Lezione 33

L'architettura Intel

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Le prime architetture Intel

- **1978 – 8086**
  - Estensione del micro-processore 8080 utilizzato per applicazioni industriali.
  - Stessa ISA, ma architettura a 16 bit (registri a 16 bit)
  - Parte dei registri è dedicata a compiti specifici
  - Limite nello spazio di indirizzamento: **1 Mbyte (20-bit address bus)**
- **1980 – 8087**
  - Coprocessore matematico per 8086.
  - Dedicato a velocizzare le operazioni in virgola mobile.
  - Estensione degli operandi a 10 byte (80bit): *Extended Double Precision*
  - Modifica nel modo di gestire gli operandi, prelevabili dallo stack o dai registri
    - **Push <operando_1> in stack** (esteso a 10 byte)
    - **Push <operando_2> in stack** (esteso a 10 byte)
    - **Comando Operazione**
    - **Pop <risultato>**
- **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.

Le architetture Intel avanzate

- **1985 – 80386**: Architettura **IA-32** "full 32–bit" (dati, indirizzi e registri a 32 bit)
  - Nuove istruzioni, vicino ad un calcolatore con general-purpose registers.
  - Pre-fetching. Paginazione della RAM.
- **1989 – 80486 = 80386+80387**: Architettura pipe-line (singola).
  - Istruzioni per la gestione delle architetture multi-processore.
  - Memoria cache. Microprogrammazione per l'Unità di controllo (FSM).
- **1992 – Pentium, PentiumPro**: Pipe-line multipla (arch. super-scalare).
  - Tecnologia MMX (Multi-media extension, SIMD).
  - Cache primaria e secondaria (separata, con bus dedicato)
- **1997 – Pentium II**: Memorie cache a doppio accesso. Cache dei registri di segmento. PentiumPro + MMX.
- **1999 – Pentium III**: "Internet Streaming Single Instruction Multiple Data extensions (SSE)
  - Estensione dell'architettura MMX ad istruzioni floating-point. L2 cache integrata nella CPU
- **2001 – Pentium 4**: Estensione del parallelismo e della superscalarità
- **2003 – Pentium M, Core**: Ottimizzazione per basso consumo
- **2006 – Core 2**: Architettura **multi-core**
- **2008 – Atom**: **Very low-power** (2÷8 W), ottimizzato per NetBooks

Architettura x86

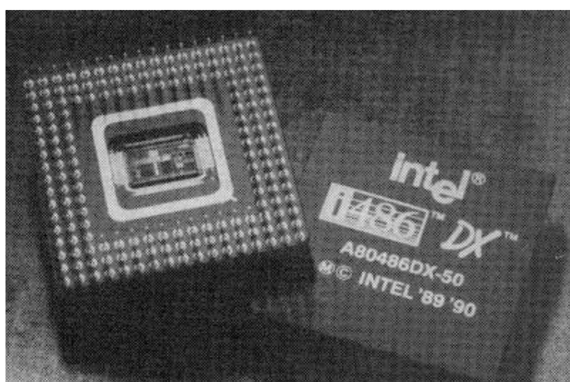
x86: Architettura CISC

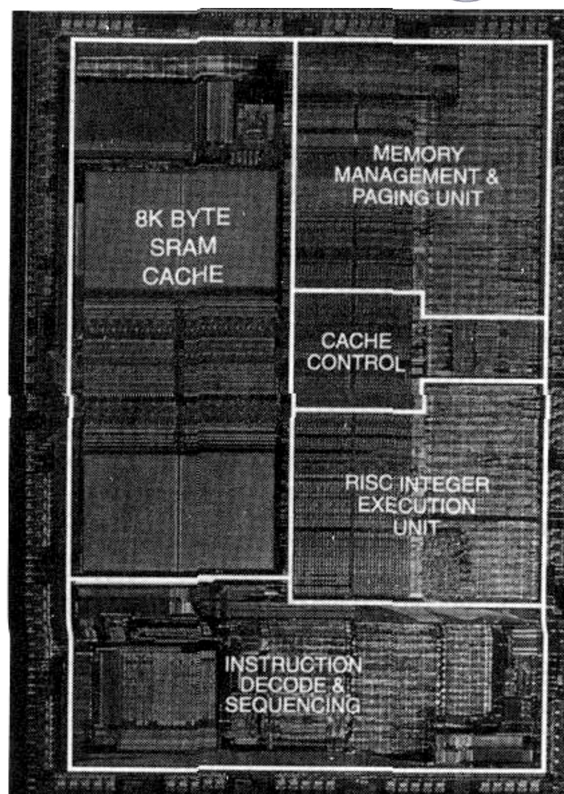
- Lunghezza variabile istruzioni: **1 – 17 bytes**
- Operazioni **aritmetico/logiche direttamente in memoria**
- Architettura condizionata dalla storia → necessità di compatibilità verso il basso
  - Real mode / Protected mode / Virtual 8086 Mode
- Registri: "not-general"-purpose register
  - ogni registro è progettato per un uso specifico
  - dal 80386 in poi (IA-32) si definiscono 8 GP registers, ma comunque non veri General Purpose Registers come in MIPS
- Gestione di memoria ed I/O tramite 3 segnali di controllo: **RD, WR, IO/MEM**

