Architettura degli Elaboratori e delle Reti

Lezione 29

L'architettura Intel

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Patterson, sezione 2.17

Sommario

Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni

Le prime architetture Intel

- **1978 – 8086**
  - Estensione del micro-processore 8080 utilizzato per applicazioni industriali. Stessa ISA. Architettura a 16 bit con registri a 16 bit. Parte dei registri è dedicata a compiti specifici, non è un'architettura con registri general-purpose.
- **1980 – 8087**
  - Coprocessore matematico. Dedicato a velocizzare le operazioni in virgola mobile. Modifica nel modo di gestire gli operandi. Questi potevano essere presi dallo stack oppure, uno di loro, dai registri. Gli operandi venivano presi sempre dalla cima dello stack.
  - Estensione degli operandi a 10 byte (80bit), *Extended Double Precision*.
    - E' il compilatore a potere dichiarare variabili su 10byte (Extended Double).
  - Ciclo di esecuzione di un'operazione aritmetica:
    - Push <operando_1> in stack (esteso a 10byte); Push <operando_2> in stack (esteso a 10byte).
    - Operazione.
    - Pop <risultato>.
- Limite nello spazio di indirizzamento: 1 Mbyte (2^{20}).
  - Indirizzamento operato come: `Base shl 4 + Offset`
- Gestione di memoria ed I/O tramite 3 segnali di controllo: **RD, WR, IO/MEM**

Le architetture Intel avanzate

- **1982 – 80286**: L'architettura diventa a 24 bit. Viene utilizzata una modalità di utilizzo protetta che consente di mappare le pagine di memoria in indirizzi privati. Aggiunta di istruzioni specifiche.
- **1985 – 80386**: Estensione a 32 bit. Archtettura, spazio di indirizzamento e registri a 32 bit. Nuove istruzioni, molto vicino ad un calcolatore con general-purpose registers. Pre-fetching. Paginazione della RAM.
- **1989-1992 – 80486**: Istruzioni per la gestione delle architetture multi-processore e per il trasferimento di dati condizionato. Cache. Pipe-line singola. Microprogrammazione per l'Unità di controllo (FSM).
- **1992-1995 – Pentium, PentiumPro**: Pipe-line multiple (super-scalare). Doppia CPU. Tecnologia MMX (Multi-media extension, SIMD). Cache primaria e secondaria (separata, con bus dedicato)
- **1997 – Pentium II**: Memorie cache a doppio accesso per ciclo di clock. Cache dei registri di segmento. PentiumPro + MMX.
- **1999 – Pentium III**: "Internet Streaming single instruction multiple data extensions (ISSE). Estensione dell'architettura MMX ad istruzioni floating-point. L2 cache e CPU nello stesso integrato.
- **2001 – Pentium 4**: Estensione del parallelismo e della Superscalarità. *Hyper-Threading Technology.*

Architettura x86

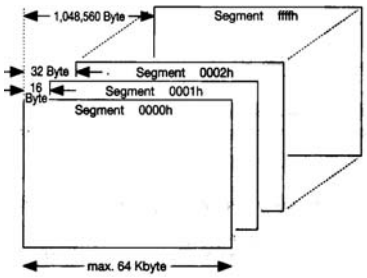
- CISC
  - Lunghezza istruzioni: 1 – 17 bytes
  - Operazioni direttamente in memoria
- Architettura condizionata dalla storia → necessità di compatibilità verso il basso
  - Real mode/Protected mode/Virtual 8086 Mode
- Nasce come architettura "not" general-purpose register
  - ogni registro è progettato per un uso specifico
  - dal 80386 in poi (IA-32) si definiscono 8 GP registers, ma non veri GP registers come in MIPS

Modi di funzionamento IA-32

- Modalità reale (Real mode)
  - Modalità compatibile DOS
  - Max memoria indirizzabile: 1 MByte → 2^{20}
  - modo attivo all'accensione (power-on)
- Modalità protetta (Protected mode)
  - modalità "nativa" di IA-32
  - Memoria indirizzabile: 4 GByte → 2^{32}
  - Memoria protetta: evita corruzione memoria da parte di altri programmi
  - Memoria virtuale: permette ad un programma di disporre di più memoria di quella fisica disponibile
- Modalità "8086 virtuale" (Virtual-8086 mode)
  - "Real mode" simulato all'interno del "Protected Mode"
  - Esecuzione di programmi DOS in multitasking con altri programmi che girano in Protected Mode.

