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 Xi
- un registro da 32 bit in uscita ESITOi (in complemento a due, quando usato per la terza operazione)
- 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 INDOUT 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, che però comporta un aumento delle frasi del μ-programma e dunque un aumento di complessità della PC.

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. *// se non ci sono richieste di operazione da U1 e U2, si rimane in attesa con una nop*
   (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 anche U2 chiede operazioni, ma è stata l'ultima unità servita e quindi la precedenza
// è alla U1

(=101-00--,111100--) X1->M[IND1], set ACK1, reset RDY1,
OP1->OPOUT, IND1->INDOUT, X1->XOUT, 0->UNITA, 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[IND1]->ESITO1, set ACK1, reset RDY1,
OP1->OPOUT, IND1->INDOUT, X1->XOUT, 0->UNITA, 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],X1) -> 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) X2->M[IND2], set ACK2, reset RDY2,
OP2->OPOUT, IND2->INDOUT, X2->XOUT, 1->UNITA, 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[IND2]->ESITO2, set ACK2, reset RDY2,
OP2->OPOUT, IND2->INDOUT, X2->XOUT, 1->UNITA, 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],X2) -> 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->ESITO1, set ACK1, reset RDY1, 0 -> LAST,
   OP1->OPOUT, IND1->INDOUT, X1->XOUT, 0->UNITA, set RDY3, reset ACK3, 0
   // non trovato, non a fine memoria, prova posizione successiva
   (=0-0) I+1 -> I, uguale(M[I],X1)->U, 1
   // non trovato, esito negativo e termina operazione
   (=011) -1 -> ESITO1, set ACK1, reset RDY1, 0 -> LAST,
   OP1->OPOUT, IND1->INDOUT, X1->XOUT, 0->UNITA, set RDY3, reset ACK3, 0
3. // trovato, manca ACK di U3
   (U,ACK3,I_0=10-) nop, 1

// trovato, termina operazione

(=11-) I->ESITO2, set ACK2, reset RDY2, 1 -> LAST,
OP2->OPOUT, IND2->INDOUT, X2->XOUT, 1->UNITA, set RDY3, reset ACK3, 0

// non trovato, non a fine memoria, prova posizione successiva

(=0-0) I+1 -> I, uguale(M[I],X2)->U, 1

// non trovato, esito negativo e termina operazione

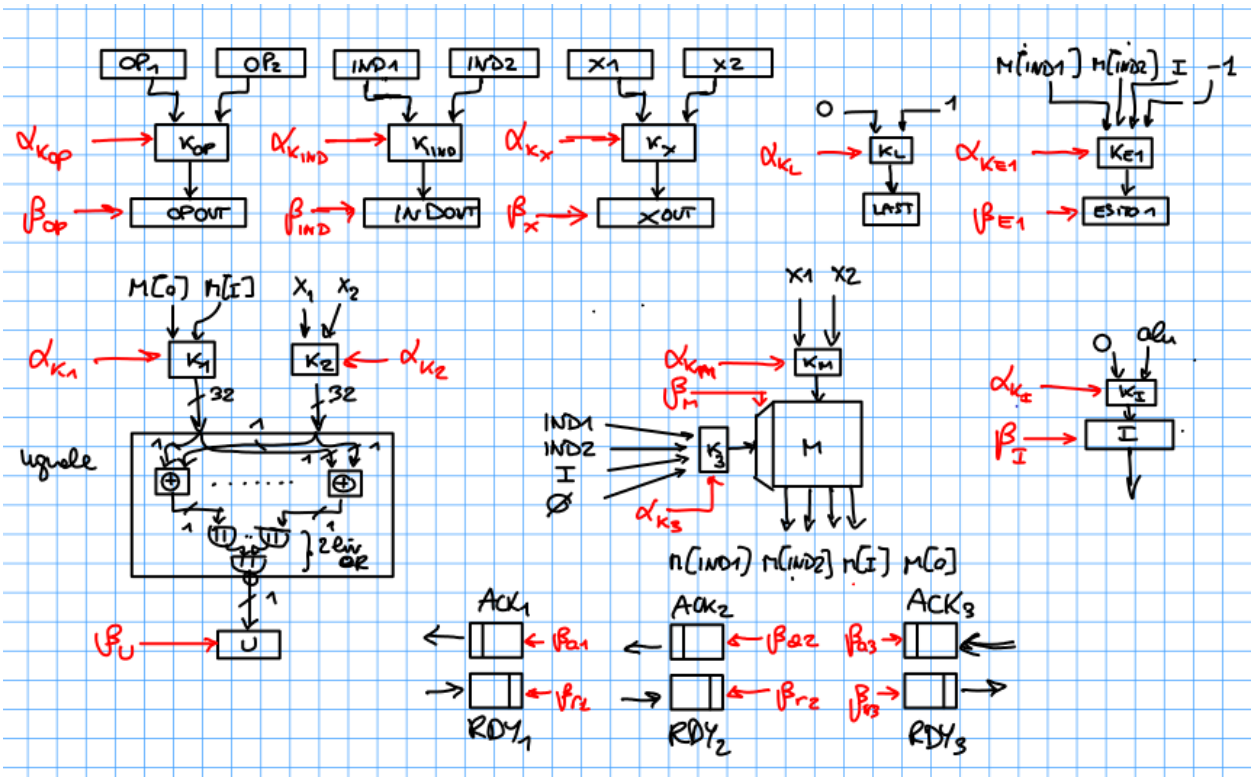
(=011) -1 -> ESITO2, set ACK2, reset RDY2, 1 -> LAST,
OP2->OPOUT, IND2->INDOUT, X2->XOUT, 1->UNITA, set RDY3, reset ACK3, 0

