



## Lezione 34

# L'architettura Intel

*Proff. A. Borghese, F. Pedersini*

Dipartimento di Scienze dell'Informazione  
Università degli Studi di Milano

## Sommario



### Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni

Le pipeline

## 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. Architettura, 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

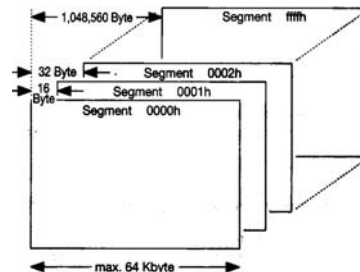


- ❖ **Modalità reale (Real mode)**
  - Modalità compatibile DOS
  - Max memoria indirizzabile: 1 MByte →  $2^{20}$
  - modo attivo all'accensione (power-on)
- ❖ **Modalità protetta (Protected mode)**
  - modalità “nativa” di IA-32
  - Memoria indirizzabile: 4 GByte →  $2^{32}$
  - Memoria protetta: evita 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 protetta



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



### ❖ Modalità reale

- $\text{Indirizzo} = 16 * \text{Segmento} + \text{Offset}$
- In tal modo posso indirizzare  $64\text{KB} \times 16 = 1 \text{ MByte}$  di MP
- Modalità **Virtual 8086**: Indirizzo e offset sono separati.

### ❖ Modalità protetta

- Spazio di indirizzamento: 32 bit
  - SALTO: indirizzo = composizione registri CS e IP
    - CS = 0x80B8, IP = 0x019D
    - indirizzo istruz. successiva: **0x 80B8 019D**

## Sommario



Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

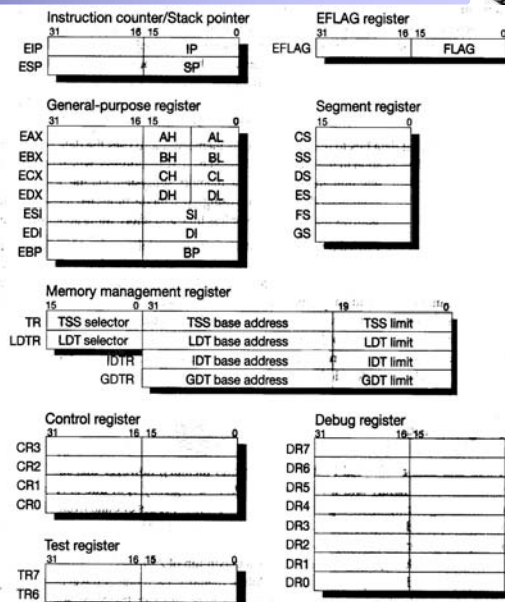
La codifica delle istruzioni

Le pipeline

## I registri dell'architettura IA-32



- ❖ Così a partire dal 80386 (IA-32)
- ❖ 8 registri “general-purpose” a 32 bit
  - 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



A.A. 2004/05

Copyright: A. Borghese, F. Pedersini - DSI, UniMI

9/44

## 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)
    - ECX: counter (ottimizzato per i loops)
    - EDX: data register (GP)
  - General Address Registers
    - 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)
- ❖ Floating-point Stack registers
  - ST(0) – ST(7): 80 bit, accessibili come LIFO (stack)
- ❖ SIMD Registers
  - MMX, SSE, 3DNow!

A.A. 2004/05

Copyright: A. Borghese, F. Pedersini - DSI, UniMI

10/44

## Registri di Segmento



Nome	Descrizione	Utilizzo
<b>CS</b>	Code Segment	Contiene l'indirizzo base dei (dati) ed istruzioni ad accesso immediato. Le istruzioni del segmento sono indirizzate tramite il registro <b>EIP</b> (Extended Instruction Pointer). Per modificare <b>CS</b> occorre una <i>chiamata a procedura far</i> o una <i>far jump</i> oppure un interrupt ( <i>int</i> ). In modo protetto viene verificato se il nuovo segmento può essere utilizzato dal task.
<b>DS (EBX)</b>	Data Segment	Contiene l'indirizzo base del segmento dati del programma. Molte istruzioni quali la <b>mov</b> utilizzano questo segmento. E' il segmento in cui sono contenuti i dati di un task.
<b>SS (EBP)</b>	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.
<b>ES, FS, GS</b>	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)

