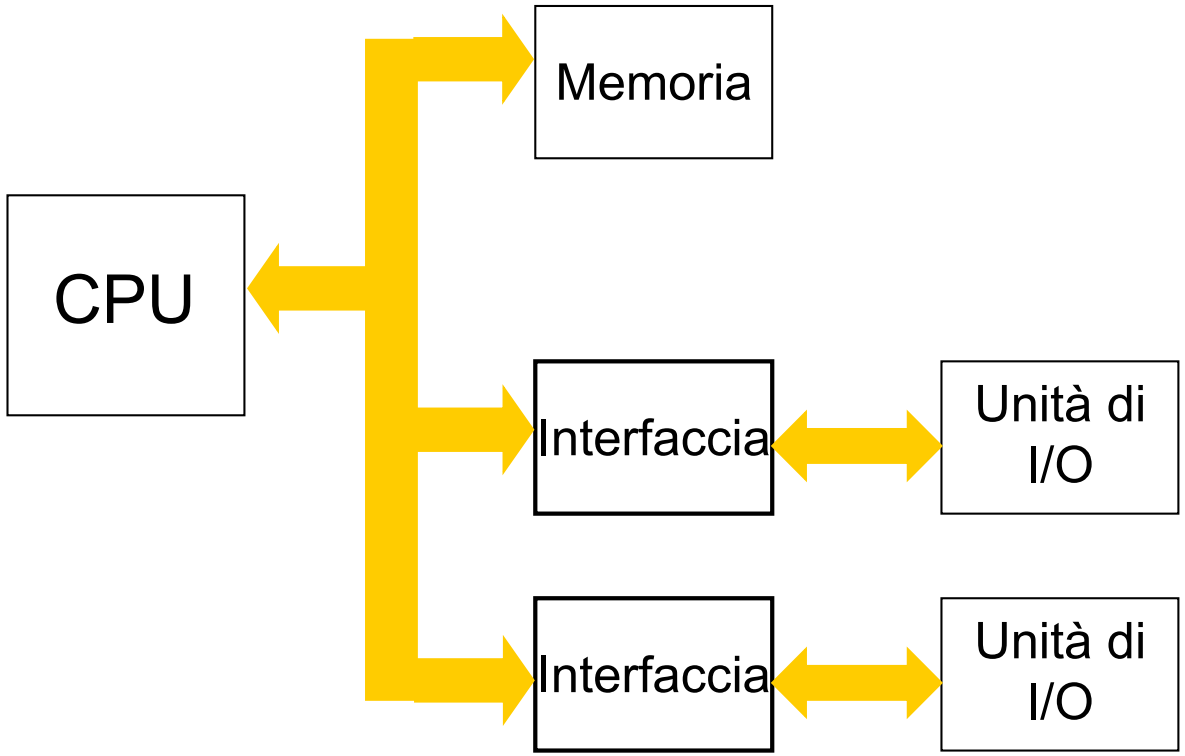
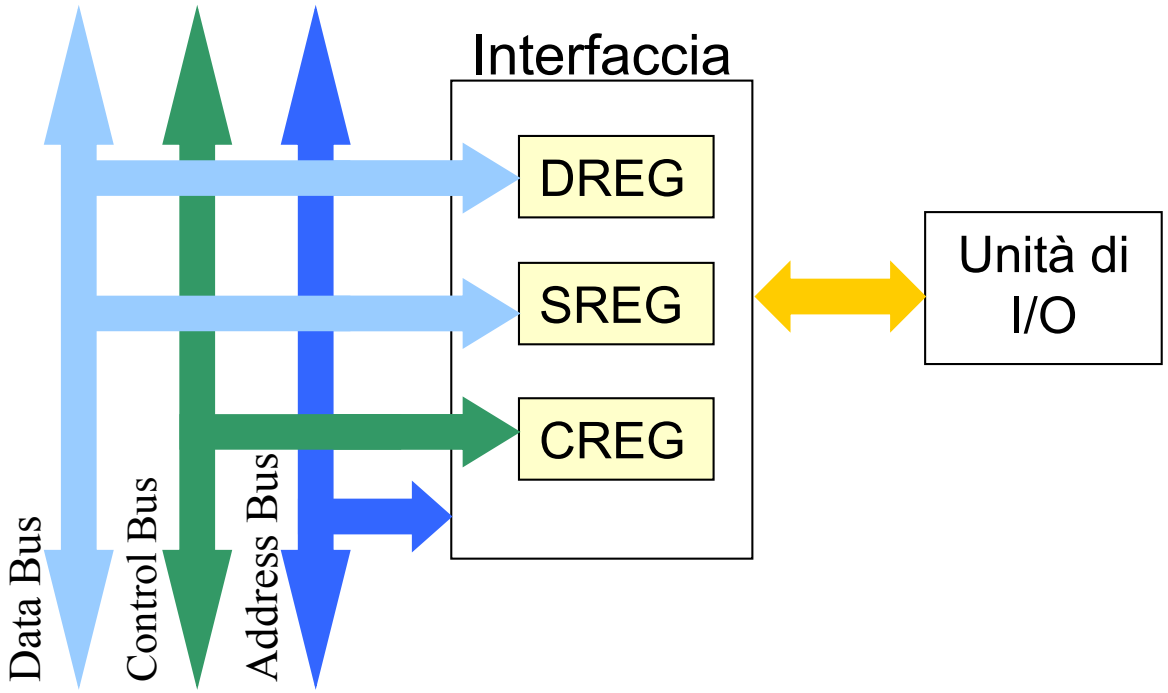
Il sistema di I/O

Architettura a bus singolo



Interfaccia

- Svolge la funzione di adattamento sia elettrico sia logico tra le unità periferiche e il calcolatore



