Architettura degli Elaboratori, 2009-2010

Prima prova di verifica intermedia

17 dicembre 2009

Domanda 1

Una unità di elaborazione U contiene due componenti logici memoria A e B di N = 64K parole da 32 bit, e può comunicare in ingresso con l'unità U_M e in uscita con le unità U_0, … , U_{31}.

Siano OP un valore booleano, J un indirizzo di A e B, W l'identificatore unico di una delle unità U_0, … , U_{31}, e OUT un valore di 32 bit da inviare ad una delle unità U_0, … , U_{31}. U Riceve da U_M messaggi (OP, J, W).

- Se OP = 0: se le locazioni di A e B di indirizzo J hanno uguale contenuto, U calcola *A[J] + B[J]/64*, altrimenti calcola *A[J] – B[J]%128*; il risultato è scritto nella medesima locazione di A ed inviato all'unità identificata da W.
- Se OP = 1: U scambia i contenuti di tutte le celle di A con quelle di B aventi lo stesso indirizzo, e comunica la fine di tale operazione a U_M. U_M prosegue la sua elaborazione e può richiedere una ulteriore operazione senza attendere il risultato della precedente.

Le due operazioni esterne sono equiprobabili.

Sono imposti i seguenti vincoli:

1) *l'operazione con OP = 0 deve essere eseguita in un singolo ciclo di clock,*
2) *i componenti logici memoria hanno un solo ingresso per l'indirizzamento.*

I numeri sono rappresentati in complemento a due. È noto il ritardo di stabilizzazione t_p di una porta logica con al più 8 ingressi. *Le ALU disponibili sono a 32 bit* ed hanno un ritardo di stabilizzazione uguale a $5t_p$. Ogni componente logico memoria A, B è realizzato mediante 16 componenti logici memoria identici con tempo di accesso $4t_p$.

a) É richiesto il microprogramma di U in modo da minimizzare il tempo medio di elaborazione, e la valutazione di tale tempo, spiegando la soluzione.

b) In generale, si indichino le due operazioni elementari $f(A, B) \rightarrow A$ e $g(A) \rightarrow C$ rispettivamente con **μ1** e **μ2**, dove *f* e *g* sono funzioni realizzate con reti combinatorie note. Dire se la microoperazione

μ1, μ2

è computazionalmente equivalente alla sequenza **μ1; μ2**, oppure alla sequenza **μ2; μ1**, oppure a nessuna delle due. Spiegare la risposta *utilizzando concetti delle reti sequenziali*. Esemplificare questo caso con riferimento alla soluzione del punto *a)*.

Domanda 2

Si consideri un programma che opera su una lista linkata, in cui ogni elemento contiene due puntatori ad altrettanti array di interi, ed applica a tali array una procedura data. *Ogni array ha una propria dimensione*. I parametri d'ingresso della procedura, che includono anche le dimensioni dei due array, sono allocati in celle di memoria consecutive a partire dall'indirizzo logico 2^{20}. La procedura non ha parametri di uscita.

a) Compilare il programma in D-RISC, spiegando chiaramente come il compilatore effettua la scelta della, e realizza la, modalità per il passaggio dei parametri alla procedura.

b) Spiegare come il compilatore tiene conto del fatto che certi valori sono di tipo intero ed altri sono di tipo indirizzo.

c) Spiegare quali registri generali sono inizializzati a compilazione; spiegare come vengono rese possibili l'inizializzazione dei registri generali e l'inizializzazione di variabili in memoria; spiegare come si comporta il compilatore nei confronti dei registri generali non inizializzati e delle variabili in memoria non inizializzate.

Soluzione

Domanda 1

a) I due vincoli del problema permettono di individuare, in modo deterministico, la soluzione che minimizza il tempo medio di elaborazione:

- per poter eseguire la prima operazione esterna in un singolo ciclo di clock, non può essere usata una soluzione basata su una variabile di condizionamento complessa come *OR(A[J] ⊕ B[J])* oppure *zero(A[J] − B[J])*: le memorie devono essere a singolo indirizzamento e, poiché la seconda operazione esterna deve utilizzare un registro diverso da J per il loro indirizzamento, una tale variabile di condizionamento verrebbe a dipendere da variabili di controllo e non sarebbe soddisfatta la condizione necessaria per la correttezza della Parte Operativa;
- invece, la funzione della prima operazione esterna deve essere implementata interamente come microoperazione; questo si ottiene realizzando *l'intera* espressione condizionale

  $$F(A[J], B[J]) = (\text{if } (A[J] = B[J]) \text{ then } A[J] + B[J]/64 \text{ else } A[J] - B[J]\%128$$

  come una rete combinatoria costituita dalla composizione di componenti standard;
- le operazioni di divisione intera e di modulo, con secondo operando potenze di due, sono realizzate, *con ritardo nullo*, semplicemente prendendo campi di bit all'uscita della memoria B (outB) e concatenandoli con un opportuno numero di zeri per tener conto che la ALU (addizione/sottrazione) deve operare su 32 bit. Tenendo anche contro del bit del segno (in posizione 31):

