Calcolatori

Giacomo Fantoni

4 dicembre 2019

Indice

Capitolo 1

Introduzione

1.1 Tipi di calcolatori

I calcolatori vengono raggruppati in quattro classi:

- Personal computer: offrono buone prestazioni per un singolo utente mantenendo il costo limitato. Tipicamente eseguono software di terze parti.
- Server: calcolatori di dimensioni maggiori orientati verso l'elaborazione di grandi carichi di lavoro come applicazioni scientifiche o per il web.Tipicamente eseguono software di terze parti personalizzato. Stesse tecnologie dei personal computer ma con maggiore potenza di calcolo, maggiore velocità di input-output e maggiore capacità di memoria. Estremamente affidabili.
- Sistemi embedded: microprocessori progettati per l'esecuzione di applicazioni collegate tra loro implementate con l'hardware. Hanno prestazioni limitate.
- Dispositivi mobili: si basano su alimentazione a batteria e tecnologie wireless e solitamente i programmi vengono eseguiti in parte su di essi e in parte su appositi server dedicati al cloud computing.

1.2 Esecuzione di un programma

Il calcolatore è in grado di eseguire solamente istruzioni di basso livello semplici e passare da un'applicazione complessa alle semplici istruzioni è un processo che coinvolge interpretatori e traduttori che dalle operazioni definite ad alto livello ottengono le istruzioni macchina. Questa gerarchia di operazioni viene divisa in tre componenti secondo questa gerarchia:

1. l'hardware: il calcolatore fisico che esegue le istruzioni.
2. Il sistema operativo che gestisce la base dell'input-ouput, alloca spazio nelle memorie e consente il multi-tasking.

3. Le applicazioni.

1.2.1 Da linguaggio ad alto livello a linguaggio macchina

Per comunicare con una macchina elettronica è necessario inviare segnali elettrici (acceso o spento) rappresentati dai numeri 0 e 1, che composti formano sequenze di numeri binari dei quali ogni cifra è detta bit. Un'istruzione è pertanto una stringa di bit. Per semplificare il processo di scrittura venne creata una notazione in grado di tradurre da un linguaggio di più facile comprensione a queste stringhe, in questo modo è il compilatore stesso a programmare il compilatore. Il primo di questi programmi fu chiamato assembler. Se la sequenza di bit viene chiamato linguaggio macchina questa nuova notazione viene chiamata linguaggio assembler. Questa traduzione puntuale rimaneva ancora di difficile comprensione, vennero creati pertanto linguaggi di programmazione ad alto livello, con particolari compilatori (o interpreti) in grado di tradurrre questi linguaggi in linguaggio macchina. Questi ultimi pertmettono un incremento di produttività (condensano più operazioni) e l'indipendenza dal particolare calcolatore sul quale vengono sviluppati. Alcuni compilatori eliminano lo stadio intermedio tra linguaggio di alto livello, assembler e macchina.

1.3 Componenti di un calcolatore

L'hardware di un calcolatore acquisice dati, li elebora e fornisce il risultato. È dotato pertanto di dispositivi di input per la ricezione e di ouptut per l'invio, un'unit à per l'elaborazione dei dati e un'unità di controllo. La parte che esegue le operazioni è detta CPU ed è formata dalle ultime due parti responsabili per le operazioni logico-matematiche e per lo spostamento inerno dei dati. La memoria invece è il luogo in cui venfono slavati i programmi e i loro dati, costituita da CHIP di DRAM (memoria dinamica ad accesso casuale). All'interno del processore è presente una memoria cache piccola ma veloce che agisce da buffer per la memoria più grande, costruita di SRAM. Per permettere l'esecuzione di istruzioni macchina è inoltre presente un'interfaccia tra linguaggio di basso livello e hardware: l'architettura dell'insieme di istruzioni, checontiene tutte le istruzioni che permettono al calcolatore di funzionare, in modo da evitare al programmatore questo lavoro. Questa architettura è indipendente dall'hardware che la implementa, che ne realizza la descrizione. Architettura e sistema operativo costituiscono l'interfaccia binaria delle applicazioni.

1.3.1 Salvare i dati

La memoria utilizzata per memorizzare i programmi in esecuzione viene detta memoria primaria, è volatile e costituita da DRAM. La memoria secondaria, o memoria di massa, salva i dati tra un'esecuzione e un'altra, non volatile. Esistono vari tipi di memoria di massa come gli hard disk, o memorie flash a semiconduttore.

1.3.2 Comunicazione tra calcolatori

Il collegamento alla rete ha permesso la condivisione di dati e risorse tra diversi calcolatori attraverso la tecnologia ethernet per le LAN, a fibra per le WAN e wireless per i dispositivi portatili.

1.3.3 Produrre un chip

Un chip è formato da transistor su di un circuito integrato prodotto da un wafer di silicio che viene più volte mascherato, tagliato e "impacchettato". Il costo di un circuito integrato si può esprimere con il costo per piastrina: $\frac{\text{Costo per wafer}}{\text{Piatrine per wafer} \cdot \text{Rendimento}}$, le piastrine per wafer: $\frac{\text{Superficie del wafer}}{\text{SUperfiie della piastrina}}$ e rendimento: $\frac{1}{1+(\text{Difetti per area} \cdot \frac{\text{Area della piastrina}}{2})}$.

1.4 Le prestazioni

Le prestazioni di un calcolatore dipendono da come viene utilizzato: un utente singolo vorrà migliorare il tempo di esecuzione, mentre il gestore di un centro di calcolo il throughput, ovvero il numero di task nell'unità di tempo. In molti casi sono codipendenti. Tenedo conto del tempo di esiscuzione si avrà che $Prestaziono = \frac{1}{\text{Tempo di esecuzione}_X}$. Per controllare quantitativamente le prestazioni di due calcolatori si farà il loro prodotto. Le prestazioni di un calcolatore si misurano nel tempo totale richiesto ad un calcolatore per completare una task. Siccome i calcolatori lavorano in condizione di condivizione di risorse il sistema potrebbe cercare di massimizzare il troughput mettendo in pausa il programma. Si considererà pertanto anche il tempo di CPU, ovvero il tempo effettivamente speso dalla CPU per risolvere il programma, che può essere a sua volta diviso tra tempo di CPU utente (per svolgere il programma) e in quello di sistema necessario per eseguire le funzioni del sistema operativo. Per rendere più facile predirre il tempo di esecuzione di un programma si esprime la velocità con cui il processore esegue le istruzioni e attraverso il ciclo di clock, un segnale periodico per la sincronizzazione delle funzioni implementate nell'hardware.

1.4.1 Prestazione della CPU

Il tempo di CPU relativo ad un programma si calcola come il prodotto tra i cicli di clock relativi ad esso e il periodo di clock, o il primo diviso la frequenza di clock.

1.4.2 Prestazione delle istruzioni

Il tempo di esecuzione di un programma dipende dal numero di istruzioni che il calcolatore dovrà eseguire. Il tempo di esecuzione totale può perciò venire espresso come il prodotto tra il numero di istruzioni da eseguire e il tempo medio di esecuzione di ciascuna istruzione. Il numero di cicli di clock per eseguire il

programma sarà il numoer di istruzioni del programma per il numero medio di cicli di clock per istruzione (CPI).

