Architetture RISC

Introduzione

- L’analisi del comportamento in fase di esecuzione dei programmi in linguaggi ad alto livello ha fornito suggerimenti per la progettazione dell’architettura dei processori che hanno portato allo sviluppo dei calcolatori con insieme ridotto di istruzioni (RISC – Reduced Instruction Set Computer)
- Le istruzioni più frequenti che dovrebbero essere ottimizzate sono:
  - Assegnazione
  - Scelta
  - Cicli

Introduzione

- Questi studi hanno motivato le principali caratteristiche delle macchine RISC:
  - Un insieme limitato di istruzioni con un formato fissato
  - Un grande numero di registri (o comunque l'uso di un compilatore che ottimizzi l'uso dei registri)
  - Enfasi rivolta all'ottimizzazione della pipeline (critica per costrutti di scelta e cicli)

Introduzione

- Per comprendere le linee guida che hanno portato allo sviluppo di architetture RISC è necessario esaminare le caratteristiche di esecuzione delle istruzioni:
  - Operazioni eseguite. Determinano le funzionalità che devono essere realizzate dal processore e la sua interazione con la memoria
  - Operandi utilizzati. Il tipo di operandi e la frequenza del loro utilizzo determinano l'organizzazione della memoria
  - Ordine di esecuzione. Determina l'organizzazione del controllo e della pipeline

Registri

- Nei linguaggi ad alto livello è presente una grande percentuale di istruzioni di assegnazione
- Inoltre, vi è un numero significativo di accessi ad operandi per ogni istruzione
- La maggior parte degli accessi avvengono a variabili scalari locali che possono essere gestite efficacemente mediante banchi di registri interni:
  - I registri sono più veloci della memoria e della cache

Finestre di registri

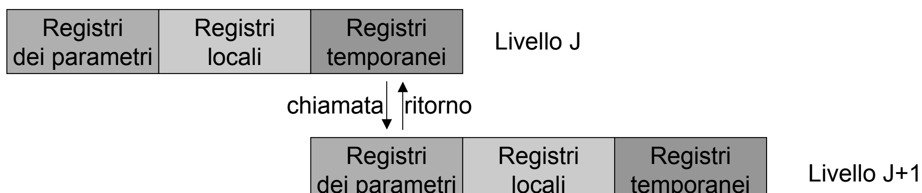
- L'uso di un grande insieme di registri dovrebbe diminuire il bisogno di accedere alla memoria:
  - Il compito progettuale è quello di organizzare i registri in modo che questo obiettivo sia raggiunto
- La maggior parte dei riferimenti riguarda variabili scalari locali
  - Il concetto di locale cambia ogni qual volta si chiama una procedura

Finestre di registri

- L'organizzazione a "finestre" divide i registri in piccoli insiemi ciascuno assegnato ad una diversa procedura:
  - Una chiamata a procedura fa sì che il processore, invece di salvare i registri in memoria, utilizzi automaticamente una diversa finestra di registri
- Le finestre corrispondenti a procedure adiacenti si sovrappongono in modo da consentire il passaggio dei parametri

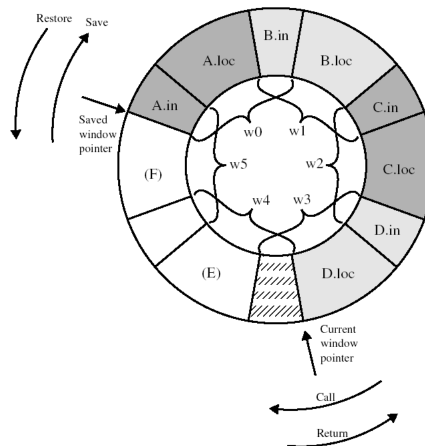
Finestre di registri

- In ogni istante è visibile una sola finestra di registri che è indirizzabile come se fosse l'unico insieme di registri
- La finestra è suddivisa in tre aree:
  - Registri dei parametri
  - Registri locali
  - Registri temporanei



Finestre di registri

- Per gestire ogni possibile schema di chiamate e di ritorni a procedura, il numero di finestre di registri dovrebbe essere illimitato:
  - Le finestre di registri possono essere utilizzate per memorizzare solo le attivazioni più recenti, mentre quelle più vecchie sono salvate in memoria
- Una reale organizzazione di un banco di registri potrebbe essere a buffer circolare di finestre sovrapposte



