PROGETTO DELLA CPU

ARCHITETTURA DELLE CPU come ARCHITETTURA DEL SET DI ISTRUZIONI

SCELTE PROGETTUALI:

1. DOVE SONO MEMORIZZATI GLI OPERANDI NELLA CPU?

2 QUANTI OPERANDI SONO CHIAMATI IN MODO ESPLICITO

3. DOVE SONO COLLOCATI GLI OPERANDI?

4. FORMATO DELLE ISTRUZIONI

5. CHE MODALITA' DI INDIRIZZAMENTO?

6. TIPO E STRUTTURA DEGLI OPERANDI?

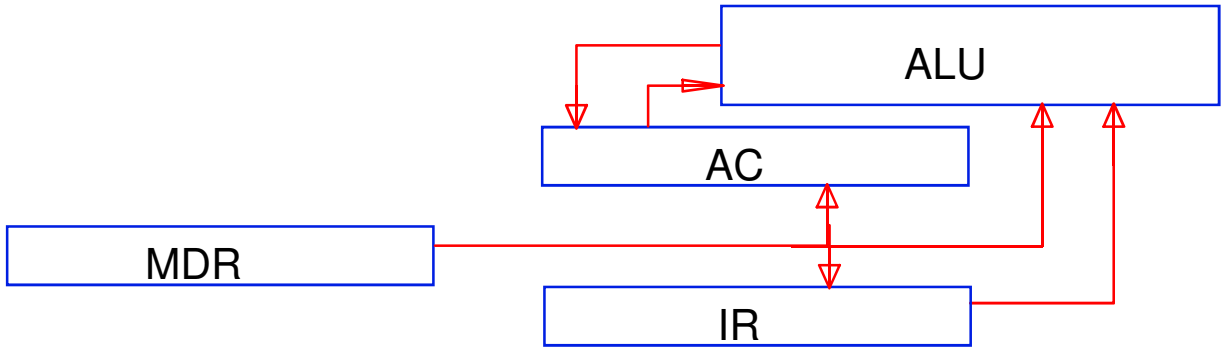
7. CHE TIPO DI OPERAZIONI SONO PREVISTE?

MEMORIZZAZIONE DEGLI OPERANDI

DOVE SONO MEMORIZZATI GLI OPERANDI NELLA CPU?

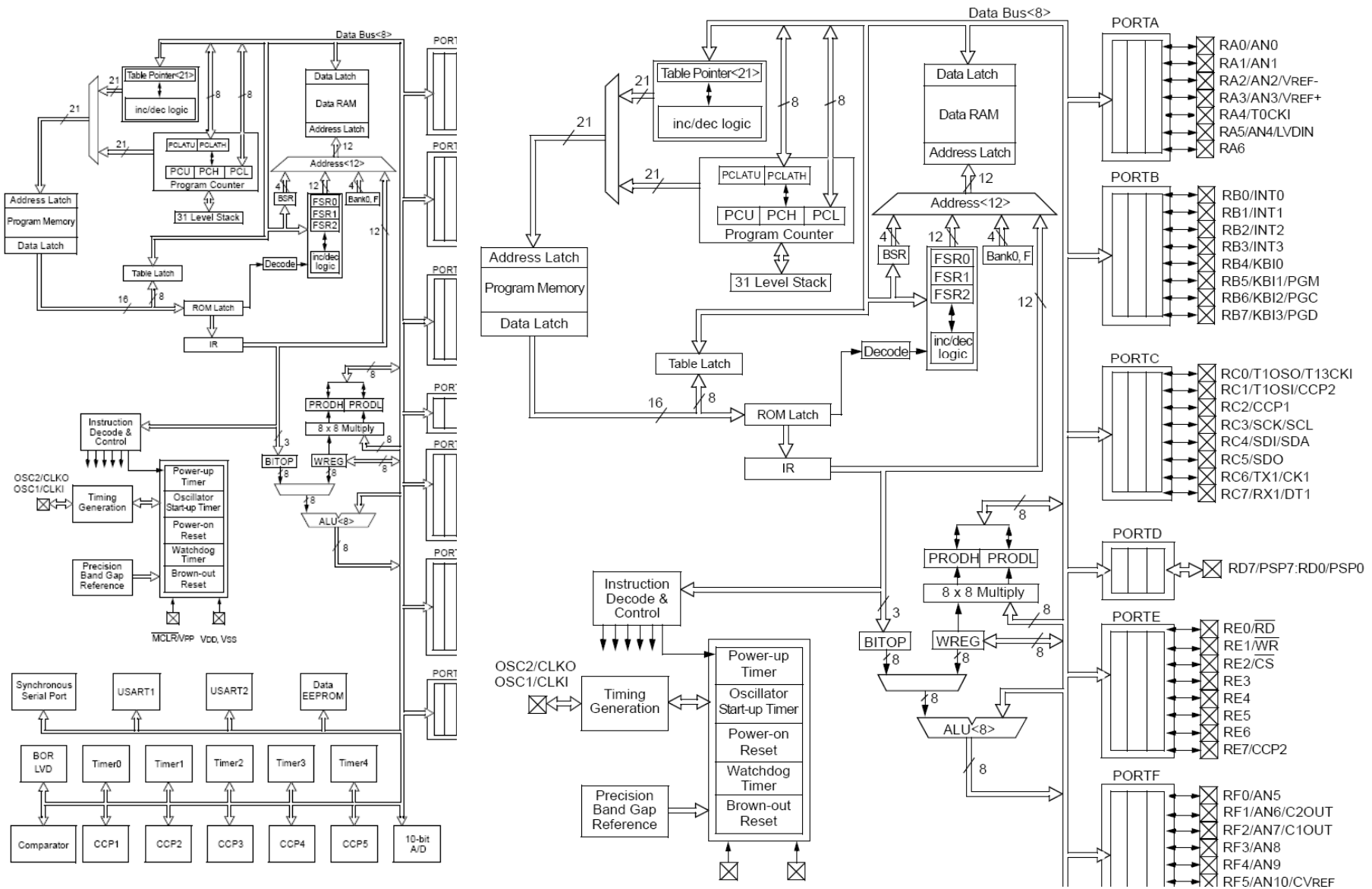
Memorizzazione degli operandi:
1. STACK
2. ACCUMULATORE
3. SET DI REGISTRI

MACCHINA AD ACCUMULATORE:



AC <- f(AC, MDR)

Esempio di macchina a registro accumulatore



ESEMPIO

C = A + B

STACK	*ACCUM.*	*GENERAL (register)*
PUSH A	LOAD A	LOAD R1,A
PUSH B	ADD B	ADD R1,B
ADD	STORE C	STORE C,R1
POP C		

- STACK -> Difficoltà di accesso, collo di Bottiglia; vantaggi: indipendenza dal register set. Esempi: Java Virtual Machine; unità floating point dei processori Intel x86
- ACCUM -> Gestione più semplice, ma accumulatore collo di bottiglia
- REGISTER -> Molto generale, tutti gli operandi espliciti, codice più lungo

ARCHITETTURA
A SET DI REGISTRI

Altra scelta: set di registri ortogonali o no

Ortogonalità: possibilità per le istruzioni di usare come operandi registri qualsiasi

Intel x86:

8 registri **non** ortogonali (4 dati e 4 indice)

Motorola 68000:

8 registri di dato a 32 bit
8 registri indirizzo a 32 bit

Sun SPARC:

32 registri a 32 bit, ortogonali

Digital Alpha AXP:

32 registri a 64 bit, ortogonali

I registri sono la risorsa più veloce. Se le variabili sono molte e i registri sono pochi, bisogna in continuazione allocare parte delle variabili sui registri, e salvare in memoria il valore delle variabili deallocate.

