Architettura degli Elaboratori, 2007-08

Esercitazione 4 di verifica

Soluzione: entro venerdì 23 novembre

Domanda 1

Si consideri il programma D-RISC risultante dalla soluzione della Domanda 1 dell'Esercitazione 3.

L'architettura dispone di una memoria principale avente ciclo di clock uguale a 100 volte il ciclo di clock τ della CPU. I collegamenti inter-chip hanno latenza di trasmissione uguale a 10 τ.

a) Esprimere il tempo di completamento del programma in funzione di τ, del numero di elementi n della lista e della probabilità p che $A_i \geq B_i$.

b) Valutare la performance del calcolatore per un campo di applicazione nel quale il programma in oggetto sia ritenuto caratterizzante, nell'ipotesi che la frequenza del clock della CPU sia uguale a 4 GHz, $n = 100$, $p = 0{,}5$.

c) Spiegare se il programma assembler, il suo interprete firmware, e le prestazioni (tempo di completamento, performance) cambiano, ed eventualmente come, nel caso che gli array A, B, C siano memorizzati in unità di ingresso-uscita, nell'ipotesi che tutte le memorie del sistema, esterne alla CPU, abbiano lo stesso ciclo di clock della memoria principale e che tutti i collegamenti inter-chip abbiano latenza di trasmissione uguale a 10 τ.

Domanda 2

Valutare, in funzione di τ, del ciclo di clock della memoria principale e della latenza di trasmissione inter-chip, il tempo di elaborazione del trattamento interruzioni, a partire dall'istante in cui si verifica l'interruzione fino all'istante in cui inizia l'elaborazione dell'handler specifico, nell'ipotesi che le unità di I/O abbiano lo stesso ciclo di clock della CPU.

Dare la valutazione nei due casi seguenti:

a) tutte le informazioni utilizzate nella fase firmware sono allocate in registri generali,

b) tutte le informazioni utilizzate nella fase firmware sono allocate nel PCB del processo.

Soluzione

Domanda 1

a) Riprendiamo il programma D-RISC dall'Esercitazione 3:

programma principale:

```
            LOAD   Rtesta, 0, Rfine
            IF ≠ 0   Rfine, FINE
            LOAD   Rtesta, 1, Rpun
LOOP:       LOAD   Rpun, 0, RA
            LOAD   Rpun, 1, RB
            LOAD   Rpun, 2, RC
            MOVE   RN, RM
            CALL   Rproc, Rret
            LOAD   Rpun, 3, Rfine
            LOAD   Rpun, 4, Rpun
            IF = 0   Rfine, LOOP
FINE:       END
```

eseguite **1** volta

Operazioni sulla testa della lista per inizializzare la scansione

Passaggio dei parametri alla procedura

Loop **do while** per la scansione della lista

ripetuto **n** volte; anche la prima e l'ultima istruzione della procedura

procedura:

```
            CLEAR   Ri
LOOP1:      MUL   Ri, RM, Rt
            ADD   RC, Rt, RCriga
            LOAD   RA, Ri, Ra
            CLEAR   Rj
LOOP2:      LOAD   RB, Rj, Rb
            IF ≥   Ra, Rb, THEN
            SUB   Rb, Ra, Rc
            GOTO   CONT
THEN:       SUB   Ra, Rb, Rc
CONT:       STORE   RCriga, Rj, Rc
            INCR   Rj
            IF <   Rj, RM, LOOP2
            INCR   Ri
            IF <   Ri, RM, LOOP1
            GOTO   Rret
```

if then else

Loop più interno

Loop più esterno

ripetuto **n N** volte

ripetuto **n N^2** volte

Sono date informazioni sul **peso relativo delle istruzioni appartenenti alle diverse sezioni**. Essendo $N = 1K$, ed essendo paragonabile il numero di istruzioni delle diverse sezioni, se ne deduce che le prestazioni dipendono, con ottima approssimazione, solo dal *loop più interno della procedura*, le cui istruzioni sono ripetute $n\ N^2$ volte; già quelle del loop più esterno della procedura, essendo ripetute $n\ N$ volte, hanno peso trascurabile, ed maggior ragione quelle eseguite un numero di volte indipendente da N.

Quindi il tempo di completamento è espresso come:

$$T_c \sim n\ N^2\ T_{iter}$$

dove T_{iter} è il tempo di completamento medio della sequenza di istruzioni appartenenti al loop più interno:

```
0. LOOP2:   LOAD   RB, Rj, Rb
1.          IF ≥   Ra, Rb, THEN
2.          SUB    Rb, Ra, Rc
3.          GOTO   CONT
4. THEN:    SUB    Ra, Rb, Rc
5. CONT:    STORE  RCriga, Rj, Rc
6.          INCR   Rj
7.          IF <   Rj, RM, LOOP2
```

La sequenza di istruzioni 0, 1, 4, 5, 6, 7 è eseguita con probabilità ***p***.
La sequenza di istruzioni 0, 1, 2, 3, 5, 6, 7 è eseguita con probabilità ***(1 – p)***.

