Livello 2: Macchina hardware

Livello 5 : Macchina utente
Livello 4 : Macchina programmatore
Livello 3 : Macchina virtuale
Livello 2 : Macchina hardware
Livello 1 : Macchina microprogram
Livello 0 : Macchina digitale

E' governata dal linguaggio macchina. Offre visione ed accesso *diretto* a tutte le risorse *fisiche*, tramite una *specifica* interfaccia di livello (funzionalmente simile alle altre implementazioni). Fornisce un'interfaccia prettamente *software* (i.e. programmabile) ai livelli superiori.

Linguaggi Macchina (Stallings: cap. 10)

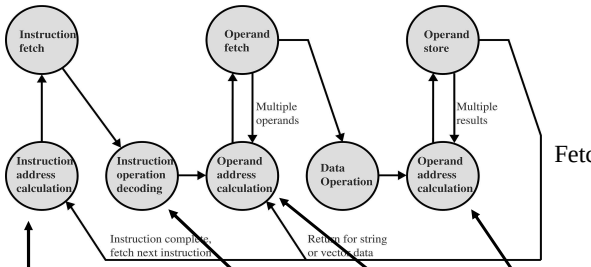
- L'instruction set (o linguaggio macchina) costituisce il "linguaggio" messo a disposizione dal livello hardware al livello superiore: specifica le **operazioni (esterne**, cioè visibili all'esterno**)** della CPU
- Le istruzioni possono prevedere **operandi** con varie modalità di localizzazione
- Le istruzioni possono avere **vari formati** e **diversa durata**
- Il tipo di istruzioni previsto da un instruction set determina il costo e le prestazioni della CPU

Progettazione di Istruction set

Gli elementi fondamentali sono:

- Repertorio: quante e quali operazioni fornire, e di che complessità
- Tipi di dato: tipi di dato su cui eseguire le operazioni
- Indirizzamento: modo/i di specificare gli indirizzi degli operandi
- Formato: lunghezza in bit, dimensione campi, etc.
- Registri: numero registri referenziabili, e loro uso

Istruzione Macchina



Ciclo Fetch/Execute

Elementi di una istruzione macchina generica:

- codice operativo
- riferimento/i operando/i sorgente
- riferimento/i operando/i risultato
- riferimento istruzione successiva

Istruzione Macchina

Operandi sorgente e risultato si possono trovare in:
- memoria centrale (o virtuale, che vedremo in seguito)
- registri della CPU
- dispositivi di I/O

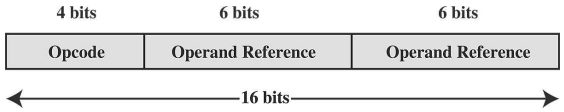
Istruzione successiva posizionata in:
- memoria centrale
- se si usa memoria virtuale:
  - memoria centrale o secondaria (disco)

Riferimento istruzione successiva:
- implicito (nel PC) se si riferisce alla istruzione successiva
- esplicito se diverso dalla istruzione successiva

Rappresentazione Istruzioni

Ogni istruzione è:
- rappresentata da una sequenza di bit
- suddivisa in campi che corrispondono agli elementi costitutivi appena discussi

es.



Tipicamente un instruction set prevede più di un formato
(lo vedremo meglio di seguito)

Esecuzione: istruzione posta nel registro IR della CPU

Tassonomia di istruzioni

In generale si possono riconoscere le seguenti categorie di istruzioni:

- Elaborazione: istruzioni aritmetico/logiche
- Memorizzazione: lettura e scrittura memoria
- Trasferimento: istruzioni di I/O
- Controllo: istruzioni di test e salto

Tassonomia di istruzioni

Tipo	*Esempi*
aritmetiche e logiche	aritmetica intera e operazioni logiche: add, and, subtract, or
trasferimento dati	trasferimento dati da/per memoria/registri: load, store
controllo	branch, jump, chiamata e ritorno da procedure, traps
sistema	chiamate di sistema operativo, istruzioni per la gestione della memoria virtuale
virgola mobile	operazioni in virgola mobile: add, multiply
decimale	operazioni decimali: decimal add, decimal multiply, decimal-to-character conversions
stringa	spostamento, comparazione, ricerca di stringhe
grafica	operazioni su pixel, operazioni di compressione/decompressione

Altri esempi

- Trasferimento di controllo
  - Operanti su Instruction Pointer (IP), detto anche Program Counter (PC)
- Operazioni su pila (*stack*) (vediamo di seguito)
  - Push, Pop
- Operazione specializzate
  - Aritmetica BCD (vediamo di seguito), trattamento I/O

