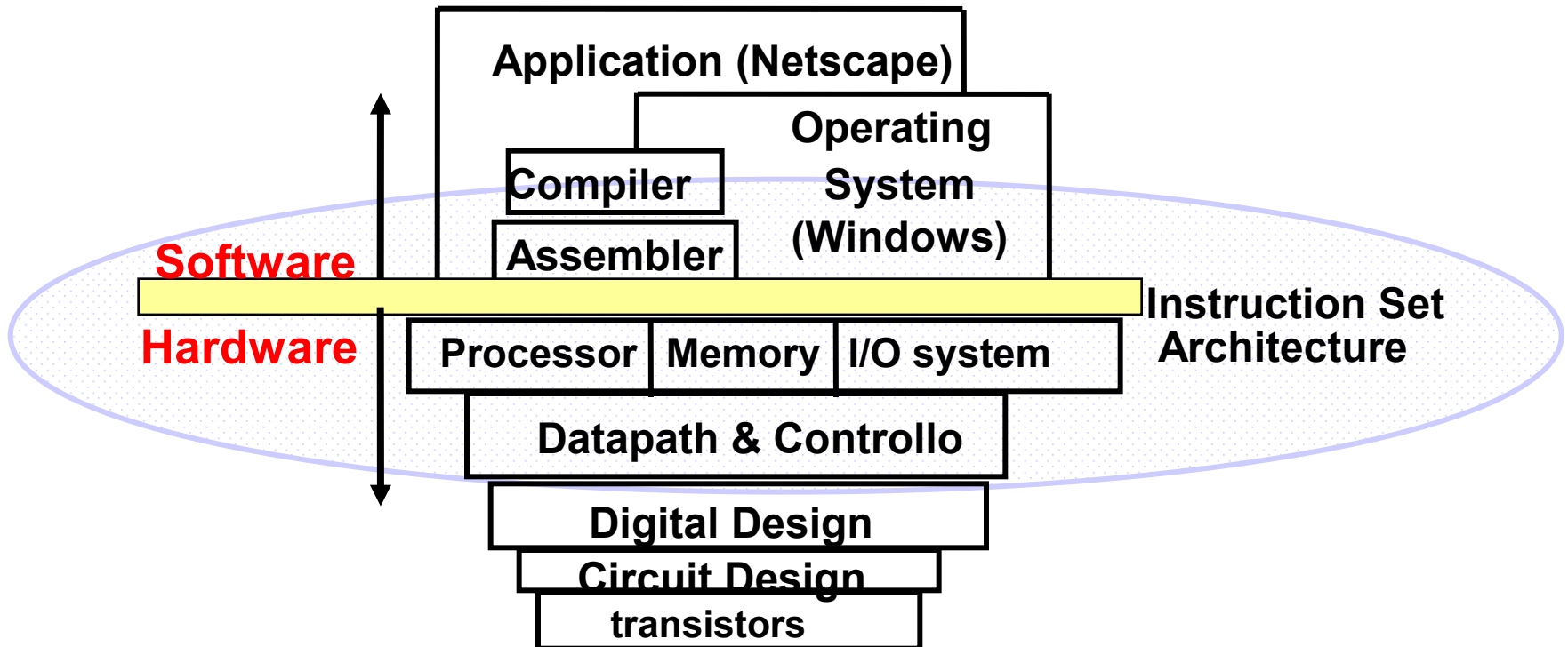


# Architettura del set di istruzioni (ISA)

# Instruction Set Architecture

Prof. G. Ascia



# Design Space of ISA

---

Prof. G. Ascia

## Five Primary Dimensions

- Number of explicit operands
- Operand Storage
- Effective Address
- Type & Size of Operands
- Operations add, sub, mul, . . .

( 0, 1, 2, 3 )

Where besides memory?

How is memory location specified?

byte, int, float, vector, . . .

How is it specified?

How is it specified?

## Other Aspects

- Successor
- Conditions
- Encodings
- Parallelism

How is it specified?

How are they determined?

Fixed or variable? Wide?

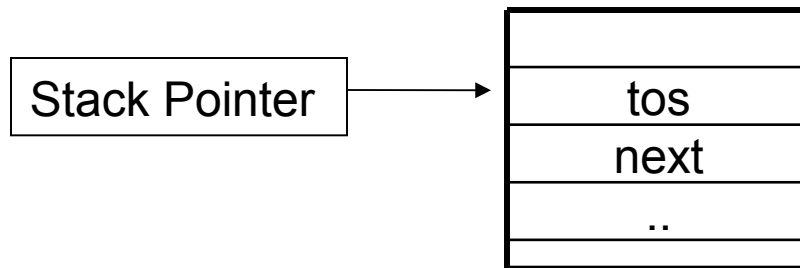
# Classificazione dell'ISA

Prof. G. Ascia

## Stack:

Gli operandi sono implicitamente sulla cima dello stack

0 operandi    `add`                     $\text{tos} \leftarrow \text{tos} + \text{next}$



## Accumulatore:

Un operando è implicitamente l'accumulatore

1 operando    `add A`     $\text{acc} \leftarrow \text{acc} + \text{mem}[A]$

1+x operandi    `addx A`     $\text{acc} \leftarrow \text{acc} + \text{mem}[A + x]$

# Classificazione dell'ISA

Prof. G. Ascia

## General Purpose Register

Gli operandi sono tutti espliciti: possono essere registri o memoria.