  $$B[J]/64 = outB_{31} \circ sei\_zeri \circ outB[30 \ldots 6]$$

  $$B[J]\%128 = outB_{31} \circ ventiquattro\_zeri \circ outB[6 \ldots 0]$$

  dove le costanti *sei_zeri* e *ventiquattro_zeri*, di ovvio significato, sono cablate nella PO;
- la rete combinatoria *F* è realizzata come la cascata di un commutatore, con ingressi le due parole sopra indicate, e di una ALU (addizione/sottrazione), dove la variabile di controllo, tanto della ALU quanto del commutatore, è data da *OR(A[J] ⊕ B[J])*. Quest'ultima scelta comporta un ritardo inferiore rispetto ad una realizzazione come *zero(A[J] − B[J])*, quindi contribuisce a *minimizzare il ciclo di clock;*
- la variabile di condizionamento *ACK[W]*, che fa uso di controllo residuo per operare parametricamente sulle interfacce con $U_0, \ldots, U_{31}$ in funzione del valore W, ha ritardo di stabilizzazione non nullo, ma questo è *ineliminabile* per soddisfare il vincolo che la prima operazione esterna deve essere eseguita in un singolo ciclo di clock e non fare esplodere la complessità di progettazione della PC.

La seconda operazione esterna è facilmente descrivibile da un microprogramma iterativo, completato in *N* cicli di clock, che utilizza esclusivamente le variabili di condizionamento I_{16} e *ACKM_OUT*, con *I* registro contatore di 32 bit (inizializzato a zero e incrementato fino al valore massimo 64K). Di conseguenza

$$T \sim N\tau/2.$$

Il microprogramma è il seguente (W di 5 bit, J di 16 bit, I di 32 bit, $I_m = I[15 \ldots 0]$):

0. (RDYM_IN, OP, ACK[W] = 0 − − , 1 0 0) nop, 0;

   (= 1 0 1) reset RDYM_IN, set ACKM_IN, F(A[J], B[J]) → (A[J], OUT[W]), set RDY[W], reset ACK[W], 0;

   (= 1 1 −) reset RDYM_IN, set ACKM_IN, 0 → I, 1

1. (I_0, ACKM_OUT = 0 −) $A[I_m] \rightarrow B[I_m]$, $B[I_m] \rightarrow A[I_m]$, I + 1 → I, 1;

   (= 1 0) nop, 1; (= 1 1) set RDYM_OUT, reset ACKM_OUT, 0

La sincronizzazione con U_M *non* è a domanda-risposta. La risposta di U a U_M è un messaggio vuoto, implementato dalla coppia RDYM_OUT, ACKM_OUT.

Nella Parte Operativa la stessa ALU, usata per l'addizione/sottrazione all'interno della funzione *F*, viene usata anche per l'incremento di I, con una variabile di controllo in più generata da PC; quindi esiste un commutatore anche sul primo ingresso della ALU (scelta tra outA e I).

Il tempo di accesso delle memorie A e B è dato da

$$t_a =$$

$$= 4t_p \text{ (tempo di accesso dei 16 componenti di capacità 4K, indirizzate dai 12 bit meno significativi dell'indirizzo) +}$$

$$+ 3t_p \text{ (commutatore a 16 ingressi rappresentati dalle uscite dei 16 componenti, comandato dai 4 bit più significativi dell'indirizzo)}$$

$$= 7t_p.$$

Per la valutazione del ciclo di clock τ si ha:

- $T_{\omega PO} = 3t_p$

  stabilizzazione del commutatore a 32 ingressi per ottenere ACK[W];

- $T_{\sigma PO}$ = ritardo di *if (A[J] = B[J]) then A[J] + B[J]/64 else A[J] − B[J]%128* $\rightarrow$ *(A[J], OUT[W]) =*

  $= t_K$ *(indirizzamento memorie con* I_m *e J)* $+ t_a + 2t_p$ *(OR esclusivo a 32 bit)* $+ 2t_p$ *(OR a 32 bit)* $+ t_K$ *(commutatori in ingresso alla ALU)* $+ t_{ALU} + t_K$ *(dato in ingresso alla memoria A)* $= 22\ t_p$

  Non viene pagato il tempo di accesso per la scrittura in A, completamente mascherato dagli altri ritardi della microoperazione; lo stesso dicasi per la stabilizzazione del selezionatore del segnale di abilitazione alla scrittura nelle interfacce di uscita verso $U_0, \ldots, U_{31}$.

  L'altra transizione di stato relativamente "pesante" è causata da $A[I_m] \rightarrow B[I_m]$, $B[I_m] \rightarrow A[I_m]$, avente ritardo di stabilizzazione $2t_K + t_a = 11t_p$;

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

  in quanto nessun termine AND, OR delle espressioni logiche di tali funzioni contiene più di 8 variabili.