1.4.3 Misura delle prestazioni

Si può pertanto esprimere il tempo di CPU come:

$$\text{Tempo di CPU} = \frac{\text{Numero di istruzioni} \cdot \text{CPI}}{\text{Frequenza di clock}} \tag{1.1}$$

Che evidenziano i tre fattori che influenzano le prestazioni. Essendo che il CPI varia da programma a programma risulta sempre difficile determinare le prestazioni in maniera univoca.

1.5 La barriera dell'energia

L'aumento della frequenza di clock e della potenza elettrica assorbita sono aumentate di pari passo fino a che è diventato impossibile dissipare il calore generato dalla CPU. A questo punto si è ridotta riducendo la tensione di alimentazione, processo a cui si è arrivato ad un limite in quanto diminuendola si aumenta la dispersione di corrente, rendendolo pertanto svantaggioso. L'assorbimento di potenza si è pertanto rivelato una barriera invalicabile. Essendo infatti la potenza richiesta metà del prodotto tra il carico capacitivo, il quadrato della tensione e la frequenza di commutazione.

1.6 Sisitemi multiprocessore

Per migliorare le prestazioni si è passati a creare microprocessori contenenti più processori o core. Questo procedimento presenta problemi nel suo utilizzo in quanto ora è necessario bilangiare il carico di lavoro tra tutti i core e ridurre il loro tempo di comunicazione e sincronizzazione.

Capitolo 2

Aritmetica dei calcolatori

2.1 La codifica

Una sequenza di zeri e uni viene utilizzata per rappresentare numeri, caratteri, programmi, immagini e suoni. Questi vengono distinti e riconosciuti attraverso una codifica.

2.1.1 Codifica dei naturali

Viene utilizzata la base due per rappresentare i numeri da 0 a 2^{k-1}. La sua conversione in base 16 consiste nel prendere gruppi di quattro bit e trasformarli nella cifra corrispondente in base sedici e viceversa. Per convertire da base 10 si fa la divisione intera e il resto è la cifra da porre a sinistra del numero. Per moltiplicare per una potenza n di due faccio lo shift a sinistra di n cifre.

2.1.2 Codifica degli interi

Per codificare i numeri negativi possono venire utilizzate le codifiche:

- Modulo e segno.
- Complemento a 1.
- Complemento a 2.

2.1.2.1 Modulo e segno

Il bit più significativo viene utilizzato per codificare il segno 1 se negativo. Pone dei problemi di efficienza e presenta più zeri.

2.1.2.2 Complemento a 1

Questa codifica si ottiene invertendo gli zeri con gli uni e viceversa. Contiene comunque due zeri, ma gli algoritmi di somma sono più veloci: facco la somma bit a bit e sommo il riporto della cifra più significativa. Il risultato è attendibile se i riporti delle ultime due cifre sono uguali.

2.1.2.3 Complemento a 2

Per ottenere il complemento a due scorro il numero dal bit meno significativo e comincio a invertire il valore dopo aver incontrato il primo 1, o sommando 1 al complemento a 1. In questo modo si ottiene una codifica unica dello zero. L'overflow si verifica quando l'ultimo bit di segno è in disaccordo con il risultato dell'operazione.

2.1.3 Codifica dei reali

2.1.3.1 Virgola fissa

Si pone un punto in cui i bit meno significativi rappresentano la parte decimale. Per cambiare di base la parte decimale moltiplico ricorsivamente per due per quante cifre contiene la parte decimale e considero la parte intera.

2.1.3.2 Virgola mobile

Un numero reale può essere rappresentato, analogalmente alla notazione scientifica come un numero compreso tra 1 e 2 moltiplicato per un esponente della sua base. Con questa codifica il bit più significativo rappresenta il segno, 8 bit sono dedicati all'esponente in complemento a 2 e i restanti dedicati alla mantissa.

Capitolo 3

Le istruzioni MIPS

Le parole del linguaggio del calcolatore sono dette istruzioni e l'intero vocabolario isieme delle istruzioni, interpretati dall'alto al basso. La semplicità dei dispositivi è un parametro di grande valore per i caclolatori. Un programma è un insieme di istruzioni che viene memorizzato come numeri interpretati dalla macchina come istruzioni. Le operazioni aritmetiche contengono sempre tre elementi, il primo il luogo in cui viene salvato il risultato e gli altri due i due elementi. Questa descrizione permette di mantenere l'hardware semplice.

3.1 Gli operando dell'hardware del calcolatore

Le istruzioni aritmetiche del MIPS possono essere svolte tra un numero limitato di locazioni particolari: i registri, che rappresentano le primitive utilizzate nella prograttazione dell'hardware: i registri. Nei calcolatori della classe dei MIPS sono presenti 32 registri in quanto minori dimensioni implicano maggiore velocità. Per questo i tre operandi delle istruzioni aritmetiche devono essere scelti tra i 32 registri a 32 bit. Strutture dati più complesse possono contenere un numero di elementi maggiore degli elementi presenti nel calcolatore e di conseguenza vengono allocate in memoria. Si rendono pertanto necessari delle istruzioni di trasfermiento che interagiscano tra la memoria e i registri. Per questo l'istruzione deve contenere l'indirizzo di memoria corrispondente al dato. La memoria può essere considerata un vettore monodimensionale. Attraverso l'istruzione di load viene caricato un valore da memoria a registro. L'indirizzo del dato in memoria vene ottenuto dalla somma della costanto e del contenuto del secondo registro. L'indirizzo di due parole consecutive differisce di quattro unità. L'operazione complementare, che dal registro salva in memoria si chiama store ed ha sintassi simile. I registri richiedono un tempo di accesso minore rispetto alla memoria e pertanto aumentano il throughput. Per evitare l'operazione di load una versione delle istruzioni aritmetiche permette di fare un'operazione tra un registro e una costante. L'operazione di spostamento di due registri richiede la

somma con la costante zero, che per questo viene sempre contenuta nel registro *$zero*.

3.2 Numeri con e senza segno

I numeri vengono rappresentati in base 2 con una sequenza di segnali elettrici alti o bassi e il valore della i-esima cifra d è $d \cdot mBase^i$, dove i assume un valore da 0 e viene incrementato spostandosi verso sinistra. Queste combinazioni di bit sono rappresentazioni di numeri. È possibile progettare un circuito logico che svolga le operazioni aritmetiche con questi numeri, se il risultato di questa operazione supera i bit disponibili avviene un overflow.

3.2.1 Numeri con segno

Una rappresentazione dei numeri con segno è data dalla rappresentazione modulo e segno, ovvero un bit, solitamente l'ultimo viene riservato per il segno: 0 se positivo 1 se negativo. Presenta degli svantaggi di efficienza e contiene due zeri. Viene pertanto utilizzata la rappresentazione in complemento a 2, ovvero la metà dei numeri da 0 a $2^{31}-1$ utilizza la rappresentazione precedente, mentre da quel numero in poi sono rappresentati i numeri negativi crescenti. Pertanto i numeri positivi presentano degli zeri iniziali, mentre i negativi degli uni e si ha un solo zero. Il bit di segno, ovvero quello più significativo, viene moltiplicato per -2^{31}. La condizione di overflow si verifica quando il bit più a sinistra non è uguale agli infniti bit alla sua sinistra, ovvero quando il bit di segno non è corretto. Il bit di segno viene ripetuto fino a riempire tutti i bit disponibili. Pertanto si utilizza *lb* per caricare un numero con segno e *lbu* per un byte con un numero senza segno. Per invertire il segno di un numero si invertono tutti i bit e si somma poi 1. Per estendere il segno si riempie a sinistra del valore del bit di segno. Con il complemento a uno non si aggiunge a uno al numero quando si cambia di segno.

