

Calcolatori Elettronici L-A

Obiettivo del modulo è lo studio dei seguenti aspetti dell'hardware dei calcolatori:

- l'architettura
- i principi di funzionamento
- le tecniche di progettazione
- l'impatto dell'architettura sulle prestazioni

si consiglia di seguire il corso di Calcolatori Elettronici L-A solamente dopo aver studiato Reti Logiche L-A

Modalità di svolgimento dell'esame: prove intermedie e prova finale

Esame finale costituito da tre esercizi:

- 1) esercizio sull'ISA "DLX"
- 2) esercizio di progetto sul processore 8088 e suo linguaggio assembler
- 3) domanda di "teoria"

Calcolatori LA - Materiale didattico

Le dispense sono più che sufficienti per una ottima preparazione

<http://www.ingce.unibo.it> → **selez. il proprio C.d.L.** → **piano degli studi** →
Calcolatori Elettronici L A

Gli argomenti sono suddivisi in files.pdf (Acrobat Reader). Per ogni file esistono due formati, 1 slide per facciata con spazio per appunti e 2 slide per facciata.

Per approfondimenti:

John L. **Hennessy**, David A. **Patterson**

ARCHITETTURE DEI CALCOLATORI, METODI DI VALUTAZIONE E PROGETTO - ISBN 88-08-14198-5

1997 - **Zanichelli** - □ **64,00** (nel 2002) - **Capp. 1..5**

Libro ben fatto a cui si ispira la parte del corso sul DLX, un po' datato tecnicamente ma didatticamente valido da punto di vista della impostazione del progettista. Costoso. Traduzione dell'originale "Computer architecture, a quantitative approach" edito in inglese da Morgan Kauffman Publishers, da NON confondere con un libro degli stessi autori edito in italiano da Zanichelli ma con TITOLO DIVERSO.

Giacomo **Bucci**

ARCHITETTURE DEI CALCOLATORI ELETTRONICI - ISBN 88-386-0889-X

2001 - **McGraw Hill** Libri Italia - □ **39,00** (nel 2002) - **Capp. 1..9**

Libro ben fatto, originale italiano, molto più aggiornato e con un approccio molto più manualistico (ancora più utile dopo la laurea)

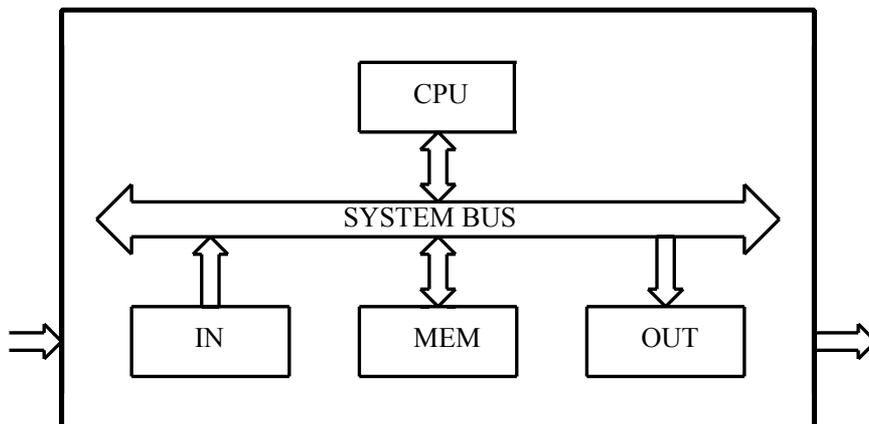
Il modello di Von Neumann (1)

- La macchina di Von Neumann
 - Funzionamento
 - Architettura dell'hardware
 - Prestazioni

Il modello di Von Neumann (2)

- Architettura del tutto generale che porta a realizzazioni poco dipendenti dal funzionamento desiderato
- il funzionamento è quello di una **macchina digitale a esecuzione sequenziale e programma memorizzato**
 - * **sequenza di istruzioni (programma)**
 - * **memorizzate** su un supporto di memoria
- Per cambiare funzionamento e' sufficiente cambiare il programma
- L'architettura è adatta a trattare problemi molto piu' complessi di quelli visti nel corso di reti logiche ma con **efficienza** molto minore
- l'importanza e la diffusione dei calcolatori dipende fortemente dall'unicità di questo modello rappresentato visivamente nel prossimo lucido

