

Processore Intel Pentium

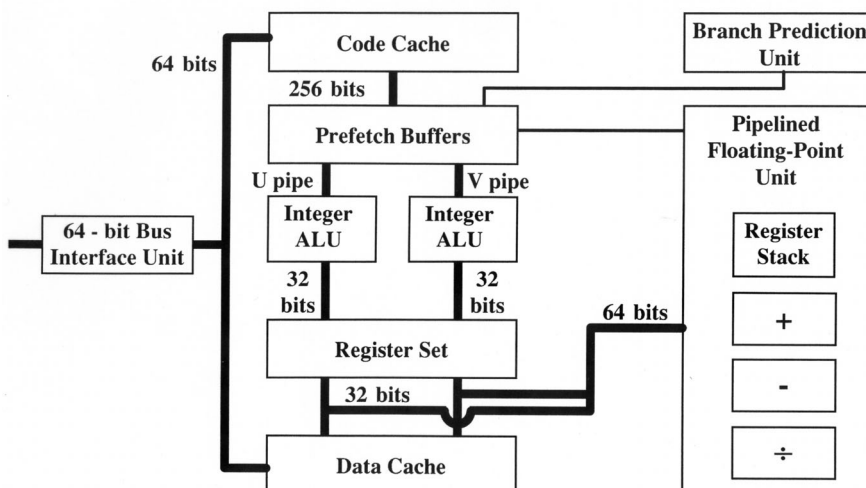
Evoluzione tecnologica

| Evoluzione tecnologica microprocessori Intel | | | | |
|--|------|--|----------------|--------------------|
| Microprocessore | Anno | Tecnologia | N. transistors | Clock Freq. (MHz.) |
| i386 CPU | 1986 | 1.5 μm CMOS due strati metalliz. | 275 K | 16 |
| i486 CPU | 1989 | 1.0 μm CMOS due strati metalliz. | 1.2 M | 33 |
| Pentium CPU | 1993 | 0.8 μm CMOS tre strati metalliz. | 3.1 M | 66 |

Caratteristiche

- Architettura superscalare:
 - Due unità per il calcolo delle operazioni sugli interi
- Branch prediction dinamica
- Unità floating point a pipeline
- Due cache di primo livello da 8KB:
 - Una per i dati e una per il codice
- Data cache con protocollo MESI
- Bus dati a 64 bit
- Bus indirizzi a 32 bit

Pentium: schema a blocchi



Pipeline a interi

- PF: stadio di prefetch
 - Durante questa fase la CPU preleva il codice dell'istruzione dalla cache e la allinea al primo byte della prossima istruzione dal decodificare
- D1: stadio di prima decodifica
 - Durante questa fase la CPU decodifica l'istruzione e genera una control word (eventuale utilizzo di microcodice per istruzioni lunghe)
- D2: stadio di seconda decodifica
 - Durante questa fase viene decodificata la control word prodotta in D1 e vengono generati eventuali indirizzi per gli operandi

Pipeline a interi

- E: stadio di execute
 - Durante questa fase la CPU accede ai dati nella cache, usa la ALU, o altre unità funzionali per i dati
- WB: stadio di write back
 - Durante questa fase vengono aggiornati i registri e i flag alla luce del risultato prodotto dallo stadio precedente. Tutte le possibili eccezioni devono essere gestite prima che una nuova istruzione raggiunga lo stadio di WB