3.3 Le istruzioni nel calcolatore

Le istruzioni del calcolatore sono memorizzate come una sequenza di segnali che possono pertanto essere rappresentate come numeri. Dato che i registri vengono indirizzati dalle istruzioni si deve assegnare un numero ad ogni registro.

Capitolo 4

Assembly intel

Si basa su un'architettura CISC

4.1 Gestione dei registri

Per indicare un registro si prepone %, per un valore immediato $.Intel posside 16 registri a 64 bit general purpose: %rax, %rbx, %rcx, %rdx, %rsi, %rdi, %r8, %r9, %r10, %r11, %r12, %r13, %r14, %r15, %rbp il frame pointer e %rsp lo stack pointer. Per accedere ai 32 bit meno significativi si sostituisce r con e, per accedere ai 16 meno significativi si omette r, di questi 16 per accedere agli 8 più significatisi si prepone h, ai meno significativi l. Si trovano inoltre due registri specializzati: %rip instrucion pointer (program counter) e %rflags, ovvero il registro per i flags.

4.2 Convezioni di chiamata

Il passaggio di parametri di una funzione è considerato da 6 registri: %rsi, %rdi, %rcx, %rdx, %r8, %r9, mentre ulteriori argomenti possono essere salvati sullo stack. I valori di ritorno vengono salvati nei registri %rax e %rdx. I registri da preservare sono: %rbp, %rbx, %r12, %r13, %r14, %r15. Esistono due modalità per richiamare una funzione attraverso l'istruzione call: call address che chiama la subroutine all'indirizzo indicato e call *register che chiama la funzione all'indirizzo contenuto nel registro indicato. Il link register viene salvato automaticamente sullo stack, facilitando le funzioni ricorsive. L'istruzione ret preleva dallo stack l'indirizzo di ritorno da utilizzare nell'instrucion pointer.

4.3 Modalità di indirizzamento

Esistono varie modalità di indirizzamento in Intel:

- <displacement>

<displacement> (<base register>)

<displacement> (<index register>, [<scale>])

<displacement> (<index register>, <base register>, [<scale>])

L'indirizzo di memoria corrispondente viene calcolato come <displacement> + <base> + <index> · <scale>. Dove displacement è una costante a 8, 16, 32 o 64 bit, index e base sono indirizzi e scale è 1, 2, 4 o 8.

4.4 Sintassi istruzioni

Le istruzioni si presentano nella forma <opcode>[<size>]<source>, <destination>, dove il secondo elemento è sia operando che destinazione e deve essere un registro o un indirizzo di memoria. Il primo può essere un valore immediato. I due operandi non possono essere contemporaneamente indirizzi in memoria e non è possibile specificare due operandi e una destinazione diversa. <size> viene utilizzato per determinare la grandezza degli operandi: b per 8 bit, w per 16, l per 32, q per 64. È opzionale quando uno dei due operandi è un registro, obbligatorio altrimenti.

4.5 Istruzioni frequenti

A seguire un elenco delle istruzioni aritmetico-logiche più frequenti:

- mov per scambiare dati fra memoria e registri e viceversa (la destinazione non può essere un valore immediato);
- push e pop per leggere e salvare dati sullo stack, senza dover modificare lo stack pointer come avviene in MIPS;
- add (somma) e addc (somma con carry);
- sub (sottrazione) e subc (sottrazione con carry);
- mul (moltiplicazione con segno) e imul (moltiplicazione senza segno);
- div (divisione con segno) e idiv (divisione senza segno);
- inc (somma 1) e dec (sottrae 1);
- and, or, xor e not: operazioni logiche bit a bit;
- lea: load effective address;
- rcl, rcr, rol, ror: varie forme di rotate;
- sal, sar, shl, shr: shift aritmetico e logico;

- jmp: salto incondizionato;
- je (jump if equal), jnz (jump if not zero), jc (jump if carry), jnc (jump if not carry);
- neg;
- cmp: setta i flag facendo una sottrazione, ma senza salvarne il risultato;
- call e ret: indirizzo di ritorno sullo stack;
- nop;
- eventuali istruzioni condizionali.

4.5.0.1 Load effective address

Nonoestante questa operazione sia stata implementata per calcolare indirizzi con indirizzamento diverso senza effettuare accessi, risulta utile per effettuare la somma tra due registri e salvare il risultato in un terzo: lea (%rax, %rbx), %rcx. Nonostante la sintassi sia simile a quella di un indirizzamento non lo si sta effettuando.

4.5.0.2 Incremento e decremento

Utilizzate invece di add in quanto permettono di risparimiare 16 bit sull'operazione.

Capitolo 5

Toolchain

La toolchain è lo strumento necessario a tradurre il codice scritto in linguaggio di alto livello in linguaggio macchina: per esempio nel C ci sono quattro componenti che permettono questa traduzione:

1. Preprocessore: gestisce le direttive # (come include o define) sostituendo pezzi di codice ed eliminando i commenti.
2. Compilatore: traduce il codice da C ad Assembly (.s).
3. Assembler: traduce da assembly a linguaggio macchina (.o).
4. Linker: collega tra loro diversi file .o per linkare al file oggetto librerie o altri file .o e produce un eseguibile.

L'eseguibile successivamente viene chiamato in memoria attraverso una system call per essere eseguito.

5.1 Da codice sorgente a eseguibile

5.1.1 Da codice sorgente a file oggetto

Quando viene compilato il codice sorgente e trasformato in assembly questo viene ottimizzato e semplificato in modo da essere più performante e leggero. Successivamente il file assembly viene trasformato in un file oggetto. Durante questo passaggio le pseudo istruzioni vengono convertite in istruzioni standard, le istruzioni assembly in linguaggio macchina, tutti i numeri in binario le label tradotte in indirizzi, vengono gestiti i salti (se l'indirizzo non può essere contenuto dal campo j si passa ad un jr), vengono creati i metadati, le informazioni di alto livello che serviranno al loader per caricare il codice binario. Dopo aver completato queste operazioni il file oggetto contiene:

- Header: contiene dati come le posizioni degli altri dati all'interno del file.

- Segmenti: contiene codice e dati come le variabili globali.
- Tabella di rilocazione: contiene tutti i simboli con indirizzo relativo alla posizione nell'area .data e tipo di istruzione.
- Tabella dei simboli: contiene tutti i simboli undefined non contenuti nel file oggetto e simboli definiti con indirizzo assoluto.
- Altre informazioni come per esempio per il debugging.

5.1.2 Da file oggetto a eseguibile

Durante questa traduzione viene deciso come codice e dati sono disposti in memoria compattando tra loro le sezioni con funzioni uguali, vengono associati indirizzi assoluti a tutti i simboli anche non locali, patcha le istruzioni di salto dopo che gli indirizzi sono stati modificati durante la prima fase della traduzione. Vengono pertanto eliminate le tabelle dei simboli e di rilocazione sostituendo tutti gli indirizzi con indirizzi assoluti. Per fare ciò si potrebbe necessitare di collegare tra loro più file .o. Durante questo processo si trovano tre tipi di simboli:

1. Locali definiti e visibili solamente all'interno del file.
2. Definiti associati ad un indirizzo relativo nella tabella dei simboli ma definiti all'interno del file stesso.
3. Non definiti presenti nella tabella dei simboli ma definiti in un file diverso.

