Aumentare il parallelismo a livello di istruzione (2)

Architetture Avanzate dei Calcolatori

Valeria Cardellini

Processori multiple-issue

- Nei processori multiple-issue vengono lanciate più istruzioni in parallelo per ciclo di clock
  - L'hardware determina il massimo numero di istruzioni (tipicamente da 2 a 6)
- Due tipi di processori multiple-issue: dinamici e statici
- Processori multiple-issue dinamici (*superscalari*)
  - Il numero di istruzioni lanciate per ciclo di clock (dimensione della finestra di issue) è *variabile*
  - Scheduling *dinamico* (eseguito dall'hardware) o *statico* (eseguito dal compilatore)
  - I processori superscalari con scheduling dinamico eseguono le istruzioni fuori ordine
    - Esempi: IBM Power4, Intel Pentium 3,4
  - I processori superscalari con scheduling statico eseguono le istruzioni in ordine
    - Esempi: SUN UltraSPARC

Processori multiple-issue statici

- Il numero di istruzioni lanciate insieme in un ciclo di clock è tipicamente ***fisso***
- Il compilatore identifica le istruzioni eseguibili in parallelo
  - Istruzioni assemblate in pacchetti (*pacchetto di issue*), che sono poi decodificati ed eseguiti dal processore
- Il compilatore gestisce le criticità

- *Motivazione*:
  - Nei processori multiple-issue dinamici il controllo occupa il 30-50% di tempo del processore
  - Affidando al compilatore l'ottimizzazione della sequenza di istruzioni:
    - Si riduce la complessità del processore
    - Si libera spazio sul chip

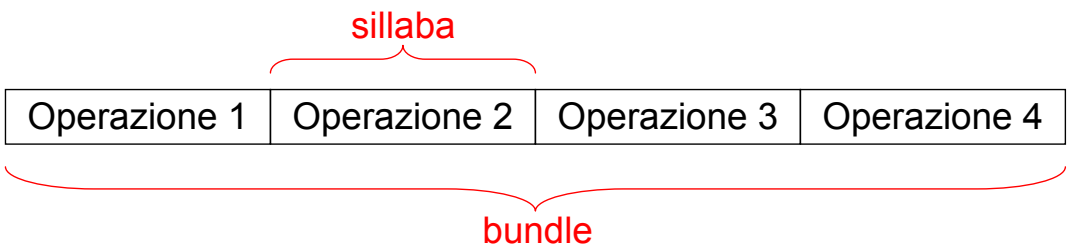
Processori multiple-issue statici (2)

- Processori **VLIW** (Very Long Instruction Word)
  - Istruzione lunga in cui si impacchettano più istruzioni elementari
  - Istruzioni lunghe eseguite in *ordine strettamente sequenziale*
  - Esempi:
    - Philips Trimedia (processore embedded per media processing, 5 operazioni per ciclo di clock)
    - Transmeta Crusoe (istruzione lunga di 128 bit, max 4 istruzioni per ciclo di clock)
  - Approccio VLIW puro: rigidità eccessiva → EPIC

- Processori **EPIC** (Explicitly Parallel Instruction Computer)
  - Intel Itanium, Itanium 2

VLIW: concetti base

- Il pacchetto di issue rappresenta un'istruzione lunga (ad es. 128 bit)
- Nell'istruzione lunga (detta *bundle*) si impacchettano più operazioni (dette *sillabe*), con i relativi operandi, che saranno eseguite in parallelo da altrettante unità funzionali
- Ad ogni operazione viene associato un percorso nell'unità di elaborazione, da cui viene eseguita l'operazione
- Esempio di bundle con 4 sillabe:



VLIW: schema di funzionamento

- *Ipotesi*: istruzione lunga composta da 4 operazioni
- Il compilatore identifica 4 operazioni *mutuamente indipendenti*, che possono essere impacchettate nello stesso bundle ed eseguite simultaneamente
- Criticità sui dati e strutturali: risolte dal compilatore selezionando opportunamente diversi slot temporali per svolgere le operazioni
- Predizione dei salta di tipo statico (usando anche informazioni di tipo statistico) ad opera del compilatore
- Al più una operazione che modifica il flusso di controllo per bundle
  - Predizione errata: viene interrotta l'esecuzione delle operazioni coinvolte

