Appunti di architetture degli elaboratori e sistemi operativi

Appunti

Prime lezioni

Il termine informatica trae la sua origine dal francese "*informatique*" ed è stato usato per la prima volta nel 1957 e vuold idre informazione automatica. Al concetto di informatica è strettamente collegato quello di **informazione** che vuol dire ridurre il grado di incertezza. Questa informazioni è misurabile e la sua unità è il **BIT** (Binary digit) che può assumere solo 2 valori, vero o falso.
L'informazione è tratta dalla raccolta e successivo sviluppo dei **dati**, unità fondamentale dell'informazione. Questi dati vengono trattati tramite **sistemi**, *ovvero un insieme di parti eterogenee ed interagenti volte a raggiungere un unico fine*, come disse il padre della teoria dei sistemi generali **Ludwing Von Bertalanffy**.
Il sistema che il nostro ramo della scienza informatica tratta è quello dei sistemi informativi, ovvero quel sistema che gestisce inforamzioni. Il sistema informatico invece è quel sistema che automatizza tutte le sequenza di raccolta e lavorazione dati in quello informativo.
Il calcolatore è un insieme hardware e software che permette a noi la gestione delle informazioni. Ma nel dettaglio questo calcolatore ha bisogno di un ponte che permetta alla parte software di parlare con quella hardware. Il **firmware** è proprio quella parte che ha questo lavoro di permettere alle 2 parti di *convivere*.

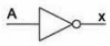
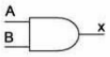
Un po' di storia

Il primo calcolatore venne inventato nel 1642 da Blaise Pascal per fare addizioni e sottrazioni, poi nel 1716 Gottfried Von Leibniz costruì una macchina in grado di fare anche moltiplicazioni e divisioni.Nel 1834, Charles Babbage, realizzò la analytical engine il primo dispositivo in grado di supportare funzioni diverse in base al tipo di programma che veniva caricato. Nel 1925 l'MIT progettò il primo calcolatore a valvole termoioniche e relè. Durante la seconda guerra mondiale spicca Alan Mathison Turing, uno dei primi e veri padri fondatori del computer. Negli anni successivi vediamo l'**ENIAC**, uno dei computer più grandi mai fatti fatto da **Von Neumann** e i suoi consulenti. Von Neumann ha progettato l'architettura della nostra CPU odierna e dei registri necessari al suo funzionamento.

Dopo le valvole i transistor presere il loro posto. Inventati nel 1948 alla Bell Labs il transistor preesnta il primo passo verso la miniaturizzazione. Sempre parlando di miniaturizzazione nel 1958 Robert Noyce inventa i circuiti integrati, che permise di inserire tanti transistor in un solo pezzo di silicio molto piccolo. tutta questa evoluzione è stata anche predetta dalla **legge di Moore** che ancora oggi ci rende partecipi di un concetto che è in continua espansione. La legge dice che ogni 18 mesi la quantità di transistor in un microprocessore sarebbe raddoppiata e infatti continuano ancora a crescere.

Logica cobinatorio e porte logiche

Queste sono le porte logiche su cui si basa la logica combinatoria. Per ora tutte le operazioni principali si fanno con la **XOR, NOT e AND**. La caratteristica principale della **XOR** è che il risultato è sempre vero se in input si hanno valori differenti, è falso se sono uguali. Questo è molto importante perchè nell'addizione binaria la XOR permette di addizionare e riportare il resto degli addendi.

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Operazioni

Quando porti i numeri da una qualsiasi base a binario devi prima di tutto ricordarti che essendo 1 byte devi avere sempre 8 bit come ad esempio 00100000

Conversioni

Ecco qui una tabella con tutto il necessario per la conversione tra basi.

Base	Base finale	Procedimento
Decimale	Binario	Divisione successiva prendendo il resto dal basso verso l'alto
Decimale	Ottale	Prendere ogni singola cifra del numero decimale e trasformala in binario con un massimo di 3 bit, quindi 3 cifre
Decimale	Esadecimale	Prendere ogni cifra edel numero decimale e trasformala in binario con un massimo di 4 bit, quindi 4 cifre
Binario	Deciamle	Puoi usare la ***formaula*** 8 4 2 1 oppure moltiplicare ogni cifra per la base (2) ed elevarla alla sua posizione
Binario	Ottale	Devi dividere il numero binario in gruppi di 3 cifre partendo da destra e trasformare ogni singolo gruppo di bit in numero
Binario	Esadecimale	Devi dividere il numero binario in gruppi di 4 partendo da destra e rappresentare quel numero in formato decimale. Atetnzione perchè l'esadecimale parte da 0 e arrivo fino a F, quindi 15 valori
Ottale	Decimale	Devi passare priama al bianrio e poi al decimale
Ottale	Bianrio	Prendere la singola cifra e rappresentala con 3 bit in binario

Base	Base finale	Procedimento
Ottale	Esadecimale	Devi passare prima al binariio e poi puoi trasformare in Esadecimale
Esadecimale	Decimale	Devi passare prima in binario e poi poi trasformarlo in decimale
Esadecimale	Binario	Prendere la singola cifra e rappresentarla con 4 bit in binario
Esadecimale	Ottale	Devi passare prima in binario e poi trasformarlo in ottale

Addizioni binarie

Dati 2 numeri in input dobbiamo prima di tutti trasformarli in binario e poi possiamo agire. Ecco un po' di regole per non commettere errori:

- Allinea sempre a destra per non commettere errori.
- Segui il principio della XOR e ricordati i ***riporti***.
- I riporti possono essere molteplici e per questo andranno avanti fino a quando non trovi 2 zeri, questo perchè la combinazione 01 con un 1 di riporto continuerà a dare 1 di riporto per la successiva.

