Architettura degli Elaboratori, 2011-12

Seconda prova di verifica intermedia

30 maggio 2012

Domanda 1

Per una CPU pipeline si descrivano le modalità di funzionamento dell'Unità Istruzioni *out-of-order*.

Successivamente se ne discuta il tipo di supporto firmware richiesto, per differenza rispetto al supporto richiesto per l'implementazione di una Unità Istruzioni *in-order*.

Considerare sia il caso di architettura scalare che superscalare a due vie.

Domanda 2

Si consideri la seguente procedura assembler (opera su una lista unidirezionale; il parametro d'ingresso, in Rtesta, rappresenta l'indirizzo del puntatore di testa; ogni elemento è di due interi + puntatore in avanti; stacca il primo elemento della lista; calcola la moltiplicazione dei due interi e memorizza il risultato in una locazione nota distinta dalla lista):

```
LOAD   Rtesta, 0, Rprimo
ADD    Rprimo, 2, Rnext
STORE  Rtesta, 0, Rnext
LOAD   Rprimo, 0, Rd1
LOAD   Rprimo, 1, Rd2
MUL    Rd1, Rd2, Rd3
STORE  Rrisultato, 0, Rd3
GOTO   Rret
```

Si consideri una architettura D-RISC con cache primaria associativa, Write-Through, con blocchi di 4 parole. La cache secondaria on-chip è assunta avente probabilità di fault trascurabile.

Compilare in forma ottimizzata e valutare il tempo di completamento per una architettura di CPU pipeline scalare e per una superscalare a due vie.

In entrambi i casi, valutare l'efficienza relativa complessiva (tenendo conto tanto delle degradazioni dovute alla struttura pipeline quanto delle degradazioni dovute alla cache).

Traccia di soluzione

(da espandere opportunamente)

Domanda 1

Spiegare prima il meccanismo di sincronizzazione per la consistenza dei registri generali.

Nel caso che una istruzione sia bloccata su un registro Ri, viene memorizzata internamente (sia una memorietta WAIT) e, nella locazione i-esima delle memoria SEM dei semafori, viene messo un riferimento (SEM[i].rif) alla locazione di WAIT contenente l'istruzione in attesa di Ri. Quando Ri diviene aggiornato (semaforo uguale a zero; il controllo viene effettuato ad ogni ciclo di clock) viene sbloccata ed eseguita l'istruzione in WAIT puntata da SEM[i].rif.

Per out-of-order FIFO, se una istruzione si blocca, si considerano e si eseguono le istruzioni successive provenienti da IM finché non se ne trova una a sua volta bloccata. Per out-of-order generale, si esamina in modo associativo un certo numero (determinato) di istruzioni in coda da IM per scoprire eventuali istruzioni abilitate.

Passando dal caso scalare al caso superscalare, il meccanismo rimane quello descritto in quanto applicato separatamente alle istruzioni scalari presenti nelle istruzioni lunghe. Ovviamente, tutti i controlli di abilitazione vengono svolti in parallelo sulle due istruzioni scalari di una stessa istruzione lunga.

Domanda 2

1. LOAD Rtesta, 0, Rprimo
2. ADD Rprimo, 2, Rnext
3. STORE Rtesta, 0, Rnext
4. LOAD Rprimo, 0, Rd1
5. LOAD Rprimo, 1, Rd2
6. MUL Rd1, Rd2, Rd3
7. STORE Rrisultato, 0, Rd3
8. GOTO Rret

Nel codice dato esistono le dipendenze logiche della 2 sulla 3 e della 6 sulla 7, oltre alla degradazione dovuta alla 8.

Un possibile codice scalare ottimizzato è:

1. LOAD Rtesta, 0, Rprimo
2. ADD Rprimo, 2, Rnext
3. LOAD Rprimo, 0, Rd1
4. LOAD Rprimo, 1, Rd2
5. MUL Rd1, Rd2, Rd3
6. STORE Rtesta, 0, Rnext
7. GOTO Rret, delayed_branch
8. STORE Rrisultato, 0, Rd3

Ora si ha la dipendenza della 1 sulla 3 (nuova: k = 2, NQ = 2) e della 5 sulla 8 (allontanate: k = 3, NQ = 2, Lpipe-k = 4), mentre non hanno effetto la dipendenza della 2 sulla 6 e il salto.

Applicando il modello dei costi, in assenza di degradazioni dovute alla cache:

$$\Delta = \Delta_1 = 5t/8 \qquad T = 13t/8 \qquad T_{c\text{-}1} = 13\ t = 26\tau$$

Il codice superscalate VLIW, a partire da quello scalare ottimizzato, è:

1-x. LOAD Rprimo | NOP

2-3. ADD .. Rnext | LOAD Rprimo …

4-x. LOAD Rprimo … | NOP

5-6. MUL … Rd3 | STORE … Rnext

7-8. GOTO … | STORE … Rd3

con le stesse dipendenze logiche del caso scalare ma distanze diverse. Applicando il modello dei costi con $\theta = 2/8$, si ha:

$$\Delta = \Delta_1 = 4t/8 \qquad T = 12t/8 \qquad T_{c\text{-}1} = 12\ t = 24\tau$$

osservando che le stesse prestazioni si ottengono anche senza applicare il delayed branch.

Data le sequenzialità insita in questa procedura, l'architettura superscalare permette un guadagno piccolo; in pratica, solo il parallelismo nella 2-3 comporta un miglioramento.

Nei due casi il tempo di completamento ideale è:

$$T_{scalare\text{-}id} = 8\ t = 16\ \tau$$

$$T_{supercalare\text{-}id} = 8\ t/2 = 8\ \tau$$

Considerando l'effetto della cache, si hanno due fault per il caricamento del codice, un fault per la lettura della struttura di testa, un fault per la lettura del primo elemento, ed un fault per la scrittura del risultato. Quest'ultimo non ha effetto sulle prestazioni se l'architettura adotta l'ottimizzazione che i blocchi in solo scrittura non sono trasferiti prima di essere usati. Nell'ipotesi data sulla cache secondaria, la degradazione dovuta ai fault di cache si misura come:

$$T_{fault} = 4\ T_{trasf} = 32\ \tau$$

La procedura, avente solo località su quattro blocchi non consecutivi, è anche caratterizzata da una carente utilizzazione della cache.

Quindi:

$$T_{scalare} = T_{scalare\text{-}id} + T_{fault} = 58\ \tau$$

$$T_{superscalare} = T_{superscalare\text{-}id} + T_{fault} = 56\ \tau$$

$$\varepsilon_{scalare} = \frac{T_{scalare-id}}{T_{scalare}} \sim 0{,}28$$

$$\varepsilon_{scalare} = \frac{T_{superscalare-id}}{T_{superscalare}} \sim 0{,}14$$