Abbiamo tre classi di GPR Architecture:

- **Memoria-Memoria:**

2 operandi `add A B`       $\text{mem}[A] \leftarrow \text{mem}[A] + \text{mem}[B]$

3 operandi `add A B C`       $\text{mem}[A] \leftarrow \text{mem}[B] + \text{mem}[C]$

- **Registro Memoria:**

2 operandi `add Ra, C`       $Ra \leftarrow Ra + \text{mem}[C]$

3 operandi `add Ra, Rb, C`       $Ra \leftarrow Rb + \text{mem}[C]$

- **Registro-Registro (Load/Store):**

3 operandi `add Ra Rb Rc`       $Ra \leftarrow Rb + Rc$

# General purpose register ISA

Prof. G. Ascia

Number of memory address	Maximum number of operands allowed	Examples
0	3	SPARC, MIPS, PowerPC, ALPHA
1	2	Intel 80x86, Motorola 68000
2	2	VAX (also has three-operans formats)
3	3	VAX (also has three-operans formats)

# Confronto

Prof. G. Ascia

- In riferimento all'istruzione di alto livello  $A := B + C$

STACK	REGISTRO- REGISTRO	MEMORIA- REGISTRO	MEMORIA- MEMORIA
PUSH B	LOAD R1, B	LOAD B	ADD A,B,C
PUSH C	LOAD R2, C	ADD C	
ADD	ADD R3,R1,R2	STORE A	
POP A	STORE A, R3		

# Alcuni esempi

Prof. G. Ascia

Machine	Number of registers	Architecture Style	Year
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	1	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



# General Purpose Registers

---

Prof. G. Ascia

- 1975-2011 tutte le macchine usano general purpose registers
- Vantaggi dei registri:
  - 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, cosicché i programmi sono più veloci
  - La densità del codice aumenta (poiché la codifica del nome di un registro richiede meno bit dell'indirizzo di una locazione di memoria)

# Indirizzamento della memoria

---

Prof. G. Ascia

- Ogni macchina usa indirizzi a livello di 8 bit (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** ? (big endian vs little endian)
  - **L'indirizzo iniziale di una word può essere qualsiasi?**

# Ordinamento degli indirizzi

Prof. G. Ascia

- **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 (Windows NT)

3	2	1	0
7	6	5	4

**Little Endian**

0	1	2	3
4	5	6	7

**Big Endian**



# Modalità di indirizzamento

Prof. G. Ascia

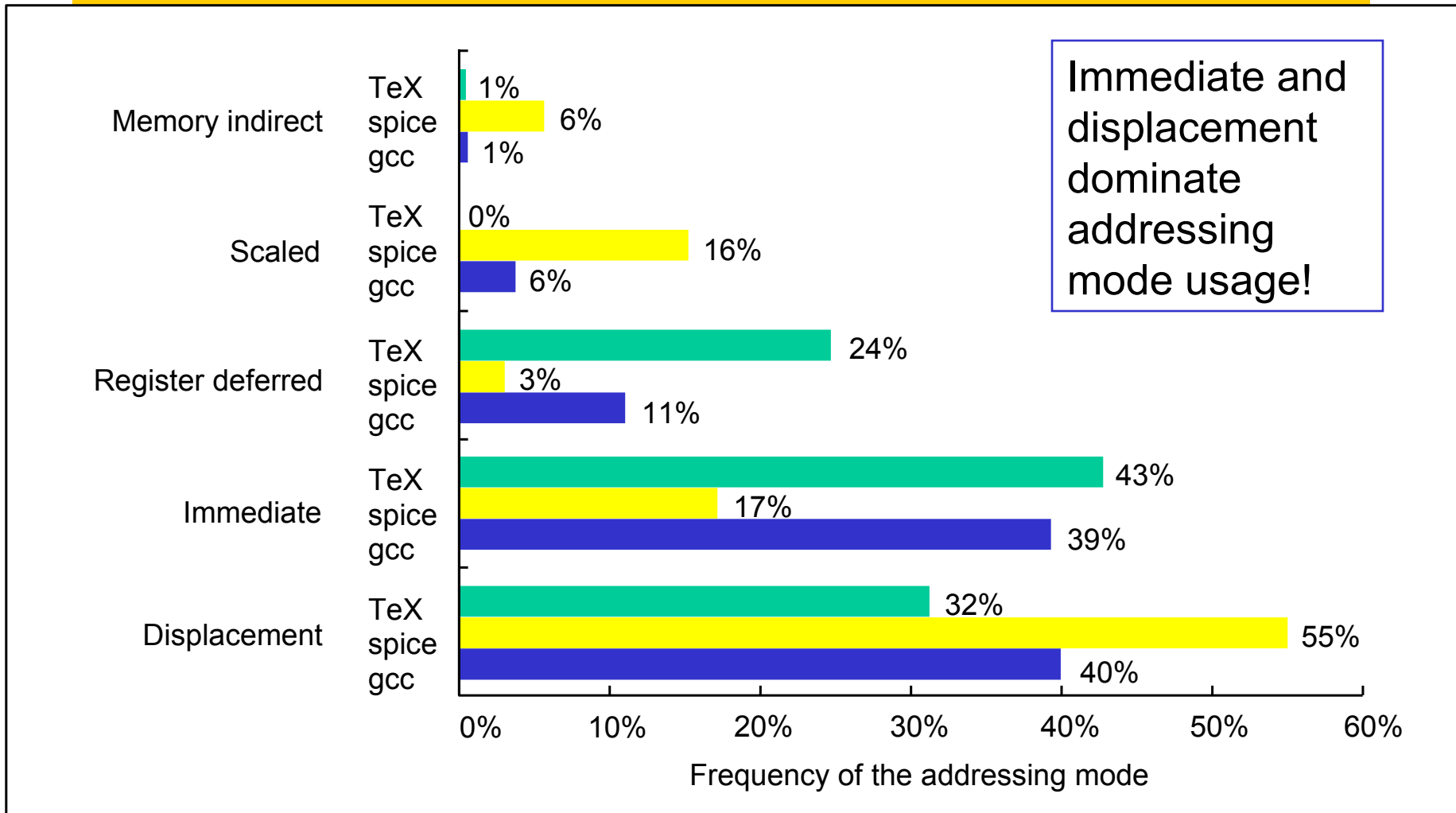
<b>Addressing mode</b>	<b>Example instruction</b>	<b>Meaning</b>	<b>When used</b>
<i>Register</i>	Add R4, R3	$\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Regs}[R3]$	<i>When a value is in a register.</i>
<i>Immediate</i>	Add R4, #3	$\text{Regs}[R4] \leftarrow \text{Regs}[R4] + 3$	<i>For constants.</i>
<i>Displacement</i>	Add R4, 100 (R1)	$\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Mem}[100 + \text{Regs}[R1]]$	<i>Accessing local variables.</i>
<i>Register deferred or indirect</i>	Add R4, (R1)	$\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Mem}[\text{Regs}[R1]]$	<i>Accessing using a pointer or a computed address.</i>
<i>Indexed</i>	Add R3, (R1 + R2)	$\text{Regs}[R3] \leftarrow \text{Regs}[R3] + \text{Mem}[\text{Regs}[R1] + \text{Regs}[R2]]$	<b>Sometimes useful in array addressing:</b> R1 = <b>base of array</b> , R2 = <i>index amount</i> .
<i>Direct or absolute</i>	Add R1, (1001)	$\text{Regs}[R1] \leftarrow \text{Regs}[R1] + \text{Mem}[1001]$	<b>Sometimes useful for accessing static data;</b> address constant may need to be large.

