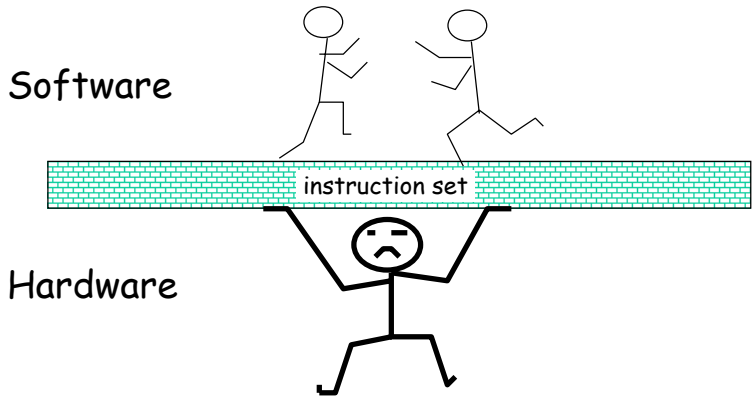
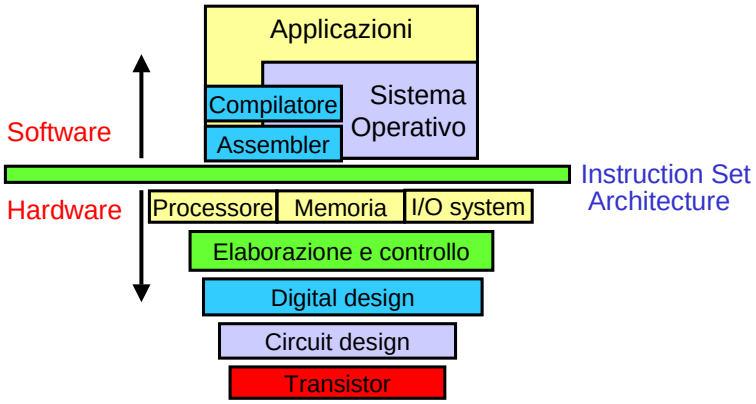
Architettura del Set di Istruzioni (ISA)

Maurizio Palesi

Instruction Set Architecture (ISA)

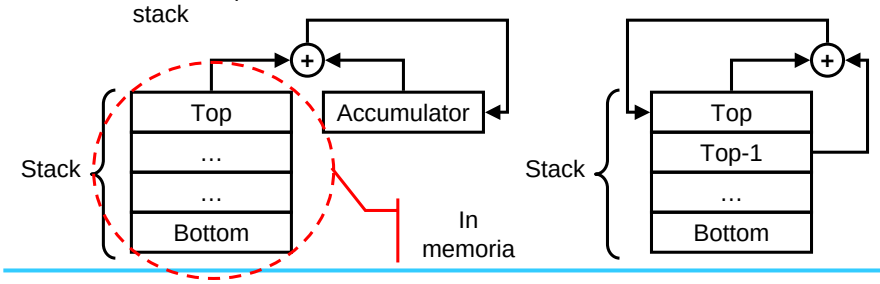


Instruction Set Architecture (ISA)



Classificazione dell'ISA

- Architetture **Stack-based** (prima degli anni '80)
  - ➔ Le istruzioni non hanno operandi
  - ➔ Gli operandi delle istruzioni sono impliciti
    - ✓ Accumulatore e Stack
  - ➔ Es. **ADD**
    - ✓ Somma il contenuto dell'accumulatore con il dato presente in cima allo stack e memorizza il risultato nell'accumulatore
    - ✓ Somma i primi due dati nello stack e memorizza il risultato in cima allo stack



Classificazione dell'ISA

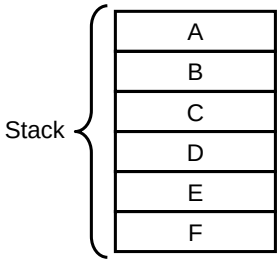
- Architetture **General Purpose Register** (dopo gli anni '80)
  - ➔ General Purpose Registers (GPR)
    - ✓ Interni al processore (+ veloci della memoria)
    - ✓ Più semplici da gestire da parte del compilatore
    - ✓ **Esempio** **(A*B) – (C*D) – (E*F)**

R1	A
R2	D
R3	B
R4	E
R5	C
R6	F

```
mul R1, R1, R3    ; R1 = A*B
mul R2, R5, R2    ; R2 = C*D
mul R3, R4, R6    ; R3 = E*F
sub R1, R1, R2
sub R1, R1, R3
```

Per calcolare l'espressione i dati possono essere memorizzati in qualsiasi registro

Classificazione dell'ISA (cont.)

