Architettura degli Elaboratori

Lezione 18 – Sottosistema di I/O

Giuseppe Cota
Dipartimento di Scienze Matematiche Fisiche e Informatiche
Università degli Studi di Parma

Indice

- Sottosistema di I/O
- Gestione a controllo di programma
- Gestione a interrupt
- Accesso diretto alla memoria

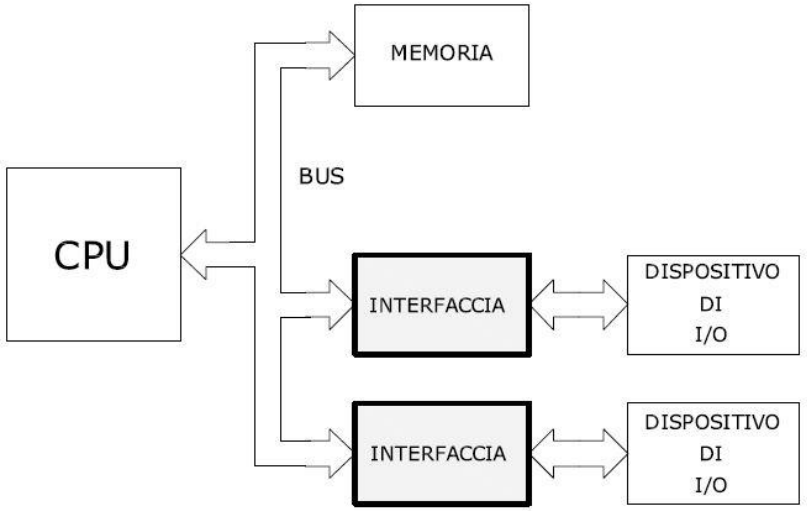
Premessa: tipi di interruzioni

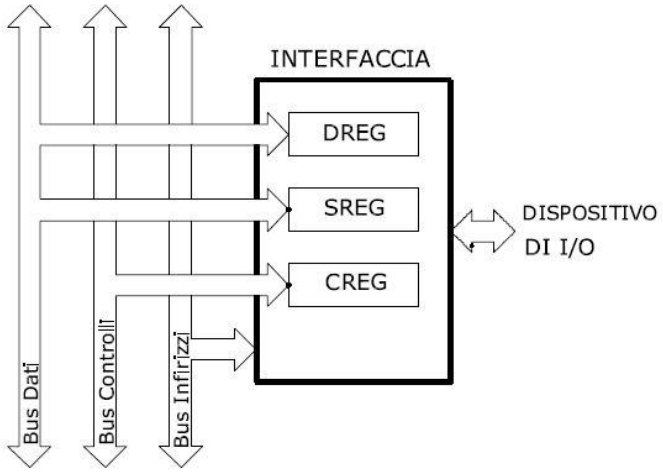
- Un'**interruzione** è un qualsiasi evento che, pur non essendo un salto o una chiamata/ritorno da procedura, altera il normale flusso di esecuzione del programma.
  - La CPU deve interrompere la normale esecuzione del programma ed eseguire un pezzo di codice dipendente dall'interruzione.
- Tipi di interruzione:
  - **Interruzioni esterne** (*external interrupt*): sono interruzioni generate dall'esterno.
  - **Eccezioni**: sono causate da situazioni anomale rilevate durante l'esecuzione del programma. Ad es. overflow, underflow, divisione per zero, accesso fallito alla memoria, ecc.
  - **Trappole** (*traps*): interruzioni software. Sono interruzioni generate da apposite istruzioni presenti nel programma. Sono particolari istruzioni di salto che hanno l'effetto di portare la macchina in opportune modalità di funzionamento.

Sottosistema di I/O

- La comunicazione tra il calcolatore e il mondo esterno avviene attraverso il **sottosistema di ingresso/uscita (Sottosistema I/O, da Input/Output).**
- Fanno parte del sottosistema di I/O:
  - Dispositivi attraverso i quali l'utente umano comunica con la macchina:
    - Monitor
    - Stampanti
    - Tastiera
    - Mouse
    - …
  - Dispositivi che estendono le funzionalità del sistema di elaborazione:
    - SSD
    - Hard disk
    - …

Schema di riferimento





- Ogni **dispositivo di I/O** è collegato al bus di sistema attraverso un'**interfaccia**, anche chiamata **porta**.
- **Ciascun dispositivo procede alla propria velocità e in modo asincrono rispetto alla CPU.**
- L'interfaccia deve contenere:
  - Registri dove poggiare i dati da trasferire/ricevere (DREG).
  - Registri per i comandi (CREG).
  - Registri per lo stato (SREG).