# Modalità di indirizzamento

Prof. G. Ascia

<b>Memory indirect or memory deferred</b>	Add R1, @ (R3)	Regs [R1] ← Regs [R1] + Mem [Mem [Regs [R3] ] ]	<i>If R3 is the address of a pointer p, then mode yields *p.</i>
<b>Autoincrement</b>	Add R1, (R2) +	Regs [R1] ← Regs [R1] + Mem [Regs [R2] ] Regs [R2] ← Regs [R2] + d	<i>Useful for stepping through arrays within a loop. R2 points to start of array, each reference increments R2 by size of an element, d.</i>
<b>Auto-decrement</b>	Add R1, ⓓ (R2)	Regs [R2] ← Regs [R2] ⓓ d Regs [R1] ← Regs [R1] + Mem [Regs [R2] ]	<b>Same use as autoincrement.</b> Autodecrement/increment can also act as push/ pop to implement a stack.
<b>Scaled</b>	Add R1, 100 (R2) [R3]	Regs [R1] ← Regs [R1] + Mem [100+Regs [R2] + Regs [R3] * d]	<i>Used to index arrays. May be applied to any indexed addressing mode in some machines.</i>

# Measurement on add. modes



# Uso delle modalità di indirizzamento

Prof. G. Ascia

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

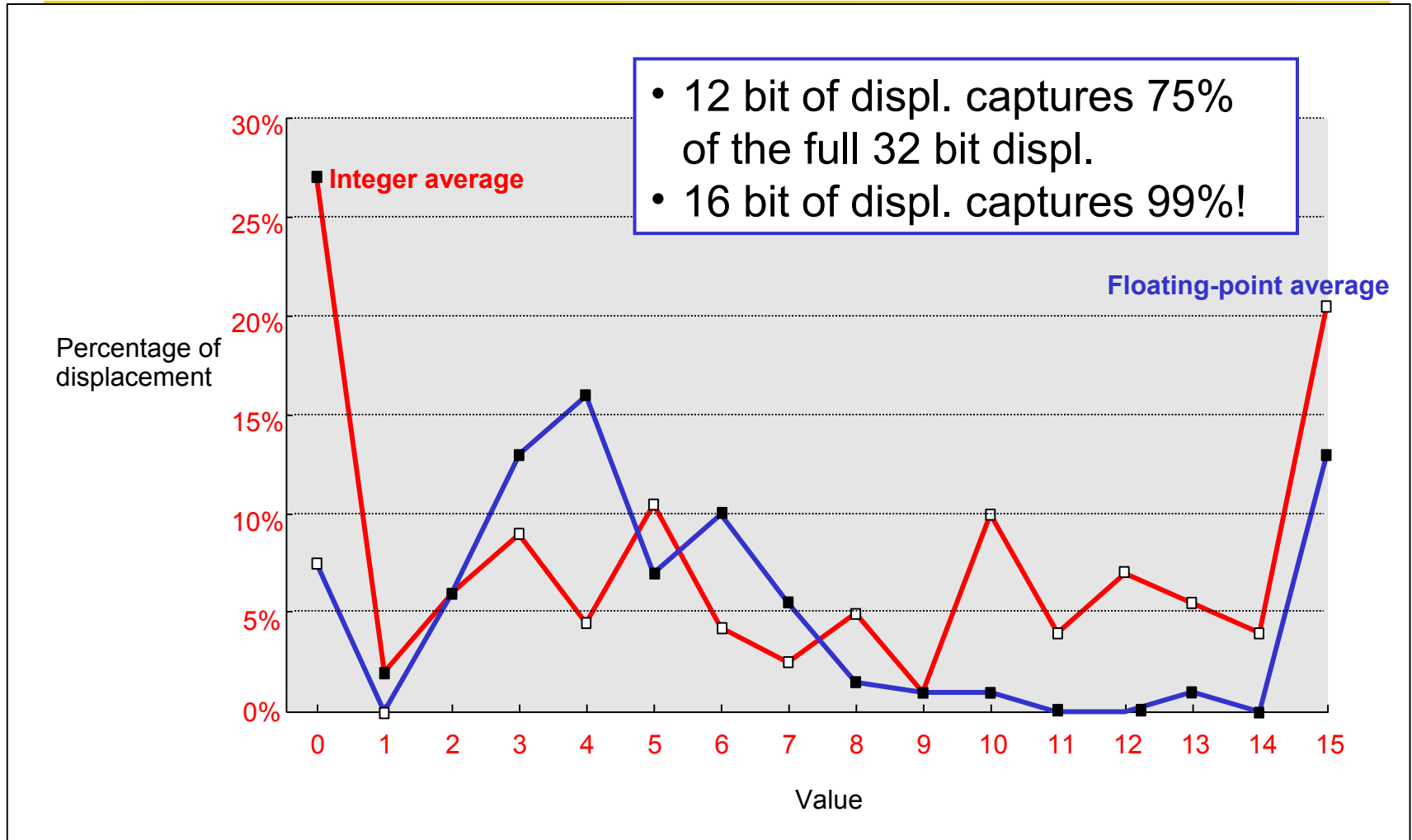
--- Displacement:	42% avg, 32% to 55%	↑ 75%
--- Immediate:	33% avg, 17% to 43%	↓ 88%
--- Register deferred (indirect):	13% avg, 3% to 24%	
--- Scaled:	7% avg, 0% to 16%	
--- Memory indirect:	3% avg, 1% to 6%	
--- Misc:	2% avg, 0% to 3%	

