

## Principali Innovazioni nei Computer (1)

- Il concetto di famiglia
  - IBM System/360, anno 1964
  - DEC PDP-8
  - Separa l'architettura dall'implementazione
- Unità di Controllo Microprogrammata
  - Idea iniziale di Wilkes, anno 1951
  - Introdotta nell' IBM System/360, anno 1964
- Memoria Cache
  - IBM System/360 model 85, anno 1969

## Principali Innovazioni nei Computer (2)

- RAM a Semiconduttori
  - anno 1970
- Microprocessori
  - Intel 4004, anno 1971
- Pipeline
  - già vista a lezione
- Processori Multipli

## Passo successivo - RISC

- **Reduced Instruction Set Computer**
- **Caratteristiche chiave**
  - Numero elevato di registri ad uso generale
  - ... oppure utilizzo di compilatori per ottimizzare l'uso dei registri
  - Set istruzioni semplice e limitato
  - Ottimizzazione della pipeline (basata sul formato fisso per le istruzioni, metodi indirizzamento semplici,...)

## Comparazione fra vari processori

Characteristic	Complex Instruction Set (CISC) Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2-6	2-57	1-11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

## Perchè erano nate le architetture **Complex Instruction Set Computer (CISC)** ?

- Costo del software molto maggiore del costo dell' hardware
  - Linguaggi ad alto livello sempre più complessi
  - Gap semantico
- conseguenze:
- Set di istruzioni ampio
  - Svariati modi di indirizzamento
  - Implementazione hardware di costrutti di linguaggi ad alto livello
    - Ad esempio: CASE (switch) su VAX

## Scopi del CISC

- Facilitare la scrittura del compilatore
- Migliorare l'efficienza dell' esecuzione
  - Operazioni complesse implementate tramite microcodice
- Supportare i linguaggi ad alto livello più complessi

## Studio delle caratteristiche di esecuzione delle istruzioni

- Operazioni eseguite
  - determinano le funzioni da eseguire e le modalità di interazione con la memoria
- Operandi usati
  - tipo e frequenza determinano le modalità di salvataggio e i modi di indirizzamento
- Serializzazione dell'esecuzione
  - determina l'organizzazione della pipeline e del controllo
- Studi sviluppati a partire dalle istruzioni macchina generate dai programmi scritti in un linguaggio ad alto livello
- Utilizzate misure dinamiche raccolte durante l'esecuzione di programmi

## Operazioni

- Assegnamento
  - Trasferimento di dati
- Costrutti condizionali (IF, LOOP)
  - Controllo di serializzazione
- Chiamata/ritorno da procedura impiega molto tempo
- Alcuni costrutti dei linguaggi ad alto livello richiedono molte operazioni macchina

## Frequenza relativa di istruzioni ad alto livello [PATT82a]

	Occorrenza Dinamica		Occorrenza ponderata sulle istruzioni macchina		Occorrenza ponderata sugli accessi a memoria	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

108

## Operandi

- Principalmente variabili scalari locali
- L'ottimizzazione si deve concentrare sull'accesso alle variabili locali

	Pascal	C	Media
Costanti Intere	16%	23%	20%
Variabili scalari	58%	53%	55%
Array/Strutture	26%	24%	25%

Architettura degli elaboratori - I

Pagina 109

## Chiamate di Procedura

- Consumano molto tempo
- Dipendono dal numero di parametri passati
- Dipendono dal livello di annidamento
- La maggior parte dei programmi non eseguono chiamate multiple annidate di procedure
- La maggior parte delle variabili sono locali

## Implicazioni

Il miglior supporto si ottiene ottimizzando le caratteristiche più utilizzate e più onerose dal punto di vista del consumo di tempo

1. Ampio numero di registri o loro uso ottimizzato dal compilatore
  - Ottimizzazione dei riferimenti agli operandi
2. Progettazione accurata della pipeline
  - Predizione dei salti condizionali, etc.
3. Set istruzioni semplificato (ridotto)

## Trattamento dei registri

- Soluzione hardware
  - Usare più registri
  - In questo modo si possono mantenere più variabili nei registri
- Soluzione software
  - Registri allocati dal compilatore
  - Allocazione basata sulle variabili più usate per ogni intervallo di tempo
  - Richiede l'utilizzo di tecniche di analisi dei programmi molto sofisticate

