Making Everything Easier!™

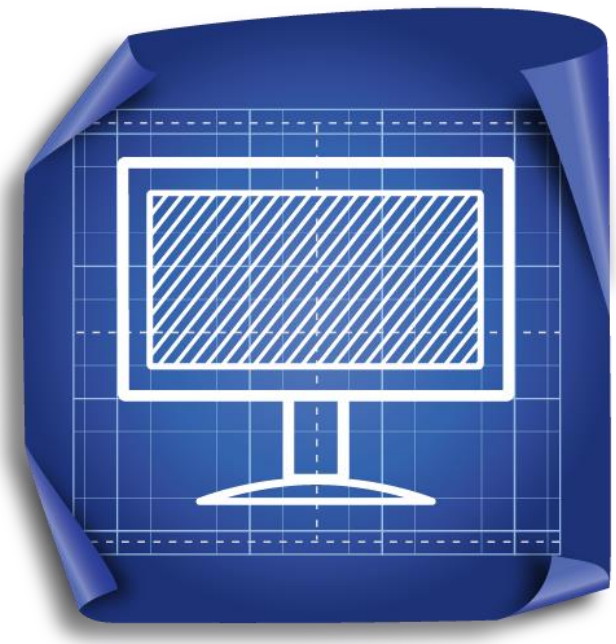
3rd Edition

Architettura degli elaboratori

FOR DUMMIES®

Learn to:

- Rispondere alle FAQ
- Risolvere gli esercizi
- Passare l'esame ASAP



Ciprian Voinea

Tabelle di conversioni utili da avere sotto mano e da imparare:

Kilo, Mega, Giga, Tera, ...

- Byte = 8 bit
- Kilo, dal greco khiloi (1000 = 10^3)
  - 2^{10} = 1024 = 1K (vicino a 1000)
- Mega, dal greco mega (grande)
  - 1.000.000 = 10^6
  - 2^{20} = 1.048.576
- Giga, dal latino gigas (gigante)
  - 1.000.000.000 = 10^9
  - 2^{30}
- Tera, dal greco tera (mostro)
  - 10^{12}
  - 2^{40}
- Peta, dal greco pente (5)
  - $1000^5 = 10^{15}$
  - 2^{50}

TABELLA DI CONVERSIONE DECIMALE-ESADECIMALE-BINARIO

DECIMALE	ESADECIMALE	BINARIO
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Formule per il calcolo del tempo di trasferimento:

$$Tempo\ medio\ totale\ di\ trasferimento\ = T_S + T_L + T_T$$

- TS → Tempo di Seek (ms in cui la testina si posiziona sulla traccia)
- TL → Tempo di Latenza (ms in cui il settore passa sotto la testina)
- TT → Termpo di Trasferimento (ms in cui vengono trasferiti tot byte)

$$T_L = \frac{60}{rpm} * \frac{1}{2} * 1000$$

$$T_T = \frac{b}{rN} * 1000$$

b → quantità di byte da trasferire

N → numero di byte per traccia

r → rps → 60 / rpm

N.B.: nel caso in cui i settori fossero contigui su un singolo cilindro, la durata di T_T viene divisa per il numero di facce.

Formule per l'indirizzamento della cache di tipo SET ASSOCATIVE:

$$Numero\ di\ linee\ presenti\ nella\ cache\ = \frac{dimensione\ cache}{dimensione\ linea}$$

Dove la dimensione di una linea è la grandezza di un blocco presente in memoria centrale.

$$Numero\ di\ set = \frac{numero\ linee}{numero\ vie}$$

Sapendo il n° di bit totali in un indirizzo, la dimensione del campo ***TAG*** si calcola facendo:

$$n\ bit\ totali\ dell'indirizzo - (n\ bit\ set + bit\ parola)$$

N.B.: il numero di bit totali di un indirizzo di cache rappresenta la quantità massima indirizzabile in memoria centrale (ovvero la capacità della RAM).

Formula per il numero di bit aggiuntivi calcolati con il codice di Hamming:

$$2^k - 1 \geq M + K$$

Parte I:

1) Descrivere la differenza tra architettura e organizzazione di un calcolatore.

L'architettura di un elaboratore è costituita dall'insieme degli attributi visibili al programmatore:

- il repertorio delle istruzioni
- il numero di bit usati per rappresentare i dati
- i meccanismi di I/O
- le tecniche di indirizzamento della memoria.

} impatto diretto sull'esecuzione logica del programma

L'organizzazione invece riguarda gli aspetti hardware trasparenti al programmatore:

- segnali di controllo
- interfacce fra periferiche e calcolatore
- tecnologia della memoria

Gli attributi architetturali sono gli elementi che costituiscono la struttura dell'elaboratore, quelli organizzativi sono dati dal modo in cui i primi vengono implementati.

Per esempio, la disponibilità o meno dell'operazione moltiplicazione è un aspetto architetturale, il modo in cui viene implementata (circuito dedicato o somme ripetute) organizzativo.

Una particolare architettura può durare per molti anni e permettere la retro-compattibilità del software.

Pag 8

2) Descrivere in cosa consiste l'architettura di Von Neumann.

La progettazione di quasi tutti i calcolatori odierni è basata sull'*Architettura di Von Neumann*, una struttura che consiste di:

1. Una memoria centrale accessibile per indirizzo che contiene dati e istruzioni
2. Un' unità artitmetico - logica (ALU) in grado di operare sui dati binari
3. Un' unità di controllo che interpreta e manda in esecuzione le istruzioni
4. Dispositivi di I/O

Per eseguire un programma basta inviare alla CPU i segnali di controllo che sono definiti dalle istruzioni salvate in memoria (*programazione software*), prima bisognava comporre i circuiti logici fisicamente in modo tale che potessero risolvere uno specifico problema (*programmazione cablata*).

Si prevede l'utilizzo di registri interni alla CPU (PC, IR, accumulatore, ecc.).

Pag 16-20

3) Spiegare a cosa serve il bus di sistema com'è strutturato e in che modo viene usato dal calcolatore.

Il bus di sistema è un mezzo di trasmissione condiviso che collega i componenti principali di un elaboratore (processore, memoria, I/O) fra di loro.

Il bus è formato da linee che trasportano il segnale in forma binaria (0 / 1), il numero di linee determina quanti bit possono essere trasportati contemporaneamente e viene definito *ampiezza del bus*.

Ci sono 3 tipi di linee:

- *Linee dati*: percorso su cui viaggiano i dati tra i moduli di sistema
- *Linee indirizzi*: contiene l'indirizzo di destinazione del dato da prelevare per la CPU (l'ampiezza del bus di indirizzi determina la quantità massima di memoria supportata)
- *Linee di controllo*: controllano l'uso e l'accesso al bus, trasmettono comandi e segnali di temporizzazione (clock) ai moduli del sistema collegati ad esso

Se un modulo desidera inviare dati ad un altro deve ottenere l'uso del bus e poi potrà cominciare il trasferimento dei dati.

Pag 87 - 88

4) Descrivere in dettaglio la gestione I/O da programma.

Nell'I/O da programma i dati vengono scambiati tra processore e modulo I/O. Quando il processore sta eseguendo un programma e incontra un'istruzione correlata con l'I/O, esso esegue l'istruzione inviando un comando al modulo di I/O appropriato.

Il modulo eseguirà l'azione richiesta e imposterà i bit appropriati nel suo registro di stato. Il modulo non intraprende nessun'altra azione per allertare la cpu. In particolare, non interrompe il processore, perciò quest'ultimo periodicamente dovrà controllare lo stato del modulo I/O finchè non rileva il completamento dell'operazione.

Ci sono 4 tipi di comandi che il processore può inviare al modulo I/O:

- Controllo: avvia una periferica e le dice cosa fare
- Test: testa le condizioni di stato dei moduli di I/O
- Lettura: ottiene i dati dalla periferica attraverso il modulo I/O
- Scrittura: impone al modulo di trasmettere tramite bus i dati alla periferica

C'è una stretta corrispondenza tra le istruzioni di I/O e i comandi che il processore invia ai moduli di I/O per eseguirle.

Pag 232 – 233,

5) Nel contesto della gestione dell'I/O si spieghi la differenza fra tecnica I/O memory mapped e I/O isolated. Discuterne vantaggi e svantaggi reciproci.

Quando processore, I/O e memoria centrale condividono un bus in comune, ci sono due modalità di indirizzamento: memory mapped e separato (isolated).

- I/O memory mapped: non c'è distinzione tra lo spazio degli indirizzi delle locazioni di memoria e dei dispositivi di I/O. Il processore tratta i registri dei moduli I/O come locazioni di memoria e utilizza le stesse istruzioni macchina per accedere alla memoria centrale e ai dispositivi I/O. Questo meccanismo necessita di una sola linea di lettura e una di scrittura sul bus.

- I/O Isolated: memoria e dispositivi non condividono lo stesso spazio di indirizzamento necessitando quindi da parte della cpu dei comandi speciali e linee di selezione apposite fra I/O e memoria. Le porte I/O sono accessibili solo tramite speciali comandi di I/O che attivano le linee di comandi di I/O del bus.

La maggior parte dei processori dispone di svariate istruzioni di riferimento alla memoria, ma nel caso di I/O separato, ci possono essere solo poche istruzioni di I/O.

Il vantaggio dell'I/O memory mapped consiste nel poter utilizzare questo vasto repertorio di istruzioni permettendo una programmazione più efficente, questo risulta uno svantaggio dell'I/O isolated che non ha tante così tante istruzioni dedicate per i comandi I/O. Uno svantaggio dell'I/O memory mapped consiste nel maggior utilizzo di prezioso spazio d'indirizzamento in memoria.

Pag 233 - 235

6) Spiegare il vantaggio di utilizzare il sistema di interruzioni.

Utilizzando un sistema di interruzioni, attraverso i segnali di interrupt un dispositivo di I/O può richiedere l'attenzione del processore, il quale dopo aver eseguito i comandi dell'interrupt ritorna al suo normale processo.

Questo evita che il processore resti in attesa del completamento delle operazioni da parte del dispositivo di I/O (come ad esempio aspettare la fine di un trasferimento dei dati), il quale comunicherà con il processore solamente in caso di errore o di completamento.

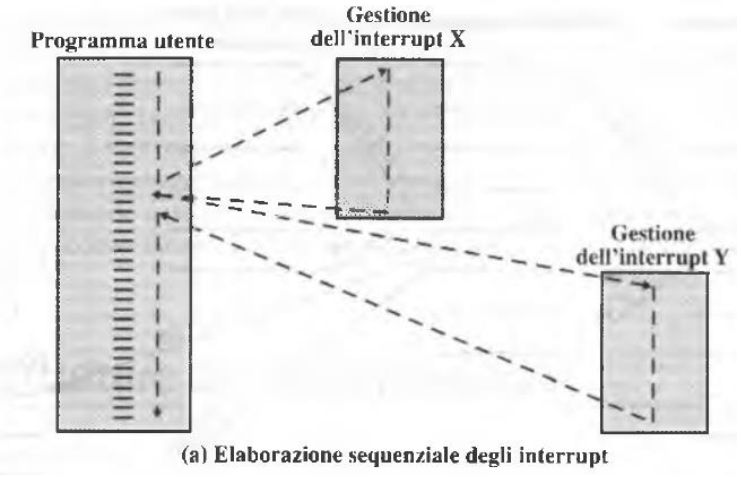
Il tempo del processore viene quindi sfruttato in modo tale che non ci siano periodi di inattività e l'efficenza dell'elaborazione viene migliorata notevolmente.

Le interruzioni possono essere messe in coda una dopo l'altra (interruzioni multiple) o annidate, ovvero possono interrompere altre interruzioni nel caso di operazioni che hanno la priorità.

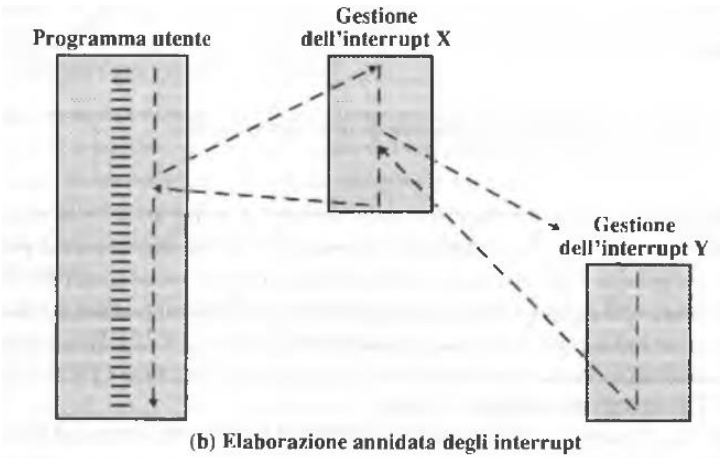
7) Spiegare la differenza fra interruzioni multiple e interruzioni annidate discutendo criticamente le differenti modalità di trattamento da loro richieste.

Per trattare le interruzioni si possono usare due approcci:

- *Interruzioni multiple*: disabilitare gli interrupt durante l'elaborazione di un interrupt (il processore ignora gli interrupt e restano pendenti fino alla terminazione di quello in esecuzione); non tiene conto di priorità o tempistica (svantaggio)
- *Interruzioni annidate*: vengono dati gradi di priorità agli interrupt permettendo ad un interrupt di priorità maggiore di interrompere l'esecuzione di uno con priorità inferiore



(a) Elaborazione sequenziale degli interrupt

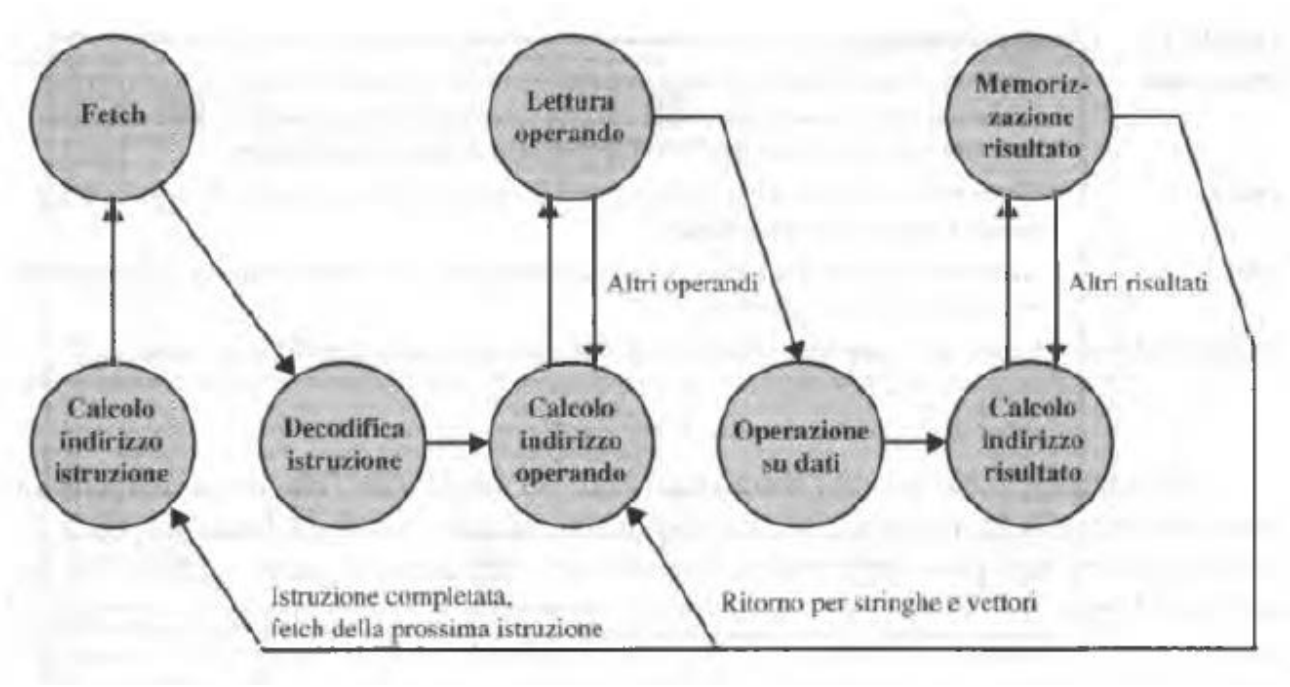


(b) Elaborazione annidata degli interrupt

Pag 81 – 84

8) Si descriva in dettaglio il ciclo completo di fetch execute delle istruzioni.

1. Calcolo indirizzo istruzione: determina l'indirizzo dell'istruzione da prelevare ed eseguire (dal PC)
2. Fetch: legge l'istruzione dalla memoria e la trasferisce nel processore (registro IR)
3. Decodifica istruzione: analizza l'istruzione e determina i tipi di operandi
4. Calcolo indirizzo operando: determina l'indirizzo dell'operando (in base al formato dell'istruzione)
5. Lettura operando: legge l'operando dalla memoria o da periferica
6. Operazione sui dati: esegue l'istruzione
7. Memorizzazione risultato: scrive il risultato nella memoria o lo manda ad una periferica



Pag 71 – 75

9) Descrivere in dettaglio il ciclo di esecuzione con trattamento delle interruzioni.

Il processore, ad ogni ciclo esecutivo esegue il fetch dell'istruzione il cui indirizzo è contenuto nel PC, il quale viene successivamente incrementato.

L'istruzione viene caricata nel registro IR del processore e analizzata per determinare il tipo di istruzione da eseguire e i suoi operandi che vengono prelevati dalla memoria (o da periferica).

L'istruzione viene dunque eseguita e il suo risultato viene scritto nella memoria.

Il processore controlla se è avvenuto interrupt, se non ce ne sono stati ricomincia il ciclo, altrimenti sospende l'esecuzione del programma salvandone lo stato attuale in memoria (*cambiamento di contesto*) e imposta il PC all'indirizzo di partenza di una routine per la gestione dell'interrupt.

Il processore procede dunque all'esecuzione dell'interrupt e quando questa routine termina viene ripresa l'esecuzione del programma salvato precedentemente in memoria (vengono ripristinati i valori salvati in precedenza).

Pag 76 - 82

10) Descrivere in che modo vengono gestite le interruzioni (sia per la componente hardware che per quella software) nel caso di I/O interrupt driven.

Nell'I/O interrupt driven il processore invia un comando al dispositivo I/O, prosegue nell'esecuzione di altre istruzioni e viene interrotto dal modulo quando quest'ultimo ha completato il proprio lavoro, il processore esegue lo scambio dei dati con il dispositivo e riprende poi l'elaborazione interrotta.

Il controllore di una periferica invia un segnale di interrupt al processore, che completa l'esecuzione dell'istruzione corrente e controlla se c'è stata richiesta di interrupt.