  Es: Memory read: IO/MEM = 0 ; RD = 1 ; WR = 0

CPU IA-32: 80386, 80486

- **80386**: prima architettura IA-32 (full 32 bit)
- **80486**: integrazione nel chip di:
  - **CPU (80386) + Coprocessore** matematico (80387)
  - **Cache interna e controller**
  - Gestione memoria (**MMU**)



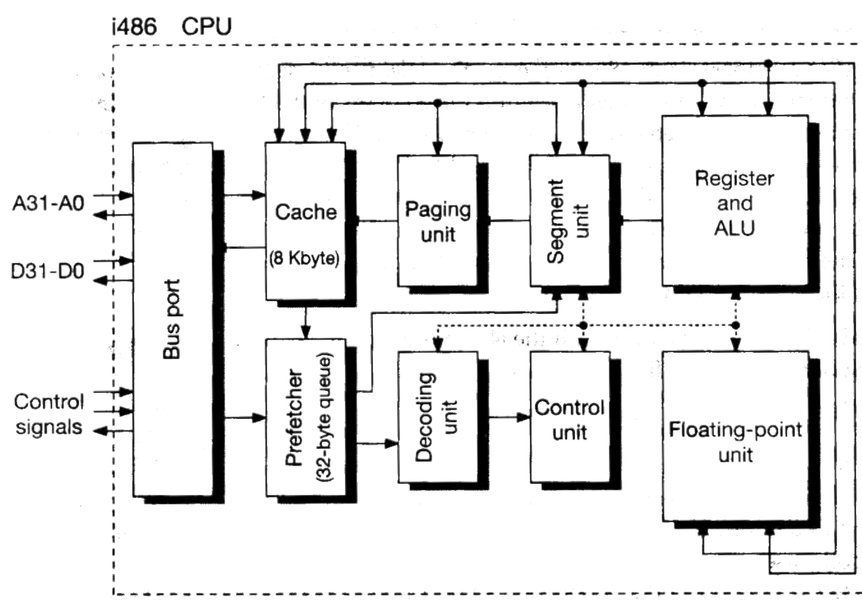


Intel 80486 – Struttura interna

- Interfacciamento con il resto della macchina tramite la porta verso il bus (bus port).
  - Sul bus vengono inviati i dati, gli indirizzi ed i segnali di controllo.

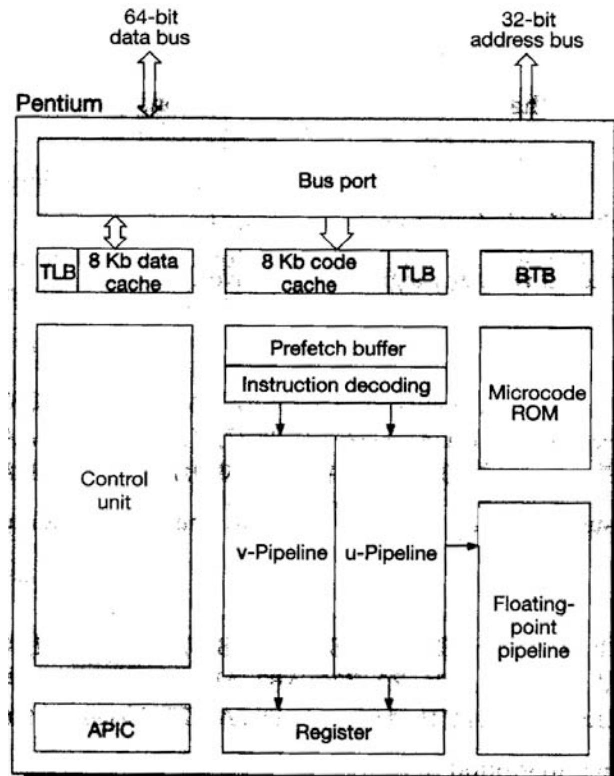
Caratteristiche:

- Pipeline a 5 stadi
- Bus esterni a 32 bit
- Bus interno a 64 bit
- Cache di 8 kbyte (modalità: write-through)
- Control unit micro-programmata



Pentium – struttura interna

- Architettura 32/64 bit
  - Bus dati: **64 bit**
  - Bus indirizzi / registri: **32 bit**
- 2 pre-fetch queues:
  - pipelines **u, v**
  - pipeline **u** si interfaccia con la pipeline **floating point**.
- APIC:
  - Advanced Programmable Interrupt Controller
- Cache interne, collegate al bus del processore
  - TLB: Transition Lookaside Buffer: tabella di corrispondenza degli indirizzi fisici delle **pagine** più usate (*cache di traduzione*)



Sviluppi futuri: IA-64

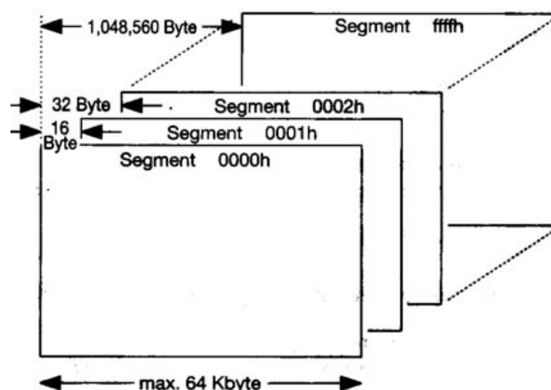
- Architettura a 64 bit
  - Progetto MERCED
  - Primo esemplare: **Intel ITANIUM**
- Massiccio aumento delle risorse
  - 128 registri GP, 128 registri FP
  - 16 GP-EU + 16 FP-EU (EU: Execution Unit)
- Esecuzione parallela di istruzioni, con parallelismo esplicito
  - si specifica a livello del codice Assembly se le istruzioni possono essere eseguite contemporaneamente o sequenzialmente
- Compatibilità con IA-32
- Macchina molto complessa → iter progettuale troppo difficoltoso
  - Progetto ITANIUM “venduto” alla Digital e abbandonato

Modi di funzionamento *IA-32*

- Modalità **reale** (Real mode)
  - Modalità **compatibile DOS (8086-compatibile)**
  - Max memoria indirizzabile: 1 MByte → 2^{20}
  - modo attivo all'accensione (power-on)
  - Nessun meccanismo di protezione della memoria
- 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 (paginazione)
- 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.

x86: gestione della memoria

- **Modalità reale** (16 + 4 address bit): interleaving dei segmenti:
  - Spazio di indirizzamento di 1 Mbyte suddiviso in 16 segmenti di 64 kbyte
  - Indirizzo = 16 * Segmento + Offset
  - Modalità Virtual 8086: Indirizzo e offset sono separati.



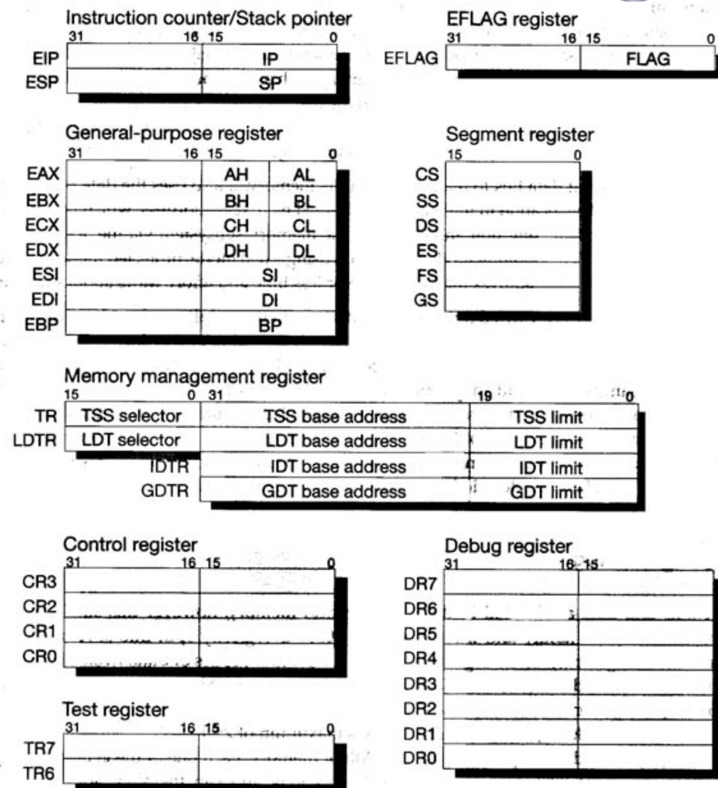
- **Modalità protetta** (32 address bit)
  - Registro **CS: Code Segment** (16 bit)
  - SALTO: **indirizzo = composizione registri CS (hi) e IP (lo)**
  - CS = **0x80B8**, IP = **0x019D** → indirizzo salto: **0x 80B8 019D**