5.1.2.1 Linking

Durante il linking vengono disposti in memoria i vari segmenti riordinandoli, ovvero prendendo tutte le parti .text nei file .o linkati e unificandole e con tutte le altre sezioni, in modo da avere un unico file che contiene tutto il codice necessario e che sia strutturalmente ordinato. Successivamente si assegna un indirizzo assoluto ad ogni simbolo presente nella tabella dei simboli, facendo attenzione alla modifica degli indirizzi relativi che avviene nel primo passaggio. Alla fine si modificano tutte le istruzioni con gli indirizzi appena calcolati, sistemando tutti i simboli nella tabella di rilocazione. Il file risultante viene incapsulato in un eseguibile che conterrà i vari segmenti, informazioni per il caricamento in memoria e informazioni aggiuntive per il debugging.

5.2 Librerie

Data la complessità dei programmi si è reso necessario l'uso di librerie, collezioni di file .o per funzioni già create. Si dividono in:

- Librerie statiche (.a): collezioni di file .o. Il linker linka l'intera libreria. Dispendioso dal punto di vista della memoria, ma la fase di esecuzione risulta semplice.

- Librerie dinamiche (.so): il linking avviene durante il caricamento ed esecuzione: la libreria può essere consultata a runtime. L'eseguibile rimane leggero, ma presenta delle difficoltà di implementazione.

5.2.1 Librerie dinamiche

Al momento del linking non viene linkata la libreria, ma solo dei riferimenti ad essa, e un riferimento ad un linker dinamico che viene caricato ed eseguito al momento dell'esecuzione passandogli come argomento il programma. Sarà tale linker a linkare le librerie quando necessario. Oltre ad un eseguibile leggero permette di modularizzare il codice in modo che la ricompilazione non sia necessaria in caso di aggiornamento di una libreria. Il processo di caricamento diventa però complesso. Si può effettuare un lazy linking, in cui il linking viene effettuato solo quando strettamente necessario. In questa strategia invece dell'indirizzo della libreria viene inserito uno stub che solo se invocato va a linkare la libreria.

Capitolo 6

Il processore

Si descriverà la struttura di un processore facendo riferimento a un set di istruzioni ridotto: accesso a memoria, aritmetico logiche e di salto. Le prime due fasi dell'esecuzione sono comuni a tutte le istruzioni e sono:

- Prelievo dell'istruzione da memoria.
- Lettura del valore di uno o più registri operandi estratti direttamente dai campi dell'istruzione.

Gli altri passi sono simili: tutte le istruzioni a parte quella di jump incondizionato utilizzano la ALU dopo aver letto gli operandi, o per calcolare l'indirizzo nel caso di istruzioni di accesso a memoria, il calcolo del risultato nel caso di operazioni logico-aritmetiche o per il calcolo del confronto nel caso delle operazioni di salto condizionato. Dopo l'utilizzo dell'ALU le operazioni si differenziano: le istruzioni di accesso a memoria richiedono o salvano il dato in memoria, le operazioni logico aritmetiche salvano il risultato nel registro target, le operazioni di salto condizionato cambiano il valore del program counter.

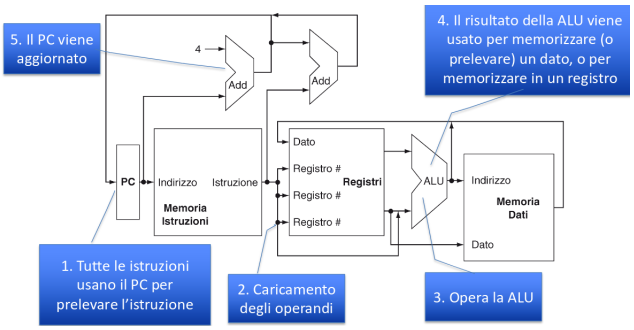

Figura 6.1: Un processore base

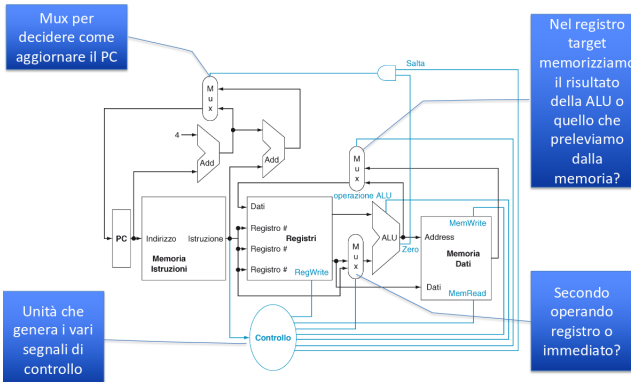

Figura 6.2: Un processore più dettagliato

La figura precedente è incompleta in quanto i dati arrivano da diverse sorgenti dalle quali bisogna scegliere, come per esempio per l'incremento del PC o la differenza tra istruzioni di tipo R o di tipo I, in cui un dato può provenire da un registro o essere contenuto nell'istruzione. Per selezionare quale delle operazioni svolgere viene utilizzato un multiplexer, che sulla base di un ingresso di controllo sceglie quale degli input debba finire in un output. Le linee di controllo del multiplexer vengono impostate sulla base del tipo di istruzioni. I vari gruppi funzionali hanno ulteriori ingressi di controllo: la ALU per decidere quale operazione effettuare, il banco registri ha un ingresso per decidere se scrivere in un ingresso o meno, la memoria dati ha degli ingressi per decidere se si vuole svolgere un'operazione di lettura o scrittura. La decisione dell'impiego degli ingressi di controllo è associata ad un'ulteriore unità funzionale.

Si può fare un'assunzione semplificativa dicendo che il processore lavora sincronizzandosi con i cicli di clock, e si assuma inoltre che tutte le operazioni si svolgano in un unico ciclo abbastanza lungo.

6.1 Temporizzazione

Questa metodologia esplicita quando i segnali possono essere letti o scritti in relazione al clock. Quella più utilizzata è quella sensibile ai fronti, in cui il dato viene memorizzato in corrispondenza della salita o della discesa del fronte di clock. I dati presi dagli elementi di stato sono relativi al ciclo precedente. Il tempo di clock deve essere scelto in modo da permettere ai dati di attraversare la rete combinatoria. Questa metodologia permette di rendere determinate operazioni che senza di essa sarebbero indecidibili.

6.2 realizzazione del datapath

Si passino in rassegna le componenti necessarie per realizzare un datapath:

- Una memoria istruzioni, dove sono salvate le istruzioni da eseguire.
- Il program counter, l'indirizzo dell'istruzione da eseguire.
- Il sommatore, un ALU specializzata a incrementare il PC.