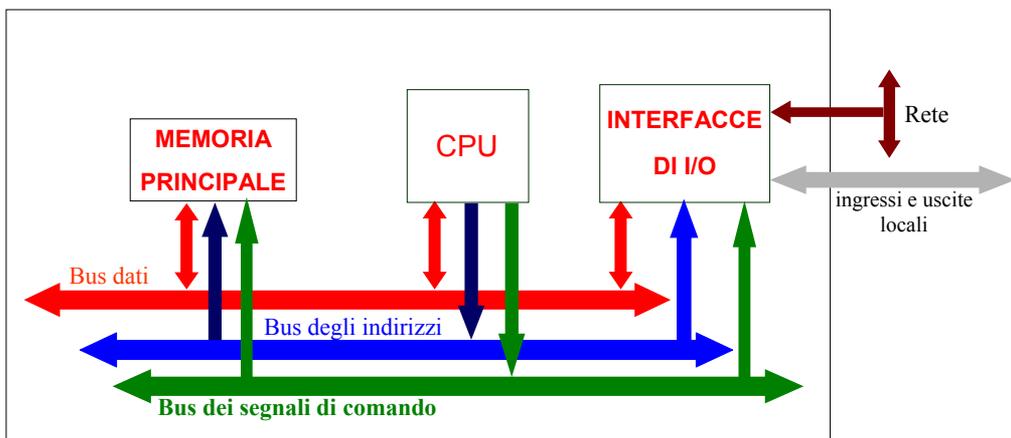
Sistemi a microprocessore



Struttura del bus di sistema

- I segnali del bus di sistema sono suddivisi in tre gruppi:
 - bus dati, bus degli indirizzi e bus dei segnali di comando
- il bus dati è costituito dai segnali che portano istruzioni e operandi; di solito il suo parallelismo è multiplo di un byte secondo una potenza di 2 (es. 1, 2, 4, 8, 16 byte); il **parallelismo del bus dati** è un altro **parametro caratteristico dell'architettura** della CPU. Il bus dati è identificato dal vettore di bit **D[m-1..0]**
- il bus degli indirizzi è costituito dai segnali che identificano la posizione delle informazioni trasferite nello spazio di indirizzamento a cui si intende accedere; il bus degli indirizzi è solitamente identificato dal vettore di bit **A[n-1..0]**
- il bus dei segnali di comando è composto dai segnali che comandano i trasferimenti di dati sul bus; esempi di segnali di comando sono:
 - il comando con cui la CPU esegue una lettura nello spazio di indirizzamento in memoria (**MRDC#**)
 - il comando con cui la CPU esegue una scrittura in memoria (**MWRC#**)
 - il comando con cui la CPU esegue una lettura nello spazio di indirizzamento in I/O (**IORDC#**)
 - il comando con cui la CPU esegue una scrittura in I/O (**IOWRC#**)

Bus di un sistema a μP



In questo schema a blocchi la CPU genera i segnali di indirizzo e di comando per la memoria e le interfacce

Accesso a dati e istruzioni da parte della CPU indirizzamento della memoria

- Durante l'esecuzione del programma la CPU legge le istruzioni e scambia dati e altre informazioni con la memoria principale attraverso il bus
- La memoria principale è vista dalla CPU come un **vettore** $M[0..2^n-1]$ di 2^n elementi detti **celle** o **parole** di memoria; questo **vettore** è detto "spazio di indirizzamento in memoria"
- L'indice i che **identifica la cella** $M[i]$ si chiama **indirizzo della cella**
- L'accesso da parte della CPU all'informazione (istruzione o operando) residente in $M[i]$ avviene sempre mediante il rispettivo indirizzo; pertanto **sul bus di sistema devono transitare non solo i contenuti ma anche gli indirizzi** delle celle di memoria
- L'indirizzo di una cella in uno spazio di indirizzamento di 2^n celle è una configurazione binaria di **n bit**; la **dimensione dello spazio di indirizzamento** di una CPU è uno dei **parametri che ne caratterizza l'architettura**
- Ogni cella è solitamente composta da 8 bit (un byte); in questo caso si dice che la memoria è organizzata in byte; il byte è quindi la più piccola quantità di memoria indirizzabile

