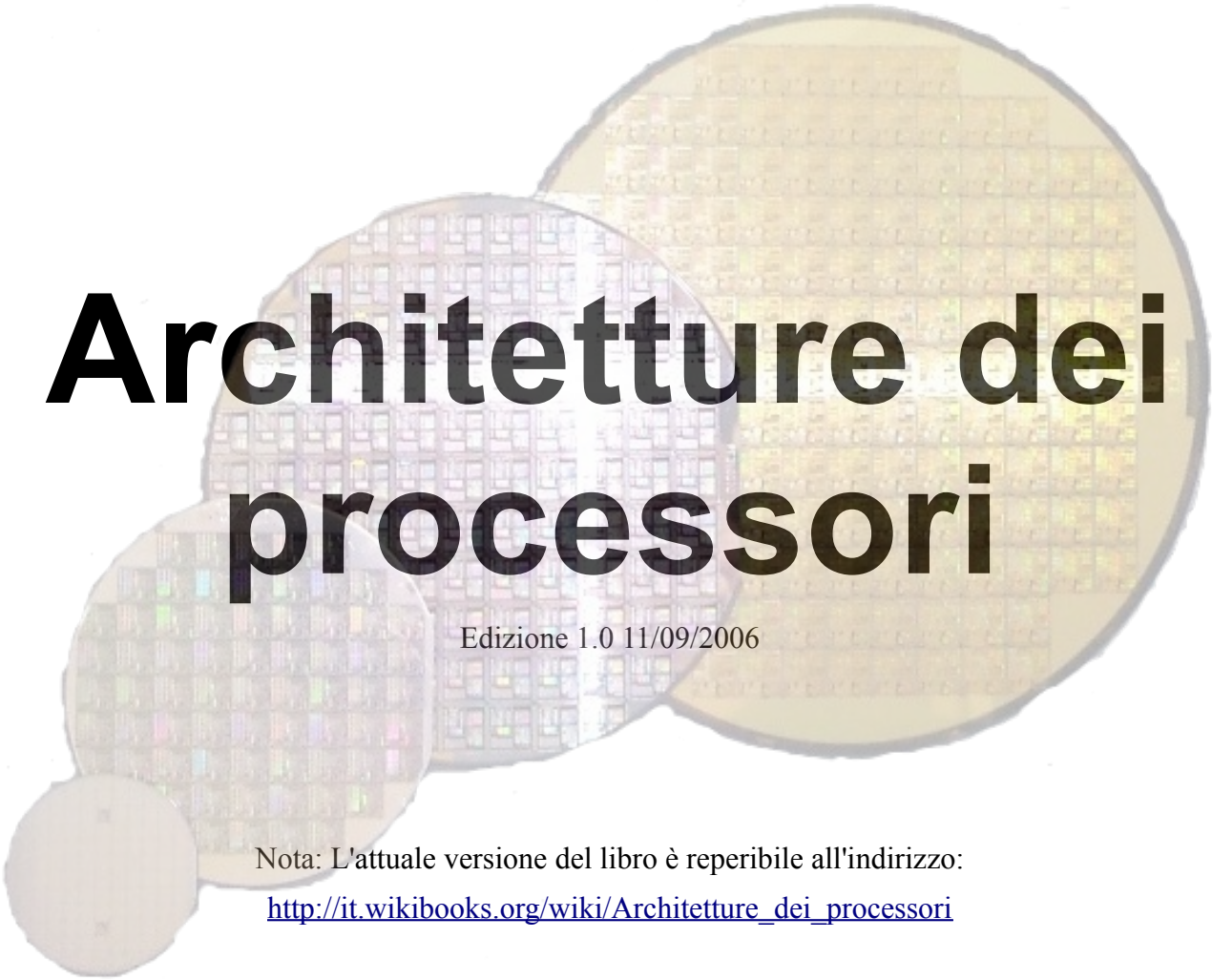


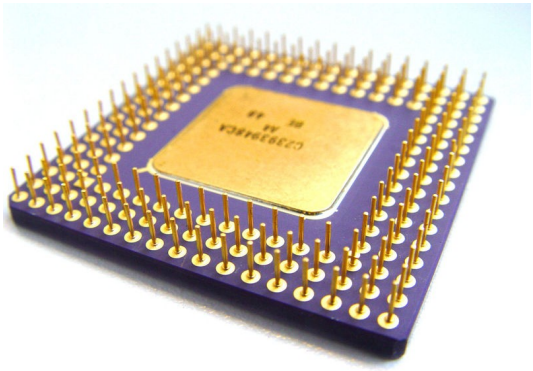
Architetture dei processori

Edizione 1.0 11/09/2006

Nota: L'attuale versione del libro è reperibile all'indirizzo:
http://it.wikibooks.org/wiki/Architetture_dei_processori

Indice dei contenuti

Introduzione



Queto è un libro prodotto dal sito it.wikibooks.org con il lavoro collaborativo degli utenti del sito. La versione aggiornata del libro è disponibile sul sito dove è anche possibile visionare l'elenco completo degli autori accedendo alla cronologia delle singole sezioni. Questo libro tratterà le architetture interne dei microprocessori elencandone le principali unità funzionali con annessi vantaggi e svantaggi. Il libro non affronterà tutte le tipologie possibili essendo molte architetture sviluppate nel corso degli anni, maggior parte delle quali senza successo commerciale o riservate ad applicazioni specializzate. Il libro affronterà le architetture dal punto di vista teorico non concentrandosi sui singoli aspetti implementativi che rendono l'architettura X86 diversa da quella PowerPC per esempio.

Definizione

Un processore è un singolo circuito integrato in grado di effettuare operazioni decisionali, di calcolo o di elaborazione dell'informazione; il microprocessore principale di un computer viene chiamato processore o CPU; il microprocessore che si occupa delle operazione legate alla visualizzazione delle informazioni in un computer viene chiamato GPU o VPU.

L'Intel 4004 il primo processore

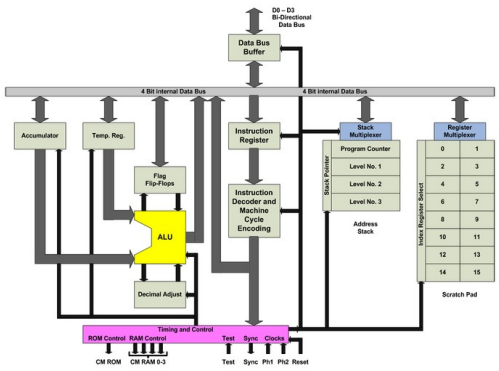
I processori sono circuiti contenenti da migliaia a milioni di transistor ed altri componenti elettronici, ottenuti sfruttando le caratteristiche di semiconduttività del silicio e la sua relativa facilità di essere convertito in isolante tramite drogaggio. Questi transistor conservano informazioni sotto forma di carica elettrica, variando il livello a seconda della logica usata nel funzionamento del circuito.

All'inizio i processori venivano progettati seguendo la classica architettura di von Neumann. Secondo questa architettura la memoria del computer era vista come un nastro infinito e il processore era una testina che leggeva sequenzialmente i dati sul nastro, li elaborava e si spostava sul nastro di conseguenza.

L'architettura di von Neumann risultava inefficiente nella gestione di più flussi di dati essendo il flusso delle operazioni e dei dati mischiati. Per superarne i limiti venne sviluppata l'Architettura Harvard. Questa architettura prevede che il flusso dati e il flusso delle istruzioni viaggino su due canali separati all'interno del processore in modo da non disturbarsi a vicenda. Praticamente tutti i moderni processori sono basati su questa architettura dati che la separazione dei dati e delle istruzioni permette agli algoritmi che gestiscono le cache dei processori di funzionare al meglio. L'ultimo processore ad elevate prestazioni basato su architettura di von Newmann è stato l'Intel 80486, mentre gli altri concorrenti avevano abbandonato questa architettura già da alcune generazioni. Intel prosegui così a lungo con questa architettura per via della difficoltà di ottenere processori basati su architettura Harvard compatibili a livello binario con il software scritto per l'architettura x86.

Storia

Il primo microprocessore mai realizzato fu l'Intel 4004, che lavorava con parole di soli 4 bit: fu progettato dal vicentino Federico Faggin e i primi prototipi videro la luce nel gennaio del 1971.



Schema interno dell'architettura del 4004

Fu un successo limitato, ma i successivi Intel 8008 e 8080, che invece usavano parole di 8 bit (un byte, finalmente) riscossero molto più interesse. Nel 1974 Faggin, che nel frattempo alla Intel era divenuto il responsabile di tutti i progetti di circuiti integrati MOS tranne le RAM dinamiche, lascia la Intel per fondare la Zilog, che nel 1976 lancia il mitico processore Z80, che sarà il cuore di molti home computer del decennio successivo. Il successo dello Z80 è immediato, ed eclissa in pratica la serie 8080 della Intel nonostante il lancio dei nuovi 8080A e 8085: nel 2000, a quasi 25 anni dal debutto, lo Z80 veniva ancora prodotto in grandi volumi e utilizzato come microcontroller per sistemi embedded.

L'apparire dei microprocessori provocò una rivoluzione nel mercato degli elaboratori elettronici; insieme con la tecnologia delle memorie RAM a stato solido, i microprocessori resero possibile costruire e vendere un computer completo ad una frazione del prezzo dei minicomputer, che fino ad allora erano le macchine più economiche disponibili: il prezzo di queste nuove macchine era tanto basso da essere alla portata anche dei privati e degli hobbysti. Poiché erano costruiti attorno a dei microprocessori, questi nuovi elaboratori vennero chiamati microcomputer.

Successivamente la Motorola e altri concorrenti entrarono nel mercato, sviluppando altri tipi di microprocessori, come il 6502 usato nell'Apple II e la versione custom 6510 prodotta appositamente per il Commodore 64; il 6800 e il 6809, tutti con registri a 8 bit.

Dal 1980 in poi i fabbricanti e i modelli prodotti iniziarono a moltiplicarsi: ricordiamo soltanto il Motorola 68000, lo Zilog Z8000 e l'Intel 8086, tutti e tre a 16 bit anziché a 8, capostipiti di tre numerose e longeve famiglie di microprocessori. IBM scelse l'Intel 8088 per il suo primo PC al posto dell'8086 perché era compatibile con tutto il software per l'8086, ma aveva un bus dati esterno a 8 bit invece che a 16 ed era compatibile a livello hardware con tutti i circuiti esistenti sviluppati per l'8085. La Apple invece si basò sui chip della famiglia Motorola 68000 per i suoi prodotti della serie Macintosh.

Con il corso degli anni i processori subirono un notevole incremento passando a parole di 32 bit (Motorola 68040, Intel 80486) e infine ad strutture a 64 bit (PowerPC 970 e Athlon 64). Le frequenze di funzionamento si innalzarono notevolmente passando da pochi Megaherz dei primi processori a 16 bit a diversi GigaHerz degli ultimi microprocessori. Nel ventennio intercorso tra gli anni 80 e il 2000 molte società grandi e piccole svilupparono propri microprocessori, alcuni dei quali basati su architetture peculiari ma la maggior parte delle quali negli ultimi anni ha deciso di abbandonare il settore.

Attualmente le architetture più diffuse sono l'architettura X86 sviluppata da Intel e nel corso degli anni espansa da AMD e altri contendente che è alla base di quasi tutti i personal computer attualmente prodotti. In seconda posizione abbiamo l'architettura Power prodotta da IBM che con varie incarnazioni viene utilizzata dalle consolle, da macchine per applicazioni specializzati e dai più potenti computer del pianeta. Un'altra architettura molto importante è l'architettura ARM che viene utilizzata in telefoni, palmari, e macchine dedicate. Altre architetture minoritarie sono le architetture MIPS e SPARC.

Set di istruzioni

Una prima classificazione dei microprocessori si può fare a seconda dei tipo di istruzioni implementate, queste possono essere CISC o RISC.