I registri dell'architettura **IA-32**

- A partire dal 80386: **IA-32**

Caratteristiche:

- **8 registri "general-purpose" a 32 bit**
  - istruzioni per accedere ai primi 8 / 16 bit
  - compatibilità con 8086
  - non sono poi così "general purpose"…
- I registri di segmento sono rimasti a 16 bit
  - usati come registri base



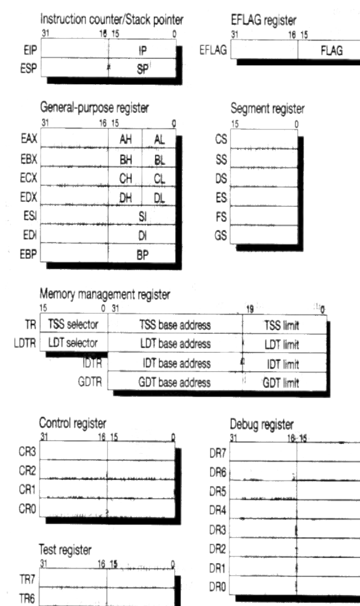
Registri IA-32

- **8 General Purpose Registers**

  4 General Data Registers (32 bit)
  - EAX: accumulatore (ottimizzato per op. aritmetico-logiche)
  - EBX: base register (registro base nel segmento dati)
  - ECX: counter (ottimizzato per i loops)
  - EDX: data register (GP)

  4 General Address Registers (32 bit)
  - EIP: instruction pointer
  - EBP: stack base pointer (base address dello stack)
  - ESP: stack pointer
  - ESI: source index (ottimizzato per op. su stringhe)
  - EDI: destination index (ottimizzato per op. su stringhe)

- **Segment Registers**
- **Floating-point Stack registers**
  - **ST(0) – ST(7):** 80 bit, accessibili come LIFO (stack)
- **SIMD Registers**
  - MMX, SSE, 3DNow!



IA-32: registri "General Data"

Nome simbolico			Nome descrittivo	Funzione
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.				

EBX	**BX**	**BH, BL**	**Base Register**	Puntatore all'indirizzo base segmento dati
`mov ecx, [ebx]` ; trasferisci in `ecx` il contenuto all'indirizzo 0(`$ebx`) – MIPS: `lw $ecx, 0($ebx)`				

ECX	**CX**	**CH, CL**	**Count Register**	Indice di conteggio (cicli, rotazioni, 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 0x70.
        loop start      ; ritorna ad inizio ciclo, che verrà ripetuto 16 volte (decr. ECX finché 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)**				

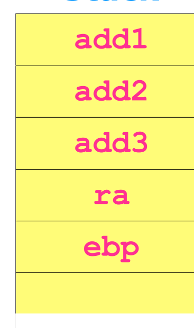
IA-32: registri di stack

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

Esempio di utilizzo dello stack per passare dati a funzione (Assembly IA-32):

```
    push add1       ; push the first summand (ESP decremented automatically)
    push add2       ; push the second summand
    push add3       ; create the third summand
    call addition
    ...

addition:
    proc near           ; dichiarazione "proc near" (inside 64k segment)
    push ebp            ; salva l'indirizzo base per il ritorno
    move ebp, esp       ; copia lo StackP nel BaseP (frame procedura)
    move eax, [ebp+16]  ; carica sum1 in EAX
    add eax, [ebp+12]   ; somma in eax sum1 + sum2
    add eax, [ebp+8]    ; somma in eax aum1 + sum2 + sum3
    pop ebp             ; recupera l'indirizzo base precedente
    ret                 ; ritorno al programma chiamante (cf. jr $ra)
```



