



Lezione 34

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. 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 (^{2²⁰}).
 - 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 GPR come in MIPS

Modi di funzionamento IA-32

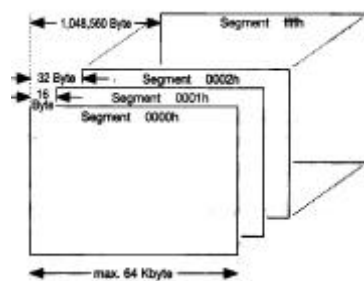


- ❖ **Modalità reale (Real mode)**
 - Modalità compatibile DOS
 - Max memoria indirizzabile: 1 MByte $\rightarrow 2^{20}$
 - modo attivo all'accensione (power-on)
- ❖ **Modalità protetta (Protected mode)**
 - modalità "nativa" di IA-32
 - Memoria indirizzabile: 4 GByte $\rightarrow 2^{32}$
 - Memoria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

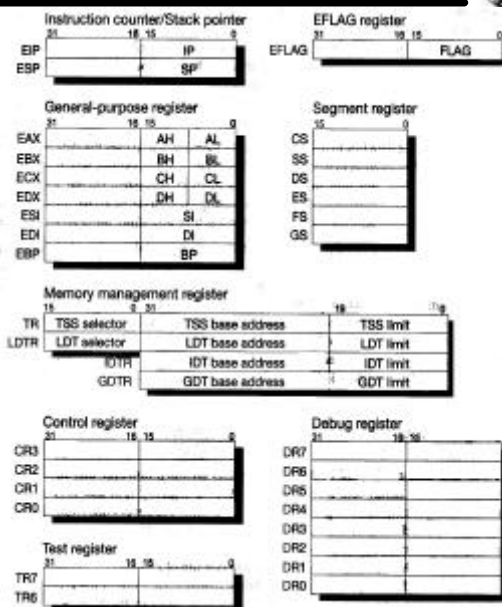


- ❖ **Modalità reale**
 - $\text{Indirizzo} = 16 * \text{Segmento} + \text{Offset}$
 - In tal modo posso indirizzare $64\text{KB} \times 16 = 1 \text{ MByte}$ di MP
 - Modalità **Virtual 8086**: Indirizzo e offset sono separati.
- ❖ **Modalità protetta**
 - Spazio di indirizzamento: 32 bit
 - SALTO: indirizzo = composizione registri CS e IP
 - CS = 0x80B8, IP = 0x019D
 - indirizzo istruz. successiva: **0x 80B8 019D**

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



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!**

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 CS occorre una <i>chiamata a procedura far o una far jump</i> oppure un interrupt (<i>int</i>). 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)

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. | |
| 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 \$ecx, 0(\$ebx) | | | | |
| 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). | |

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 add1      ; create the first summand
                ; (automaticamente ESP viene decrementato)
push add2      ; create the second summand
push add3      ; create the third summand

addition: proc near      ; call near (inside 64k segment, cf. branch)
    push ebp      ; salva l'indirizzo base per il ritorno (record attivaz.)
    move ebp, esp ; copia lo StackP nel BaseP (individua frame di 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 sum1 + sum2 + sum3
    pop ebp          ; recupera l'indirizzo base precedente
    ret             ; ritorno al programma chiamante (cf. jr $ra)
addition endp
    
```

Registri di gestione stringhe



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

Esempio: output della stringa: "abcdefghijklm" su un monitor monocromatico

```

string db 20 dup ('abcdefghijklm') ; definizione della stringa
mov eax, @data      ; caricare l'indirizzo (di inizio) dei dati in EAX
mov ds, eax         ; impostare ds a questo segmento dati
mov eax, b800H     ; caricare l'indirizzo del segmento di RAM video in eax
cld                ; sequenza ascendente
mov ecx, 5         ; trasferisce 5 parole di 4 byte ciascuna
mov esi, OFFSET string ; carica indirizzo stringa nel segmento esteso (stringa sorgente)
mov edi, 00h       ; carica indirizzo del primo carattere in alto a sinistra (stringa destinazione)
movsw              ; trasferisce 5 parole (20 caratteri)
    