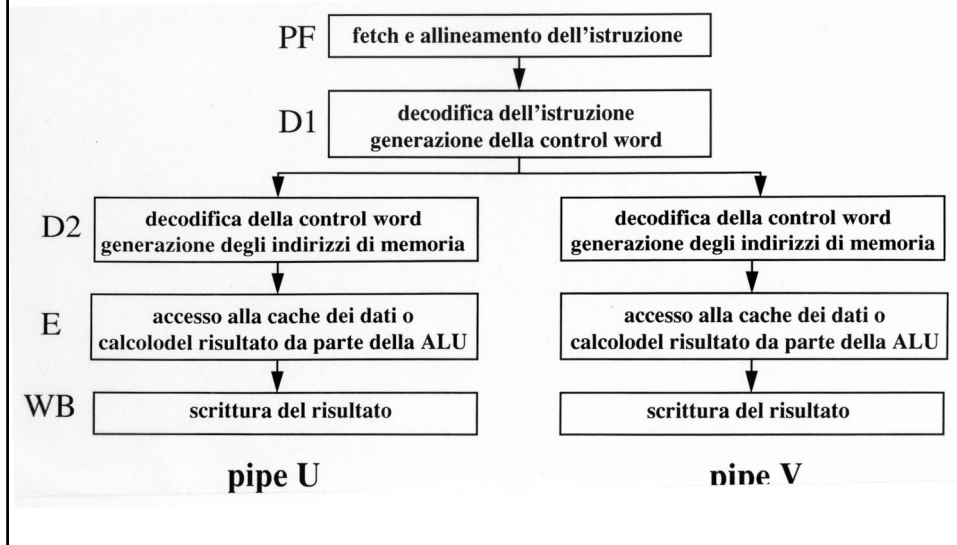
Pipeline a interi

- È garantito che qualunque istruzione rimanga in D1 per un solo colpo di clock
- Non è garantito che tutte le istruzioni permangano in execute (E) per un solo colpo di clock:
 - Le istruzioni che usano la ALU e la cache necessitano più di un colpo di clock

Esecuzione superscalare

- Due code di pipeline per gli interi U e V che possono eseguire le operazioni in parallelo
- Il throughput per ciascuna di esse è di una istruzione semplice per ogni ciclo di clock
 - Le istruzioni semplice non richiedono microcodice per la loro esecuzione
- Le due code non sono perfettamente simmetriche

Esecuzione superscalare



Esecuzione superscalare

- Negli stadi PF e D1 la CPU è in grado di eseguire il fetch e la prima decodifica di due istruzioni semplici in parallelo e di inviarle alle due code U e V
- In PF e D1 operano due unità di prefetch e due unità di decodifica in parallelo
- Per istruzioni complesse la CPU è in grado di generare le sequenze di microcodice per entrambe le code

Esecuzione superscalare

- Affinché le due code possano operare in parallelo "a pieno ritmo" è necessario che:
 - Non vi siano dipendenze tra le istruzioni
 - Siano gestiti accuratamente i salti per non dover svuotare entrambe le code
 - Siano gestiti correttamente gli accessi simultanei alla cache per i dati
 - Sia mantenuta la consistenza della cache per le istruzioni nel caso di codice automodificante

Dipendenza tra le istruzioni

- Il criterio usato da D1 per gestire le dipendenza tra le istruzioni è:
 - Decodifica due istruzioni consecutive I1 e I2
 - Se valgono tutte le seguenti condizioni:
 - I1 è un'istruzione semplice
 - I2 è un'istruzione semplice
 - I1 non è un'istruzione di salto
 - Destinazione di I1 \neq sorgente di I2
 - Destinazione di I1 \neq destinazione di I2
 - Allora invia I1 alla coda U e I2 alla coda V
 - Altrimenti invia I1 alla coda U

Dipendenza tra le istruzioni

- La dipendenza tra le istruzioni si divide in:
 - Dipendenza tra le risorse
 - Dipendenza tra i dati
 - Dipendenza nel controllo

Dipendenza tra le risorse

- Si ha dipendenza tra risorse quando due (o più istruzioni) richiedono la stessa unità funzionale
 - D1 elimina la maggior parte delle dipendenze tra risorse non permettendo l'esecuzione a istruzioni che non siano entrambe semplici (circa il 90% delle istruzioni rientra in questa categoria SPECint92)
- Le due code U e V non sono intercambiabili:
 - La coda V è in grado di eseguire istruzioni: che necessitano della ALU, che facciano riferimento alla memoria, di salto
 - La coda U in più esegue operazioni che necessitano di logica non replicata presente solo nella coda U (ad esempio il barrel shifter)

Dipendenza tra i dati

- Si verifica dipendenza tra i dati quando un'istruzione scrive un risultato che deve essere letto o scritto da un'altra istruzione:
 - La logica allo stadio D1 assicura che i registri sorgente e destinazione dell'istruzione inviata alla coda V differiscano dal registro destinazione dell'istruzione inviata alla coda U

