


Trend di sviluppo delle pipeline

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
alberto.borghese@.unimi.it

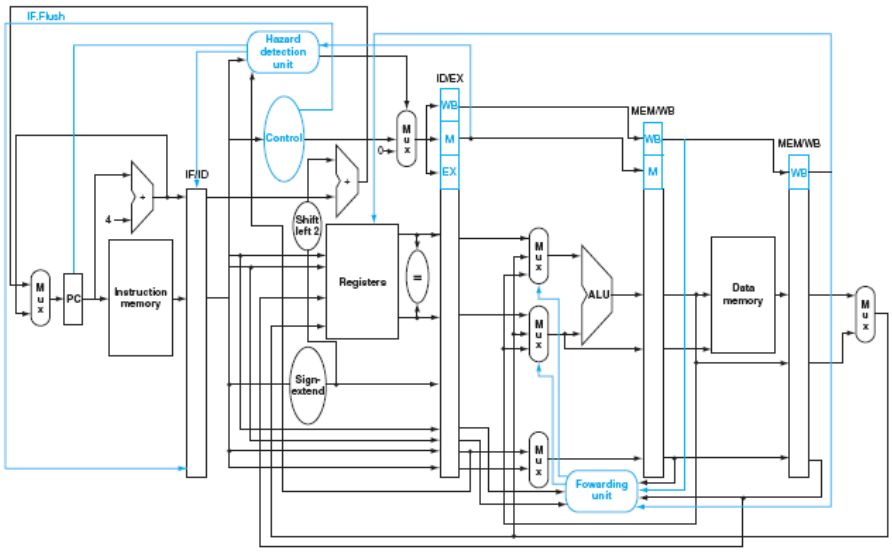
Università degli Studi di Milano

Patterson 4.10, 4.11; 3.7 subword parallelism


Sommario

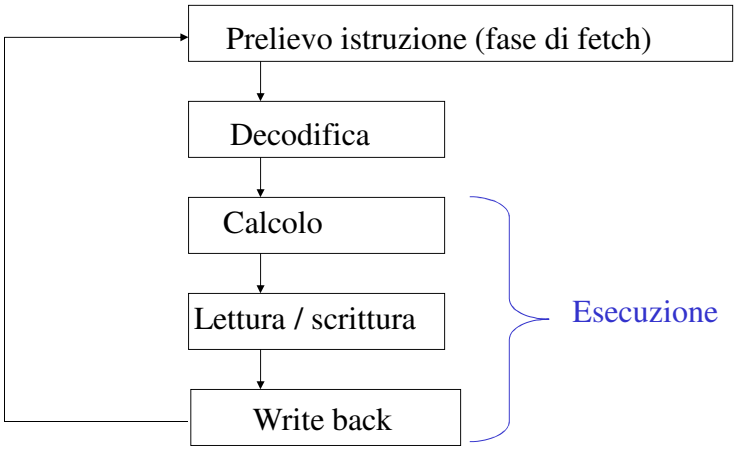
Superpipeline
Multiple-Issue Statici
Multiple-Issue Dinamici
Alcune pipeline

CPU con pipeline completa della gestione degli hazard



CPI = # Cicli_clock per istruzione ≥1

Ciclo di esecuzione di un'istruzione MIPS



Pipeline avanzate: esecuzione parallela

Instruction level parallelism (ILP), **parallelismo implicito**.

Come si può aumentare?

- Superpipelining (pipeline più lunga).
- "Multiple-issue" (esecuzione parallela di un issue packet).
  Static multiple issues (schedulazione decisa dal compilatore)
  Dynamic multiple issues (schedulazione decisa run-time dalla CPU).

Corrisponde alla suddivisione del lavoro tra SW e HW, cioè tra il compilatore ed il processore.

Speculazione e Scheduling dinamico.

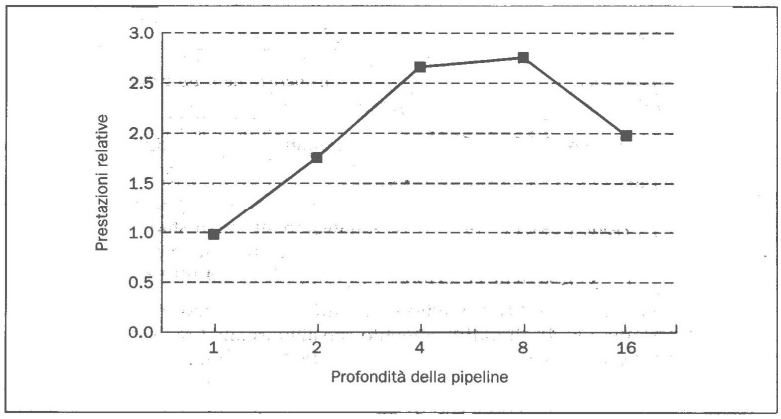
Superpipeline

Pipeline più lunghe (Arm A-8 e Intel Core i7 hanno 14 stadi).
Teoricamente: guadagno in velocità proporzionale al numero di stadi.

