Architettura degli Elaboratori, a.a. 2006-07

Appello del 8 febbraio 2007

Domanda 1

Una unità di elaborazione U contiene un componente logico memoria A di capacità $N = 4K$ parole. Ricevendo da U1 una parola X, sostituisce il valore X a tutti gli elementi di A aventi valore minore di X, ed invia a U2 il numero di sostituzioni effettuate.

Il tempo di stabilizzazione di una ALU e il tempo di accesso di A valgono $5t_p$, dove t_p è il ritardo di stabilizzazione di una porta logica con al più 8 ingressi.

Una *prima versione* di U è espressa dal seguente microprogramma (il significato e il numero di bit di ogni registro usato è a carico dello studente):

0. (RDYIN = 0) nop, 0;

   (= 1) reset RDYIN, set ACKIN, X $\rightarrow$ B, 0 $\rightarrow$ I, 0 $\rightarrow$ COUNT, 1

1. (I_0, ACKOUT = 0 –) segno (A[I_m] – B) $\rightarrow$ S, 2;

   (= 1 0) nop, 1;

   (= 1 1) COUNT $\rightarrow$ OUT, set RDYOUT, reset ACKOUT, 0;

2. (S = 0) I + 1 $\rightarrow$ I, 1;

   (= 1) B $\rightarrow$ A[I_m], I + 1 $\rightarrow$ I, COUNT + 1 $\rightarrow$ COUNT, 1

Si chiede una *seconda versione* di U che renda possibile un netto miglioramento del tempo medio di elaborazione di U: precisamente, esprimendo come $T_1 \cong aN$ il tempo medio di elaborazione della prima versione, la seconda versione deve avere tempo medio di elaborazione esprimibile come $T_2 \cong bN$, con $b < a$.

Scrivere il microprogramma della seconda versione e valutare le costanti *a* e *b*. *È essenziale che venga spiegato chiaramente quale ragionamento è stato fatto per definire la seconda versione.*

Domanda 2

a) Scrivere una versione equivalente della seguente microistruzione che *non* faccia uso di controllo residuo:

4. ($A_0 = 0$) A[I] $\rightarrow$ B, 5; ($A_0 = 1$) A $\rightarrow$ C, 6

dove A e C sono registri di 32 bit, B di 8 bit, e I di 2 bit.

b) Si consideri l'implementazione di una unità cache con il metodo completamente associativo.

- Spiegare perché, concettualmente, è possibile che il ritardo introdotto dall'unita cache, per l'accesso in assenza di fault, sia uguale ad un ciclo di clock.
- Spiegare per quale motivo si preferisce, in realtà, implementare le funzionalità di cui sopra dell'unità cache con un ritardo di due cicli di clock.

c) In una qualunque gerarchia di memoria a due livelli, si consideri la funzione $h = h(\sigma \mid \gamma = cost)$ dove *h* è la probabilità di fault, σ l'ampiezza del blocco e γ la capacità del supporto di livello più basso. Spiegare perché *h* ha un minimo h_{min} in corrispondenza ad un valore finito σ_{min}, e spiegare se la coppia (σ_{min}, h_{min}) varia o no da programma a programma.

d) Spiegare se la seguente affermazione è vera, falsa o vera sotto certe condizioni: il trattamento delle interruzioni ha come effetto di realizzare trasferimenti di dati da unità di I/O al processore.

e) Spiegare se la seguente affermazione è vera, falsa o vera sotto certe condizioni: il codice del supporto a tempo di esecuzione di una primitiva *send* (di una primitiva *receive*) su canale asincrono consiste nell'inserzione (nell'estrazione) del messaggio nella (dalla) coda FIFO del descrittore del canale di comunicazione.

Soluzione

Domanda 1

Il tempo medio di elaborazione della prima versione di U è dato da:

$$T_1 = 2N\tau_1 + 2\tau_1 \cong 2N\tau_1$$

Valutiamo il ciclo di clock di questa versione:

$$T_{\omega PO} = 0$$

$$T_{\omega PC} = T_{\sigma PC} = 2\,t_p$$

$$T_{\sigma PO} = T_A + T_{ALU} = 10\,t_p \text{ (operazione } segno\ (A[I_m] - B) \rightarrow S)$$

$$\tau_1 = 13\,t_p$$

Si tratta del ciclo di clock minimo, in quanto $T_{\omega PO} = 0$ e tutte le operazioni significative che concorrono alla σ_{PO} si stabilizzano contemporaneamente. Quindi:

$$T_1 \cong 26\,t_p\,N$$

$$a = 26\,t_p$$

La prima versione non è caratterizzata dal minimo numero di cicli di clock, in quanto, ad ogni iterazione, la determinazione del valore del segno ed il suo test sono effettuati in due cicli di clock distinti. Sono possibili soluzioni che minimizzano il numero di cicli di clock, riducendo ad uno il numero di cicli di clock per iterazione:

$$T_2 \cong N\ \tau_2$$

Queste soluzioni potranno comportare un ciclo di clock $\tau_2 \geq \tau_1$. Quello che interessa è che T_2 risulterà minore di T_1 se $\tau_2 < 2\ \tau_1$; tale condizione è, ad esempio, soddisfatta da una realizzazione di U corrispondente al seguente microprogramma:

0. (RDYIN = 0) nop, 0;

   (= 1) reset RDYIN, set ACKIN, X $\rightarrow$ B, 0 $\rightarrow$ I, 0 $\rightarrow$ COUNT, 1

1. (I_0, *segno (A[I_m] – B)*, ACKOUT = 0 0 –) I + 1 $\rightarrow$ I, 1;

   (= 0 1 –) B $\rightarrow$ A[I_m], I + 1 $\rightarrow$ I, COUNT + 1 $\rightarrow$ COUNT, 1;

   (= 1 – 0) nop, 1;

   (= 1 – 1) COUNT $\rightarrow$ OUT, set RDYOUT, reset ACKOUT, 0;

ottenuto utilizzando direttamente la variabile di condizionamento *segno (A[I_m] – B)*. La correttezza della PO (modello di Moore) è assicurata dal fatto che tale variabile di condizionamento è ricavata come uscita secondaria di una ALU che esegue solo *A[I_m] – B* e dal fatto che A è indirizzata solo da I_m.

In questa versione si ottiene:

$$T_{\omega PO} = T_A + T_{ALU} = 10\,t_p \text{ (stabilizzazione di } segno\ (A[I_m] - B))$$

$$T_{\omega PC} = T_{\sigma PC} = 2\,t_p$$

$$T_{\sigma PO} = T_K + T_{ALU} = 7\,t_p \text{ (operazioni I + 1 } \rightarrow \text{ I, COUNT + 1 } \rightarrow \text{ COUNT)}$$

$$\tau_2 = 20\,t_p$$

$$T_2 \cong 20\,t_p\,N$$

$$b = 20\,t_p$$

Un risultato ancora migliore ($b = 13\ t_p$, corrispondente a $\tau_2 = \tau_1$) si può ottenere conservando la variabile di condizionamento S in un registro, testandola *ed* aggiornandola nella stessa microistruzione 1, a condizione di inizializzare opportunamente i registri e di trattare opportunamente la terminazione.

Domanda 2

a) In questo esempio A è considerato un array di quattro byte; il controllo residuo consiste nel pilotare, mediante i due bit di uscita del registro I, un commutatore avente in ingresso i quattro byte di A.

Senza usare controllo residuo, la microistruzione sarebbe la seguente:

4. (A_0, I_0, I_1 = 0 0 0) A[0] $\rightarrow$ B, 5;

   (A_0, I_0, I_1 = 0 0 1) A[1] $\rightarrow$ B, 5;

   (A_0, I_0, I_1 = 0 1 0) A[2] $\rightarrow$ B, 5;

   (A_0, I_0, I_1 = 0 1 1) A[3] $\rightarrow$ B, 5;

   (A_0, I_0, I_1 = 1 – –) A $\rightarrow$ C, 6

dove l'indirizzo di A è ottenuto mediante due variabili di controllo in uscita dalla PC.

Questa realizzazione, che usa due variabili di condizionamento in più (I_0, I_1), è caratterizzata da una complessità di progettazione della Parte Controllo che, come ordine di grandezza, è 2^2 volte maggiore rispetto a quella che fa uso di controllo residuo.

b) Nel metodo completamente associativo l'accesso in assenza di fault comporta l'elaborazione di *due funzioni in cascata*:

- la prima per ricavare l'indirizzo di cache a partire dall'indirizzo di memoria principale,
- la seconda per accedere al componente logico memoria cache.

Entrambe hanno ritardo commisurato a quello di un accesso ad una memoria di almeno qualche migliaio di registri. Per la prima funzione si tratta del ritardo di una memoria associativa, sommato al ritardo di una rete combinatoria a pochi livelli di logica (codifica dell'identificatore del blocco di cache).