CISC

Motorola 68000 uno dei meglio riusciti processori CISC

Un architettura CISC (Complex Instruction Set Computer) è dotata di un set di istruzioni che consentono di eseguire operazioni anche complesse. I primi microprocessori erano di questa tipologia e fino agli anni 80 questo è stato il paradigma imperante. I microprocessori con questa architettura erano pensati per essere programmati in assembler e quindi il progettista metteva a disposizione del programmatore istruzioni anche molto complesse che sfruttavano le peculiarità della macchina in modo da generare programmi veloci e che occupassero poca memoria. Queste istruzioni potevano nascondere alcune caratteristiche del processsore per rendere la programmazione assembler più semplice. Per esempio potevano contenere istruzioni che formalmente modificavano i dati in memoria senza passare dai registri. In realtà queste istruzioni caricavano il dato in un registro nascosto, modificavano il dato e poi lo risalvavano in memoria. In sostanza creavano un illusoreo teatro per il programmatore. I processori CISC spesso sono dotati di un set di istruzioni definito con codici di lunghezza variabile e sono dotato di un numero ridotto di registri dato. I codici di lunghezza variabile permettono di sfruttare al pieno la memoria disponibile mentre i pochi registri interni derivano dai pochi transistor integrabili negli anni 80 e dal fatto che avendo istruzioni interne molto complesse spesso non sono necessari molti registri per eseguire i programmi.

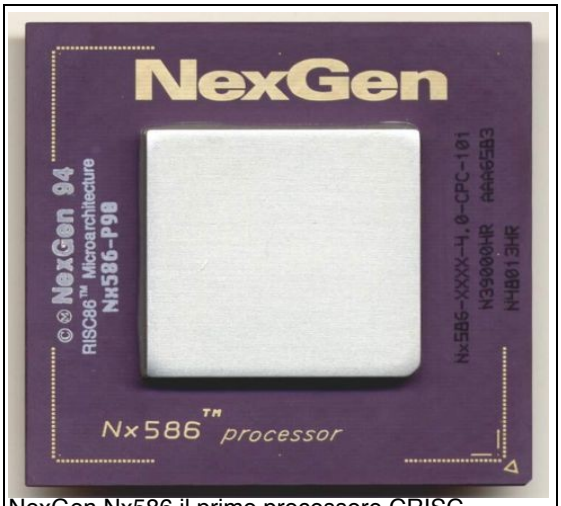
RISC

Processore MIPS prodotto da Toshiba basato su architettura RISC

I microprocessori RISC (Reduced Instruction Set Computer) invece sono basati sulla presenza di un numero ridotto di istruzioni rispetto a un microprocessore CISC e di un ridotto numero di modi di indirizzamento. Questo permetteva di realizzare architetture semplici e con pochi transistor, i transistor rimanenti potevano essere utilizzati per realizzare molti registri o per implementare strutture come le pipeline (vedi oltre). Nei microprocessori RISC quasi tutte (se non tutte) le istruzioni erano eseguite in numero costante di cicli di clock (usualmente 1 o 2). Questo a differenza dei microprocessori CISC dove alcune operazioni erano eseguite in modo rapido (1 o 2 cicli di clock) mentre altre richiedevano decine di cicli di clock (usualmente le divisioni). I processori RISC non erano nati per essere programmati in assembler dato che non implementavano semplificazioni per il programmatore che invece i processori CISC avevano. Questa tipologia di processori era nata per essere programmata con linguaggi ad alto livello che poi opportuni compilatori avrebbero tradotto in linguaggio macchina. I processori RISC normalmente hanno un set di istruzioni di lunghezza fissa per facilitare il caricamento di pezzi di codice ad alta velocità in modo semplice e sono quasi sempre dotati di molti registri (spesso centinaia) dato che per eseguire un programma spesso sono necessarie molte

operazioni elementari e quindi molti registri ove salvare i dati intermedi.

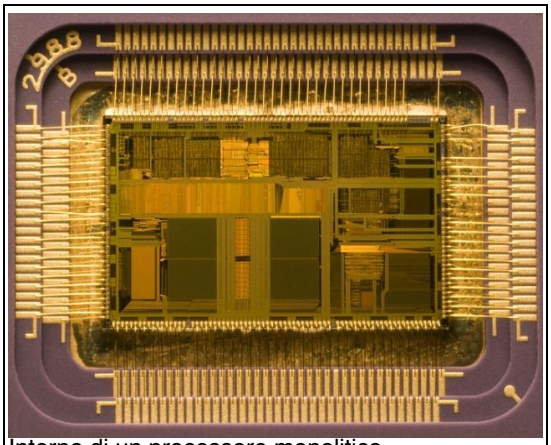
CRISC



NexGen Nx586 il primo processore CRISC compatibile con l'architettura X86

Negli ultimi anni tutti i microprocessori sono di tipologia RISC internamente. Anche i microprocessori che in teoria sono CISC (architettura X86 per esempio) in realtà traducono le istruzioni internamente in istruzioni RISC. In questi casi a volte si parla di architettura CRISC per sottolineare l'architettura ibrida. I processori RISC si sono diffusi sul mercato dato che oramai solo una ridottissima cerchia di programmatori utilizza l'assembler dato che i linguaggi ad alto livello permettono di scrivere programmi in minor tempo. Le istruzioni complesse dei CISC e i particolari modi di indirizzamento e i vari ammennicoli che venivano inseriti per i programmatori assembler rendono invece i programmi compilati più lenti e in generale deprimono le prestazioni generali quindi in generale i processori CISC risultano meno performanti a parità di transistor utilizzati di un equivalente RISC. Questa architettura quindi cerca di sfruttare le prestazioni dei processori RISC pur mantenendo la compatibilità con i passati programmi scritti con set di istruzioni CISC. Questo approccio ha lo svantaggio di rendere il processore molto più complesso di un equivalente processore RISC e meno performante. Difatti i compilatori per architetture CISC compilano il codice tenendo conto di alcune considerazioni tecniche che con questi processori non sono vere. Inoltre questi processori per effettuare la conversione da CISC a RISC implementano alcune tecniche (come la ridenominazione dei registri) che incrementano la complessità dei processori e comunque non fornisce le prestazioni di un RISC nativo. Questa architettura quindi va intesa come un *trucco* utilizzato per aumentare le prestazioni senza compromettere la compatibilità con il passato e non come una scelta tecnica primaria. Se non si hanno problemi di compatibilità soluzioni RISC o VLIW sono sempre migliori.

Processore monolitico



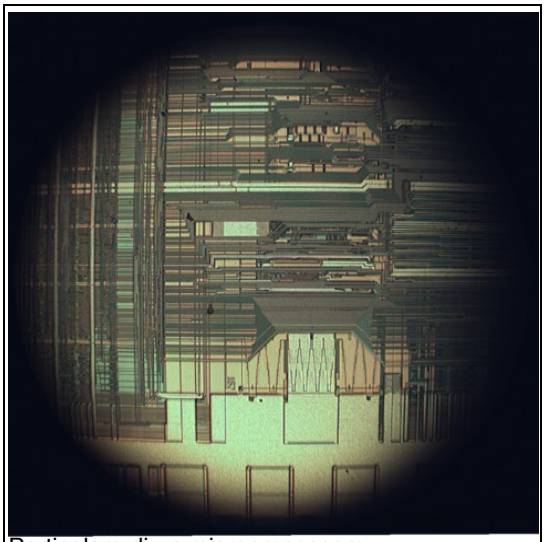
Interno di un processore monolitico
(Intel 80486 dx2)

I primi microprocessori erano costruiti interno all'unità di calcolo (chiamata ALU). Le istruzioni venivano caricate dalla memoria, decodificare, mandate all'ALU che le elaborava e se era necessario richiedeva il caricamento degli eventuali operandi da processare. Una volta elaborata l'istruzione il risultato veniva salvato in un registro o in memoria. L'esecuzione delle istruzioni poteva essere visto come un flusso di dati che scorreva lungo le varie unità, per eseguire alcune operazioni potevano essere necessarie anche decine di cicli di clock e mentre un'istruzione era in esecuzione tutte le unità interne erano assegnate all'istruzione anche se questa magari non utilizzava questa unità interna. Per incrementare le prestazioni dei processori si puntò ad innalzare le frequenze di funzionamento, ed a rendere le unità di calcolo più efficienti quindi si iniziò ad introdurre più componenti interni che lavoravano in parallelo in modo da eseguire alcune operazioni di una nuova istruzione in parallelo con l'esecuzione dell'istruzione corrente. Studi su come rendere più efficienti ie veloci processori portarono allo sviluppo di strutture come le pipeline le cache, le unità FPU, le unità SIMD, ecc.

- Unità di decodifica
    - Unità predizione dei salti
- Arithmetic Logic Unit
- Floating Point Unit
- Cache
- Pipeline
- Memory Management Unit
- Single Istruction Multiple Data

Unità di decodifica

L'unità di decodifica si occupa di ricevere le istruzioni in ingresso e di attivare le opportune unità interne del processore per eseguire l'istruzione caricata. Nei primi processori questa unità provvedeva a tradurre l'istruzione in opportuno microcodice che veniva utilizzato per gestire l'ALU i registri l'accumulatore e le altre unità funzionali dei processori. Con l'avvento delle unità dotate di pipeline il microcodice è stato sostituito da segnali di controllo che temporizzano le varie unità.


Particolare di un microprocessore

Alcuni microprocessori comunque utilizzano ancora parzialmente il microcodice per codificare alcune istruzioni che vengono considerate poco utilizzate ma che devono essere mantenute per ragione di compatibilità. Questa strategia permette di risparmiare transistor ma ovviamente produce un degrado delle prestazioni.

Molti microprocessori moderni sono progettati per eseguire set di istruzioni molto complessi, sviluppati per processori CISC. Questo set di istruzioni renderebbe difficile l'esecuzione di più istruzioni contemporaneamente. Quindi molti microprocessori traducono le istruzioni complesse in sequenze di istruzioni più semplici da eseguire rendendo il set di istruzioni simile a quello di un'architettura RISC. Anche questo lavoro viene svolto dall'unità di decodifica.

Nei microprocessori superscalari l'unità di decodifica diventa una componente critica del processore dato che l'unità si occupa di riceve le unità e di organizzarle in modo da ridurre al minimo possibile gli stalli delle pipeline. Questo viene ottenuto eseguendo le istruzioni fuori ordine, ridenominando i registri, prevedendo le condizioni di salto, precaricando e dati ed implementando l'esecuzione speculativa. Non tutti i processori implementato tutte queste tecniche dato che alcune sono complementari ed in alcuni casi l'implementazione di queste tecniche sarebbe molto onerosa e porterebbe modesti miglioramenti delle prestazioni.

Esecuzione fuori ordine

L'esecuzione fuori ordine indica la capacità di molti processori di eseguire le singole istruzioni senza rispettare necessariamente l'ordine imposto dal programmatore. Il processore in sostanza analizza il codice che dovrà eseguire, individua le istruzioni che non sono vincolate da altre istruzioni e le esegue in parallelo con il codice principale. La rilevazione delle dipendenza tra le varie istruzioni è un compito complesso dato che le istruzioni possono essere vincolare in vari modo. Per esempio se ho un'istruzione che esegue una somma tra due registri e poi il risultato della somma viene utilizzato come operando per una divisione va da se che la divisione e la somma non possono essere eseguite in parallelo senza generale conflitti. Il problema può essere risolto replicando delle unità funzionali (vedi sezione sulla pipeline) oppure eseguendo in parallelo all'istruzione di somma un'istruzione che non ha dipendenze e rimandando la divisione. In generali queste unità funzionali decodificano le istruzioni e pongono le istruzioni che non hanno vincoli (o che hanno vincoli che sono stati risolti) entro un buffer che alimenta le pipeline. Dopo le pipeline vi è un'unita che utilizzando delle etichette collegate alle istruzioni ricostruisce l'ordine cronologico delle istruzioni facendo uscire dal processore i risultati delle istruzioni con l'ordine cronologico impostato dal programma. Questa unità inoltre provvede ad eliminare le eventuali operazioni eseguite erroneamente dal processore. La presenza di unità di predizione dei salti implica che il processore spesso esegue delle istruzioni presupponendo che il processore esegua (o non esegua) un

certo salto. Ma, se la previsione fornita dall'unità di predizione dei salti si dovesse dimostrare non corretta le istruzioni eseguite erroneamente vanno eliminate per preservare il corretto funzionamento del programma.