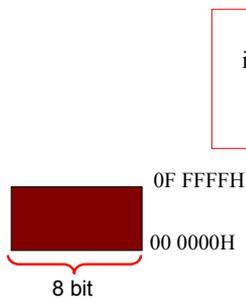
Locazioni di memoria

MEMORIA	INDIRIZZI
INFO W	2^{N-1}
⋮	⋮
INFO K	51
⋮	⋮
INFO Z	2
INFO Y	1
INFO X	0

N = linee hardware di indirizzamento del bus indirizzi

Spazio di indirizzamento in memoria

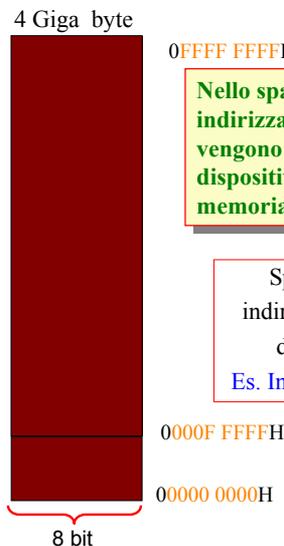
- Gli indirizzi si indicano solitamente in codice esadecimale
- la dimensione di uno spazio di indirizzamento si può esprimere in:
 - Kilobyte ($1\text{KB} = 2^{10}\text{Byte} = 1024\text{ B}$)
 - Megabyte ($1\text{MB} = 1\text{K KB} = 2^{20}\text{Byte} = 1.048.576\text{ B}$)
 - Gigabyte ($1\text{GB} = 1\text{K MB} = 2^{30}\text{Byte} = 1.0\text{e}+9$)
- Esempi
 - l'8085 ha uno spazio di indirizzamento di 64 KB (indirizzi di 16 bit)
 - l'8086 ha uno spazio di indirizzamento di 1 MB (indirizzi di 20 bit rappresentabili con 5 cifre esadecimali)



Spazio di indirizzamento di un MB
Es. Intel 8086

Spazio di indirizzamento in memoria

- Gli indirizzi si indicano solitamente in codice esadecimale
- la dimensione di uno spazio di indirizzamento si può esprimere in:
 - Kilobyte ($1\text{KB} = 2^{10}\text{Byte} = 1024\text{ B}$)
 - Megabyte ($1\text{MB} = 1\text{K KB} = 2^{20}\text{Byte} = 1.048.576\text{ B}$)
 - Gigabyte ($1\text{GB} = 1\text{K MB} = 2^{30}\text{Byte} = 1.0\text{e}+9$)
- Esempi
 - l'8085 ha uno spazio di indirizzamento di 64 KB (indirizzi di 16 bit)
 - l'8086 ha uno spazio di indirizzamento di 1 MB (indirizzi di 20 bit rappresentabili con 5 cifre esadecimali)
 - il Pentium ha uno spazio di indirizzamento di 4 GB (indirizzi di 32 bit rappresentabili con 8 cifre esadecimali)
 - Pentium III e Pentium IV indirizzano 64 GB (indirizzi di 36 bit)