DREG: Reg. dati da trasferire

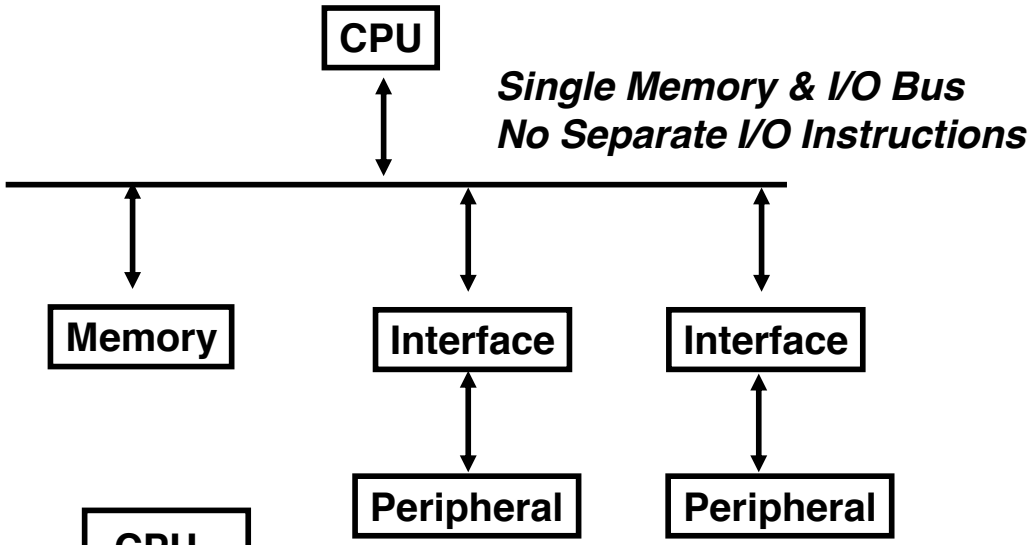
CREG: Reg. comandi alla periferica

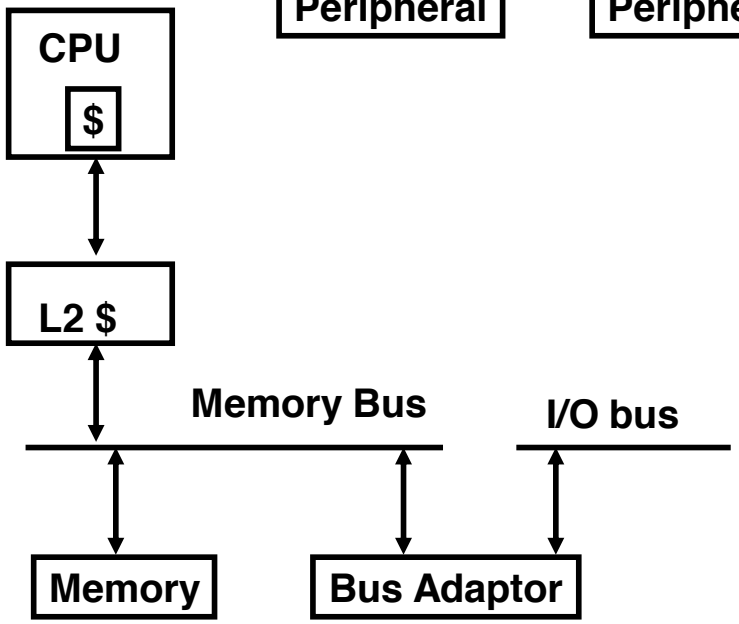
SREG: Reg. di stato della periferica

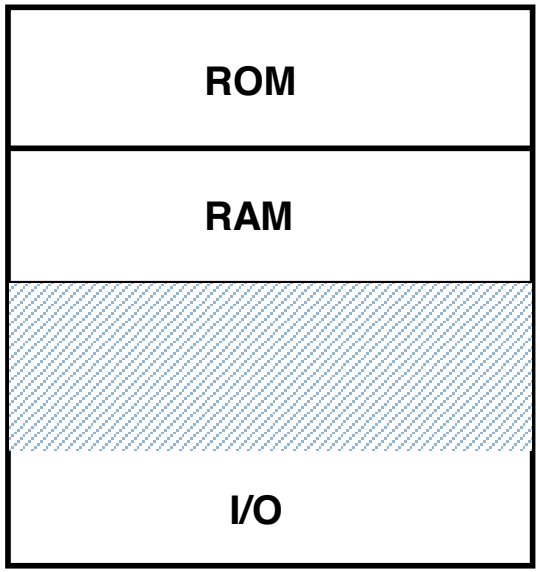
Types of bus-based I/O: memory-mapped I/O and standard I/O

- Processor talks to both memory and peripherals using same bus – two ways to talk to peripherals
  - Memory-mapped I/O
    - Peripheral registers occupy addresses in same address space as memory
    - e.g., Bus has 16-bit address
      - lower 32K addresses may correspond to memory
      - upper 32k addresses may correspond to peripherals

Memory Mapped I/O



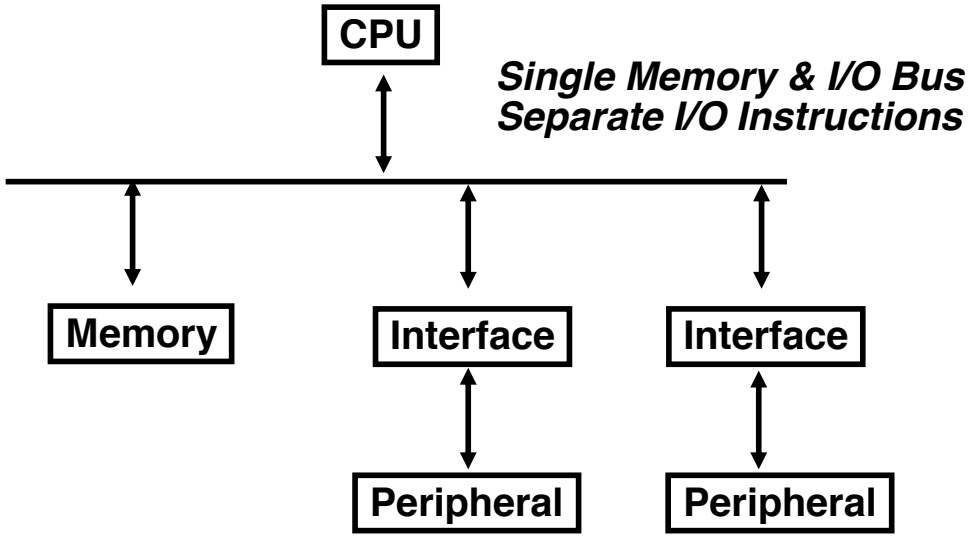


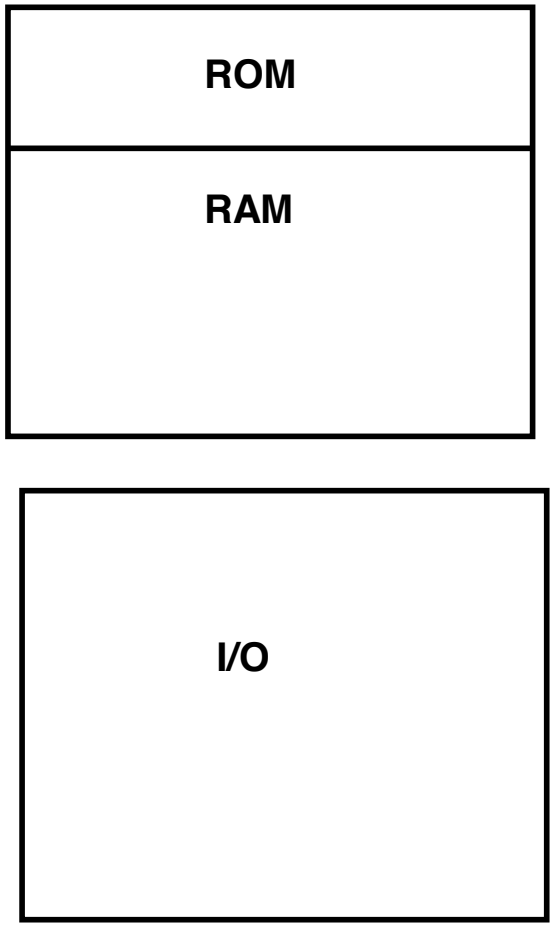


Types of bus-based I/O: memory-mapped I/O and standard I/O

- Standard I/O (I/O-mapped I/O)
  - Additional pin (*M/IO*) on bus indicates whether a memory or peripheral access
  - e.g., Bus has 16-bit address
    - all 64K addresses correspond to memory when *M/IO* set to 0
    - all 64K addresses correspond to peripherals when *M/IO* set to 1