IA-32: registri di gestione stringhe

Nome simbolico			Nome descrittivo	Funzioni
32 bit	16 bit	8 bit		
ESI	**SI**	---	**Source Index**	Indice per la stringa sorgente o indice di caratteri/array
EDI	**DI**	---	**Destination Index**	Indice per la stringa destinazione o indice di caratteri/array

Esempio: output della stringa: "abcdefghijabcdefghij" su monitor alfanumerico:

```
string: db 20 ('abcdefghijabcdefghij')   ; definizione della stringa

   mov eax, @data     ; carica indirizzo (di inizio) dei dati in EAX
   mov ds, eax        ; imposta DS a questo segmento dati

   cld                ; sequenza ascendente
   mov ecx, 5         ; trasferisce 5 parole di 4 byte ciascuna
   mov esi, string    ; carica indirizzo stringa in ESI (stringa sorgente)
   mov edi, video     ; carica indirizzo del primo carattere
                      ; in alto a sinistra in EDI (stringa destinazione)
   movsw              ; trasferisce 5 parole (20 caratteri)
```


IA-32: registri di Segmento

- **Segment Registers**
  - Registri a 16 bit
  - **Si sovrappongono alla parte alta** di altri registri a 32 bit

    Instruction Pointer: CS → [CS, IP] = EIP

    Stack Pointer: SS → [SS, SP] = ESP

Nome	Descrizione	Funzione
CS (EIP)	**Code Segment**	Contiene l'indirizzo base, per accesso ad istruzioni. Le istruzioni sono indirizzate tramite il registro **EIP** (Extended IP). Per modificare **CS** occorre una ***far call*** o una ***far jump*** oppure un interrupt. In protected mode viene verificato se il nuovo segmento può essere utilizzato.
DS (EBX)	**Data Segment**	Contiene l'indirizzo base del segmento dati del programma. Molte istruzioni quali la `mov` utilizzano questo segmento.
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 A/L 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.

Registro **EFLAG**

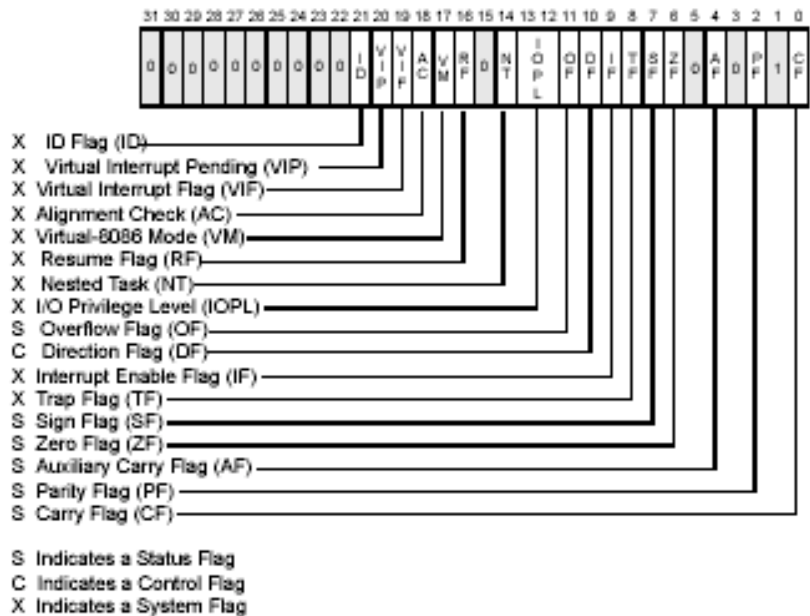
- I risultati notevoli vengono salvati in questo registro
  - Le istruzioni di branch si riferiscono sempre a EFLAG
  - **Carry, zero, overflow, segno, parità, ...**

MIPS:

```
lw $s0, 0($s1)
beq $s0, $zero, label
```

IA-32:

```
move eax, [ebx]
jz label
```



IA-32: **Instruction Set**

- Istruzioni **general purpose**
  - Istruzioni per lo **spostamento dei dati**
    - push, pop, utilizzo dello stack e della memoria dati
  - Istruzioni **aritmetico-logiche**, confronto e operazioni
  - Istruzioni di **controllo di flusso**
    - basati sui flag, allineamento al byte
  - Istruzioni della gestione delle **stringhe**
  - Istruzioni di **I/O**
- Istruzioni di **sistema**
  - Cambio di modo, Halt, Reset, …
- 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)

