Accesso alle periferiche

Introduzione

Interfaccia di una periferica

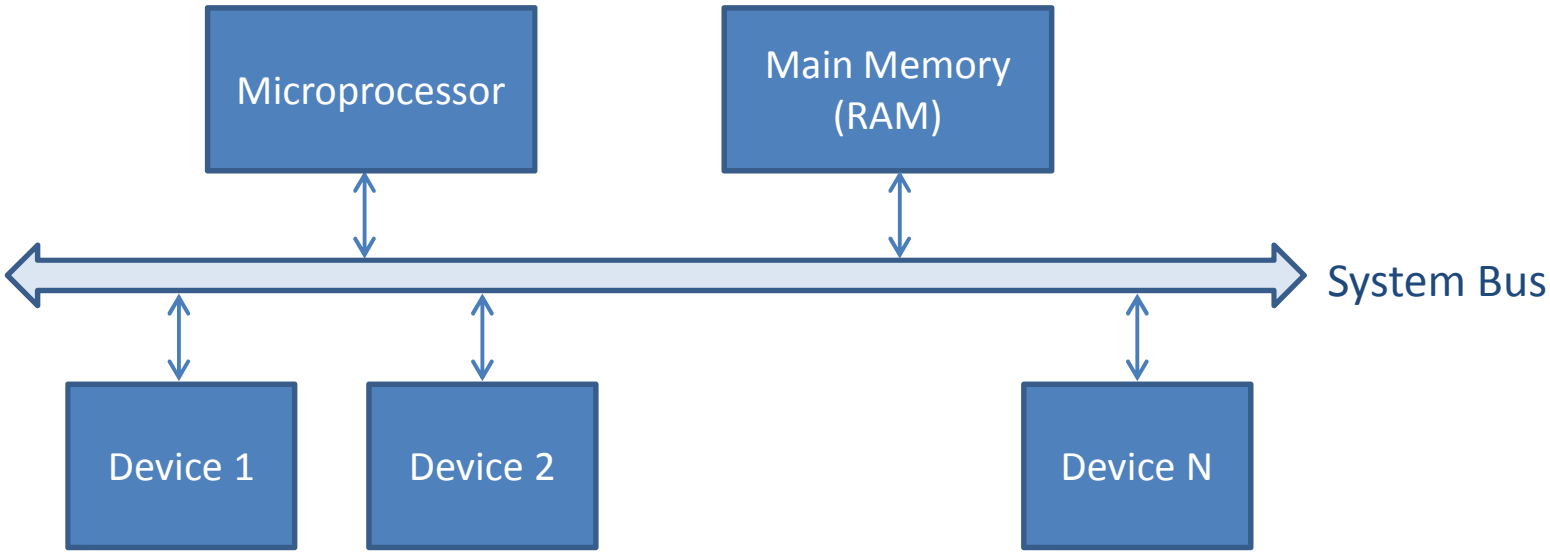
Polling

Interrupt

Introduzione

- **Il calcolatore interagisce con il mondo esterno e con altri calcolatori mediante opportune periferiche o "device"**

- **Le periferiche sono connesse al microprocessore mediante il "bus di sistema"**
  - Qui considereremo solamente un calcolatore con un unico bus di sistema



Interfaccia di una periferica

- **Una periferica deve poter scambiare dati con il microprocessore**

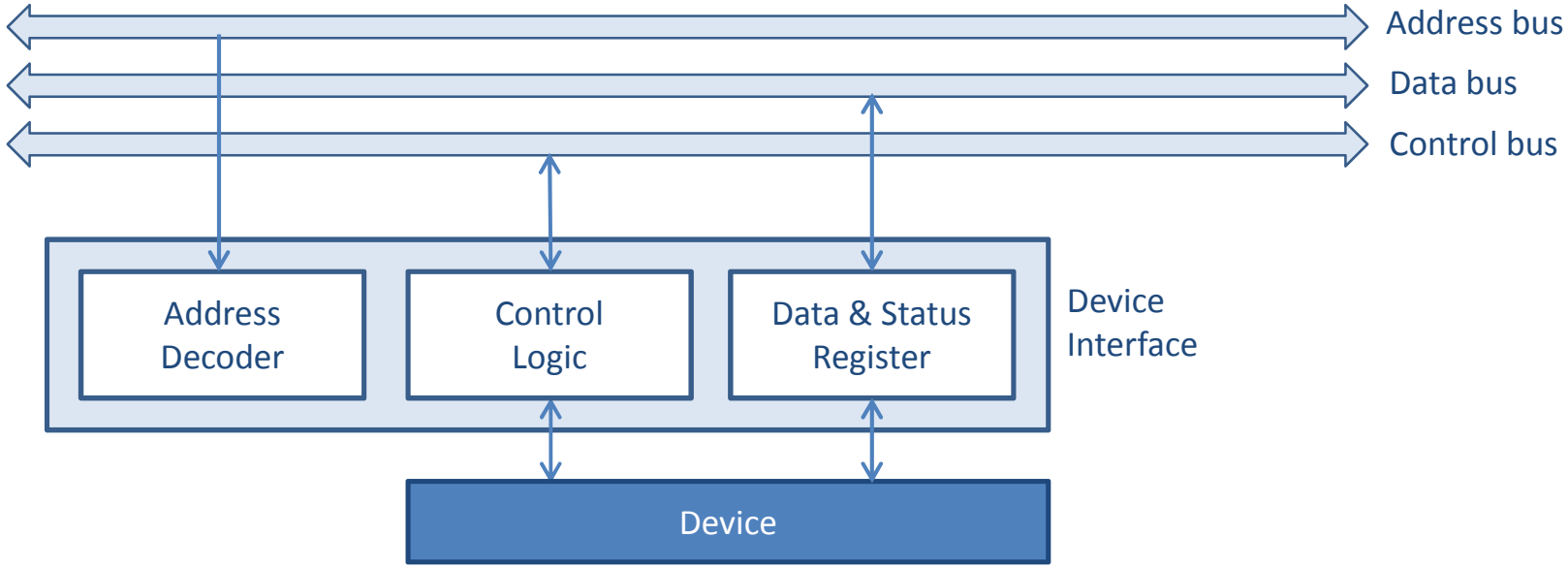
- **A questo scopo è necessario**
  - Identificare la periferica tra tutte quelle disponibili
    - La periferica deve pertanto avere un indirizzo
  - Specificare il tipo di trasferimento che si vuole eseguire (lettura o scrittura)
    - la periferica deve disporre di un meccanismo di controllo
  - Prelevare o scrivere dei dati
    - La periferica deve disporre di una zona di memoria locale per i dati

- **Tutto ciò avviene mediante il bus di sistema**

- **Il bus di sistema è un insieme di linee, raggruppate in**
  - Linee di indirizzo, o "address bus"
  - Linee di dati, o "data bus"
  - Linee di controllo, o "control bus"

Interfaccia di una periferica

- **La connessione di una periferica al bus avviene mediante la sua interfaccia**



- **Address Decoder**
  - Decodifica l'indirizzo, attivando la periferica se l'indirizzo sul bus è il proprio
- **Contro Logic**
  - Realizza la logica di controllo necessaria svolgere le operazioni richieste
- **Data & Status registers**
  - Sono registri destinati a contenere i dati e le informazioni di stato della periferica

Interfaccia di una periferica

- **I registri di una periferica**
  - Hanno un nome simbolico
  - Sono associati ad indirizzi univoci
  - Possono essere acceduti dal processore "come se" fossero normali indirizzi di memoria
- **Consideriamo, per esempio, l'interfaccia di una tastiera**
  - Registro dati: Contiene il carattere letto

TTY_D	0x0080	data

  - Registro di stato: Indica lo stato della periferica
    - IRQ: Richiesta interrupt (vedi oltre)
    - RDY: Indica se è presente un dato valido

TTY_S	0x0081						IRQ	RDY

  - Registro di controllo: Permette di controllare la periferica
    - IE Abilita e disabilita gli interrupt

