Architetture degli Elaboratori - A. A. 2009-2010
Seconda prova di verifica intermedia

Riportare su tutti i fogli consegnati Nome, Cognome, numero di matricola e corso di appartenenza.
I risultati verranno pubblicati sulle pagine web del corso/dei docenti appena disponibili.

Domanda 1

Si consideri il seguente frammento di codice assembler eseguito sull'architettura pipeline D-RISC:

```
        MOV    R0, Ri
        MOV    Rn, Rj
loop:   LOAD   RbaseA, Ri, R1
        LOAD   RbaseB, Ri, R2
        ADD    R3, R2, R3
        SUB    R1, R2, R1
        IF>    R1, R0, cont
        DECR   Rj
        STORE  RbaseC, Rj, R1
cont:   INCR   Ri
        IF<    Ri, Rn, loop
```

Se ne valutino possibili ottimizzazioni fornendone la valutazione del tempo di completamento (utilizzando il metodo analitico) in assenza di fault di cache e assumendo che N sia noto e che il salto **IF> R1,R0,cont** sia preso con probabilità p. Si discutano le differenze delle prestazioni rispetto alla versione non ottimizzata.

Si assuma poi che la cache istruzioni sia ad accesso diretto e operi su domanda (1K blocchi, σ=8) e che la cache dati sia associativa su insiemi (1K insiemi, 4 blocchi per insieme, σ=8) e si determini working set e numero di fault del codice ottimizzato.

Domanda 2

1) Si descrivano le diverse fasi di trattamento di un'interruzione mettendo in evidenza, in particolare, quali strutture dati e/o parti di codice vengono accedute/eseguite e di quali spazi di indirizzamento logici fanno parte.

2) Si consideri un codice assembler, per un processore D-RISC pipeline, con una dipendenza logica di distanza uno (k=1). Si supponga di riuscire a spostare un'istruzione successiva a quella che induce la dipendenza in modo che la dipendenza stessa si trasformi in una dipendenza di distanza due (k=2). Si discutano i possibili effetti dello spostamento.

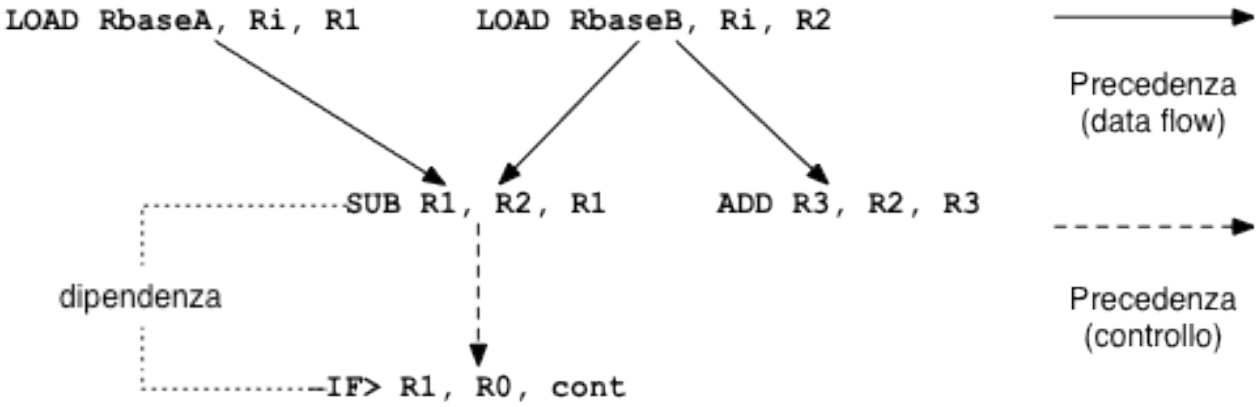
Domanda 1

Il codice è chiaramente risultato della compilazione di un loop. Consideriamo solo le istruzioni del ciclo interno (tralasciando quindi le due MOV iniziali).

```
1.loop:LOAD RbaseA, Ri, R1
2.     LOAD RbaseB, Ri, R2
3.     ADD R3, R2, R3
4.     SUB R1, R2, R1
5.     IF> R1, R0, cont
6.     DEC Rj
7.     STORE RbaseC, Rj, R1
8.cont:INC Ri
9.     IF< Ri, Rn, loop
```

