Architettura degli Elaboratori, a.a. 2005-06

Appello del 12 gennaio 2006

Correzione

Le seguenti caratteristiche sono comuni alle Domande 1 e 2 che seguono:

(I) dati due array unidimensionali A e B, ognuno di $N = 4K$ interi, costruire un array bidimensionale M di $N \times N$ interi tale che, $\forall\ i, j = 0 \ldots N - 1$, sia $M[i, j] = A[i] + B[j]$;

(II) una gerarchia di memoria è costituita da memoria principale M di capacità 1G parole e memoria cache C di capacità 32K parole. Sia UC l'unità, avente ciclo di clock τ, su cui risiede C. C ha le seguenti caratteristiche: blocchi di 8 parole, funzionamento su domanda, indirizzamento con il metodo completamente associativo, e scritture gestite con il metodo Write-Through. M è interallacciata con 8 moduli ed ha ciclo di clock uguale a 20τ. Tutti i collegamenti inter-chip hanno latenza di trasmissione uguale a 5τ.

Domanda 1

Una unità di elaborazione U svolge il compito **(I)**, ricevendo da una unità U1 gli indirizzi base di A, B e M ed inviando ad una unità U2 una segnalazione alla fine del calcolo **(I)**. U usa la gerarchia di memoria **(II)** ed è realizzata sullo stesso chip di UC.

a) Valutare, in funzione di t_p, il tempo medio di elaborazione di U. t_p è il ritardo di una porta logica con al massimo 8 ingressi; una ALU ha ritardo uguale a $5t_p$, e la durata dell'impulso di clock è uguale a t_p.

Spiegare quale impatto hanno avuto tutte le caratteristiche della memoria cache.

b) Dire se la seguente affermazione è vera o falsa, spiegando la risposta: "per qualunque unità di elaborazione, tutte le informazioni per verificare la condizione necessaria per il corretto funzionamento della Parte Operativa sono ricavabili direttamente osservando il microprogramma e solo da questo".

Domanda 2

Valutare, in funzione del ciclo di clock, il tempo di completamento di un programma assembler Risc (Cap. V) che svolge il compito **(I)**. La CPU usa la gerarchia di memoria **(II)** ed include, nello stesso chip, UC.

Domanda 3

a) Siano E1, E2 elaboratori general-purpose aventi lo stesso ciclo di clock e con set di istruzioni rispettivamente S1, S2. S2 è dato da S1 con l'aggiunta di istruzioni per il calcolo delle funzioni trigonometriche base. Spiegare se la performance di E1 è maggiore, uguale o minore di quella di E2.

b) Spiegare la differenza tra spazio di indirizzamento e insieme di lavoro (working set), quest'ultimo riferito alla gerarchia memoria virtuale – memoria principale.

c) Sia un elaboratore con sistema operativo scritto nel linguaggio concorrente Lc della "Note sul livello dei processi". Spiegare perché: *i)* il sistema operativo non deve essere ricompilato ogni volta che viene compilato un nuovo programma applicativo; *ii)* la compilazione di una programma applicativo non cambia se le funzionalità di un processo di sistema operativo sono sostituite da una unità di I/O.

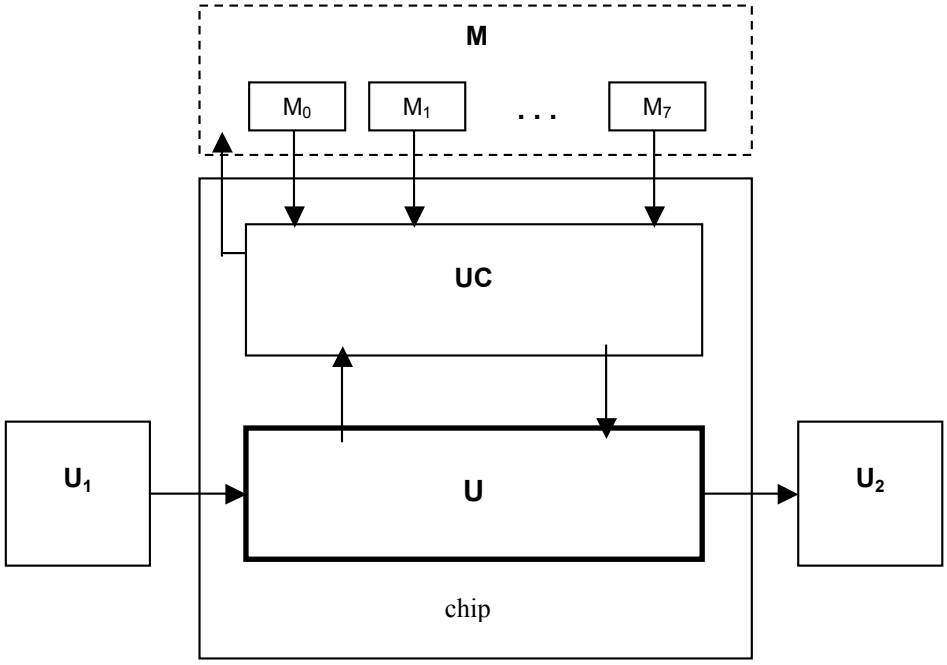
d) Spiegare le azioni svolte dal processore per effettuare una commutazione di contesto ed a quale spazio di indirizzamento tali azioni fanno riferimento.