TTY_C	0x0082				IE			

Polling

- **Il metodo più semplice di accesso ad una periferica è il "polling"**
  - Il programma controlla continuamente il registro di stato
  - Se il registro indica che è presente un dato valido, lo preleva
- **Il programma seguente legge un carattere da tastiera mediante polling**

```
TTY_D   EQU       0x0080
TTY_S   EQU       0x0081
TTY_C   EQU       0x0082
CHAR    DATAWORD  0

        ORIGIN    0xF000
POLL    LOAD      R0, TTY_S
        AND       R0, #0x01
        BEQ       POLL
        LOAD      R1, TTY_D
        STORE     CHAR, R1
```

- **Mediante polling**
  - Il processore è sempre impegnato a verificare lo stato della periferica
  - Questo meccanismo è poco efficiente

Interrupt

- **Grazie al meccanismo di interrupt si può migliorare l'accesso alle periferiche**
- **Il principio di funzionamento è il seguente**
  - Il programma configura la periferica in modo che sia in grado si "sollevare un interrupt"
    - Cioè di segnalare al microprocessore in modo asincrono la presenza di un nuovo dato
  - Il programma specifica una routine da eseguire in corrispondenza dell'interrupt
    - Si tratta appunto della interrupt service routine
  - Il flusso del programma procede quindi autonomamente
  - Quando si verifica un'interruzione
    - Il processore riceve un segnale di interrupt sul bus di controllo
    - Il processore decide se "servire" o meno l'interrupt
  - Se il processore decide di servire l'interrupt
    - Interrompe il flusso del programma
    - Esegue la interrupt service routine
    - Ritorna al punto in cui il flusso del programma era stato interrotto
- **Ciò comporta**
  - Uno speciale cambio di contesto, diverso da quello della chiamata a funzione
  - Un modo per indicare quale routine eseguire

Interrupt

- **Per salvare il contesto di esecuzione**
  - Il programma non può fare nulla, poiché l'interrupt è asincrono
  - E' necessario che sia il microprocessore stesso ad eseguire il salvataggio
    - Almeno del program counter e dello status register
- **La routine di servizio svolge poi le seguenti operazioni**
  - Disabilta gli interrupt
    - In modo che il processore ignori altri interrupt finché la ISR non è terminata
  - Salva tutti i registri sullo stack
  - Esegue le operazioni richieste
  - Ripristina tutti i registri dallo stack
  - Cancella il flag di interrupt nel registro di stato della periferica
  - Riabilita gli interrupt
  - Esegue il "ritorno da interrupt" mediante l'istruzione IRET
    - Ripristina lo status register salvato sullo stack dal processore
    - Salta al valore del program counter salvato sullo stack
    - A questo punto lo stack è esattamente nello stato precedente all'interruzione

Interrupt

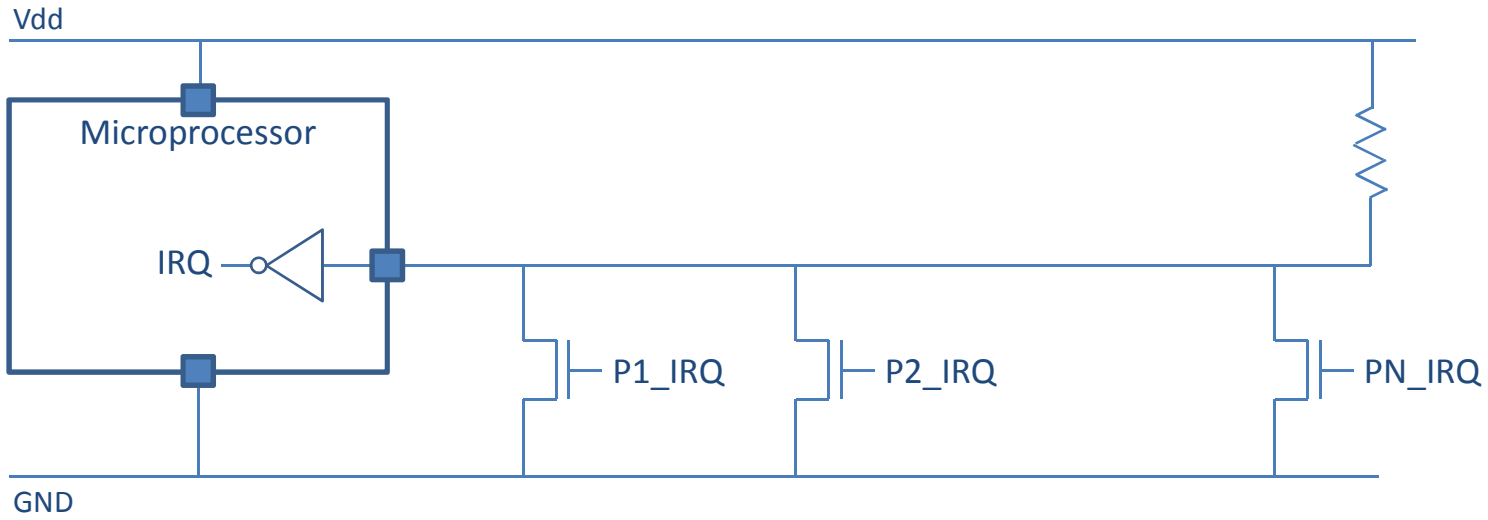
- **Lettura di un carattere mediante interrupt service routine**
  - Non specifichiamo come associare questa routine allo specifico segnale di interrupt
  - Supponiamo che il processore abbia solo due registri R0 ed R1
- **Il codice della ISR è il seguente**

```
        ...
        ORIGIN    0xF000
MAIN    LOAD      R1, TTY_C
        OR        R1, 0x10    ; TTY_C:IE <= 1
        ...

        ORIGIN    0xC000
TTY_ISR DI
        PUSH      R0
        PUSH      R1
        LOAD      R0, TTY_D
        STORE     CHAR, R0
        LOAD      R1, TTY_S
        AND       R1, 0xFD    ; TTY_S:IRQ <= 0
        POP       R1
        POP       R0
        EI
        IRET
```

Logica di interrupt

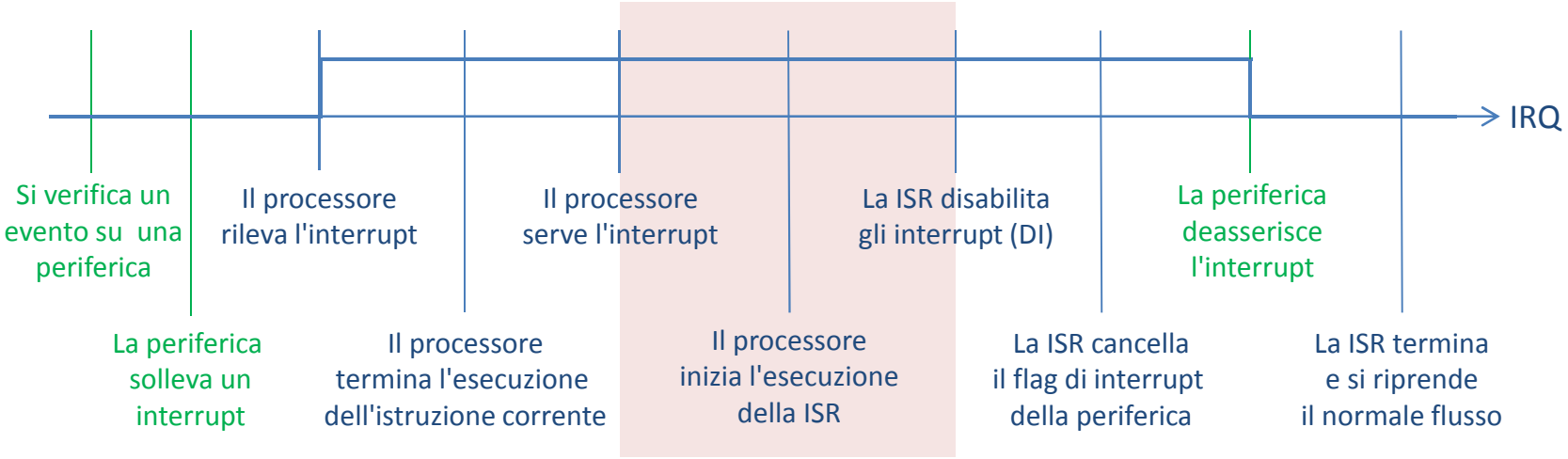
- **Il modo più semplice consiste nel ricorrere ad una sola linea IRQ**