Istruzioni più usate (80x86)

istruzione	*percentuale di uso*
load	22%
branch condizionale	20%
compare	16%
store	12%
add	8%
and	6%
sub	5%
spostamento reg./reg.	4%
call	1%
return	1%
Totale	**96%**

Architettura di CPU

- Un instruction set definisce (corrisponde ad) una particolare **architettura** di CPU
  - Architettura ad accumulatore
  - Architettura a pila (*stack*)
  - Architettura a registri generali
- Il tipo di architettura è anche correlata con il numero di indirizzi (riferimenti) specificati esplicitamente dalle istruzioni (tipicamente aritmetico/logiche)
  - Architettura ad accumulatore: **1 indirizzo**
  - Architettura a pila: **0 indirizzi**
  - Architettura a registri generali: **1 o più indirizzi**

Architettura ad accumulatore

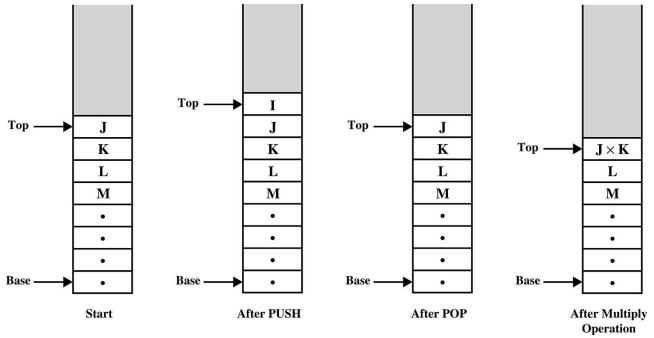
- Uno degli operandi dell'istruzione è assunto trovarsi sempre in un registro speciale detto '**accumulatore**' (registro AC)
- Gli altri operandi sono generalmente localizzati in memoria
- Architettura arcaica e poco efficace
- Programmazione concisa

Architettura a pila

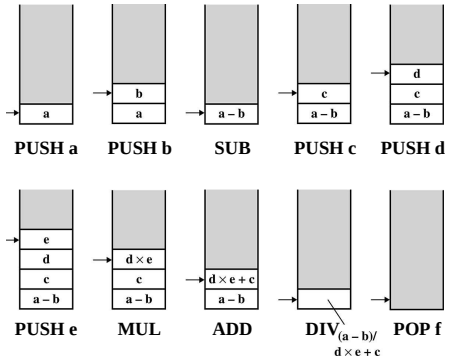
- Gli operandi sono assunti essere presenti **in testa** ad una particolare zona di memoria detta '**pila di sistema**'
- Le operazioni di calcolo sono precedute da inserimento di operandi (**PUSH**) in pila e seguite da prelievo del risultato (**POP**) dalla pila
- Architettura arcaica e poco efficace
- Programmazione verbosa

Architettura a pila

Es. di PUSH, POP, e applicazione di una operazione aritmetica (moltiplicazione)



Architettura a pila



Esempio di elaborazione di una espressione aritmetica

f = (a-b)/(d×e + c)

usando una architettura a pila

Architettura a registri generali

- Gli operandi sono tutti **espliciti**
- Gli operandi sono localizzati in **memoria** o in **registri generali**
- Esiste una vasta gamma di modalità di localizzazione (**indirizzamento**) degli operandi (vediamo di seguito)

Esempio

A + B → C

Architettura a pila

Push A | Push B | Add | Pop C

A→Stack | B→Stack | <risultato>→Stack | Stack→C

Architettura ad accumulatore

Load A | Add B | Store C

A→Acc | Acc+B→Acc | Acc→C

Architettura a registri generali

Load R1,A | Load R2,B | Add R1,R2,R3 | Store C,R3

A→R1 | B→R2 | R1+R2→R3 | R3→C

Esempio più complesso

Esempio di calcolo della espressione aritmetica

f = (a-b)/(d x e + c)

usando le 3 architetture viste in precedenza

	Pila	**Registri**	**Accumulatore**
	Push a Push b Sottrai Push c Push d Push e Moltiplica Somma Dividi Pop f	Carica R1, a Sottrai R1, b Carica R2, d Moltiplica R2, e Somma R2, c Dividi R1, R2 Salva R1, f	Carica d Moltiplica e Somma c Salva f Carica a Sottrai b Dividi f Salva f
Numero di istruzioni	10	7	8
Accessi alla memoria	10 op + 6 d	7 op + 6 d	8 op + 8 d

