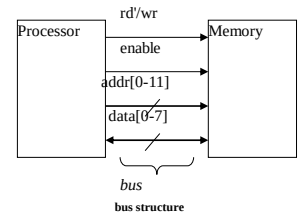
Interfacing

Introduction

- Embedded system functionality aspects
  - Processing
    - Transformation of data
    - Implemented using processors
  - Storage
    - Retention of data
    - Implemented using memory
  - Communication
    - Transfer of data between processors and memories
      - Read/Write a memory
      - Read/Write peripheral'register
    - Implemented using buses
    - Called *interfacing*

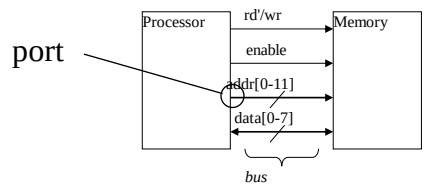
A simple bus

- Wires:
  - Uni-directional or bi-directional
  - One line may represent multiple wires
- Bus
  - Set of wires with a single function
    - Address bus, data bus
  - Or, entire collection of wires
    - Address, data and control
    - Associated protocol: rules for communication



bus structure

Ports

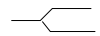


- Conducting device on periphery
- Connects bus to processor or memory
- Often referred to as a *pin*
  - Actual pins on periphery of IC package that plug into socket on printed-circuit board
  - Sometimes metallic balls instead of pins
  - Today, metal "pads" connecting processors and memories within single IC
- Single wire or set of wires with single function
  - E.g., 12-wire address port

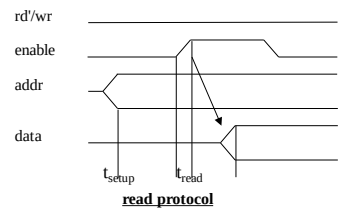
Timing Diagrams

- Most common method for describing a communication protocol
- Time proceeds to the right on x-axis
- Control signal: low or high

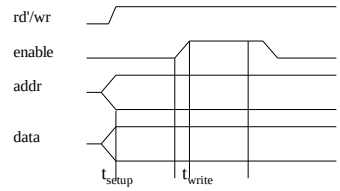
  - May be active low (e.g., go', /go, or go_L)
  - Use terms *assert* (active) and *deassert*
  - Asserting go' means go=0
- Data signal: not valid or valid



- Protocol may have subprotocols
  - Called bus cycle, e.g., read and write
  - Each may be several clock cycles
- Read example
  - *rd'/wr* set low,address placed on *addr* for at least t_{setup} time before *enable* asserted, enable triggers memory to place data on *data* wires by time t_{read}



read protocol

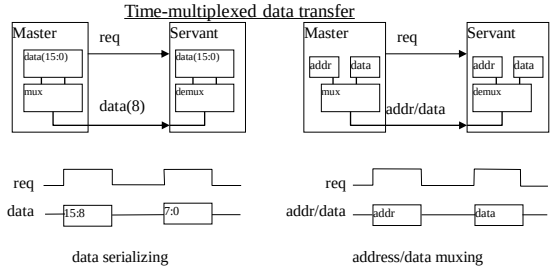


write protocol

Basic protocol concepts

- Actor: master initiates, servant (slave) respond
- Direction: sender, receiver
- Addresses: special kind of data
  - Specifies a location in memory, a peripheral, or a register within a peripheral
- Time multiplexing
  - Share a single set of wires for multiple pieces of data
  - Saves wires at expense of time

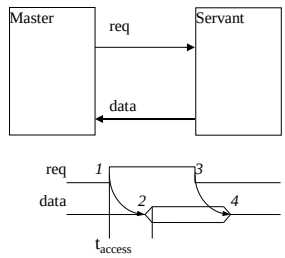
Time-multiplexed data transfer



data serializing

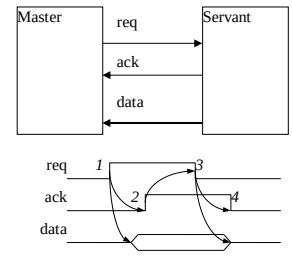
address/data muxing

Basic protocol concepts: control methods



1. Master asserts *req* to receive data
2. Servant puts data on bus **within time t_{access}**
3. Master receives data and deasserts *req*
4. Servant ready for next request

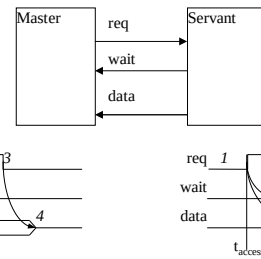
Strobe protocol



1. Master asserts *req* to receive data
2. Servant puts data on bus **and asserts *ack***
3. Master receives data and deasserts *req*
4. Servant ready for next request

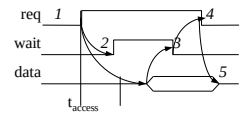
Handshake protocol

A strobe/handshake compromise



1. Master asserts *req* to receive data
2. Servant puts data on bus **within time t_{access}** (wait line is unused)
3. Master receives data and deasserts *req*
4. Servant ready for next request

Fast-response case

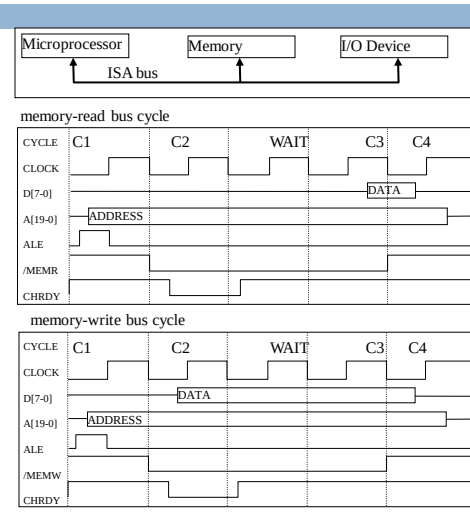


1. Master asserts *req* to receive data
2. Servant can't put data within t_{access}, **asserts *wait*** ack
3. Servant puts data on bus and **deasserts *wait***
4. Master receives data and deasserts *req*
5. Servant ready for next request

Slow-response case

ISA bus protocol – memory access

- ISA: Industry Standard Architecture
  - Common in 80x86's
- Features
  - 20-bit address
  - Compromise strobe/handshake control
    - 4 cycles default
    - Unless CHRDY deasserted – resulting in additional wait cycles (up to 6)



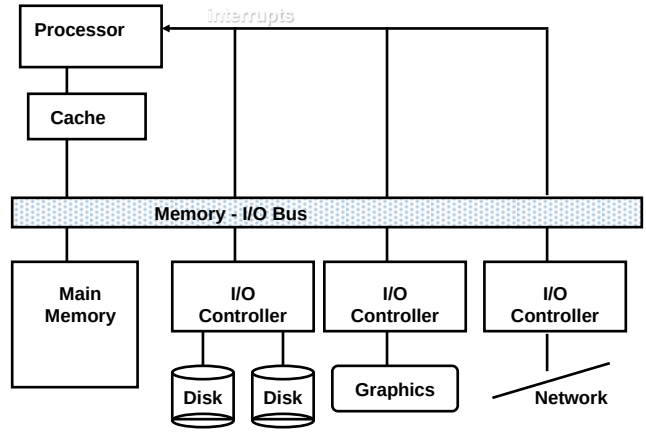
Motivation: Who Cares About I/O?