Analizzato il segnale, il processore salva il contenuto dei registri PC e PSW e le pone in cima alla pila di sistema (cambiamento di contesto).Il contenuto del PC viene modificato puntando alla prima istruzione della routine di gestione dell'interrupt considerato.

Quando l'elaborazione dell'interrupt è stata completata vengono recuperati dalla pila e ripristinati i valori dell'istruzione precedente (PC → Program Counter e PSW → Program Status Word).

Pag 233, 235 - 239

11) Si descrivano le caratteristiche e le operazioni principali delle memorie a semiconduttore considerando sia le memorie RAM che ROM.

L'elemento base delle memorie a semiconduttore è la cella di memoria che è caratterizzata da specifiche proprietà:

- Presenta 2 stati stabili, che rappresentano 0 o 1
- La possibilità di scrivere nella cella per impostarne lo stato
- La possibilità di leggerne lo stato

Tutti i tipi di memoria a semiconduttore sono ad accesso casuale (si accede alla cella tramite circuito dedicato).

La memoria a semiconduttore più comune è la RAM (Random Access Memory), la cui caratteristica principale è la possibilità di leggere e scrivere dati in e da memoria in modo semplice e rapido tramite segnali elettrici. Tale memoria è inoltre definita volatile per il fatto che necessita di alimentazione costante se si vuole preservare i dati, ed è per questo che quindi la RAM viene utilizzata solo per una memorizzazione temporanea.

Essa si divide in 2 tipologie: SRAM e DRAM.

La DRAM (Dynamic RAM) è composta di celle che memorizzano i dati sotto forma di cariche nei condensatori. L'assenza o la presenza di cariche determina lo stato 0 o 1. Poichè i condensatori tendono a scaricarsi naturalmente dopo un certo periodo le DRAM necessitano di un refresh periodico e da ciò deriva l'aggettivo "dinamiche" per il fatto che la carica scompare dinamicamente anche in presenza di alimentazione.

Le SRAM (Static RAM) utilizzano invece gli stessi elementi di base di un processore, infatti i valori binari vengono memorizzati mediante porte logiche e, diversamente dalle DRAM non necessitano di un refresh delle cariche.

Un altro tipo di memoria ad accesso casuale è la ROM (Read Only Memory), la quale presenta uno schema di dati predefinito e non modificabile. Tali memorie non sono volatili e mantengono quindi i dati anche in assenza di alimentazione, però questo tipo di memoria permette la sola lettura.

Il vantaggio di questa memoria è che i dati e programmi di piccole dimensioni possono essere scritti in essa e non devono essere caricati da dispositivi esterni. In esse inoltre, i dati vengono inseriti durante il processo di produzione, costoso e che richiede grande precisione, perchè l'errata scrittura di un bit causa la perdita di tutto il lotto ROM.

Un'alternativa meno costosa è la PROM (Programmable ROM): una memoria programmabile (1 volta) e non volatile.

Un'altra variante è la memoria "principalmente di lettura", utile quando le operazioni di lettura sono più frequenti di quelle di scrittura. Ce ne sono 3 tipi:

- EPROM: memorie cancellabili e riprogrammabili otticamente, lette e scritte elettricamente come la PROM. Prima della scrittura le celle di memoria devono essere cancellate e portate tutte allo stesso stadio mediante esposizione ai raggi UV.
- EEPROM: memorie cancellabili e programmabili elettricamente, in esse è possibile scrivere in qualunque momento senza cancellare i dati, aggiornando solo i byte indirizzati (la scrittura richiede molto più tempo della lettura). Tali memorie sono più costose e meno dense delle EPROM. Esempio chip ATMEL.
- Flash: chiamate così per la velocità con la quale possono essere riprogrammate. Così come le EEPROM adottano un sistema di cancellazione elettrica e la sua memoria può essere eliminata molto più velocemente di una EPROM, è inoltre possibile cancellare solo alcuni blocchi piuttosto che tutta la memoria.

Pag 164 - 168

12) Si descriva in dettaglio la memoria DRAM.

La DRAM acronimo di Dynamic RAM è composta di celle che memorizzano i dati sotto forma di cariche nei condensatori. L'assenza o presenza di cariche determina lo stato 0 o 1.

Poichè i condensatori tendono a scaricarsi naturalmente dopo un certo periodo di tempo le RAM necessitano di un refresh periodico e da ciò deriva l'aggettivo "dinamiche" per il fatto quindi che la carica scompare dinamicamente anche in presenza di alimentazione.

Per le operazioni di scrittura si applica tensione alla linea di bit (che a seconda dell'intensità determina se il valore del bit è 0 o 1) successivamente si applica segnale alla linea degli indirizzi trasferendo la carica al condensatore.

Per quelle di lettura, invece, prima di tutto si seleziona la linea indirizzo poi la carica viene convogliata in una linea di bit collegata ad un amplificatore che confronta la tensione con un valore di riferimento e determina se la cella contiene il valore 0 o 1, infine si applica un refresh per ripristinare la carica nel condensatore.

Pag 165 - 166

13) Si descriva in dettaglio la memoria SRAM.

La SRAM, acronimo di Static RAM, è un dispositivo digitale che utilizza gli stessi elementi alla base del processore. I valori binari sono memorizzati mediante porte logiche. Come le DRAM mantengono i propri valori finchè sono alimentate.

La struttura tipica di una cella di memoria SRAM è composta da 4 transistor connessi in modo da costruire uno stato logico stabile.

Come nelle DRAM, la linea indirizzo viene usata per aprire o chiudere un interruttore, controlla due transistor i quali, quando viene applicato un segnale, vengono accesi permettendo la lettura / scrittura dello stato della cella.

Pag 167

14) Si faccia un confronto tra DRAM e SRAM.

Sia le DRAM (Dynamic RAM) che le SRAM (Static RAM) sono memorie volatili, ovvero richiedono una continua alimentazione per conservare i valori dei bit.

Una cella di memoria dinamica (formata da condensatori) è più semplice e più piccola di una cella di memoria statica (formata da porte logiche). Dunque, a pari capacità, le DRAM sono più dense e meno costose.

Le DRAM richiedono inoltre circuiti aggiuntivi per il refresh periodico. Queste tendono ad essere favorite per memorie di grandi cacpacità, come la memoria centrale.

Le SRAM sono più veloci delle DRAM, non richiedono circuiti di refresh e vengono usate per la memoria cache (sia on-chip che off-chip).

Pag 167

15) Spiegare in dettaglio le differenze tra un modulo di memoria DRAM e un modulo di memoria SRAM. Discuterne vantaggi e svantaggi.

La DRAM (Dynamic RAM) è composta di celle che memorizzano i dati sotto forma di cariche nei condensatori. L'assenza o presenza di cariche determina lo stato 0 o 1.

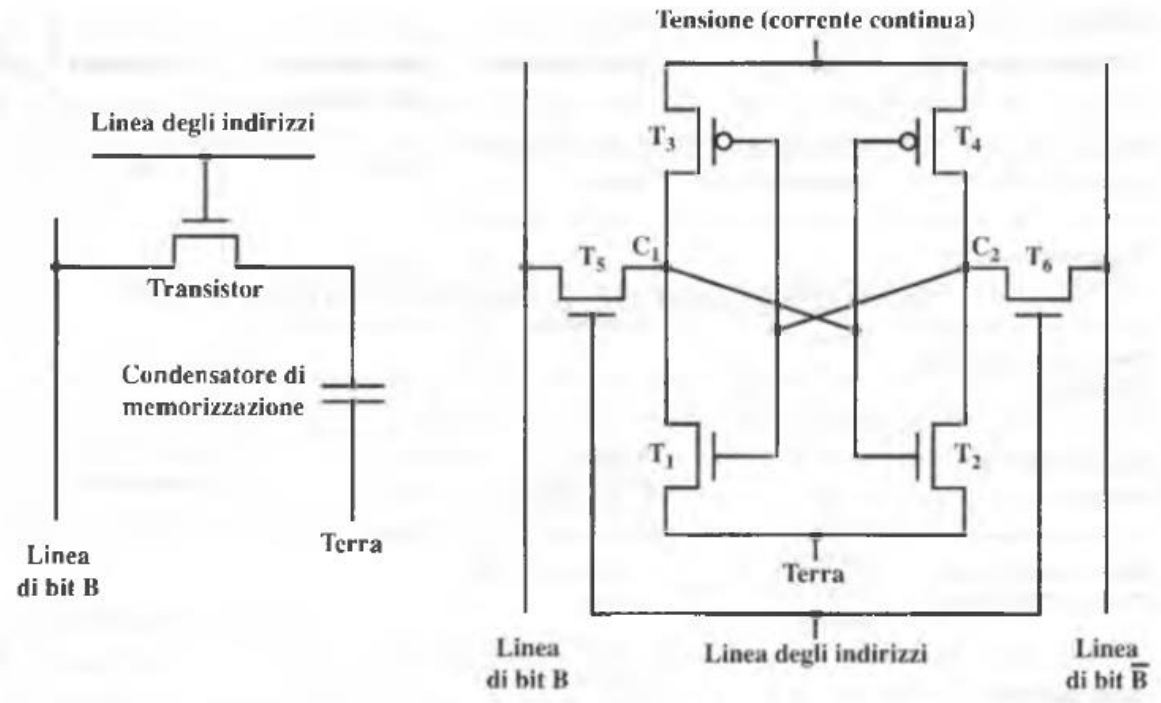
Poichè i condensatori tendono a scaricarsi naturalmente dopo un certo periodo di tempo le RAM necessitano di un refresh periodico e da ciò deriva l'aggettivo "dinamiche" per il fatto quindi che la carica scompare dinamicamente anche in presenza di alimentazione.

Le SRAM (Static RAM) utilizzano invece gli stessi elementi base di un processore, infatti i valori binari vengono memorizzati mediante porte logiche e diversamente dalle DRAM non necessitano di un refresh delle cariche.

Sia le RAM dinamiche che quelle statiche sono volatili, una cella di memoria dinamica però è più piccola e semplice di una statica perciò le DRAM sono più dense e meno costose ma richiedono circuiti aggiuntivi per il refresh periodico.

In sostanza le SRAM sono più veloci delle DRAM e vengono usate per la memoria cache mentre le DRAM per la memoria centrale.

Pag 165 – 167



(a) Cella di RAM dinamica (DRAM)

(b) Cella di RAM statica (SRAM)

16) Spiegare in dettaglio come funziona il codice di correzione di Hamming dare un'esempio concreto di codifica nel caso di memorizzazione di un insieme di 8 bit.

Quando i dati stanno per essere memorizzati si esegue un calcolo su di essi per produrre un codice che verrà memorizzato assieme a questi. Durante la lettura di una parola tale codice verrà impiegato per rilevare eventuali errori negli M bit di dati.Un nuovo codice è generato a partire dagli M bit dati e confrontato con i K bit precedentemente prelevati. Il confronto ottiene 3 risultati:

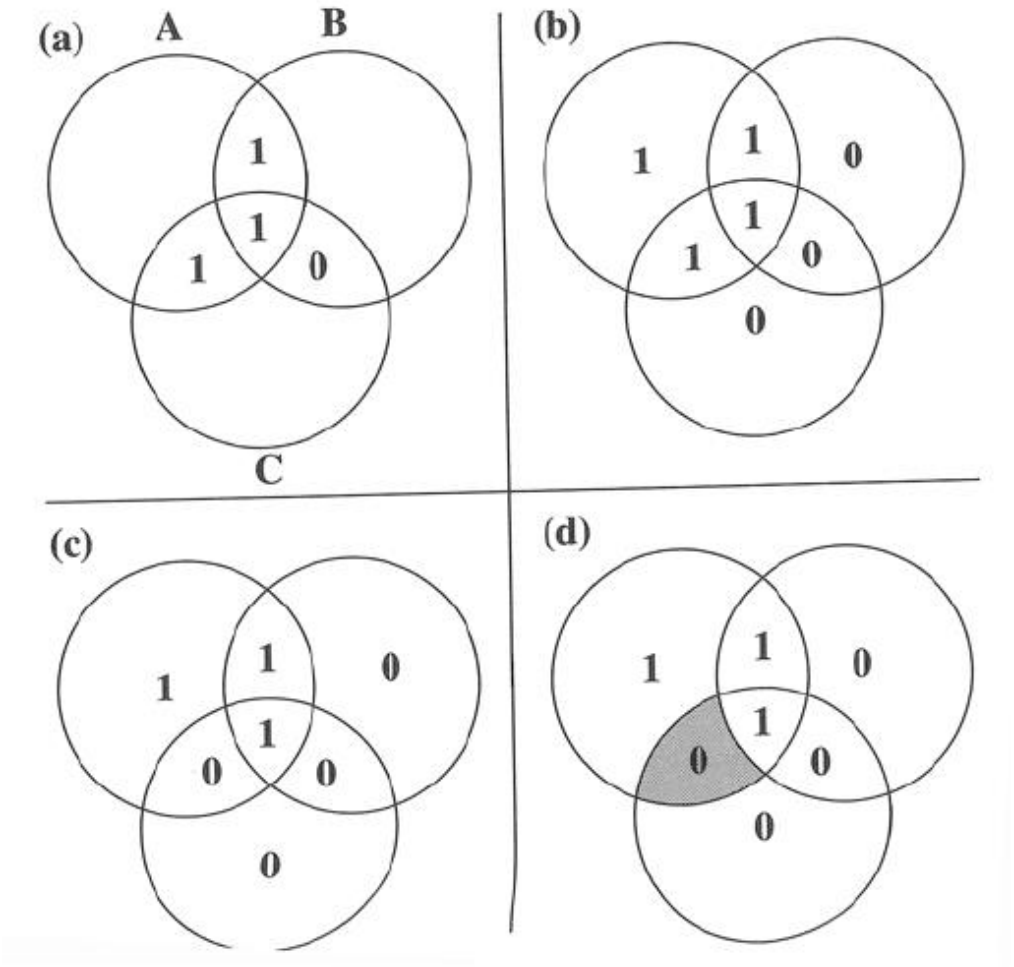
- nessun errore viene rilevato
- viene rilevato un errore ed è possibile correggerlo, perciò i bit dati e i bit per la correzione vengono inviati ad un correttore che produce un insieme corretto di M bit da emettere
- viene rilevato un errore ma non è possibile correggerlo, ciò viene segnalato

Il codice a correzione d'errore più semplice è quello di Hamming che deve rispettare la seguente formula:

$$2^k - 1 \geq M + K$$

con *M* il numero di bit della parola data e *K* il numero di bit di controllo che si possono aggiungere.

In un insieme di parole di 4 bit utilizziamo il diagramma di Eulero Venn per dividere il piano in 3 cerchi che determinano 7 regioni, i 4 bit dati verranno assegnati agli scomparti interni. I restanti scomparti vengono riempiti con i cosiddetti bit di parità. Ciascun bit di parità è scelto in modo che il numero totale di 1 nel proprio cerchio sia pari. Ora se avvenisse una modifica dei bit dati l'errore sarebbe facilmente rilevato e potrebbe essere corretto cambiando il determinato bit che lo causa.



Quanti bit di controllo servono ? $2^K-1 \geq M + K$

Bit di dati	Bit di controllo	% incremento
8	4	50
16	5	31,25
32	6	18,75
64	7	10,94

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data Bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C8				C4		C2	C1

Esempio generazione bit di controllo

12 1100	11 1011	10 1010	9 1001	8 1000	7 0111	6 0110	5 0101	4 0100	3 0011	2 0010	1 0001
D8	D7	D6	D5		D4	D3	D2		D1		
				C8				C4		C2	C1
1	1	0	0		0	0	1		0		

$$C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7$$
$$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$$
$$C4 = D2 \oplus D3 \oplus D4 \oplus D8$$
$$C8 = D5 \oplus D6 \oplus D7 \oplus D8$$

si ha

$$C1 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0$$
$$C2 = 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$$
$$C4 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$$
$$C8 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$$

I bit Cx sono i bit di controllo generati e vengono posizionati sulle posizioni delle potenze di 2 ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, ecc.).

Correzione degli errori: disposizione bit

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check Bit					0				0		0	1

XOR XOR XOR XOR

risultato XOR 0 1 1 0

È possibile aggiungere un bit finale alla parola per il controllo della parità sui bit di controllo generati con l'algoritmo di Hamming. Nel caso ci fosse un errore nei bit di controllo questo può venire rilevato ma non corretto.

Pag 174 - 178

17) Nel contesto di una gerarchia di memoria spiegare perchè la memoria viene suddivisa in blocchi e, relativamente alle prestazioni della cache, discutere pregi e difetti dell'adozione di una dimensione di blocco elevata.

Per realizzare un'organizzazione gerarchica della memoria, che soddisfi i parametri di velocità, ampiezza e costo, conviene suddividere la memoria in blocchi. La dimensione di un blocco è la quantità minima indivisibile di dati prelevabile (copiare) dal livello inferiore.

L'indirizzo di un dato diventa l'indirizzo del blocco che lo contiene sommato alla posizione del dato all'interno del blocco.

Se si adotta un blocco con dimensione elevata si guadagna in termini di costi e capacità ma si perde in termini di velocità.

Quando la dimensione del blocco cresce, vengono portati all'interno della cache più dati utili e la percentuale di successo inizialmente aumenta per il principio della località. Successivamente però la frequenza si successo comincerà a diminuire.

Entrano in gioco due specifici effetti:

- blocchi più larghi riducono il numero di blocchi nella cache. Un minor numero di blocchi porta alla sovrascrittura dei dati in fasi immediatamente successive al loro prelievo
- quando i blocchi diventano troppo grandi ogni parola addizionale è più lontana dalla parola richiesta, diminuisce quindi la probabilità che venga richiesta nell'immediato futuro.

La relazione fra dimensione blocco e percentuale di HIT è complessa, non esistono valori ottimali definitivi ma dimensioni da 8 a 64 byte sembrano ottimali.

Pag 141

18) Nel contesto di una gerarchia di memoria spiegare i possibili modi di realizzazione del mapping dei blocchi, discutendo criticamente i vantaggi e gli svantaggi di ogni modo.

L'algoritmo di mapping esegue l'indirizzamento dei blocchi di memoria centrale nelle linee della cache.

I possibili metodi di mapping dei blocchi in una gerarchia di memoria sono tre:

- diretto (direct mapping)
- associativo completo (fully associative)
- set associativo (n-way set associative)

L'indirizzamento diretto è la tecnica più semplice e meno costosa in quanto assegna ad ogni blocco una sola e specifica possibile linea di cache. Per accedere alla cache poi, ogni indirizzo in memoria centrale viene diviso in tre campi: tag, linea, parola. Questo metodo di traduzione si distingue per la semplicità con cui quest ultima avviene da indirizzo ILI (memoria) ad ILS (cache) e per la veloce determinazione di HIT o MISS. D'altro canto gli svantaggi possono essere necessità di contraddistinguere il blocco presente in cache tramite un'etichetta (tag) e porta inoltre ad un numero elevato di swap per accedere ai dati di blocchi adiacenti.