Variabili globali

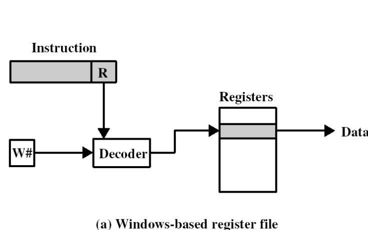
- I registri risolvono il problema dell'accesso a variabili locali ma non affrontano il problema di memorizzare variabili globali condivise da più procedure
- Una possibile soluzione è quella di mantenere le variabili globali solo in memoria → inefficiente
- In alternativa, alcuni registri all'interno del processore potrebbero essere adibiti a memorizzare solo variabili globali
  - Ad esempio i riferimenti ai registri da 0 a 7 corrispondono a registri globali unici, mentre i riferimenti ai registri da 8 a 31 potrebbero essere "traslati" per indirizzare i registri fisici nella finestra corrente

Registri vs. cache

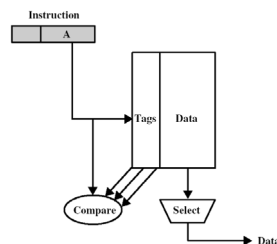
- Un banco di registri può fare un uso inefficiente dello spazio:
  - Non tutte le procedure avranno bisogno dell’intero spazio di finestra ad esse assegnato
- Per contro, la cache soffre del problema che i dati sono trasferiti a blocchi:
  - Parte dei dati trasferiti non saranno mai utilizzati
- La cache è in grado di gestire le variabili globali allo stesso modo di quelle locali
- Nel caso del banco di registri il trasferimento dati tra la memoria e i registri è determinato dal livello di annidamento
  - livello mediamente basso → uso della memoria poco frequente

Registri vs. cache

- La memoria cache è tipicamente di tipo set-associativo con insiemi di piccole dimensioni
  - Rischio che dati o istruzioni sovrascrivano frequentemente variabili utilizzate
- L’aspetto che fa propendere per il banco di registri è la velocità di indirizzamento di una variabile scalare locale



(a) Windows-based register file



(b) Cache

Ottimizzazione dei registri basata sul compilatore

- Nel caso un piccolo numero di registri sia disponibile, è compito del compilatore ottimizzarne l'uso
- In un linguaggio ad alto livello non c'è, in generale, nessun riferimento esplicito ai registri
  - Il compilatore deve mantenere gli operandi il più possibile nei registri

Ottimizzazione dei registri basata sul compilatore

- Ad ogni "quantità" del programma che può candidarsi a risiedere in un registro viene assegnato un registro simbolico
  - Il numero dei registri simbolici è illimitato
- Registri simbolici il cui utilizzo non si sovrappone possono condividere lo stesso registro fisico
- Se in una porzione di codice il numero di registri simbolici allocati supera quello dei registri fisici, alcune "quantità" vengono assegnate a locazioni di memoria
  - L'operazione di ottimizzazione consiste nel decidere quali operandi devono risiedere nei registri e quali in memoria

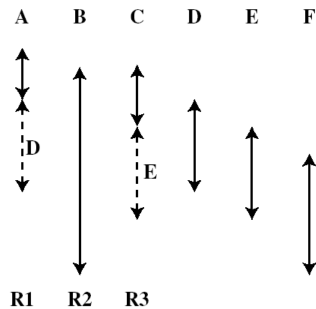
Ottimizzazione dei registri basata sul compilatore

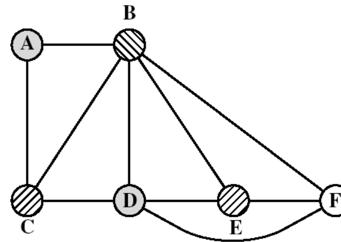
- La tecnica più comunemente usata dai compilatori per architetture RISC è quella nota come "colorazione dei grafi"
- Il problema della colorazione di un grafo (costituito da nodi e archi) è quello di assegnare dei colori ai nodi in modo che i nodi adiacenti abbiano colori diversi e che il numero di colori diversi sia il minimo possibile