- **Il funzionamento è il seguente**
  - Quando tutti gli nMOS sono aperti, cioè quando tutti i segnali Pi_IRQ sono a 0
    - La linea di interrupt è cortocircuitata a Vdd ed ha pertanto valore logico 1
    - Il segnale IRQ interno al microprocessore vale pertanto 0
  - Non appena una periferica richiede un'interrupt
    - Uno degli nMOS cortocircuita a massa la linea di interrupt, che assume valore logico 0
    - Il segnale IRQ interno al microprocessore vale pertanto 1

Logica di interrupt

- **Quando una periferica solleva un interrupt si verificano i seguenti eventi**



- **Esiste pertanto un intervallo di tempo in cui**
  - Il programma non ha ancora disabilitato gli interrupt
  - Il processore potrebbe entrare in un "ciclo" di richieste di interruzione
    - Molto pericoloso

Logica di interrupt

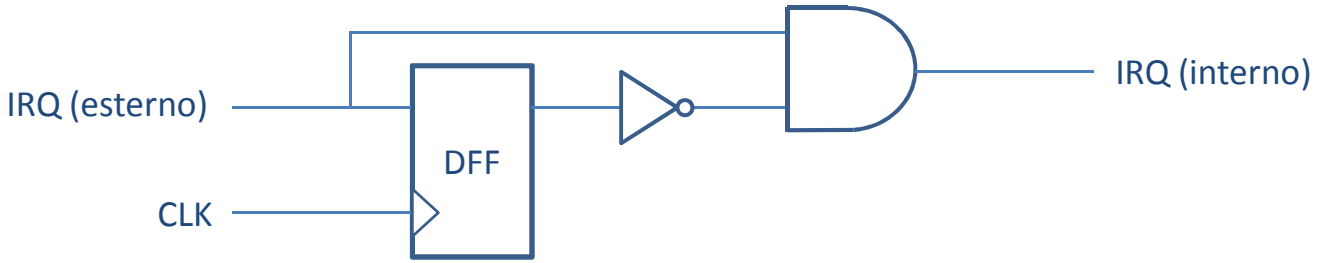
- **Esistono alcune soluzioni per ovviare al problema**
  - Richiedono un'apposita logica di controllo
  - Alcune richiedono la cooperazione della ISR
- **Disabilitazione semi-automatica degli interrupt**
  - Al termine dell'istruzione corrente
    - Il microprocessore ignora automaticamente la linea di interrupt per una ulteriore istruzione
    - Se la prima istruzione della ISR è una DI, non si può innestare un nuovo interrupt
    - Il microprocessore salva PC e SR sullo stack
  - Inizia l'esecuzione della ISR, che inizia con una DI
  - Il meccanismo di interruzione viene riabilitato da un'istruzione EI
    - Per evitare che si innesti un nuovo interrupt prima del rientro dalla ISR
    - L'istruzione EI deve essere la penultima della ISR
    - L'effetto dell'istruzione EI deve essere ritardato di un ciclo di clock
    - In questo modo il processore può eseguire la IRET senza problemi
  - La ISR ritorna al flusso di esecuzione del programma mediante l'istruzione IRET
    - Recupera PC e SR dallo stack
    - Ripristina lo stack allo stato in cui si trovava prima dell'interruzione

Logica di interrupt

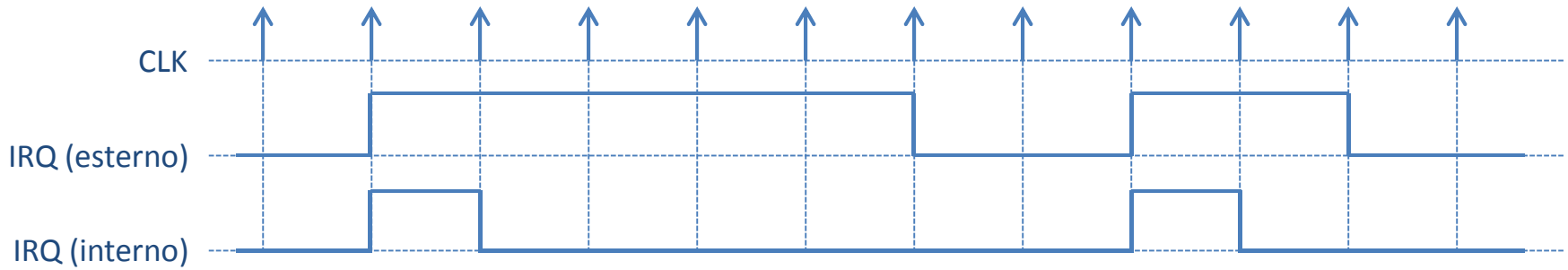
- **Disabilitazione automatica degli interrupt**
  - Al termine dell'istruzione corrente
    - In questo momento il bit IE di SR vale 1
    - Il microprocessore salva PC e SR sullo stack
  - Il microprocessore pone a zero il bit IE del registro di stato
    - Ciò equivale a disabilitare gli interrupt mediante l'istruzione DI
  - Inizia l'esecuzione della ISR
    - La ISR viene eseguita con interrupt disabilitati
    - Non si deve curare di disabilitare/riabilitare esplicitamente gl interrupt
  - La ISR ritorna al flusso di esecuzione del programma mediante l'istruzione IRET
    - Il bit IE del registro SR salvato valeva 1
    - Recupera PC e SR dallo stack
    - A questo punto il bit IE di SR vale 1, cioè gli interrupt sono riabilitati
    - Anche se arrivasse ora un interrupt il processore attenderebbe comunque la fine dell'istruzione corrente (la IRET, appunto) prima di servire il nuovo interrupt
  - Il flusso riprende nel programma interrotto

Logica di interrupt

- **Interrupt sul fronte**
  - Si realizza un circuito di rilevamento sensibile al fronte di salita del segnale IRQ
    - E' necessario "passare" per un fronte di discesa di IRQ prima di avere un nuovo fronte di salita
    - Per generare un fronte di discesa su IRQ è necessario resettare il flag di interrupt della periferica
    - Ciò viene fatto esplicitamente su richiesta della ISR
- **Il cicuito di rilevamento dei fronti è il seguente**



