Architettura degli elaboratori – A.A. 2016-17

Primo Appello – 12 gennaio 2017

Riportare nome , cognome, numero di matricola e corso di appartenenza in alto a destra su tutti i fogli consegnati.
I risultati saranno pubblicati via web appena disponibili

Domanda 1

Si consideri un programma che, dati N vettori K_1, … K_N ed un vettore A_0 ciascuno di interi e di N posizioni, calcola un vettore RES di N posizioni la cui i-esima posizione è il risultato del prodotto fra il vettore K_i ed il vettore A_0. Il prodotto fra due vettori di interi A e B di N posizioni è definito come la somma degli N prodotti A[i]*B[i]. Gli indirizzi di partenza dei vettori K_1 , …, K_N sono contenuti in un vettore IND di N posizioni. Lo pseudo codice potrebbe essere:

```
for(int v=0; v<N; v++) {
      int sum = 0;
      for(int i=0; i<N; i++)
            sum += (INDS[v])[i] * A0[i];
      res[v] = sum;
}
```

Si fornisca il codice D-RISC del programma e quindi, assumendo di lavorare in un sistema con gerarchia di memoria a 2 livelli (cache set associativa a 2 vie, σ=8, 1K insiemi, on chip, memoria principale interallacciata con 4 moduli da 1M parole ciascuna, $\tau_M = 50\tau$, off chip):

- se ne valutino le prestazioni su un'architettura D-RISC pipeline con unità EU_{slave} che calcola la moltiplicazione fra interi in *2t*
- se ne fornisca il working set e il numero di fault di cache
- si ottimizzi il codice prodotto, stimando il guadagno ottenuto in termini di tempo di servizio

Successivamente, si consideri il caso in cui il calcolo del prodotto fra vettori avviene mediante una funzione (parametri in ingresso: indirizzi dei vettori e lunghezza, parametri in uscita: risultato, tutti passati in memoria). Si fornisca il codice della procedura e si discutano le eventuali variazioni prestazionali.

Domanda 2

Si consideri un'unità U che contiene al proprio interno una memoria M da 8K parole. L'unità è interfacciata a due unità U_1 ed U_2 dalla quale riceve rispettivamente interi e un segnale di pura sincronizzazione. E' interfacciata inoltre a due unità U_3 ed U_4 alle quali invia interi. L'unità implementa un unica operazione esterna:

- ricerca del numero di elementi pari in M minori dell'intero ricevuto da U_1 e invio ad una tra U_3 o U_4 del risultato.

All'inizio delle operazioni, U invia il risultato calcolato ad U_3. In caso riceva il segnale da U_2, a partire da quel momento U invia i risultati ad U_3 , se prima li inviava a U_4, oppure a U_4 se prima li inviava a U_3 . Lo scambio vale anche per l'operazione eventualmente in esecuzione quando U riceve il segnale da U_2. Dell'unità U si determini il tempo di servizio in funzione di t_p. Si hanno a disposizione componenti standard ALU in grado di sommare o sottrarre interi in $5t_p$.

Bozza di soluzione

Domanda 1

Discutiamo prima il WS. L'accesso al codice avviene con località e riuso. L'accesso al vettore A0 avviene pure con località e riuso, mentre per i vettori Ki e per RES c'è solo località. Il working set contiene quindi:

- tutto il codice
- tutto il vettore A0
- una linea del vettore K utilizzato al momento
- una linea del vettore RES
- una linea del vettore INDS

La dimensione (in numero di linee della cache) del Working Set è quindi pari a (assimiamo 0 fault per RES che è in sola scrittura, ma occorre comunque allocare le N/s linee man mano che servono; semplicemente non occorre trasferire dalla memoria principale il contenuto una volta allocate, visto che viene immediatamente soprascritto):

$$\#istruzioni/\sigma + N/\sigma + 1 + 0 + 1$$

Il numero di fault "fisiologici" può essere calcolato come numero di fault per il codice, per A0, per i vettori Ki, per RES e per INDS, ovvero

$$\#istruzioni/\sigma + N/\sigma + N*(N/\sigma) + N/\sigma + 0 = (\#istruzioni + 2N + N^2)/\sigma$$

Va notato, che, essendo la cache primaria associativa a 2 vie, ci potremmo trovare nella situazione che gli indirizzi fisici di A0 Ki e RES mappano tutti sullo stesso insieme e dunque avremmo un questo caso un fenomeno di trashing (elementi del working set continuamente eliminati dalla cache). Qualora questo non succedesse ed utilizzando il flag NON_DEALLOCARE per gli accessi ad A0 e il flag PREFECTH per tutti gli accessi ai vettori, avremmo, oltre ai fault per il codice, solo i fault per il primo accesso ad A0, INDS e RES e per ciascuno dei vettori K, cioè N+3 fault oltre ai fault dovuti al codice.