VLIW: pro e contro

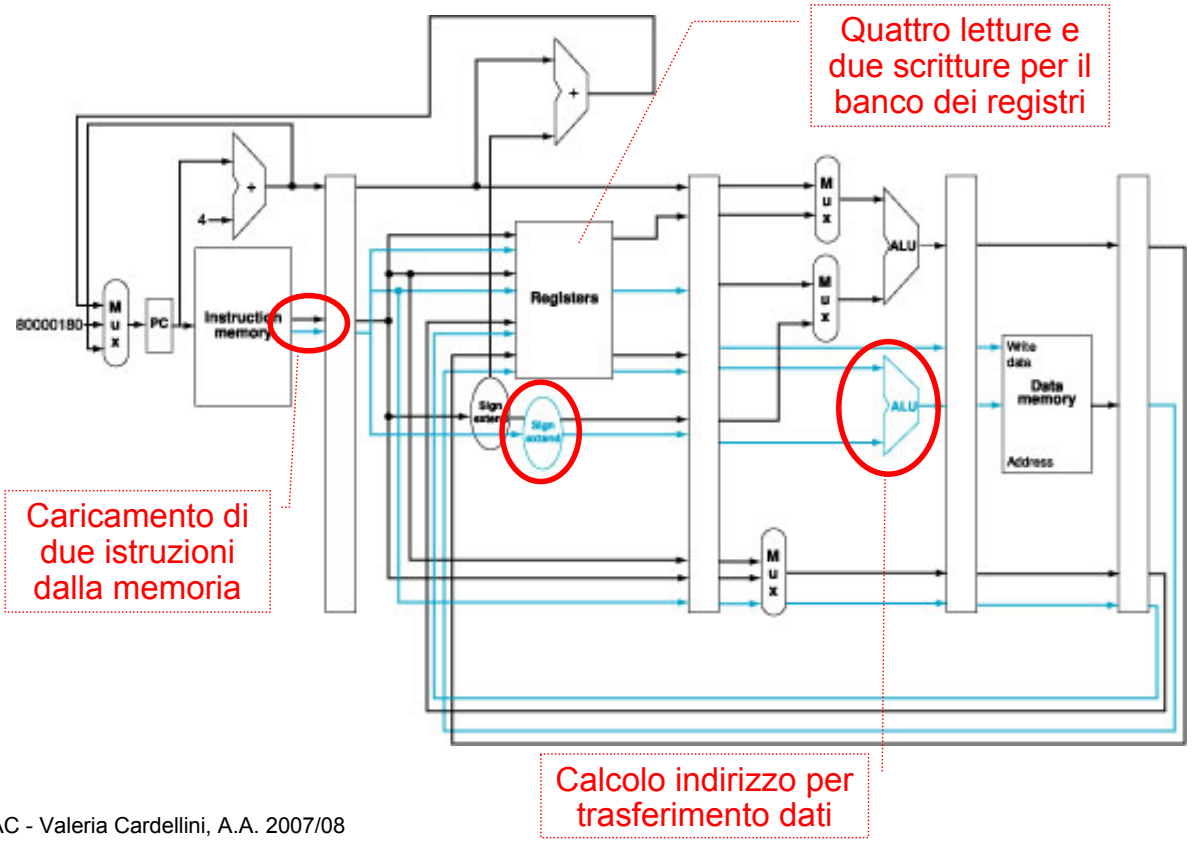
- *Vantaggio*: architettura più semplice rispetto a quella di un processore superscalare
  - Più veloce (ciclo di clock più breve)
  - Riduzione dell'area del chip
  - Minori costi e minor consumo di potenza
- *Svantaggi*: complessità del compilatore e riduzione della portabilità
  - Il compilatore deve conoscere non solo l'architettura del processore, ma anche parametri tecnologici
  - Si riduce la portabilità del codice oggetto, anche tra successive generazioni della stessa famiglia (manca la *compatibilità binaria*)
  - Soluzione possibile per il problema della compatibilità binaria: *emulazione*
    - Traduzione del codice oggetto da parte dell'architettura obiettivo
    - Es.: Code Morphing nel Transmeta Crusoe

