Architettura degli elaboratori – A.A. 2016-17

Quarto appello—4 luglio 2017

Riportare nome, cognome, numero di matricola e corso di appartenenza in alto a destra su tutti i fogli consegnati.
I risultati saranno pubblicati via web appena disponibili

Domanda 1

Il calcolo dell'integrale di una funzione *f(x)* fra *a* e *b* può essere approssimato sommando l'area di *n* rettangoli di larghezza $\frac{b-a}{n}$ e altezza $f(x_i)$ con $x_i = a + i\frac{(b.a)}{n}$, per esempio utilizzando lo pseudocodice:

```
float res = 0.0, x = a, delta = (b-a)/n;
for(int i=0; i<n; i++) {    res += f(x)*delta;  x += delta;    }
```

Supponendo che sia $f(x) = c_1x^2 + c_2x + c_3$ e che il calcolo di *f(x)* avvenga utilizzando un'apposita funzione con parametri in ingresso e in uscita passati mediante registri, si calcolino:

- La traccia degli indirizzi logici (anche utilizzando indirizzi simbolici, purché coerenti con le assunzioni utilizzate durante le lezioni) generati dall'esecuzione dei calcolo dell'integrale, assumendo che *a*, *b* ed *n* vengano caricati da un'area di memoria il cui indirizzo base é contenuto in un registro
- Il tempo di completamento del calcolo dell'integrale, in funzione di *t* ed *n*, utilizzando un processore D-RISC pipeline, assumendo che i calcoli in virgola mobile utilizzino una EU parallela con EU slave che calcolano somma e sottrazione in virgola mobile in *2t* e moltiplicazione e divisione in virgola mobile in *4t*
- Il tempo di completamento con un processore D-RISC pipeline superscalare a 2 vie.

Domanda 2

Dato il microcodice:

0. (RDY,IN_0=0-) nop, 0
   (=10) 0 → C, 0 → I 1.
   (=11) 1 → C, 0 → I 1.
1. (zero(N-I)=0) C * IN→ M[I], I+1 → I, 1
   (=1) set ACK, reset RDY, 0.

Si dica se l'unità firmware può essere realizzata con un'unica rete sequenziale motivando dettagliatamente la risposta.

Successivamente si dica se le seguenti affermazioni sono vere o false (sempre motivando dettagliatamente la risposta):

a. Il numero dei livello di porte OR di una PC è determinato unicamente dal numero di microistruzioni del microprogramma di controllo
b. Il numero di bit del registro di stato della PC è determinato unicamente dal numero di microistruzioni del microprogramma di controllo
c. Il numero di livelli di porte AND di una PC è determinato dal numero complessivo delle variabili di condizionamento e delle microistruzioni del microprogramma di controllo.

Traccia di soluzione

Domanda 1

La compilazione dello pseudo codice secondo le regole viste a lezione genera il codice:

```
init:   ADDF Rf0, Rf0, Rres          // azzeramento Rres
        LOAD Rconst, #0, Ra          // carico le costanti dalla memoria di indirizzo noto
        LOAD Rconst, #1, Rb
        LOAD Rconst, #2, Rn
        LOAD Rconst, #3, Rc1
        LOAD Rconst, #4, Rc2
        LOAD Rconst, #5, Rc3
        ADDF Ra, R0, Rx              // x = a
        SUBF Rb, Ra, Rdiff           // calcolo (b-a)/n
        DIVF Rdiff, Rn, Rdelta
        ADD R0, R0, Ri               // inizializzazione variabile d'iterazione (solo per contare)

loop:   CALL Rf, Rret                // ciclo principale: chiamo la funzione
        MULF Rfx, Rdelta, Rcontrib   // calcolo area rettangolo i-esimo
        ADDF Rres, Rcontrib, Rres    // sommo contributo
        ADDF Rdelta, Rx, Rx          // calcolo prossimo xi
        INC Ri                       // numero di iterazioni
        IF< Ri, Rn, loop             // se non ho finito, ciclo
        END

f:      MULF Rx, Rx, Rx2             // ingresso in Rx, uscita in Rfx, calcolo x^2
        MULF Rx2, Rc1, Rx2           // calcolo c1 x^2
        MULF Rx, Rc2, Rxb            // calcolo c2 x
        ADDF Rx2, Rxb, Rxb           // calcolo c1 x^2 + c2 x
        ADDF Rxb, Rc, Rfx            // calcolo c1 x^2 + c2 x + c3
        GOTO Rret                    // ritorno
        END
```

La traccia degli indirizzi logici generati sarà dunque:

- *init, init+1, memparam, init+2, memparam+1, init+3, memparam+2, … init+6, memparam+5, init+7, init+8, init+9, init+10, loop, f, f+1, f+2, … , f+5, loop+1, loop+2, … , loop+5, loop, …*

intendendo per *init*, *loop* e *f* gli indirizzi logici che corrispondono alla prima istruzione del codice compilato, del loop e della funzione f e per *memparam* l'indirizzo dell'area da cui vengono prelevati i valori delle costanti.

Per la valutazione delle prestazioni, consideriamo come al solito il solo corpo del ciclo, eseguito *n* volte. Qui abbiamo un bel numero di dipendenze logiche e bolle legate ai salti:

- dipendenze logiche EU-EU: fra la prima e la seconda istruzione di f, fra la seconda, la terza e la quarta e fra la quarta e la quinta. Inoltre, c'è una dipendenza fra l'ultima ADDF della f e la prima MUL del corpo del loop
- dipendenze logiche IU-EU: fra la INC e la IF<
- bolle legate ai salti: CALL e IF< nel corpo del ciclo, GOTO nella funzione f

L'effetto delle dipendenze e dei salti è notevole, come si può vedere dalla simulazione di una singola iterazione del ciclo:

				0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	loop: CALL Rf, Rret																																	
2	MUL Rfx, Rdelta, Rcontrib																																	
3	ADDF Rres, Rcontrib, Rres		IM	CALL	M2	M8	M9	M10					A11	A12					GOT(	END	M2	A3			A4					INC	IF<	END	CALL		
4	ADDF Rdelta, Rx, Rx		IU		CALL	M2	M8	M9	M10				M10	A11	A12				A12	GOT(	END	M2	A3		A3	A4				A4	INC	IF<	END	CALL	
5	INC Ri		DM																															
6	IF< Ri, Rn, loop		EU					M8	M9	M9	M9	M9	M9	M10	A11	A11	A11	A11	A11	A12	A12	A12	M2	M2	M2	A3	A3	A3	A3	A3	A4	INC				
7	END		EUF*/						M8	M8	M8	M8		M9	M9	M9	M9									M2	M2	M2	M2						
8	f: MULF Rx, Rx, Rx2														M10	M10	M10	M10																
9	MULF Rx2, Rc1, Rx2		EUF+-																	A11	A11		A12	A12						A3	A3			
10	MULF Rx, Rc2, Rxb																														A4	A4		
11	ADDF Rx2, Rxb, Rxb																																	
12	ADDF Rxb, Rc, Rfx																																	
13	GOTO Rret																																	
14	END																																	

Di fatto impieghiamo *29t* per una iterazione (tempo di completamento pari a *29nt*) a fronte di 12 istruzioni per iterazione. Il tempo di servizio sarà quindi T = $\frac{29}{12}t$ e l'efficienza $\varepsilon = \frac{12}{29} = 0.414$

Volendo considerare un processore D-RISC pipeline, superscalare a due vie, possiamo utilizzare il codice D-RISC precedente per riempire i due slot di ciascuna delle istruzioni *lunghe*, ricordando le condizioni che precludono all'utilizzo di due particolari istruzioni D-RISC in un'istruzione lunga che obbligano all'inserimento di eventuali NOP.

In questo caso, sempre considerando il solo corpo del ciclo e il calcolo di f (escludendo quindi le istruzioni dell'inizializzazione), avremo

loop:	CALL Rf, Rret	NOP
	MUL Rfx, Rdelta, Rcontrib	NOP
	ADDF Rres, Rcontrib, Rres	ADDF Rdelta, Rx, Rx
	INC Ri	NOP
	IF< Ri, Rn, loop	NOP
f:	MULF Rx, Rx, Rx2	NOP
	MULF Rx2, Rc1, Rx2	MULF Rx, Rc2, Rxb
	ADDF Rx2, Rxb, Rxb	NOP
	ADDF Rxb, Rc, Rfx	GOTO Rret

con un numero di NOP (necessarie) relativamente alto. Come si può vedere con la simulazione, non si ha un guadagno significativo rispetto al caso non superscalare e l'efficienza è chiaramente più bassa:

								0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	loop:	CALL Rf, Rret	NOP				IM	1	2	6	7	8					9						2		3			4				5			1		
2		MUL Rfx, Rdelta, Rcontrib	NOP				IU		1	2	6	7	8				8	9				9		2	2	3		3	4				4	5	5		1	
3		ADDF Rres, Rcontrib, Rres	ADDF Rdelta, Rx, Rx				DM																														
4		INC Ri	NOP				EU					6	7	7	7	7	7	8	8	8	8	8	9	9	9	2	2	2	3	3	3	3	3	4				
5		IF< Ri, Rn, loop	NOP				EUF*/						6	6	6	6		7	7	7	7		8	8		9	9		2	2	2	2		3	3			
6	f:	MULF Rx, Rx, Rx2	NOP																7	7	7	7							3	3							
7		MULF Rx2, Rc1, Rx2	MULF Rx, Rc2, Rxb				EUF+-																														
8		ADDF Rx2, Rxb, Rxb	NOP																																		
9		ADDF Rxb, Rc, Rfx	GOTO Rret																																		

Il tempo di completamento è di 28*nt*.

Il codice potrebbe essere chiaramente ottimizzato. Il calcolo della funzione f può essere fatto in modo da interporre le istruzioni che calcolano $c_2x + c_3$ fra le istruzioni che calcolano $c_1 x^2$. Possiamo anche utilizzare un salto ritardato sia per il ritorno dalla funzione che per la chiamata. Per lo slot della GOTO possiamo utilizzare l'ultima istruzione che calcola il risultato finale (una ADDF). Nello slot della CALL possiamo inserire la INC Ri, per esempio. Il salto ritardato può essere utilizzato anche per l'IF< di fine iterazione, utilizzando l'istruzione che aggiorna il valore dell''integrale per riempire lo slot. Tuttavia, tutto questo può essere ulteriormente migliorato utilizzando la tecnica dell'inlining, ovvero inserendo le poche istruzioni che calcolano f all'interno del corpo del ciclo, sostituendole alla CALL.

In questo caso il codice potrebbe essere riscritto come segue (abbiamo anche utilizzato la proprietà associativa della addizione, ovvero compiliamo $f(x) = c_1x^2 + (c_2x+c_3)$ anziché $f(x) = (c_1x^2 + c_2x)+c_3$):

loop:

MULF Rx, Rx, Rx2
MULF Rx, Rc2, Rxb
ADDF Rxb, Rc3, Rxb
MULF Rx2, Rc1, Rx2
ADDF Rxb, Rx2, Rfx
INC Ri
MULF Rfx, Rdelta,
Rcontrib
ADDF Rdelta, Rx, Rx
IF< Ri, Rn, loop, delayed
ADDF Rres, Rcontrib, Rres

In questo caso, la simulazione ci fa vedere un guadagno migliore:

							0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	MULF Rx, Rx, Rx2																																	
2	MULF Rx, Rc2, Rxb					IM	M1	M2	A3	M4					A5	INC					M7	A8		IF<d	A10	M1								
3	ADDF Rxb, Rc3, Rxb					IU		M1	M2	A3	M4				M4	A5	INC				INC	M7	A8	A8	IF<d	A10	M1		M1					
4	MULF Rx2, Rc1, Rx2					DM																												
5	ADDF Rxb, Rx2, Rfx					EU			M1	M2	A3	A3	A3	A3	A3	M4	A5	A5	A5	A5	A5	INC	M7	M7	A8		A10	A10	A10	M1				
6	INC Ri					EUF*/				M1	M1	M1	M1												M7	M7	M7	M7			M1	M1	M1	M1
7	MULF Rfx, Rdelta, Rcontrib										M2	M2	M2	M2			M4	M4	M4	M4														
8	ADDF Rdelta, Rx, Rx					EUF+-										A3	A3					A5	A5			A8	A8			A10	A10			
9	IF< Ri, Rn, loop, delayed																																	
10	ADDF Rres, Rcontrib, Rres																																	

Visto che tempo di completamento diventa 21nt. Se consideriamo questo codice per il caso del processore superscalare otteniamo:

loop:

MULF Rx, Rx, Rx2	MULF Rx, Rc2, Rxb
ADDF Rxb, Rc3, Rxb	MULF Rx2, Rc1, Rx2
ADDF Rxb, Rx2, Rfx	INC Ri
MULF Rfx, Rdelta, Rcontrib	ADDF Rdelta, Rx, Rx
IF< Ri, Rn, loop	ADDF Rres, Rcontrib, Rres

								0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	MULF Rx, Rx, Rx2	MULF Rx, Rc2, Rxb				IM	1	2	3					4					5						1										
2	ADDF Rxb, Rc3, Rxb	MULF Rx2, Rc1, Rx2				IU		1	2	3				3	4				4	5		5		1				1						
3	ADDF Rxb, Rx2, Rfx	INC Ri				DM																													
4	MULF Rfx, Rdelta, Rcontrib	ADDF Rdelta, Rx, Rx				EU			1	2	2	2	2	2	2	3.1	3.2	3.2	3.2	3.2	4	4	4	5	5	5	5	5	1						
5	IF< Ri, Rn, loop	ADDF Rres, Rcontrib, Rres				EUF*/				1.1	1.1	1.1	1.1			2.2	2.2	2.2	2.2					4.1	4.1	4.1	4.1		5	5	5	5			
											1.2	1.2	1.2	1.2									4.2	4.2						1.1	1.1	1.1	1.1		
						EUF+-								2.1	2.1						3.2	3.2									1.2	1.2	1.2	1.2	

Adesso il tempo di completamento è 20*nt*.

Domanda 2

Se consideriamo il microcodice assegnato, possiamo notare che le micro operazioni della seconda micro istruzione soddisfano le condizioni di Bernstein rispetto alle micro operazioni della prima micro istruzione, tranne che nel caso dell'inizializzazione del registro I (sempre posto a 0 nella prima microistruzione) e del registro C. I viene scritta da 0.1 e 0.2 e letta nella 1.0, violando le condizioni di Bernstein. Abbiamo tuttavia assunto che qualunque registro utilizzato in una unità firmware sia automaticamente inizializzato a 0 all'avvio dell'unità, dunque potremmo rimuovere le inizializzazioni di I nella 0. e considerare il codice:

0. (RDY,IN_0=0-) nop,
   (=10) 0 → C, 1.
   (=11) 1 → C, 1.
1. (zero(N-I)=0) C * IN→ M[I], I+1 → I, 1
   (=1) set ACK, reset RDY, 0

Anche il registro C viene scritto nella 0. e letto nella 1. Osservando che il valore di C è costante e dipende dalla IN_0, possiamo eliminare l'inizializzazione e utilizzare una costante nella 1. a seconda del valore di IN_0. Con queste ipotesi il microcodice potrebbe essere riscritto così:

0. (RDY, IN0, zero(N-I)=0-- nop, 0
   (=1-1) set ACK, reset RDY, 0
   (=100) 0 * IN→ M[I], I+1 → I, 0
   (=110) 1 * IN→ M[I], I+1 → I, 0

ovvero, semplificando

0. (RDY, IN0, zero(N-I)=0-- nop, 0
   (=1-1) set ACK, reset RDY, 0
   (=100) 0 → M[I], I+1 → I, 0
   (=110) IN→ M[I], I+1 → I, 0

Dunque l'unità può essere realizzata come un'unica rete sequenziale, visto che la parte controllo non avrebbe stato interno dal momento che il programma di controllo ha un'unica micro istruzione.

Riguardo alle affermazioni, ultima parte della seconda domanda:

a. Il numero dei livello di porte OR di una PC è determinato unicamente dal numero di microistruzioni del microprogramma di controllo
Falso, dobbiamo prevedere una riga della tabella di verità di ωPC e σPC per ognuna delle frasi del microprogramma, che possono essere più del numero delle micro istruzioni

b. Il numero di bit del registro di stato della PC è determinato unicamente dal numero di microistruzioni del microprogramma di controllo
Vero. Il numero di microistruzione corrente è lo stato della PC, dunque i bit del registro di stato devono essere quelli necessari a rappresentare il numero dell'istruzione.

c. Il numero di livelli di porte AND di una PC è determinato dal numero complessivo delle variabili di condizionamento e delle microistruzioni del microprogramma di controllo
Falso. E' determinato dal numero di bit del registro di stato (quindi dal numero delle micro istruzioni, vedi sopra) e dal numero di variabili di condizionamento testate contemporaneamente in una micro istruzione.