ARCHITETTURA
A SET DI REGISTRI

QUANTI OPERANDI SONO CHIAMATI IN MODO ESPLICITO?

DOVE SONO COLLOCATI GLI OPERANDI?

CLASSIFICAZIONE IN BASE AGLI OPERANDI NELLE ISTRUZIONI PER LA ALU:

1. N. DI INDIRIZZAMENTI A MEMORIA (0:3)
2. N. DI OPERANDI ESPLICITI (0:3)

(n. op. memoria, n. op. registri)

register/register	(0,3)	Macchine RISC: MIPS, SPARC,I860
register/memory	(1,2)	macchine CISC Motorola, Intel, IBM 360..
memory/memory	(3,3)	VAX

esempi:
RISC I ADD, RS, S2, Rd
x86: ADD AX,mem_var
i860: ADDS src1,src2,rdest

ARCHITETTURA

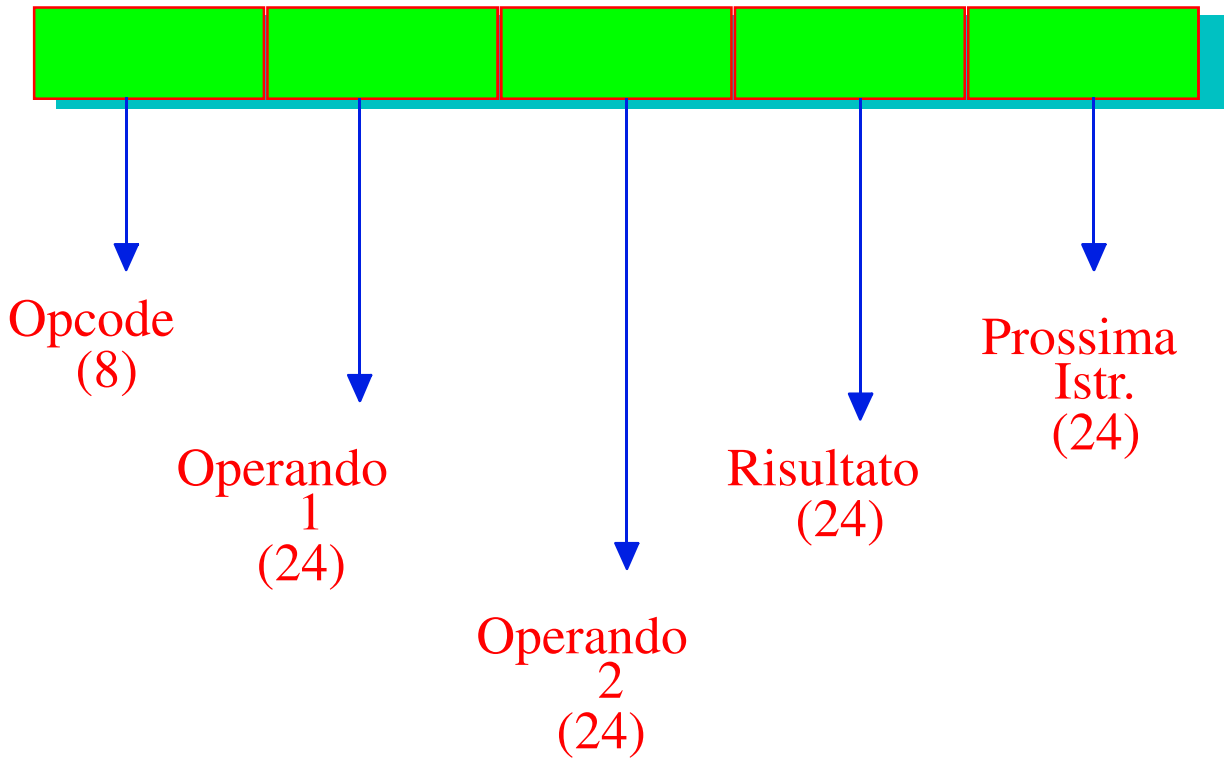
- Architetture *Register-Register (stile RISC)*
  Istruzioni di lunghezza fissa
  Tipicamente ugual numero di clock per istruzione

- Architetture *Register - Memory (stile Intel x86)*
  Istruzioni di lunghezza variabile (unità di controllo MOLTO più complessa); tipicamente anche diversa durata in tempo di esecuzione

- Architetture *Memory - Memory (stile IBM 360)*

Danno la massima flessibilità ma la memoria è un collo di bottiglia insormontabile

FORMATO DELLE ISTRUZIONI
per una macchina a 3 operandi espliciti



4 * 24 + 8 BITS !!!!!!!

FORMATO DELLE ISTRUZIONI

* LUNGHEZZA DELLE ISTRUZIONI?

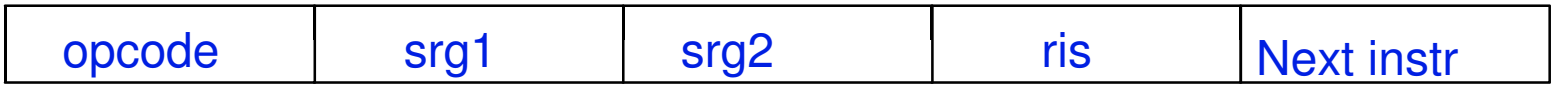
* lunghezza fissa o variabile

* struttura delle informazioni

* allineamento in memoria

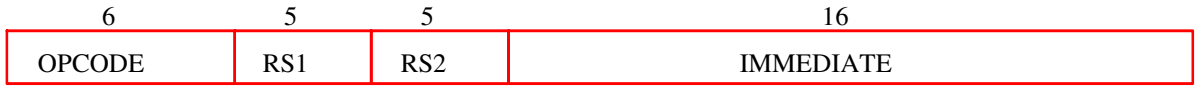
Formato base:

- cosa fare
- quali operandi
- come proseguire



FORMATO DELLE ISTRUZIONI NEL DLX

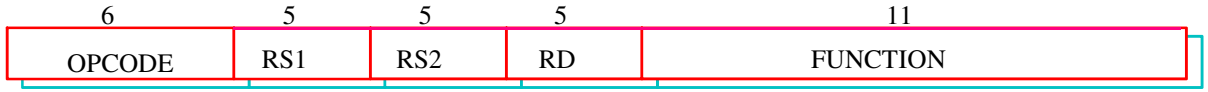
I



- (Rd ≡ RS2) ← [RS1 op imm] se LOAD
- [RS1 op imm] ← (RS2) se STORE
- RS2 non usato in BRANCH (posto=0)

- RS2 = 0, RS1 dest. in JR e JALR (via registro)
- ALU con immediate

R



- Rd ← RS1 funct RS2 (funct ← datapath)
  - SUB R1, R2, R3

    Rd RS1 RS2

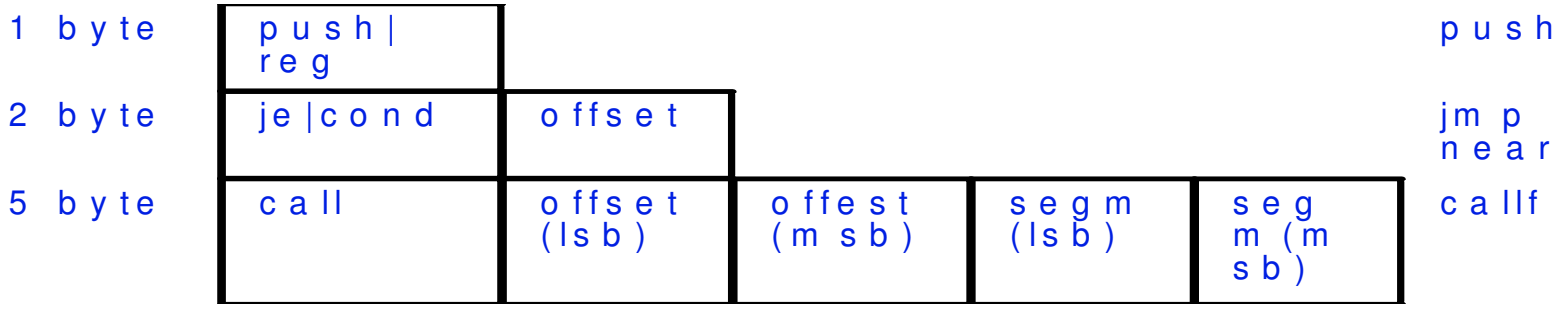
n.b. Qui la posizione degli identificativi dei registri è diversa rispetto al formato I. Dipende dall'OPCODE!

J

- J e JL (non register)
- TRAP (IAR ← PC; PC ← PC +offs)
- RFE (return form exception)

FORMATO DELLE ISTRUZIONI NELL'8086

FORMATO NON UNICO

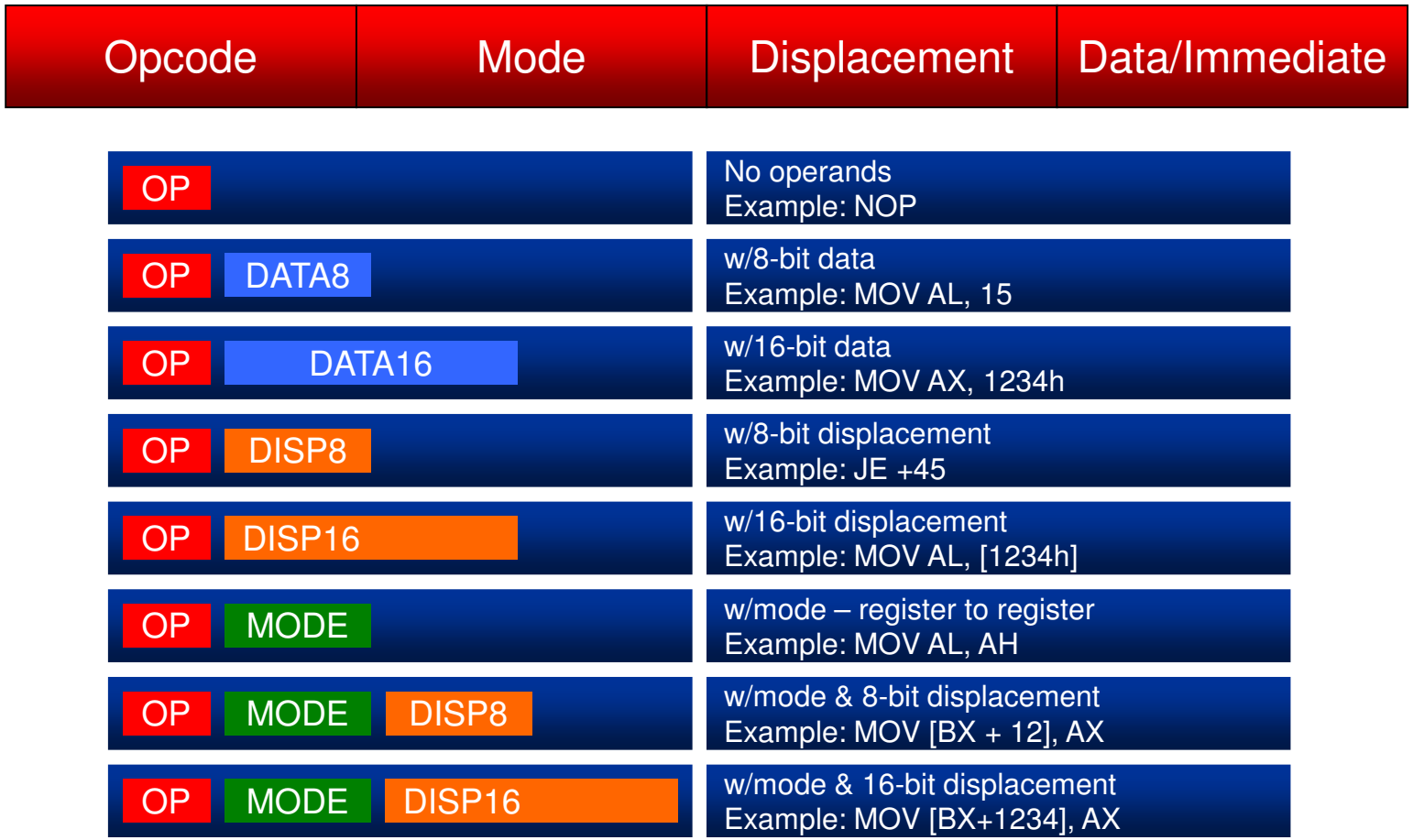


- L'istruzione può essere preceduta da un byte di prefisso
- Istruzioni con riferimento a operandi:

POST BYTE: codice operativo extra usato per specificare la modalità di indirizzamento