Indirizzi di I/O

- I registri di interfaccia devono essere letti o scritti. Per fare ciò è necessario associare un indirizzo a ciascun registro (come si fa con le celle di memoria).
- Lo spazio degli indirizzi di I/O deve essere *in qualche modo* disgiunto dagli indirizzi di memoria.
- Due possibili modi per indirizzare i registri di I/O:
  - **I/O mapped I/O** (*ingresso/uscita isolato*)
    - Spazio indirizzi di I/O separato dallo spazio di memoria
    - **Il repertorio presenta specifiche istruzioni di I/O**
  - ***Memory mapped I/O***
    - Indirizzi di I/O non distinti da quelli di memoria
    - Una fetta dello spazio di memoria viene dedicata solo e unicamente per i registri di I/O
    - **Si usano le stesse istruzioni di Load e Store usate per leggere e scrivere in memoria**

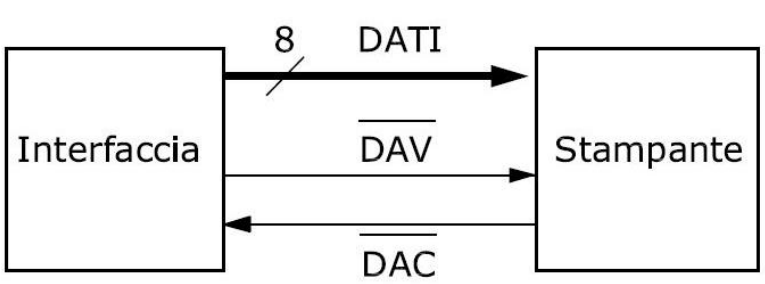
Sincronizzazione delle operazioni di I/O

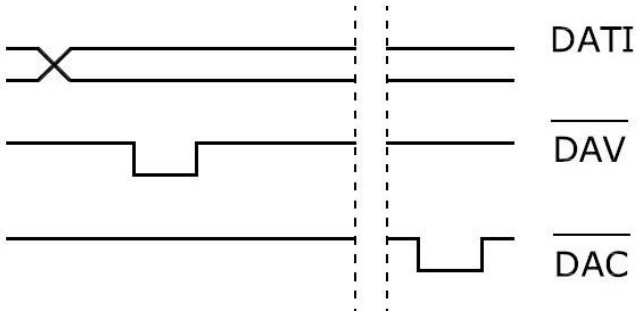
- Le periferiche sono più lente della CPU e hanno diverse frequenze operative (ogni periferica avrà un suo clock).
- È necessario introdurre dei meccanismi di sincronizzazione tra CPU e periferiche in maniera tale da controllare i tempi per trasmettere/ricevere dati.
- Tre tecniche fondamentali:
  - **Gestione a controllo di programma**.
  - **Sotto controllo di interruzione**.
  - **Tramite accesso diretto alla memoria (DMA)**.

Gestione a controllo di programma

Protocollo di handshaking

- Supponiamo di voler trasmettere una sequenza di caratteri alla stampante.
  - La stampante accetta caratteri di 1 byte.
- Si potrebbe pensare di utilizzare un protocollo di comunicazione *hand-shaking* come quelli di seguito:





1. L'interfaccia presenta il dato (carattere da stampare)
2. L'interfaccia asserisce il $\overline{\text{DAV}}$ (dato valido)
3. La stampante risponde con il $\overline{\text{DAC}}$ (dato accettato) ed è qundi pronta per un nuovo dato.

