

ESERCITAZIONE 10

Sommario

- Insieme di istruzioni
- Coprocessori Matematici
- Processori RISC

1. Insieme di istruzioni

Un insieme di istruzioni deve soddisfare le seguenti caratteristiche:

- completezza;
- efficacia;
- regolarità;
- compatibilità.

1.1. Completezza

Deve essere possibile valutare qualunque funzione che sia calcolabile con una disponibilità ragionevole di memoria.

1.2. Efficacia

Istruzioni usate frequentemente devono essere eseguite rapidamente, il controllo sull'efficienza deve essere generato sul codice generato dai compilatori, occorre fornire delle primitive e non soluzioni a problemi particolari.

Nei calcolatori VAX esisteva un'istruzione a livello Assembler che permetteva di calcolare un polinomio a livello assembler. Tale istruzione viene usata assai di rado e dunque utilizza diverse risorse per la soluzione di un problema eccessivamente specifico e poco frequentemente utilizzato.

È più opportuno avere istruzioni primitive elementari che messe insieme permettono di risolvere problemi arbitrariamente complessi, piuttosto che avere un'istruzione che risolve un intero problema.

Il tempo impiegato dalla soluzione dell'intero problema è mediamente è nella maggior parte dei casi inferiore se si hanno a disposizione istruzioni elementari. Esiste qualche condizione particolare in cui l'utilizzo di un'istruzione complessa permette una soluzione più veloce.

1.3. Regolarità

Le istruzioni devono comportarsi in modo omogeneo rispetto ai modi di indirizzamento. Si dice che l'insieme di istruzioni è *ortogonale* quando ogni istruzione consente tutti i modi di indirizzamento possibili per tutti gli operandi. Un esempio di architettura perfettamente ortogonale è stata realizzata nel caso del mini-computer PDP-11.

1.4. Compatibilità

Il codice macchina deve essere eseguibile sui processori precedenti della stessa famiglia.

Si distinguono diverse famiglie di processori:

Intel: 8086, 80186, 80286, 80386, 80486 Pentium;

Motorola: 68000, 68010, 68020, 68030, 68040, 68060;

Digital: PDP-11, VAX-11

Si distinguono due tipi di compatibilità:

- compatibilità a livello sorgente: tra due processori sono compatibili i linguaggio assembler (a livello di mnemonici);
- compatibilità a livello binario: tra due processori sono compatibili i codice macchina.

2. Tipologie di istruzioni

Si distinguono le seguenti tipologie di istruzioni:

- trasferimento dati;
- operazioni aritmetico-logiche;
- controllo del flusso di programma;
- controllo specifico della CPU;
- ingresso/uscita.

2.1. Istruzioni di trasferimento dati

MOVE: trasferimento dati da sorgente a destinazione

LOAD: trasferimento dati dalla memoria a registro interno

STORE: trasferimento dati da registro interno a memoria

EXCHANGE: scambio dati tra registri interni o locazioni di memoria

SET/RESET: pone a 0/1 bit in una parola

PUSH/POP: istruzioni di gestione dello stack

2.2. Istruzioni Aritmetiche

ADD/SUB: operazioni di somma e sottrazione

MUL/DIV: operazioni di moltiplicazione e divisione

ABS: calcolo del valore assoluto

NEG: cambiamento di segno

INC/DEC: incremento o decremento di una parola

Tipo di dato varia da macchina a macchina.

Scalari		Virgola Mobile	Vettori	CPU
+ -	X /	+ - X /	+ - X /	
<input type="checkbox"/>				Intel 8085
<input type="checkbox"/>	<input type="checkbox"/>			Motorola 68020
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		IBM 360
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Cray-1

2.3. Istruzioni logiche

AND/OR/NOT/EXOR: calcolano le funzioni logiche booleane

SHIFT: scorrimento

ROTATE: scorrimento ad anello

2.4. Controllo del flusso di programma

JUMP: salto incondizionato

CMP: confronto tra operandi

JUMP cond: salto condizionato (viene effettuato il salto in corrispondenza di una determinata combinazione di valori sui flag)

CALL: chiamata a procedura

RET: ritorno alla procedura chiamante

2.5. Controllo specifico della CPU

HALT: blocco delle operazioni (prima dello spegnimento)

WAIT/HOLD: sospensione delle operazioni (fino ad un successivo reset)

NOP: non svolge operazioni

2.6. Operazioni di ingresso/uscita

Sono legate alla particolare macchina.