Sottrazione binaria

Per fare la sottrazione in binario dobbiamo prima spiegare dei **concetti necessari** ai fini di una corretta esecuzione:

- **Primo concetto** ==> i numeri positivi portano alla loro sinistra uno 0 ad indicare la loro positività, mentre quelli negativi portano un 1. Esempio 76 = **0**1001100 mentre -76 = **1**1001100. Infatti i 2 mìnumerisono uguali tranne per la prima cifra a sinistra che differisce per la diversità di segno.
- **Secondo concetto** ==> Con **complemento a uno** definiamo l'operazione necessaria per la trasformazione di un numero da positivo a negativo. Ovvero prendiamo il numero in bianrio e sostituiamo ogni zero con un uno e viceversa. Esempio -76 = -01001100 == 10110011 così otteniamo il numero negativo in binario.
- **Terzo concetto** ==> Con **complemento a due** definiamo l'addizione di 1 al numero binario negativo ottenuto con il commplemento ad 1. *Bada bene che entrambi i complementi sono necessari alla risoluzione di una sottrazione binaria*. Esempio -76 = -01001100 == 10110011 + 1 = 10110100
- **Quarto concetto** ==> È anche il più *difficile da applicare*, ovvero l'**Eccesso 2^m-1** dove possiamo rappresentare i numeri come somma di se stessi con 2^m-1 dove m è il numero di bit utilizzati per rappresentare il valore. Esempio: supponendo di volere -76 dobbiamo pensare che il suo range di esistenza è quella di -128 e +127, quindi 255 che è **2^8**. Quindi abbbiamo bisogno di 8 bit per la rappresentazione. Ora **eleviamo il 2 a m-1** dove m è il numero di bit (8) e otterremo **2^7 = 128**. Con questo 128 andiamo a fare -76 + 128 = 52 che in binario è 00110100ed infatti rispecchia il complemento a 2 ma positivo.

- **NOTA BENE!** Quando poi andrai a risolvere una sottrazione dovrai ricordarti che il numero massimo di bit a tua disposizione sono 8 e che quindi se il riporto arriva fino alla **cifra numero nove dovrai ignorarla**. Ecco un esempio completo di sottrazione binaria.

	(Decimale)	*(Binario)*
Addendo	10	00001010
Addendo	-3	11111101
Somma	7	00000111
Riporti		**1**1111000

Ignorato

Ora con un po' di attenzione la posiamo risolvere da 0. Prima cosa si **trasformano entrambi i numeri** in binario con un massimo di **8 bit**. Il numero 10 è positivo e quindi in binario sarà 00001010, il numero 3 pur essendo negativo lo prendiamo per positivo e lo trasformiamo in bianrio ottenendo 00000011. Ora con il **complemento ad uno** andiamo a sostituire gli 0 con 1 e viceversa, quindi otteniamo 11111100. Adesso con il **complemento a due** andiamo a sommare 1 al prodotto del complemento ad uno ottenendo ***11111101 che è il nostro valore negativo***. Passiamo a fare la somma e alla fine ignoriamo il riporto finale perchè sfora il nostro limite di bit, ovvero 8. Ecco che il numero 00000111 è uguale a 7 in binario.

Moltiplicazione binaria

La moltiplicazione in binario si può ricondurre a una serie di addizioni successive che hanno come moltiplicando e moltiplicatore solo il vero moltiplicando della moltiplicazione stessa. Il concetto può essere un po' dispersivo ma spero di riuscire a spiegarlo nel migliore dei modi. Prendiamo per esempio una moltiplicazione come 12 * 6 e dividiamo tutto in step per vedere come funziona.

- **Step 1** ==> Individuiamo il **moltiplicando e il moltiplicatore** che, in questo caso come moltiplicando è 12 e moltiplicatore è 6.
- **Step 2** ==> Trasformiamo entrambi i numeri in binario ottenendo rispettivamente **12 = 1100** e **6 = 0110**.
- **Step 3** ==> Adesso tutto si concentra sul moltiplicando, ovvero 1100 (12). Controlliamo quanti 1 sono **presenti nel moltiplicatore**. Da qui in poi possiamo avere **due esiti**, ovvero quello dove **l'1 occorre almeno 2 volte** oppure dove è **presente un solo 1**. Ma per adesso prendiamo in considerazione solo la prima possibilità, ovvero quella dove gli 1 sono >= 2.
- **Step 4** ==> Ora dobbiamo procedere con una sola addizione se il numero di 1 presenti è uguale a 2, con 2 addizioni successive se con 3 occorrenze del numero 1 e così via. Per poter svolgere le addizioni vediamo il **posto della prima occorrenza dell'1** (partendo da destra verso sinistra e da 0 a +∞ con i posti) nel moltiplicatore. Se maggiore di 0 (perchè con 1 presente in prima posizione non si cambia niente poichè dovremo inserire 0 volte 0 a destra del moltiplicando) andremo ad aggiungere un numero di 0 uguale al numero della posizione in cui si tova il primo 1 e poi tutti i successivi. **Esempio**: in questo caso il numero 0110 presenta il primo 1 in posizione 1 quindi

aggiungiamo uno zero a 1100 facendolo diventare 11000 e lo andiamo a sommare a 1100 con l'aggiunta della seconda ricorrenza di 1 che è in posizione numero 2. Quindi avremo 11000 + 110000 = 1001000 che in decimale è 72. Con un numero di occorrenze di 1 maggiore a 2 basta continuare l'addizione sucessiva fino all'ultima ricorrenza.

- **Step 4.5** ==> Questo caso invece è per quando si verifia **una sola occorrenza di 1**. Per esempio nella moltiplicazione di 12 * 8 abbiamo 1100 e 0100. In questo caso basta aggiungere tanti 0 quanti precedono l'1 nel moltiplicatore e si otterrà il quoziente. In questo caso il quoziente è 110000 quindi in decimale 96 che è proprio 12 * 8.

