Andrea Aquino, Sara Bergonzoni, Matteo Brucato, Miro Mannino

DISPENSA GENERALE DI ARCHITETTURA

Corso di Architettura professoressa Kiziltan (Bologna, a.a. 2008/2009)

Indice generale

Informazioni

La presente dispensa

La presente dispensa è da considerarsi un semplice aiuto (speriamo valido) allo studio della materia relativa al corso tenuto dalla professoressa Kiziltan (Università di Bologna), in particolare per come ci è stato presentato nell'anno 2008/2009. Ma può essere anche un buon strumento per corsi analoghi. La sua lettura non è da considerarsi in nessun modo sostitutiva allo studio di un libro di Architettura (a tal fine vedere la Bibliografia per i testi che consigliamo), ma solo un sussidio per il ripasso di argomenti di cui si è già sentito parlare almeno una volta o meglio studiati.

E' stata scritta principalmente ad uso personale e non pretende in nessun modo di essere completa od esaustiva. Abbiamo deciso di pubblicarla per far sì che potesse essere utile a qualcun altro, oltre che a noi stessi. Ci scusiamo in anticipo per eventuali errori presenti nel testo e, anzi, vi saremmo grati se voleste segnalarceli.

Gli autori

In ordine alfabetico di cognome:

Andrea Aquino, Sara Bergonzoni, Matteo Brucato, Miro Mannino (Bologna 2009).

Per contattare Matteo Brucato: brucato AT cs PUNTO unibo PUNTO it

Licenza d'uso

Changelog

- 28 maggio 2009: Prima pubblicazione.
- 30 giugno 2009: correzione piccoli errori, modifiche minori al layout.

Bibliografia

- Andrew S. Tanenbaum, *Architettura degli calcolatori. Un approccio strutturale*, Milano, Pearson Prentice Hall, 2006
- M. Morris Mano e Charles R. Kime, *Reti logiche – quarta edizione*, Milano, Pearson Prentice Hall, 2008

Macchina virtuale

Traduzione

Un programma scritto in un generico linguaggio viene tradotto quando, in una fase chiamata compilazione, ogni istruzione viene sostituita da una serie di istruzioni semanticamente equivalenti, ma in un linguaggio comprensibile alla macchina. Una volta che tale programma è stato tradotto sarà possibile eseguirlo programma direttamente al livello hardware.

Interpretazione

Un programma scritto in un generico linguaggio viene interpretato quando, in fase di esecuzione, ogni istruzione viene eseguita nella macchina. Il lavoro di esecuzione di tutte le istruzioni è svolto da un altro programma chiamato interprete.

Macchina virtuale

Talvolta piuttosto che vedere l'interprete come un programma che funge da esecutore di codice è più semplice vederlo come una macchina virtuale che ha un linguaggio diverso dal linguaggio macchina. Essenzialmente il compito di una macchina virtuale non è differente dal lavoro svolto da un interprete, è solo questione di comprensibilità e appeal. Di solito si comprende meglio la cosa si pensa alla JVM che è letteralmente una macchina virtuale che ha un proprio linguaggio (studiato a tavolino per permettere una facile compilazione Java → ByteCode). Ma dall'altra parte la Microsoft non parla di macchina virtuale ma di interprete! Ma anche C# viene compilato in un codice molto simile al ByteCode di Java. Questo ci fa capire come le due cose siano essenzialmente le stesse

Traduzione e interpretazione, similitudini

Sono simili poiché tutti e due eseguono l'identico compito di tradurre un istruzione. Mentre però la traduzione traduce una volta sola un programma e tale programma tradotto può successivamente essere eseguito senza altre modifiche o aiuti; l'interpretazione traduce ed esegue il programma in fase di esecuzione del programma. Ciò vuol dire che i programmi interpretati hanno sempre bisogno di qualcosa in più rispetto a quelli tradotti, cioè l'interprete.

Traduzione e interpretazione, velocità a confronto

Sicuramente il più veloce è il metodo della traduzione. Per quanto lunga, lenta e complessa possa essere la traduzione è irrilevante poiché viene fatta una sola volta. I programmi tradotti possono essere eseguiti poi alla massima velocità poiché non hanno bisogno di ulteriori metamorfosi.

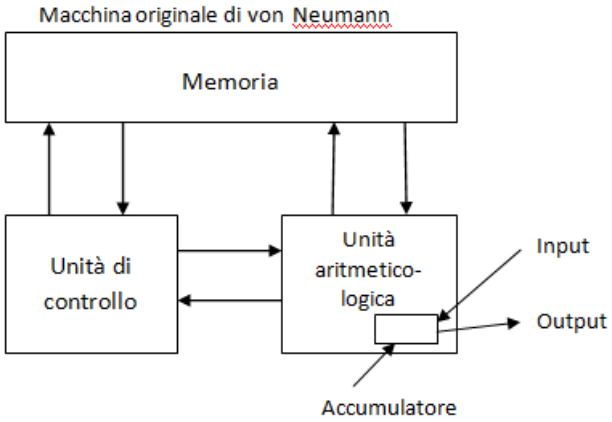
Pietre miliari nell'architettura dei computer

La storia del computer si suddivide in 6 generazioni: Computer meccanici, Valvole, Transistor, circuiti integrati, Integrazione a grandissima scala, Computer invisibili.

Difference Engine e Analytical Engine

Charles Baddage intorno al 1820 concepì la difference engine in grado di calcolare un solo algoritmo. La macchina utilizzava come sistema di output una lastra di rame su cui incideva il risultato. In seguito creò la analytical engine anch'essa interamente meccanica ma in grado di eseguire diversi programmi che leggeva da schede perforate che prendeva in input. L'analytical engine era formata da: magazzino (la memoria), mulino (l'unità computazionale), dispositivo di input (le schede perforate) e dispositivo di output (output stampato e perforato). Il mulino poteva prelevare dati dal magazzino eseguirvi sopra operazioni di somma, sottrazione, moltiplicazione e divisione e reinserire il risultato nel magazzino.

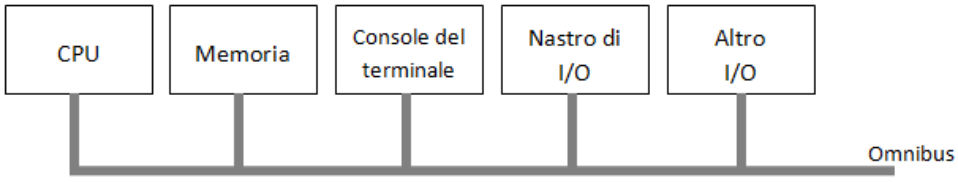
Macchina di von Neumann



Dopo la fine della guerra John von Neumann ideo quella che oggi viene definita la macchina di von Neumann la quale rappresentava i programmi così come i dati in forma numerica e rappresentava i numeri in forma binaria. Queste idee insieme ad altre permisero di realizzare il computer ISA, la prima macchina che memorizzava il programma in memoria. L'architettura di ISA è alla base di quasi tutti gli odierni computer digitali.

La macchina di von Neumann era composta da cinque componenti principali: la memoria, l'unità aritmetico-logica, l'unità di controllo e i dispositivi di input e output.

PDP-8



Progettato dalla DEC (Digital Equipment Corporation), la versione PDP-8, ancora più economica della versione precedente, era dotata di un unico bus chiamato omnibus, questa architettura differiva dalla macchina ISA che era centralizzata rispetto alla memoria.

6600

La CDC (Control data corporation) creò il modello 6600, estremamente veloce grazie alla sua Cpu altamente parallela, essa era dotata al suo interno di varie unità funzionali che potevano lavorare contemporaneamente. Inoltre il modello 6600 aveva un certo numero di piccoli computer che lo aiutavano, la CPU poteva spendere tutto il suo tempo a macinare numeri, lasciando ai piccoli computer tutti i dettagli della gestione dei programma di input/output.

System/360

La System/360 dell'IBM, fu la prima linea di computer che presentava la compatibilità tra tutti i modelli della serie. Questa era una famiglia composta da macchine dotate tutte dello stesso linguaggio assemblativo, ma di dimensione e potenza crescenti. Il software creato per una di queste macchine poteva essere eseguito in linea di principio su qualsiasi altra macchina della famiglia La famiglia 360 era dotata anche della multiprogrammazione, grazie a cui poteva tenere in memoria più di un programma alla volta, in modo che mentre si era in attesa di completare un operazione di input/output era possibile eseguire dei calcoli. Il 360 fu il primo a essere capace di emulare altri vecchi computer dell'IBM.

Legge di Moore e conseguenze

Legge di Moore

La legge di Moore, enunciata da Gordon Moore nel 1965, afferma che: "Le prestazioni dei processori, e il numero di transistor ad esso relativo, raddoppiano ogni 18 mesi" ovvero un aumento annuo del numero di transistor in un chip di circa il 60%. Essa si è mantenuta valida fino ad oggi e ha creato quello che gli economisti chiamano un ciclo virtuoso: i prodotti migliorano e diventano più economici, così si crea concorrenza che a sua volta genera un nuovo ciclo identico.