Ridenominazione dei registri

Nei processori che implementano l'esecuzione fuori ordine non vi è una reale corrispondenza tra ciò che il programmatore si aspetta di trovare nei registri e ciò che realmente si trova nei registri dato che l'ordine cronologico del programma non è rispettato. Quando un'istruzione di trova ad accedere ad un registro che non contiene un dato corretto per il programma la pipeline dovrebbe andare in stallo fino a quando il registro non contenga il dato corretto o fino a quando non sia libero. Per evitare questo spreco di cicli di calcolo i processori implementano la ridenominazione dei registri. In sostanza il processore dispone di molti registri fisici nascosti che vengono assegnati alle istruzioni quando servono dei registri che risultano già occupati. Quindi in un dato istante per esempio un ipotetico registro R0 potrebbe essere in realtà assegnato a tre registri fisici che *fingono* di essere il registro R0 per le istruzioni che li usano. La corrispondenza tra questi registri fisici e i registri visti dal programma viene mantenuta da una tabella che viene aggiornata ad ogni decodifica di un'istruzione.

Predizione dei salti

Un processore basato su pipeline viene influenzato molto negativamente dalla presenza di salti condizionati dato che questi possono costringere il processore a svuotare la pipeline. Mediamente ogni 6 istruzioni un processore basato su architettura x86 incontra un salto condizionato e quindi i processori implementano delle unità molto complesse per cercare di prevedere se il salto condizionato verrà eseguito in modo da caricare in anticipo il blocco di istruzioni corretto nella pipeline. Vista l'importanza di questa unità si rimanda al capitolo apposito per spiegazioni più approfondite.

Precaricamento dei dati

I processori sono diventati sempre più veloci e nel giro di un decennio sono passati da frequenze di poche decine di Megahertz a frequenze di funzionamento dell'ordine dei Gigahertz. Le memorie invece sono non sono diventate molto più veloci di quanto fossero dieci anni fa, aumentando invece la quantità di dati trasmessi per secondo. I processori per attutire il problema hanno implementato cache di primo, secondo e a volte anche di terzo livello. La presenza delle cache con gli ultimi processori non è più sufficiente per evitare un eccessivo deperimento delle prestazioni e quindi alcuni processori implementano delle unità che analizzano il codice cercando di prevedere in anticipo quali dati o quali istruzioni serviranno al processore e provvedendo al loro caricamento in cache (o direttamente nel processore) prima della loro reale necessita. Alcune architetture prevedono delle istruzioni apposite per indicare quali blocchi precaricare ma la maggior parte delle architetture non sono dotate di queste istruzioni e quindi devono basarsi solamente sull'ispezione del codice in tempo reale. Il precaricamento dei dati e delle istruzioni introduce dei problemi, per esempio se il microprocessore carica delle istruzioni dipendenti da un salto e quel salto non viene eseguito il processore deve provvedere ad eliminare le istruzioni caricate erroneamente prima di caricare le istruzioni da eseguire. Problemi anche maggiori si hanno nel caso di generazione di un'eccezione, per esempio l'accesso a una locazione con consentita genera una eccezione che va segnalata al sistema operativo, ma la segnalazione va effettuata quando dovrebbe effettivamente avere luogo e non prima per via del precaricamento. Il precaricamento dei dati invece deve garantire la coerenza e la validità dei dati quindi se un dato viene precaricato nessuna istruzione deve modificarlo prima del suo effettivo utilizzo da parte delle istruzioni che hanno forzato il precaricamento. Questi problemi rendono il precaricamento dei dati e delle istruzioni molto

complesso da implementare in hardware senza un supporto diretto del set di istruzioni. Invece se il set di istruzioni supporta nativamente questa caratteristica la sua gestione diventa molto più semplice.

Esecuzione predicativa

L'esecuzione predicativa è un approccio che mira a ridurre la dipendenza del processore dall'unità di predizione delle diramazioni. I microprocessori moderni sono dotati di molte unità funzionali in grado di eseguire operazioni in parallelo ma queste unità sono quasi sempre vuote. Processori anche molto complessi come il Pentium 4 pur potendo in teoria eseguire fino a 6 operazioni in contemporanea in realtà per la maggior parte del tempo eseguono una o due operazioni in parallelo. Partendo da questa constatazione l'approccio predicativo punta a riempire al massimo le unità di elaborazioni eliminando le istruzioni eseguite per errore. Un processore dotato di esecuzione predicativa oltre a leggere le istruzione legge dei predicati associate a esse, questi predicati indicano al processore in caso di salto condizionato quale ramo risulta effettivamente valido. In sostanza in caso di salto condizionato il processore associa alla condizione di salto un predicato (per esempio il predicato P0). Al blocco di istruzioni da eseguire se il salto viene eseguito viene associato il predicato P0=0 mentre al blocco di istruzioni da eseguire nel caso il salto non venga eseguito viene associato il predicato P0=1. Poi il processore esegue in parallelo l'istruzione di salto in un'unità funzionale, le istruzioni del primo blocco in una seconda unità funzionale e l'istruzione del secondo blocco in una terza unità funzionale. Quando l'istruzione di salto viene calcolata si vede se il predicato vale 0 oppure 1 e quindi si può stabilire quale blocco di istruzioni sia valido. Il blocco con le istruzioni non valide vengono individuate tramite i predicati loro associati ed eliminate. Questa filosofia di sviluppo quindi preferisce eseguire anche istruzioni inutili pur di mantenere sempre piene le pipeline. Questa metodologia di esecuzione fornisce buone prestazioni se le pipeline non sono formate da troppi stadi. Per esempio un processore come il Penium 4 è dotato di pipeline da 31 stadi e quindi un'esecuzione predicativa potrebbe portare ad eseguire fino a 31 operazioni inutili prima del completamento del salto e quindi dell'individuazione delle istruzioni eseguite erroneamente. Il processore Itanium 2 che implementa questa modalità di esecuzione difatti utilizza pipeline corte, a 10 stadi. L'esecuzione predicativa se non è gestita dal set di istruzioni del processore è molto costosa da implementare. Il processore dovrebbe associare in tempo reale alle istruzioni i vari predicate e tenerne traccia durante l'esecuzione. É da notare che non tutti i salti possono convenientemente essere risolti con i predicati, a volte conviene basarsi sull'unità di predizione dei salti. Quale delle due alternative scegliere dipende dal codice e solo un'analisi approfondita del codice permette di individuare l'alternativa migliore ma un processore dovendo scegliere in tempo reale quale alternativa scegliere dovrebbe far affidamento su delle euristiche con non sempre danno il risultato ottimo. Invece un processore dotato nativamente di questa modalità di esecuzione deve semplicemente caricare i predicati e regolare correttamente i registri appositi dato che tutta la fase di analisi del codice è stata svolta precedentemente dal compilatore che non avendo problemi di tempo può individuare l'alternativa migliore.

Ovviamente tutte queste infrastrutture aggiuntive rendono le unità di decodifica molto complesse e queste occupano buona parte dei transistor dei moderni processori. Per ridurre il problema si sono sviluppate architetture come le architetture VLIW e derivate che affrontano il problema alla radice con un paradigma diverso eliminando alcune unità funzionali e semplificandone altre.

Unità di predizione dei salti

Pentium IV, processore utilizzante la predizione della linea successiva

Esistono molte tecniche per implementare la predizione dei salta. Tecniche più complesse portano ad ottenere percentuali di previsione migliori ma comportano anche costi maggiori per via del maggior numero di transistor impiegato e quindi spesso non viene utilizzata la migliore strategia di previsione ma strategie più semplici e quindi più economiche da implementare.

- Predizione elementare

I primi esemplari di SPARC e MIPS (due delle prime architetture RISC commerciali) facevano una specie di predizione, molto elementare: non consideravano mai il salto accettato, e decodificavano l'istruzione seguente. L'operazione di salto veniva effettutata solo dopo che la condizione veniva valutata.

Entrambe le CPU effettuavano questa "predizione" in fase di decodifica e dedicavano al fetch delle istruzioni un solo ciclo di clock. In questo modo per effettuare un salto servivano due cicli di clock, ma dopo il primo la CPU aveva già effettuato il fetch dell'istruzione subito successiva al salto; piuttosto che sprecare questo lavoro, entrambi i microprocessori eseguivano anche queste istruzioni, avvantaggiandosi magari per fasi successive del lavoro.

- Predizione statica

I processori che impiegano questa tecnica considerano sempre i salti verso la parte precedente del codice come "accettati" (ipotizzando che siano le istruzioni riguardanti un ciclo) e i salti in avanti sempre come "non accettati" (ipotizzando che siano uscite precoci dal ciclo o altre funzioni di programmazione particolari). Per cicli che si ripetono molte volte, questa tecnica fallisce solo alla fine del ciclo.

La predizione statica è usata come "paracadute" quando non ci sono elementi per usare altre tecniche come la predizione dinamica e il processore deve andare "alla cieca".

Predizione della linea successiva Alcuni processori superscalari (es.: MIPS R8000 e DEC Alpha EV6/EV8 eseguivano col fetch di una linea di istruzioni, quello di un puntatore alla successiva. Questo metodo è piuttosto diverso dagli altri trattati qui perché esegue la previsione sia della scelta della diramazione che dell'obiettivo del salto.

Quando un puntatore indica un gruppo di 2, 4 o 8 istruzioni, solitamente l'istruzione ricercata non è la prima (per un fatto statistico), così la scansione delle prime istruzioni è tempo perso. Generalizzando, vengono scartate rispettivamente 0,5, 1,5 e 3,5 istruzioni decodificate. Lo stesso discorso vale per le istruzioni successive all'istruzione di salto, che devono essere scartate con identica distribuzione media.

Le istruzioni scartate dall'unità di predizione fanno guadagnare quasi un completo ciclo di fetch, anche con predizioni eseguite solo sulla linea successiva e in un solo ciclo di clock.

- Predizione bimodale

Questa tecnica sfrutta una tabella di contatori a due bit, e indicizzati con i bit meno significativi dell'indirizzo dell'istruzione cui si riferiscono (a differenza della cache per le istruzioni, questa tabella non ha tag e quindi uno stesso contatore può essere riferito a più istruzioni: ciò è definito interferenza o aliasing e porta a una perdita di precisione nella previsione). Ogni contatore può trovarsi in uno di questi quattro stati:

- Strongly not taken, "non accettato molto spesso";
- Weakly not taken, "non accettato poco spesso";
- Weakly taken, "accettato poco spesso";
- Strongly taken, "accettato molto spesso".

Ogni volta che una condizione è valutata, il contatore relativo viene aggiornato secondo il risultato, e la volta successiva viene preso come riferimento per la previsione. Un pregio di questo sistema è che i cicli vengono sempre accettati, e viene fallita solo la previsione relativa all'uscita del ciclo, mentre un sistema con contatori a bit singolo fallisce sia la prima che l'ultima istruzione.

Esempi di unità di predizione molto grandi basate su questo sistema hanno ottenuto una percentuale di successo pari al 93,5% su benchmark SPEC.

- Predizione locale

La predizione bimodale fallisce all'uscita di ogni ciclo: per cicli che si ripetono con andamento sempre simile a sè stesso si può fare molto meglio.

Con questo metodo ci si avvale di due tabelle. Una è indicizzata con i bit meno significativi dell'istruzione relativa, e tiene traccia della condizione nelle ultime n esecuzioni. L'altra è una tabella molto simile a quella usata nella predizione bimodale, ma è indicizzata sulla base della prima. Per effettuare una predizione, l'unità cerca grazie alla prima tabella la parte della seconda che tiene traccia del comportamento della condizione non in media, ma a quel punto del ciclo.

Sui benchmark SPEC, sono stati ottenuti risultati intorno al 97,1%.

Questa tecnica è più lenta perché richiede il controllo di due tabelle per effettuare ogni previsione. Una versione più veloce organizza un insieme separato di contatori bimodali per ogni istruzione cui si accede, così il secondo accesso all'insieme può procedere in parallelo con l'accesso all'istruzione. Questi insiemi non sono ridondanti, in quanto ogni contatore traccia il comportamento di una singola condizione.

- Predizione globale

Nella predizione globale si fa affidamento sul fatto che il comportamento di molte condizioni si basa su quello di condizioni vicine e valutate da poco. Si può così tenere un unico registro che tiene conto del comportamento di ogni condizione valutata da poco, e usarne i valori per indicizzare una tabella di contatori bimodali. Questo sistema è di per sè migliore della predizione bimodale solo per grandi tabelle, e non è migliore della predizione locale in nessun caso.

Se invece si indicizzano i contatori bimodali con la storia recente delle condizioni concatenata ad alcuni bit dell'indirizzo delle istruzioni si ottiene un previsore gselect, che supera la previsione locale in tabelle piccole e viene staccato di poco in tabelle maggiori di un KB.

