Architettura degli elaboratori—A. A. 2016-2017—2° Appello—7 febbraio 2017

Riportare nome, cognome, numero di matricola e corso (A/B) in alto a destra su tutti i fogli consegnati.
I risultati e il calendario degli orali saranno pubblicati su WEB appena disponibili.

Domanda 1

Si consideri un sistema costituito da 8 unità firmware (U_1, ... ,U_8) collegate in pipeline: l'unità U_i riceve dati da U_{i-1} e invia risultati alla unità U_{i+1}. L' unità U_1 riceve dati da una unità U_a mentre la U_8 invia i propri risultati ad una unità U_b. Ciascuna U_i ha al suo interno una memoria M con 2^{13} posizioni da 32 bit. Ogni interfaccia fra due unità permette l'invio di una terna di valori interi X, I e OP, rispettivamente da 32, 16 e 3 bit. I 16 bit di I vengono interpretati come 3 bit di indirizzo unità (parte più significativa) e 13 bit di indirizzo da utilizzare per accedere alla memoria interna M. Ciascuna delle unità U_j esegue una delle seguenti operazioni esterne:

- se OP = 00 e i bit di indirizzo unità di I valgono j, il valore X in ingresso viene memorizzato alla posizione di M determinata dai 13 di indirizzo di I e il vecchio valore di tale posizione viene trasmesso alla unità U_{j+1}, utilizzando come valore I lo stesso ricevuto in ingresso e come valore OP il valore 10; se i bit di indirizzo unità di I non valgono j, I, X e OP vengono passati all'unità successiva
- se OP = 01 e i bit di indirizzo unità di I valgono j, all'unità U_{j+1} viene inviato il valore in memoria M alla posizione determinata dai 13 bit di indirizzo di I, utilizzando lo stesso valore I quello ricevuto in ingresso e come valore OP il valore 10; se i bit di indirizzo unità di I non valgono j, I, X e OP vengono passati all'unità successiva
- se OP =10 l'unità passa all'unità successiva senza modificarli i valori appena ricevuti per X, I e OP

L'unità U_a invia ad U_1 solo richieste per operazioni OP=00 o 01. Si vuole conoscere la latenza, in t_p, richiesta dall'esecuzione di una operazione ordinata da U_a considerando come esecutore dell'operazione l'intero pipeline formato dalle 8 unità.

Domanda 2

Si consideri un sistema costituito dal processore D-RISC sequenziale (quello dell'interprete firmware) e dotato di una gerarchia di memoria che comprende un'unico livello di cache, comune per dati e istruzioni, set associativa su insiemi (4 linee per insieme), con σ = 16 e 1K insiemi. La memoria principale è modulare, interallacciata, con 4 moduli da 1G parole ciascuno e τ_M=400τ (con τ ciclo di clock della CPU). Il t_{tr} fra memoria principale e cache è pari a 4τ.

Si consideri quindi il codice D-RISC che deriva dalla compilazione dello pseudo codice

```
int N = 1024, a[N], b[N], c[N];
for(int i=0; i<N; i++) {
       if(a[i] == b[i]) c[i] = a[i]*4;
       else             c[i] = a[b[i]%N]*2;
}
```

e si calcolino:

- il numero di fault generati dal programma
- la traccia degli indirizzi generati dal programma e quelli utilizzati per l'accesso alla cache durante la prima iterazione del for, indicando per ciascun indirizzo il numero di insieme nella cache
- il tempo di completamento della prima iterazione, assumendo che A[0] sia diverso da B[0]

assumendo che:

- i vettori A, B e C e il codice siano allocati rispettivamente dal compilatore agli indirizzi 1K, 2K, 3K e 0
- che le pagine in memoria centrale siano di 2K parole
- che tabril[] = {<12,1>,<6,1>,<24,1>,<1,1>,...} dove ogni coppia è data da <IPF, bit di presenza>

Traccia di soluzione

Domanda 1

Traccia

Va progettata la singola unità Ui. Dovrebbe essere possibile realizzarla con una unica micro istruzione o al massimo con 2 microistruzioni. Va valutato il ciclo di clock, considerando anche il tempo di accesso alla memoria, in funzione di tp. Per la comunicazione servono 2 cicli di clock. Dunque la latenza non potrà essere inferiore a 16 τ, qualunque sia il valore di τ.