## Registri "General Data"



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

## 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 (inizio 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 sum1 + sum2 + sum3
    pop  ebp            ; recupera l'indirizzo base precedente
    ret                 ; ritorno al programma chiamante (cf. jr $ra)
addition endp
    
```

### Architettura ad accumulatore

## IA-32 - operazioni logico-aritmetiche



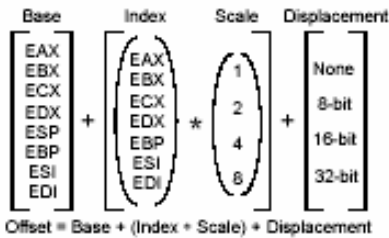
Tipo operando 1	Tipo operando 2	Tipo risultato
Registro	Registro	Registro
Registro	Immediato	Registro
Registro	Memoria	Registro
Memoria	Registro	Memoria
Memoria	Memoria	Memoria

- ❖ 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
<b>Ind. diretto (registro)</b> Register Addressing	Indirizzamento tramite registro (l'operando è in un registro)	No ESP e EBP	<code>move \$s0, \$s1</code>	<code>move eax, ebx</code>
<b>Register Indirect</b>	Base register	No ESP e EBP	<code>lw \$s0, 0(\$s1)</code>	<code>move eax, [ebx]</code>
<b>Base + offset (8 - 32bit)</b>	Base + offset addressing (INTEL - displacement)	No ESP e EBP	<code>lw \$s0, 100(\$s1)</code>	<code>move eax, 100</code>
<b>Base + scale*offset</b>	Ind = base + offset*scale	No ESP e EBP	...	<code>move eax, [esi*4]</code>
<b>Base + scale*offset + displacement</b>	Scaling Factor displacement	No ESP e EBP	...	<code>move eax, [esi*4 + 2]</code>



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

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

A.A. 2

Pedersini - DSI, UniMI

15/44

## Modalità di indirizzamento - istruzioni "immediate"



Modo	Descrizione	Codice MIPS	Codice INTEL
<b>Indirizzamento immediato</b>	L'operando è in una parte dell'istruzione	<code>li \$s0, 0x6a02H</code>	<code>move eax, 0x6a02H</code>
<b>PC_relative addressing</b>	Indirizzamento relativo al Program Counter	<code>bne \$s0, \$s1, label</code>	<code>jnz 0x01A5</code>
<b>Pseudodirect addressing</b>	<b>MIPS:</b> indirizzo ottenuto cambiando i 26 bit dell'istruzione con i 28 LSB di PC (i 2 LSB sono 00) <b>INTEL:</b> modifica del registro Code Segment	<code>j label</code>	<code>move cs, 0x87ee</code> <code>move eip 0x0000</code>

A.A. 2004/05

Copyright: A. Borghese, F. Pedersini - DSI, UniMI

16/44