- CPU Performance: 60% per year
- I/O system performance limited by *mechanical* delays (disk I/O)
  < 10% per year (IO per sec or MB per sec)
- Amdahl's Law: system speed-up limited by the slowest part!
  10% IO & 10x CPU => 5x Performance (lose 50%)
  10% IO & 100x CPU => 10x Performance (lose 90%)
- I/O bottleneck:
  Diminishing fraction of time in CPU
  Diminishing value of faster CPUs

Two points of view for I/O

- Are CPUs ever idle?
- Throughput view: il multitasking eliminates the problem of I/O performance
- User's view: response time is the user's performance index (often only one task is running and waiting in multitasking!)

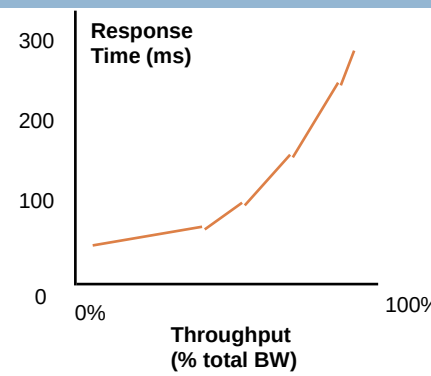
I/O Systems

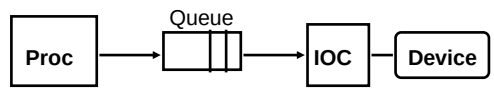


Time(workload) = Time(CPU) + Time(I/O) - Time(Overlap)

Disk I/O Performance

Metrics:
Response Time
Throughput

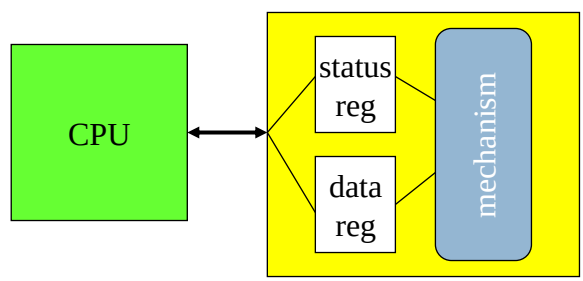




Response time = Queue + Device Service time

I/O devices

- Usually includes some non-digital component.
- Typical digital interface to CPU:

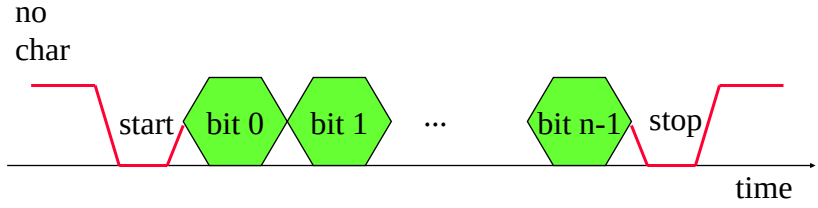


Application: 8251 UART

- Universal asynchronous receiver transmitter (UART) : provides serial communication.
- 8251 functions are integrated into standard PC interface chip.
- Allows many communication parameters to be programmed.

Serial communication

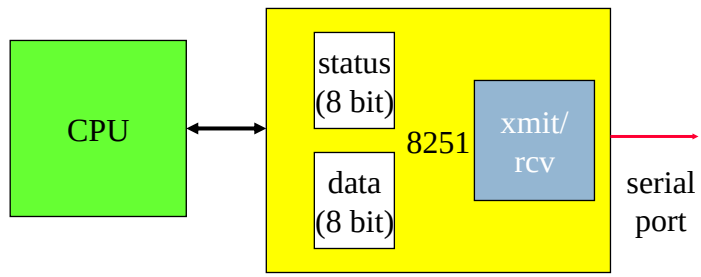
- Characters are transmitted separately:



Serial communication parameters

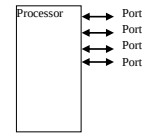
- Baud (bit) rate.
- Number of bits per character.
- Parity/no parity.
- Even/odd parity.
- Length of stop bit (1, 1.5, 2 bits).

8251 CPU interface

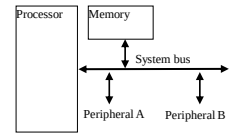


Microprocessor interfacing: I/O addressing

- A microprocessor communicates with other devices using some of its pins
  - Port-based I/O (parallel I/O)
    - Processor has one or more N-bit ports
    - Processor's software reads and writes a port just like a register
  - Bus-based I/O
    - Processor has address, data and control ports that form a single bus
    - Communication protocol is built into the processor
    - A single instruction carries out the read or write protocol on the bus



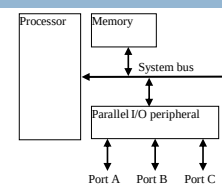
Parallel I/O



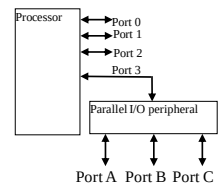
bus-based I/O processor

Compromises/extensions

- Parallel I/O peripheral
  - When processor only supports bus-based I/O but parallel I/O needed
  - Each port on peripheral connected to a register within peripheral that is read/written by the processor
- Extended parallel I/O
  - When processor supports port-based I/O but more ports needed
  - One or more processor ports interface with parallel I/O peripheral extending total number of ports available for I/O
  - e.g., extending 4 ports to 6 ports in figure



Adding parallel I/O to a bus-based I/O processor

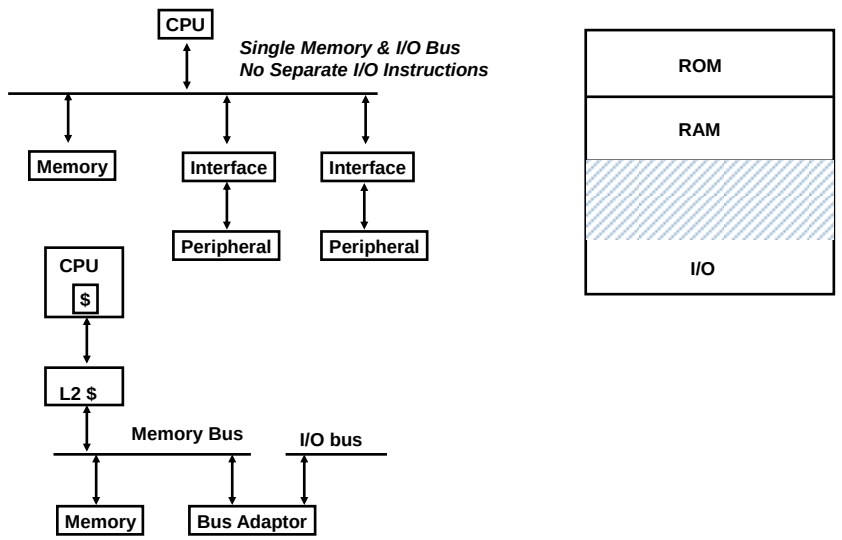


Extended parallel I/O

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

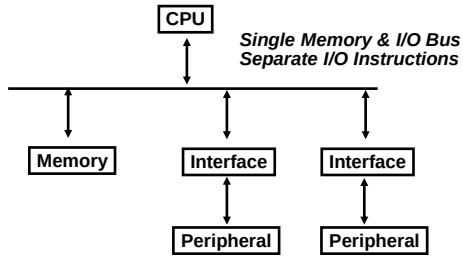
Memory Mapped I/O

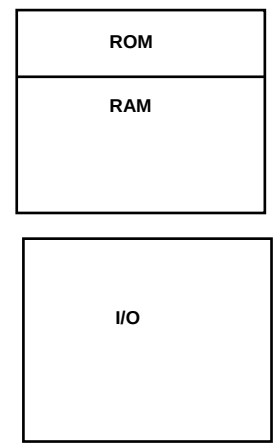


Types of bus-based I/O: memory-mapped I/O and standard I/O

- 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)

