IRET, privilegiata, andrà modificata per gestire il ritorno del processore in modalità utente.

- Introdurremo, più avanti, nuove istruzioni per gestire questo meccanismo di protezione.

Memoria di sistema

- Definiremo una porzione di memoria a cui il processore non può accedere mentre opera in modalità utente.
- Quest'area è detta memoria di sistema, ed è a esclusiva disposizione del software di sistema.
- Caratteristiche:
  - Sono inibiti al processore in modalità utente esclusivamente gli accessi in scrittura (per semplificarsi l'esistenza)
  - È costituita dalle $2M = 2^1 \cdot 2^{20} = 2^{21}$ locazioni il cui indirizzo inizia con 000 (si veda la condizione posta nel micro-sottoprogramma per scritture in memoria).
  - Una tentata scrittura in questa area di memoria in modalità utente comporta una richiesta di interruzione interna di tipo 4 (*interruzione per violazione della memoria protetta*).

Pila di sistema e pila utente

- Per separare ulteriormente la modalità sistema dalla modalità utente andremo a dotare il processore di un supporto per distinguere una pila utilizzabile in modalità sistema (detta *pila di sistema*) da una pila utilizzabile in modo utente (detta *pila utente*).

- Precisamente:
  - Doteremo il processore di un nuovo registro puntatore (quindi a 24bit) detto PMSP (*Previous Mode Stack Pointer*), che affiancato al registro SP permetterà di ricordarsi l'indirizzo di memoria di un registro quando stiamo usando l'altro.
  - Gestiremo il valore di questo nuovo registro con una nuova istruzione privilegiata EXCHSP (*EXCHange Stack Pointers*), con cui scambieremo i contenuti dei registri PMSP ed SP.
  - Prevederemo, in un qualunque passaggio fra modo sistema e modo utente, e viceversa, uno scambio automatico di contenuto tra i due registri.

- Chiaramente la pila di sistema sarà allocata nella cosiddetta memoria di sistema.
- Potremo svolgere operazioni con la PUSH e la POP solo sulla pila indirizzata da SP, e non su quella indirizzata da PMSP.

Regole per un passaggio sicuro da modalità sistema a modalità utente e viceversa

- Memorizzeremo l'attuale modalità in cui si trova il processore introducendo un nuovo flag in posizione 5 del registro dei flag: precisamente il flag U/S (*User/System flag*).
- Questo flag è uguale a zero quando il processore si trova in modalità sistema, uguale ad 1 quando si trova in modalità utente.

- Si consideri che:
  - Al momento del reset tutti i flag sono uguali a zero, quindi il processore si troverà in modalità sistema.
  - Segue che il programma bootstrap sarà eseguito in modalità sistema.
  - A un certo punto il processore si porta in modalità utente utilizzando l'istruzione STUM (*SeTUserMode*), che imposta a 1 il contenuto del flag.
- Dopo essersi spostato in modalità utente il processore potrà tornare in modalità sistema solo ed esclusivamente gestendo una richiesta di interruzione. L'istruzione IRET riporterà il processore in modalità utente dopo aver gestito l'interruzione.

- **Segue la necessità di modificare il micro-sottoprogramma per scritture in memoria**:

```
// MICROSOTTOPROGRAMMA PER SCRITTURE IN MEMORIA, CON ECCEZIONE PER TENTATA
SCRITTURA, IN MODO UTENTE, NELLA MEMORIA PROTETTA

writeB: begin D7_D0<=APP0; DIR<=1; NUMLOC<=1; STAR<=write0; end
writeW: begin D7_D0<=APP0; DIR<=1; NUMLOC<=2; STAR<=write0; end
writeM: begin D7_D0<=APP0; DIR<=1; NUMLOC<=3; STAR<=write0; end
writeL: begin D7_D0<=APP0; DIR<=1; NUMLOC<=4; STAR<=write0; end

write0: begin
            MW <= ({A23_A0[23:21], F[5]}== 'B000_1)? 1 : 0;
            STAR <= ({A23_A0[23:21], F[5]}== 'B000_1)? nvma : write1;
        end

write1: begin MW_<=1; STAR<=(NUMLOC==1)?write11:write2; end
write2: begin D7_D0<=APP1; A23_A0<=A23_A0+1; NUMLOC<=NUMLOC-1; STAR<=write3;
end

write3: begin
            MW <= ({A23_A0[23:21], F[5]}== 'B000_1)? 1 : 0;
            STAR <= ({A23_A0[23:21], F[5]}== 'B000_1)? nvma : write4;
        end

write4: begin MW_<=1; STAR<=(NUMLOC==1)?write11:write5; end
write5: begin D7_D0<=APP2; A23_A0<=A23_A0+1; NUMLOC<=NUMLOC-1; STAR<=write6;
end

write6: begin
            MW <= ({A23_A0[23:21], F[5]}== 'B000_1)? 1 : 0;
            STAR <= ({A23_A0[23:21], F[5]}== 'B000_1)? nvma : write7;
        end

write7: begin MW_<=1; STAR<=(NUMLOC==1)?write11:write8; end
write8: begin D7_D0<=APP3; A23_A0<=A23_A0+1; STAR<= write9; end

write9: begin
            MW <= ({A23_A0[23:21], F[5]}== 'B000_1)? 1 : 0;
            STAR <= ({A23_A0[23:21], F[5]}== 'B000_1)? nvma : write10;
        end

write10: begin MW_<=1; STAR<= write11; end
write11: begin DIR<=0; STAR<=MJR; end

nvma:    begin DIR <= 0; SOURCE <= 'H04; STAR <= int; end
```