Microcodice

μ-codice della generica unità U_i (indichiamo con i_1,i_2,i_3 la configurazione di bit corrispondente a *i*)

0. *// attendo una richiesta di operazione e la disponibilità a ricevere da parte della unità succ* (RDYin,ACKout,OP, I_{15},I_{14},I_{13}=0------,10-----) nop, 0
   // richiesta OP=00, bit di indirizzo indirizzano questa unità
   (=1100i_1,i_2,i_3)
   // memorizza X e prepara i dati in uscita
   X→M[and(I,0001..1)] , M[and(I,0001..1)] →XOUT, I→IOUT, 01→OPOUT,
   // sistema sincronizzatori
   set ACKin, reset RDYin, set RDYout, reset ACKout,0
   // richiesta OP=01, bit di indirizzo indirizzano questa unità
   (=1101i_1,i_2,i_3)
   // prepara dati in uscita
   M[and(I,0001..1)] →XOUT, I→IOUT,01→OPOUT,
   // sistema sincronizzatori
   set ACKin, reset RDYin, set RDYout, reset ACKout,0
   // se OP=10 o è una delle prime due operazioni ma indirizzate ad un'altra unità
   (=1110---,110----)
   // passa dati in ingresso sull'uscita
   X→XOUT, I→IOUT,OP→OPOUT,set ACKin, reset RDYin, set RDYout, reset ACKout,0

In questo caso si sono testati i bit più alti di I direttamente nelle variabili di condizionamento. In alternativa, si poteva utilizzare controllo residuo per scegliere (nel caso OP=00 o OP=01) se scrivere in uscita i valori nuovi o quelli in ingresso e per comandare il β della memoria.

0. (RDYin, ACKout, OP= 00--) nop,0
   // OP=00
   (=1100) (X→M[and(I,0001..1)] , M[and(I,0001..1)] →XOUT, I→IOUT, 01→OPOUT)| (I_{15},I_{14},I_{13}==i_1,i_2,i_3), (X→XOUT, I→IOUT,OP→OPOUT)| (I_{15},I_{14},I_{13}==i_1,i_2,i_3), set ACKin, reset RDYin, set RDYout, reset ACKout,0
   // OP=01
   (=1101) (M[and(I,0001..1)] →XOUT, I→IOUT, 01→OPOUT)| (I_{15},I_{14},I_{13}==i_1,i_2,i_3), (X→XOUT, I→IOUT,OP→OPOUT)| (I_{15},I_{14},I_{13}==i_1,i_2,i_3), set ACKin, reset RDYin, set RDYout, reset ACKout,0
   // OP=10
   (=1110) X→XOUT, I→IOUT,OP→OPOUT,set ACKin, reset RDYin, set RDYout, reset ACKout,0

In entrambi i casi, abbiamo una sola μ-istruzione. L'unità si riduce ad una unica rete sequenziale. Le variabili di condizionamento sono meno di 8 e quindi le variabili di controllo sono calcolate con un ritardo di $2t_p$ (max 8 ingressi, 4 frasi). La m-operazione più lunga (consideriamo il primo μ-codice) è la *M[and(I,0001..1)] →XOUT* che richiede un t_p per calcolare l'and, un t_a per l'accesso in memoria e un t_k per la scrittura di XOUT. Il t_a in questo caso vale $\lceil \log_8(14) \rceil + \lceil \log_8(2^{13}) \rceil t_p = (2 + 5)\, t_p = 7\, t_p$. Quindi complessivamente servono $10t_p$.

Il ciclo di clock τ può dunque essere calcolato come

$$\tau = T_{\omega PO} + T_{\omega PC} + T_{\sigma PO} + \delta = 0 + 2t_p + 10t_p + t_p = 13\, t_p$$

Il tempo di completamento di ognuna delle operazioni esterne è pari ad un τ. Tuttavia, l'implementazione del protocollo di comunicazione richiede almeno 2τ. La latenza (dal momento in cui arriva una richiesta di operazione ad U_1 al momento in cui U_8 è pronta a spedire il risultato sarà di 8 (numero unità) volte $2\tau = 16\tau = 16 * 13\, t_p$

Domanda 2

Traccia

Compilazione del codice -> standard. C'è da ricordarsi che le moltiplicazione e l'operazione di modulo sono per potenze di due dunque vanno fatte con gli shift. L'accesso ai vettori è sequenziale, senza riuso per B e C, random e sequenziale per A. Dunque utilizzando il flag "prefetch" per B e C abbiamo il solo fault iniziale, mentre per A dovremmo prevedere comunque N/σ faults, visto l'accesso con indirezione su B. Il codice, dopo il fault iniziale non ne produrrà altri, utilizzando il prefetch. Il tempo di completamento deve tener conto di #fault * Ttransf. Quest'ultimo sarà dato dalla solita formula $2(\tau + T_{tr}) + \sigma\, t_M /m + \tau$. Gli indirizzi generati saranno quelli tradotti dalla tabril, tenendo conto che entra tutto in due pagine, dunque serve solo utilizzare la pagina IPF = 12 e IPF = 6.

