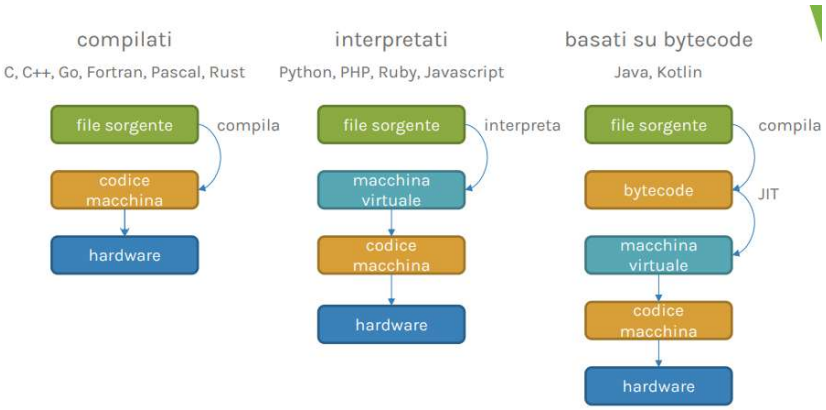
Lezione 2 Introduzione 2 + Rappresentazione dati 26/9/23

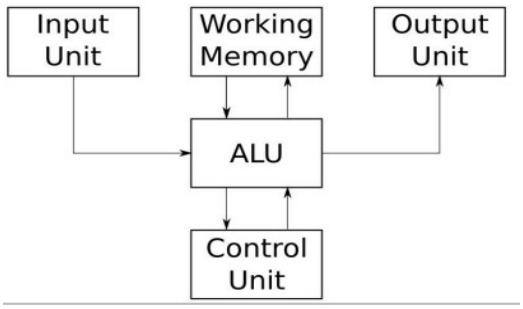
martedì 26 settembre 2023 15:36

Principalmente oggi giorno i linguaggi di programmazione si dividono in tre categorie, ognuna delle quali con un modello specifico di sviluppo e progettazione:



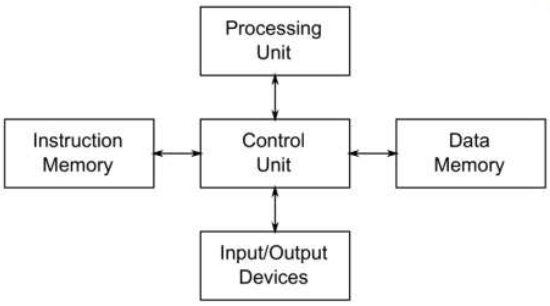
Dove la macchina virtuale è uno strato che maschera realizzazione di hw dal sw. Quindi questo componente essenziale viene usato come emulatore di macchina a stati finiti. E' caratterizzato da un'insieme di istruzioni, un'insieme di registri, due aree di memoria : stack e heap e da un formato file delle class. Per quanto riguarda le architetture attuali vi è un trade-off tra architettura Von-Neumann , Architettura Harvard e quella moderna : nessuna delle tre prevale sulle altre quindi si cerca di arrivare ad un compromesso.

Architettura Von-Neumann:



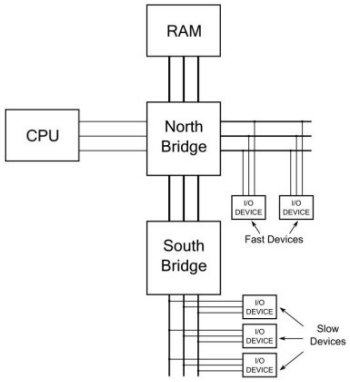
Questa architettura è composta principalmente da 3 componenti : ALU + Memory Unit + Control Unit , dove il primo fa da interprete per le istruzioni , accede in modo conteso alla memoria di lavoro con la Control Unit e comunica direttamente con i dispositivi di i/o , la seconda invece salva i dati della computazione , mentre l'ultima si contende accesso alla memoria con alu , quindi si ha una fatica della memoria a fornire i sati, soprattutto a dispositivi con velocità diverse. Diciamo inoltre che la Alu e le unità di controllo sono realizzate in modo differente , in base agli obiettivi , quindi in generale si distingue una parte che esegue il lavoro (ALU) ed una parte più intelligente (Control Unit). La pecca di questo tipo di architettura è che ha una sola memoria : latenza di accesso maggiore , quindi si ha un collo di bottiglia in termini prestazionali.