Nello spazio di indirizzamento vengono "mappati" i dispositivi fisici di memoria

Spazio di indirizzamento di 4 GB
Es. Intel Pentium

Accesso alle interfacce di ingresso/uscita Spazio di indirizzamento in I/O

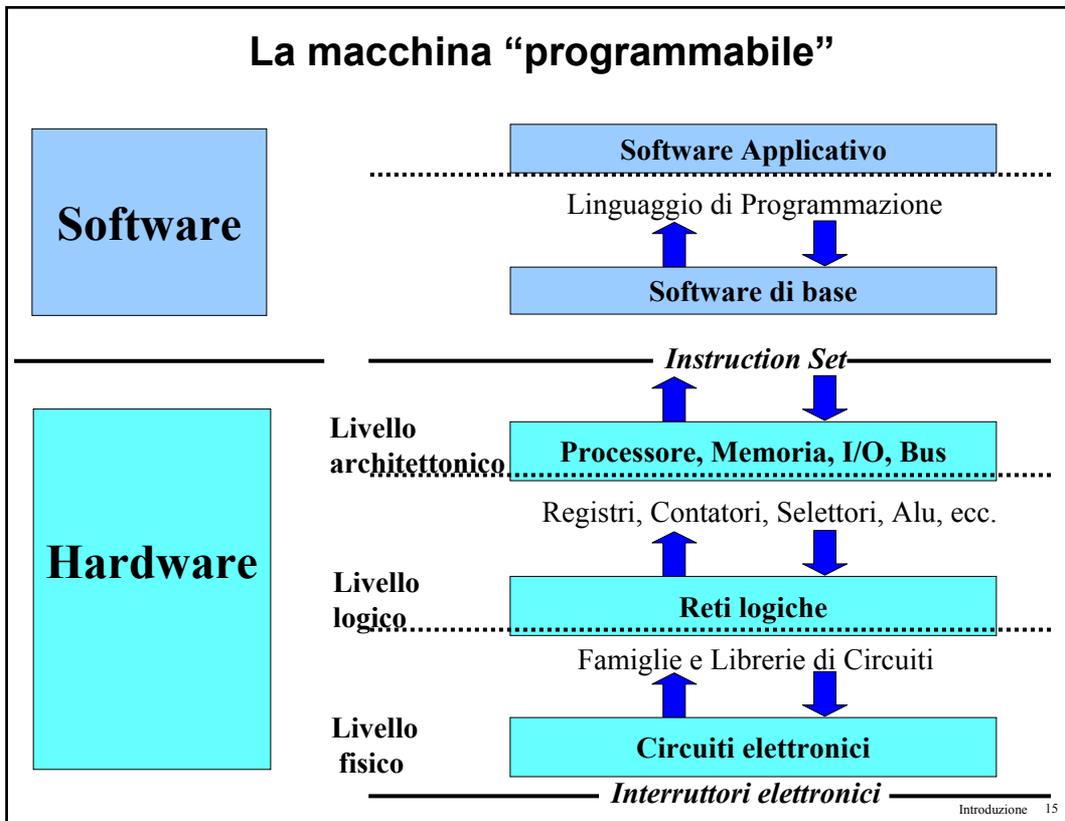
- Durante l'esecuzione del programma anche i dati scambiati tra CPU e interfacce di ingresso/uscita transitano per il bus
- Così come i dispositivi di memoria, anche le interfacce di ingresso/uscita sono mappate in uno spazio di indirizzamento
- Le interfacce di I/O possono essere mappate in uno spazio distinto da quello della memoria oppure nello stesso; in quest'ultimo caso si dice che l'I/O è mappato in memoria (*memory mapped I/O*)
- Lo spazio di indirizzamento in I/O è solitamente più piccolo dello spazio di indirizzamento in memoria; es: nelle architetture Intel IA16 e IA32 lo spazio di indirizzamento in I/O è di 64 KB
- Anche se i due spazi di indirizzamento sono distinti, i segnali del bus che portano l'indirizzo sono comuni (i segnali che portano gli indirizzi di I/O sono un sottoinsieme di quelli che portano gli indirizzi negli accessi alla memoria)
- La distinzione tra i due spazi di indirizzamento viene affidata a appositi segnali del bus



Trasferimenti di informazioni sul bus comandati dalla CPU

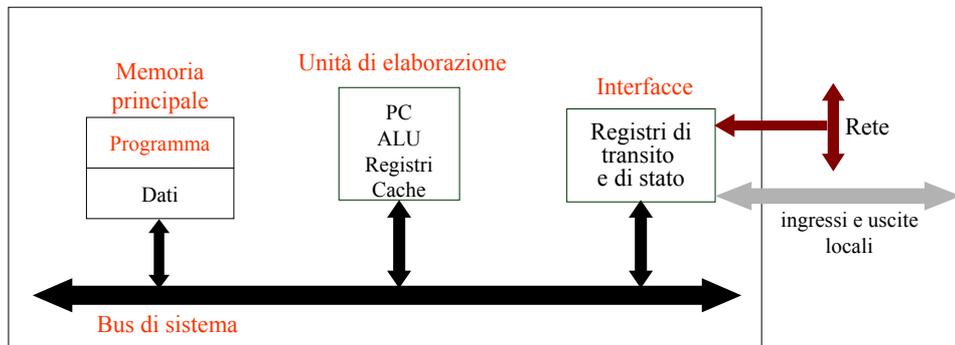
- La CPU può comandare i seguenti trasferimenti di informazioni sul bus:
 - trasferimenti da e verso dispositivi mappati nello spazio di indirizzamento in memoria; esempi:
 - ✦ lettura di istruzioni e dati (operandi delle istruzioni) **dalla memoria**
 - ✦ scrittura di risultati **in memoria**
 - trasferimenti da e verso dispositivi mappati nello spazio di indirizzamento in I/O; esempi:
 - ✦ lettura di dati **dalle interfacce**
 - ✦ scrittura di risultati **sulle interfacce**
- Per gestire correttamente questi trasferimenti il bus di sistema deve disporre dei tre gruppi di segnali già descritti (dati, indirizzi e comandi)