Indirizzamento in modalità reale

Interleaving dei segmenti:
Spazio di indirizzamento di 1Mbyte suddiviso in segmenti di 64kbyte



- Modalità reale
  - Indirizzo = 16 * Segmento + Offset
  - In tal modo posso indirizzare 64KB x 16 = 1 MByte di MP
  - Modalità Virtual 8086: Indirizzo e offset sono separati.
    - CS = 0x80B8, IP = 0x019D
      → indirizzo istruz. successiva:

0x 80B8 0 +
019 D =

0x 80D1 D

Indirizzamento in modalità protetta

- Modalità protetta
  - Spazio di indirizzamento: 32 bit
  - I segmenti sono di 64Kbyte e non più di 16 byte.
    - SALTO: indirizzo = composizione registri CS e IP
    - Esempio:
      - CS = 0x80B8, IP = 0x019D
        → indirizzo istruz. Successiva mediante pipe dei due registri:
        0x 80B8 || 019D

Sommario

Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni

I registri dell'architettura 80286

AH	AL
BH	BL
CH	CL
DH	DL

SI
DI
BP

Segment Register

CS	
SS	
DS	
ES	
FS	
GS	

IP
SP

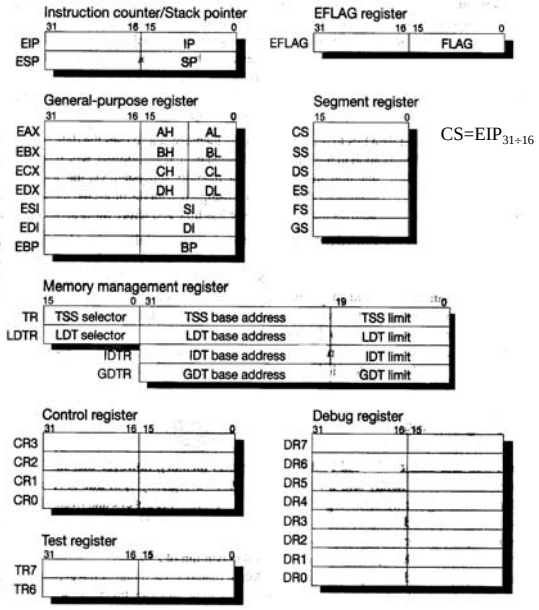
Registri di Segmento

Nome	Descrizione	Utilizzo
CS	Code Segment	Contiene l'indirizzo base dei (dati) ed istruzioni ad accesso immediato. Le istruzioni del segmento sono indirizzate tramite il registro **EIP** (Extended Instruction Pointer). Per modificare il registro **CS** occorre una *chiamata a procedura far* o una *far jump* oppure un interrupt (*int*). In modo protetto viene verificato se il nuovo segmento può essere utilizzato dal task.
DS (EBX)	Data Segment	Contiene l'indirizzo base del segmento dati del programma. Molte istruzioni quali la **mov** utilizzano questo segmento. E' il segmento in cui sono contenuti i dati di un task.
SS (EBP)	Stack Segment	Quasi del tutto simile allo stack del MIPS. Cresce verso il basso. Contiene i dati locali delle procedure e gli argomenti di chiamata. Contiene anche gli operandi per le operazioni aritmetiche di tipo accumulatore.
ES, FS, GS	Extra Segments	Principalmente utilizzati per operazioni su stringhe. Possono essere utilizzati in sostituzione di DS per accedere a dati al di fuori di DS. Il DOS ed il BIOS utilizzano spesso ES come buffer per le loro chiamate.

Si sovrappongono ai registri a 32 bit (e.g. SS → [BP, SS] = EBP)

I registri dell'architettura IA-32

- Estensione del set dei registri dell'80286.
- Così a partire dal 80386 (IA-32)
- 8 registri "general-purpose" a 32 bit (general puropose + ESP).
  - ma si deve potere accedere ad 1 o 2 byte al loro interno (compatibilità con 8086)
  - non sono poi così "general purpose"…
- I registri di segmento sono rimasti a 16 bit
  - usati come registri base
  - Possono essere utilizzati in alternativa ai registri estesi (e.g. DS <-> EBX).