Qualora valesse (ca.) N>=8K, la dimensione del WS sarebbe maggiore della dimensione della cache di primo livello e pertanto avremmo un numero di fault molto maggiore al numero di fault fisiologici appena calcolato in quanto A0, con una politica LRU, andrebbe ricaricato interamente ad ogni iterazione del loop interno

Lo pseudo codice è compilato nel codice assembler D-RISC come segue:

```
loopv: CLEAR Rsum
       LOAD RbaseINDS, Rv, RbaseVv
```

```
       CLEAR Ri
loopi: LOAD RbaseVv, Ri, Rvvi
       LOAD RbaseAo, Ri, Rai
       MUL Rvvi, Rai, Rtemp
       ADD Rsum, Rtemp, Rsum
       INC Ri
       IF< Ri, RN, loopi
contv: STORE RbaseRes, Rv, Rsum
       INC Rv
       IF< Rv, RN, loopv
cont:  END
```

Nel codice abbiamo dipendenze IU-EU (fra la **`INC Ri`** e la **`IF< Ri, RN, loopi`**, fra la **`INC Rv`** e la **`IF< Rv, RN, loopv`** e fra la **`LOAD RbaseINDS, Rv, RbaseVv`** e la **`LOAD RbaseVv, Ri, Rvvi`** che però conta solo per la prima iterazione del ciclo interno) e dipendenze EU-EU (fra la **`MUL Rvvi, Rai, Rtemp`** e la **`ADD Rsum, Rtemp, Rsum`**).

Consideriamo l'ottimizzazione del ciclo più interno. Il ciclo impiega 11t per un'iterazione da 6 istruzioni (T=11t/6, ε=6/11)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IM	LOAD	LOAD	MUL	ADD	INC	IF<					STORE	LOAD			
IU		LOAD	LOAD	MUL	ADD	INC	IF<			IF<	IF<	STORE	LOAD		
DM			LOAD	LOAD										LOAD	
EU				LOAD	LOAD	MUL	ADD		ADD	INC					LOAD
EUslave							MUL	MUL							

Possiamo anticipare la INC per ridurre l'effetto della dipendenza sulla IF< e allontanare la MUL dalla ADD (dipendenza EU-EU). Dal momento che la ADD non induce altre dipendenze, possiamo anche pensare di mettere la ADD nel delay slot della IF< ottenendo il codice (per il solo loop interno):

```
loopi: LOAD RbaseVv, Ri, Rvvi
       LOAD RbaseAo, Ri, Rai
       INC Ri
       MUL Rvvi, Rai, Rtemp
       IF< Ri, RN, loopi, dealyed
       ADD Rsum, Rtemp, Rsum
```

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

IN questo modo T = 7t/6 e dunque ε=6/7, nettamente migliore di prima.

In caso utilizzassimo una procedura con passaggio dei parametri per memoria (parametri in ingresso 2 indirizzi e un intero, parametro in uscita un intero), dovremmo considerare il codice assembler:

```
Fprod: LOAD Rparam, #0, Rbase1
       LOAD Rparam, #1, Rbase2
```

```
        LOAD Rparam, #3, Rn
        CLEAR Ri
        CLEAR Rsum
loop:   LOAD Rbase1, Ri, R1
        LOAD Rbase2, Ri, R2
        MUL R1, R2, R1
        ADD Rsum, R1, Rsum
        INC Ri
        IF< Ri, Rn, loop
        STORE Rparam, #4, Rsum
        GOTO Rret

main:   CLEAR Rv
loopv:  LOAD RbaseINDS, Rv, R1
        STORE Rparam, #0, R1
        STORE Rparam, #1, RbaseA0
        STORE Rparam, #2, RN
        CALL Rfvett, Rret
        LOAD Rparam, #4, R2
        STORE RbaseRES, Rv, R2
        INC Rv
        IF< Rv, RN, loopv
cont:   END
```

Il ciclo interno alla funzione fvett potrebbe essere ottimizzato assoultamente come prima e non si osserva alcun rallentamento dovuto alle istruzioni di inizializzazione per il reperimento dei parametri:

	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	
IM	LOAD	LOAD	LOAD	CLEAR	CLEAR	LOAD	LOAD	INC	MUL	IF<		ADD	LOAD			
IU		LOAD	LOAD	LOAD	CLEAR	CLEAR	LOAD	LOAD	INC	MUL	IF<z	IF<	ADD	LOAD		
DM			LOAD	LOAD	LOAD			LOAD	LOAD						LOAD	
EU				LOAD	LOAD	LOAD	CLEAR	CLEAR	LOAD	LOAD	INC	MUL		ADD	ADD	LOAD
EUslave													MUL	MUL		

Dunque il T (per il for i interno) rimane pari a 7t/6.
Quello che cambia (e molto) è il ciclo più esterno, dove ora dobbiamo preparare i parametri per la chiamata della funzione. È da notare, tuttavia, che il solo parametro caricato inizialmente in R1 varia fra una chiamata e l'altra e quindi il codice si poteva semplificare significativamente rimuovendo gli invarianti dal loop:

```
main:   CLEAR Rv
        STORE Rparam, #1, RbaseA0
        STORE Rparam, #2, RN
loopv:  LOAD RbaseINDS, Rv, R1
        STORE Rparam, #0, R1
        CALL Rfvett, Rret
        LOAD Rparam, #4, R2
        STORE RbaseRES, Rv, R2
        INC Rv
        IF< Rv, RN, loopv
cont:   END
```

Va considerato che sebbene l'efficienza nell'esecuzione del ciclo interno rimanga la stessa, il numero di istruzioni da eseguire è più alto.

Commento generale

Indipendentemente dalla versione di codice compilato, la valutazione di qualunque misura di prestazioni può essere fatta o sul ciclo più interno (quello eseguito N^2 volte) o, se si vuole considerare l'intero codice, sull'esecuzione di N iterazioni del codice del ciclo esterno con N iterazioni del ciclo interno. Quello che NON si deve fare è valutare il costo dell'esecuzione di tutte le istruzioni del codice compilato come se fossero eseguite una volta sola (dunque una sola iterazione del ciclo esterno, con una sola iterazione del ciclo interno).

Domanda 2

Assumendo che:

- l'interfaccia da U1 sia costituita dagli indicatori a transizione di livello RDY1 (in ingresso) e ACK1 (in uscita) e dal registro di ingresso X
- l'interfaccia da U2 abbia solo gli indicatori RDY2 (in ingresso) e ACK2 (in uscita)
- l'interfaccia verso U3 abbia il registro OUT3 in uscita e gli indicatori RDY3 (in uscita) e ACK3 (in ingresso)
- come al solito che i registri siano tutti inizializzati a 0 al momento dell'avvio
- di non dare la disponibilità ad una nuova richiesta ad U1 prima che la precedente non sia stata soddisfatta
- di avere un registro TURNO che indica quale delle due unità deve essere utilizzata per l'invio
- di utilizzare controllo residuo per
    - selezionare ACK dell'unità cui si vuole inviare (In lettura, per testare se l'unità cui si vuole inviare è pronta a ricevere: ACK3 e ACK4 ingressi di un commutatore il cui ingresso di controllo è il registro TURNO; l'uscita del commutatore sarà $ACK|_{TURNO}$, è una variabilie di condizionamento che non viola la condizione di correttezza in quanto funzione del solo stato interno. In scrittura, per il reset, come per i RDY3 e RDY4, vedi punto seguente)
    - selezionare RDY dell'unità cui si invia (β_{RDYout} ingresso di un selezionatore comandato da TURNO; le uscite del selettore saranno i β_{RDY3} e β_{RDY4} che servono per il set dell'indicatore corrispondente)
    - selezionare il registro di interfaccia utilizzato per inviare ad U3 o U4 il risultato (come per il RDY, si usa un selettore comandato da TURNO per ottenere $OUT|_{TURNO}$ ovvero per trasformare un β_{OUT} in β_{OUT3} o β_{OUT4} a seconda di TURNO)
- che f(X,Y) calcoli la coppia di valori da 1 bit S,Z,P tali per cui:
    - S = segno(X-Y)
    - P = X_0 (bit meno significativo di X)

il micro codice potrebbe essere scritto come segue:

1. (RDY1,RDY2=00) nop, 0 // nessuna richiesta, attesa
   (=01) not(TURNO) → TURNO, reset RDY2, set ACK2, 0 // richiesta da U2: cambio turno