Questi elementi permettono il prelievo dell'istruzione:

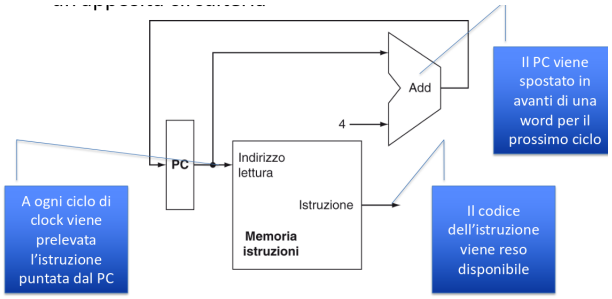

Figura 6.3: Il processo di prelievo di un'istruzione

6.2.1 Istruzioni di tipo R

Le operazioni di tipo R sono istruzioni aritmetico-logiche che operano tra registri e producono un risultato in un altro registro. Per operare queste operazioni si necessita di altri due blocchi funzionali:

- Il banco registri, che dà in output i registri specificati dell'istruzione e se abilitato in scrittura (regWrite) 0da un multiplexer scrive nel registro specificato il dato in ingresso.
- Una ALU che effettua l'operazione specificata da una codifica a 4 bit, setta un bit in uscita se il risultato è zero.

6.2.2 Istruzioni load store

Per entrambe si deve calcolare un indirizzo di memoria dato dalla somma di un registro con un offset e nel caso di un'operazione di scrittura leggere il registro dal register file. Si necessiterà ancora pertanto di ALU e register file. Si noti che essendo l'offset memorizzato in un campo a 16 bit occorrerà un'unità funzionale in grado di estendere il segno a 32 bit. Dovrà essere inoltre essere aggiunta un'unità di memoria dati da dove leggere e salvare i dati. Verrà utilizzata in scrittora (MemWrite) o in lettura (MemRead) in base al segnale dato dall'apposito multiplexer.

6.2.3 Salto condizionato

Per compiere un salto condizionato si necessita di sommare un offset che dovrà essere esteso a 32 bit al PC. Siccome quest ultimo verrà sempre automaticamente aumentato di 4 verrà aggiunto al PC già aumentato. L'offset inoltre viene automaticamente traslato di due bit in modo da esprimerlo come word e non come byte e estendento così l'intervallo degli offset raggiungibile. Occorre pertanto un ulteriore multiplexer che decida se operare sul PC un'operazione di Pc+4 o PC+4+offset. Per il salto incondizionato basta sostituire il campo offset shiftato di 4 al PC.

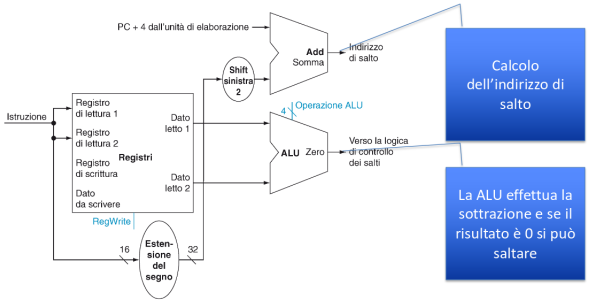

Figura 6.4: Il processo di Salto Condizionato

6.2.4 Progetto di un'unità di elaborazione

Siccome ogni unità funzionale può essere utilizzata un'unica volta per ogni ciclo di clock, si necessiterà di distinguere tra memoria dati e memoria istruzioni. Occorre inoltre condividere il più possibile le varie unità.

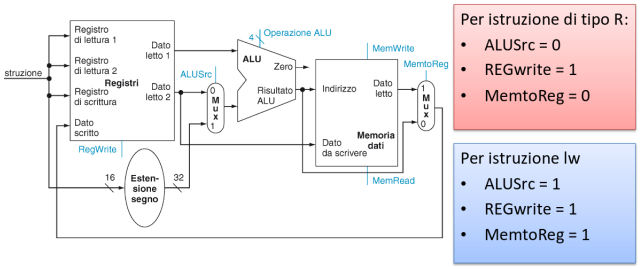

Figura 6.5: Circuito per istruzioni R e di trasferimento in memoria

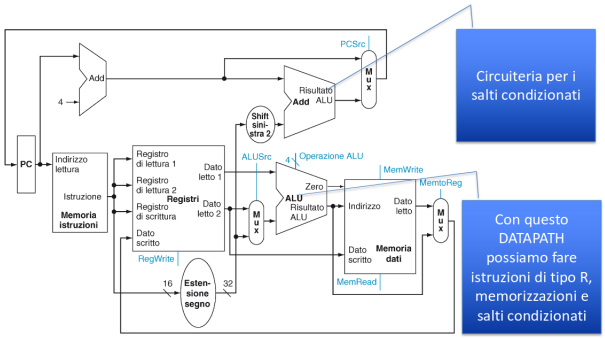

Figura 6.6: Circuito per istruzioni R e di trasferimento in memoria dettagliato

6.2.5 Prima implementazione comleta

Per arrivare a una prima implementazione completa si parta dal datapath mostrato e si aggiunga la parte di controllo, implementando le istruzioni add, sub, and, or, slt, lw, sw, beq e successivamente il jump.

6.2.5.1 ALU

La ALU viene utilizzata per realizzare operazioni logico aritmentiche di tipo R e slt, calcolare indirizzi di memoria e la sottrazione per beq. Per queste operazioni si ha una diversa configurazione degli input di controllo (a 4 bit), si veda la tabella sulle slide. Per generarli si utilizza un'unità di controllo che prende in ingresso il campo funct prelevato dall'istruzuione e due bit detti ALUop, ovvero 00 per sw e lw, 01 per beq, 10 per operazioni di tipo R specificate dal funct. Si genera pertanto un comando attraverso due livelli: il primo genera i segnali di controllo ALUop per l'unità di controllo della ALU, il secondo genera i segnali di controllo per la ALU. I segnali di controllo della ALU sono generati da una rete logica combinatoria e per evitare di elencare tutte le combinazioni di ingresso ALUop e dei campi funct vengono compressi attraverso l'utilizzo della wildcard X. Si vedano le tabelle delle slides.

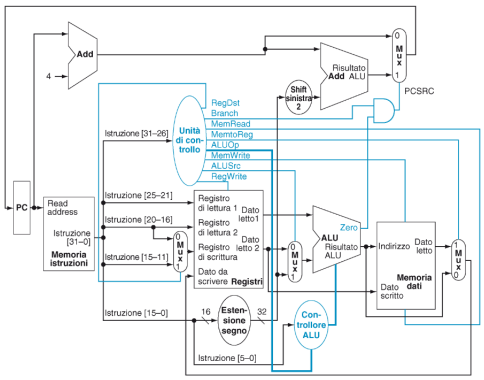

Figura 6.7: Una CPU molto dettagliata

Si faccia riferimento alle slides per una spiegazione dei segnali di controllo.

6.2.5.2 Unità di controllo

È un'unità funzionale combinatoria che prende come input il codice operativo dell'istruzione e genera i comandi del caso

6.2.5.3 Salto incondizionato

In questa operazione il PC viene incrementato di 4, i suoi quattro bit significativi rimangono uguali, dal 27 al 2 sostituiti con il campo indirizzo dell'istruzione i due bit meno significativi settati a 0.

6.3 Conclusione

Le istruzioni raramente vengono concluse in un unico ciclo in quanto sono le più lente a dettarlo e non si riesce a fare ottimizzazioni aggressive sulle cose più frequenti.