Dipendenza nel controllo

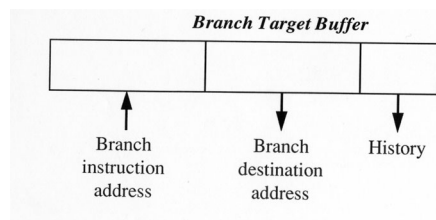
- Si ha una dipendenza nel controllo qualora il risultato di un'istruzione determina se un'altra istruzione possa o meno essere eseguita:
 - Se un'istruzione di salto viene inviata alla coda U, D1 non invia un'altra istruzione alla coda V

Dipendenza tra le istruzioni

- D1 non risolve la dipendenza tra dati e risorse nel caso di riferimenti alla memoria
- Istruzioni che causano potenziale conflitto tra riferimenti a memoria possono essere inviate in parallelo alle due code
 - La risoluzione di tali conflitti dipende dalla gestione della cache di dati

Branch prediction

- Il Pentium utilizza un branch target buffer (BTB) costituito da una memoria set-associativa a quattro vie da 256 entry.
- Il BTB è usato per incrementare le prestazioni nel caso in cui un'istruzione di salto alteri il flusso sequenziale delle istruzioni



Branch prediction

- Quando un'istruzione di salto viene eseguita per la prima volta, la CPU alloca una entry nel BTB per associare all'indirizzo dell'istruzione di salto l'indirizzo della prima istruzione da eseguire:
 - Viene inizializzata la "storia"
- Quando un'istruzione di salto viene decodificata (D1), la CPU accede al BTB per vedere se quest'ultimo contiene una entry relativa all'istruzione di salto in questione:
 - Se la entry è presente, la CPU usa la storia per prevedere il salto
 - La conferma (o la smentita) della previsione avviene nello stadio di WB:
 - Predizione errata → svuotamento della coda
 - Predizione corretta → le istruzioni di salto non introducono ritardi

Organizzazione della cache

- Il Pentium utilizza due cache separate per dati e istruzioni nonostante che una cache unificata (ad esempio Intel 80486) presenti i seguenti vantaggi:
 - La ripartizione tra dati e codice è automaticamente bilanciata in ragione delle esigenze del programma
 - Deve essere progettata solo una cache

Organizzazione della cache

- Per contro una cache unificata presenta i seguenti svantaggi:
 - Un'efficiente gestione dei salti prevede che si faccia accesso ad una istruzione mentre altre istruzioni in esecuzione nelle code di pipeline accedono a dati e operandi in memoria
 - La sola cache dati deve già supportare gli accessi contemporanei da parte delle due code U e V

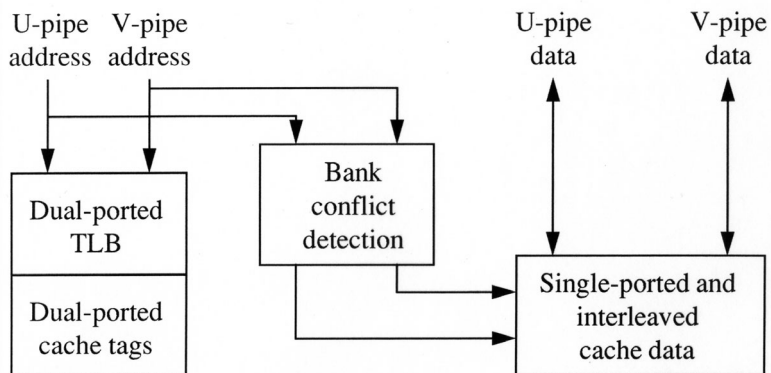
Cache dati

- La cache dati è una memoria set-associativa a due vie di 8 KB con linee da 32 byte
- La cache dati supporta accessi contemporanei provenienti da entrambe le code U e V
- Poiché la cache "risponde" a indirizzi fisici è necessario gestire la traduzione da indirizzo virtuale a indirizzo fisico "in loco" (TLB)

Cache dati

- Il data path è invece a singola porta (interleaving a otto vie di banchi da 32 bit)
- Nel caso di conflitto la U ha priorità sulla coda V
- La politica adottata per la coerenza della cache è di tipo write-back con protocollo MESI