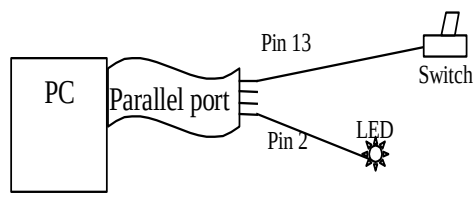




Memory-mapped I/O vs. Standard I/O

- 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

Intel x86 Standard I/O



LPT Connection Pin	I/O Direction	Register Address
1	Output	0^{th} bit of register #2
2-9	Output	0^{th} bit of register #2
10,11,12,13,15	Input	6,7,5,4,3^{th} bit of register #1
14,16,17	Output	1,2,3^{th} bit of register #2

Intel x86 Standard I/O

```
; This program consists of a sub-routine that reads the state of the input pin,
determining the on/off state of our switch and asserts the output pin, turning
the LED on/off accordingly
.386

CheckPort    proc
    push ax       ; save the content
    push dx           ; save the content
    mov  dx, 3BCh + 1 ; base + 1 for register #1
    in   al, dx       ; read register #1
    and  al, 10h  ; mask out all but bit # 4
    cmp  al, 0        ; is it 0?
    jne  SwitchOn ; if not, we need to turn the LED on
```


Intel x86 Standard I/O

```
SwitchOff:
    mov dx, 3BCh + 0 ; base + 0 for register #0
    in  al, dx       ; read the current state of the port
    and al, f7h  ; clear first bit (masking)
    out dx, al       ; write it out to the port
    jmp Done             ; we are done

SwitchOn:
    mov dx, 3BCh + 0 ; base + 0 for register #0
    in  al, dx       ; read the current state of the port
    or  al, 01h  ; set first bit (masking)
    out dx, al       ; write it out to the port

Done:   pop dx       ; restore the content
    pop ax       ; restore the content
CheckPort   endp
```

ARM memory-mapped I/O

- Define location for device:

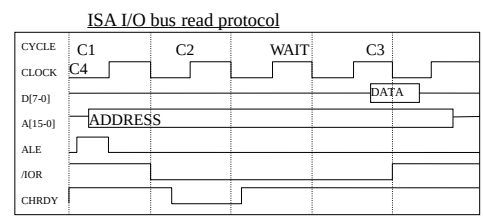
```
DEV1 EQU 0x1000
```

- Read/write code:

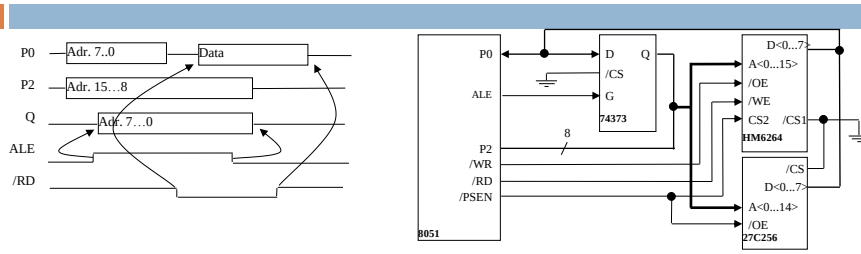
```
LDR r1,#DEV1 ; set up device adrs
LDR r0,[r1] ; read DEV1
LDR r0,#8 ; set up value to write
STR r0,[r1] ; write value to device
```


ISA bus

- ISA supports standard I/O
  - /IOR distinct from /MEMR for peripheral read
    - /IOW used for writes
  - 16-bit address space for I/O vs. 20-bit address space for memory
  - Otherwise very similar to memory protocol



A basic memory protocol



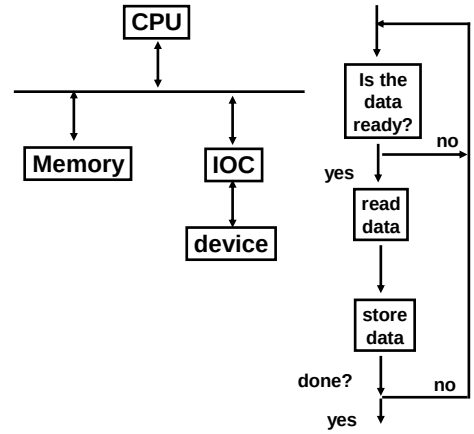
- Interfacing an 8051 to external memory
  - Ports P0 and P2 support port-based I/O when 8051 internal memory being used
  - Those ports serve as data/address buses when external memory is being used
  - 16-bit address and 8-bit data are time multiplexed; low 8-bits of address must therefore be latched with aid of ALE signal

I/O techniques

- Programmed I/O (Polling)
- Interrupt
- Direct Memory Access

Polling

- In questo caso la gestione dei dispositivi di I/O è totalmente demandata alla CPU.
- E' basato sulla scansione periodica 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.



Peek and poke

- Traditional HLL interfaces:

```
int peek(char *location) {
  return *location; }

void poke(char *location, char newval)
  {
  (*location) = newval; }
```

Busy/wait output

- Simplest way to program device.
  - Use instructions to test when device is ready.

```
current_char = mystring;
while (*current_char != '\0') {
  poke(OUT_CHAR,*current_char);
  while (peek(OUT_STATUS) != 0);
  current_char++;
}
```


Simultaneous busy/wait input and output

```
while (TRUE) {
  /* read */
  while (peek(IN_STATUS) == 0);
  achar = (char)peek(IN_DATA);
  /* write */
  poke(OUT_DATA,achar);
  poke(OUT_STATUS,1);
  while (peek(OUT_STATUS) != 0);
  }
```


Polling

- La maggior parte del tempo è impiegata dal programma principale nell'esecuzione del ciclo di polling.
- Viene normalmente utilizzato nei sistemi più piccoli e meno complessi, in quanto ha le seguenti caratteristiche:
  - poco costoso in termini di hardware
  - poco efficiente.