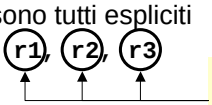



```
mul    ; top = A*B
rotate ; Ruota lo stack
pop    ; Elimina A
pop    ; Elimina B
mul    ; top = C*D
rotate ; Ruota lo stack
pop    ; Elimina C
pop    ; Elimina D
mul    ; top = E*F
rotate ; Ruota lo stack
pop    ; Elimina E
pop    ; Elimina F
sub    ; top = A*B – C*D
rotate ; Ruota lo stack
pop    ; Elimina A*B
pop    ; Elimina C*D
rotate
sub    ; Top = A*B - C*D - E*F
```

Il compilatore deve "faticare" per generare il codice in un'architettura stack-based!

Architetture GPR

- Gli operandi sono tutti espliciti
  - Es. `add r1, r2, r3`



Sto esplicitamente specificando gli operandi sorgente (r2 ed r3) e destinazione (r1) dell'istruzione.

- 3 Classi di architetture GPR
  - Memoria – Memoria
    - ✓ 2 operandi: add A, B → mem[A] = mem[A] + mem[B]
    - ✓ 3 operandi: add A, B, C → mem[A] = mem[B] + mem[C]
  - Registro – Memoria
    - ✓ 2 operandi: add Ra, B → Ra = Ra + mem[B]
    - ✓ 3 operandi: add Ra, Rb, C → Ra = Rb + mem[C]
  - Registro – Registro (Load/Store)
    - ✓ 3 operandi: add Ra, Rb, Rc → Ra = Rb + Rc

Confronto

- In riferimento all'istruzione di alto livello

A = B + C

Dove **A**, **B** e **C** sono tre variabili intere si ha

Stack	Registro-Registro	Registro-Memoria	Memoria-Memoria

`push B`
`push C`
`add`
`pop A`
	`load R1, B`
	`load R2, C`
	`add R3, R1, R2`
	`store A, R3`
		`load R1, B`
		`add R1, C`
		`store A, R1`
			`add A, B, C`

Alcuni Esempi

Macchina	Numero di registri	Stile architetturale	Anno
EDSAC	1	accumulator	1949
IBM 701	1	accumulator	1953
CDC 6600	8	load-store	1963
IBM 360	16	register-memory	1964
DEC PDP-8	1	accumulator	1965
DEC PDP-11	8	register-memory	1970
Intel 8008	1	accumulator	1972
Motorola 6800	2	accumulator	1974
DEC VAX	16	register-memory, memory-memory	1977
Intel 8086	8	extended-accumulator	1978
Motorola 68000	16	register-memory	1980
Intel 80386	8	register-memory	1985
MIPS	32	load-store	1985
HP PA-RISC	32	load-store	1986
SPARC	32	load-store	1987
PowerPc	32	load-store	1992
DEC Alpha	32	load-store	1992

Architetture GPR - Sommario

- Dal 1975 tutte le macchine usano GPR
- Vantaggi
  - ➔ Sono più veloci della memoria
  - ➔ Sono più facili da usare per un compilatore
    - ✓ Es., (A*B) – (C*D) – (E*F) a differenza dello stack, i prodotti possono essere eseguiti in qualsiasi ordine
  - ➔ Possono conservare variabili
    - ✓ Il traffico con la memoria è ridotto
      - – Programmi sono più veloci
  - ➔ La densità del codice aumenta
    - ✓ Infatti la codifica del nome di un registro richiede meno bit dell'indirizzo di una locazione di memoria

Indirizzamento della Memoria

- Di norma l'indirizzamento avviene al byte
- Due problemi per il progetto dell'ISA
  - ➔ Dal momento che una word di 32 bit può essere letta come una sequenza di quattro byte
    - ✓ Come mappare gli indirizzi a livello di byte all'interno della word?
    - ✓ L'indirizzo iniziale di una word può essere qualsiasi?