Problemi:
•Criticità sui dati: stalli più frequenti.
•Criticità sul controllo: numero maggiore di stadi di cui annullare l'esecuzione (flush).
•Maggior numero di registri: il clock non si riduce linearmente con il numero degli stadi, perché?

Superpipeline

Pipeline più lunghe (e.g. Digital DecAlpha 21264, 5-7 stadi).
Teoricamente: guadagno in velocità proporzionale al numero di stadi.



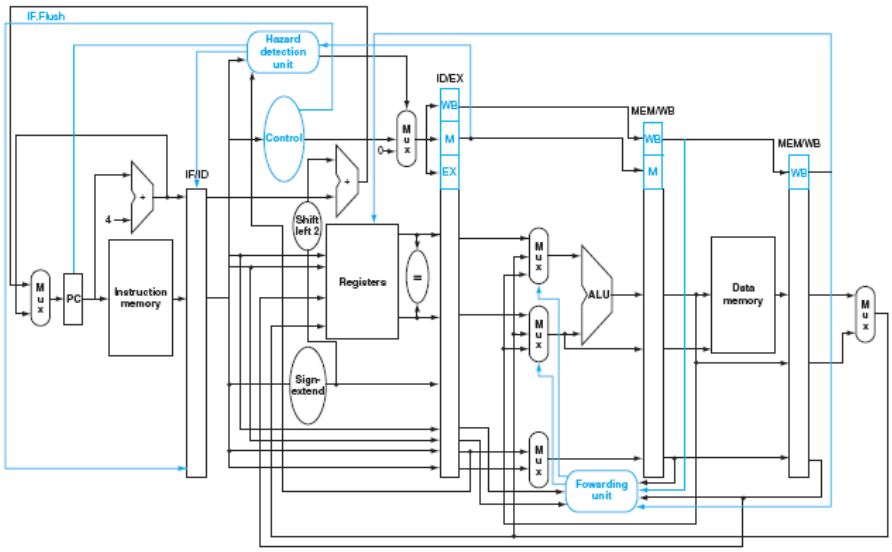
Problemi:

- Criticità sui dati: stalli più frequenti.
- Criticità sul controllo: numero maggiore di stadi di cui annullare l'esecuzione.
- Maggior numero di registri: il clock non si riduce linearmente con il numero degli stadi.

Sommario

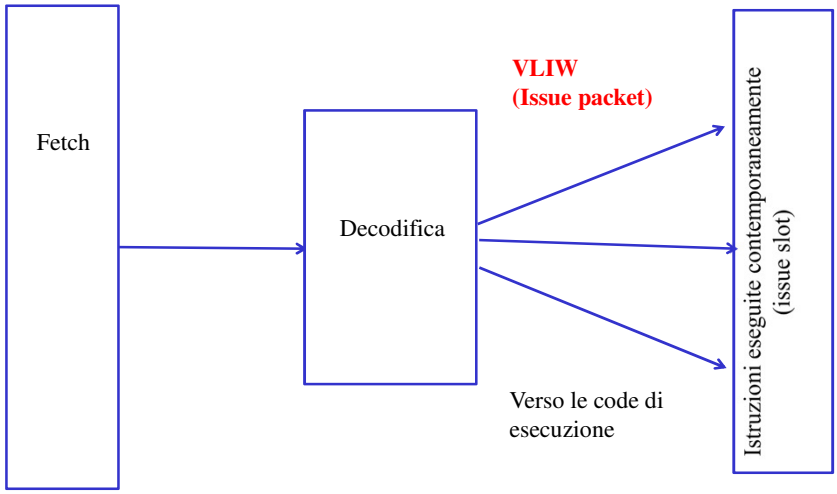
Superpipeline
Multiple-Issue Statici
Multiple-Issue Dinamici
Alcune pipeline