Riassunto

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T – 1) OP T

AC = accumulator
T = top of stack
A, B, C = memory or register locations

Ulteriore classificazione 1

Esistono **tre** sottoclassi di architetture a registri generali

- **Registro-registro (Load/Store)**
  Tutti gli operandi sono su registro. Gli accessi a memoria si effettuano con **Load** (prelievo) e **Store** (deposito)
- **Memoria-registro**
  Prevedono operazioni con alcuni operandi in memoria ed altri su registro
- **Memoria-memoria**
  Prevedono operazioni che prelevano operandi in memoria e restituiscono il risultato in memoria

Ulteriore classificazione 2

- Architetture commerciali di tipo 1 (Load/Store)
  - SPARC, MIPS, PowerPC
- Architetture commerciali di tipo 2 (memoria/registro)
  - Intel x86, Motorola 68k
- **Non** esistono architetture commerciali di tipo 3 (memoria/memoria)

Architettura Load/Store (RISC)

- Necessita di poche istruzioni, con formato molto semplice ed uniforme
- Le istruzioni hanno tempi di esecuzione molto simili tra loro
- Facilita il compito dei generatori automatici di codice macchina (compilatori)
- Nota sotto il nome di architettura **RISC** (Reduced Instruction Set Computer)

Architettura CISC

- Alternativa alla regolarità dell'architettura RISC è la cosiddetta architettura **CISC** (Complex Instruction Set Computer), che utlizza istruzioni a **formato variabile**, con grande potenza espressiva e tempi di esecuzioni variabili e spesso elevati
- L'architettura Intel x86 è di tipo CISC

Tipi di dato

Categorie generali di dati:

- Indirizzi: di fatto considerati interi senza segno
- Numeri:
  - interi (o in virgola fissa) [rapp. vista a Inf. di Base]
  - decimali, cioè in base 10 (vediamo di seguito)
  - in virgola mobile [rapp. vista a Inf. di Base]
- Caratteri: (vediamo di seguito)
- Dati logici: si interpretano i singoli bit come variabili logiche

Dati numerici
Codifica dei dati interi senza segno

	TIPO	bit	rango
UB	unsigned byte	8	0 .. 255
UW	unsigned word	16	0 .. 65525
UD	unsigned double word	32	0 .. ~ 4GB
BU	byte unpacked BCD	8	0 .. 9
BP	byte packed BCD	8	0 .. 99

Dati numerici
Codifiche UB, UW, UD

Dato il numero ad **n** bit

$$A = a_{n-1}\ a_{n-2}\ ...\ a_2\ a_1\ a_0$$

la i-esima cifra ha un peso di 2^i, ad esempio

$1010_{bin} = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 8 + 2 = 10_{dec}$

L'intervallo di valori va da 0 a 2^n-1.

Aritmetica (senza segno)
Bit di zero e di carry

- **Bit di zero**, vale 1 se tutti i bit del risultato sono a zero
- **Bit di carry**, vale 1 se il risultato di una operazione tra operandi senza segno supera il limite di valore consentito
  - nelle somme rappresenta il riporto
  - nelle differenze vale -2^n

Dati numerici
Codifica dei dati interi con segno

	TIPO	bit	rango
SB	signed byte	8	-128 .. +127
SW	signed word	16	-32768 .. +32767
SD	signed double word	32	~ $-/+ 2.15x10^9$
SQ	signed quad word	64	~ $-/+ 9.22x10^{18}$
PD	packed decimal	80	$-/+ 9{,}999... x10^{18}$

*Rappresentazione del numero in **complemento a 2***

Aritmetica (con segno)
Bit di segno e di overflow

- **Bit di segno** (*MSb*), vale 1 se il numero è negativo
- **Bit di overflow**, vale 1 se il risultato di un'operazione tra operandi con segno supera il rango consentito
  - nelle somme vale 1 ***se e solo se*** i due operandi sono dello stesso segno ed il risultato è di segno opposto

Dati decimali
Codifiche BU, BP

Si utilizzano 8 bit (*octet*) o due gruppi di 4 bit (*nibble*) per rappresentare l'informazione in codifica Binary Coded Decimal
Esempio:

$3_{dec} = 0000\ 0011_{BU}$

$45_{dec} = 0100\ 0101_{BP}$

dec	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Dati decimali con Segno
Codifica PD

- Campo **S**, segno, un byte per il segno (negativo = *MSb* a 1)
- Campo **G**, grandezza, 9 byte in codifica *byte packed BCD*

