Appello straordinario Architettura degli Elaboratori
5 novembre 2019

Riportare in alto a destra di tutti i fogli consegnati Nome, Cognome, Matricola e Corso di appartenenza (A o B). I risultati e il calendario degli orali saranno resi disponibili su web (pagine docenti o didawiki) appena disponibili.

Domanda 1

Si consideri lo pseudo codice che copia il contenuto di un array X di lunghezza N in un altro array Y di lunghezza N/2 facendo sì che per ogni i in [0, N/2 -1] risulti Y[i] = X[2*i] + X[2*i +1]:

```
for(int i=0; i<N/2; i++)
        Y[i] = X[2*i] + X[2*i +1];
```

Successivamente

- si compili il codice in assembler D-RISC, secondo le regole di compilazione standard
- se ne valutino le prestazioni su un processore pipeline con EU in grado di eseguire solo operazioni aritmetico logiche "corte" (no moltiplicazioni e divisioni), evidenziando eventuali inefficienze prestazionali
- si determini il working set del programma e si valuti il numero di fault che si verificano durante l'esecuzione
- si valutino possibili ottimizzazioni del codice assembler

Domanda 2

Si progetti un'unità firmware U_F che filtra i messaggi inviati da una unità U_A ad una unità U_B in modo da evitare che vi siano duplicati in sequenze di 2^n+1 messaggi consecutivi (messaggi ripetuti devono essere scartati) e se ne forniscano le prestazioni. U_A ed U_B interagiscono entrambe con U_F richiedendo una che U_A accetti un invio di un dato e l'altra richiedendo l'invio di un nuovo dato (quando disponibile), rispettivamente. Tutti i messaggi sono da una parola[1].

[1] Nella versione consegnata in aula questa frase non c'era, ma è stata comunicata a tutti dal docente.

Bozza di soluzione

Domanda 1

Pseudo codice

```
for(int i=0; i<N/2; i++) Y[i] = X[2*i] + X[2*i +1];
```

Compilazione standard

```
        CLEAR Ri
        SHR Rn, 1, Rnmezzi
LOOP:   SHL Ri, 1, Rdisp
        LOAD Rbasex, Rdisp, R2x
        ADD Rdisp, 1, Rdisp
        LOAD Rbasex, Rdisp, R2x1
        ADD R2x, R2x1, Ry
        STORE Rbasey, Ri, Ry
        INC Ri
        IF< Ri, Rnmezzi, LOOP
        END
```

Valutazione delle prestazioni

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
IM	SHL	LOAD		ADD	LOAD			ADD	STORE			INC	IF<		END	SHL		
IU		SHL	LOAD	LOAD	ADD	LOAD	LOAD	LOAD	ADD	STORE	STORE	STORE	INC	IF<	IF<	END	SHL	
DM					LOAD				LOAD				STORE					
EU			SHL			LOAD	ADD			LOAD	ADD			INC				SHL

Impieghiamo 15t per eseguire un'iterazione (a regime) che è composta da 8 istruzioni. Efficienza appena superiore al 50%. La SHL induce una dipendenza sulla prima LOAD, la ADD sulla seconda, la seconda ADD induce dipendenza sulla STORE, la INC sulla IF< e infine abbiamo la bolla da salto preso.

Working set

Entrambi i vettori vengono acceduti sequenzialmente, senza riutilizzo. Il working set è dunque formato da tutto il codice (acceduto con località e riuso) da una linea di X e una linea di Y. Se utilizziamo prefetch avremo un solo fault per X e Y oltre ai fault per il codice.

Ottimizzazione del codice

Possiamo eseguire immediatamente le due istruzioni che calcolano gli indici per l'accesso a X e anche la INC Ri. Inoltre, possiamo utilizzare il delayed branch, se presente, per posticipare la store, avendo cura di utilizzare il registro base decrementato per Y:

```
LOOP:  SHL Ri, 1, Rdisp
       ADD Rdisp, 1, Rdisp
       INC Ri
       LOAD Rbasex, Rdisp, R2x
       LOAD Rbasex, Rdisp, R2x1
       ADD R2x, R2x1, Ry
       IF< Ri, Rnmezzi, LOOP, delayed
       STORE Rbasey', Ri, Ry
```

Con questo codice riusciamo ad eseguire una iterazione in 9t, che è quasi il tempo ideale. Rimane infatti solo la dipendenza indotta dalla ADD sulla STORE, che, seppure a distanza 1, ha effetto per via delle LOAD nella sequenza di istruzioni che portano alla dipendenza.

	0	1	2	3	4	5	6	7	8	9	10	11
IM	SHL	ADD	INC	LOAD	LOAD	ADD	IF<	STORE	SHL			
IU		SHL	ADD	INC	LOAD	LOAD	ADD	IF<	STORE	STORE	SHL	
DM						LOAD	LOAD				STORE	
EU			SHL	ADD	INC		LOAD	LOAD	ADD			SHL

Domanda 2

Possiamo utilizzare una memoria per mantenere copia degli 2^n messaggi precedenti e controllare che il messaggio corrente non sia nella memoria. Va distinto il funzionamento a regime (la memoria sarà sempre piena) dal transitorio iniziale (la memoria si riempie un messaggio alla volta). Utilizziamo un registro N che contiene il numero di elementi in memoria da n bit e un registro I che punta alla prossima posizione da utilizzare (in modo circolare, sempre da n bit).

Utilizziamo RDYa e ACKa per interagire con UA e RDYb e ACKb per interagire con UB. Assumiamo che tutte le operazioni siano effettuate modulo 2 alla n.

1. (RDYa, N0, or(N) = 0, -, -) nop, 0
   (=1, 1, 0) eq(M[0],INm)->F, 1->J, 2. // mem piena. Comincio a controllare
   (=1, 0, 1) eq(M[0], INm) -> F, 1-> J, eq(1, I)->E, 4 // transitorio, comincio a controllare
   (=1, 0, 0) INm -> M[0], 1 -> N, 3. // primo messaggio, lo memorizzo e spedisco a richiesta
2. // controllo con memoria piena
   (F, J0 = 1, -) reset RDYa, set ACKa, 1. // scarto il messaggio e ricomincio
   (=0,1) INm -> M[I], I + 1 -> I, 3. // pronto a mandare a UB quando chiede, mi ricordo il msg
   (=0,0) J+1 -> J, eq(M[J],INm}->F, 2. // altrimenti scorro
3. // spedisco messaggio corrente appena mi viene richiesto
   (RDYb=0) nop, 3.
   (=1) INm -> OUTm, set ACKb, reset RDYb, set ACKa, reset RDYa, 0.
4. // controllo con memoria non piena
   (F, E = 1, -) eset RDYa, set ACKa, 1. // scarto il messaggio e ricomincio

(=0, 0) J+1 -> J, eq(I,J)->E, eq(M[J], INm)->F, 2. // altrimenti scorro
(=0,1) INm -> M[I], I+1 -> I, 3. // memorizzo e mando a richiesta

Prestazioni

4 stati e max 3 variabili di condizionamento testate contemporaneamente. 1 Livello AND per la PC.12 frasi. 2 livello OR per la PC, che possono essere ridotti a 1codificando gli zeri e negando le uscite se necessario. Per la parte operativa, non abbiamo variabili di condizionamento complesse, tranne le OR(N) che potrebbe richiedere più di un livello di porte se n>8. La microistruzione più lunga è la eq(M[I],INm) -> F che richiede tk + ta + talu (commutatore sugli indirizzi, tempo di accesso alla memoria e tempo per produrre il flag mediante una ALU).

Il ciclo di clock sarà pertanto

Tau = tp (assumendo n<=8) + 2tp + tk + talu + ta + tp = 6tp + ta + talu.

In media, a regime, occorre eseguire $2^n/2$ iterazioni se l'elemento è presente nei messaggi precedenti, e 2^n iterazioni se non è presente, più un paio di microsistruzioni per inizializzare la computazione e per mandare il messaggio alla unità UB. Quindi T sarà circa $(½(2^{n-2} + 2^{n-1}) + 2)$tau che ca. $3x2^{n-2}$ tau.