Architettura degli Elaboratori

a.a. 2013/14 - quinto appello, 12 febbraio 2014

Riportare nome, cognome, numero di matricola e corso A/B

Domanda 1

Definire e realizzare una rete sequenziale che calcola il massimo di 256 valori interi non-negativi che costituiscono la sequenza d'ingresso, e valutarne il ciclo di clock. Spiegare come si è ragionato per ottenere la definizione formale della rete.

L'ingresso IN e l'uscita OUT sono sincronizzati con interfacce a transizione di livello. Una volta terminata la sequenza d'ingresso, la rete deve rimanere in uno stato stabile. A parte il valore iniziale, il registro OUT deve contenere solo il risultato finale.

Domanda 2

Un'unità di elaborazione U utilizza una gerarchia di memoria: memoria principale M, cache secondaria C2, cache primaria C1, con C1 e C2 sullo stesso chip di U. La cache a due livelli C1-C12 è su domanda, con blocchi di ampiezza $\sigma_1 = 8$, C1 di capacità 16K e C2 di capacità 1M.

U opera su tre array A, B, C di *M* interi, con *M* = 128 K:

$$\forall\ i = 0\ ..\ M-1\text{: } \{\ C[i] = 0;\ \ \forall\ j = 0\ ..\ M-1\text{: } C[i] = F(A[i], B[j], C[i])\ \}$$

L'array A è allocato staticamente in M all'indirizzo 0. U riceve l'indirizzo di M in cui trovare il nuovo valore dell'array B e l'indirizzo di M in cui scrivere il valore calcolato dell'array C.

Definire l'insieme di lavoro della gerarchia di memoria e valutare la penalità sul tempo di servizio di U dovuta ai fault di cache. I parametri della memoria principale possono essere fissati dallo studente.

Domanda 3

Un'unità di I/O riceve dal dispositivo associato uno stream di parole e le invia, una alla volta, alla CPU usando direttamente il meccanismo delle interruzioni. Il servizio richiesto con l'interruzione è di bufferizzare la parola in una data coda FIFO di 20 posizioni. L'operazione di inserzione in coda è eseguita dallo *handler* dell'interruzione, l'estrazione verrà effettuata da parte di determinati processi.

La coda è una struttura dati con i seguenti campi: indice di inserzione, indice di estrazione, numero di parole attualmente presenti, vettore di 20 posizioni. Handler conosce l'indirizzo logico base di tale struttura, e questo è unico per qualunque processo.

Come di regola, in caso di coda piena la richiesta di servizio di inserzione deve attendere che si svuoti almeno una posizione, ma questo deve avvenire *senza sospendere il processo* che esegue lo handler. Inoltre, nell'operazione di inserzione non deve essere effettuata alcuna azione di scheduling a basso livello nei confronti dei processi estrattori.

a) Scrivere il codice del trattamento dell'interruzione, incluso handler.
b) Valutare la latenza complessiva dell'operazione di inserzione in coda, a partire dall'istante in cui l'unità di I/O dispone del dato da inviare fino all'istante in cui termina il trattamento dell'interruzione.

La CPU è D-RISC, pipeline, scalare, a singola bufferizzazione, con EU parallela, e cache C1-C2 come nella Domanda 2. Si assume che il PCB del processo in esecuzione e la sua tabella delle interruzioni risiedano in C1, e la struttura dati coda in C2. Qualunque collegamento inter-chip (inclusa la connessione di I/O) va considerato come dedicato con latenza di trasmissione 5τ.

Soluzione

Domanda 1

Definiamo la rete sequenziale (X, Z, S, ω, σ) secondo il modello di Moore, in particolare con lo stato di uscita Z dato dal contenuto del registro OUT (funzione identità delle uscite, ω).

Lo stato interno S è costituito dai contenuti dei registri IN e OUT, del registro MAX destinato a contenere il valore massimo della sequenza, del registro I di 9 bit operante come contatore di passi, e dai quattro indicatori di interfaccia RDYIN, ACKIN, RDYOUT, ACKOUT. L'automa ha stato iniziale: MAX, I, OUT e gli indicatori di interfaccia sono inizializzati a zero, e IN è non specificato.