Standard I/O (I/O-mapped I/O)





Memory-mapped I/O vs. Standard I/O

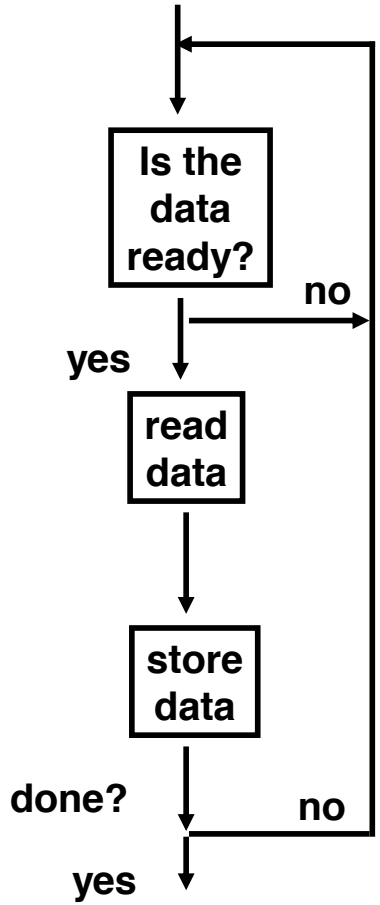
- Memory-mapped I/O
  - Requires no special instructions
    - Assembly instructions involving memory like MOV and ADD work with peripherals as well
    - Standard I/O requires special instructions (e.g., IN, OUT) to move data between peripheral registers and memory
- Standard I/O
  - No loss of memory addresses to peripherals
  - Special-purpose I/O instructions
  - Intel x86 provides `in`, `out` instructions.
    - (Es. IN AL, port e OUT port, AL with port= interface address)
  - Simpler address decoding logic in peripherals possible
    - When number of peripherals much smaller than address space then high-order address bits can be ignored
      - smaller and/or faster comparators

Tecniche di gestione delle unità periferiche

- Controllo di programma diretto
- Controllo di programma con polling
- Controllo di interruzione
- Accesso diretto alla memoria

Controllo di programma diretto

- In questo caso la gestione dei dispositivi di I/O è totalmente demandata alla CPU.
- Se un programma ha bisogno di eseguire un'operazione di I/O
  1. La CPU verifica che il dispositivo è pronto al trasferimento (***ready_bit*** = '1')
  2. Se il dispositivo è pronto, la CPU esegue il trasferimento, altrimenti ripete la verifica
- Viene normalmente utilizzato nei sistemi più piccoli e meno complessi, in quanto ha le seguenti caratteristiche:
  - poco costoso in termini di hardware
  - poco efficiente, la CPU può restare per molti cicli in attesa (busy waiting)



Keyboard Input

- Receiver control (at $FFFF0000_{16}$)
  - Bit 0 is the 'ready' bit – unread input from keyboard
- Receiver data (at $FFFF0004_{16}$) contains last pressed key
  - Reading data register resets 'ready' bit

```
        addi $t0, 0xffff0000, 0
rd_poll: lw $v0, 0($t0)
        andi $v0, $v0, 0x01
        beq $v0, $zero, rd_poll
        lw $v0, 4($t0)
```

Read from control register
AND off the 'ready' bit
If zero, no keypress Keep waiting
Else read data register

Character Display

- Transmitter control (at $FFFF0008_{16}$)
  - Bit 0 is the 'ready' bit – display accepting new character
- Transmitter data (at $FFFF000C_{16}$) takes ASCII character
  - Writing data register resets 'ready' bit

```
# character in $a0
                addi $t0, 0xffff0008, 0
wr_poll:        lw $v0, 0($t0)          Read from control register
                andi $v0, $v0, 0x01     AND off the 'ready' bit
                beq $v0, $zero, wr_poll If zero, not ready    Keep waiting
                sw $a0, 4($t0)          Else send to data register
```

Controllo di programma con polling

- E' basato sulla scansione periodica del ***ready_bit*** di tutti i dispositivi per verificare se qualcuno di essi richiede un servizio
- La gestione del polling viene fatta attraverso un ciclo software di lettura dello stato dei dispositivi di I/O.
- La maggior parte del tempo è impiegata dal programma principale nell'esecuzione del ciclo di polling.

Limiti del polling

- Ogni dispositivo deve dipendere dalla CPU per essere servito. Ne consegue che:
  - il tempo che nel caso peggiore il dispositivo deve attendere prima che la CPU esegua il trasferimento richiesto può essere alto, e dipende da quanti dispositivi di I/O sono connessi
  - tutti i dati devono passare attraverso la CPU, e non esiste connessione diretta tra dispositivo e memoria
  - la CPU dedica una parte del suo tempo ad eseguire banali operazioni di test e trasferimento dati.

Costo del Polling?

Assumiamo che per un processore a 500 MHz siano richiesti 400 cicli di clock per l'operazione di polling.

- Mouse: testato 30 volte/sec per non perdere i movimenti dell'utente
  - Polling Clock/sec= 30*400=12000 clock/sec
  - %Processore per polling = $12*10^3/(500*10^6)$= 0,002% *Impatto limitato*

- Floppy: trasferimento dati di 2 byte alla velocità di 50 KB/sec per non perdere dati
  - Polling Clock/sec=50KB/2*400=10,000,000 clock/sec
  - %Processore per polling= $10*10^6/(500*10^6)$= 2% *Basso*

- Hard disk: trasferimento di blocchi di 16 byte alla velocità di 8MB/sec per non perdere dati
  - Hard Disk Polling Clock/sec=8MB/16*400=$200*10^6$ clock/sec
  - %Processore per polling= $200*10^6/(500*10^6)$= 40% *Inaccettabile*

Un alternativa al polling: il sistema di interruzione

È il dispositivo di I/O che comunica al processore il suo stato di pronto.