Esempio: MIPS con two-issue

- Il pacchetto di issue è composto da 2 istruzioni (dimensione complessiva pari a 64 bit)
  - Nel pacchetto: un'istruzione è di tipo ALU su interi o un branch, l'altra istruzione è una load o store

Istruzione	Stadi della pipeline							
ALU o branch	IF	ID	EX	MEM	WB			
Load o store	IF	ID	EX	MEM	WB			
ALU o branch		IF	ID	EX	MEM	WB		
Load o store		IF	ID	EX	MEM	WB		
ALU o branch			IF	ID	EX	MEM	WB	
Load o store			IF	ID	EX	MEM	WB	
ALU o branch				IF	ID	EX	MEM	WB
Load o store				IF	ID	EX	MEM	WB

Esempio: MIPS con two-issue (2)



Processori VLIW e criticità

- Criticità sui dati di tipo RAW
  - Il compilatore inserisce staticamente istruzioni **nop** o altre istruzioni
- Criticità sui dati di tipo WAR e WAW
  - Generalmente risolte dal compilatore selezionando opportunamente gli slot temporali per le operazioni
- Criticità strutturali
  - Risolte dal compilatore selezionando opportunamente gli slot temporali per le operazioni
  - Operazioni nello stesso bundle devono utilizzare risorse distinte
- Criticità sul controllo
  - Predizione statica dei salti ad opera del compilatore
  - Risolte dall'hardware annullando l'esecuzione di salti predetti erroneamente

Esempio di scheduling per VLIW

- Sia dato il seguente codice scalare

```
Loop:   lw $t0, 0($s1)
        add $t0, $t0, $s2
        sw $t0, 0($s1)
        addi $s1, $s1, -4
        bne $s1, $zero, Loop
```

Potenziale WAR su $t0

Potenziale WAW su $t0

In rosso criticità RAW

- Indicare il possibile scheduling sulla pipeline MIPS di tipo statica two-issue
- Assunzione: predizione dei salti

Esempio di scheduling per VLIW (2)

- Assumendo che la prima operazione dell'istruzione lunga è relativa ad un'istruzione ALU o branch, mentre la seconda è per istruzioni di accesso in memoria:

	Istruzione ALU o branch	Istruzione load o store	Ciclo di clock
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1, -4	nop	2
	add $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- Si ha

$CPI_{ideale} = 1/2 = 0.5$

$CPI_{two\text{-}issue} = 4/5 = 0.8$

$CPI_{scalare} = 7/5 = 1.4$ ⇐ Necessari due stalli per criticità RAW (tra lw e add, tra addi e bne)

Inserimento di istruzioni nop

- Se il parallelismo non è elevato, le istruzioni lunghe contengono un numero elevato di sillabe nop
  - Problema: esplosione della memoria
- Per superare il problema dell'esplosione della memoria: in memoria i bundle sono impacchettati
  - Contengono solo le sillabe significativa (nessuna nop)
  - Codici separatori indicano quali sillabe appartengono ad un bundle e quali al bundle successivo

Scheduling del codice

- Obiettivo: riordinare le istruzioni del codice oggetto in modo tale che siano eseguite nell'ordine ottimo e semanticamente corretto
  - Eseguire le operazioni critiche in modo efficiente
  - Aumentare il numero di istruzioni significative (no nop) lanciate simultaneamente
  - Con lo scopo di minimizzare il tempo di esecuzione del programma
- Tecniche di ***scheduling locale***
  - Lavorano su una sequenza di istruzioni appartenenti ad un *singolo* blocco di base
    - Il blocco di base non contiene salti e destinazioni di salti
  - Esempi: loop unrolling, software pipelining
- Tecniche di ***scheduling globale***
  - Lavorano su una sequenza di istruzioni appartenenti a *multipli* blocchi di base
  - Esempi (*non trattati*): trace scheduling, superblock scheduling