Premessa

*La domanda **D1a** è dello stesso tipo degli esercizi UC1 e UC2 delle Note "Esercizi sull'architettura del processore e memoria cache". **D1b** è completamente svolto nelle Note "Reti sequenziali e strutturazione firmware" sez. 2.4; si veda anche la "Correzione della Prima Prova di Verifica Intermedia", domanda D1d. **D2** è un tipico esercizio di compilazione in assembler e di valutazione delle prestazioni, come esemplificato in esercizi svolti nelle dispense (esercizi del tutto analoghi, con loop annidati, sono stati svolti a lezione) e nelle suddette Note "Esercizi sull'architettura del processore e memoria cache". **D3a** è l'esercizio 5b di tali Note. Le risposte alle domande **D3b**, **D3c**, **D3d** sono spiegate ampiamente nelle dispense, nelle Note "Processi e processori" e nelle "Note sul livello dei processi"; in ogni caso, le riposte a **D3b** e **D3**c sono contenute nella "Correzione della Seconda Prova di Verifica Intermedia".*

Domanda 1

a) Lo schema del sistema è il seguente:



***Nota importante:** è un grave errore considerare la memoria cache C come una o più memorie di registri di U, in quanto C è contenuta in una unità distinta UC. Anche nel caso in cui fosse stato richiesto di realizzare C nella stessa unità U, i dati A, B, M sarebbero comunque stati memorizzati in una memoria cache: la traduzione degli indirizzi (da M a C) e la gestione del fault di blocco di cache avrebbero dovuto comparire esplicitamente tanto nel microprogramma quanto nella struttura della PO. In questo esercizio, essendo C realizzata in una unità distinta UC, ad U non è assolutamente visibile come C sia realizzata: U si limita a chiedere accessi in memoria di una parola alla volta, in lettura o scrittura, attraverso le interfacce. L'impatto delle caratteristiche di C si ha nella valutazione del tempo medio di elaborazione di U.*

L'algoritmo da implementare è:

```
int A[N]; int B[N]; int M[N][N];
    { ricevi ( base_A, base_B, base_M ) da U1;
      for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                  M[i[j] = A[i] + B[j];
       invia ( ) a U2
    }
```

Le interfacce a transizione di livello con U1 e U2 contengono rispettivamente i registri (RDY1, ACK1, base_A (30 bit), base_B (30), base_M (30)) e (RDY2, ACK2; la segnalazione è un messaggio senza valore). La comunicazione con UC è a domanda-risposta con interfaccia di uscita (RDYOUTC, IND (30), DATAOUT (32), OP (1)) e di ingresso (RDYINC, DATAIN (32)).

Il microprogramma di U è riportato di seguito. L'algoritmo dell'interprete segue fedelmente quello ad alto livello e (a parte non usare un indirizzamento base più indice, che non comporta vantaggi a livello firmware) è lo stesso utilizzato per la compilazione nella D2. Il significato dei registri è insito nei nomi adottati. I registri contatori I e J sono di 13 bit.