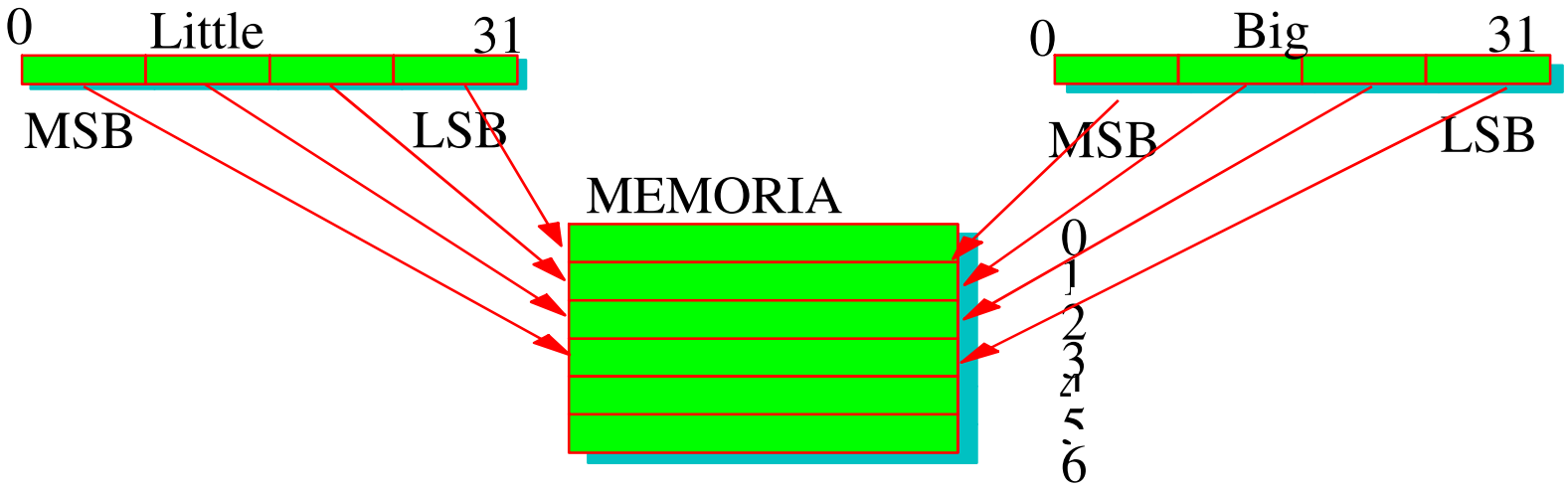
- Nel Pentium: fino a 12 byte! + prefisso di max 4 byte!

Formato generico delle istruzioni



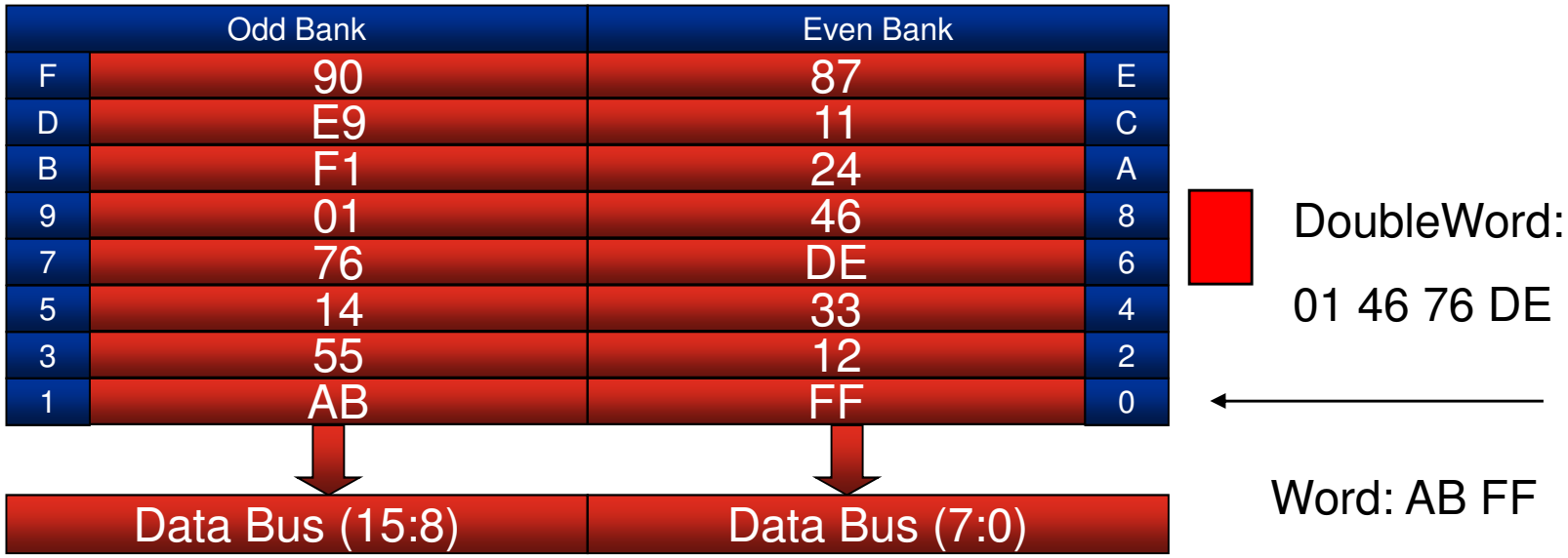
STRUTTURA DELLE INFORMAZIONI

LITTLE / BIG ENDIAN



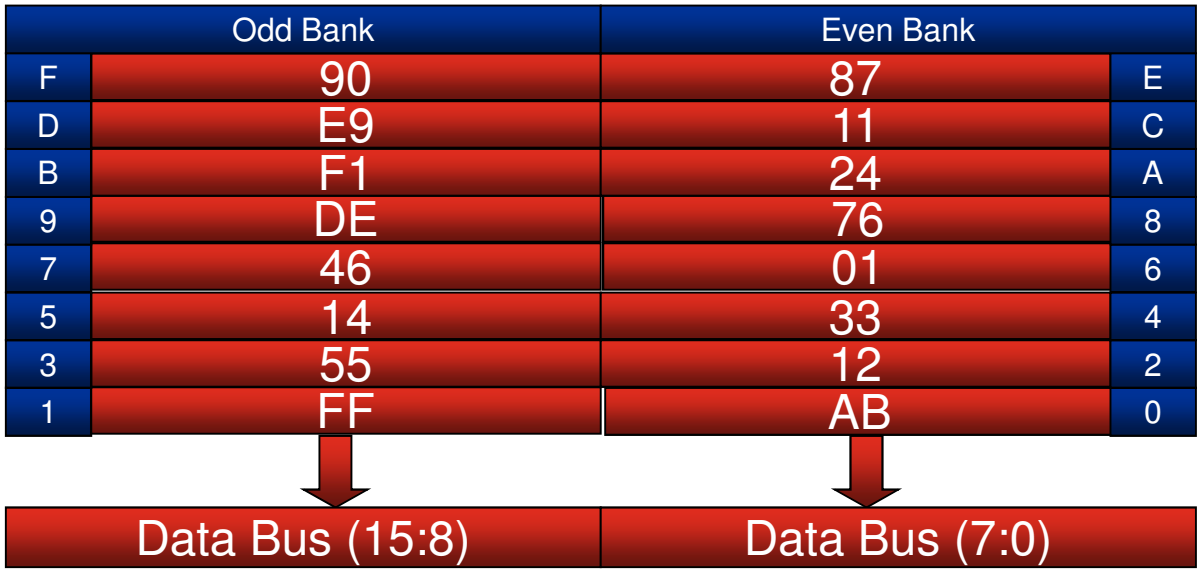
Attenzione: i bit all'interno del byte sono sempre in big-endian (il più significativo a sinistra)