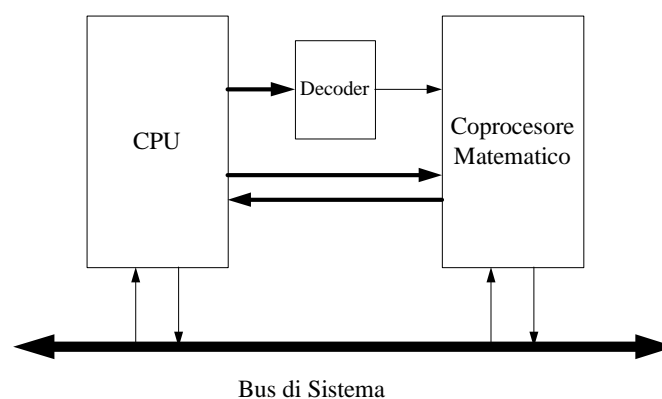
INPUT: trasferimento dati da I/O a memoria o registro interno

OUTPUT: trasferimento dati da memoria o registro interno a I/O

3. Coprocessori Matematici

I coprocessori matematici sono moduli separati dalla CPU che eseguono operazioni in virgola mobile, estremamente importanti per il calcolo scientifico. Ciascun coprocessore è dedicato ad una famiglia di microprocessore.

3.1. Collegamento coprocessore-CPU



Le istruzioni eseguite dal coprocessore matematico compaiono nel linguaggio assembler come istruzioni normali e fanno parte della sequenza di istruzioni che costituiscono un programma.

Le istruzioni per il coprocessore sono formate dai seguenti campi:

- identificatore di istruzione del coprocessore;
- indirizzo coprocessore;
- tipo di istruzione ed operando.

Esistono due possibili modi di funzionamento per un coprocessore:

1. la CPU decodifica l'istruzione ed invia gli ordini al coprocessore e si mette in attesa del risultato (la CPU controlla il coprocessore matematico);
2. il coprocessore identifica e decodifica tutte le istruzioni che passano sul bus ed esegue quelle istruzioni che sono a se stesso indirizzate (gestione controllata direttamente dal coprocessore matematico).

4. Processori RISC

Negli ultimi 3 decenni si sono contrapposte 2 tipologie di microprocessori :

- CISC (*Complex Instruction Set Computer*): processori con un set di istruzioni sempre più ampio e complesso, in grado di colmare il salto tra linguaggi macchina e linguaggi ad alto livello

- RISC (*Reduced Instruction Set Computer*): processori con un set di istruzioni ridotto.

4.1. Ragioni del RISC

Nei processori di tipo CISC la porzione di silicio occupata per la realizzazione di un'unità di controllo arriva ad occupare fino a metà dell'area totale. Questo è causato dalla eccessiva complicazione delle istruzioni macchina disponibili su processori CISC. Nei processori RISC si privilegia:

- ridurre il numero di istruzioni
- ridurre la complessità delle istruzioni
- ridurre il numero dei modi di indirizzamento
- aumentare il numero di registri interni.

Le scelte che conducono al progetto di una CPU dipendono anche dall'analisi delle caratteristiche dei programmi che questa dovrà eseguire, in quanto è auspicabile che le istruzioni ed i modi di indirizzamento più utilizzati siano particolarmente ottimizzati.

Da una analisi generale sulle caratteristiche dei programmi si conclude che:

- le istruzioni ad alto livello più frequenti sono quelle di assegnazione (circa 40%) e di salto condizionato (40%).

Se si considera il costo pesato in termini di istruzioni macchina richieste, si rileva che:

- le istruzioni di chiamata e ritorno da procedura corrispondono a circa 1/3 del totale;
- un altro terzo corrisponde alle istruzioni di iterazione;
- le istruzioni di assegnazione e salto condizionato corrispondono soltanto al 10-15% del totale.

Per quanto riguarda gli operandi, le analisi rivelano che:

- nel 55% dei casi l'operando è una *variabile scalare*; nell'80% di questi casi, tale variabile è *locale*
- nel 25% dei casi è un elemento *divettore* o *struttura*
- nel 20% dei casi è un *acostante*.

Esiste quindi una forte percentuale di accessi a dati scalari locali.

Per quanto riguarda le *procedure*, le analisi dimostrano che

- il 98% delle chiamate a procedura comporta il passaggio di un numero di parametri inferiore a 6
- il 92% di tali chiamate utilizza meno di 6 variabili locali.

Questo di nuovo significa che il numero di parole di memoria richieste da ciascuna chiamata è relativamente basso.

Le prestazioni possono essere migliorate significativamente se:

- si dispone di un numero di registri ragionevole, attraverso cui si può ridurre significativamente il numero di accessi in memoria;
- si utilizza un meccanismo intelligente di *pipeline* delle istruzioni.

I registri sono la forma di memoria con minore tempo di accesso in quanto:

- risiedono sullo stesso chip della CPU;
- sono costruiti con la tecnologia più veloce;
- sono accessibili con un meccanismo di indirizzamento semplice.

