I/O con DMA

Argomento:

- DMA
- DMA Cache
- Advanced I/0

Materiale di studio

- Capitolo 7 (da 7.5 a 7.8)

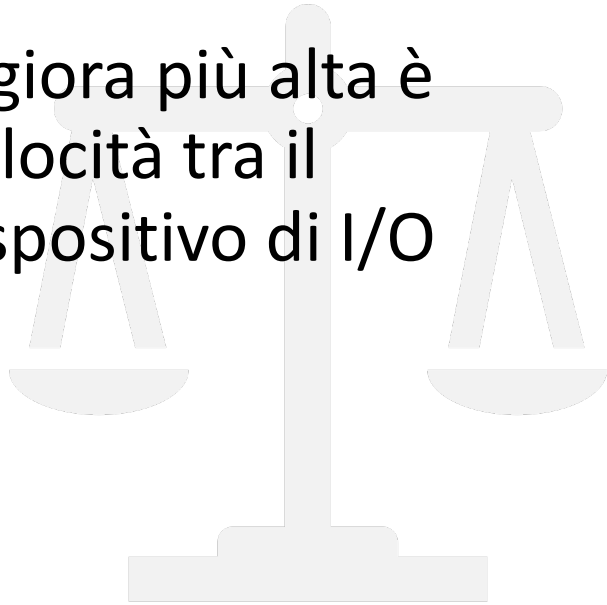
I/O Programmato: pro e contro

PRO

- Molto facile da realizzare
- Tempo di risposta molto veloce(*): appena il dato è pronto viene letto / inviato

CONTRO

- Mentre è in attesa che il dispositivo sia pronto, il processore non può fare niente altro!
- L'inefficienza peggiora più alta è la differenza di velocità tra il processore e il dispositivo di I/O



(*) se il dispositivo è pronto …

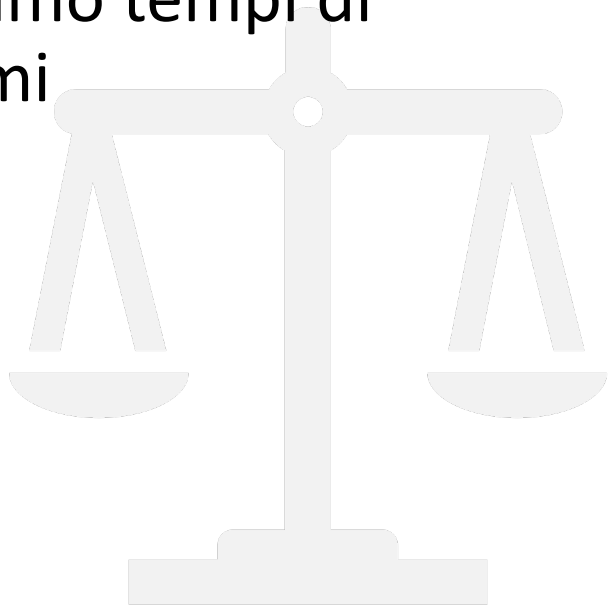
I/O a Interruzioni: pro e contro

CONTRO

- Sistema più complesso
- È necessario software dedicato
- È necessario hardware dedicato

PRO

- Efficiente: il processore non deve stare in busy waiting
- Usando HW nell'identificazione e gerarchia, abbiamo tempi di risposta rapidissimi
  - Hardware poll
  - Vectored IRQ
  - Banked registers



Performance dell'I/O

- Parlando di gestione di I/O ci interessa
  - L'impatto sulle performance dell'elaboratore
  - L'efficienza, ovvero il tempo di risposta

- Parlando di I/O un altro parametro che ci interessa è il «**data rate**», ovvero quanti dati al secondo si possono trasferire
- In un sistema a interrupt in cui la gestione dell'I/O in totale richiede 10µs (tempo di esecuzione della RSI, salvataggio del contesto compreso), quanti dati ci permette di trasferire?

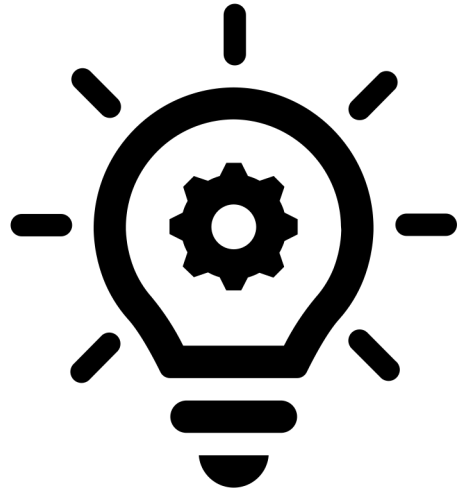
Data rate

- Ad ogni esecuzione la RSI gestisce un tot limitato di dati
- Se una RSI impiega 10µs per essere eseguita, potrà essere eseguita al massimo 10^5 volte in un secondo
  - Se ad ogni esecuzione la RSI gestisce ad es. 16 bit, il limite massimo del data rate sarà di $2 \cdot 10^5$ B/s

  Nota: a questa velocità, il processore sarà **costantemente** impegnato a gestire gli IRQ e non potrà fare altro!
  - Se il device produce più di 10^5 dati/s il processore non riuscirà a leggerli tutti e avremo una **perdita di dati** !
  - Dimezzando la velocità, a $10^5/2$ dati/s, il processore sarà comunque impegnato per il 50% del tempo!

Una nuova idea!

- Per il trasferimento di dati da/verso la memoria ad alta velocità il sistema ad interrupt non va bene.
- Il problema è che il processore viene interrotto ogni volta che una piccola parte dei dati è pronta



- Ma se il compito del processore è solo quello di prelevare i dati dal dispositivo e metterli in memoria (o viceversa), questo semplice compito non potrebbe farlo qualcun altro?

DMA – Direct Memory Access

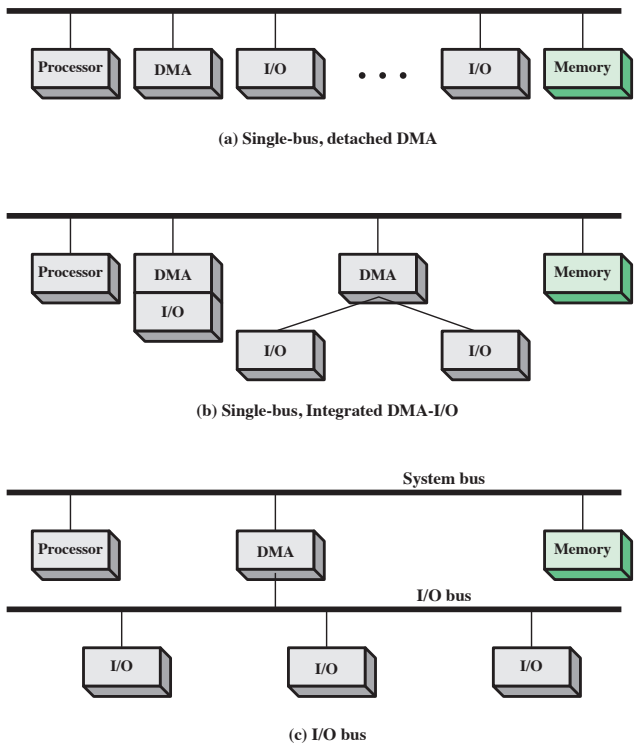
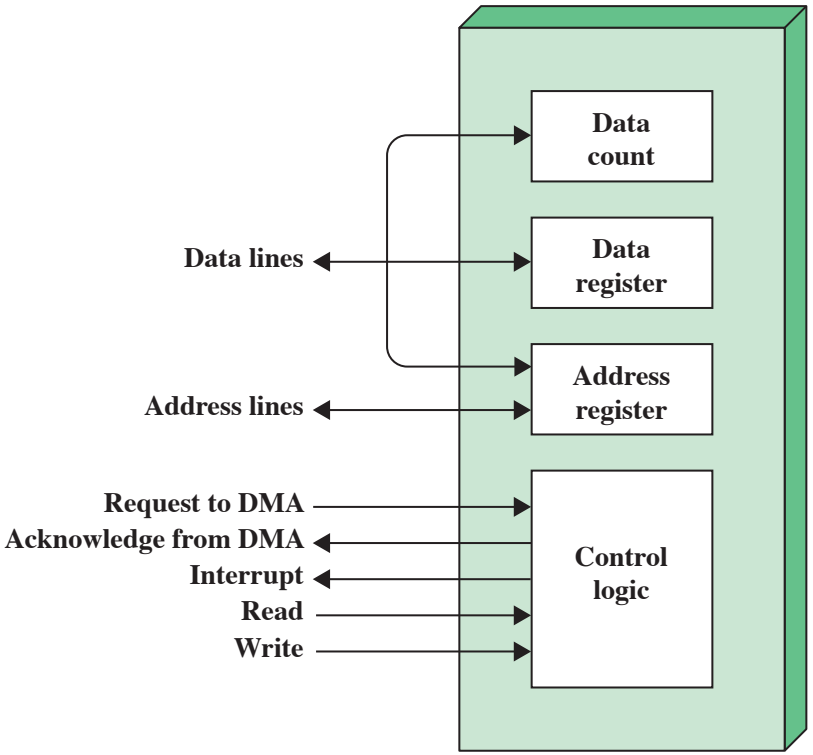


Figure 7.14 Alternative DMA Configurations

- Aggiungiamo al nostro elaboratore un DMA Controller
- Il DMA ha accesso ai dispositivi di I/O e alla memoria tramite il bus
- Il processore gli dice quanti dati vuole leggere, da che dispositivo e la posizione iniziale dove scriverli in memoria
- Il DMA si occupa del trasferimento e quando ha terminato avverte il processore con un IRQ

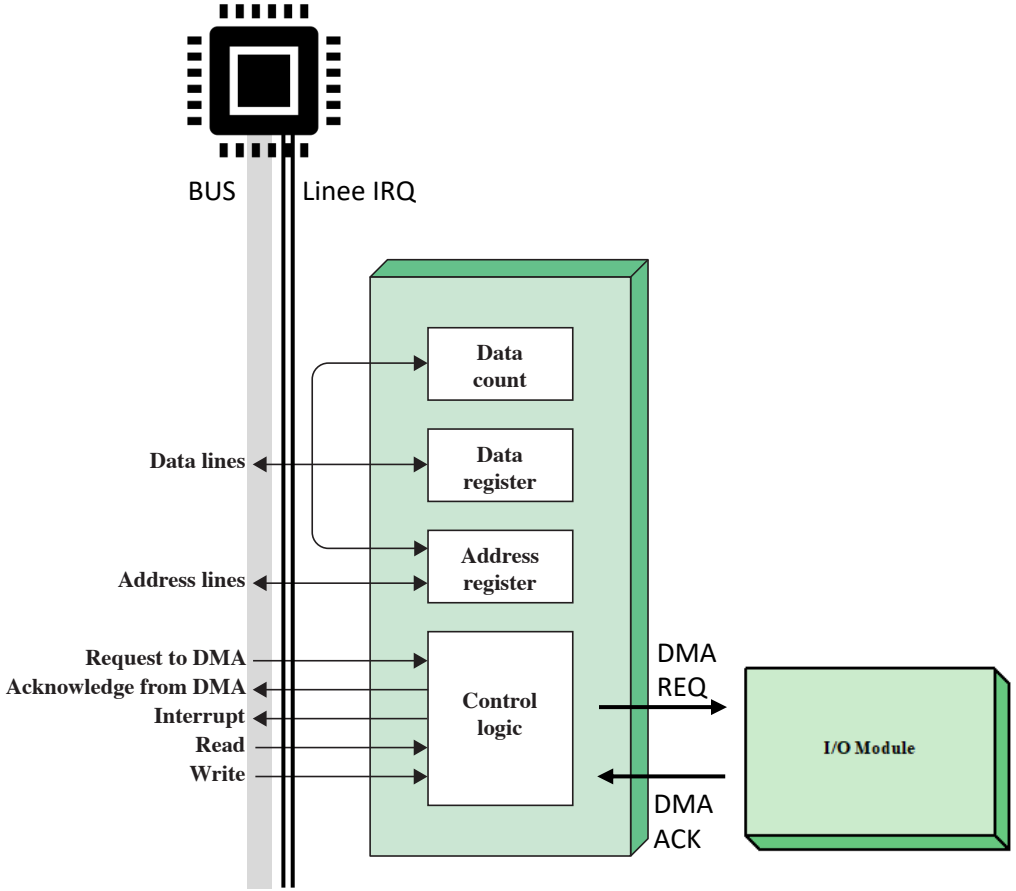
DMA Controller



- Al suo interno troviamo:
  - Un registro per l'indirizzo di memoria
  - Un registro per il numero di dati da trasferire
- Un DMA controller può controllare uno o più dispositivi di I/O
- È di fatto un piccolo processore

DMA in (input)

- Il processore indica
  - Device di I/O su cui (leggere)
  - Quanti dati trasferire
  - Indirizzo di memoria da cui (scrivere)
  - Da il comando START
- Il controller
  - È in busy waiting sul dispositivo
  - Trasferisce il dato appena pronto
  - Decrementa il contatore dei dati e aggiorna il puntatore a memoria
  - Quando il contatore è zero, avverte il processore con un IRQ



Accesso al bus

- Per trasferire i dati, il DMA deve passare per il BUS, che però è condiviso con il processore
- Il DMA avverte il processore quando gli serve il bus, con un comando HOLD request
- Il processore consente l'accesso con un comando HOLD ack

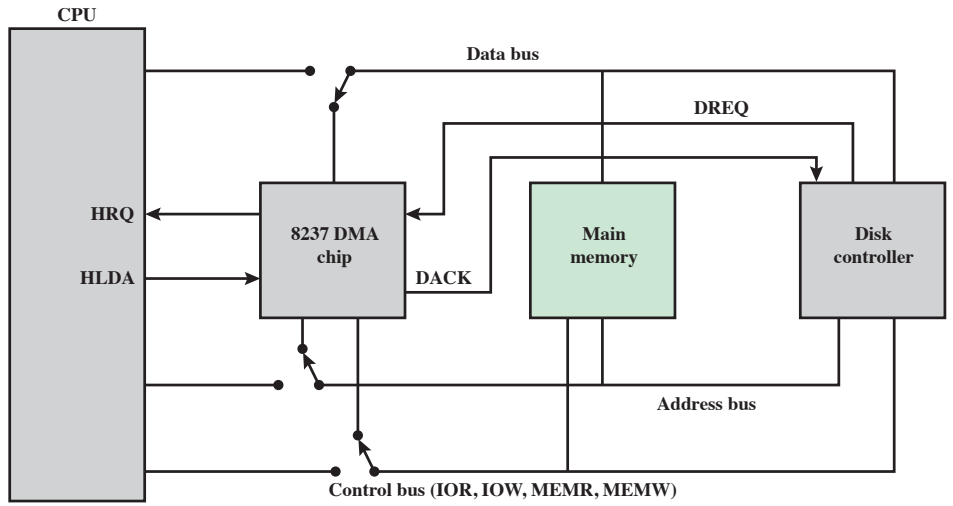


Figure 7.15 8237 DMA Usage of System Bus

Arbitraggio del bus

- Può capitare che il bus serva contemporaneamente sia alla CPU che al DMA:
- in questo caso tipicamente si da precedenza al DMA (la perdita di dati è considerata più grave):
  - Il DMA procede a trasferire i dati
  - Il processore dovrà attendere per accedere alla memoria

Arbitraggio del bus: cycle stealing

Il processore accede alla memoria per il fetch: se il fetch non si può fare, il processore non ha nulla su cui lavorare!

→Quando il bus è occupato dal DMA, il processore è di fatto fermo: Il processore perde un ciclo «cycle stealing»

- Per evitare che il processore resti bloccato troppo a lungo, il trasferimento non viene fatto tutto di un colpo (burst) ma viene fatto un dato alla volta: l'arbitraggio del bus è fatto ad ogni ciclo di memoria

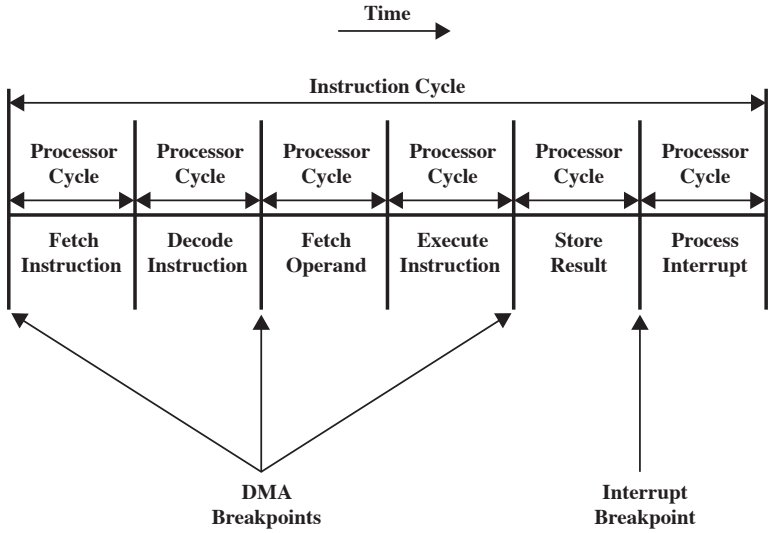


Figure 7.13 DMA and Interrupt Breakpoints During an Instruction Cycle

I/O con DMA: pro e contro

CONTRO

- Sistema più complesso
- È necessario software dedicato
- È necessario hardware dedicato
- Impegna il bus di sistema

PRO

- Permette il trasferimento ad alta velocità di dati dai dispositivi di I/O alla memoria (e viceversa)
- Il processore è impegnato solo quando il trasferimento è stato completato

Sistemi di I/O avanzato

- Fly-by DMA: i dati in transito non vengono memorizzati temporaneamente sul DMA
- DDIO DMA con accesso alla cache (più esterna) del processore (per operazioni di output): tipicamente infatti i dati da scrivere sono appena stati prodotti e sono ancora in cache
- I/O channels: il controller di I/O è un piccolo processore che può eseguire istruzioni e la CPU gli indica delle routine da eseguire; il controller avverte la CPU quando ha terminato
- I/O processor: il controller è a tutti gli effetti un piccolo elaboratore in grado di controllare autonomamente vari dispositivi senza l'intervento della CPU

Sistemi multiprocessore

- Abbiamo visto una tecnica con cui condividere il bus (e la memoria) tra due dispositivi: il DMA controller e la CPU
- Con la stessa tecnica possiamo pensare di condividere queste risorse con altri dispositivi, da esempio dei processori supplementari:
  - Processori ausiliari (es. specializzati per la grafica)
  - Processori paritari multipli (multiprocessore), che permettono il calcolo parallelo e la ridondanza
  - Acceleratori hardware (e.g., per machine learning)