

Architettura degli Elaboratori

Secondo appello sessione invernale: 8 febbraio 2016

Riportare su tutti i fogli consegnati nome, cognome, matricola e corso di appartenenza. I risultati saranno pubblicati via web appena disponibili, insieme al calendario degli orali

Esercizio n. 1

Si consideri il seguente frammento di pseudo codice

```
for {i=1, i<1024, i++}
{
  a[i]=b[i]*b[i]+c[1024-i]+c2;
  c[i]=c[i-1]+1;
  c2=c2+2;
  b[i]=c[i]+c2
}
```

Si assuma che tutte le variabili siano di tipo intero e che tutti i vettori contengano N elementi ($N > 1024$). Il risultato della compilazione è eseguito su un processore pipeline con salto ritardato ed EU parallela pipeline che esegue le operazioni brevi in t e quelle lunghe in $4t$. Le pagine della cache contengono ciascuno 128 parole ($\sigma=128$). Si chiede di:

- scrivere il codice prodotto dalla compilazione con regole standard,
- calcolare il numero di fault generati dalla esecuzione del frammento assumendo che la dimensione della cache permetta di mantenere in memoria 2 pagine per istruzioni e 4 pagine per i dati,
- ottimizzare il codice e calcolare il guadagno di prestazioni ottenuto,
- calcolare il tempo di esecuzione del codice prodotto dalla compilazione con regole standard supponendo che oltre alle caratteristiche precedenti l'architettura sia multithreading con interleaving di grado 2 ed il secondo thread esegua le stesse istruzioni su dati diversi.

Esercizio n. 2

Una unità U interfaccia una unità M e otto unità D_1, \dots, D_8 . M invia dati da 32 bit che possono essere ricevuti da una qualsiasi delle unità D_i . Ogni D_i ($i:1..8$), richiede un dato a U e poi ne attende la comunicazione. U ha una memoria interna di 64 posizioni per mantenere i messaggi ricevuti da M e non ancora trasmessi. Eventuali messaggi ricevuti e non trasmessi vengono gestiti secondo una disciplina FIFO. Si chiede di progettare PC e PO di U, calcolare il ciclo di clock τ (funzione di t_a , tempo di accesso alla memoria interna di U, e t_p , tempo di stabilizzazione di una porta logica con al più 8 ingressi) ed il tempo necessario per ricevere un dato da M.

Esercizio 1

Secondo le regole di compilazione standard lo pseudo codice produrrà il seguente codice assembler:

```

ADD R0, #1, Ri           ; inizializzazioni
ADD R0, #1024, Rk        ; serve come costante per la 1024-i
loop: LOAD Rb, Ri, Rbi    ; primo statement: prima parte calcolo di b[i]*b[i]
    MUL Rbi, Rbi, Rbiq
    SUB Rk, Ri, Rki       ; seconda parte calcolo di c[1024-i]+1
    LOAD Rc, Rki, Rcki
    ADD Rcki, Rc2, Rai
    ADD Rai, Rbiq, Rai
    STORE Ra, Ri, Rai
    SUB Ri, #1, Rii       ; secondo statement calcola c[i-1]+1
    LOAD Rc, Rii, Rcii
    INC Rcii
    STORE Rc, Ri, Rcii
    ADD Rc2, #2, Rc2      ; terzo statement: calcola c2=c2+2
    ADD Rcii, Rc2, Rbi    ; quarto statement: calcola c[i]+c2 (c[i] già in un reg. temp.)
    STORE Rb, Ri, Rbi
    INC Ri
    IF< Ri, Rk, loop
  
```

Vi sono diverse dipendenze logiche:

EU-EU: MUL sulla seconda ADD

EU-IU: SUB su LOAD, seconda ADD sulla STORE, SUB su LOAD, INC su STORE, ADD su STORE e INC su IF

Simulazione:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26		
IM	LOAD	MUL	SUB	LOAD	LOAD	ADD	ADD	STORE	ADD	STORE	STORE	SUB	LOAD	INC	STORE	STORE	STORE	ADD	ADD	STORE	ADD	INC	IF<	IF<	IF<	XXX	LOAD	LOAD	
IU		LOAD	MUL	SUB	LOAD	LOAD	ADD	ADD	STORE	STORE	STORE	SUB	STORE	LOAD	LOAD	INC	STORE	STORE	STORE	ADD	ADD	STORE	STORE	INC	IF<	IF<	IF<	LOAD	LOAD
DM			LOAD																										
EU m			LOAD	MUL	SUB				LOAD	ADD	ADD			SUB			LOAD	INC			ADD	ADD			INC				
EU *					MUL	MUL	MUL	MUL																					