Ordinamento degli Indirizzi

- **Big Endian**
  - ➔ L'indirizzo del byte più significativo = indirizzo word (xx00 = Big End of word)
  - ➔ IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- **Little Endian**
  - ➔ L'indirizzo del byte meno significativo = indirizzo word (xx00 = Little End of word)
  - ➔ Intel 80x86, DEC Vax, DEC Alpha
- **Esempio:** scrivendo la word 0xAABBCCDD all'indirizzo di memoria 0, i 4 byte della parola vengono così ordinato

In Big Endian

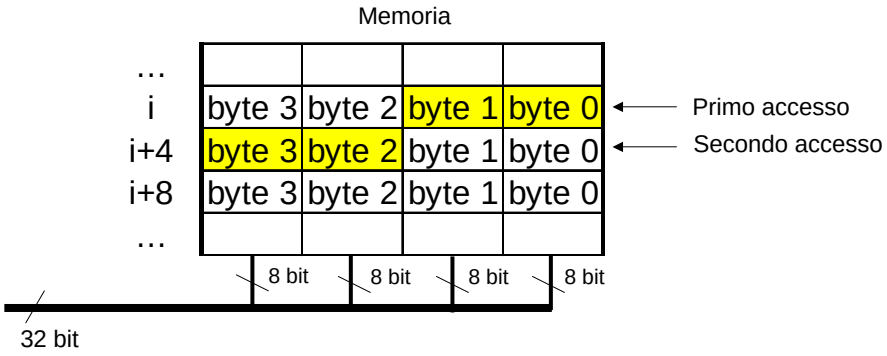
Indirizzo	Cella di memoria (1 byte)
0	AA
1	BB
2	CC
3	DD
4	
5	
…	

In Little Endian

Indirizzo	Cella di memoria (1 byte)
0	DD
1	CC
2	BB
3	AA
4	
5	
…	

Allineamento dei Dati

- Si consideri una memoria con bus dati di 32 bit (4 byte)
  - ➔ Con un solo accesso è possibile cioè prelevare 4 byte
  - ➔ Se la word richiesta non è allineata (cioè non inizia da un indirizzo multiplo di 4) occorrono 2 accessi alla memoria per prelevarla



Modalità di Indirizzamento (Dati)

Modalità	Esempio	Significato
Register	`add r4,r3`	`r4 = r4 + r3`
Immediate	`add r4,#3`	`r4 = r4 + 3`
Displacement	`add r4,100(r1)`	`r4 = r4 + mem[100+r1]`
Register indirect	`add r4,(r1)`	`r4 = r4 + mem[r1]`
Indexed/Base	`add r3,(r1+r2)`	`r3 = r3 + mem[r1+r2]`
Direct or absolute	`add r1,(1000)`	`r1 = r1 + mem[1000]`
Memory indirect	`add r1,@(r3)`	`r1 = r1 + mem[mem[r3]]`
Post-increment	`add r1,(r2)+`	`r1 = r1 + mem[r2]` `r2 = r2 + d`
Pre-decrement	`add r1,-(r2)`	`r2 = r2 - d` `r1 = r1 + mem[r2]`

Utilizzazione delle Modalità di Indirizzamento

- 3 programmi misurati su una macchina con tutte le modalità di indirizzamento (VAX)

Indirizzamento	Utilizzazione	Range
Displacement	42%	32%-55%
Immediate	33%	17%-43%
Register deferred (indirect)	13%	3%-24%
Scaled	7%	0%-16%
Memory indirect	3%	1%-6%
Misc	2%	0%-3%