Principi di una pipeline Multiple Issue



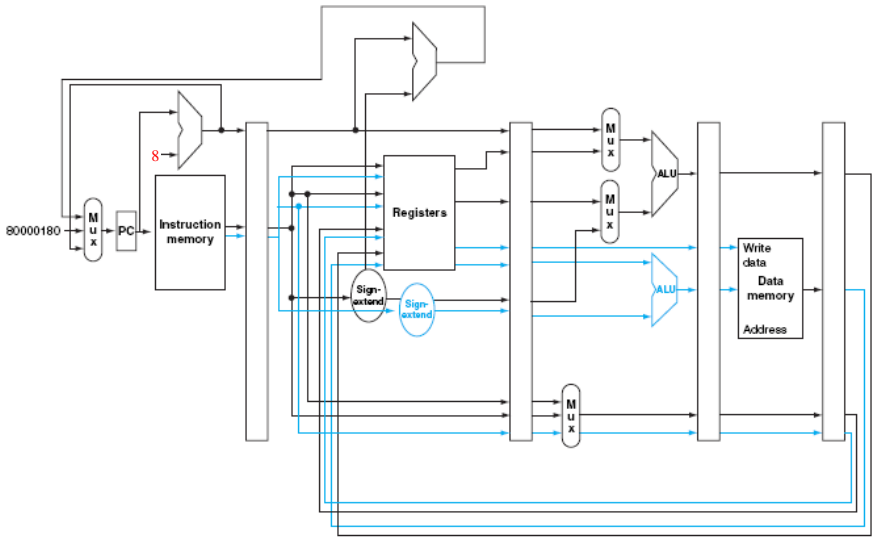
A.A. CPI = # Cicli_clock per istruzione diventa < 1 => Instructions Per Clock cycle (IPC) > 1 ii.it\

Struttura Pipeline Multiple Issue statico



Esistono dei vincoli su quali istruzioni inserire nello stesso issue packet (VLIW)
L'ordine di esecuzione viene deciso dal compilatore.
Primi progetti: Itanium (2000), Itanium-2 (2002) by Intel.

Multiple-issue statici: il MIPS64



Schedulazione delle istruzioni in MIPS a 2 Vie (MIPS64)

La prima via è riservata alle operazioni aritmetico-logiche o di branch
La seconda via è riservata alle operazioni di load / store
(load / store architecture)

Tipo di istruzioni	Stadi di pipeline							
ALU o salto	IF	ID	EX	MEM	WB			
Load o store	IF	ID	EX	MEM	WB			
ALU o salto		IF	ID	EX	MEM	WB		
Load o store		IF	ID	EX	MEM	WB		
ALU o salto			IF	ID	EX	MEM	WB	
Load o store			IF	ID	EX	MEM	WB	
ALU o salto				IF	ID	EX	MEM	WB
Load o store				IF	ID	EX	MEM	WB

CPI = 0,5 – 2 istruzioni per ogni ciclo di clock. E' vero?

Problemi nella creazione dello issue packet

Hazard sul controllo. Due istruzioni vengono eliminate dalla pipeline.

Hazard sui dati oltre a quelli insiti nel codice sequenziale, si generano **hazard dovuti alla parallelizzazione del codice**.

add $t0, $t1, $t2	Non provoca stallo su una pipeline single issue	
lw $s0, 40($t0)	stallo di 1 istruzione nella pipeline a due vie:	
	Q1	Q2
è errata:	add $t0, $t1, $t2	lw $s0, 40($t0)
è corretta:	add $t0, $t1, $t2	nop
		lw $s0, 40($t0)

Hazard sui dati (stall on load).

lw $s0, 40($t0)	Provoca stallo di 1 istruzione su una pipeline single issue	
add $t0, $t1, $s0	Provoca stallo di 2 istruzioni nella pipeline a due vie:	
	Q1	Q2
è errata:	add $t0, $t1, $s0	lw $s0, 40($t0)
è corretta:	nop	lw $s0, 40($t0)
	nop	
	add $t0, $t1, $s0	

Esempio

```
Ciclo: lw   $t0, 0($s1)
       addu $t0, $t0, $t2
       sw   $t0, 4($t0)
       addi $s1, $s1, -4
       bne  $s1, $zero, Ciclo
       or   $s6, $s7, $s5
```

Senza riorganizzazione del codice, occorre inserire una bubble dopo la lw

Il contenuto del registro $t0 viene riscritto da 3 istruzioni consecutive, non fa a tempo a depositarsi nel register file.

Ciclo:		lw $t0, 0($s1)	1
	addi $s1, $s1, -4	~~sw~~	2
	addu $t0, $t0, $s2	~~sw~~	3
	bne $s1, $zero, Ciclo	sw $t0, 4($s1)	4
	or $s6, $s7, $s5		5

CPI = ?