Sarebbe quindi concettualmente possibile implementare l'accesso in un singolo ciclo di clock, in quanto l'uscita della prima funzione è l'ingresso della seconda. In questa soluzione il ciclo di clock sarebbe approssimativamente (considerando solo il ritardo prevalente della funzione σ_{PO}) uguale al doppio di un tempo di accesso ad una memoria di registri.

Questo ritardo, in realtà, sarebbe sostanzialmente maggiore (circa il doppio) di quello che caratterizza le altre unità sul chip CPU: per il processore domina la somma del ritardo della memoria dei registri generali e di una ALU in cascata, per la MMU il tempo di accesso di una piccola memoria associativa; entrambe le memorie hanno capacità di poche decine di celle.

Poiché raddoppiare il ciclo di clock del chip comporterebbe dimezzare la performance, si preferisce spezzare l'elaborazione dell'unità cache in due cicli di clock più brevi, in ognuno dei quali viene speso il ritardo di un solo accesso ad una memoria di registri. La penalità di un ciclo di clock viene pagata solo per quanto riguarda l'accesso ad informazioni in memoria.

c) Valgono i due limiti:

$$\lim_{\sigma \to 1} h = 1$$

$$\lim_{\sigma \to \gamma} h = 1$$

Il primo è giustificato dal fatto che blocchi troppo piccoli abbasserebbero la *località* all'interno di ogni singolo blocco fino a renderla nulla.

Il secondo limite è in relazione al significato più completo del concetto di *località di un programma*: istante per istante, ogni programma utilizza un certo numero di oggetti (codice, una o più strutture dati), uno o più blocchi di ognuno dei quali deve risiedere nel livello più basso della gerarchia allo scopo di minimizzare la probabilità di fault; tale minimizzazione dipende anche da quanti e quali blocchi per ogni oggetto vi risiedono contemporaneamente. Blocchi più grandi di σ_{min} avrebbero come conseguenza una diminuzione del numero di blocchi del livello più basso, e quindi una diminuzione del numero di oggetti allocabili contemporaneamente.

Per i motivi suddetti (quali e quanti oggetti devono risiedere nel livello più basso della gerarchia allo scopo di minimizzare la probabilità di fault di un programma: concetto di "insieme di lavoro" del programma), i valori di σ_{min} e di h_{min} sono specifici di ogni programma.

d) Falsa. Il trattamento delle interruzioni ha lo scopo di *gestire eventi, verificatisi nel sottosistema di I/O,* che sono *asincroni* rispetto all'esecuzione del processo attualmente in esecuzione sul processore. Tali eventi interesseranno, direttamente o indirettamente, la cooperazione tra vari processi attivi nel sistema (in generale, non necessariamente il processo in esecuzione all'atto dell'interruzione).

Il trattamento dell'interruzione comporta sì un certo trasferimento di dati da unità di I/O a processore (ad esempio, due parole), ma questi dati hanno l'unico significato di *rappresentare lo specifico evento* che I/O intende segnalare.

D'altra parte, il trasferimento di dati vero e proprio, che avverrà in altre fasi, non ha certo come destinazione il processore, bensì aree di memoria virtuale di uno o più processi.

e) Falsa. Il codice di una primitiva di comunicazione comprende, oltre alla copia di valori nel/dal descrittore di canale, altre azioni di *sincronizzazione* e di *scheduling a basso livello*.

Le azioni di sincronizzazione riguardano lo stato di avanzamento del partner (in attesa o meno) ed il numero di messaggi attualmente presenti nel buffer del descrittore di canale. Il descrittore di canale contiene informazioni al riguardo, che vengono restate e modificate.

Le azioni di scheduling a basso livello riguardano la sveglia del partner eventualmente in attesa dell'esecuzione della primitiva in oggetto (nella *send*: sveglia del destinatario, se nel descrittore di canale è segnalata la sua attesa; nella *receive*: sveglia del mittente, se nel descrittore di canale è segnalata la sua attesa), o la sospensione del processo esecutore della primitiva in seguito a condizioni che rendano impossibile il completamento della primitiva stessa (*receive*: assenza di messaggi nel canale, *send*: saturazione del grado di asincronia del canale).

Se, per assurdo, l'affermazione fosse vera, la cooperazione tra processi non potrebbe essere effettuata correttamente, ad esempio i processi potrebbero rimanere indefinitamente in attesa o scambiarsi informazioni non significative.