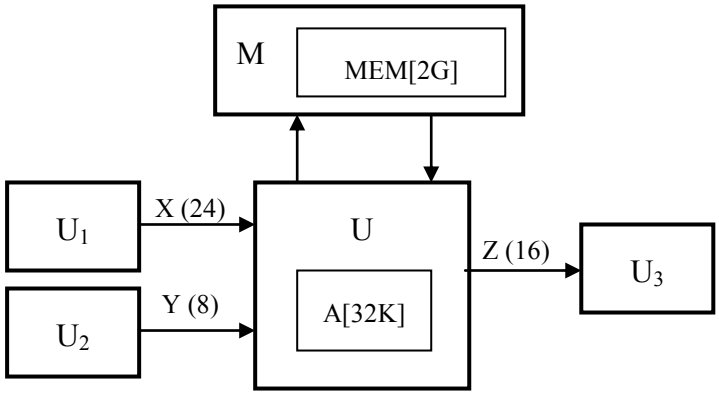
Architettura degli Elaboratori, 2010-2011

Prima prova di verifica intermedia

2 febbraio 2011

Domanda 1

Si consideri il seguente sistema a livello firmware:



L'unità U contiene un componente logico memoria A di capacità 32K parole, e l'unità M un componente logico memoria MEM di capacità 2G parole. Le operazioni esterne di M sono lettura o scrittura di una parola su richiesta di U (per semplicità, senza esplicita informazione di esito). M ha ciclo di clock uguale a 11 volte quello di U. I collegamenti tra unità hanno latenza di trasmissione uguale a 4 cicli di clock di U.

Le informazioni contenute in MEM e in A sono logicamente organizzate in blocchi disgiunti, ciascuno costituito da 128 parole contigue.

Ricevendo da U_1 e U_2 le informazioni X (24 bit) e Y (8 bit) rispettivamente, U identifica l'operazione esterna come segue:

- se Y è uguale a due o tre dei byte di X: operazione esterna 0;
- se Y è uguale ad uno e solo uno dei byte di X: operazione esterna 1;
- se Y è diverso da ognuno dei tre byte di X: operazione esterna 2.

Le operazioni esterne hanno il seguente effetto:

- operazione esterna 0: copiare il blocco di MEM identificato da X nel blocco di A identificato da Y;
- operazione esterna 1: copiare il valore assoluto di ogni parola del blocco di A identificato da Y nelle corrispondenti parole del blocco di MEM identificato da X;
- operazione esterna 2: inviare a U_3 il numero di parole di A che hanno valore positivo e uguale a una potenza di 2.

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. Una ALU ha ritardo di stabilizzazione uguale a $5t_p$. Il componente logico memoria A è a singolo indirizzamento e ha tempo di accesso $4t_p$.

É richiesto il microprogramma di U con il requisito di *minimizzare il tempo medio di elaborazione dell'operazione esterna 2*, e la valutazione di tutti i tempi medi di elaborazione.

Prima del microprogramma deve essere riportata la spiegazione di quali scelte sono effettuate per soddisfare il suddetto requisito sui tempi di elaborazione.

Ogni rete combinatoria utilizzata, che sia diversa da una ALU o un commutatore, deve essere completamente definita e implementata.

Domanda 2

a) Compilare in D-RISC un programma che effettui la moltiplicazione matrice-vettore secondo il seguente algoritmo:

int A[M][M]; int B[M], C[M];

$\forall$ *i = 0 .. M-1*:

{ *C[i] = 0;*

$\forall$ *j = 0 .. M-1*:

C[i] = C[i] + A[i][j] ∗ B[j]

}

Il programma compilato deve consistere di un programma principale e di una procedura che opera su due vettori *V1*, *V2* di *M* interi, con $M > 0$, e restituisce l'intero *x*:

$$x = \sum_{h=0}^{M-1} V1[h] * V2[h]$$

con passaggio di parametri via memoria.

b) Supponendo *M* dell'ordine del migliaio e il tempo di accesso in memoria t_a dell'ordine del centinaio di cicli di clock della CPU, valutare il tempo di completamento in modo *approssimato*, in funzione solo di *M* e t_a, stimando l'entità di tale approssimazione *senza* ricorrere al calcolo dettagliato del tempo di completamento.

