Architettura degli Elaboratori, 2007-08

Appello del 26 giugno 2008

Domanda 1

Una unità di elaborazione U è così definita:

- ha al suo interno una memoria Q contenente una matrice quadrata 16 × 16 di interi, ed un registro S di 16 bit;
- riceve da una unità U_M una coppia di valori *(i, d),* con *i* di 4 bit e *d* di 5 bit, ed invia il risultato dell'elaborazione all'unità U_d appartenente all'insieme (U_0, …, U_{31});
- sapendo che il valore *i* rappresenta l'indice di una riga di Q, il valore calcolato da U è uguale a

$$\sum_{j=0}^{15} (-1)^{S_j} * Q_{i,j}$$

Nell'ipotesi che il tempo di accesso di Q sia uguale a $10t_p$ e che il ritardo di stabilizzazione di una ALU sia uguale a $5t_p$, con t_p ritardo di stabilizzazione di una porta logica con al più 8 ingressi, scrivere il microprogramma e determinare il tempo di elaborazione di U. Spiegare e motivare le scelte fatte per la progettazione di U e la scrittura del microprogramma.

Domanda 2

Una unità di I/O collegata ad una CPU D-RISC è definita in maniera tale che il messaggio di interruzione alla CPU, per segnalare un certo evento *ev*, consiste in *n* parole, dove il valore di *n* è di volta in volta variabile e noto all'unità stessa.

Lo handler di *ev* provvede a copiare le *n* parole del messaggio di interruzione in una certa struttura dati B condivisa tra tutti i processi e nota ad ogni processo.

Spiegare come progettare lo handler, e scrivere lo handler in D-RISC.

Domanda 3

Si consideri un programma che, dato un array Q di N × N interi ed un array S di N bit, con N = 1K, calcola un array P di N interi tale che:

$$\forall i = 0 \,..\, N-1 : \; P_i \; = \; \sum_{j=0}^{N-1} (-1)^{S_j} * Q_{i,j}$$

Il programma viene eseguito su una CPU D-RISC con cache primaria completamente associativa di capacità 64K parole, blocchi di 8 parole, e scritture con il metodo Write-Through.

a) Fornendo adeguate spiegazioni, valutare l'insieme di lavoro del programma, il numero di trasferimenti di blocchi di cache che avvengono durante la sua esecuzione, ed il loro peso sul tempo di completamento inteso come ordine di grandezza.

b) Compilare il programma.

Sintesi di soluzione

Domanda 1

Il calcolo consiste nel sommare tutti gli elementi della riga *i*-esima di *Q*, dove l'elemento *j*-esimo, incluso il primo, è preso con il suo segno oppure negato a seconda che il bit *S[j]* sia uguale rispettivamente a zero oppure a uno.

Applichiamo il controllo residuo in tre occasioni:

- per leggere il bit *S[j],* inserendo all'uscita di *S* un commutatore comandato dal valore *j*;
- detto *P* il risultato parziale del calcolo, inizializzato a zero, una ALU, comandata dal bit *S[j],* esegue la seguente funzione:

$$Alu\ (P, outQ, S[j]) = if\ (S[j] = 0)\ \ then\ P + outQ\ \ else\ P - outQ;$$

- per considerare come memorie di 16 locazioni, indirizzate dal valore *d*, gli insiemi dei vari indicatori di interfaccia e l'insieme dei registri di uscita verso le unità destinatarie.

Essendo la matrice memorizzata per righe, il valore ricevuto di *i* viene moltiplicato per 16 (shift di 4 posizioni a sinistra) per ottenere l'indirizzo base *IND* dell'array Q.