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*

Interrupts

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

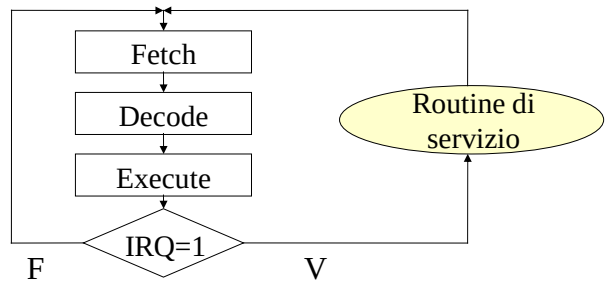
Tale meccanismo viene realizzato dal ***sistema di interruzione***

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



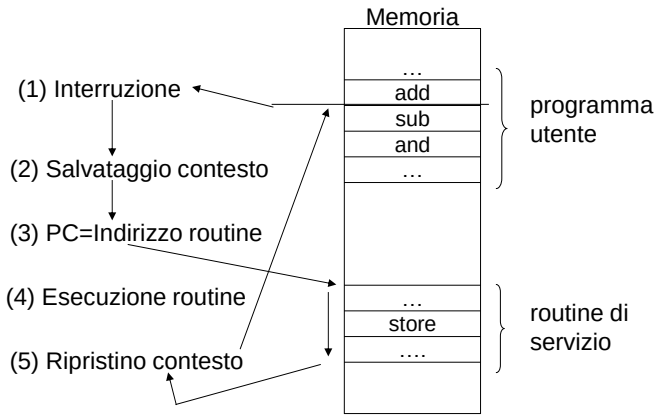
Funzioni del sistema di interruzione

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

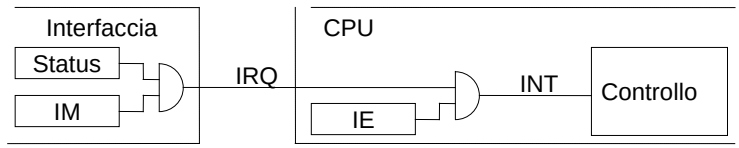
- PC e SR devono essere salvati via hardware.
- Esempio:
  - MEM[SP]=SR; SP--;
  - MEM[SP]=PC; SP--;

- Gli altri registri possono essere salvati via software
  - 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
- Esempio
  - 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

Additional interrupt issues

- Maskable vs. non-maskable interrupts
  - Maskable: programmer can set bit that causes processor to ignore interrupt
    - Important when in the middle of time-critical code
  - Non-maskable: a separate interrupt pin that can't be masked
    - Typically reserved for drastic situations, like power failure requiring immediate backup of data to non-volatile memory

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

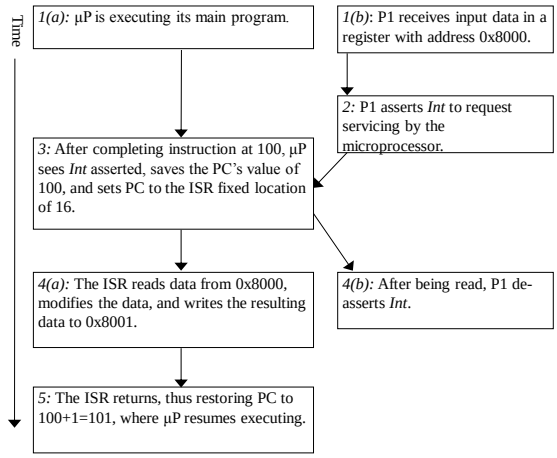
- In seguito alla richiesta di interruzione:
  1. viene salvato il contesto del programma interrotto
  2. Viene mandata in esecuzione una sequenza di identificazione dell'interruzione che interroga ad uno ad uno i dispositivi fino a trovare quello che ha fatto la richiesta
  3. Viene mandata in esecuzione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

Microprocessor interfacing: interrupts

- What is the address of the Interrupt Service Routine?
  - Fixed interrupt
    - Address built into microprocessor, cannot be changed
    - Either ISR stored at address or a jump to actual ISR stored if not enough bytes available
    - Multiple Int pins to support multiple peripherals or one pin and polling of the peripherals
  - Vectored interrupt
    - Peripheral must provide the address
    - Common when microprocessor has multiple peripherals connected by a system bus
  - Compromise: interrupt address table

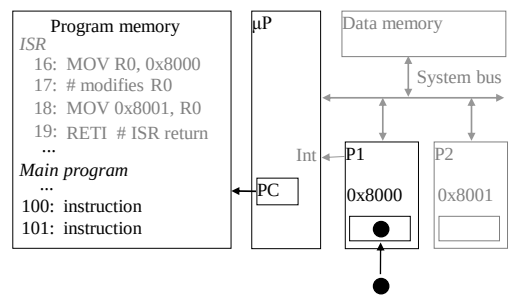
Interrupt-driven I/O using fixed ISR location



Interrupt-driven I/O using fixed ISR location

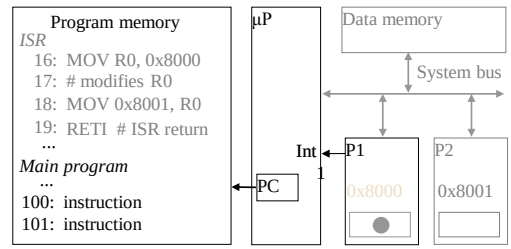
1(a): µP is executing its main program

1(b): P1 receives input data in a register with address 0x8000.



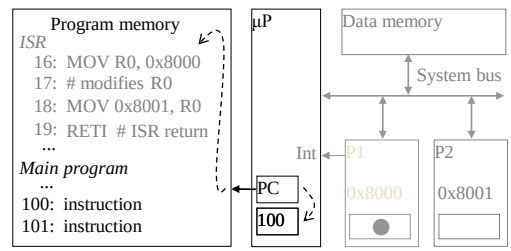
Interrupt-driven I/O using fixed ISR location

2: P1 asserts *Int* to request servicing by the microprocessor



Interrupt-driven I/O using fixed ISR location

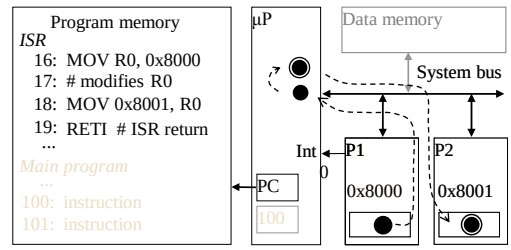
3: After completing instruction at 100, μP sees *Int* asserted, saves the PC's value of 100, and sets PC to the ISR fixed location of 16.



Interrupt-driven I/O using fixed ISR location

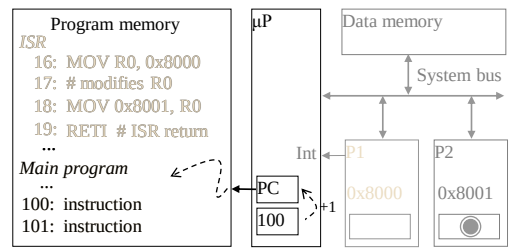
4(a): The ISR reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001.

4(b): After being read, P1 deasserts *Int*.



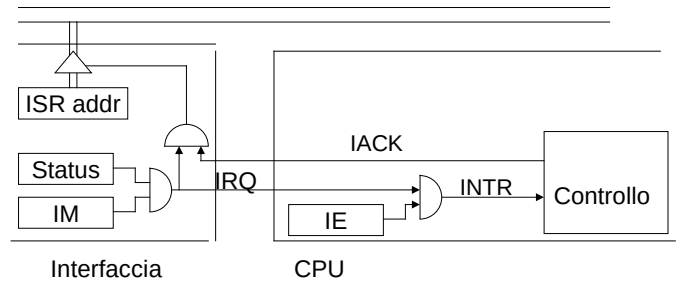
Interrupt-driven I/O using fixed ISR location

5: The ISR returns, thus restoring PC to 100+1=101, where μP resumes executing.

