Architettura degli Elaboratori, a.a. 2005-06

Correzione della Prima Prova di Verifica Intermedia

Domanda 1

Una unità di elaborazione U contiene una memoria *A* di capacità 16K parole ed una memoria *B* di capacità 256 parole (parola di 32 bit).

U riceve da U_a messaggi del tipo *(OP, J, X, Y, INDA)*, con *OP* di 1 bit, *J* di 2 bit, *X* e *Y* interi, e *INDA* indirizzo di A, ed invia a U_b messaggi (OUT) di valore intero.

Se OP = 0, il massimo tra i valori di *X* ed *Y* è scritto nella locazione di *B* il cui indirizzo è dato dal valore del byte *J*-esimo della locazione di *A* avente indirizzo *INDA*.

Se OP = 1, viene inviata a U_b una sequenza di 256 valori, il generico dei quali è calcolato come $B[i] + X$, con $i = 0 \ldots 255$.

È noto il ritardo t_p di una porta logica con al massimo 8 ingressi. Una ALU ha ritardo uguale a $5t_p$. La durata dell'impulso di clock è uguale a t_p. La memoria *B* è data, ed ha tempo di accesso uguale a $3t_p$. La memoria *A* deve essere realizzata mediante composizione di memorie con le stesse caratteristiche della memoria *B*.

a) Scrivere il microprogramma di U, spiegando sinteticamente le principali scelte, e determinare il tempo medio di elaborazione di ognuna delle due operazioni esterne in funzione di t_p.

<u>*Facoltativo*</u>: spiegare se, nella soluzione adottata per U, è stato fatto uso del minimo numero di variabili d'ingresso delle reti combinatorie della Parte Controllo e, nel caso ciò non sia, indicare una soluzione che minimizzi tale numero.

Le seguenti domande hanno carattere di generalità :

b) dare una definizione di ciclo di clock di una unità di elaborazione;

c) spiegare perché il risultato della funzione delle uscite di PC dipende, ad ogni ciclo di clock, sia dallo stato interno presente che dallo stato d'ingresso di PC;

d) nella realizzazione di una certa PO, la modifica del contenuto di un registro di un bit, la cui uscita è una variabile di condizionamento, è ottenuta mediante una rete combinatoria tra i cui ingressi figurano anche variabili di controllo. Dire se la seguente affermazione è vera o falsa, spiegando la risposta: " di conseguenza, la realizzazione della PO non soddisfa la condizione necessaria per la correttezza".

Domanda 2

a) Per ognuna delle seguenti espressioni, dove RG indica la memoria dei registri generali del processore e MV la memoria virtuale del programma:

1) RG [5]
2) MV [RG [5]]
3) MV [RG [5] + RG [13]]
4) MV [43971]
5) MV [MV [43971]]
6) MV [MV [RG [5] + RG [13]]]
7) MV [IC + 3825]

dire se esiste una sequenza di (una o più) istruzioni assembler Risc (Cap. V) per assegnare il valore dell'espressione ad un certo registro generale d'indirizzo *x* e, nei casi in cui esita, mostrare tali sequenze.

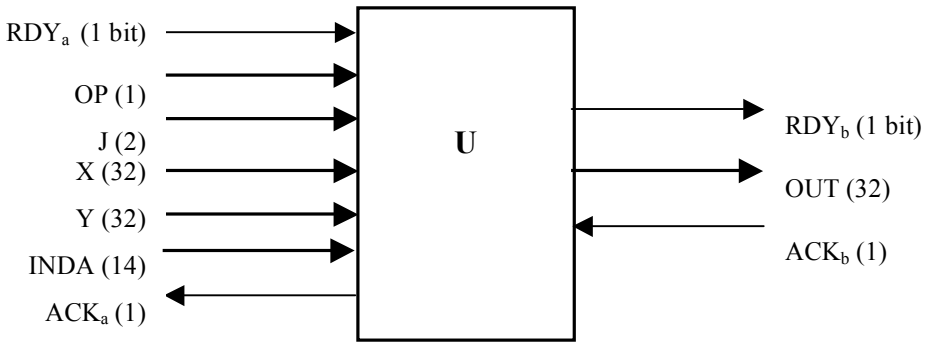
b) Dare il formato e la semantica delle istruzioni assembler Risc: STORE, IF<; GOTO, quest'ultima con indirizzamento indiretto via registro.