(=10) 1 → I, f(M[0],X) →(S,P), 0→C,1 // richiesta da U1: avvio calcolo

(=11) 1 → I, f(M[0],X) →(S,P), 0→C , // richiesta da entrambe: avvio

not(TURNO) → TURNO, reset RDY2, set ACK2,1 // calcolo e cambio turno immediato

2. (RDY2,ACK|$_{TURNO}$, S, P, I_0=1----) // cambio turno, può avvenire

not(TURNO) → TURNO, reset RDY2, set ACK2, 1 // durante un ciclo di calcolo

(=0-0--) I+1 → I, f(M[I],X) →(S,Z,P), 1 // non minore, vai avanti

(=0--1-) I+1 → I, f(M[I],X) →(S,Z,P), 1 // non pari vai avanti

(=0-100) I+1 → I, C+1 → C, f(M[I],X) →(S,Z,P), 1, // minore e pari vai avanti

(=01101) C+1 → OUT|$_{TURNO}$, // fine ciclo, va contato anche l'ultimo

reset RDY1, set ACK1, // output del valore calcolato

set RDY|$_{TURNO}$, reset ACK|$_{TURNO}$, 0 // mediante controllo residuo

(=01--1) C→ OUT|$_{TURNO}$, // fine ciclo, output del valore calcolato

reset RDY1, set ACK1, // mediante controllo residuo

set RDY|$_{TURNO}$, reset ACK|$_{TURNO}$, 0

(=00--1) nop, 1 // attesa disponibilità a ricevere

Abbiamo, due microistruzioni e 6 variabili di condizionamento, delle quali per non se ne testano mai più di 5 contemporaneamente (6 input al livello AND: 1 bit di stato e 5 variabili di condizionamento), con 10 frasi (al più 5 1 in input al livello OR, se sono di più codifichiamo gli zeri e neghiamo l'uscita). Dunque la PC può essere realizzata con σ e ω che lavorano in $2t_p$.

Le variabili di condizionamento sono tutte letture da registri, tranne ACK|$_{TURNO}$, che richiede un commutatore (comandato mediante controllo residuo per garantire la condizione di correttezza) per il calcolo (e dunque per quella microistruzione $T_{\omega PO}=2t_p$).

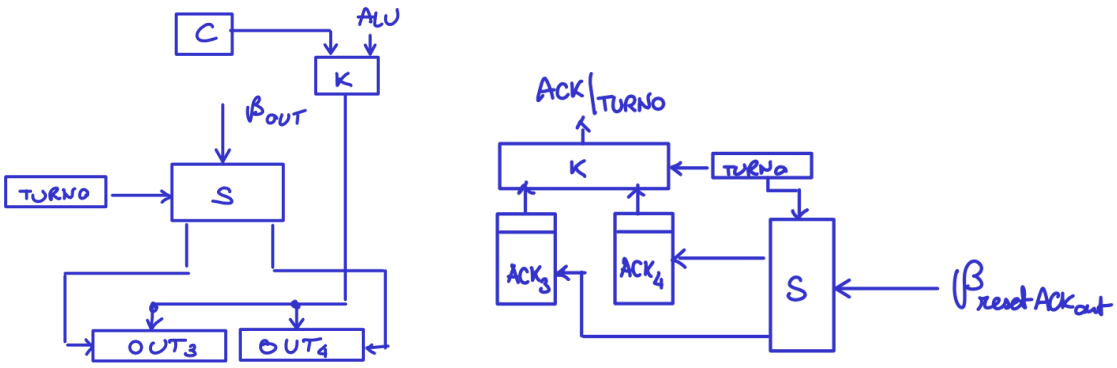
La scrittura in I e C richiede un commutatore per scegliere fra il valore dell'inizializzazione e l'uscita della ALU (dunque I+1 I e C+1 C costano $2t_p+t_{alu}=7t_p$).

Il calcolo di f(M[..],X) (dunque del segno(M[...]-X)) richiede t_k (perchè c'è un commutatore sugli indirizzi per inizializzazione e scorrimento) + t_a (per leggere il valore M[...]) + t_{alu} (per calcolare il flag del segno della sottrazione).

Il tempo di accesso a una memoria da 8K (#celle) posizioni è dominato dal tempo del commutatore di lettura. Sappiamo che un commutatore ha (($\log_2$(#celle))+1) ingressi al livello AND e #celle ingressi al livello OR. Nel nostro caso, 14 ingressi al livello AND (servono 2 livelli di porte, quindi stabilizza in $2t_p$) e 8K ingressi al livello OR (servono la parte intera superiore di $\log_8$(8K) ($\log_8$(8K) = $\log_2$(8K) / $\log_2$(8) = 13/3 = 4.xx) livelli di porte OR, quindi stabilizza in $5t_p$). Di conseguenza $t_a = 7\ t_p$.