Spettro di computer

Il guadagno fornito dalla legge di Moore viene sfruttato in due modi: costruendo computer sempre più potenti allo stesso prezzo; costruendo di anno in anno sempre lo stesso computer ad un prezzo via via inferiore. Grazie a questi approcci (ed altri) oggi si ha una vasta gamma di computer: computer usa e getta (es. RFID), microcontrollori (es. negli elettrodomestici), computer per giocare (es. PS3), personal computer, server, cluster di workstation, mainframe.

Istruzioni dei calcolatori

Ciclo PDE

Con ciclo PDE si indica quella particolare sequenza di passi della CPU per eseguire delle istruzioni. Tale ciclo preleva le istruzioni dalla memoria, le decodifica e quindi le esegue.

Il ciclo PDE [prelievo decodifica esecuzione] si compone fondamentalmente di 6 fasi.

1. Prelevare l'istruzione corrente e memorizzarla nell'IR [Istruction Register: registro della CPU che contiene l'istruzione attualmente in fase di esecuzione]
2. Modificare PC [Program Counter: registro della CPU che punta alla prossima istruzione da eseguire] in modo da farlo puntare alla prossima istruzione
3. Determinare il tipo dell'istruzione prelevata
4. Se l'istruzione necessita di parole di memoria queste vanno eventualmente localizzate e se necessario prelevate e salvate in opportuni registri [in quanto la CPU non è in grado di calcolare operazioni tra registri e memoria]
5. L'istruzione viene quindi eseguita
6. Si ritorna al punto 1. e il ciclo ricomincia

Potrebbe essere importante tenere presente che il ciclo PDE non è necessariamente programmato attraverso l'hardware seppur questo accade nella stragrande maggioranza dei Computer moderni. Essendo fondamentalmente un algoritmo può essere interpretato.

CISC contro RISC

Fondamentalmente RISC [Reduced Istruction Set Computer] è una tipologia di processore dotato di un ridotto insieme di istruzioni [generalmente intorno alle 50, ben di meno delle 200-300 dei processori di tipo CISC] estremamente veloci in grado di essere lanciate rapidamente. CISC [Complex Istruction Set COmputer] è fondamentalmente l'opposto, un sistema che consta di numerose istruzioni di velocità limitata ma in grado di svolgere una discreta mole di lavoro. Quando il primo processore di tipo RISC fu messo a punto sembrava avere netta prevalenza su gli altri di tipo CISC, effettivamente nonostante impiegasse 5 o 6 istruzioni per svolgere il lavoro di una sola istruzione CISC la sua velocità per istruzione era circa 10 volte maggiore in quanto le istruzioni RISC sono NON interpretate.

Occorre spendere una parola sul fatto che le macchine RISC furono ideate e realizzate senza alcuna presunzione di retrocompatibilità con qualsivoglia sistema, dunque le prime macchine RISC nonostante effettivamente convenienti a livello di potenza e di qualità non divennero uno standard considerando i miliardi investiti dalle compagnie software nelle precedenti macchine CISC [Intel Pentium nè è un esempio lampante]. Oltretutto Intel riuscì ad applicare miracolosamente le idee dell'architettura RISC all'interno delle proprie macchine trovando un compromesso decisamente accettabile tra le operazioni che dovevano essere svolte da un nucleo di istruzioni semplici e veloci e quelle invece particolarmente complicate per cui un'istruzione complessa risultava di poco peggiore rispetto lo standard RISC.

Parallelismo a livello di istruzione

Buffer di prefetch

Considerando che la lettura dalla memoria centrale è piuttosto lenta un modo per velocizzare l'esecuzione delle istruzioni è prelevarle dalla memoria mentre altre istruzioni vengono eseguite e conservate fino al loro turno in dei registri speciali chiamati appunto "Buffer di Prefetch". Questo concetto sfrutta il parallelismo tra carimento ed esecuzione delle istruzioni in maniera da velocizzare il sistema.

Pipeline

E' un tipo di parallelismo a livello di istruzione usato per aumentare il numero di istruzioni eseguite in singole unità di tempo. Consiste nel dividere il cilco PDE in sotto parti detti stadi, ognuno dei quali identifica un'operazione semplice, facendoli eseguire all'hardware contemporaneamente. Ciò permette di bilanciare latenza e larghezza di banda del processore. Esistono anche pipeline doppie dove c'è un'unica unità di fetch e due pipeline che eseguono le altre operazioni contemporaneamente.

Architettura superscalare

Consiste in un'unica pipeline con più unità funzionali specializzate sullo stadio di esecuzione (per esempio ALU, LOAD, STORE, etc.). E' ovvio che lo stadio precedente a quello di esecuzione (e in generale tutti gli altri stadi) debba essere sensibilmente più veloce di quest'ultimo per permettere alle unità funzionali ivi presenti di eseguire contemporaneamente più istruzioni.

Parallelismo a livello di processore

Array di computer

Queste strutture risultano particolarmente utili per eseguire lo stesso calcolo su un gran numero di dati. Un array computer (ovvero un array di processori) consiste in un gran numero di processori identici che eseguono la stessa sequenza di istruzioni su insiemi diversi di dati. Una singola unità di controllo trasmette in broadcast le istruzioni ad un “array”, in cui ogni cella è rappresentata da un processore e da una memoria. Questo tipo di architettura viene anche chiamata SIMD (single Instruction- stream Multiple Datastream, “Istruzioni singole, dati multipli”)

Un processore vettoriale esegue una sequenza di istruzioni su coppie di dato. La differenza tra i due è che per eseguire la somma a coppie degli elementi di due vettori, l’array di processori lo fa usando tanti sommatori quanti sono gli elementi dei vettori, un processore vettoriale esegue la somma tra due coppie provenienti da sue registri vettoriali che alimentano un sommatore strutturato a pipeline.

Multiprocessori

Il multiprocessore è un sistema composto da più CPU complete con una memoria in comune. Dato che ogni CPU può leggere e scrivere una qualsiasi parte della memoria, esse devono coordinarsi (via software) per evitare di ostacolarsi a vicenda. Esistono vari sistemi d’implementazione uno dei quali consiste in un singolo bus con più CPU, tutte connesse ad un'unica memoria, questo però provoca dei conflitti tra le CPU che tentano di accedere contemporaneamente alla memoria attraverso un unico bus. Per risolvere questo problema si può dotare ogni CPU di una propria

memoria locale, a cui può accedere senza passare per il bus principale e che non è accessibile agli altri. Le CPU interagiscono con un legame chiamato tighly coupled (legate strettamente).

Multicomputer

Il problema dei multiprocessori consiste nel non poter collegare più di 256 processori senza incorrere nel problema di come connetterli alla memoria. Per aggirare questo problema sono stati realizzati i multicomputer che sono sistemi composti da un gran numero di calcolatori interconnessi tra loro ciascuno dotato di una propria memoria privata. Le CPU dei multicomputer comunicano tra loro inviandosi messaggi, simili a e-meil, ma molto più veloci. Le CPU interagiscono con un legame chiamato loosely coupled (legamento lasco).

Ciclo di clock, latenza e larghezza di banda

Ciclo di clock

Il **ciclo di clock** è il tempo che intercorre tra il verificarsi di due colpi di clock successivi.

La **frequenza** o **velocità di clock** è il numero di colpi di clock che vengono eseguiti in una determinata unità di tempo e di solito si misura in Herz (cicli di clock / secondo).

Di solito i fronti di salita e di discesa del clock sono usati per stabilire gli attimi in cui devono avvenire le transazioni elettriche all'interno dell'hardware del computer.

Latenza

La **latenza** di un processore è il tempo necessario affinché venga eseguita una singola istruzione. Si misura in ns (nanosecondi = 10^{-9} secondi). In un processore senza pipeline la latenza è semplicemente data dal tempo del ciclo di clock, poiché ad ogni colpo di clock viene eseguita un'istruzione. In un processore con pipeline ad n stadi la latenza è data dal prodotto del ciclo di clock per il numero di stadi, poiché ogni stadio esegue una piccola parte dell'istruzione ed essa sarà eseguita interamente solo dopo che avrà passato tutti gli n stadi.

Larghezza di banda

La **larghezza di banda** di un processore è il numero di istruzioni eseguite in una unità di tempo. Di solito si misura in MIPS (milioni di istruzioni al secondo). Essa si calcola allo stesso modo sia in un processore senza pipeline che in uno con pipeline, come 1000 fratto il tempo del ciclo di clock MIPS.

I MIPS non sono altro che una frequenza dove è sotto inteso il 10^{-9} (il milione) e dove si contano il numero di istruzioni al secondo. Nulla vieta di calcolare la larghezza di banda in Mhz, i cui valori sarebbero del tutto analoghi a quelli espressi in MIPS. Quella di utilizzare i MIPS è solo una convenzione.