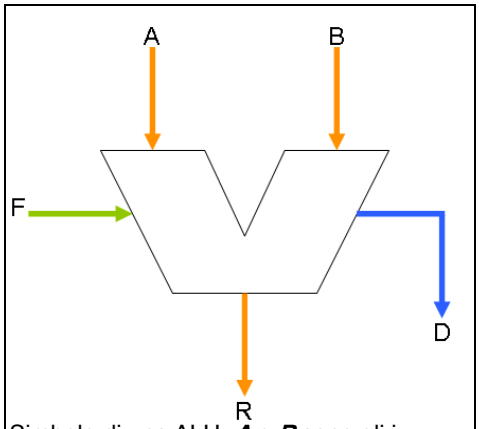
Si ottiene un metodo ancora migliore per le tabelle più grandi di 256 B, detto gshare, sostituendo nel gselect la concatenazione con l'operazione logica XOR.

Quest'ultimo metodo ottiene nei benchmark un'efficienza del 96,6%, di poco inferiore alla predizione locale.

Le predizioni globali sono più facili da rendere più veloci della predizione locale in quanto richiedono in controllo di una sola tabella per ogni previsione.

Arithmetic Logic Unit

L'Arithmetic Logic Unit (ALU) è un'unità dedita allo svolgimento di operazioni matematiche ed è formata da un insieme di porte logiche opportunamente collegate. Queste porte logiche nel loro complesso provvedono ad eseguire tutte le operazioni aritmetiche e logiche gestite dal microprocessore.



Simbolo di una ALU. ***A*** e ***B*** sono gli i segnali con gli addendi, ***R*** sono i segnali con il risultato, ***F*** sono i segnali di controllo dell'ALU e ***D*** è il segnale con l'eventuale riporto/resto

Le prime ALU erano in grado di eseguire nativamente solo le operazioni più semplici (addizione, sottrazione e shifting di bit ecc.) e le operazioni logiche booleane (AND, OR, XOR e NOT). Le operazioni più complesse come le operazioni di moltiplicazioni o divisione venivano emulate utilizzato ripetutamente somme o sottrazioni. Con l'evolvere dell'elettronica si è riuscito a integrare nelle ALU anche le operazioni di divisione e moltiplicazione. Le ALU non sono in grado di svolgere tutte le operazioni supportate dai microprocessori moderni, infatti le operazioni in virgola mobile o le operazioni multimediali (tipo AltiVec o MMX) sono svolte da unità specializzate che non risiedono nell'ALU. Le ALU utilizzate dai primi processori erano unità composte da sottounità base. Queste sottounità venivano utilizzate congiuntamente per svolgere le operazioni aritmetiche ma essendo ogni unità in grado di svolgere una parte delle operazioni aritmetica servivano più cicli di clock per completare un'operazione. La richiesta di prestazioni migliori portò alla realizzazioni di ALU più efficienti (che richiedevano un maggior numero di transistor) e alla loro inclusione in strutture come le pipeline.

Per esempio nei primi processori per realizzare le divisioni veniva utilizzata l'unità di sottrazione per emulare la divisione. Questa soluzione pur essendo molto semplice ed economica da implementare era molto lenta. Nei primi processori le somme e le moltiplicazioni venivano realizzate con semplice reti logiche formate da porte AND o OR. Queste reti erano veloci da realizzare ma fornivano i risultati lentamente dato che i riporti dovevano propagarsi e nelle implementazioni più semplici i riporti per propagarsi in modo completo lungo tutta la rete impiegavano un tempo pari al numero di bit da processare. Le implementazioni successive invece utilizzano reti logiche più complesse che effettuano una serie di precalcoli sui dati e sui riporti in modo da parallelizzare le operazione ed rendere le ALU più veloci. Ovviamente questi circuiti sono più complessi e quindi costosi da realizzare.

Floating Point Unit

Esempio di FPU

La Floating Point Unit (FPU), unità di calcolo in virgola mobile) è una parte della CPU specializzata nello svolgere i calcoli in virgola mobile. La maggior parte delle operazioni svolte sono semplice aritmetica (come addizioni o moltiplicazioni), ma alcuni sistemi sono in grado di svolgere calcoli esponenziali o trigonometrici (come estrazioni di radice o il calcolo del seno).

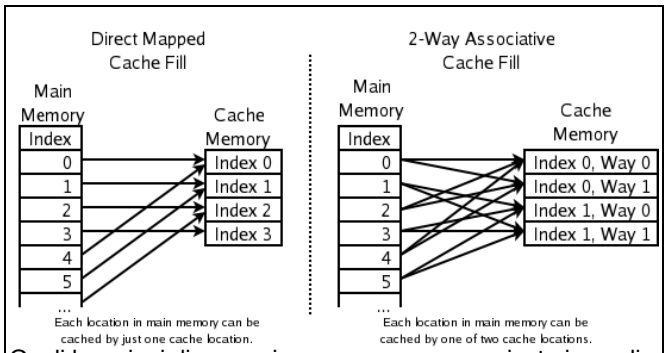
Non tutte le CPU possiedono una FPU dedicata: alcune usano degli emulatori per svolgere il calcolo in virgola mobile con una ALU; ciò risparmia il costo di una unità di calcolo aggiuntiva, ma il procedimento è sensibilmente più lento.

In alcune CPU il calcolo in virgola mobile è gestito in modo completamente separato dal calcolo intero, con registri dedicati è schemi di clock indipendenti. Le operazioni di addizione e di moltiplicazione sono generalmente eseguite all'interno di una pipeline, ma operazioni più complicate, come la divisione, a volte non lo sono, e alcuni sistemi hanno addirittura un circuito dedicato alla divisione.

Fino alla metà degli anni '90 era comune che le FPU fossero completamente separate dalle CPU all'interno degli home computer ed essere incluse solo come optional, utile solo per applicazioni con intenso calcolo matematico. Ne sono esempi i i387 e i487 della Intel, da affiancarsi rispettivamente ai Intel 80386 e Intel 80486|80486 SX, e il Motorola 68881, usato nei processori Motorola 68000, diffusi nei computer Macintosh. Da allora, la FPU ha cominciato ad essere integrata nei processori.

Cache

I processori nel corso degli anni sono diventati sempre più veloci mentre le memorie per computer non hanno avuto un incremento paragonabile quindi mentre negli anni '70 le memorie erano significativamente più veloci dei processori dagli anni '90 in poi i microprocessori sono diventati molto più veloci delle memorie. Per impedire al processore di passare più del 90% del suo tempo ad attende i dati dalla memoria si è deciso di dotare i processori di una ridotta quantità di memoria molto veloce dove conversare i dati utilizzati di frequente. Questa scelta deriva dalla constatazione che sebbene i programmi utilizzino molta memoria in realtà passano più del 90% del loro tempo ad accedere alle stesse locazioni di memoria.



Quali locazioni di memoria possono essere caricate in quali locazioni della cache

Essendo le cache una copia della memoria la copia deve essere fedele ed aggiornata, disallineamenti tra ciò che si trova in cache e lo stato della memoria può portare ad errori di elaborazione. Nel caso si voglia realizzare una cache che contenga anche i dati da elaborare bisogna progettare accuratamente il processore in modo da impedire disallineamenti tra lo stato della memoria e lo stato della cache. In un sistema con più processori infatti esistono appositi protocolli che invalidano il contenuto della cache di un microprocessore nel caso la cache non sia aggiornata con lo stato nella memoria. Questi meccanismi aumentano la complessità circuitale e riducono le prestazioni dei processori ma sono indispensabili per ottenere un'elaborazione corretta dei programmi.

Le prestazioni della cache dipendono moltissimo dalla scelta di un'adeguata politica di riempimento della stessa. Nella cache vanno memorizzati i dati utilizzati più di frequente e quindi i microprocessori includono dei circuiti che provvedono a scegliere i dati che nel futuro verranno probabilmente utilizzati. Sebbene vi siano stati molti studi teorici sull'argomento allo stato attuale l'algoritmo più utilizzato provvede a eliminare dalla cache i dati utilizzati meno di recente con i nuovi dati da caricare.

La politica di rimpiazzamento decide dove, nella cache, può risiedere una copia di una particolare locazione di memoria. Se la politica di rimpiazzamento è libera di scegliere in quale linea di cache caricare il dato, la cache è chiamata fully associative. Invece, se ogni dato in memoria può essere posizionato solo in una particolare linea di cache, essa è detta direct mapped. La maggior parte delle cache, però, implementa un compromesso chiamato set associative.

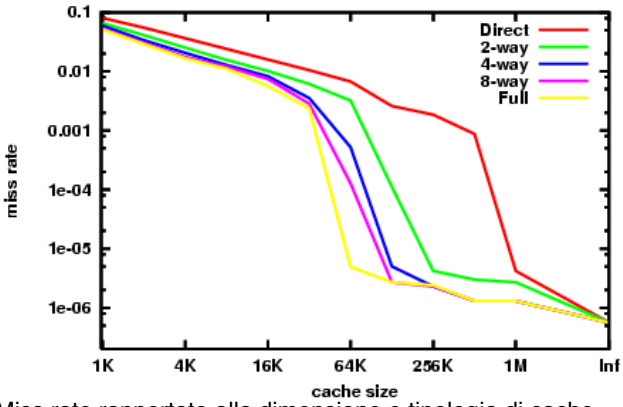
Per esempio, la cache dati di livello 1 dell'AMD Athlon è 2-way set associative, cioè una particolare locazione di memoria può essere caricata in cache in due distinte locazioni nella cache dati di livello 1.

Se ogni locazione in memoria principale può essere caricata in due locazioni diverse, la domanda sorge spontanea: quali? Lo schema utilizzato più frequentemente è mostrato nel diagramma a lato: i bit meno significativi dell'indice della locazione di memoria vengono usati come indici per la cache e ad ognuno di questi indici sono associate due linee di cache. Una buona proprietà di questo schema è che le etichette dei dati caricati in cache non devono includere quella parte dell'indice già codificata dalla linea di cache scelta. Poichè i tag sono espressi su meno bit, occupano meno memoria ed il tempo per processarli è minore.

Sono stati suggeriti altri schemi, come quello della skewed cache, dove l'indice della way 0 è diretto, come sopra, mentre l'indice per la way 1 è calcolato attraverso una funzione di hash. Una buona funzione di hash ha la proprietà che gli indirizzi che sono in conflitto con il direct mapping

tendono a non collidere quando sono mappati con la funzione di hash, così è meno probabile che un programma soffra di un numero imprevedibilmente grande di collisioni dovuti ad un metodo d'accesso particolarmente patologico. Lo svantaggio è il ritardo aggiuntivo necessario per calcolare il risultato della funzione di hash. In aggiunta, quando diventa necessario caricare una nuova linea ed eliminarne una vecchia, potrebbe rivelarsi difficile determinare quale tra le linee esistenti è stata usata meno recentemente, in quanto la nuova linea entra in conflitto con differenti "set" di linee per ogni "way"; il tracciamento LRU è infatti normalmente calcolato per ogni set di linee.

L'associatività è un compromesso. Se ci sono dieci posizioni, la politica di rimpiazzamento può riempire una nuova linea, ma quando bisogna cercare un dato devono essere controllate tutte e 10 le posizioni. Controllare più posizioni necessita di più potenza, area e tempo. D'altra parte, le cache con più associatività soffrono di meno cache miss (vedere anche più in basso). La regola di massima è che raddoppiare l'associatività ha circa lo stesso effetto sull'hit rate che il raddoppio della dimensione della cache, da 1-way (direct mapping) a 4-way. Aumenti dell'associatività oltre il 4-way hanno molto meno effetto sull'hit rate e sono generalmente utilizzati per altri motivi (vedere il virtual aliasing, più in basso).



Miss rate rapportato alla dimensione e tipologia di cache secondo il test SPEC CPU2000

Uno dei vantaggi della cache direct mapped è che permette una esecuzione speculativa semplice e veloce. Una volta che l'indirizzo è stato calcolato, è nota quale sia la linea di cache che potrebbe contenere il dato. Questa può essere letta ed il processore può continuare a lavorare con quel dato prima che finisca di controllare che l'etichetta effettivamente combaci con l'indirizzo richiesto.

