Architettura degli elaboratori – A. A. 2017–2018

Primo appello Sessione Estiva – 11 giugno 2018

Indicare in alto a destra di ciascun foglio consegnato, nome, cognome, matricola e corso di appartenenza (A o B).
I risultati e la bozza di correzione saranno pubblicati via WEB appena disponibili.

Domanda 1

Un'unità firmware *U* riceve da una seconda unità firmware U_c due indirizzi di vettori di numeri interi in memoria (*inda* e *indb*), la lunghezza dei vettori (*n*) e due costanti intere (*x* e *y*) e calcola la combinazione lineare dei vettori in cui l'*i*-esimo elemento del vettore a (*a[i]*) è sostituito da

$$a[i]\text{-}x * b[i] + y$$

L'accesso alla memoria, esterna all'unità U, avviene utilizzando un'interfaccia di memoria standard. L'unità U_c interagisce con U utilizzando un protocollo a domanda-risposta.

Si richiede la progettazione di U in modo che il numero di cicli di clock richiesti per il completamento dell'operazione esterna sia minimo.

In particolare, si richiede di calcolare il tempo di completamento relativo ad una singola esecuzione dell'unica operazione esterna implementata in funzione di *n*, t_p e t_a.

Successivamente, si richiede di calcolare il tempo di completamento in funzione dei soli *n* e t_p, assumendo che i dati siano tutti nella cache di primo livello, set associativa a 4 vie e che la cache sia sullo stesso chip di U.

Domanda 2

Si fornisca una procedura assembler D-RISC che implementi la stessa operazione della domanda 1. La procedura riceve tramite registri l'indirizzo di ritorno e l'indirizzo di un'area contigua di memoria che contiene tutti i parametri in input.

Di detta procedura si calcoli

- il tempo di completamento su un processore D-RISC pipeline dotato di EU slave che calcola la moltiplicazione fra interi in *2t*
- il tempo di completamento su un processore pipeline con le stesse caratteristiche ma superscalare a due vie

Si considerino eventuali ottimizzazioni.

Bozza di soluzione

Domanda 1

L'interfaccia d'ingresso avrà 5 registri da 32 bit (inda, indb, n, x, y) e una coppia di indicatori (RDYop, ACKop). L'interfaccia verso la memoria è quella standard.

Un possibile microcodice è il seguente:

0. *// attesa della ricezione di una richiesta di operazione, ordina 1ª lettura*
   (RDYop=0) nop, 0
   (=1) N-1 + INDa → IND, "read" → OP, set RDYm, N-1→ ITER, 1.

1. (ACKm,OR(ESITO),$ITER_0$=--1) 0→ ESITOop, reset RDYop, set ACKop, 0. *// fine lavori*

   (=0--) nop, 1. *// ricevo a[i] e ordina lettura b[i]*
   (=10-) ITER + INDb → IND, DATAIN + Y→ TEMP, reset ACKm, set RDYm, 2.
   (=11-) NOP, trattamento_eccezione.

2. (ACKm,OR(ESITO)=0-), nop, 2. *// ricevo b[i] completo ris, ordino scrittura*
   (=10) TEMP+DATAIN*X → DATAOUT, ITER+INDa→IND, "write"→OP,
   reset ACKm, set RDYm, 3. (=11) NOP, trattamento_eccezione.

3. (ACKm,OR(ESITO)=0--) nop, 3 // attendo esito scrittura e procedo
   (=101) ITER-1 + INDa → IND, "read"→ OP, reset ACKm, set RDYm, ITER-1→ ITER, 1
   (=11-) NOP, trattamento_eccezione

Il numero di microistruioni è minimo dal momento che servono, per ognuna delle iterazioni almeno tre microistruzioni per attendere l'esito dell'operazione richiesta al sottosistema di memoria. Un'ulteriore istruzione è necessaria per ordinare la prima operazione in memoria.

Per calcolare il tempo di completamento dobbiamo calcolare la lunghezza del ciclo di clock e moltiplicarlo per il numero di microistruzioni eseguite, sommando i tempi necessari per gli accessi in memoria.

Occorrono

1 + 3*N + 1

istruzioni (la 0. poi N volte la 1. 2. e 3. ed infine una 1. che prende la prima frase). Nelle le 3*N istruzioni centrali costano ($\tau+t_a$), per via dell'accesso in memoria.