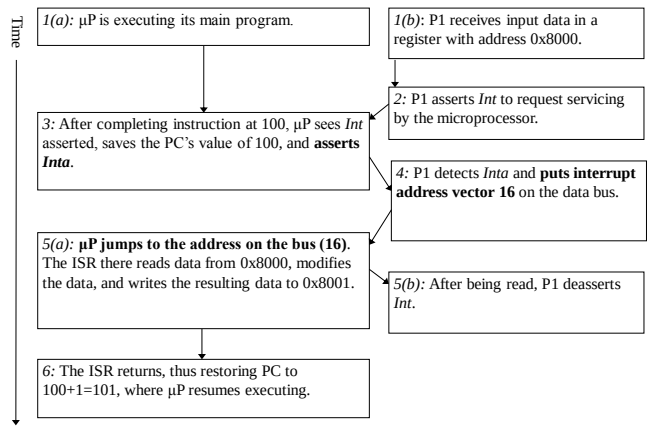


Riconoscimento mediante vettore delle interruzioni

- Nell'interfaccia è presente un registro in cui è memorizzato l'indirizzo della routine di servizio.
- In seguito alla richiesta di interruzione IRQ=1, se il processore è interrompibile (IE=1), attiva in risposta il segnale IACK.
- L'indirizzo della ISR viene inviato sul bus dati in risposta al segnale IACK generato dal processore.

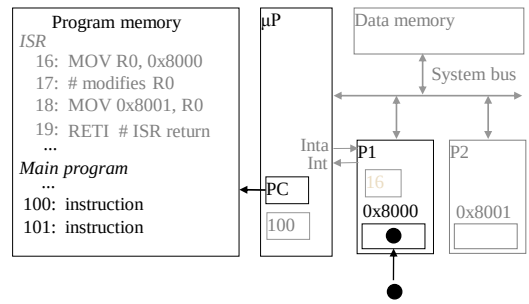


Interrupt-driven I/O using vectored interrupt



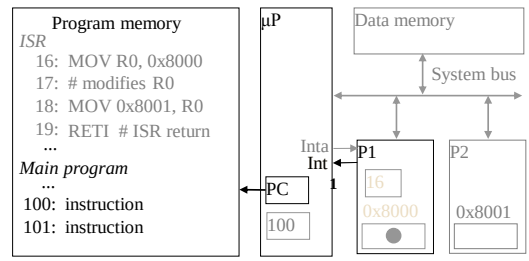
Interrupt-driven I/O using vectored interrupt

1(a): P is executing its main program
1(b): P1 receives input data in a register with address 0x8000.



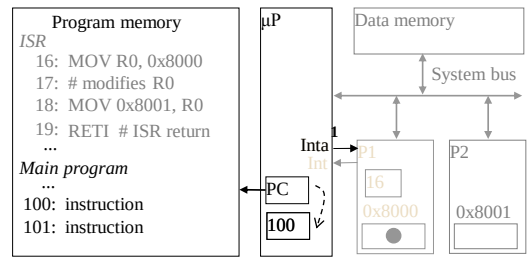
Interrupt-driven I/O using vectored interrupt

2: P1 asserts *Int* to request servicing by the microprocessor



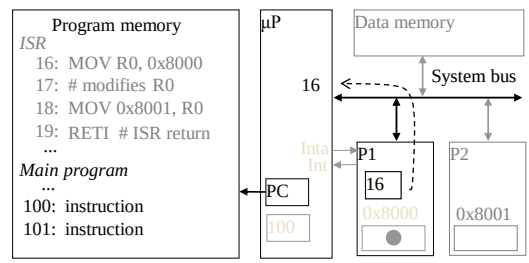
Interrupt-driven I/O using vectored interrupt

3: After completing instruction at 100, µP sees *Int* asserted, saves the PC's value of 100, and **asserts *Inta***



Interrupt-driven I/O using vectored interrupt

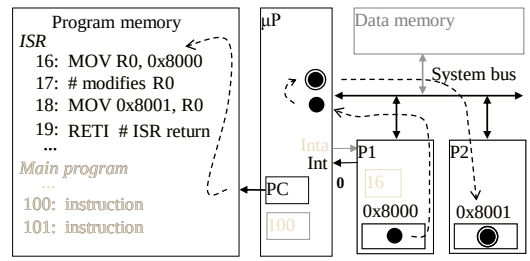
4: P1 detects *Inta* and puts **interrupt address vector 16** on the data bus



Interrupt-driven I/O using vectored interrupt

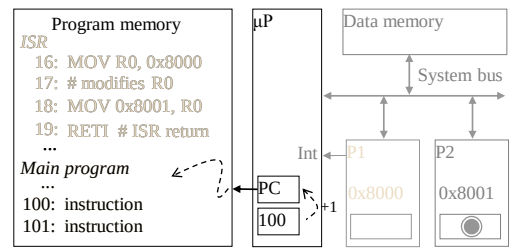
5(a): PC jumps to the address on the bus (16). The ISR there reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001.

5(b): After being read, P1 deasserts *Int*.



Interrupt-driven I/O using vectored interrupt

6: The ISR returns, thus restoring the PC to 100+1=101, where the µP resumes

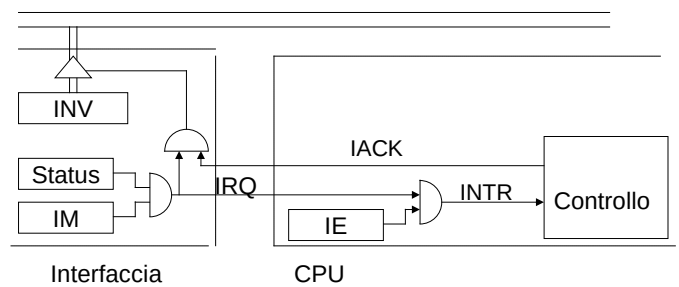


Interrupt address table

- Compromise between fixed and vectored interrupts
  - One interrupt pin
  - Table in memory holding ISR addresses (maybe 256 words)
  - Peripheral doesn't provide ISR address, but rather index into table
    - Fewer bits are sent by the peripheral
    - Can move ISR location without changing peripheral

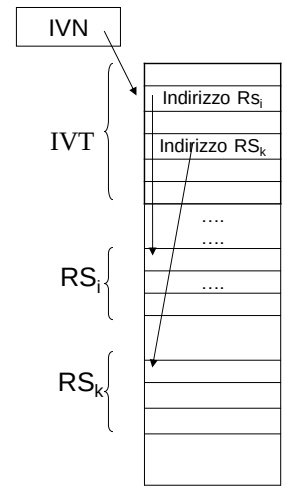
Riconoscimento mediante vettore delle interruzioni

- 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.