Sommario

Superpipeline
Multiple-Issue Statici
Multiple-Issue Dinamici
Alcune pipeline

Dynamic multiple issues

Questi processori sono detti anche **superscalari**.

La scelta di quali istruzioni inviare alla pipe-line viene eseguita durante l'esecuzione stessa. Dipende dalla compatibilità tra le varie istruzioni e da eventuali hazard su dati e controllo.

Nella versione più semplice, le istruzioni sono processate in sequenza ed il processore decide se elaborarne nessuna (stallo), una o più di una a seconda delle criticità riscontrate.

L'ottimizzazione del codice da parte del compilatore è comunque richiesta.

E' la CPU che garantisce la correttezza dell'esecuzione.

Confronto tra Multiple-issue statici e dinamici

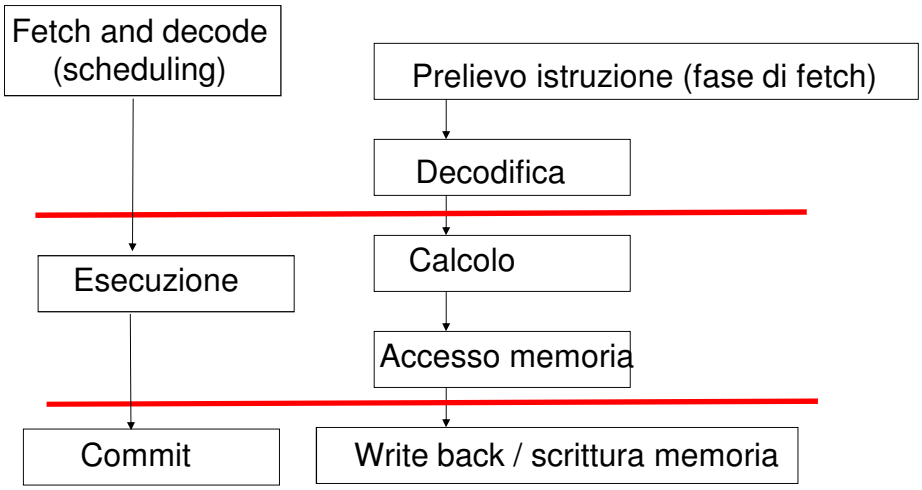
L'hardware di una pipe-line superscalare garantisce la correttezza del codice.

Il codice verrà eseguito correttamente qualunque sia la CPU sul quale viene fatto girare (purchè contenga l'ISA su cui il codice è basato!).

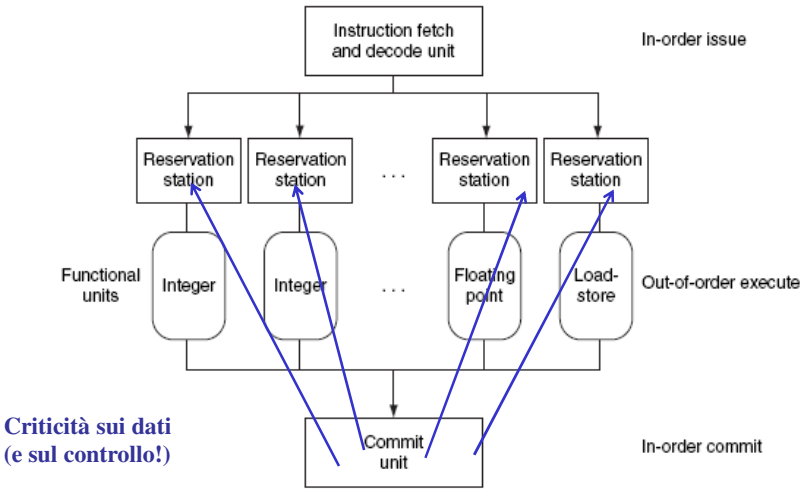
Nelle multiple-issue statiche, spesso occorre ricompilare passando da una CPU ad un'altra oppure il codice può girare con prestazioni molto scadenti. Nelle multiple-issue dinamiche, ciò non è necessario.

La speculazione può essere fatta sia dal compilatore (multiple-issue statici, SW) che dal processore (multiple-issue dinamici, HW).

Ciclo di esecuzione di un'istruzione MIPS



Pipeline con schedulazione dinamica



Esistono diversi cammini paralleli (per la fase di esecuzione e memoria) dell'istruzione, vengono scelti dinamicamente, run-time.