Per 16 istruzioni impieghiamo 26 t, quindi

$$T = 26t/16 = 13t/8$$

Con efficienza di poco più del 60% (8/13)

Il tempo di completamento in questo caso sarebbe

$$T_{c1} \cong 1023 * 13t/8$$

Il working set del programma contiene

- tutto il codice
- una pagina per a (solo scrittura)
- una pagina per b (lettura e scrittura)
- due pagine per c (alla posizione i e alla posizione 1024-i)

Dunque le 2 pagine per il codice e 4 per i dati sono sufficienti a mantenere il working set (assumiamo che la costante c2 sia compilata in un registro e dunque non richieda accessi in memoria per il caricamento).

Il numero totale dei fault sarà:

- $1024 / \sigma = 8$, per il vettore b (non contiamo fault per a, visto che si scrive completamente senza leggerlo)
- $1024 / \sigma + 1024 / \sigma = 16$ per il vettore c (serve una pagina in alto e una in basso e ne possiamo mantenere solo 2 in cache)
- 1 per il codice

Ovvero #fault = (1+8+16) = 25. Quindi al T_{c1} di prima dovremmo aggiungere $25 T_{transf}$

Ottimizzazione del codice:

```

ADD R0, #1, Ri           ; inizializzazioni
ADD R0, #1024, Rk        ; serve come costante per la 1024-i
loop: LOAD Rb, Ri, Rbi    ; primo statement: prima parte calcolo di b[i]*b[i]
      MUL Rbi, Rbi, Rbiq
      SUB Rk, Ri, Rki
      SUB Ri, #1, Rii
      LOAD Rc, Rki, Rcki
      ADD Rcki, Rc2, Rai
      ADD Rai, Rbiq, Rai
      LOAD Rc, Rii, Rcii
      INC Rcii
      ADD Rc2, #2, Rc2    ; terzo statement: calcola c2=c2+2
      ADD Rcii, Rc2, Rbi  ; quarto statement: calcola c[i]+c2 (c[i] già in un reg. temp.)
      INC Ri
      STORE Ra', Ri, Rai
      STORE Rc', Ri, Rcii
      IF< Ri, Rk, loop, delayed
      STORE Rb', Ri, Rbi
  
```

Abbiamo spostato prima possibile le operazioni di calcolo con le costanti, posticipato le store (di cui una nel delay slot). In questo caso tutte le STORE devono utilizzare registri base decrementati per tener conto del valore del registro Ri già incrementato.

Simulazione:

	0	5	10	15	16	17	18	19												
IM	L	M	S	S	L	A	A	L	I	A	A	I	ST	ST	IF	ST	L			
IU		L	M	S	S	L	A	A	L	I	A	A	I	ST	ST	ST	ST	IF	ST	L
DM		L					L								ST	ST		ST	L	
EU₁			L	M	S	S		L	A	A	L	I	A	A	I					L
EU₂				M	M	M	M													

In questo caso otteniamo $T = 19t/16$ (efficienza circa al 85%).

Nel caso eseguiamo il codice non ottimizzato su architettura multithreading interleaved a due vie (come nelle ipotesi dell'esercizio) le bolle da 1 sparirebbero completamente e quelle da 2

diventerebbero da 1. Dunque avremmo $T = 40t / 32$ (ovvero $T = 20t / 16$, riferito al singolo thread) con un'efficienza dell'80%.

Esercizio 2

Le interazioni fra unità vengono gestite con le usuali coppie di indicatori a transizione di livello. Una richiesta a trasmettere di M viene vista come un RDYm da U. Una volta trattata la richiesta, U invia a M un ACKm. Una richiesta a ricevere da Di viene vista da U come un RDYi. Una volta trattata si risponde a Di con il messaggio e un ACKi.