Capitolo 7

La pipeline

Si è notato come compiere un'operazione per ciclo di clock è inefficiente, pertanto si utilizza una pipeline. Si noti come le istruzioni del MIPS hanno 5 fasi di esecuzione: prelievo dell'istruzione dalla memoria, lettura dei registri e decodifica dell'istruzione, esecuzione di un'istruzione o calcolo di un indirizzo, accesso a un operando nella memoria dati, scrittura del risultato in un registro, verrà utilizzata pertanto una pipeline a 5 stadi. In una pipeline quando un istruzione termina l'utilizzo di un elmento funzionale questo viene subito occupato dall'istruzione successiva.

7.0.0.1 Confronto di prestazione

Il confronto può essere fatto come tempo fra due istruzioni=$\frac{Tempo\ tra\ due\ istruzioni\ senza\ pipeline}{numero\ stadi\ della\ pipeline}$.

7.1 Vantaggi del RISC

1. Essendo tutte le istruzioni della stessa lunghezza il prelievo è più facile.
2. Essendo i codici degli operandi in posizione fissa ci si può accedere leggendo il register file in parallelo con la decodifica dell'istruzione.
3. Gli operandi residienti in memoria sono utilizzabili solo da sw e lw, pertanto si può utilizzare la ALU per calcolare gli indirizzi. .
4. L'uso di accessi allineati permette agli accessi in memoria di avvenire in un solo ciclo impegnando un solo stadio della pipeline.

7.2 Hazard

In condizioni normali la pipeline permette di eseguire un'operazione per ciclo di clock, ma ci sono dei casi in cui questo non è possibile per il verificarsi di condizioni critiche.

7.2.1 Hazard strutturali

L'architettura del calcolatore rende impossibile l'esecuzione di alcune istruzioni in pipeline, ad esempio, disponendo di un'unica memoria non è possibile caricare istruzioni e prelevare operandi nello stesso ciclo di clok.

7.2.2 Hazard sui dati

Si verifica quando la pipeline deve essere messa in stallo per ottenere informazioni dagli stadi precedenti, come somme che utilizzando gli stessi operandi. Questo tipo di hazard blocca la pipeline per tre cicli di clock.

7.2.2.1 Possibili soluzioni

- In certi casi si può eliminare il problema a livello di compilatore invertendo delle istruzioni.
- È utile osservare come non è necessario salvare il risultato, attraverso un operazione di operand forwarding o propagazione, in cui il risultato viene reso disponibile bypassando l'operazione di storage.

7.2.3 Hazard sul controllo

Si verifica in presenza di salti condizionati, supponendo di avere un circuito sofisticato che permette di calcolare l'indirizzo di salto si necessita comunque di saltare uno o due cicli. Se la pipeline è troppo lunga questo stallo diventa troppo costoso. Vengono pertanto implementati dei circuiti che prevedano dei salti e fanno delle assunzioni su di essi, mettendo le operazioni nella pipeline in base ad esse e poi, se necessario si corregge. Si può considerare che il salto non venga utilizzato, o che il comportamento rimanga costante per esempio.

Capitolo 8

Le memorie

La memoria è fondamentale per il funzionamento di un compilatore. È necessario poterci leggere e scrivere dati. La memoria indirizzata direttamente (memoria principale o cache) è volatile ed è limitata dallo spazio di indirizzamento dal compilatore. Esiste una memoria indirizzata indirettamente o periferica, permanente con uno spazio di indirizzamento software non limitato dal processore. Le informazioni sulla memoria principale sono disponibili al processore in qualsiasi momento, mentre le informazioni nella memoria periferica devono essere prima trasferite a quella principale. Questo trasferimento è tipicamente mediato dal sistema operativo.

- Si intende per tempo di accesso il tempo di un'operazione di lettura o scrittura nella memoria.
- Il tempo di ciclo è il tempo che intercorre tra l'inizio di due operazioni tra locazioni diverse.
- Accesso casuale: non vi è alcuna relazione o ordine tra i dati salvati, tipico delle memorie a semiconduttori.
- Accesso sequenziale: l'accesso alla memoria è ordinato o semi ordinato, il tempo di accesso dipende dalla posizione, tipico dei dischi o dei nastri.
- RAM (random access memory) memoria scrivibile leggibile a semiconduttori, con tempo di accesso indipendente dalla posizione del dato.
- ROM (read only memory) memoria a semiconduttori in sola lettura può avere accesso casuale o sequenziale.

8.1 Memorie RAM a semiconduttori

Queste memorie memorizzano singoli bit spesso organizzati in byte e o word. Data una capacità N la memoria può essere organizzata in diversi modi a seconda

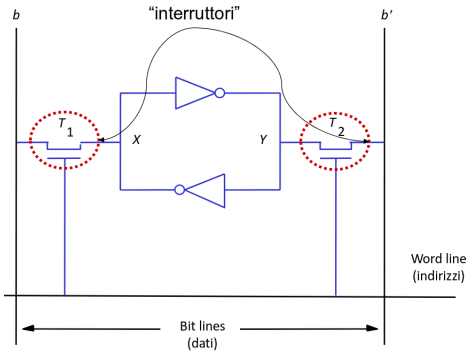

Figura 8.1: Un bit in SRAM

del parallelismo, influenzando il numero di pin I/O necessari al circuito integrato che la implementa.

8.1.1 Memorie statiche SRAM

Sono memorie in cui i bit possono essere salvati indefinitamente fintanto che rimangono alimentate, sono estremamente veloci e consumano poca corrente, ma sono care in quanto richiedono molte componenti per ciascuna cella di memorizzazione. Si consideri che $b = NOT(b')$, i circuiti di terminazione della linea di bit (sense/write circuit) interfacciano il mondo esterno che non accede mai direttamente alle celle, la loro presenza contemporanea riduce gli errori. In caso di scrittura la linea di word è alta e chiude T_1 e T_2, il valore presente su b e b', linee di pilotaggio viene salvato nel latch a doppio NOT. In caso di lettura la linea di word è alta e chiude T_1 e T_2, il le linee b e b' sono tenute in stato di alta impedenza e il valore in X e Y viene copiato in b e b'. Se la linea di word è bassa il consumo è nullo.

8.1.2 Memorie DRAM

Sono le memorie più diffuse in quanto economiche e a densità elevata, in quanto la memoria viene ottenuta sotto forma di carica di un condensatore. Hanno bisogno però di un costante refresh altrimenti la carica si disperderebbe a causa di correnti parassite. In caso di operazione di scrittura la linea di word è alta e chiude T e il valore di b viene copiato su C. In caso di lettura la linea di word chiude T e si utilizza un apposito circuito che se la tensione di C è sopra una certa soglia pilota la linea b alla tensione di alimentazione ricaricando C, altrimenti mette b a terra scaricando C.

8.1.2.1 Refresh

Nel momento in cui T è aperto il condensatore comincia a caricarsi o scaricarsi a causa di correnti parassite sul semiconduttore e si rende necessario quindi

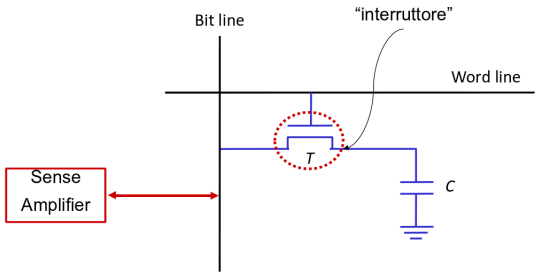

Figura 8.2: Un bit in DRAM

periodcamente rinfrescare i dati attraverso un'operazione di lettura. In genere la memoria contiene un circuito per la lettura periodica della memoria, pertanto l'utente non se ne deve preoccupare.