Notazione scinetifica

Per la rappresentazione di numeri molto grandi o molto piccoli utilizziamo la notazione scientifca per poterli rappresentare in modo più veloce come per esempio un numero che è seguito da 10 zeri verrà scritto così:

$$n * 10^{10}$$

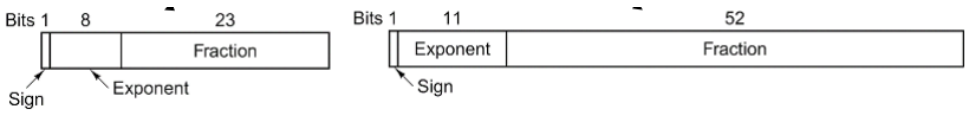
Questo tipo di rappresentazione permette in particolar modo di rappresentare i bit in virgola mobile (floating point) e anche di poterli scrivere in **codifica standard**. Ecco alcuni esempi di codifica standard:

$$3.14 = 0.314 * 10^{1}$$
$$0.00001 = 0.1 * 10^{-4}$$
$$2^{16} =$$

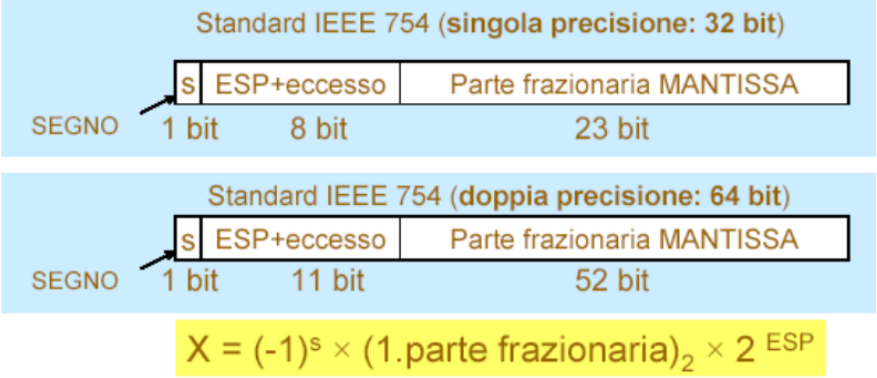
La **mantissa** (parte decimale) deve essere compresa tra 0.1 <= f < 1.
Con i float non è possibile rappresentare tutti i numeri reali perchè la densità di numeri stessi nonè infinita, ecco perchè nummeri periodici non opssono essere rappresentati e quindi vanno arrotondati.

IEEE 754

Con l'avvento dello standard **IEEE 754** negli anni 80' abbiamo un formato standard di rappresentazione dei numeri floating point binari.



Elemento	Singola Precisione	Doppia Precisione
Numero bit nel segno	1	1
Numero bit nell'esponente	8	11
Numero bit nella mantissa	23	52
Numero bit totale	32	64
Rappresentazione dell'esponente	Eccesso 127	Eccesso 1023
Campo dell'esponente	Da −126 a +127	Da −1022 a +1023
Numero più piccolo	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Numero più grande	$2^{127} \approx 10^{38}$	$2^{1023} \approx 10^{308}$



$$X = (-1)^s \times (1.\text{parte frazionaria})_2 \times 2^{ESP}$$

ATETNZIONE La formula in giallo permette di determinare il numero originale partendo da un numero standard.

Con *Singola precisione e Doppia precisione* vogliamo dire le architetture delle CPU, ovvero quella a 32 e 64 bit. Con differenti architetture cambiano i canoni da rispettare per la rappresentazione dei bit.
Nel caso del singola precisione vediamo come il primo bit è sempre quello che definisce il segno del nostro numero a virgola mobile, quindi 1 per negativi e 0 per positivo. Poi abbiamo **l'eccesso** che viene calcolato **normalizzando** il nostro numero trasformato in binario e, sommando l'esponente al creiterio di rappresentazione preso come standard (nel caso del singolaprecisione abbiamo 8 bit a disposizione quindi 2 elevato a 8 che sono 255. Ma in realtà è un range compreso tra -128 e +127. Mentre in quella a doppia precisione abbiamo 11 bit quindi 2047 che sono rispettivamente -1024 e +1023) che in questo caso è 127 otterremo 139 e, trasformandola in binario avremo il nostro eccesso. Ora con i bit rimasti a disposizione possiamo riportare la parte frazionaria e inserire tanti 0 quante sono le cifre mancanti per arrivare a 23 bit. *Sulle dispense è anche presente un esercizio.*
Stessa cosa vale per il prcedimento inverso, ovvero inizio col prendere il bit di segno e l'esponente. Con l'esponente in bianrio faccciamo un passaggio in decimale e poi sottraiamo 127 per ottenere l'esponente vero del nostro numero. Ora riportiamo la mantissa e moltiplichiamo per 2 elevato l'esponente per avere il numero in binario. Ora con la conversione abbiamo il nostro numero decimale.

ASCII

Il codice ASCII è stato un metodo sviluppato per risolvere un problema nato con la diffusione del calcolatore a livello mondiale, ovverlo la trascrizione delle lettere dell'alfabeto e delle caratteristiche dei singoli caratteri di ogni lingua. All'inizio l'ASCII veniva rappresentato con 7 bit e 1 bit di parità. Poi è arrivato l'ASCII esteso con 8 bit (1 byte) per rappresentare il doppio dei caratteri. Ma, con l'evoluzione delle lingue e l'aumento dei caratteri la codifica ASCII non è bastata e quindi siamo arrivati ad usare l'**UNICODE** con ben 2 byte a disposizione per rappresentare i caratteri, ovvero 2 elevato 16 che sono 65.536 caratteri. In ogni caso anche con l'UNICODE standard non avevamo abbastanza bit per rappresentare tutti i caratteri a livello mondiale, per questo vediamo l'**UTF-8** che, a differenza degli altri standard, questo è dinamico e la quantità di bit utilizzati può cambiare da un minimo di 8 a un massimo di 32 (quindi da 1 a 4 byte). Ha anche un controllo sulla concatenazione dei bit dove i 2 bit più a sinistra di un ottetto (con ottetto definisco un gruppo di 8 bit) possono avere valore 10 o 0. Con 0 avremo sapremo che quel byte è la *testa* e che dietro di se ci possono essere altri byte. Invece con 10 sappiamo con certezza che quel byte è seguito da almeno un altro byte.

