Architettura degli elaboratori Corsi A e B

Appello straordinario per studenti lavoratori e fuori corso, 4 novembre 2014

Indicare su ognuno dei fogli consegnati nome, cognome, matricola e corso (A o B).
Risultati e calendario degli orali saranno pubblicati su didawiki.cli.di.unipi.it, pagina del corso di AE, non appena disponibili.

Domanda 1 Si progetti una unità firmware U, interfacciata con un sottosistema di memoria M e con due altre unità U_a e U_b. U riceve da U_a una coppia di valori A_ID e N, rispettivamente da 10 bit e da 8 bit. U:

- interpreta A_ID come identificatore di un array in M di N parole,
- lo traduce in indirizzo logico INDL utilizzando una apposita tabella di corrispondenza contenuta nella stessa memoria M a partire dall'indirizzo specificato in un registro INDTAB. Gli array sono tutti memorizzati in modo da cominciare ad un indirizzo multiplo di 256. L'unità U contiene al suo interno una piccola memoria (CACHE) utilizzata per mantenere le associazioni fra identificatori di array e indirizzi utilizzate più di frequente.
- invia ad U_b il checksum CHK (somma modulo 2^{32}) dell'array identificato da A_ID e lungo N posizioni.

L'interfaccia verso il sottosistema di memoria è standard e il sottosistema di memoria risponde a U con una latenza τ_M. CACHE è una memoria di solo 8 parole gestita con il metodo dell'indirizzamento diretto. Ogni entry memorizzata in CACHE contiene quanto necessario a tradurre un singolo identificatore di array, incluso la parte più significativa (24 bit) dell'indirizzo INDL. Qualora CACHE non contenga le informazioni necessarie alla traduzione dell'identificatore A_ID, U accede alla tabella di corrispondenza in M e usa i bit meno significativi del valore letto (una singola parola da 32 bit) per il campo INDL della nuova entry di CACHE.

Dell'unità U si fornisca il tempo medio di elaborazione in funzione di N, τ_M e t_p assumendo $5t_p$ come tempo di stabilizzazione della ALU per operazioni intere di somma e sottrazione a 32 bit.

Domanda 2 Si consideri il frammento di codice:

```
int A[N], B[N], C[N], D[N];

for(i=0;i<N;++) {
        E[i]=(A[i]+B[i])*D[C[i]];
}
```

e se ne valuti il tempo si completamento su una architettura pipeline con unità esecutiva parallela dotata di EU slave per la moltiplicazione pipeline a 4 stadi e gerarchia di memoria con unico livello di cache, con cache dati implementata mediante una cache set associativa a due vie. Il vettore C contiene tutti valori compresi nell'intervallo [0, N-1].

Per tutti gli elementi non specificati nel testo dell'esercizio si facciano *esplicitamente* e *concisamente* tutte le assunzioni del caso.

Traccia di soluzione

Domanda 1

La memoria CACHE conterrà entry nel formato TAG (7 bit) IND (24 bit). Si può considerare la presenza di un bit P (bit di presenza) che indica se l'entry è significativa o vuota (inizialmente saranno tutte vuote). Considerando A_ID diviso in due parole da 3 bit (nella parte meno significativa, campo BC) e 7 bit (nella parte più significativa, campo TAG), l'accesso alla cache sarà implementato confrontando il campo TAG dell'indirizzo con il campo TAG di CACHE[A_ID.BC] e controllando che sia CACHE[A_ID.CB].P=1.

Dunque il microprogramma dell'unità potrebbe essere il seguente:

0. (RDYin, CACHE[A_ID.BC].P, zero(CACHE[A_ID.BC].TAG - A_ID.TAG)=0--) nop, 0,
   (=111) CACHE[A_ID.BC].INDL..(N-1) → IND, "read"→ OP, set RDYMout, N-1→C, 0→CHK,1,
   (=1--) INDTAB+A_ID→IND, "read"→OP, set RDYMout,2
1. (RDYMin, ESITO, zero(C), ACKout=0---) nop, 1
   (=100-) CHK + DATAIN → CHK, C-1 → C, CACHE[A_ID.BC].INDL..(C-1) →IND, "read"→OP, reset RDYMin, set RDYMout, 1,
   (=11--) *trattamento errore,*
   (=1011) CHK+DATAIN→CHKOUT, reset RDYMin, reset ACKout, set RDYout, set ACKin, reset RDYin, 0
2. (RDYMin, ESITO=0-) nop, 2,
   (=10) A_ID.TAG .. 1 .. DATAIN.INDL → CACHE[A_ID.BC], N-1→C, 0→CHK, A_ID.TAG .. 1 .. DATAIN.INDL → IND, "read"→ OP, set RDYMout, 1,
   (=11) *trattamento errore*

La parte controllo ha 7 ingressi (RDYin, CACHE[].P, zero(CACHE[].TAG-A_ID.TAG), RDYMin, ZERO(C), ESITO, ACKout) e due bit per la rappresentazione dello stato interno. Per ogni frase vengono testati al più 4 variabili di condizionamento, quindi per ω_{PC} e σ_{PC} il livello di porte AND è uno solo. Le frasi sono 10, dunque c'è anche un solo livello di porte OR (se ci fossero più di 8 "1" per uno degli α e β, potremmo esprimere gli zeri negando il risultato). Dunque $T\omega_{PC} = T\sigma_{PC} = 2t_p$.