Soluzione

Domanda 1

Occorre realizzare, mediante una rete combinatoria, una funzione che,operando su ogni coppia di valori *X* e *Y* ricevuti da U_1 e U_2, identifichi l'operazione esterna da eseguire:

$$OP = F\ (X.byte_0, X.byte_1, X.byte_2, Y)$$

con *OP* di due bit (OP_0, OP_1), tali che:

$OP_0\ OP_1 = 0\ 0$ *:* Y è uguale a due o tre dei byte di X: *operazione esterna 0*

$OP_0\ OP_1 = 0\ 1$ *:* Y è uguale ad uno e solo uno dei byte di X: *operazione esterna 1*

$OP_0\ OP_1 = 1\ -$ *:* Y è diverso da ognuno dei tre byte di X: *operazione esterna 2*

La funzione *F* è definita nel seguente modo:

- vengono confrontati contemporaneamente i tre byte di *X* con il byte *Y*, e gli 8 bit di ogni singolo risultato messi in OR. Siano W_0, W_1, W_2 le tre variabili booleane risultanti:

$$i = 0, 1, 2: W_i = OR\ (X.byte_i \oplus Y)$$

- dalla specifica delle tre operazioni esterne si ricava una tabella di verità con variabili d'ingresso W_0, W_1, W_2 e variabili di uscita OP_0, OP_1. Da essa si ottengono le espressioni logiche (del tutto intuitive applicando direttamente le specifiche):

$$OP_0 = W_0\ W_1\ W_2$$

$$OP_1 = \overline{W_0}\ W_1\ W_2 + W_0\ \overline{W_1}\ W_2 + W_0\ W_1\ \overline{W_2}$$

La realizzazione di *F* è una rete combinatoria con ritardo di stabilizzazione

$$T_F = 2t_p \text{ (confrontatori)} + t_p \text{ (porta OR a 8 ingressi)} + 2t_p = 5t_p$$

Effettuando la decodifica del codice operativo nella prima microistruzione con variabili di condizionamento complesse *F(X.byte0, X.byte1, X.byte2, Y)*, nella valutazione del ciclo di clock avremmo almeno $T_{\omega PO} = 5t_p$.

Le operazioni esterne 0 e 1 non hanno particolari problemi dal punto di vista della scrittura del microprogramma ottimo, in entrambi i casi trattandosi di un loop ripetuto 128 volte con variabili di condizionamento semplici. Nell'operazione 1 il calcolo del valore assoluto viene effettuato con una rete combinatoria:

abs(num) = if (bit_segno(num) = 0) then num else negazione_complemento_a_due(num)

così da non introdurre inutilmente una variabile di condizionamento complessa. La funzione è realizzabile con una ALU controllata dal bit del segno dello stesso operando *num*.

L'operazione esterna 2 consta di un loop ripetuto 32K volte. Per minimizzarne il tempo medio di elaborazione, occorre anzitutto fare uso di una rete combinatoria che realizzi la funzione booleana:

$$P = G(a) = if\ (a > 0)\ and\ (a = 2^h)\ |\ 0 \le h \le 30$$

con *a* contenuto della generica locazione di *A*.

Dalla proprietà delle potenze di 2 si ottiene quindi la funzione G come OR di 31 termini AND, ognuno di 32 variabili (i bit di *a*), tutti i termini AND aventi a_{31} negato, ed ogni termine AND avente una sola variabile affermata e tutte le altre negate:

$$P = \overline{a_{31}}\ a_{30}\ \overline{a_{29}}\ \ldots\ \overline{a_0} + \overline{a_{31}}\ \overline{a_{30}}\ a_{29}\ \ldots\ \overline{a_0} + \ \ldots + \overline{a_{31}}\ \overline{a_{30}}\ \overline{a_{29}}\ \ldots\ a_0$$

La rete combinatoria corrispondente ha ritardo $2t_p$ (livello AND) + $2t_p$ (livello OR), a cui va sommato il tempo di accesso nella memoria *A*: complessivamente *almeno* $T_G = 8t_p$ ("almeno": a seconda che esistano altri ritardi significativi, ad esempio per l'indirizzamento di *A*).

Per minimizzare il tempo medio di elaborazione dell'operazione 2, occorre inoltre che il corpo del loop consti di una sola microistruzione. Questo si può ottenere

i) con una variabile di condizionamento complessa, nella quale si valuti la funzione *G*,

ii) con una variabile di condizionamento semplice, scrivendo il valore di *G* in un registro di un bit *P*, ed anticipandone opportunamente l'esecuzione di un ciclo di clock.

La prima soluzione comporta che, nella valutazione del ciclo di clock, sarebbe almeno $T_{\omega PO} = 8t_p$, che è ancora maggiore del tempo di stabilizzazione della funzione *F*.