La macchina "programmabile"



Rappresentazione astratta dell'hardware di un calcolatore

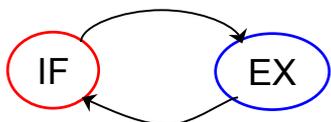
- L'hardware del calcolatore si interfaccia con il software attraverso il suo set di istruzioni (linguaggio macchina)



- Struttura del calcolatore (macchina digitale a esecuzione sequenziale e programma memorizzato)
- Ogni blocco della struttura è costituito da circuiti elettronici digitali

Modello di esecuzione del programma

- Il programma risiede in memoria ed è costituito da istruzioni **codificate in forma binaria**
- In memoria risiedono anche gli **operandi delle istruzioni**, cioè i dati elaborati e da elaborare
- Le istruzioni vengono eseguite **in sequenza** dalla CPU
- La CPU è una macchina sequenziale sincrona
- A livello di massima astrazione il suo automa ha due stati:
 - **Stato** in cui la CPU legge in memoria **la prossima istruzione** da eseguire (INSTRUCTION FETCH o **IF**)
 - **Stato** in cui la CPU esegue **l'istruzione letta in IF** (EXECUTE o **EX**)

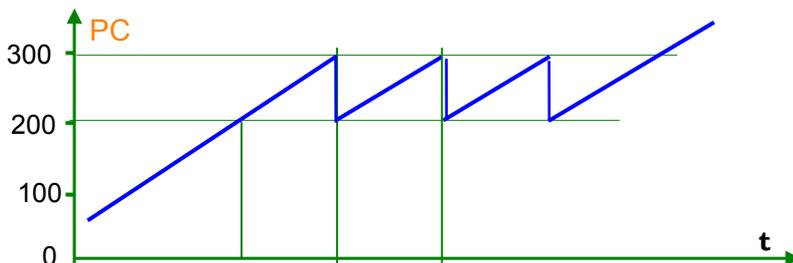


- per funzionare la CPU ha bisogno almeno degli ingressi di **reset** e **clock**
- Se il reset non è attivo la CPU **perennemente** legge e esegue istruzioni, cambiando stato ad ogni impulso di clock
- **la frequenza del clock è uno dei parametri che caratterizzano la CPU**

Introduzione 17

Il program counter e la sua dinamica durante l'esecuzione di un programma

- Per poter eseguire le istruzioni in sequenza la CPU dispone al suo interno di un contatore detto **Program Counter (PC)**
- Il **PC** viene incrementato ad ogni **FETCH**
- Il **PC** contiene **l'indirizzo in memoria** della prossima istruzione da leggere nella prossima fase di **FETCH**; in questo modo il **PC** individua la prossima istruzione da eseguire

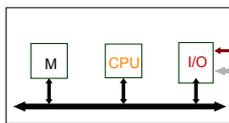


Il grafico mostra la dinamica del PC quando il calcolatore esegue un loop di 4 iterazioni
La pendenza delle linee è un indicatore del numero di istruzioni eseguite nell'unità di tempo

Introduzione 18

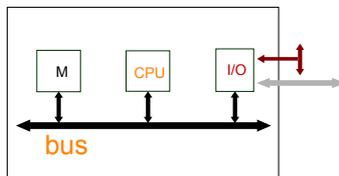
Il calcolatore è una macchina digitale

- all'interno di un calcolatore tutte le informazioni (es. dati e istruzioni) sono codificate in forma **binaria**, quindi:
 - nell'unità di elaborazione vengono elaborate variabili binarie
 - in memoria dati e istruzioni risiedono sotto forma di variabili binarie
- Il bus è il **supporto di interconnessione** tra i blocchi che costituiscono il calcolatore quindi:
 - sul bus transitano variabili **binarie**, pertanto i segnali del bus sono segnali digitali



- I blocchi interconnessi al bus si chiamano **agenti del bus (bus agent)**
- L'unità di elaborazione viene chiamata anche **"processore" o "CPU" (Central Processing Unit)**