Il secondo metodo (associativo) invece supera lo svantaggio dell'indirizzamento diretto poichè ogni blocco della memoria centrale può essere caricato su qualsiasi linea della cache. Così facendo l'indirizzo di memoria presenta solamente due campi: tag e parola garantendo nel complesso una massima efficenza di allocazione. L'unico svantaggio è dato dalla complessità circuitale richiesta per esaminare in parallelo i tag di tutte le linee di cache quando viene richiesta una parola dalla CPU.

L'indirizzamento set-associativo, invece, è un compromesso che unisce i punti di forza dei precedenti riducendo i loro svantaggi. Qui la cache è suddivisa in insiemi (set) di k linee e il blocco può essere assegnato a qualunque linea dell'insieme, l'indirizzo in memoria è suddiviso nei campi tag, set e parola. Questo tipo di indirizzamento vanta una buona efficienza di allocazione a fronte di una discreta complessità di ricerca.

Pag 126 - 135

19) Nel contesto di una gerarchia di memoria spiegare come i MISS possono essere categorizzati in diversi tipi e dire quali sono le strategie (anche quelle che coinvolgono il compilatore), per ogni tipo, che si possono adottare per tentare di diminuirne il numero. Discutere tali strategie.

I miss in una gerarchia di memoria possono essere categorizzati in 3 tipi diversi:

- Miss di primo acesso: all'accensione del calcolatore la cache è vuota e non contiene alcun dato, inevitabile e non riducibile
- Miss per capacità insufficiente: quando la cache non può contenere tutti i blocchi necessari all'esecuzione del programma
- Miss per conflitto: quando più blocchi possono essere mappati (con associazione diretta o a gruppi) su uno stesso gruppo

Le tecniche risoluzione classiche per i miss per capacità insufficiente possono essere una maggiore dimensione del blocco la quale è una buona tecnica per fruire di località spaziale che però causa un aumento di miss per conflitto (a causa del numero ridotto di blocchi disponibili). Per i miss per conflitto una soluzione efficiente può essere la maggiore associatività che causa però un incremento del tempo di localizzazione in gruppo ed è soggetta alla regola del 2:1 (cache di N blocchi stessa probabilità di miss di cache a N/2 con associazione a 2 vie).

Altre tecniche di risoluzione possono essere l'adozione di una cache multilivello, la separazione di cache dati e cache istruzioni e l'ottimizzazione dei dati mediante compilatori che permettono il posizionamento accurato delle procedure ripetitive, la fusione di vettori in strutture (località spaziale) e la trasformazioni di iterazioni annidate (località spaziale).

Slide

20) Spiegare cosa sono gli errori soft, come ovviare a tali errori e fare eventualmente un esempio.

Le memorie primarie (semiconduttore) sono soggette ad errori. Questi possono essere classificati in guasti hardware, che sono permanenti, e guasti software, chiamati anche “soft error”, che sono casuali e non distruttivi. Essi infatti alterano i contenuti di una o piu celle di memoria, senza però danneggiarla fisicamente.

Possono essere causati da problemi di alimentazione o da particelle α che modificano il valore di uno più bit.

Gli errori “soft” possono essere rilevati ed eventualmente corretti usando codici che aggiungono bit in più alla parola in base a criteri come la parità.

Un esempio è il codice di Hamming: questo funziona tramite bit di parità, ovvero,(per esempio) con insiemi da 4 bit, si hanno 3 cerchi che si intersecano e all’interno di ogni cerchio il numero di bit complessivo deve essere pari, se no viene rilevato l’errore il quale può essere eventualmente corretto cambiando il numero di bit.

Pag 173 - 174

21) Nel contesto di una gerarchia di memoria spiegare come funzionano le politiche di scrittura write through e write back. Discuterne criticamente i problemi che possono sorgere nell’addottarle, vantaggi e svantaggi di ciascuna.

- *Write through*: Ogni dato modificato nella cache viene contemporaneamente modificato nella memoria centrale. In questo modo i dati sono sempre coerenti tra i vari livelli di memoria. Lo svantaggio principale però è che per frequenti scritture sul medesimo blocco si verifica un aumento di traffico nel bus con conseguente collo di bottiglia.

- *Write back*: La scrittura in memoria centrale avviene solo quando il corrispettivo blocco in cache viene rimpiazzato. Ciò consente un ottimizzazione del traffico ma causa periodi di incoerenza i vari livelli di cache e la memoria, inoltre occorre sempre ricordare se sono avvenute operazioni di scrittura nel blocco tramite “dirty bit”. Il problema di questa tecnica è che parti della memoria centrale non sono aggiornate, e dunque gli accessi tramite moduli I/O possono essere consentiti solo attraverso la cache.

Nel caso in cui cache e / o memoria centrale fossero condivise fra CPU ed altri dispositivi (processori ad nel caso dei multicore) l’alterazione di dati in una cache invalida i dati nella memoria centrale e nelle altre cache in cui sono stati prelevati. Anche addottando una politica di scrittura write through è possibile che il problema possa sussistere. Alcuni modi per risolverlo sono:

- monitoraggio del bus con write through: i controllori delle cache osservano le linee indirizzi per rilevare operazioni di scrittura in memoria da parte di altri gestori del bus, nel caso in cui uno di questi scrive in una locazione di memoria condivisa il cui dato è già presente in cache, il contollore invalida tale valore nella cache
- trasparenza hardware: viene utilizzato hardware aggiuntivo per assicurare che tutti gli aggiornamenti alla memoria centrale, tramite cache, si riflettano in tutte le cache
- memoria *noncacheable*: soltanto una porzione di memoria centrale detta *noncacheable* può essere condivisa. Gli accessi alla memoria sono detti cache miss (perchè non potrà mai essere portata in cache) e questa memoria viene identificata via hardware (*chip-select*) o tramite indirizzi riservati.

Pag 140, slide

22) Nel contesto di una gerarchia di memoria, discutere la modalità e la granularità delle informazioni fra i vari livelli della gerarchia.

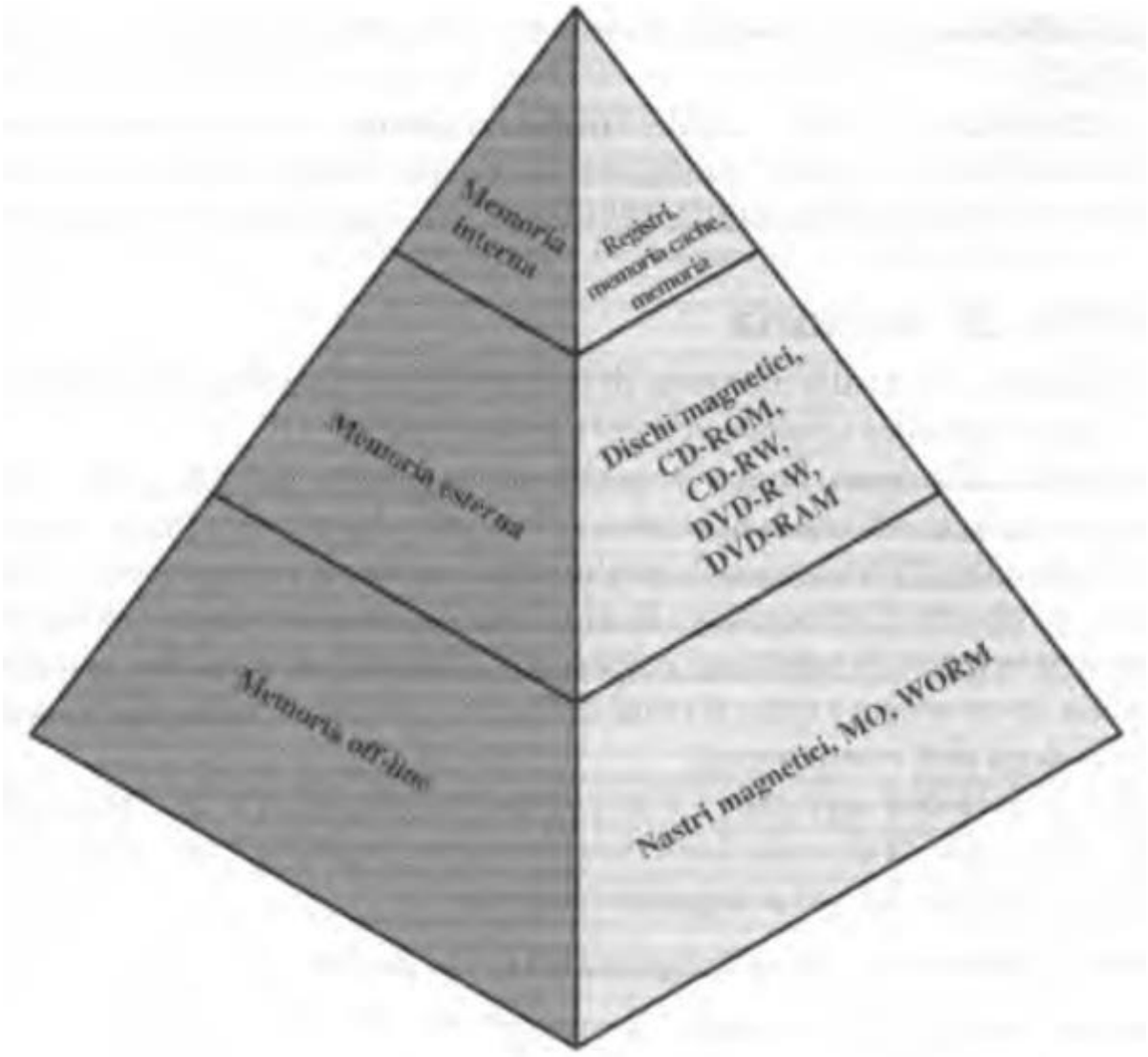
È possibile organizzare gerarchicamente le memorie in base a fattori come velocità / prezzo / grandezza.

Scendendo lungo la gerarchia, troviamo un minor costo per bit, una capacità maggiore, un tempo di accesso crescente e una minore frequenza di accesso da parte del processore.

In questo modo, memorie più piccole, più costose e più veloci sono integrate da memorie più grandi, economiche e più lente.

La CPU utilizza direttamente il livello più alto della gerarchia. Ogni livello inferiore, deve contenere tutti i dati presenti ai livelli superiori (più altri).

La dimensione di un blocco è la quantità minima indivisibile di dati prelevabile dal livello inferiore. L'indirizzo di un dato diviene l'indirizzo del blocco che lo contiene, sommato alla posizione del dato all'interno del blocco.



Pag 117 – 118

23) Discutere il modo in cui le informazioni sono organizzate in un CD-ROM (formato dati).

Il CD (Compact Disk) è un dispositivo ottico non cancellabile che sfrutta una serie di pozzetti (pit) per memorizzare dati binari su una superfice di policarbonato.

Tali informazioni, scritte da un laser ad alta intensità, vengono poi recuperate da un laser a bassa potenza.

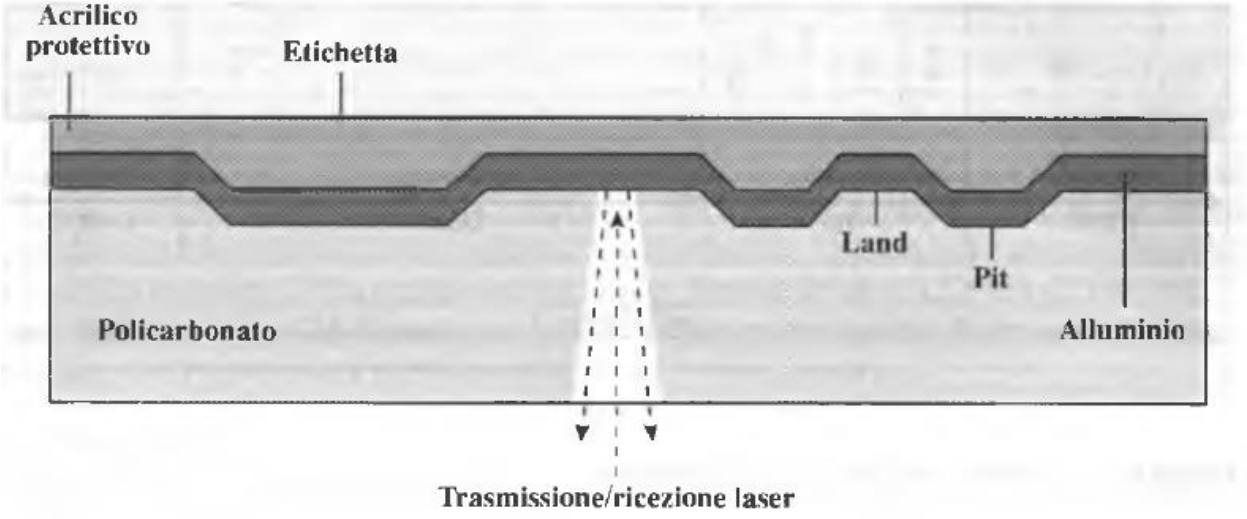
Le informazioni sono organizzate su una traccia a spirale che inizia vicino al centro e si svolge verso il bordo del disco. I settori in prossimità dell'estremità del disco sono della stessa lunghezza di quelli interni.

Le informazioni sono quindi impacchettate uniformemente sul disco in segmenti della stessa dimensione e questi vengono letti ruotando il disco a velocità variabile.

I pit sono poi letti dal laser a velocità lineare costante. Il disco ruota più lentamente per gli accessi sul lato esterno rispetto a quelli vicini al centro. L'intensità della luce del laser riflessa cambia quando incontra un pit.

A differenza dei CD audio, i CD-ROM presentano dispositivi di correzione degli errori per assicurare la correttezza del trasferimento dati tra disco e processore.

I pit rappresentano l'1, i land lo 0.

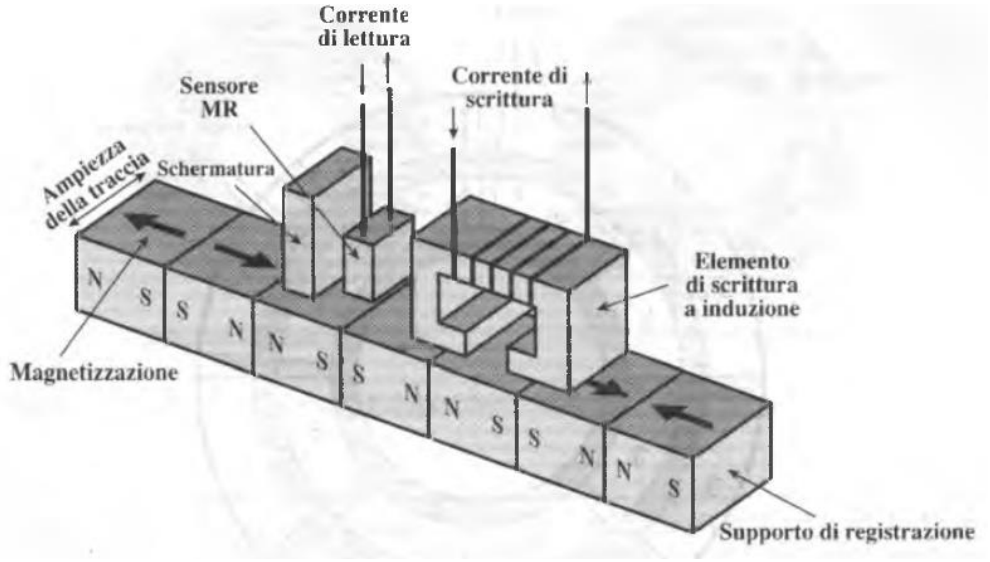


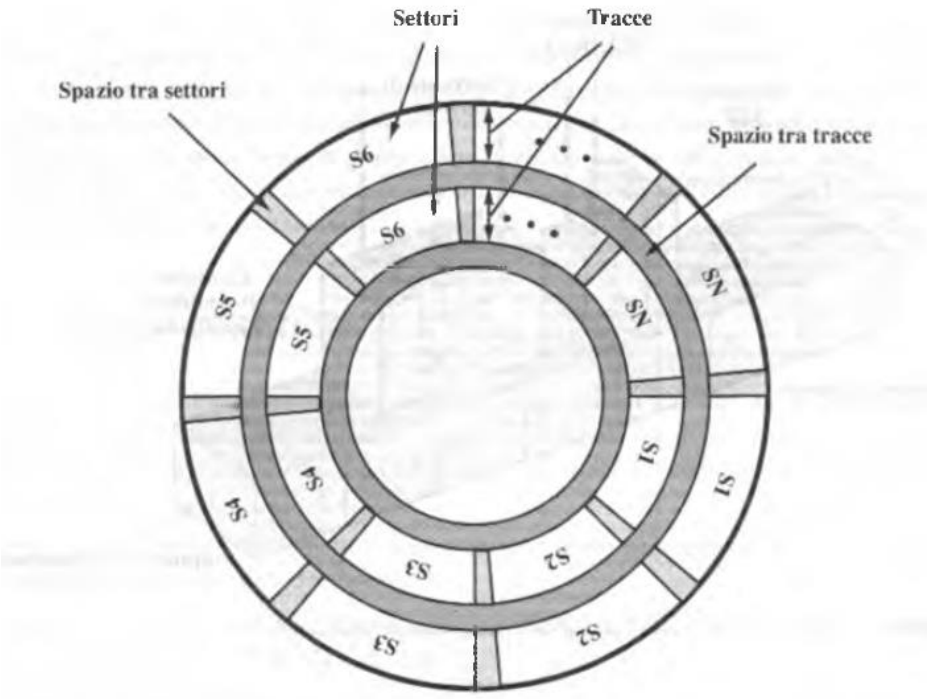
Pag 212 – 213

24) Descrivere l'organizzazione e formattazione dei dati nei dischi rigidi

Nei dischi magnetici la testina è in grado di leggere e scrivere su una porzione del disco rotante. Ciò origina la disposizione fisica dei dati in anelli concentrici, chiamati tracce (tracks). Le tracce hanno la stessa larghezza della testina, e ne esistono migliaia per ciascun piatto. Tracce adiacenti sono separate da spazi (gaps). Ciò minimizza gli errori dovuti al disallineamento della testina o all'interferenza tra i campi magnetici. Il trasferimento dati avviene per settori, che generalmente sono un centinaio per traccia, di lunghezza fissa o variabile (odiernamente 512 byte), i settori adiacenti sono inoltre separati da spazi. I bit più vicini al centro del disco ruotano attorno al punto fisso,come la testina, più lentamente dei bit esterni, questa velocità viene quindi compensata per permettere alla testina di leggere tutti i bit alla stessa velocità facendo ruotare il disco a velocità angolare costante.