In conclusione, la soluzione che minimizza il tempo di elaborazione dell'operazione esterna 2 è la *ii)*, a condizione che

- *nella valutazione del ciclo di clock sia* $T_{\omega PO} = 0$.

Questo si ottiene impiegando *due cicli di clock per la ricezione di X, Y e la decodifica del codice operativo*, utilizzando il primo ciclo per scrivere il valore della funzione *F* in un registro *OP* di 2 bit. Il ciclo di clock speso in più non ha alcun effetto sui tempi medi di elaborazione, *anzi*: è fondamentale per minimizzare il tempo medio di elaborazione dell'operazione esterna 2, e minimizza anche i tempi di elaborazione delle operazioni 0 e 1 in quanto permette di minimizzare il ciclo di clock (come detto, il loop nelle operazioni 0 e 1 impiega solo variabili di condizionamento semplici).

Il microprogramma che ne deriva è il seguente:

****** *Attesa X e Y, decodifica e inizializzazione. L'interfaccia verso la memoria usa il registro IND per l'indirizzamento. Il registro I è di 8 bit, per controllare il loop delle operazioni 0 e 1. Il registro J è di 16 bit, per controllare il loop dell'operazione 2. C contiene il risultato dell'operazione 2.*

0. (RDY_1, RDY_2 = 0 0, 0 1, 1 0) nop, 0;

   (= 1 1) reset RDY_1, set ACK_1, reset RDY_2, set ACK_2, *F(X, Y)* $\rightarrow$ *OP*, X ° *settezeri* $\rightarrow$ INDM,

   Y ° *settezeri* $\rightarrow$ INDA, 1

1. (OP_0, OP_1 = 0 0) INDM $\rightarrow$ IND, 'read' $\rightarrow$ OPM, set RDYOUTM, 1 $\rightarrow$ I, INDM + 1 $\rightarrow$ INDM, 2;

   (= 0 1) INDM $\rightarrow$ IND, *abs*(A[INDA]) $\rightarrow$ DATAOUT, 'write' $\rightarrow$ OPM, set RDYOUTM, 1 $\rightarrow$ I,

   INDM + 1 $\rightarrow$ INDM, INDA + 1 $\rightarrow$ INDA, 3;

   (= 1 –) *G(A[0])* $\rightarrow$ *P*, 1 $\rightarrow$ INDA, 1 $\rightarrow$ J, 0 $\rightarrow$ C, 4

****** *Operazione esterna 0*

2. (I_0, RDYINM = – 0) nop, 2;

   (= 0 1) reset RDYINM, DATAIN $\rightarrow$ A[INDA], INDM $\rightarrow$ IND, 'read' $\rightarrow$ OPM, set RDYOUTM,

   I + 1 $\rightarrow$ I, INDM + 1 $\rightarrow$ INDM, INDA + 1 $\rightarrow$ INDA, 2;

   (= 1 1) reset RDYINM, DATAIN $\rightarrow$ A[INDA], 0

****** *Operazione esterna 1*

3. (I_0, RDYINM = – 0) nop, 3;

   (= 0 1) reset RDYINM, INDM $\rightarrow$ IND, *abs*(A[INDA]) $\rightarrow$ DATAOUT, 'write' $\rightarrow$ OPM,

   set RDYOUTM, I + 1 $\rightarrow$ I, INDM + 1 $\rightarrow$ INDM, INDA + 1 $\rightarrow$ INDA, 3;

   (= 1 1) reset RDYINM, 0

****** *Operazione esterna 2*

4. (J_0, P, ACK_3 = 0 0 –) J + 1 → J, INDA + 1 → INDA, *G(A[INDA])* → *P*, 4;

   (= 0 1 –) J + 1 → J, INDA + 1 → INDA, *G(A[INDA])* → *P*, C + 1 → C, 4;

   (= 1 – 0) nop, 4;

   (= 1 0 1) reset ACK_3, set RDY_3, C → OUT3, 0;

   (= 1 1 1) reset ACK_3, set RDY_3, C + 1 → OUT3, 0;