Agenti master e slave e sistemi a singolo master



- Gli agenti del bus si suddividono in:
 - agente master
 - agente slave
 - agenti master/slave

- Si chiama agente del bus (**bus agent**) un modulo che si affaccia al bus
- Si chiama **agente master** un agente che genera indirizzi e segnali di comando; per i master **indirizzi e comandi sono segnali di uscita**
- Si chiama **agente slave** un agente indirizzato dal master; per gli agenti slave **indirizzi e comandi sono segnali di ingresso**
- Un sistema in cui c'è un solo agente master è detto sistema a singolo master
- Nell'architettura del lucido precedente la CPU è l'unico **agente master**; memoria e interfacce sono **agenti slave**
- Un agente che alterna nel tempo il ruolo di master con quello di slave si chiama agente master/slave

I cicli di bus

- Il trasferimento di un'informazione tra agenti del bus avviene con una sequenza di eventi detti nel loro insieme *ciclo di bus*
- in ogni ciclo di bus un agente master indirizza un agente slave
- Ad esempio, per leggere in memoria **all'indirizzo i** la CPU (master del bus) genera un ciclo di bus costituito dalla seguente sequenza di eventi:
 - la CPU mette sul **bus degli indirizzi** l'indirizzo **i**
 - la CPU attiva **un segnale di comando** che identifica l'operazione desiderata (MRDC# per la lettura in memoria)
 - la CPU attende che la memoria indirizzata (slave) metta l'informazione desiderata sul **bus dati**
 - la CPU legge (cioè campiona) sul **bus dati** i segnali generati dalla memoria
- Questa sequenza di eventi si chiama *ciclo di lettura in memoria* (**memory read cycle**)

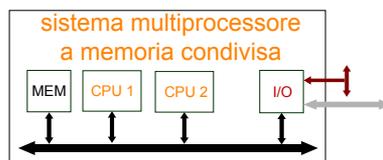
Esempi di cicli di bus

- Altri esempi di cicli di bus sono i seguenti:
 - ciclo di scrittura in memoria (**memory write cycle**) caratterizzato dal segnale di comando **MWRC#**
 - ciclo di lettura in I/O (**I/O read cycle**) caratterizzato dal segnale di comando **IORDC#**
 - ciclo di scrittura in I/O (**I/O write cycle**) caratterizzato dal segnale di comando **IOWRC#**
- Durante il corso studieremo in dettaglio questi e altri cicli di bus

Sistemi multimaster

- Un sistema in cui più bus master sono interconnessi a un bus si chiama “sistema multimaster”
- In questo caso il bus diventa una risorsa condivisa gestita a divisione di tempo: **un solo bus master è attivo (bus owner) per ogni ciclo di bus**
- Quando un master è attivo gli altri master possono:
 - osservare l’attività sul bus al fine di stabilire se ciò che avviene sul bus li riguarda (in questo caso l’agente si chiama anche **snooping agent**)
 - rimanere temporaneamente isolati dal bus
- E’ necessario prevedere un meccanismo di arbitraggio che **assegna in base a una regola di priorità** il bus ai master che ne fanno richiesta
- I cicli di bus sono indivisibili; se un master inizia un ciclo, allora lo termina prima di cedere il bus a un altro master (questa è una regola generale con qualche eccezione non considerata in questo corso)

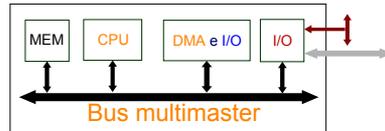
Esempi di Sistemi Multimaster Architetture multiprocessor



- Un sistema con più CPU affacciate al bus di sistema è un sistema multimaster
- Un sistema di questo tipo si chiama **sistema multiprocessore a memoria condivisa**
- L’architettura viene anche chiamata architettura **UMA (Uniform Memory Access)** a indicare che tutte le CPU accedono nello stesso modo alla memoria
- Non solo la memoria ma anche le interfacce vengono condivise da tutte le CPU

Esempi di Sistemi Multimaster

Sistemi con accesso diretto alla memoria (DMA)