Cache dati



Dual-access data cache

Cache codice

- La cache per il codice è una memoria da 8kB set-associativa con linee da 32 byte
- Anch'essa è provvista di un TLB per la traduzione degli indirizzi
- Utilizza un sottoinsieme del protocollo MESI per garantire la consistenza con il codice in memoria:
 - Quando un'istruzione viene modificata dal codice stesso si parla di codice automodificante

Codice automodificante

- Nel caso di istruzioni di salto c'è il seguente problema:
 - Il codice dell'istruzione prevista come destinazione viene caricato prima che l'istruzione precedente quella di salto completi l'esecuzione
 - Se questa istruzione modifica quella di destinazione del salto si genera un'inconsistenza
 - La CPU dispone di una logica per verificare quando viene modificata un'istruzione il cui indirizzo, memorizzato nel BTB, è stato usato per la predizione di un salto → In tal caso coda svuotata
 - Quando un'istruzione ne modifica un'altra il risultato viene scritto direttamente in memoria → MESI semplificato

Pipeline floating-point

- L'unità floating point è integrata nella CPU:
 - È aderente allo standard IEEE P754 (singola precisione su 32 bit, doppia precisione su 64 bit)
 - Offre precisione estesa su 80 bit
 - Pipeline su otto stadi
 - Le istruzioni floating point "base" (add, sub, multiply, compare) vengono eseguite in un ciclo di clock

Pipeline floating-point

| | | | | | | | | |
|---------------------|----|----|----|---|----|----|----|----|
| Integer pipe | PF | D1 | D2 | E | WB | | | |
| Floating-point pipe | PF | D1 | D2 | E | X1 | X2 | WF | ER |

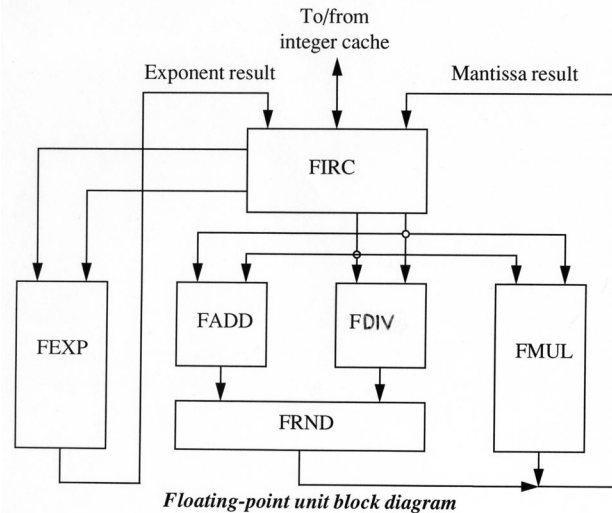
Pipeline floating-point

- PF: prefetch (lo stesso della coda ad interi)
- D1: prima decodifica (lo stesso della coda ad interi)
- D2: seconda decodifica (lo stesso della coda ad interi)
- E: fetch degli operandi. Si fa accesso alla cache di dati ed al floating-point register file per il caricamento degli operandi
- X1: primo passo del calcolo
- X2: secondo passo del calcolo
- WF: stadio di write float. Si completa l'operazione e si scrive il risultato nel register file
- ER: error reporting. In questo stadio si settano i flag nella floating-point status word e vengono riportate eventuali eccezioni

Pipeline floating-point

- Le istruzioni floating-point utilizzano le risorse hardware della coda di pipeline U ed in generale non possono essere eseguite in parallelo ad altre istruzioni
- Il progetto della FPU è stato ottimizzato per quelle istruzioni che utilizzano un operando di 64 bit in memoria ed un operando nel register file della FPU
 - Per tali istruzioni, durante lo stadio E, i due canali della cache dati vengono unificati per portare un dato su 64 bit alla FPU