La formattazione è l'operazione con la quale si prepara il disco per renderlo idoneo all'archiviazione e consiste ad esempio nell'inserimento di un criterio che determini la posizione di un dato settore il suo inizio. e la sua fine e un punto di partenza sulla traccia. Questi requisiti sono rispettati tramite dati di controllo memorizzati sul disco con i quali esso viene inizializzato.





Pag 193 – 195

25) Descrivere la gestione dell'I/O tramite DMA

Il DMA (Direct Memory Access) è un modulo hardware addizionale sul bus di sistema che sostituisce la CPU (emulandola) per la maggior parte delle attività di I/O.

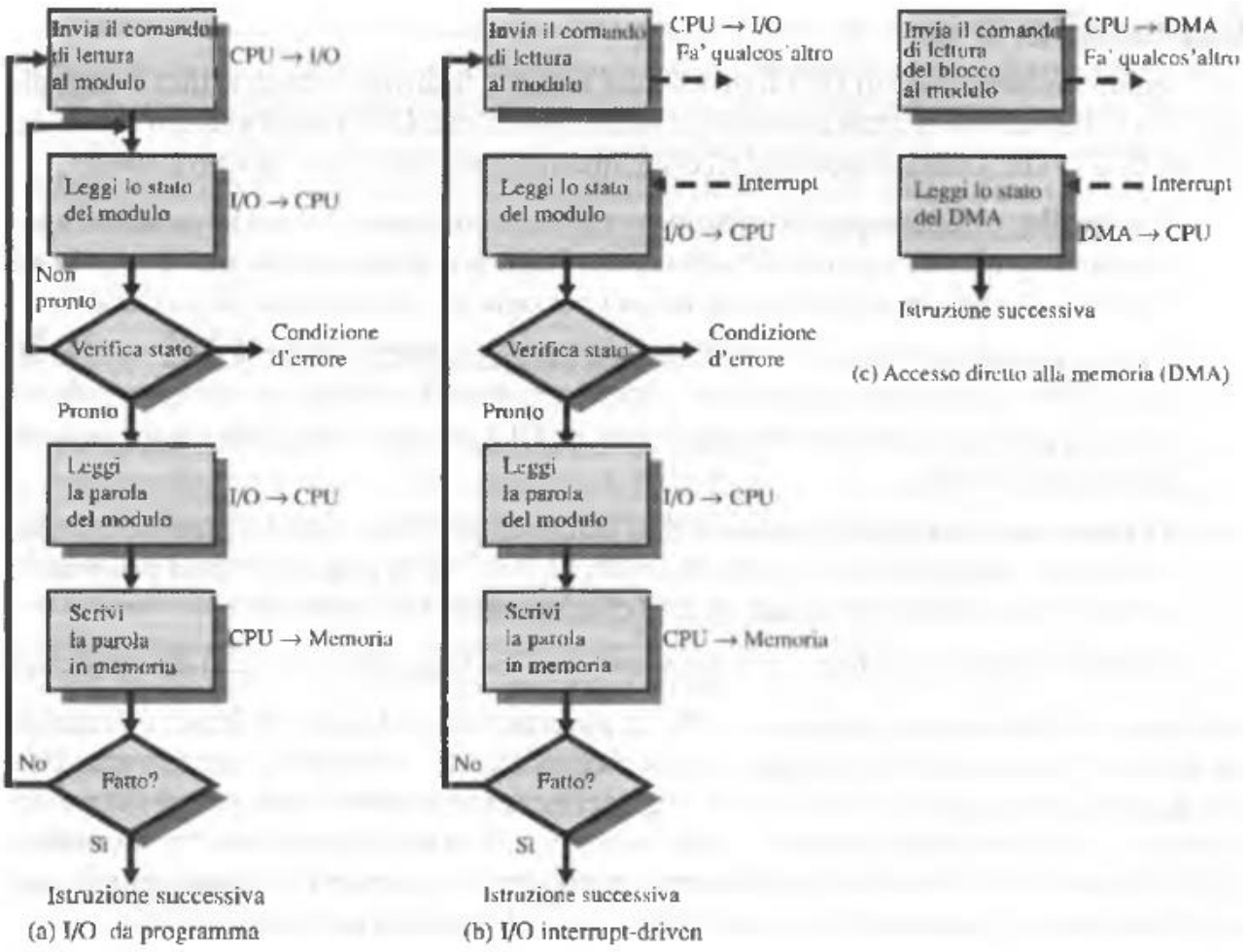
Esso si occupa del trasferimento dei dati da o verso la memoria senza passare attraverso il processore al quale invia un segnale di interrupt quando il trasferimento è completato.

In questo modo la CPU è coinvolta solo all'inizio e alla fine del trasferimento e nel frattempo può eseguire altre attività.

Prima di delegare una determinata operazione di I/O al DMA, il processore gli comunica le seguenti informazioni:

- lettura/scrittura
- indirizzo dispositivo interessato
- indirizzo iniziale in memoria del blocco dati coinvolto nell'operazione
- quantità di dati da trasferire (numero di parole da leggere / scrivere)

Il DMA è connesso al bus di sistema e può accedere al canale dati in 2 modi: una parola alla volta, sottraendo di tanto in tanto il controllo del canale alla cpu (cycle stealing) o per blocchi, prendendo in possesso il canale per una serie di trasferimenti (burst mode). Una configurazione ideale per consumare meno cicli di bus è integrare le funzioni di DMA e I/O. In questo modo il DMA utilizza il bus solo per scambiare dati con la memoria, mentre comunica in altri modi con i moduli I/O.



Pag 244 – 245

26) Discutere le ragioni per cui è stato sviluppato il sistema RAID. Inoltre si descriva in dettaglio il livello 0, 1, 2, 3, 4, 5, 6 del RAID

Il sistema RAID (Redundant Array of Indipendent Disks) venne elaborato per aumentare le prestazioni di un calcolatore grazie all'utilizzo di più dischi in parallelo (visti come uno solo dal sistema operativo), garantendo in questo modo una gestione più veloce delle diverse operazioni di input/output e, con l'aggiunta della ridondanza, una maggiore affidabilità poichè i dati vengono distribuiti su più dischi e sono recuperabili in caso di guasto di uno dei dischi (backup immediato).

- RAID 0: non inlcude la ridondanza a favore delle prestazioni, i dati sono distribuiti sui vari dischi quindi se vi sono due richieste di blocchi che si trovano in dischi diversi esse vengono servite in parallelo. I dati vengono distribuiti in strisce (strips) e sono distribuite a rotazione (round robin) sui dischi. In caso di guasto i dati sono persi e non possono essere recuperati.

- RAID 1: la ridondanza viene ottenuta duplicando tutti i dati (mirroring), questo implica un numero doppio di dischi fissi. Si usa lo striping dei dati e ogni striscia si trova fisicamente su due dischi quindi una lettura può essere soddisfatta dal disco al quale si accede più velocemente e che presenta la minor latenza di rotazione, la scrittura avviene su entrambi i dischi quindi le prestazioni sono condizionate dalla più lenta delle due scritture (quella del disco più lento). In caso di guasto, i dati sono disponibili nell'altro disco,questo permette un'alta affidabilità ma un costo elevato.

- RAID 2: sfrutta l'accesso parallelo ossia le testine di ciascun disco si trovano nella stessa posizione in ogni momento. Si usa lo striping dei dati con strisce molto piccole (singoli byte o parola), vengono generati codici di correzione errori di Hamming memorizzati in più dischi, perciò se si riscontra un errore il controllore può riconoscre e correggere l'errore istantaneamente in modo che il tempo di accesso per la lettura non venga rallentato (tale configurazione non viene commercializzata).

- RAID 3: solamente uno dei dischi nell'array è ridondante, indipendentemente dal numero totale dei dischi, utilizza un accesso parallelo con i dati distirbuiti in piccole strisce. Invece di un codice a correzione d'errore si calcola un bit di parità per l'insieme dei bit nella stessa posizione su tutti i dischi dei dati. In caso di guasto si accede al disco di parità e i dati vengono ricostruiti attravereso lo XOR con i bit degli altri dischi, oppure se il disco non è ancora stato sostituito, vengono generati sul momento. Ha una velocità di trasferimento elevata ma può eseguire solo una richiesta di I/O.

- RAID 4: invece ogni disco opera indipendentemente, così facendo le richieste di I/O possono essere gestite contemporaneamente. Usa lo striping dei dati con strisce grandi e utilizza una striscia di parità bit a bit sulle corrispondenti strisce di ciascun disco dati, e i bit di parità sono memorizzati nella corrispondente striscia sul disco di parità. RAID 4 risulta penalizzato dalle richieste di scrittura di piccola dimensione. Ciascuna operazione di scrittura coinvolge il disco di parità che può dunque diventare un collo di bottiglia in quanto aumentano la durata di accesso ai dati e della scrittura.

- RAID 5: organizzato in maniera simile al RAID 4, si differenzia in quanto distribuisce le strisce di parità su tutti i dischi attraverso l'algoritmo round robin. Per un sistema a *n* dischi la striscia di parità si trova su un disco diverso dalle prime *n* strisce, poi lo schema si ripete. Questa strategia evita il collo di bottiglia creato dal RAID 4.

- RAID 6: si effettuano due distinti calcoli della parità che sono memorizzati su dischi differenti. Così uno schema RAID 6 i cui dati richiedono *n* dischi consiste di *n + 2* dischi. Questo rende possibile rigenerare i dati anche se due dischi contenenti i dati utente si guastano (a tre i dati vengono persi). Il vantaggio consiste nell'altissima disponibilità dei dati. Incorre in una penalità di scrittura in quanto ciascuna riguarda due blocchi di parità.

(a) RAID 0 (non ridondante)

(b) RAID 1 (mirror)

(c) RAID 2 (ridondanza tramite codice di Hamming)

(d) RAID 3 (parità con bit interlacciato)

(e) RAID 4 (parità a livello di blocco)

(f) RAID 5 (parità distribuita a livello di blocco)

(g) RAID 6 (ridondanza duale)

Parte II

1) Spiegare la differenza tra la codifica dei numeri interi in complemento a 2 rispetto a quella in modulo e segno.

Nella rappresentazione modulo e segno, in una parola di n bit, gli n - 1 bit a destra contengono il modulo del numero, mentre il bit più a sinistra rappresenta il segno: + = 0 e - = 1.

Il più grande svantaggio di questa rappresentazione è che lo 0 può essere rappresentato come + 0 (0000) sia come - 0 (1000). Questo diventa difficoltoso per i calcolatori durante le operazioni, per questo tale codifica è raramente utilizzata.

La rappresentazione in complemento a due risolve questo problema codificando con il pimo bit il segno, mentre con i successivi il valore del numero:

- Avendo n bit, è possibile rappresentare numeri da -2^{n-1} a $+2^{n-1}$ -1
- Se il numero da rappresentare è positivo si procede come nella rappresentazione modulo e segno
- Se il numero è negativo viene rappresentato tramite somma pesata dei bit, ovvero viene sommato il valore in binario del bit di segno (posto a 1), agli altri bit.

Questo significa che il numero negativo più grande rappresentabile sarà 1 (bit di segno) seguito da n-1 zeri (il più piccolo sarà 1 segutio da n-1 uni).

Per rappresentare un numero negativo - k esistono due metodi:

- Si calcola la sua versione positiva k in binario, viene complementata e al complemento si somma 1
- Si calcola la sua versione positiva k in binario, vegono scritti gli stessi bit da destra a sinistra sino al primo 1, i numeri successivi vengono complementati

Pag 321 - 322

2) Si spieghi in dettaglio la codifica in complemento a due dei numeri interi. Si discutano poi i problemi legati alla realizzazione della moltiplicazione di due interi rappresentati in complemento a due, esempificando tali problemi su un caso concreti di moltiplicazione.

Attraverso la rappresentazione in complemento a due, con n bit a disposizione possiamo rappresentare tutti i numeri interi da $-2^{(n-1)}$ a $+2^{(n-1)}$-1

Il segno del numero viene indicato mediante il bit più a sinistra, il quale rappresenta - se è a 1 e + se è a 0.

La rappresentazione per i numeri positivi è identica a quella in modulo e segno, entrambe con lo zero a sinistra.

Per rappresentare un numero negativo - k esistono due metodi:

- si calcola la sua versione positiva k in binario, viene complementata e al complemento si somma 1
- si calcola la sua versione positiva k in binario, vegono scritti gli stessi bit da destra a sinistra sino al primo 1, i numeri successivi vengono complementati

Nella moltiplicazione di due interi rappresentati in complemento a due nascono problemi legati al bit più significativo, cioè al bit di segno. Se nella moltiplicazione di due numeri è presente almeno un numero negativo, il suo bit di segno verrà calcolato nella moltiplicazione ed andrà a rendere errato il risultato.

Per risolvere tali problemi bisogna utilizzare la rappresentazione in complemento a due per i prodotti parziali. La soluzione adottata è l'algoritmo di Booth, che presenta anche il vantaggio di rendere più veloce la moltiplicazione.

1001	(9)		1001	(−7)
× 0011	(3)		× 0011	(3)
00001001	1001 × 2^0		11111001	(−7) × 2^0 = (−7)
00010010	1001 × 2^1		11110010	(−7) × 2^1 = (−14)
00011011	(27)		11101011	(−21)

(a) Interi senza segno (b) Numeri in complemento a due

Pag 334 - 335

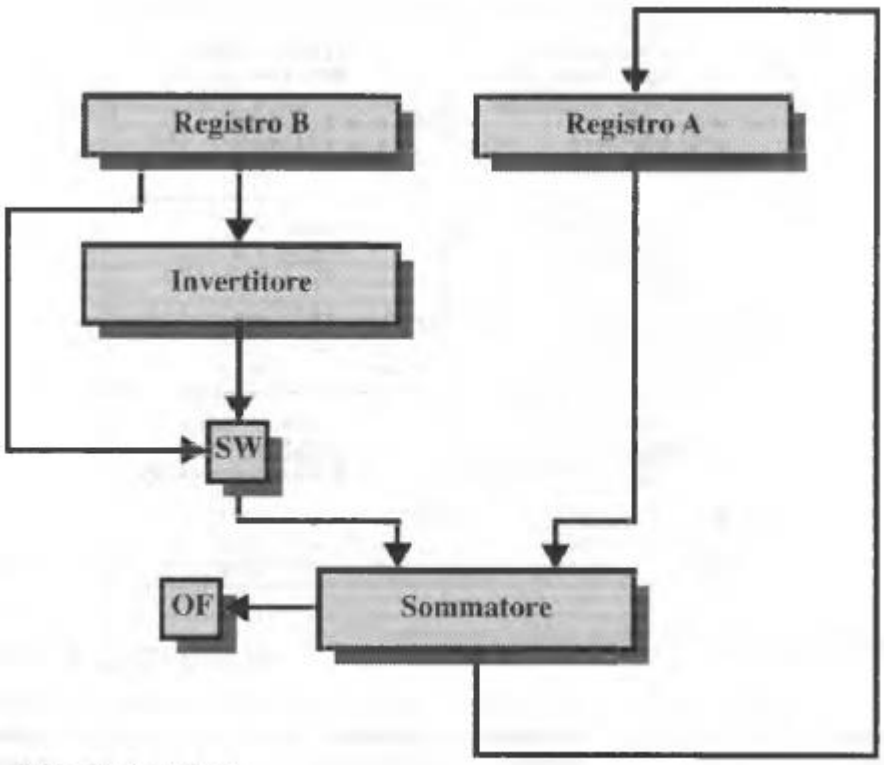
3) Si spieghi in dettaglio l'HW addotato per la realizzazione della somma e sottrazione di numeri interi rappresentati in complemento a due.

L'elemento centrale dell'hardware è il *sommatore binario* (adder) al quale vengono forniti gli addendi (o sottraendo e minuendo).Esso può produrre in risultato della somma (o sottrazione) oppure un overflow, che genera un flag che viene inserito in un apposito registro.

Un overflow può capitare quando il risultato di una somma è maggiore di quello contenibile nella dimensione della parola da usare e quando il risultato di una somma di due numeri dello stesso segno genera un numero di segno opposto.

Per la somma i due numeri provengono da due registri ed il risultato può essere memorizzato in uno dei due o in un terzo.

Nella sottrazione al sottraendo viene applicato il complemento a due (passando per un invertitore) e il sommatore legge questa nuova modifica, l'opposto del sottraendo viene quindi sommato al minuendo.



OF = bit di overflow
SW = seleziona la somma o la sottrazione

Pag 328 – 330

4) Si spieghi in dettaglio la rappresentazione dei numeri reali secondo lo standard IEEE 754.

La più importante rappresentazione in virgola mobile è definita nello standard IEEE 754, sviluppato per facilitare la portabilità dei programmi e per incoraggiare lo sviluppo di programmi sofisticati dal punto di vista numerico.

Lo standard è ampliamente usato in tutti i processori moderni. Vengono definiti un formato singolo a 32 bit e un formato doppio a 64, rispettivamente con esponenti da 8 e 11 bit e mantisse da 23 e 52 bit. Come negli altri formati il primo bit rappresenta il segno (- = 1 e + = 0). La base è 2, e l'1 è implicito a sinistra della virgola.

I formati estesi includono bit aggiuntivi nell'esponente e nel significando (o mantissa). Essi devono essere utilizzati per i calcoli intermedi. Grazie alla loro precisione superiore diminuiscono la possibilità di un risultato finale contaminato da eccessivi errori di arrotondamento: con il loro intervallo più ampio, essi diminuiscono anche la possibilità di un overflow intermedio il cui risultato sarebbe rappresentabile in un formato di base. Una motivazione aggiuntiva per il formato esteso è che racchiude alcuni benefici del formato doppio senza incorrere nella penalità di tempo associata solitamente a una precisione più alta.

Non tutte le combinazioni di bit nei formati IEEE sono interpretate nel solito modo; infatti alcune sono usate per rappresentare le seguenti classi di numeri:

- esponenti nell'intervallo da 1 a 254 in formato singolo e da 1 a 2046 in formato doppio, vengono rappresentati numeri normalizzati non nulli in virgola mobile. L'esponente è polarizzato e varia da -126 e +127 nel formato singolo e tra -1022 e +1023 in quello doppio.
- un esponente 0 con una frazione di 0 rappresenta lo zero positivo o negativo
- un esponente di tutti 1 con una frazione di 0 rappresenta l'infinito positivo o negativo, a seconda del bit di segno
- un esponente 0 con frazione non nulla rappresenta un numero denormalizzato (in questo caso il bit a sinistra è 0 e l'esponente è -126 o -1022)
- un esponente di tutti 1 con una frazione non nulla ha il valore simbolico NaN (Not a Number)

Pag 345 - 346

5) Si spieghi in dettaglio la divisione fra numeri reali secondo lo standard IEEE 754.

