Architettura degli Elaboratori, 2007-08

Appello del 14 luglio 2008

Domanda 1

Una unità di elaborazione U è così definita ha al suo interno due memorie, A e B, ognuna di 64K parole. U riceve da una unità U_1 messaggi del tipo *(op, J, L)*, con *op* di 1 bit, *J* indirizzo delle memorie A, B, e *L* di 10 bit, e da una unità U_2 messaggi *x* di tipo parola.

Se *op = 0*: scrive in A, a partire dall'indirizzo *J*, una sequenza di *L* parole inviate da U_2; A è considerato un array circolare.

Se *op = 1*: se *A[J] mod 256 > B[J] mod 256*, scrive B[J] in A[J] e invia B[J] a U_1.

Il tempo di accesso di A e B è uguale a $10t_p$ e il ritardo di stabilizzazione di una ALU è uguale a $5t_p$, con t_p ritardo di stabilizzazione di una porta logica con al più 8 ingressi.

Rispettando il seguente vincolo:

- il tempo di elaborazione dell'operazione esterna con *op = 1* deve essere uguale a un ciclo di clock,

scrivere il microprogramma e determinare il tempo di elaborazione di U per entrambe le operazioni esterne in funzione di t_p. Spiegare e motivare le scelte fatte per la progettazione di U e la scrittura del microprogramma.

Domanda 2

Dire se le seguenti affermazioni sono vere, false, oppure vere sotto condizioni da specificare, spiegando la risposta:

a) se la dimensione della memoria virtuale di un certo processo è maggiore della capacità della cache, allora l'insieme di lavoro di quel processo non può risiedere interamente in cache;

b) considerando la strutturazione a livelli di un sistema, la strategia di non deallocare blocchi dalla cache è implementata solo a livello firmware;

c) si supponga di realizzare la MMU contenente, al posto di una memoria associativa, una RAM la cui capacità sia sufficiente a contenere l'intera tabella di rilocazione di qualunque processo. Prescindendo da considerazioni di costo, l'affermazione da valutare è la seguente: con questa realizzazione il tempo di completamento di un processo è maggiore (o uguale) a quello che si ha dotando la MMU di una memoria associativa.

Domanda 3

Un sistema a livello di processi contiene *n* processi interni identici, $A_0, \ldots, A_{n-1}$, ed un processo esterno B. B ha una variabile locale costituita da una coda FIFO di blocchi ampi 1K parole inviati dal dispositivo associato. Un generico A_i può fare richiesta a B di un blocco; il blocco ricevuto viene assegnato ad una certa variabile targa.

Spiegare come implementare questo sistema:

a) nel caso che i processi, interni ed esterni, siano espressi in LC;

b) nel caso che i processi, interni ed esterni, siano espressi direttamente in assembler.

La scrittura completa e dettagliata dei processi in LC e in assembler è facoltativa, mentre la spiegazione a parole deve essere chiara ed eventualmente accompagnata da frammenti significativi di codice in LC o in assembler.

Sintesi di soluzione

Domanda 1

Notiamo che il risultato dell'operazione *w mod 256* è rappresentata dai $lg_2(256) = 8$ bit meno significativi della parola *w* (campo *w[24..31]*).

Concentrandoci sull'operazione esterna con *op = 1*, ci sono due modi di rispettare il vincolo di eseguirla in un unico ciclo di clock:

1. usare una variabile di condizionamento data da *segno(B[J][24..31] - A[J][24..31])*: la sua realizzazione corretta (PO di Moore) impone che la memoria A sia indirizzata separatamente dalle due sorgenti di indirizzo (la seconda sorgente è usata nell'altra operazione esterna) e che si usi una ALU dedicata a questa operazione;
2. usare controllo residuo: il valore *segno(B[J][24..31] - A[J][24..31])* fornisce il segnale di abilitazione alla scrittura in A e nell'interfaccia di uscita verso U_1.

Nel caso 1 il ciclo di clock (che, si può verificare, è imposto dall'operazione esterna con *op = 1*) si valuta come segue:

$T_{\omega PO} = T_{mem} + T_{ALU} + T_{mem} = 25t_p$

$T_{\sigma PO} = T_{mem} = 10t_p$ Il commutatore sull'ingresso del dato di A si stabilizza in parallelo alla stabilizzazione di β_A

$T_{\omega Pc} = T_{\sigma PC} = 2t_p$

$$\tau = 28t_p$$

Nel caso 2 si ottiene lo stesso valore di τ a condizione che A sia indirizzata separatamente, tecnica che, però, ora non è necessaria e (per convenzione) non viene quindi applicata. Inserendo un commutatore anche sull'indirizzo di A si ottiene:

$$\tau = 30t_p$$

Vediamo il microprogramma nel caso 2 (per l'operazione con *op = 1*, la comunicazione tra U_1 e U è a domanda-risposta; U_1 sa che, avendo chiesto *op = 1*, il risultato è significativo per ACK1 = 1):