Qui DAV e DAC sono impulsivi (non sempre è così)

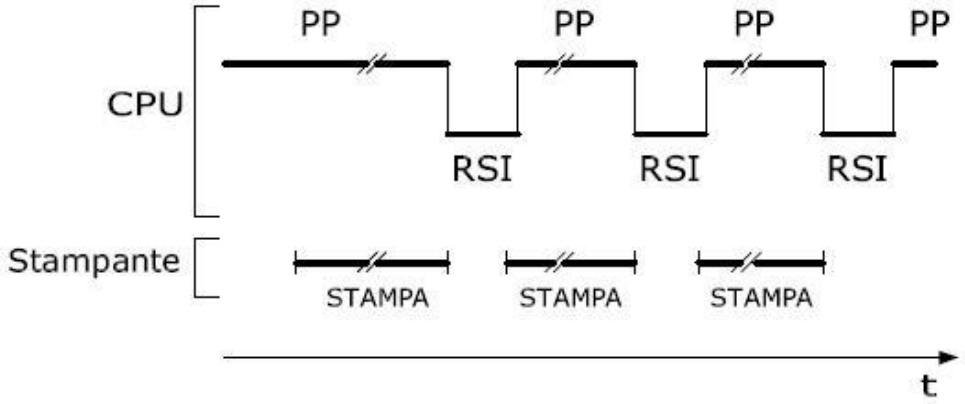
Controllo a gestione di programma

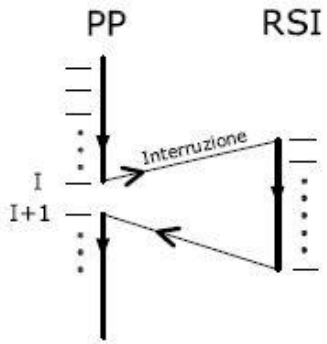
- Si supponga che l'interfaccia della stampante abbia un registro BUSY che
  - BUSY = 0 se la stampante è disponibile ad accettare un nuovo byte (la periferica ha asserito $\overline{\mathrm{DAC}}$)
  - BUSY = 1 se la stampante è occupata
- Se la CPU vuole far stampare una sequenza di caratteri alla stampante allora gestirà la sincronizzazione attraverso un programma che:
  1. Attende che la stampante sia pronta (finché BUSY = 1 attende)
  2. Se la stampante è pronta (BUSY = 0) invia un byte all'interfaccia e poi torna al punto 1
- Durante il punto 1 la CPU interroga ripetutamente l'interfaccia per vedere la stampante è pronta.
  - Questa tecnica è chiamata ***polling***
- ***PROBLEMA:*** la CPU perde un sacco di tempo nell'attesa, quando invece potrebbe fare altro!

Gestione a interrupt

Gestione a interrupt

- Una volta trasmesso il carattere la CPU riprende il programma che aveva sospeso.
- Una volta ricevuto il carattere, la stampante nel frattempo stampa.
- Terminata la stampa, la stampante è pronta a ricevere un nuovo carattere allora l'interfaccia asserisce un segnale di richiesta di interruzione (INTR).
- Ad un segnale di richiesta di interruzione è associata una Routine di Servizio dell'Interruzione (RSI) che contiene le istruzioni per effettuare il trasferimento dati.



