Architettura degli Elaboratori

a.a. 2012/13 – quinto appello, 10 febbraio 2014

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

Domanda 1

Un'unità di elaborazione U_{sw} comunica in ingresso con due unità UIN_0, UIN_1 e in uscita con due unità $UOUT_0$, $UOUT_1$.

Dalla generica UIN_i (i = 0, 1) riceve la coppia (H_i, V_i), dove V_i è una parola di 32 bit, e H_i è una stringa di 19 bit suddivisa logicamente nei campi *C* (3 bit), *S* (8 bit), *D* (8 bit). Se il bit *C*-esimo di *S* e quello di *D* sono uguali allora V_i deve essere inviato a $UOUT_0$, altrimenti a $UOUT_1$.

Quando possibile, richieste contemporanee da UIN_0, UIN_1 devono essere servite in parallelo. Quando ciò non sia possibile, deve essere adottata una politica che assicuri che una richiesta rimanga in attesa solo per il tempo necessario a servire l'altra.

Progettare U_{sw} (microprogramma, definizione della struttura, ciclo di clock) in modo che la massima banda di elaborazione raggiungibile sia uguale a $2/\tau$ parole al secondo.

Fornire spiegazioni adeguate.

Domanda 2

Un processo in esecuzione su una CPU D-RISC provoca il trasferimento del contenuto di un proprio array *A[N]* nella memoria locale di una determinata unità di I/O. Si vuole valutare la latenza complessiva del trasferimento, dall'istante in cui viene iniziato nel processo all'istante in cui viene concluso nella memoria di I/O, in due casi distinti:

a) l'unità di I/O accede in DMA ai valori di *A*, supposto allocato per intero e in pagine consecutive in memoria principale. I/O opera solo su indirizzi fisici, per cui deve ricevere l'indirizzo fisico di *A*. Il processo non deve essere avvertito della fine del trasfrimento;

b) l'unità di I/O non opera in DMA e l'array *A* è presente nella cache secondaria della CPU.

L'unità di I/O è diversa nei due casi; il suo comportamento deve essere specificato.

La latenza richiesta deve essere valutata con precisione in termini di numero di cicli di clock, senza approssimazioni. Specifiche:

- la CPU ha architettura pipeline scalare con Unità Esecutiva parallela, con comunicazioni a singola bufferizzazione, cache primaria su domanda con blocchi di 8 parole, e cache secondaria on-chip;
- agli effetti della valutazione, si suppone che nel caso *a)* la cache secondaria contenga le strutture dati del supporto del processo in esecuzione, nel caso *b)* l'array A;
- CPU e I/O hanno lo stesso ciclo di clock τ;
- il tempo di accesso dell'unità di I/O alla propria memoria locale è uguale a 2τ;
- la memoria principale ha organizzazione sequenziale con ciclo di clock di 30τ;
- tutti i collegamenti inter-chip sono dedicati ed hanno latenza di trasmissione uguale a 9τ;
- la memoria fisica ha capacità massima 64G parole e le pagine di memoria virtuale hanno ampiezza 1K parole.

Fornire spiegazioni adeguate.

Soluzione

Domanda 1

La massima banda viene raggiunta quando, essendo presenti contemporaneamente (nello stesso ciclo di clock) due richieste in ingresso, esse risultano dirette a interfacce di uscita distinte. Ciò richiede che, nello stesso ciclo di clock, si testi la presenza delle due richieste e degli ACK delle interfacce di uscita, si applichi ad ogni richiesta una funzione booleana (*route*) per determinare l'interfaccia di uscita e si determini se, in caso di due richieste, esse siano o meno in conflitto per la stessa interfaccia di uscita.

Nel caso di conflitto, all'interfaccia viene inviata una delle due richieste scelta in base ad un booleano P che viene complementato ad ogni invio (priorità pseudo-casuale che assicura quanto richiesto dalle specifiche). L'altra richiesta attende normalmente nell'interfaccia d'ingresso e verrà servita al ciclo di clock successivo.

La funzione che determina l'interfaccia di uscita di una richiesta:

$$route_i = if\ (H_i.S[C] \neq H_i.D[C]) \qquad i = 0,1$$

è implementabile da una rete combinatoria con componenti standard:

$$route_i = not\ exor\ (commutatore\ (H_i.S, C), commutatore\ (H_i.\ D, C))$$

Serve inoltre la funzione:

$$conflitto = not\ exor\ (route_0, route_1)$$

Il microprogramma consta quindi di una sola microistruzione:

0. ($RYIN_0$, $RDYIN_1$, conflitto, P, ACKOUT[$route_0$], ACKOUT[$route_1$] = 0 0 - - - -, 0 1 - - - 0, 1 0 - - 0 - , 1 1 0 - 0 0, 1 1 0 - 0 1, 1 1 0 - 1 0, 1 1 1 0 0 -, 1 1 1 1 - 0) nop, 0;

