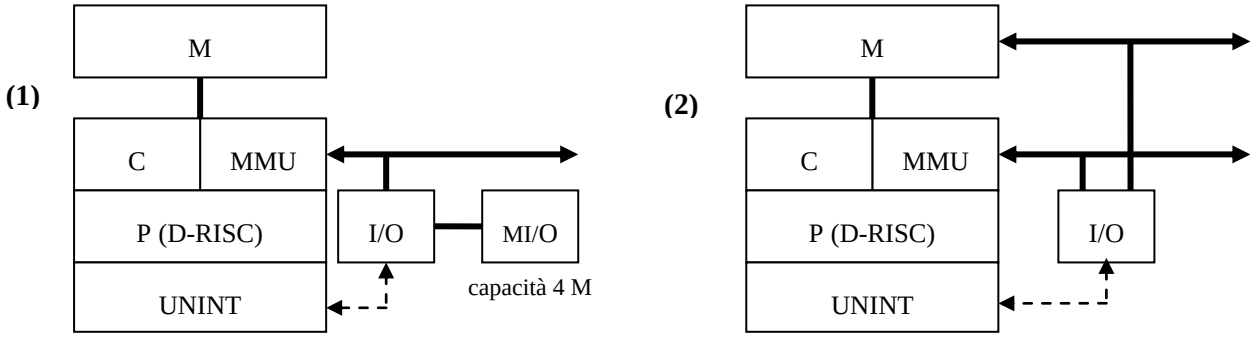
Architettura degli Elaboratori, 2007-08

Appello del 16 gennaio 2008

Domanda 1

Un processo Q opera su due array di interi, A[N][N] e B [N][N], con N = 1K. Il valore di A è ottenuto da un'unità di I/O; il valore di B è il risultato di una certa elaborazione su A.

La CPU ha processore D-RISC e cache primaria. Si considerino le seguenti configurazioni, (1) e (2), dell'architettura dell'elaboratore, con particolare riguardo all'unità di I/O:



Nel caso (1) l'unità di I/O ha associata una unità di memoria MI/O contenente un componente logico memoria di capacità 4M parole, mentre nel caso (2) una simile unità non è prevista. L'unità di I/O acquisisce le parole di A dal dispositivo associato (non indicato in figura).

a) Si supponga che I/O *non* sia implementato come processo esterno e che *non* sia virtualizzato da un processo driver. Spiegare come, nei casi (1) e (2), avviene la cooperazione tra Q e l'unità di I/O affinché Q possa operare su A: cioè le azioni svolte dall'unità di I/O, dal processore, e dal processore quando è in esecuzione il processo Q. Spiegare se il tempo di completamento di Q è uguale o diverso nei casi (1) e (2) ed eventualmente in quale caso è maggiore, assumendo che tutte le unità di memoria abbiano lo stesso ciclo di clock e che tutti i collegamenti inter-chip abbiano la stessa latenza di trasmissione.

b) Si supponga che I/O *sia* implementato come processo esterno e che *non* sia virtualizzato da un processo driver. Si assuma l'esistenza di un livello con linguaggio LC. Spiegare le stesse cose del caso *a)* e, rispetto a tale caso, mettere in evidenza le differenze funzionali ed eventualmente di prestazioni.

Domanda 2

L'elaborazione interna del processo Q della Domanda 1 è la seguente:

$$\forall i = 0 .. N - 1$$
$$\quad \forall j = 0 .. N - 1$$
$$\quad\quad B[i][j] = F(A[i][j], A[0][j])$$

Con riferimento alla cache, determinare l'insieme di lavoro e il numero di fault del programma, sapendo che la cache opera su domanda, ha capacità 16K parole, i blocchi sono ampi 16 parole, e la funzione F consta di circa 1000 istruzioni, tutte eseguite. Spiegare come è possibile garantire l'insieme di lavoro determinato.