0. (RDY1, OP = 0 –) nop, 0;

   (= 1 0) reset RDY1, set ACK1, J $\rightarrow$ IND, – L $\rightarrow$ I, 1;

   (= 1 1) reset RDY1, set ACK1, B[J] $\rightarrow$ (A[J], OUT1) | (segno(B[J][24..31] - A[J][24..31]) = 1), 0

1. (I_0, RDY2 = 1 -) nop, 0; // nop eliminabile //

   (= 1 0) nop, 1;

   (= 1 1) reset RDY2, set ACK2, IN2 $\rightarrow$ A[IND], IND + 1 $\rightarrow$ IND, I + 1 $\rightarrow$ I, 1

Quindi:

$$T_0 = (L + 2)\,\tau = 30t_p\,(L + 2)$$

$$T_1 = \tau = 30t_p$$

Domanda 2

a) Falsa: l'insieme di lavoro è definito come l'insieme di blocchi che, se presenti contemporaneamente in cache, minimizza la probabilità di fault. Tale numero è certamente minore (normalmente molto minore, se non altro per la presenza di molti oggetti del supporto a meccanismi di concorrenza utilizzati con bassa probabilità) della dimensione della memoria virtuale del processo espressa in blocchi di cache.

b) Falsa: l'unità cache ha bisogno che le venga esplicitamente indicato se un blocco può essere deallocato liberamente o se deve essere mantenuto in cache. Poter implementare questa scelta, effettuata a tempo di compilazione (o anche a tempo di esecuzione, ma comunque secondo una strategia prevista

esplicitamente nel codice del processo), ha precise ripercussioni sulla definizione del livello assembler del sistema: prevedere istruzioni speciali oppure annotazioni in istruzioni LOAD/STORE.

c) Vero: la tabella (acceduta per indice) verrebbe ad essere caricata per intero, in un'unica soluzione all'atto della commutazione di contesto oppure in più soluzioni in istanti diversi, ma comunque non su domanda. In questo modo, il numero di accessi in memoria da parte della MMU sarebbe maggiore (in un caso puramente teorico potrebbe anche essere uguale) rispetto a quelli effettuati su domanda con una memoria associativa di dimensione molto minore rispetto alla tabella di rilocazione, in quanto un certo numero (un grande numero) di entrate della tabella di rilocazione non vengono usate durante l'esecuzione del processo (vedi il concetto di insieme di lavoro applicato alla memoria virtuale).

Domanda 3

In entrambi i casi il processo B (unità di I/O) incapsula interamente tutte la variabili per realizzare la coda FIFO (array circolare, indici di inserzione ed estrazione, dimensione degli elementi, ecc), quindi è l'unico processo che esegue le operazioni di inserzione ed estrazione. Di conseguenza, un processo A_i può solo fare richiesta di ricevere un blocco: B, quando accetterà la richiesta di A_i, invierà il primo blocco in coda quando presente.

Nel caso *a)*, il generico A_i fa richiesta con un comando del tipo:

send (ch_to_B, ch_from_B)

e attende la risposta con un comando del tipo:

receive (ch_from_B, vtg)

B contiene una variabile *channelname* alla quale assegnare il valore *ch_from_B*. Poiché A_i si sospenderà sulla receive, l'esecuzione della send da parte di B provocherà la scrittura del blocco direttamente nella variabile targa, minimizzando così il numero di copie dei blocchi. Tale esecuzione provocherà anche una interruzione per svegliare A_i: il messaggio di interruzione contiene, nella seconda parola, l'identificatore di A, oppure l'indirizzo logico del PCB_A, oppure l'indirizzo fisico del PCB_A, oppure la capability del PCB_A, a seconda del metodo adottato per risolvere il problema delle strutture condivise riferite indirettamente (spiegare).

Nel caso *b)*, a livello assembler occorre emulare esplicitamente un funzionamento simile a quello che avremmo a livello di supporto di LC. Un modo è il seguente:

- A_i fa richiesta del blocco (mediante istruzioni STORE con indirizzi mappati nello spazio di I/O), comunicando i "riferimenti" della variabile targa e del proprio PCB. La natura di tali "riferimenti" dipende dal metodo adottato per risolvere il problema delle strutture condivise riferite indirettamente (vedi sopra; spiegare). Una volta eseguite tali istruzioni, A_i provoca la commutazione di contesto. Se l'unità di I/O non fosse immediatamente pronta ad eseguire la propria parte del supporto delle istruzioni di STORE (perché impegnata in altre inserzioni/estrazioni), ciò si traduce in una attesa del processore che si concluda una comunicazione a livello firmware (quindi è garantita la sincronizzazione, anche in questa fase, tra A_i e B);
- l'unità di I/O, quando serve la richiesta, scrive il primo blocco in coda nella variabile targa, utilizzando il relativo "riferimento", e invia una interruzione di sveglia, il cui messaggio di interruzione contiene il "riferimento" al PCB del processo da svegliare.