75%

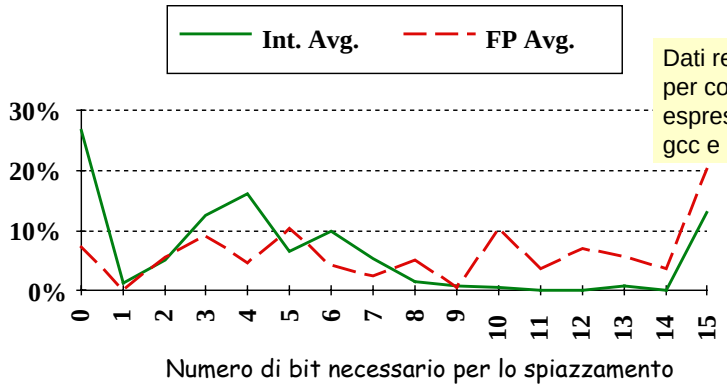
88%

Modalità di Indirizzamento (controllo)

- Salto, salto condizionato, chiamata e ritorno da sottoprogrammi
  - ➔ Diretto
  - ➔ Relativo al PC o ad altro registro
- **Esempi**

```
JMP   DEST        ; Diretto o relativo a PC
JZ    wait        ; Di solito relativo a PC
call  sub         ; Di solito diretto
BR    R30         ; Destinazione = R30
```

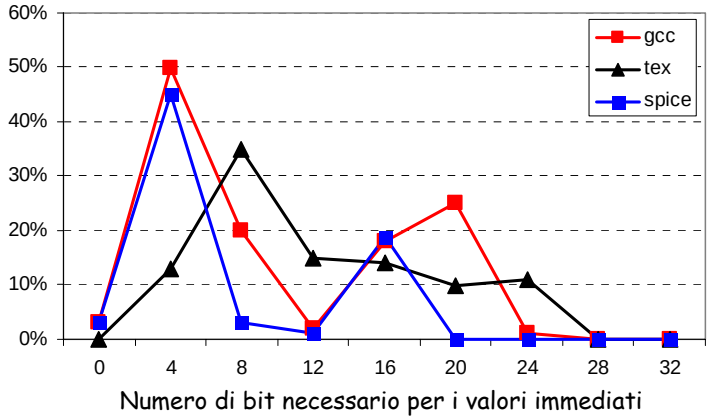
Dimensione dello Spiazzamento



Dati relativi a MIPS per compress, espresso, eqntott, gcc e li

Con **12** bit di spiazzamento si cattura circa il **75%** dei casi
Con **16** bit di spiazzamento si cattura circa il **99%** dei casi

Dimensione degli Immediati



Misure effettuate su un VAX

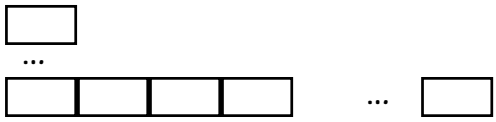
Con **8** bit si cattura circa il **50%-70%** degli immediati
Con **16** bit si cattura circa il **75%-80%** degli immediati

Indirizzamento - Sommario

- Modalità di indirizzamento dei dati più importanti
  - ➔Displacement, Immediate, Register Indirect (catturo circa il 90% dei casi)
- Dimensione dello spiazzamento
  - ➔Dai 12 ai 16 bit (catturo dal 75% al 99% dei casi)
- Dimensione degli openandi immediati
  - ➔Dagli 8 ai 16 bit (catturo dal 50% all'80% dei casi)

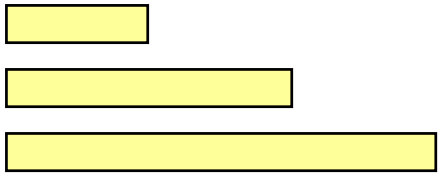
Formato delle Istruzioni

Variabile
(VAX)



Fissa
(MIPS, PowerPC)

Ibrida
(Intel 80x86)

