Architettura degli Elaboratori – A.A. 2009-2010

4° appello – 7 settembre 2010

Riportare su tutti i fogli consegnati nome e cognome, numero di matricola, corso di appartenenza, programma d'esame (NEW, OLD-0, OLD-1). risultati saranno pubblicati sulle pagine web dei docenti, appena disponibili, insieme al calendario degli orali.

Domanda 1 (*tutti*) Una unità firmware U implementa uno stack di interi da 32 bit, con capacità massima pari a 4K posizioni. Lo stack è realizzato utilizzando una memorietta interna. L'unità riceve richieste di inserimento (*push*) da parte di un'unità U_1 e di estrazione (*pop*) da parte di un'unità U_2, e serve prioritariamente le richieste di *pop*. Inoltre, ad ogni operazione, invia ad una unità U_3 il numero di elementi in quel momento presenti nello stack. Si assume che l'unità U_3 sia sempre pronta a ricevere tale contatore. Si richiede: i) di progettare l'unità, ii) di valutarne il ciclo di clock e iii) di fornirne il tempo di servizio, assumendo che *push* e *pop* siano equiprobabili.

Domanda 2 (*tutti*) Su una serie di vettori in memoria A_0, ... A_N (tutti di dimensione N=1K), si vuole calcolare il vettore di N elementi in cui l'elemento *i-esimo* corrisponde al prodotto scalare A_0 x A_i (si ricorda che il prodotto scalare è la somma dei prodotti degli elementi corrispondenti nei due vettori, ovvero $A_0 \times A_i = \Sigma_{K=1,N} (A_0[k] * A_i[k])$).

Si fornisca il codice assembler D-RISC della funzione che calcola il prodotto fra due vettori, nonchè il codice del programma assembler che calcola il vettore risultato a partire dall'indirizzo di un vettore da N+1 posizioni che contiene gli indirizzi base in memoria dei vettori A_i e dalla costante N.
Si fornisca la mappa dettagliata della memoria virtuale del processo che esegue il programma.
Si discuta il working set del programma e il tipo di accessi effettuati in memoria, assumendo la presenza di una cache associativa su insiemi con insiemi da 4 posizioni, σ=16.

Domanda 3

(*new, old-0*) Si discuta, fornendo tutti i dettagli ritenuti necessari, la seguente affermazione: «l'introduzione della tecnica di salto ritardato (delayed branch) permette la risoluzione di tutte le bolle derivanti dai salti condizionati nell'architettura pipeline semplificata».

(*old-1*) Si descriva la struttura dati utilizzata per implementare un canale asincrono con grado di asincronia 1, e si fornisca lo pseudocodice della send e della receive che operano su questo canale.

Per questo insegnamento sono previsti, durante l'intero a.a., 6 appelli ed un numero massimo di 5 prove.

Gli studenti del vecchio ordinamento devono scegliere se presentarsi sul programma di esame 2009-10 con l'aggiunta del Cap. VII (Processi), sez. 5, 6 e 7, oppure sul programma di esame 2008-09. La scelta va fatta la prima volta che lo studente si iscrive all'esame, e vale per tutti gli eventuali successivi appelli dell'a.a. ai quali lo studente si presenti. Riportare su tutti i fogli consegnati nome, cognome, numero di matricola, corso di appartenenza, e la sigla NEW(per nuovo ordinamento), oppure OLD-0 (per vecchio ordinamento, programma 2009-10), oppure OLD-1 (per vecchio ordinamento, programma 2008-09).

Bozza di soluzione

Domanda 1

Lo stack è implementato mediante una memoria (M) da 4K posizioni (e quindi indirzzato utilizzando un registro da 12 bit), un registro puntatore (SP, da 12 bit), che punta alla prima posizione libera dello stack, a un registro booleano (EMPTY) che dice se lo stack è vuoto, e da un registro booleano (FULL) che dice se lo stack è pieno.

I registri FULL ed EMPTY sono aggiornati durante l'operazione di PUSH e POP corrente, in modo da poter essere testati alla prossima richiesta di operazione.

In caso di stack pieno e richiesta di push o di stack vuoto e richiesta di pop ci bocchiamo, ovvero non si esegua la richiesta e si aspetta invece una richiesta che possa essere soddisfatta.

La priorità alle operazioni di pop viene implementata testando prima il RDY da U2 e poi quello da U1. Nel caso di stack vuoto, con richiesta di pop e push contemporanee, si passa direttamente il dato in arrivo da U1 ad U2, senza aggiornare lo stack vero e proprio.

