Architettura degli Elaboratori

a.a. 2013/14 – appello straordinario, 13 aprile 2015

Riportare nome, cognome, numero di matricola e corso A/B

Domanda 1

Si consideri la seguente gerarchia di memoria memoria principale – cache:

- metodo associativo su insiemi, con insiemi di 4 blocchi;
- memoria principale di capacità massima 1 Tera parole;
- cache di capacità 32 K parole;
- blocchi di 8 parole;
- architettura dell'elaboratore di tipo elementare.

a) Dare la definizione della legge di corrispondenza tra identificatori di blocco di memoria e identificatori di blocco di cache.

b) Dare un'implementazione dettagliata della Parte Operativa dell'unità cache, senza considerare quanto richiesto dalle azioni per il caricamento di blocchi in seguito a fault.

c) Si supponga che il ciclo di clock della CPU sia fissato a priori a 1 nsec, che $t_p = 33 \times 10^{-3}$ nsec , e che qualunque componente logico memoria con locazioni ampie fino a 128 bit debba essere realizzato mediante componenti aventi capacità 1K locazioni e ritardo di stabilizzazione di $5t_p$. Determinare il tempo di accesso in cache come visto dal processore.

Domanda 2

Compilare e valutare il tempo di completamento della seguente computazione:

int A[N]; int x = 0, c1 = ..., c2 = ...;

$\forall$ *i = 0 .. N–1*: *x = x + (A[i] – c1)* div *(A[i] + c2)*

eseguita su un elaboratore D-RISC con CPU pipeline scalare a singola bufferizzazione, Unità Esecutiva parallela, cache primaria su domanda di 32K (I) + 32K (D) con blocchi-C1 di 8 parole, e cache secondaria su domanda di 512K con blocchi-C2 di 128 parole.

La memoria principale è interallacciata con 8 moduli e ciclo di clock uguale a 30τ. La CPU è collegata alla memoria principale attraverso una struttura di interconnessione ad albero binario. Qualunque collegamento inter-chip ha latenza di trasmissione uguale a 4τ.

Domanda 3

Si consideri il problema dell'implementazione dell'Unità Esecutiva di una CPU pipeline.

Spiegare perché la realizzazione delle operazioni aritmetiche lunghe tipica di un elaboratore con CPU elementare e la realizzazione di una EU di latenza unitaria non sono adatte e vengono sostituite da un'opportuna realizzazione parallela.

Spiegare come è caratterizzata tale realizzazione e perché garantisce che, in assenza di dipendenze logiche, si ottenga la stessa performance di una architettura con EU di latenza unitaria.

Le spiegazioni devono contenere chiare considerazioni di prestazioni e di costo delle varie realizzazioni prese in esame.

Soluzione

Domanda 1

a) La memoria, indirizzata con 40 bit, consta di 2^{37} blocchi. La cache, indirizzata con 15 bit, consta di 2^{12} blocchi suddivisi in NS = 2^{10} insiemi contigui.

Ogni identificatore di blocco di memoria (BM, 37 bit) corrisponde ad un preciso identificatore di un insieme di blocchi di cache (SET, 10 bit) secondo una *funzione nota a priori*. Una funzione facile da implementare è

$$\text{SET} = \text{BM } \textit{mod } \text{NS}$$

All'interno dell'insieme, il blocco di memoria può essere allocato in *uno qualunque* dei quattro blocchi di cache. La corrispondenza all'interno dell'insieme è quindi *dinamica*, implementabile in modo associativo. Per ogni accesso occorre determinare, mediante una ricerca associativa, in quale blocco dell'insieme risiede il blocco identificato da BM (o se non risiede attualmente in nessuno). Per semplificare tale ricerca, invece che su BM il confronto può essere effettuato sulla quantità di 27 bit:

$$\text{TAG} = \text{BM } \textit{div } \text{NS}$$

che identifica univocamente il blocco di memoria tra tutti quelli corrispondenti all'insieme identificato da SET.

Se uno dei quattro blocchi, sia $b = 0 \ldots 3$, contiene il blocco cercato, l'identificatore del blocco di cache (BC, 12 bit) è dato dalla concatenazione di SET e b.

b) Essendo NS una potenza di 2, le quantità TAG e SET sono ottenute direttamente come i 27 bit più significativi e i 10 bit meno significativi, rispettivamente, del campo BM all'uscita del registro di interfaccia d'ingresso che contiene l'indirizzo di memoria.

La ricerca associativa è implementa usando una memoria RAM di 1K locazioni, indirizzata da SET. Ogni locazione j ($j = 0 \ldots \text{NS}-1$) contiene quattro coppie, la generica delle quali (ampia $28 \times 4 = 112 < 128$ bit) sia (P_k, TAG_k) con P_k bit di presenza e TAG_k che identifica il blocco di memoria presente nel blocco k-esimo ($k = 0 \ldots 3$) dell'insieme j.