Il processore, presumibilmente impegnato nella esecuzione di una sequenza di istruzioni "utili", la interrompe temporaneamente per eseguire la sequenza prevista dal protocollo del I/O

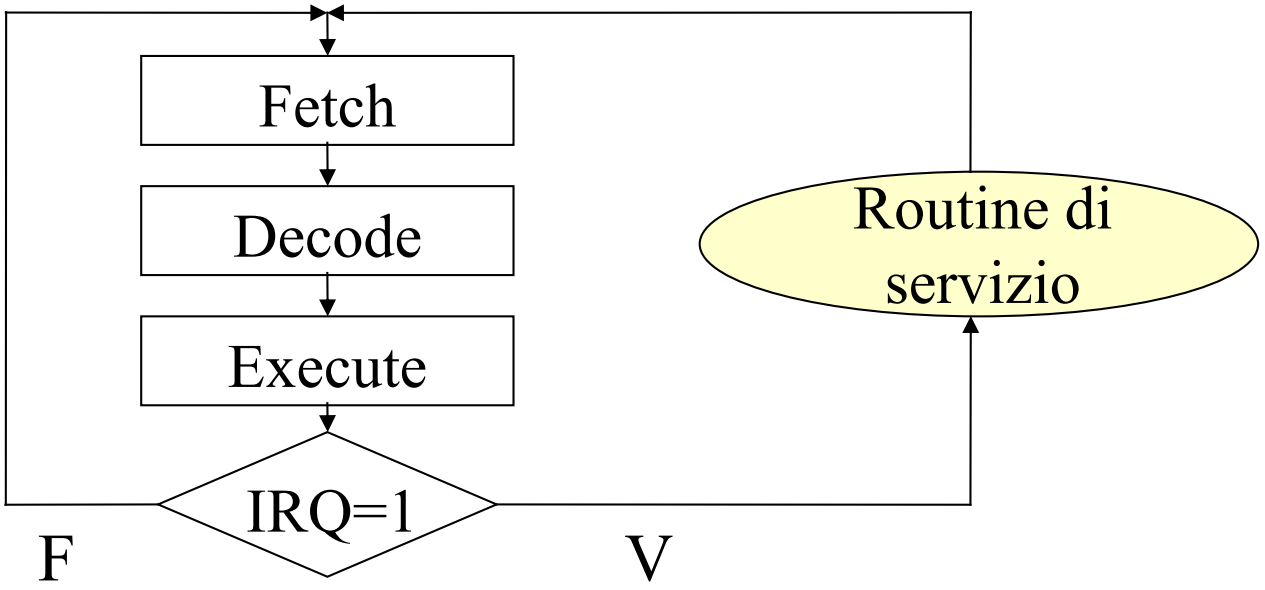
Tale meccanismo viene realizzato dal ***sistema di interruzione***

Il segnale inviato dal dispositivo di I/O (IRQ) prende il nome di ***richiesta di interruzione.***

La sequenza di istruzioni che il processore esegue in seguito ad una interruzione viene detta ***routine di servizio.***

Sistema di interruzione

- La richiesta di interruzione di un dispositivo di I/O è asincrona rispetto all'esecuzione delle istruzioni
  - Non è associata ad alcuna istruzione e può essere attivata durante l'esecuzione di ogni istruzione
  - Viene valutata solo alla fine dell'esecuzione di ogni istruzione



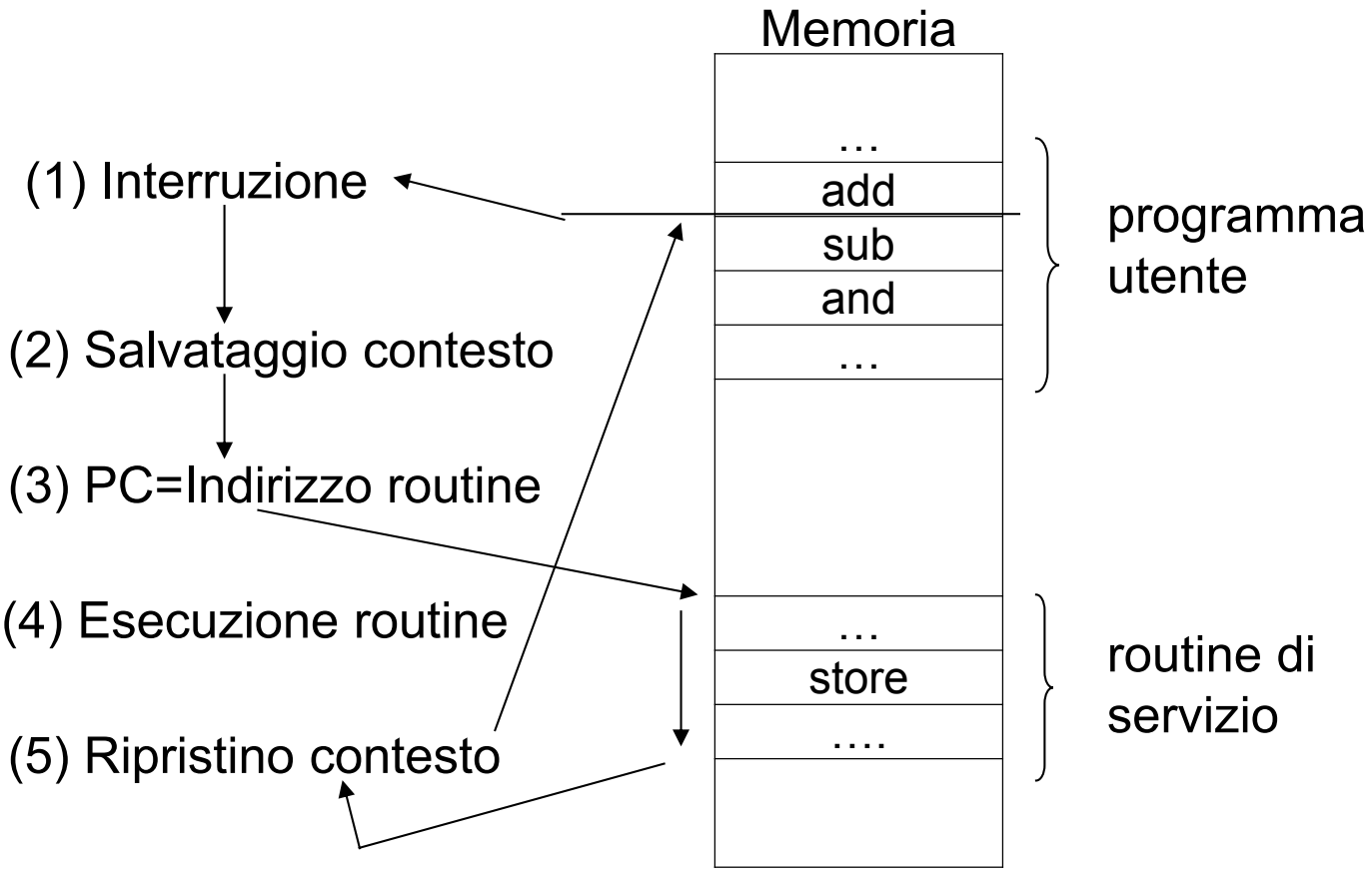
Funzioni del sistema di interruzione

F1. Deve garantire che una interruzione non provochi interferenze sul programma interrotto.

