Architettura degli Elaboratori e delle Reti

L'ISA di 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

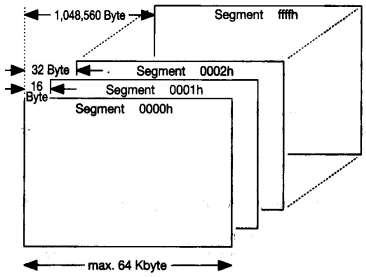
Le prime architetture Intel

- **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 (virgola mobile sempre)** oppure, uno di loro, dai registri.
  - 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>da stack

- 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**

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

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. Cache primaria e secondaria (separata, con bus dedicato). Introduzione dell'esecuzione condizionata. ISA quasi identico. Ampia ristrutturazione interna.
- **1997 – Pentium II**: Memorie cache a doppio accesso per ciclo di clock. Cache dei registri di segmento. PentiumPro + MMX (Multi-media extension, SIMD su dati piccoli: byte).
- **1999 – Pentium III**: "Internet Streaming single instruction multiple data extensions (SSE). Estensione dell'architettura MMX ad istruzioni floating-point. L2 cache e CPU nello stesso integrato. 8 registri da 128 bit per supporto ad esecuzione parallela (multiple-issue).

Le architetture più recenti

- **2001 – Pentium 4**: Estensione del parallelismo e della Superscalarità. *Hyper-Threading Technology*. SSE2. I registri a 128 bit supportano anche virgola mobile in doppia precisione. Register renaming e gestione separate delle code di esecuzione. Massimizzazione delle prestazioni.
- **2003 – AMD**: Architettura a 64 bit. Aumento dello spazio di indirizzamento a 64 bit e registri a 64 bit. 16 registri GP + 16 registri a 128 bit.
- **2004 – EM64T**: (Extended Memory 64 Techniology). SSE3 che supporta numeri complessi e strutture. Operazione atomica su blocchi di memoria di 128 bit.
- 2006 – Virtual machines support: SSE4 con operazioni quali somma di differenze in modulo, conteggio elementi, ... Supporto alle macchine virtuali.
- **2007 – AMD.** Istruzioni su 3 operandi.
- **2007 – Intel.** Advanced Vector Extension. Registri da 128 a 256 bit.

Architettura x86

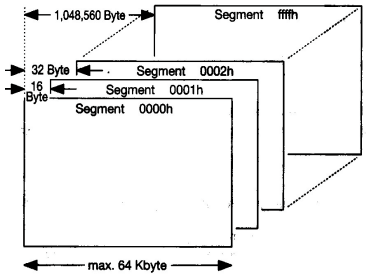
- CISC
  - Lunghezza istruzioni: 1 – 15 bytes
  - Operazioni direttamente in memoria
- Architettura condizionata dalla storia → necessità di compatibilità verso il basso
  - Real mode/Protected mode/Virtual 8086 Mode
- Nasce come architettura "not" general-purpose register
  - ogni registro è progettato 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

64 bits architectures

Operating mode	Operating sub-mode	Operating system required	Type of code being run	Default address size	Default operand size	Supported typical operand sizes	Register file	Typical GPR width
Long mode	64-bit mode	64-bit operating system or bootloader	64-bit code	64 bits	32 bits	8, 16, 32, or 64 bits	16 registers per file	64 bits
	Compatibility mode	64-bit operating system or bootloader	32-bit protected mode code	32 bits	32 bits	8, 16, or 32 bits	8 registers per file	32 bits
		64-bit operating system or bootloader	16-bit protected mode code	16 bits	16 bits	8, 16, or 32 bits	8 registers per file	32 bits
Legacy mode	Protected mode	32-bit operating system or bootloader, or 64-bit bootloader	32-bit protected mode code	32 bits	32 bits	8, 16, or 32 bits	8 registers per file	32 bits
		16-bit or 32-bit operating system or bootloader, or 64-bit bootloader	16-bit protected mode code	16 bits	16 bits	8, 16, or 32 bits	8 registers per file	16 or 32 bits
	Virtual 8086 mode	16-bit or 32-bit operating system	16-bit real mode code	16 bits	16 bits	8, 16, or 32 bits	8 registers per file	16 or 32 bits
	Real mode	16-bit or 32-bit operating system or bootloader, or 64-bit bootloader	16-bit real mode code	16 bits	16 bits	8, 16, or 32 bits	8 registers per file	16 or 32 bits