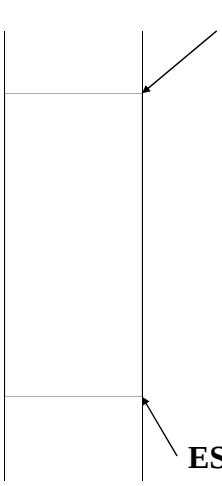
Utilizzo dei registri IA-32

- General Purpose Registers
  - General Data Registers
    - **EAX**: accumulatore (ottimizzato per op. aritmetico-logiche)
    - **EBX**: base register (registro base nel segmento dati, $gp)
    - **ECX**: counter (ottimizzato per i loops)
    - **EDX**: data register
  - General Address Registers
    - **EBP**: stack base pointer (base address dello stack, $fp)
    - **ESP**: stack pointer ($sp) (SS || SP)
    - **ESI**: source index (ottimizzato per op. su stringhe)
    - **EDI**: destination index (ottimizzato per op. su stringhe)
- Program counter
    - **EIP**: extended instruction pointer (CS + EIP <-> Program Counter)
- Floating-point Stack registers
  - ST(0) – ST(7): 80 bit, accessibili come LIFO (stack)
- SIMD Registers
  - MMX, SSE (SSE1, SSE2, SSE3, SSE4)

Registri “General Data”

Nome simbolico			Nome descrittivo	Funzioni
32 bit	16 bit	8 bit		
EAX	AX	AH, AL	Accumulator	Moltiplicazione/Divisione, I/O, shift veloce
`out 70h, al` ;Il contenuto di al viene trasferito alla porta 70h. `sb $al,offset($k1)` `# in MIPS, accesso memoria > 2Gbyte`				
EBX	BX	BH, BL	Base Register	Puntatore all’indirizzo base segmento dati
`mov ecx, [ebx]` ;trasferisci quanto presente all’indirizzo 0($ebx) in ecx `lw $s0, 8000($gp)`				
ECX	CX	CH, CL	Count Register	Indice di conteggio per cicli, rotazioni e shift
`move ecx, 10h` ; load ecx con 10h (=16), valore di inizio conteggio (associato a “loop”) `start: out 70h, al` ; Il contenuto di al viene trasferito alla porta 70h. `loop start` ; ritorna ad inizio ciclo, il quale verrà ripetuto 16 volte (fino a che ecx = 0)				
EDX	DX	DH, DL	Data Register	Moltiplicazione/Divisione, I/O
`mul edx` ; moltiplica edx con eax (implicito), il risultato è contenuto nella coppia edx:eax (hi:lo).				

Lo stack viene delimitato



L'area di memoria privata della procedura è compresa tra EBP ed ESP. Viene utilizzato per memorizzare l'indirizzo di ritorno, per memorizzare i parametri di procedure e per memorizzare le variabili locali della procedura.

Registri di stack

Nome simbolico			Nome descrittivo	Funzioni
32 bit	16 bit	8 bit		
ESP	SP	x,x	Stack Pointer	Stack Pointer
EBP	BP,SS	x, x	Base Pointer	Indirizzo base del segmento di Stack (32 bit)

```
   push sum1    ; create the first summand
                ; (automaticamente ESP viene decrementato)
   push sum2    ; create the second summand
   push sum3    ; create the third summand

addition: proc near    ; call near (inside 64k segment, cf. branch)
   push ebp            ; salva l'indirizzo base per il ritorno
                       ;(inzio del record attivaz.)
   move ebp, esp       ; copia lo StackP nel BaseP (individua frame
                       ; di procedura)
   move eax, [ebp+12]  ; carica sum1 in EAX
   add eax, [ebp+8]    ; somma in eax sum1 + sum2
   add eax, [ebp+4]    ; somma in eax aum1 + sum2 + sum3
   pop ebp             ; recupera l'indirizzo base precedente
   ret                 ; ritorno al programma chiamante (cf. jr $ra)
addition endp
```

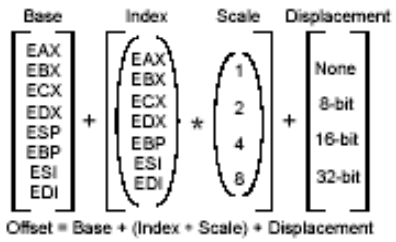
IA-32 – operazioni logico-aritmetiche

Tipo operando 1	Tipo operando 2	Tipo risultato
Registro	Registro	Registro
Registro	Immediato	Registro
Registro	Memoria	Registro
Memoria	Registro	Memoria
Memoria	Memoria	Memoria

- Uno dei registri (memoria) deve fungere sia da operando che da registro destinazione (architettura ad accumulatore) E.g.: add eax, [ebp + 8] ➔ eax + [ebp + 8] -> eax.
  - MIPS permette di avere operandi e risultato in registri differenti
- Uno od entrambi gli operandi può provenire direttamente dalla memoria
  - Nel MIPS o nel PowerPC solo dai registri
- Gli operandi Immediati possono arrivare a 32 bit, gli altri ad 80 bit