La lettura di tale memoria, implementata da un singolo componente logico elementare, richiede un tempo di accesso uguale a $5t_p$.

La quantità b si ottiene come segue.

Si ricavano quattro bit x_0, x_1, x_2, x_3, il generico dei quali è il risultato della funzione logica:

$$x_k = P_k \wedge \overline{orbit\ (TAG \oplus TAG_k)}$$

Al più uno degli x_k vale uno. Questa funzione si stabilizza in $5t_p$.

Se tutti gli x_k valgono zero, la funzione OR negato dei quattro x_k fornisce l'indicazione di presenza di fault di blocco (FAULT): questo bit viene scritto in un registro per essere testato nella microistruzione successiva.

Se FAULT è falso, sono significativi i valori dei bit di b, ottenuti dalla funzione logica che riconosce la posizione k tale che $x_k = 1$:

$$b_0 = \overline{x_0}\, x_1\, \overline{x_2}\, \overline{x_3} + \; x_0\, \overline{x_1}\, \overline{x_2}\, \overline{x_3}$$

$$b_1 = \overline{x_0}\, \overline{x_1}\, x_2\, \overline{x_3} + \; x_0\, \overline{x_1}\, \overline{x_2}\, \overline{x_3}$$

ricavata dalla tabella di verità nella quale tutte le entrate, tranne le quattro di interesse (0001, 0010, 0100, 1000), hanno uscita non specificata. Questa funzione si stabilizza, in parallelo alla funzione FAULT, in $2t_p$.

La concatenazione di *TAG*, *b* e il displacement (3 bit) dell'indirizzo dà luogo all'indirizzo di cache. Il componente logico memoria cache MC è realizzato mediante 32 componenti elementari in parallelo. Il commutatore di uscita consta di 3 livelli di logica, per cui l'accesso a MC avviene in $8t_p$. A MC perviene anche il dato presente nell'interfaccia. Nel caso che l'operazione richiesta sia di scrittura, l'abilitazione alla scrittura è condizionata da FAULT. In caso di lettura, FAULT condiziona l'operazione *set RDY* nell'indicatore di interfaccia di uscita verso il processore. Il ritardo di un t_p si sovrappone all'accesso in MC.

c) Provando a realizzare la traduzione dell'indirizzo e l'accesso a MC in *una sola microistruzione*, dal punto *b)* si ottiene, in base ai ritardi indicati:

$$T_{\sigma-PO} = 20\, t_p$$

Il microprogramma sarebbe tale per cui:

$$T_{\omega-PO} = 0 \qquad T_{\omega-PC} = 2\, t_p$$

Il minimo ritardo di stabilizzazione dell'unità cache vale dunque $22\ t_p = 0.73\ nsec$. Anche tenendo conto dell'eventuale presenza di commutatori sugli indirizzi delle due memorie (al più, ulteriori $4t_p$ da aggiungere alla $T_{\sigma\text{-}PO}$), il ritardo per la traduzione dell'indirizzo e l'accesso a MC sarebbe di *0.86 nsec*, che risulta minore del ciclo di clock $\tau = 1\ nsec$ imposto per la CPU. Quindi, la realizzazione della traduzione dell'indirizzo e accesso a MC in un *singolo ciclo di clock* è fattibile.

Tenendo conto del ritardo introdotto dalla MMU, il tempo di accesso in cache, come visto dal processore, *con questa tecnologia* vale

$$t_c = 2\,\tau = 2\ nsec$$

Domanda 2

Il codice D-RISC non ottimizzato, con Rx e Ri inizializzati a zero a tempo di compilazione, è:

```
LOOP:  LOAD   RA, Ri, Ra
       SUB    Ra, Rc1, Ra1
       ADD    Ra, Rc2, Ra2
       DIV    Ra1, Ra2, Ra
       ADD    Rx, Ra, Rx
       INCR   Ri
       IF <   Ri, RN, LOOP
```

Codice ottimizzato per la CPU pipeline:

```
LOOP:  1. LOAD   RA, Ri, Ra
       2. INCR   Ri
       3. SUB    Ra, Rc1, Ra1
       4. ADD    Ra, Rc2, Ra2
       5. DIV    Ra1, Ra2, Ra
       6. IF <   Ri, RN, LOOP, delayed_branch
       7. ADD    Rx, Ra, Rx
```

Non ci sono degradazioni dovute a dipendenze logiche IU-EU né a salti. Il tempo di servizio per istruzione vale quindi:

$$T = t$$

Le dipendenze EU-EU, pur in presenza della latenza "lunga" in EU dell'istruzione 5, *non hanno alcun effetto sul tempo di servizio per istruzione*, in quanto, per questo programma, la CPU riesce a funzionare come un pipeline puro con tempo di servizio *t*, essendo l'unità funzionale divisore realizzata a sua volta in pipeline (più divisioni possono essere eseguite contemporaneamente).