Esempio

```
add  $t0, $t1, $t2
sub  $s0, $s1, $s2
beq  $s3, $s4, salta
or $f5, $f6, $f7
lw $t4, 20($t5)
```

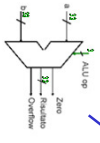
Fetch and decode (scheduling)

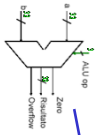
#s1, #s2, 'sub'
#s0
#t1, #t2, 'add'
#t0

#s3, #s4, 'diff'
PC, costante

#f5, #f6, 'or'
#f7

#t5, 20, 'read'
#t4





ALUfp

Memoria

Buffer da cui si trasferisce a memoria o register file

Commit

Principi della schedulazione dinamica

Obbiettivo: mettere in esecuzione istruzioni che non presentino criticità.

Le istruzioni vengono bufferizzate dalla **reservation station**, la quale gestisce la coda delle istruzioni che hanno bisogno della stessa unità funzionale.

Al termine dell'esecuzione la **reorder station**, provvede ad ordinare le istruzioni nella sequenza con la quale devono essere restituite.

Per eseguire un'operazione è sufficiente che il dato sia già pronto nel reorder buffer, senza che sia necessariamente scritto nel register file.

Un'operazione viene lanciata, quando i dati sono pronti. Se un dato non è pronto viene inserita un'etichetta che associa (traccia) il dato al cammino che lo deve produrre. Quando il dato viene eseguito, tramite etichetta si libera il blocco all'esecuzione dell'istruzione.

NB Le istruzioni non sono eseguite sequenzialmente.

La speculazione

Ricorso massiccio alla **speculazione** (= immaginare degli scenari). La speculazione richiede **l'analisi del valore assunto run-time dai registri**.

La speculazione consente di spostare le istruzioni all'interno del codice per rimuovere l'hazard.

Esempio: si può speculare sul risultato di una branch, e quindi inserire nel branch delay slot l'istruzione opportuna.

Esempio: si può speculare che una sw non abbia come indirizzo di memoria lo stesso della lw successiva. In questo caso, si possono scambiare le due istruzioni.

sw $t0, 4($s0)

lw $t1, 16($s0)

Solo se gli indirizzi di memoria sono indipendenti, posso scambiare l'ordine di lettura e scrittura dalla memoria (questo sia che i registri base siano uguali o diversi).

Register renaming e roll-back

Il compilatore può utilizzare la speculazione per riordinare le istruzioni, decidere il nome dei registri della pipeline da associare ai registri del register file (visibili al programmatore).

Register renaming. Posso associare più registri interni ad uno stesso registro dell'architettura in modo da evitare hazard. Esempio:

```
lw $t0, 20($t1)
add $s0, $t0, $t1        hazard che si crea con la parallelizzazione del codice
lw $t0, 24($t1)
add $s1, $t0, $t2
```

Questa dipendenza non ci sarebbe se allocassi alla seconda coppia di istruzioni un registro diverso.

Codice all'interno della pipeline dopo il renaming:

```
lw $t01, 20($t1)
add $s0, $t01, $t1
lw $t02, 24($t1)
add $s1, $t02, $t2
```


Come viene corretta la speculazione nelle multiple-issue

La speculazione può essere fatta sia dal compilatore che dal processore (mediante logica di controllo).

E se la speculazione risulta sbagliata? Deve esistere un meccanismo di correzione (**roll-back**). La speculazione si paga in termini di meccanismi per **controllare** se la speculazione è stata corretta e di **correggerla**.

Nelle multiple-issue statiche, il compilatore inserisce delle istruzioni di controllo e di correzione a speculazioni errate, anche chiamando procedure opportune che correggono quanto fatto (e.g. Procedure che eseguono le operazioni inverse erroneamente eseguite).

Nell multiple-issue dinamiche, i buffer collezionano i risultati che vengono scritti nel register file solamente quando la speculazione è stata verificata come corretta.

Occorre speculare quando si hanno degli elementi validi, altrimenti si possono inserire problemi (vedi eccezioni generate dall'esecuzione di un'istruzione sbagliata o con dati sbagliati, eccezioni "speculative") che rendono il funzionamento meno efficiente.

