Architettura degli Elaboratori

Appello del 28 giugno 2011

Riportare su tutti i fogli consegnati nome, cognome, numero di matricola, corso A/B, e la sigla NEW (per nuovo ordinamento), oppure OLD-0 (per vecchio ordinamento, nuovo programma), oppure OLD-1 (per vecchio ordinamento, vecchio programma). I risultati verranno pubblicati sulle pagine web del corso/dei docenti appena disponibili.

Domanda 1 (tutti)

a) Spiegare come è definito il metodo di indirizzamento diretto per memorie cache.

b) Si consideri una unità cache primaria C con metodo di indirizzamento diretto e scritture con il metodo write-through. Il chip CPU contiene anche una cache secondaria. La MMU non ha interfacce verso la memoria esterna.
Scrivere il microprogramma dell'unità C, esclusa la fase di gestione del fault di blocco, e, sulla base di tale microprogramma, mostrarne la parte Operativa e valutarne il ciclo di clock e il tempo di servizio utilizzando simboli opportuni. A seconda del programma di esame, fare riferimento ad una CPU pipeline oppure convenzionale.

c) Completare il microprogramma dell'unità C per quanto riguarda anche la gestione del fault di blocco, e, sulla base di tale microprogramma, valutare il tempo di trasferimento di un blocco dalla cache secondaria alla cache primaria.

Fornire adeguate spiegazioni per ogni risposta *a), b), c).*

Domanda 2 (tutti)

Si consideri una gerarchia di memoria memoria principale – memoria cache su domanda, senza ulteriori livelli di cache e con memoria principale avente banda di elaborazione uguale a una parola per tempo di accesso.

Dimostrare che, in tali ipotesi, programmi che godano solo della proprietà di località hanno un tempo di completamento maggiore o uguale rispetto al caso in cui l'architettura non preveda cache.

A seconda del programma di esame, fare riferimento ad una CPU pipeline oppure convenzionale.

Domanda 3 (NEW, OLD-0)

Si consideri il seguente programma D-RISC:

```
        LOAD   RA, 0, Ra
        LOAD   RB, 0, Rb
        LOAD   RC, 0, Rc
        LOAD   RD, 0, Rd
        IF <   Rc, Rd, THEN
        DIV    Ra, Rc, Rx
        INCR   Rx
        GOTO   CONT
THEN:   MUL    Ra, Rb, Ra
        DIV    Rc, Rd, Rc
        ADD    Ra, Rc, Rx
CONT:   STORE  RX, 0, Rx
        END
```

a) Con riferimento ad una CPU pipeline scalare con Unità Esecutiva parallela, avente unità pipeline in virgola fissa a quattro stadi, fornirne una versione ottimizzata e valutarne il tempo di completamento in funzione del ciclo di clock e della probabilità che si verifichi il predicato dell'istruzione IF, senza considerare i fault di cache. *Fornire adeguate spiegazioni.*

b) Spiegare l'interprete firmware dell'Unità Istruzioni per l'istruzione STORE RX, 0, Rx.

Domanda 3 (OLD-1)

a) Descrivere e *spiegare* il trattamento interruzioni per una CPU D-RISC, dall'istante in cui viene rilevata l'interruzione all'istante in cui inizia l'elaborazione dello handler.

b) Come al punto *a),* ma nell'ipotesi che le unità di I/O siano capaci solo di inviare il segnale di interruzione e non siano predisposte ad inviare il messaggio di I/O in seguito all'accettazione dell'interruzione.

c) Con riferimento alla fase di commutazione di contesto, *spiegare* quali sono le strutture dati condivise riferite indirettamente e come vengono gestite *nel caso specifico*.

Traccia di soluzione

Vengono riportati solo gli elementi essenziali per comprendere la soluzione, che va completata con adeguate spiegazioni, codici e schemi. Per molte spiegazioni e per il formalismo si rimanda al libro di testo.