```

IA-32 – operazioni logico-aritmetiche



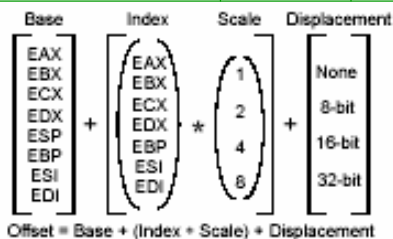
| Tipo operando 1 | Tipo operando 2 | Tipo risultato |
|-----------------|-----------------|----------------|
| Registro | Registro | Registro |
| Registro | Immediato | Registro |
| Registro | Memoria | Registro |
| Memoria | Registro | Memoria |
| Memoria | Memoria | Memoria |

- ❖ Uno dei registri (memoria) deve fungere sia da operando che da registro destinazione (architettura ad accumulatore)
 - 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 |
|--|---|--------------------------|--------------------|-----------------------|
| (registro) Register | Indirizzamento tramite registro (l'operando è in un registro) | No ESP e EBP | move \$s0, \$s1 | move eax, ebx |
| Addressing 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] |



Modalità di indirizzamento – istruzioni “immediate”



| Modo | Descrizione | Codice MIPS | Codice INTEL |
|--------------------------|---|------------------------------------|--|
| Indirizzamento immediato | L'operando è in una parte dell'istruzione | <code>li \$s0, 0x6a02H</code> | <code>move eax, 0x6a02H</code> |
| PC_relative addressing | Indirizzamento relativo al Program Counter | <code>bne \$s0, \$s1, label</code> | <code>jnz 0x01A5</code> |
| Pseudodirect 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 | <code>j label</code> | <code>move cs, 0x87eeae</code> <code>move eip 0x000000</code> |

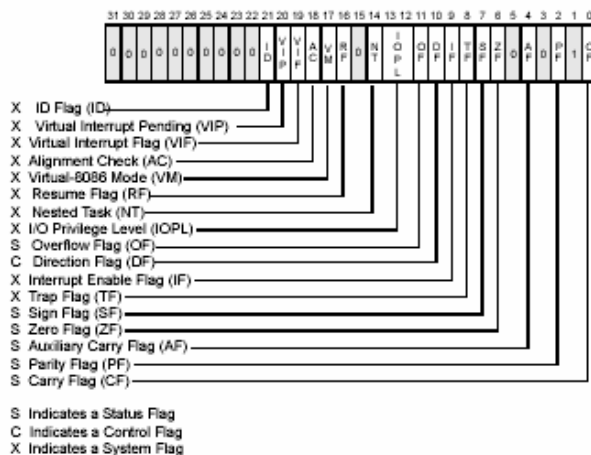
Registro EFLAG



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

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

IA-32:
`move eax, [ebx]`
`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.

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 e 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 |
| CMPL | Confronta il sorgente con la destinazione; formato registro-memoria |
| SHL, SHR, RCL | 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 ₁₆ |
| MOVSL | M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4 |

- JE:** jump equal - near (± 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 42**
- MOVSL:** Sposta 4 byte e incrementa EDI ed ESI (stringhe/aree dati)

Codifica delle istruzioni



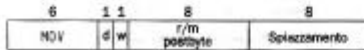
a. JE EIP + spostamento



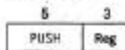