0. (RDY1 = 0) nop, 0;

   (= 1) reset RDY1, set ACK1, base_A → INDA, base_B → INDB, base_B → INDB_BASE, base_M → INDM, 0 → I, 0 → J, 1

1. (I_0, ACK2 = 0 -) INDA → IND, 'read' → OP, set RDYOUTC, INDA + 1 → INDA, 2;

   (= 1 0) nop, 1;

   (= 1 1) reset ACK2, set RDY2, 0

2. (RDYINC = 0) nop, 2;

   (= 1) reset RDYINC, DATAIN → TEMP, 3

3. (J_0 = 0) INDB → IND, 'read' → OP, set RDYOUTC, INDB + 1 → INDB, 4;

   (= 1) I + 1 → I, 0 → J, INDB_BASE → INDB, 1

4. (RDYINC = 0) nop, 4;

   (= 1) reset RDYINC, DATAIN + TEMP → DATAOUT, INDM → IND, 'write' → OP, set RDYOUTC, 5

5. (RDYINC = 0) nop, 5;

   (= 1) reset RDYINC, J + 1 → J , INDM + 1 → INDM, 3

Nota: *sarebbe possibile ridurre di uno il numero delle microistruzioni, rendendo più complicato l'algoritmo, ma senza con questo ridurre il numero medio di cicli di clock.*

Il tempo medio di elaborazione di U è dato da:

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

Costruito il grafo degli stati della PC, si vede che si ha ciclo esterno ripetuto N volte durante il quale

- si esegue la sequenza [3, 1, 2] ([0, 1, 2] solo la prima volta), che comprende un accesso in memoria (ad A),
- si esegue il ciclo interno per N volte costituito dalla sequenza [3, 4, 5], che comprende due accessi in memoria (a B e ad M).

Quindi:

$$T_{id} = N\,[\,3\tau + t_c + N\,(3\tau + 2t_c)\,]$$

Tenendo conto che $t_c = 2\tau$ (metodo completamente associativo), si ottiene:

$$T_{id} = N\,[\,5\tau + 7N\tau] \cong 7N^2\,\tau$$

Per valutare il numero di fault di cache si osservi che, essendo N sostanzialmente minore della capacità di C, è possibile fare in modo che, dopo la gestione di N/σ fault per l'accesso ai blocchi di B durante la prima iterazione del *for* più esterno, l'intero array B rimanga in cache per tutta la successiva durata del microprogramma. *L'insieme di lavoro* del microprogramma è quindi dato da un blocco di A (l'accesso ad A causa quindi N/σ fault), tutti i blocchi di B, ed un blocco di M. Poiché M è in sola scrittura, il numero di fault che interessa valutare è dato da quelli generati per accedere ad A e da quelli per accedere a B:

$$N_{fault} = \frac{N}{\sigma} + \frac{N}{\sigma} = \frac{N}{4}$$

Poiché UC adotta la tecnica Write-Through, e M è interallacciata con 8 moduli, la scrittura di ogni parola in M non provoca alcun ritardo nell'elaborazione di UC e quindi di U, sovrapponendosi al calcolo interno.

Il tempo di trasferimento di un blocco da M a C è dato da:

$$T_{trasf} = t_M + m\tau = \tau_M + 2T_{tr} + \sigma\tau = 38\tau$$

Quindi:

$$T_{fault} = \frac{38N\tau}{4} = 9{,}5N\tau$$

che risulta trascurabile rispetto a T_{id}. In conclusione:

$$T \cong T_{id} \cong 7N^2\,\tau$$

con una efficienza relativa della cache ≅ 1.

Per valutare il ciclo di clock, si ha che

- $T_{\omega PO} = 0$;
- $T_{\omega PC} = T_{\sigma PC} = 2t_p$, in quanto il massimo numero di variabili significative di un termine AND, sia per la funzione σ_{PC} che per ω_{PC}, è 5 (3 variabili dello stato interno e 2 variabili di condizionamento), e che il numero di termini AND messi in OR, sia per σ_{PC} che per ω_{PC}, è minore di 8;
- per quanto riguarda $T_{\sigma PO}$, osserviamo che la PO contiene due ALU sufficienti a supportare, con il massimo parallelismo presente nel microprogramma, 6 operazioni aritmetiche (cinque incrementi ed una somma). Quindi, all'ingresso delle ALU sono presenti commutatori a due livelli di logica, così come per i registri in cui viene scritto il valore all'uscita di una qualunque ALU:

$$T_{\sigma PO} = 2T_K + T_{ALU} = 4t_p + 5t_p = 9t_p$$

Quindi:

$$\tau = T_{\omega PO} + T_{\omega PC} + T_{\sigma PO} + \delta = 2t_p + 9t_p + t_p = 12t_p$$

$$T \cong 7N^2\,\tau = 1344\,M\,t_p = 1{,}344\,G\,t_p$$

b) L'affermazione è falsa, in quanto la condizione deve essere verificata facendo l'analisi della struttura della rete sequenziale PO.