Srotolamento del ciclo

- Lo srotolamento del ciclo (*loop unrolling*) modifica la struttura del ciclo
  - Il corpo del ciclo è replicato più volte (per un numero di volte pari al *fattore di srotolamento*), modificando il codice di terminazione del ciclo
  - Il compilatore deve verificare se le iterazioni del ciclo sono tra loro indipendenti
- Vantaggi
  - Aumenta il grado di ILP
  - Riduce l'overhead del ciclo
    - Numero di incrementi del contatore e di salti
  - Aumenta la lunghezza di un blocco di base
    - Più istruzioni disponibili per lo scheduling → diminuisce il numero di stalli da inserire
- Svantaggi
  - Aumenta il numero di registri necessari
  - Aumenta la lunghezza del codice

Srotolamento del ciclo: esempio

```
for (i=1000; i>0; i=i-1)
        x[i] = x[i] + s;
```

← Le iterazioni del ciclo sono *indipendenti*

- Consideriamo il codice con una singola iterazione

  x è un vettore di interi, s è un intero

  $s2 contiene s, $t0 è inizializzato con l'indirizzo più alto degli elementi del vettore

```
Loop:   lw $t0, 0($s1)
        add $t0, $t0, $s2
        sw $t0, 0($s1)
        addi $s1, $s1, -4
        bne $s1, $zero, Loop
```

Srotolamento del ciclo: esempio (2)

- Senza riordinamento (scheduling), il ciclo è eseguito dal processore MIPS con pipeline così:

```
Loop:   lw $t0, 0($s1)
        nop
        add $t0, $t0, $s2
        sw $t0, 0($s1)
        addi $s1, $s1, -4
        nop
        bne $s1, $zero, Loop
        nop
```

Confronto nello stadio ID

- Prestazioni: 8 cicli di clock per iterazione

Srotolamento del ciclo: esempio (3)

- Riordinamento (scheduling) sulla singola iterazione

```
Loop:   lw $t0, 0($s1)
        addi $s1, $s1, -4
        add $t0, $t0, $s2
        bne $s1, $zero, Loop
        sw $t0, 4($s1)
```

- Prestazioni: 5 cicli di clock per iterazione
  - 3 cicli di clock per operazioni su vettore (lw, add, sw)
  - 2 cicli di clock per overhead del loop (addi, bne)

Srotolamento del ciclo: esempio (4)

- Srotolando il ciclo per 4 volte
  - Fattore di srotolamento = 4

```
for (i=1000; i>0: i=i-4) {
        x[i] = x[i] + s;
        x[i-1] = x[i-1] + s;
        x[i-2] = x[i-2] + s;
        x[i-3] = x[i-3] + s;
}
```

```
Loop:  lw $t0, 0($s1)      it. i
       add $t0, $t0, $s2
       sw $t0, 0($s1)
       lw $t0, -4($s1)     it. i-1
       add $t0, $t0, $s2
       sw $t0, -4($s1)
       lw $t0, -8($s1)     it. i-2
       add $t0, $t0, $s2
       sw $t0, -8($s1)
       lw $t0, -12($s1)    it. i-3
       add $t0, $t0, $s2
       sw $t0, -12($s1)
       addi $s1, $s1, -16
       bne $s1, $zero, Loop
```

Fusione della parte di controllo del ciclo

Srotolamento del ciclo: esempio (5)

- Per evitare dipendenze sui nomi il compilatore ridenomina i registri

```
Loop:  lw $t0, 0($s1)
       add $t0, $t0, $s2
       sw $t0, 0($s1)
       lw $t1, -4($s1)
       add $t1, $t1, $s2
       sw $t1, -4($s1)
       lw $t2, -8($s1)
       add $t2, $t2, $s2
       sw $t2, -8($s1)
       lw $t3, -12($s1)
       add $t3, $t3, $s2
       sw $t3, -12($s1)
       addi $s1, $s1, -16
       bne $s1, $zero, Loop
```