c) Spiegare come si implementa il fatto che tutte quelle variabili o costanti, allocate in memoria o in registri generali, i cui valori iniziali sono fissati a tempo di compilazione, vengano effettivamente inizializzate a tali valori a tempo di esecuzione.

d) <u>*Facoltativo*</u>: Volendo tradurre un programma che calcola il massimo degli elementi di un array di interi, indicare almeno una ottimizzazione che un compilatore può riuscire ad utilizzare rispetto ad un interprete.

Domanda 1

a) La figura seguente mostra i nomi dei registri delle interfacce, a transizione di livello, dell'unità U:



Scelte principali

Nel microprogramma dell'operazione esterna con OP = 0 (O_0), la scelta fondamentale è quella della variabile di condizionamento per valutare se X ≥ Y:

i) una soluzione è quella di testare *segno(X – Y)* nella prima microistruzione, riuscendo quindi ad eseguire O_0 in un solo ciclo di clock;

ii) un'altra soluzione consiste nel memorizzare, nella prima microistruzione, il risultato di *segno(X – Y)* in un registro S, testato nella microistruzione successiva, eseguendo O_0 quindi in due cicli di clock.

Usando la prima soluzione viene minimizzato il tempo di elaborazione T_0 di O_0, ma il maggior valore del ciclo di clock, che ne risulta, va a discapito del tempo di elaborazione T_1 di O_1. La seconda soluzione, invece, minimizza T_1. Scegliendo la seconda soluzione, occorre però osservare che la minimizzazione del ciclo di clock deve essere ottenuta minimizzando, oltre che il ritardo della funzione ω_{PO}, anche quello della σ_{PO}: ciò si ottiene anticipando la lettura della memoria A nella prima microistruzione.

[Ovviamente, soluzioni che testino *segno(X – Y)*, ma che usino più di un ciclo di clock, non sono accettabili in quanto con certezza sono inefficienti (mentre la scelta a priori tra le soluzioni "buone" *i)* e *ii)* è un problema indecidibile se non in presenza di altre informazioni).]

[È fondamentale rendersi conto che il ciclo di clock viene determinato considerando i massimi ritardi che le quattro funzioni di PC e PO assumono in qualunque microistruzione. In altre parole, il ciclo di clock non viene determinato calcolandolo separatamente per ogni microistruzione e poi prendendo il massimo: questo procedimento avrebbe complessità lineare nel numero delle microistruzioni, invece che costante rispetto a tale numero.]

Per ricavare l'indirizzo della memoria B nella O_0, la memoria A viene vista come un *array bidimensionale di byte* (16K righe e 4 colonne). L'implementazione di *A[INDA][J]* (il byte *J*-esimo della parola di A avente indirizzo *INDA*) è ottenuta, con la tecnica del controllo residuo, mediante un commutatore KA, avente in ingresso i quattro byte della parola letta da A e controllato dai bit in uscita a *J* [inserire una figura].

Il microprogramma di O_1 utilizza un registro contatore di passi, *I*, di 9 bit, dei quali il più significativo I_0 è la variabile di condizionamento per controllare il loop, ed i rimanenti bit I_m servono a indirizzare la memoria B.

Microprogramma, soluzione i):

0. (RDY$_a$, OP, segno (X – Y) = 0 – –) nop, 0;

   (= 1 0 0) reset RDY$_a$, set ACK$_a$, X → B[A[INDA][J]], 0;

   (= 1 0 1) reset RDY$_a$, set ACK$_a$, Y → B[A[INDA][J]], 0;

   (= 1 1 –) reset RDY$_a$, set ACK$_a$, X → M, 0 → I, 1

1. (I_0, ACK$_b$ = 0 0) nop, 1;

   (= 0 1) set RDY$_b$, reset ACK$_b$, B[I_m] + M → OUT, I + 1 → I, 1;

   (= 1 -) "nop", 0 // questa "nop" è una solo abbreviazione: può essere eliminata fondendo la microistruzione 0 nella 1 //

Microprogramma, soluzione ii):