75% displacement & immediate

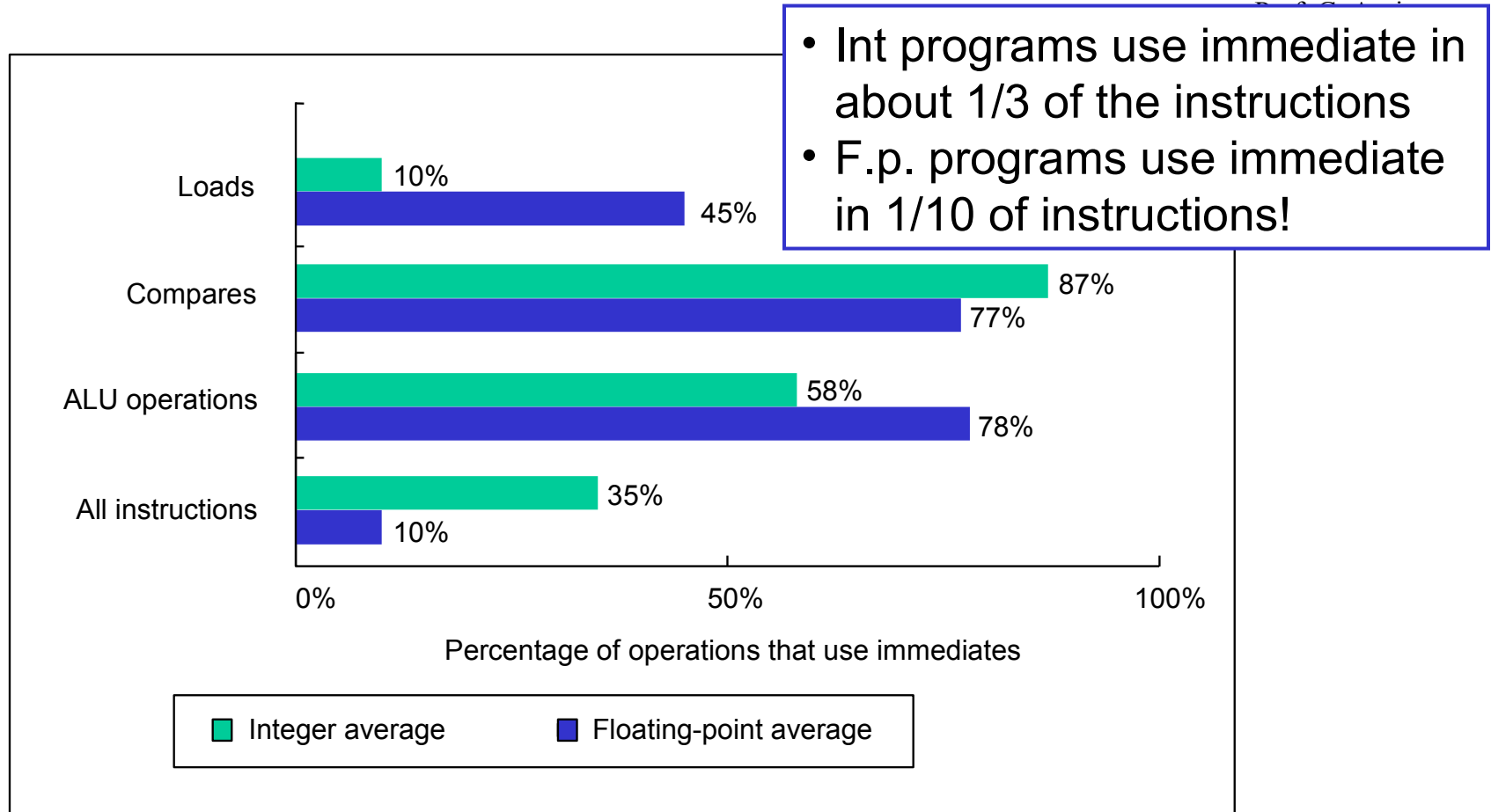
85% displacement, immediate & register indirect