Abbiamo 3 dipendenze logiche (k=1) fra la 4 e la 5, la 6 e la 7 e fra la 8 e la 9. Quando il salto è preso (con probabilità *p* quindi) le istruzioni per la singola iterazione sono 7 (1,2,3,4,5,8,9), le dipendenze sono 2 (k=1, d1=2/7, Nq1=(2+1)/2) e i salti presi 2 (λ=2/7). Nel caso il salto all'istruzione 5 non sia preso, invece, abbiamo 9 istruzioni (tutte, in sequenza) tre dipendenza (k=1, d1=3/7, Nq1=(2+1+1)/3) e i salti presi sono 1 (λ=1/7).

Per ottimizzare il codice analizziamo le relazioni fra le varie istruzioni. Le relazioni sono quelle in figura:



Per ridurre l'effetto della dipendenza indotta dalla IF> possiamo dunque riordinare queste istruzioni come segue (senza che la semantica del programma cambi):

```
loop:LOAD RbaseA, Ri, R1
     LOAD RbaseB, Ri, R2
     SUB R1, R2, R1
     ADD R3, R2, R3
     IF> R1, R0, cont
     DEC Rj
     STORE RbaseC, Rj, R1
cont:INC Ri
     IF< Ri, Rn, loop
```

A questo punto la dipendenza indotta dalla IF> è diventata di distanza 2. Dal momento che la sequenza di istruzioni eseguite sulla EU che porta alla istruzione che induce la

dipendenza contiene in questo una LOAD, la bolla per k=1 è da 2 posizioni, mentre per k=2 diventa da 1 posizione sola, dimezzando l'effetto della dipendenza.

Considerando adesso la parte del codice che segue il codice compilato dall'*if-then*, codice che deve essere eseguito indipendentemente dal salto preso o non preso, notiamo come la INC scriva un registro utilizzato per l'ultima volta nelle LOAD. Dunque può essere spostata in un punto qualunque del programma che segua le LOAD, purchè in una posizione attraversata sia nel caso di salto preso (per la IF>) che di salto non preso. Possiamo dunque spostarla, con l'intento di ridurre la terza dipendenza, quella indotta dalla IF<.

```
loop:LOAD RbaseA, Ri, R1
     LOAD RbaseB, Ri, R2
     SUB R1, R2, R1
     ADD R3, R2, R3
     INC Ri
     IF> R1, R0, cont
     DEC Rj
     STORE RbaseC, Rj, R1
cont:IF< Ri, Rn, loop
```

In questo caso, sia la dipendenza indotta dalla IF< che quella indotta dalla IF> diventano dipendenze con k≥2, quindi tali da non determinare bolle nel processore pipeline D-RISC.

Dal momento che la dipendenza indotta dalla IF< dava luogo ad una bolla da 1 (niente LOAD nella sequenza che porta a questa istruzioni), questa soluzione annulla completamente l'effetto di 2 delle 3 dipendenze del codice.

La dipendenza indotta dalla STORE è difficilmente eliminabile visto che le istruzioni in gioco sono di fatto confinate nel ramo "salto preso" dell'IF>.

Secondo il modello analitico, il tempo medio di servizio è dato da

$$T = (1+\lambda)t + t\Sigma(dk(Nqk+1-k))$$

Consideriamo separatamente il caso "salto IF> preso" e quello "salto IF> non preso".