Le immagini

Partiamo dalla definizione di immagini di tipo **raster** e **vettoriale**.

- Con raster definiamo quell'immagine che è un continuo o meglio un intero. Non ha una rappresentazione reale e quindi tende a perdere di chiarezza se modficata.
- Con vettoriale definiamo quell'immagine che rappresenta forme geometriche che anche se cambiate nello spazio avrenna sempre la stessa forma e valenza.

Nelle immagini digitali il **PIXEL** è quel singolo ***quadratino*** in cui viene scomposa l'immagine. Questi sono essenziali nella rappresentazione delle immagini. Infatti nella fase di rappresentazione troviamo due fasi, ovvero la **discretizzazione** e la **quantizzazione**.

- La fase di discretizzazione prevede i processi di **definizione** e **risoluzione**.
-
    - Il primo definisce il numero di pixel chiamati **DPI (dots per inch)**.
-
    - La risoluzione invece rappresenta la dimensione della nostra *griglia* come per esempio 1920x1080.
- La fase di quantizzazione prevede la rappresentazione di ogni pixel con una sequenza di bit. Per esempio in un'immagine in bianco e nero useremo 2 bit la rappresentazione, quindi i pixel bianchi verranno quantificati con 1 e quelli neri con 0. Se invece l'immagine fosse a colori, in modo analogo assegneremo ad ogni pixel una sequenza di bit per definire il colore rappresentato. Quando andiamo ad utilizzare i colori ci basiamo sui 3 colori generatori **RGB (red, green, blue)** e quindi se assegnassimo 8 bit per ogni colore andremo a rappresentare ogni pixel con 3 byte.
  Anche per le immagini il concetto di ridurre spazio di archiviazione vale e infatti vediamo che molte sono le estensioni delle nostr immagini. La differenza essenziale tra ogni estensione è l'utilizzo dell'immagine stessa e il peso (ovvero la quantità di colori utilizzati o **palette**).
  Le immagini **vettoriali** si affidano ad equazioni matematiche per disegnare le imamgini, per questo è *flessibile* e non si perde qualità quando la si ingrandisce o rimpicciolisce. I formati più comuni sono l'**SVG, JPG e PNG**. Una caratteristica molto importante dei vettoriali è che possono essere trasformati in raster ma non il contrario.

Algebra Booleana

Teoria

Ecco le leggi basiche della matematica **booleana**

Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	AB = BA	A + B = B + A
Associative law	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive law	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption law	A(A + B) = A	A + AB = A
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$
Absorption law 2	$A + \bar{A}B = A + B$	

Queste sono le leggi fondamentali per l'algebra booleana. Ogni formula può essere spiegata con addizioni e sottrazioni in colonna con la scomposizione della singola formula

La **semplificazione di funzioni booleane** è un processo che permette la riduzione di una funzione booleana in una equivalente così da essere rappresentabile graficamente e avere una migliore rappresentazione tramite le tabelle di verità.

f(x, y, z) = $\bar{x}y\bar{z} + \bar{x}yz + yz =$

$\bar{x}y(z + \bar{z}) + yz =$

$\bar{x}y + yz =$

$y(\bar{x} + z)$

f(x, y) = $x + \bar{y} + \bar{x}y + (x + \bar{y})\bar{x}y =$

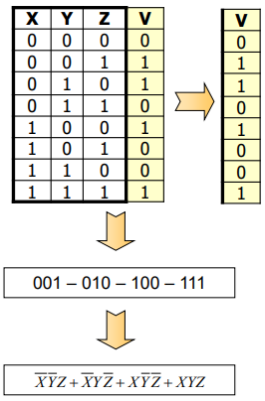
$x + y + \bar{y} + (x + \bar{y})\bar{x}y =$

$x + 1 + (x + \bar{y})\bar{x}y =$

1

Tramite questaimmagine vediamo 2 funzoini che vengono ridotte al minimo. Generalmente si usano le formule matematiche classiche come la raccolta o scomposizione di singole funzioni.

Con tabella di verità definiamo tabelle che permetono di verificare tutte le conbinazioni vere dati in input n valori.



Tramite questa tabella infatti vediamo che dati in input 3 valori avremo alla fine solo 4 possibili combinazioni. Il numero di combinazioni è dettato dal contesto della nostra tabella. Poi possiamo trasformare le combinazioni in espressione negando le variabili con valore 0 e prendendo quelle con valore 1 positive e mettendole insieme con OR (+).

Il calcolatore

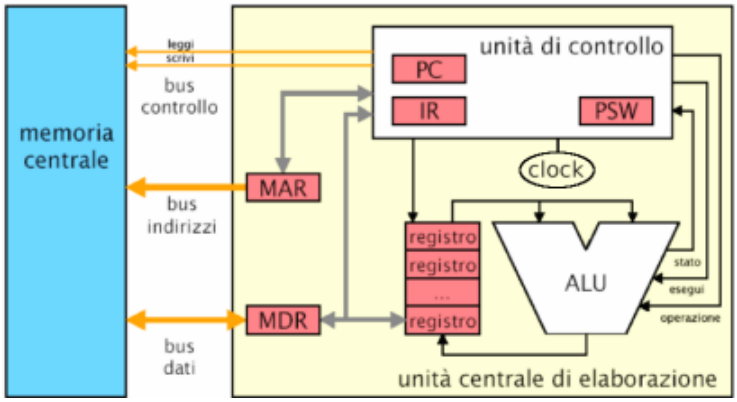
Con calcolatore definiamo una macchina che può elaborare, memorizzare e trasferire dati, tutto questo in modo controllato. Il primo calcolatore venne inventato da Von Neumann che lo rese possibile grazie ai componenti principali, ovvero **la CPU, memoria e bus** (collegamenti fra le varie unità). Con bus in realtà definiamo una specie di strade dove i dati passano. La gestione dei bus e dei dati è fatta dalla CPU (master) che gestisce il flusso di dati alle altre unità (slave). Di bus ne esistono di 3 tipi e sono:

- **Bus dati** dove viaggiano in ingresso e in uscita i dati dalla CPU;
- **Bus indirizzi** che trasmette l'indirizzo della locazione di memoria ed è unidirezionale dalla CPU verso l'esterno.
- **Bus controllo** è un insieme di linee che svolgono diversi compiti raggruppati in un solo bus che invia segnali alla CPU.