Little Endian



- La famiglia x86 usa il byte order di tipo “little endian”
  - L’indirizzo richiamato (addr. 0) punta al byte meno significativo del dato da trattare
  - Il byte di ordine maggiore del dato risulta essere all’indirizzo immediatamente superiore adiacente al precedente (addr. 1)

Big Endian



DoubleWord invertita:

01 46 76 DE

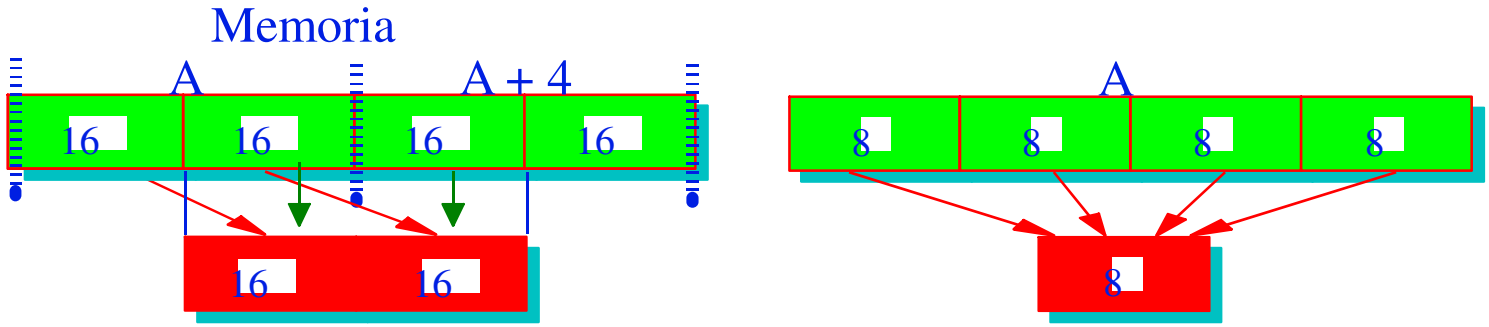
Word invertita:

AB FF

- Little endian vs. big endian
  - In big endian il byte più significativo del dato si trova all'indirizzo richiamato dal data pointer
- Il Byte ordering è una caratteristica della architettura! (es. Motorola 68000)

ALLINEAMENTO

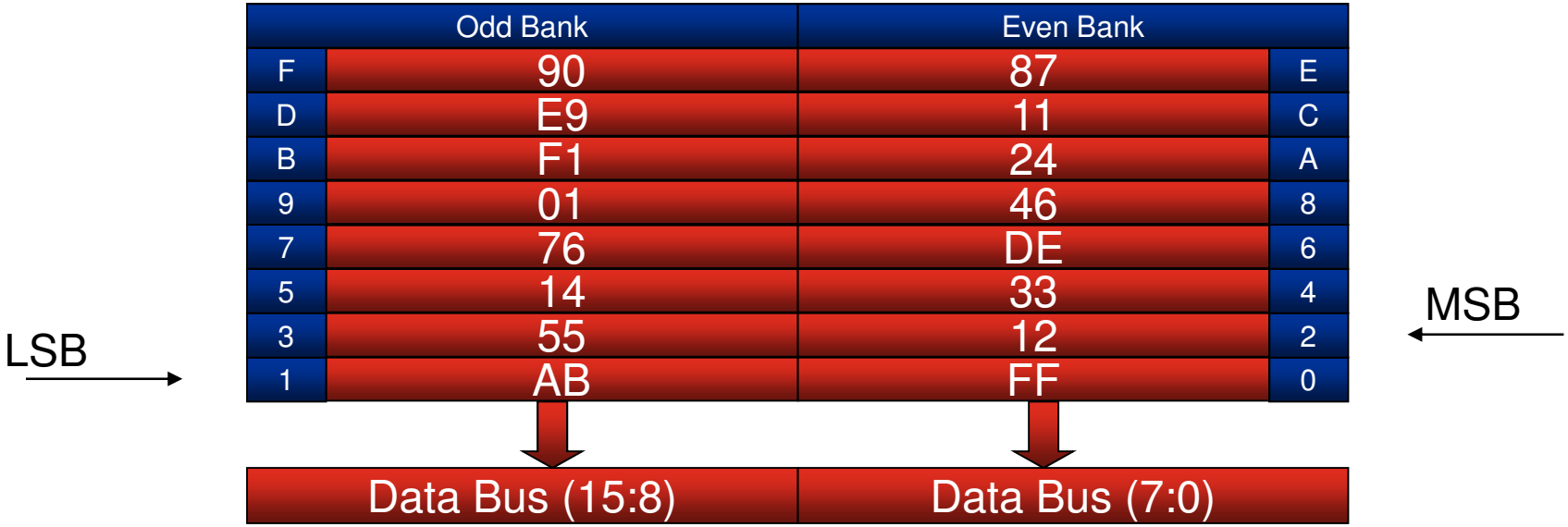
A mod S = 0 (S = n. bytes ; A = indirizzo)



- Il disallineamento obbliga a multiplexer più complessi e a meccanismi più complessi del controllo memorie (ex. scrittura dei soli bytes da modificare)
- Con l'allineamento ogni accesso alla memoria richiede un solo ciclo

allineamento

- Esempio di disallineamento in sistema in Little endian



- Cosa accade se leggo una word all'address 1 ?
- L'accesso a indirizzi dispari è permesso, l'addr. 1 rappresenta un indirizzo valido in memoria
- Il risultato della lettura della word è (dovrebbe essere): 12AB
- Ma i bytes nel data bus non sono allineati data(15:8)=AB,data(7:0)=12
- Servono due accessi al bus!