FPU



FPU

- La Floating-point Interface, Register file e Control (FIRC) è l'unica interfaccia della FPU verso il resto del processore
- FIRC alloca alle istruzioni floating-point le risorse di cui necessitano
- Essendo la coda di pipeline per le istruzioni floating-point agganciata alla coda U, la FIRC vede passare tutte le istruzioni eseguite in U
 - È in grado di allocare risorse anche per istruzioni intere che ne abbiano bisogno
- FEXP: la Floating-point EXponent unit calcola esponente e segno delle operazioni



FPU

- Il Floating-point Multiplier (FMUL) è in grado di gestire moltiplicazioni floating point in:
 - Singola precisione (mantissa su 24 bit)
 - Doppia precisione (mantissa su 53 bit)
 - Precisione estesa (mantissa su 64 bit)
- FMUL gestisce anche gli arrotondamenti delle operazioni di moltiplicazione
- FMUL esegue anche le moltiplicazioni intere implementate in microcodice per la coda U
- Il Floating-point Adder (FADD) esegue:
 - Addizioni, sottrazioni e compare
 - Operazioni BCD tramite microcodice
 - Conversioni di formato



FPU

- Il Floating-point Divider (FDIV) esegue:
 - Divisioni
 - Calcolo del resto
 - Radici quadrate
- Il Floating-point Rounder (FRND) arrotonda i risultati forniti da FADD e FDIV
- I register file è costituito dall'insieme di 8 registri, usati come uno stack, per memorizzare i risultati delle operazioni floating-point:
 - Il risultato dell'operazione appena conclusa viene posta sulla sommità dello stack

Cicli di BUS

- Il Pentium è in grado di gestire diversi tipi di cicli di BUS tra cui:
 - Cicli di trasferimento di dati per trasferire un singolo dato di dimensioni non superiori agli 8 byte
 - Cicli burst in cui avvengono 4 trasferimenti. I trasferimenti cache ↔ memoria avvengono soltanto per mezzo di cicli burst
 - Operazioni sul BUS per consentire operazioni particolari quali locked read modify-write

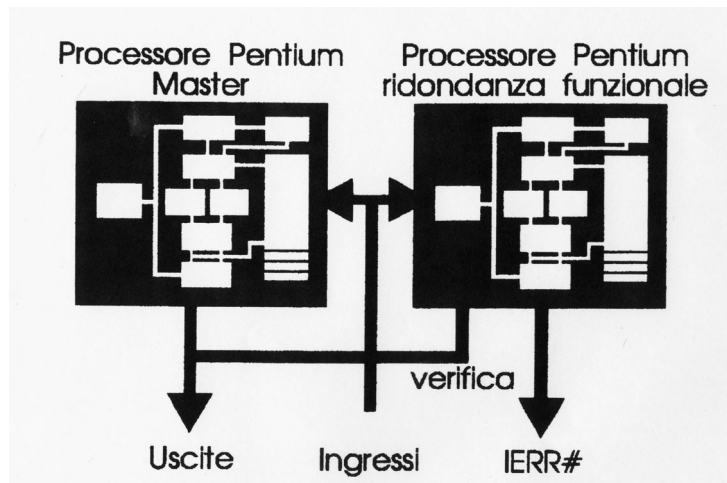
Cicli di BUS

- Usati per trasferire multipli: ad esempio le linee della cache su 32 byte (4 dati singoli su 8 byte)
- Per ogni ciclo burst il processore genera il primo indirizzo (quello del primo dato) mentre gli altri devono essere generati da un'opportuna logica esterna
- In un trasferimento di tipo burst il tipo di operazione non cambia (o solo lettura o solo scrittura)

