Architettura degli Elaboratori

appello del 3 giugno 2010

informazioni e indicazioni sul retro

Domanda 1 *(per tutti)*

Una unità di elaborazione U

- contiene 256 registri di una parola, considerati appartenere a due insiemi disgiunti denominati *A0, .., A127* e *B0, …, B127*. I registri del primo insieme sono in sola lettura e tutti di valore diverso, quelli del secondo modificabili;
- riceve in ingresso messaggi *(OP, X, Y, I)*, con *OP* di 2 bit, *X* e *Y* di una parola, e *I* di 7 bit, e può inviare messaggi *(OUT, ESITO)* con OUT di una parola ed *ESITO* di 1 bit;
- se OP = 00, scrive il valore *Y* nel registro *Bh*, con *h = I*;
- se OP = 01, invia su *OUT* il risultato di *(Bh + Y) div 128*, con *h = I*, e su *ESITO* il valore 1;
- se OP = 1–, invia su OUT il risultato di *(Bj + Y) mod 128*, con *j* tale che *Aj = X*, e su *ESITO* il valore 1. Se non esiste alcun *j* tale che *Aj = X*, invia su *ESITO* il valore 0.

Fornendo adeguate spiegazioni, progettare U in modo che il tempo di elaborazione sia uguale ad *un ciclo di clock*, e valutare il ciclo di clock in funzione del ritardo di stabilizzazione t_p di una porta logica con al più 8 ingressi, considerando che una ALU ha tempo di stabilizzazione $5t_p$.

Di tutte le reti combinatorie utilizzate, diverse da una ALU, deve essere esplicitamente data la definizione, l'espressione logica ed il ritardo di stabilizzazione.

Domanda 2

a - TUTTI

Si consideri un programma eseguito su una architettura la cui gerarchia di memoria memoria principale – cache è costituita solo dalla memoria principale M non interallacciata e dalla cache primaria C. Il programma opera su una certa struttura dati in modo da utilizzarne tutti i blocchi, ognuno mediamente una sola volta e, mediamente, una parola per blocco. Confrontare il tempo di completamento con quello ottenibile da una architettura con la stessa M e senza C. Estendere il confronto al caso di M interallacciata con *m* moduli, determinando un eventuale rapporto ottimo σ/m, se esiste.

b - NEW, OLD-0

Si consideri il seguente frammento di codice D-RISC, da eseguire su una CPU pipeline:

```
        LOAD   Rx, Ry, Ra
        ADD    Ra, Rb, Ra
        IF > 0   Ra, CONT
        STORE   Rd, 0, Ra
CONT:   INCR Rb
        END
```

Si assuma trascurabile tanto la probabilità che il predicato dell'istruzione IF sia *falso*, quanto la probabilità di fault della cache. Si assuma uguale a 2τ il tempo di servizio e di latenza dell'istruzione END.

Fornire due ottimizzazioni alternative e tali da permettere lo stesso valore del tempo di completamento: una che usi la tecnica del delayed branch, e l'altra che non la usi. Valutare il tempo di completamento e dire se è ritenuto ottimo.

continua

c – OLD-1, facoltativo per OLD-0 (ma non sostituzione di uno dei precedenti)

Dire se la seguente affermazione è vera, falsa, o vera sotto certe condizioni, spiegando la risposta:

"è necessario che i canali di comunicazione di un processo P, che esegua un certo servizio di sistema operativo, siano allocati staticamente nella memoria virtuale di P; ciò è dovuto alla creazione dinamica dei processi applicativi che possono utilizzare tale servizio. Lo stesso dicasi per l'allocazione, nella memoria virtuale di P, dei PCB di tali processi applicativi."

Per questo insegnamento sono previsti, durante l'intero a.a., 6 appelli ed un numero massimo di 5 prove.

Gli studenti del vecchio ordinamento devono scegliere se presentarsi sul programma di esame 2009-10 con l'aggiunta del Cap. VII (Processi), sez. 5, 6 e 7, oppure sul programma di esame 2008-09. La scelta va fatta la prima volta che lo studente si iscrive all'esame, e vale per tutti gli eventuali successivi appelli dell'a.a. ai quali lo studente si presenti.

Riportare su tutti i fogli consegnati nome, cognome, numero di matricola, corso di appartenenza, e la sigla ***NEW*** *(per nuovo ordinamento), oppure* ***OLD-0*** *(per vecchio ordinamento, programma 2009-10), oppure* ***OLD-1*** *(per vecchio ordinamento, programma 2008-09).*

I risultati verranno pubblicati sulle pagine web del corso/dei docenti appena disponibili.