F2. È necessario che il sistema di interruzione riconosca il dispositivo interrompente

F3. Deve provvedere alla gestione delle priorità delle richieste di interruzioni

Cambio di contesto



Salvataggio del contesto

- **Contesto:** Program Counter (PC), Registro di Stato (SR), Registri di uso generale

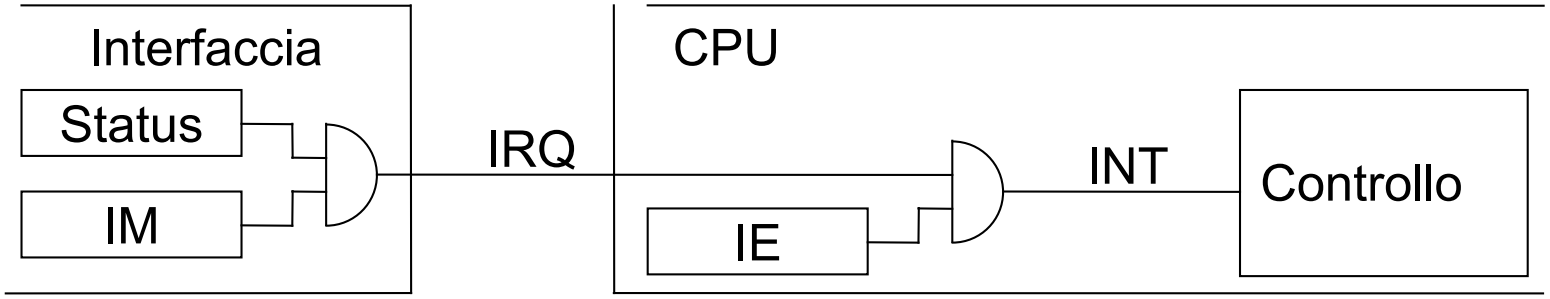
- PC e SR devono essere salvati via hardware nello stack.
  - MEM[SP]=SR; SP--;
  - MEM[SP]=PC; SP--;

- Gli altri registri possono essere salvati via software nello **stack** (PUSH $i)
  - Vengono salvati solo i registri che verranno utilizzati.
  - Questo compito è demandato alla routine di servizio che lo svolge nel suo preambolo.

Salvataggio del contesto

- Il salvataggio del contesto deve essere non interrompibile per evitare situazioni anomale
  - Il processore viene dotato di un flag IE indicante la interrompibilità del processore.
  - Nel momento in cui viene accettata la richiesta di interruzione IE viene resettato e il processore diventa non interrompibile
  - Per rendere nuovamente interrompibile il processore bisogna usare un'apposita istruzione

Un sistema di interruzione



IM=Interrupt Mask

- Quando il dispositivo è pronto, pone STATUS=1
- Se IM=1 e le interruzioni sono abilitate (IE=1) viene servita la richiesta di interruzione (al termine dell'istruzione corrente).
  - Viene posto IE=0
  - Viene salvato il contesto
  - PC=Indirizzo della Routine di servizio

Ripristino del contesto

- Via software, nell'epilogo della routine di servizio, vengono ripristinati i valori dei registri salvati nello stack (POP)
- L'uscita dalla routine di servizio avviene mediante un'apposita istruzione di ritorno da interruzione (RTI) che ripristina la parte di contesto salvata via hardware
  - SR=MEM[SP]; SP++;
  - PC=MEM[SP]; SP++;
- In alcuni processori la RTI riabilita anche il flip flop IE (IE=1), in altri è necessario utilizzare un'apposita istruzione;

Benefici dell'I/O mediante sistema di interruzione

- Supponiamo che siano richiesti 500 cicli di clock (a 500 MHz) per ogni trasferimento, compreso l'interrupt.
- Trovare la % del tempo consumato dal processore se lo hard disk è attivo il 5% del tempo.
- Interrupt rate = polling rate
  - Disk Interrupts/sec = 8 MB/s /16B
    = 500K interrupts/sec
  - Disk Polling Clocks/sec = 500K * 500
    = 250,000,000 clocks/sec
  - % Processor for during transfer: $250*10^6/(500*10^6) = 50\%$
- Disk active 5% $\Rightarrow$ 5% * 50% $\Rightarrow$ 2.5% busy

Interruzioni non mascherabili

- Il flip-flop IE serve a mascherare le richieste di di interruzioni.
- Quasi tutti i calcolatori sono dotati di una linea di interruzione non mascherabile (NMI) che viene sempre rilevata se asserita.
- È impiegata per gestire situazioni di emergenza (tipicamente la caduta di tensione).
- Questo tipo di interruzione deve essere prioritaria rispetto ad altre richieste

Riconoscimento del dispositivo di I/O

- In un calcolatore possono essere collegati diversi dispositivi ciascuno caratterizzato da un propria routine di servizio
- È necessario che il sistema di interruzione riconosca il dispositivo interrompente per attivare la relativa routine
- Approcci possibili:
  - via software (*polling*): tramite una sequenza di istruzioni viene individuato il dispositivo interrompente e viene mandata in esecuzione la relativa routine
  - via hardware (*interruzioni vettorizzate*): il dispositivo interrompente invia il codice identificativo

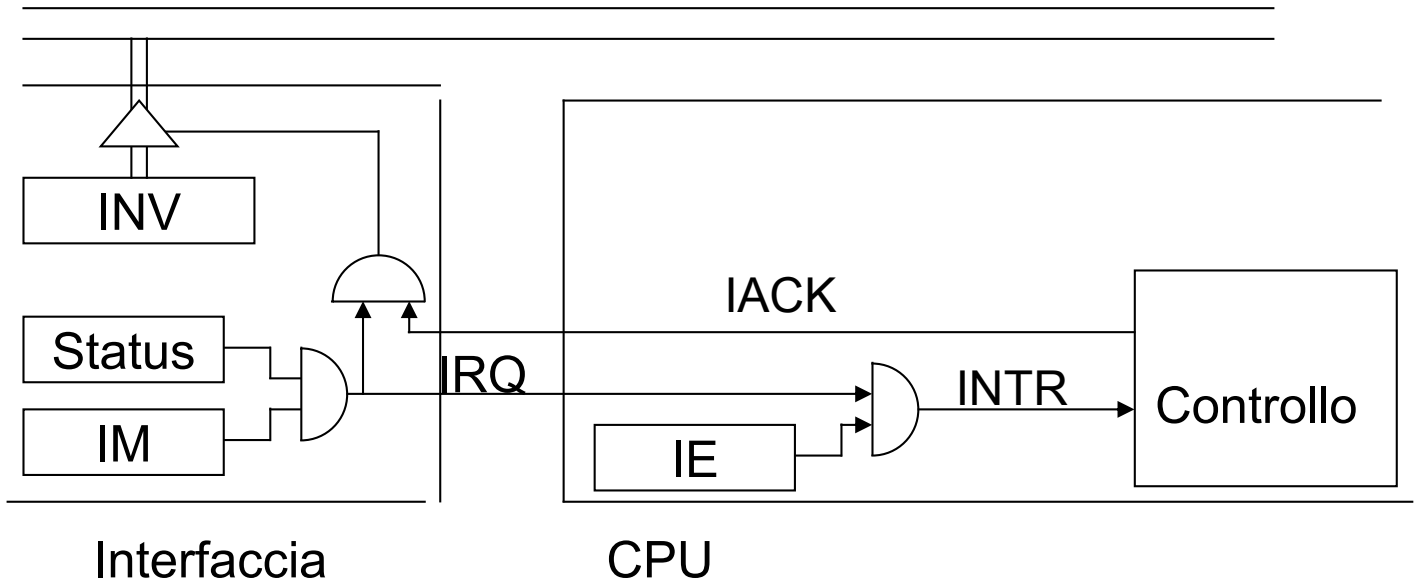
Riconoscimento mediante polling

- In seguito alla richiesta di interruzione:
  1. viene salvato il contesto del programma interrotto
  2. Viene mandata in esecuzione una sequenza di identificazione dell'interruzione che interroga ad uno ad uno i dispositivi fino a trovare quello che ha fatto la richiesta
  3. Viene mandata in esecuzione la routine individuata al passo 2.