Nuove istruzioni privilegiate: EXCHSP e STUM

- EXCHSP

  Il processore scambia il contenuto del registro SP con quello del registro PMSP, permettendone anche l'inizializzazione.
  - **Formato**: F0, non c è da fare nulla per quanto riguarda gli operandi.

```
//-------------- istruzione EXCHSP -------------------
exchSP: begin
              SP <= PMSP;
              PMSP <= SP;
              STAR <= test_intr;
        end
```

- STUM

  Il processore mette a 1 il contenuto del flag U/S passando a lavorare in modo utente e lasciando inalterato il contenuto di tutti gli altri flag. Inoltre, scambia il contenuto del registro SP con quello del registro PMSP rendendo non ulteriormente accessibile la pila di sistema
  - **Formato**: F0, non c'è da fare nulla per quanto riguarda gli operandi.

```
//-------------- istruzione STUM -------------------
stum:   begin
              F <= {F[7:6], 1'B1, F[4:0]};
              SP <= PMSP;
              PMSP <= SP;
              STAR <= test_intr;
        end
```

Modifiche alle istruzioni INT e IRET

- INT $operando

  Il processore passa a operare in modalità sistema, maschera le richieste di interruzioni esterne ed effettua la chiamata di un sottoprogramma di servizio. Precisamente:
  - se il processore si trova in modalità utente scambia il contenuto del registro SP con quello del registro PMSP;
  - salva nella pila di sistema il contenuto del registro IP e del registro dei flag;
  - azzera il contenuto del registro dei flag, portandosi così a operare definitivamente in modalità sistema e mascherando le richieste di interruzioni esterne;
  - si procura nella tabella delle interruzioni l'indirizzo della prima istruzione del sottoprogramma di servizio e immette tale indirizzo nel registro IP.

```
//----------- istruzione INT $operando -----------------
int:  begin
              SP <= (F[5] == 1) ? PMSP : SP;
              PMSP <= (F[5] == 1) ? SP : PMSP; STAR <= int1;
      end
int1: begin
              A23_A0 <= SP-4;
              SP <= SP - 4;
              {APP3, APP2, APP1, APP0} <= {F, IP};
              F <= 'H00;
              MJR <= int2; STAR <= writeM;
      end
int2: begin
              A23_A0 <= IDTP + {SOURCE, 3'B000};
              MJR <= int3;
              STAR <= readM;
      end
int2: begin
              IP <= {APP2, APP1, APP0};
              STAR <= fetch0;
      end
```

- IRET

  Equivalente della RET, ma in aggiunta rimuove 4 byte dalla pila di sistema rinnovando il contenuto dei due registri IP ed F. Inoltre, se l'alterazione dei flag comporta un ritorno in modalità utente viene scambiato il contenuto del registro SP con quello del registro PMSP.

```
//----------- istruzione IRET -----------------
iret:  begin
               A23_A0 <= SP;
               SP <= SP+4;
               MJR <= iret1;
               STAR <= readL;
       end
iret1: begin
               {F,IP} <= {APP3, APP2, APP1, APP0};
               STAR <= iret2;
       end
iret2: begin
               SP <= (F[5] == 1) ? PMSP : SP;
               PMSP <= (F[5] == 1) ? SP : PMSP;
               STAR <= fetch0;
       end
```

Modifica della descrizione Verilog del processore nella parte relativa alla fase di fetch

- Modifichiamo la fase di fetch per introdurre il meccanismo di protezione spiegato.
- Per prima cosa è necessario modificare la function `valid_fetch` in modo che questa riceva in ingresso anche il flag U/S. Inoltre l'uscita deve essere estesa a due bit. Precisamente
  - se l'uscita è 00 il byte in ingresso non coincide con il codice operativo di una istruzione valida;
  - se l'uscita è 01 il flag U/S è a 1 e l'istruzione richiesta è una delle istruzioni privilegiate (HLT, IN, OUT, CLI, STI, IRET, LIDTP, EXCHSP, STUM);
  - se l'uscita è 11 tutto ok.

```
function [1:0] valid_fetch;
input [7:0] opcode;
input flag;
...  ...
endfunction
```

- Il resto è automatico, modifichiamo l'assegnamento procedurale di STAR in fetch2 (abbiamo modificato la function) e introduciamo una condizione in nvi per determinare SOURCE. Contrariamente a prima dobbiamo verificare il contenuto restituito dalla `valid_fetch` per determinare il tipo di interruzione.

```
fetch0: begin A23_A0<=IP; IP<=IP+1; MJR<=fetch1; STAR<=readB; end
fetch1: begin OPCODE<=APP0; STAR<=fetch2; end
fetch2: begin
MJR<=(OPCODE[7:5]==F0)? fetchEnd:
(OPCODE[7:5]==F1)? fetchEnd:
(OPCODE[7:5]==F2)? fetchF2_0:
(OPCODE[7:5]==F3)? fetchF3_0:
(OPCODE[7:5]==F4)? fetchF4_0:
(OPCODE[7:5]==F5)? fetchF5_0:
(OPCODE[7:5]==F6)? fetchF6_0:
/* default */ fetchF7_0;
STAR<=(valid_fetch(OPCODE, F[5])== 'B11) ? fetch3 : nvi;
end

// SALTO AL PRIMO PASSO SPECIFICO DELLA FASE DI FETCH (Lo abbiamo determinato prima)
fetch3: begin STAR<=MJR; end
// Eseguo se l'OPCODE non è valido (rivedere fetch2)
nvi: begin
        SOURCE <=(valid_fetch(OPCODE, F[5])== 'B00) ? 'H06 : 'H05;
        STAR <= int;
     end
```