Soluzione

Domanda 1 *(per tutti)*

I registri A sono utilizzati solo per il confronto contemporaneo con X, mediante 128 confrontatori in parallelo; per il *j*-esimo, i 32 bit di uscita sono messi in OR generando un bit c_j:

$$c_j = OR(A_j \oplus X)$$

I registri B sono usati come una memoria indirizzata in modo diverso a seconda di OP: per le prime due operazioni esterne, l'indirizzo è I su 7 bit in forma completamente codificata, mentre per la terza operazione esterna l'indirizzo è dato, in forma completamente decodificata, direttamente dai 128 bit $c_0, \ldots, c_{127}$. Ciò comporta l'uso di un selezionatore S, comandato da I, e due commutatori K1, K2, il primo comandato da I, il secondo da $c_0, \ldots, c_{127}$. La *j*-esima porta AND di K2 ha in ingresso B_j e c_j:

$$outK2 = OR\ (B_0\ AND\ c_0, \ldots, B_{127}\ AND\ c_{127})$$

Il microprogramma consta di una sola microistruzione

0. (RDYIN, OP_1, OP_0, ACKOUT = 0 - - -) nop, 0;

   (= 1 0 0 -) reset RDYIN, set ACKIN, Y → B[I], 0;

   (= 1 0 1 0, 1 1 - 0) nop, 0;

   (= 1 0 1 1) reset RDYIN, set ACKIN, (B[I] + Y) *div* 128 → OUT, 1 → ESITO, reset ACKOUT, set RDYOUT, 0;

   (= 1 1 - 1) reset RDYIN, set ACKIN, (B[j] + Y) *mod* 128 → OUT | j: A_j = X, *not* AND ($c_0, \ldots, c_{127}$) → ESITO, reset ACKOUT, set RDYOUT, 0

quindi l'unità è realizzata come singola rete sequenziale di Moore così definita:

funzione delle uscite, ω:

- out = OUT, esito = ESITO, oltre agli indicatori di sincronizzazione in ingresso e in uscita

funzione di transizione dello stato interno, σ:

- in_{Bh} = *when* β_B = 1 *and* h = I *do* Y
- in_{OUT} = *when* β_{OUT} = 1 *do*

  if α_1 = 0 *then* (B[I] + Y) *div* 128

  else (B[j] + Y) *mod* 128 | j: A_j = X
- in_{ESITO} = *when* β_{ESITO} = 1 *do*

  if α_1 = 0 *then* 1

  else not AND ($c_0, \ldots, c_{127}$)
- oltre agli indicatori di sincronizzazione in ingresso e in uscita

dove:

$$\beta_B = RDYIN\ \overline{OP_1}\ \overline{OP_0}$$

$$\beta_{OUT} = \beta_{ESITO} = RDYIN\ ACKOUT\ \overline{OP_1}\ OP_0 + RDYIN\ ACKOUT\ OP_1$$

$$\alpha_1 = OP_1$$

Il ciclo di clock è dato da:

$$\tau = T_\sigma + \delta$$

Il ritardo maggiore della funzione σ è dato dalla cascata dei ritardi delle risorse per la terza operazione esterna: confrontatore ($2t_p$), OR a 32 ingressi ($2t_p$), commutatore K2 (livello AND t_p, livello OR $3t_p$), commutatore delle uscite di K1 e K2 in ingresso all'unica ALU, comandato da α_1 ($2t_p$), ALU ($5t_p$), commutatori all'ingresso di OUT ed ESITO ($2t_p$), comandati da α_1. Tutti i β e α si stabilizzano in parallelo. Le operazioni di *mod* e *div* sono a costo zero (rispettivamente gli 8 bit meno significativi e gli 8 bit più significativi dell'uscita della ALU); il valore da scrivere in OUT va allineato in hardware a 32 bit (rispettivamente i 24 bit più significativi a zero e i 24 bit meno significativi a zero).