Caso 1 (prob = p)

$\lambda = 2/7$ $d1 = 0$ $T = (1+1/7)\ t = 9t/7$

Caso 2 (prob = 1-p)

$\lambda = 1/9$ $d1 = 1/9$ $Nq1 = 1$ $T = (1+1/9)\ t + t\ 1/9 = 12t/9$

Quindi complessivamente il tempo medio di servizio sarà

$T = p\ (9t/7) + (1-p)\ (12t/9)$

Il tempo di completamento sarà pertanto dato da

$$Tc = N\ (7\ p + 9\ (1-p))\ T$$

Consideriamo adesso i salti. Possiamo mitigarne l'effetto utilizzando delayed branch sia per la IF> che per la IF< di fine ciclo. Nel primo caso, possiamo utilizzare la INC per riempire lo slot del salto ritardato. Nel secondo caso possiamo utilizzare la LD di inizio ciclo.

```
     LOAD RbaseA, Ri, R1
loop:LOAD RbaseB, Ri, R2
```

```
     SUB R1, R2, R1
     ADD R3, R2, R3
     IF> R1, R0, cont, delayed_branch
     INC Ri
     DEC Rj
     STORE RbaseC, Rj, R1
cont:IF< Ri, Rn, loop, delayed_branch
     LOAD RbaseA, Ri, R1
```

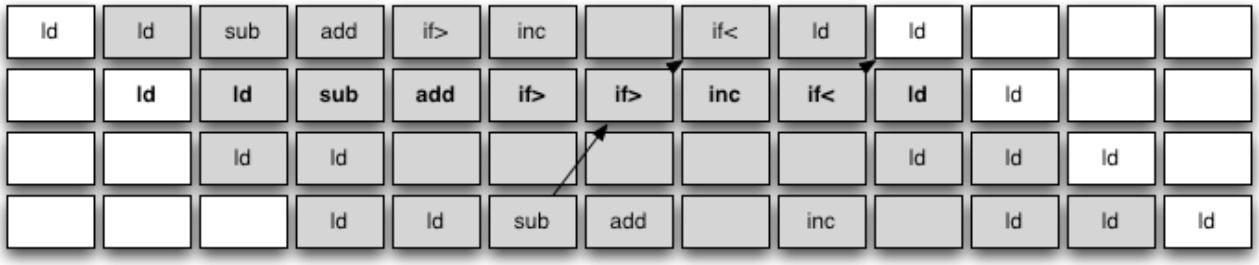
Tuttavia, lo spostamento della INC fa' sì che la dipendenza indotta dall'IF> ritorni visibile (k=2, N=2, quindi bolla da 1 nel pipeline D-RISC), il che bilancia l'eliminazione della bolla da 1 dovuta al salto preso ed eliminata con il delayed branch. L'utilizzo della LOAD per il delay slot del salto di fine ciclo comporta lo spostamento dell'etichetta di inizio ciclo e lascia in R1 un valore non uguale a quello lasciato dal codice originale. Ciò è ammissibile considerando che R1 non verrà più letto come A[i] fuori dal ciclo.

Con queste ottimizzazioni, in entrambi i casi (salto IF> preso o non preso) abbiamo λ=0, ma rimane una bolla da 1 indotta dalla IF> e una bolla, ancora da 1 e solo nel caso di salto IF> non preso, indotta dalla STORE.

Riportiamo (quella con il metodo della simulazione non era richiesta nel testo) la valutazione del tempo di servizio nei due casi di salto IF> preso e non preso.

Salto IF> preso (prob = p)

λ=0 d1=0 d2=1/7 Nq2=2 T1=t + t(1/7(2+1-2)) = 8t/7

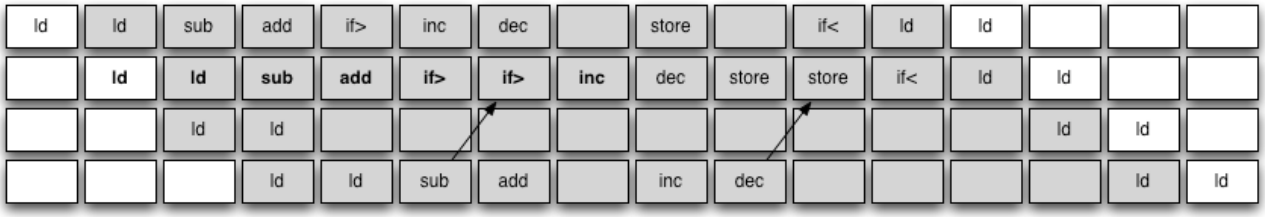


e dalla simulazione, infatti T1 = 8t/7

Salto IF> non preso (prob = (1-p))

λ=0 d1=1/9 NQ1=1 d2=1/9 Nq2=1

T2 = t + t (1/9(1+1-1) + 1/9(1+1-1)) = t + t(2/9) = 11t/9



e dalla simulazione, infatti T2 = 11t/9.