Codice

La compilazione con le regole standard produce il codice:

```
        CLEAR Ri
loop:   LOAD Rba, Ri, Rai
        LOAD Rbb, Ri, Rbi
        IF= Rai, Rbi, then
else:   AND Rbi, #1023, Rind
        LOAD Rb1, Rind, Rai
        SHL Rai, #1, Rai
        STORE Rbc, Ri, Rai
        GOTO cont
then:   SHL Rai, #2, Rai
        STORE Rbc, Ri, Rai
cont:   INC Ri
        IF< Ri, RN, loop
        END
```

Numero di fault

Il programma scorre i vettori A e B, per il calcolo della condizione, e C, per la memorizzazione del risultato. Per i tre vettori avremo dunque N/σ fault, riducibili a 1 in caso di utilizzo del flag "prefetch" sulle relative LOAD. Tuttavia, A viene indirizzato anche mediante i valori di B. Questi accessi non seguono un pattern sequenziale. Se tuttavia si

verificasse un fault per un accesso ad A non ancora toccato dagli accessi sequenziali per il test della condizione, il fault non verrebbe ripetuto quando si arriva a tale linea di cache per effetto degli accessi seuquenziali. Questo perchè la capacità complessiva della cache (1K insiemi da 4 linee ciascuna) è sufficiente a contenere tutto il vettore A, oltre alla singola linea di B e C nel working set ed al codice. Dunque per A non vale la riduzione ad unico fault mediante prefecth.

Traccia degli indirizzi

La traccia degli indirizzi logici generati dal programma per la prima iterazione è la seguente:

0, 1, 1024, 2, 2048, 3,

che corrispondono, scomposti nei campi <IPL, OFF> ai seguenti indirizzi

<0,0>, <0,1>, <0,1024>, <0,2>, <1,0>, <0,3>

e prosegue, in caso il test IF= abbia successo con

9, 10, 3072, 11, 12

ovvero

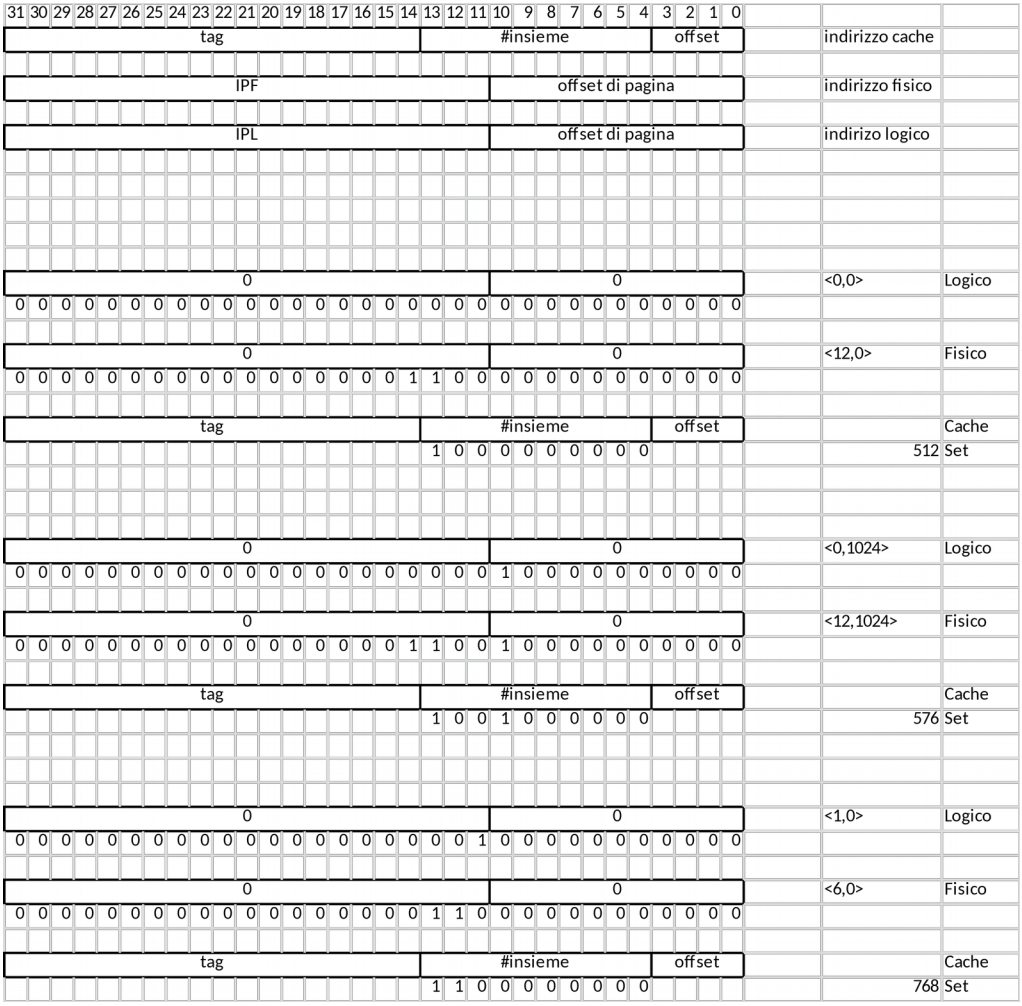
<0,9>, <0,10>, <1, 1024>, <0,11>, <0,12>