L'idea che il processore utilizzi i dati in cache prima ancora che sia verificata la corrispondenza tra etichetta ed indirizzo può essere applicata anche alle cache associative. Un sottoinsieme dell'etichetta, chiamato in inglese hint, può essere utilizzato per scegliere temporaneamente una delle linee di cache associate all'indirizzo richiesto. Questo dato può essere utilizzato dalla CPU in parallelo, mentre l'etichetta viene controllata completamente. Questa tecnica lavora al meglio quando usata nel contesto della traduzione degli indirizzi, come spiegato più in basso.

Quando un dato è scritto nella cache, dopo un po' di tempo deve comunque essere scritto in memoria principale. La decisione del momento in cui questa scrittura deve aver luogo è controllata dalla politica di scrittura. In una cache write-through, ogni scrittura sulla cache comporta una scrittura contemporanea nella memoria principale. In alternativa, una cache write-back non esegue immediatamente questa azione: al contrario, la cache tiene traccia delle linee che contegono dati da aggiornare settando opportunamente quello che viene chiamato il dirty bit. Il dato viene effettivamente scritto in memoria solo quando esso deve essere eliminato dalla cache per far spazio a nuove informazioni. Per questa ragione, una ricerca fallita in una cache write-back spesso genera due accessi alla memoria: uno per leggere il nuovo dato, l'altro per scrivere la vecchia informazione (se indicato dal dirty bit).

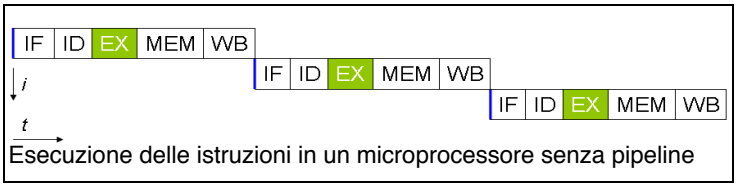
Esistono anche alcune politiche intermedie. La cache potrebbe essere ad esempio write-through, ma le scritture potrebbero essere temporaneamente inserite in una coda, così da processare insieme scritture multiple, ottimizzando l'accesso al bus.

Con l'evoluzione dei processori si è deciso di realizzare più livelli di cache. Questi livelli vengono indicato con L1, L2 e L3 a seconda che sia il primo, secondo o terzo livello. Il primo livello usualmente funziona alla stessa frequenza della CPU, il secondo livello funziona a un mezzo o un

terzo della frequenza di clock del processore e non sempre è incluso nello stesso processore mentre il terzo livello che non sempre esiste è una cache spesso montata sulla scheda madre del computer ed è una memoria funzionante a frequenza maggiore di quella della memoria RAM ma con frequenza inferiore a quella della cache L2.

Quasi tutti i processori moderni inoltre suddividono le cache in cache dati e cache istruzioni. Questa suddivisione permette di ottimizzare la gestione della cache dato che usualmente il processore esegue sempre le stesse istruzioni mentre i dati possono cambiare con frequenza e quindi una suddivisione dei flussi evita conflitti tra dati che cambiano rapidamente e istruzioni che tendono ad essere sostituite con lentezza.

Pipeline



Esecuzione delle istruzioni in un microprocessore senza pipeline

L'elaborazione di un istruzione da parte di un processore si compone di cinque passaggi fondamentali:

IF: Lettura dell'istruzione da memoria

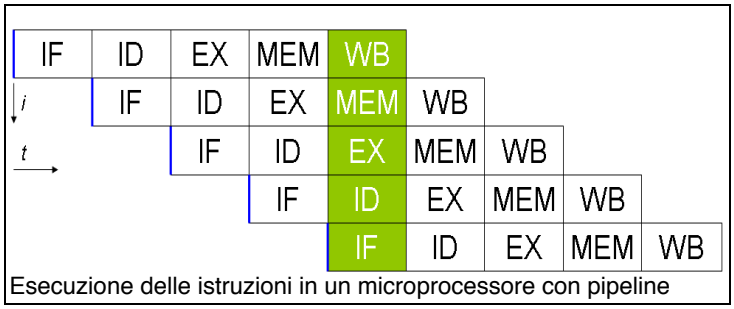
ID: Decodifica istruzione e lettura operandi da registri

EX: Esecuzione dell'istruzione

MEM: Attivazione della memoria (solo per certe istruzioni)

WB: Scrittura del risultato nel registro opportuno

Praticamente ogni CPU in commercio è gestita da un clock centrale e ogni operazione elementare richiede almeno un ciclo di clock per poter essere eseguita. Le prime CPU erano formate da un'unità polifunzionale che svolgeva in rigida sequenza tutti e cinque i passaggi legati all'elaborazione delle istruzioni. Una CPU classica richiede quindi almeno cinque cicli di clock per eseguire una singola istruzione.



Esecuzione delle istruzioni in un microprocessore con pipeline

Con il progresso della tecnologia si è potuto integrare un numero maggiore di transistor in un microprocessore e quindi si sono potute parallelizzare alcune operazioni riducendo i tempi di esecuzione. La pipeline dati è la massima parallelizzazione del lavoro di un microprocessore.

Una CPU con pipeline è composta da cinque stadi specializzati, capaci di eseguire ciascuno una operazione elementare di quelle sopra descritte. La CPU lavora come in una catena di montaggio e quindi ogni stadio provvede a svolgere solo un compito specifico. Quando la catena è a regime, ad ogni ciclo di clock esce dall'ultimo stadio un'istruzione completata. Nello stesso istante ogni unità sta elaborando in parallelo i diversi stadi delle successive istruzioni. In sostanza si guadagna una maggior velocità di esecuzione a prezzo di una maggior complessità circuitale del microprocessore, che non deve essere più composto da una sola unità generica ma da cinque unità specializzate che devono collaborare tra loro.

Problematiche

L'implementazione di una pipeline non sempre migliora le prestazioni. Questo è dovuto al fatto che le istruzioni possono richiedere l'elaborazione di dati non ancora disponibili e alla presenza dei salti condizionati.

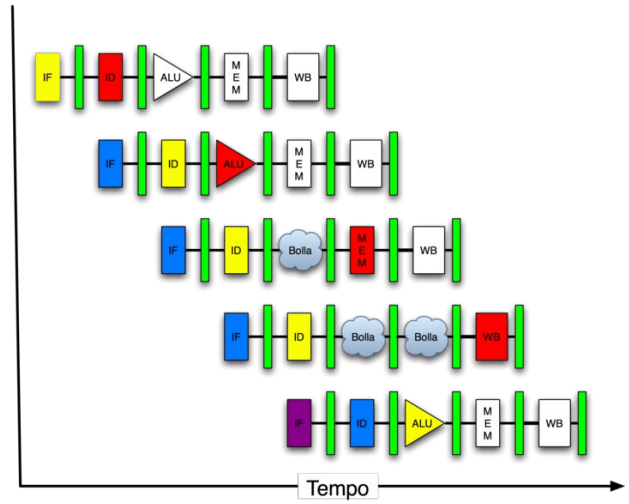
Notare come l'istruzione rossa fornisce il risultato dell'operazione direttamente all'istruzione gialla del ciclo successivo permettendole di essere eseguita senza rallentamenti

- Il primo problema deriva dal lavoro parallelo delle unità.

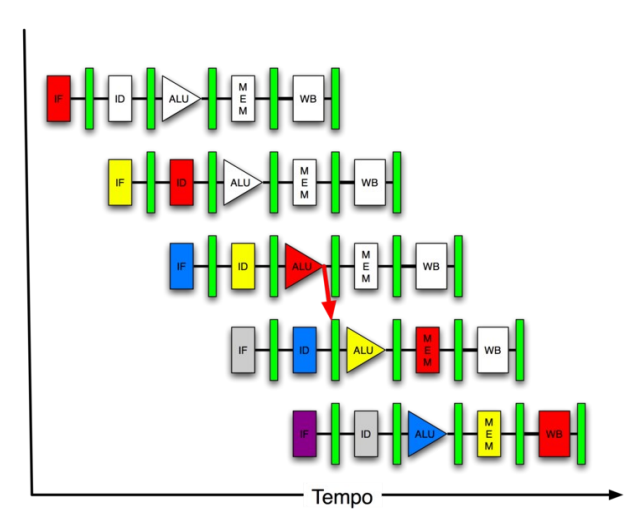
Supponiamo che la CPU con pipeline debba eseguire il seguente frammento di codice:

A+B=C (istruzione rossa)

C-1=D (istruzione gialla)



In questa pipeline non è previsto il riporto indietro dei risultati quindi l'istruzione gialla deve attendere la memorizzazione dell'istruzione rossa introducendo due bolle e bloccando la pipeline



Notare come l'istruzione rossa fornisce il risultato dell'operazione direttamente all'istruzione gialla del ciclo successivo permettendole di essere eseguita senza rallentamenti

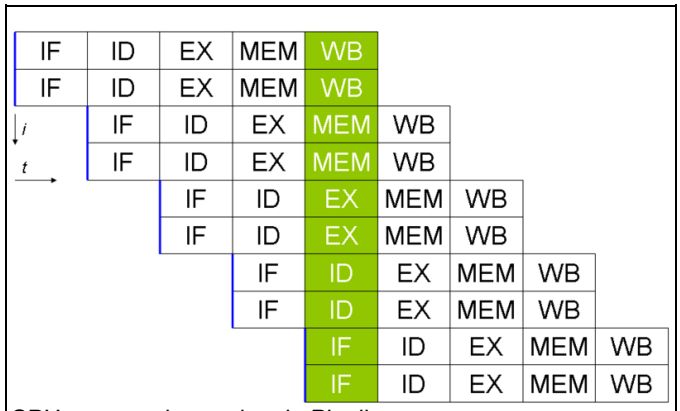
La prima istruzione deve prelevare i numeri contenuti nelle variabili A e B, sommarli e porli nella variabile C. La seconda istruzione deve prelevare il valore contenuto nella variabile C, sottrarlo di uno e salvare il risultato in D. Ma la seconda istruzione non potrà essere elaborata (EX) fino a quando il dato della prima operazione non sarà disponibile in memoria (MEM) e quindi la seconda operazione dovrà bloccarsi per attendere il completamento della prima e quindi questo ridurrà il throughput complessivo. Questo problema viene affrontata implementando all'interno dei registri a doppia porta. Questi registri sono in grado di riportare i risultati appena elaborati alle istruzioni successive senza dover attendere il loro salvataggio in memoria. Quindi una volta eseguita la fase 3 (fase EX della pipeline) i risultati possono essere utilizzati dalla istruzione successiva. Quindi seguendo l'esempio sopra esposto alla fine del terzo ciclo di clock il risultato dell'operazione A+B=C può essere utilizzato dalla operazione successiva (C-1=D) che essendo solo al suo ciclo di clock è ancora nella fase di decodifica e quindi non viene rallentata. Questa propagazione all'indietro dei risultati permette di eliminare gli stalli di elaborazione o comunque permette di limitarli fortemente.

- Il secondo problema consiste nei salti condizionati.

I programmi contengono delle istruzioni condizionate che se una specifica condizione è verificata provvedono a interrompere il flusso abituale del programma e a mandare in esecuzione un altro pezzo di programma indicato dall'istruzione di salto. Ogni volta che questo accade il microprocessore si trova a dover eseguire un nuovo flusso di operazioni e quindi deve svuotare la

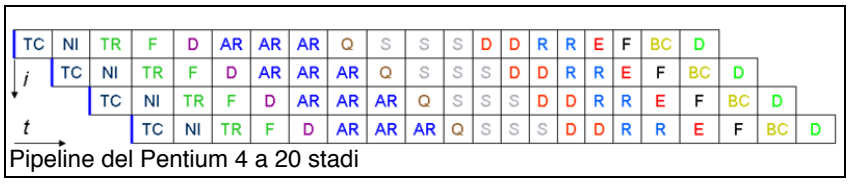
pipeline del precedente flusso e caricare il nuovo flusso. Ovviamente queste operazioni fanno sprecare cicli di clock e quindi deprimono il throughput. Per ridurre questo problema le CPU adottano delle unità chiamate unità di predizione delle diramazioni (in inglese *Branch Prediction Unit*) che fanno delle previsioni sul flusso del programma. Queste unità riducono notevolmente i cicli persi per i salti. Queste unità fanno un'analisi speculativa del codice cercando di prevedere se il salto verrà eseguito oppure no e fanno eseguire alle pipeline il codice più probabile.