Srotolamento del ciclo: esempio (6)

- Riordinamento (scheduling) per evitare criticità sui dati e sul controllo

```
Loop:  lw $t0, 0($s1)
       lw $t1, -4($s1)
       lw $t2, -8($s1)
       lw $t3, -12($s1)
       add $t0, $t0, $s2
       add $t1, $t1, $s2
       add $t2, $t2, $s2
       add $t3, $t3, $s2
       sw $t0, 0($s1)
       sw $t1, -4($s1)
       addi $s1, $s1, -16
       sw $t2, 8($s1)
       bne $s1, $zero, Loop
       sw $t3, 4($s1)
```

- Prestazioni:
  - 14 cicli di clock per 4 iterazioni
  - 12/4 = 3 cicli di clock per operazioni su vettore per 1 iterazione
  - 2 cicli di clock per overhead del loop per 4 iterazioni
  - CPI = 14/14 = 1

Srotolamento del ciclo: esempio (7)

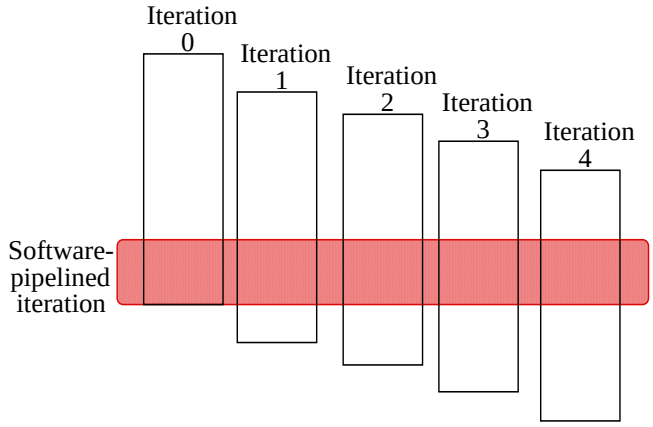
- Scheduling su pipeline MIPS di tipo statico two-issue

	Istruzione ALU o branch	Istruzione load o store	Ciclo di clock
Loop:	addi $s1, $s1, -16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	add $t0, $t0, $s2	lw $t2, 8($s1)	3
	add $t1, $t1, $s2	lw $t3, 4($s1)	4
	add $t2, $t2, $s2	sw $t0, 16($s1)	5
	add $t3, $t3, $s2	sw $t1, 12($s1)	6
	bne $s1, $zero, Loop	sw $t2, 8($s1)	7
	nop	sw $t3, 4($s1)	8

- Prestazioni:
  - 8 cicli per 4 iterazioni
  - 8/4 = 2 cicli per iterazione
  - CPI = 8/14 = 0.57

Software pipelining

- Assumiamo che un ciclo presenti delle istruzioni indipendenti in diverse iterazioni (evidenziate in rosso)



- E' possibile riorganizzare il ciclo in un nuovo ciclo tale che ciascuna nuova iterazione esegua istruzioni appartenenti ad iterazioni diverse del ciclo originale