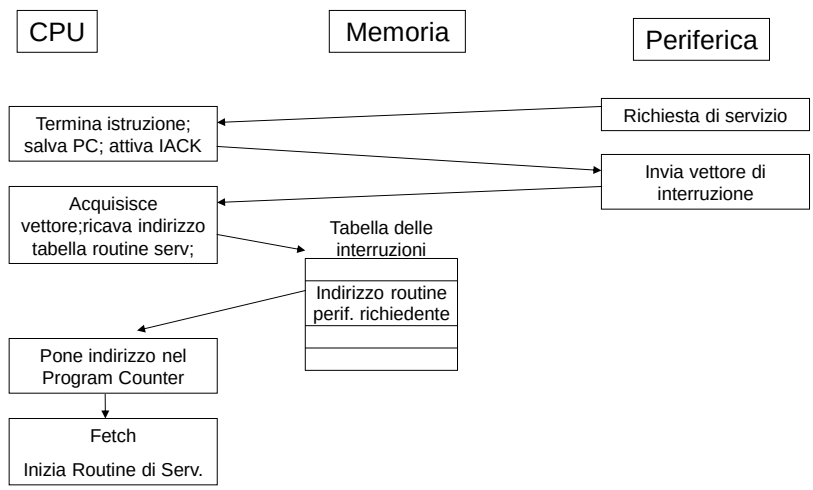


Riconoscimento mediante vettore delle interruzioni

- 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.
- 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

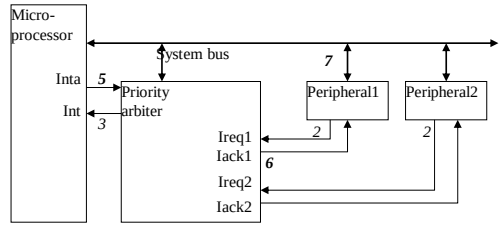


Gestione dei conflitti tra richieste di interruzione

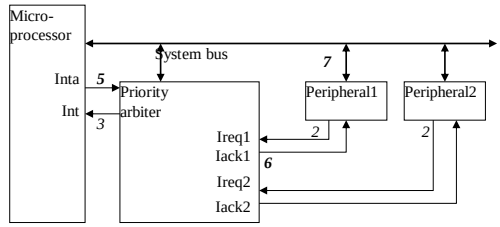
- Più richieste di interruzioni possono entrare in conflitto
- Cause conflitto:
  - Richieste contemporanee
  - Richieste quando è in esecuzione la routine di servizio di una interruzione

Arbitration: Priority arbiter

- Consider the situation where multiple peripherals request service from single resource (e.g., microprocessor, DMA controller) 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



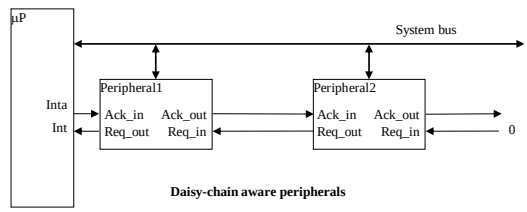
1. 1. Microprocessor is executing its program.
2. 2. Peripheral1 needs servicing so asserts *Ireq1*. Peripheral2 also needs servicing so asserts *Ireq2*.
3. 3. Priority arbiter sees at least one *Ireq* input asserted, so asserts *Int*.
4. 4. Microprocessor stops executing its program and stores its state.
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

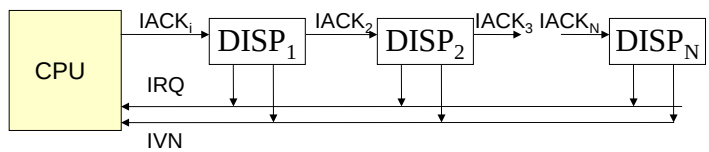
- Types of priority
  - Fixed priority
    - each peripheral has unique rank
    - highest rank chosen first with simultaneous requests
    - preferred when clear difference in rank between peripherals
  - Rotating priority (round-robin)
    - priority changed based on history of servicing
    - better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

Arbitration: Daisy-chain arbitration

- Arbitration done by peripherals
  - Built into peripheral or external logic added
    - *req* input and *ack* output added to each peripheral
- Peripherals connected to each other in daisy-chain manner
  - One peripheral connected to resource, all others connected "upstream"
  - Peripheral's *req* flows "downstream" to resource, resource's *ack* flows "upstream" to requesting peripheral
  - Closest peripheral has highest priority



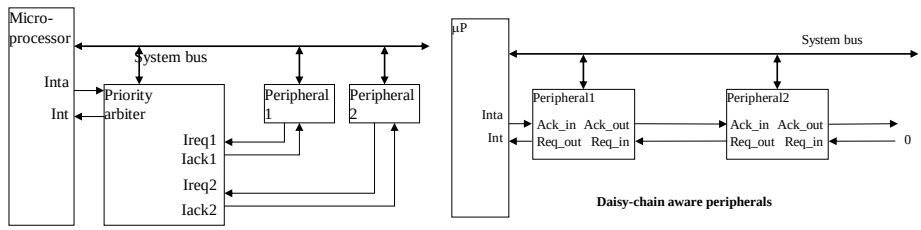
Interruzione vettorizzata con Daisy chain



- 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
- **Problema:** 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.

Arbitration: Daisy-chain arbitration

- Pros/cons
  - Easy to add/remove peripheral - no system redesign needed
  - Does not support rotating priority
  - One broken peripheral can cause loss of access to other peripherals



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

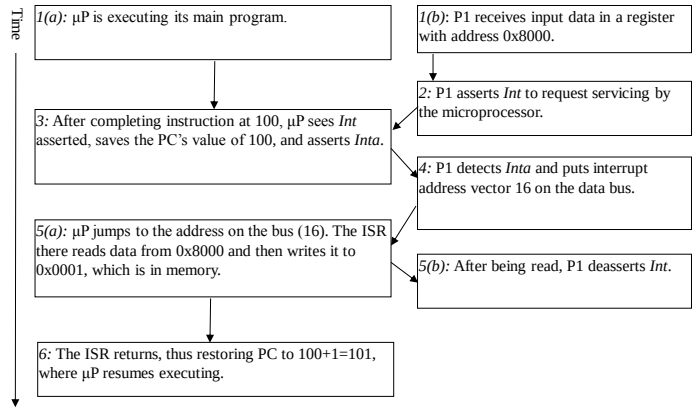
Approccio hardware

- 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

Direct memory access

- Buffering
  - Temporarily storing data in memory before processing
  - Data accumulated in peripherals commonly buffered
- Microprocessor could handle this with ISR
  - Storing and restoring microprocessor state inefficient
  - Regular program must wait
- DMA controller more efficient
  - Separate single-purpose processor
  - Microprocessor relinquishes control of system bus to DMA controller
  - Microprocessor can meanwhile execute its regular program
    - No inefficient storing and restoring state due to ISR call
    - Regular program need not wait unless it requires the system bus
      - Harvard archictecture – processor can fetch and execute instructions as long as they don't access data memory – if they do, processor stalls

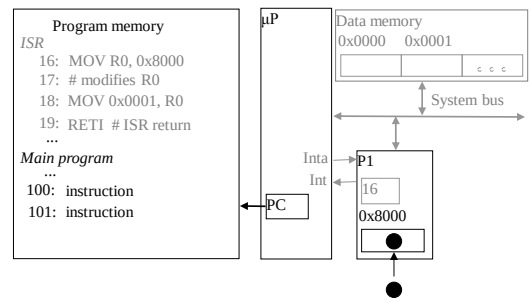
Peripheral to memory transfer *without* DMA, using vectored interrupt



Peripheral to memory transfer *without* DMA, using vectored interrupt

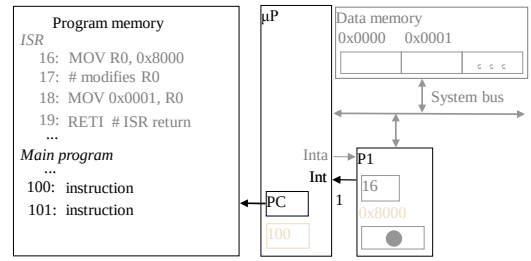
1(a): µP is executing its main program

1(b): P1 receives input data in a register with address 0x8000.