Renaming e hazard

- La CPU mette in buffer i risultati dell'esecuzione fino a quando non si è potuto verificare la correttezza della speculazione (esecuzione condizionata).
- Nel caso di speculazione errata, la cancellazione del lavoro fatto viene ottenuta semplicemente svuotando i buffer e correggendo la sequenza di istruzioni (meccanismo di **roll-back**).
- Nel caso in cui l'esecuzione sia corretta, il risultato viene copiato in memoria dati e/o nel register file. Nell'esempio precedente: $t0 = $t02, viene copiato il valore più recente di $t0.
- Il register renaming può essere utilizzato anche per la gestione degli hazard => invece di correggere il register file è sufficiente cancellare l'associazione registro interno – registro del register file.
- Ampliamento del numero dei registri. Limitazione dello spilling dei registri.

Sommario

Superpipeline

Multiple-Issue Statici

Multiple-Issue Dinamici

Alcune pipeline

Tendenze di sviluppo delle pipeline

- Il costo della speculazione è cresciuto fino alla barriera dell'energia => multi-core.
- Le dipendenze sono difficili da eliminare => è difficile riempire tutti gli slot di esecuzione.
- Alimentazione da parte del sistema di memoria => è difficile avere sempre i dati pronti.

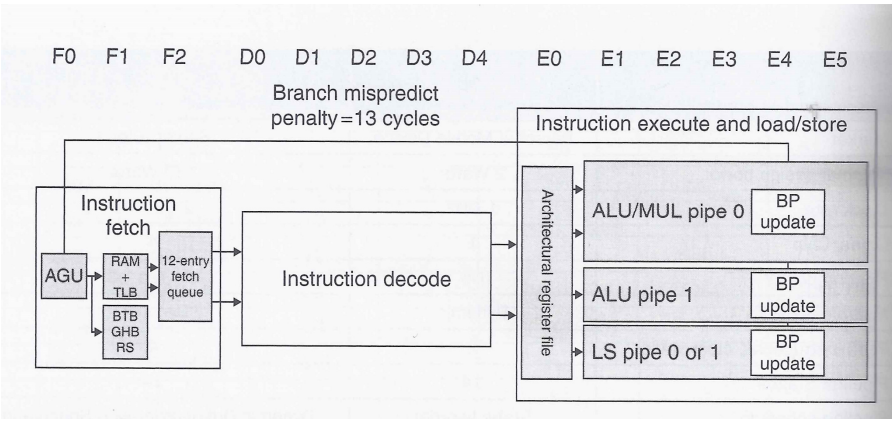
Roof model per i muti-core.

Micro processore	Anno	Freq clock	Stadi pipeline	Larghezza pacchetto	Fuori-ordine / Specuazione	Core / chip	Potenza
Intel 486	1989	25 MHz	5	1	No	1	5 W
Intel Pentium	1993	66 MHz	5	2	No	1	10 W
Intel PentiumPro	1997	200 MHz	10	3	Sì	1	29 W
Intel Pentium 4 Willamette	2001	2 GHz	22	3	Sì	1	75 W
Intel Pentium 4 Prescott	2004	3,6 GHz	31	3	Sì	1	103 W
Intel Core	2006	2,93 GHz	14	4	Sì	2	75 W
Intel Core i5 Nehalem	2010	1,95 GHz	14	4	Sì	1	87 W
Intel Core i5 Ivy Bridge	2012	1,2 GHz	14	4	Sì	8	77 W

ARM Cortex-A8 e Intel Core i7 920

Processore	ARM 8	Intel Core i7
Mercato	Dispositivi mobili	Server, Cloud
Obbiettivi di consumo	2 W	130 W
Frequenza di clock	1 GHz	2,66 GHz
Core / chip	1	4
Virgola mobile?	No	Sì
Multiple-issue	Dinamico	Dinamico
ICP di picco	2	4
Stadi pipeline	14	14
Scheduling pipeline	Statico In-order	Dinamico Out-of-order con Speculazione
Predizione salti	A 2 livelli	A 2 livelli
Cache 1° livello	32 KB I; 32 KB, D	32 KB I; 32 KB, D
Cache 2° livello	128 – 1024 KB	256 KB
Cache 3° livello	-	2-8 MByte

Arm-8 Pipeline

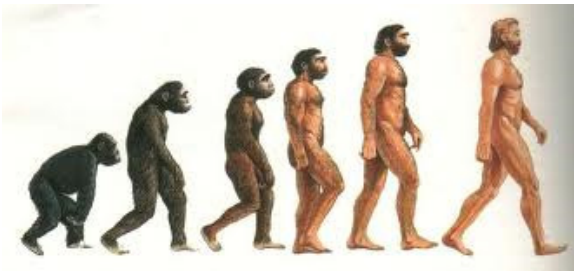


- Static in-order pipeline. 3 Main phases.
- 12 Instruction queue buffer
- Fetch 2 instructions / clock
- 13 clock cycles for branch misprediction
- Dependencies are evaluate in Instruction decode and forces sequential execution and sends to the proper execution queue
- Forwarding in the exercution stage

A..