From wikipedia

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

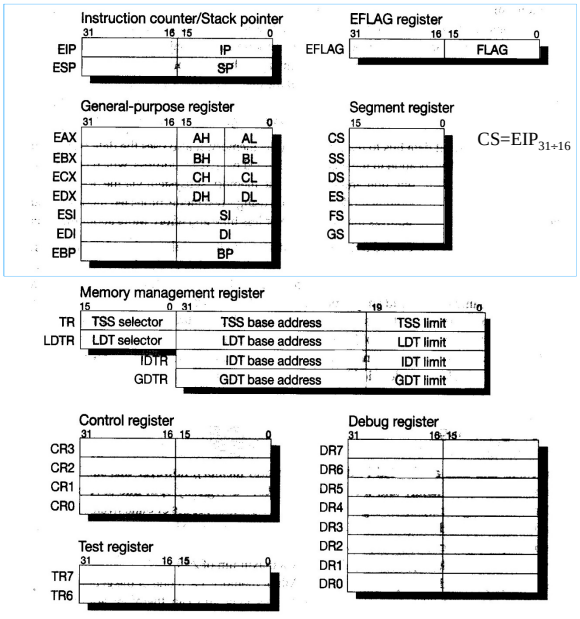
IP
SP

Segment Register

CS	
SS	
DS	
ES	
FS	
GS	

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).



Registri di Segmento (di memoria)

Nome	Descrizione	Utilizzo
CS || IP (EIP)	Code Segment	Puntatore al segmento di codice. 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	Puntatore al segmento dati. 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 || SP (ESP)	Stack Segment	Puntatore al segmento di stack. 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	Puntatori addizionali al segmento dati. 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)

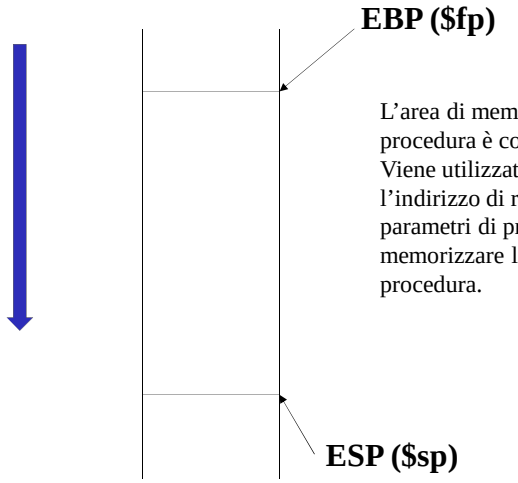
Utilizzo dei registri IA-32

- General Purpose Registers
  - General Data Registers
    - **EAX**: accumulatore (ottimizzato per op. aritmetico-logiche)
    - **EBX**: base register (registro base nel segmento dati, indirizzo corrente, $gp)
    - **ECX**: counter (ottimizzato per i loops)
    - **EDX**: data register (contiene i dati)
  - General Address Registers
    - **EBP**: stack base pointer (base address dello stack, $fp)
    - **ESI**: source index (ottimizzato per op. su stringhe)
    - **EDI**: destination index (ottimizzato per op. su stringhe)
- Program counter e stack pointer
  - **EIP**: extended instruction pointer (EIP = Program Counter = SS || IP)
  - **ESP**: stack pointer ($sp) (SS || SP)
- Floating-point Stack registers
  - ST(0) – ST(7): 80 bit, accessibili come LIFO (stack)
- Debug registers
  - DR(0) – DR(7): 32 bit, debugging
- SIMD Registers
  - MMX, SSE (SSE1, SSE2, SSE3, SSE4, SSE5)
- Coordinamento tra: EBX e DS; tra EIP e SS; tra EBP, ESP e SS.

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 edx, [ebx]` ;trasferisci quanto presente all’indirizzo 0($ebx) in edx `lw $s0, 0($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