Software pipelining: esempio

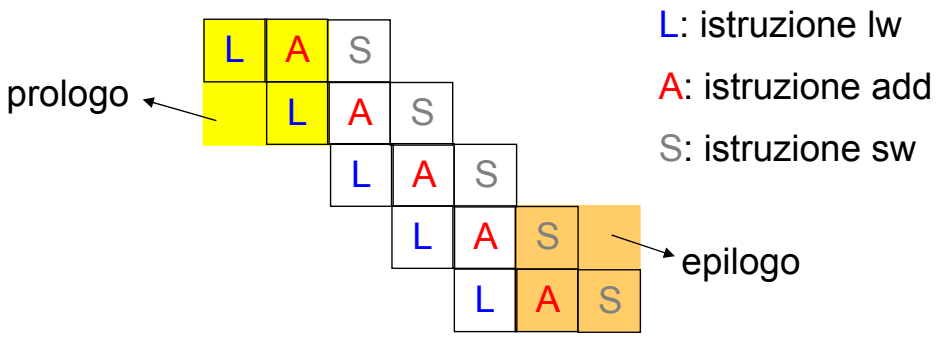
```
Loop:   lw $t0, 0($s1)
        add $t4, $t0, $s2
        sw $t4, 0($s1)
        addi $s1, $s1, -4
        bne $s1, $zero, Loop
```

```
it. i       lw $t0, 0($s1)
            add $t4, $t0, $s2
            sw $t4, 0($s1)
it. i+1     lw $t0, 0($s1)
            add $t4, $t0, $s2
            sw $t4, 0($s1)
it. i+2     lw $t0, 0($s1)
            add $t4, $t0, $s2
            sw $t4, 0($s1)
```

```
Loop:   sw $t4, 8($s1)
        add $t4, $t0, $s2
        lw $t0, 0($s1)
        addi $s1, $s1, -4
        bne $s1, $zero, Loop
```

Software pipelining: esempio (2)

- Analogia con il pipelining nel processore:



Prologo (prima del loop):

```
lw $t0, 0($s1)
    add $t4, $t0, $s2
    lw $t0, -4($s1)
    addi $s1, $s1, -8
```

Epilogo (dopo il loop):

```
sw $t0, 4($s1)
    add $t4, $t0, $s2
    sw $t0, 0($s1)
```

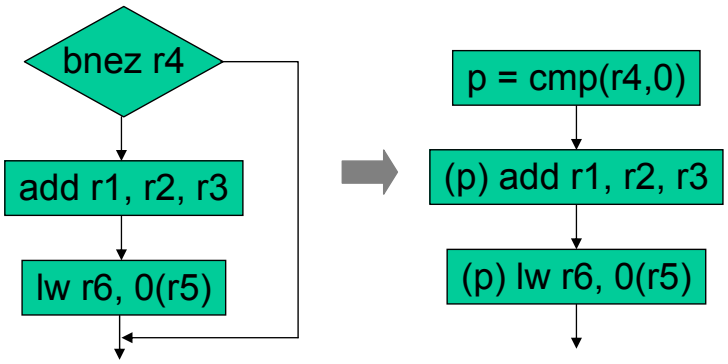

Speculazione del compilatore

- Tecniche come il loop unrolling ed il software pipelining possono essere usate per aumentare l'ILP quando il comportamento dei salti è facilmente predicibile a tempo di compilazione
- Per evitare che le dipendenze sul controllo limitino l'ILP, è possibile estendere l'insieme di istruzioni per includere istruzioni *predicate* o condizionali
  - **Esecuzione predicata** o **condizionale**
- L'esecuzione predicata è un insieme di tecniche che convertono le dipendenze di controllo in dipendenze di dati
  - Il compilatore ha un'alternativa ai salti condizionati
- L'esecuzione predicata implica una forma di esecuzione condizionale di istruzioni basata su una guardia booleana

Esecuzione predicata

- L'istruzione fa riferimento ad una condizione, che viene valutata come parte dell'esecuzione dell'istruzione
  - Condizione vera: l'istruzione viene eseguita normalmente
  - Condizione falsa: l'istruzione continua come se fosse nop
- Istruzione predicata:

  (p) op rd, r1, r2
- Esempio: *conversione if*



VLIW vs EPIC

- Approccio VLIW (Very Long Instruction Word)
  - Formato fisso dei campi delle operazioni nel bundle
  - Parallelismo implicito tra operazioni nel bundle
- Approccio EPIC (Explicitly Parallel instruction Computer)
  - Variante dell'approccio VLIW
    - Seconda generazione VLIW?
  - Maggiore flessibilità nel formato dell'istruzione
  - Il compilatore indica quando un'istruzione non può essere eseguita in parallelo con i suoi successori