Il microprogramma potrebbe essere quello che segue (esistono ovviamente versioni alternative con microistruzioni e prestazioni diverse):

```
0.  // inizializzazioni
    1→EMPTY, 0→FULL, 0→TOP, 1

1.  (RDY1,RDY2,RDY3,EMPTY,FULL=00---) nop, 1      // nessuna richiesta
    // richiesta di pop con stack non vuoto
    (=0110-) M[TOP-1]→OUT2, reset RDY2, set ACK2,
          TOP →OUT3, reset RDY3, set ACK3,
          TOP-1→TOP, zero(TOP-1)→empty, 0→full, 1
    // richiesta di push con stack non pieno e nessuna richiesta di pop
    (=101-0) IN1→M[TOP], reset RDY1, set ACK1,
          TOP+1→OUT3, set ACK3, reset RDY3,
          TOP+1→TOP, zero((TOP+1)[0..11])→FULL, 0→EMPTY, 1
    // richiesta di push e pop contemproranea con stack vuoto
    (=11110) IN1→OUT2, reset RDY1, set ACK1, reset RDY2, set ACK2,
          reset RDY3, set ACK3, TOP→OUT3,1
    // richiesta di push e pop contemporanea con stack non vuoto
    (=11100) M[TOP-1]→OUT2, reset RDY2, set ACK2,  // pop dallo stack
          IN1→M[TOP-1], reset RDY1, set ACK1,      // push sullo stack
          TOP→OUT3, reset RDY3, set ACK3,          // comunicazione conto a U3
    (=10--1) nop, 1 // stack pieno con richista di push
    (=01-1-) nop, 1 // stack vuoto, richiesta di pop
```

Assumendo che un accesso in M costi 10τ, possiamo calcolare il ciclo di clock come segue:

$T_{\omega PO} = 0$

$T_{\sigma PO} = 19\tau$ (ALU + accesso in memoria (con commutatore su indirizzo) + commutatore per scrivere in OUT2, l'operazione più lunga è la scrittura M[TOP-1]→OUT2)

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

$\delta = \tau$

per un totale di 22τ.

Il tempo di servizio è dato comunque dal tempo necessario per eseguire una microistruzione, quindi è pari ad 1τ. (consideriamo nullo il contributo della 0, visto che viene eseguita una volta sola in fase di inizializzazione. Tale inzializzazione è necessaria dal momento che EMPTY deve essere inizialmente 1)

Domanda 2

Il codice ad alto livello dell'operazione da eseguire è il seguente:

```
for(int i=1; i<=N; i++) {
    s[i]=prod_vett(V[0], V[i], N);

prod_vett(v0, vi, N) {
    s=0;
    for(int k=0; k<N; k++) {
        s+=v0[k]*vi[k];
    }
    return s;
}
```

Assumendo che i parametri alla procedura prod_vett siano passati per registro, mediante i registri Rv0, Rvi e Rn, il codice assembler è il seguente:

```
PROD_VETT:  CLEAR Rs
        CLEAR Rk;
loop:   LOAD Rvi, Rk, R1
        LOAD Rv0, Rk, R2
        MUL R1, R2, Rtemp
        ADD Rtemp, Rs, Rs
        INC Rk
        IF< Rk, RN, loop
        GOTO Rret

main:   CLEAR Ri
mloop:  LOAD Rbasevettore, R0, Rv0
        LOAD Rbasevettore, Ri, Rvi
        //   assumiamo   che   RN   sia   già   il   regsitro
        //   per il terzo parametro N
        CALL PROD_VETT, Rret
        STORE Rbaserisultati, Ri, Rs
        INC Ri
        if<= Ri, RN, mloop
```

La memoria virtuale del processo conterrà il codice del programma, i dati (vettori iniziali, vettore degli indirizzi (inizializzati) e vettore dei risultati), il codice delle routine appartenenti al nucleo del S.O. (schedulazione, trattamento interruzioni), le strutture dati del S.O. (tabella dei processi, etc.) (vedi dispensa/libro).

Per gli accessi al primo vettore valgono sia la località che il riuso. Per gli altri vettori in input vale solo la località. Per il vettore dei risultati vale località e riuso.

Il working set del programma contiene il primo vettore, il vettore *i-esimo* e la *i-esima* posizione del vettore dei risultati, oltre ai blocchi relativi al codice. La cache associativa su insiemi con insiemi da 4 posizioni garantisce che non vi siano fault oltre a quelli fisiologici Se fosse stata a 2 vie, avremmo potuto avere gli indirizzi di A0, Ai e codice mappati sullo

stesso insieme e questo avrebbe comportato una situazione con un altissimo numero di fault (dell'ordine di N invece the N/σ dati).

Domanda 3

(*new. old-0*) l'affermazione è vera a patto che esistano istruzioni nel codice originale che possono essere spostate negli slot del salto ritardato senza violare la semantica del codice. Qualora non esistesse un numero sufficienti di tali istruzioni, l'uso del salto ritardato richiederebbe l'intruduzione di NOP, che di fatto hanno lo stesso effetto di una «bolla».

(*old-1*) vedi libro/dispensa.