Si può guadagnare in efficienza di esecuzione in 2 modi:

- aumentando il numero di registri
- ottimizzando il loro uso.

Nell'ambito di una procedura, i registri possono essere usati:

- per contenere le variabili locali;
- per passare i parametri.

È però necessario un meccanismo per il salvataggio del loro valore quando si esegue la chiamata.

Poichè il livello di annidamento delle procedure è mediamente basso, si possono organizzare i registri in blocchi (*Register Window*).

Ogni procedura vede solo la propria *Register Window*; ad ogni chiamata a procedura o ritorno da procedura, viene cambiata la *Register Window* attiva.

Il passaggio di parametri alle procedure si può realizzare facendo parzialmente sovrapporre le *Register Window* della procedura chiamante e di quella chiamata.



Quando si supera il livello massimo di annidamento permesso, il primo blocco di registri viene salvato in memoria.

I blocchi di registri possono essere organizzati a buffer circolare.

Il *Current Window Pointer* (CWP) punta al blocco di registri della procedura corrente: i registri vengono indirizzati come offset rispetto al CWP.

Le variabili globali possono essere messe:

- in memoria;
- in un blocco speciale di registri (ad esempio i primi).

Nel secondo caso ogni procedura vede:

- n registri (i primi) per le variabili globali, il cui indice virtuale coincide con quello reale;

- m registri per le variabili locali ed i parametri, il cui indice reale viene ottenuto sommando l'indice virtuale al CWP.

4.2. Caratteristiche dei RISC

Ogni istruzione può essere eseguita in un ciclo macchina, in quanto corrisponde a:

- fetch di 2 operandi dai registri;
- attivazione della ALU;
- memorizzazione del risultato in un registro.

Le istruzioni RISC hanno la complessità delle microistruzioni CISC; per questa ragione l'unità di controllo dei RISC non è microprogrammata macabblata.

Le uniche istruzioni che coinvolgono la memoria sono LOAD (memoria P registro) e STORE (registro P memoria). Questo permette di semplificare il formato delle istruzioni e di ridurre sensibilmente il loro numero (ades, nel VAX ci sono 25 tipi di ADD; nei RISC 2).

I RISC possiedono un numero limitato di modi di indirizzamento.

I RISC hanno un formato delle istruzioni fisso o con poche alternative. Il codice operativo ha di solito una lunghezza fissa; ne conseguono alcuni vantaggi:

- la decodifica del codice operativo può avvenire in parallelo con il caricamento degli operandi dai registri;
- l'unità di controllo è più semplice;
- la fase di fetch è più ottimizzata.

4.3. Vantaggi dei processori RISC

I compilatori per RISC sono più semplici, in quanto producono codice composto da istruzioni più semplici; le sequenze ottimizzate per operazioni complesse possono essere predeterminate.

Il codice generato per un RISC ha dimensioni comparabili con quelle per un CISC in quanto il numero di istruzioni RISC generate è maggiore ma ogni istruzione occupa un numero inferiore di byte (anche perché si riducono gli accessi in memoria).

Codici più corti sono più efficienti perché si riduce il numero di fetch. La gestione ottimizzata della pipeline delle istruzioni è più semplice con i RISC.

Le istruzioni CISC, per quanto più complesse, non sono più veloci in quanto richiedono un hardware più complicato.

4.4. Esempio di processore RISC: RISC1

Il microprocessore RISC1 è stato progettato da David Patterson (Università della California, Berkeley) nel 1980. Tale microprocessore è il predecessore della famiglia di RISC SPARC.

Il processore RISC dispone di un'unità aritmetica ALU a 32 bit e contiene 138 registri general-purpose a 32 bit.

La tipica operazione eseguita da un'istruzione del processore RISC1 è la seguente:

$$R_d \leftarrow F(R_s, S2)$$

in cui R_d rappresenta il registro destinazione, R_s rappresenta il primo registro sorgente e i primi 5 bit di $S2$ definiscono il secondo registro sorgente.

Tutte le istruzioni nel processore RISC1 sono lunghe 32 bit ed hanno il seguente formato generale:

0		7 8		13		18 19		31
Codice Operativo		R_s	R_d			$S2$		

Se il bit 18 dell'istruzione è settato ad 1, allora il campo $S2$ è interpretato come valore costante su 13 bit oppure come indirizzo immediato (in questo caso l'indirizzo è automaticamente espanso su 32 bit attraverso l'estensione del segno).

A titolo di esempio si mostrano ora alcune istruzioni del processore RISC1.