8.1.2.2 Multiplazione degli indirizzi

Data l'elevata integrazione delle DRAM il numero di pin I/O diventa un problema, pertanto è usuale multiplare nel tempo l'indirizzo delle righe e delle colonne negli stessi fili. Le memorie non sono indirizzabili al bit, per cui righe e colonne si riferiscono a byte e non a bit.

8.1.2.3 Modo di accesso veloce

L'accesso ai dati della memoria DRAM avviene da blocchi o pagine di memoria: l'indirizzo viene separato in row address selector e column address selector, il primo dei quali estrae una riga dalla pagina e il secondo la colonna dalla riga. È possibile, attraverso il fast page mode (FPM) evitare di riselezionare la riga ad ogni accesso se le posizioni sono consecutive con aumenti delle prestazioni significativi.

8.1.3 Memorie DRAM sincrone SDRAM

Le DRAM viste prima sono dette asincrone in quanto non esiste una precisa temporizzazione di accesso, ma la dinamica viene governata dai RAS e dai CAS. Il processore deve tenerne conto in quanto può generare problemi in fase di refresh. Aggiungendo dei buffer di memorizzazione degli ingressi e delle uscite si può ottenere un funzionamento sincrono disaccoppiando lettura e scrittura dal refresh, ottenendo automaticamente un FPM pilotato dal clock.

8.1.4 Double data rate SDRAM: DDR-SDRAM

Una SDRAM che consente il trasferimento dei file sia sul fronte positivo che quello negativo del clock. Latenza uguale a una DRAM normale ma banda

doppia, le memorie sono separate in due banchi uno per le locazioni pari sul fronte positivo e l'altro per le locazioni dispari sul fronte negativo. Le locazioni contigue sono in banchi separati pertanto si può fare un accesso interlacciato.

8.2 Velocità e prestazione

8.2.0.1 Latenza

Il tempo di accesso ad una singola parola, dà un indicazione al tempo che un processore dovrà aspettare un dato dalla memoria nel caso peggiore.

8.2.0.2 Velocità o banda

Indica la velocità di trasferimento massima in FPM, importante per operazioni in FPM che sono legate all'uso di memorie cache interne ai processori.

8.3 Gerarchie di memoria

Si può ottimizzare l'utilizzo di memorie in modo da dare l'impressione di avere una grande memoria veloce.

8.3.0.1 Località temporale

Quando si fa uso di una locazione è molto probabile che verrà riutilizzata presto.

8.3.0.2 Località spaziale

Quando si fa riferimento ad una locazione è molto probabile che nelle operazioni successive si farà riferimento alle locazioni successive.

8.3.1 Struttura della gerarchia

Costi e velocità delle memorie creano una propensione a utilizzare delle piccole e veloci vicino al processore che si ingrandiscono e rallentano allontanandosi da esso.

8.3.1.1 Terminologia

- Si indica con blocco l'unità minima di informazione che può essere presente o assente in un livello.
- Hit rate: la frequenza di accesso, la frazione degli accessi in cui trovo il blocco nel livello superiore.
- Miss rate: il complementare dell'hit rate.

- Tempo di hit: il tempo che occorre per trovare il dato quando lo trovo nel livello superiore.
- Penalità di miss: il tempo necessario per accedere al dato se non lo trovo nel livello superiore

La penalità di miss è molto maggiore del tempo di hit e del trasferimento in memoria di un singolo dato, pertanto è cruciale che non avvenga troppo spesso.

8.3.2 Cache

Un posto sicuro, o nascosto dove riporre i dati, nascosto in quanto è impossibile accedervi direttamente. Per capire se un dato è nella cache o no si può utilizzare una cache a mappatura diretta, ovvero a ogni indirizzo della memoria corrisponde una precisa locazione della cache, l'indirizzo di locazione dove un indirizzo è mappato è l'indirizzo del blocco modulo numero di blocchi nella cache. Se il numero di elementi della cache è a potenza di due è sufficiente prendere i bit meno significativi dell'indirizzo in numero pari al logaritmo in base due della potenza della cache. Siccome molte parole possono essere mappate sullo stesso blocco di cache per capire se in un dato momento vi si trova l'indirizzo ricercato si ricorre ad un campo, detto tag, che contiene informazione sufficiente a risalire al blocco correntemente salvato in memoria, si possono utilizzare per esempio i bit più significativi di una parola per trovare la locazione in cache dove l'indirizzo è mappato. Vengono utilizzati i bit più significativi per capire se nel blocco di cache viene memorizzato l'elemento richiesto, inoltre c'è un blocco di validità che dice se quello che viene memorizzato in un blocco di cache in un dato momento è quello richiesto.

8.3.2.1 Prestazioni

Blocchi di cache molto grande esaltano la località spaziale e diminuiscono la probabilità di miss. Tuttavia a parità di grandezza della cache blocchi molto grandi diminuisce l'efficacia della località temporale, oltre a generare dei miss con costo elevato. Si necessita pertanto di dover trovare un equilibrio.

8.3.2.2 Gestione delle miss

La presenza di una cache modifica la gestione della pipeline solo nel caso in cui ci sono delle miss, quando bisogna generare uno stallo nella pipeline per gestire il trasferimento da memoria principale a cache. Per una miss sulla memoria istruzioni bisognerà inviare alla memoria il valora PC-4, eseguire una lettura, scriverne il risultato aggiornando il tag e far ripartire la fetch. Gli accessi in lettura alla memoria dati avvengono alla stessa maniera.

8.3.2.3 Miss in scrittura

Gli accessi in scrittura sono delicati in quanto possono generare delle inconsistenze. Possono venire implementate due politiche:

1. Write-through, in cui ogni scrittura viene fatta direttamente in memoria principale, eliminando i problemi di consistenza ma aumentando i costi, si può impiegare un buffer di scrittura, una coda con tutte le scritture in attesa di essere completate.

2. Write-back, in cui se il blocco è in cache le scritture avvengono localmente in cache e l'update viene fatto solo quando il blocco viene rimpiazzato. Conveniente in presenza di numerose scritture.

8.3.2.4 Cache completamente associativa

In questo tipo di cache si può mappare qualsiasi tipo di blocco in qualsiasi blocco di cache, il loro problema è che rendono necesario cercare ovunque il dato, per essere efficiente su tutti i blocchi in parallelo, si necessitano pertanto n comparatori che operino su n blocchi. Il costo è molto elevato, pertanto viene utilizzata solo per cache molto piccole.

8.3.2.5 Cache set-associativa

È una via di mezzo tra la mappatura diretta e la completamente associativa: ogni blocco di memoria può essere mappato su una linea di n blocchi diversi di cache. Si combinano pertanto le due idee: a ciascun blocco di memoria viene associata una certa linea, pertanto uno degli n blocchi di quella liena su cui si può mappare il blocco di memoria e all'interno della linea si effettua una ricerca parallela come se fosse una cache completamente associativa. Ovvero un blocco di memoria viene mappato nella linea data da indirizzo blocco modulo numero linee della cache. Pertanto per trovare il blocco all'interno della linea si deve confrontare in parallelo il tag del blocco con tuti i tag dei blocchi di quella linea. Aumentando l'associatività diminuisce la frequenza di miss, ma si rendono necessarie operazioni complesse. Inoltre nelle cache a mappatura diretta in caso di miss so sicuramente chi sostituire, nelle cache associative si possono utilizzare diverse strategie, come una FIFO o una Least Recently Used.