Evoluzioni



CPU superscalare a doppia Pipeline

Per realizzare CPU con prestazioni migliori col tempo si è affermata la strategia di integrare in un unico microprocessore più pipeline che funzionano in parallelo. Questi microprocessori sono definiti superscalar] dato che sono in grado di eseguire mediamente più di un'operazione per ciclo di clock. Queste pipeline ovviamente rendono ancora più complessa la gestione dei problemi di coerenza e dei salti condizionati. Nelle CPU moderne inoltre le pipeline non sono composte da soli cinque stadi ma in realtà ne utilizzano molti di più (il Pentium 4 ne utilizza da 20 fino a 30). Questo si è reso necessario per potere innalzare la frequenza di clock. Spezzettando le singole operazioni necessarie per completare un istruzione in tante sotto operazioni si può elevare la frequenza della CPU dato che ogni unità deve svolgere un'operazione più semplice e quindi può impiegare meno tempo per completare la sua operazione. Questa scelta di progettazione consente effettivamente di aumentare la frequenza di funzionamento delle CPU ma rende critico il problema dei salti condizionati. In caso di un salto condizionato non previsto il Pentium 4 per esempio può essere costretto a svuotare e ricaricare una pipeline di 30 stadi perdendo fino a 30 cicli di clock contro una classica CPU a pipeline a 5 stadi che avrebbe sprecato nella peggiore delle ipotesi 5 cicli di clock.

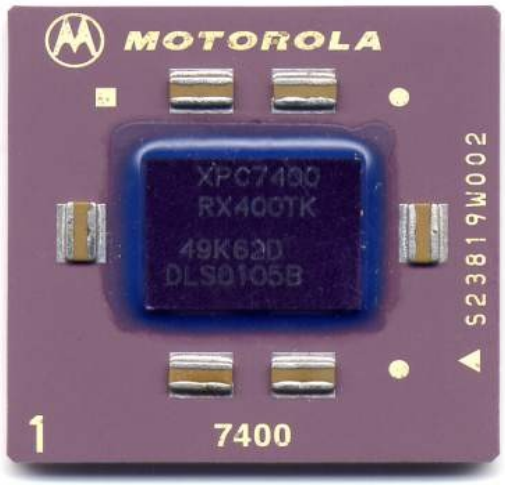


Pipeline del Pentium 4 a 20 stadi

La sempre maggior richiesta di potenza di calcolo ha spinto le industrie produttrici di microprocessori a integrare in un unico chip più microprocessori. Questo strategia consente al computer di avere due CPU separate dal punto di vista logico ma fisicamente risiedenti nello stesso chip. Questa strategia progettuale attenua i problemi di coerenza e di predizione dei salti. Infatti ogni CPU logica esegue un programma separato e quindi tra i diversi programmi non si possono avere problemi di coerenza tra le istruzioni. Questa scelta progettuale aumenta le prestazioni solo nel caso in cui il sistema operativo sia in grado di utilizzare più programmi contemporaneamente e i programmi siano scritti per poter utilizzare le CPU disponibili, quindi se i programmi sono parallelizzabili.

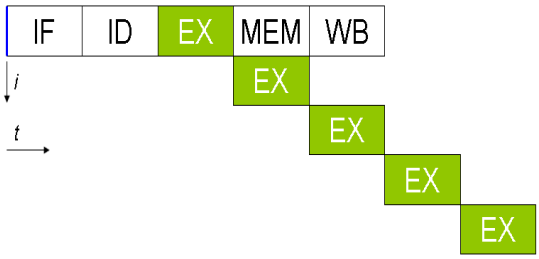
Single Instruction Multiple Data

Quasi tutti i moderni processori per incrementare le loro prestazioni dell'elaborazione dei dati multimediali fanno uso di unità SIMD. Le unità SIMD sono unità basate su più unità specializzate per elaborare dati diversi in parallelo. Questa architettura viene utilizzata dai processori vettoriali o da processori che funzionano in parallelo.



Motorola PowerPC G4, primo processore PowerPC dotato di unità SIMD

In passato venivano prodotti un numero elevato di dispositivi dedicati allo svolgimento di compiti specifici. Usualmente questi dispositivi erano DSP opportunamente programmati. La differenza fondamentale tra le istruzioni SIMD e i DSP e che questi sono dotati di un set di istruzioni completo e quindi sono in grado di svolgere teoricamente qualsiasi compito. Invece le istruzioni SIMD sono progettate per manipolare elevate quantità di dati in parallelo e per le usuali operazioni si appoggiano ad un altro insieme di istruzioni usualmente gestito dal microprocessore. Inoltre i DSP tendono a includere un certo numero di istruzioni dedicate ad elaborare tipi specifici di dati come possono essere i dati audio o video mentre le istruzioni SIMD vengono utilizzate per elaborare dati generici.



Pipeline di un processore SIMD, dopo aver decoditia to l'istruzione e caricato i dati applica le operazioni a un blocco di dati sequenzalmente senza dover ogni volta decoditia re le istruzioni

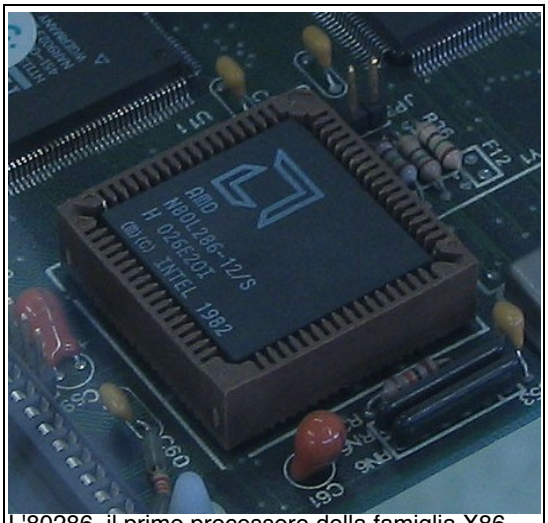
Nell'elaborazioni di dati multimediali spesso si incontrano algoritmi che possono avvantaggiarsi di un'architettura SIMD. Per esempio per cambiare la luminosità di un immagine un microprocessore dovrebbe caricare ogni pixel che compone l'immagine dei suoi registri, effettuare la modifica della luminosità e poi salvare i risultati in memoria. Un processore SIMD eseguirebbe prima un' istruzione che caricherebbe con un unica operazione un certo numero di pixel (il numero preciso dipende dall'architettura) poi il processore modificherebbe tutti i dati in parallelo e in seguito li salverebbe tutti contemporaneamente in memoria. Eseguire le operazioni a blocchi invece che agire sui singoli pixel rende le operazioni molto più efficienti dato che i moderni computer sono progettati per trasferire i dati a blocchi e sono inefficienti nei singoli accessi alla memoria.

Un altro vantaggio deriva dal fatto che tipicamente le istruzioni SIMD sono in grado di manipolare tutti i dati caricati contemporaneamente: Quindi se un processore SIMD è in grado di caricare 8 dati, questo sarà anche in grado di processarli tutti contemporaneamente. Anche i microprocessori superscalari sono in grado di elaborare più dati contemporaneamente ma con un'efficienza inferiore.

Problematiche

Le architetture basate su SIMD richiedono un numero elevato di registri e quindi a volte i progettisti per ridurre i costi decidono di utilizzare i registri della FPU. Questa scelta rende impossibile utilizzare istruzioni SIMD e FPU contemporaneamente a meno di lenti cambi di contesto. Questo era l'approccio scelto da intel per le istruzioni MMX che infatti sono notoriamente lente se associate a operazioni in virgola mobile.

Memory Management Unit



L'80286, il primo processore della famiglia X86 dotato di MMU

Una Memory Management Unit (MMU) in un processore ha vari compiti, tra cui la traslazione (o traduzione) degli indirizzi virtuali in indirizzi fisici (necessaria per la gestione della memoria virtuale), la protezione della memoria, il controllo della cache della CPU, l'arbitraggio del bus, e, in architetture più semplici (specialmente nei sistemi a 8-bit), la commutazione di banchi di memoria.

Le MMU moderne generalmente suddividono lo spazio degli indirizzi virtuali (l'intervallo di indirizzi accessibili dal processore) in pagine di memoria dimensione 2N, tipicamente pochi kilobytes. Gli N bit meno significativi dell'indirizzo (l'offset all'interno della pagina) rimangono invariati, mentre i bit restanti rappresentano il numero virtuale della pagina. La MMU contiene una tabella delle pagine indicizzata (possibilmente associativamente) dal numero della pagina. Ogni elemento di questa tabella (detto PTE o Page Table Entry) restituisce il numero fisico della pagina corrispondente a quello virtuale, che, combinato con l'offset della pagina, forma l'idirizzo fisico completo.

Un PTE può includere anche informazioni su quando la pagina è stata usata per l'ultima volta (per l'algoritmo di sostituzione LRU), quale tipo di processi (utente o supervisore) puo leggerla e scriverla, e se dovrebbe essere inserita nella cache.

È possibile che non esista memoria fisica (RAM) allocata a una data pagina virtuale. In questo caso, la MMU segnala una condizione di «pagina di memoria mancante» (page fault) alla CPU. Il sistema operativo interviene per gestire questa condizione, cerca di trovare una pagina libera nella RAM e di creare una nuova PTE nella quale mappare l'indirizzo virtuale richiesto nell'indirizzo fisico della pagina trovata. Quando non c'è spazio disponibile nella RAM per una nuova pagina di memoria, può essere necessario scegliere una pagina esistente utilizzando un algoritmo di sostituzione, farne una copia su disco rigido e rimpiazzarla con quella nuova. Analogamente, quando non ci sono PTE inutilizzate a disposizione, il sistema operativo deve liberarne una.

In alcuni casi un page fault puo indicare un errore nel software. Uno dei benefici della MMU è la protezione della memoria: un sistema operativo puo usarla per proteggere la memoria da processi erranti, impedendo a un processo di accedere a locazioni di memoria non autorizzate. Tipicamente, il sistema operativo assegna ad ogni processo il suo spazio di indirizzamento virtuale.

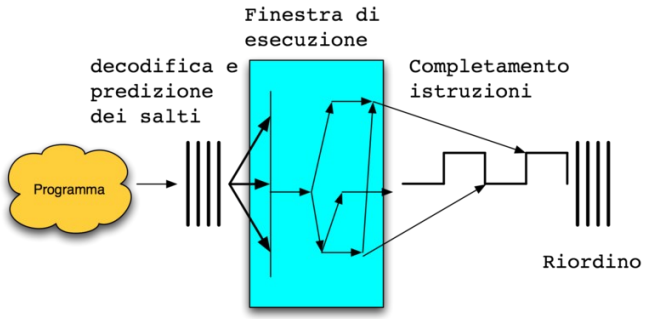
La MMU riduce anche il problema della frammentazione della memoria. Dopo che blocchi di memoria precedentemente allocati sono stati liberati, la memoria libera puo diventare frammentata (discontinua) quindi il blocco più grande di memoria libera contigua puo essere molto più piccolo del totale. Con la memoria virtuale, blocchi non contigui di memoria fisica possono essere mappati in indirizzi virtuali contigui.

Nelle prime realizzazioni, la gestione della memoria era eseguita in un circuito integrato separato dalla CPU, come ad esempio l'MC 68851 usato dalla CPU Motorola 68020 nel Macintosh II, o lo Z8015 usato dalla famiglia di processori Z80 della Zilog. CPU più moderne quali il Motorola 68030 e lo Zilog Z280 possiedono MMU integrate nello stesso chip.

La maggior parte delle MMU moderne, come quelle descritte, funzionano con sistemi di memoria paginata. Esistono tuttavia altri sistemi per organizzare la memoria, come la segmentazione e l'indirizzamento base-limite, che ne è uno sviluppo. Alcune MMU funzionano anche con questi

sistemi di memoria, che benché meno frequenti sono utilizzati in alcune architetture moderne di rilevanza notevole. Ad esempio, l'architettura x86 può funzionare con memoria segmentata oltre che paginata.