## Registri per variabili locali

- Strategia: memorizzare variabili scalari locali nei registri
- Vantaggio: riduce l'accesso alla memoria
- Problemi:
  - ogni chiamata a procedure (o funzione) cambia la località (scope delle variabili)
  - si devono passare i parametri della chiamata
  - al ritorno dalla procedura si devono ritornare i risultati
  - si devono ripristinare (i valori del)le variabili del programma chiamante al ritorno della procedura

## Finestre di registri

- Osservazioni: tipicamente le chiamate di procedura
  - coinvolgono pochi parametri (< 6 nel 98% dei casi)
  - non presentano grado di annidamento elevato



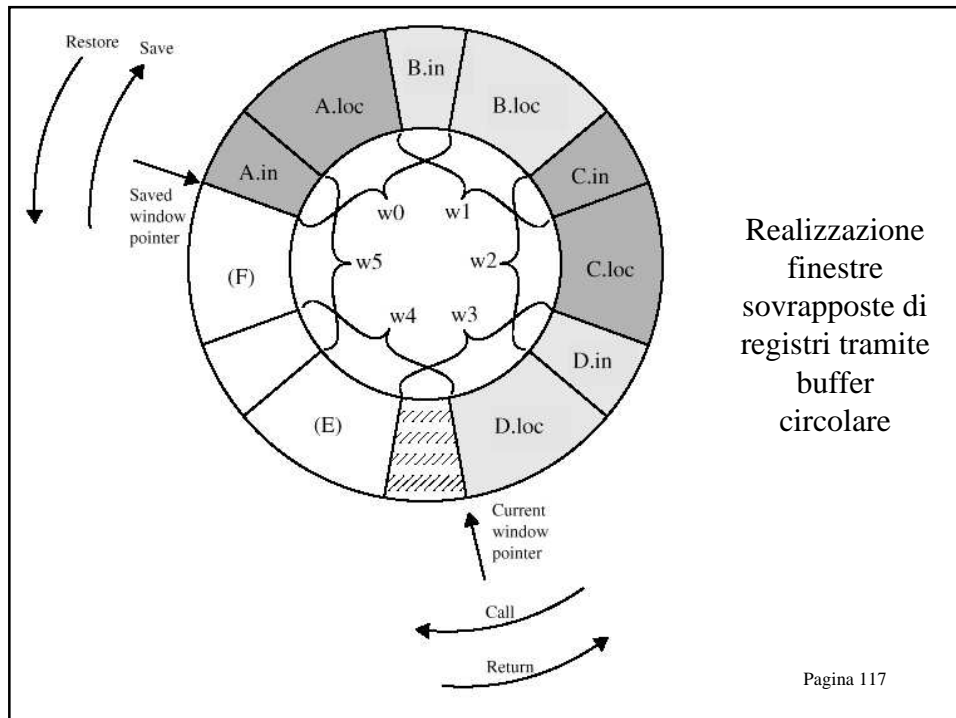
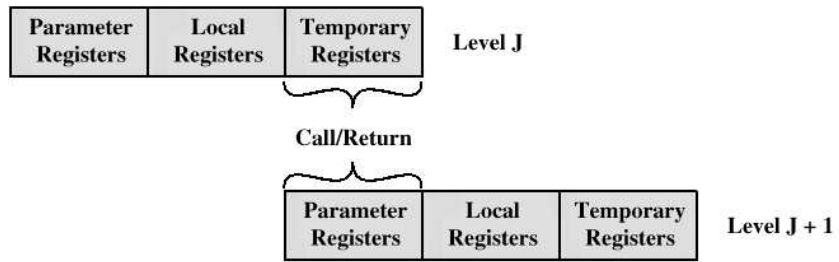
- Suggestisce la seguente soluzione: usare molti gruppi (con cardinalità limitata) di registri:
  - una chiamata di procedura seleziona automaticamente un nuovo gruppo di registri
  - il ritorno da una procedura (ri)seleziona il gruppo di registri assegnato precedentemente alla procedura chiamante