- **L'andamento dei segnali mostra l'effetto del circuito**



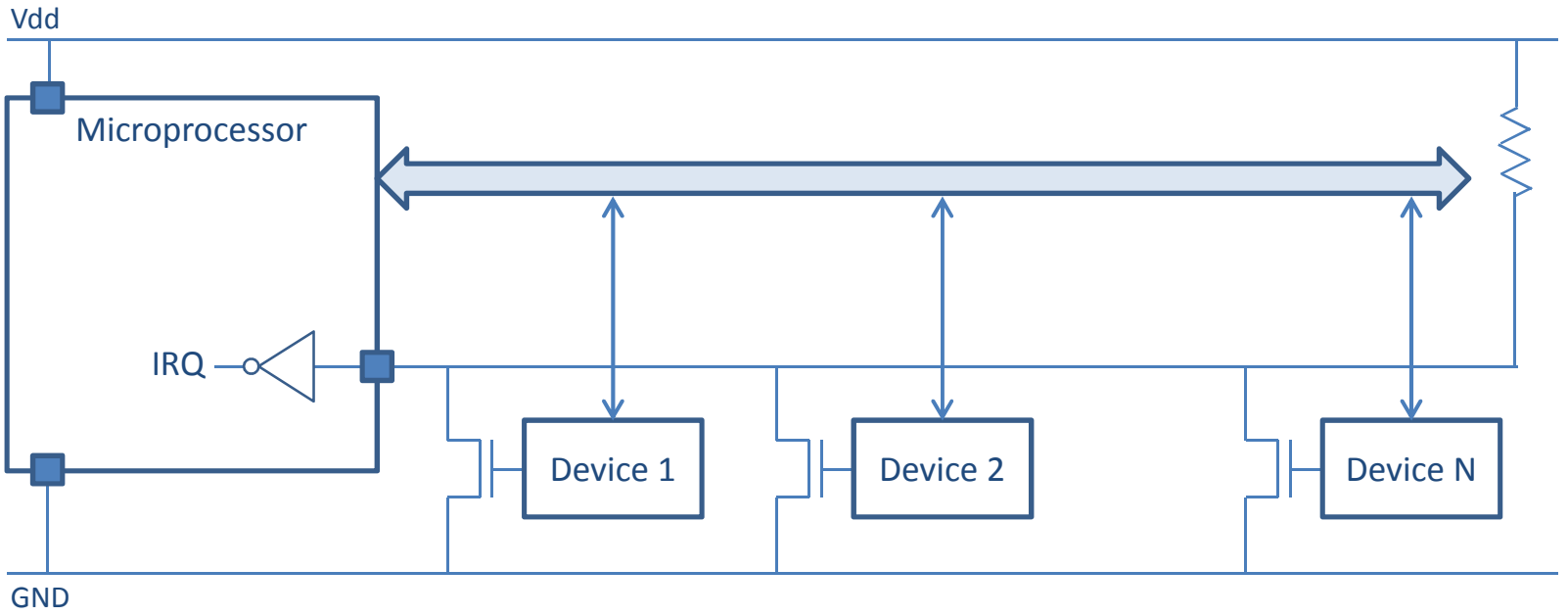
Interrupt multipli

- **Nel caso vi siano molti dipositivi connessi al microprocessore è necessario**
  - Identificare il dispositivo che ha sollevato l'interrupt
  - Associare una specifica ISR ad ogni interrupt
  - Secegliere la periferiche da servirer nel caso si verifichino due o più interrupt

- **Pe l'identificazione dell'interrupt si può ricorrere a tre schemi**
  - Linee di interrupt specifiche per ogni dispositivo
  - Identificazione da parte della periferica
  - Connessione delle periferiche in daisy-chain

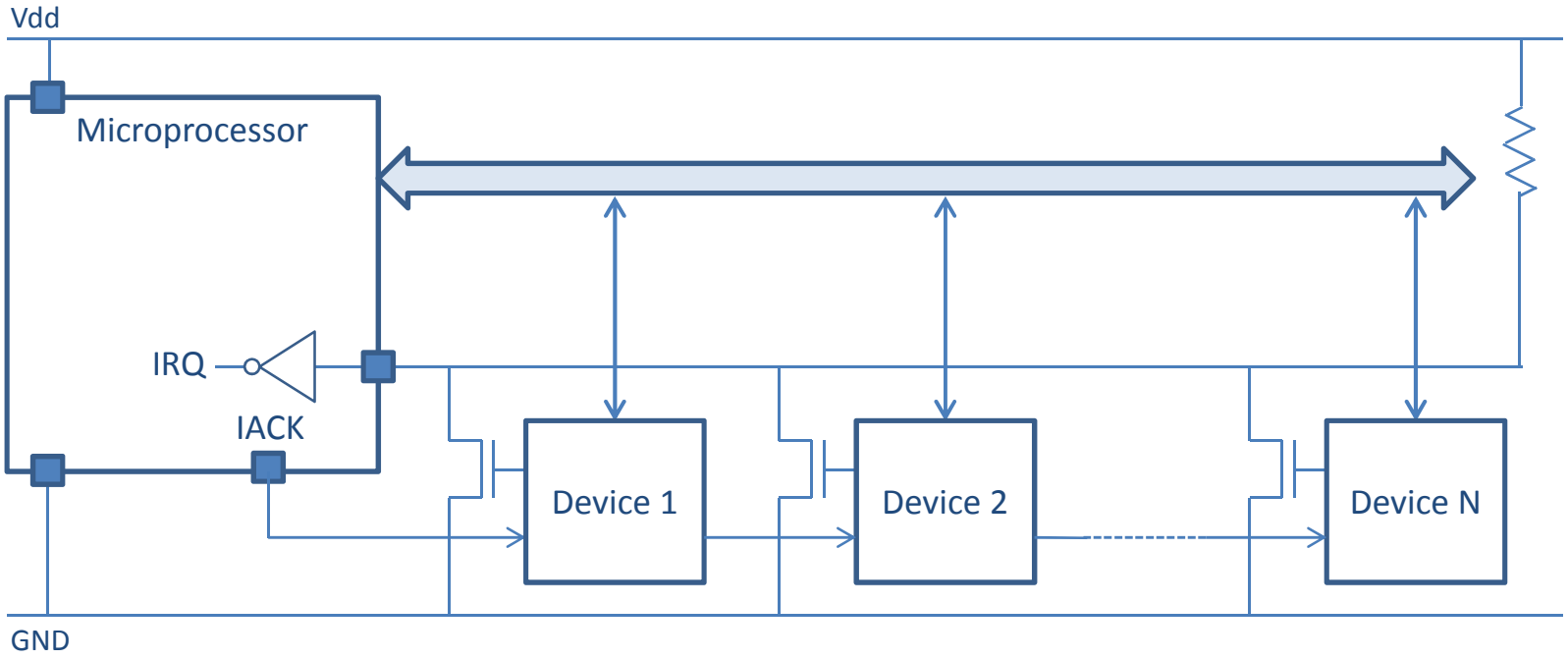
Identificazione da parte della periferica

- **Il sistema dispone di una sola linea di interrupt**
  - Tutte le periferiche controllano la stessa linea
  - Quando una periferica richiede un interruzione
    - Asserisce la linea di interrupt
    - Pone sul bus un "codice" che la identifica
  - Questa soluzione non consente di gestire interruzioni multiple e priorità