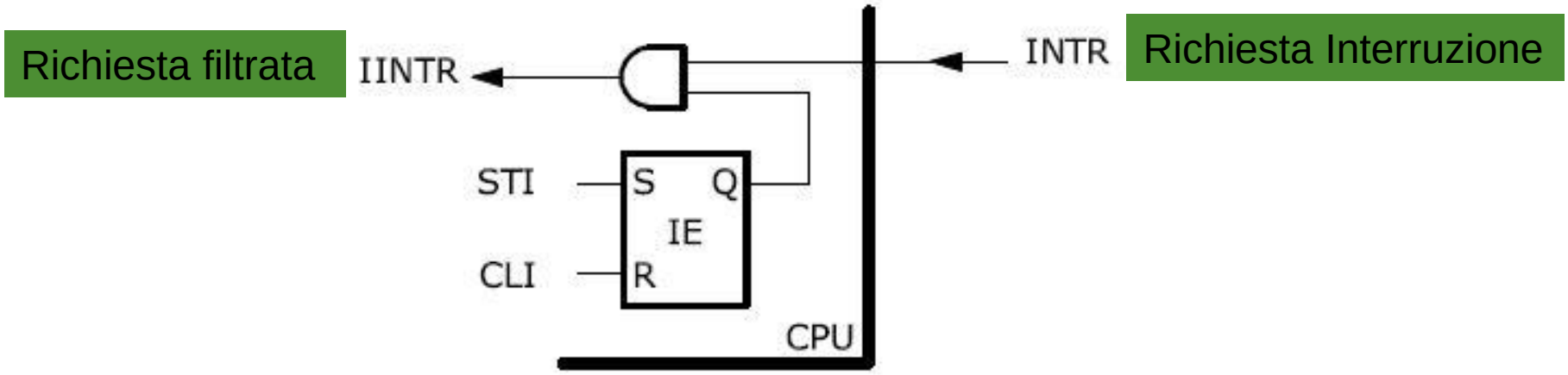


Gestione a interrupt
Driver

- In questo ambito parliamo sempre e solo di interrupt esterni
  - Generati da eventi esterni
  - Asincroni, impredicibili
- La RSI deve essere eseguita in modo *trasparente* rispetto al programma interrotto, nel senso che, al termine della RSI bisogna riprendere il programma da dove era stato interrotto:
  - *Uso dello stack*
- Il **driver** è un programma per la gestione della periferica composto di due parti:
  - Sezione di inizializzazione
  - RSI
- Con la gestione ad interrupt perdo un po' di tempo quando passo dal programma principale alla RSI e poi ritorno, ossia introduco dell'**overhead**.

Interrupt mascherati

- In tutte le macchine si prevede la possibilità di abilitare/disabilitare le interruzioni esterne.
- Si dice che le interruzioni sono *mascherabili*
  - Esistono anche interruzioni non mascherabili, usate per eventi gravi (corrente troppo alta).



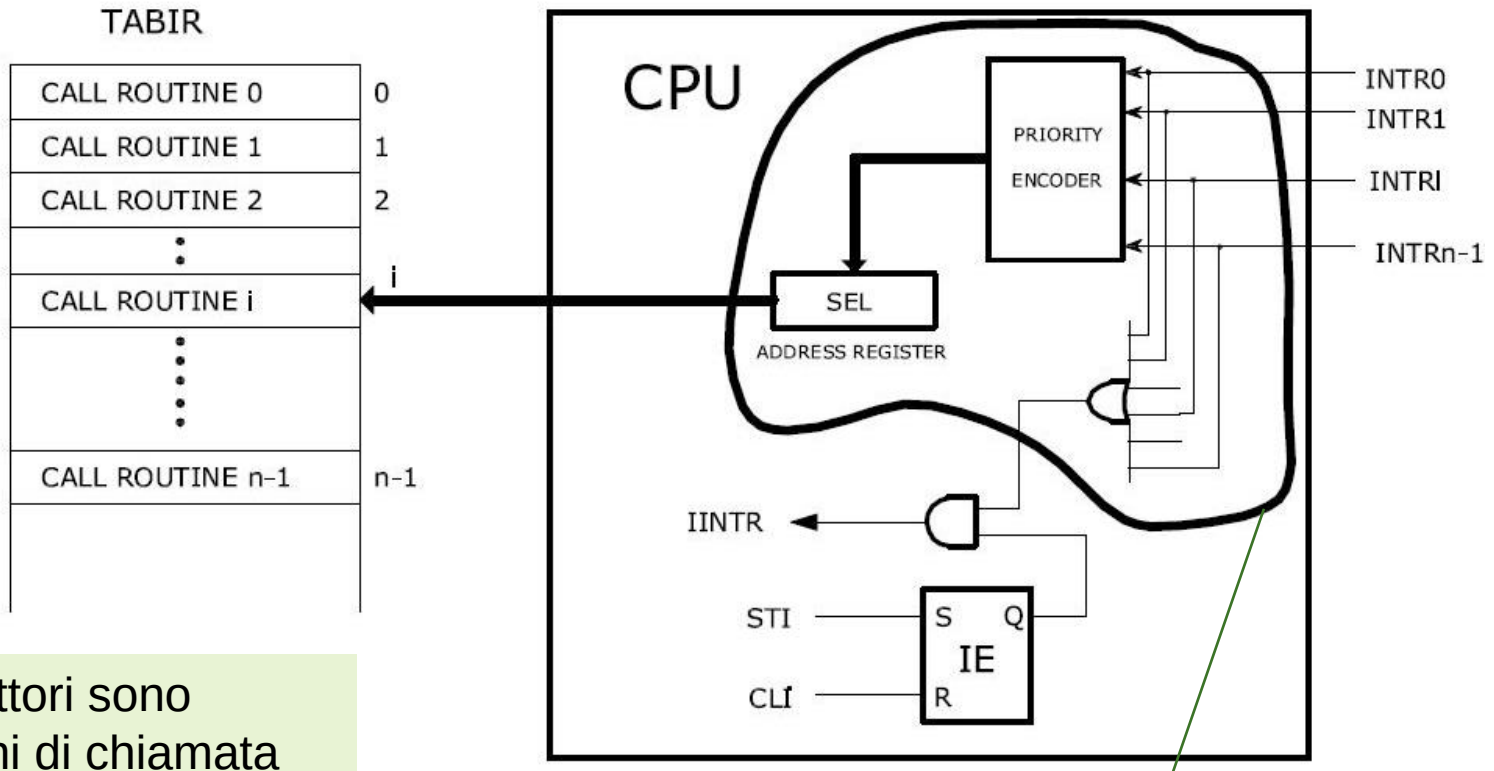
Interruzione da parte di più periferiche