La condizione necessaria per il corretto funzionamento di PO è che, *ad ogni ciclo di clock*, il valore delle variabili di condizionamento dipenda esclusivamente dallo stato interno di PO (contenuti di registri) e non dallo stato d'ingresso di PO (valori delle variabili di controllo). Anche se nelle condizioni logiche di un microprogramma compaiono solo contenuti di registri di PO (e non potrebbe essere altrimenti), la realizzazione della PO deve essere tale da rispettare tale variabili di condizionamento.

Come esempio, consideriamo una variabile di condizionamento espressa, nel microprogramma, come *x = segno (A – B)*. Nella realizzazione della PO, questa funzione non può essere realizzata in modo qualunque, ma deve essere ricavata come uscita secondaria di una ALU che esegua *solo* A – B. Qualunque altra realizzazione farebbe dipendere il valore di *x*, *ad ogni ciclo di clock*, da variabili di controllo.

Domanda 2

La compilazione in assembler Risc dell'algoritmo visto in D1a è la seguente:

```
LOOP1:  LOAD   RA, Ri, Ra
LOOP2:  LOAD   RB, Rj, Rb
        ADD    Ra, Rb, Rb
        STORE  RM, Rj, Rb
        INCR   Rj
        IF<    Rj, RN, LOOP2
        ADD    RM, RN, RM
        CLEAR  Rj
        INCR   Ri
        IF<    Ri, RN, LOOP1
        END
```

loop più interno
(anche Rj è inizializzato a zero a tempo di compilazione)

Per le spiegazioni da dare sugli aspetti dell'allocazione e inizializzazione dei registri generali, si vedano le Note con altri esercizi svolti e le correzioni delle prove di verifica intermedia.

Per le stesse ragioni viste in D1a, il tempo di completamento ideale è praticamente dato dal tempo di completamento del *loop più interno*:

$$T_{cid} \cong N^2 \, (5 \, T_{ch} + 2 \, T_{ex\text{-}LD} + 2 \, T_{ex\text{-}ADD} + T_{ex\text{-}IF}) = N^2 \, [5 \, (2\tau + t_c) + 2 \, (2\tau + t_c) + 2\tau + 2\tau]$$

Tenendo conto che ora $t_c = 3\tau$, si ha:

$$T_c \cong T_{cid} \cong 39 \, N^2 \, \tau$$

visto che T_{fault}, valutato come in D1a (la probabilità di fault delle istruzioni è trascurabile), è trascurabile rispetto a T_{cid}.

Domanda 3

a) La risposta deriva immediatamente dalla definizione di performance $\mathcal{P}$ come inverso del tempo medio di elaborazione, valutato come:

$$T = \sum_i p_i T_i$$

con p_i e T_i rispettivamente probabilità di occorrenza e tempo medio di elaborazione dell'istruzione *i-esima* (o della classe di istruzioni *i-esima*).

Considerando le istruzioni che S2 ha in più rispetto a S1, si può senz'altro affermare, senza alcuna necessità di scriverne l'interprete, che esse sono certamente del tipo "lungo", cioè hanno un tempo di elaborazione di molti cicli di clock (almeno come MUL, DIV, ecc). Quindi, purché abbiano una probabilità non nulla, si ha che

$$T_2 > T_1 \qquad \Rightarrow \qquad \mathcal{P}_2 < \mathcal{P}_1$$

L'uguaglianza si avrebbe solo in campioni di programmi per i quali la probabilità delle suddette istruzioni fosse uguale a zero.

b) Lo spazio di indirizzamento di un processo è l'insieme di tutti gli indirizzi logici che il processo può generare a tempo di esecuzione (dove "può" significa "con probabilità non nulla").

Considerato come un insieme di pagine logiche, lo spazio di indirizzamento comprende l'insieme di lavoro. Questo, riferito ad un certo istante (ad una certa finestra temporale), è il sottoinsieme delle pagine dello spazio di indirizzamento che, in quell'istante (in quell'intervallo di tempo), devono risiedere nella memoria principale per minimizzare (per contenere il più possibile) la probabilità di fault di pagina.

