Architettura degli elaboratori – A. A. 2016—2017

Prima prova di verifica intermedia – 2 novembre 2016

Una unità U_m è interfacciata con tre unità U_1, U_2 ed U_3 e contiene al suo interno una memoria M da 1K posizioni da 32bit. U_m riceve richieste da U_1 ed U_2 per la memorizzazione di un valore a 32bit X all'indirizzo IND, per la lettura ad un indirizzo IND o per conoscere l'indirizzo della posizione in cui si trova un certo valore X (in caso il valore non venga trovato, va restituito -1). U_1 ed U_2 interagiscono con U_m a domanda risposta. Al termine del servizio di una qualsiasi richiesta da U_1 ed U_2, U_m invia ad U_3 un report sull'operazione svolta costituito da I (indice dell'unità U_i che ha richiesto l'operazione), OP (operazione eseguita), IND e X (se significativi) che riportano dato e indirizzo coinvolti nell'operazione.

Si fornisca il microcodice e l'implementazione della unità U_m insieme al tempo medio di elaborazione delle operazioni esterne, considerando prioritario, nell'ordine:

- implementare una politica di servizio fair per U1 ed U2
- la minimizzazione del ciclo di clock
- la minimizzazione del tempo medio di elaborazione

Si giustifichino dettagliatamente tutte le decisioni prese nell'implementazione dell'unità. Si facciano le seguenti assunzioni:

- sono disponibili ALU intere a 32 bit che operano in $15t_p$
- il tempo di accesso in memoria va calcolato tenendo conto che ogni porta logica ha al più 8 ingressi
- non si può fare alcuna ipotesi sul tempo medio di risposta di U_3

Bozza di soluzione

Interfacce verso le altre unità

L'interfaccia verso Ui (i ∈ [1,2]) sarà costituita da:

- un indicatore a transizione di livello in ingresso RDYi
- un indicatore a transizione di livello in uscita ACKi
- un registro da 10 bit in ingresso INDi
- un registro da 32 bit in ingresso X
- un registro da 11 bit in uscita ESITO (in complemento a due)
- un registro OPi da 2 bit in ingresso

L'interfaccia verso U3 sarà invece costituita da:

- un indicatore a transizione di livello in ingresso ACK3
- un indicatore a transizione di livello in uscita RDY3
- un registro da 1 bit UNITA in uscita
- un registro da 2 bit OPOUT in uscita
- un registro da 32 bit XOUT in uscita

Politica di gestione delle richieste da U1 e U2

Per implementare una politica di gestione fair delle richieste da U1 ed U2 si utilizza un registro LAST da 1 bit che ricorda il numero dell'unità che ha inviato l'ultima richiesta servita. Si suppone che il registro sia inizializzato a 0 ed aggiornato opportunamente ogni volta che si serve una richiesta da una delle due unità.

In alternativa, si può considerare un microcodice composto da due porzioni diverse che operano nello stesso modo ma che danno priorità, in caso di richieste multiple, a una unità specifica.

Microprogramma

Versione 1: politica implementata utilizzando un registro da 1 bit LAST

Assumiamo che OP valga 0 per la prima operazione (scrittura X->M[IND]), 1 per la seconda (lettura M[IND]) e 2 per la terza (ricerca del primo IND per cui M[IND]==X). LAST vale 0 se l'ultima unità servita fosse U1 o 1 se invece fosse U2. Dal momento che non possiamo fare assunzioni sul tempo medio di risposta di U3, occorre esplicitamente testare l'ACK a trasmettere prima di impegnare l'interfaccia verso tale unità.

1. (RDY1,RDY2,ACK3,LAST,OP11,OP12,OP21,OP22=00------) nop, 0
   // prima operazione da U1, U2 non richiede operazioni, si può mandare il risultato a U3
   // oppure U2 chiede operazioni, ma è stata l'ultima unità servita
   (=101-00--,111100--) X->M[IND], set ACK1, reset RDY1,
   OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0 -> LAST, 0