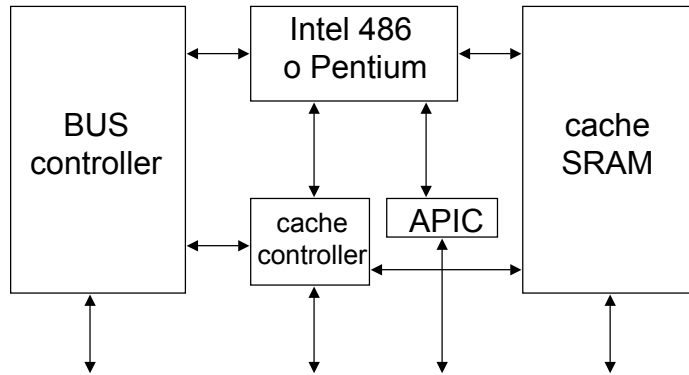
Integrità dell'informazione

- Il Pentium è in grado di supportare un controllo di parità:
 - Sui dati
 - Sugli indirizzi
 - Interno
 - Nella cache di codice c'è un bit di parità ogni 8 byte ed un bit di parità per ogni tag
 - Nella cache dati c'è un bit di parità per ogni byte di dato ed uno per ogni tag
- Il Pentium supporta un controllo di ridondanza funzionale:
 - Un secondo Pentium assume la funzione di checker lavorando in parallelo al processore "master"

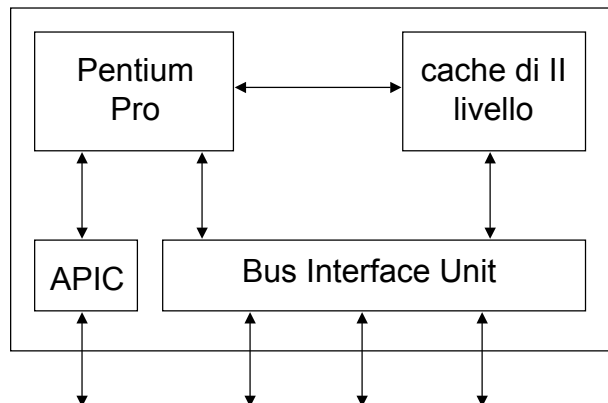
Ridondanza funzionale



Configurazione Pentium



Configurazione PentiumPro





PentiumPro

- La prima differenza del Pentium rispetto al PentiumPro è nell'esecuzione dinamica delle istruzioni. Il PentiumPro adotta:
 - prevenzione degli stalli del processore durante gli accessi alla cache, alla memoria e agli I/O
 - esecuzione speculativa delle istruzioni e branch prediction multiplo.
 - esecuzione di tutte le istruzioni in modo out of order O-O-O (tutte le dipendenze tra i dati e le istruzioni sono risolte)
- Istruzioni che non hanno dipendenze, o le cui dipendenze sono state risolte, possono essere eseguite in qualunque ordine; il risultato può essere "congelato" in attesa di doverlo utilizzare.



PentiumPro

- Quando si verifica un cache miss nel PentiumPro la macchina non va in stallo per aspettare i dati che devono essere letti dalla memoria principale, ma continua ad eseguire istruzioni successive a quella che hanno provocato il cache miss finché il dato mancante non sarà caricato nella cache.
- La fase di fetch delle istruzioni avviene in order (cioè sequenzialmente), mentre la fase di esecuzione è di tipo out of order; le istruzioni che non possono essere eseguite (manca il dato cache) vengono "accantonate" e riprese appena possibile.

PentiumPro

- Una falsa dipendenza dei dati deriva spesso da un numero limitato di registri a disposizione, ad esempio:

- MOV EAX, 17
- ADD MEM, EAX
- MOV EAX, 3
- ADD EAX, EBX

il codice mostra una falsa dipendenza delle istruzioni sul registro EAX.

- Il PentiumPro ha 40 registri interni; quando avviene una scrittura nel registro EAX avviene anche la scrittura su un registro alias. Tutte le seguenti letture referenzieranno questo registro alias. Una ulteriore scrittura di EAX comporterà semplicemente (in modo trasparente) la scrittura in un altro registro alias.
- Le istruzioni del codice precedente non hanno più dipendenze se gestite con il PentiumPro, in quanto le due scritture su EAX non implicano più un ordine sequenziale.

PentiumPro

- Il PentiumPro è dotato di un meccanismo di predizione dei salti più complesso di quello del Pentium.
- Il BTB è stato raddoppiato sia in larghezza sia in profondità e può fornire fino a un massimo di 4 previsioni (il Pentium poteva predire solo un indirizzo per i salti).
- Le cache di primo e di secondo livello sono non bloccanti nel senso che possono gestire un cache miss e contemporaneamente gestire accessi ai dati presenti in cache.
- Le cache possono "soportare" fino a 4 cache miss contemporanei.