Ottimizzazione dei registri basata sul compilatore

- Il programma viene analizzato in modo da costruire il cosiddetto grafo di interferenze dei registri
  - I nodi sono i registri simbolici
  - Se due registri simbolici sono attivi nello stesso frammento di codice i corrispondenti nodi sono uniti da un arco che rappresenta l'interferenza
- Successivamente si tenta di colorare il grafo con n colori, dove n indica il numero di registri
  - I nodi che condividono lo stesso colore possono essere assegnati allo stesso registro
  - I nodi che non possono essere colorati equivalgono a "quantità" da memorizzare in memoria

Esempio





(a) Time sequence of active use of registers

(b) Register interference graph

Caratteristiche dei RISC

- Alcune caratteristiche comune a tutte le architetture RISC sono:
  - Un'istruzione per ciclo
  - Operazione da registro a registro
  - Semplici modalità di indirizzamento
  - Semplici formati delle istruzioni

Un’istruzione per ciclo

- Il ciclo macchina è definito come il tempo necessario per caricare due operandi dai registri, eseguire un’operazione nella ALU e memorizzare il risultato in un registro
- Con istruzioni “semplici” ad un solo ciclo non vi è quasi necessità di microcodice, in quanto le istruzioni possono essere eseguite da logica cablata

Operazioni da registro a registro

- Avere solo istruzioni che operano da registro a registro (ad eccezione di operazioni di LOAD e STORE per caricare e memorizzare gli operandi) semplifica l’insieme di istruzioni e quindi l’unità di controllo
- Inoltre, operandi usati frequentemente tendono a rimanere nei registri e questo aumenta le prestazioni del sistema

Semplici modalità di indirizzamento

- Quasi tutte le istruzioni RISC usano l’indirizzamento mediante registro e possono essere incluse diverse modalità aggiuntive, come l’uso di scostamento o l’indirizzamento relativo al PC
- Altre modalità di indirizzamento più complesse possono essere ottenute da quelle più semplici via software
- Questa caratteristica semplifica il progetto e il funzionamento dell’unità di controllo

Semplici formati delle istruzioni

- Generalmente sono utilizzati solo uno o due formati
  - La posizione dei campi e, in particolare, quella del codice operativo è fissata
- Con campi di lunghezza fissa la decodifica del codice operativo e l’accesso agli operandi nei registri può avvenire simultaneamente
  - I formati semplificati “snelliscono” l’unità di controllo
  - Il caricamento dell’istruzione è ottimizzato

RISC vs. CISC

- Dopo un entusiasmo iniziale per le macchine RISC vi è stata una crescente consapevolezza del fatto che i progettisti RISC potevano beneficiare di alcune caratteristiche delle architetture CISC e viceversa
- Ad esempio il PowerPC non è più un RISC puro come pure il PentiumII incorpora alcune caratteristiche RISC