b. CALL



c. MOV EBX, [EDI + 45]



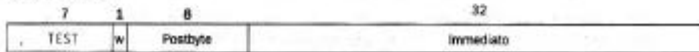
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



- ❖ Molti 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**

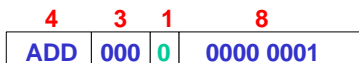


❖ Campo **reg**:

- 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



| 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 | 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 | CK | ECX | ECX | 1 | Ind=BX+DI | =ECX | * | * | * | * | * |
| 2 | DL | DK | 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 | =(sb) | SI+disp8 | (sb)+disp8 | SI+ disp8 | (sb)+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)

IA-32: Ciclo di esecuzione di un'istruzione

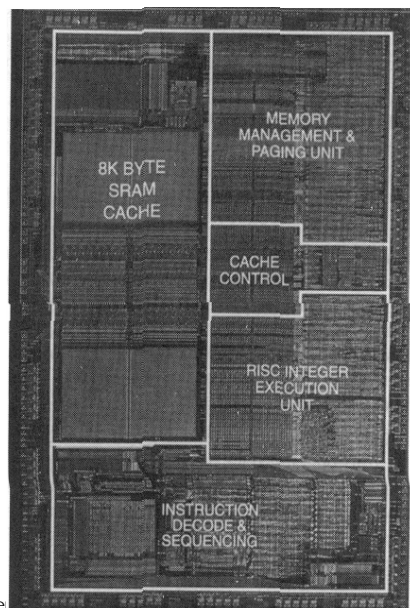
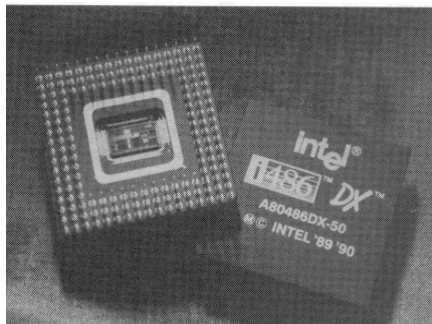


- ❖ **Fase di fetch:** lettura dell'istruzione mediante la coppia:
 - CS:IP dal segmento codice + instruction counter
 - Quindi: $IP \leftarrow IP + \#byte_istruzione$
- ❖ **Pre-fetch Queue** (Coda di pre-fetch). Streaming dal segmento codice di RAM in un buffer fino al riempimento. L'UC legge il primo 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,
Coprorocessore e Cache controller.



L'Intel 80486 – Struttura interna



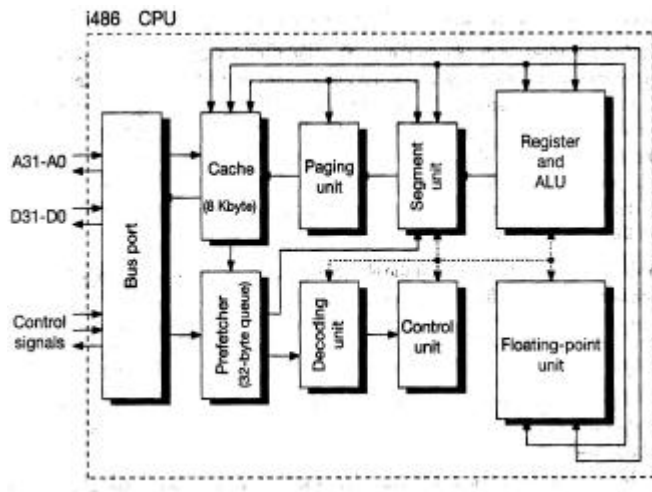
Interfacciamento con il resto della macchina tramite la porta verso il bus. Sul bus vengono inviati i dati, gli indirizzi ed i segnali di controllo.

Pipeline a 5 stadi.

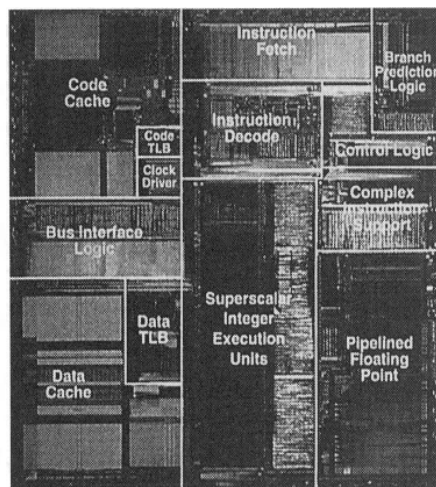
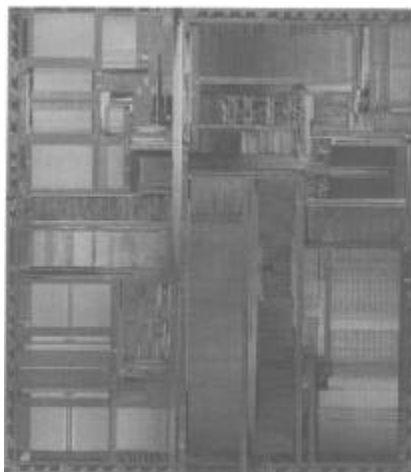
Bus interno a 64bit.

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

Control unit micro-
programmata



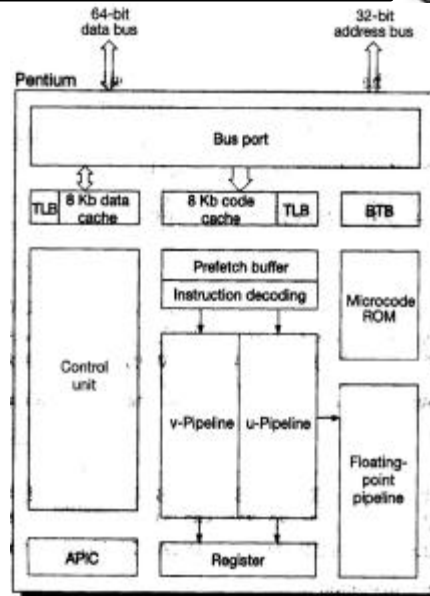
Il pentium



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.



A.A. 2003/04

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

L. 34 – 29/30

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
- ❖ Esecuzione parallela di istruzioni, con parallelismo esplicito
 - si specifica in assembly se istruzioni possono essere eseguite contemporaneamente o sequenzialmente
- ❖ Compatibilità con IA-32

- ❖ Macchina molto complessa → progetto non ancora terminato
 - Progetto ITANIUM “venduto” alla Digital
 - In corso di sviluppo: ITANIUM 2

A.A. 2003/04

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

L. 34 – 30/30