Il primo passo è verificare la presenza di 0. Se il divisore è 0, viene inviata una segnalazione di errore, o il risultato viene posto a infinito, a seconda dell'implementazione. Un dividendo pari a 0 produce un risultato nullo.

Poi, l'esponente del divisore viene sottratto dall'esponente del dividendo.Ciò rimuove la polarizzazione, che deve essere risommata.

Vengono poi eseguiti dei controlli sull'underflow o sull'overflow dell'esponente.

Il passo successivo consiste nel dividere i significandi e poi si esegue la normalizzazione e l'arrotondamento.

Pag 352

6) Si spieghi in dettaglio la moltiplicazione fra numeri reali secondo lo standard IEEE 754.

Innanzitutto se almeno uno degli operandi è 0, il risultato è 0.

Il passo successivo consiste nel sommare gli esponenti; essendo memorizzati in forma polarizzata, la loro somma raddoppierebbe la polarizzazione. Quindi il valore della polarizzazione deve essere sottratto dalla somma. Il risultato potrebbe comportare un underflow o un overflow dell'esponente, che verrebbe riportato provocando la conclusione dell'algoritmo. Se l'esponente del prodotto è compreso nel proprio intervallo , è necessario moltiplicare i significandi tenendo conto dei loro segni.

La moltiplicazione viene eseguita come per gli interi. Concluso il calcolo del prodotto, il risultato viene normalizzato e arrotondato, così come avviene per la somma e la sottrazione (la normalizzazione potrebbe comportare un overflow dell'esponente).

Pag 350 - 351

7) Quali sono i fattori che determinano la lunghezza delle istruzioni di una CPU?

La lunghezza delle istruzioni condiziona (ed è condizionata) da:

- Dimensione della memoria
- Organizzazione della memoria
- Struttura del bus
- Complessità della CPU
- Velocità della CPU

Il compromesso che si cerca è ovviamente quello di avere un repertorio di istruzioni vasto e potente e la necessità di risparmiare spazio.

Più codici operativi e operandi semplificano infatti la vita del programmatore, ma ovviamente aumentano le dimensioni della memoria.

La lunghezza delle istruzioni dovrebbe essere uguale (o multipla) alla dimensione del bus di memoria. Inoltre, essa dovrebbe essere anche multipla della lunghezza di un char , quindi di 8 bit.

Pag 430 - 431, Slide

8) Si discuta in dettaglio in cosa consiste il formato variabile per le istruzioni. Se possibile dare esempi di formati variabili.

Abbiamo visto diversi progetti di macchine storiche che hanno utilizzato diversi formati di istruzioni.

Il formato di un'istruzione definisce la disposizione dei suoi bit, in termini delle sue parti costituenti e deve includere un codice operativo e, in modo implicito o esplicito, zero o più operandi.

Avere un formato variabile delle istruzioni agevola un ampio repertorio di codici operativi e di lunghezze differenti. L'indirizzamento può essere più flessibile, con varie combinazioni di riferimenti a registri e memoria e più modi di indirizzamento. Disponendo di istruzioni a lunghezza variabile, tali possibilità possono essere fornite in modo efficente e compatto. Lo svantaggioè l'incremento della complessità del processore.

Il PDP-11 è stato progettato per un linguaggio macchina potente e flessibile che rispettasse i requisiti di un microcomputer a 16 bit. Esso dispone di 8 registri generici da 16 bit (uno viene usato come puntatore alla pila). Inoltre gode dell' indipendenza definita ortogonalità. I formati comprendono istruzioni a zero, uno e due indirizzi. La lunghezza del codice operativo varia dai 4 ai 16 bit. I riferimenti ai registri sono lunghi 6 bit. Le istruzioni del PDP-11 sono lunghe solitamente una parola (16 bit), oppure da 32 e 48 bit.

Il formato delle istruzioni VAX invece è stato progettato adottando due criteri:

- Tutte le istruzioni dovrebbero avere un numero "naturale" di operandi
- Tutti gli operandi dovrebbero presentare la stessa generalità nelle specifiche.

Il risultato consiste in un codice operativo di 1 o 2 byte seguito da alcuni specificatori di operando a seconda del codice operativo.

La lunghezza minima di un'istruzione è 1 byte (che è sufficiente per descrivere la maggior parte delle istruzioni del VAX), ma si possono costruire istruzioni da 37 byte.

Il numero di specificatori di operando può arrivare a 6. Uno di essi occupa 1 byte in cui i 4 più a sinistra specificano il modo di indirizzamento. Uno specificatore di operando spesso consiste di un solo byte, dove i 4 più a destra specificano uno dei 16 registri generici.

Pag 435 - 440

9) Si descrivano i possibili formati di codifica di un'istruzione, specificando per ogni formato la sua composizione tipica, i pregi e i difetti.

Il formato di un'istruzione definisce la disposizione dei suoi bit in termini di parti costituenti, e deve includere (implicitamente o meno) il codice operativo detto opcode.

Il problema fondamentale è la lunghezza delle istruzioni: essa infatti dovrebbe essere uguale alla larghezza del bus di memoria o l'una dovrebbe essere multiplo dell'altra.

Inoltre, dovrebbe essere multiplo della lunghezza di un carattere, che di solito è di 8 bit.

Il formato di un istruzione può essere a lunghezza:

- Fissa:tutte le istruzioni hanno la stessa lunghezza, posso avere più formati cambiando i campi. Estremamente efficiente nel caso di pipeline.
- Variabile: ogni istruzione ha una lunghezza che dipende dal numero degli operandi specificato nel campo opcode, permettono grande flessibilità ma incrementano la complessità
- Ibrida: ho più formati con differenti lunghezze fisse

Le istruzioni a lunghezza variabile o ibrida sono sicuramente più flessibili, permettendo più modi di indirizzamento e più riferimenti, ma incrementano di molto la complessità dell' hardware (CPU). Queste caratteristiche si avvicinano di più alla filosofia CISC.

Dall'altro lato, una lunghezza fissa ha come problema la mancanza di operandi indipendenti dall' OPCODE, ne consegue che sono disponibili meno operandi da utilizzare per le operazioni. Vi è in generale mancanza di flessibilità, ma la CPU ne guadagna in termini di complessità e costi, inoltre il caricamento di un' istruzione avviene in modo più veloce rispetto alle istruzioni a lunghezza variabile, ed una dimensione fissa favorisce l'uso della pipeline. Queste caratteristiche si avvicinano alla filosofia RISC.

Pag 430 - 438

10) Si descriva in dettaglio la modalità di indirizzamento indiretto, discuterne pregi e difetti l'adozione di tale indirizzamento è favorito o sfavorito in un'architettura RISC? Motivare la risposta.

Nell'indirizzamento indiretto ogni istruzione è suddivisa nei campi op-code ed indirizzo.

Il campo indirizzo contiene l'indirizzo di una cella di memoria, la quale a sua volta contiene l'indirizzo dell'operando.

Il vantaggio di tale tecnica è la possibilità di indirizzare un grande quantità di celle, precisamente 2k con k la lunghezza del campo indirizzo.

Lo svantaggio di tale metodo è che esso richiede due accessi in memoria per ottenere l'operando.

L'adozione di tale modo di indirizzamento è sfavorito in un'architettura RISC, in quanto tale architettura ottimizza l'accesso alle variabili locali mediante l'utilizzo di un ampio numero di registri e utilizza un set di istruzioni semplificato, cercando di minimizzare gli accessi in memoria.

11) Si descrivano in dettaglio le modalita di indirizzamento con spiazzamento e a pila. In particolare si confrontino criticamente i 2 modi di indirizzamento e se ne discutano i pregi e i difetti.

Un metodo di indirizzamento molto potente è l'indirizzamento con spiazzamento che combina l'indirizzamento diretto con l'indirizzamento di registro indiretto.

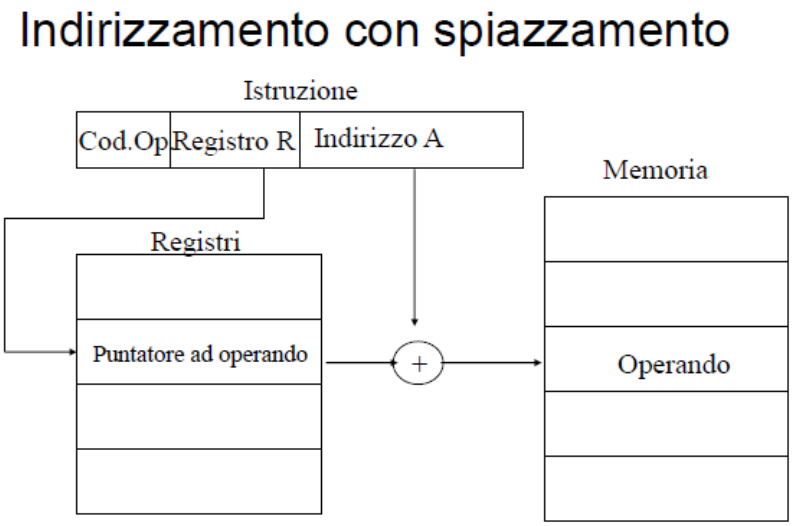
Questa tecnica richiede che l'istruzione abbia due campi indirizzo e che almeno uno sia esplicito.

EA = A + (R)

EA = indirizzo effettivo del dato in memoria

A = valore di base (diretto)

R = registro che contiene l'indirizzo di un valore da sommare ad A per ottenere l'indirizzo



La pila è una sequenza lineare di locazioni riservate nella memoria, associato a questa vi è un puntatore (SP, Stack Pointer) il cui valore è l'indirizzo della cima della pila. In alternativa i due elementi più in alto nella pila si possono trovare nella CPU, in tal caso SP referenzia il terzo elemento della pila.

Pertanto i riferimenti alla locazione della pila nella memoria sono di fatto indirizzi a registro indiretto.

Questa tecnica è una forma di indirizzamento implicito, le istruzioni macchina operano implicitamente sulla cima della pila senza accedere alla memoria.

Pag 422, 424

12) Descrivere i possibili approcci per trattare l'indirizzo di ritorno da una chiamata di una procedura.

Una procedura è una sequenza di istruzioni autonome incorporata all'interno di un programma più grande. In ciascun punto del programma la procedura può essere invocata, o chiamata. Il processore la esegue e poi ritorna al punto in cui si è verificata la chiamata.

Le due ragioni principali per l'uso di procedure sono il risparmio e la modularità.

Essa prevede due istruzioni di base: una chiamata che provoca il salto dalla locazione attuale alla prima istruzione della procedura, e una di ritorno che rimanda dalla procedura al punto della chiamata.

Esistono 3 modi efficaci per trattare il ritorno da una procedura:

- Usare un registro: se viene adottato questo approcio, CALL X (chiamata di procedura) provoca il salvataggio nel registro RN del PC e della lunghezza dell'istruzione. La chiamata di ritorno userà i dati salvati in RN
- Memorizzare l'indirizzo a inizio procedura: la procedura salva l'indirizzo di ritorno all'interno di se stessa. Ciò è pratico e sicuro
- Usare la cima di una pila: questo è l'approcio più potente e funzionale. Quando la CPU esegue la chiamata, posiziona l'indirizzo di ritorno nella cima della pila, e quando esegue il ritorno, lo riprende e lo utilizza

L' ultimo approcio è decisamente il più funzionale poiché, oltre a trasmettere l'indirizzo di ritorno, la procedura deve anche salvare dei parametri, che anziché venire salvati in registri, possono invece essere posizionati in coda alla pila.

Pag 386 - 390

13) Nel contesto di una pipeline descrivere la problematica della dipendenza dal controllo e si discuta in particolare la tecnica del buffer circolare spiegando in quali situazioni tale tecnica è particolarmente efficace.

La dipendenza da controllo si verifica quando la pipeline intraprende una decisione errata su una predizione di salto e di conseguenza porta nella pipeline istruzioni che devono essere successivamente eliminate.

Tutte le istruzioni che modificano il PC (salti condizionati e non, chiamate a e ritorni da procedure, interruzioni) invalidano la pipeline.

La fase fetch successiva carica l'istruzione seguente che non può essere quella giusta (tali istruzioni sono circa il 30% del totale medio di un programma).

Le soluzioni possono essere:

- mettere in stallo la pipeline fino a quando non si è calcolato l'indirizzo della prossima istruzione (pessima efficienza ma massima semplicità)
- individuare le istruzioni critiche per anticiparne l'esecuzione, eventualmente mediante apposita logica di controllo

La soluzione del buffer circolare per i salti condizionati consiste nell'utilizzare una memoria piccola e molto veloce (appunto il buffer circolare) formata da registri dove mantenere le ultime n istruzioni prelevate.

In caso di salto, si controlla se l'istruzione destinazione è già presente nel buffer, così da evitare il fetch della stessa.

I vantaggi sono:

- anticipando il fetch, alcune delle istruzioni successive a quella corrente saranno già presenti nel buffer e se non si ha salto non ci sarà bisogno di caricarle in memoria
- se si salta in avanti di poche istruzioni, l'istruzione destinazione sarà già presente nel buffer
- se il salto condizionale realizza un ciclo le cui istruzioni possono essere tutte contenute nel buffer, non c'è bisogno di effettuare fetch ripetuti delle sue stesse istruzioni

Pag 475 - 476

14) Nel contesto di una pipeline descrivere nel dettaglio la tecnica del data forwarding: a cosa serve?

Il data forwarding è una tecnica che cerca di ridurre il problema di hazard dei dati.

Individuata la dipendenza, il data forwarding consiste nel prelevare il dato richiesto direttamente all'uscita della ALU, attraverso appositi ciruti di bypass e MUX (regolati da unità di controllo e altre unità).

Nel caso di architettura MIPS, sono presenti circuiti EX → EX e MEM → EX.

Questa soluzione riduce notevolmente il numero di stalli di un'istruzione, e di conseguenza anche i cicli di clock che la pipeline impiega.

Per mandare i dati dall'ALU alle istruzioni successive, si potrebbe pensare di utilizzare un nuovo multiplexer da mettere davanti ad ogni ingresso in ALU. Questo consentirebbe di verificare se occorre seguire il percorso normale di istruzioni, o di inoltrare il dato nel circuito di bypass. Nella realtà, il circuito di data-forwarding può avere diverse implementazioni, ma solitamente abbiamo 3 componenti fondamentali:

- Forwarding Unit: decide se attivare il bypass attivando opportunamente MUX e multiplexer
- Hazard Detection Unit: riconosce le dipendenze, genera stalli se non sono risolvibili
- Control Unit: Manda segnali di controllo che regolano il forward dei dati

Pipeling MIPS in English

15) Nel contesto di una pipeline si consideri la tecnica che utilizza la tabella della storia dei salti. A cosa serve? Descriverla in dettaglio.

La tabella di storia dei salti è una piccola memoria cache associata allo stadio di IF.

Ciascuna riga della tabella è costituita da 3 elementi:

- indirizzo di istruzione del salto
- un certo numero di bit di storia (conservano lo stato di tale istruzione)
- informazioni inerenti all'istruzione destinazione / indirizzo istruzione / istruzione stessa

Memorizzando l'indirizzo della destinazione si ottiene una tabella più piccola ma con un tempo di prelievo maggiore rispetto a memorizzare l'istruzione stessa.

Ogni prefetch innesca una ricerca all'interno della tabella. Se non viene trovata una corrispondenza, si utilizza per il prelievo il successivo indirizzo in sequenza. Viceversa, se si riscontra una corrispondenza, si effettua una previsione sulla base dello stato dell'istruzione.

Quando l'istruzione di salto viene eseguita lo stadio di esecuzione segnala il risultato alla tabella di storia dei salti.

Lo stato dell'istruzione viene aggiornato per riflettere una previsione corretta o errata.

Quando si incontra un'istruzione di salto condizionato non presente nella tabella, quest'ultima viene aggiornata cancellando una riga esistente (scelta in base a uno degli algorimti di sostituzione in vigore nella cache).

Pag 477 - 479

16) Nel contesto di una pipeline, descrivere nel dettaglio la tecnica di predizione del salto utilizzando 2 bit di predizione.

Per anticipare se un salto verrà intrapreso possono essere utilizzati vari approcci dinamici di predizione che cercano di migliorare la previsione memorizzando la storia delle istruzioni di salto di uno specifico programma.

È possibile associare a ogni istruzione di salto condizionato uno o più bit che ne riflettano la storia recente. Tali bit, detti "taken/not taken switch", spingono il processore a prendere una particolare decisione alla successiva occorrenza dell'istruzione.

Disponendo di un solo bit si può memorizzare solamente se l'ultima esecuzione dell'istruzione ha provocato o no un salto.

Se si utilizzano 2 bit di predizione è possibile memorizzare il risultato delle ultime 2 istanze di esecuzione dell'istruzione associata, oppure memorizzare uno stato in qualche altro modo.

L'algoritmo inizia leggendo la prossima istruzione di salto condizionato.

Fino a quando le successive istruzioni di salto condizionato vengono effettuate, il processo decisionale prevede che il prossimo salto verrà intrapreso.

Se una singola previsione è errata, l'algoritmo continua a prevedere che il prossimo salto verrà effettuato. L'algoritmo deve prevedere che i salti non vengano effettuati fino a quando non siano intrapresi due salti consecutivi. L'algoritmo richiede due previsioni errate consecutive prima di cambiare la propria decisione di predizione.

L'uso dei bit di storia presenta uno svantaggio: se viene presa la decisione di effettuare il salto, l'istruzione di destinazione non può essere prelevata fino a quando l'indirizzo di destinazione non viene decodificato.

Pag 477 - 480

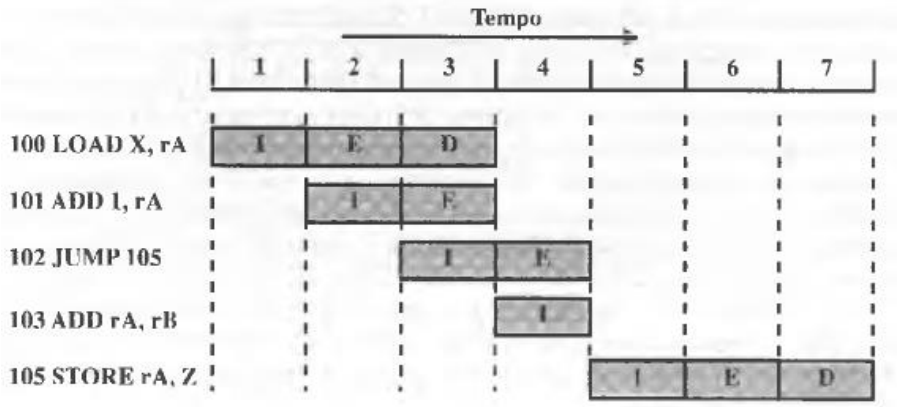
17) Nel contesto di una pipeline si spieghi in dettaglio la tecnica del salto ritardato. Fornire un esempio.