## Finestre di registri

- Ogni gruppo di registri suddiviso in tre sottogruppi
  - **registri** che contengono i **parametri** passati alla procedura chiamata
  - **registri** che memorizzano il contenuto delle **variabili locali** alla procedura
  - **registri temporanei**
  - i registri temporanei di un gruppo si sovrappongono perfettamente con quelli che contengono i parametri del gruppo successivo (cioè, sono fisicamente gli stessi registri)
  - Questo permette il passaggio dei parametri senza spostare i dati



# Finestre sovrapposte di registri



## Operazioni sul buffer circolare

- Quando avviene una chiamata, il puntatore alla finestra corrente (**Current Window Pointer**) viene aggiornato per mostrare la finestra di registri corrente attiva
- Se si esaurisce la capacità del buffer (tutte le finestre sono in uso a causa di chiamate annidate), viene generata una interruzione e la finestra più “vecchia” viene salvata in memoria principale
- Un puntatore (**Saved Window Pointer**) indica dove si deve ripristinare l’ultima finestra salvata in memoria principale

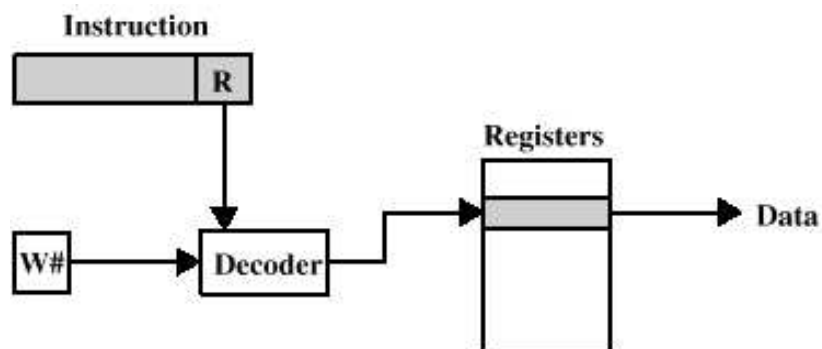
## Variabili globali

- Vengono allocate dal compilatore nella memoria
  - scelta inefficiente per variabili riferite frequentemente
- Soluzione: utilizzare un gruppo di registri per memorizzare le variabili globali

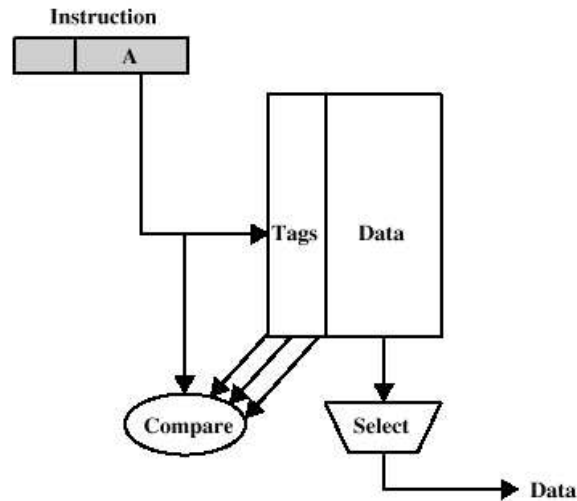
## Registri “contro” Cache

Banco di Registri Ampio	Cache
Tutti gli scalari locali	Scalari locali utilizzati di recente
Variabili individuali	Blocchi di memoria
Variabili globali assegnate dal compilatore	Variabili globali usate di recente
Save/Restore basato sulla profondità di annidamento delle procedure	Save/Restore basato sull' algoritmo di sostituzione adottato dalla cache
Indirizzamento a registro	Indirizzamento a memoria

## Riferimento a scalare - Banco di registri



## Riferimento a scalare - Cache



Architettura degli elaboratori -1

Pagina 122

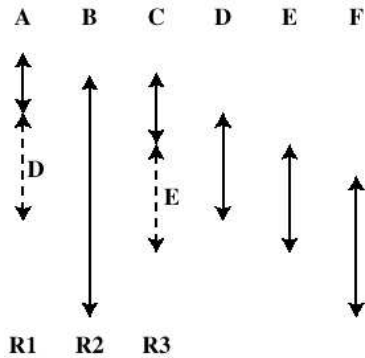
## Ottimizzazione dei registri tramite compilatore