La scheda madre

La scheda madre è una componente principale che regge tutto il nostro calcolatore e permette alla CPU e agli altri componenti di interagire tra di loro. Molto importanti sono i connetori interni o meglio **PCI (Peripheral Component Interconnect)**. Di questi ne esistono 2 tipi, il PCI e il PCIe che differiscono in larghezza di tramissione e comunicazione in serie e parallelo. Sulla scheda madre troviamo anche tutti i connettori per le periferiche come gli USB, cavi lan, hdmi...

La CPU



La CPU ha come compito quello di eseguire i comandi dei programmi immagazzinati nella memoria centrale in linguaggio macchina. Ma la CPU non è un unico componente, al suo interno ci sono molte parti, come i registri che sono delle memorie molto veloci, che gli permettono di lavorare. Queste sono:

Nome	Descrizione
Unità di controllo	Legge les istruzioni della memoria centrale e ne determina il tipo
ALU	Esegue le operazioni necessarie all'esecuzione delle istruzioni (Aritmetic Logic Unit)
PC	Programm Coutner che mantiene in memoria l'indirizzo di memoria della prossima stringa da far eseguire

Nome	Descrizione
IR	Instruction Register che immagazzina una copia dell'istruzione che sta venendo elaborata
MAR	Memory Address Register contine l'indirizzo della memoria RAM dal quale si andrà a prendere o scrivere un dato
MDR	Memory Data Register contiene momentaneamente i dati per la CPU, quseto perchè è una memoria molto veloce ad accesso diretto
PSW	Processor Status Word indica l'esito di operazioni aritmetiche

La grandezza dei regisri può variare e questo è determinato dall'**architettura** o meglio **word** della CPU stessa. La word indica la dimensione dei registri che in quelli odierni può essere a 32 o 64 bit.
Quando la CPU esegue dei processi lo fa in modo ciclico, ovvero si basa su quattro operazioni che vengono ripetute fino alla fine dell'esecuzione del programma. Queste operazioni sono:

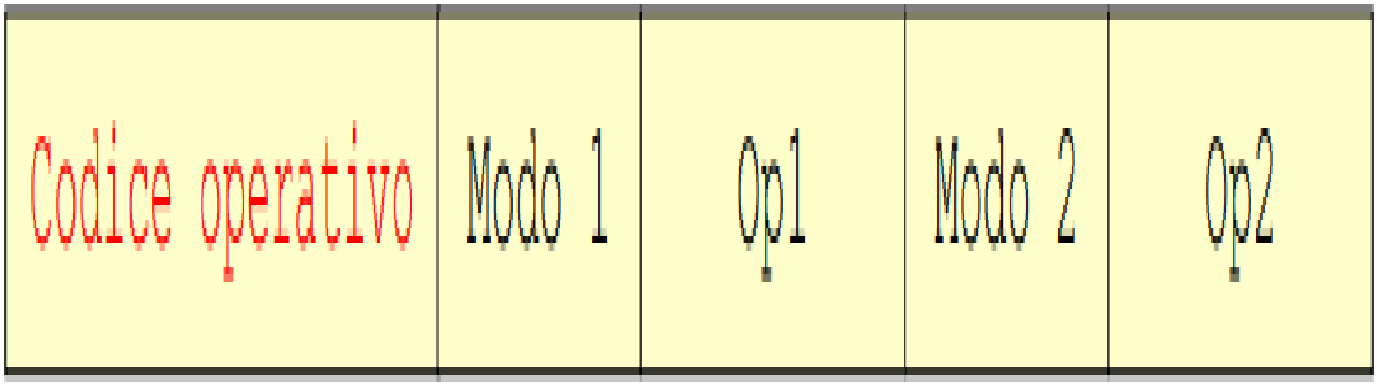
- **Fetch** (Caricamento) dove acquisisce dalla memoria un'istruzione del programma;
- **Decode** (Decodifica) che identifica il tipo di operazione da eseguire;
- **Execute** (Esecuzione)effettua le operazioni corrispondenti all'istruzione.
  Queste operazioni però sono in realtà più articolate e richiedono l'utilizzo di tutti i registri. Il processo di esecuzione è il seguente:
  Alla CPU viene dato il sesgnale di avvio, al Programm Counter viene assegnata la prima istruzione del programma. La CPU imposta nel MAR l'indirizzo contenuto nel PC e viene attivata la modalità di lettura sul bus di controllo. Grazie al MAR i bus vanno a prendere le istruzione dalla RAM. La CPU imposta nell'IR l'indirizzo del MDR e **qui finisce la fase di fetch**. La CPU incrementa il PC in modo che punti alla prossima istruzione da eseguire ed esamina l'IR e determina la prossima operazione da svolgere, **questa è infatti la fase di decode**. Tutte le unità richiamate dalla CPU si attivano ad eseguire il programma fino alla fine, andando a prelevare dati dalle memorie o anche da periferiche. Una volta terminata la fase di execute si torna alla fase 2.

Assembly

La CPU permette di eseguire le applicazioni in memoria centrale. Grazie a un **assembler** i programmatori possono dare istruzioni alla CPU traducendo il loro codice sorgente in linguaggio macchina. L'**assembly 8086** è un tipo di linguaggio che lavoro sui registri e permette di fare operazioni elementari con la CPU.