0. (RDY$_a$, OP = 0 –) nop, 0;

   (= 1 0) reset RDY$_a$, set ACK$_a$, X → M, Y → N, segno (X – Y) → S, A[INDA][J] → INDB, 1;

(= 1 1) reset RDY_a, set ACK_a, X → M, 0 → I, 2

1. (S = 0) M → B[INDB], 0;

   (S = 1) N → B[INDB], 0;

2. (I_0, ACK_b = 0 0) nop, 2;

   (= 0 1) set RDY_b, reset ACK_b, B[I_m] + M → OUT, I + 1 → I, 2;

   (= 1 -) "nop", 0 // questa "nop" è una solo abbreviazione: può essere eliminata fondendo la microistruzione 0 nella 1 //

<u>*Facoltativo*</u>:

La soluzione *i)* minimizza il numero di variabili dello stato interno di PC, ma non il numero di variabili di condizionamento: infatti, la variabile di condizionamento *segno(X – Y)* può essere eliminata usando tale valore per controllare, con la tecnica del controllo residuo, il commutatore KB1 posto sull'ingresso del dato della memoria B (ingressi X e Y) [qui si può inserire una figura, per chiarezza].

Tempo medio di elaborazione, soluzione i)

Nella PO la variabile di condizionamento *segno(X – Y)* è ottenuta come uscita secondaria di una ALU, avente come soli ingressi *X* e *Y* affinché la PO risulti un automa di Moore. Altre due ALU serviranno per le operazioni $B[I_m] + M$ e $I + 1$. Quindi :

$$T_{\omega PO} = t_{ALU =} 5t_p$$

Per ricavare $T_{\sigma PO}$, tra le operazioni elementari $B[I_m] + M \rightarrow OUT$ e $I + 1 \rightarrow I$ la prima ha senz'altro ritardo di stabilizzazione maggiore, e precisamente:

$$t_{KB2} + t_{aB} + t_{ALU =} 10\ t_p$$

dove KB2 è il commutatore sull'ingresso dell'indirizzo di B (possibili indirizzi di B: *A[INDA][J]* e I_m).

Il ritardo dell'operazione $X \rightarrow B[A[INDA][J]]$ si ricava considerando le seguenti risorse in cascata [qui può essere messa la figura del pezzo di PO interessato:

lettura da A, commutatore KA, commutatore KB2, scrittura in B

la cui stabilizzazione avviene in parallelo a quella del commutatore KB1. Il ritardo di tale cascata è quindi:

$$t_{aA} + t_{KA} + t_{KB2} + t_{aB}$$

Il commutatore KA è, come KB1 e KB2, a due livelli di logica, in quanto KA ha quattro ingressi primari e due ingressi secondari.

Il tempo di accesso di A si ottiene considerando che A è costituita da una memoria modulare di 64 moduli, ognuno dei quali ha capacità 256 parole e tempo di accesso $3t_p$ Dei 14 bit dell'indirizzo di A, gli 8 bit meno significativi indirizzano ogni modulo, e i 6 bit più significativi controllano un commutatore a 64 ingressi all'uscita di A [queste considerazioni possono essere sostituite da una figura]. Tale commutatore ha ritardo di stabilizzazione uguale a $3t_p$, in quanto i termini AND sono implementabili con un solo livello di logica (porte a 7 ingressi) ed il termine OR con 2 livelli di logica (albero di arietà 8 con 64 foglie). [Equivalentemente, si può usare la formula che si trova nelle "Note sull'Utilizzazione di Componenti Logici di Tipo Memoria"]. Quindi:

$$t_{aA} = 6t_p$$

$$T_{\sigma PO} = t_{aA} + t_{KA} + t_{KB2} + t_{aB} = 13t_p$$

[La scelta di implementare la memoria B a doppio indirizzo (uno per la lettura e uno per la scrittura) permetterebbe di eliminare il commutatore KB2, e quindi comporterebbe una riduzione di $T_{\sigma PO}$ e di τ di $2t_p$. Volendo adottare questa ottimizzazione, essa deve però essere chiaramente giustificata].