Il numero medio di tali istruzioni è quindi dato da:

$$n_{iter} = 6p + 7\ (1 - p) = 7 - p$$

Il *numero medio di istruzioni eseguite dal programma* è:

$$n_{istr} \sim n_{iter}\ n\ N^2 = (7 - p)\ n\ N^2$$

Usando i simboli del cap. VI:

$$T_{iter} = n_{iter}\ T_{ch} + 2\ T_{ex\text{-}LD} + 2\ T_{ex\text{-}IF} + 2\ T_{ex\text{-}SUB} + (1 - p)\ T_{ex\text{-}GOTO}$$

dove:

$$T_{ch} = 2\ \tau + t_a$$

$$T_{ex\text{-}LD} = 2\ \tau + t_a \text{ (vale anche per STORE)}$$

$$T_{ex\text{-}IF} = 2\ \tau$$

$$T_{ex\text{-}SUB} = \tau \text{ (vale anche per INCR)}$$

$$T_{ex\text{-}GOTO} = \tau$$

da cui:

$$T_{iter} = (23 - 3p)\ \tau + (9 - p)\ t_a$$

Essendo:

$$t_a = 2\ (\tau + T_{tr}) + \tau_M = 122\ \tau$$

si ha:

$$T_{iter} = (1121 - 125\ p)\ \tau$$

e infine:

$$T_c \sim n\ N^2\ (1248 - 250\ p)\ \tau \sim (1175 - 131\ p)\ 10^6\ n\ \tau$$

Con i dati $n = 100$, $\tau = 1/4GHz = 0{,}25\ nsec$, $p = 0{,}5$:

$$T_c \sim 27{,}7\ sec$$

b) Il tempo medio di elaborazione si ricava dal tempo di completamento:

$$T = \frac{T_c}{n_{istr}} = \frac{1175 - 125\,p}{7 - p}\,\tau$$

Si verifichi che, per questo programma, *T* deve essere sostanzialmente indipendente da *n* e *N*.

Numericamente, con i dati del problema:

$$n_{istr} = 682 \bullet 10^6$$

da cui:

$$T = \frac{29{,}45}{734}\,10^{-6} = 0{,}041\ \mu sec$$

La performance per questo campo di applicazione vale quindi:

$$\wp = 24{,}6\ \ MIPS$$

c) Poiché (come da specifiche) gli array A, B, C sono *già* memorizzati nelle memorie associate a unità di ingresso-uscita, *non conviene* trasferire questi dati in memoria principale, visto che una macchina D-RISC è dotata di memory mapped I/O. In queste condizioni, nessuna modifica va apportata al programma compilato, visto che gli accessi alle memoria di I/O sono effettuati, in maniera invisibile, con le stesse istruzioni di LOAD e STORE.

Nessuna modifica va apportata all'interprete firmware, *in quanto* le istruzioni di LOAD e STORE operano su indirizzi logici, e quindi il microprogramma del processore non ha alcuna visibilità dell'allocazione delle informazioni in memoria fisica (principale o di I/O).

Le prestazioni subiscono una leggera variazione, a causa del fatto che, agli effetti della latenza per effettuare un accesso ad una memoria di I/O, occorre attraversare anche

- due volte l'unità di I/O associata, nell'ipotesi che questa sia realizzata a firmware,
- due volte l'unità di interfaccia tra unità di I/O e memoria di I/O, nell'ipotesi che l'unità di I/O sia realizzata con un processore general-purpose.

In entrambi i casi, con i dati del problema, occorre aggiungere solo 2τ al tempo di accesso in memoria:

$$t_a = 124\,\tau$$

da cui le leggere modifiche alle prestazioni:

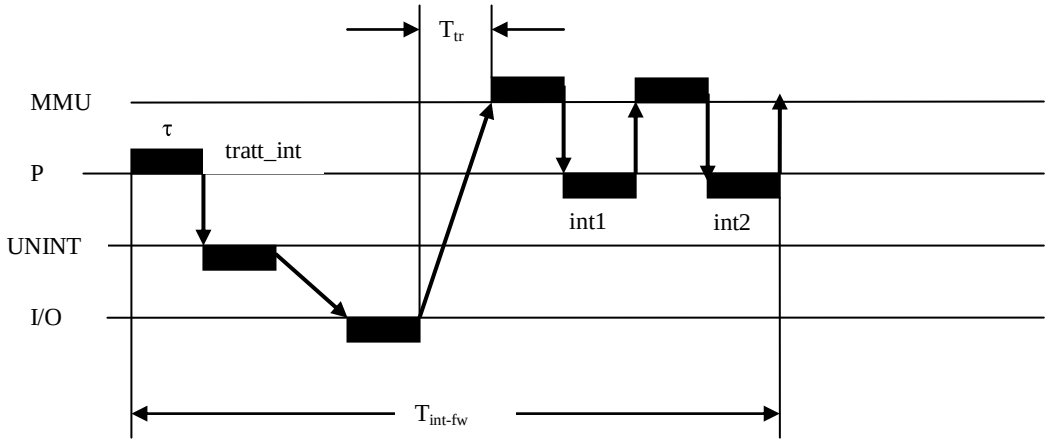
$$T_c \sim 28{,}1\ sec$$

$$\mathcal{P} = 24{,}3\ MIPS$$

Ritorniamo sulla frase iniziale: "Poiché (come da specifiche) gli array A, B, C sono *già* memorizzati nelle memorie associate a unità di ingresso-uscita, *non conviene* trasferire questi dati in memoria principale, …". Questo è vero in quanto il tempo di accesso alle memorie di I/O è paragonabile a quello in memoria principale. Questa condizione non sempre è soddisfatta, ad esempio (con dati diversi del problema) a causa della maggiore latenza del Bus di I/O oppure per la presenza di memoria cache (come vedremo). In un caso in cui si rivelasse conveniente trasferire gli array in memoria principale, a condizione che l'unità di I/O disponga di DMA, tale trasferimento andrebbe provocato a programma, e questo comporterebbe *modifiche al programma* stesso (come modificarlo verrà visto nella parte del corso dedicata ai processi).

Domanda 2

a) Basandoci sul microprogramma della sez. 5.3.2, Cap. VI, la valutazione del tempo di elaborazione della *fase firmware* del trattamento interruzioni può essere effettuata con l'aiuto del seguente diagramma temporale:



Tale latenza è quindi data da:

$$T_{int\text{-}fw} = 7\ \tau + 2\ T_{tr}$$

Per fare iniziare lo handler occorre eseguire le prime due istruzioni della routine di interfacciamento interruzioni (vedi sez. 5.3.3, Cap. VI), che comportano un tempo di completamento:

$$T_{int\text{-}rout} = 2\ T_{ch} + T_{ex\text{-}LD} + T_{ex\text{-}CALL} = 3\ (2\ \tau + t_a) + \tau = 7\ \tau + 3\ t_a = 13\ \tau + 6\ T_{tr} + 3\ \tau_M$$

Complessivamente il tempo di elaborazione del trattamento interruzione, fino all'inizio dello handler, è dato da:

$$T_{int} = T_{int\text{-}fw} + T_{int\text{-}rout} = 20\ \tau + 8\ T_{tr} + 3\ \tau_M$$

b) La soluzione del punto a) comporta il minimo tempo di elaborazione del trattamento interruzione. La contropartita è *dedicare* un certo numero di registri generali esclusivamente a questa fase ("esclusivamente", in quanto non è, ovviamente, possibile inserire codice assembler per salvare il contenuto dei quattro registri usati nella fase firmware!). Inoltre, la routine di interfacciamento ne usa altri per la base della tabella interruzioni, l'indirizzo di handler, e l'indirizzo di ritorno da handler: questi possono essere normalmente salvati a programma prima dell'inizio della routine stessa, e ripristinati prima del ritorno al programma interrotto; la stessa cosa vale per i registri usati da handler.

Le informazioni della fase firmware possono essere allocate nel PCB del processo in posizioni note. Noto l'indirizzo del PCB (implicito, ad esempio uguale a zero, oppure in un registro generale *dedicato*), nel PCB trovano spazio le due parole del messaggio di I/O, l'indirizzo della routine di interfacciamento interruzioni, e l'indirizzo di ritorno da tale routine. Questo comporta che la fase firmware effettui quattro accessi in memoria, quindi al costo di un maggior tempo di elaborazione della fase firmware stessa.

Sinteticamente, il microprogramma, che usa registri temporanei visibili a livello firmware, diviene:

tratt_int.	come sempre
int1.	ricevi prima parola da MMU e richiedi la scrittura in memoria
int2.	ricevi seconda parola da MMU
int3.	attendi fine prima scrittura e richiedi la scrittura in memoria della seconda parola
int4.	attendi fine seconda scrittura
int5.	richiedi la lettura da memoria dell'indirizzo della routine
int6.	attendi la fine della lettura e scrivi il dato letto in IC, richiedi la scrittura in memoria di IC
int7.	attendi la fine della scrittura, vai a ch0

I due cicli di clock spesi in *int2* sono sovrapposti alla scrittura della prima parola. Il tempo di elaborazione della fase firmware è quindi:

$$T_{int\text{-}fw} = 5\ \tau + 2\ T_{tr} + 5\ \tau + 4\ t_a$$

dove il terzo addendo (5τ) è costituito dai cicli di clock delle microistruzioni *int3, int4, int5, int6, int7* al di fuori delle attese delle risposte da memoria.

La routine comprende anche tre LOAD per la lettura delle due parole del messaggio di I/O e dell'indirizzo di ritorno al programma interrotto, ed eventualmente un certo numero di LOAD e STORE per salvare e ripristinare gli altri registri usati.