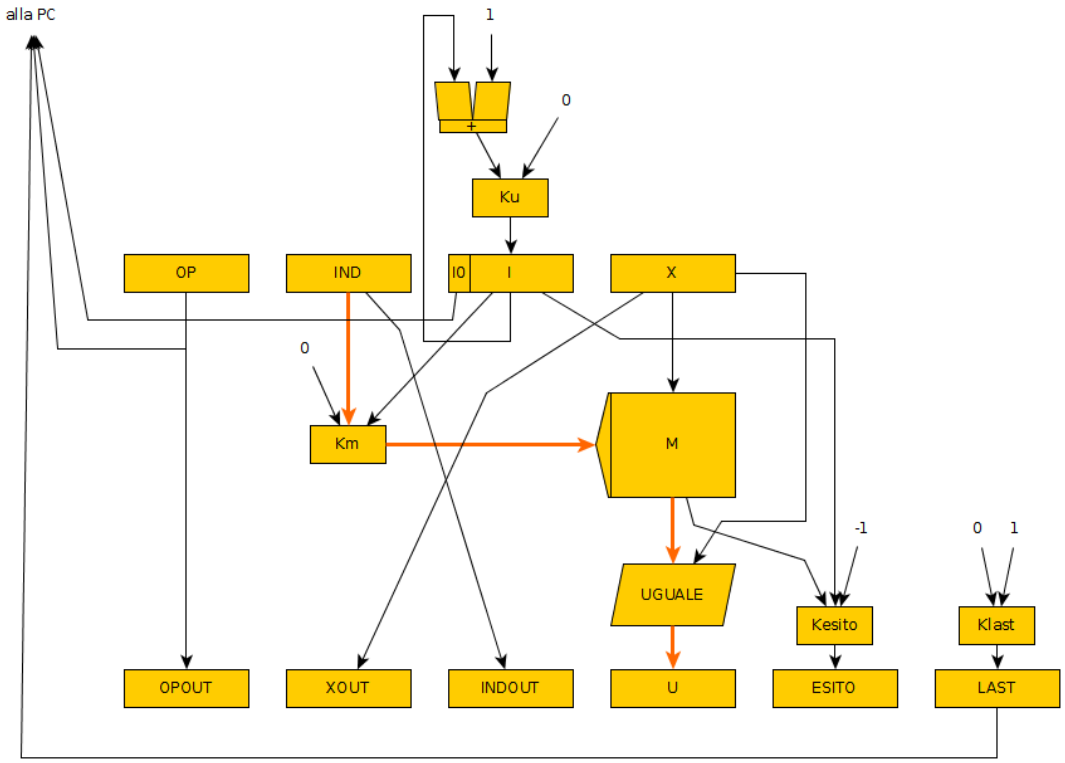
// seconda operazione da U1, U2 non richiede operazioni, si può mandare il risultato a U3
// oppure U2 chiede operazioni, ma è stata l'ultima unità servita
(=101-01--,=111101--) M[IND]->ESITO, set ACK1, reset RDY1,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0 -> LAST, 0
// terza operazione da U1, U2 non richiede operazioni, prepara ciclo e vai alla 1
// oppure U2 chiede operazioni, ma è stata l'ultima unità servita
(=10--10--,=11-110--) 1 -> I, uguale(M[0],X) -> U, 1
// prima o seconda operazione da U1, nessuna op da U2 manca ACK per U3, attesa
// oppure U2 chiede operazioni, ma è stata l'ultima unità servita
(=100-0---,=11010---) nop, 0
// stesse frasi con operazoine richiesta solo da U2
// prima operazione da U2, U1 non richiede operazioni, si può mandare il risultato a U3
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=011---00,1110--00) X->M[IND], set ACK2, reset RDY2,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 1 -> LAST, 0
// seconda operazione da U2, U1 non richiede operazioni, si può mandare il risultato a U3
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=011---01,=1110--01) M[IND]->ESITO, set ACK2, reset RDY2,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 1 -> LAST, 0
// terza operazione da U2, U1 non richiede operazioni, prepara ciclo e vai alla 1
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=01----10,11-0--10) 1 -> I, uguale(M[0],X) -> U, 2
// prima o seconda operazione da U2, nessuna op da U1 manca ACK per U3, attesa
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=010---0-,1100--0-) nop, 0