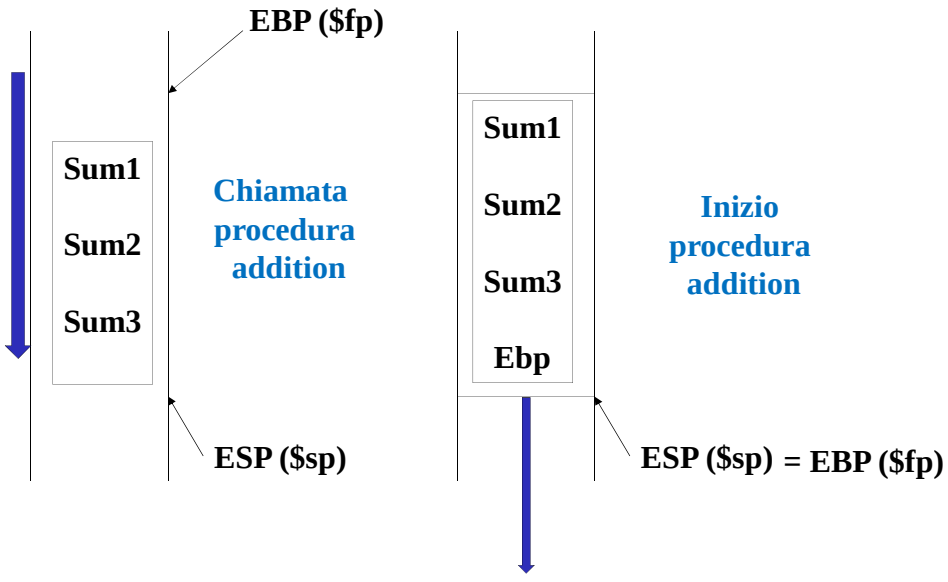
Nome simbolico			Nome descrittivo	Funzioni
32 bit	16 bit	8 bit		
ESP	SP	x,x	Stack Pointer	Stack Pointer
EBP	BP,SS	x, x	Base Pointer	Indirizzo base del segmento di Stack (32 bit)

Architettura ad accumulatore

```
   push sum1    ; create the first summand
                ; addi $sp, $sp, -4; lw $s0, 0($sp)
                ; (automaticamente ESP viene decrementato)
   push sum2            ; create the second summand
   push sum3            ; create the third summand
   call addition        ; chiama procedura addition

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 (esp) 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
```


Lo stack viene delimitato



IA-32 – operazioni logico-aritmetiche e confronto con il MIPS

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

Dati di tipo diverso

- Possibilità di indirizzare dati da 8, 16, 32 (e 64 nell'estensione a 64 bit) bit.
- INTEL definisce un'ampiezza di default dei dati (16, 32 o 64 bit) .
- L'ampiezza di default viene impostata e modificata attraverso un **prefisso di 8 bit** inserito prima dell'istruzione.

Ruolo del prefisso (1 byte):

- modificare i registri di segmento
- impedire l'accesso al bus
- ripetere l'istruzione (fino a quando ECX = 0, pensato per la move di stringhe o blocchi)
- dimensione di default dei dati e degli indirizzi.

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 dal setting dei dati di default)
    - IA-32: 32 bit di default
- **w** determina il tipo di dato (short o long: 8 o 16/32 bit).