Esempi di architetture multiple-issue

La soluzione Itanium

- Obiettivo: spostare la complessità fuori dalla logica del processore e portarla dentro il compilatore
- L'Itanium (Itanium I e II) è il primo processore Intel a 64 bit (IA-64); basato sull'approccio EPIC
- Incompatibile con IA-32
  - Soluzione: nei chip Itanium un emulatore hardware che traduce le istruzioni IA-32 in istruzioni VLIW
- Elevato numero di registri
  - 128 registri interi a 64 bit, 128 registri in virgola mobile a 82 bit, 8 registri speciali per branch, 64 registri di condizione ad 1 bit
- Pipeline di larghezza 6, con profondità 10
- Risorse funzionali
  - 4 unità su interi, 4 unità su interi multimediali (MMX), 2 unità load/store, 3 unità di salto, 4 unità floating point

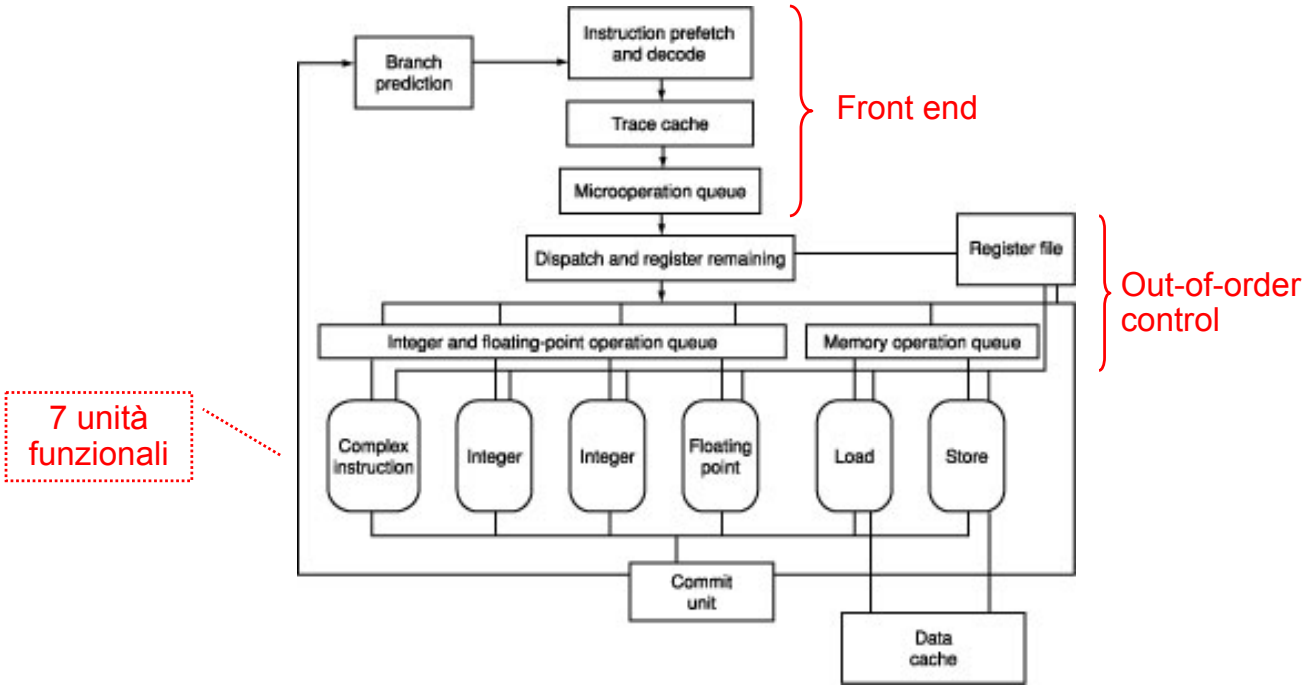
La soluzione Itanium (2)