Il ciclo di clock, valutato con la formula ottenuta con procedimento indipendente dal numero della frasi, è quindi dato da:

$$\tau = 28t_p$$

Il tempo medio di elaborazione:

$$T \sim N\tau/2 = 14\ N\ t_p$$

b) La sequenza **μ2; μ1** è equivalente alla microoperazione parallela **μ1, μ2**.

Essendo il modello di unità di elaborazione realizzato mediante due reti sequenziali sincrone, alla fine del ciclo di clock, in cui viene eseguita **μ1, μ2**, i valori delle variabili dello stato successivo in_A e in_C nella rete PO sono stabili rispettivamente ai valori di *f(A, B)* e di *g(A)*; in PO l'effettiva transizione di stato delle variabili A e C avverrà all'inizio del prossimo ciclo di clock.

Per l'applicazione delle condizioni di Bernstein, la sequenza **μ1; μ2** non è equivalente alla microoperazione parallela **μ1, μ2**.

Nel punto *a)* una applicazione di questo caso si ha nella microoperazione:

$$A[I_m] \rightarrow B[I_m],\ B[I_m] \rightarrow A[I_m],\ I + 1 \rightarrow I$$

Domanda 2

a) I parametri d'ingresso della procedura sono gli indirizzi logici base di due array e le rispettive dimensioni: *passaggio dei parametri per indirizzo per quanto riguarda gli array e per valore per quanto riguarda le dimensioni*.

Il generico elemento di lista consta di 6 parole: indirizzo base del primo array, dimensione del primo array, indirizzo base del secondo array, dimensione del secondo array, indicatore di fine lista, puntatore all'elemento successivo in lista.

Il programma principale legge i quattro parametri dal generico elemento di lista e li scrive nelle quattro locazioni di memoria di indirizzo logico 1Mega + 0, … , 1Mega + 3: *passaggio dei parametri via memoria*.

Il valore 1Mega è inizializzato in un registro *Ractual* usato dal programma principale. La procedura, in generale, farà uso di un registro diverso contenente lo stesso valore.

La scansione della lista utilizza lo schema usuale. La struttura dati Testa della Lista (così come ogni puntatore di un elemento all'elemento successivo) è costituita da due parole: un booleano che indica se la lista è vuota (se è l'ultimo elemento della lista) e un indirizzo al primo (al successivo) elemento se esiste. Sia *Rtesta* l'indirizzo del registro generale inizializzato all'indirizzo di Testa.

Il codice compilato del programma principale è il seguente, dove gli altri registri hanno significato auto-esplicativo:

```
        LOAD   Rtesta, 0, Rfine
        IF < > 0   Rfine, FINE
        LOAD   Rtesta, 1, Rpun
LOOP:   LOAD   Rpun, 0, Rbase1
        LOAD   Rpun, 1, Rdim1
        LOAD   Rpun, 2, Rbase2
        LOAD   Rpun, 3, Rdim2
        STORE  Ractual, 0, Rbase1
        STORE  Ractual, 1, Rdim1
        STORE  Ractual, 2, Rbase2
        STORE  Ractual, 3, Rdim2
        CALL   Rproc, Rret
        LOAD   Rpun, 4, Rfine
        LOAD   Rpun, 5, Rpun
        IF = 0   Rfine, LOOP
FINE:   END
```

b) Il compilatore per la macchina D-RISC rappresenta sia gli interi (come le dimensioni degli array) che gli indirizzi di memoria virtuale con parole di 32 bit. Questo ha ripercussione sia nell'allocazione della memoria virtuale che nell'allocazione di registri.

c) Nel programma principale solo i tre registri generali *Rtesta*, *Ractual*, *Rproc* sono inizializzati a tempo di compilazione. Gli altri registri (*Rfine*, *Rpun*, *Rbase1*, *Rdim1*, *Rbase2*, *Rdim2*, *Rret*) sono allocati ma non inizializzati.

La memoria virtuale del processo forma il file binario eseguibile (file oggetto), nel quale alcune informazioni hanno un valore inizializzato e le altre hanno un valore non significativo.

All'atto del caricamento in memoria (*creazione* del processo), una parte delle informazioni in memoria virtuale verrà copiata in opportune locazioni della memoria principale M: a queste informazioni appartiene certamente almeno il PCB e le prime istruzioni, e possibilmente almeno i primi dati riferiti. Dunque, alcune locazioni di M assumeranno il desiderato valore iniziale contenuto nel file eseguibile e scelto a tempo di compilazione (le informazioni non caricate a tempo di creazione verranno caricate successivamente a tempo di esecuzione).

Quando il processo passerà in stato di esecuzione, l'operazione di *commutazione di contesto* provocherà che le immagini dei registri generali e di IC, presenti nel PCB, vengano copiate nei registri del processore, provocando così la loro effettiva inizializzazione.

Per quanto riguarda i registri generali non inizializzati, il compilatore si limita semplicemente a lasciare non specificato il contenuto delle rispettive parole nella parte PCB del file eseguibile, allo stesso modo con cui viene trattata qualunque altra informazione in memoria non inizializzata.