Il microprogramma (con *J* di 5 bit, *IND* di 8 bit) è:

0. (RDYM = 0) nop, 0;

   (= 1) reset RDYM, set ACKM, $sh\_left^4(i) \rightarrow$ IND, d $\rightarrow$ DEST, 0 $\rightarrow$ P, 0 $\rightarrow$ J, 1

1. (J_0, ACK[DEST] = 0 –) Alu (P, Q[IND], $S[J_m]$) $\rightarrow$ P, J + 1 $\rightarrow$ J, IND + 1 $\rightarrow$ IND, 1;

   (= 1 0) nop, 1;

   (= 1 1) reset ACK[DEST], set RDY[DEST], P $\rightarrow$ OUT[DEST], 0

Si usano tre ALU per permettere il massimo parallelismo nelle microperazioni; una ALU esegue sia $sh\_left^4(i)$ che *J + 1*. L'operazione *Alu (P, Q[IND], S[J_m])* è quella con il massimo ritardo di stabilizzazione: si stabilizzano in parallelo la memoria *Q* ($10t_p$) e il commutatore all'uscita di *S* ($3t_p$), quindi la ALU e infine il commutatore all'ingresso di *P*. Quindi

$$T_{\sigma PO} = 17t_p$$

Inoltre:

$$T_{\omega PO} = 3t_p \qquad \text{(commutatore dei 32 ACK)}$$

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

Quindi:

$$\tau = 23t_p$$

$$T = 18\ \tau = 414t_p$$

Domanda 2

In un calcolatore con CPU D-RISC le unità di I/O, dove aver ricevuto l'ack dell'interruzione, inviano alla CPU, via Bus di I/O, due parole: la prima codifica l'evento, la seconda un valore usato come parametro dallo handler. Nel nostro caso, l'unità di I/O memorizza, in una struttura dati *S* nella sua memoria (MI/O), le *n* parole precedute dal valore di *n*. Il valore *X* inviato alla CPU nella seconda parola deve permettere allo handler di indirizzare *S* mediante Memory Mapped I/O.

Se il sistema adotta il metodo a *capability* per riferire le strutture dati condivise riferite indirettamente, X è la capability di S. Altrimenti, se si adotta il metodo a indirizzi logici distinti, X è un intero che identifica univocamente S nello spazio logico di indirizzamento del processo che esegue lo handler.

Nel caso a capability, il processo acquisisce dinamicamente S nel proprio spazio logico di indirizzamento. Per ogni processo, l'indirizzo logico della struttura dati B, in cui copiare S, è contenuto in un a posizione nota del proprio PCB.

Se RG[Rsecond] contiene la capability di S, il codice dello handler è:

```
        // acquisisci capability di S, supposta di una parola //
                LOAD   Rpcb, Roffset1, Rtabril
                STORE  Rtabril, Rfree, Rsecond
                MUL    Rfree, Rpagesize, RS
        // leggi n //
                LOAD   RS, 0, Rn
        // leggi indirizzo di B //
                LOAD   Rpcb, Roffset2, RB
        // loop di copia di S in B //
                INCR   RS
                CLEAR  Ri
LOOP:           LOAD   RS, Ri, Rtemp
                STORE  RB, Ri, Rtemp
                INCR   Ri
                IF <   Ri, Rn, LOOP
        // rilascia capability di S //
                STORE  Rtabril, Rfree, 0
        // ritorno da procedura //
                GOTO   Rret
```

Domanda 3

a) Il programma consta di un doppio loop, complessivamente con N^2 iterazioni.

Il compilatore è in grado di verificare che, per ogni riga di Q, si ha riuso dell'intero vettore S. Poiché S può essere contenuto tutto in cache, viene imposto che i blocchi di S non siano deallocati dalla cache una volta caricativi.

Per gli array Q e P è sufficiente che risieda in cache solo il blocco corrente. Quindi:

$$working\ set = 2 + \frac{N}{\sigma}$$

Essendo P in sola scrittura, il numero di trasferimenti di blocchi è dato da:

$$numero\ trasferimenti\ blocchi = 2\,\frac{N}{\sigma}$$

che, essendo di ordine N, è trascurabile rispetto al tempo di completamento ideale, di ordine N^2.

b) La compilazione del programma in D-RISC è (RG[Ri] inizializzato a zero):

```
LOOPi:  CLEAR   Rj
        CLEAR   Rp
LOOPj:  LOAD    RQ, Rj, Rq
        LOAD    RS, Rj, Rs, non_deallocare
        IF ≠ 0  Rs, CONT
        SUB     0, Rq, Rq
CONT:   ADD     Rp, Rq, Rp
        INCR    Rj
        IF <    Rj, RN, LOOPj
        STORE   RP, Ri, Rp
        ADD     RQ, RN, RQ
        INCR    Ri
        IF <    Ri, RN, LOOPi
        END
```