Il salto ritardato (*delayed brench*) è un modo per incrementare l'efficienza della pipeline, esso fa uso di un salto che non ha effetto fino al termine dell'istruzione seguente (da qui ritardato).

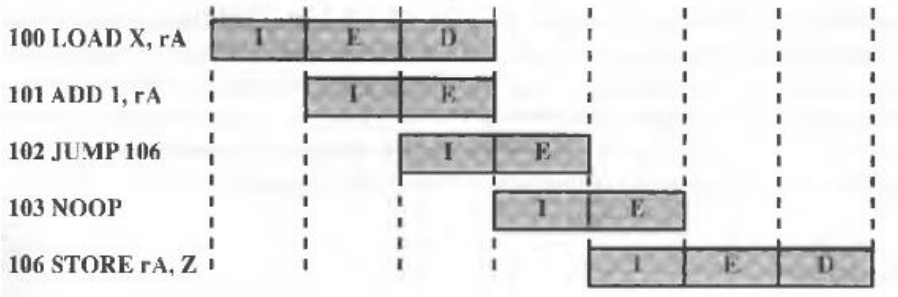
La CPU, infatti esegue sempre l'istruzione di salto, e solo in seguito altera (se necessario) la sequenza di esecuzione delle istruzioni.

La locazione successiva a quella di salto viene detta *brench delay slot* e contiene l'istruzione da eseguire.

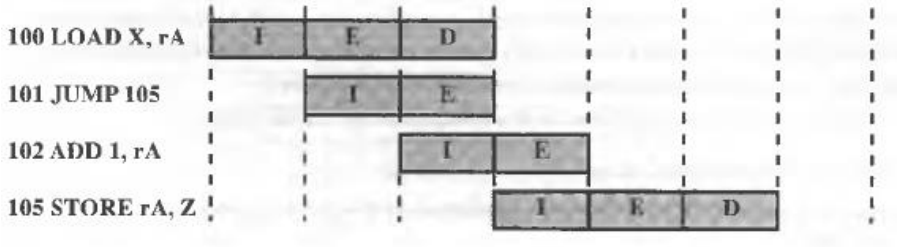
Esempio di pipeline con istruzioni di salto e come viene ottimizzata la loro esecuzione.



(a) Pipeline tradizionale



(b) Pipeline RISC con NOOP inserite



(c) Scambio di istruzioni

Questo scambio di istruzioni funziona con successo per i salti incondizionati e per le procedure. Per i salti condizionati questa procedura non può essere applicata ciecamente, la condizione da verificare può venire alterata a causa dello scambio di istruzioni effettuato dal compilatore.

Pag 526 - 527

18) Nella pipeline MIPS spiegare come lo stadio ID (Instruction Decode) è in grado di rilevare le dipendenze fra i dati.

Quando una istruzione passa dalla fase ID a quella EX si dice che la istruzione è stata "rilasciata" (issued).

Nella pipeline MIPS è possibile individuare tutte le dipendenze dai dati nella fase ID.

Se si rileva una dipendenza dai dati per una istruzione, questa va in stallo prima di essere rilasciata. Inoltre, sempre nella fase ID, è possibile determinare che tipo di data forwarding adottare per evitare lo stallo ed anche predisporre gli opportuni segnali di controllo.

La logica per decidere come effettuare il forwarding è simile a quella per individuare le dipendenze, ma considera molti più casi.

Una osservazione chiave è che i registri di pipeline contengono i dati su cui effettuare il forwarding e i campi registro sorgente e destinazione. Tutti i dati su cui effettuare il forwarding provengono dall'output della ALU e dalla memoria dati, e sono diretti verso l'input della ALU (fase EX), l'input della memoria dati (fase MEM) e il comparatore con 0.

Quindi occorre confrontare i registri destinazione di IR in EX/MEM e MEM/WB con i registri sorgente di IR in ID/EX e EX/MEM.

Pipelineing in english

19) Nel contesto della pipeline, descrivere la problematica della dipendenza dei dati e si discutano in dettaglio le tecniche viste a lezione per trattare il problema.

Un hazard si verifica quando esiste un conflitto nell'accesso alla locazione di un operando.

I tipi di conflitto generabili sono 3:

- *Scrittura dopo Lettura (RAW, Read After Write)*, o dipendenza effettiva: quando un'istruzione modifica un registro o una locazione di memoria e un'istruzione successiva legge il dato in quella locazione di memoria o quel registro
- *Lettura dopo Scrittura (WAR, Write After Read)*, o antidipendenza: un'istruzione legge un registro o una locazione di memoria e un'istruzione successiva scrive nella stessa posizione
- *Scrittura dopo Lettura (WAW, Write After Write)*, o dipendenza di output: due istruzioni devono scrivere nella stessa locazione.

Se si verifica una dipendenza di due istruzioni nel caso WAR, la seconda istruzione deve essere ritardata di tanti cicli di clock quanti sono richiesti per rimuoverne la dipendenza (un'istruzione deve essere ritardata fino a che non sono prodotti i suoi input). A lezione abbiamo visto che possibili soluzioni possono essere:

- introduzioni di fasi non operative (dette nop)
- individuazione del rischio e del prelievo del dato direttamente dall'uscita dell'ALU (data forwarding)
- risoluzione a livello del compilatore (tramite architettura MIPS)
- riordino delle istruzioni (pipeline scheduling)

Pag 474 - 475

20) Spiegare nel dettaglio la differenza fra un'architettura CISC e una RISC.

I processori con architettura RISC possono trarre beneficio dall'inconclusione di alcune caratteristiche dell'architettura CISC e viceversa. Questo significa che lle moderne macchine con architettura RISC non sono '*pure*', ma incorporano alcune caratteristiche dell'architettura CISC.

Questi sono considerati i punti tipici per un'architettura di tipo RISC:

1- Formato fisso delle istruzioni
2- Dimensione generalmente di 4 byte

3- Pochi modi di indirizzamento

I primi 3 punti indicano la complessità della codifica delle istruzioni.

4- Assenza dell'indirizzamento indiretto
5- Assenza di istruzioni che combinano operazioni aritmetiche con gli accessi in memoria
6- Assenza di istruzioi con più di un operando in memoria
7- Nessun supporto all'allineamento arbitrario di dati per le operazioni di load e store
8- Massimo utilizzo dell'unità di gestione della memoria (*MMU, Memory Managment Unit*) per l'indirizzo dei dati di un'istruzione

I punti dal 4 all' 8 suggeriscono la facilità / difficoltà del pipelining in presenza di requisiti di memoria virtuale.

9- Almeno 5 bit per l'identificatore dei registri interi
10- Almeno 4 bit per l'identificatore dei registri in virgola mobile

I punti 9 e 10 sono relativi alle capacità di avvalersi di compilatori.

Pag 523

21) Spiegare le caratteristiche base dell'architettura RISC.

Sebbene siano stati escogitati svariati approcci alle architetture RISC (Reduced Instruction Set Computer), determinate caratteristiche sono comuni a tutte:

- ***completamento di un'istruzione per ciclo macchina***: prelevare due operandi dai registri, eseguire un'operazione ALU e memorizzare i risultati in un registro. La necessità di usare microcodice si annulla con istruzioni di durata un ciclo

- ***operazioni da registro a registro***: ovvero gli operandi sono contenuti nei registri della CPU, accedendo alla memoria solamente con le operazioni di LOAD e STORE. In questo modo si semplifica l'insieme delle istruzioni e quindi l'unità di controllo. Un vantaggio è che questa architettura incoraggia l'ottimizzazione dell'uso dei registri

- ***semplici modi di indirizzamento***: quasi tutte le istruzioni RISC utilizzano l'indirizzamento a registro (R), altri modi possono essere sintetizzati via software a partire da quelli semplici. Ancora una volta questa caratteristica semplifica l'insieme delle istruzioni e l'unità di controllo

- ***semplici formati delle istruzioni***: la lunghezza delle istruzioni è fissata e allineata sui confini di parola. Le posizioni dei campi, specialmente l'op-code, sono di dimensione fissa. Questa scelta presenta numerosi vantaggi: con campi fissi la decodifica dei codici operativi e l'accesso agli operandi nei registri possono avvenire simultaneamente. Ancora una volta ciò concorre alla semplificazione dell'unità di controllo. Il fetch è ottimizzato dato che si prelevano istruzioni lunghe quanto una parola. L'allineamento delle istruzioni in memoria implica pure che le istruzioni non possono scavalcare i confini di pagina.

Pag 520 - 521

22) Spiegare le motivazioni alla base dell'architettura CISC.

Le architetture CISC (Complex Instruction Set Computer) sono state sviluppate diversi motivi.

Innanzitutto in seguito all'evoluzione dei calcolatori si riscontrò un costo del software molto maggiore rispetto al costo dell'hardware. Inoltre i linguaggui di programmazione ad alto livello stavano diventato sempre più potenti e complessi.

Questo fenomeno originò un altro problema, il gap semantico, ossia la differenza tra le operazioni consentite dai linguaggi di alto livello e quelle fornite dall'architettura del calcolatore.

Come conseguenza ne derivarono l'inefficienza dell'esecuzione, la dimensione eccessiva del codice e la complessità dei compilatori.

Gli scopi dell'architettura CISC consistevano infatti nel

- facilitare la scrittura del compilatore
- migliorare l'efficienza dell'esecuzione
- supportare i linguaggi ad alto livello più complessi

Tramite il CISC si ottengono programmi (teoricamente) più piccoli e più veloci composti da un numero sempre minore di istruzioni (e quindi di codice da prelevare) ma non è sicuro che sia più piccolo in termini di dimensioni, e di un utilizzo minore di pagine, riducendo la probabilità di un page fault

Pag 519 - 520

23) Si metta a confronto criticamente il modo in cui un'architettura RISC utilizza l'ampio banco di registri a sua disposizione rispetto alla gestione di una cache.

Il banco dei registri organizzato in finestre agisce come un piccolo buffer che conserva alcune delle variabili che hanno un'altà probabilità di essere usate con maggiore frequenza. Proprio per questo esso assomiglia molto ad una memoria cache.

Essi hanno però diverse differenze. Il banco dei registri, basato su finestre, contiene le variabili scalari locali delle N-1 procedure più recentemente attivate. La cache contiene una selezione delle variabili scalari usate di recente.

Il banco dei registri è più veloce, ma la cache usa lo spazio in modo più efficiente, dato che opera dinamicamente. Inoltre generalmente le cache trattano tutti i riferimenti alla memoria allo stesso modo, incluse le istruzioni e altri tipi di dato.

Il banco dei registri fa un uso inefficiente dello spazio quando non tutte le procedure richiedono l'intera dimensione della finestra ad esse allocata.

La cache è in grado di trattare variabili locali e globali. Con il banco dei registri il trasferimento dati tra registri e memoria è determinato dalla profondità di annidamento della procedura. Poiché tale profondità solitamente varia poco, l'uso della memoria è relativamente poco frequente.

Pag 512 - 513

24) Spiegare in dettaglio come le architetture RISC trattino efficacemente le chiamate annidate di procedura.

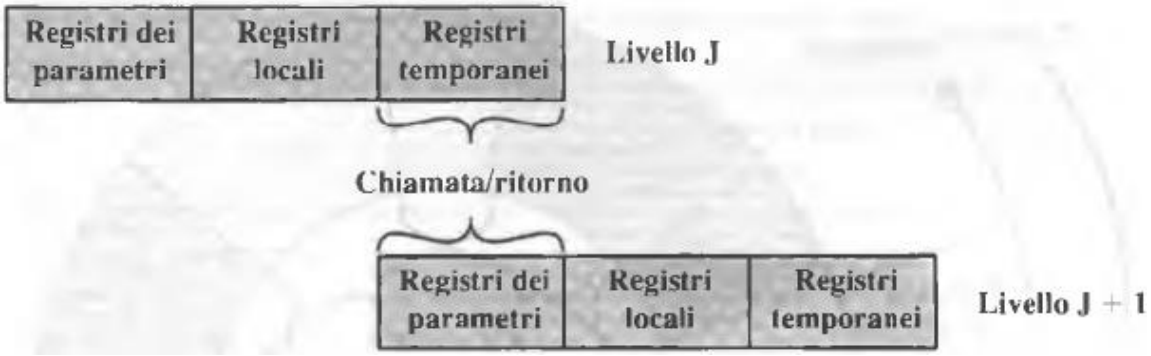
Dall'osservazione che le chiamate di procedura tipicamente coinvolgono pochi parametri e non presentano un grado di annidamento elevata, si è pensato di usare molti gruppi di registri, detti finestre di registri, per gestire le chiamate annidate.

Una chiamata seleziona automaticamente un nuovo gruppo di registri e quando essa è conclusa ed effettua il ritorno, riseleziona il gruppo di registri riferito alla chiamata che l'aveva invocata.

Ogni gruppo di registri è suddiviso in tre sottogruppi:

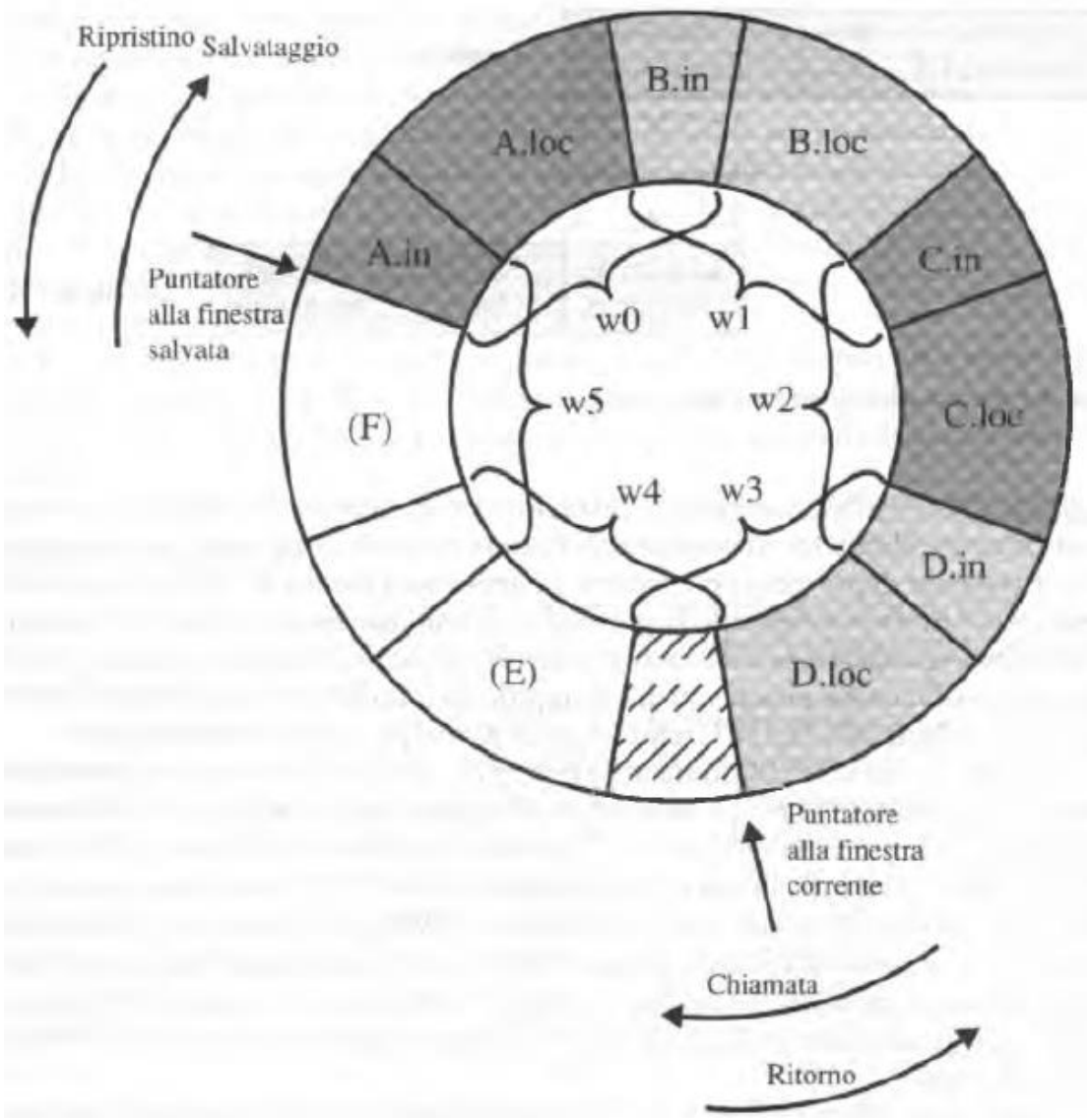
- parametri passati alla procedura
- registri che memorizzano il contenuto delle variabili locali della procedura
- registri temporanei che gestiscono il ritorno della procedura

I registri temporanei di un gruppo si sovrappongono perfettamente con quelli che contengono i parametri del gruppo successivo, cioè del gruppo riferito ad una chiamata annidata. Tali registri sono fisicamente gli stessi e ciò permette il passaggio dei parametri senza trasferimento dei dati.



La realizzazione fisica di finestre di registri sovrapposte avviene tramite buffer circolare.

Nel buffer circolare, quando avviene una chiamata il puntatore alla finestra corrente (Current Window Pointer) viene aggiornato per farlo puntare alla finestra attiva. Se si esaurisce la capacità del buffer, cioè tutte le finestre sono in uso a causa di chiamate annidate, la finestra che per prima è stata inserita nel buffer viene salvata in memoria principale e quindi sovrascritta dalla nuova. Quando una procedura termina, una finestra viene liberata e grazie ad un apposito puntatore (SWP, Stack Window Pointer) è possibile ripristinare l'ultima finestra salvata in memoria principale.



Pag 511 - 514

25) Spiegare in quale modo un compilatore possa aiutare l'utilizzo efficace dei registri da parte di un'architettura RISC.

In una macchina con architettura RISC, l'uso ottimale dei registri è lasciato al compilatore.

I programmi scritti ad alto livello non hanno espliciti riferimenti a registri, ma fanno riferimento alle variabili in maniera simbolica.