Quindi, il tempo di completamento in assenza di fault di cache vale:

$$T_{c-0} = 7\,N\,T = 14\,N\,\tau$$

Per quanto riguarda i fault di cache, il working set è costituito dal blocco corrente di A (sola località). Siamo nelle condizioni – *fault consecutivi appartenenti allo stesso blocco-C2* – in cui la cache secondaria, pur operante su domanda, garantisce alla cache primaria il trasferimento anticipato dei blocchi-C1 richiesti. Poiché la distanza tra due fault di C1 è 14 σ τ = 112 τ > τ_M, C1 trova sempre già in C2 tutti i blocchi-C1 appartenenti allo stesso blocco-C2. Inoltre, per ogni blocco-C2, occorre aggiungere la latenza di trasferimento del primo blocco-C1 da M a C2. Dunque:

$$T_{fault} = N_{fault-C2}\,T_{trasfM-C2} + N_{fault-C1}\,T_{trasfC2-C1} = \frac{N}{\sigma_2}\,T_{trasfM-C2}(\sigma_1) + \frac{N}{\sigma_1}\,T_{trasfC2-C1}(\sigma_1)$$

La latenza di trasferimento di un blocco-C1 da M a C2, tramite la struttura di interconnessione ad albero, vale:

$$T_{trasfM-C2} = 2T_{tr} + \tau_M + lg_2m\,(\tau + T_{tr}) = 53\,\tau$$

Inoltre, $T_{trasfC2\text{-}C1}\,(\sigma_1) = 2\,\sigma_1\,\tau$. Quindi:

$$T_{fault} = \frac{53}{128}\,N\,\tau + 2\,N\,\tau = 2.4\,N\tau$$

In conclusione, il tempo di completamento vale:

$$T_c = T_{c-0} + T_{fault} = 16.4\,N\,\tau$$

Domanda 3

Realizzando a microprogramma un'operazione lunga, come la moltiplicazione, questa avrebbe un tempo di servizio *e* una latenza dell'ordine di *O(n)*, con *n* numero di bit della parola: ad esempio, circa 50τ per il processore D-RISC elementare con *n* = 32.

Realizzando l'operazione come rete combinatoria, il tempo di servizio e la latenza avrebbero ordine *O(1)*, cioè (1 ÷ 2)τ. Questa realizzazione – *EU di latenza unitaria* – garantisce il tempo di servizio richiesto (*t*) per la CPU pipeline, ma comporta un costo troppo elevato in termini di area del chip CPU, dell'ordine di $O(2^n)$.

Con una realizzazione parallela si può ridurre il *tempo di servizio* rispetto alla realizzazione sequenziale, portandolo a *O(1)*; con alcune forme di parallelismo la *latenza* può essere ugualmente minimizzata, con altre risulta maggiore. La scelta della forma di parallelismo dipende principalmente dal *costo*, inteso come complessità *O(...)* della struttura logica, che deve risultare drasticamente abbattuto rispetto a quello della EU di latenza unitaria.

La realizzazione adottata è quella basata sulla forma di parallelismo *pipeline con loop-unfolding*, dove ogni stadio contiene una rete combinatoria moltiplicatore su basso numero di bit: ad esempio, per *n* = 32, 4 stadi con ogni stadio contenente un moltiplicatore su 8 bit. La complessità, e quindi l'area, è ora dell'ordine di $O(2^8)$, contro $O(2^{32})$ della EU di latenza unitaria. Una realizzazione di tipo *farm* avrebbe invece un costo dell'ordine di $O(2^{32})$, in quanto ogni worker sarebbe una EU di latenza unitaria.

Poiché ogni stadio del pipeline ha tempo di servizio *e* latenza uguale a t ($t = \tau$ oppure 2τ a seconda della realizzazione delle comunicazioni), il tempo di servizio dell'intera operazione risulta uguale a t, mentre la latenza è uguale alla somma delle latenze degli stadi ($4t$ nell'esempio).

Il tempo di servizio uguale a t garantisce che si raggiunga la massima performance in assenza di dipendenze logiche, proprio come nell'ipotetica CPU con EU di latenza unitaria, ma a un costo di molti ordini di grandezza inferiore.

La contropartita è rappresentata dalla latenza della realizzazione pipeline, che si ripercuote sul ritardo dovuto a dipendenze logiche, da cui discende tutta la problematica dell'ottimizzazione dei programmi.