Domanda 1 (tutti)

a) Si tratta di spiegare la legge di corrispondenza tra blocchi e la modalità di rimpiazzamento in caso di fault, senza ancora descrivere l'implementazione (Cap. VIII, sez. 3.2).

b) Ora si descrive l'implementazione. Una volta spiegato come dalla legge di corrispondenza si ricava la strutturazione dell'indirizzo fisico in campi opportuni, si passa a definire le interfacce con MMU, con il processore (o con l'Unità Esecutiva di una CPU pipeline), con la cache secondaria C2 e con la memoria principale M; la richiesta di scrittura viene inoltrata in parallelo sia a C2 che a M. Il microprogramma, facente uso di controllo residuo comandato dalla condizione di fault, è il seguente:

0. (RDYMMU, OP, ACKC2, ACKM, ACKP = 0 – – – –, 1 1 0 – –, 1 1 1 0 –, 1 0 – – 0) nop, 0;

   (= 1 0 – – 1) **(MC[IND.BC ° IND.D] → OUTP, set RDYP, reset ACKP) | (TAB[IND.BC]. TAG = IND.TAG and TAB[IND.BC]. PRESENZA = 1),**

   not (TAB[IND.BC]. TAG = IND.TAG and TAB[IND.BC]. PRESENZA = 1) → FAULT,

   (IND, OP, DATAIN) → SAVE, 1;

   (= 1 1 1 1 –) (IND, OP, DATAIN) → INTERF_C2, set RDYC2, reset ACKC2,

   (IND, OP, DATAIN) → INTERF_M, set RDYM, reset ACKM,

   (DATAIN → MC[IND.BC ° IND.D]) | (TAB[IND.BC]. TAG = IND.TAG and TAB[IND.BC]. PRESENZA = 1),

   not (TAB[IND.BC]. TAG = IND.TAG and TAB[IND.BC]. PRESENZA = 1) → FAULT,

   (IND, OP, DATAIN) → SAVE, 1

   1. (FAULT …. = …)

Mostrare la PO. Spiegare che i segnali di abilitazione comandati dal controllo residuo devono anche tenere conto dello stato interno (0) della PC e delle configurazioni opportune delle variabili di condizionamento.

Il ciclo di clock è determinato come segue:

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

$T_{\sigma PO}$: utilizzando la tecnica del controllo residuo, le memorie TAB e MC si stabilizzano in parallelo (condizione essenziale per contenere la lunghezza del ciclo di clock a valori compatibili con quelli del processore); assumendo che TAB e MC abbiano lo stesso ritardo di stabilizzazione t_a, si ottiene $T_{\sigma PO} = t_a + 2t_p$, da cui

$$\tau = t_a + 5t_p.$$

che rappresenta anche il tempo di servizio di C in assenza di fault.

c) Il trasferimento del blocco da C2 a C avviene con un'unica richiesta ed un ciclo di σ risposte:

1. (FAULT = 0) “nop”, 0;

   (= 1) (SAVE.IND.TAG, 1) → TAB[SAVE.IND.BC], (SAVE.IND.BM, ‘read’, –) → INTERF_C2, set RDYC2, 0 → I, 2 // I è un registro di $1 + \log_2\sigma$ bit //

2. (I_0, RDYINC2 = 0 0) nop, 2;

   (= 0 1) reset RDYINC2, set ACKINC2, DATAINC2 → MC[SAVE.IND.BC ° I_m], I + 1 → I, 2;

   (= 1 –) “nop”, 0

Considerando che l’accesso in lettura a C2 (con tempo di accesso uguale a 2τ) avviene in parallelo alla scrittura in MC, e notando che il microprogramma contiene delle “nop” eliminabili:

$$T_{trasf} = 2\ \sigma\ \tau$$

Domanda 2 (tutti)

Se tali programmi prevedono *N* accessi in memoria, nell’architettura senza cache vengono spesi *N* tempi di accesso alla memoria, mentre nell’architettura con cache viene speso un numero maggiore o uguale di accessi, precisamente:

$$\left\lceil \frac{N}{\sigma} \right\rceil \sigma$$

Spiegare che tutto il resto della valutazione del tempo di completamento rimane invariato.

Domanda 3 (NEW, OLD-0)

```
        LOAD   RA, 0, Ra
        LOAD   RB, 0, Rb
        LOAD   RC, 0, Rc
        LOAD   RD, 0, Rd
        IF <   Rc, Rd, THEN
        DIV    Ra, Rc, Rx
        INCR   Rx
        GOTO   CONT
THEN:   MUL    Ra, Rb, Ra
        DIV    Rc, Rd, Rc
        ADD    Ra, Rc, Rx
CONT:   STORE  RX, 0, Rx
        END
```

a) Le degradazioni delle prestazioni sono dovute:

- a *(1 + p)* casi di salto, con *p* probabilità che il predicato dell’istruzione IF sia vero;
- alle dipendenze logiche IU-EU di LOAD RD su IF (k = 1, NQ = 2, senza ulteriori effetti di dipendenze EU-EU), nel ramo con probabilità *(1 – p)* di INCR su STORE (k = 2, NQ = 1, con latenza ulteriore di *4t* causata della dipendenza EU-EU di DIV Ra su INCR), nel ramo con probabilità *p* della ADD su STORE (k = 1, NQ = 1, con latenza ulteriore di *4t* causata della dipendenza EU-EU della sola DIV Rc su INCR, essendo MUL e DIV indipendenti).