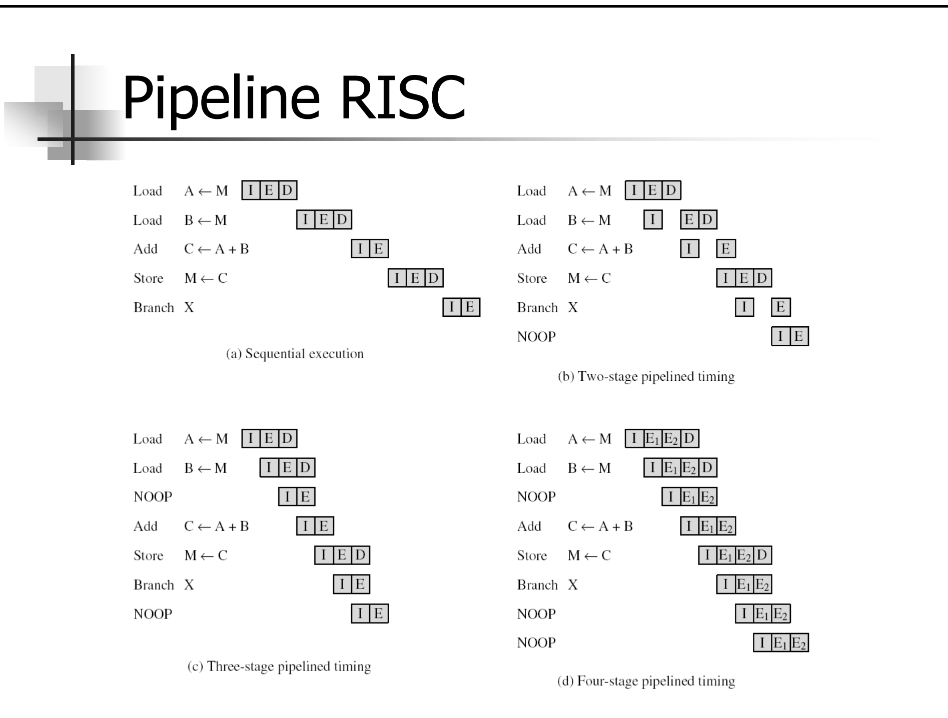
Pipeline RISC

- L'ottimizzazione della pipeline è un aspetto fondamentale delle architetture RISC
- Essendo la maggior parte delle istruzioni da registro a registro, un ciclo di istruzioni ha le seguenti fasi:
  - I: caricamento dell'istruzione
  - E: esecuzione in cui viene eseguita un'operazione dalle ALU con input e output in registri

Pipeline RISC

- Per le istruzioni che coinvolgono operazioni di lettura e scrittura in memoria sono necessarie tre fasi:
  - I: caricamento dell’istruzione
  - E: esecuzione in cui viene calcolato l’indirizzo di memoria
  - D: memoria in cui viene eseguita l’operazione di trasferimento dal registro alla memoria e viceversa

Pipeline RISC



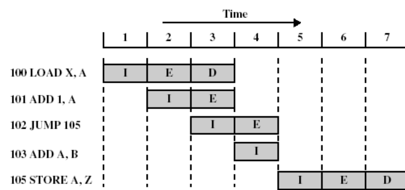
Pipeline RISC

- In uno schema di pipeline a due vie (nel quali le fasi I ed E di due diverse istruzioni possono essere eseguite contemporaneamente) può dimezzare il tempo di esecuzione rispetto ad uno schema "seriale"
- La pipeline può essere ulteriormente migliorata permettendo due accessi alla memoria in ogni fase
- La pipeline lavorerebbe meglio se le tre fasi avessero la stessa durata
  - La fase E è più lunga e può essere ulteriormente scomposta in E1 (lettura dal banco dei registri) ed E2 (operazione della ALU e scrittura nel registro)
- L'istruzione NOOP evita l'uso di apposito HW per la "normalizzazione" della pipeline

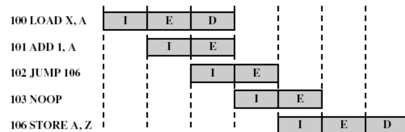
Pipeline RISC – salti ritardati

- Il salto ritardato è un modo per aumentare l'efficienza della pipeline
  - Il codice viene riorganizzato in modo da non dover inserire NOOP per la normalizzazione della pipeline
- Tale tecnica può essere applicata senza problemi per i salti incondizionati, mentre è più problematici per salti di tipo condizionato
  - È il compilatore che deve valutare quando poter applicare questa tecnica

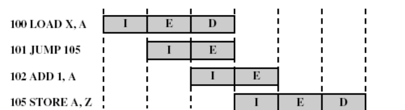
Pipeline RISC – salti ritardati



(a) Traditional Pipeline



(b) RISC Pipeline with Inserted NOOP



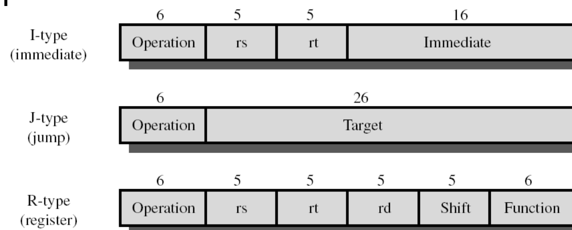
(c) Reversed Instructions

MIPS R4000

- L'architettura MIPS (sviluppata dalla MIPS Technology Incorporation) ha le seguenti caratteristiche:
  - Spazio di indirizzamento a 64 bit
  - 32 registri interni a 64 bit
  - Cache di I° livello da 128KB divisa a metà tra dati e istruzioni
  - Tutte le istruzioni sono codificate mediante un formato di una singola parola a 32 bit
  - Tutte le operazioni sui dati sono da registro a registro e gli unici riferimenti alla memoria sono semplici operazioni di lettura e scrittura