Ogni quantità che nel programma è candidata a risiedere in un registro è assegnata a un registro simbolico (o virtuale). Quindi il compilatore mappa (un numero virtualmente illimitato) di registri simbolici su registri reali del processore).

I registri simbolici il cui uso non si sovrappone temporalmente possono condividere lo stesso registro reale, cioè possono essere mappati sullo stesso registro reale.

Se i registri non sono sufficienti per contenere tutte le variabili riferite in un dato intervallo di tempo, alcune variabili vengono mantenute in memoria principale.

La tecnica comunemente usata per decidere quali quantità debbano essere assegnate ai registri in ogni dato punto del programma è la "colorazione di un grafo".

Dato un grafo, costituito da nodi connessi di archi, si assegnano dei colori ai suoi nodi in modo che nodi adiacenti non abbiano lo stesso colore, e che il numero dei colori usati sia minimo.

I nodi del grafo corrispondono a registri simbolici. Due registri che sono in "vita" all'interno di uno stesso frammento di codice sono connessi da un arco.

L'idea di fondo è colorare il grafo con n colori, dove n è il numero di registri reali. I nodi che possono essere colorati sono perciò memorizzati in memoria principale.

Pag 512 – 518

26) Nel contesto MIPS spiegare come e quando la CPU attiva i circuiti di bypass.

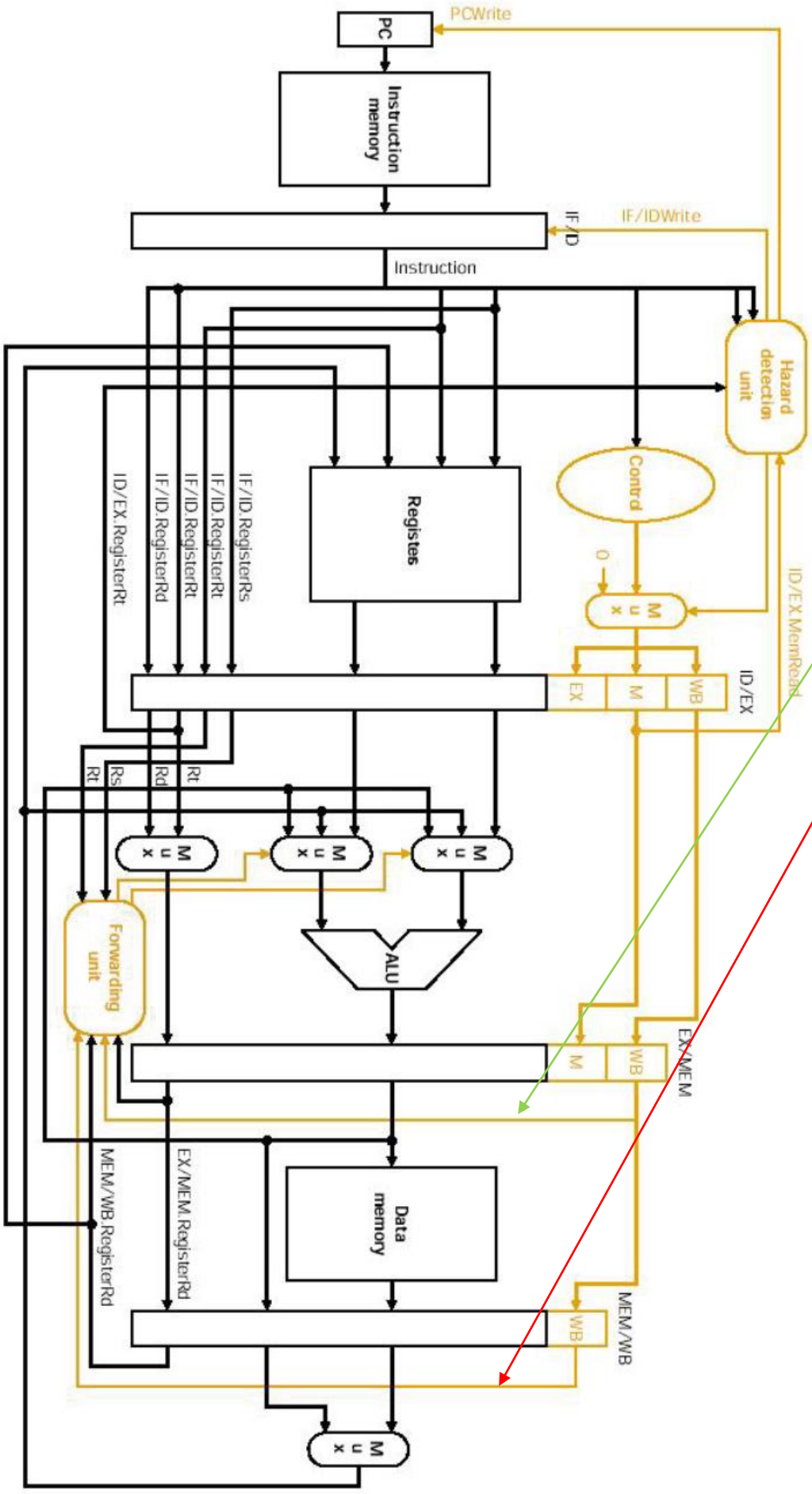
La CPU può attivare i circuiti di bypass (o propagazione) per prelevare un valore che è stato calcolato dalla ALU e che viene richiesto in input da istruzioni successive. Questo è il caso in cui viene attivato ***EX.ALUOutput_istr1 → EX.Top(o Bottom)_ALU_input_ istr2***, il dato passa dall'output della fase exe precedente all'input della fase exe successiva.

Nel caso in cui il dato fosse disponibile all'uscita dello stadio MEM, la CPU può attivare il circuito di bypass in modo da prelevare il dato dalla fase MEM per portarlo in input alla ALU in fase EXE di un'istruzione successiva. Viene attivato ***MEM.LMD_lw → EX.Top_ALU_input_istr*** (generalmente questo caso capita quando il dato da utilizzare nell'operazione aritmetica viene precedentemente prelevato da un'struzione di LOAD dalla memoria).

Slide

Il senso del data forwarding è quello di fornire alla ALU in fase di EX i dati di cui ha bisogno il prima possibile, prendendoli quindi dalle fasi EX / MEM e MEM / WB. Non ci sono altre possibilità di forwarding.

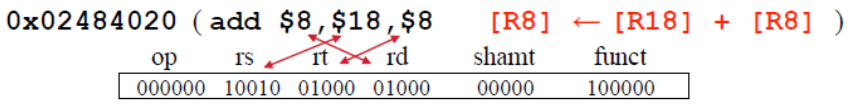
N.B. Nel primo registro della sw non è possibile effettuare alcun tipo di forwarding in quanto i dati forniti alla ALU sono lo spiazzamento e il valore del secondo registro. Il primo deve aspettare la normale esecuzione dell'istruzione che lo scrive. Questo perchè lo si usa in fase MEM e non in EX



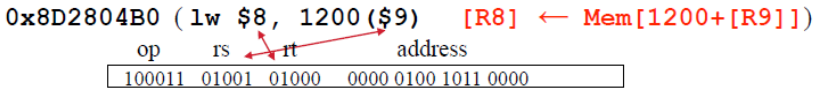
27) Si descrivano i formati delle istruzioni MIPS visti a lezione specificando per ogni formato la sua composizione tipica.

La MIPS è un tipo di architettura RISC con pipeline ottimizzata, le sue istruzioni sono da 32 bit e di 3 formati:

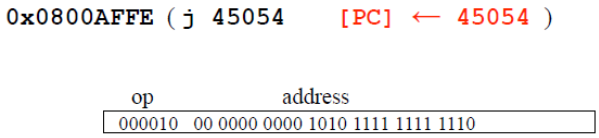
- Formato R (Registro): le operazioni avvengono sempre fra registri, quindi questo formato è utilizzato per le operazioni aritmetiche. I suoi campi sono:
  - OP: codice operativo dell'istruzione da eseguire
  - RS: primo argomento (operando) dell'istruzione, registro top
  - RT: secondo argomento (operando) dell'istruzione, registro bottom
  - RD: registro in cui verrà memorizzato il risultato
  - SHAMT: quantità di shift
  - FUNCT: variante operativa (unsigned, overflow handling...)



- Formato I (Istruzioni Load / Store): istruzioni per il trasferimento dati tra memoria e registiri (indirizzamento con displacement), può essere anche utilizzato per operazioni logico aritmetiche con operando immediato. I suoi campi sono:
  - OP: codice operativo dell'istruzione da eseguire
  - RS: primo argomento (operando) dell'istruzione
  - RT: secondo argomento (operando) dell'istruzione
  - ADDRESS: bit di displacement (da sommare al valore contenuto nel registro)



- Formato J (Jump): per le istruzioni di salto. I suoi campi sono:
  - OP: codice operativo dell'istruzione da eseguire
  - ADDRESS: indirizzo prossima istruzione da eseguire



Il formato a lunghezza fissa delle istruzioni facilita le operazioni di fetch e decode.

Slide MIPS

28) Si descriva sinteticamente l'implementazione delle istruzioni attraverso la tecnica della microprogrammazione. Si dica se questa tecnica viene utilizzata per i processori CISC o RISC, motivare la risposta.

Nell'implementazione microprogrammata la fase di fetch e quella di execute possono essere descritte in un linguaggio di microprogrammazione.

Implementazione microprogrammata: ad ogni codice operativo si fa corrispondere l'indirizzo di inizio di un microprogramma, formato da una sequenza di microistruzioni che a loro volta sono formate da microordini (ognuno corrispondente ad un segnale di controllo) registrati nella memoria ROM (Read Only Memory) detta anche memoria di controllo ed organizzati in una word chiamata word di controllo.

Nella word di controllo ogni bit corrisponde ad un microordine, ovvero rappresenta una linea di controllo.

Le unità di controllo microprogrammate si possono suddividere in due categorie: unità di controllo realizzate con:

- microprogrammazione orizzontale: quando le microistruzioni sono composte da un numero elevato di bit e quindi possono essere svolti molti compiti in parallelo, generando svariati segnali di controllo contemporaneamente;
- microprogrammazione verticale: quando le microistruzioni presentano un numero limitato di bit. La microprogrammazione verticale conduce ad una minore velocità di funzionamento, in quanto, diminuendo il numero di bit, si ha bisogno di più microistruzioni per specificare tutte le operazioni svolte con una sola microistruzione "orizzontale".

Nell'unità di controllo microprogrammata il codice operativo dell'istruzione in linguaggio macchina indirizza una locazione della memoria di mapping (memoria che contiene gli indirizzi di partenza dei microprogrammi) nella quale è registrato l'indirizzo di partenza del corrispondente microprogramma. Il μPC (micro Program Counter) contiene l'indirizzo della memoria di controllo e il μIR (micro Instruction Register) contiene la microistruzione.

Ci sono 2 alternative per realizzare la PC:

- Cablata: si realizza direttamente tramite circuiti digitali (lv di astrazione 0), soluzione tipica dell'architettura RISC
- Microprogrammata: si realizza tramite microprogrammazione (lv di astrazione 1), soluzione tipica CISC, maggior flessibilità in fase di progettazione rendendo facile modificare la sequenza di micro-operazioni.

Slide

29) Discutere le motivazioni alla base dei processori multicore.

I microprocessori hanno visto una crescita esponenziale delle prestazioni (organizzazione e frequenza clock). Il miglioramento dell'organizzazione del chip, è stato fortemente focalizzato sul parallelismo: è stata introdotta la pipeline, poi si è passati a pipeline parallele e infine a processori SMT (Simultaneous Multi Threading).

Tutta questa complessità non è però gratis: richiede infatti una logica più complessa, e di conseguenza un'area del chip maggiore per supportare il parallelismo, che però è più difficile da progettare, realizzare e testare.

Inoltre, le CPU SMT erano giunte al imite per quanto riguarda potenza richiesta e calore prodotto, per questo si è scelto di passare alle architetture multicore.

Si è supposto infatti che quest'ultima tipologia di CPU ha il potenziale per ottenere un miglioramento lineare, anche se i vantaggi prestazionali dipendono in parte dai programmi, che devono sfruttare adeguatamente questo parallelismo.

Slide

30) Si descrivano le possibilità alternative di organizzazione di un processore multicore.

L'organizzazione multicore dipende da:

- Numero di core per chip
- Numero di livelli di cache per chip (L1,L2,L3...)
- Quantità di cache condivisa

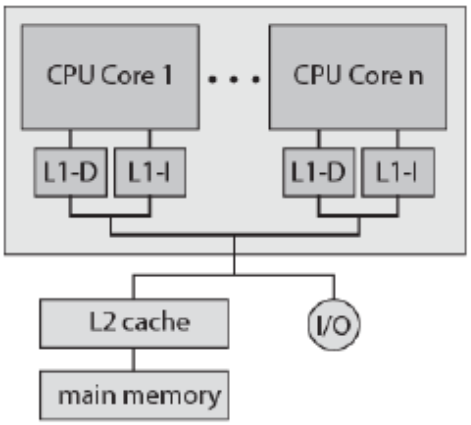
Questo divide i multicore in 4 gruppi:

- Cache L1 dedicata: Ogni CPU ha la propria cache L1
- Cache L2 dedicata: Ogni CPU possiede la propria cache L2 (oltre che L1)
- Cache L2 condivisa: Ogni CPU possiede la propria cache L1, inoltre è prevista una cache L2 condivisa tra i core

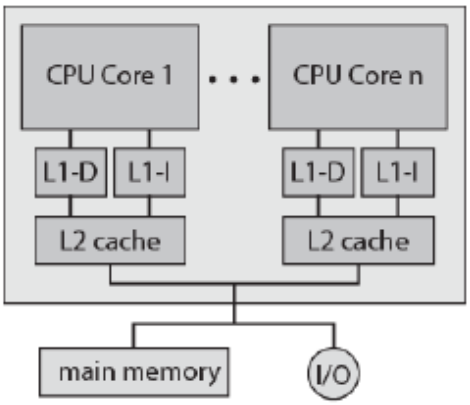
- Cache L3 condivisa: Ogni CPU possiede i propri L1 e L2 cache. Inoltre, è presente una cache L3 condivisa tra i core;

Le ultime due alternative sono le migliori, in quanto riducono il numero di miss totali. Inoltre non esiste ridondanza in cache L2 o L3 tra i due core.

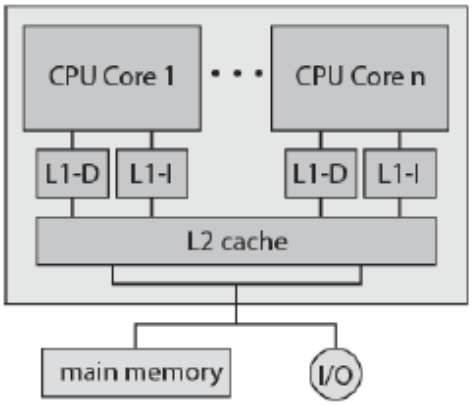
È anche possibile, tramite appositi algoritmi di sostituzione blocchi, allocare dinamicamente la cache, in modo che ogni core abbia cache dedicatache fornisce un miglioramento alla comunicazione tra processi,anche su core diversi.



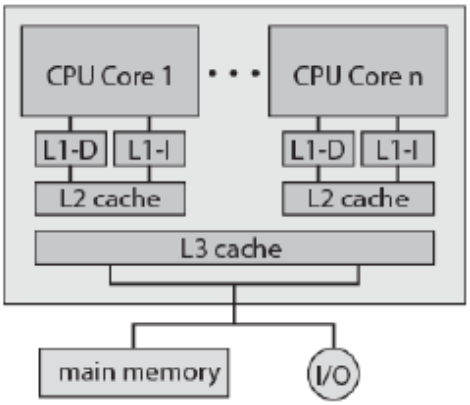
(a) Dedicated L1 cache



(b) Dedicated L2 cache



(c) Shared L2 cache



(d) Shared L3 cache

Slide

Esercizi trasferimento dati memoria centrale / cache:

Es5: Sia data la seguente sequenza di indirizzi in lettura (l) o scrittura (s) emessi dalla CPU:

	Indirizzo	l/s	dato scritto (in esadecimale)
1	000100000000	l	
2	000100001000	l	
3	000100001100	s	B1
4	000100001100	l	
5	000100010000	s	B4
6	000100010000	l	
7	000100010100	s	B7

Si assuma che la dimensione di parola coincida con un byte, e la presenza di una cache di ampiezza 16B, dimensione di blocco 4B, inizialmente vuota, e ad associazione a 2 vie (con politica di rimpiazzo LRU e politica di scrittura write-through).
Si assuma che la memoria abbia il contenuto esadecimale mostrato di seguito:

1 blocco

ind	byte	ind	byte	ind	byte	ind	byte
100	0C	101	00	102	07	103	02
104	00	105	00	106	00	107	00
108	AE	109	13	10A	A1	10B	23
10C	A1	10D	42	10E	90	10F	75
110	B9	111	16	112	00	113	00
114	0A	115	07	116	03	117	71

ind = indirizzo

Si mostri come sia il contenuto della cache che il contenuto della memoria cambia.