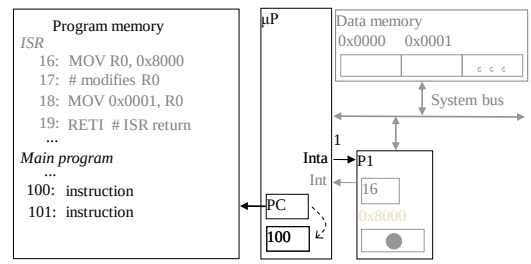
Peripheral to memory transfer *without* DMA, using vectored interrupt

2: P1 asserts *Int* to request servicing by the microprocessor



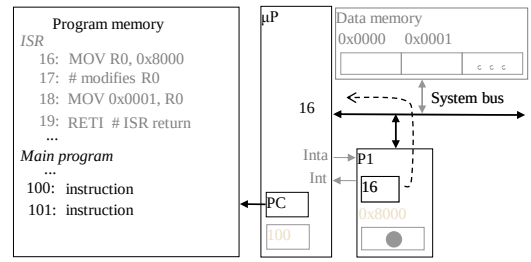
Peripheral to memory transfer *without* DMA, using vectored interrupt

3: After completing instruction at 100, µP sees *Int* asserted, saves the PC's value of 100, and asserts *Inta*.



Peripheral to memory transfer *without* DMA, using vectored interrupt (cont')

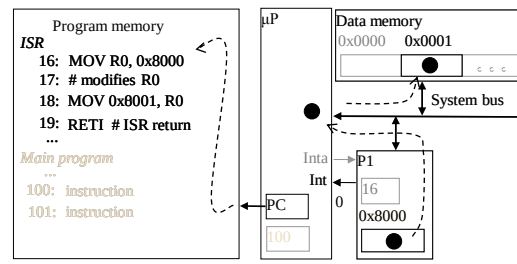
4: P1 detects *Inta* and puts interrupt address vector 16 on the data bus.



Peripheral to memory transfer *without* DMA, using vectored interrupt (cont')

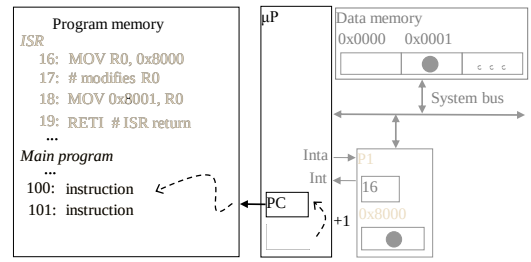
5(a): µP jumps to the address on the bus (16). The ISR there reads data from 0x8000 and then writes it to 0x0001, which is in memory.

5(b): After being read, P1 de-asserts *Int*.

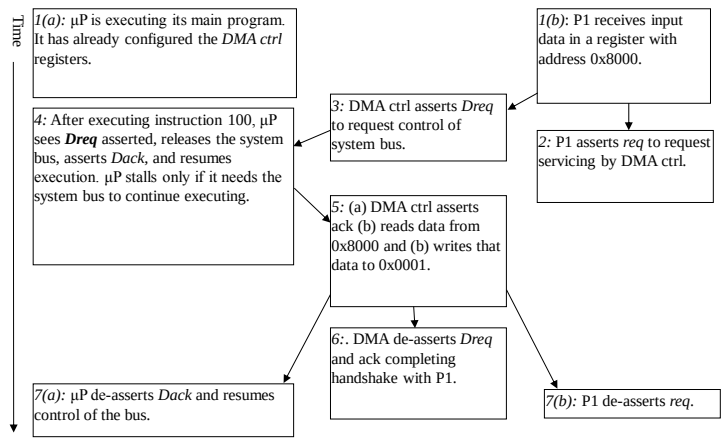


Peripheral to memory transfer *without* DMA, using vectored interrupt (cont')

6: The ISR returns, thus restoring PC to 100+1=101, where µP resumes executing.



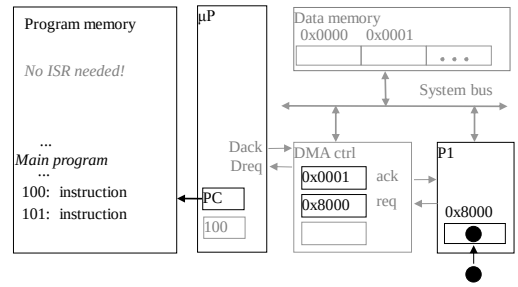
Peripheral to memory transfer with DMA



Peripheral to memory transfer with DMA (cont')

1(a): µP is executing its main program. It has already configured the DMA ctrl registers

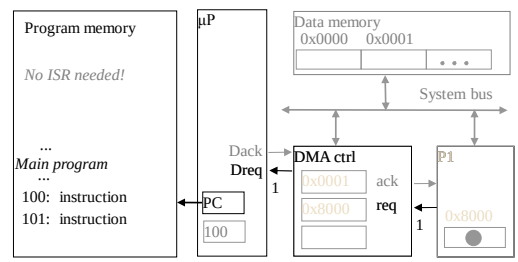
1(b): P1 receives input data in a register with address 0x8000.



Peripheral to memory transfer with DMA (cont')

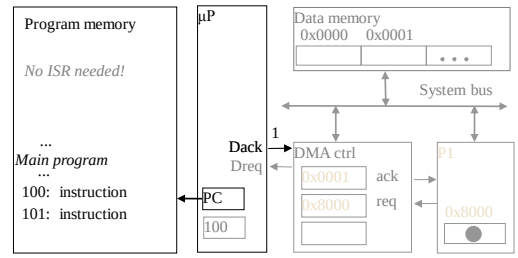
2: P1 asserts *req* to request servicing by DMA ctrl.

3: DMA ctrl asserts *Dreq* to request control of system bus



Peripheral to memory transfer with DMA (cont')

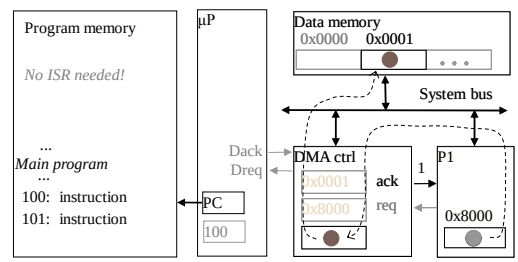
4: After executing instruction 100, µP sees *Dreq* asserted, releases the system bus, asserts *Dack*, and resumes execution, µP stalls only if it needs the system bus to continue executing.



Peripheral to memory transfer with DMA (cont')

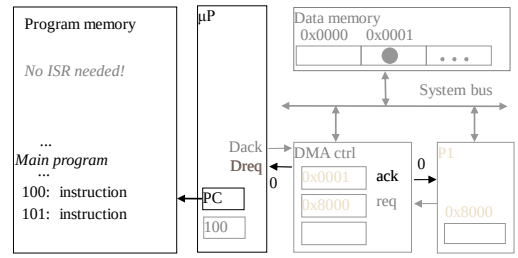
5: DMA ctrl (a) asserts ack, (b) reads data from 0x8000, and (c) writes that data to 0x0001.