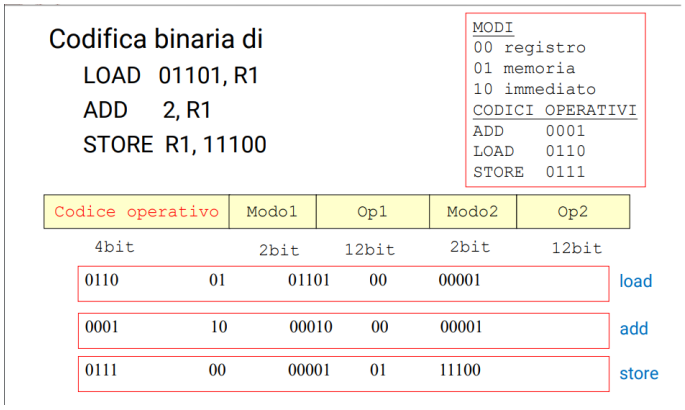
Un esempio di programma assembly può essere l'addizione x = y + 2 ottenendo:
LOAD Y, R1
ADD 2,R1
STORE R1,X
Questo codice permette di caricare il valore della variabile Y nel registro R1, sommare 2 al registro R1 e infine memorizzare il valore di R1 sulla variablie X. Tutto questo però deve essere espresso in linguaggio riconoscibile per la macchina, quindi in binario. Iniziamo traducendo la Y con 01101 e X con

11100, ottenendo così l'inidirzzo di memoria delle 2 variabili. Ora dobbiamo definire un formato semplificato per trascrivere queste informazioni alla macchina.



Questo è l'assegnamento dei bit e la loro versione semplificata. Ora però dobbiamo definire quali sono i pisto per i singoli bit.In Assembly dobbiamo definire che le istruzioni vengono lette in modo differente dalla macchina(naturalmente la lunghezza di queste stringhe varia in base alla *word* del processore, quindi il numero di bit del processore stesso). Per esempio dati dei comandi per fare un'addizione vedremo che il codice finale letto dalla CPU sarà formato da:

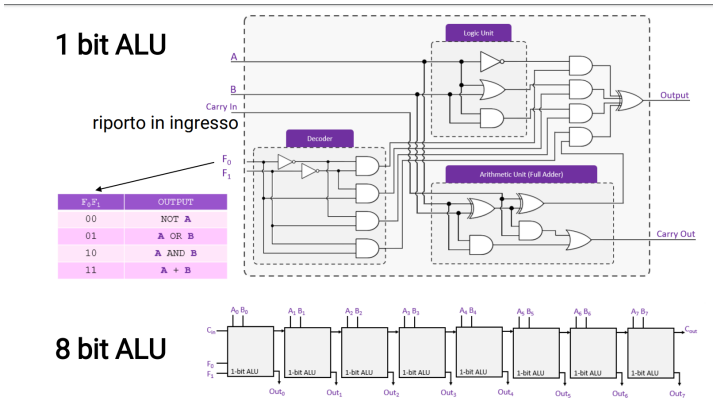
- **Codice Operativo** che codifica i singoli operatori in binario, quindi per esempio l'operatore ADD può essere scritto come 0001;
- **Modo N** indica dove si trova l'operando in base a una tabella, ovvero **00 per registri, 01 per valroi in memoria e 10 se un valore dato in input**;
- **Op** indica la posizione dove andare a prendere l'operazioni che deve andare a fare la CPU.
  ESEMPIO A PAGINA 30 LUCIDI 5



Il processore può fare tutto questo seguendo 3 fasi, ovvero:

- **Fetch** dove legge una nuova istruzione da eseguire dalla memoria centrale;
- **Decode** dove risale alla operazione decodificando i bit;
- **Execute** dove porta a termine l'operazione richiesta.

Questa immagine rappresenta come l'ALU esegue delle operazioni.



Vediamo come possiamo avere in entrata solo 2 valori, A e B, insieme a un **Carry in** ovvero il resto dell'operazione precedente. Il Entrata nel **decoder** abbiamo F0 e F1 che sono una sequenza di bit che permettono di capire quale operazione verrà svolta. Come output avremo il valore finale della nostra operazione e il **Carry out**, ovvero il possibile resto dell'operazione.

Come dovrebbe funzionare l'addizione seguendo tutto il processo interno. Prendendo ADD 2,R1 come esempio avremo che l'istruzione verrà presa dalla memoria (fetch) tramite il MAR e letta dal **PC** come indirizzo di memoria. Ora tramite il bus dati il valore codificato in binario del codice sorgente e, tramite l'**MDR** viene messo nell'**IR**.Ora il tutto viene mandato all'**ALU** e il valore 2 viene messo nello **stack dei registri** insieme al registro R1. Tramite la fase di esecuzione l'**ALU** esegue l'operazione e inserisce il risultato nel registro R1.

Il processore esegue le istruzioni in modo sincorna, ovvero esiste un segnale (**tick**) che scandisce l'inizio di un'operazione ed è riconosciuto da tutti i componenti. La frequenza con cui viene inviato il tick è misurata in *numero di tick al secondo* e si misura in **Hertz**.
La **frequenza di clock** indica il numero di cicli di clock per secondo.

Esistono due correnti di pensiero per la modalità di esecuzione delle istruzioni della CPU.