# Range of displacement

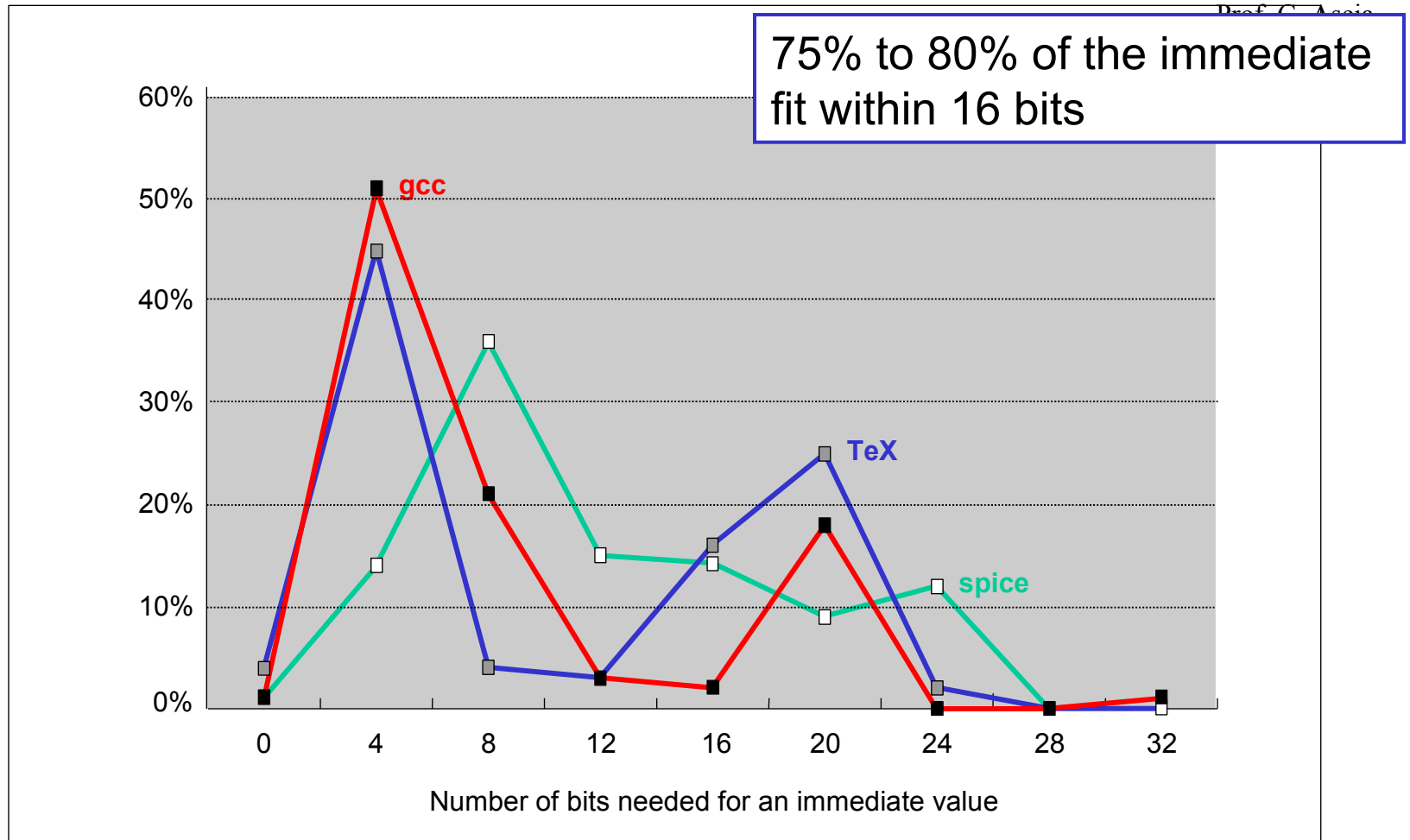


# Where Immediate has to be used?



# Values for Immediate

Prof. C. Ascia



# Frequency of operations in 80x86 Inst.Set.

## Average values for 5 programs in SPECint92

Prof. G. Ascia

Rank	instruction	Integer Average Percent total executed
1	load	22 %
2	conditional branch	20 %
3	compare	16 %
4	store	12 %
5	add	8 %
6	and	6 %
7	sub	5 %
8	move register-register	4 %
9	call	1 %
10	return	1 %
	Total	96 %

Simple instructions dominate instruction frequency

# Modalità di indirizzamento (controllo)

---

Prof. G. Ascia

- 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** ; EA destinazione = R30