-353_{dec} = `1000 0000  0000 0000  0000 0000`
`0000 0000  0000 0000  0000 0000`
`0000 0000  0000 0000  0000 0011`
`0101 0011`

Numeri Reali
Codifica dei numeri in virgola mobile (1)

- I **numeri reali** rappresentabili da un elaboratore sono un **sottinsieme finito** dei numeri razionali
- Si usa la notazione a *virgola mobile*:

$$\mathbf{N = +/- M \times B^E}$$

  - **M**, *mantissa*, numero razionale con segno
  - **E**, *esponente*, numero intero con segno
  - **B**, *base*, valore standard = 2

Numeri Reali
Codifica dei numeri in virgola mobile (2)

Lo standard ANSI/IEEE 754 usa la rappresentazione $\mathbf{(-1)^S \times 1.M \times 2^{E-bias}}$ a 3 campi:

- **S**, *segno*, su 1 bit (0 → numero positivo)
- **M**, *mantissa*, su **m** bit, normalizzata escludendo il bit più significativo che vale sempre 1
- **E**, *esponente* o *caratteristica*, su **e** bit, valutata in codifica ad eccesso $2^{e-1}-1$ (**bias**)
- Con precisione **p = 1 + m + e**

Numeri Reali
Codifica dei numeri in virgola mobile (3)

	TIPO	p	S	e	m	bias
SR	short real	32	1	8	23	127
	$\sim 1.8 \times 10^{-38} < \lvert x \rvert < \sim 3.40 \times 10^{+38}$					
LR	long real	64	1	11	52	1023
	$\sim 2.23 \times 10^{-308} < \lvert x \rvert < \sim 1.80 \times 10^{+308}$					
XR	extended real	80	1	15	64	16383
	$\sim 3.4 \times 10^{-4932} < \lvert x \rvert < \sim 1.2 \times 10^{+4932}$					

Numeri Reali
Codifica dei numeri in virgola mobile (4)

Esempio

Il valore in codifica SR (*single-precision*)

0 10000110 00000001000000000000000

2^{-1} 2^{-2} 2^{-3} 2^{-4} ...

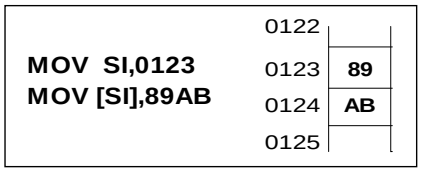
dove **S** = 0, **E** = 10000110 = 134_{dec}, e

M = 000000010...$0_{IEEE754}$ = 00000001_2/ **1**00000000_2 = 0.00390625 (equivalente a 2^{-8}, con le cifre in mantissa aventi peso negativo) corrisponde al valore decimale:

$x = (-1)^0 \times 1.00390625 \times 2^{134-127} = 1.00390625 \times 128 = 128.5$

Memorizzazione dei dati
Big Endian

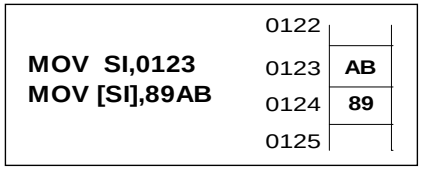
- **Big Endian**: si memorizzano i byte più significativi del dato a partire dagli indirizzi inferiori
- L'indirizzo del dato è quello dell'MSB (*big end*)



Motorola 68xxx
IBM360/370
MIPS, SPARC

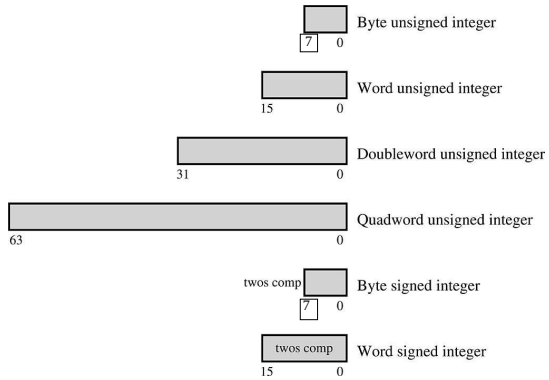
Memorizzazione dei dati
Little Endian

- **Little Endian**: si memorizzano i byte meno significativi dell'informazione a partire dagli indirizzi inferiori
- L'indirizzo del dato è quello dell'LSB (*small end*)



Intel x86
DEC Vax

Esempio: tipi di dato nel Pentium



Esempio: tipi di dato nel Pentium