- Esistono interfacce in grado di generare cicli di bus di accesso alla memoria
- Queste interfacce vengono attivate dalla CPU, dopodichè gestiscono da sole lo scambio di informazioni tra la memoria e il mondo esterno, senza richiedere ulteriori interventi alla CPU
- La funzione di accesso autonomo alla memoria si chiama **DMA** (Direct Memory Access)
- Un sistema con una CPU e **un'interfaccia con DMA** è ancora un sistema multimaster
- Esistono anche moduli il cui unico ruolo è quello di gestire il trasferimento dati tra agenti slave (es. memoria e interfacce slave); questi moduli si chiamano **DMA Controller (DMAC)**
- **I DMAC e le interfacce con DMA sono in generale agenti master/slave (ma possono anche essere solo master)**

Esercizio: calcolo del numero di cicli di bus necessario per eseguire un programma

- Domanda: quante volte accede al bus la CPU se si desidera eseguire il seguente programma?
 $A = B + C$
 $D = E - F$
 $G = A * D$
(si facciano le seguenti tre ipotesi:
 - tutte le variabili A, B, C, D, E, F, G sono residenti in memoria
 - l'accesso a ciascuna di esse richiede un ciclo di bus
 - la lettura di ogni istruzione in memoria richiede un ciclo di bus)
- Risposta:
 - il programma è lungo tre istruzioni, quindi la CPU deve eseguire tre fasi di FETCH e tre fasi di EXECUTE
 - ogni fase di FETCH (lettura dell'istruzione da eseguire) **implica un accesso al bus**
 - ogni fase di EXECUTE **implica 3 cicli di bus** (2 per la lettura in memoria dei due operandi e uno per la scrittura in memoria del risultato)
 - dunque per eseguire il programma dato nelle ipotesi fissate **sono necessari 12 cicli di bus**

Il livello di astrazione a cui si colloca il corso

- Abbiamo visto quali sono i blocchi che compongono un calcolatore
- abbiamo visto quali sono i principi di funzionamento del calcolatore
- Ora daremo le seguenti definizioni:
 - architettura dell'hardware di un calcolatore
 - prestazioni di un calcolatore
- quindi andremo a studiare come l'architettura influenza le prestazioni
- successivamente analizzeremo le più comuni architetture oggi impiegate
- Infine impareremo a progettare semplici sistemi che basano il loro funzionamento su una CPU

Architettura dell'hardware di un calcolatore

L'architettura dell'hardware è definita dalla seguente terna:

- Il set di istruzioni (architettura vista dall'utente, detta anche **linguaggio macchina**)
- La struttura interna
- La realizzazione circuitale (cioè la tecnologia microelettronica impiegata nella realizzazione)

- **Uno stesso set di istruzioni può essere realizzato con strutture interne diverse (es. 386, 486, e Pentium)**
- **La stessa struttura interna può essere realizzata con tecnologie diverse (es. 486, 486-DX2 e 486-DX4)**
- **Architetture diverse avranno in generale prestazioni diverse**

Principi di progettazione 1

Rendere veloce il caso più comune

Nella progettazione di un calcolatore si cerca di rendere la risposta più veloce nei riguardi del compito che deve essere eseguito più spesso a discapito del caso meno frequente.

Legge di Amdhal

Il miglioramento di prestazione ottenibile mediante l'uso di alcune modalità di esecuzione più veloci sarà tanto maggiore quanto maggiore è la frazione di tempo in cui esse vengono usate (nel calcolo va considerata la frazione di tempo su cui avrà effetto il miglioramento e non quella misurata dopo che è esso è stato apportato).

Regole Pratiche

-Il solo 10% del codice di un programma impegna la CPU per il 90% del tempo della sua esecuzione...

-Località temporale: gli elementi (dati e codice) usati di recente saranno usati anche nel futuro prossimo

-Località spaziale: gli elementi i cui indirizzi sono tra loro vicini tendono ad essere usati in tempi ravvicinati

Principi di progettazione 2

Per progettare un calcolatore bisogna tenere conto dei REQUISITI FUNZIONALI:

Area d'applicazione: impiego speciale (alte prestaz. per appl. specifiche)

- generale (prestaz. bilanciate per svariati compiti)
- scientifico (elevate prestazioni per calcoli con virgola mobile)
- commerciale (supporto per Cobol -aritm. decimale- e database)

Livello di compatibilità SW:

- alta compatibilità per la programmazione ad alto livello (sono accettati i sorgenti scritti per altre macchine grazie a compilatori scritti ad hoc)
- alta compatibilità per gli eseguibili (sono accettati gli eseguibili compilati per altre macchine grazie ad una architettura costruita ad hoc)

continua...