Modalità di indirizzamento – dati

Modo	Descrizione	Restrizioni sui registri	Codice MIPS	Codice INTEL
Ind. diretto (registro) Register Addressing	Indirizzamento tramite registro (l'operando è in un registro)	No ESP e EBP	move $s0, $s1	move eax, ebx
Register Indirect	Base register	No ESP e EBP	lw $s0, 0($s1)	move eax, [ebx]
Base + offset (8 - 32bit)	Base + offset addressing (INTEL – displacement)	No ESP e EBP	lw $s0, 100($s1)	move eax, 100 ([ebx] è sottinteso)
Base + scale*offset	Ind = base + offset*scale	No ESP e EBP	…	move eax, [esi*4] ([ebx] è sottinteso)
Base + scale*offset + displacement	Scaling Factor displacement	No ESP e EBP	…	move eax, [esi*4 + 2] ([ebx] è sottinteso)



Esempio, caricamento in EAX (sottointeso) dell'i-esimo elemento di un vettore:

mov EBX, 0x0x224H
mov ESI, i
*mov EAX , [ESI*4]*

Modalità di indirizzamento – istruzioni "immediate"

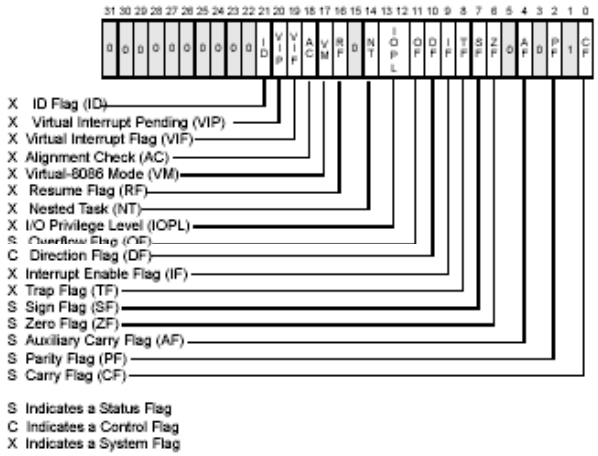
Modo	Descrizione	Codice MIPS	Codice INTEL
Indirizzamento immediato	L'operando è in una parte dell'istruzione	li $s0, 0x6a02H	move eax,0x6a02H
PC_relative addressing	Indirizzamento relativo al Program Counter	bne $s0, $s1, label	sub eax jnz label
Psudodirect addressing	**MIPS**: indirizzo ottenuto cambiando i 26 bit dell'istruzione con i 28 LSB di PC (i 2 LSB sono 00) **INTEL**: modifica del registro Code Segment	j label	move cs, 0x87ee move ip 0x0000 = jump label

Registro EFLAG

- I risultati notevoli vengono salvati in questo registro
  - Carry, zero, overflow, segno, parità, …
  - Le istruzioni di branch si riferiscono sempre a EFLAG

MIPS:
beq $s0, $s1, label

IA-32:
sub eax
jz label




Sommario

Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni


Operazioni di I/O

- IA-32 prevede istruzioni dedicate per in/out
  - Dati presenti nel registro accumulatore (EAX).
  - Distinzione tra memoria ed input/output mediante il segnale di controllo **M/IO**
- Le periferiche sono viste mediante le porte di I/O.
- Spazio di indirizzamento su 16 bit
  - 64k porte da 1 byte (32k porte da 2 byte; 16k porte da 4 byte)
- Lettura/scrittura avviene verso il device controller (DR), mediante i segnali di controllo di **R/W**
- Spazio di indirizzamento duplicato: 16 bit (+4 bit di offset) per la RAM e 16 bit per le periferiche nell'8086.