(Meanwhile, processor still executing if not stalled!)

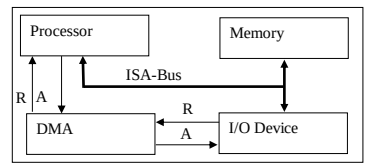


Peripheral to memory transfer with DMA (cont')

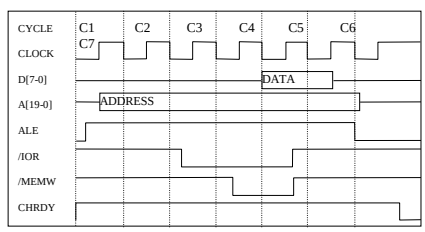
6: DMA de-asserts *Dreq* and *ack* completing the handshake with P1.



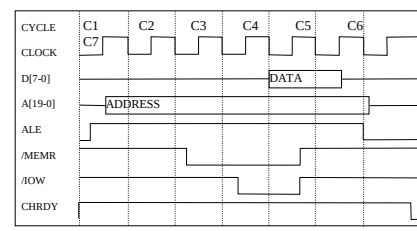
ISA bus DMA cycles



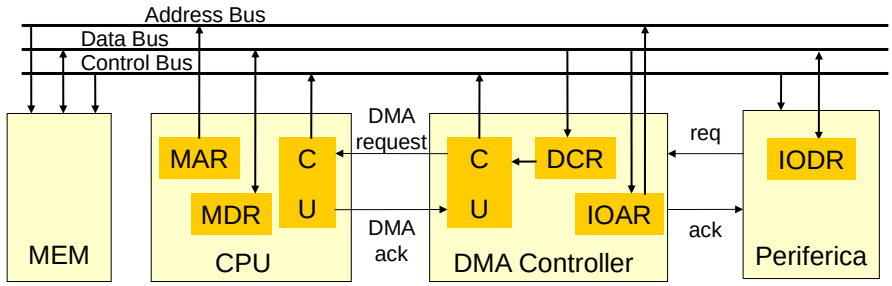
DMA Memory-Write Bus Cycle



DMA Memory-Read Bus Cycle



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:
  - la CPU carica nei registri IOAR e DCR l'indirizzo dell'area di memoria ed il numero di parole da trasferire;
  - 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;
  - il DMA Controller esegue il trasferimento; dopo il trasferimento di ciascuna parola, IOAR e DCR vengono decrementati;

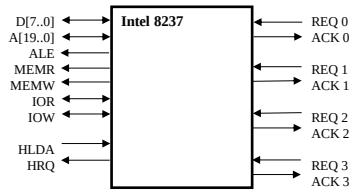
Trasferimento in DMA

- 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

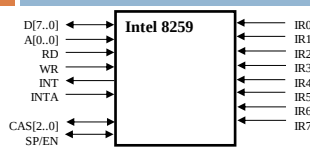
- **Il trasferimento dei dati in DMA può avvenire in vari modi:**
  - **trasferimento a blocchi *(burst transfer)***
    - Prevede che il DMA Controller, una volta acquisito il controllo del bus, lo mantenga per tutto il tempo richiesto per trasferire un blocco di dati.
    - Il trasferimento a blocchi 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.

Intel 8237 DMA controller



Signal	Description
D[7..0]	These wires are connected to the system bus (ISA) and are used by the microprocessor to write to the internal registers of the 8237.
A[19..0]	These wires are connected to the system bus (ISA) and are used by the DMA to issue the memory location where the transferred data is to be written to. The 8237 is
ALE*	This is the address latch enable signal. The 8237 use this signal when driving the system bus (ISA).
MEMR*	This is the memory write signal issued by the 8237 when driving the system bus (ISA).
MEMW*	This is the memory read signal issued by the 8237 when driving the system bus (ISA).
IOR*	This is the I/O device read signal issued by the 8237 when driving the system bus (ISA) in order to read a byte from an I/O device
IOW*	This is the I/O device write signal issued by the 8237 when driving the system bus (ISA) in order to write a byte to an I/O device.
HLDA	This signal (hold acknowledge) is asserted by the microprocessor to signal that it has relinquished the system bus (ISA).
HRQ	This signal (hold request) is asserted by the 8237 to signal to the microprocessor a request to relinquish the system bus (ISA).
REQ 0,1,2,3	An attached device to one of these channels asserts this signal to request a DMA transfer.
ACK 0,1,2,3	The 8237 asserts this signal to grant a DMA transfer to an attached device to one of these channels.
*See the ISA bus description in this chapter for complete details.	

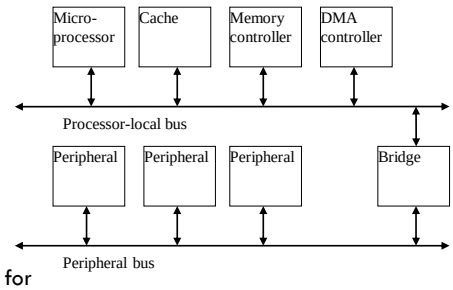
Intel 8259 programmable priority controller



Signal	Description
D[7..0]	These wires are connected to the system bus and are used by the microprocessor to write or read the internal registers of the 8259.
A[0..0]	This pin actis in cunjunction with WR/RD signals. It is used by the 8259 to decipher various command words the microprocessor writes and status the microprocessor wishes to read.
WR	When this write signal is asserted, the 8259 accepts the command on the data line, i.e., the microprocessor writes to the 8259 by placing a command on the data lines and asserting this signal.
RD	When this read signal is asserted, the 8259 provides on the data lines its status, i.e., the microprocessor reads the status of the 8259 by asserting this signal and reading the data lines.
INT	This signal is asserted whenever a valid interrupt request is received by the 8259, i.e., it is used to interrupt the microprocessor.
INTA	This signal, is used to enable 8259 interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the microprocessor.
IR 0,1,2,3,4,5,6,7	An interrupt request is executed by a peripheral device when one of these signals is asserted.
CAS[2..0]	These are cascade signals to enable multiple 8259 chips to be chained together.
SP/EN	This function is used in conjunction with the CAS signals for cascading purposes.

Multilevel bus architectures

- Don't want one bus for all communication
  - Peripherals would need high-speed, processor-specific bus interface
    - excess gates, power consumption, and cost; less portable
  - Too many peripherals slows down bus
- Processor-local bus
  - High speed, wide, most frequent communication
  - Connects microprocessor, cache, memory controllers, etc.
- Peripheral bus
  - Lower speed, narrower, less frequent communication
  - Typically industry standard bus (ISA, PCI) for portability
- Bridge
  - Single-purpose processor converts communication between busses



Network-oriented arbitration

- When multiple microprocessors share a bus (sometimes called a network)
  - Arbitration typically built into bus protocol
  - Separate processors may try to write simultaneously causing collisions
    - Data must be resent
    - Don't want to start sending again at same time
      - statistical methods can be used to reduce chances
- Typically used for connecting multiple distant chips
  - Trend – use to connect multiple on-chip processors