La memoria è gestita come un buffer circolare. Un registro N (da 7 bit) indica quanti elementi vi sono nel buffer. Un registro INS indica la posizione dove si inserisce, e inizialmente vale 0. Una volta effettuata l'inserzione, si incrementa. Un registro PRE indica la posizione da cui si preleva, ed è pure inizializzato a 0. Una volta effettuato il prelievo, si incrementa PRE modulo 64. Una volta inserito, si incrementa INS modulo 64. $N=0$ ($OR(N)=0$) indica buffer vuoto (non si possono soddisfare richieste da Di). $N=64$ ($N_0=1$) indica buffer pieno (non si possono soddisfare richieste da M).

Possiamo decidere una politica per gestire conflitti nelle richieste dalle unità Di. La più semplice è una politica che fa coincidere la priorità con il numero della unità: richieste di Dj hanno priorità su quelle di Dk se $j < k$.

Situazioni possibili:

- Invio da M, buffer vuoto e nessuna richiesta dalle Di : inserzione del messaggio nel buffer
- Invio da M, buffer vuoto e una richieste pendenti da una o più Di : spedizione diretta del messaggio
- Invio da M, buffer non pieno: inserzione del messaggio nel buffer
- Invio da M, buffer pieno: NOP
- Richiesta da una o più Di e buffer non vuoto: arbitraggio e invio alla unità vincente del messaggio prelevato dal buffer
- Richiesta da una o più Di, buffer vuoto e richiesta di invio pendente: arbitraggio e invio diretto del messaggio in spedizione
- Richiesta da una o più Di, buffer vuoto: NOP

Variabili di condizionamento:

- $OR(N)$: 0 significa buffer vuoto
- $OR(RDYi)$: 1 significa richiesta di prelievo messaggio pendente
- $RDYm$: 1 significa richiesta di invio messaggio pendente

La funzione f_i è la funzione di arbitraggio delle richieste. Ha come input i RDYi e come output 3 bit che rappresentano la codifica dell'indice i dell'unità che ha vinto l'arbitraggio.

Può essere implementata utilizzando una tabella di verità tipo:

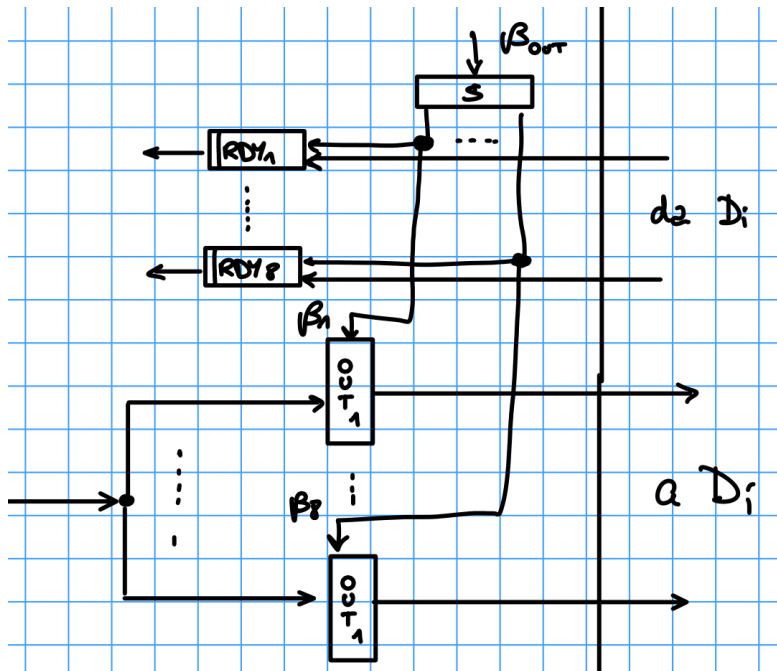
RDY1	RDY2	RDY3	RDY4	RDY5	RDY6	RDY7	RDY8	z0	z1	z2
1	-	-	-	-	-	-	-	0	0	0
0	1	-	-	-	-	-	-	0	0	1
0	0	1	-	-	-	-	-	0	1	0
0	0	0	1	-	-	-	-	0	1	1
0	0	0	0	1	-	-	-	1	0	0
0	0	0	0	0	1	-	-	1	0	1
0	0	0	0	0	0	1	-	1	1	0
0	0	0	0	0	0	0	1	1	1	1

che implementa la precedenza alle unità con indice più basso. La funzione si può calcolare chiaramente in $2 t_p$ (livello AND max 8 ingressi, livello OR max 4 uni per colonna).