- Finora abbiamo visto il caso con una sola periferica (di output). Cosa succede se ho più periferiche?
- **Problemi**:
  - **Capire chi ha fatto richiesta di interruzione.**
  - **Chiamare la routine di servizio (RSI) corretta.**
  - **Stabilire la priorità delle periferiche.**
  - Una routine può essere interrotta a sua volta da un altro interrupt esterno (caso di routine annidate)
- Occorre un meccanismo per attivare la routine opportuna e per tornare al programma interrotto
  - Ogni architettura ha il suo
- Una delle tecniche più utilizzate è *la discriminazione tramite* ***vettorizzazione*** *delle interruzioni esterne*.

Vettorizzazione

- Ci sono molteplici tecniche di vettorizzazione relative a differenti architetture.
- In generale, si ha una tabella (TABIR) in memoria contenente i vettori di interruzione
  - L'interruzione fa generare un selettore che individua il vettore in TABIR.
  - Un vettore fa saltare alla routine associata.
- Due tipi diversi di vettore:
  - istruzione di chiamata alla RSI;
  - Indirizzo (dell'istruzione da eseguire per attivare la RSI)

Schema di principio della vettorizzazione



Qui i vettori sono istruzioni di chiamata

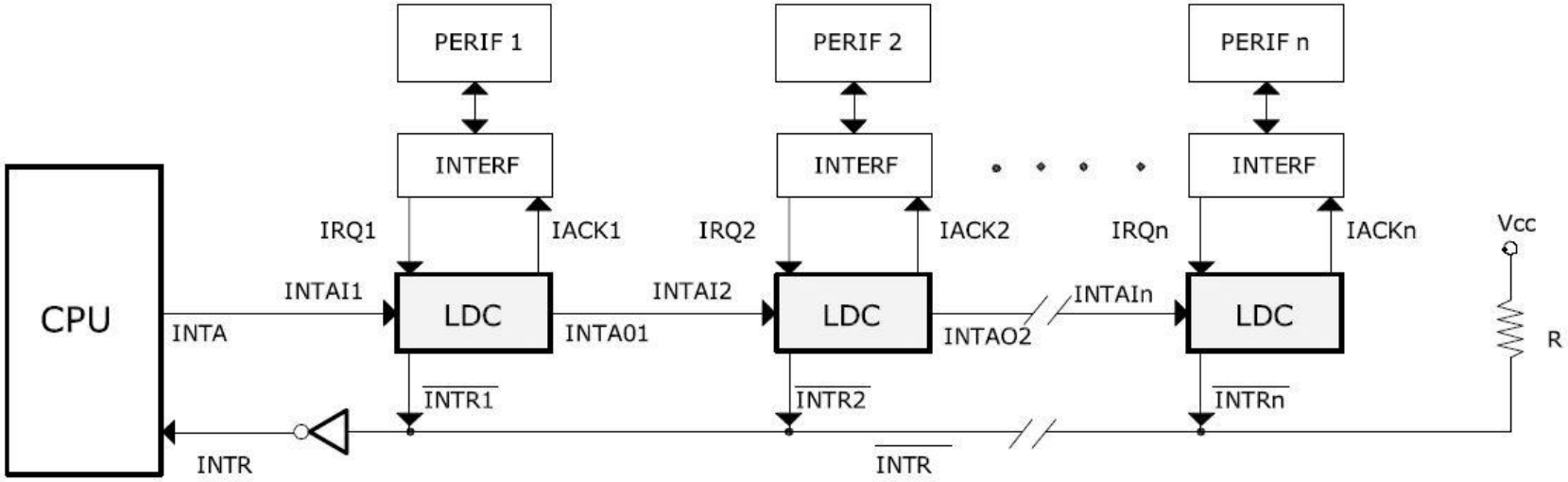
Concettualmente questo blocco può essere portato all'esterno a costituire un controllore distinto