Il ciclo di clock lo possiamo valutare calcolando i tempi per la stabilizzazione della ω_{PO}, σ_{PO}, ω_{PC}, σ_{PC} nella frase più lunga. ω_{PC}, σ_{PC} sono ovviamente uguali per tutte le frasi. Abbiamo 4 microistruzioni (2 bit di stato) e un massimo di 3 variabili di condizionamento testate contemporaneamente. Dunque 5 ingressi per il livello AND. Abbiamo 12 frasi, quindi al più 8 ingressi per il livello OR (se sono di più codifichiamo gli zeri e neghiamo l'uscita). Questo comporta che la PC introduce al più un ritardo di 2 t_p sia per la ω_{PC} che per la σ_{PC}. La frase più lunga dal punto di vista della PO è senz'altro la

(ACKm,OR(ESITO)=01) TEMP+DATAIN*X → DATAOUT, ITER+INDa→IND, "write"→OP, reset ACKm, set RDYm, 3.

che richiede ca. 2_{talu} (supponendo di avere una alu che fa addizione e moltiplicazione intere). Il calcolo delle variabili di condizionamento costa un t_p.

Questo porta ad avere un ciclo di clock

$$\tau = t_p + 2t_p + 2t_{alu} + t_p = 4t_p+2t_{alu}$$

Dunque il tempo di completamento dell'operazione esterna sarà

$$(3N+2)\ (4t_p+2t_{alu}) + (3Nt_a)$$

Nel caso in cui i dati sia tutti presenti nella cache di primo livello, avremo che

$$t_a = 2t_p$$

(un tp per la MMU, un tp per l'accesso alla memoria associativa su insiemi implementata secondo il modello "Hennessy-Patterson" visto nel corso), il che porta il tempo di completamento a

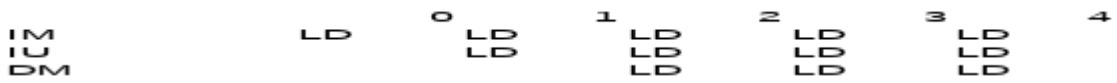
$$(3N+2)\ (4t_p+2t_{alu}) + (3N2t_p)$$

Domanda 2

Il codice assembler D-RISC per la procedura sarà il seguente:

```
proc:   LOAD Rparam, #0, Rinda
        LOAD Rparam, #1, Rindb
        LOAD Rparam, #2, Rn
        LOAD Rparam, #3, Rx
        LOAD Rparam, #4, Ry
        ADD R0, R0, Ri
loop:   LOAD Rinda, Ri, Rai
        LOAD Rindb, Ri, Rbi
        MUL Rbi, Rx, Rta
        SUB Rta, Rai, Rta
        ADD Rta, Ry, Rta
        STORE Rinda, Ri, Rta
        INC Ri
        IF< Ri, Rn, loop
cont:   GOTO Rret
```

Dalla simulazione vediamo che il tempo di completamento è pari a 9t+13Nt+t



Tuttavia potremmo facilmente ridurre le dipendenze nel ciclo interno considerando che a[i] + y può essere calcolato prima di avere il risultato di b[i]*x. Dunque se riorganizziamo il codice del codice interno come segue:

```
loop:   LOAD Rindb, Ri, Rbi
        MUL Rbi, Rx, Rtempb
        LOAD Rindb, Ri, Rai
        ADD Rai, Ry, Rtempa
        SUB Rtempa, Rtempb, Rres
        INC Ri
        IF< Ri, Rn, loop, delayed
        STORE Rinda', Ri, Rres
cont:   GOTO Rret
```

riduciamo il peso dell'iterazione interna a 11t:

Se consideriamo l'esecuzione su un processore pipeline superscalare a due vie, possiamo pensare di raggruppare le istruzioni come segue (ci limitiamo al ciclo interno):

```
loop:   LOAD Rindb, Ri, Rbi          LOAD Rindb, Ri, Rai
        MUL Rbi, Rx, Rtempb          ADD Rai, Ry, Rtempa
        SUB Rtempa, Rtempb, Rres     INC Ri
        IF< Ri, Rn, loop             STORE Rinda', Ri, Rres
        GOTO Rret                    NOP
```

In questo caso, per il ciclo interno

spendiamo 9t invece, risparmiando un 10% rispetto al processore pipeline (non superscalare)