Capitolo 9

Input-output

I dispositivi di I/O sono necessari al computer per comunicare con l'esterno, devono essere espandibili ed eterogenei. La tipologia di prestazione varia dai casi: nel caso di tastiere e mouse interessa più la latenza, nel caso di dischi o interfacce di rete interessa più il throughtput. Questi dispositivi sono collegati al processore da un dispositivo di comunicazione chiamato bus. I dispositivi i I/O possono essere classificati in base alle operazioni che compiono: read o write, in base al loro partner (uomo o macchina) e in base alla velocità di trasferimento.

9.1 Connesione tra processori e periferiche

Le connessioni avvengono tramite delle strutture di comunicazione chiamate bus che si possono dividere in due categorie:

1. Bus processore-memoria, specializzato, corto e veloce.
2. Bus I/O, possono essere lunghi e permettere la connessione con periferiche eterogenee, tipicamente non sono collegati direttamente alla memoria ma richiedono un bus processore-memoria o un bus di sistema.

Se prima era presente un unico bus parallelo che collegava tutto, ora per problemi di clock e frequenze troppo elevate si usano architetture composte da più bus paralleli condivisi e bus seriali punto-punto. Si indicherà con transizione I/O invio di indirizzo e spedizione o ricevimento di dati, con input il trasferimento di dati da una periferica verso una memoria dove il processore può leggerla, con output il trasferimento di dati dalla memoria al dispositivo.

9.1.1 Bus sincrono

Tra le linee di controllo di questo bus esiste un clock e le comunicazioni avvengono con un protocollo collegato al il ciclo di clock. Questo tipo di bus è molto semplice da implementare e molto veloce, ma presenta poca robustezza al drift del clock e necessita che tutte le periferiche vadano alla velocità del clock.

9.1.2 Bus asincrono

Per rimediare agli inconvenienti del bus sincrono si utilizza il bus asincrono, in cui tutte le transazioni sono governate da una serie di handshake e non più dal clock. Questo richiede l'introduzione di nuove linee di controllo per segnalare inizio e fine di transazioni ma permette di collegare periferiche con velocità diversa. Questo tipo di bus è robusto rispetto ai ritardi e consente di comunicare con periferiche di tipo diverso, ma è lento nelle connessioni in quanto richiede diversi segnali di controllo che devono circolare per permettere il passaggio di informazioni, la circuiteria di controllo è inoltre complessa. Si utilizzano spesso tecnologie ibride, ma prevalentemente asincrone. Il trend più recente è quello di implementare all'interno del processore gli hub per il controllo di I/O e della memoria.

9.2 Prospettiva del programmatore

Al programmatore interessa come trasformare una richiesta di I/O in un comando per la periferica e come trasferire i dati. Riguardo al sistema operativo occorre osservareche i programmi che condividono il processore condividono anche il sistema di I/O, i trasferimenti dati utilizzano delle interrupt che impattano le funzionalit à del SO, devono pertanto essere eseguiti in una modalità del processore detta supervisor cui solo il codice kernel può accedere, il controllo di I/O si interseca con problematiche di concorrenza. Un sistema operativo deve garantire che un utente con i permessi possa accedere ai sistemi di I/O cui può accedere, deve fornire comandi di alto livello per gestire operazioni di basso livello, gestire le interruzioni generate dall' I/O, ripartire l'accesso a un dispositivo tra i programmi che lo richiedono. Per implementare tali funzionalità occorre rendere possibile al SO di inviare comandi alle periferiche, rendere possibile ai dispositivi notificare il corretto avvenimento di un'operazione, consentire trasferimenti diretti tra dispositivi e memoria. Questo si fa fornendo sulle relative linee di bus delle parole di controllo scrivendo o leggendo in particolari locazioni di memoria (memory mapped I/O) o tramite delle istruzioni speciali legate all'I/O. Scrivendo una particolare parola in una locazione di memoria associata al dispositivo il sistema di memoria ignora la scrittura e il controllore di I/O intercetta l'indirizzo particolare e trasmette il dato al dispositivo sotto forma di comando. Queste particolari locazioni di memoria sono inaccessibili ai programmi utante ma solo al sistema operativo, pertanto deve esserci una chiamata di sistema che faccia commutare il processore in modalità supervisore. Il dispositivo può utilizzare queste locazioni per trasmettere dati o pre-segnalare il suo stato.

9.3 Trasmettere o ricevere dati

Per trasferire dati la maniera più semplice è l'attesa attiva o polling: si manda un comando di lettura scrittura alla periferica e si fa un ciclo di attesa testando

il bit di stato per vedere quando il dispositivo è pronto.

9.3.0.1 Costo del polling

Percentuale del processore utilizzata dal polling: prodotto tra frequenza di operazione e clock utilizzati dall'operazione di polling diviso la frequenza del processore.

9.3.1 Considerazioni

L'attesa attiva fa perdere tempo al processore che si dedica a cicli di lettura inutili, può pertanto essere utilizzato quando le operazioni avvengono con velocità determinata e se i dati vengono trasferiti con bitrate elevati il ciclo di attesa attiva dura poco. Se lo spreco è inaccettabile viene utilizzato I/O a interruzione di programma (interrupt driven I/O).

9.3.2 Interruzioni di programma

Un'interruzione I/O è un segnale utilizzato per segnalare al processore che la periferica è pronta a eseguire il trasferimento richiesto, possono avere diverso grado di urgenza e occorre un modo per segnalare al processore quale periferica richiede l'interruzione. Questo tipo di interruzioni sono sempre asincrone rispetto all'esecuzione delle istruzioni. Quando arriva un'interruzione l'istruzione corrente viene terminata prima di considerare l'interruzione. Si può differire l'esecuzione di un programma a un altro momento. Con questo metodo non occorre interrompere l'esecuzione del programma se non quando il dato può essere effettivamente riferito in memoria, ma si necessita di hardware speciale per interrompere l'esecuzione del programma per permettere alle periferiche di generare un'interruzione e per rilevare l'istruzione, salvare lo stato del processore per esegure una routine di servizio (Interrupt Service Routine ISR) e poi riprendere dal punto dove si era interrotto. Il guadagno rispetto al polling è che l'overhead necessario alla lettura dei dati viene pagato solo quando il dato è effettivamete presente.

9.4 Eccezioni

Un'eccezione è un trasferimento del controllo del programma non programmato. Il sistema effettua delle azioni per gestire le eccezioni, ome per esempio deve sapere dove registrare il punto di interruzione e dove salvare lo stato e poi, ad eccezione finita, come riprendere dal punto immediatamente successivo al punto di interruzione.

9.4.1 Interrupts

Sono eccezioni causate da eventi esterni, sono asincrone, possono essere gestite nello spazio tra due istruzioni, sospendono il programma e riprendono dallo stato in cui si era interrotto.

9.4.2 Traps

Sono eccezioni causate da eventi interni al programma, sincrone, gestite da un trap handler è possibile riprovare a eseguire l'istruzione che ha generato l'istruzione o abortire il programma.