MODALITA' DI INDIRIZZAMENTO

INDIRIZZO EFFETTIVO:

CALCOLO DELL'INDIRIZZO FISICO DELLA LOCAZIONE DI MEMORIA

MAGGIORI MODALITA' DI INDIRIZZAMENTO,

-> MAGGIORE FLESSIBILITA',

-> DIMINUZIONE DEL NUMERO DI ISTRUZIONI,

COMPROMESSO

-> MAGGIORE LUNGHEZZA DELLE ISTRUZIONI

-> MAGGIORE COMPLESSITA' NELLA CODIFICA DELLE ISTRUZIONI E DECODIFICA NELLA UNITA' DI CONTROLLO

METODI DI INDIRIZZAMENTO

Modo	Esempio	Funzionamento
Registro	Add R4,R3	R4 ← R4 + R3
Immediato	Add R4, #3	R4 ← R4 +3
Base	Add R4,100(R1)	R4 ← R4+M[100+R1]
Indiretto(R)	Add R4,(R1)	R4 ← R4+M[R1]
Indicizzato	Add R3,(R1+R2)	R3 ← R3+M[R1+R2]
Diretto	Add R1,(1001)	R1 ← R1+M[1001]
Indiretto(M)	Add R1,@(R3)	R1 ← R1+M[M[R3]]
Autoincr.	Add R1,(R2)+	R1 ← R1+M[R2]
		R2 ← R2+d
Autodecr	Add R1,- (R2)	R2 ← R2-d
		R1 ← R1+M[R2]
Scalato	Add R1,100(R2)[R3]	
		R1 ← R1+M[100+R2+R3*d]

(d = dimensione dell'elemento)

- Codifica del tipo di indirizzamento: specificatore
  (parzialmente utilizzato come estensione del codice operativo)

Esempi di indirizzamento

Instruction	Comment	Addressing Mode	Memory Contents
MOV AX, BX	Move to AX the 16-bit value in BX	Register	89 D8 OP MODE
MOV AX, DI	Move to AX the 16-bit value in DI	Register	89 F8 OP MODE
MOV AH, AL	Move to AH the 8-bit value in AL	Register	88 C4 OP MODE
MOV AH, 12h	Move to AH the 8-bit value 12H	Immediate	B4 12 OP DATA8
MOV AX, 1234h	Move to AX the value 1234h	Immediate	B8 34 12 OP DATA16
MOV AX, CONST	Move to AX the constant defined as CONST	Immediate	B8 lsb msb OP DATA16
MOV AX, X	Move to AX the address or offset of the variable X	Immediate	B8 lsb msb OP DATA16
MOV AX, [1234h]	Move to AX the value at memory location 1234h	Direct	A1 34 12 OP DISP16
MOV AX, [X]	Move to AX the value in memory location DS:X	Direct	A1 lsb msb OP DISP16

Esempi di indirizzamento (2)

Instruction	Comment	Addressing Mode	Memory Contents
MOV [X], AX	Move to the memory location pointed to by DS:X the value in AX	Direct	A3 lsb msb OP DATA16
MOV AX, [DI]	Move to AX the 16-bit value pointed to by DS:DI	Indexed	8B 05 OP MODE
MOV [DI], AX	Move to address DS:DI the 16-bit value in AX	Indexed	89 05 OP MODE
MOV AX, [BX]	Move to AX the 16-bit value pointed to by DS:BX	Register Indirect	8B 07 OP MODE
MOV [BX], AX	Move to the memory address DS:BX the 16-bit value stored in AX	Register Indirect	89 07 OP MODE
MOV [BP], AX	Move to memory address **SS:BP** the 16-bit value in AX	Register Indirect	89 46 OP MODE
MOV AX, TAB[BX]	Move to AX the value in memory at DS:BX + TAB	Register Relative	8B 87 lsb msb OP MODE DISP16
MOV TAB[BX], AX	Move value in AX to memory address DS:BX + TAB	Register Relative	89 87 lsb msb OP MODE DISP16
MOV AX, [BX + DI]	Move to AX the value in memory at DS:BX + DI	Base Plus Index	8B 01 OP MODE

Esempi di indirizzamento (3)

Instruction	Comment	Addressing Mode	Memory Contents
MOV [BX + DI], AX	Move to the memo. Location pointed by DS:BX + DI the value in AX	Base Plus Index	89 01 OP MODE
MOV AX, [BX + DI + 1234h]	Move word in memory location DS:BX + DI + 1234h to AX register	Base Rel Plus Index	8B 81 34 12 OP MODE DISP16
MOV [BX + DI + 1234h], 5678h	Move immediate value 5678h to memory location BX + DI + 1234h	Base Rel Plus Index	C7 81 34 12 78 56

- MOV data
  - da memoria a registro
  - da registro a memoria
  - da registro ad altro registro
  - Mai da memoria a memoria!
- Syntassi
  - MOV destination, source (destination=source)
  - Destination e source possono essere registri o indirizzi di memoria

  Definiti in modi diversi che chiamiamo “addressing modes”

METODI DI INDIRIZZAMENTO

Quali sono i più utili?

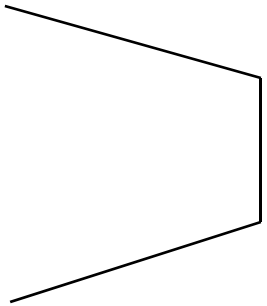
Esempio: benchmark sul VAX:

	spice	Tex
a mem. indiretta	6%	1%
scalato	0%	16%
a reg. indiretto	24%	3%
immediato	43%	17%
con base	32%	55%

METODI DI INDIRIZZAMENTO

Indirizzamento

con base (displacement)
immediato
indiretto di registro



75%-99%
dei test

Lunghezza dello spiazzamento ottimale:
12-16 bit (75%-99% dei casi)

Lunghezza del campo immediato 8-16 bit
(50% dei casi)

OPERAZIONI

- **TIPI DI OPERAZIONI**
- Aritmetiche e logiche (ALU op)
- Trasferimento di dati (LD, ST, MOV)
- Controllo (branch, jmp, call, ret, trap..)
- Sistema (memoria virtuale, protezione)
- Virgola mobile
- Decimale
- Stringhe (move, compare, search)
- Shift e rotazione
- Grafiche

- Trasferimento del controllo

  Branch condizionale (PC-relative ->
  Salti Principio di località)
  Chiamate
  Ritorni
  Nella maggioranza dei casi si tratta di un test sullo zero