Architettura Harvard:



In questa architettura si ha la risoluzione del problema del collo di bottiglia di quella di Von - Neumann. Usa datapath alternativi per istruzioni e per i dati , usa memorie cache , quindi in generale si ha un aumento delle prestazioni in termini di reattività e complessità. Quindi in definitiva questa architettura migliora le prestazioni ma soffre delle dimensioni del programma , ovvero programma piccolo ma con molti dati oppure programma grande e pochi dati : di solito si usa in particolari scenari anche se è presente nei moderni computer.

Architettura moderna:



Questa architettura usa due coprocessori : separano la logica e la fisica dell'applicazione tra device a diverse velocità, ma disturbano il processore se il dispositivo più lento è pronto. Di solito questi due coprocessori sono all'interno della CPU.

Come già detto i moderni computer elaborano vari tipi di file audio, immagini , ma alla base di questi file vi sono le sole informazioni riconosciute : due livelli di informazione alto/basso oppure 0/1 oppure acceso/spento , quindi per manipolare qualunque tipo di dato bisogna scegliere una rappresentazione adeguata : la codifica (alfabeto finito associato a qualche configurazione) . Nella codifica bisogna distinguere **numeri e numerali** : i primi sono delle entità astratte che esistono a prescindere dalla rappresentazione, mentre i secondi rappresentano una sequenza di caratteri in un qualsiasi sistema posizionale . In virtù di ciò si possono definire sistemi numerici posizionali : sequenze di numerali :

$$(x)_b = \langle a_n a_{n-1} \cdots a_1 a_0, c_1 c_2 c_3 \cdots \rangle = \sum_{i=0}^{n} (a_i \cdot b^i) + \sum_{k=1}^{\infty} c_k b^{-k}$$

Di questi sistemi numerici è possibile fare la conversione da una base ad altra sia per la parte intera (sx della virgola) sia per la parte frazionaria (dx della virgola). Nel primo caso si divide il numero per la base corrispondente , mentre per la parte frazionaria si moltiplica perla base . Si parte dalla fine a ritroso. Si usano principalmente tre basi : base 2 , base 8 , base 16 : base 2 -> {0,1} , base 8 -> {0,1,2,3,4,5,6,7} , base 16 ->{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} . Possibile passare da base ad altra , raggruppando opportunamente le cifre : 3 se base 8, 4 se base 16.Vediamo un esempio:

8->2	2->8
65->101 110	101 100->54
17->001 111	110 010->62

16->2	2->16
AF->1010 1111	1011 1001->B9
1C->0001 1100	0000 1001->09

Quindi in generale si vede il valore di k intero tale che :

Intervallo di rappresentazione [0,2^$k-1$]
Numero di bit necessari k=log(base)n

Questo discorso vale solo per i numeri positivi, ma visto che i processori attuali possono manipolare anche i numeri negativi, bisogna implementare le stesse logiche : **Rappresentazione in modulo e segno, complemento ad uno e complemento a due.**
Nella prima si va a vedere la cifra più significativa (a sx) e se è "0" è positivo , mentre se "1" è negativo :

10001100	-12
00001100	+12
0000000	+0
1000000	-0

Occhio alla rappresentazione doppia dello 0!!!
Quindi questa rappresentazione è inefficiente se circuiti più complessi , aumentando i costi e diminuendo le prestazioni .
Nel secondo si ha sempre la doppia rappresentazione dello "0", bit più significativo ancora quello più a sx , ma questo numero si ottiene invertendo tutti i bit:

00001100	+12
11110011	-12

Quindi intervallo di rappresentazione : [-(2^(n-1)-1),2^(n-1)-1]
Da notare che in questa rappresentazione potrebbe verificarsi la non veridicità del risultato in quanto si ha effetto end-around-carry:

```
 1111 1110 +    -1 +
 0000 0010 =     2 =
10000 0000      0
```

Quindi devo fare somma ulteriore : riporto al risultato
Nell'ultima invece si ha la rappresentazione più flessibile semplice ed efficiente che un calcolatore possa computare. Si ottiene facendo cosi': partendo dal bit meno significativo , si lasciano inalterati tutti i bit fino al primo "1" **compreso** e poi si complementa bit a bit, oppure si complementa bit a bit e si aggiunge "1":

Primo modo	1011 1001	0100 0111		0100 0111
Secondo modo	1011 1001	0100 0110	+1	0100 0111

Per quanto riguarda le somme / sottrazioni in questo ambito può essere svolta ignorando il segno , quindi possibile usare stesso circuito per due operazioni.

Quindi intervallo di rappresentazione : [-(2^(n-1)),2^(n-1)-1]