La funzione σ di *transizione dello stato interno* è definita come segue:

- con la sequenza in corso ($I_0 = 0$), il nuovo valore di IN, quando presente (RDYIN = 1), viene scritto in MAX a condizione che *segno*(MAX – IN) = 1:

$$in_{MAX} = when\ clock\ and\ (\overline{I_0}\ \ RDYIN\ \ segno(MAX - IN)\ )\ do\ IN$$

- con la sequenza in corso e un nuovo IN presente, I viene incrementato di uno:

$$in_I = when\ clock\ and\ (\overline{I_0}\ \ RDYIN\ )\ do\ I + 1$$

- quando la sequenza è terminata, MAX viene scritto in OUT a condizione che ACKOUT = 1:

$$in_{OUT} = when\ clock\ and\ (I_0\ ACKOUT\ )\ do\ MAX$$

- per gli indicatori di interfaccia:

$$in_{Y\_RDYIN} = when\ clock\ and\ (\overline{I_0}\ \ RDYIN\ )\ do\ \overline{Y\_RDYIN}$$

$$in_{ACKIN} = when\ clock\ and\ (\overline{I_0}\ \ RDYIN\ )\ do\ \overline{ACKIN}$$

$$in_{RDYOUT} = when\ clock\ and\ (I_0\ ACKOUT\ )\ do\ \overline{RDYOUT}$$

$$in_{Y\_ACKOUT} = when\ clock\ and\ (I_0\ ACKOUT\ )\ do\ \overline{Y\_ACKOUT}$$

Dalla definizione si ricava immediatamente la realizzazione della rete, facente uso di due ALU e con i valori dei β dei registri ottenuti come uscite di porte AND definite dalla funzione σ di cui sopra. Essendo nullo il ritardo della funzione delle uscite, il ciclo di clock è dato da:

$$\tau = \max(T_\omega, T_\sigma) + \delta = T_\sigma + \delta = T_{ALU} + \delta$$

Esattamente lo stesso risultato si può ottenere progettando la rete sequenziale come unità di elaborazione descritta da un microprogramma di una sola microistruzione:

0. (I_0, RDYIN, segno(MAX – IN), ACKOUT = 0 0 – –, 1 – – 0) nop, 0;
   (= 0 1 1 –) reset RDYIN, set ACKIN, IN → MAX, I + 1 → I, 0;
   (= 0 1 0 –) reset RDYIN, set ACKIN, I + 1 → I, 0;
   (= 1 – – 1) MAX → OUT, set RDYOUT, reset ACKOUT, 0

Con le usuali regole, si deriva la *definizione* delle funzioni ω e σ viste sopra.

Domanda 2

A è allocato staticamente in M ed è in sola lettura: poiché U opera su stream, A è quindi caratterizzato, oltre che da località, da *riuso*. Dati i valori di *M*, γ_1 e γ_2, il riuso di A è esplicitabile in C2 (dopo l'elaborazione del primo elemento dello stream), ma non in C1. Analogamente per il nuovo array B, caratterizzato anche da riuso per la durata dell'elaborazione di un elemento dello stream: dopo la prima

iterazione su *i*, B viene mantenuto in C2 fino alla ricezione del prossimo elemento dello stream. L'array C è caratterizzato da sola località ed è usato in sola scrittura (non c'è trasferimento di blocchi da M a C2-C1).

L'insieme di lavoro è quindi costituito da tutti i blocchi di A e di B e dal blocco corrente di C.

La penalità sul tempo di servizio dovuta ai fault di cache è data da:

$$T_{fault} = N_{fault-A-C1}\, \boldsymbol{T_{trasfC2-C1}}(\sigma_1) + N_{fault-B-C2}\, \boldsymbol{T_{trasfM-C2}}(\sigma_1) + N_{fault-B-C1}\, \boldsymbol{T_{trasfC2-C1}}(\sigma_1))$$

Si ha:

$$N_{fault-A-C1} = N_{fault-B-C2} = \frac{M}{\sigma_1} \qquad N_{fault-B-C1} = \frac{M^2}{\sigma_1}$$

Quindi, grazie gli ordini di grandezza in gioco:

$$T_{fault} \sim \frac{M^2}{\sigma_1}\, \boldsymbol{T_{trasfC2-C1}}(\sigma_1) = \frac{M^2}{\sigma_1}\ 2\,\sigma_1\,\tau\ = 2\,M^2\,\tau$$

indipendentemente dalla realizzazione della memoria principale. Il calcolo è sua volta $O(M^2)$.