   (= 1 0 - - 1 - , 1 1 1 0 1 -, 1 1 0 - 1 0) reset $RDYIN_0$, set $ACKIN_0$, $V_0 \rightarrow$ OUT[$route_0$], set RDYOUT[$route_0$], reset ACKOUT[$route_0$], $\bar{P} \rightarrow P$, 0;

   (= 0 1 - - - 1, 1 1 1 1 - 1, 1 1 0 - 0 1) reset $RDYIN_1$, set $ACKIN_1$, $V_1 \rightarrow$ OUT[$route_1$], set RDYOUT[$route_1$], reset ACKOUT[$route_1$], $\bar{P} \rightarrow P$, 0;

   (= 1 1 0 - 1 1) reset $RDYIN_0$, set $ACKIN_0$, $V_0 \rightarrow$ OUT[$route_0$], set RDYOUT[$route_0$], reset ACKOUT[$route_0$], reset $RDYIN_1$, set $ACKIN_1$, $V_1 \rightarrow$ OUT[$route_1$], set RDYOUT[$route_1$], reset ACKOUT[$route_1$], $\bar{P} \rightarrow P$, 0

Esistono altri modi di esprimere il microprogramma, tutti *equivalenti* nella semantica *e nelle prestazioni*, concentrando una o più espressioni condizionali nelle microoperazioni invece che testando tutte le possibili situazioni nelle condizioni logiche.

L'unità è realizzata come singola rete sequenziale, avente *funzione delle uscite* (ω) identità rispetto ai contenuti dei registri di uscita, e *funzione di transizione dello stato interno* (σ) che, limitatamente ai registri di uscita e P (gli indicatori d'interfaccia hanno una funzione standard), è definita come:

$i = 0,1$: $in_{OUT_i} = when\ (\beta_i = 1)\ do\ if\ \alpha_i\ then\ V_1\ else\ V_0$

$$\beta_0 = (\overline{route_0} + \overline{route_1})\ ACK_0\ (RDY_0 + RDY_1)$$

$$\beta_1 = (route_0 + route_1)\ ACK_1\ (RDY_0 + RDY_1)$$

$$\alpha_0 = (\overline{route_0} + \overline{route_1})\ RDY_1\ (RDY_0\ conflitto\ P + \overline{RDY_0})$$

$$\alpha_1 = (route_0 + route_1)\ RDY_1\ (RDY_0\ conflitto\ P + \overline{RDY_0})$$

$in_P = when\ (\beta_P = 1)\ do\ \bar{P}$

$\beta_P = \beta_0 + \beta_1$

Il ritardo di stabilizzazione della rete combinatoria corrispondente è dato da:

$$T_\sigma = T_{conflitto} + T_{rete\_\alpha\beta} + T_K = 6\,t_p + 2\,t_p + 2\,t_p = 10\,t_p$$

(La "rete_αβ" è definita della funzione σ scritta in forma AND-OR. Ogni registro di uscita ha in ingresso un commutatore a due ingressi V_0, V_1)

Quindi:

$$\tau = T_\sigma + \delta = 11\,t_p$$

Il tempo di servizio varia da τ/2 (inverso della massima banda) a τ (inverso della minima banda).

Domanda 2

a) La latenza richiesta viene valutata come somma di tre latenze:

1) traduzione dell'indirizzo base di A e invio dell'indirizzo fisico all'unità di I/O con modalità Memory Mapped I/O;
2) trasmissione dell'indirizzo fisico sul collegamento tra CPU e di I/O;
3) trasferimento di A a domanda-risposta tra unità di I/O e memoria principale.

1) Il processo traduce esplicitamente l'indirizzo logico base di A (presente nel registro generale RA) nel corrispondente l'indirizzo fisico, utilizzando la propria tabella di rilocazione il cui indirizzo si trova in una locazione nota del proprio PCB (indirizzo in Rpcb).

Gli indirizzi fisici sono di 36 bit. L'identificatore di pagina fisica IPF è di 26 bit. È possibile rappresentare l'entrata delle tabella di rilocazione mediante una singola parola avente IPF nei 26 bit meno significativi.

Il codice assembler per la traduzione dell'indirizzo e il suo invio all'unità di I/O è il seguente:

1. LOAD Rpcb, Rscarto_tabril, Rtabril // indirizzo tabella di rilocazione //
2. DIV RA, Rpage_size, Ripl // ident. della prima pagina logica di A; page_size = 1024 //
3. LOAD Rtabril, Ripl, Rentry // entrata della tabella di rilocazione //
4. MOD Rentry, Rnum_pagine_fisiche, Ripf // ident. di pagina fisica; num_pagine_fisiche = 2^{26} //
5. MUL Ripf, Rpage_size, Rind_fis // indirizzo fisico base dell'array A //
6. STORE RI/O, 0, Rind_fis

Nell'istruzione 6, RG[RI/O] contiene l'indirizzo logico cui corrisponde l'indirizzo fisico della locazione della memoria locale di I/O in cui scrivere l'informazione di interesse, in questa caso l'indirizzo fisico base di A. Poiché l'interfaccia di I/O fa capo alla MMU_D del sottosistema DM, è MMU_D che smista la richiesta di scrittura all'unità di I/O; ciò non comporta alcuna modifica al modello dei costi della CPU pipeline (stesso tempo di servizio di una STORE normale).