Viene utilizzata per il controllo residuo necessario a (i indice dell'unità che ha vinto l'arbitraggio):

- Settare l'indicatore di uscita verso l'unità destinazione D_i
- Resetare l'indicatore di ingresso dall'unità D_i
- Scrivere nel registro OUT_i relativo all'unità D_i

Secondo uno schema tipo (gli ACK sono omessi per semplicità):



Microcodice

0. $(RDY_m, OR(RDY_i), OR(N), N_0=00_)$ NOP, 0 // nessuna richiesta non si fa nulla
// priorità alle estrazioni
// estrazione da coda con almeno un elemento
 $(=11-)$ $M[PRE] \rightarrow OUT|_{fi}$, $PRE+1\%64 \rightarrow PRE$, $N-1 \rightarrow N$, set $ACK|_{fi}$, reset $RDY|_{fi}$, 0
 $(=101)$ $M[PRE] \rightarrow OUT|_{fi}$, $PRE+1\%64 \rightarrow PRE$, $N-1 \rightarrow N$, set $ACK|_{fi}$, reset $RDY|_{fi}$, 0
// estrazione da buffer vuoto ma con richiesta di inserzione pendente
 $(=1100)$ $INM \rightarrow OUT|_{fi}$, set $ACK|_{fi}$, reset $RDY|_{fi}$, reset RDY_m , set ACK_m , 0
// se non c'è niente da estrarre, passare allo stato con priorità alle inserzioni
 $(=0100)$ NOP, 1
// inserzioni
// inserzione in buffer non pieno in assenza di richiesta di estrazione (trattate prima)
 $(=10-0)$ $INM \rightarrow M[INS]$, $INS+1\%64 \rightarrow INS$, $N+1 \rightarrow N$, reset RDY_m , set ACK_m , 0
// inserzione in buffer pieno, si passa alla priorità alle estrazioni
 $(=10-1)$ NOP, 1
1. // come la 0 ma con priorità alle inserzioni
 $(RDY_m, OR(RDY_i), OR(N), N_0=00_)$ NOP, 0 // nessuna richiesta non si fa nulla
 $(=10-0)$ $INM \rightarrow M[INS]$, $INS+1\%64 \rightarrow INS$, $N+1 \rightarrow N$, reset RDY_m , set ACK_m , 1
 $(=110-)$ $INM \rightarrow OUT|_{fi}$, set $ACK|_{fi}$, reset $RDY|_{fi}$, reset RDY_m , set ACK_m , 1
 $(=100-)$ nop, 0
 $(=011-)$ $M[PRE] \rightarrow OUT|_{fi}$, $PRE+1\%64 \rightarrow PRE$, $N-1 \rightarrow N$, set $ACK|_{fi}$, reset $RDY|_{fi}$, 1
 $(=010-)$ nop, 0

Valutazione dei tempi:

- $T_{\omega PO} = 1t_p$ (OR di 8 valori max, OR(N) considera l'OR dei soli primi 8 bit, il bit N_0 è testato separatamente)
- $T_{\sigma PO} = t_k + t_a + t_{sel} = 2t_p + t_a + 2t_p = t_a + 4t_p$
(operazione più lunga è la $M[PRE] \rightarrow OUT|_{fi}$ che richiede un tempi di accesso in memoria ($t_k + t_a$) e un t_{sel} per il selettore di scrittura comandato da fi , t_{fi} non si conta perchè stabilizza in parallelo all'accesso in memoria)
- $T_{\omega PC} = 2t_p$
(ingressi : 1 bit per lo stato, 4 bit per var di cond. Quindi 5 bit in totale, un livello di porte AND; uscite : 13 frasi, quindi max 13 uno, ma se sono più di 8 codifico gli zeri e nego il risultato, dunque 1 livello OR)
- $T_{\sigma PC} = 2t_p$ (come per $T_{\omega PC}$)

$$\tau = T_{\omega PO} + \max \{ T_{\sigma PC}, T_{\omega PC} + T_{\sigma PO} \} + \delta = t_p + 2t_p + t_a + 4t_p + t_p = t_a + 8t_p$$

Per ricevere un dato da M, U utilizza un singolo ciclo di clock, in caso vi sia posto nel buffer. Qualora non vi sia posto nel buffer, il tempo dipende dai tempi di presentazione delle richieste di ricezione da parte delle unità Di.