$T\omega_{PO}$ è il risultato di un'ALU che ha la memoria come ingresso. La memoria è sempre acceduta mediante lo stesso indirizzo, dunque non abbiamo commutatori sull'ingresso degli indirizzi e la condizione di correttezza è verificata, a patto di utilizzare una ALU dedicata per il calcolo di *segno(…)*.

x0	x1	x2	x3	x4	x5	x6	x7	$\alpha 0$	$\alpha 1$	$\alpha 2$	z
1	-	-	-	-	-	-	-	0	0	0	1
-	1	-	-	-	-	-	-	0	0	1	1
-	-	1	-	-	-	-	-	0	1	0	1
-	-	-	1	-	-	-	-	0	1	1	1
-	-	-	-	1	-	-	-	1	0	0	1
-	-	-	-	-	1	-	-	1	0	1	1
-	-	-	-	-	-	1	-	1	1	0	1
-	-	-	-	-	-	-	1	1	1	1	1

Tabella di verità del commutatore per la lettura di CACHE

La memoria da 8 posizioni avrà un commutatore che richiede 2 t_p (1 t_p per il livello AND, prodotto di 4 termini, e 1 t_p per il livello or (somma di otto termini), quindi $t_{acache}=2t_p$ e, considerando i 5 t_p della ALU, avremo $T\omega_{PO} = 7t_p$.

Il $T\sigma_{PO}$ è dominato dalla operazione che vede il risultato della ALU scritto in un registro con un commutatore al proprio ingresso, dunque $T\sigma_{PO} = 7\ t_p$.

Il ciclo di clock τ può essere dunque stimato con $\tau = T\omega_{PO} + \max\{ T\sigma_{PC} , T\omega_{PC} + T\sigma_{PO} \} + \delta = 7\ t_p + \max\{2t_p, 2t_p + 7t_p\} + 1\ t_p = 17\ t_p$.

Per completare l'operazione esterna sono necessari $T = \tau + N(\tau + \tau_M)$ se la cache contiene la traduzione dell'indirizzo. Se non la contiene, occorre un ulteriore $(\tau + \tau_M)$.

Domanda 2

Compilazione del codice (standard):

1. loop: LOAD RA, Ri, R1
2. LOAD RB, Ri, R2
3. ADD R1, R2, R3
4. LOAD RC, Ri, R4
5. LOAD RD, R4, R5
6. MUL R3, R5, R6
7. STORE RE, Ri, R6
8. INC Ri
9. IF< Ri, RN, loop

Ci sono dipendenze EU-IU fra la 4 e la 5, fra la 6 e la 7 e fra la 8 e la 9. Non ci sono dipendenza EU-EU.

Dalla simulazione vediamo come occorrano 19*t* per completare la singola iterazione:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
IM	LD1	LD2	ADD	LD4	LD5	MUL			ST	INC							IF<			LD1	
IU		LD1	LD2	ADD	LD4	LD5	LD5	LD5	MUL	ST	ST	ST	ST	ST	ST	ST	INC	IF<	IF<		LD1
DM			LD1	LD2		LD4			LD5								ST				
EUm				LD1	LD2	ADD	LD4			LD5	MUL							INC			
EU*												MUL	MUL	MUL	MUL						

Per ridurre l'effetto delle dipendenze logiche possiamo anticipare la LD4 e la INC, avendo cura di effettuare la STORE con un registro base decrementato di 1, come d'abitudine. Inoltre possiamo utilizzare la tecnica del delayed branch per mitigare ulteriormente la dipendenza indotta dalla MUL sulla STORE, spostando la store nello slot del salto ritardato di fine ciclo. Otteniamo dunque il codice:

1. loop: LOAD RC, Ri, R4
2. LOAD RA, Ri, R1
3. LOAD RB, Ri, R2
4. INC Ri
5. ADD R1, R2, R3
6. LOAD RD, R4, R5
7. MUL R3, R5, R6
8. IF< Ri, RN, loop, delayed
9. STORE RE, Ri, R6

la cui simulazione

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IM	LD4	LD1	LD2	INC	ADD	LD5	MUL	IF<	ST	LD1											
IU		LD4	LD1	LD2	INC	ADD	LD5	MUL	IF<	ST	ST	ST	ST	ST	ST	LD1					
DM			LD4	LD1	LD2			LD5								ST	LD1				
EUm				LD4	LD1	LD2	INC	ADD	LD5	MUL								LD1			
EU*											MUL	MUL	MUL	MUL							

ci fa vedere che il tempo per la singola iterazione scende a 14*t*.

Il tempo di completamento ideale è dunque

$$T_{id} = 14\ N\ t$$

cui va aggiunto il tempo necessario al trattamento dei fault. In questo caso abbiamo:

- località ma non riuso per A, B, C ed E (in scrittura), e
- riuso ma non località per D

Dunque il working set è dato da un blocco di A, B, C ed E e da tutto D. Potremmo utilizzare il flag "non deallocare" sulla LOAD 5 e, a patto che la capacità della cache sia tale da poter contenere tutto D, non avremo altri fault per D che non siano quelli fisiologici. In queste ipotesi, il numero di fault sarà pertanto

$$N_{fault} = 4N/\sigma$$

assumendo che il compilatore e il sistema operativo non allochino la memoria in modo da mappare più di due degli array A, B, C ed E nello stesso insieme della cache. Se così fosse, il numero dei fault crescerebbe, potenzialmente fino a diventare pari al numero degli accessi effettuati.

Considerando una memoria interallacciata con *m* moduli con tempo di accesso t_a, il tempo di trattamento del singolo fault sarà dato dalla classica

$$T_{fault} = 2(\tau+T_{tr}) + \tau_M \sigma/m + \tau$$

e quindi per ottenere l'effettivo tempo di completamento dovremmo considerare

$$T = T_{id} + N_{fault} * T_{fault}$$