IA-32 Instruction Set

- Istruzioni general purpose
  - Istruzioni per lo spostamento dei dati
    - push, pop, utilizzo dello stack e della memoria dati
  - Istruzioni aritmetiche logiche, confronto e operazioni
  - Istruzioni di controllo del flusso
    - basati sui flag, allineamento al byte
  - Istruzioni della gestione delle stringhe
  - Istruzioni di I/O
- Istruzioni floating point x87
  - funzioni trigonometriche, potenze di 2 (2^x, $\log_2 x$)
- Istruzioni SIMD
  - MMX, SSE (SSE2, SSE3), 3DNow! (IA-32 by AMD)
- Istruzioni di sistema
  - Cambio di modo, Halt, Reset, …

IA-32 Instruction Set

Istruzione	Significato
Controllo	**Salti condizionati e incondizionati**
JNZ, JZ	Salto condizionato a EIP+offset a 8bit; nomi alternativi sono JNE (per JNZ) e JE (per JZ)
JMP	Salto incondizionato; offset a 8 o a 16 bit
CALL	Chiamata a procedura, con offset a 16 bit; l'indirizzo di ritorno è memorizzato nello stack
RET	Prende dallo stack l'indirizzo di ritorno ed esegue il salto
LOOP	Ciclo: decrementa ECX e salta a EIP+offset a 8bit se ECX è diverso da 0
Trasferimento dati	**Spostamento di dati fra registri o fra registri e memoria**
MOV	Sposta un dato da un registro ad un altro o fra registro e memoria
PUSH, POP	Mette nello stack l'operando sorgente; recupera dallo stack un operando e lo mette in un registro
LES	Carica dalla memoria ES ed uno dei GPR
Aritmetiche e logiche	**Operazioni aritmetiche e logiche su dati nei registri o in memoria**
ADD, SUB	Somma il sorgente alla destinazione, sottrae il sorgente alla destinazione; formato registro-memoria
CMP	Confronta il sorgente con la destinazione; formato registro-memoria
SHL, SHR, RCR	Scalamento a sinistra, scalamento a destra, rotazione verso destra con inserimento del flag di carry
CBW	Converte il byte che si trova negli 8 bit meno significativi di EAX in una parola che occupa i 16 bit meno significativi di EAX
TEST	Mette i valori opportuni nei flag facendo l'AND logico fra sorgente e destinazione
INC, DEC	Incrementa la destinazione, decrementa la destinazione; formato registro-memoria
OR, XOR	OR logico, OR esclusivo; formato registro-memoria
Stringhe	**Trasferimenti fra operandi stringhe; lunghezza data da un prefisso di ripetizione**
MOVS	Copia una stringa sorgente in una stringa destinazione incrementando ESI ed EDI; può essere ripetuta
LODS	Carica nel registro EAX un byte, una parola o una double word appartenente ad una stringa


IA-32 Instruction Set: esempi

Istruzione	Funzione
JE nome	if equal (condition code) {EIP=nome}; EIP-128 <= nome < EIP+128
JMP nome	EIP=nome;
CALL nome	SP=SP-4; M[SP]=EIP+5; EIP=nome
MOVW EBX, [EDI+45]	EBX=M[EDI+45]
PUSH ESI	SP=SP-4; M[SP]=ESI
POP EDI	EDI=M[SP]; SP=SP+4
ADD EAX, #6765	EAX=EAX+6765
TEST EDX, #42	Setta i flag con il risultato dell'and fra EDX e 42_{esa}
MOVSL	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

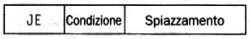
JE: jump equal - near (±128 byte)
JMP: jump (near → uso CS; far → uso EIP)
CALL: jump; SP=SP–4 (MIPS: `jal`)
MOVW: (MIPS: `lw`)
PUSH,POP: aggiornamento implicito SP
TEST: Carica nei flag i risultati di $EDX AND 0x42
MOVSL: Sposta 4 byte e incrementa EDI ed ESI (stringhe/aree dati)


Sommario

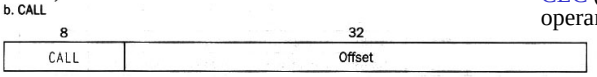
Le architetture Intel
I registri ed il loro utilizzo
L'ISA degli Intel
La codifica delle istruzioni

Codifica delle istruzioni

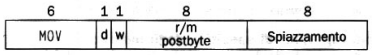
a. JE EIP + spiazzamento



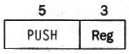
b. CALL



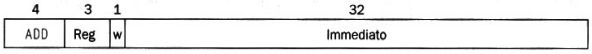
c. MOV EBX, [EDI + 45]



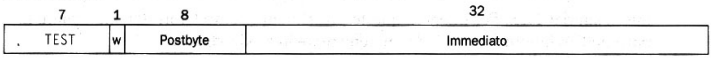
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42