Evoluzione degli Intel

La compatibilità backwards ha costretto la ricerca di soluzioni innovative.



Dall'8080 al 386, al Pentium Pro, ai multi-core...

Le pipeline Intel

Ultimo processore Intel senza pipeline: 386, 1985.

Esecuzione multi-ciclo:
Durata diversa per istruzioni molto diverse (cf. `MOVS`).
Riutilizzo di unità funzionali in diversi passi di esecuzione.

UC cablata per le istruzioni semplici e microprogrammata per le istruzioni più complesse.

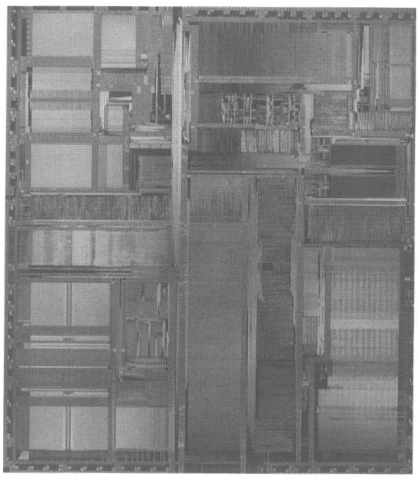
Pentium 4: Superpipeline superscalare. Fino a 3 istruzioni per ciclo di clock.

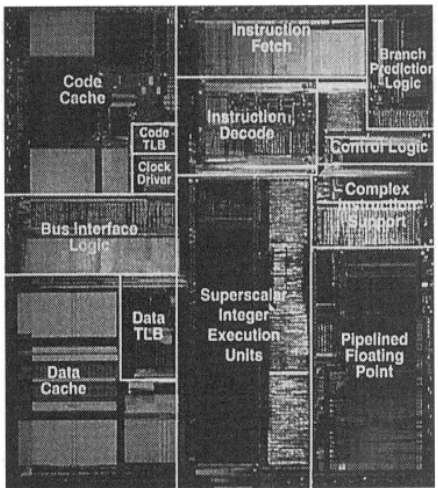
All'interno della pipeline, abbiamo microistruzioni di ampiezza pari a 70 bit (fissa), RISC.

A partire dal codice operativo, vengono generati i segnali di controllo: 120 per le ALU intere, 275 per le unità ALUfp e 400 per le istruzioni SS2.

Trasformo le istruzioni dell'ISA Intel in microistruzioni di lunghezza uguale.