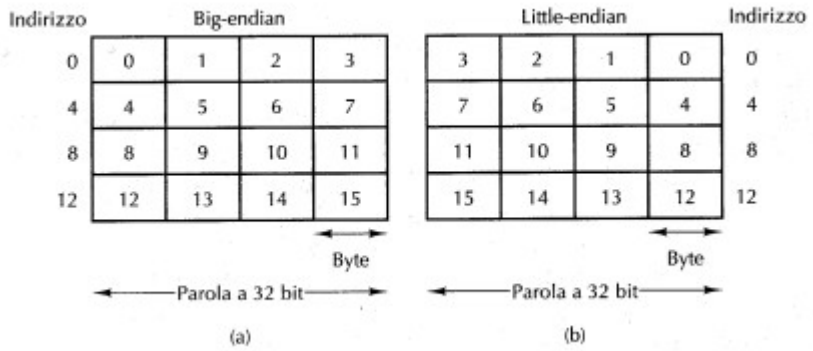
Ordinamento dei byte

Parole

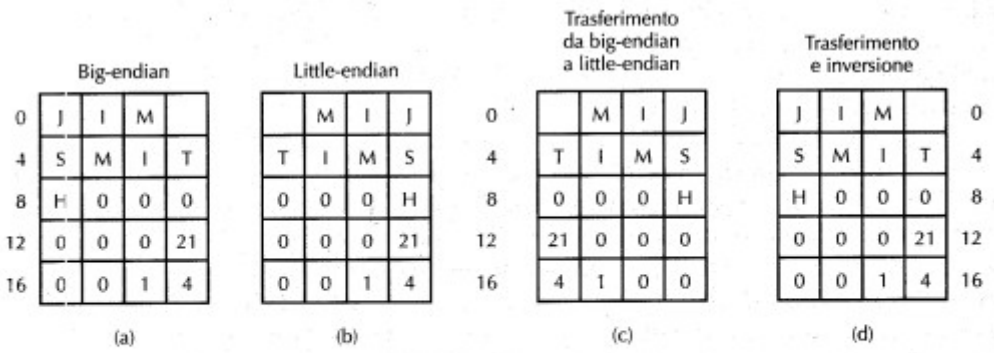
La cella rappresenta la più piccola unità di memoria rappresentabile essa si è standardizzata e ha assunto le dimensioni di un byte; i byte sono raggruppati in parole (word). I moderni calcolatori hanno parole da 4 byte o da 8 byte (32 o 64 bit rispettivamente). Un calcolatore con parole da m bit è formato da n byte per parola (m = n * 8bit). Le parole risultano importanti poiché la maggior parte delle istruzioni lavora su parole intere, piuttosto che su singole celle di memoria.

Little-endian e big-endian

All'interno di una parola i byte possono essere numerati da sinistra a destra o da destra verso sinistra. Questo ha portato alla creazione di due diversi sistemi di numerazione il little-endian che inizia a contare dal byte meno significativo e big-endian che inizia a contare dal byte più significativo.



Il problema tra questi due sistemi di numerazione si ha quando si vuole trasferire una stringa non numerica da una macchina big-endian in una little-endian o viceversa. A questo problema non è stata trovata soluzione, anche invertendo l'ordine dei byte le stringhe risultano scritte al contrario.



(a) Un record dell'archivio del personale per una macchina *big endian*.
(b) Lo stesso elemento per una macchina *little endian*. (c) Il risultato del trasferimento dell'elemento da una macchina *big endian* a una *little endian*.
(d) Il risultato dell'inversione dei byte di (c).

Codici correttori

Distanza di Hamming

La distanza di Hamming tra due parole di codice è il numero di inversioni necessarie per far diventare identiche le due parole, ovvero il numero di bit differenti tra le due parole.

La distanza di Hamming in un codice intero invece è la minima distanza di Hamming esistente considerando tutte le coppie di parole.

Rivelazione e correzione di errori

Per rilevare n errori è necessario un codice con distanza di Hamming pari a n + 1.

Per correggere n errori è necessario un codice con distanza di Hamming pari a 2n + 1.

Bit di parità

Sono quei bit che vengono impostati ad 1 o 0 a seconda della parità dei bit che stanno controllando. Di solito vengono impostati in modo da rendere il numero di 1 presenti nei bit che stanno controllando un numero pari. Se ad esempio un codice ha un solo bit di parità che controlla tutti i bit presenti nelle parole, la parola 001 diventerà 001**1**, per rendere il numero di 1 pari, mentre 101 diventerà 101**0**, per lasciare il numero di bit 1 pari.

Algoritmo di Hamming

E' un metodo per costruire codici con correzione di errore, che raggiungano il limite teorico.

m = numero di bit per i dati

r = numero di bit di controllo (es. parità)

Formula di Hamming: $2^r \geq m + r + 1$

Si numerano i bit a partire da 1 e i bit che stanno in una posizione che è una potenza di 2 (es. 1, 2, 4, 8, 16...) saranno bit di controllo, tutti gli altri bit per i dati. I bit di controllo controlleranno solo alcuni bit secondo questa legge: ogni bit di controllo controlla se stesso e tutti i bit che sono in una posizione che è la somma delle posizioni di altri bit di controllo con se stesso.

Cache

La cache è una memoria veloce e costosa che serve per migliorare lettura e scrittura dalla memoria.

Hit ratio

E' una percentuale (indicata con una frazione) che esprime le volte che la parola viene letta dalla cache piuttosto che direttamente dalla memoria. Per esempio se k istruzioni fanno riferimento ad una parola si avrà che essa è stata letta la prima volta dalla memoria e le successive (k-1) volte dalla cache. In questo caso si avrà quindi un hit ratio di (k – 1)/k.

Miss ratio

E' una percentuale (indicata con una frazione) che esprime le volte che la parola viene letta dalla memoria piuttosto che dalla cache. Il miss ratio è definito come 1 – h, dove h è l'hit ratio.

Tempo medio di accesso

E' il tempo di accesso alla memoria medio ed è espresso dalla formula:

TempoMedioDiAccesso = c + (1 - h)*m Dove: c = tempo di accesso alla cache

(1 – h) = miss ratio

m = tempo di accesso alla memoria

Per avere un effettivo miglioramento nelle prestazioni è necessario che

$$h > \frac{c}{m}$$

Principio di località

La cache si basa sul principio di località. Tale principio consiste nel copiare sulla cache, insieme alla parola letta, anche un intera zona adiacente ad essa (chiamata linea di cache). Tale principio si basa sulla semplice considerazione che, una volta che si accede ad una determinata cella è molto probabile che i prossimi accessi verranno effettuati nelle celle vicine ad essa.

Cache unificate e cache specializzate e cache multilivello

La cache che memorizza istruzioni e dati sulla stessa cache è chiamata cache unificata. La cache che memorizza istruzioni e dati su cache differenti è chiamata cache specializzata. Quest'ultima organizzazione è chiamata architettura Havard. La differenza fra le due consiste che caricamento di istruzioni e dati possono essere eseguite in parallelo solo con una cache specializzata. La cache multilivello consiste nel suddividere la cache in cache più lente e grandi e cache più veloci e piccole. Generalmente operano controllando prima la cache di livello 1, se la parola non viene trovata si procede nei successivi livelli fino a quando non si è costretti a prelevare la parola sulla memoria..

Speed-up

E' la frazione che esprime il rapporto fra il tempo di esecuzione con e senza cache.

Talvolta è utile la seguente formula:

$$\frac{T_m}{T_c} = \frac{1}{1 + A(\frac{c}{m} - h)}$$

Dove:

- c = tempo di accesso alla cache
- h = hit ratio
- m = tempo di accesso alla memoria
- A = percentuale (in frazione) delle istruzioni del programma che fanno riferimento alla memoria

Dischi magnetici

Il **Tempo di seek** è il tempo che impiega la testina a posizionarsi nel punto giusto del disco (inteso come giusta distanza dal centro).

Il **Tempo di latenza** è il tempo che impiega il settore giusto del disco a trovarsi sotto la testina (generalmente il tempo di latenza è la metà del tempo di rotazione del disco).

Il **Tempo di accesso** la somma dei due tempi precedenti.

Il **Tempo di trasferimento** è il tempo che impiega a trasferire un certo numero di dati.

$$TempoDiTrasferimento = \frac{Dimensione[MB]}{VelocitàDiTrasferimento[MB/s]}$$

Tipico esercizio:

Viene dato:

Tempo di seek: T_s

Rotazioni del disco al minuto: R

Velocità di trasferimento: V_t

Il tempo di seek è già esplicitato ed è immediato. (se vogliamo però metterlo in relazione sommandolo con gli altri occorrerà trasformarlo in secondi)

Il tempo di latenza è metà di quello di rotazione:

- Sapendo che il disco fa R rotazioni ogni minuto cominciamo a chiederci quante ne fa ogni secondo visto che vogliamo tutto in secondi, e allora dividiamolo per 60
- Sapendo adesso quante volte gira in un secondo noi vogliamo però quanto tempo ci mette per fare un giro, allora dividiamo il nostro secondo per il numero di rotazioni che fa in un secondo, cioè facciamo 1/(R/60)
- Ma noi vogliamo metà di questo tempo, allora lo dividiamo ulteriormente per 2

Infine si ottiene che $Tempo\ di\ latenza = \frac{60}{R * 2}$

Il tempo di trasferimento poi è dato dalla formula di sopra sapendo quanto si vuole trasferire e la velocità che conosciamo che è Vt (quest'ultimo di solito è espresso in MB/s quindi se la dimensione è in byte per esempio bisogna trasformarlo in MB)

Ad esempio 512B sono 0,5KB, cioè 0,5x10-3MB.

Organizzazione dei dischi magnetici

IDE

E' uno standard per collegare dischi magnetici e ottici al computer. IDE (Integrated Drive Electronics, "memoria di massa con elettronica integrata"), questo fu il primo disco con integrato il controllo, che in precedenza si trovava su una scheda esterna, venne poi introdotto il modello EDIE (extended IDE), che potevano indirizzare una quantità maggiore di memoria. I controllori EIDE

potevano avere due canali, ciascuno dei quali poteva avere un disco primario e uno secondario (4 dischi in tutto). Venne poi introdotta un'interfaccia serial ATA molto più veloce delle precedente, che utilizzava invece un enorme quantità di fili paralleli un cavo seriale, inoltre essa veniva alimentata a 0,5 V.

SCSI

Le uniche differenze con IDE sono la maggiore velocità di trasferimento dati e il fatto che lo si può utilizzare per collegare dispositivi di vario tipo. Bisogna ricordare che SCSI è anche un bus che permette di collegare ad un connettore sette dispositivi. Ciascun dispositivo SCSI possiede un ID, compreso tra 0 e 7 (15 nel caso di wide SCSI), e due connettori, uno per l'input e uno per l'output. I cavi connettono in serie l'output di un dispositivo con l'input del successivo.

RAID

Per aumentare l'affidabilità del computer al posto di un solo hard-disk si può utilizzare un controllore RAID che permette di copiare i dati su una "composizione" di dischi RAID. Vi sono sei livelli RAID che si differenziano per come il controllore suddivide il dato tra i vari dischi..

Livello 0) Lavora in modalità striping ovvero l'informazione immagazzinata è suddivisa in strip (strisce), due strip che vanno lette in maniera consecutiva sono scritte su due dischi diversi ma consecutivi. Questa modalità non presenta nessuna possibilità di recupero dati ma è ottima per lavorare con richieste di grande dimensioni.

Livello 1) Le informazioni sono scritti sempre in modalità striping, ma ogni disco ha un copia. Le prestazioni in scrittura non migliorano rispetto al precedente, ma la capacità di lettura è doppia e la tolleranza agli errori è ottima.

Livello 2) Le informazioni rappresentate in byte vengono divise in due, ai nibble così ottenuti vengono aggiunti 3 bit che rappresentano il codice di Hamming (sono quindi necessari 7 dischi). In questo caso però la rotazione dei dischi deve essere sincronizzata, inoltre lo schema è efficace solo se usano molti dischi e il controllore deve lavorare molto per controllare il codice di Hamming.

Livello 3) Come nel caso precedente l'informazione viene suddivisa in nibble e viene aggiunto un solo disco per il bit di parità. Anche in questo caso i dischi devono essere sincronizzati. Il bit di parità in caso di rottura di un disco permette di recuperare tutte le informazioni.

Livello 4) Lavora in base strip e nono richiede la sincronizzazione dei dischi, questa configurazione è simile al livello 0 ma ha in più un disco che contiene i bit di parità strip-per-strip (si calcola l'or esclusivo di tutte le prime parole di un disco, poi delle seconde….). Quando si aggiornano piccole quantità di dati è comune necessario leggere tutti i dischi per ricalcolare la strip di parità.

Livello 5) Esso è identico al livello 4 a parte per il fatto che poiché questo sollecitava troppo il disco di parità, l'informazione dello strip di parità è suddiviso tra tutti i dischi in modalità round robin.

Conversioni

Conversione da decimale in binario

Considerando che 10 non è una potenza di 2 purtroppo non c'è modo di utilizzare una trasformazione diretta che in generale è più semplice del metodo che presenterò qui di seguito. Per trasformare un numero da decimale in binario occorre dividerlo per due, memorizzare il resto e procedere ricorsivamente sul quoziente fino a quando non si ottiene quoziente 0.

Vediamo un esempio con spiegazione:
1452 in binario
1452 / 2 = 726 con il resto di 0
726 / 2 = 363 con il resto di 0
363 / 2 = 181 con il resto di 1
181 / 2 = 90 con il resto di 1
90 / 2 = 45 con il resto di 0
45 / 2 = 22 con il resto di 1
22 / 2 = 11 con il resto di 0
11 / 2 = 5 con il resto di 1
5 / 2 = 2 con il resto di 1
2 / 2 = 1 con il resto di 0
1 / 2 = 0 con il resto di 1
1452 in binario è dunque il numero ottenuto leggendo i resti ottenuti dall'ultimo al primo:
1452 = 10110101100

Qualche altro esempio banale:
16 in binario [potenza di due, tanto per farlo apposta]
16 / 2 = 8 con il resto di 0
8 / 2 = 4 con il resto di 0
4 / 2 = 2 con il resto di 0
2 / 2 = 1 con il resto di 0
1 / 2 = 0 con il resto di 1
16 = 10000 [che caso!]

9 in binario
9 / 2 = 4 con il resto di 1
4 / 2 = 2 con il resto di 0
2 / 2 = 1 con il resto di 0
1 / 2 = 0 con il resto di 1
9 = 1001

Conversione da decimale in binario di numeri in virgola mobile

Per convertire un numero decimale in virgola mobile in binario bisogna separare la parte intera dalla parte decimale e trasformarle entrambe in binario ma con metodi differenti. La parte intera va trasformata normalmente in binario con il metodo su mostrato. La parte decimale [della forma 0.xxxxx] va moltiplicata per 2 finchè non diventa intera, ad ogni passo memorizziamo la parte intera ottenuta dal raddoppiamento della parte decimale. Ogni volta che lo zero [parte intera del numero decimale su cui stiamo lavorando] diventa un uno questo va memorizzato e bisogna riportarlo a zero prima tornare a raddoppiare.

Vediamo un esempio poichè l'algoritmo non è proprio banale:
0.25 in binario

separiamo la parte intera dalla parte decimale
0 – 0.25
convertiamo la parte intera [0] in binario normalmente, in questo caso rimane esattamente 0
raddoppiamo la parte decimale ottenendo
0.50
memorizziamo la parte intera e riportiamola a 0 anche se inutile in quanto già 0 ottenendo
0.5 [0]
raddoppiamo di nuovo il numero così ottenuto
1.0
memorizziamo la parte intera e la riportiamo a 0
0 [01]
a questo punto abbiamo fatto sparire la parte decimale quindi abbiamo finito
il risultato è la parte intera trasformata in binario all'inizio, virgola il numero che abbiamo ottenuto memorizzando le parti intere con l'algoritmo sulla parte decimale
0.25 in decimale = 0.01 in binario

Conversione da binario in ottale

Questo è uno dei casi fortunati in cui dobbiamo tradurre un numero da un sistema di numerazione in uno che è una sua potenza. Per convertire un numero da binario in ottale si procede nel seguente modo. Si spezza il numero a gruppi di 3 a partire da destra [aggiungendo eventualmente degli zeri qualora ne mancassero nell'ultimo gruppo a sinistra] e si trasforma ogni terza in un numero decimale da 0 a 7 ottendo così il corrispettivo ottale.

Vediamo un esempio:
1001 in ottale
1 – 001
aggiungo due zeri perchè l'ultimo gruppo non è una terza e ottengo
001 – 001
trasformo i due numeri in decimale
1 – 1
il risultato è dunque 11
1001 in binario = 11 in ottale
altro esempio, ripeto i passaggi nell'ordine appena esposto:
10000 in ottale
10 – 000
010 – 000
2 – 0
10000 in binario = 20 in ottale [che caso fortuito! forse perchè 10000 è 16 in decimale? :D]
ultimo esempio commentato se proprio non si fosse ancora capito.
100111001 in ottale
100 - 111 – 001
4 - 7 – 1
100111001 in binario = 471 in ottale

Conversione da binario in esadecimale

Anche questo risulta un caso comodo in quanto 16 è una potenza di 2. Si procede esattamente come per le trasformazioni binario-ottale ma invece di dividere il numero in gruppi di 3 lo si divide in gruppi di 4 in modo da ottenere sempre per ogni quaterna [chissà se si può dire, io l'ho sempre associato alla tombola ma in questo caso intendo un gruppo di 4 numeri] un numero tra 0 e 15.

100011100111110 in esadecimale
100 - 0111 - 0011 – 1110
l'ultimo gruppo non è da 4 quindi aggiungo come al solito uno zero in testa

0100 - 0111 - 0011 – 1110
traduco ogni gruppo di quattro bit nel corrispettivo decimale
4 - 7 - 3 – 14
ricordo che nel sistema di numerazione esadecimale le cifre sono le seguenti:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
dunque il 14 corrisponde a E
il numero che ottengo è quindi:
473E
allora 100011100111110 in binario = 473E in esadecimale

Conversione da decimale in esadecimale o in ottale

Ci sono due possibilità per traduzioni di questo tipo. Probabilmente la più veloce è quella tale per cui si traduce il numero decimale in binario attraverso l'algoritmo su enunciato e poi lo si converte nel sistema di riferimento voluto. L'altro metodo consiste nello sfruttare una variante del primo sistema per le conversioni decimale-binario dove si divide continuamente per 8 o per 16 anzicchè per 2, e si memorizzano i resti nel solito modo [che stavolta però non saranno più 0 o 1, ma un numero compreso tra 0 e il sistema di numerazione in cui stiamo traducendo – 1]. Non riporto esempi di tali metodi perchè sono stati ampiamente commentati sopra.

Operazioni con numeri binari

Sommare numeri binari in complemento a due con bit di segno

Basta sommare a partire da destra i bit dei numeri [qualora abbiano lo stesso segno (dato dal bit di segno che considereremo essere il più a sinistra), se hanno segni diversi bisogna calcolare il complemento a due del negativo per poterlo sommare secondo questa regola all'altro].

0+0 = 0 senza riporto
0+1 = 1 senza riporto
1+0 = 1 senza riporto
1+1 = 1 con il riporto di 1
l'ultimo riporto accumulato va ignorato
esempio:

```
1  1 1 1 1   riporti
   1 0 0 0 1 +
   1 1 1 1 1 =
  ---------------
   1 0 0 0 0
```

```
l'ultimo riporto [1] va scartato.
Adesso occorre osservare se il bit più a sinistra del risultato
ottenuto è uguale al bit più a sinistra dei due addendi, se lo è
allora NON si è verificato un overflow ed il risultato è corretto se
è differente allora SI E' verificato un overflow ed il risultato è
errato.
```

Sottrarre numeri binari in complemento a due con bit di segno

Come prima accennato a - b = a + (-b)

Basterà quindi calcolare il complemento a due del numero negativo, in questo caso di b e procedere sommando i due numeri normalmente come appena appreso. Per calcolare il complemento a due di un numero binario bisogna complementare tutti i suoi bit e sommare 1.

esempio:
10000100
complemento tutti i suoi bit
01111011
sommo 1 e ottengo
01111100 che è il complemento a due di 10000100. Facile no? :)

Bisogna prestare attenzione al fatto che ciò è valido se il primo numero è maggiore del secondo. In caso contrario si può procedere analogamente ricordandosi di complementare il risultato ottenuto.

Moltiplicazione tra numeri binari

Per le moltiplicazioni tra numeri binari a n bit si procede in un modo abbastanza semplice, come per la moltiplicazione standard di numeri in base 10 ma ovviamente la moltiplicazione tra bit è differente (a livello puramente teorico) da quella decimale [anche se in pratica è esattamente identica]. Per praticità supponiamo n = 2 ma la regola è identica per un numero di bit qualunque.

Ricordiamoci che in algebra booleana il prodotto è dato dall'AND logico quindi:

0 * 0 = 0
1 * 0 = 0
0 * 1 = 0
1 * 1 = 1
[che non è altro che la tabella di verità dell'AND]

Vediamo allora come procede per moltiplicare il numero a due bit A1 A0 con il numero a due bit B1 B0 [un esempio potrebbe essere 01 * 11]:

```
          A1      A0 *
          B1      B0 =
----------------------
         B0A1   B0A0
 B1A1   B1A0   //
----------------------
 B1A1 B0A1+B1A0 B0A0
```

facciamo dunque un esempio di moltiplicazione:

```
    0 1 *
    1 1 =
-------------
    0 1
  0 1 //
-------------
  0 1 1
```

Dunque 01 * 11 = 011 [c'era da aspettarselo in quanto stiamo moltiplicando 01 (che equivale ad 1 in decimale) per 11 (che equivale a 3 in decimale). Il risultato non poteva essere altro che 3 dunque].

E' importante notare che la moltiplicazione di numeri a n bit produce un risultato di n+1 bit. Nel nostro caso moltiplicando due numeri a 2 bit abbiamo ottenuto un numero a 3 bit.

In sostanza per compiere una moltiplicazione bisogna procedere come in decimale dove il prodotto di bit è dato dal loro AND logico e alla fine sommare i numeri ottenuti normalmente.

Circuiti/Reti combinatorie e sequenziali

Un circuito combinatorio è composto da porte logiche le cui uscite sono, istante per istante, funzioni logiche dei valori assunti dagli ingressi. Essi non contengono cicli. Fanno parte di questo gruppo circuiti come il multiplexer il demultiplexer, i decoder, i comparatori…

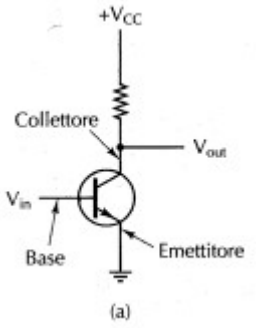
I circuiti sequenziali impiegano elementi di memoria per cui il valore delle loro uscite dipende sia dal valore assunto dagli ingressi sia dai valori memorizzati. I valori memorizzati sono vecchi valori di output, per questo motivo si può dire che i circuiti sequenziali dipendono sia dai valori di ingresso applicati in quell'istante sia dai valori d'ingresso applicati in precedenza. Fanno parte di questo gruppo circuiti come i Letch e Flip-flop.

Circuiti digitali

I circuiti digitali sono componenti che trattano informazioni binarie e sono costituiti da transistor, connessi tra loro in modo da formare i circuiti integrati. I circuiti digitali di base sono chiamate porte logiche. Questi circuiti formano i programmi poiché essi sono entità concrete e tangibili non devono essere né tradotti né interpretati.

Transistor

Può essere considerato come un velocissimo interruttore binario. Esso è fornito di tre piedini denominati collettore (collegato a VCC), base ed emettitore (collegato a massa). Il suo funzionamento dipende dal livello di tensione che è applicato alla base. Se la tensione applicata alla base scende sotto una certa soglia, il transistor si comporta come una resistenza infinita ovvero non viene attraversato dalla corrente, per questo motivo VOUT avrà una tensione circa pari a VCC. Al contrario se la tensione applicata alla base sale sopra un determinato livello il transistor si comporta come un conduttore perfetto e in VOUT si ha un valore di tensione basso.



Letterale

Un letterale è una variabile booleana oppure la negazione di una variabile booleana. (Es: A e $\bar{A}$ sono due letterali diversi).

I circuiti equivalenti

Nome	Forma OR	Forma AND
Elemento neutro	0 + A = A	1A = A
Assorbimento	1 + A = 1	0A = 0
Idempotenza	A + A = A	AA = A
Complementazione	A + $\bar{A}$ = 1	A $\bar{A}$ = 0
Doppia Negazione	not (not A) = A	
Commutativa	A + B = B + A	AB = BA
Associativa	(A + B) + C = A + (B + C)	(AB)C = A(BC)
Distributiva	A(B + C) = AB + AC	A + BC = (A + B)(A + C)
Assorbimento	A (A + B) = A	A + AB = A
De Morgan	Not (A + B) = not A not B	Not (AB) = not A + not B

Due funzioni sono equivalenti se e solo se hanno lo stesso valore di output per tutti i possibili input. La legge di De Morgan può essere estesa a più di due variabili. I cerchietti dell'inversione possono essere spostati a piacere lungo una linea, muovendoli per esempio dagli input agli output o viceversa.

- Una porta logica OR con gli input invertiti è equivalente ad una NAND
- Una porta logica AND con gli input invertiti è equivalente ad una NOR
- Una porta logica NOR con gli input invertiti è equivalente ad una AND
- Una porta logica NAND con gli input invertiti è equivalente ad una OR

Circuiti integrati (IC o chip)

I circuiti integrati sono dei circuiti elettronici miniaturizzati in modo da presentarsi come un unico componente elettrionico e sono composti all'interno da un certo numero di componenti quali ad esempio transistor o condensatori. Sui lati più lunghi sono presenti dei pin che permettono di collegare il chip ad un altro circuito. Il numero di transistor contenuti in un IC ne determina la classe di appartenenza:

- SSI (massimo 10 transistor, Small Scale Integration)
- MSI (massimo 100, Medium Scale)
- LSI (massimo 10.000, Large Scale)

- VLSI (massimo 100.000, Very-Large Scale)
- ULSI (massimo 10 milioni, Ultra-Large Scale)

Esempi di reti combinatorie

Multiplexer (selettore)

E' un particolare circuito integrato capace di selezionare uno solo tra i segnali che riceve in ingresso. La selezione avviene tramite il valore presente nei segnali di controllo di cui è dotato. Quindi ha 2^n pin di input dati, n pin per la selezione ed un'unica uscita. E' possibile utilizzare i multiplexer per implementare un qualsiasi funzione booleana. Per far ciò si imposteranno opportunamente i segnali di controllo affinché indirizzino verso l'uscita l'input corretto.

Demultiplexer

E' l'inverso del multiplexer, quindi ha 1 solo ingresso dati, n selettori che indirizzano il dato in ingresso ad una delle 2^n uscite.

Decoder (decodificatore)

E' un circuito integrato capace di produrre in uscita i 2^n prodotti di n variabili date in ingresso. Poiché ogni espressione booleana può essere espressa in forma di somma di prodotti, è sempre possibile implementarla con un decoder ed una porta OR.

Comparatore

E' un circuito integrato che controlla se due sequenze di n bit ciascuna sono o non sono identiche. Se sono identiche produce 1, altrimenti 0. Si può implementare utilizzando n porte XOR che controllano l'uguaglianza dei bit coppia per coppia. Infine si usa una porta NOR per invertire il risultato (la XOR dà 1 se i due bit sono diversi, per questo è necessario invertire il risultato).

PLA (Programmable Logic Array)

E' un circuito che permette di implementare un certo numero di funzioni in forma di somme di prodotti. Per programmare un tale circuito si devono bruciare dei fusibili che determinano i collegamenti tra le porte. Il PLA è un circuito molto generale che può implementare più di una funzione contemporaneamente.

Come implementare funzioni booleane

Utilizzando un multiplexer

Se volessimo implementare una funzione booleana di 4 variabili in un multiplexer, la maniera più semplice sarebbe quella di utilizzare un mux 16-1 a 16 ingressi e 4 bit di controllo, impostando a 0 o 1 ogni pin di ingresso in base al valore della funzione nella sua tabella di verità. Si capisce subito che per funzioni con più variabili la grandezza del multiplexer cresce esponenzialmente, e quindi i costi di implementazione.

A tal fine è meglio procedere in un altra maniera. Si utilizza un mux più piccolo sfruttando alcune

caratteristiche delle tabelle di verità. Si divide la tabella nel numero di segnali di input di cui dispone il mux. Se il mux dispone per esempio di soli 2 ingressi di controllo e 4 di input, la tabella verrà divisa in 4 parti uguali. Se per esempio volessimo implementare una funzione a 4 variabili, solo le prime due variabili farebbero da veri e propri selettori e sarebbero mandate come segnali di controllo al mux. A questo punto si potrebbe esprimere ogni sotto-parte della tabella (in questo caso avremmo 4 sotto parti) in funzione delle ultime due variabili rimaste. Basterebbe leggere ogni sotto-tabella di verità ed estrarre la relativa funzione booleana, che sarebbe per ovvie ragioni dipendente soltanto dalle ultime due variabili. In questo modo gli ultimi due segnali per le variabili (in unione se necessario ad i segnali 0 e 1), potrebbero essere inseriti in appositi circuiti che implementino le nuove sotto-funzioni trovate. Questo metodo che può sembrare a prima vista molto complicato, è molto utile ai fini del risparmio. Basta solo prenderci un po' la mano.

Le sotto-funzioni derivanti dalle sotto-tabelle di verità possono a loro volta essere implementate con altri mux, eccetera. In questo modo il problema viene diviso e semplificato mediante una tecnica nota anche in elettronica come divide et impera. Il mux più semplice, il 2-1, ha due soli ingressi dati e uno di controllo ed è formato da due porte AND, una NOT e una OR.

Utilizzando un decodificatore

In questo caso l'implementazione è banale. Basta solo esprimere la funzione come somma di prodotti (DNF). Il decodificatore non fa altro che creare la tabella di verità completa. Si indirizzeranno allora le uscite della tabella che danno 1 verso una sola porta OR, la cui uscita sarà l'uscita della funzione che si vuole implementare.

Utilizzando un PLA

L'unica particolarità del PLA sta nel fatto che può implementare più di una funzione contemporaneamente. Il numero massimo è dato dal numero di porte OR che contiene. Infatti come il decodificatore può rappresentare somme di prodotti. Basterà indicare quali saranno i fusibili da bruciare e quali da lasciare attivi.

Circuiti per l'aritmetica

Shifter (registro a scorrimento)

Non fa altro che far scorrere a destra o a sinistra gli n bit dati in ingresso. Dispone ovviamente di n uscite e di un bit di controllo che seleziona se lo shift deve avvenire a destra o sinistra. Quando si effettua lo shift a sinistra, il primo bit viene perso per dare spazio al secondo bit, dall'altra parte invece l'ultimo bit viene impostato a 0 di default. La situazione è speculare per lo shift a destra.

Half adder e full adder

Sappiamo grazie all'algebra booleana che la funzione di somma tra due bit è data dalla tabella dello XOR. D'altra parte quando si sommano due bit si produce anche un riporto. Ed è banale verificare che la tabella di verità del riporto tra due bit è la stessa dell'AND. Implementare un circuito che faccia la mera somma tra due bit e che calcoli anche il suo riporto è a questo punto banale. Si collegano entrambi i bit ad una porta XOR ed a una porta AND. Uno sarà la somma, l'altro il riporto. Questo circuito lo chiameremo half-adder.

In generale quando cominciamo a fare la somma di numeri con più di un bit il riporto va sempre trasferito nelle somme successive verso sinistra. Quindi già dal secondo passo in poi siamo sempre costretti a sommare il riporto che si è generato nel passo precedente. Un circuito completo dovrebbe

considerare anche questa terza variabile, cioè il riporto precedente. Gli output saranno sempre due: la somma dei tre bit in input (bit1, bit2, vecchio-riporto) e il nuovo riporto in uscita. Ma anche qui la cosa è semplice: basta collegare due half-adder in cascata, il primo uguale al precedente, il secondo che prende in input la somma del primo half-adder e il vecchio riporto. Così facendo l'uscita dello XOR del secondo half-adder sarà la vera e propria somma, poi si manderanno i due riporti prodotti dai due half-adder ad una porta OR che ci dirà sempre 1 se almeno uno dei due riporti è stato 1. Ed è esattamente ciò che facciamo quando eseguiamo la somma tra due numeri binari su carta: prima sommiamo i due bit dei numeri, poi gli sommiamo il riporto. Se si è verificato almeno un riporto nelle due mini-somme diamo un altro riporto alla successiva somma parziale.

Full adder a propagazione di riporto e a selezione di riporto

Un full-adder esegue una somma tra due bit e il vecchio riporto. Come fare per sommare numeri ad n bit? Semplice! Si collegano n full-adder in serie! Il riporto si propaga ai full-adder successivi. Questo genere di collegamento ha tanti vantaggi: innanzitutto è facile controllare se è avvenuto un overflow. Infatti basta controllare che gli ultimi due riporti siano uguali, se sono diversi è avvenuto un overflow (e la diversità si verifica come sempre con uno XOR). D'altra parte è possibile anche fare l'incremento di un numero semplicemente impostando ad 1 il primissimo riporto (quello che di solito viene impostato a 0 per fare la semplice somma tra due numeri). Oppure si può calcolare il complemento a due di un numero negando tutte le entrate e sommando uno come detto poc'anzi.

Lo svantaggio sta nella lentezza: il riporto deve essere propagato via via che si somma. C'è una tecnica chiamata selezione di riporto che consiste nel mandare avanti parallelamente due linee di full-adder a propagazione di riporto, le quali fanno una sorta di ipotesi su di un certo riporto ad esempio al centro della fila di full-adder. Uno scommette che il riporto sarà 1, l'altro scommette che sarà 0. Il vantaggio è che una volta che si saprà chi dei due ha vinto, entrambi avranno già finito i loro calcoli sui riporti scommessi e la somma sarà già pronta, selezionando l'uscita del vincitore.

ALU a 1 bit e a 8 bit

Una ALU a 1 bit permette di eseguire operazioni aritmetiche o logiche su due bit in ingresso, calcolando il suo unico risultato in uscita. Ha un'unità logica che consta di varie porte logiche che eseguono tutte le operazioni logiche implementate nella ALU; un'unità aritmetica (full-adder) che esegue la somma tra i due bit in ingresso; un decoder per selezionare quale tra le due unità deve essere attivata sui bit in ingresso.

Semplicemente collegando 8 di queste ALU ad 1 bit, si può formare una ALU a 8 bit, collegando in cascata i riporti (dei full-adder) che ogni ALU produce.

Reti sequenziali

Latch SR temporizzato

Il latch è un circuito di memorizzazione per un singolo bit. E' caratterizzato da tre ingressi (uno dei quali è collegato alla clock, uno viene detto S e 'ultimo R, da cui il nome del latch e del cui funzionamento parlerò tra un istante), e da due uscite, Q e Q'. l'ingresso S sta per "Set", quando viene attivato il latch forza l'uscita Q a 1, e di conseguenza l'uscita Q' a 0. l'ingresso R sta per "Reset" e quando viene attivato il latch forza l'uscita Q a 0, e quindi Q' a 1. Il vero meccanismo di memorizzazione entra in gioco quando sia R che S sono settati a 0, infatti non applicando nessuna operazione il Latch mantiene il suo stato corrente, ovvero sia Q che Q' permangono sui loro valori basilari. Quando la clock vale 0 sia S che R valgono 0 e lo stato del latch non cambia.

E' importante considerare il caso in cui invece sia S che R valgono 1, in questo caso la struttura del Latch diventa instabile e i risultati possono variare senza controllo finanche ad essere determinati in modo puramente casuale. Per questa ragione è stato studiato un nuovo tipo di Latch che espongo qui di seguito.

Tabella caratteristica di un latch SR temporizzato:

S	R	-	OUT
0	0		Q(t)
0	1		0
1	0		1
1	1		ind.

Latch D temporizzato

A differenza del latch SR il latch D (che non è altro che un miglioramento dell'atro) si compone di due soli ingressi: D e la clock. Nel latch D il caso in cui S e R valgono contemporaneamente 1 NON è possibile che si verifichi. Quando D vale 1 il latch D forza Q a 1 e Q' a 0, quando vale 0 forza Q a 0 e Q' a 1. Quando la clock vale 0 il Latch D mantiene il suo stato corrente.

Flip-Flop D

E' un nuovo circuito in grado di memorizzare un bit, esattamente come un latch D temporizzato ma con la differenza che invece di cambiare di stato se il clock vale 1, cambia di stato sul fronte di salita o di discesa del clock, ovvero nel momento in cui il clock passa da 0 a 1 o da 1 a 0 a seconda dei casi.

Flip-Flop JK

E' come un latch SR con la differenza che ammette anche lo stato S = R = 1, in quel caso produce il complemento dello stato corrente.

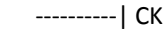
Tabella caratteristica di un flip-flop JK temporizzato:

J	K	-	OUT
0	0		Q(t)
0	1		0
1	0		1
1	1		Q(t)'

Rappresentazione di latch D e flip-flop D in relazione alla transizione di stato

Tutti i latch ed i flip-flop si rappresentano generalmente nello stesso modo, come un rettangolo con un cavo di accesso D, uno di uscita Q (ed eventualmente un'altro di uscita Q') ed un cavo di accesso CK (per la clock). L'unica differenza tra queste varie rappresentazioni è la clock che determina di quale oggetto stiamo parlando.

Se il cavo di accesso alla clock è lineare allora parliamo di un latch D temporizzato con transizione di stato quando la clock vale 1:



Se il cavo di accesso alla clock è lineare e negato allora parliamo di un latch D temporizzato con

transizione di stato quando la clock vale 0:

----------o| CK

Se il cavo di accesso alla clock termina con una virgoletta parliamo di un flip-flop positive edge triggerd a transizione di stato sul fronte di salita del clock:

----------|> CK

Se il cavo di accesso alla clock termina con una virgoletta e una negazione parliamo di un flip-flop negative edge triggerd a transizione di stato sul fronte di discesa del clock:

----------o|> CK

Buffer

Il buffer è una sorta di interruttore costituito da un cavo di ingresso, uno di uscita e uno di controllo.

Quando il cavo di controllo riceve un segnale di corrente alto il buffer è come se non esistesse e la corrente transita normalmente dal cavo di ingresso a quello di uscita. Al contrario se il buffer riceve un segnale basso si comporta come un circuito aperto e la corrente non transita affatto.

Per questa ragione il buffer viene detto "dispositivo a tre stadi" in quanto può far passare tre segnali distinti, 0, 1 oppure addirittura non far passare la corrente e quindi non produrre alcun output.

Oltretutto un buffer amplifica il segnale che riceve e permette che questo non si disperda quindi sono utilizzati moltissimo nei circuiti anche quando non sono necessarie le loro capacità da "interruttori".

Memorie

I registri

Sono costituiti da flip-flop D (uno per ogni bit del registro), riuniti in un unico chip. I clock di questi flip-flop sono tutti collegati tre loro e ricevono un clock dall'esterno, il quale viene negato per amplificarlo, essi condividono anche un segnale di clear (CLR che forza il loro valore in uscita a 0). L'ingresso D e l'uscita Q di tutti i flip-flop sono collegati con l'esterno attraverso pin distinti.

RAM

Le RAM (Random Access Memory, "Memoria ad accesso casuale") sono memorie che vengono sia lette che scritte, esse si distinguono in statiche e dinamiche.

Le RAM statiche (SRAM), sono costruite utilizzando circuiti simili ai flip-flop D e hanno la proprietà di mantenere il proprio contenuto fintanto che vi è alimentazione. Le SRAM sono molto veloci e vengono utilizzate per la memoria cache e i registri.

Le RAM dinamiche (DRAM) sono composte da un array di celle, ciascuna delle quali contiene un transistor e un piccolo condensatore. Il condensatore può essere caricato o scaricato per memorizzare i valori 0 o 1. Dato che la carica elettrica tende a disperdersi, per non perdere i dati è necessario effettuare ogni pochi ms un refresh (ricarica) di ciascun bit. I vantaggi della DRAM sono la maggior capacità e l'elevata densità di celle di memoria che stanno in un chip.

ROM, PROM, EPROM, EEPROM, flash

Queste sono tutte memorie che mantengono i dati memorizzati anche in assenza di alimentazione.

ROM (Read-Only Memory), esse non possono essere ne modificate ne cancellate. Le rom sono molto più economiche delle Ram ma meno flessibili poiché i programmi e i dati vengono inseriti durante la fabbricazione.

PROM (Programmable ROM) essa può essere programmata una sola volta direttamente dall' "utente", bruciando dei fusibili.

EPROM (Erasable PROM) essa può essere sia programmata sia cancellata più volte, essa è infatti dotata di una piccola lente al quarzo che se esposta per un certo periodo ad un intensa luce ultravioletta, forza tutti i bit della memoria a 1.

EEPROM (elettricamente EPROM), possono essere cancellate elettricamente attraverso impulsi elettrici. Questa memoria è riprogrammabile senza estrarla dal circuito, ma purtroppo sono molto meno capienti delle EPROM e veloci solo la metà.

Flash, le memorie flash sono cancellabili a blocchi e riscrivibili. Possono essere cancellate senza doverle rimuovere dal circuito, ma purtroppo si esauriscono dopo 100.000 cancellazioni.

I pin della CPU

Ogni CPU ha un insieme di pin attraverso il quale passano tutte le relative comunicazione verso il mondo esterno, essi sono di tre tipi: indirizzi, dati e controlli.

La CPU per prelevare l'istruzione, ne imposta l'indirizzamento sui pin di indirizzamento, e poi asserisce una o più linee di controllo per informare la memoria che vuole leggere una parola. La memoria risponde spedendo la parola richiesta sui pin della CPU dedicati ai dati e asserendo un

segnale che notifica il completamento dell'operazione.

I pin di controllo si dividi dono in 6 categorie principali:

- Controllo del bus: mandano principalmente segnali della CPU per notificare quando quest'ultima vuole leggere o scrivere dalla memoria oppure compiere altre azioni.
- Interrupt sono normalmente collegati alle periferiche, attraverso questi la CPU comunica a un dispositivo di I/O l'inizio di un operazione.
- Arbitraggio del bus: controllano il traffico sul bus.
- Comunicazione con il coprocessore: comunicano con i chip di virgola mobile.
- Stato.
- Altro: per esempio resettaggio del calcolatore o retro compatibilità.

Bus

Master e slave, driver e ricevitore del bus

Le periferiche che si collegano al bus e che iniziano un trasferimento dati sono chiamate master mentre quelle che si collegano aspettando una richiesta sono chiamate slave. Nel caso di una CPU che vuole scrivere su un disco la CPU è master mentre il controller del disco è slave.

I segnali delle periferiche master sono di solito molto deboli per pilotare il bus, queste periferiche allora sono connesse tramite un dispositivo che serve per amplificare questi segnali che si chiama driver del bus. Mentre gli slave hanno bisogno al contrario di un dispositivo che attenui questi segnali (ricevitore del bus). Le periferiche che fungono sia da master che da slave invece usano un dispositivo che si chiama trasmettitore-ricevitore del bus che esegue entrambi i compiti.

Ampiezza del bus

Nell'ampiezza del bus il fattore più importante da considerare è la sua ampiezza per la quantità di memoria che poi la CPU sarà in grado di indirizzare. Gli aspetti di cui bisogna tener conto sono quindi le linee per gli indirizzi, quelle per i dati. Poi la velocità di trasferimento e la disposizione di tali linee per mantenere la retro compatibilità. Di solito a causa del costo elevato si fanno scelte progettuali che gravano su futuri upgrade. E a causa della retro compatibilità i futuri upgrade renderanno la disposizione delle linee di difficile comprensione.

Bus multiplexato

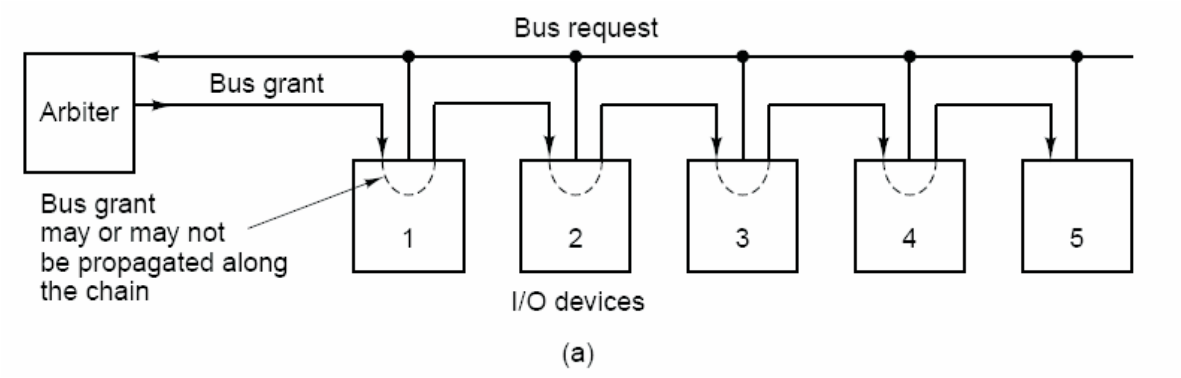
Si ricorre a questa scelta progettuale quando non si vogliono aumentare troppo il numero di linee. Invece di separare le linee dei dati da quelle degli indirizzi queste vengono multiplexate, cioè vengono mandati la prima volta gli indirizzi e poi i dati utilizzando le stesse linee. Questa scelta però dimezza le prestazioni ma diminuisce l'ampiezza del bus e quindi del suo costo.

Bus sincroni e asincroni, pregi e difetti.

I Bus sincroni sono chiamati tali perché ogni trasferimento viene fatto ad un tempo stabilito (si usa come la CPU un oscillatore che genera una frequenza). Mentre quelli asincroni hanno un flusso di dati non costante, usano dei segnali di controllo che permettono la comunicazione. Ad esempio la CPU per richiedere dalla memoria un dato imposta l'indirizzo e poi asserisce un segnale chiamato MSYN. Quando la memoria (che in questo caso ha il ruolo di slave) ha finito il lavoro chiesto asserisce invece SSYN in modo da far capire che la memoria sta inviando i dati richiesti. Una volta che poi la CPU ha letto i dati inviati dalla memoria nega MSYN e di conseguenza poi la memoria vedendo negato MSYN nega SSYN. Questo passaggio di negazioni e asserzioni di MSYN e SSYN si chiama full handshake ed è fatto in modo che i due dispositivi non possano interferire fra di loro. Il pregio dei bus sincroni è che il clock va regolato alla velocità del dispositivo più lento mentre nel bus asincrono non c'è questa esigenza. Ma in compenso i bus asincroni sono più difficili da progettare.

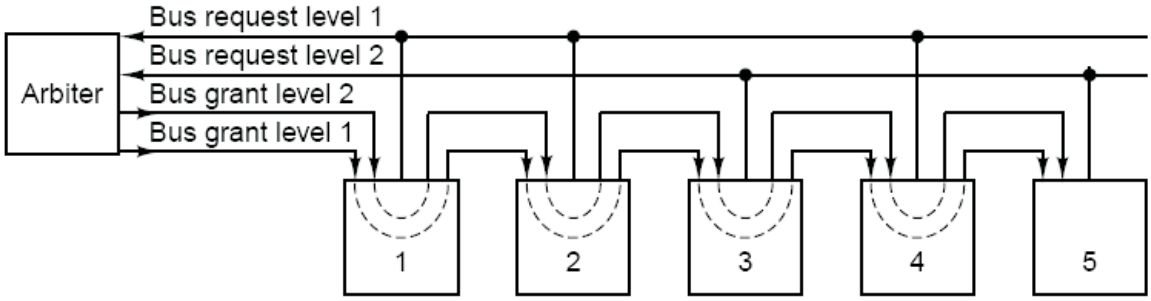
Arbitraggio del bus

L'arbitraggio centralizzato consiste in un chip chiamato Arbitro che dà la concessione al dispositivo più vicino (a patto che qualcuno abbia chiesto di accedere al bus) e se il dispositivo che voleva accedere era il più vicino allora si impossessa del bus, altrimenti passa la concessione agli altri dispositivi. Tale schema si chiama daisy chaining.



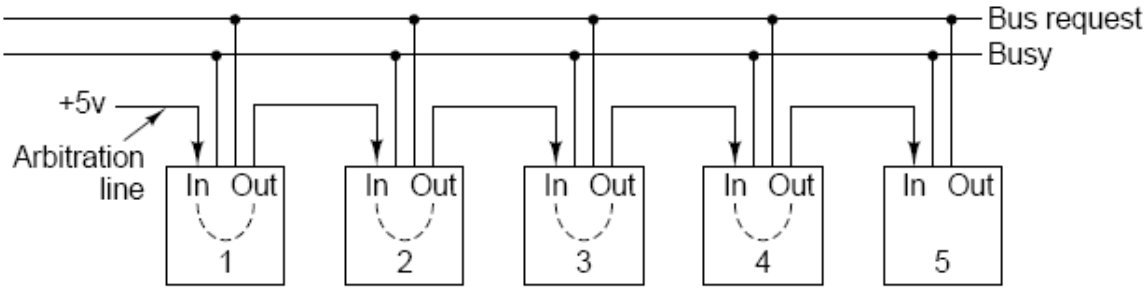
(a)

E' possibile poi dividere le richieste e le concessioni in più livelli (cioè più linee per la concessione e più linee per la richiesta) in modo da avere delle priorità.



(b)

L'arbitraggio non centralizzato invece non ha un dispositivo che fa da arbitro ma ogni dispositivo sa se si può impossessare del bus controllando la linea di Busy. Poi una linea sempre attiva dell'arbitro manda a tutti i dispositivi la concessione della linea. Sarà poi il dispositivo che vuole impossessarsi del bus che asserisce Busy e nega a tutti gli altri la linea di arbitraggio. Questo schema viene pure chiamato daisy chaining ma senza arbitro.



Altre operazioni del bus

Il bus può anche trasferire blocchi di dati piuttosto che singole parole. Questo è utile soprattutto con architetture con cache che non trasferiscono una singola parola ma tutta la linea di cache. Il bus trasferisce in modo efficiente blocchi di dati poiché non c'è da richiedere ogni volta l'accesso al bus o da aspettare che la memoria fornisca nuove parole.

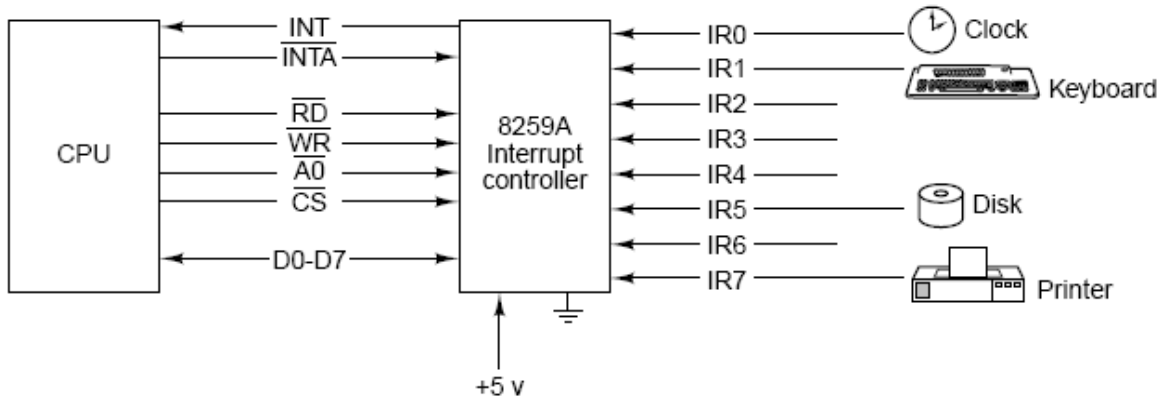
Un'altra operazione che compie è il ciclo di lettura-modifica-scrittura che consiste nel leggere una parola dalla memoria, tenere il bus occupato quando la CPU la analizza e poi scrivere la parola in memoria. Questo ciclo è utile per non permettere ad architetture multiprocessore di scrivere o

leggere sulle stesse parole causando risultati inaspettati.

Un'altra operazione è quella di gestire i segnali provenienti dai dispositivi alla CPU sugli interrupt. Questi segnali servono a comunicare alla CPU che un dispositivo ha finito l'operazione che precedentemente la CPU aveva richiesto. Esistono arbitri che decidono quali interrupt sono più importanti di altri nel caso più di un dispositivo generi un interrupt.

ISA, PCI, chip 8255A

E' un dispositivo per la gestione degli interrupt. Il chip ha il compito di mandare alla CPU l'interrupt nel caso qualcuno l'abbia richiesto. Quando la CPU comunicherà che è disponibile per accogliere l'interrupt il chip manda l'indirizzo del dispositivo che ha richiesto l'interrupt alla CPU.



I/O Mappato in memoria

I/O mappato in memoria consiste nel vedere le periferiche come se facessero parte della memoria centrale. Supponendo che si ha solo come seconda periferica la ROM si potrebbe mappare in memoria la ROM in modo che gli indirizzi che cominciano con 1 sono per la ROM mentre quelli che cominciano per 0 sono per la RAM. In questo modo si vede subito che nasce un primo problema fondamentale che è quello che diminuisce la quantità di spazio indirizzabile. Nelle architetture con I/O mappato in memoria per accedere a parole su dispositivi particolari si usano le istruzioni normali che si usano per accedere a parole che risiedono sulla RAM (quello che cambia è solamente l'indirizzo). Nelle architetture senza l'I/O mappato in memoria invece esistono istruzioni apposite per l'accesso ai dispositivi di I/O. Su piccoli sistemi l'I/O mappato in memoria è una soluzione molto semplice ed economica per risolvere il problema dei dispositivi di I/O ma nei sistemi con molti dispositivi diventa praticamente inattuabile.

In generale quando si ha l'I/O mappato in memoria:

1) Si usa lo stesso bus per indirizzare sia la memoria che gli I/O.
2) Le stesse istruzioni che leggono e scrivono dalla memoria accedono anche agli I/O.
3) Allora è ovvio che aree indirizzabili dalla CPU devono essere dedicate agli I/O invece che alla memoria.

Svantaggi: se si ha già una limitata capacità di indirizzamento, se ne avrà ancora meno per la memoria perché alcuni indirizzi saranno usati per indirizzare gli I/O.

Vantaggi: la CPU è più semplice ed economica, non ha bisogno di bus speciali per collegarsi agli I/O. E ora che ci sono architetture da 32 0 64 bit non si hanno più gli svantaggi prima elencati.