Valutiamo il tempo di completamento. Il codice contiene le dipendenze logiche IU-EU

- della 2 sulla 3 (distanza k1 = 1, $N_{Qk1} = 2$, $L_{pipe\text{-}k1} = 4$),
- della 5 sulla 6 (distanza k2 = 1, $N_{Qk2} = 2$, $L_{pipe\text{-}k2} = 4$) su cui ha impatto la dipendenza EU-EU della 4 sulla 5 (distanza h = 1, $L_{pipe\text{-}h} = 4$).

Il tempo di servizio per istruzione di questo segmento di codice:

$$T = t + \Delta = t + t\,\frac{1}{6}\left[\left(N_{Qk1} + 1 - k1 + L_{pipe\,-k1}\right) + \left(N_{Qk2} + 1 - k2 + L_{pipe\,-k2}\right) + \left(1 - h + L_{pipe\,-h}\right)\right] = \frac{22}{6}t$$

da cui la latenza in assenza di fault di cache:

$$L_{CPU-0} = 6\,T = 22\,t = 44\,\tau$$

Secondo le specifiche, PCB e tabella di rilocazione sono supposti presenti in cache secondaria. Quindi occorre pagare i trasferimenti di due blocchi di cache primaria da cache secondaria:

$$T_{fault} = 2\,T_{trasm} = 4\,\sigma\,\tau = 32\,\tau$$

Complessivamente la latenza della prima fase vale:

$$L_{CPU} = L_{CPU-0} + T_{fault} = 76\,\tau$$

2) Il trasferimento dell'indirizzo fisico da CPU a unità di I/O su collegamento dedicato ha una latenza:

$$L_{CPU-I/O} = \tau + T_{tr} = 10\,\tau$$

3) L'unità di I/O, una volta ricevuto l'indirizzo fisico di A (logicamente in una locazione di MI/O, in realtà in un registro interno) inizia un trasferimento a domanda-risposta su collegamenti dedicati con la memoria principale. La latenza è data da:

$$L_{M-I/O} = N\,[2\,(\tau + T_{tr}) + \tau_M] = 50\,N\,\tau$$

essendo la scrittura di una parola nella memoria di I/O interamente sovrapposta al trasferimento della parola successiva.

In conclusione, la latenza richiesta vale:

$$L = L_{CPU} + L_{CPU-I/O} + L_{M-I/O} = (86 + 50\,N)\,\tau \sim 50\,N\,\tau$$

b) Il processo esegue un semplice loop del tipo:

```
for (i = 0; i < N; i++)
        B[i] = A[i]
```

dove B è la copia di A allocata nella memoria locale di I/O, riferita con la tecnica del Memory Mapped I/O. In questo caso, l'unità di I/O è progettata per ricevere, in un loop, una parola e inviarla alla memoria locale in parallelo alla ricezione della parola successiva.

Il codice assembler ottimizzato per la CPU pipeline è:

```
LOOP:  LOAD   RA, Ri, Ra
       INCR   Ri
       IF <   Ri, RN, LOOP, delayed_branch
       STORE  RI/O, Ri, Ra
```

in cui l'unica degradazione è causata dalla dipendenza logica IU-EU della INCR sulla IF (k = 1, N_Q = 2). Quindi:

$$T = t + \frac{1}{4}\left(N_Q + 1 - k\right) = \frac{6}{4}\,t$$

Da cui la latenza a vuoto:

$$L_0 = 4\,N\,T = 6\,N\,t = 12\,N\,\tau$$

Il trasferimento di una parola da CPU a unità di I/O più la scrittura in MI/O ha una latenza uguale a $3\tau + T_{tr} = 12\tau$, quindi sovrapposto all'esecuzione della prossima iterazione del programma, anche senza considerare l'effetto dei fault di cache.

Essendo A interamente presente in cache secondaria, la penalità dei fault di cache primaria vale:

$$T_{fault} = N_{fault}\ T_{trasm} = \frac{N}{\sigma}\,2\,\sigma\,\tau = 2\,N\,\tau$$

Complessivamente la latenza richiesta vale:

$$L = L_0 + T_{fault} = 14\,N\,\tau$$

sensibilmente inferiore rispetto al caso con DMA, a riprova del fatto che, quando la memoria principale venga usata intensivamente, prestazioni adeguate si possono ottenere solo con una sua realizzazione a larga banda (ad esempio, memoria interallacciata: con 4-8 moduli si può vedere come le prestazioni dei casi *a)* e *b)* divengano paragonabili). D'altra parte, la soluzione con DMA ha un miglior tempo di servizio, in quanto il trasferimento è effettuato in parallelo all'elaborazione della CPU