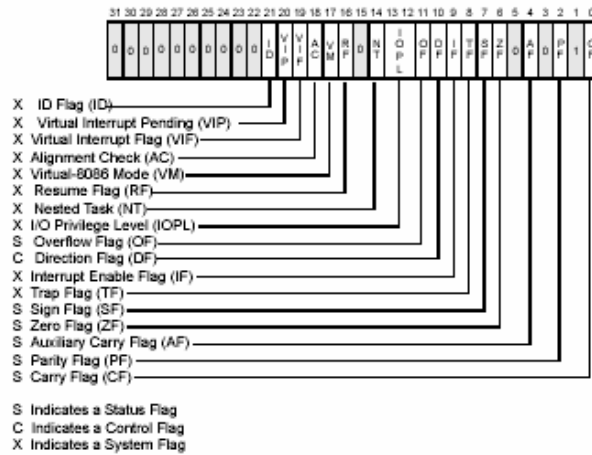
## Registro EFLAG



- ❖ 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, \$zero, label

**IA-32:**  
jz label



## Sommario



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

## 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 **MIO**
- ❖ 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
<b>Controllo</b>	<b>Salto condizionati e incondizionati</b>
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
<b>Trasferimento dati</b>	<b>Spostamento di dati fra registri o fra registri e memoria</b>
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
<b>Aritmetiche e logiche</b>	<b>Operazioni aritmetiche e logiche su dati nei registri o in memoria</b>
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
<b>Stringhe</b>	<b>Trasferimenti fra operandi stringhe; lunghezza data da un prefisso di ripetizione</b>
MOVSB	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 <sub>esa</sub>
MOVSL	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

- JE:** jump equal - near ( $\pm 128$  byte)
- JMP:** jump (near  $\rightarrow$  uso CS; far  $\rightarrow$  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

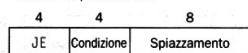
La codifica delle istruzioni

Le pipeline

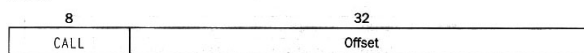
# Codifica delle istruzioni



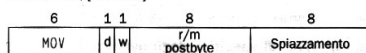
a. JE EIP + spiazamento



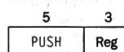
b. CALL



c. MOV EBX, [EDI + 45]



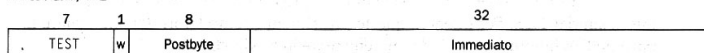
d. PUSH ESI



e. ADD EAX, #6785



f. TEST EDX, #42



❖ Multi formati: ampiezza 1-15byte.

> 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 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

4	3	1	8
ADD	000	0	0000 0001

4	3	1	32
ADD	000	1	00 ... 01

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

## 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
- Le pipeline**

## IA-32: Ciclo di esecuzione di un'istruzione

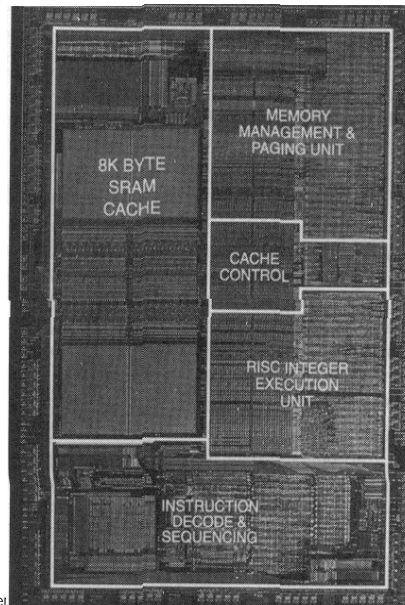
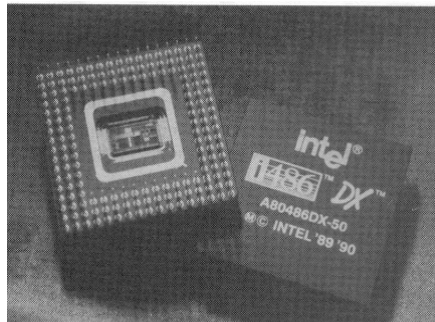


- ❖ **Fase di fetch:** lettura dell'istruzione mediante la coppia:
  - CS:IP dal segmento codice + instruction counter
  - Quindi:  $IP \leftarrow IP + \#byte\_istruzione$