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

- Architettura ad **accumulatore**: uno dei registri (memoria) deve fungere sia da operando che da registro destinazione.
  - MIPS permette di avere operandi e risultato in registri differenti
- Uno od entrambi gli operandi può provenire direttamente dalla memoria
  - Nel MIPS: solo dai registri
- Gli operandi immediati possono arrivare a 32 bit, gli altri ad 80 bit

Formato istruzioni IA-32

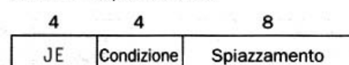
IA-32 Instruction Format:

opcode	Mod,R/M	SIB	spiazzamento	immediato
1,2 bytes	1 byte se richiesto	1 byte se richiesto	1, 2 o 4 byte se richiesto	1, 2 o 4 byte se richiesto

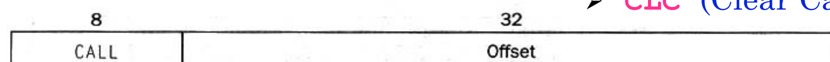
- **Opcode**: definisce il tipo di istruzione. In alcuni casi contiene i campi: **d** (*direction*, 1 bit) e **w** (*width*, 1 bit)
- **Mod R/M**: definisce il tipo di indirizzamento (a registro o a memoria). Composto dai campi: **reg** (3 bit), **Mod** (2 bit), **R/M** (3 bit)
- **SIB**: Scale-Index-Base (definisce il tipo di scansione di un'area di memoria)
- Esistono istruzioni speciali, con formato diverso

Codifica delle istruzioni

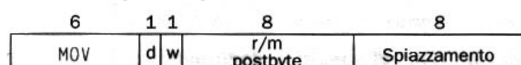
a. JE EIP + spiazzamento



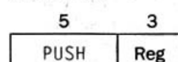
b. CALL



c. MOV EBX, [EDI + 45]

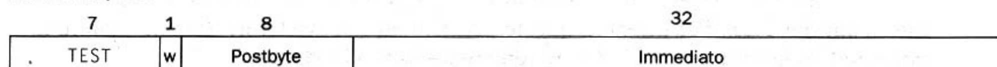


d. PUSH ESI



e. ADD EAX, #6765

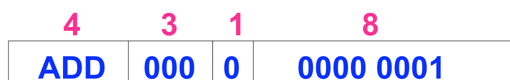
f. TEST EDX, #42

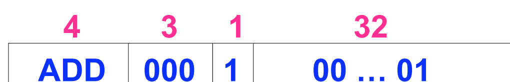


- Molti formati – **ampiezza: 1÷17 byte**
  - Codice operativo su 1 o 2 byte
  - `CLC` (Clear Carry: 1 byte, non ha operandi)
  - `mov EAX, ind1:[ind2 + ind3*4 + 2]` *richiede 17 byte*
  - **w** specifica se lavora sul **byte** o sulla parola di 32 bit (**word**)
  - **d** specifica la direzione del trasferimento
  - `Post_byte (r/m)` specifica la modalità di indirizzamento

Codifica istruzioni – campi **reg** e **w**

- Campo **reg:**
  - Definisce il registro interessato all'istruzione
  - la sua interpretazione dipende da **w**:
  - w=0 → registri a 8 bit
  - w=1 → registri a 16 o 32 bit
    - dipende dall'architettura
- **w** determina la lunghezza dell'istruzione
  - `ADD AL, 1` → 16 bit
  - `ADD EAX, 1` → 40 bit





OpCode

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
16b	32b	16b	32b	16b	32b	16b	32b				
0	AL	AX	EAX	0	ind=BX+SI	=EAX	*stesso ind di mod=0*	*stesso ind di mod=0*	*stesso ind di mod=0*	*stesso ind di mod=0*	*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+DI	=EBX	+disp8	+disp8	+disp16	+disp32	•
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**
  - r/m = 0 → Indirizzo = Registro Base (IA-32) o Reg + Segment Reg (16 bit)
  - r/m = 1 → Indirizzo = Registro Base + displacement 8 bit
  - r/m = 2 → Indirizzo = Registro Base + displacement 16/32 bit
  - r/m = 3 → Indirizzo = Registro Base selezionato dal campo **reg**
- **mod:** **(2 bit)** in combinazione con r/m, seleziona la **modalità di indirizzamento** (offset, offset+displ., …)
  - r/m = 4; mod=0,1,2 → Seleziona la modalità "**scaled index**" (necessario campo **SIB**)
  - r/m = 5; mod=1,2 → Seleziona EBP + spiazzamento (32 bit)
  - r/m = 6; mod=1,2 → Seleziona BP + spiazzamento (16 bit)