Principi di progettazione 3

...segue REQUISITI FUNZIONALI:

Compatibilità con vari OS, implica vincoli:

- nella scelta della *dimensione dello spazio di indirizzamento*
- nella *gestione della memoria* RAM (segmentata 8088, paginata Pentium, uniforme...)
- realizzazione del meccanismo della *protezione* delle aree di memoria
- gestione dei meccanismi per *interrompere e riattivare* un programma

Rispetto degli standard per:

- virgola mobile
- Bus I/O
- connessione alla rete, ...

“Piccolo è veloce” è il principio è da applicare in particolare alle memorie. Piccoli segmenti di memoria saranno in generale veloci rispetto a memorie più capaci. Figlie di questo assioma sono le memorie “cache”, bassa capacità alta velocità, contribuiscono maggiormente a velocizzare l’esecuzione di un programma quanto più contengono i dati più frequentemente usati dalla CPU

Calcolatori Elettronici L-A

Misura delle prestazioni di un Calcolatore
(o sistema a microprocessore)

Velocità di un Sistema a microprocessore o Speedup

Def: Velocità relativa (Speedup)

Definiamo un computer X più veloce di un altro Y quando il *Tempo di Esecuzione* [sec] (o *Latenza*) per un dato lavoro è più basso in X che in Y:

$$(VelocXrispettoVelocY)\% = \left(\frac{TempoEsecuzione_Y - TempoEsecuzione_X}{TempoEsecuzione_X} \right) \cdot 100$$

Per questo motivo se X esegue un calcolo in 10 secondi e Y in 15 si dirà che X è più veloce di Y del 50% (e non del 33%!).

Esiste anche il parametro *Prestazione* [eventi/sec] talvolta usato per calcolare la velocità:

$$Prestazione = \frac{1}{TempoEsecuzione}$$

Prestazioni della CPU - 1

Clock [Hz] è un segnale periodico ad onda quadra (non necess. simmetrica) e frequenza FISSA usato dai dispositivi di un calcolatore (quindi anche dalla CPU) per discretizzare gli eventi e sincronizzare tra loro i dispositivi.

Def: **Tempo di CPU** [sec] è il tempo impiegato dalla CPU per l'esecuzione di UN programma. Esso NON comprende il tempo impiegato per altri programmi (multiprogrammazione) e il tempo necessario per accedere alle risorse di I/O necessarie al programma.

$$TempoDiCPU = \frac{CicliClockPerEseguireUnProgramma}{FrequenzaClock}$$

Def: **CPI clock per (ogni) istruzione medio** Dato un programma di un certo numero di istruzioni, il **CPI medio** risulta dalla divisione dei cicli di clock serviti per eseguire il programma ed il numero di istruzioni di cui è composto il programma.

$$\overline{CPI} = \frac{CicliClockPerEseguireUnProgramma}{NumeroIstruzioni} \quad \text{da cui}$$

$$TempoDiCPU = \frac{NumeroIstruzioni \cdot \overline{CPI}}{Clock}$$

Prestazioni della CPU - 2

Talvolta è utile determinare i cicli di clock necessari ad eseguire un programma in funzione del CPI medio supponendo che quest'ultimo sia stato precedentemente misurato in modo specifico per la i -esima istruzione

$$\text{CicliClockPerEeguireUnProgramma} = \sum_{i=1}^n \overline{CPI}_i \cdot I_i$$

dove I_i rappresenta il numero di volte che l'istruzione i viene eseguita, \overline{CPI}_i è il suo CPI medio misurato ed n è il numero di istruzioni diverse inserite nel programma. Alla luce di questa ultima espressione possiamo ridefinire sia il Tempo di CPU che il CPI:

$$\text{TempoDiCPU} = \frac{\sum_{i=1}^n \overline{CPI}_i \cdot I_i}{\text{FrequenzaClock}}$$

$$\overline{CPI} = \sum_{i=1}^n \left(\overline{CPI}_i \cdot \frac{I_i}{\text{NumeroIstruzioni}} \right)$$

dove è stato messo in evidenza che ogni \overline{CPI}_i moltiplica la frazione delle occorrenze della istruzione nel programma.

A parità di clock, più è basso il CPI, migliori sono le prestazioni.