Il *massimo ritardo delle funzioni delle uscite e di transizione dello stato interno di PC* è certamente (minore o) uguale a $2t_p$, in quanto (come si vede dal microprogramma) la tabella di verità di tali funzioni ha un numero di righe distinte uguale a 7 (frasi distinte) ed un numero massimo di variabili specificate nelle righe uguale a 4 (al massimo tre variabili di condizionamento significative per frase):

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

Il ciclo di clock è quindi dato da:

$$\tau = T_{\omega PO} + max\ (T_{\omega PC} + T_{\sigma PO}\ ,\ T_{\sigma PC}) + \delta = 5t_p + 2t_p + 13t_p + t_p = 21\ t_p$$

I tempi medi di elaborazione delle operazioni esterne sono dati da:

$$T_0 = 1\ \tau = 21\ t_p$$

$$T_1 = 258\ \tau = 5418\ t_p$$

b) Il ciclo di clock di una unità di elaborazione è, tolta la durata δ dell'impulso di clock, l'intervallo di tempo necessario alla stabilizzazione degli ingressi di tutti i registri di PC e di PO a partire dall'istante (inizio del ciclo di clock, o fronte di discesa dell'impulso di clock) in cui sono stabili tutte le uscite di tali registri. Formalmente, quindi, è sufficiente dire che $\tau - \delta$ è l'intervallo di tempo necessario alla stabilizzazione delle funzioni di transizione dello stato interno di PC, σ_{PC}, e di PO, σ_{PO}.

Poiché il risultato di tali funzioni dipende dallo stato di ingresso di PC e di PO, la loro stabilizzazione è preceduta da quella delle funzioni delle uscite, rispettivamente di PO, ω_{PO}, e di PC, ω_{PC}. *Purché venga fatta questa osservazione*, si può allora anche definire $\tau - \delta$ come l'intervallo di tempo necessario alla stabilizzazione delle quattro funzioni: di transizione dello stato interno, σ_{PC} e σ_{PO}, e delle uscite, ω_{PC} e ω_{PO}.

c) L'asserto "il risultato della funzione delle uscite di PC dipende, ad ogni ciclo di clock, sia dallo stato interno presente che dallo stato d'ingresso di PC" (cioè che PC sia un automa di Mealy) va dimostrato. Esso è una conseguenza della *struttura di frase* delle microistruzioni del microlinguaggio adottato: ad ogni ciclo di clock, lo stato di uscita di PC (configurazione delle variabili di controllo) corrisponde all'esecuzione di una ben determinata microperazione, appartenente ad una ben determinata frase, *la quale è individuata dall'etichetta della microistruzione cui appartiene (stato interno) e dal valore delle variabili di condizionamento testate (stato d'ingresso):*

i. ($X_{i.0}$, $X_{i.1}$, …, $X_{i.k}$-= ……..) microperazione, …; …..

d) L'affermazione è falsa, **in quanto** la variabile di condizionamento in questione è prelevata direttamente all'uscita di un registro (X) e quindi, ad ogni ciclo di clock, il risultato della funzione delle uscite di PO dipende solo dallo stato interno di PO (almeno per quanto riguarda la variabile in questione). La modifica del contenuto del registro X *non* riguarda la funzione delle uscite di PO, *bensì* la funzione di transizione dello stato interno, il cui risultato deve dipendere (ad ogni ciclo di clock) anche dal valore delle variabili di controllo: ma questo non ha niente a che vedere con la condizione di correttezza della PO (automa di Moore), che riguarda solo la funzione delle uscite.

Domanda 2

a) Tutte le espressioni, tranne la 7) sono emulabili con il set di istruzioni Risc del Cap. V:

1)	MOVE R5, Rx	
2)	LOAD R5, 0, Rx	dove la costante corta (6 bit) uguale a zero può anche essere sostituita dal contenuto di un registro inizializzato a zero
3)	LOAD R5, R13, Rx	
4)	LOAD Rcost, 0, Rx	dove il registro di indirizzo *cost* è inizializzato a 43971
5)	LOAD Rcost, 0, Rtemp LOAD Rtemp, 0, Rx	
6)	LOAD R5, R13, Rtemp LOAD Rtemp, 0, Rx	

Il caso 7) non è emulabile in quanto, in questa macchina assembler, il registro IC può essere utilizzato implicitamente solo in istruzioni di salto, mentre nel caso specifico sarebbe richiesto di assegnare esplicitamente ad un registro generale il valore IC + 3825.