2. // trovato, manca ACK di U3
(U,ACK3,I_0=10-) nop, 1
// trovato, termina operazione
(=11-) I->ESITO, set ACK1, reset RDY1, 0 -> LAST,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0
// non trovato, non a fine memoria, prova posizione successiva
(=0-0) I+1 -> I, uguale(M[I],X)->U, 1
// non trovato, esito negativo e termina operazione
(=011) -1 -> ESITO, set ACK1, reset RDY1, 0 -> LAST,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0
3. // trovato, manca ACK di U3
(U,ACK3,I_0=10-) nop, 1
// trovato, termina operazione
(=11-) I->ESITO, set ACK2, reset RDY2, 1 -> LAST,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0
// non trovato, non a fine memoria, prova posizione successiva
(=0-0) I+1 -> I, uguale(M[I],X)->U, 1

// non trovato, esito negativo e termina operazione
(=011) -1 -> ESITO, set ACK2, reset RDY2, 1 -> LAST,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0

Il microprogramma è stato scritto in modo da minimizzare la durata del ciclo di clock. Non ci sono variabili di condizionamento complesse.

Dal microprogramma possiamo derivare una PO fatta come nella figura che segue (abbiamo omesso gli indicatori a transizione di livello nonchè le variabili di controllo α e β, per semplicità):



Il cammino in grassetto rosso è il cammino più lungo per la stabilizzazione della σPO.

Per la PC abbiamo 3 microistruzioni, 17 frasi, 11 variabili di condizionamento di cui al 6 testate contemporaneamente (al massimo). Quindi gli ingressi delle porte AND saranno al più 8 (2 bit di stato, 6 variabili di controllo). Con 17 frasi servono 2 livelli di porte OR (anche assumendo di codificare gli zeri e negare le uscite). Dunque $T_{\sigma PC}$ e $T_{\omega PC}$ saranno entrambi $3t_p$. Per ora consideriamo che questo sia il minimo.

Per la PO:

- Non ci sono variabili di condizionamento complesse, dunque in tutte le frasi $T_{\omega PO}$ sarà 0

- Nelle μ–operazioni, quelle che durano di più sono quelle che coinvolgono gli accessi alla memoria. La memoria viene indirizzata con 3 diversi valori, 0, IND e I, dunque serve un commutatore sull'ingresso indirizzi. La memoria è da 1K posizioni. Il commutatore di lettura avrà quindi in ingresso 10 bit di controllo oltre ai 1024 valori da commutare di cui uno solo è significativo. Il livello AND sarà costituito dunque da un albero di porte a due livelli. Il livello OR ha 4 livelli (parte intera superiore del logaritmo in base 8 di 1024). Quindi in totale il tempo di accesso sarà di $6t_p$. L'operazione più lunga sulla memoria è quella specificata come uguale(M[I],X)->U. In questo caso:
    - Deve stabilizzare il commutatore sull'ingresso degli indirizzi
    - Deve stabizzare il commutatore di lettura (t_a 0 = $6t_p$ come appena calcolato)
    - Dobbiamo calcolare se M[IND] è uguale a X. Per questo, possiamo utilizzare una ALU per calcolare M[IND]-X e poi controllare il flag Z. Questo richiederebbe però $15t_p$, secondo le assunzioni. Se usassimo invece 32 confrontatori e calcolassimo un OR delle uscite (due livelli di porte OR da 8 ingressi) potremmo calcolare un bit che vale 0 se sono M[IND] e X sono uguali in 4 t_p (2 per i confrontatori, che lavorano tutti in parallelo, e 2 per l'OR). Il risultato va negato per ottenere il bit U, ma al solito assumiamo che un NOT all'uscita di una porta AND o OR costi 0. Dunque utilizziamo questa soluzione per calcolare uguale(M[I],X).
    - Complessivamente dunque avremo un tempo di stabilizzazione per uguale(M[I],X)->U pari a $12t_p$.