- Il compilatore ottimizza a livello globale
  - Supporto hw per ottimizzazioni dinamiche a run time, così da garantire massimo throughput
- Ampio uso di speculazione ed esecuzione predicata
  - Es.: istruzione di load avanzata, con un supporto per controllare la presenza di alias che possono invalidare la load speculativa
- *Gruppo di istruzioni*
  - Sequenza di istruzioni consecutive senza dipendenze sui dati nei registri
  - Istruzioni nel gruppo eseguibili in parallelo
    - Se esistono sufficienti risorse funzionali e se sono preservate le dipendenze in memoria
  - Lunghezza arbitraria del gruppo di istruzioni
    - Il compilatore deve indicare il confine tra due gruppi consecutivi (stop)
- Le istruzioni IA-64 sono raggruppate in *bundle* di 128 bit
  - Ciascun bundle contiene un campo *template* (di 5 bit) e 3 istruzioni (da 41 bit ciascuna)
    - Il template specifica le risorse funzionali necessarie all'esecuzione delle istruzioni nel bundle

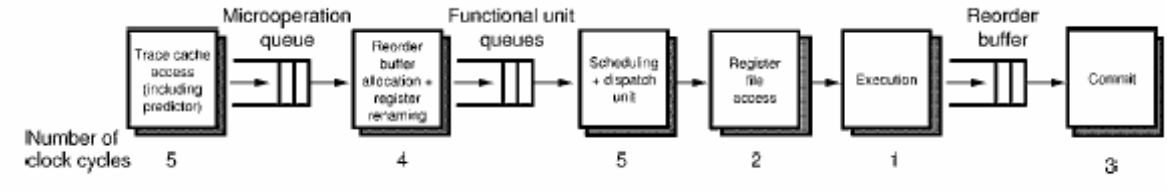
La soluzione Pentium 4

- Nella micro-architettura del Pentium 4 (*NetBurst*) le istruzioni IA-32 vengono tradotte in microistruzioni (simili ad istruzioni RISC), dette *micro-operazioni* nella terminologia Intel
  - Uso di *trace cache* (una cache istruzioni) per mantenere memorizzate le micro-operazioni che implementano una data istruzione IA-32
    - Una volta decodificata l'istruzione IA-32, non sarà più necessario ritradurla in micro-operazioni, ma si potranno ottenere le micro-operazioni ad essa corrispondenti direttamente dalla trace cache
    - Nel caso in cui di istruzioni IA-32 complesse (che richiedono più di 3 micro-operazioni), viene usata una ROM
  - Le micro-operazioni sono eseguite da una pipeline sofisticata, con scheduling dinamico e speculazione
  - Esecuzione di 3 micro-operazioni per ciclo di clock
- Soluzioni adottate per massimizzare le prestazioni:
  - Multiple issue dinamico e superpipelining

Micro-architettura del Pentium 4



Pipeline del Pentium 4



- Circa 20 cicli di clock per un'istruzioni tipica
  - Il numero di cicli di clock può variare in base allo scheduling dinamico e al tipo di micro-operazione
- Pipeline molto profonda
  - 20 stadi nei primi core
  - Con il core Prescott (commercializzato dal 2004 fino a fine 2005, frequenza di clock fino a 3,8 GHz) ulteriore aumento del numero di stadi della pipeline (da 20 a 31)

Speculazione nel Pentium 4

- Esecuzione fuori ordine speculativa
  - In grado di vedere nella pipeline fino a 126 istruzioni, di cui 48 load e 24 store
- Predizione dei salti
  - Avviene in due livelli
    - Prima del prefetch e decodifica dell'istruzione IA-32
      - Branch Target Buffer da 4K entry (L1 BTB)
    - Durante l'uso della trace cache (Trace BTB)
      - Branch Target Buffer da 512 entry
  - Predizione con tecniche sia statiche che dinamiche

La soluzione Pentium M

- Processore Intel usato nella tecnologia Centrino
- Perché ha prestazioni confrontabili con quelle del Pentium 4, sebbene operi ad una frequenza di clock inferiore?
- Per quanto riguarda il pipelining…
  - Riduzione del numero di stadi della pipeline
  - Tecniche di predizione dei salti più sofisticate
- L'*Intel Core* (la micro-architettura multi-core di Intel) estende il Pentium M
  - NetBurst sostituita per problemi di sovrariscaldamento e impossibilità di aumentare ulteriormente la frequenza di clock