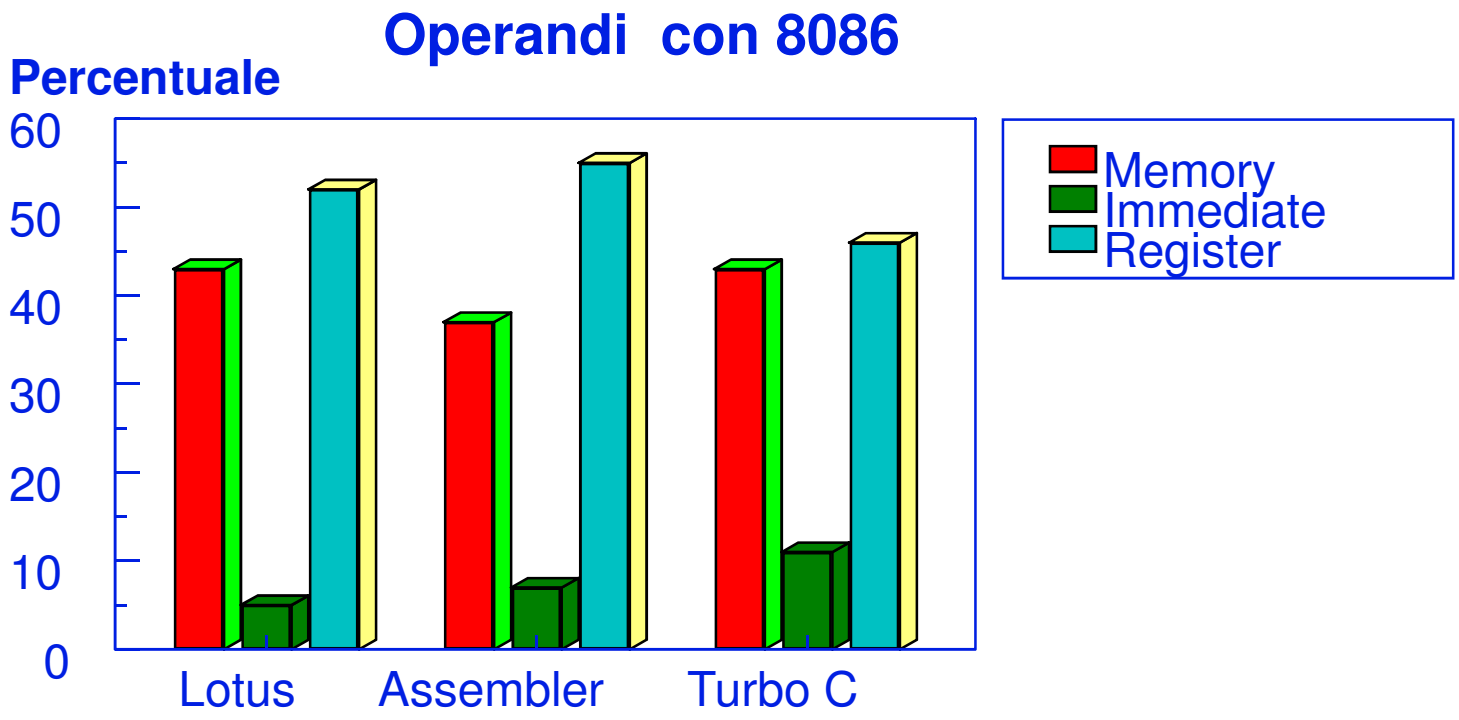
OPERANDI

- Operandi:
  - interi a 1, 2, 4 byte (esistono casi di operandi a 3 byte ma NON sono standard C e quindi non si usano: vedere ad esempio gli *shortlong* dei PIC18)
  - floating point a 4, 8 byte

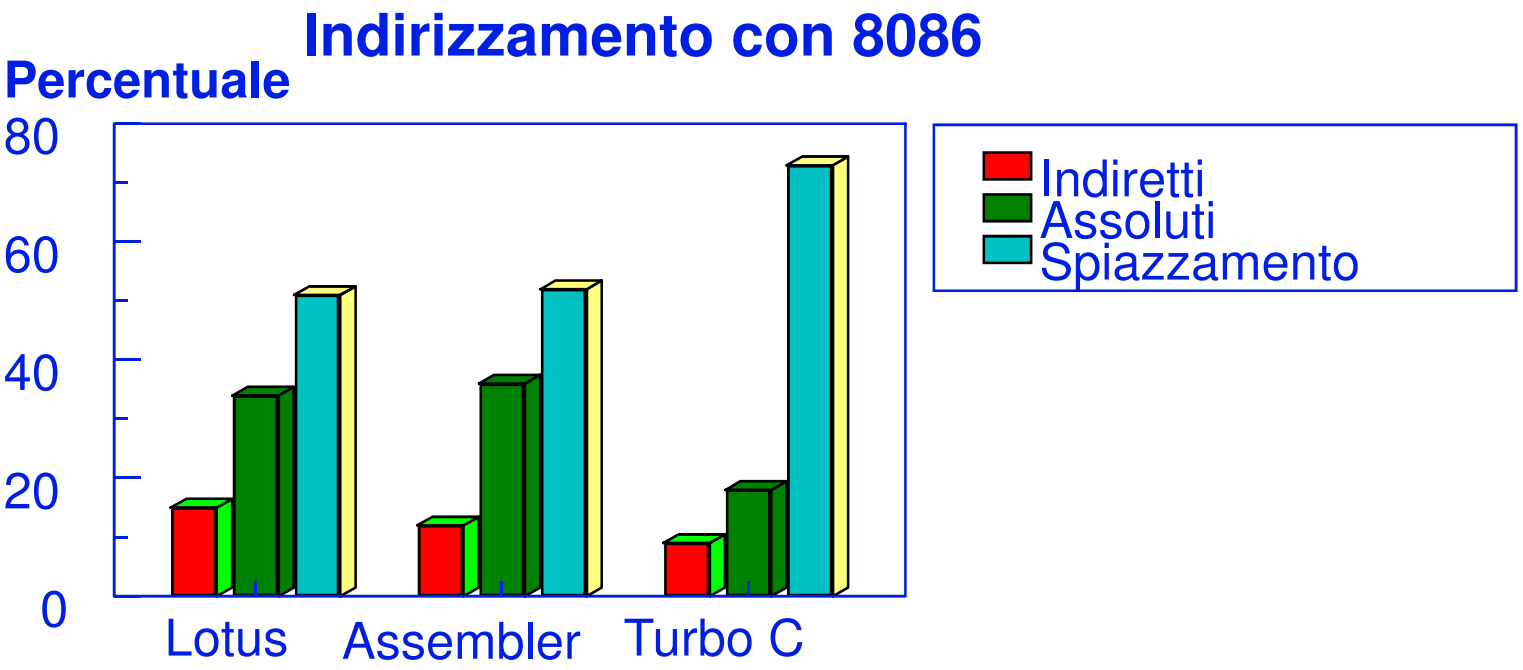
  Strutture dati non trattate nel set di istruzioni

  - Specifica del tipo di operando:
    - Indicato nel codice operativo
    - Tags attaccato al dato (ex. 8087)
    - Integer diverso da floating point (standard IEEE 754)
    - Caratteri: codice ASCII
  - Per tutti esiste il problema dell'allineamento
  - Operazioni di I/O