Modalità di indirizzamento

Istruzioni che contengono un campo immediate:

- A/L immediate, branch, jump, …

- Confronto MIPS vs. IA-32

Modo	Descrizione	Codice MIPS	Codice INTEL
Indirizzamento immediato	L'operando è parte dell'istruzione	`li $s0, 0x6a02H`	`move eax,0x6a02H`
Indirizzamento relativo al PC	Il dato immediato è sommato al Program Counter	`bne $s0,$s1,label`	`jnz 0x01A5`

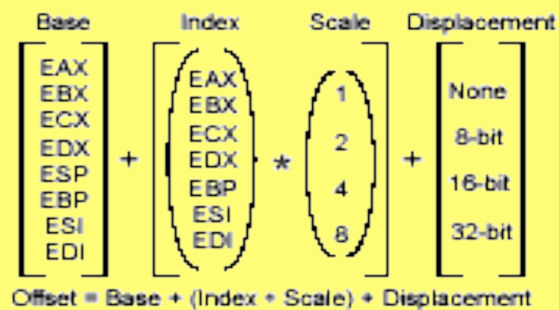
Indirizzamento Pseudodiretto	**MIPS**: indirizzo ottenuto cambiando i 26 bit dell'istruzione con i 28 LSB di PC (i 2 LSB sono 00)
	
	INTEL: modifica del registro **Code Segment**
		`j label`	`move cs, 0x87ae`
			
			`move eip 0x00000000`

Modalità di *indirizzamento* – dati

Modo	Descrizione	Restrizioni	Codice MIPS	Codice IA-32
Diretto (registro) Reg. Addressing	Indirizz. tramite registro (l'operando è in un registro)	No ESP e EBP	`add $s0,$s1,$0`	`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, array[100]`
Base + scale*offset	Ind = base + offset*scale	No ESP e EBP	…	`move eax, [esi*4]`
Base + scale*offset + displacement	Scaling Factor displacement	No ESP e EBP	…	`move eax, [esi*4 + 2]`

- **Campo SIB (3 bit: 0÷7)**

  definisce il modo in cui si accede alla memoria dati, a partire dall'indicazione di 2 registri + displacement



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 42
MOVSL: Sposta 4 byte e incrementa EDI ed ESI (stringhe/aree dati)

Operazioni di **I/O**

- IA-32 prevede **istruzioni dedicate per I/O**:
- Dati presenti nel registro accumulatore (**EAX**).
- Distinzione tra accesso a memoria ed input/output mediante il segnale di controllo: **M/IO**
- Per lettura/scrittura (in memoria o I/O) si utilizzano i segnali di controllo: **RD, WR**
- Spazio di indirizzamento su **16 bit**
  - 64k porte da 1 byte (32k porte da 2 byte; 16k porte da 4 byte)
  - **Spazio di indirizzamento duplicato**:
    16 bit (+4/+32 bit) per la RAM e 16 bit per le periferiche nell'8086.
- Le periferiche sono viste mediante le **porte di I/O**
  - 8086: I/O : 2^{16} porte di I/O
  - 8086: MEM : $2^{16(+4)} = 2^{20}$ celle di memoria (words)

Codifica istruzioni: osservazioni

- Architettura CISC
  - **Lunghezza variabile** sia delle **istruzioni** che dell' **OpCode**
  - La lunghezza dell'istruzione dipende dal contenuto di alcuni campi
  - Devo iniziare la decodifica dell'istruzione per sapere quant'è lunga → prima di terminare la fase di fetch
  - **Pre-fetch Queue** (Coda di pre-fetch). Streaming dal segmento codice di RAM in un buffer fino al riempimento.

- Architettura complessa
  - Fino a 300 cicli di clock per i task più complessi quali il task switch tramite gate che viene operata in modalità protetta.
  - Conseguenza anche della volontà di mantenere la compatibilità verso il basso (8, 16, 32 bit)