- **CISC**(Complex Instruction Set Computer) sostiene che il set di istruzioni di un compilatore debba contenere quante più istruzioni possibili;
- **RISC**(Reduced Instruction Set Computer) sostiene che ogni istruzione debba essere eseguita in un solo cilco sebbene saranno necessarie più istruzioni RISC.
  Con il tempo si è arrivati ad avere la necessità di migilorare le prestazioni delle CPU, per questo siamo arrivati ad avere il **parallelismo** e il **pipelining**.
  Per parallelismo abbiamo quello a livello di **processore** (fisico) dove più CPU cooperano per la soluzione del problema e quello di **istruzione** (virtuale) dove più istruzioni vengono eseguite contemporaneamente all'interno della stessa CPU.
  Il **pipelining** invece divide ogni istruzione in più fasi e viene gestita da un hardware dedicato. Questo permette di eseguire più istruzioni allo stesso tempo. Bisogna fare attenzione però alla dipendenza delle singole istruzioni, ovvero se un'istruzione dipende da quella precedente, questo perchè in questo caso non potrà essere eseguita perchè in attesa di input.

$$T_{esecuzione} = T_{clock} \cdot \left(\sum_{i=1}^{n} N_i \cdot CPI_i \right)$$

- T_{clock} -> periodo di clock della macchina
  dipende dalla tecnologia e dalla organizzazione del sistema
- CPIi -> numero di clock per istruzione di tipo i
  il numero di clock occorrenti affinché venga eseguita l'istruzione di tipo *i*. Se il set di istruzioni contiene istruzioni di tipo semplice si ha un CPI basso, se invece le istruzioni eseguono operazioni più complesse si ha un CPI elevato
- Ni -> numero di istruzioni di tipo i (somme , sottrazioni , salti, ecc.)
  dipende dal set di istruzioni e dalla qualità dei compilatori (ossia la capacità dei compilatori di ottimizzare il programma). Se il set di istruzioni contiene istruzioni di tipo semplice si ha un N_i alto; caso opposto per istruzioni di tipo più complesso che permettono di abbassare N_i. Infatti, con un set di istruzioni di tipo semplice ne occorrono di più per svolgere le stesse funzioni.

La logica dei calcolatori RISC è quella di avere un set di istruzioni molto semplici che mantengono basso il CPI a scapito del valore di N_i, mentre quella dei calcolatori CISC è esattamente opposta.

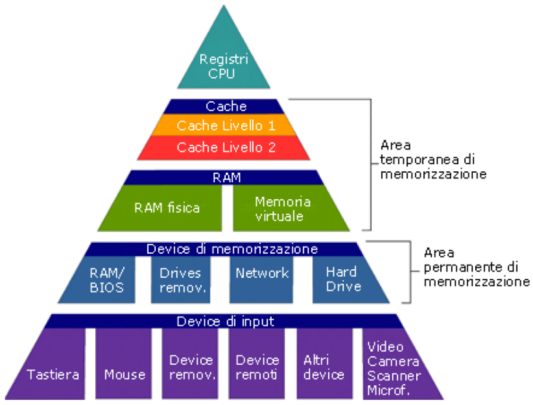
Piccolo schema sul calcolo del periodo di clock

La memoria

Introduzione

La memoria in un qualsiasi tipo di device elettronico è il posto in cui vengono memorizzate le informazioni. Di memorie però ne esistono di diversi tipi e con caratteristiche differenti che le contraddistinguono, alcune delle caratteristiche fondamentali sono:

- **Velocità**
- **Dimensione**
- **Permanenza dei dati**, ovvero che mantengono il contenuto anche senza alimentazione
  Queste caratteristiche però non sono conciliabili tra i diversi tipi di memorie disponibili sul mercato, per questo dobbiamo distinguere le memorie in due famiglie, ovvero le **Memorie centrali** e le **Memorie di massa**.
  Le prime, ovvero le centrali (RAM) , sono memorie molto veloci che salvano i programmi in esecuzione con i relativi dati, ma non hanno permanenza. Quelle di massa (HDD) invece sono generalmente più lente di quelle centrali, ma con una capienza massima molto più grande. Inoltre sono **persistenti**, ovvero che anche senza alimentazioni mantengono i dati al loro interno.



Le memorie si possono distinguere anche per le tecnologie come:

- **memorie elettroniche**, ovvero con velocità elevate ma necessitano di alimentazione (memoria centrale);

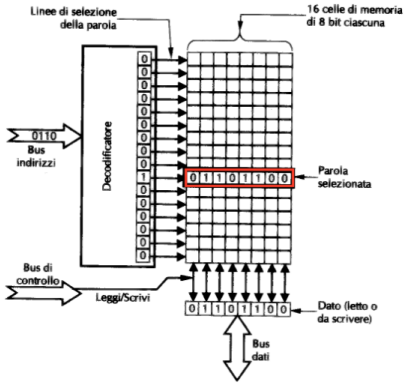
- **Memorie magnetiche**, ovvero permanenti e più lente (memoria di massa);
- **Memorie ottiche**, ovvero delle memorie riscrivibili come i CD e DVD

La memorie è strutturata in una successione ordinata di elementi binari, ogni unità è detta cella ed ha un indirizzo ben specifico per essere riconosciuta. La dimensione massima della memorie indirizzabile dipende dal numero di bit utilizzati per codificare gli indirizzi.

L'accesso alle celle di memoria può avvenire in diversi modi, ovvero:

- **Acesso sequenziale**, l'accesso a un dato comporta la lettura di tutti quelli che lo precedono;
- **Accesso casuale** il tempo di accesso non dipende dalla posizione del dato ed è costante per tutte le celle;
- **Accesso misto** dove la posizione della cella non è determinabile nello specifico e per questo si accede in prossimità della cella e si controllano tutte le celle in modo sequenziale (dischi);
- **Accesso associativo** è un metodo ad accesso casuale nel quale le celle vengono selezionate in base al loro contenuto anzichè utilizzando un indirizzo (memoria cache)

A livello fisico l'accesso alla memoria centrale avviene tramite il *MAR* (Memory Address Register) e si prelevano o scrivono dati tramite l'*MDR* (Memory Data Register) in base al segnale definito dal bus di controllo (il segnale può essere lettura o scrittura). In un computer odierno lo spazio di indirizzamento è di almeno 32 bit (dimensione del MAR) e quindi possiamo indirizzare 2^32 bit quindi 4 GB.