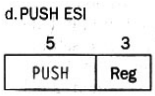


- Molti formati: ampiezza 1-15 byte.
  - Codice operativo su 1 o 2 byte.
  - CLC (Clear Carry:1 byte, non ha operandi)
  - **w** specifica se lavora sul byte o sulla parola a 32 bit (word)
  - **d** specifica la direzione del trasferimento
  - **Post_byte** r/m, specifica la modalità di indirizzamento

Codifica istruzioni – campi reg e w

- Campo reg (cf. numero registro Register File in MIPS):
  - la sua interpretazione dipende da w:
  - w=0 → registri a 8 bit
  - w=1 → 16 o 32 bit (dipende dall'architettura
    - IA-32: 32 bit
- w determina la lunghezza dell'istruzione
  - ADD AL, #1 → 16 bit
  - ADD EAX, #1 → 40 bit

d. PUSH ESI



4	3	1	8
ADD	000	0	0000 0001

4	3	1	32
ADD	000	1	00 … 01

campo reg	w=0	w=1	
	8 bit	16 bit	32 bit
0	AL	AX	EAX
1	CL	CX	ECX
2	DL	DX	EDX
3	BL	BX	EBX
4	AH	SP	ESP
5	CH	BP	EBP
6	DH	SI	ESI
7	BH	DI	EDI

Codifica istruzioni – campi r/m e mod

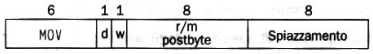
reg	w = 0	w = 1		r/m	mod = 0		mod = 1 mod = 2				mod = 3
	8b	16b	32b		16b	32b	16b	32b	16b	32b	
0	AL	AX	EAX	0	ind=BX+SI	=EAX	*stesso ind di mod=0 + disp8*	*stesso ind di mod=0 + disp8*	*stesso ind di mod=0 + disp16*	*stesso ind di mod=0 + disp32*	*come il campo reg*
1	CL	CX	ECX	1	ind=BX+DI	=ECX					
2	DL	DX	EDX	2	ind=BP+SI	=EDX					
3	BL	BX	EBX	3	ind=BP+SI	=EBX					
4	AH	SP	ESP	4	ind=SI	=(*sib*)	SI+disp8	(*sib*)+disp8	SI+disp8	(*sib*)+disp32	"
5	CH	BP	EBP	5	ind=DI	=disp32	DI+disp8	EBP+disp8	DI+disp16	EBP+disp32	"
6	DH	SI	ESI	6	ind=disp16	=ESI	BP+disp8	ESI+disp8	BP+disp16	ESI+disp32	"
7	BH	DI	EDI	7	ind=BX	=EDI	BX+disp8	EDI+disp8	BX+disp16	EDI+disp32	"

- **r/m (3 bit)** seleziona il registro usato come **registro base**
- **mod (2 bit)** seleziona la **modalità di indirizzamento** (+ offset, offset+displacement,...)
  - **mod = 0** → Indirizzo = Registro Base (IA-32) o Reg + Segment Reg (16 bit)
  - **mod = 1** → Indirizzo = Registro Base + displacement 8 bit
  - **mod = 2** → Indirizzo = Registro Base + displacement 16/32 bit
  - **mod = 3** → Indirizzo = Registro Base selezionato dal campo reg

  Eccezioni
  - **r/m = 4; mod=0,1,2** → Seleziona la modalità "**scaled index**"
  - **r/m = 5; mod=1,2** → Seleziona EBP + spiazzamento (32 bit)
  - **r/m = 6; mod=1,2** → Seleziona BP + spiazzamento (16 bit)

Esempi

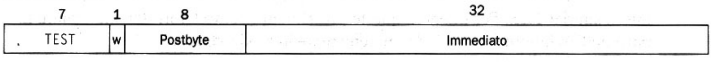
c. MOV EBX, [EDI + 45]



7 45

d = 1
w = 1 **[EBP] + EDI + 45**
r/m = 5
mod = 1
EBP sottointeso nella codifica binaria

f. TEST EDX, #42



w = 1
r/m = 5
mod = 2
EBP sottointeso nella codifica binaria

Codifica istruzioni: osservazioni

- Architettura CISC
  - Lunghezza variabile istruzioni
  - Lunghezza variabile OpCode
- La lunghezza dell'istruzione dipende dal contenuto di alcuni campi dell'istruzione stessa
  - mod, r/m, reg, w
  - Devo iniziare la decodifica dell'istruzione per sapere quant'è lunga → prima di terminare la fase di fetch
- Architettura complessa e poco razionale
  - prezzo da pagare per mantenere la compatibilità verso il basso (8, 16, 32 bit)

Sommario

Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni?