AE – A.A. 2018-2019 – 2ª prova di verifica intermedia

Riportare nella prima facciata in alto a destra di ciascuno dei fogli consegnati Nome, Cognome, numero di matricola e corso di appartenenza (A o B). I risultati saranno pubblicati via WEB appena disponibili.

Domanda 1

Si consideri un algoritmo che esegue l'ordinamento di un vettore di interi utilizzando il *bubble sort* (per ogni posizione dalla prima alla penultima scambiare il valore corrente con il minimo dei valori trovati nelle rimanenti posizioni del vettore). Il confronto fra due valori avviene in modo non standard:

→ dati x e y allora $x<y$ se e solo se $(3x) \bmod 8 < (3y) \bmod 8$.

Per tale algoritmo:

- Si produca uno pseudo codice e se ne fornisca il working set
- Si fornisca la compilazione in assembler D-RISC utilizzando le regole standard
- Si fornisca la traccia degli indirizzi logici generati dal programma
- Si forniscano le prestazioni (tempo di servizio) su un processore D-RISC pipeline con unità EU slave pipeline a 2 stadi per l'esecuzione delle moltiplicazioni e divisioni intere
- Si individuino le cause di degrado delle prestazioni e se ne forniscano eventuali ottimizzazioni, quantificando il guadagno in termini di tempo di servizio.

Si considerino le istruzioni dell'assembler D-RISC descritto nelle note distribuite a lezione con l'aggiunta delle istruzioni che calcolano operazione logiche bit a bit su registri (AND, OR, NOT) e l'istruzione aritmetico logica lunga MOD che calcola il resto della divisione intera.

Domanda 2

Si dica se le seguenti affermazioni sono vere o false, dandone opportuna e sintetica motivazione:

1. In una cache di primo livello organizzata secondo il modello associativo su insiemi con insiemi da 4 linee e s=16 parole e con un modello di memoria paginata con pagine da 1K parole, indirizzi della stessa pagina di memoria principale risiedono necessariamente nella stessa linea di cache.
2. Un processore D-RISC pipeline superscalare a 2 vie esegue qualunque codice D-RISC a una velocità doppia rispetto ad un processore D-RISC di identiche caratteristiche ma privo del supporto per l'esecuzione superscalare.
3. Una CPU costituita da processore D-RISC sequenziale, MMU, cache di primo livello set-associativa a 4 vie e memoria principale interallacciata esegue qualunque programma D-RISC più velocemente di un processore con le stesse caratteristiche ma con cache (della stessa capacità complessiva) ad indirizzamento diretto.

Traccia di soluzione

Come al soltio la traccia è indicativa e alcune parti sono presenti solo a titolo esplicativo.

Domanda 1

Innanzitutto, osserviamo come opera l'algoritmo. Assumiamo di avere il vettore di interi da ordinare

4	19	12	7	16	21	32	18

Riportiamo i vari passi, ciascuno indicando il contenuto del vettore (prima riga della tabella) e i rispettivi risultati dell'operazione f(x) = 3x mod 8:

Vettore	**4**	**19**	**12**	**7**	**16**	**21**	**32**	**18**
3*x %8	4	1	4	5	0	7	0	6
Passo 1	**16**	**19**	**12**	**7**	**4**	**21**	**32**	**18**
	0	1	4	5	4	7	0	6
Passo 2	**16**	**32**	**12**	**7**	**4**	**21**	**19**	**18**
	0	0	4	5	4	7	1	6
Passo 3	**16**	**32**	**19**	**7**	**4**	**21**	**12**	**18**
	0	0	1	5	4	7	4	6
Passo 4	**16**	**32**	**19**	**4**	**7**	**21**	**12**	**18**
	0	0	1	4	5	7	4	6
Passo 5	**16**	**32**	**19**	**4**	**12**	**21**	**7**	**18**
	0	0	1	4	4	7	5	6
Passo 6	**16**	**32**	**19**	**4**	**12**	**7**	**21**	**18**
	0	0	1	4	4	5	7	6
Passo 7	**16**	**32**	**19**	**4**	**12**	**7**	**18**	**21**
	0	0	1	4	4	5	6	7
Passo 8	**16**	**32**	**19**	**4**	**12**	**7**	**18**	**21**
	0	0	1	4	4	5	6	7

Pseudo codice

Lo pseudo codice è costituito da due cicli annidati. Utilizziamo la funzione f per calcolare l'operazione f(x) = (3*x) mod 8, ma in realtà la utilizzeremo come codice inline, viste le dimensioni.

```
for(int i=0; i<N-1; i++)
       for(int j=i+1; j<N; j++)
              if(f(x[j]) < f(x[i])) then swap(x[i], x[j]);

int f(int x) = (3*x) mod 8;
```

Working set

Ad ogni iterazione j servono la linea di cache che contiene x[i] e quella che contiene x[j]. X gode della proprietà di località (da utilizzare il flag prefetch) e del riuso (da utilizzare il non deallocare). Nel seguito omettiamo i flag delle LOAD per semplicità.

Compilazione in codice D-RISC

Il codice D-RISC, secondo la compilazione standard, è il seguente:

```
                                    CLEAR Ri
```

```
LOOPI  ADD Ri, #1, Rj
LOOPJ  LOAD Rbasex, Ri, Rxi
       MUL Rxi, #3, Rtemp1
       MOD Rtemp1, #8, Rtemp1
       LOAD Rbasex, Rj, Rxj
       MUL Rxj, #3, Rtemp2
       MOD Rtemp2, #8, Rtemp2
       IF>= Rtemp1, Rtemp2, CONT
 THEN  STORE Rbasex, Ri, Rxj
       STORE Rbasex, Rj, Rxi
 CONT  INC Rj
       IF< Rj, Rn, LOOPJ
       INC Ri
       SUB Rn, #1, Rn1
       IF< Ri, Rn1, LOOPI
       END
```

Traccia degli indirizzi

Assumendo che per le prime due iterazioni il ramo **then** sia saltato e preso, avremo una traccia tipo (In indica l'istruzione n-esima, contando a partire da I0, e Xn indica un accesso alla posizione n dell'array X):

I0, I1, I2, I3, X1, I4, I5 I6, X0, I7, I8, I9 (salto preso), I12, I13, I3, X2, I4, I5, I6, X1, I7, I8, I9 (salto non preso), I10, X1, I11, X2, I12, I13, I3 …

Prestazioni

Consideriamo il solo ciclo più interno. Abbiamo:

- Una dipendenza EU-EU fra le MUL e le MOD.
- Una dipendenza IU-EU fra la seconda MOD e la IF>=
- Una dipendenza IU-EU fra la INC e la IF<
- Una dipendenza IU-EU fra la seconda MOD e la prima STORE, in caso di ramo **then** preso

Inoltre avremo una o due penalità legati ai salti:

- Quello di fine ciclo, normalmente preso, e
- Il salto per nn eseguire il ramo **then**, in caso non occorra fare lo swap fra le due posizioni del vettore.

Per valutare il tempo di completamento di un'iterazione, consideriamo i due casi. Esecuzione di un'iterazione che non richiede uno swap:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
IM	LD	MUL	MOD	LD	MUL			MOD	IF>=							ST	INC	IF<		END	LD		
IU		LD	MUL	MOD	LD	MUL			MOD	IF>=						IF>=	ST	INC	IF<	IF<	END	LD	
DM			LD			LD																	LD
EU master				LD	MUL	MOD		MOD	LD	MUL	MOD		MOD						INC				
EU slave */						MUL	MUL		MOD	MOD	MUL	MUL		MOD	MOD								

ed esecuzione con ramo **then** preso (quindi con lo swap):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
IM	LD	MUL	MOD	LD	MUL			MOD	IF>=							ST	ST	INC	IF<		END	LD	
IU		LD	MUL	MOD	LD	MUL			MOD	IF>=						IF>=	ST	ST	INC	IF<	IF<	END	LD
DM			LD			LD												ST	ST				
EU master				LD	MUL	MOD		MOD	LD	MUL	MOD		MOD							INC			
EU slave */						MUL	MUL		MOD	MOD	MUL	MUL		MOD	MOD								

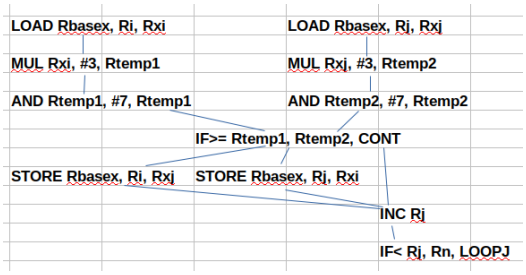
Nel primo caso occorrono 20tper completare un'iterazione, nel secondo ne occorrono 21. In media quindi un'iterazione richiede 20.5t per eseguire (9+11)/10 istruzioni. Il tempo di servizio risulta di circa 2t.

Ottimizzazione

Innanzitutto, occorre osservare che la MOD ha per operando una potenza di 2. Le MOD …, #8, … possono quindi essere sostituite da una operativa corta AND …, #7, … (si noti che l'AND è fatto con 2^k-1). Il codice del loop centrale potrebbe essere riscritto quindi come segue:

```
        CLEAR Ri
LOOPI   CLEAR Rj
LOOPJ   LOAD Rbasex, Ri, Rxi
        MUL Rxi, #3, Rtemp1
        AND Rtemp1, #7, Rtemp1
        LOAD Rbasex, Rj, Rxj
        MUL Rxj, #3, Rtemp2
        AND Rtemp2, #7, Rtemp2
        IF>= Rtemp1, Rtemp2, CONT
 THEN   STORE Rbasex, Ri, Rxj
        STORE Rbasex, Rj, Rxi
 CONT   INC Rj
        IF< Rj, Rn, LOOPJ
        INC Ri
        SUB Rn, #1, Rn1
        IF< Ri, Rn1, LOOPI
        END
```

Questo porterebbe ad un miglior tempo di esecuzione, perché la dipendenza EU-EU si riduce in lunghezza. Tuttavia, osservando il grafo delle dipendenze fra le istruzioni:



Notiamo che le prime sei istruzioni si possono eseguire in un ordinamento diverso, purché la LOAD proceda la relativa MUL e quest'ultima preceda l'esecuzione della relativa AND. Per mitigare l'effetto della dipendenza EU-EU MUL-AND potremmo dunque trasformare il codice così (vediamo solo il ciclo più interno, che è quello eseguito il maggior numero di volte):

```
LOOPJ  LOAD Rbasex, Ri, Rxi
       MUL Rxi, #3, Rtemp1
       LOAD Rbasex, Rj, Rxj
       MUL Rxj, #3, Rtemp2
       AND Rtemp1, #7, Rtemp1
       AND Rtemp2, #7, Rtemp2
       IF>= Rtemp1, Rtemp2, CONT
THEN   STORE Rbasex, Ri, Rxj
       STORE Rbasex, Rj, Rxi
CONT   INC Rj
       IF< Rj, Rn, LOOPJ
```

Il che porta ad un tempo di esecuzione decisamente migliore:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IM	LD	MUL	LD	MUL	AND	AND	IF>=				ST	INC	IF<		END	LD	
IU		LD	MUL	LD	MUL	AND	AND	IF>=			IF>=	ST	INC	IF<	IF<	END	LD
DM			LD		LD												
EU master				LD	MUL	LD	MUL	AND	AND	AND				INC			
EU slave */						MUL	MUL	MUL	MUL								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IM	LD	MUL	LD	MUL	AND	AND	IF>=				ST	ST	INC	IF<		END	LD
IU		LD	MUL	LD	MUL	AND	AND	IF>=			IF>=	ST	ST	INC	IF<	IF<	END	LD
DM			LD		LD								ST	ST			
EU master				LD	MUL	LD	MUL	AND	AND	AND					INC		
EU slave */						MUL	MUL	MUL	MUL								

In questo caso, abbiamo un tempo per il calcolo dell'iterazione che è 15t nel caso non sia richiesto lo swap e 16t nel caso di swap, a fronte, rispettivamente di 9 o 11 istruzioni, lo stesso numero dei casi precedenti. Dal momento che il tempo di completamento diminuisce, a parità di numero di istruzioni anche il tempo di servizio diminuisce.

Potremmo ancora anticipare la INC prima della IF>= utilizzando una base anticipata per la LOAD che utilizza Rj. Questo permetterebbe sia di mitigare (annullare, di fatto) l'effetto della dipendenza EU-EU sulla seconda AND che di eliminare la dipendenza IU-EU INC-IF>:

```
LOOPJ  LOAD Rbasex, Ri, Rxi
       MUL Rxi, #3, Rtemp1
       LOAD Rbasex, Rj, Rxj
       MUL Rxj, #3, Rtemp2
       AND Rtemp1, #7, Rtemp1
       INC Rj
       AND Rtemp2, #7, Rtemp2
       IF>= Rtemp1, Rtemp2, CONT
 THEN  STORE Rbasex', Ri, Rxj
       STORE Rbasex, Rj, Rxi
 CONT  IF< Rj, Rn, LOOPJ
```

Come si vede nel codice, occorre utilizzare un registro base decrementato per la prima STORE per annullare l'effetto dell'anticipato incremento del registro indice. In quest'ultimo caso, l'esecuzione col ramo **then** preso arriva a concludere un'iterazione in 15t contro I 21t della versione originale, con un risparmio netto di 6t per iterazione. Di nuovo, il numero di istruzioni non cambia e quindi l'incremento di tempo di completamento dell'iterazione comporta anche un incremento del tempo di servizio.

Va osservato che nel codice prodotto con le regole di compilazione standard ci sono alcune cose che sono ripetute ad ogni iterazione ma che in realtà sono invarianti, che qui di seguito evidenziamo in giallo:

```
       CLEAR Ri
LOOPI  ADD Ri, #1, Rj
LOOPJ  LOAD Rbasex, Ri, Rxi
       MUL Rxi, #3, Rtemp1
       AND Rtemp1, #7, Rtemp1
       LOAD Rbasex, Rj, Rxj
       MUL Rxj, #3, Rtemp2
       AND Rtemp2, #7, Rtemp2
```

```
        IF>= Rtemp1, Rtemp2, CONT
THEN    STORE Rbasex, Ri, Rxj
        STORE Rbasex, Rj, Rxi
CONT    INC Rj
        IF< Rj, Rn, LOOPJ
        INC Ri
        SUB Rn, #1, Rn1
        IF< Ri, Rn1, LOOPI
        END
```

Se muovessimo le prime tre istruzioni evidenziate all'esterno del loop for j e la SUB che decrementa Rn fuori dal ciclo for i otterremmo il codice:

```
        CLEAR Ri
        SUB Rn, #1, Rn1
LOOPI   ADD Ri, #1, Rj
        LOAD Rbasex, Ri, Rxi
        MUL Rxi, #3, Rtemp1
        MOD Rtemp1, #8, Rtemp1
LOOPJ   LOAD Rbasex, Rj, Rxj
        MUL Rxj, #3, Rtemp2
        AND Rtemp2, #7, Rtemp2
        IF>= Rtemp1, Rtemp2, CONT
THEN    STORE Rbasex, Ri, Rxj
        STORE Rbasex, Rj, Rxi
CONT    INC Rj
        IF< Rj, Rn, LOOPJ
        INC Ri
        IF< Ri, Rn1, LOOPI
        END
```

In cui è nuovamente evidenziato il ciclo interno. L'esecuzione del ciclo interno adesso prende 13t o 14t a seconda del salto (then preso o no). Il grafo delle dipendenze evidenza una catena lineare fra LOAD, MUL, AND e IF>= che non c'è modo di migliorare, a meno di non anteporre la INC Rj:

```
LOOPJ  LOAD Rbasex, Rj, Rxj
       MUL Rxj, #3, Rtemp2
       INC Rj
       AND Rtemp2, #7, Rtemp2
       IF>= Rtemp1, Rtemp2, CONT
 THEN  STORE Rbasex, Ri, Rxj
       STORE Rbasex, Rj, Rxi
 CONT  IF< Rj, Rn, LOOPJ
```

Questa piccola ottimizzazione porta il tempo di esecuzione a 11t/12t (a seconda del salto). Vista l'anticipazione della in Rj potremmo anche anticipare la LOAD nel delay slot della IF<:

```
       LOAD Rbasex, Rj, Rxj
LOOPJ  MUL Rxj, #3, Rtemp2
       INC Rj
       AND Rtemp2, #7, Rtemp2
       IF>= Rtemp1, Rtemp2, CONT
 THEN  STORE Rbasex, Ri, Rxj
       STORE Rbasex, Rj, Rxi
 CONT  IF< Rj, Rn, LOOPJ, DELAYED
       LOAD Rbasex, Rj, Rxj
```

Questa ulteriore piccola ottimizzazione porta il tempo di esecuzione dell'iterazione a 10t/11t (a seconda del salto).

Dunque, complessivamente il tempo di servizio (calcolato relativamente al solo ciclo più interno) passa dai 2t calcolati all'inizio ai 11.5t/7 che è poco più di 1.6t.

Domanda 2

Punto 1

In una cache di primo livello organizzata secondo il modello associativo su insiemi con insiemi da 4 linee e σ=16 parole e con un modello di memoria paginata con pagine da 1K parole, indirizzi della stessa pagina di memoria principale risiedono necessariamente nella stessa linea di cache.

Falso: la pagina di memoria principale è di dimensioni maggiori della pagina di cache, dunque le informazioni appartenenti ad una pagina di memoria centrale saranno memorizzate in più pagine (linee) di cache. Nella fattispecie, in 64 linee diverse (1024/16).

Punto 2

Un processore D-RISC pipeline superscalare a 2 vie esegue qualunque codice D-RISC a una velocità doppia rispetto ad un processore D-RISC di identiche caratteristiche ma privo del supporto per l'esecuzione superscalare.

Falso: dipende dal tipo di codice. Se nel codice sono presenti istruzioni consecutive che dovrebbero finire nella stessa istruzione superscalare ma che presentano dipendenze logiche, la velocità di esecuzione è diminuita dalla necessità di inserire delle NOP.

Punto 3

Una CPU costituita da processore D-RISC sequenziale, MMU, cache di primo livello set-associativa a 4 vie e memoria principale interallacciata esegue qualunque programma D-RISC più velocemente di un processore con le stesse caratteristiche ma con cache (della stessa capacità complessiva) ad indirizzamento diretto.

Vero. L'unico caso in cui potrebbe andare alla stessa velocità è quello in cui nella cache ad indirizzamento diretto non abbiamo mai indirizzi appartenenti al working set che vengano mappati su un'unica linea di cache.