[L'aspetto dell'eventuale penalità dovuta alle scritture di C in M non è richiesto dall'esercizio. Per completezza:

- se la gerarchia fa uso di write-through, le singole scritture in M sono sovrapposte, almeno parzialmente, al calcolo;
- in caso di write-back, ogni blocco di C viene scritto in M una volta calcolato. Le scritture dei blocchi sono almeno parzialmente sovrapposte al calcolo.

In entrambi i casi, si tratta di valutare l'eventuale penalità partendo dal grado di sovrapposizione.]

Domanda 3

a) Si noti che lo handler è eseguito da qualunque processo che sia in esecuzione all'atto dell'interruzione, non necessariamente il processo che eseguirà anche l'estrazione. Quindi la coda è condivisa: dalle specifiche si deduce come i puntatori condivisi siano gestiti come indirizzi logici coincidenti.

Per rispettare il vincolo sulla corretta bufferizzazione, ogni volta che riempie completamente la coda handler provvede a *disabilitare l'interruzione* corrispondente all'evento in questione (istruzione MASKINT). In altre parole, *questo tipo di interruzione non viene accettata se la coda è piena*. Con l'interruzione disabilitata, l'unità di I/O rimane in attesa di ACKINT, e UNINT conserva il segnale di interruzione; successivamente, quando l'interruzione pendente verrà *riabilitata dall'operazione di estrazione*, il processore (IU) la accetterà e innescherà lo handler per l'inserzione in coda (in questa situazione, è probabile che handler sia eseguito dallo stesso processo che ha effettuato l'estrazione).

Trattamento dell'interruzione, fase assembler:

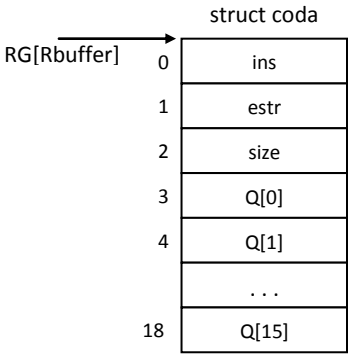
// la fase firmware ha provveduto a memorizzare il codice-evento (inserzione in coda) in *Revento*, la parola-messaggio in *Rdato*, e l'indirizzo di ritorno in *Rret_int* //

Routine di interfacciamento interruzioni:

1. LOAD Rpcb, Roffset, Rtabella_interruzioni
2. LOAD Rtabella_interruzioni, Revento, Rindirizzo_handler
3. CALL Rindirizzo_handler, Rret_handler
4. GOTO Rret_int

Questa prima parte di codice contiene le dipendenza logiche IU-EU 1-2 e 2-3, entrambe di distanza k = 1 con $N_Q = 2$, e due salti ineliminabili (3, 4).

Handler:



// l'indirizzo logico della struttura coda è in *Rbuffer* //

// il test di coda piena viene effettuato dopo l'inserzione //

// *codice non ottimizzato* //

```
5.  LOAD   Rbuffer, 0, Rins          // indice di inserzione //
6.  ADD    Rbuffer, 3, RQ            // indirizzo base di Q //
7.  STORE  RQ, Rins, Rdato           // scrittura in coda //

8.  INCR   Rins
9.  MOD    Rins, 20, Rins            // ins = (ins + 1)%20 //
10. STORE  Rbuffer, 0, Rins

11. LOAD   Rbuffer, 2, Rsize
12. INCR   Rsize                     // size = size + 1; il valore risultante è sempre ≤ 16 //
13. STORE  Rbuffer, 2, Rsize

14. IF <   Rsize, 20, EXIT           // test di coda piena //
15. MASKINT  Rmask, Rvalore_maschera     // se coda piena, disabilita interruzione //

16. EXIT:  GOTO  Rret_handler
```