Trascurando i cicli di clock spesi nelle microistruzioni 0 e 1, i tempi medi di elaborazione valgono:

$$T_0 = T_1 = 128\ (\tau + t_a) = 128\ (\tau + 2T_{tr} + \tau_M) = 2560\ \tau$$

$$T_2 = 32K\ \tau$$

Per la valutazione del ciclo di clock, abbiamo $T_{\omega PO} = 0$. Una volta appurato facilmente che $T_{\omega PC} = T_{\sigma PC} = 2t_p$, valutiamo $T_{\sigma PO}$: il numero di ALU necessarie per il parallelismo delle micro operazioni è 3, usando tutte ALU a 32 bit e completando con opportuni zeri i numeri rappresentati con meno di 32 bit; tali ALU hanno quindi commutatori in ingresso. L'operazione più costosa è *abs(A[INDA])*, che comporta i seguenti ritardi in serie: attraversamento di un commutatore per l'indirizzamento di *A* (con la costante *0* oppure con il contenuto di *INDA*), lettura di *A*, attraversamento del commutatore di una ALU e ritardo di tale ALU. Quindi,

$$T_{\sigma PO} = 13\ t_p$$

da cui

$$\tau = 16\ t_p$$

Domanda 2

a) I vettori *V1* e *V2* sono passati alla procedura *per riferimento via memoria*: gli indirizzi base si trovano in due locazioni di memoria virtuale indirizzate dai registri generali **Rvet1** e **Rvet2**. La dimensione *M* è passata *per valore, via la locazione di memoria virtuale* indirizzata dal registro **Rdim**. Il risultato *x* è passato *per valore, via la locazione di memoria virtuale* indirizzata dal registro **Rris**. I registri citati sono inizializzati a tempo di compilazione; gli altri che compariranno nel codice della procedura sono temporanei.

Applicando le regole di compilazione, il codice della procedura è:

```
            LOAD   Rvet1, 0, RV1          // indirizzo base vettore V1
            LOAD   Rvet2, 0, RV2          // indirizzo base vettore V2
            LOAD   Rdim, 0, Rsize         // dimensione dei vettori
            CLEAR  Rh
            CLEAR  Rx
LOOP_proc:  LOAD   RV1, Rh, Rt1           // componente di V1
            LOAD   RV2, Rh, Rt2           // componente di V2
            MUL    Rt1, Rt2, Rt1
            ADD    Rx, Rt1, Rx
            INCR   Rh
            IF <   Rh, Rsize, LOOP_proc
            STORE  Rris, 0, Rx            // scrittura parametro risultato in memoria
            GOTO   Rret
```

Il programma principale è (sono inizializzati a tempo di compilazione RbaseA, RbaseB, RbaseC, Ri, RM, Rproc; quelli per interfacciarsi con la procedura sono stati definiti sopra, gli altri sono temporanei):

```
LOOP:       STORE  Rvet1, 0, RbaseA       // passaggio indirizzo base vettore A[i][*]
            STORE  Rvet2, 0, RbaseB       // passaggio indirizzo base vettore B
            STORE  Rdim, 0, RM            // passaggio dimensione M
            CALL   Rproc, Rret
            LOAD   Rris, 0, Rc            // lettura parametro risultato da memoria
            STORE  RbaseC, Ri, Rc                 // scrittura risultato in C[i]
            ADD    RbaseA, RM, RbaseA
            INCR   Ri
            IF <   Ri, RM, LOOP
            END
```

Si noti che, in assenza di ulteriori ipotesi, il passaggio dell'indirizzo base di B e della dimensione M deve essere ripetuto per ogni invocazione della procedura (ad esempio, non potrebbero essere passati solo la prima volta se la procedura fosse condivisa e il programma principale fosse interrompibile).

b) Visto l'ordine di grandezza di *M*, ed essendo il tempo di completamento $O(M^2)$, è sufficiente valutare solo l'impatto del *loop della procedura*:

- questo consta di 6 istruzioni ripetute M^2 volte, per cui sono necessari $6M^2$ accessi alla memoria per la chiamata delle istruzioni stesse;
- l'utilizzo dei dati di A comporta M^2 letture dalla memoria, così come per i dati di B. Quindi, approssimativamente si hanno $2M^2$ accessi in memoria per i dati.

Complessivamente, la valutazione approssimata fornisce:

$$T_c \sim 8\, M^2\, t_a$$

L'approssimazione, pur per difetto, è del tutto accettabile per quanto riguarda aver trascurato tutti gli accessi in memoria, sia in LOAD che in STORE, effettuati $O(M)$ volte, inclusi quelli per il passaggio dei parametri: l'errore è dell'ordine di 0,1% o inferiore.

Per quanto riguarda aver trascurato il tempo medio speso in cicli di clock del processore per chiamata, decodifica ed esecuzione delle istruzioni, nel loop della procedura il numero di tali cicli è dell'ordine di poche decine, ripetuti M^2 volte, quindi molto minore di $8t_a$: l'errore è dell'ordine di 1% o inferiore.