- Assume un numero limitato di registri (16-32)
- L'ottimizzazione è lasciata al compilatore
- Linguaggi ad alto livello non fanno riferimento esplicito ai registri
  - eccezione in C: **register int**
- Il compilatore assegna un registro simbolico (o virtuale) ad ogni variabile candidata...
- ...quindi mappa (un numero virtualmente illimitato di) registri simbolici su registri reali del processore
- 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 reali non sono sufficienti per contenere tutte le variabili riferite in un dato intervallo di tempo, alcune variabili vengono mantenute in memoria principale

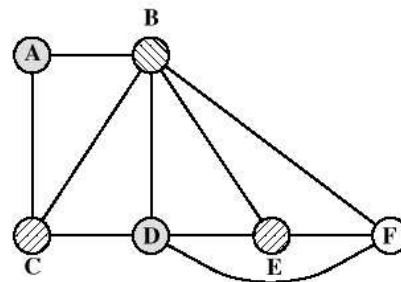
Architettura degli elaboratori -1

Pagina 123

Mapping: equivale a risolvere un problema di “colorazione” di un grafo  
[problema difficile da risolvere in generale]



(a) Time sequence of active use of registers



(b) Register interference graph

## Colorazione di un grafo

- Dato un grafo, costituito da nodi connessi da archi...
- ...si assegna un colore per ogni nodo, in modo tale che
  - nodi adiacenti (connessi da un arco) abbiano colori diversi
  - Si usi il numero minore possibile di colori
- Nel nostro caso, i nodi corrispondono a registri simbolici
- Due registri che sono “in vita” all’interno di uno stesso frammento di codice sono connessi da un arco
- Idea di fondo: colorare il grafo con  $n$  colori, dove  $n$  è il numero di registri reali
- Nodi che non possono essere colorati sono memorizzati in memoria principale

## Valutazione critica del CISC

- Semplifica il compilatore ?
  - Controverso ...
  - Istruzioni macchina complesse difficili da sfruttare
  - Ottimizzazione più difficile
- Programmi più piccoli ?
  - I programmi occupano meno memoria, ma ...
  - La memoria è diventata economica
  - Possono non occupare meno bit, ma semplicemente sembrano più corti in forma simbolica (codice mnemonico)
    - numero maggiore di istruzioni → codici operativi più lunghi
    - riferimenti a registri richiedono meno bit ...

## Valutazione critica del CISC

- Esecuzione dei programmi più veloce ?
  - Si tende ad usare istruzioni più semplici
  - Unità di controllo più complessa
  - Controllo microprogrammato necessita di più spazio ...
  - ... e quindi le istruzioni più semplici (e più usate) diventano più lente
- Non è ovvio che una architettura CISC sia la soluzione migliore

## Caratteristiche RISC

- Un'istruzione per ciclo
- Operazioni da registro a registro
- Pochi e semplici modi di indirizzamento
- Pochi e semplici formati per le istruzioni
- Formati fissi per le istruzioni
- Controllo cablato (hardware, meno flessibile ma più veloce) e non microprogrammato (più flessibile ma meno veloce)
- Maggior utilizzo della ottimizzazione a livello del compilatore

## RISC “contro” CISC

- Non emerge un “vincitore” netto
- Molti processori utilizzano idee da entrambe le filosofie:
  - ad esempio, PowerPC e Pentium II

## Controversia tra RISC e CISC

- Criterio quantitativo
  - Paragonare dimensione dei programmi e loro velocità
- Criterio qualitativo
  - Esame del merito nel supportare linguaggi ad alto livello o l'ottimizzazione dell'area di integrazione del chip
- Problemi
  - Non esistono architetture RISC e CISC che siano direttamente confrontabili
  - Non esiste un set completo di programmi di test
  - Difficoltà nel separare gli effetti dovuti all'hardware rispetto a quelli dovuti al compilatore
  - Molti confronti sono stati svolti su macchine prototipali e semplificate e non su macchine commerciali
  - Molte CPU commerciali utilizzano idee provenienti da entrambe le filosofie

## Esempio di architettura RISC: famiglia MIPS

- Riferimenti bibliografici: **Stallings + Hennessy & Patterson**
- Architettura molto regolare con insieme di istruzioni semplice e compatto
- Architettura progettata per una implementazione efficiente della pipeline (lo vedremo più avanti)
- Codifica delle istruzioni omogenea: 32 bit
- Co-processore per istruzioni a virgola mobile e gestione delle eccezioni