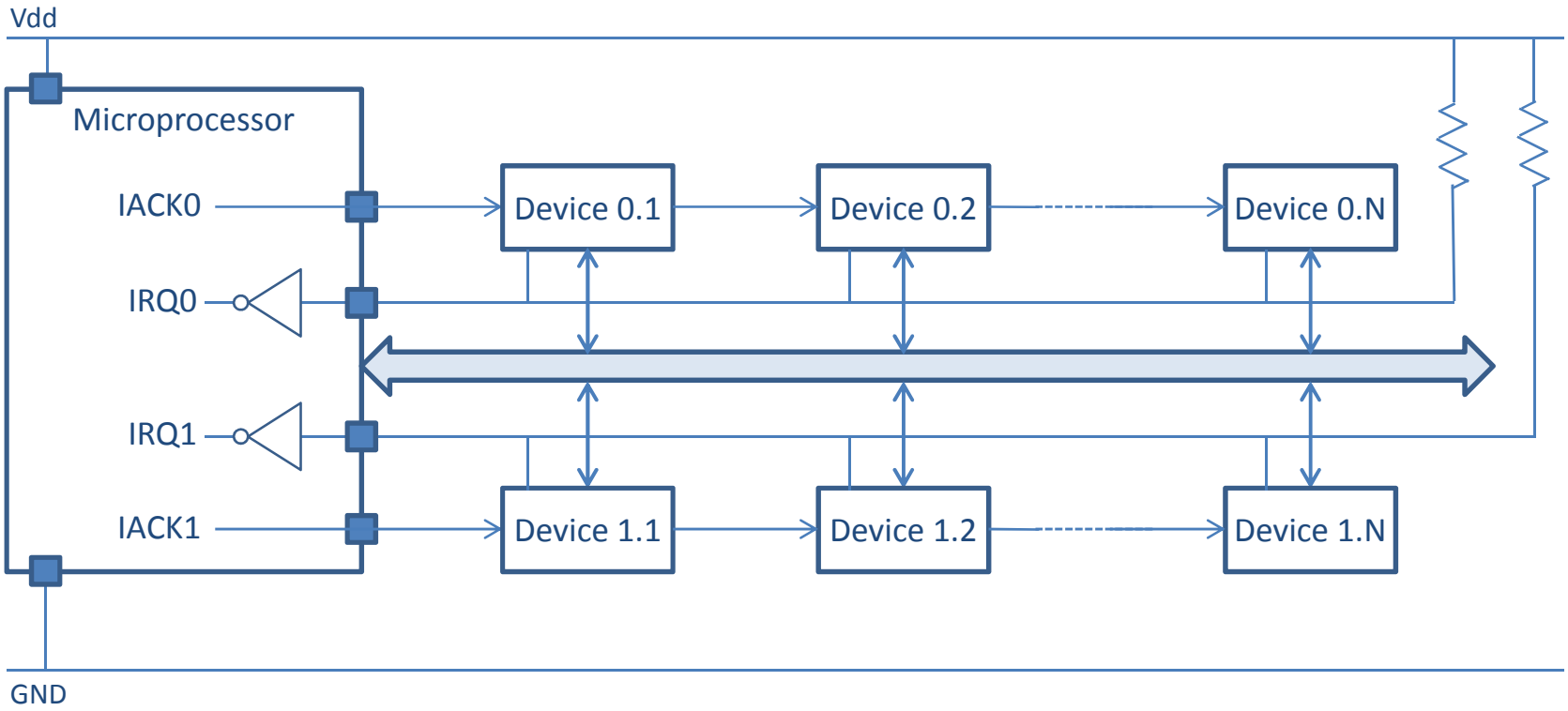
Collegamento in daisy-chain

- **Per gestire interruzioni multiple e priorità**
  - Si ricorre ad una linea aggiuntiva detta di "acknowledge" o IACK
    - Quando decide di servire un interrupt, il processore asserisce la linea di acknowledge
  - Se una periferica non ha sollevato l'interrupt
    - Propaga il segnale di acknowledge
  - La periferica che ha sollevato l'interrupt
    - Non propaga il segnale e pone sul bus un "codice" che la identifica



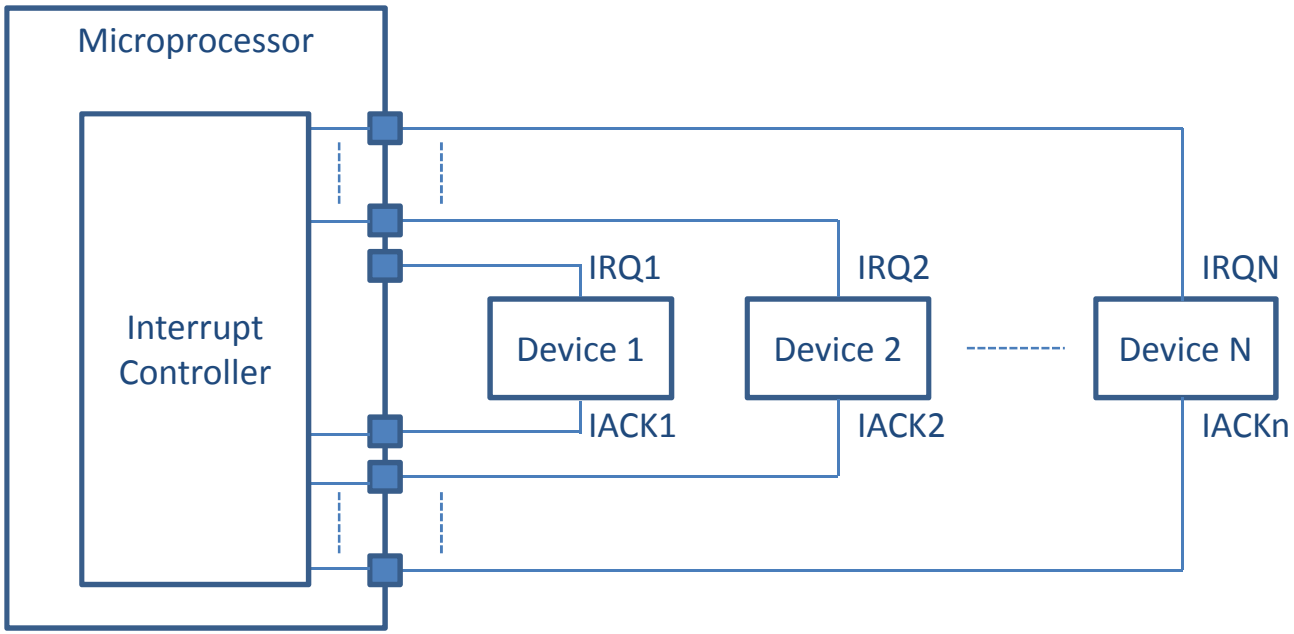
Collegamento in daisy-chain

- **Secondo questo schema**
  - La priorità delle periferiche è fissata
    - Quelle più vicine al processore hanno priorità maggiore
- **Per rendere il meccanismo più flessibile**
  - Si ricorre due (o più) coppie di linee IRQ / IACK



Linee di interrupt multiple

- **In questa solzuione**
  - Ogni dispositivo dispone di due linee dedicate di richiesta e acknowledge
  - Il microprocessore dispone di un modulo detto "interrupt controller"
    - Definisce e gestisce le priorità degli interrupt
    - Gestisce l'abilitazione/disabilitazione selettiva degli interrupt



Identificazione della routine di servizio

- **Nel caso di una sola linea di interrupt, si ha una sola routine di servizio**
  - Preleva il valore presente sul bus, ad un indirizzo noto, che indichiamo con DBUS_IN
  - In base a tale valore salta alla specifica sezione di codice che gestisce la periferica

```
ISR       DI
          PUSH  R0
          PUSH  R1

         ; Read bus
          LOAD  R0, DBUS_IN

          ; Selects the specific code
          CMP   R0, #0x01
          BEQ   ISR_01
          CMP   R0, #0x02
          BEQ   ISR_02
          ...
          CMP   R0, #0x1B       ; Last id
          BEQ   ISR_1B

          ; Set error
          STORE ISR_ERROR, #1
          B     ISR_END
```

```
          ; ISR code for id 0x01
ISR_01    ...                   ; Processing
          LOAD  R0, DEV_01_S
          AND   R0, #0xEF       ; Clear flag
          B ISR_END

          ; ISR code for id 0x01
ISR_02    ...                   ; Processing
          LOAD  R0, DEV_02_S
          AND   R0, #0xFB       ; Clear flag
          B ISR_END

          ...

          ; Restore stack and returns
ISR_END   POP   R1
          POP   R0
          EI
          IRET
```

Identificazione della routine di servizio

- **Una diversa implementazione consiste nel utilizzare una tabella**
  - Il codice presente sul bus indirizza la tabella
  - Ogni elemento della tabella contiene l'indirizzo della specifica sezione di codice

```
          ; Interrupt vector
          ORIGIN    0x0000
IVECT     DATAWORD  ISR_01
          DATAWORD  ISR_02
          ...
          DATAWORD  ISR_1B
```

```
ISR       DI
          PUSH  R0
          PUSH  R1

          ; Read bus
          LOAD  R0, DBUS_IN

          ; Checks if code is valid
          CMP   R0, #0x1B    ; Last id
          BGT   ISR_ERR
          ; Indirect branch
          ADD   R0, IVECT
          B     (R0)

          ; Set error
          STORE ISR_ERROR, #1
          B     ISR_END
ISR_END   ...
          IRET
```