MIPS R4000

- Sono possibili tre formati di istruzione che condividono una formattazione comune dei codici operativi e dei riferimenti ai registri

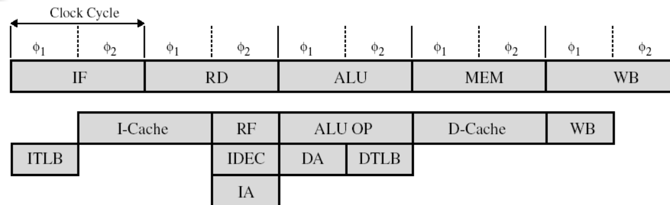


Operation	Operation code
rs	Source register specifier
rt	Source/destination register specifier
Immediate	Immediate, branch, or address displacement
Target	Jump target address
rd	Destination register specifier
Shift	Shift amount
Function	ALU/shift function specifier

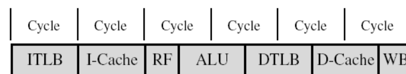
MIPS - pipeline

- L’architettura MIPS realizza la cosiddetta “superpipeline”
  - Fa uso di un numero maggiore di fasi di pipeline
- Tutte le istruzioni eseguono la stessa istruzione di cinque fasi di pipeline:
  - Caricamento dell’istruzione
  - Caricamento degli operandi sorgente dal banco dei registri
  - Operazione della ALU, oppure generazione dell’indirizzo dei dati operandi
  - Riferimento alla memoria dei dati
  - Riscrittura nel banco dei registri

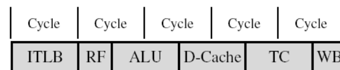
MIPS - pipeline



(a) Detailed R3000 pipeline



(b) Modified R3000 pipeline with reduced latencies



(c) Optimized R3000 pipeline with parallel TLB and cache accesses

IF = Instruction fetch
RD = Read
MEM = Memory access
WB = Write back
I-Cache = Instruction cache access
RF = Fetch operand from register
D-Cache = Data cache access
ITLB = Instruction address translation
IDEC = Instruction decode
IA = Compute instruction address
DA = Calculate data virtual address
DTLB = Data address translation
TC = Data cache tag check

IA-64/Merced

- Intel e Hewlett-Packard (HP) si sono unite per dare vita ad una nuovo architettura a 64 bit
  - Non è un’estensione dell’architettura Pentium a 32 bit
  - Non è un’estensione dell’architettura HP PA-RISC a 64 bit
- I concetti fondamentali di questa nuova architettura sono:
  - Parallelismo a livello di istruzione
  - Parole di istruzioni lunghe oppure molto lunghe
  - Predicazione dei salti (da non confondere con predizione dei salti)
  - Caricamento speculativo del codice

IA-64/Merced

- L'obiettivo principale dei progettisti era quello di utilizzare in modo efficiente l'elevatissimo numero di transistor che la tecnologia consente di mettere su un chip
  - Tanti transistor → tante unità che possono lavorare in parallelo
  - In un'architettura superscalare tradizionale ci vuole logica aggiuntiva (complessità HW) per gestire e coordinare le varie unità

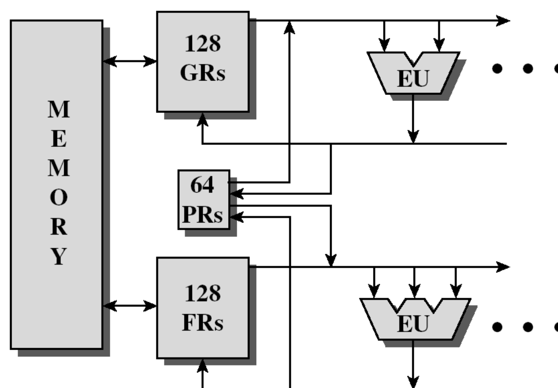
IA-64/Merced

- Il punto principale dell'approccio IA-64 è il concetto di parallelismo esplicito con il quale il compilatore organizza staticamente le istruzioni in fase di compilazione, invece di obbligare il processore a farlo dinamicamente in fase di esecuzione:
  - Il compilatore determina quali istruzioni possono essere eseguite in parallelo e inserisce queste informazioni nell'istruzione macchina