Numero di linee = 2^4 / 2^2 = 2^2 → Numero di set = 2^2 / 2^1 = 2^1 → 1 bit per il campo set

Dimensione di blocco = 4B = 2^2 → 2 bit per il campo parola

LRU → Least Recently Used

Write-through → aggiornamento della memoria centrale immediatamente dopo la modifica del blocco in cache

Indirizzo (HEX / bin)	H / M	Cache	Modifica memoria centrale
0x100 0001 0000 0000	M	SET 0 SET 1 [0C|00|07|02] [| | |] T: 0001 0000 0 T: R: R: [| | |] [| | |] T: T: R: R:	
0x108 0001 0000 1000	M	SET 0 SET 1 [0C|00|07|02] [| | |] T: 0001 0000 0 T: R: R: [AE|13|A1|23] [| | |] T: 0001 0000 1 T: R: R:	
0x10C 0001 0000 1100	M	Prima della scrittura: SET 0 SET 1 [0C|00|07|02] [A1|42|90|75] T: 0001 0000 0 T: 0001 0000 1 R: R: [AE|13|A1|23] [| | |] T: 0001 0000 1 T: R: R: Dopo la scrittura: SET 0 SET 1 [0C|00|07|02] [B1|42|90|75] T: 0001 0000 0 T: 0001 0000 1 R: R: [AE|13|A1|23] [| | |] T: 0001 0000 1 T: R: R:	Mem[10C] = B1
0x10C 0001 0000 1100	H	SET 0 SET 1 [0C|00|07|02] [B1|42|90|75] T: 0001 0000 0 T: 0001 0000 1 R: R: [AE|13|A1|23] [| | |] T: 0001 0000 1 T: R: R:	
0x110 0001 0001 0000	M	Prima della scrittura: SET 0 SET 1 [B9|16|00|00] [B1|42|90|75] T: 0001 0001 0 T: 0001 0000 1 R: R: [AE|13|A1|23] [| | |] T: 0001 0000 1 T: R: R: Dopo la scrittura: SET 0 SET 1 [B4|16|00|00] [B1|42|90|75] T: 0001 0001 0 T: 0001 0000 1 R: R: [AE|13|A1|23] [| | |] T: 0001 0000 1 T: R: R:	Mem[110] = B4

0x110 0001 0001 0000	H	SET 0 [B4|16|00|00] T: 0001 0001 0 R: [AE|13|A1|23] T: 0001 0000 1 R:	SET 1 [B1|42|90|75] T: 0001 0000 1 R: [| | |] T: R:	
0x114 0001 0001 0100	M	Prima della scrittura: SET 0 [B4|16|00|00] T: 0001 0001 0 R: [AE|13|A1|23] T: 0001 0000 1 R: Dopo la scrittura: SET 0 [B4|16|00|00] T: 0001 0001 0 R: [AE|13|A1|23] T: 0001 0000 1 R:	SET 1 [B1|42|90|75] T: 0001 0000 1 R: [0A|07|03|71] T: 0001 0001 0 R: SET 1 [B1|42|90|75] T: 0001 0000 1 R: [B7|07|03|71] T: 0001 0001 0 R:	Mem[114] = B7

Esercizi trasferimento dati memoria centrale / cache:

Sia data la seguente sequenza di indirizzi in lettura (l) o scrittura (s) emessi dalla CPU e che la memoria abbia il contenuto esadecimale mostrato di seguito:

#	indirizzo (binario)	l/s	byte scritto (HEX)
1	000100000100	l	
2	000100001100	s	3F
3	000100001111	l	
4	000100001101	s	A9
5	000100010100	l	
6	000100011111	s	5E
7	000100000111	s	66
8	000100100110	l	

ind	byte	ind	byte	ind	byte	ind	byte
100	08	101	D0	102	07	103	02
104	00	105	00	106	00	107	00
108	AE	109	13	10A	A1	10B	23
10C	A1	10D	42	10E	90	10F	75
110	BB	111	16	112	00	113	00
114	0A	115	87	116	03	117	71
118	3E	119	13	11A	A1	11B	23
11C	A1	11D	82	11E	90	11F	15
120	F9	121	86	122	A0	123	00
124	E9	125	16	126	05	127	00

Si assuma che la dimensione di parola coincida con un byte, e la presenza di una cache di ampiezza 32B, dimensione di blocco 4B, inizialmente vuota, e ad associazione a 2 vie (politica di rimpiazzo FIFO, politica di scrittura write-through e gestione dei miss in scrittura con la politica write allocate).

Si mostri come sia il contenuto della cache che il contenuto della memoria cambia.

Numero di linee = 2^5 / 2^2 = 2^3 → Numero di set = 2^3 / 2^1 = 2^2 → 2 bit per il campo set

Dimensione di blocco = 4B = 2^2 → 2 bit per il campo parola

FIFO → First In First Out

Write-through → aggiornamento della memoria centrale immediatamente dopo la modifica del blocco in cache

Write allocate → nel caso di miss in scrittura il blocco su cui scrivere viene prima portato in cache e poi modificato

Indirizzo (HEX / bin)	H / M	Cache		Modifica memoria centrale
0x104 0001 0000 0100	M	SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [00|00|00|00] T:0001 0000 R: [| | |] T: R:	
		SET 10 [| | |] T: R: [| | |] T: R:	SET 11 [| | |] T: R: [| | |] T: R:	
0x10C 0001 0000 1100	M	Prima della scrittura: SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [00|00|00|00] T:0001 0000 R: [| | |] T: R:	Mem[10C] = 3F
		SET 10 [| | |] T: R: [| | |] T: R: Dopo la scrittura:	SET 11 [A1|42|90|75] T:0001 0000 R: [| | |] T: R:	
		SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [00|00|00|00] T:0001 0000 R: [| | |] T: R:	
		SET 10 [| | |] T: R: [| | |] T: R:	SET 11 [3F|42|90|75] T:0001 0000 R: [| | |] T: R:	
0x10F 0001 0000 1111	H	SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [00|00|00|00] T:0001 0000 R: [| | |] T: R:	
		SET 10 [| | |]	SET 11 [3F|42|90|75]	

		T: R: [| | | |] T: R:	T:0001 0000 R: [| | | |] T: R:
0x10D 0001 0000 1101	H	Prima della scrittura: SET 00 [| | | |] T: R: [| | | |] T: R: SET 01 [00|00|00|00] T:0001 0000 R: [| | | |] T: R: SET 10 [| | | |] T: R: [| | | |] T: R: SET 11 [3F|42|90|75] T:0001 0000 R: [| | | |] T: R: Dopo la scrittura: SET 00 [| | | |] T: R: [| | | |] T: R: SET 01 [00|00|00|00] T:0001 0000 R: [| | | |] T: R: SET 10 [| | | |] T: R: [| | | |] T: R: SET 11 [3F|A9|90|75] T:0001 0000 R: [| | | |] T: R:	Mem[10D] = A9
0x114 0001 0001 0100	M	SET 00 [| | | |] T: R: [| | | |] T: R: SET 01 [00|00|00|00] T:0001 0000 R: [0A|87|03|71] T: 0001 0001 R: SET 10 [| | | |] T: R: [| | | |] T: R: SET 11 [3F|A9|90|75] T:0001 0000 R: [| | | |] T: R:	
0x11F 0001 0001 1111	M	Prima della scrittura: SET 00 [| | | |] T: R: SET 01 [00|00|00|00] T:0001 0000 R:	Mem[11F] = 5E

		[| | |] T: R:	[0A|87|03|71] T: 0001 0001 R:	
		SET 10 [| | |] T: R: [| | |] T: R:	SET 11 [3F|A9|90|75] T:0001 0000 R: [A1|82|90|15] T: 0001 0001 R:	
		Dopo la scrittura: SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [00|00|00|00] T:0001 0000 R: [0A|87|03|71] T: 0001 0001 R:	
		SET 10 [| | |] T: R: [| | |] T: R:	SET 11 [3F|A9|90|75] T:0001 0000 R: [A1|82|90|5E] T: 0001 0001 R:	
0x107 0001 0000 0111	H	SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [00|00|00|66] T:0001 0000 R: [0A|87|03|71] T: 0001 0001 R:	Mem[107] = 66
		SET 10 [| | |] T: R: [| | |] T: R:	SET 11 [3F|A9|90|75] T:0001 0000 R: [A1|82|90|5E] T: 0001 0001 R:	
0x126 0001 0010 0110	M	SET 00 [| | |] T: R: [| | |] T: R:	SET 01 [E9|16|05|00] T:0001 0010 R: FIFO [0A|87|03|71] T: 0001 0001 R:	
		SET 10 [| | |] T: R: [| | |] T: R:	SET 11 [3F|A9|90|75] T:0001 0000 R: [A1|82|90|5E] T: 0001 0001 R:	

Esercizi trasferimento dati memoria centrale / cache:

Sia data la seguente sequenza di indirizzi in lettura (l) o scrittura (s) emessi dalla CPU:

#	indirizzo (binario)	l/s	byte scritto (esadecimale)
1	000100001000	l	
2	000100001100	l	
3	000100001111	s	1F
4	000100011101	s	AD
5	000100101000	s	09
6	000100011000	l	
7	000100000011	l	
8	000100100101	l	

b) Si assuma che la dimensione di parola coincida con un byte, e la presenza di una cache di ampiezza 16B, dimensione di blocco 4B, inizialmente vuota, e ad associazione a 2 vie (politica di rimpiazzo FIFO, politica di scrittura write-back e gestione dei miss in scrittura con la politica write allocate).

ind	byte	ind	byte	ind	byte	ind	byte
100	08	101	00	102	07	103	02
104	00	105	00	106	00	107	00
108	AE	109	13	10A	A1	10B	23
10C	A1	10D	42	10E	90	10F	75
110	B9	111	16	112	00	113	00
114	0A	115	07	116	03	117	71
118	3E	119	13	11A	71	11B	23
11C	A1	11D	82	11E	90	11F	15
120	F9	121	86	122	A0	123	00
124	E9	125	16	126	05	127	00

Si mostri come sia il contenuto della cache che il contenuto della memoria cambia.

Sapendo che i bit totali dell'indirizzo sono 12 e che il tipo di associazione in uso è set associative, bisogna dividere l'indirizzo in tag set e parola.

Sapendo che la dimensione di blocco è 4B, il numero di bit per il campo parola è 2.

Il numero totale di linee in cache è $2^4 / 2^2 = 2^2$ → il numero totale di set è $2^2 / 2^1 = 2^1$ → quindi 1 bit per il campo set.

I 9 bit rimanenti sono usati per il campo tag.

Scrittura write back → aggiornamento del blocco in memoria quando viene portato fuori dalla cache.

Rimpiazzo FIFO (First In First Out) → in caso di rimpiazzo il blocco più vecchio in cache è quello che deve essere portato fuori.

Indirizzo (HEX / bin)	H / M	Cache	Modifica memoria centrale
0x108 0001 0000 1000	M	SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [| | |] T: R: SET 1 [| | |] T: R: [| | |] T: R:	
0x10C 0001 0000 1100	M	SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [| | |] T: R: SET 1 [A1|42|90|75] T: 0001 0000 1 R: [| | |] T: R:	
0x10F 0001 0000 1111	H	SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [| | |] T: R: SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [| | |] T: R:	
0x11D 0001 0001 1101	M	Prima della scrittura: SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [| | |] T: R: SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [A1|82|90|15] T: 0001 0001 1 R: Dopo la scrittura (write allocate): SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [| | |] T: R: SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [A1|AD*|90|15] T: 0001 0001 1 R:	
0x128 0001 0010 1000	M	Prima della scrittura: SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [00|00|00|00] T: 0001 0010 1 R: SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [A1|AD*|90|15] T: 0001 0001 1 R: Dopo la scrittura (write allocate): SET 0 [AE|13|A1|23] T: 0001 0000 1 R: [09*|00|00|00] T: 0001 0010 1 R: SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [A1|AD*|90|15] T: 0001 0001 1 R:	

0x118 0001 0001 1000	M	SET 0 [3E|13|71|23] T: 0001 0001 1 R: FIFO [09*|00|00|00] T: 0001 0010 1 R:	SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [A1|AD*|90|15] T: 0001 0001 1 R:	
0x103 0001 0000 0011	M	SET 0 [3E|13|71|23] T: 0001 0001 1 R: [08|00|07|02] T: 0001 0000 0 R: FIFO	SET 1 [A1|42|90|1F*] T: 0001 0000 1 R: [A1|AD*|90|15] T: 0001 0001 1 R:	Mem[128] = 09
0x125 0001 0010 0101	M	SET 0 [3E|13|71|23] T: 0001 0001 1 R: [08|00|07|02] T: 0001 0000 0 R: FIFO	SET 1 [E9|16|05|00] T: 0001 0010 0 R: FIFO [A1|AD*|90|15] T: 0001 0001 1 R:	Mem[10F] = 1F

Esercizi conversione virgola mobile:

Si consideri la seguente rappresentazione in virgola mobile nello standard IEEE 754 a singola precisione:

00111110001110000000000000000000

Che numero rappresenta in decimale?

Sapendo che è a precisione singola è formato da 32 bit, di cui il primo rappresenta il segno, i seguenti 8 sono l'esponente polarizzato e i restanti 23 sono la mantissa (tenendo a mente che l'1 prima della virgola è impicito).

Il primo bit è 0 quindi il numero è positivo.

01111100_{BIN} → 124_{DEC}

A 124 bisogna togliere la polarizzazione → 124 – 127 = -3

N.B. Il valore della polarizzazione è dato da $2^{k-1}-1$ con k il numero di bit dedicati per l'esponente.

Si shiftano di 3 posti la virgola verso sinistra ricordando che l'1 prima della virgola è implicito.

La mantissa diventa quindi: 0,00101110000000000000000000

Adesso si converte normalmente il numero:

$2^{-3}+2^{-5}+2^{-6}+2^{-7}$ = 0.125 + 0.03125 + 0.015625 + 0.0078125 = 0.1796875

-.*-

Si consideri la rappresentazione di numeri a virgola mobile che utilizza 1 bit per il campo segno, 4 bit per il campo esponente e 11 bit per la mantissa. Il numero -41,125 da quale sequenza di bit viene rappresentato?

Sapendo che il numero è negativo il primo bit sarà 1.

Si converte 41 in binario: 101001

Si converte 0.125 in binario: 0.001

Si mettono assieme la parte intera con quella con la virgola e si sposta la virgola verso sinistra o destra modificando l'esponente in modo da avere un solo uno prima della virgola:

$$101001.001 \times 2^0 \rightarrow 1.01001001 \times 2^5$$

Sapendo che abbiamo a disposizione 4 bit per l'esponente, i restanti per la mantissa, il numero in binario è:

1010101001001000

Possibili domande a risposta multipla:

Parte I:

Relativamente ad un bus sincrono, quale fra le seguenti affermazioni è falsa?

A) gli eventi sono determinati da un clock

B) tutti i dispositivi connessi possono leggere la linea di clock

C) tutti gli eventi partono dall'inizio di un ciclo di clock

D) non occorre un arbitro per assegnare l'uso del bus

Quale fra le seguenti funzioni non è di pertinenza del Modulo di I/O?

A) contollo e temporizzazione

B) comunicazione con CPU

C) comunicazione con i dispositivi

D) buffering dei dati

E) tutte le funzioni sopra sono di pertinenza del modulo di I/O

Nella gestione programmata dell'I/O con I/O memory-mapped, quale delle seguenti informazioni non è vera?

A) non si presta all'uso di cache

B) I/O sembra proprio come lettura / scrittura di memoria

C) non necessita di istruzioni speciali per l'I/O

D) dispositivi e memoria condividono lo stesso spazio di indirizzamento

E) è compatibile con architetture a bus multipli

F) nessuna delle risposte precedenti è corretta

La riduzione dei problemi di congestione del traffico del bus di sistema può essere ottenuta grazie all'introduzione di uno o più bus di espansione. Tale approccio tuttavia produce problemi in caso di:

A) utilizzo di uno o più dispositivi di DMA

B) gestione da programma dell'I/O con tecnica memory-mapped

C) presenza di uno o più moduli di I/O

D) presenza di cache

E) presenza di dispositivi di memorizzazione secondaria a tecnologia ottica

F) nessuna delle precedenti

Fra le informazioni riportate di seguito dire quale non è comunicata dalla CPU al dispositivo di DMA:

A) se l'operazione è di lettura

B) se l'operazione è di scrittura

C) quantità di dati da trasferire

D) codice di interruzione

E) indirizzo iniziale in memoria del blocco coinvolto nell'operazione

F) tutte le informazioni sopra sono comunicate

Fra i tempi di attesa a cui è soggetto il recupero di informazioni in un disco rigido, quale fra quelli elencati di seguito aumenta nel caso in cui si diminuisca la velocità di rotazione del disco?

A) tempo di attesa del dispositivo

B) tempo di attesa del canale

C) tempo di posizionamento

D) tempo di latenza

E) nessuna delle precedenti

Parte II:

Quante volte la CPU deve accedere alla memoria quando preleva ed esegue un'istruzione che ha 2 operandi, uno con modo di indirizzamento diretto e uno con indirizzamento indiretto?

A) 2

B) 3

C) 1

D) 4

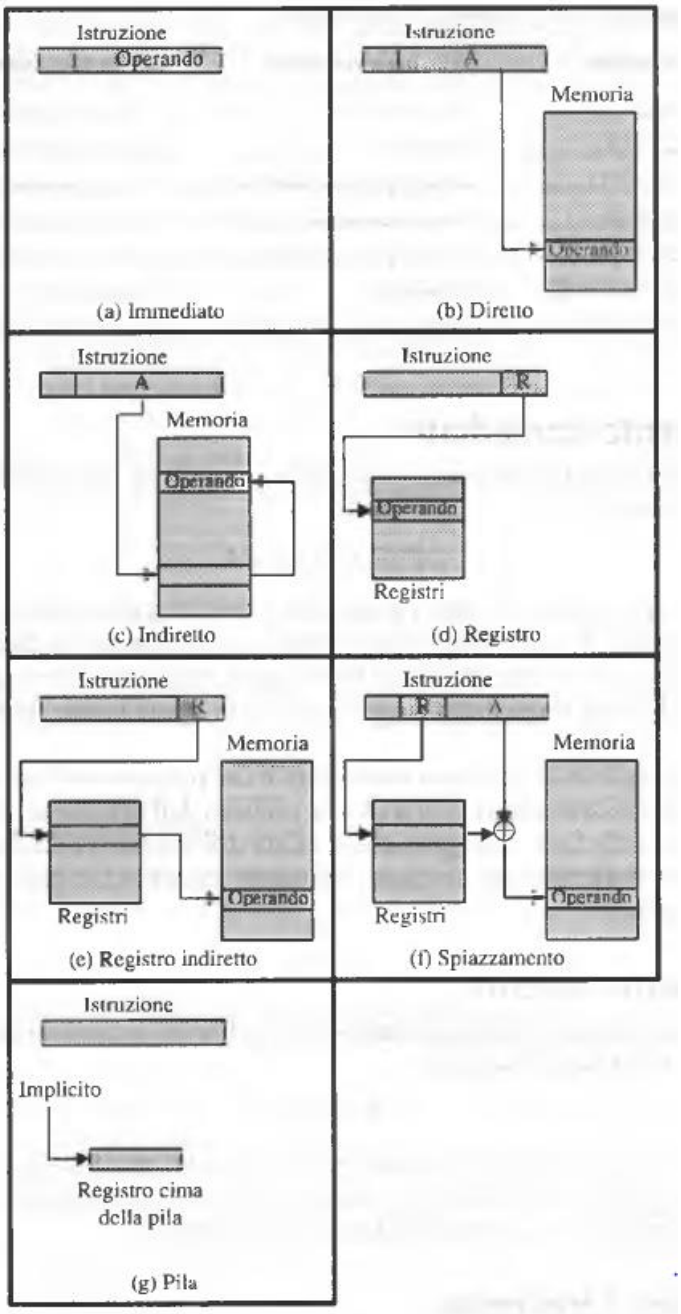
Quante volte la CPU deve accedere alla memoria quando preleva ed esegue un'istruzione che ha 2 operandi, uno con modo di indirizzamento registro e uno con indirizzamento immediato?

A) 2

B) 3

C) 1

D) 4



NB: il fetch accede di suo 1 volta alla memoria per prelevare l'istruzione da eseguire