COMPILATORI



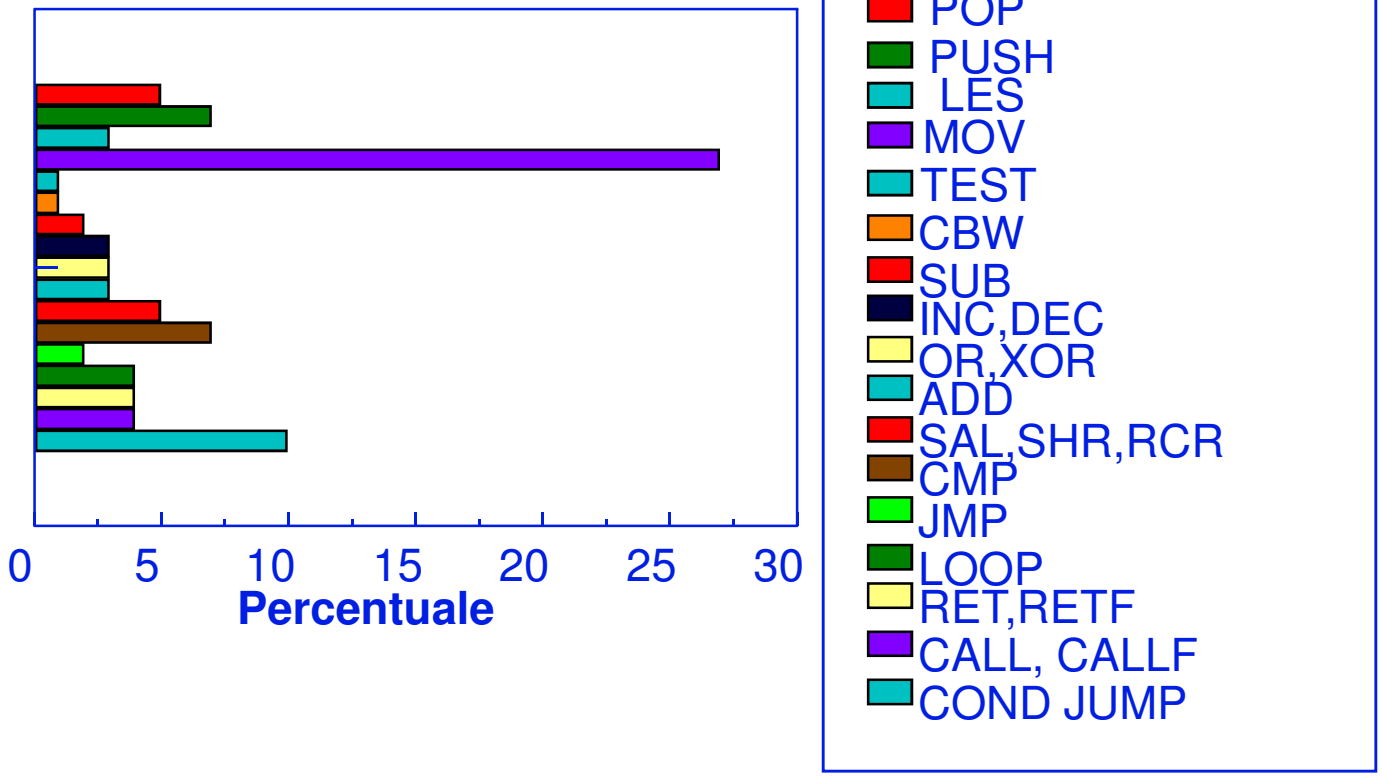
COMPILATORI



Indicizzati e based sommati insieme

COMPILATORI

Tipologia delle istruzioni con 8086



EVOLUZIONE DELLA ARCHITETTURA DI CPU

ARCHITETTURA CISC (anni 70)

PROGETTO E SVILUPPO MACCHINE CISC

'70 EVOLUZIONE DEGLI HLL (High Level Language)

COSTO DEL SOFTWARE

Aumento del gap semantico tra HLL e ML (Machine Language)

Aumento set di istruzioni (numero e complessità)

Supporto hardware agli HLL

Microprammazione

VAX 11/760 300 istruzioni

68020 18 modi di addressing.

80% programmi 20 % del set di istruzioni

sviluppo delle cache

analisi delle prestazioni

sviluppo dei compilatori

'80 PROGETTO E SVILUPPO MACCHINE RISC

(RISC i, RISC ii, MIPS, IBM801, i 860...)

RISC

RISC (1980 Diezel, Patterson)

1) STUDIARE LE OPERAZIONI PIU' FREQUENTI
2) OTTIMIZZARE L'HW IN BASE A 1)
AGGIUNGERE SOLO CIO' CHE NON DEGENERA 1) E 2)

Set di istruzioni "ridotto" (MIPS 31, RISC II 39)
PIC16 (Microchip 35 istruzioni)
PIC18 il doppio (dieci anni più tardi) e assembler che passa da 14 a 16 bit
complessità demandata ai compilatori: trasparente per i programmatori
No microcodice: controllo cablato
macchina load/store

istruzioni della stessa lunghezza
1 clock/cycle
pipeline
Superscalare
ILP..

Nel 2000

- Lo sviluppo della tecnologia dei semiconduttori ha consentito ai processori CISC di avvantaggiarsi dei concetti RISC, mantenendo un set di istruzioni complesso

- Molti altri concetti sono stati integrati:

  - Pipelining delle istruzioni
  - Out-of-order execution
  - Multithreading
  - Esecuzione speculativa
  - Predicate execution
  - …

ESEMPI

VAX 11/780 (1977 macchina CISC)

Lunghezza Istruzione (bit)	8,16,32,.. > 80
N. indirizzamenti a memoria	3
N. operandi espliciti	3
Registri interi	16
Registri f.p.	0
allineamento dei dati	no , Little Endian
spazio di indirizzamento	32 piatto
pagine	1/2 K
I/O	Memory mapped
Modi di indirizzamento	14

Es.:
add3 r1, 734(r2), #257

1 byte add3, 1 byte r1, 1 byte address +2 offset,

1 byte immediato+3 valore long ---> 10 byte

ESEMPI

Intel 8086 (1978)

Lung. istruzione	8,16,24,32,48
N. indirizzamenti a memoria	1
N. operandi espliciti	2
Registri interi	8 dedicati a 16 bit
Registri f.p.	0pz. 8 a 80
allineamento dei dati	no Little Endian
spazio di indirizzamento	20 segmentato
pagine	no (4k nell'80486)
I/O	Opcode (Istruzioni particolari)
Modi di indirizzamento	10

ESEMPI

SPARC (1992)

Lunghezza istruzione	32
N. indirizzamenti a memoria	0
N. operandi espliciti	3
Registri interi	32 a 32
Registri f.p.	32 a 32 IEEE754
allineamento dei dati	si Big Endian
spazio di indirizzamento	32 piatto
pagine	4k (64k)
I/O	Memory mapped
Modi di indirizzamento	2

ESEMPI

Pentium III (1998)

Lunghezza istruzione	Fino a 12 byte
N. indirizzamenti a memoria	1
N. operandi espliciti	2
Registri interi	8 (40) a 32
Registri f.p.	8 a80 bit IEEE754
allineamento dei dati	No Little Endian
Address bus	36 bit
pagine	4k (4M)
I/O	Opcode (Istruzioni particolari)
Modi di indirizzamento	11