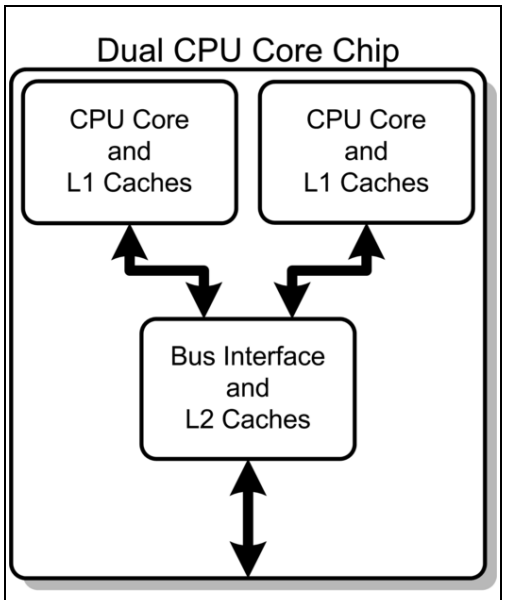
Processore superscalare



Schema generico di un processore superscalare. Il programma viene caricato come istruzioni seriali, le istruzioni vengono analizzate, riordinate tenendo conto delle dipendenze, eseguite in parallelo, ed riordinate prima di provvedere al salvataggio dei dati in memoria.

In un processore superscalare sono presenti diverse unità funzionali dello stesso tipo, con dispositivi addizionali per distribuire le istruzioni alle varie unità. Per esempio, sono generalmente presenti numerose unità per il calcolo intero (ALU). Le unità di controllo stabiliscono quali istruzioni possono essere eseguite in parallelo e le inviano alle rispettive unità. Questo compito non è semplice, dato che un'istruzione può richiedere il risultato della precedente come proprio operando, oppure può dover impiegare il dato conservato in un registro usato anche dall'altra istruzione; il risultato può quindi cambiare secondo l'ordine d'esecuzione delle istruzioni. La maggior parte delle CPU moderne dedica un elevato numero di transistor allo svolgimento di questo compito, per permettere al processore di funzionare a pieno regime in modo costante; compito che si è reso sempre più importante con l'aumento del numero delle unità. Mentre le prime CPU superscalari possedevano due ALU ed una FPU, un processore attuale come ad esempio il PowerPC 970 possiede quattro ALU, due FPU e due unità SIMD. Se l'unità di decodifica delle istruzioni non mantiene occupate tutte le unità funzionali del processore, le prestazioni ne soffrono grandemente.

Le architetture superscalari ebbero origine nell'ambiente RISC, dato che questo tipo di design richiede unità funzionali semplici, che possono essere incluse in più esemplari in una unica CPU. Questa è la ragione per cui questi processori erano più veloci dei CISC tra gli anni '80 e gli anni '90. Tuttavia, col progresso della tecnologia, anche design ingombranti come l'IA-32 poterono essere progettati in modo superscalare.



Schema generico di una CPU dual core

Attualmente è impensabile un futuro miglioramento sensibile del sistema di controllo, ponendo di fatto un limite ai miglioramenti prestazionali dei processori superscalari. Il progetto VLIW (very long instruction word) cerca una soluzione scaricando parte del processo di controllo delle istruzioni in fase di scrittura del programma e di compilazione, evitando al processore di doverlo ripetere ad ogni esecuzione del programma.

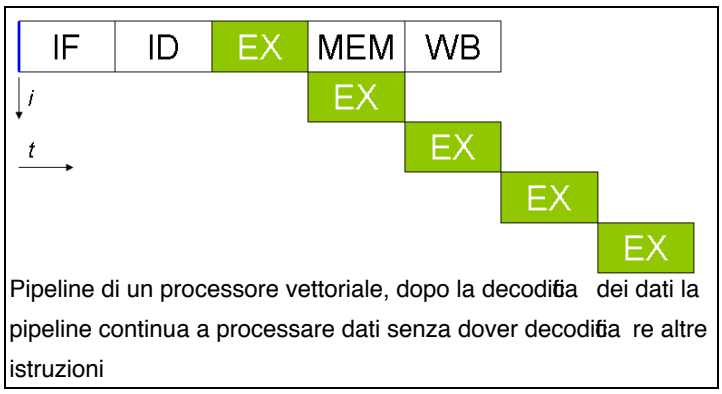
Un'altra evoluzione dei processori superscalari è l'integrazione di più processori indipendenti (core) in un singolo processori. Questi processori non sono solo dotati di più pipeline la le varie pipeline sono totalmente separate e quindi sono in grado di eseguire programmi diversi cosa non possibile nelle cpu classiche. I processori Core Duo dell'Intel per esempio sono di questo tipo. Un'approccio intermedio prevede una separazione logica e non fisica delle pipeline con le pipeline separate ma i circuiti di controllo e gestione ancora in comune. Questo permette di eseguire più tread in parallelo senza dover duplicare tutte le unità funzionali di un processore e quindi risparmiando molti transistor rispetto a una soluzione pura. esempi di questa soluzione sono la

tecnologia Hyper-Threading. Il già citato Core Duo è dotato di due core con supporto dell'Hiper_threading e quindi è in grado di eseguire fino a quattro tread simultaneamente.

La realizzazione di processori con più core è una soluzione migliore rispetto alla semplice aggiunta di nuove unità pipeline dato che ogni nuova pipeline aumenta la possibilità di eseguire istruzioni che siano in conflitto con altre e quindi spingersi oltre quattro pipeline risulta spesso sconveniente. Eseguendo più tread in parallelo si eliminano i problemi dati che i tread sono separati e quindi le varie pipeline non possono entrare in conflitto tra di loro. Questi processori però costringono i programmatori a realizzare programmi paralleli per fruttare al meglio i processori moderni e la realizzazione di programmi paralleli non è semplice ed per alcuni algoritmi non esistono nemmeno metodi per renderli paralleli in modo efficiente.

Processore vettoriale

Un processore vettoriale o array processor è una CPU progettata per svolgere operazioni matematiche su più dati elementari contemporaneamente. Questo in contrasto con l'architettura classica di un processore scalare che prevede l'elaborazione di un singolo dato per volta. La maggior parte dei processori sono scalari (o esternamente lo sembrano). I processori vettoriali sono comuni nelle applicazioni scientifiche e sono spesso alla base dei supercomputer fin dagli anni 80.



Pipeline di un processore vettoriale, dopo la decodifica dei dati la pipeline continua a processare dati senza dover decodificare altre istruzioni

Con la fine degli anni 90 i microprocessori sono cresciuti di prestazioni e molti processori per applicazioni generiche si sono dotati di unità vettoriali o sono diventati vettoriali al loro interno. Nel 2000 IBM Toshiba e Sony hanno iniziato lo sviluppo del processore Cell, un microprocessore ad elevate prestazioni dotato di svariate unità vettoriali e rivolto ad applicazioni che spaziano dalle consolle al supercalcolo.

Attualmente praticamente ogni CPU moderna include istruzioni vettoriali tipicamente conosciute come istruzioni SIMD. Le consolle per i videogiochi e le schede grafiche fanno un ampio uso di processori vettoriali dato che l'elaborazione di flussi audio e video in tempo reale è un campo che ben si presta all'elaborazione vettoriale.

Vantaggi

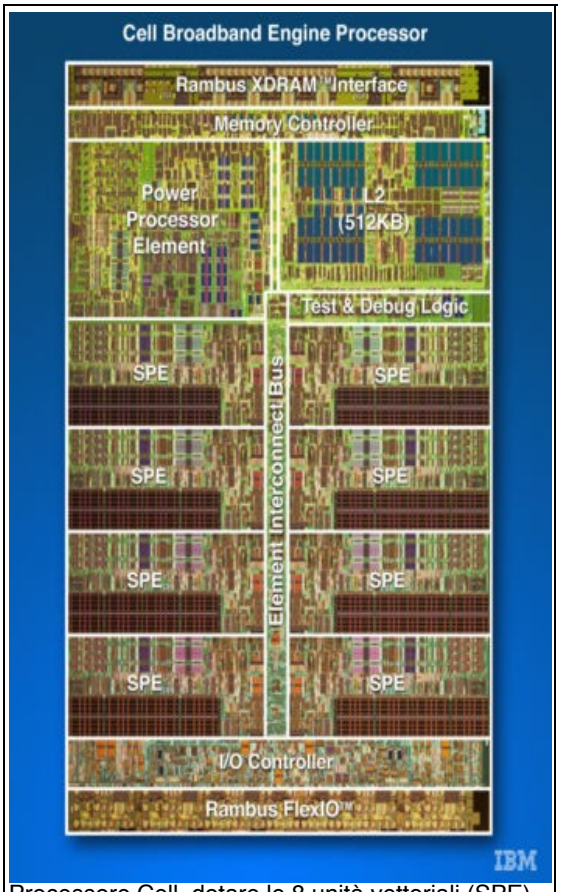
Dato che un'unica operazione vettoriale opera su più dati contemporaneamente questo consente di leggere meno dati rispetto a un classico processore. I dati vettoriali sono tra di lo indipendenti e quindi si può realizzare unità con un elevato parallelismo con unità di controllo semplici e quindi con pochi transistor. Un numero ri dotto di transistor consentono di ottenere frequenze di funzionamento elevate. Essendo che il compilatore provvede a ridurre le dipendenze le unità di gestione diventano ancora più semplici. Le istruzioni vettoriali accedono alla memoria secondo schemi noti quindi si possono ottimizzare gli accessi alle memorie dato che i dati vengono salvati in ampi registri vettoriali. Inoltre queste unità possono fare a meno di cache dati.

Architettura

Internamente un processore vettoriale è formato da un'architettura basata su registri vettoriali. Ogni registro vettoriale è composto da un insieme di registri caratterizzati da un unico nome e un indice che permette di accedere al singolo dato. Le operazioni vettoriali lavorano unicamente tra registri vettoriali tranne ovviamente le operazioni di load e di store che provvedo a caricare e scaricare i dati dai registri verso la memoria.

I componenti di un processore vettoriale sono:

- CPU scalare: Questa unità è composta da registri, logica per la lettura delle istruzioni e della loro decodifica.
- Registri vettoriali: Questi registri sono formati da un insieme di registri accorpati per nome e indirizzabili tramite un indice. Sono dotati di almeno due porte di lettura e di una di scrittura e possono essere da 8 fino a 32. A volte i registri supportano parole di lunghezza variabile (8,16,32,64 bit) questo risulta comodo per applicazioni multimediali.
- Unità funzionale vettoriale: Questa unità è di tipo pipeline per poter iniziare una nuova operazione ogni ciclo di clock. Tipicamente ne esistono da 2 a 8 in grado di lavorare su interi o su numeri in virgola mobile.
- Unità vettoriale di load-store: Questa unità è un'unità pipeline che provvede a leggere e scrivere i dati dai registri alla memoria. L'unità può leggere o scrivere più dati in contemporanea e in un processore possono esserci più unità di load-store.
- Matrice di commutazione: Questa matrice mette in comunicazione le varie unità funzionali del processore.



Processore Cell, dotare le 8 unità vettoriali (SPE)

Metodi di accesso

L'unità vettoriale di load-store usualmente supporta almeno tre modalità di accesso.

- Metodo di accesso a passo unitario, il più veloce.
- Metodo di accesso a passo costante
- Metodo di accesso indicizzato. Accede alla memoria tramite un indice, è molto utile per accedere alle matrici sparse e permettere di vettorializzare molti programmi

Svantaggi

Questi processori sono poco adatti all'elaborazione di dati distribuiti in modo non costante e quindi le loro reali prestazioni dipendono dalla tipologia di programma in esecuzione e in alcuni casi anche dai dati trattati.

Unità vettoriali

Tutti i processori moderni supportano operazioni vettoriali. Questo perché i processori classici mal si prestano all'elaborazione di dati multimediali. L'utilizzo di chip dedicati per l'elaborazione multimediali non ha mai preso piede dato che quieti chip sono limitati nell'utilizzo, complicano lo sviluppo dei computer e non sono mai stati ben supportati dal software. Invece l'inclusione di queste unità nei processori moderni permette di migliorare le prestazione nel campo del multimedia senza incrementare i costi in modo significativo. Difatti basta aggiungere qualche registro (o utilizzarne alcuni poco usati come quelli del processore matematico), modificare le pipeline in modo da poter gestire gruppi di dati in parallelo e aggiungere la decodifica di alcune istruzioni in più.