- ❖ **Pre-fetch Queue** (Coda di pre-fetch). Streaming dal segmento codice di RAM in un buffer fino al riempimento. L'UC legge il primo byte dell'istruzione dalla coda di pre-fetch e **trasferisce un certo numero di byte nell'IR.**
- ❖ **Decodifica, Esecuzione.** Esecuzione è pipe-line e multi-ciclo
  - Fino a 300 cicli di clock per i task più complessi quali il task switch tramite gate che viene operata in modalità protetta.

## L'Intel 80486



Integrazione in un solo chip di CPU, Coprocessore e Cache controller.



## L'Intel 80486 - Struttura interna



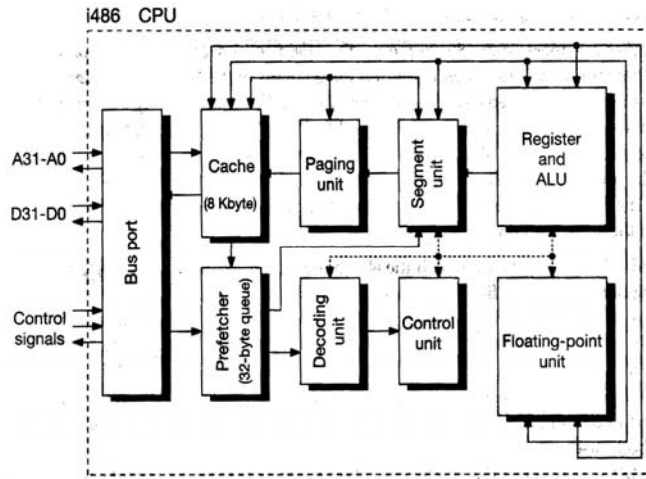
Interfacciamento con il resto della macchina tramite la porta verso il bus. Sul bus vengono inviati i dati, gli indirizzi ed i segnali di controllo.

Pipeline a 5 stadi.

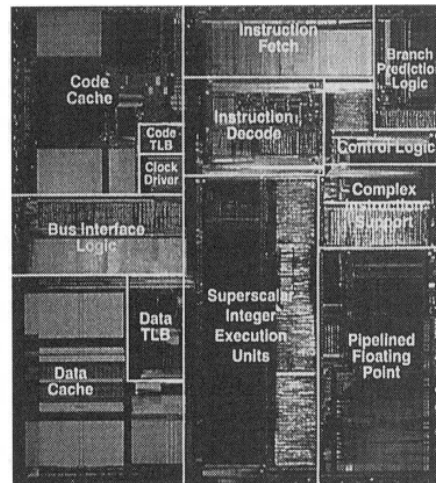
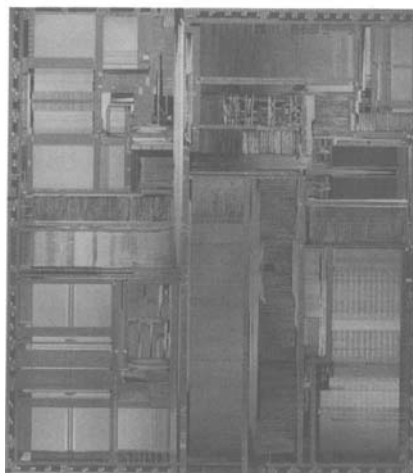
Bus interno a 64bit.

Cache a 8kbyte.  
Modalità di scrittura:  
write-through con  
buffer.

Control unit micro-  
programmata



## Il pentium

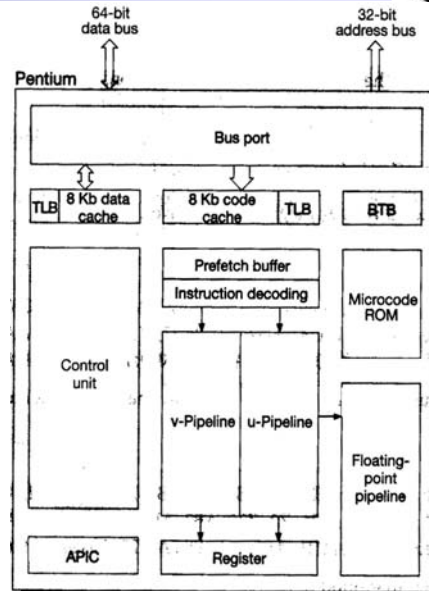




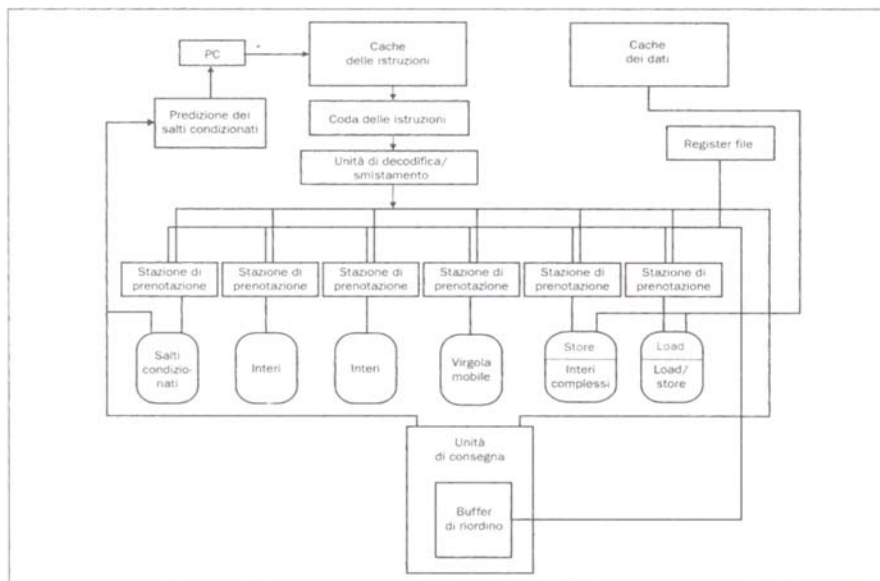
## Pentium – struttura interna



- ❖ 2 pre-fetch queues: la pipeline u si interfaccia con la pipeline floating point.
- ❖ Advanced Programmable Interrupt Controller (è attivo per la doppia CPU).
- ❖ Microcodice per l'esecuzione del controllo.
- ❖ Cache interfacciate sul bus del processore.



## La pipeline del PentiumPro





## Il funzionamento della pipeline del PentiumPro



**Fetch:** Viene prelevato un blocco di memoria istruzioni di 16 byte ed inviato alla coda istruzioni.

**Decodifica:**

1. Le istruzioni dell'ISA INTEL vengono convertite in microistruzioni di lunghezza fissa pari a 72 bit. **Le microistruzioni sono estremamente simili all'ISA del MIPS.**
2. In questa fase vengono valutate le branch (su un certo numero di istruzioni) attraverso una **Branch Prediction Table** con 512 elementi.
3. Recupero degli operandi ed assegnazione loro dei registri replicati (rename buffer).

**Esecuzione:**

1. Smistamento. Le istruzioni vengono alla stazione di prenotazione corrispondente all'unità funzionale richiesta.
2. Un riferimento all'istruzione viene inserito nel buffer di riordino.
3. Esecuzione vera e propria (calcolo)
