Architettura degli Elaboratori, 2008-09

Appello del 3 luglio 2009

Domanda 1

Una unità di elaborazione U

- contiene un componente logico memoria A di capacità 64K parole, ognuna di 32 bit;
- riceve in ingresso da una unità U_M un indirizzo IND di A;
- è collegato in ingresso e in uscita a due unità, U_1 e U_2: i messaggi in ingresso, IN_1 e IN_2, e di uscita, OUT_1 e OUT_2, sono di 32 bit.

Il funzionamento è il seguente:

> ricevuto il valore IND da U_M invia A[IND] a U_1 e U_2, attende da esse i valori IN_1 e IN_2 (risultati di elaborazioni che U_1 e U_2 effettuano sui valori OUT_1 e OUT_2), e memorizza in A[IND] il valore così definito: il suo *i*-esimo byte è uguale al massimo tra l'*i*-esimo byte di IN_1 e l'*i*-esimo byte di IN_2.

È noto il ritardo t_p di una porta logica con al più 8 ingressi. Il tempo di accesso di A e il ritardo di una ALU valgono $5t_p$.

È imposto il seguente vincolo: *U deve essere realizzata come singola rete sequenziale*.

a) Scrivere il microprogramma di U.

b) Definire completamente la funzione delle uscite e la funzione di transizione dello stato interno della rete sequenziale.

c) Calcolarne il ciclo di clock.

Le risposte devono essere adeguatamente spiegate.

Domanda 2

Si consideri un calcolatore la cui macchina assembler è D-RISC e la cui architettura firmware è quella del Cap. VI con cache associativa.

Il calcolatore contiene, tra le unità di I/O, quattro unità IN_1, IN_2, IN_3 e IN_4 e quattro unità OUT_1, OUT_2, OUT_3 e OUT_4, le prime capaci di ricevere, le seconde di inviare, al dispositivo associato blocchi di ampiezza N = 8K interi.

Le unità indicate contengono ognuna una memoria di capacità qualche decina di K parole, e non sono capaci di operare in DMA.

Si vuole trasferire blocchi di N parole dalla memoria dell'unità IN_i alla memoria dell'unità OUT_j, per qualunque combinazione di i e j. L'unità OUT_j è scelta di volta in volta da IN_i, e qualunque OUT_j è sempre disposta a ricevere da una qualunque IN_i.

Realizzare tale funzionalità secondo due approcci diversi:

a) considerando le unità di I/O implementate come processi esterni e disponendo del linguaggio LC,

b) non disponendo di alcuna forma di supporto a processi esterni.

spiegando adeguatamente le risposte. Inoltre, nel caso *b)*

c) valutare il tempo di completamento del trasferimento di un blocco dalla memoria di IN_i alla memoria di OUT_j in funzione del ciclo di clock τ del processore. Un protocollo completo a domanda-risposta sul Bus di I/O abbia latenza $T_{I/O} = 20\tau$ per ogni parola letta/scritta.

d) pronunciarsi esplicitamente sulla convenienza di prevedere caching dei dati di I/O nella CPU.

Soluzione

Domanda 1

Per rispettare il vincolo imposto, il microprogramma deve constare di una sola microistruzione.

Sia S un registro di 1 bit, inizializzato a zero, tale che: per S = 0 U è disposta a ricevere solo da U_M, altrimenti solo da U_1 e U_2.

Il protocollo con U_1 e U_2 sia a domanda-risposta.

Il microprogramma è:

0. (S, RDYM, RDYIN1, RDYIN2 = 0 0 – –) nop, 0;

   (= 0 1 – –) reset RDYM, set ACKM, M[IND] → OUT1, set RDYOUT1, A[IND] → OUT2, set RDYOUT2, 1 → S, IND → J, 0;

   (= 1 – 00, 1 – 01, 1 – 10) nop, 0;

   (= 1 – 11) reset RDYIN1, reset RDYIN2, *F(IN1, IN2)* → A[J], 0 → S, 0

La funzione *X = F(IN1, IN2)*:

$$\forall\, i = 0 \,..\, 3\text{: } X[i] = max\,(IN1[i], IN2[i])$$

è implementata da quattro ALU in parallelo operanti su byte, per calcolare le funzioni *segno (IN1[i] – IN2[i])*, e da quattro commutatori corrispondenti avente in ingresso *IN1[i]* e *IN2[i]* e comandati da *segno (IN1[i] – IN2[i])*. Il concatenamento delle uscite ordinate dei quattro commutatori rappresenta il valore da scrivere in A.