oppure, in caso il test IF= non abbia successo con

4, 5, X (fra 1024 e 2047 compresi), 6, 7, 3072, 8, 11, 12

<0,4>, <0,5>, <0,X>, <0,6>, <0,7>, <1,1024>, <0,8>, <0,11>, <0,12>

Gli indirizzi logici corrispondono ad indirizzi fisici ricavati mediante la tabella di rilocazione. Le pagine sono da 2K parole, quindi tutti gli indirizzi fra 0 e 2047 (codice e vettore A) cadono nella prima pagina logica e quindi sono mappati nella pagina fisica con IPF = 12 (come da tabril), mentre quelli fra 2048 e 4095 (vettori B e C) cadono nella seconda pagina logica e vengono mappati nella pagina fisica con IPF = 6.



e dunque la traccia degli indirizzi fisici generati verso la cache dalla MMU sarà interpretata come (considerando solo la parte iniziale per semplicità):

set=512, offset=0, tag=1
set=512, offset=1, tag=1
set=576, offset=0, tag=1
…

dal momento che gli accessi in cache considerano i 32 bit dell'indirizzo fisico come (dai bit meno significativi a quelli più significativi):

- 4 bit di offset
- 10 bit di numero di blocco
- 18 bit di tag

Tempo di completamento prima iterazione

La prima iterazione esegue (nell'ipotesi che venga preso il ramo else) 3 LOAD, 1 STORE, 2 IF, 1 GOTO e 3 aritmetico logiche corte. Complessivamente fanno $37\tau+14t_a$:

		Texec		CH0, CH1		Totale	
	#	τ	t_a	τ	t_a	τ	t_a
LOAD	3	2	1	2	1	12	6
STORE	1	3	1	2	1	5	2
SALTI COND	2	2	0	2	1	8	2
SALTI INCOND	1	1	0	2	1	3	1
ARIT LOG CORTE	3	1	0	2	1	9	3
				Tot		37	14

A questo vanno aggiunti i tempi necessari per risolvere il fault relativo al codice, quello relativo ad A[0] quello relativo a B[0], quello relativo alla A[B[i]%N] (probabilmente non risolto con la stessa linea che contiene A[0]) e infine (in sola scrittura), il costo dell'allocazione della linea di cache per C[0] (trascurabile rispetto agli altri), che in totale costano

$4 * (2(\tau+ T_{tr}) + (\sigma \tau_M)/m + \tau) = 3*(2(\tau+ 4 \tau) + (16 * 400 \tau)/4 + \tau) = 3 (10 \tau + 1600 \tau + \tau) = 4 * 1611 \tau = 6444 \tau$

e quindi il tempo di completamento della prima iterazione sarà

$37\tau+14t_a + 6444 \tau$

Considerando che t_a, il tempo di accesso in cache di primo livello, sarà di 2 o 3 τ a seconda del modo scelto per implementare la cache set associativa (1 o 2 τ + 1 τ per la MMU), il tempo complessivo speso per la prima iterazione può essere calcolato come (cache set associativa con accesso in 1 τ) 6444 τ + 14 (2 τ) = 6472 τ