Identificazione della routine di servizio

- **La soluzione più flessibile è detta "vectored interrupt" e si basa su**
  - Una tabella detta "vettore di interrupt" o "interrupt vector"
    - Tale tabella contiene gli indirizzi delle diverse routine di servizio
  - Una specifica routine per ogni perriferica, ossia per ogni interrupt
  - Il processore
    - Identifica l'interrupt e preleva l'indirizzo dal vettore
    - Esegue la routine che si trova a quell'indirizzo
  - A volte è necesario specificare dove si trova il vettore degli interrupt
    - Scrivendo tale indirizzo in un registro (memory mapped), che indichiamo con IVBA

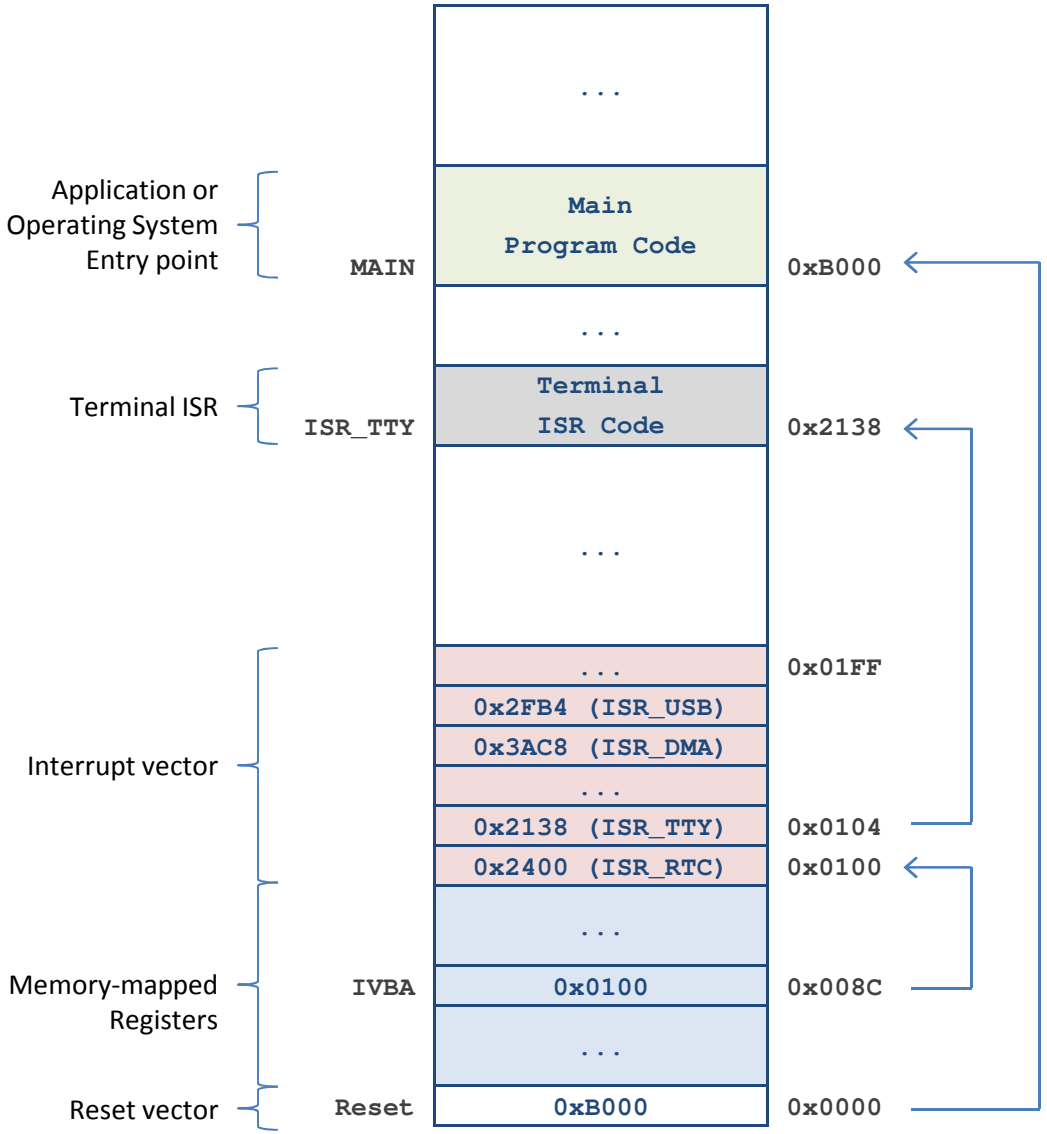
```
          ; Interrupt vector
          ORIGIN    0xFF00
IVECT     DATAWORD  ISR_01
          DATAWORD  ISR_02
          ...
          DATAWORD  ISR_1B

          ORIGIN    0x8000
MAIN      STORE     IVECT, IVBA
          ...
```

```
ISR_XX    DI
          PUSH  R0
          PUSH  R1
          ; Processing
          ; Clear flag
          POP   R1
          POP   R0
          IRET
```

Identificazione della routine di servizio

- **Nel caso di vectored interrupt la memoria è composta dalle seguenti zone "logiche"**
  - Reset vector
    - Una o più celle adiacenti che specificano il o gli entry point dell'applicazione o del S.O.
  - Memory-mapped register
    - Banco di registri delle periferiche e di configurazione del processore
  - Interrupt vector
    - Tabella degli indirizzi delle interrupt service routines
  - ISR
    - Zone di codice contenenti le ISR
    - In generale, non sono adiacenti
  - Main code
    - Codice dell'applicazione o del S.O.



Interrupt multipli

- **Si è visto come a livello hardware sia possibile definire la priorità di un interrupt**
  - Daisy-chain singola: priorità fissa, scarsa flessibilità
  - Daisy chain multipla: priorità fissa, maggiore flessibilità
  - Linee multiple: priorità flessibile sotto la gestione dell'interrupt controller

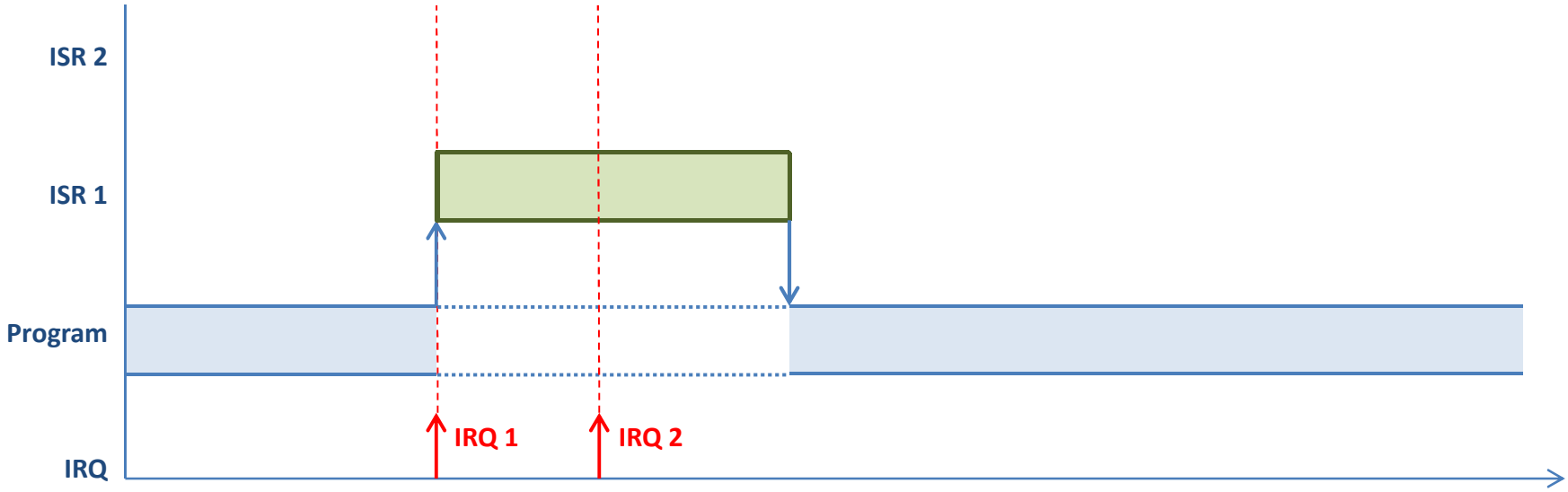
- **Cosa accade quando il processore riceve un interrupt mentre ne sta già servendo uno arrivato in precedenza?**