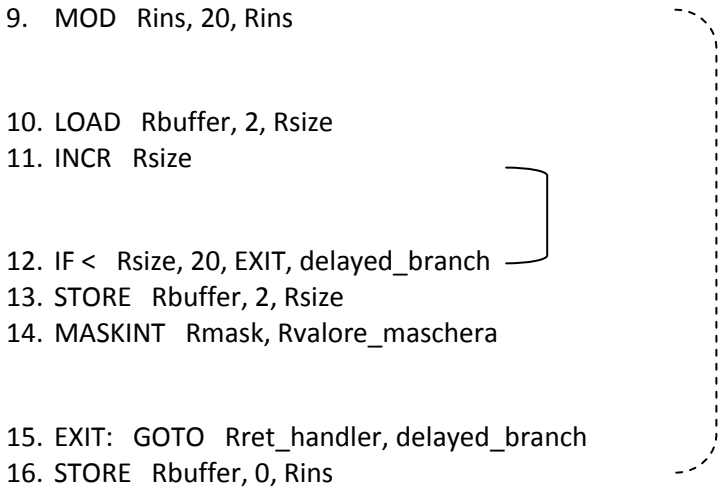
Il codice dello handler contiene le dipendenze logiche IU-EU indicate, tutte con k = 1 (6 su 7 con $N_Q = 2$, 9 su 10 con $N_Q = 1$ e $L_{pipe\text{-}k} = 4$, 12 su 13 con $N_Q = 2$) e due salti (14, 16).

b) Nella valutazione supporremo che l'istruzione MASKINT non venga eseguita (caso più probabile), per cui, globalmente, si tratta di eseguire 15 istruzioni.

Trattamento dell'interruzione, fase assembler - codice ottimizzato:

```
1.  LOAD   Rpcb, Roffset, Rtabella_interruzioni
2.  LOAD   Rtabella_interruzioni, Revento, Rindirizzo_handler
3.  CALL   Rindirizzo_handler, Rret_handler
4.  GOTO   Rret_int

5.  LOAD   Rbuffer, 0, Rins
6.  ADD    Rbuffer, 2, RQ            // indirizzo_base_vettore_Q – 1 //
7.  INCR   Rins
8.  STORE  RQ, Rins, Rdato
```



Non possono essere effettuati spostamenti di codice tra la routine di interfacciamento e lo handler, per cui la cause di degradazione nelle istruzioni 1-4 rimangono.

Nello handler, si è scelto di eliminare la dipendenza logica più pesante (indotta dalla MOD di latenza lunga) e le degradazioni dovute ai due salti. Si può inoltre alleviare la dipendenza indotta sulla STORE RQ. Rimane quella della 11 sulla 12 (la cui bolla contribuisce, comunque, a raggiungere la distanza necessaria per eliminare la dipendenza indotta dalla MOD).

Quindi:

$$\lambda = \frac{2}{15}$$

$$\Delta = \Delta_{1-2}\left(k = 1, N_Q = 2\right) + \Delta_{2-3}\left(k = 1, N_Q = 2\right) + \Delta_{6-8}\left(k = 2, N_Q = 2\right) + \Delta_{11-12}\left(k = 1, N_Q = 2\right) = \frac{7}{15}\, t$$

Il tempo di servizio effettivo per istruzione è:

$$T = (1 + \lambda)t + \Delta = \frac{24}{15}\, t$$

La latenza della fase assembler del trattamento interruzioni, in assenza di fault di cache, vale dunque:

$$L_{assem-trattint-0} = 15\, T = 24\, t = 48\, \tau$$

La penalità dovuta ai fault si ha a causa del trasferimento da C2 a C1 dei blocchi della struttura dati coda: almeno il primo blocco viene certamente acceduto e, nel caso peggiore (probabile), un altro blocco appartenente al vettore Q.

Quindi valutiamo:

$$N_{fault} = 2$$

$$T_{fault} = N_{fault}\ 2\, \sigma\, \tau = 32\, \tau$$

da cui:

$$L_{assem-trattint} = L_{assem-trattint-0} + T_{fault} = 80\, \tau$$

Va poi considerata la latenza della *fase firmware del trattamento interruzioni*:

- da I/O a UNINT: $1\tau + T_{tr}$
- da UNINT a IU e da IU a I/O (ACKINT): $2\tau + T_{tr}$, se l'interruzione è abilitata
- da I/O a DM: $1\tau + T_{tr}$
- da DM a EU (2 parole) e a IU, dove ha luogo la chiamata della routine di interfacciamento: 5τ

Complessivamente:

$$L_{fw-trattint} = 9\,\tau + 3\,T_{tr} = 24\,\tau$$

In conclusione, la latenza di una inserzione in coda vale:

$$L_{inserzione} = L_{fw-trattint} + L_{assem-trattint} = 104\,\tau$$