Tipologie di memorie

Nome	Descrizione
RAM	**Random Access Memory**, memoria ad accesso casuale di tipo volatile (si resetta ad ogni avvio del PC)
ROM	**Read Only Memory**, memoria non volatile scritta al momento della produzione e che non può essere cancellata. Utilizzata per contenere operazioni del BIOS
Memorie Flash	Memorie elettroniche non volatili ma riscrivibili. Una pendrive è un tipo di memoria flash
SIMM	**Single Inlne Memory Module**, un tipo di RAM basata su condensatori che immagazzinano un bit per condensatore. Se il condensatore perde la carica l'informazione è perduta. Ha ampiezza di bus di 32 bit e sono usate per i bus dati

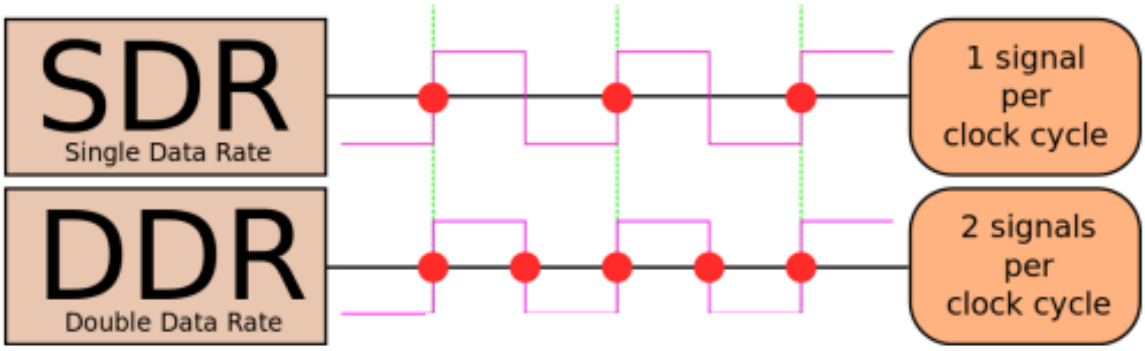
Nome	Descrizione
DIMM	**Dual Inline Memory Module** sono RAM con i connettori su entrambi i lati della scheda e un'ampiezza del bus a 64 bit
SO-DIMM	**Small Outline Dual In-Line Memory Module** è un formato di RAM diffuso nei computer a circuito integrato come portatili, router e stampanti

Tutte queste memorie hanno dei parametri che le rendono differenti per il loro scopo, abbiamo infatti:

- **Tempo di accesso** ovvero il tempo necessario a rendere disponibile un valore in una cella di memoria;
- **Ciclo di memoria** valutato sommando al tempo di accesso la durate del *periodo refrattario*, ovvero il tempo in cui la memoria non è utilizzabile;
- **Velocità di trasferimento** ovvero la quantità di dati trasferiti nell'unità di tempo, si misura in bit/sec.

Sul mercato odierno abbiamo due tipi principali di RAM, ovvero le **SDR** e le **DDR**.

- Con **DDR** (Double Data Rate)indichiamo le RAM più performanti al giorno d'oggi, questo eprchè possono trasferire un byte sil sul fronte di salita che su quello di discesa del clock. Al giorno d'oggi contiamo 5 generazioni di DDR, la più recente è la DDR5.
- Con **SDR** (Single Data Rate) invece, possiamo trasferire dati solo sul fronte di salita del clock.



Controllo degli errori

Oggi abbiamo RAM molto affidabili, ma nonostante ciò i sistemi effettuano un controllo all'avvio usando la tecnica del **bit di parità**, ovvero un controllo dove si aggiunge un bit alla fine di una sequenza. Queto bit può avere valore 1 se è presente un numero di 1 dispari, altrimenti varrà 0.

Memoria cache

La **memoria cache** è un tipo di memoria molto veloce che troviamo vicino alla CPU, questo perchè permette di salvare le istruzioni da svolgere in modo che il processo non diventi più lento andandole a cercare nella memoria centrale, questo perchè sarebbe troppo lenta. In generale la cache richiede alla memoria centrale e, insieme all'istruzione necessaria richiede anche le celle successive, così che non dovrà attendere per la prossima richiesta e le invia alla CPU. Questo ci fa capire che la cache è un chip speciale, molto e veloce e quindi costota. Attualmente abbiamo tre livelli di cache:

- **L1 o cache di primo livello** è un tipo di cache presente fisicamente nel processore;

- **L2 o cache di secondo livello** si può trovare alloggiato sia nella scheda madre che nel core;
- **L3 o cache di terzo livello** che è condivisa tra i core dell'elaboratore.
  Qundi alla fine avremo L1 e L2 situate in ogni core della CPU e L3 condivisa da tutta la CPU.

Le ROM

Con **ROM** definiamo delle memorie di sola lettura (Read Only Memory) che contengono istruzioni necessarie al funzionamento della macchina e che non possono essere riscritte da un utente. Vengo scritte e installate al momento della saldatura sulla scheda madre. Di queste ROM ne abbiamo di diverso tipo:

- **PROM** ovvero Programmable Read Only Memory, sono delle ROM programmabili una sola volta;
- **EPROM** ovvero Erasable Programmable Read Only Memory che possono essere riprogrammate con apparecchiature speciali;
- **EEPROM** Electrically Erasable Programmable Read Only Memory sono delle memorie flash che possono essere sempre riscritte

Memoria di massa

- **I Nastri Magnetici** permetto solo accesso sequenziale ai dati, quindi non è possibile decidere quale celle prendere. Inoltre queste memorie sono usate per scopi particolari, come archivi storici;
- **I Floppy Disk** sono stati i prmi supporti magnetici con accesso diretto ai dati, la loro grandezza massima è di 1,44 MB e al giorno d'oggi non sono più utilizzati.
- **I Dischi Magnetici** o meglio Hard disk sono dei dispositivi elettro-meccanico per conservare informazioni sotto forma magnetica tramite la lettura e scrittura fatta da una testina rotante. Questa testina contine un induttore ed è sospesa sopra i dischi. L'hard disk è struttorato in cilindri, tracce e settori. Le tracce sono dei cerchi concentrici sul disco, i settori sono uan suddivisione delle tracce mentri i cilindri sono dei gruppi di tracce.

ARRIVATI A PAG 26 PARTE 6