4. Riconsegna. **L'unità di consegna** tiene traccia di tutte le istruzioni pendenti all'interno del buffer di riordino. Essa **riordina le istruzioni** in modo che terminino in modo sequenziale. Può fornire in uscita più istruzioni in un ciclo di clock.



## Osservazioni sulla fase di esecuzione



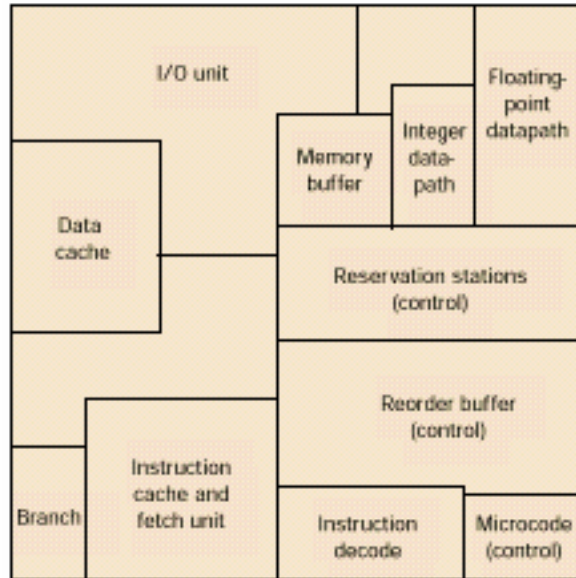
- **Per considerare un'istruzione occorre che ci sia spazio nel Buffer di Riordino e nella stazione di prenotazione relativa alla pipeline di esecuzione richiesta.**
- La CPU contiene duplicazione dei registri in cui scrivere il risultato. Con così tante istruzioni in esecuzione, è possibile avere il registro di destinazione occupato e occorre scrivere quindi su un registro ausiliario (**rename registers or rename buffers**). In questi registri vengono memorizzati i risultati in attesa che l'unità di consegna **dia il permesso di scrivere i registri effettivi della CPU.**
- L'istruzione viene eseguita quando il corrispondente elemento all'interno della stazione di prenotazione possiede tutti gli operandi, e la relativa unità funzionale è libera.
- Un'istruzione non viene terminata (riconsegnata) fino a quando non lo decide l'unità di consegna. Se la predizione di un salto è scorretta, vengono scartate le istruzioni sbagliate dalle stazioni di prenotazione e dai buffer di riordino e liberati i registri interni.



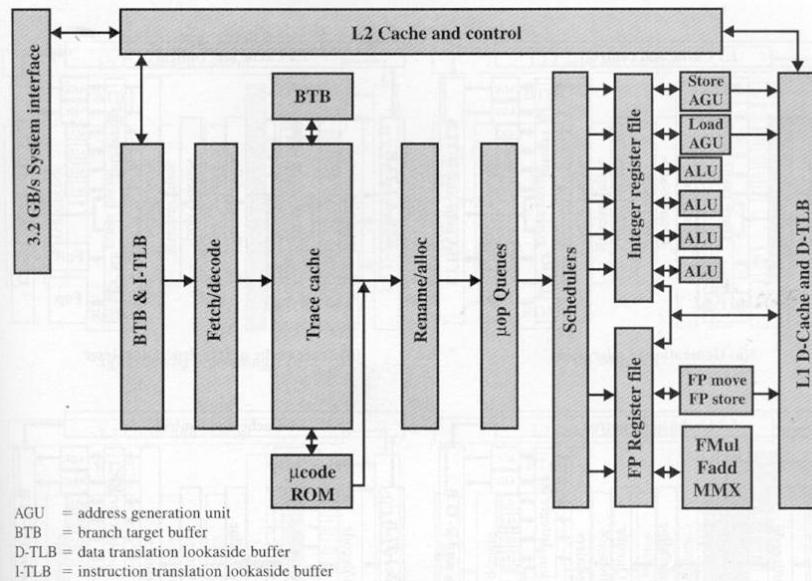
# Il processore Pentium Pro



Esecuzione “speculativa”:  
scheduling dinamico +  
predizione dei salti (e.g. Intel  
80x86 dal Pentium).



# LA CPU del Pentium 4





## Funzionamento della Pipeline del Pentium 4



Fetch delle istruzioni della Memoria (Cache)

Ciascuna istruzione viene tradotta in una o più microistruzioni di lunghezza fissa.

Il processore esegue le micro-istruzioni in una pipeline superscalare a schedulazione dinamica.

Il processore restituisce il risultato (finale) dell'esecuzione delle micro-istruzioni relative alla stessa istruzione ai registri nell'ordine nel quale le istruzioni sono state inviate in esecuzione.

Il Pentium 4 trasforma un'ISA CISC in un'ISA interno RISC costituito dalle micro-operazioni.

Esecuzione parallela di istruzioni, con [parallelismo esplicito](#) si specifica in assembly se istruzioni possono essere eseguite contemporaneamente o sequenzialmente (estensione Processor del compilatore Microsoft Visual C++) o librerie dell'Intel.