[Nota: in realtà, nella seconda parte del corso introdurremo un'istruzione assembler che permette di copiare il contenuto di IC in un registro generale].

b)

STORE Ri, Rj, Rk

Campi del formato su singola parola:

- codice operativo per "STORE" con i valori di entrambi gli addendi (base e indice), per il calcolo dell'indirizzo di memoria, contenuti in registri generali (8 bit),
- indirizzo *i* di registro generale RG (6),
- indirizzo *j* di registro generale RG (6),
- indirizzo *k* di registro generale RG (6),
- inutilizzati (6).

Semantica: il contenuto del registro generale di indirizzo *k* viene scritto nella locazione di memoria, il cui indirizzo è dato dalla somma dei contenuti dei registri generali di indirizzo *i* e *j*; il contatore istruzioni viene incrementato di 1:

$$RG[k] \rightarrow MEM[\ RG[i] + RG[j]\ ],\ IC + 1 \rightarrow IC.$$

È possibile la variante in cui il secondo addendo per il calcolo dell'indirizzo è dato da una costante corta esprimibile con 6 bit (codice operativo diverso dal precedente).

IF< Ri, Rj, offset

Campi del formato su singola parola:

- codice operativo per "IF<" (8 bit),
- indirizzo *i* di registro generale RG (6),
- indirizzo *j* di registro generale RG (6),
- costante con segno (12).

Semantica: se i contenuti dei registri generali di indirizzo *i* e *j* soddisfano il predicato <, allora al contatore istruzioni viene sommata la costante con segno *offset*, altrimenti il contatore istruzioni viene incrementato di 1:

$$\textbf{\textit{if}}\ (RG[i] < RG[j])\ \textbf{\textit{then}}\ IC + offset \rightarrow IC\ \textbf{\textit{else}}\ IC + 1 \rightarrow IC.$$

GOTO Ri

Campi del formato su singola parola:

- codice operativo per "Goto calcolato" con indirizzo di salto indiretto via registro (8 bit),
- indirizzo *i* di registro generale RG (6),
- inutilizzati (18).

Semantica: il contenuto del registro generale di indirizzo *i* viene scritto nel contatore istruzioni:

$$RG[i] \rightarrow IC$$

c) Quando viene lanciato il comando di esecuzione del programma, una opportuna funzionalità del sistema operativo (gestore della memoria principale) provvede a individuare (se esiste momentaneamente) una parte della memoria principale in cui allocare almeno una porzione significativa del programma. Tale porzione, che include il PCB, viene quindi copiata da memoria secondaria (file eseguibile in cui è codificata la memoria virtuale del programma) in memoria principale. Il PCB viene concatenato ad una lista di programmi eseguibili.

Inizializzazione di variabili/costanti in memoria: quelle che sono inizializzate nel file eseguibile verranno automaticamente copiate in memoria principale all'atto del loro trasferimento.

Inizializzazione di variabili/costanti in RB e inizializzazione di IC: quando il Processore si rende disponibile, dal PCB primo in lista vengono copiate le immagini dei registri generali e del contatore istruzioni nei registri corrispondenti del Processore: (seconda parte della) *fase di commutazione di contesto*. Questo rende quindi possibile l'inizializzazione dei registri RG e IC mediante una normale sequenza di istruzioni assembler. La prossima istruzione eseguita, dopo questa sequenza, sarà quindi proprio la prima istruzione da cui deve iniziare (o riprendere) l'esecuzione del programma con il contenuto corretto di RG.

d) <u>*Facoltativo*</u>: una tipica ottimizzazione consiste nel mantenere il valore temporaneo *MAX* del massimo in un registro generale invece che in una locazione di memoria. Un compilatore può riconoscere questa situazione da un'analisi dei costrutti *e dei tipi* (*MAX* è una variabile intera riassegnata all'interno di un loop), cosa che non è in generale possibile con un interprete. Altre ottimizzazioni possono riguardare l'implementazione del controllo del loop, scartando alcuni casi teoricamente possibili in funzione dei valori di alcune costanti in gioco (in questo caso, la dimensione dell'array, certamente maggiore di zero).