- **Il comportamento dipende da diversi fattori**
  - Supporto dei livelli di priorità
  - Supporto dell'annidamento delle ISR
  - Livelli di priorità relativi delle interruzioni

- **A seconda dei casi sono possibili diverse soluzioni**

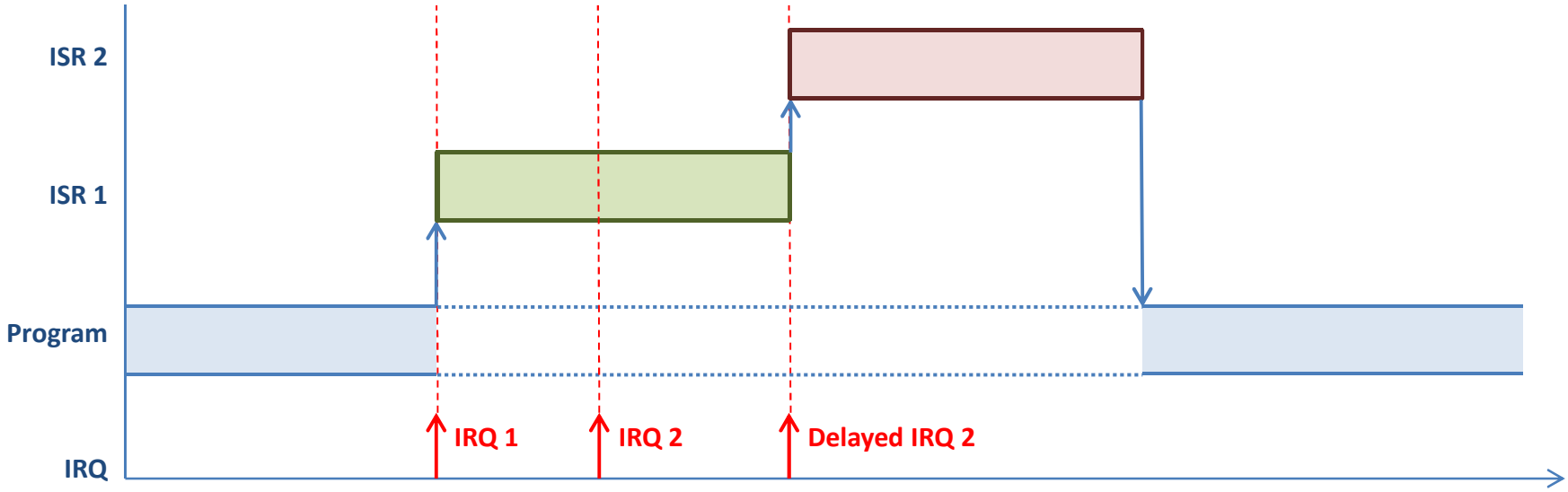
Interrupt multipli

- **Soluzione 1: Si ignora il nuovo interrupt**
  - Molto semplice da realizzare, per esempio mediante i bit del registro di stato
  - Poco flessibile, non sempre adeguata a tutte le situazioni
    - Alcuni interrupt non possono essere mai ignorati



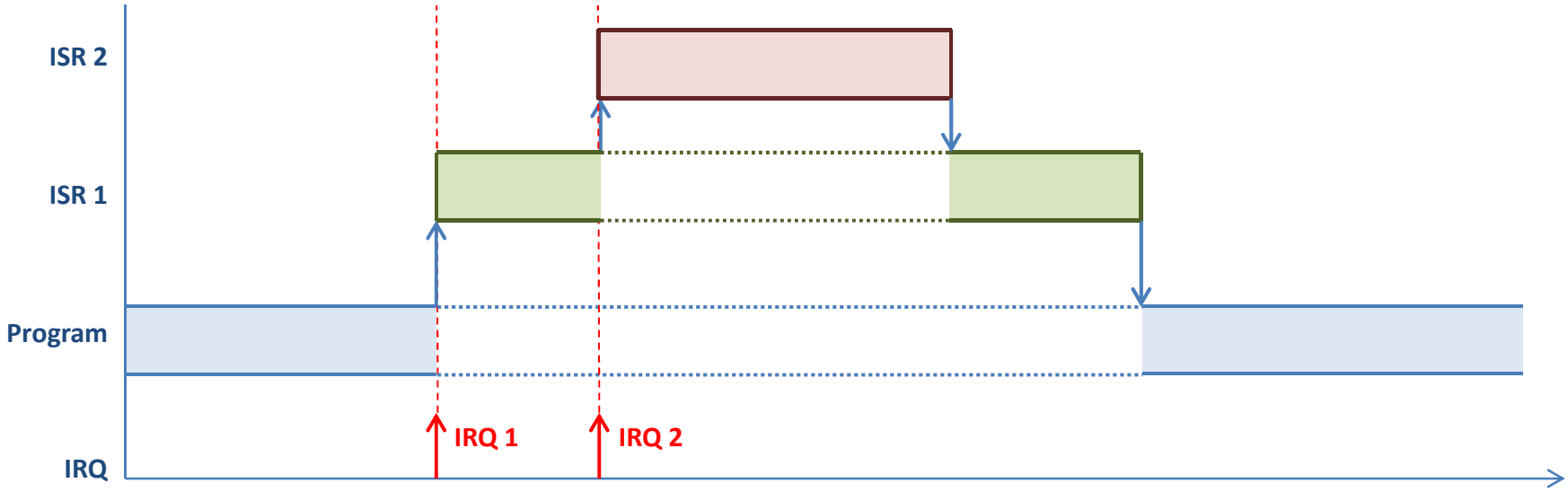
Interrupt multipli

- **Soluzione 2: Si ritarda il servizio del nuovo interrupt (delayed interrupt)**
  - Semplice da realizzare, richiede supporto hardware
  - Poco flessibile, non sempre adeguata a tutte le situazioni
    - Alcuni interrupt richiedono di essere serviti in tempi molto rapidi
    - Si pensi, per esempio, al real-time clock



Interrupt multipli

- **Soluzione 3: Si prevede il supporto per interrupt annidati (interrupt nesting)**
  - Richiede un supporto hardware abbastanza complesso
    - Si possono supportare solo un numero fissato di routine annidate, spesso solo 2
  - Molto flessibile
    - Permette al programmatore di stabilire il comportamento nel caso degli specifici interrupt



Interrupt multipli

- **Nel caso di interrupt annidati si possono avere diverse situazioni**
  - Tutti gli interrupt hanno la stessa priorità
    - Ogni ISR può essere interrotta da qualsiasi altra ISR
  - Gli interrupt hanno diversi lievelli di priorità
    - Una ISR può essere interrotta solo da una ISR di priorità maggiore
    - In questo caso è necessario un meccanismo per tenere traccia del livello di priorità corrente
    - Se si accetta un solo livello di nesting, è sufficiente un registro
    - Nel caso di nesting più profondo, è necessario ricorrere ad un meccanismo basato su stack

- **Spesso un processore prevede uno o più "non maskable interrupts" o NMI**
  - Interrupt che non possono essere mai "mascherati", ma devono sempre essere serviti
  - Una realizzazione frequente consiste nel definire un livello di priorità speciale
    - Indicato generalmente con 0 (valori più bassi indicano priorità maggiori)
    - Nessuna altro interrupt può avere questo livello