Daisy chain

- Una catena di periferiche nella quale la priorità delle unità collegate è determinata dalla loro posizione.
- Funzionamento:
  1. In risposta alla richiesta di interruzione la CPU esegue il ciclo di INTA; il segnale INTA appare all'interfaccia più a sinistra.
  2. Se l'interfaccia sta asserendo la richiesta di interruzione la relativa logica di controllo (LDC) non lascia propagare il segnale a valle e asserisce verso la periferica il segnale IACK (interruzione riconosciuta).
  3. Se l'interfaccia non ha asserito la richiesta di interruzione, LDC lascia passare il segnale a destra.
  4. Il segnale si propaga fino ad incontrare la prima interfaccia che ha asserito la richiesta di interruzione.
- Più la periferica è vicina alla CPU, più è alta la sua priorità.

Daisy chain

Vettorizzazione senza un controllore esterno



Accesso diretto alla memoria

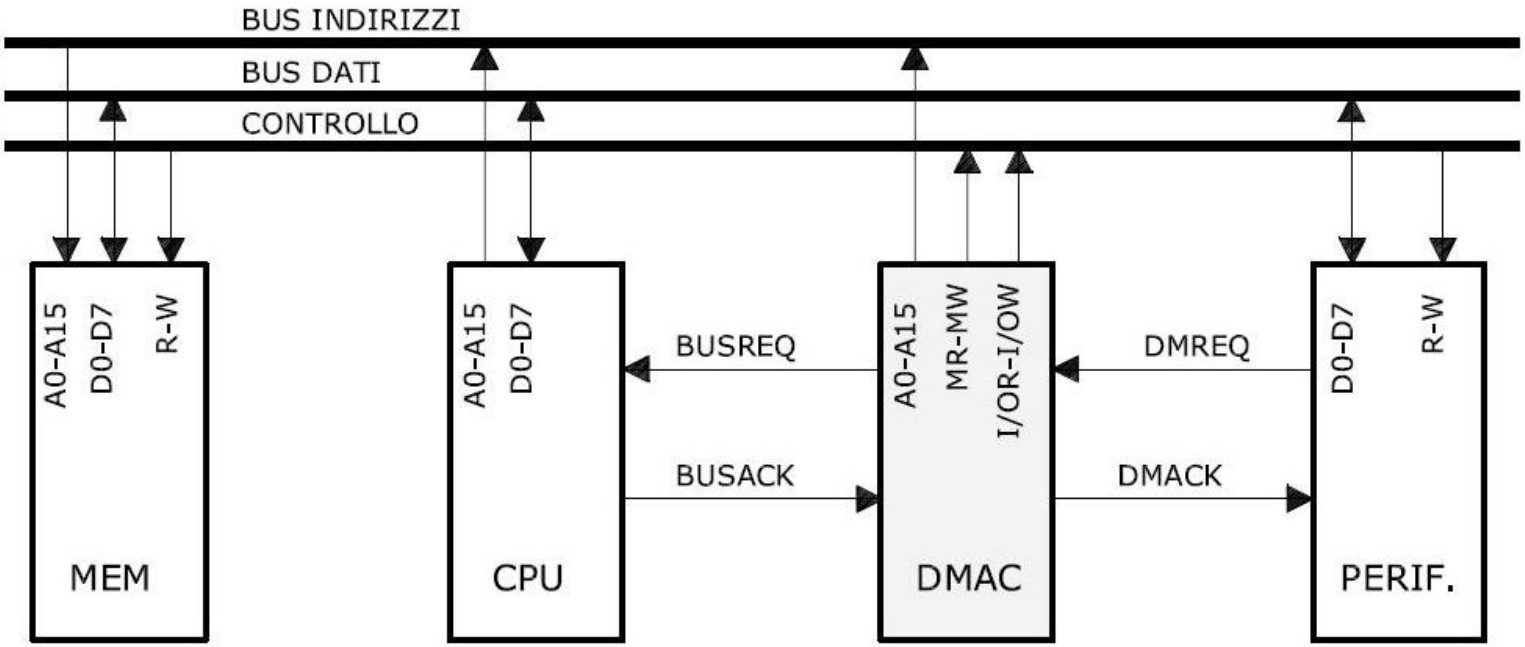
Accesso diretto alla memoria

- La gestione a interrupt è molto più efficiente della gestione a programma…
- …tuttavia a ogni interruzione è necessario eseguire delle istruzioni di inizializzazione per salvare e ripristinare i registri CPU, mettere le istruzioni nello stack, ecc. (***tempo di overhead***)
- Se la periferica è molto veloce è possibile che la frequenza delle interruzioni sia così alta da non permettere l'esecuzione della routine di servizio all'interruzione.
- Per questo è stata introdotta la tecnica dell'accesso **diretto alla memoria (Direct Memory Access, DMA)**.
  - **Permette il trasferimento diretto dei dati tra memoria e periferiche (senza passare per la CPU).**
- Il dispositivo che si occupa del trasferimento dei dati è chiamato **DMA Controller**.
  - Mentre il DMAC gestisce il trasferimento la CPU fa altro lavoro.

Protocollo per il controllo del bus da parte del DMAC

- Il bus (dati, indirizzi e controllo) sono normalmente gestiti dalla CPU. Per permettere il trasferimento dati, il DMAC *deve prendere il controllo del bus*
- Protocollo utilizzato:
  1. L'interfaccia della periferica asserisce DMREQ per indicare che è disponibile ad inviare/trasmettere un dato.
  2. Il DMAC asserisce BUSREQ, richiedendo di entrare in controllo del bus.