# Mesurements on Instruction for Control Flow

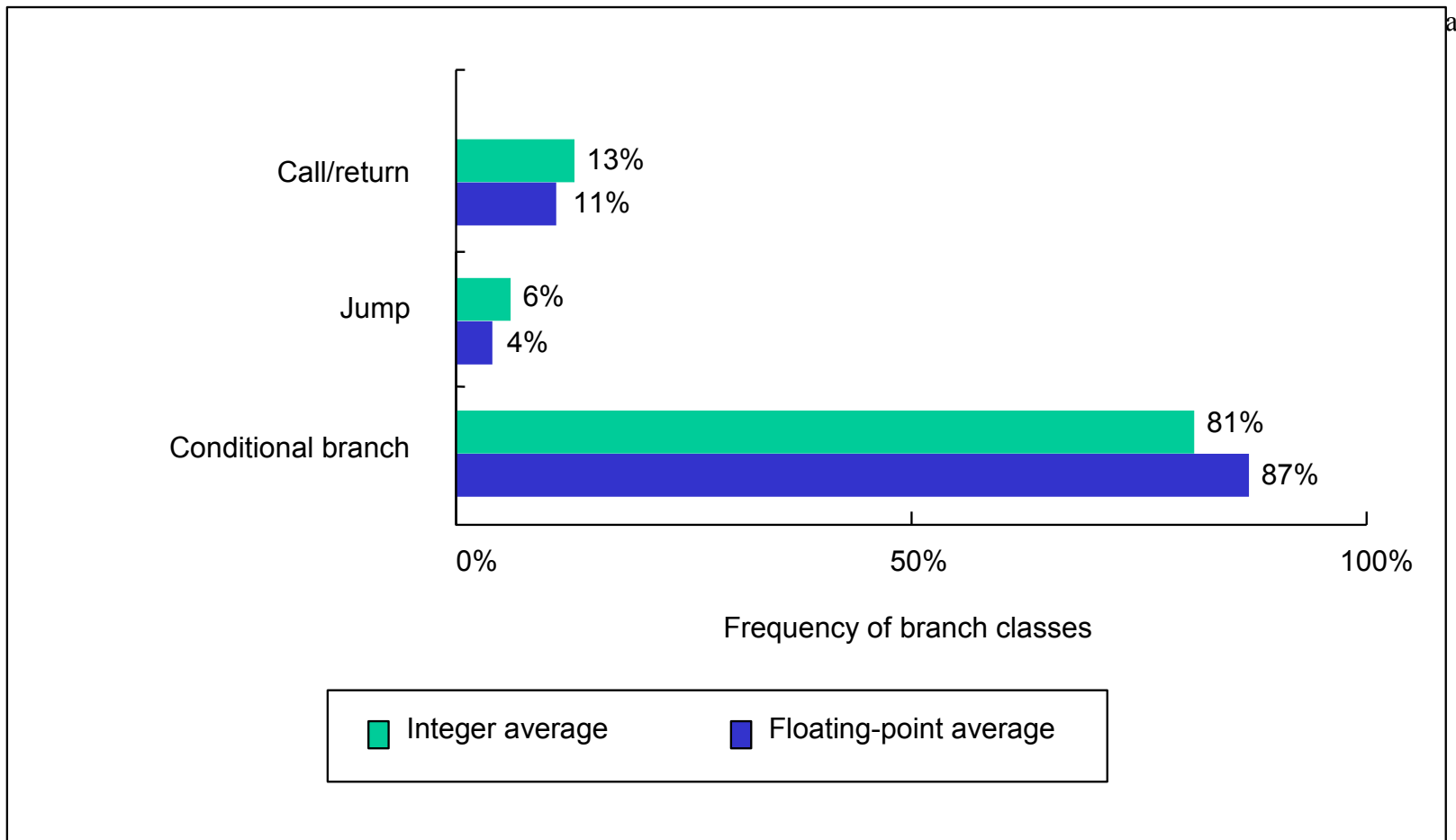


FIGURE 2.12 Breakdown of control flow instructions into three classes: calls or returns, jumps, and conditional branches.

# Size for Branch Displacement

Prof. G. Ascia

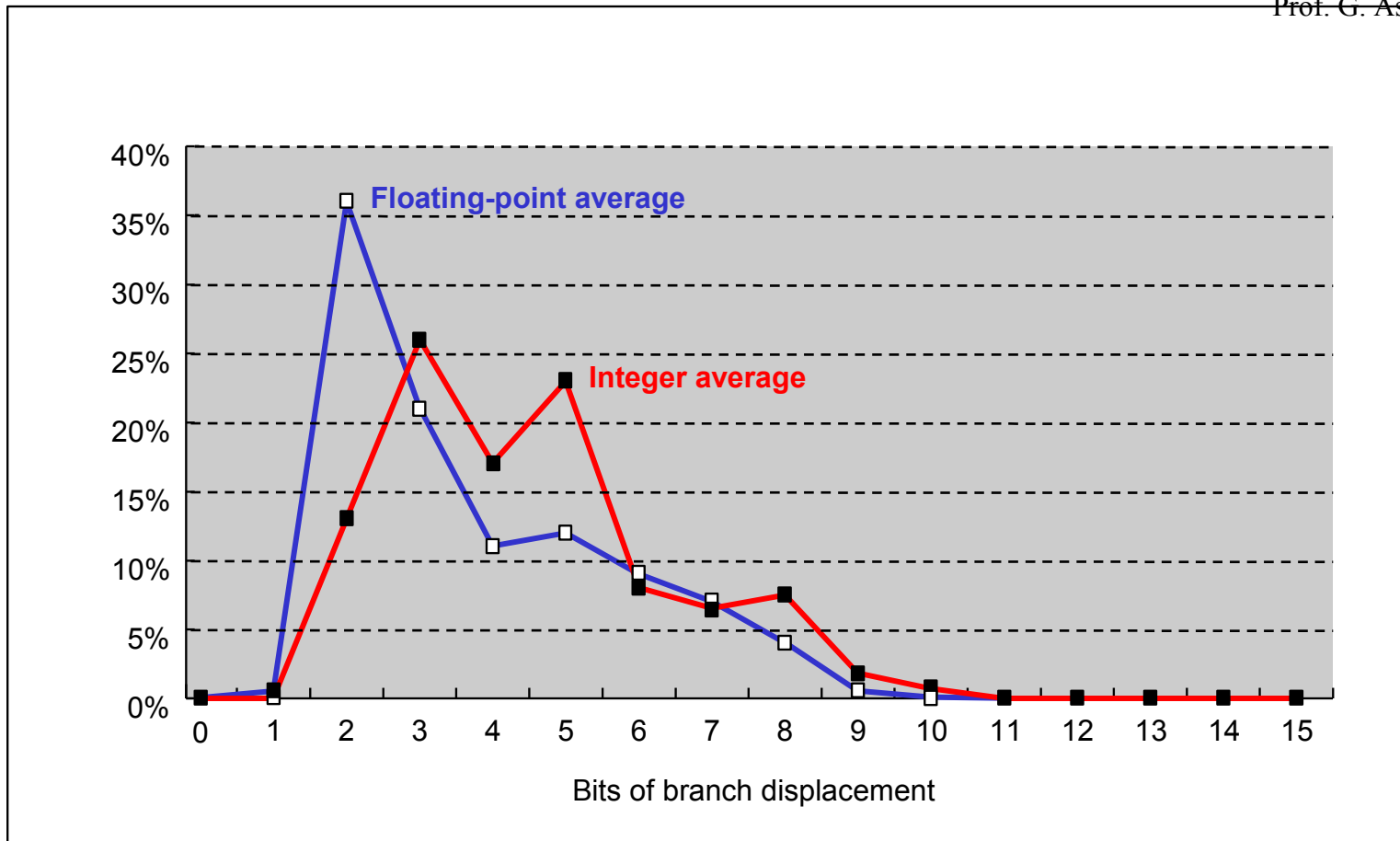


FIGURE 2.13 Branch distances in terms of number of instructions between the target and the branch instruction.