Very Long Instruction Word

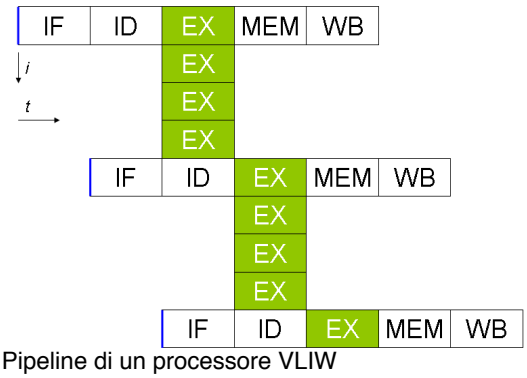
Le architetture Very Long Instruction Word sono basate sull'utilizzo del parallelismo intrinseco presente delle istruzioni. Similmente ai microprocessori superscalari queste CPU sono dotate di più unità di calcolo indipendenti (per esempio due moltiplicatori) per permettere alla CPU di eseguire più calcoli contemporaneamente (per esempio due moltiplicazioni).

L'Intel i860 uno dei primi processori VLIW prodotti

Progetto

In un progetto superscalare il numero di unità di calcolo non è visibile nel set di instruzioni. Ogni istruzione codifica una sola instruzione, per molti microprocessori è di 32 bit o meno.

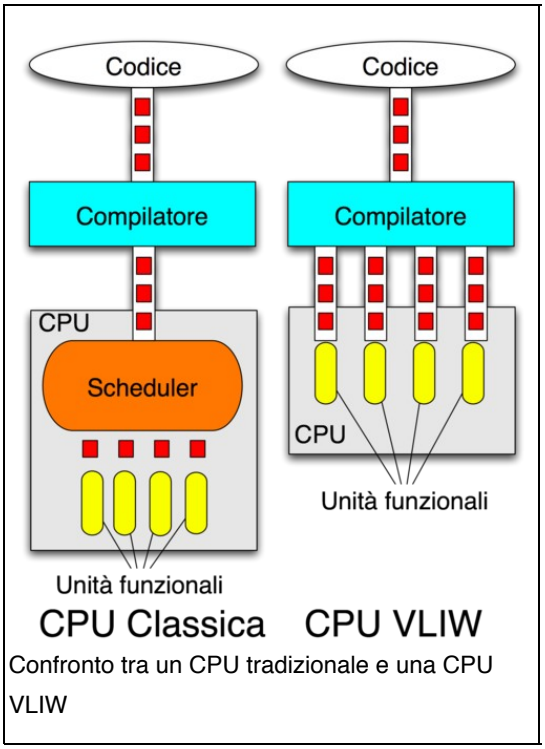
Invece ogni istruzione VLIW codifica più istruzioni elementari specificando ogni istruzione per ogni unità di calcolo. Per esempio un dispositivo VLIW con 5 unità di calcolo sarà dotato di istruzioni con cinque campi, ogni campo specifico per ogni unità di calcolo. Ovviamente le istruzioni VLIW sono molto più lunghe delle classiche istruzioni, sono lunghe almeno 64 bit ma spesso sono di 128 bit o più.



Pipeline di un processore VLIW

Sin dalle prime architetture ci si è resi conto che aggiungendo unità di calcolo alle macchine si potevano incrementare le prestazioni senza aumentare i costi in maniera eccessiva. Nelle CPU superscalari è la CPU stessa che durante l'esecuzione decide dinamicamente quali istruzioni mandare in esecuzione in parallelo. nelle CPU VLIW è il compilatore che durante la fase di traduzione decide quali istruzioni vadano eseguite in parallelo.

Per esempio una CPU può essere in grado di eseguire due moltiplicazioni contemporaneamente. Supponendo che la CPU riceva le due moltiplicazioni, la prima sarà mandata in esecuzione nella prima unità ma se la seconda moltiplicazione dipendesse dal risultato della prima questa non potrebbe essere mandata in esecuzione e al suo posto verrebbe effettuato un blocco in hardware. In un'istruzione VLIW il compilatore individuerebbe il conflitto e introdurrebbe una NOP per la seconda unità di calcolo. Questo riduce la complessità della CPU.



Confronto tra un CPU tradizionale e una CPU VLIW

Inoltre un compilatore VLIW può riconoscere il problema delle due moltiplicazioni e quindi anticipare una istruzione che non ha precondizioni per poter incrementare le prestazioni della CPU evitando l'utilizzo dell'istruzione NOP. Un simile approccio viene seguito anche da alcune CPU superscalari moderne che però dovendo eseguire queste dicisioni in tempo reali

forniscono modeste prestazioni e incrementano ulteriormente la complessità del progetto.

Un simile problema si presenta se il risultato di un'istruzione viene utilizzato per definire se uscire da un ciclo o no. Molte CPU moderne scelgono in anticipo un percorso in modo da poter caricare i dati corrispondente. Alcune CPU sono dotate di una unità di predizione delle diramazioni che effettua una analisi del codice per prevedere la diramazione più probabile. Questi metodi incrementano la complessità del progetto e corrompono la filosofia originaria delle architetture RISC anche perché la CPU deve contenere anche l'elettronica che in caso di errore della predizione elimina le istruzioni in esecuzione e elimina le eventuali modifiche già eseguite.

In un'architettura VLIW il compilatore utilizza delle euristiche o dei profili per predeterminare in anticipo il ramo più probabile. Avendo il compilatore molto più tempo della CPU e la possibilità di analizzare tutto il codice e non solo qualche istruzione le sue previsioni sono molto più precise di quelle effettuate da una CPU in tempo reale. Comunque il compilatore sviluppa il codice con il ramo più probabile già codificato nel codice e fornisce anche il codice per eliminare le istruzioni già eseguite nel caso le previsione non sia quella corretta.

Problematiche

Il principale problema di questa architettura è l'estrema dipendenza dei programmi dal compilatore. Un programma ottimizzato per un processore VLIW per lavorare in modo efficiente sulla generazione successiva di microprocessori va quasi sempre ricompilato. Questo rende problematico per un utente cambiare il computer dato che anche il suo parco software andrebbe adattato al nuovo processore a meno che i programmi non siano in grado scritti con un linguaggio come il Java che essendo in realtà compilato durante l'esecuzione possa essere adattato alla macchina durante l'esecuzione. Un'altra strategia è utilizzare uno strato software che interpreti il vecchio codice e lo adatti al nuovo processore ma in questo caso si ha un deperimento delle prestazioni che può essere anche molto marcato. Questa strategia viene utilizzata per esempio dal processore Efficeon della Transmeta che interpreta codice Intel X86 standard e internamente lo traduce in codice VLIW per la CPU.

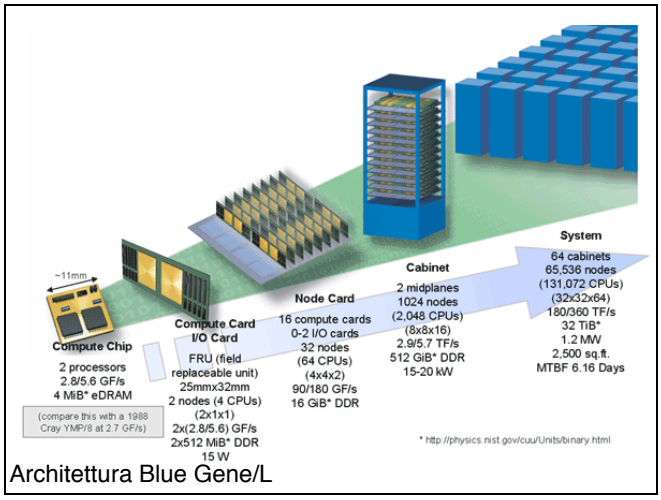
Evoluzioni

L'Itanium 2, l'ultimo esponente della famiglia basata su architettura EPIC

L'architettura VLIW ha indubbiamente molti vantaggi ma i suoi problemi ne rendono problematico l'utilizzo in processori per computer. La necessita di ricompilare il codice per ogni generazione di processori in particolare si scontra con la necessita degli utenti di poter mantenere il parco software. Per eliminare questi problemi diverse società hanno sviluppato delle evoluzioni dell'architettura VLIW, tra le varie evoluzioni la più famosa è l'architettura EPIC sviluppata da Intel e HP congiuntamente. L'architettura EPIC (Explicitly Parallel Instruction Computing) raggruppa le istruzioni elementari in parole come una classica architettura VLIW e inserisce inoltre delle informazioni sul parallelismo tra le varie parole. In questo modo le varie generazioni del processore possono variare internamente la loro architettura senza troppi problemi. Le informazioni sul parallelismo permettono di realizzare unità di decodifica che sfruttano il parallelismo efficientemente ma sono nel contempo semplici dato che l'analisi del codice parallelo e la sua suddivisione è stata effettuata dal compilatore. Inoltre l'architettura EPIC per migliorare le prestazioni aggiunge molti registri (diverse centinaia) per evitare di implementare l'unità di

ridenominazione dinamica dei registri, aggiunge delle istruzioni predicative per evitare lo svuotamento delle pipeline e altre innovazioni per velocizzare i cambi di contento tra le subroutine e per migliorare la gestione della cache. L'architettura EPIC è stata implementata da Intel nei processori Itaniun e Itanium 2. Questa famiglia di processori dopo un avvio molto stentato nel settore dei server ha ultimamente conquistato quote di mercato. La prima versione dell'Itanium forniva prestazioni deludenti, la cache di primo e secondo livello era piccola ed il codice EPIC per via delle istruzioni aggiuntive risulta più grande dell'equivalente codice per x86. Quindi le cache potevano contenere porzioni di codice ridotto e i continui accessi alla memoria penalizzavano il processore. Le successive generazioni dell'Itanium fornirono i processori di cache generose per arrivare fino all'Itanium 2 MP 9050, un processore dotato di due core separati con cache di primo livello da 32 Kilobyte per core, 512 Kilobyte di cache di secondo livello per core e 24 Megabyte di cache di terzo livello unificata. Ovviamente tutta questa memoria incide sul numero di transistor che in questo modello arrivano ad essere 1.72 Miliardi.

Evoluzioni future



Architettura Blue Gene/L

Attualmente il più promettente campo per incrementare le prestazioni dei processori è l'elaborazione parallela. Le strade legate alle altre tecnologie come cache e pipeline sono state intensamente studiate e significativi aumenti di prestazioni seguendo queste strade sembrano improbabili. Incrementi di frequenza dei processori diventano sempre più difficili per limiti tecnologici e fisici, le attuali tecniche litografiche infatti riescono a ridurre le dimensioni dei transistor ma non ad innalzarne le frequenze in modo rilevante sia per problemi di produzione sia per problemi di consumo e di dissipazione, processori che consumano 100 Watt sono difficili da dissipare e da integrare in macchine a basso consumo. Quindi l'unica strada percorribile rimane appunto l'elaborazione parallela cioè l'integrazione in un unico integrato di più processori in grado di eseguire uno o più processi contemporaneamente. Allo stato attuale (maggio 2006) esistono in commercio per applicazioni domestiche processori con due nuclei di calcolo indipendenti e ogni unità di calcolo è in grado di eseguire due processi indipendenti e quindi il processore è in sostanza in grado di eseguire quattro programmi contemporaneamente. Per applicazioni professionali società come la Sun Microsystems hanno presentato processori in grado di eseguire decine di processi in contemporanea come l'UltraSPARC T1 un processore con 8 unità di calcolo e con ogni unità in grado di eseguire 4 processi per un totale di 32 processi in contemporanea. Lo stato dell'arte di questo settore è detenuto dall'IBM che con l'architettura Blue Gene domina il settore dei supercomputer. Il Blue Gene/L è dotato di 131.072 processori con ogni processore dotato di due unità di calcolo per un totale 262.144 unità di calcolo indipendenti. Le attuali ricerche riguardanti le architetture dei processori infatti riguardano l'integrazione delle varie unità in modo efficiente, veloce e con ridotti consumi energetici dato che l'integrazione di molte unità di calcolo in uno spazio ridotto pone seri problemi di dissipazione e di consumi energetici dato che i supercomputer possono consumare anche diversi megawatt di potenza.

Bibliografia

Il libro si è basato sulle seguenti voci dell'enciclopedia online it.wikipedia.org con varie aggiunte e integrazioni.

- Processore
- CPU
- Arithmetic Logic Unit
- Floating Point Unit
- Pipeline dati
- Unità di predizione dei salti
- Cache dati
- Single Instruction Multiple Data
- Memory Menagement Unit
- Processore superscalare
- Processore vettoriale
- Very Long Istruction Word

Licenza

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with

generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your

agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.
If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights,

from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.