  3. La CPU risponde asserendo BUSACK. La CPU da ora in poi non controlla più il bus (i collegamenti sono in alta impedenza) fino a quando BUSREQ è alto.
  4. Il DMAC può pilotare il bus e risponde alla periferica con DMACK e gestisce il trasferimento dati.
  5. Finito il trasferimento BUSREQ viene disattivato e il bus torna al controllo della CPU che disattiva BUSACK.

Direct Memory Access



- **DMREQ: DMA Request**, richiesta di trasferimento da parte del periferico;
- **BUSREQ: Bus Request**, richiesta di entrare in possesso del bus da parte del DMAC alla CPU;
- **BUSACK: Bus Ackowledgement**, risposta di richiesta accolta da parte della CPU;
- **DMACK: DMA Acknowledgement**, risposta di richiesta accolta da parte del DMAC all'interfaccia della periferica

Direct Memory Access Controller

- L'architettura di un DMAC prevede:
  - Un contatore del numero di byte da trasferire
  - Un (registro) puntatore alla posizione di memoria in cui verrà letto/scritto il prossimo dato.
  - Un registro di comando che indichi il tipo di trasferimento.
  - Un registro di stato.
- Il DMAC prevede due fasi principali di funzionamento:
  - **Fase di programmazione:** la CPU trasmette al DMAC le informazioni e le modalità per il trasferimento (informazioni scritte sui registri del DMAC). Il DMAC in questa fase viene visto come se fosse un'interfaccia di una periferica.
  - **Fase di trasferimento dati**
- Esistono due modalità di trasferimento:
  - **Trasferimento singolo** (*cycle stealing*), il DMAC occupa il bus solo per uno o pochi cicli di clock.
  - **Trasferimento a blocchi** (*burst*), il DMAC occupa il bus per tutto il tempo necessario per trasferire il numero di byte definito nel contatore in fase di programmazione.

Accesso diretto alla memoria
Vantaggi e svantaggi

- Vantaggi:
  - La CPU fa altro mentre il trasferimento dati è gestito dal DMAC
- Svantaggi:
  - Serve un componente in più (il DMAC)
  - La CPU mentre il DMAC controlla il bus non può fare operazioni che richiedono il suo utilizzo (ad es. accesso alla memoria).

Domande?

Riferimenti principali

- Capitolo 5 di ***Calcolatori elettronici. Architettura e Organizzazione,*** Giacomo Bucci. McGraw-Hill Education, 2017.