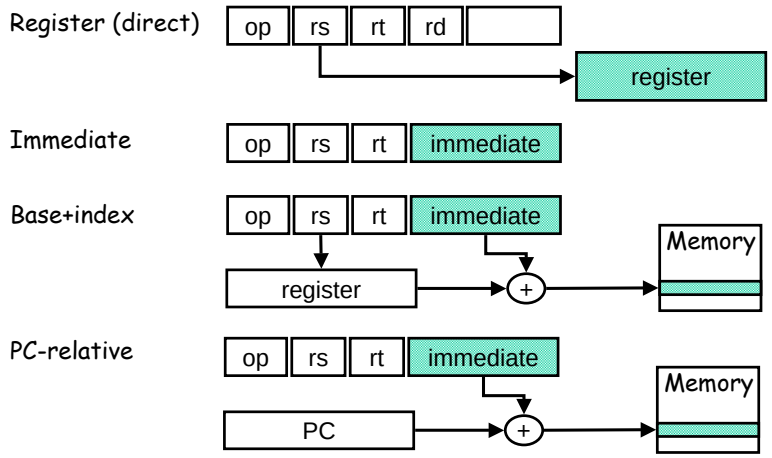


Formato delle Istruzioni

- Se la dimensione del codice è il requisito più importante, si usano istruzioni a lunghezza variabile
  - ➔Aumento della densità del codice
  - ➔Questo stile è il migliore quando sono presenti molte modalità di indirizzamento e operandi
- Se le prestazioni sono il requisito più importante, si usano istruzioni a lunghezza fissa
  - ➔Questa scelta è particolarmente indicata quando l'operazione e la modalità di indirizzamento sono combinate nel codice operativo
  - ➔Lavora bene quando sono presenti poche modalità di indirizzamento

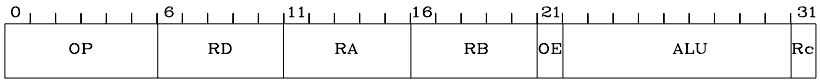
Esempio MIPS

- Formato fisso dell'istruzione **32 bit**



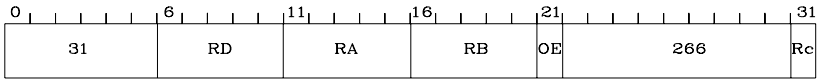
Esempio PowerPC

Formato istruzioni aritmetiche



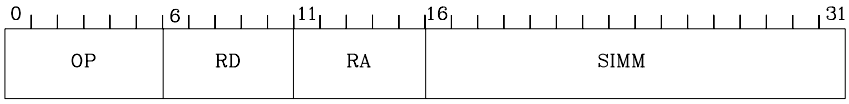
Esempio di istruzione aritmetica:
Istruzione di somma

```
ADD    RD,RA,RB     (OE=0, Rc=0)
ADD.   RD,RA,RB     (OE=0, Rc=1)
ADDO   RD,RA,RB     (OE=1, Rc=0)
ADDO.  RD,RA,RB     (OE=1, Rc=1)
```

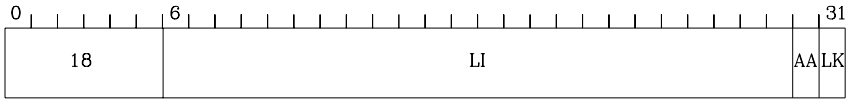


Esempio PowerPC

Formato istruzioni aritmetiche con operando immediato e istruzioni di caricamento e e memorizzazione dei registri



Formato istruzione di salto incondizionato



Esempio Intel x86

PREFISSI				ISTRUZIONE				
Istruzione	Dimensione Indirizzo	Dimensione Operando	Segmento	Codice	MOD R/M	SIB	Scostamento	Immediato
0/1	0/1	0/1	0/1	1/2	0/1	0/1	0/1/2/4	0/1/2/4

Numero di byte possibili per ciascuno dei campi dell'istruzione

MOV	**EAX, ES:V[EBX+EIX]**	6 byte
PUSH	**EAX**	1 solo byte
PUSH	**Var[EBX]**	Fino a 6 byte

x86 Top Ten!

	Classe	Percentuale eseguite
1	load	22%
2	branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move	4%
9	call	1%
10	return	1%
	Total	95%

Le istruzioni maggiormente eseguite sono semplici!