Quindi:

$$\tau = 18\ t_p$$

Domanda 2

a – TUTTI

Nel caso di M non modulare, nella versione con cache dati occorre aggiungere, alle elaborazioni effettuate su cache istruzioni e registri ("puro calcolo"), il ritardo del trasferimento di *N* blocchi. Organizzando la memoria in modo da essere capace di trasferire sequenzialmente un blocco in seguito ad un'unica richiesta, questo ritardo è dato da:

$$N\ (2T_{tr} + \sigma\ \tau_M + \tau)$$

Nel caso di assenza di cache dati, gli accessi in memoria costano:

$$N\ (2T_{tr} + \tau_M + 2\tau)$$

A causa della mancanza di località e di riuso sui dati di questo programma, l'architettura con cache dati è penalizzante rispetto a quella senza, in quanto per ogni accesso ad una parola viene inutilmente letto l'intero blocco che la contiene (ovviamente la differenza non sarebbe significativa solo nel caso che la parte di puro calcolo sia di ordine superiore).

Nel caso di memoria interallacciata, l'architettura con cache dati paga un ritardo per trasferimento di blocchi:

$$N\ (2\ T_{tr} + \frac{\sigma}{m}\ \tau_M + m\ \tau)$$

che è ancora penalizzante rispetto al caso senza cache dati, anche se la differenza è meno marcata. La funzione differenza nei tempi di accesso in memoria è monotona decrescente in *m*, quindi il miglior risultato si ottiene con $m = \sigma$: in questo caso la differenza non dipende da τ_M e si riduce a pochi cicli di clock.

b - NEW, OLD-0

Le degradazioni di prestazioni sono dovute alla dipendenza logica indotta dall'istruzione 2 sulla 3 ($k = 1$, $N_Q = 2$) e dal salto nella 3 (che per ipotesi avviene sempre). Complessivamente, viene introdotta una bolla ampia $3t$. La probabilità di dipendenza logica e di salto sono uguali a 1/5; il tempo di servizio è uguale a $8/5\ t$ (si applichi il modello dei costi), quindi il tempo di completamento è uguale a $8t$, con $t = 2\tau$.

L'unica istruzione che può essere spostata, o per applicare il delayed branch o per aumentare la distanza di dipendenza logica, è la *INCR Rb*.

Nel caso di delayed branch:

1. LOAD Rx, Ry, Ra
2. ADD Ra, Rb, Ra
3. IF > 0 Ra, CONT, delayed_branch
4. INCR Rb
5. STORE Rd, 0, Ra
6. CONT: END

ottenendo un tempo di servizio uguale a *7/5 t* e un tempo di completamento uguale a *7t*.

L'altra ottimizzazione è:

1. LOAD Rx, Ry, Ra
2. ADD Ra, Rb, Ra
3. INCR Rb
3. IF > 0 Ra, CONT
4. STORE Rd, 0, Ra
6. CONT: END

ottenendo le stesse prestazioni. In entrambi i casi, infatti, la bolla complessiva passa da *3t* a *2t*.

Il tempo di completamento *7t* è ottimo, in quanto altre ottimizzazioni non sono possibili con una CPU pipeline.

c – OLD-1, facoltativo per OLD-0 (ma non sostituzione di uno dei precedenti)

L'affermazione è falsa, in quanto è possibile allocare staticamente tutti i canali di comunicazione di P nella memoria virtuale di P. Infatti, è noto e costante il numero massimo *N* di processi applicativi che possono essere creati: ad un processo creato viene assegnato un identificatore unico compreso tra 0 e *N-1* e lasciato disponibile da un processo terminato.

I canali d'ingresso di P, se asimmetrici, hanno *N* mittenti (tutti i potenziali processi applicativi). Ogni canale di uscita di P è in realtà costituito da *N* canali (con destinatari tutti i potenziali processi applicativi), ognuno riconoscibile a programma in base ad informazioni associate alle richieste.

P è scritto in modo da ricevere richieste di servizio da uno qualsiasi dei processi applicativi, e da distinguere a programma i canali di uscita.

La compilazione dei processi applicativi provvede ad utilizzare i canali d'ingresso di P (con identificatori noti una volta per tutte) per inviare richieste di servizio, con associate le suddette informazioni, ed a ricevere le risposte al servizio su canali di uscita da P (con identificatori noti una volta per tutte).

Si può introdurre un'ottimizzazione dello spazio di indirizzamento di P, consistente nell'allocare dinamicamente i canali di uscita di P in funzione delle informazioni passate dai processi applicativi nei messaggi di richiesta di un servizio. Ma ciò *NON* cambia la natura del problema studiato, trattandosi di una ottimizzazione, non di una condizione necessaria di fattibilità.

Le stesse considerazioni (inclusa l'eventuale ottimizzazione) si applicano agli *N* PCB dei processi applicativi.