4.4.1. Istruzioni di trasferimento dati

STL $R_s, (R_d)S2$; $M(R_d+S2) \leftarrow R_s$
 LDL $R_s (S2), R_d$; $R_d \leftarrow M(R_s+S2)$

4.4.2. Istruzioni aritmetico-logiche

ADD $R_s, S2, R_d$; $R_d \leftarrow R_s + S2$
 SUB $R_s, S2, R_d$; $R_d \leftarrow R_s - S2$
 AND $R_s, S2, R_d$; $R_d \leftarrow \text{AND}(R_s, S2)$
 OR $R_s, S2, R_d$; $R_d \leftarrow \text{OR}(R_s, S2)$
 EXOR $R_s, S2, R_d$; $R_d \leftarrow \text{EXOR}(R_s, S2)$
 SLL $R_s, R2, R_d$; $R_d \leftarrow R_s(S2:31), 0$ (shift logico di $S2$ bit)
 SRL $R_s, R2, R_d$; $R_d \leftarrow 0, R_s(0:31-S2)$ (shift logico di $S2$ bit)

4.4.3. Istruzioni per il controllo del flusso di programma

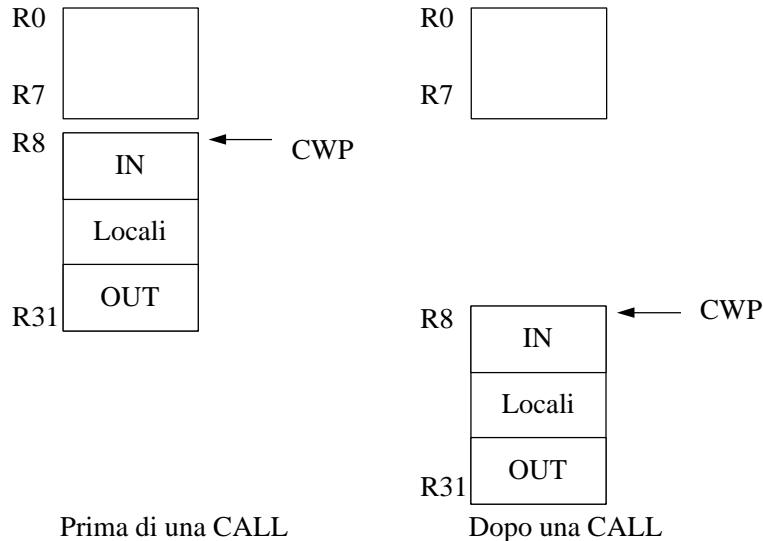
JMP cond, $S2(R_s)$; IF cond = 1 then $PC \leftarrow R_s + S2$
 CALL $R_d, S2(R_s)$; $R_d \leftarrow PC; PC \leftarrow R_s + S2; CWP \leftarrow CWP - 1$
 RET $R_s, S2$; $PC \leftarrow R_s + S2; CWP \leftarrow CWP + 1$

4.4.4. Uso dei registri in un sottoprogramma

In un processore CISC il passaggio dei parametri a procedure ed il salvataggio dei registri è un'operazione estremamente dispendiosa in quanto coinvolge accessi in memoria nello stack. In un processore di tipo RISC si è cercato di evitare tale costo utilizzando un vasto numero di registri interni.

Il processore dispone di un numero di registri complessivi pari a 138. Ad ogni istanti non tutti sono visibili contemporaneamente, sono accessibili i seguenti registri:

1. un insieme di 8 registri globali;
2. una finestra di 24 registri.



Le finestre di registri sono parzialmente sovrapposte. Le istruzioni di CALL e RET aggiornano il puntatore alla finestra corrente (CWP), ossia il puntatore alla finestra di registri a cui si fa riferimento.

Le finestre di registri sono parzialmente sovrapposte per permettere di effettuare agevolmente il passaggio di parametri.

Se il numero di chiamate a procedure cresce, può succedere che il file di registri si esaurisca: in questo caso viene salvata la parte di registri relativa alla procedura più esterna nello stack in memoria.

4.5. Futuro dei RISC

I RISC hanno avuto una costante evoluzione che ha portato alla complicazione del progetto ed all'aumento del numero delle istruzioni. La famiglia SPARC è passata da 31 istruzioni nel RISC1 alle 75 istruzioni nel processore SPARC; la famiglia MIPS è passata dalle 39 istruzioni del processore MIPS alle 183 istruzioni del processore RISC6000 (è da notare che il processore CISC Intel 80486 dispone di 183 istruzioni!)

La distanza in termini di prestazioni tra RISC e CISC si sta riducendo a vantaggio dei CISC, in quanto i notevoli miglioramenti tecnologici hanno permesso l'integrazione su un chip di sempre più raffinate funzionalità.

Inoltre i RISC iniziano a soffrire il problema dei CISC di dover mantenere la compatibilità con le versioni precedenti della famiglia.

[Torna al Sommario](#)