→ istruzioni di lunghezza diversa

  - ADD AL, #1 → 16 bit
  - ADD EAX, #1 → 40 bit

4	3	1	8
ADD	000	0	0000 0001

4	3	1	32
ADD	000	1	00 … 01

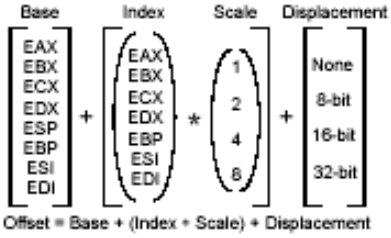
d. PUSH ESI

5	3
PUSH	Reg

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

Modalità di indirizzamento – dati

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



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

```
mov EBX, 0x0x224H   ; indirizzo base
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, 0x6a02	mov eax,0x6a02
PC_relative addressing	Indirizzamento relativo al Program Counter ([ebx] sottointeso)	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	mov cs, 0x87ee mov ip 0x14fa = jump label = mov eip 0x87ee14fa

Non c’è più il limite dell’allineamento ai 32 bit della parola

Sommario

Le architetture Intel

I registri ed il loro utilizzo

L’ISA degli Intel

La codifica delle istruzioni

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, …

Le operazioni IA-32

- Spostamento di dati: move, push, pop
- Aritmetico-logiche su interi e float
- Controllo di flusso: salti condizionati ed incondizionati, chiamata e ritorno da procedura.
- Operazioni su stringhe

I salti condizionati si basano sul controllo del risultato dell'operazione: **condition codes** o **flag**.

NB: l'offset rispetto al PC viene espresso in byte; non può essere espresso in word!!

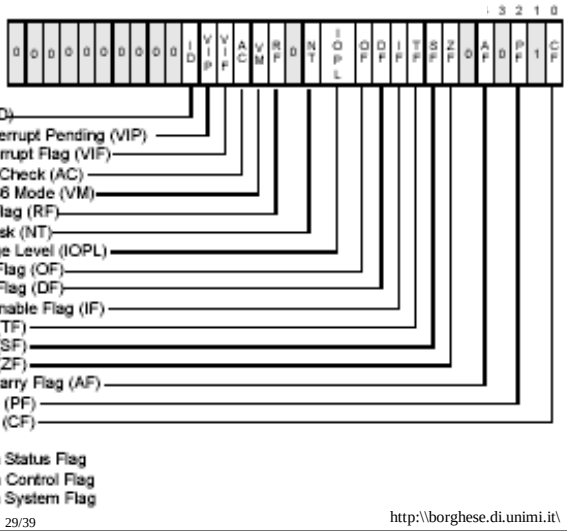
Le operazioni su stringhe sono un'eredità (legacy) dell'8080.

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



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. L'indirizzo di base viene memorizzato nel CS (cf. MIPS).


IA-32 Instruction Set - I

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
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	Set condition code (flags) with EDX and 42
movsl	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4


IA-32 Instruction Set - I

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
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	Set condition code (flags) with EDX and 42
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)

IA-32 Instruction Set: esempi

Instruction	Meaning
Control	**Conditional and unconditional branches**
jnz, jz	Jump if condition to EIP + 8-bit offset; JNE (for JNZ), JE (for JZ) are alternative names
jmp	Unconditional jump—8-bit or 16-bit offset
call	Subroutine call—16-bit offset; return address pushed onto stack
ret	Pops return address from stack and jumps to it
loop	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠ 0
Data transfer	**Move data between registers or between register and memory**
move	Move between two registers or between register and memory
push, pop	Push source operand on stack; pop operand from stack top to a register
les	Load ES and one of the GPRs from memory
Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
add, sub	Add source to destination; subtract source from destination; register-memory format
cmp	Compare source and destination; register-memory format
shl, shr, rcr	Shift left; shift logical right; rotate right with carry condition code as fill
cbw	Convert byte in eight rightmost bits of EAX to 16-bit word in right of EAX
test	Logical AND of source and destination sets condition codes
inc, dec	Increment destination, decrement destination
or, xor	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
movs	Copies from string source to destination by incrementing ESI and EDI; may be repeated
lods	Loads a byte, word, or doubleword of a string into the EAX register

Sommario

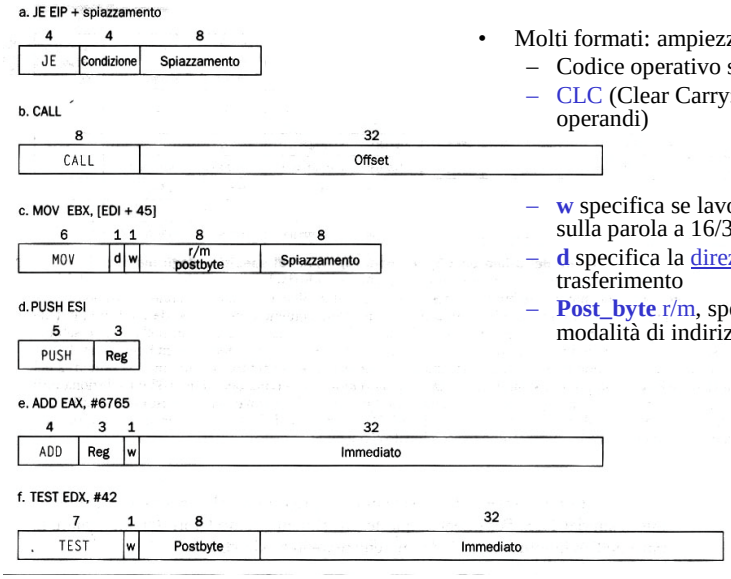
Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni

Codifica delle istruzioni



- 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 16/32 bit (word)
  - **d** specifica la direzione del trasferimento
  - **Post_byte** r/m, specifica la modalità di indirizzamento

Codifica istruzioni – campi r/m e mod

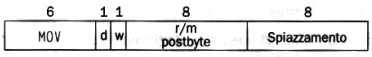
reg	w = 0	w = 1		r/m	mod = 0		mod = 1		mod = 2		mod = 3
		16b	32b		16b	32b	16b	32b	16b	32b	
0	AL	AX	EAX	0	addr=BX+SI	=EAX	*same*	*same*	*same*	*same*	*same*
1	CL	CX	ECX	1	addr=BX+DI	=ECX	*addr as*	*addr as*	*addr as*	*addr as*	*as*
2	DL	DX	EDX	2	addr=BP+SI	=EDX	*mod=0*	*mod=0*	*mod=0*	*mod=0*	*reg*
3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

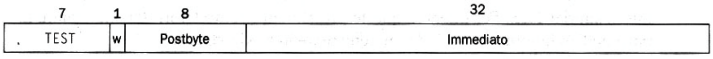
c. MOV EBX, [EDI + 45]



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?