Le ottimizzazioni consistono:

- nello spostare le LOAD RA, LOAD RB dopo le LOAD RC, LOAD RD; la dipendenza indotta dalla LOAD RD sulla IF diviene di distanza 3 con NQ = 2, quindi non ha effetto;
- nell'eliminare l'effetto del GOTO con la tecnica del delayed branch replicando la STORE dopo il GOTO stesso. Non è utile applicare delayed branch alla IF, in quanto lo spostamento di istruzioni di LOAD dopo la IF stessa contribuirebbe ad aumentare il valore di NQ per le dipendenze logiche di INCR su STORE e di ADD su STORE, che quindi rimangono come nel programma originario:

```
        LOAD   RC, 0, Rc
        LOAD   RD, 0, Rd
        LOAD   RA, 0, Ra
        LOAD   RB, 0, Rb
        IF <   Rc, Rd, THEN
        DIV    Ra, Rc, Rx
        INCR   Rx
        GOTO   CONT, delayed_branch
        STORE  RX, 0, Rx
THEN:   MUL    Ra, Rb, Ra
        DIV    Rc, Rd, Rc
        ADD    Ra, Rc, Rx
        STORE  RX, 0, Rx
CONT:   END
```

La valutazione, sia con modello dei costi che con simulazione grafica (in entrambi i casi: spiegare), fornisce come risultato che al tempo di completamento ideale uguale a *9t* si aggiunge una bolla ampia *6t* con probabilità *p* oppure ampia *4t* con probabilità *(1 – p)*, quindi $T_c = (13 + 2p)\ t = (13 + 2p)\ 2\tau$.

b) L'interprete firmware in IU effettua tutte le seguenti azioni in un singolo ciclo di clock:

controlla la validità dell'istruzione ricevuta; nel caso specifico la STORE è sempre valida, anche dopo GOTO con delayed branch;

decodifica l'istruzione; trattandosi di una STORE:

verifica l'aggiornamento dei registri generali sorgente testando i rispettivi semafori; nel caso specifico, trova rosso il semaforo associato a RG[Rx] e attende che diventi verde; se sono applicate forme di out-of-ordering, nel frattempo verifica se ulteriori istruzioni in arrivo sono valide e abilitate (non ce ne sono nel caso specifico);

se/appena tutti i dati sono disponibili, invia a DM il messaggio (RG[RX] + 0, RG[Rx], 'write');

contemporaneamente, verifica la presenza di messaggi (dato, indice) da EU; se uno è presente aggiorna RG[indice] = dato, SEM[indice] = SEM[indice] – 1; se SEM[indice] assume il valore zero verifica se ci sono istruzioni in attesa dell'aggiornamento di RG[indice] ed eventualmente provvede ad eseguirle.

Domanda 3 (OLD-1)

a) Vedi Cap. IX, sez. 2.1, 2.2.

b) La fase firmware si limita ad inviare l'accettazione dell'interruzione e a chiamare la routine di interfacciamento interruzioni. Questa effettua una scansione (polling) delle unità di I/O, effettuando una lettura in Memory Mapped I/O a indirizzi logici noti, finché non trova la condizione di interruzione richiesta; a questo punto legge i due parametri (codice evento, dato) da altre due locazioni note o puntate dalla stessa locazione letta. Ora la fase assembler prosegue normalmente.

c) Le strutture condivise riferite indirettamente sono

- il PCB del processo da portare in esecuzione, riferito indirettamente tramite la testa o un altro elemento della Lista Pronti;
- la Tabella di Rilocazione del processo da portare in esecuzione, riferita attraverso un campo del PCB stesso: A seconda di come viene definita l'istruzione START_PROCESS e implementata la MMU; questa informazione serve o per ottenere l'indirizzo fisico della Tabella di Rilocazione (traduzione usando esplicitamente la Tabella), o per disporre dell'entrata della Tabella di Rilocazione relativa alla struttura stessa.

Scegliere un metodo per strutture condivise riferite indirettamente e applicarlo al caso specifico.