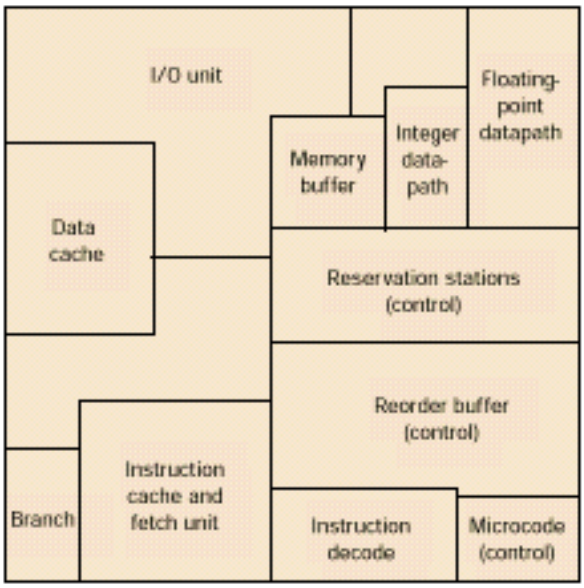
Il pentium



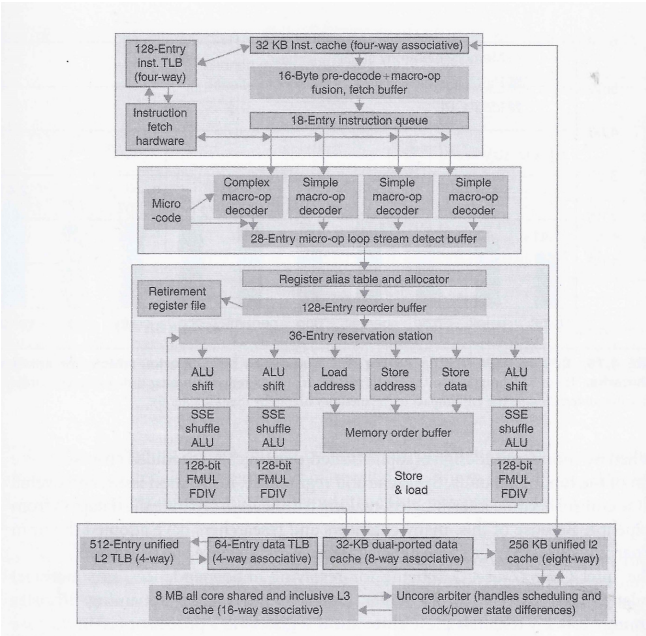


Il processore Pentium Pro

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



Core i7 Pipeline



Le fasi di esecuzione

- Fetch di istruzioni Intel a trasformazione in micro-istruzioni RISC.
- A **micro-architettura** è implementata nella CPU.
- Renaming and speculation
- Out-of-order exection

1) Fetch
- Lettura di gruppi di 16 Byte.
- Speculazione sui salti condizionati.

2) Decodifica
- Identificazione delle istruzioni Intel
- Conversione in micro-operazioni RISC => micro-op buffer (28 micro-op)
- Controllo della presenza di cicli brevi (< 28 istruzioni)
- Recupero degli operandi e allocazione dello spazio del reorder buffer per il risultato.

3) Esecuzione
- 6 code di esecuzione con stazione di prenotazione centralizzata (36 elementi)
- Esecuzione dell'operazione, invio dei risultati alle reservation stations a al riorder buffer

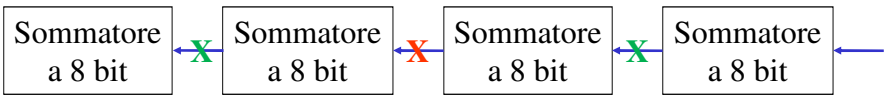
4) Write back
- Scrittra nei registri e nella memoria.

Modalità di calcolo vettoriale

- Operazioni sui vettori richiedono la stessa operazione su elementi adiacenti (multi-media, grafica, calcolo strutturale…)

- Vettore di dati + comando operazione
- Vettore di elementi HW che eseguono l'operazione su ogni coppia di elementi

- Struttura modulare flessibile: 1 somma a 32 bit, 2 somme a 16 bit, 4 somme a 8 bit…
- Le parole avviate in esecuzione sono su 128 / 256 bit.



Come sfruttare il parallelismo a livello di parola

Estensioni MMX e SSE:

Trasferimento dati	Aritmetica	Comparazione
MOV{A/U}{SS/PS/SD/PD} xmm, mem/xmm	ADD {SS/PS/SD/PD} xmm, mem/xmm	CMP {SS/PS/SD/PD}
	SUB {SS/PS/SD/PD} xmm, mem/xmm	
MOV {H/L} {PS/PD} xmm, mem/xmm	MUL {SS/PS/SD/PD} xmm, mem/xmm	
MOV {H/L} {PS/PD} xmm, mem/xmm	DIV {SS/PS/SD/PD} xmm, mem/xmm	
	SQRT {SS/PS/SD/PD} xmm, mem/xmm	
	MAX {SS/PS/SD/PD} xmm, mem/xmm	
	MIN {SS/PS/SD/PD} xmm, mem/xmm	

Different data quantities can be inserted into an xmm register (128 bit):
SS – Scalar, Signle precision FP: 1 operand on 32 bit
PS – Packed Single precision FP: 4 operands on 32 bit
SD – Scalar, Double precision FP, 1 operand on 64 bit
PD – Packed Double precision FP, 2 operands on 64 bit
A – 128 bit aligned in memory

Sub-words vector operation

Single operation specifies more data.

```
#include <x86intrin.h>
addpd %xmm0, %xmm4          # Multiply two 64 bits FP variables in registers xmm
```

In 2011 ***Advanced Vector Exentsion*** (AVX) has been provided by Intel, with registers of 256 bit (internal registers ymm).

```
#include <x86intrin.h>
vaddpd %ymm0, %ymm4          # Multiply four 64 bits FP variables
```

It supports also operations on three registers.

Efficient use when FP adjacent in memory are loaded into registers.

Sommario

Superpipeline
Speculazione
Multiple-Issue Statici
Multiple-Issue Dinamici
Alcune pipeline