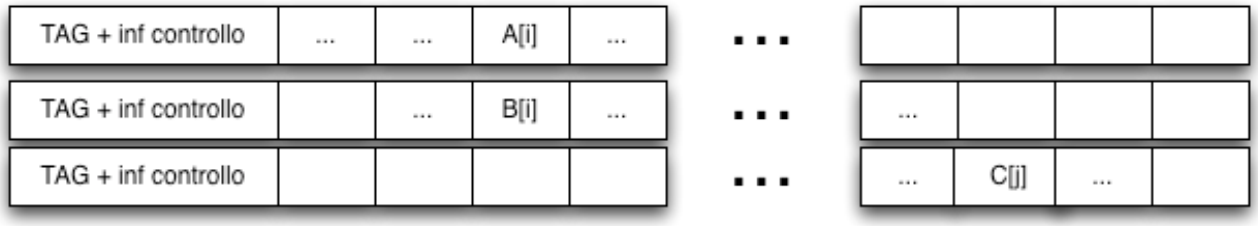
Il tempo di completamento in questo caso è

$$Tc = N * (7\ (p)\ T1 + 9\ (1\text{-}p)\ T2)$$

Consideriamo ora il secondo punto della domanda: numero di fault generati dal programma. In questo programma scorriamo tre vettori: A e B sono acceduti sequenzialmente dalla posizione 0 alla posizione N-1. C è acceduto sequenzialmente dalla

posizione N-1 verso il basso, non interamente visto che con probabilità *p* si salta la fase che decrementa j ed accede in scrittura C[j].

Il working set del programma, all'iterazione *i* sarà costituito dai blocchi di cache seguenti:



per quanto riguarda la cache dati e dagli 1 o 2 blocchi (a seconda dell'indirizzo della prima istruzione del blocco stesso) di cache che contengono il codice, per quanto riguarda la cache istruzioni. Le dimensioni del working set (che è relativamente piccolo) di fatto permettono di assumere che il WS stesso sia in cache per qualunque dimensione totale della cache stessa.

La cache dati opera su domanda e quindi possiamo concludere che avremo N/σ fault per A, altrettanti per B e (1-p)N/σ per C. Complessivamente il numero dei fault sarà dunque

$$\#\text{fault} = 2\,(N/\sigma) + (1 - p)(N/\sigma) + 2 \cong (3 - p)(N/\sigma)$$

Domanda 2

1) Si dovrebbe dire sommariamente che il trattamento delle interruzioni consta di una fase firmware e di una fase assembler. Nella fase firmware, dopo aver rilevato l'interruzione mediante il segnalatore INT nell'ultima micro istruzione dell'interprete firmware che relativa all'esecuzione dell'istruzione assembler corrente, si manda un ACKINT e di seguito si prelevano le due parole fornite dal dispositivo che interrompe mediante l'interfaccia di memoria. Le due parole sono salvata in registri generali. Quindi si salta alla fase assembler. Si utilizza la prima parola per individuare la routine che dovrà gestire l'interruzione e vi si salta dopo aver salvato l'indirizzo di ritorno. In tutto questo procedimento, lo spazio di indirizzamento è quello del processo interrotto e l'unica struttura dati di rilievo acceduta è la tabella dei gestori delle interruzioni, la cui base si trova nel PCB del processo in esecuzione.
2) Se la dipendenza era indotta da un'istruzione al termine di una sequenza di istruzioni eseguite sulla EU che contengono almeno una LOAD, la bolla da 2 viene ridotta ad una bolla da 1. Se viceversa, la sequenza non conteneva una LOAD, la bolla era da 1 e la trasformazione della dipendenza in una dipendenza con k=2 ne annulla l'intero effetto. Infatti, in questo caso, il ritardo prodotto dalla dip sarebbe pari a

$$t\ (d2\ (Nq2 + 1 - k) = t\ d2\ (1 + 1 - 2) = 0t$$