# Mesurements on Conditional Branches

Prof. C. Agnini

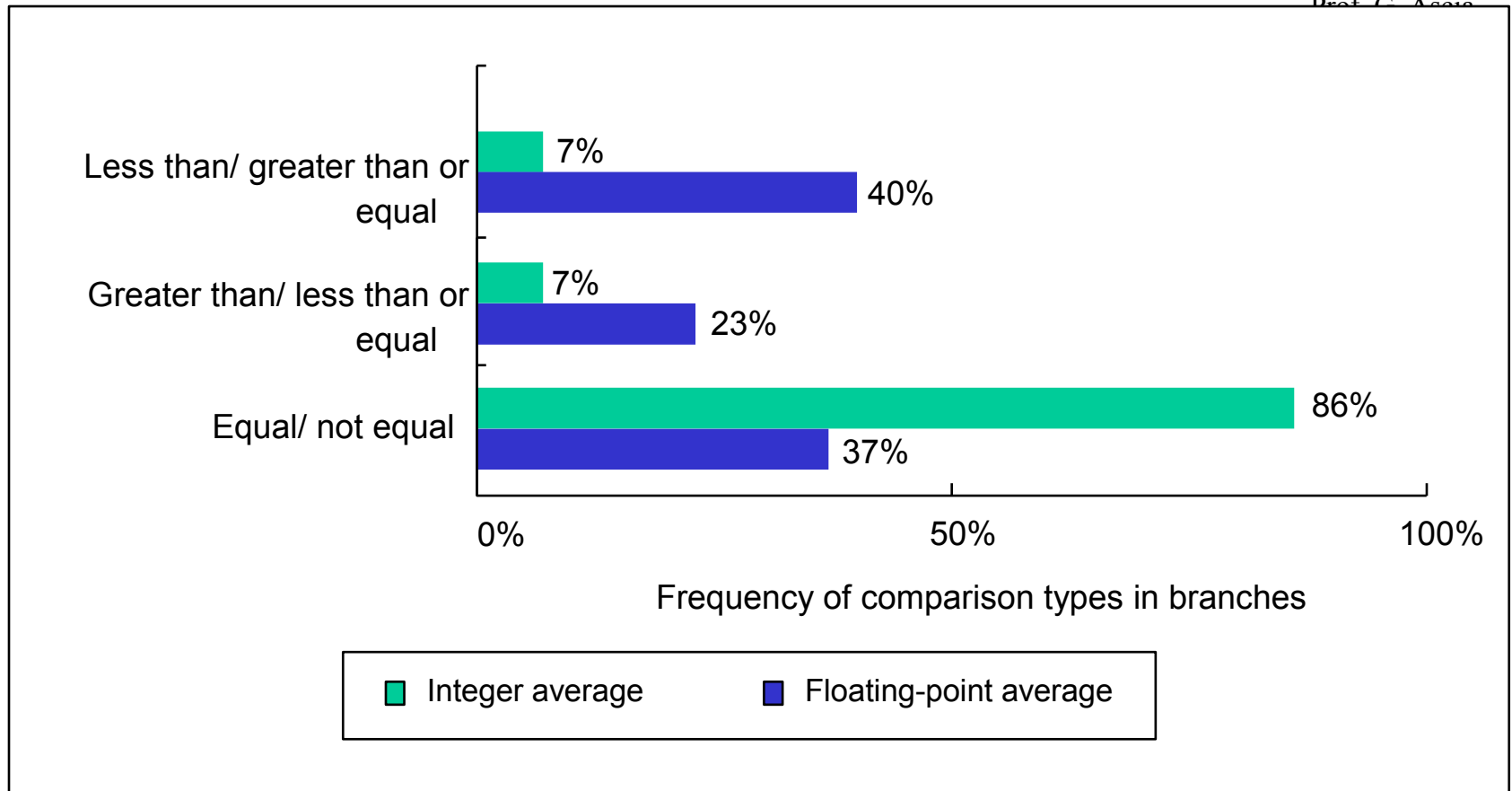


FIGURE 2.15 Frequency of different types of compares in conditional branches.



# Indirizzamento: riassumendo

---

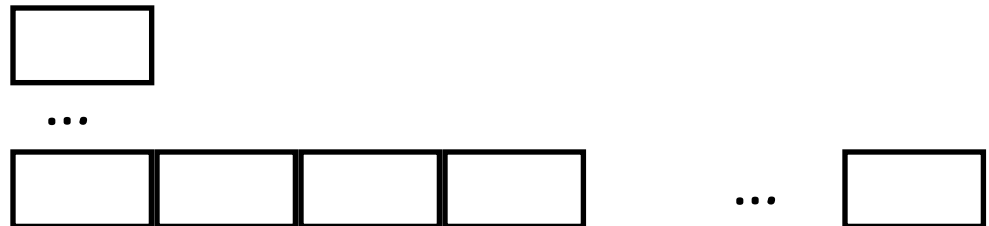
Prof. G. Ascia

- Modalità di indirizzamento dei dati più importanti:
  - Displacement, Immediate, Register Indirect
- Dimensione dello spiazzamento:
  - dai 12 ai 16 bit
- Dimensione degli operandi immediati:
  - dagli 8 ai 16 bit