IA-64: organizzazione

- IA-64 prevede l'utilizzo di:
  - 256 registri: 128 per l'uso intero, logico e general-purpose e 128 per l'uso in virgola mobile
  - 64 registri da un bit per l'esecuzione predicata
  - Unità di esecuzione multiple (superscalarità). Il numero di unità di esecuzione è funzione del numero di transistor disponibili in una particolare implementazione

IA-64: organizzazione



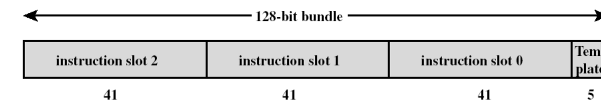
IA-64: formato delle istruzioni

- IA-64 definisce un gruppo di 128 bit che contiene:
  - Tre istruzioni
  - Un campo template
- Il processore può caricare le istruzioni un gruppo alla volta
- Il campo template contiene informazioni su quali istruzioni possono essere eseguite in parallelo e la sua interpretazione non è confinata ad un solo gruppo
  - Il processore può osservare gruppi multipli per determinare quali istruzioni possono essere eseguite contemporaneamente

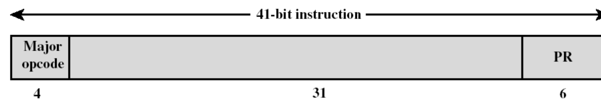
IA-64: formato delle istruzioni

- Ogni istruzione ha un formato di lunghezza fissa pari a 41 bit (più lunga della lunghezza delle istruzioni delle macchine RISC e RISC superscalari)
  - IA-64 fa uso di più registri rispetto ad una macchina RISC classica
  - Devono essere gestiti i 64 registri di predicato

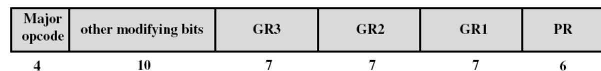
IA-64: formato delle istruzioni



(a) IA-64 bundle



(b) General IA-64 instruction format



(c) Typical IA-64 instruction format

PR = Predicate register
GR = General or floating-point register

IA-64: esecuzione predicata

- La predicazione è una tecnica con la quale il compilatore determina quali istruzioni può eseguire in parallelo ed elimina dai programmi i salti grazie all'esecuzione condizionata
- Si supponga di analizzare il classico statement if-then-else
- Un compilatore tradizionale:
  - Inserisce un salto condizionato nel punto if del costrutto
  - Se la condizione ha un certo valore il salto non viene intrapreso e viene eseguito il "ramo then"

IA-64: esecuzione predicata

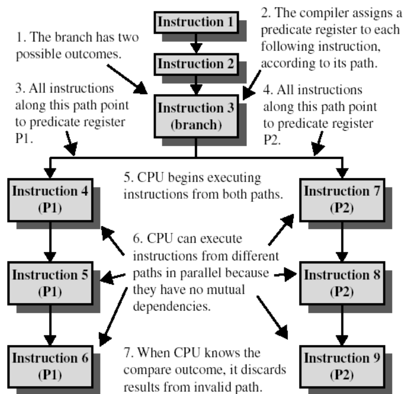
- Alla fine del ramo then vi è un salto incondizionato al allo statement successivo all'if-then-else
- Se la condizione è falsa viene eseguito il salto condizionato al "ramo else"

- Un compilatore IA-64 esegue invece le seguenti azioni:
  - Nel punto if inserisce un'istruzione di confronto che genera due predicati: se il confronto determina un risultato vero, il primo predicato è impostato a vero e il secondo a falso e viceversa

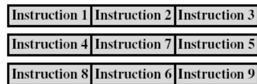
IA-64: esecuzione predicata

- Aumenta ogni istruzione del cammino then con un riferimento ad un registro di predicato che contiene il valore del primo predicato ed aumenta ogni istruzione del cammino else con un riferimento al registro di predicato che contiene il valore del secondo predicato
- Il processore esegue le istruzioni lungo i due percorsi e quando si conosce il risultato del confronto scarta i risultati di un cammino

IA-64: esecuzione predicata



The compiler might rearrange instructions in this order, pairing instructions 4 and 7, 5 and 8, and 6 and 9 for parallel execution.