- Nel caso in cui più richieste di interruzione sono presenti, viene servita quella che per prima viene incontrata nella sequenza.

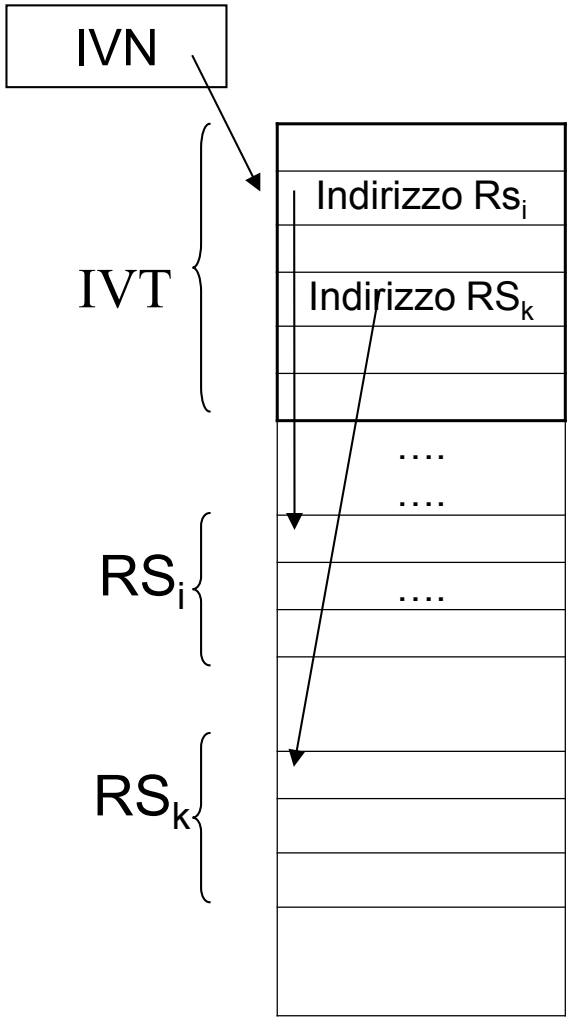
Riconoscimento mediante vettore delle interruzioni

- Nell'interfaccia è presente un registro in cui è memorizzato il codice identificativo del dispositivo (INV).
- In seguito alla richiesta di interruzione IRQ=1, se il processore è interrompibile (IE=1), attiva in risposta il segnale IACK.
- Il codice identificativo viene inviato sul bus dati in risposta al segnale IACK generato dal processore.

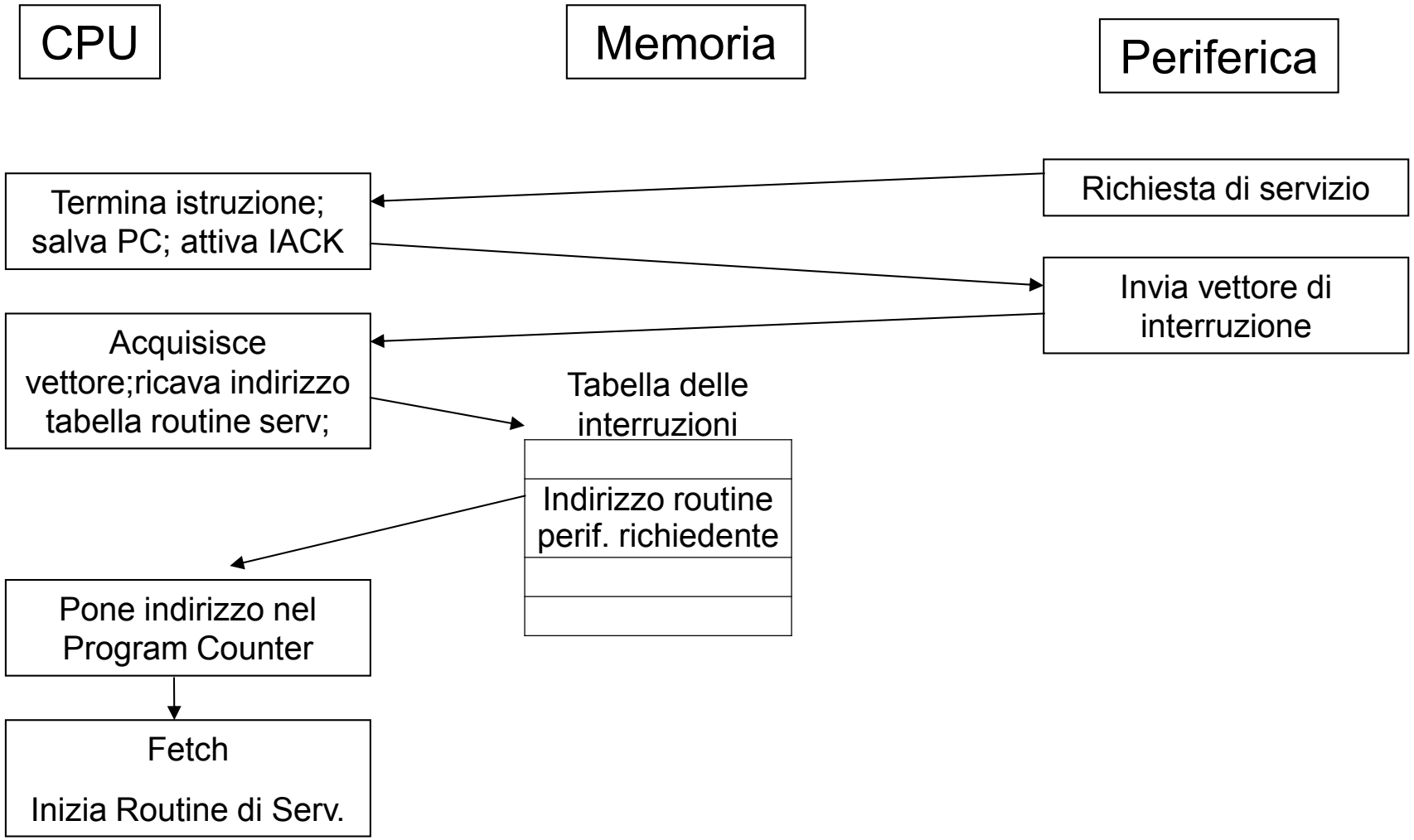


Riconoscimento mediante vettore delle interruzioni

- In memoria è presente una tabella, ***Interrupt Vector Table (IVT)***, che contiene gli indirizzi delle routine di servizio dei dispositivi.
- Il codice inviato dal dispositivo di I/O, ***Interrupt Vector Number (IVN)***, rappresenta l'indice della tabella corrispondente alla routine di servizio (ISR).
- La IVT di norma è memorizzata a partire dalle prime posizioni della memoria in modo da codificare l'IVN con pochi bit.
- Il riempimento della IVT (o di parte di essa) viene eseguita ad opera del programmatore



Riconoscimento mediante vettore delle interruzioni

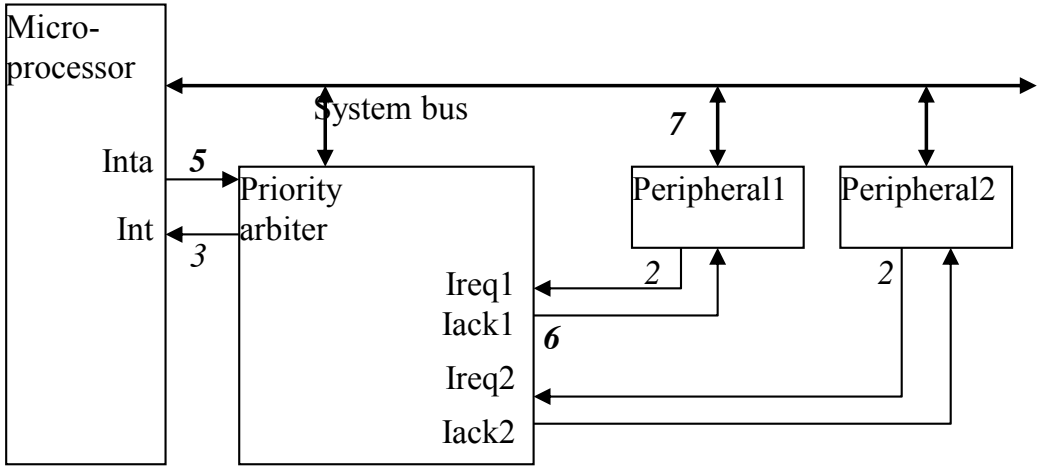


Gestione dei conflitti tra richieste di interruzione

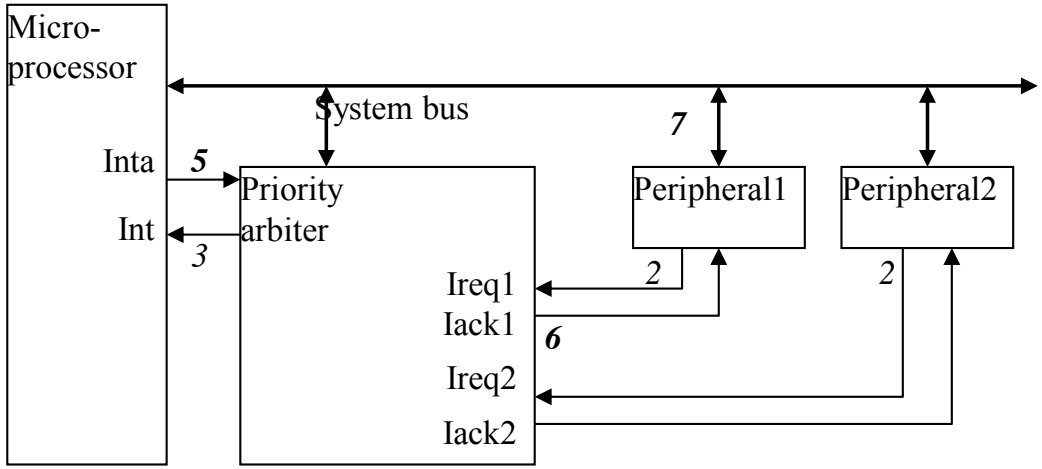
- Più richieste di interruzioni possono entrare in conflitto
- Cause conflitto:
  - Richieste contemporanee
  - Richieste quando è in esecuzione la routine di servizio di una interruzione

Arbitration: Priority arbiter

- Consider the situation where multiple peripherals request service from single resource (e.g., microprocessor) simultaneously - which gets serviced first?
- Priority arbiter
  - Single-purpose processor
  - Peripherals make requests to arbiter, arbiter makes requests to resource
  - Arbiter connected to system bus for configuration only



Arbitration using a priority arbiter

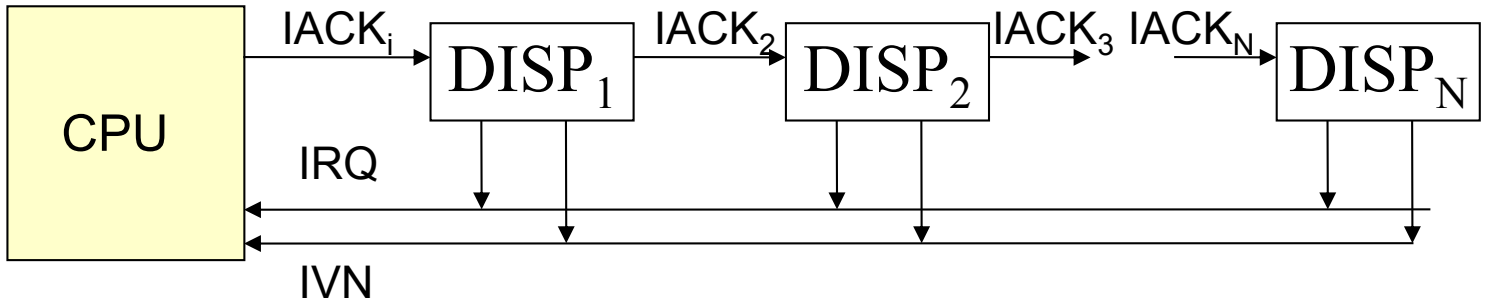


1. 1. Microprocessor is executing its program.
2. 2. Peripheral1 needs servicing so asserts *Ireq1*. Peripheral2 also needs servicing so asserts *Ireq2*.
3. 3. Priority arbiter sees at least one *Ireq* input asserted, so asserts *Int*.
4. 4. Microprocessor stops executing its program and stores its state.
5. 5. Microprocessor asserts *Inta*.
6. 6. Priority arbiter asserts *Iack1* to acknowledge Peripheral1.
7. 7. Peripheral1 puts its interrupt address vector on the system bus
8. 8. Microprocessor jumps to the address of ISR read from data bus, ISR executes and returns
9. (and completes handshake with arbiter).
10. 9. Microprocessor resumes executing its program.

Arbitration: Priority arbiter

- Types of priority
  - Fixed priority
    - each peripheral has unique rank
    - highest rank chosen first with simultaneous requests
    - preferred when clear difference in rank between peripherals
  - Rotating priority (round-robin)
    - priority changed based on history of servicing
    - better distribution of servicing especially among peripherals with similar priority demands

Interruzione vettorizzata con Daisy chain



- In seguito ad richiesta di interruzione IRQ, la CPU attiva il segnale IACK che viene inviato ai dispositivi mediane un'unica linea.
- Se un dispositivo DISP$_i$ ha richiesto una interruzione non propaga il segnale IACKi, ponendo IACK$_{i+1}$=0
- Se il dispositivo DISPi non ha fatto alcuna richiesta propaga il segnale in ingresso ponendo IACK$_{i+1}$=1
- **Problemi:**
  - la priorità dei dispositivi dipende dalla posizione nella daisy chain: i più vicini alla CPU hanno una più alta priorità statica rispetto a quelli che seguono.
  - Il guasto su una periferica inibisce l'accesso delle altre a valle

Gerarchia di priorità: interruzione di una routine di servizio

- Approccio software
  - Nel preambolo della routine di servizio attivata, mentre la CPU è non interrompibile, oltre a salvare il contesto vengono definiti i dispositivi di I/O che hanno priorità superiore e che possono interrompere la routine corrente.
  - I dispositivi di I/O vengono abilitati settando i rispettivi flip-flop IM
- Il principale problema è il tempo perso per settare i bit dei dispositivi di I/O

Gerarchia di priorità: interruzione di una routine di servizio

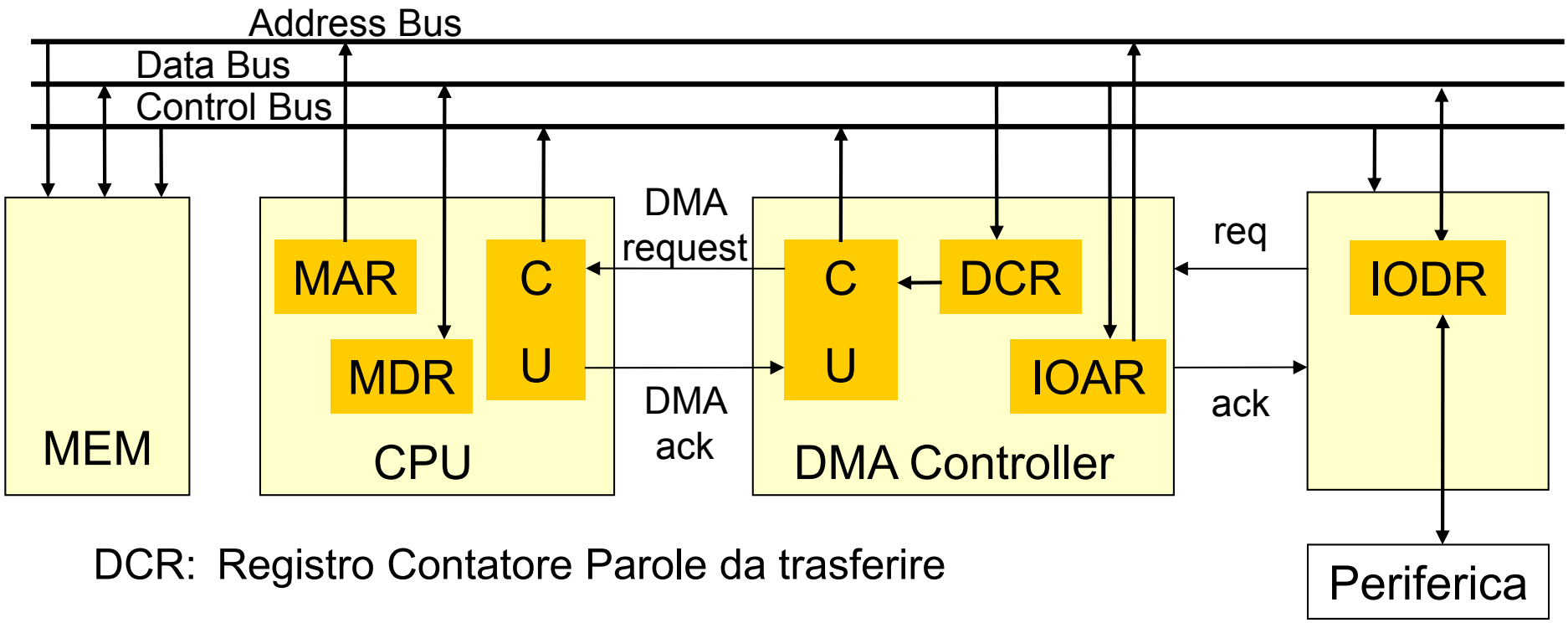
Approccio hardware

- Ad ogni dispositivo di I/O è associata un livello di priorità
- Il processore in ogni istante è caratterizzato da un livello di priorità corrente NPR
- Quando si verifica una richiesta di interruzione prodotta da un dispositivo con priorità K, NPR assumerà il valore K
- La routine di servizio può essere interrotta solo dai dispositivi con priorità >NPR
  - Se NPR=0 non è servita alcuna interruzione
  - Se NPR=N la routine è non interrompibile
- Poiché NPR fa parte del contesto, prima di eseguire la routine di servizio è necessario salvare il suo valore

Accesso diretto alla memoria (DMA)

- È il metodo preferito quando si devono trasferire grosse moli di dati.
- Una circuiteria apposita (*DMA Controller*) provvede ad eseguire il trasferimento di dati da una periferica alla memoria (o viceversa).
- La circuiteria deve essere in grado di fungere da *bus master*, ossia deve generare gli indirizzi ed i segnali di controllo secondo la tempistica opportuna.
- Il DMA Controller deve inoltre essere in grado di negoziare con la CPU l'acquisizione del controllo del bus ed il suo rilascio.

Circuiteria per il DMA



DCR: Registro Contatore Parole da trasferire

IOAR: Registro Indirizzo di memoria del prossimo dato da trasferire

IODR: Registro dati da trasferire in lettura o scrittura

Trasferimento in DMA

- Il trasferimento di un blocco in DMA si articola in varie fasi:
  - 1) la CPU carica nei registri IOAR e DCR l'indirizzo dell'area di memoria ed il numero di parole da trasferire;
  - 2) Quando la periferica è pronta al trasferimento invia una richiesta al DMA Controller
  - 3) Quando il DMA Controller è pronto per eseguire il trasferimento invia un segnale di DMA Request alla CPU; quando questa giunge ad un punto di rilevamento di tale segnale, rilascia il bus e attiva il segnale di DMA Acknowledge;
  - 4) il DMA Controller invia l'ack alla periferica per eseguire il trasferimento
    - il DMA C invia su bus IOAR
    - In lettura viene inviato su bus IODR, in scrittura il contento del bus è scritto su IODR
  - 5) dopo il trasferimento di ciascuna parola, IOAR e DCR vengono decrementati;

Trasferimento in DMA

- quando il DMA Controller termina il trasferimento (ad esempio perchè la periferica non invia più dati) disattiva DMA Request; la CPU disattiva DMA Acknowledge, e riprende il controllo del bus;

Modalità di trasferimento

- **Il trasferimento dei dati in DMA può avvenire in vari modi:**
  - **trasferimento a blocchi *(burst transfer)***
    - Prevede che il DMA Controller, una volta acquisito il controllo del bus, lo mantenga per tutto il tempo richiesto per trasferire un blocco di dati.
    - Il trasferimento a blocchi è importante per periferiche quali i dischi magnetici, dove il trasferimento del blocco non può essere interrotto)
  - **trasferimento con *cycle stealing***
    - Il DMA Controller trasferisce i dati in piccoli blocchi, occupando il bus per periodi limitati di tempo.