## Stadi della pipeline del Pentium 4: Front end



Le istruzioni passano dalla cache primaria al buffer L2 (64 byte). A questo stadio vengono gestite le miss ed i trasferimenti da cache.

In parallelo i dati vengono caricati da cache nel buffer dati L1.

Nel trasferimento da L1 ed L2 agiscono Branch Target Buffer e lookaside buffer che gestiscono le predizioni sulle branch.

A questo punto inizia la fase Fetch e Decodifica che legge il numero di byte pari alla lunghezza dell'istruzione.

Il decoder traduce l'istruzione in da 1 a 4 micro-operazioni: 118bit, appartenenti ad un'ISA RISC.



## Stadi della pipeline del Pentium 4: Trace cache



Riordina localmente il codice in modo da evitare le criticità: la pipeline ha 20 stadi.

Invia alla fase di esecuzione (Allocate e Renaming).

Per istruzioni complesse (> 4 microistruzioni) vengono create in questa fase con una macchina a stati finiti implementata con una ROM + memoria.



## Stadi della pipeline del Pentium 4: Esecuzione



Elemento centrale è il **Reorder Buffer – ROB** (Buffer circolare che può contenere fino a 126 micro-istruzioni).

Contiene le micro-istruzioni con il loro stato, la presenza o assenza dei dati, per tutta la durata dell'esecuzione.

Alloca i registri (traducendo i registri simbolici dell'Assembly in uno dei 128 registri di pipeline).

Avvia alla coda di esecuzione l'istruzione (coda per le lw/sw e coda per le altre istruzioni).

Lo scheduling avviene prendendo la prima istruzione che può essere eseguita nella coda. Dagli scheduler si può passare alle ALU in modi diversi.

La fase di esecuzione scrive poi i risultati nei registri o nella cache L1.

C'è una parte dedicata ai flag che vengono poi inviati alla BTB per predire correttamente i salti.



## Sviluppi futuri: IA-64



- Architettura a 64 bit
  - Progetto MERCED
  - Primo esemplare: **Intel ITANIUM**
- Massiccio aumento delle risorse
  - 128 registri GPR, 128 FPR
  - 16 GP-EU + 16 FP-EU
- Compatibilità con IA-32
  
- Macchina molto complessa → progetto non ancora terminato
  - Progetto ITANIUM “venduto” alla Digital
  - In corso di sviluppo: ITANIUM 2

## Sommario



Le architetture Intel

I registri ed il loro utilizzo

L'ISA degli Intel

La codifica delle istruzioni

**Le pipeline**