Di regola, l'insieme di lavoro è di ampiezza molto minore dello spazio di indirizzamento, ciò che è alla base dell'efficienza del concetto di gerarchia di memoria.

c) ***i)*** I processi del sistema operativo sono caratterizzati da un insieme di canali di comunicazione d'ingresso sui quali ricevere richieste di servizio da parte di processi applicativi. Le eventuali risposte al servizio sono inviate attraverso canali di comunicazione "parametrici" il cui identificatore è specificato dai richiedenti del servizio stesso. Concettualmente, la compilazione dei processi applicativi avviene prima in $\mathcal{L}_c$ per dar luogo al collegamento, via primitive di comunicazione, con i processi di SO interessati.

Il supporto delle primitive di comunicazione è indipendente dagli specifici processi comunicanti, in quanto tutte le informazioni che caratterizzano i partner non fanno mai uso di nomi di processi, bensì sono specificate per indirizzo logico attraverso la struttura dati descrittore di canale: messaggi, variabili targa, descrittori di processo. Questo permette di realizzare i processi del SO come predefiniti, cioè compilati una volta per tutte, indipendentemente dalle applicazioni che vengono di volta in volta ad essi collegate.

ii) Se le unità di I/O sono realizzate come processi (processi "esterni") del linguaggio concorrente $\mathcal{L}_c$, ad esse possono essere trasferite *le funzionalità* ed *i canali* dei processi che sono chiamate a sostituire, senza che questa sostituzione abbia impatto sulla compilazione dei processi applicativi: infatti le primitive *send* e *receive* operano su variabili condivise, realizzate nello spazio di memoria di I/O e/o di memoria principale, *senza alcuna necessità di inserire, nel loro supporto, operazioni specifiche del sottosistema di I/O e diverse da quelle per supportare la comunicazione tra processi "interni"* (eseguibili dalla CPU). La *sveglia di processi interni* da parte di processi esterni avviene con il meccanismo delle interruzioni, in maniera del tutto invisibile ai processi interni stessi: l'Handler esegue la stessa funzionalità di sveglia processo invocata direttamente nelle primitive tra processi interni.

d) La commutazione di contesto si può avere quando il processo in esecuzione passa: *i)* in stato di attesa, oppure *ii)* in stato di pronto esaurendo il quanto di tempo (se previsto) o eseguendo una sveglia che comporti

prerilascio (se previsto), oppure *iii)* in stato di terminato eseguendo l'istruzione END (se il processo termina).

Nota: *queste considerazioni riguardano qualunque processo, non solo i processi applicativi.*

Consideriamo il caso *i)*. Sia A il nome del processo che passa in attesa. Tutte le azioni seguenti sono eseguite *nello spazio di indirizzamento di A*:

1. copia i contenuti dei registri generali (RG) e del contatore istruzioni (IC) nel descrittore di processo di A (PCB_A). Questa fase viene implementata mediante un serie di STORE. Se, come di regola, la commutazione di contesto è implementata come procedura, o costituisce la parte finale di una procedura, il valore di IC da salvare è già contenuto in un registro generale (indirizzo di ritorno da procedura);
2. stacca il primo PCB dalla Lista Pronti, il cui indirizzo è contenuto in una locazione puntata da una locazione del PCB_A. Sia B il processo il cui PCB è stato staccato dalla lista pronti (PCB_B);
3. copia i contenuti di apposite locazioni del PCB_B (immagini dei RG, di IC, indirizzo della Tabella di Rilocazione di B) nei RG, mediante una serie di LOAD. In particolare, in due registri generali verranno copiati l'indirizzo della Tabella di Rilocazione di B (IND_TABRIL) e il valore di IC da cui B deve riprendere l'esecuzione (NEW_IC); siano Ra e Rb gli indirizzi di tali registri rispettivamente;
4. esegui l'istruzione *START_PROCESS Ra, Rb, EI*, i cui argomenti sono rispettivamente *RG[Ra] = IND_TABRIL*, da passare a MMU, e *RG[Rb] = NEW_IC*, da copiare nel registro IC. Se, come di regola, la procedura in cui ci troviamo è eseguita a interruzioni disabilitate, la START_PROCESS provvede anche a riabilitare le interruzioni.

La prossima istruzione che verrà eseguita dal processore è la prima da cui riprende correttamente l'esecuzione del processo B.