Domanda 3

Una unità di elaborazione U ha tre ingressi esterni: A di 32 bit, B di 64 bit e C di 2 bit, ed una uscita esterna D di 4 bit. Per ogni tripla (A, B, C) il valore di D è uguale al numero di volte che il C-esimo byte di A compare tra i byte di B.

a) Con le opportune spiegazioni, scrivere il microprogramma di U in modo che il numero di cicli di clock impiegato sia lineare nel numero di byte di B, e valutarne il tempo di elaborazione in funzione del ritardo t_p di una porta logica con al più 8 ingressi. Una ALU ha tempo di stabilizzazione $5t_p$.

b) Con le opportune spiegazioni, realizzare U come una singola rete sequenziale, e valutarne il tempo di elaborazione in funzione di t_p. Spiegare se è possibile realizzare tale rete secondo il modello di Mealy.

Soluzione

Domanda 1

I due casi si caratterizzano per il modello di I/O, e di conseguenza per l'organizzazione fisica della memoria condivisa tra CPU e I/O:

(1) Memory Mapped I/O, con MI/O condivisa,

(2) Memory Mapped I/O e DMA, con M condivisa.

In D-RISC il modello MMI/O è primitivo, e quindi sempre disponibile, anche in presenza di DMA; anche nel caso (2), infatti, pur se non esiste fisicamente una memoria associata all'unità di I/O, MMI/O viene usato per scambiare informazioni con l'unità di I/O indirizzando locazioni fittizie.

In entrambi i casi (1) e (2), il funzionamento comprende le seguenti azioni:

1. il processo Q fa richiesta all'unità di I/O di un blocco di 1M parole. I parametri della richiesta dipendono dal formalismo disponibile (caso *a)* o *b)*); tra questi ci può essere la dimensione del blocco: per semplicità supporremo che 1M parole sia la dimensione standard del blocco;
2. il processo Q si sospende; l'invocazione della funzionalità di commutazione di contesto è esplicita o implicita a seconda del formalismo adottato;
3. l'unità I/O preleva dal dispositivo il blocco e lo scrive, parola per parola (man mano che le riceve), nella memoria condivisa: M/IO oppure M; come ricavare l'indirizzo base dipende dal formalismo;
4. I/O invia una interruzione per svegliare Q, indicando nel messaggio di I/O un riferimento a Q o al suo PCB, a seconda del formalismo. Il processo attualmente in esecuzione esegue l'handler che consiste appunto nella funzionalità di sveglia;
5. quando Q torna in esecuzione, esegue la computazione B = F (A, …) conoscendo l'indirizzo base di A in base al formalismo adottato per la descrizione.

Caso a): in qualunque linguaggio concorrente sia descritto Q, la cooperazione con l'unità di I/O deve essere espressa, per forza di cose, in assembler. Le azioni di Q si caratterizzano come segue:

azione 1: la richiesta è effettuata con una o più STORE con indirizzi logici mappati nello spazio di I/O; i parametri consistono in come riferire A e come riferire il PCB di Q. Ad esempio, adottando il metodo a capability, Q passa la capability di A e la capability del PCB. Oppure, se I/O non adotta capability, può essere passato l'identificatore di Q e un identificatore unico di A (se Q e I/O adottano indirizzi logici distinti), oppure l'indirizzo logico di Q e del PCB (indirizzi logici coincidenti), oppure l'indirizzo fisico di A e del PCB (dopo traduzione esplicita da parte di Q);

azione 2: la sospensione è esplicita, chiamando la procedura per commutare contesto;

azione 5: nel caso (2) anche per l'array A (oltre che per tutti gli altri oggetti) verrà sicuramente utilizzata la gerarchia memoria principale - cache, mentre nel caso (1) dipende se l'architettura prevede il trasferimento in cache di informazioni fisicamente presenti nello spazio di I/O in quanto, se non fosse possibile sfruttare una organizzazione della memoria di I/O a larga banda, tale trasferimento potrebbe essere troppo lento agli effetti delle prestazioni della cache (come normalmente accade).

La differenza in prestazioni dipende da questa distinzione: se non è possibile usare cache, il caso (1) ha un tempo di completamente maggiore.

Caso b): Q e I/O sono descritti con il linguaggio concorrente LC. Esiste un canale di comunicazione (*ch1*) da Q ad I/O per la richiesta, ed un canale (*ch2*) da I/O a Q per la risposta (valore di A). Le azioni si caratterizzano come segue:

azione 1: la richiesta di Q è effettuata con una *send* su *ch1*; se la dimensione del blocco è implicita, l'unico parametro è il valore di *ch2* che I/O assegnerà ad una variabile channelname. Tutti gli altri parametri del caso *a)* ora sono ricavati tramite i descrittori dei canali e, in generale, dal supporto delle comunicazioni. Il

supporto della *send* provvede ad inviare una segnalazione all'unità di I/O, che si trovava in attesa attiva nella *receive* su *ch1*;

azione 2: dopo aver eseguito la *send*, Q esegue una *receive* su *ch2*, che certamente provoca la sua sospensione. La commutazione di contesto è quindi effettuata implicitamente dal supporto;

azioni 3, 4: il processo esterno I/O, dopo aver eseguito la *receive* su *ch1*, comincia a leggere i dati dal dispositivo e, contestualmente, esegue la *send* su *ch2*. Poiché trova che Q è sospeso, il supporto della *send* provvede a scrivere il blocco di dati direttamente nella variabile targa A il cui riferimento si trova nel descrittore di canale (secondo uno dei metodi ricordati in precedenza); nel descrittore di canale trova anche il riferimento al PCB di Q, che verrà inviato nel messaggio di interruzione per svegliare il partner.

Valgono le differenze tra le prestazioni delle organizzazioni (1) e (2), dovute agli accessi della CPU tanto ai descrittori di canale *ch1, ch2*, quanto alla variabile *A*: nel caso (2) tutti questi oggetti sono fisicamente allocati in M, e quindi soggetti al trasferimento in cache.

Il tempo di completamento è praticamente lo stesso nel caso *a)* e nel caso *b)*.

Domanda 2

Gli array A e B siano memorizzati per righe. Tutti i blocchi di cache di A e di B sono acceduti almeno una volta, con piena località. Il riuso si ha solo per i blocchi della prima riga di A, che è utilizzata in tutte le iterazioni. Di conseguenze, tutta la prima riga di A fa parte dell'insieme di lavoro, assieme al blocco corrente di A ed al blocco corrente di B, complessivamente:

$$\frac{N}{\sigma} + 2 \quad blocchi\ di\ dati$$

con $\sigma = 16$. Per quanto riguarda le istruzioni, in assenza di altre informazioni si può assumere che ci sia riuso delle istruzioni della procedura, per cui fanno parte dell'insieme di lavoro all'incirca altri

$$\left\lceil \frac{1000}{\sigma} \right\rceil \quad blocchi\ di\ istruzioni$$

Infine, fanno certamente parte dell'insieme di lavoro di Q la sua Tabella di Rilocazione, il suo PCB e le informazioni per accedere alla Lista Pronti.

Data la capacità della cache, tutto l'insieme di lavoro può risiedere in cache.

Per imporre che i blocchi della prima riga di A non siano rimossi dalla cache, una volta letti per la prima volta, occorre che le istruzioni di LOAD che fanno riferimento alle componenti *A[0][j]* siano annotate con l'opzione "non_deallocare".

Il numero di fault, per l'elaborazione interna di Q, è dato da:

$$\left\lceil \frac{1000}{\sigma} \right\rceil \quad fault\ per\ istruzioni$$

$$\frac{N}{\sigma} \quad fault\ per\ la\ prima\ riga\ di\ A$$

$$\frac{N}{\sigma}(N-1) \quad fault\ per\ le\ altre\ righe\ di\ A$$

Complessivamente:

$$N_{fault} \sim \frac{N^2}{\sigma}$$

Essendo B in sola scrittura, i fault su B, sempre in numero di N^2/σ, non hanno impatto sulle prestazioni in quanto non comportano trasferimento di blocchi.

Osserviamo che, essendo il numero di fault quadratico in *N*, l'ottimizzazione di non deallocare la prima riga di A non ha un impatto significativo sulle prestazioni. D'altra parte, dal punto di vista metodologico va rimarcato che il compilatore cerca comunque di determinare le occasioni di riuso, come questa, anche se difficilmente potrà essere in grado di verificare, per ogni programma, se il riuso sia effettivamente utile o meno se non è ricavabile un modello dei costi del programma.

Domanda 3

(Traccia di soluzione: per maggiori spiegazioni e dettagli si vedano l'Esercitazione 1 e la Prima Prova di Verifica Intermedia)

a) L'unità fa uso di controllo residuo (mediante opportuni commutatori nella PO) per riferire il C-esimo bye di A ed il generico byte degli otto che fanno parte di B.

Per minimizzare il numero di cicli di clock, utilizzeremo una variabile di condizionamento "complessa" che indica se il C-esimo byte di A ed il generico byte di B sono uguali. Un'altra soluzione avrebbe potuto utilizzare una variabile di condizionamento "semplice" pur di adottare una opportuna inizializzazione e terminazione del loop, oltre che un opportuno parallelismo nelle microoperazioni.

// registri I e COUNT di 4 bit //

0. (RDYIN = 0) nop, 0; (=1) reset RDYIN, set ACKIN, A $\rightarrow$ M, B $\rightarrow$ N, C $\rightarrow$ J, 0 $\rightarrow$ I, 0 $\rightarrow$ COUNT, 1
1. (I_0, zero (M[J] – N[I_m]), ACKOUT = 0 0 –) COUNT + 1 $\rightarrow$ COUNT, I + 1 $\rightarrow$ I, 1;

   (= 0 1 –) I + 1 $\rightarrow$ I, 1;

   (= 1 – 0) nop, 1;

   (= 1 – 1) COUNT $\rightarrow$ D, set RDYOUT, reset ACKOUT, 0

La PO comprende 3 ALU, una (–) per la variabile di condizionamento zero (…), avente in ingresso solo le uscite dei commutatori di controllo residuo, le altre due (+1) per l'incremento di COUNT e di I. Si verifica la condizione necessaria per la correttezza della PO.

Agli effetti del ciclo di clock, si ha:

$$T_{\omega PO} = 7\ t_p \quad , \quad T_{\sigma PO} = 7\ t_p \quad , \quad T_{\omega PC} = T_{\sigma PC} = 2\ t_p$$

$$\tau = 17\ t_p$$

Il numero di cicli di clock è uguale a 2 (inizializzazione e terminazione) + 8 (loop), quindi il tempo di elaborazione vale $170\ t_p$.

b) Tutto il calcolo consiste in una funzione pura

$$F(A, B, C)$$

che quindi è realizzabile con una rete combinatoria, mostrata nella figura a pagina seguente. La rete, inclusa la parte *g*, viene interamente realizzata partendo da un formalismo algoritmico.

Il microprogramma consta di una sola microistruzione:

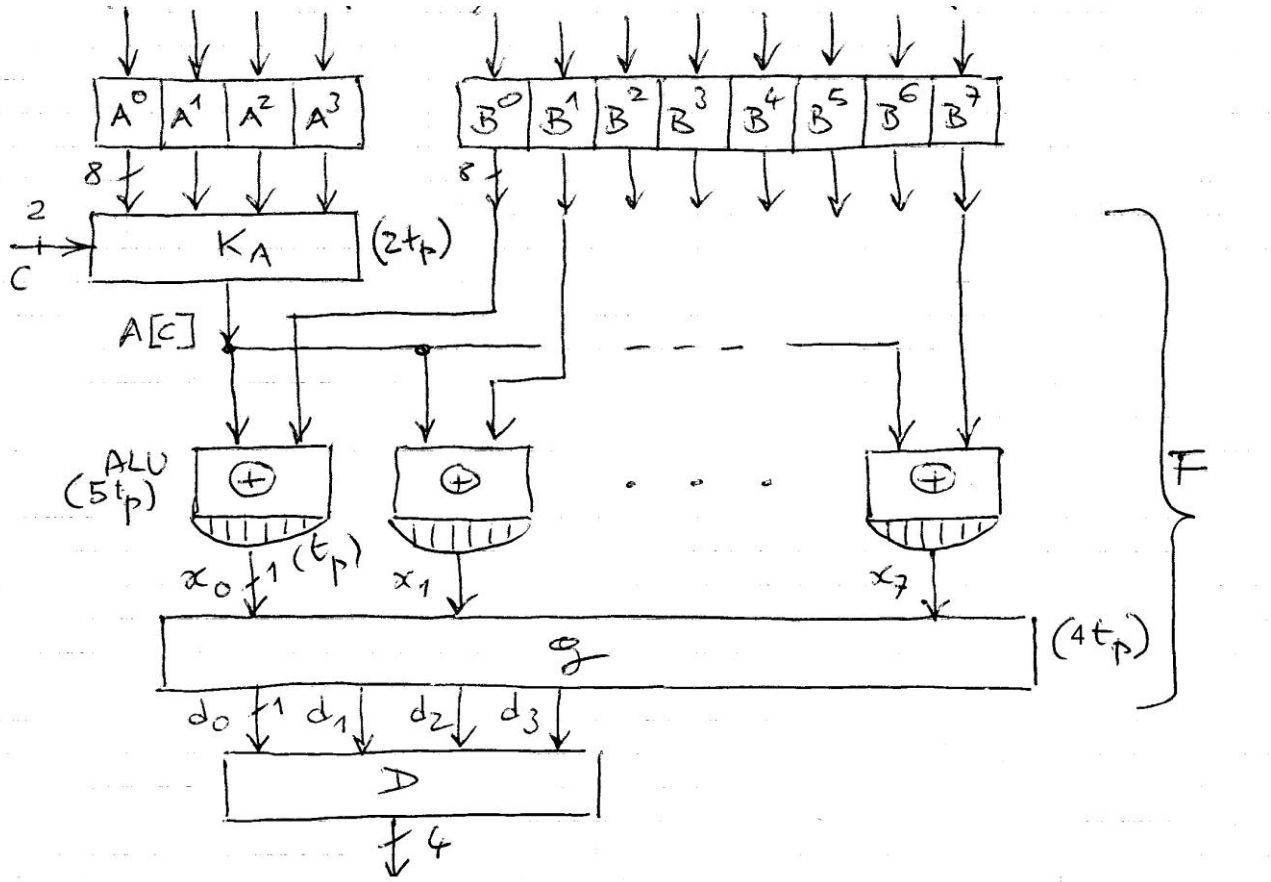
0. (RDYIN, ACKOUT = 0 – , 1 0) nop, 0;

   (= 11) reset RDYIN, set ACKIN, set RDYOUT, reset ACKOUT, *F*(A, B, C) $\rightarrow$ D, 0

La rete sequenziale consiste della rete combinatoria *F*, della rete combinatoria PC, e dei registri di interfaccia A, B, C, D, RDYIN, ACKIN, RDYOUT, ACKOUT. La rete combinatoria PC si stabilizza in parallelo alla *F*.

Si tratta di una rete di Moore, in cui la funzione di transizione dello stato interno è data da *F(A, B, C)* e dai valori degli ingressi dei registri di interfaccia, mentre la funzione delle uscite coincide semplicemente con i valori delle uscite dei registri D, ACKIN, RDYOUT.

L'eliminazione del registro D trasforma la rete di Moore in una rete di Mealy: la funzione delle uscite è ora data da *F(A, B, C)*, mentre la funzione di transizione dello stato interno è data dai valori degli ingressi dei registri di interfaccia.



La rete combinatoria *g* è definita come:

$$(d_0\ d_1\ d_2\ d_3) = numero\_di\_zeri\ (x_0\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7)$$

L'espressione logica può essere ricavata da una descrizione algoritmica in termini di *case*, con le seguenti caratteristiche:

- d_0: un solo termine AND con tutte le variabile negate;
- d_1: OR di tutti i possibili termini AND con 4 variabili negate, 5 variabili negate, 6 variabili negate, 7 variabili negate,
- d_2: OR di tutti i possibili termini AND con 2 variabili negate, 3 variabili negate, 6 variabili negate, 7 variabili negate,
- d_3: OR di tutti i possibili termini AND con 1 variabile negata, 3 variabili negate, 5 variabili negate, 7 variabili negate.

Si ha:

$$T_F = 12\ t_p \qquad , \qquad \tau = 13 t_p$$

Alternativamente, in modo più semplice ma con un maggior ritardo di stabilizzazione, la descrizione algoritmica può essere data in funzione della somma di tutti gli x_i negati. L'implementazione consiste in un albero di ALU (somma su 4 bit), su 3 livelli, quindi:

$$T_F = 23\ t_p \qquad , \qquad \tau = 24 t_p$$