Interrupt non mascherabili

- **Gli interrupt non mascherabili**
  - Sono detti "eccezioni"
  - Sono generati dal processore stesso
- **Alcuni interrupt non mascherabili indicano situzioni di errore grave**
  - Bus error
    - Bad instruction
  - Math exception
    - Divide-by-zero, Floating-point exception
  - Protection violation
    - Memory access violation
    - Privileged instruction execution in user mode
- **Altri interrupt non mascherabili sono definiti a scopo di debugging**
  - Breakpoint o trap
    - Il processore raggiunge un punto in cui il programmatore ha stabilito di volersi fermare
  - Trace mode
    - Il processore si interrompe al termine di ogni istruzione assembly

Direct Memory Access

- **Per il trasferimento di dati da/verso le perifriche esisto due tecniche**
  - Polling
  - Interrupt

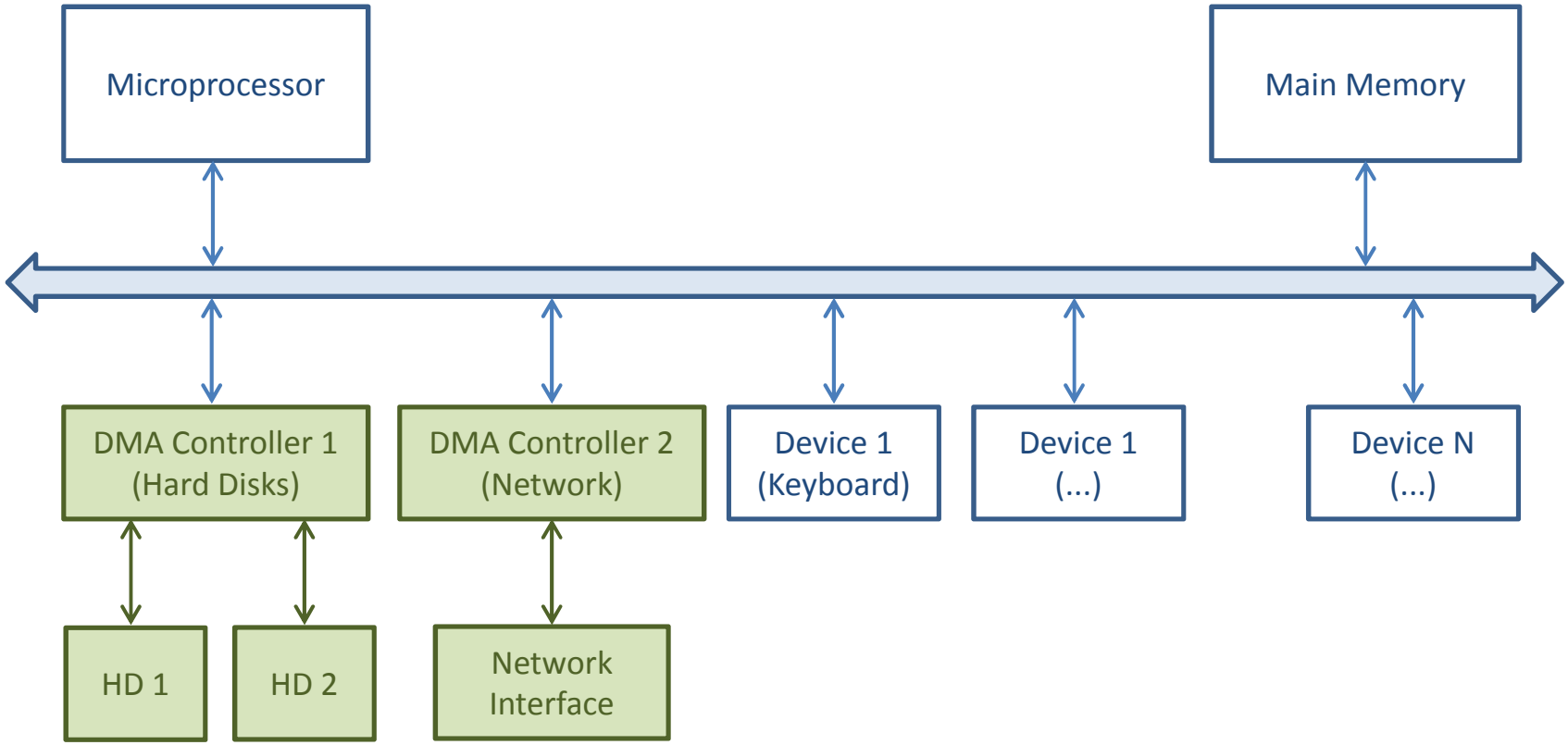
- **In entrambi i casi**
  - Il processore è coinvolto direttamente nel trasferimento di ogni singola parola o byte
  - Il processore è vincolato al tempo tipico di risposta della periferica
    - Totalmente vincolato nel caso di polling: deve eseguire un ciclo esplicito di attesa
    - Meno vincolato nel caso di interrupt: deve eseguire la specifica ISR

- **Nel caso di trasferimento di grandi quantità di dati ciò è inefficiente**
  - Si pensi ad esempio al trasferimento di dati da disco o da una interfaccia di rete

- **Una tecnica alternativa e molto più efficiente consiste nel ricorrere al trasferimento diretto alla memoria o DMA**
  - Il processore delega ad un apposito dispositivo le operazioni di trasferimento
  - Il processore è libero di eseguire altro codice

Direct Memory Access

- **La tecnica di DMA richiede una architettura del tipo seguente**
  - I controllori DMA possono gestire una o più periferiche ciascuno
  - I controllori DMA sono visti come periferiche generiche dal processore



Direct Memory Access

- **Il trasferimento DMA avviene nel modo seguente**
  - Un processo P richiede una operazione in DMA
    - Il processo P accede all'interfaccia del DMA controller per specificare l'operazione da svolgere
    - Il processo P richiede al DMA controller l'avvio dell'operazione DMA
    - Il processo P si sospende in attesa della fine dell'operazione
  - Il sistema operativo
    - Pone in esecuzione un nuvo processo Q
  - Il DMA controller
    - Inizia l'operazione di trasferimento, agendo in luogo del processore
    - Agisce da bus master
    - Si interfaccia con il dispositivo specifico e con la memoria, in modo, appunto, diretto
  - Al termine del trasferimento il DMA controller
    - Solleva un interrupt di fine trasferimento
  - Il sistema operativo
    - Intercetta l'interrupt, che dal suo punto di vista è un evento
    - Risveglia il processo P in attesa su quell'evento e lo pone in esecuzione
  - Il processo P
    - Utilizza i dati trasferiti dall'operazione in DMA

Direct Memory Access

- **L'interfaccia del DMA è costituita da alcuni registri**
- **Registro di stato**
  - DMA_IE: Interrupt enable, indica se il DMA può sollevare interrupt
  - DMA_IR: Interrupt request, indica se il DMA ha generato una richiesta di interrupt
  - DMA_TC: Transfer Complete: indica che il trasferimento è completo
- **Registro di controllo**
  - DMA_START: Transfer Start: inizia il trasferimento
  - DMA_RW: Read/Write: indica la direzione del trasferimento
- **Registro di indirizzo di base**
  - DMA_BASE: Address Base: indica l'indirizzo in memoria dell'inizio del buffer dei dati
- **Registro di dimensione dati**
  - DMA_SIZE: Data Size: indica la quantità di dati da trasferire
- **Registri specifici della periferica**
  - Sono utilizzati per controllare la specifica periferica gestita in DMA

Architettura a bus multipli

- **Il meccanismo del DMA offre il vantaggio di**
  - Eseguire in autonomia operazioni di trasferimento tra periferica e memoria
  - Consentire al processore di eseguire altre operazioni
- **A tale scopo, tuttavia, il processore necessita del bus**
  - E' impegnato dal DMA per il trasferimento, per cui non si pressoché alcun vantaggio
    - A meno di eseguire codice presente in cache su dati anch'essi presenti in cache
- **Si ricorre ad una architettura più evoluta**
  - Un bus dedicato alla sola memoria
  - Una memoria principale "dual-port"