(a) Predication

Esempio

- Si consideri il seguente codice sorgente:

```
if (a && b)
   j = j + 1;
else
   if (c)
        k = k + 1;
   else
        k = k – 1;
i = i + 1;
```

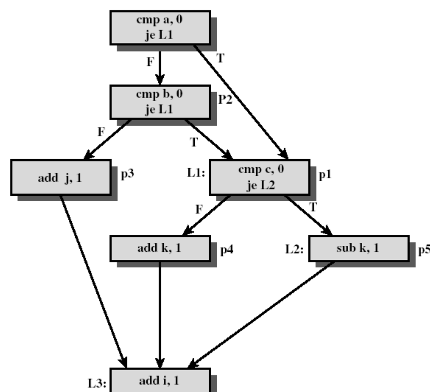
Esempio

- L’esempio recedente può essere tradotto da un compilatore tradizionale in:

```
      beq a, 0, L1
      beq b, 0, L1
      add j, j, 1
      jump L3
L1:   beq c, 0, L2
      add k, k, 1
      jump L3
L2:   sub k, k, 1
L3:   add i, i, 1
```

Esempio

- Il compilatore può assegnare un predicato ad ogni blocco che viene eseguito in modo condizionato



Esempio

- Assumendo che tutti i predicati siano stati inizializzati a falso e che esistano questi tre nuovi costrutti:
  - <PI> istruzione. L’istruzione deve essere eseguita solo se il predicato PI è vero
  - PJ, PK = cmp (relazione). Imposta il valore del predicato PJ a TRUE e del predicato PK a FALSE se la relazione è vera e viceversa
  - <PI> PJ, PK = cmp (relazione). Un’istruzione di generazione di predicato può essa stessa essere predicata

  il codice che ne deriva è il seguente:

Esempio

```
(1) P1, P2 = cmp (a == 0)
(2) <P2> P1, P3 = cmp (b == 0)
(3) <P3> add j, j, 1
(4) <P1> P4, P5 = cmp (C != 0)
(5) <P4> add k, k, 1
(6) <P5> sub k, k, 1
(7) add i, i, 1
```

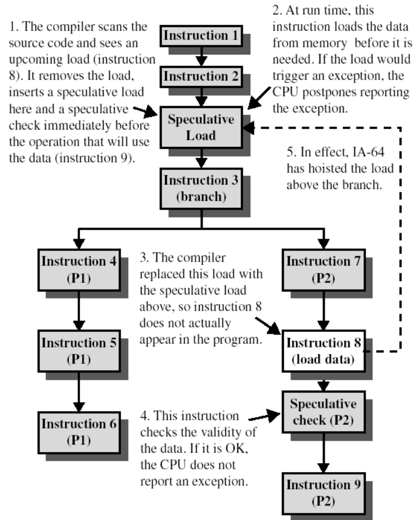
Esempio

- Una sola tra le istruzioni (3), (5) e (6) sarà eseguita
  - In un processore superscalare classico si usa la predizione dei salti per indovinare quale cammino sarà scelto. Se la previsione è sbagliata la pipeline deve essere scaricata
  - Un processore IA-64 può avviare l'esecuzione parallela di tutte e tre le istruzioni e, una volta noti i risultati dei registri di predicato, inviare solo i risultati dell'istruzione valida

Caricamento speculativo

- Il caricamento speculativo consente al processore di caricare i dati dalla memoria prima che ne abbia bisogno
- Un'istruzione di caricamento originale del programma viene sostituita da due istruzioni:
  - Un caricamento speculativo (ld.s) esegue il prelievo dalla memoria e realizza l'individuazione dell'eccezione (page fault). L'istruzione ld.s è posta in un punto appropriato più indietro nel programma
  - Un'istruzione di controllo (chk.s) prende il posto dell'originale istruzione di caricamento e può essere predicata in modo da essere eseguita solo se il predicato si è verificato

Caricamento speculativo



(b) Speculative loading