# Formato delle istruzioni

Prof. G. Ascia

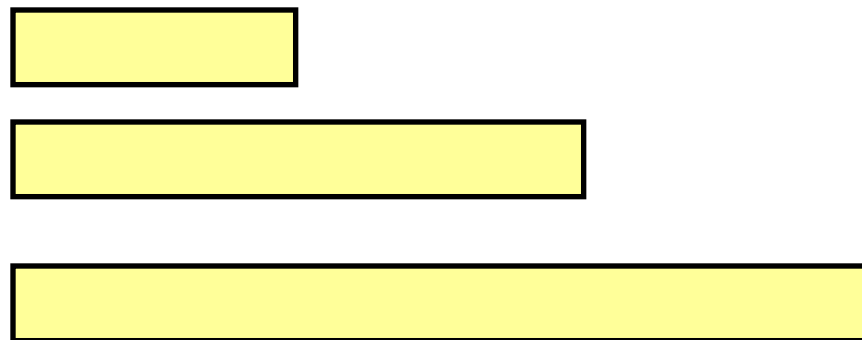
Variabile:  
(VAX)



Fissa:  
(MIPS,PowerPC)



Ibrida:  
(Intel 80x86)



# Formato delle istruzioni

---

Prof. G. Ascia

- Se la dimensione del codice è il requisito più importante, si usano istruzioni a lunghezza variabile
  - 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

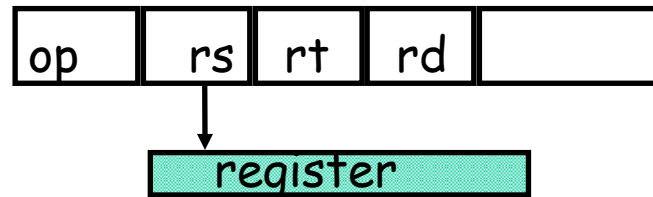
# MIPS: Modalità di indirizzamento

## Formato Istruzioni

Prof. G. Ascia

- Tutte le istruzioni solo di 32 bit

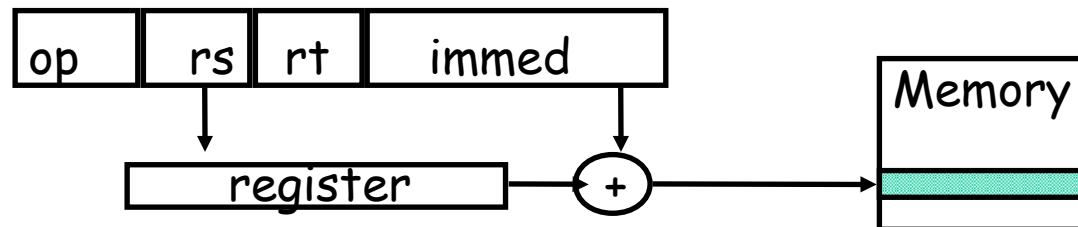
Register (direct)



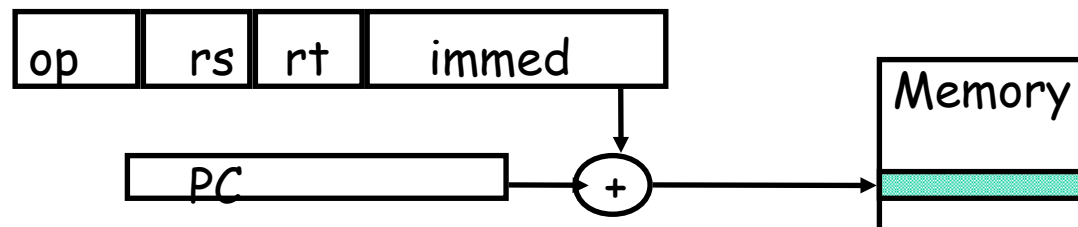
Immediate



Base+index

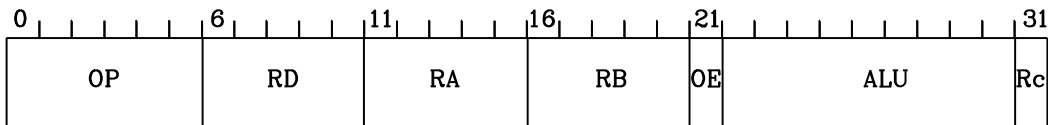


PC-relative



# Formato Power PC

Formato istruzioni aritmetiche



Esempio di istruzione aritmetica:

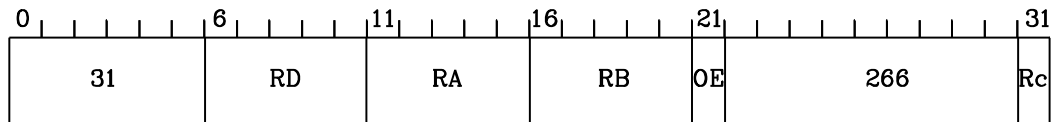
Istruzione di somma

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)



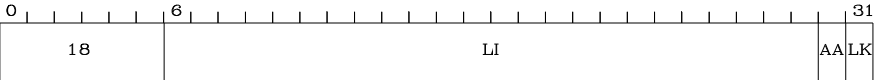
# PowerPC

Prof. G. Ascia

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



Formato istruzione di salto incondizionato



# Formato Intel

---

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

## Esempio

**MOV EAX, ES:V[EBX+EIX]** 6 byte

**PUSH EAX** 1 solo byte

**PUSH Var[EBX]** fino a 6 byte