Complessivamente abbiamo dunque che il calcolo di segno(M[...]-X) costa $2t_p + 7t_p + 5t_p = 14\ t_p$.

Le espressioni ACK|$_{TURNO}$, RDY|$_{TURNO}$, OUT|$_{TURNO}$, sono realizzati utilizzando controllo residuo. Per esempio, la scrittura in OUT|$_{TURNO}$ o la gestione deigli ACK in ingresso da U3 o U4 può essere implementata così:



Il costo di $T_{\omega PO}$ e $T_{\sigma PO}$ nelle varie frasi è

Istruzione	Frase	$T_{\omega PO}$	$T_{\sigma PO}$	Totale
0.	1,2	0	0	0
0.	3,4	0	$14t_p$	$14t_p$
1.	1	$2t_p$	0	$0t_p$
1.	2,3,4	$2t_p$	$14t_p$	$16t_p$
1.	5,6	$2t_p$	t_p (selettore)	$3t_p$
1.	7	$2t_p$	0	$2t_p$

Le frasi più lunghe, che determinano la lunghezza del ciclo di clock sono le frasi 2, 3 e 4 della seconda microistruzione. Il ciclo di clock in funzione di tp sarà calcolato applicando la solita formula ai valori di tale frase, quindi:

$$\tau = T_{\omega po} + \max\{T_{\sigma PC}, T_{\omega PC} + T_{\sigma PO}\} + \delta = 2t_p + \max\{2t_p, 2t_p + 14t_p\} + t_p = 19t_p$$

Si richiedeva di dare il tempo di servizio dell'unità U. L'operazione "cambio destinazione" richiede un singolo ciclo di clock (sia nella prima microistruzione che nella seconda). L'operazione "numero di interi pari minori di" richiede 1 istruzione per inizializzare i contatori ed i flag, e 8K istruzioni per scorrere la memoria. Il tempo di servizio sarà dunque

$$p_0\,\tau + p_1\,(1+8K)\,\tau$$

Utilizzando variabili di condizionamento complesse avremmo potuto scrivere il microprogramma diversamente:

1. (RDY1,RDY2=00) nop, 0 // nessuna richiesta, attesa
   (=01) not(TURNO) → TURNO, reset RDY2, set ACK2, 0 // richiesta da U2: cambio turno
   (=10) 0 → I, 0→C,1 // richiesta da U1: avvio calcolo
   (=11) 1 → I, 0→C , not(TURNO) → TURNO, reset RDY2, set ACK2,1// calcolo e cambio turno
2. (RDY2,ACK|$_{TURNO}$, segno(M[I]-X), $(M[i])_0$, I_0=1-----)// cambio turno, può avvenire
   not(TURNO) → TURNO, reset RDY2, set ACK2, 1 // durante un ciclo di calcolo
   (=0-0--) I+1 → I, 1 // non minore, vai avanti

(=0--0-) I+1 $\rightarrow$ I, 1 // non pari vai avanti

(=0-100) I+1 $\rightarrow$ I, C+1 $\rightarrow$ C, 1, // minore e pari vai avanti

(=01--1) C$\rightarrow$ OUT$|_{TURNO}$, // fine ciclo, output del valore calcolato

reset RDY1, set ACK1, // mediante controllo residuo

set RDY$|_{TURNO}$, reset ACK$|_{TURNO}$, 0

Abbiamo sempre 6 variabili di condizionamento, 1 bit di stato, 9 frasi e quindi la PC si può realizzare con tempo di stabilizzazione di $T_{\sigma PC}$ e $T_{\omega PC}$ entrambi pari a $2t_p$. Le frasi con la somma $T_{\sigma PO}$ e $T_{\omega PO}$ più consistente sono quelle della seconda microistruzione. Qui il $T_{\omega PO}$ é $14t_p$, come calcolato precedentemente per il segno(M[...]-...). Il $T_{\sigma PO}$ é al massimo $t_k + t_{alu}$ (I+1 $\rightarrow$ I), ovvero $7t_p$. Tutto questo porta ad un τ pari a $14t_p+2t_p+7t_p+t_p = 24t_p$. Il tempo è maggiore di quello ottenuto col codice precedente perchè in questo caso il tempo di calcolo del segno (...) non può essere sovrapposto al tempo del calcolo dell'incremento dei contatori.