Sotto queste ipotesi, possiamo calcolare la durata del ciclo di clock come:

$$\tau = 3t_p + 12t_p + t_p = 16t_p$$

Potremmo ridurre il numero sia delle frasi che delle microistruzioni del microprogramma considerando che la 1. e la 2. in realtà fanno la stessa cosa, utilizzando però ACKi, RDYi e valori in ingresso a LAST diversi. Dunque potremmo utilizzare una sola microistruzione (che porta il numero delle microistruzioni a 2 e il numero delle frasi a 13) utilizzando il controllo residuo per le operazioni di set e reset degli ACKi/RDYi e per un commutatore in ingresso a LAST. Questo allunga il $T_{\sigma PO}$ di un tk (2tp) ma questo accade in una frase dove comunque il tempo di stabilizzazione è inferiore a quello richiesto per il calcolo di uguale(M[I],X)->U e questo non compora un allungamento del $T_{\sigma PO}$. Per contro, 2 istruzioni riducono il numero di bit di stato della parte controllo da 2 a 1 (senza conseguenze sul tempo di stabilizzazione di $T_{\sigma PC}$ e $T_{\omega PC}$) e riducono anche il numero delle frasi. 13 frasi vuole dire massimo 7 "1" in ciascuna delle colonne della tabella di verità per $T_{\sigma PC}$ e $T_{\omega PC}$ (codificando semmai gli 0 e negando l'uscita in caso di un numero di "1" maggiore) e quindi il numero di livelli di porte OR scende da 2 a 1, portando il ritardo di $T_{\sigma PC}$ e $T_{\omega PC}$ a $2t_p$ e t a $15t_p$.

In questo caso, il microcodice sarebbe dunque il seguente:

1. (RDY1,RDY2,ACK3,LAST,OP11,OP12,OP21,OP22=00------) nop, 0
   // prima operazione da U1, U2 non richiede operazioni, si può mandare il risultato a U3
   // oppure U2 chiede operazioni, ma è stata l'ultima unità servita

(=101-00--,111100--) X->M[IND], set ACK1, reset RDY1,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0 -> LAST, 0
// seconda operazione da U1, U2 non richiede operazioni, si può mandare il risultato a U3
// oppure U2 chiede operazioni, ma è stata l'ultima unità servita
(=101-01--,=111101--) M[IND]->ESITO, set ACK1, reset RDY1,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0 -> LAST, 0
// terza operazione da U1, U2 non richiede operazioni, prepara ciclo e vai alla 1
// oppure U2 chiede operazioni, ma è stata l'ultima unità servita
(=10--10--,=11-110--) 1 -> I, uguale(M[0],X) -> U, 0->CHI, 1
// prima o seconda operazione da U1, nessuna op da U2 manca ACK per U3, attesa
// oppure U2 chiede operazioni, ma è stata l'ultima unità servita
(=100-0---,=11010---) nop, 0
// stesse frasi con operazoine richiesta solo da U2
// prima operazione da U2, U1 non richiede operazioni, si può mandare il risultato a U3
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=011---00,1110--00) X->M[IND], set ACK2, reset RDY2,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 1 -> LAST, 0
// seconda operazione da U2, U1 non richiede operazioni, si può mandare il risultato a U3
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=011---01,=1110--01) M[IND]->ESITO, set ACK2, reset RDY2,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 1 -> LAST, 0
// terza operazione da U2, U1 non richiede operazioni, prepara ciclo e vai alla 1
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=01----10,11-0--10) 1 -> I, uguale(M[0],X) -> U, 1->CHI, 2
// prima o seconda operazione da U2, nessuna op da U1 manca ACK per U3, attesa
// oppure U1 chiede operazioni ma è stata l'ultima unità servita
(=010---0-,1100--0-) nop, 0

2. *// trovato, manca ACK di U3*
(U,ACK3,I_0=10-) nop, 1
// trovato, termina operazione
(=11-) I->ESITO,
if(CHI==0) {set ACK1, reset RDY1} else {set ACK2, reset RDY2}, CHI -> LAST,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0
// non trovato, non a fine memoria, prova posizione successiva
(=0-0) I+1 -> I, uguale(M[I],X)->U, 1
// non trovato, esito negativo e termina operazione
(=011) –1 -> ESITO,
if(CHI==0) {set ACK1, reset RDY1} else {set ACK2, reset RDY2}, CHI -> LAST,
OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0

Utilizzando una notazione più in linea con quanto presente nelle soluzioni degli esercizi online, avremmo potuto scrivere

(set $ACK_{0,1}$, reset $RDY_{0,1}$)$|_{CHI}$

Versione 2: microcodice distinto per la precendenza.