La rete sequenziale è costituita dalla rete PO con l'aggiunta delle funzioni β_S, β_M, β_{OUT} che rappresentano i valori delle variabili di controllo per abilitare la scrittura nei vari registri:

$$\beta_S = \bar{S}\ RDYM\ +\ S\ RDYIN1\ RDIN2$$

$$\beta_M = S\ RDYIN1\ RDIN2$$

$$\beta_{OUT} = \bar{S}\ RDYM$$

Le funzioni che definiscono la rete sequenziale, secondo il modello di *Moore*, sono quindi:

- *funzione delle uscite* (ω):

$$z1 = OUT1,\ z2 = OUT2$$

- *funzione di transizione dello stato interno* (σ):

$$in_S = when\ \beta_S\ do\ \bar{S}$$

$$in_{M[h]} = when\ \beta_M\ \ and\ (h = IND)\ do\ \ F(IN1, IN2)$$

$$in_{OUT1} = in_{OUT2} = when\ \beta_{OUT}\ \ do\ \ A[IND]$$

Il ciclo di clock vale:

$$\tau\ =\ max\ (T_\omega\ , T_\sigma) + \delta\ =\ T_\sigma + \delta\ =\ T_{ALU} + T_K + \delta\ =\ 8\,t_p$$

in quanto: la stabilizzazione delle variabili β avviene in parallelo alla stabilizzazione delle funzioni che figurano nella parte destra dei comandi *do*, il ritardo maggiore tra tutte le funzioni è quello della *F*, e la scrittura in A si sovrappone alla stabilizzazione della *F* stessa.

Domanda 2

L'architettura firmware del Cap. VI *non* permette che due unità di I/O comunichino *direttamente* tra loro. Le uniche comunicazioni consentite sul Bus di I/O sono quelle tra CPU ed una unità di I/O, in quanto l'arbitraggio tra le unità di I/O è effettuato dalla CPU stessa, che è sempre implicitamente l'unico destinatario dei messaggi sul Bus inviati da unità di I/O (messaggi di interruzione o risposte a richieste di accesso in Memory Mapped I/O) e l'unico possibile mittente (richieste di accesso in Memory Mapped I/O).

Occorre quindi che la comunicazione tra una unità IN_i ed una unità OUT_j avvenga con l'aiuto di una entità intermediaria, che nei due casi *a)* e *b)* è realizzabile come descritto nel seguito.

a) L'intermediario è un processo Q avente in ingresso un canale asimmetrico dai quattro processi esterni IN_i ed in uscita quattro canali simmetrici variabili verso i processi esterni OUT_j:

```
process Q, IN1, IN2, IN3, IN4, OUT1, OUT2, OUT3, OUT4;
Q :: channel in from_I/O (1); channel out var to_I/O; int X[8K];
    while true do {
        receive ( from_I/O, (to_I/O, X) );
        send ( to_I/O, X )
        }
```

b) Non disponendo del livello dei processi per quanto riguarda l'astrazione delle unità di I/O, l'intermediario non può che essere un handler eseguito nello spazio di indirizzamento di qualunque processo (non interessa se sia un processo LC o altro) in esecuzione sulla CPU.

Supponiamo che il problema delle strutture condivise riferite indirettamente sia risolto con il metodo degli indirizzi logici coincidenti (comunque, inserire spiegazioni sulle possibilità alternative).

L'unità IN_i invia una interruzione e, una volta accettata, comunica il messaggio di I/O = (evento = trasferimento tra unità, parametro = indirizzo base della struttura parametri).

Il parametro, contenuto nel registro generale *Rparam*, indirizza una struttura dati, allocata fisicamente nella memoria di IN_i, contenente l'indirizzo base della memoria logica (allocata in OUT_j) in cui trasferire il blocco, e il valore del blocco stesso.

Utilizzando esclusivamente Memory Mapped I/O, lo handler ha il seguente codice:

```
        LOAD   Rparam, 0, Rdest, DI
        INCR   Rparam
        CLEAR Ri
LOOP:   LOAD   Rparam, Ri, Rtemp
        STORE   Rdest, Ri, Rtemp
        INCR   Ri
        IF <   Ri, RN, LOOP
        GOTO Rret, EI
```

La struttura del Bus di I/O, che permette di leggere/scrivere una sola parola alla volta in modalità MMI/O, e la latenza del trasferimento di una parola, impediscono di realizzare efficientemente il trasferimento di blocchi da memoria di I/O nella cache della CPU. Anzi, tale meccanismo comporterebbe prestazioni peggiori rispetto alla totale mancanza di caching. In cache risiede solo il codice dello handler.

Il tempo di completamento dello handler è quindi dato da:

$$T_{handler} \sim N\,(4\,T_{ch} + 2\,T_{ex\text{-}LD} + T_{ex\text{-}INCR} + T_{ex\text{-}IF}) = N\,(15\,\tau + 4\,t_{cache} + 2\,T_{I/O}) = 67\,N\,\tau$$

(dato il valore di *N*, risulta trascurabile anche il tempo di elaborazione della fase firmware del trattamento interruzioni).