In questo caso dovremmo scrivere un microcodice composto dalle microistruzioni della versione 1, replicate due volte:

- Un primo blocco, in cui la precedenza viene data comunque ad U1 e al termine dell'operazione si passa alla prima istruzione del secondo blocco, sia in caso di richieste singole che di richieste multiple.
- Un secondo blocco in cui la precendenza viene data comunque ad U2 e al termine dell'operazione si passa alla prima istruzione del primo blocco.

Questo comporta un aumento delle microistruzioni e delle frasi (entrambe raddoppiate). Utilizzando il controllo residuo per unificare la 1 e la 2, le istruzioni sono comunque 4 (2 bit di stato nella PC) ma il numero delle frasi sarebbe pari a 26 e dunque $T_{\sigma PC}$ e $T_{\omega PC}$ rimarrebbero a $3t_p$.

Osservazioni

Minimizzazione della durata del ciclo di clock

Nella soluzione è garantita dal non aver introdotto variabili di condizionamento complesse in presenza di μ-operazione "lunghe". Si sarebbero potuti eliminare i $2t_p$ del commutatore sugli ingressi indirizzo della memoria necessari a scegliere fra I, IND e 0. A tale scopo avremmo dovuto utilizzare una memoria "doppia porta" con 3 ingressi indirizzo distinti:

- IND che comanda un commutatore di lettura e il selettore di scrittura
- I e 0 che comandano altri due commutatori di lettura

Il costo (economico) è maggiore ma il costo di stabilizzazione della operazione uguale(M[I],X)->U (e quindi τ) diminuirebbe di t_p ($t = 13\ t_p$, considerando la versione con controllo residuo).

Una soluzione alternativa sarebbe stata quella nella quale in una μ-istruzione si inseriva, utilizzando un commutatore, il valore da utilizzare per l'indirizzamento della memoria in un registro e in una μ-istruzione successiva si accedeva alla memoria utilizzando il registro come indirizzamento. Tecnicamente questa soluzione garantisce la minimizzazione del ciclo di clock eliminando il peso del commutatore nella implementazione dell'operazione **uguale(M[I],X)->U** ma aumenta il numero di istruzioni eseguite, dunque il *k* da considerare per il calcolo del tempo medio di elaborazione.

Minimizzazione del tempo medio di elaborazione

Questa richiede anche la minimizzazione del numero di μ-istruzioni necessarie a completare le operazioni esterne. Nel nostro caso:

- Per le operazioni esterne di lettura e scrittura della memoria impieghiamo una sola μ-istruzione e quindi siamo già al minimo.
- Per le operazioni di ricerca, impieghiamo una microistruzione per iniziare il ciclo e al massimo N (1K) istruzioni per la ricerca. Avremmo potuto eliminare l'istruzione di inizializzazione testando

direttamente nelle variabili di condizionamento se uguale(M[0],X) e, in caso, restituendo subito l'esito a Ui. Se i due valori risultavano diversi si sarebbe potuto procedere con l'istruzione che cicla sull'intera memoria:

1. (RDY1,RDY2,ACK3, LAST,OP11,OP12,OP21,OP22,uguale(M[0],X)=00-------) nop, 0
   …
   (=101010--1) 0->ESITO, set ACK1, reset RDY1, OP->OPOUT, IND->INDOUT, X->XOUT, set RDY3, reset ACK3, 0 -> LAST, 0
   (=101010--0) uguale(M[1],X)->U, …

ma in questo caso avremmo allungato il ciclo di clock, visto che uguale(…) sarebbe comparso sia nella varibile di condizionamento che in una delle μ-operazioni della stessa μ-istruzione.

Tempo medio di elaborazione

Con la prima soluzione proposta abbiamo:

# μ-istruzioni	*probabilità*
$k_1 = 1$	$p_1 = 1/3$
$k_2 = 1$	$p_2 = 1/3$
$k_3 = 1 + N/2$ (assumendo di trovare il valore cercato a metà memoria)	$p_3 = 1/3$

Dunque il tempo medio di elaborazione sarà

$$T = \tau * \Sigma (k_i * p_i) = \tau * (1/3 + 1/3 + 1/3 (1 + N/2)) = 1 + N/6$$