Tempi della PC

Per la PC abbiamo 3 microistruzioni, 17 frasi, 11 variabili di condizionamento di cui al più 6 testate contemporaneamente. 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.

Tempi della PO

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):

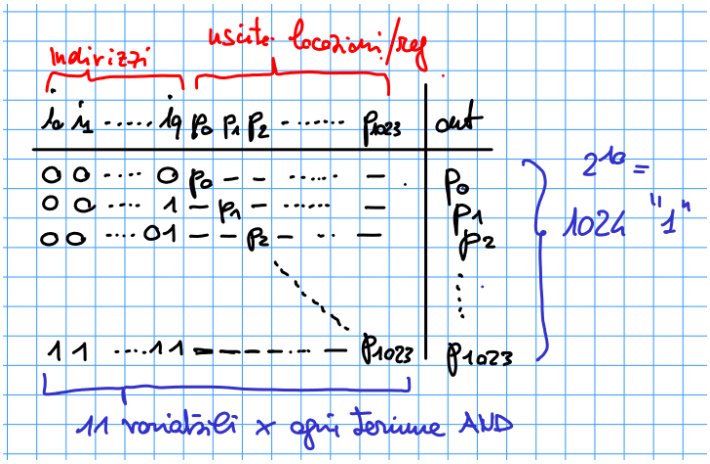


OPOUT, INDOUT e XOUT devono essere intesi come risultato della scelta fra OP1 e OP2, IND1 e IND2 e X1 e X2, quindi avranno un commutatore in ingresso. Il cammino più lungo fra l'uscita di un

registro/memoria e l'ingresso di un registro/memoria va cercato fra quello che serve a calcolare uguale(M[...], ...) e quello che calcola I+1->I, visto che la ALU richiede un tempo di stabilizzazione piuttosto lungo.

Calcolo del tempo di accesso alla memoria

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. La tabella di verità per il commutatore di lettura è infatti:



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, $\log_8 1024 = \log_2 1024 / \log_2 8 = 10 / 3 = 3.3$). Quindi in totale il tempo di accesso sarà di $6t_p$.

Il microprogramma è stato scritto in modo da minimizzare la durata del ciclo di clock.

- 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 4 diversi valori, 0, IND1, IND2 e I, dunque serve un commutatore sull'ingresso indirizzi. Il tempo di accesso è di $6t_p$, come appena calcolato. L'operazione più lunga sulla memoria sembra quella specificata come *uguale(M[0],Xi)->U* o *uguale(M[I],Xi)->U*. In questo caso, il cammino di cui va valutata la stabilizzazione è quello che legge la memoria, passa dai commutatori di ingresso per la rete uguale e termina con la scrittura in U, che richiede un tempo pari a $t_a + t_k + t_{uguale} = 2t_p + 6t_p + 2t_p + 4t_p = 14t_p$ Tuttavia va considerato che la I+1->I richiede un talu e un t_k (commutatore davanti a I) per un totale di $17t_p$, maggiore del $14t_p$ per l'altra operazione. Dunque dovremo considerare il massimo $T_{\sigma PO}$ pari a $17t_p$.

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

$$\tau = T_{\omega PC} + T_{\sigma PO} + \delta = 3t_p + 17t_p + t_p = 21t_p$$

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 OPi, INDi, Xi, 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, per gli accessi a OPi, INDi e Xi, 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 $20t_p$.

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) e dunque un aumento della complessità della PC.

Versione 3: uso di controllo residuo su TURNO

Organizziamo il codice in modo da utilizzare una variabile T (da 1 bit, numero dell'unità che deve essere servita) per indicare a chi tocca essere servito. Si spende un ciclo per l'arbitraggio, poi il resto del codice si semplifica, utilizzando il controllo residuo per accedere i registri giusti e le interfacce giuste.

1. (RDY1,RDY2,T=00-) nop
   (10-) 0->T, 1
   (01-) 1->T, 1
   (110) 1->T, 1
   (111) 0->T, 1
2. (OP(T),ACK3=0-0) nop // aspetta se non puoi mandare il report a U3
   // prima operazione
   (=001) X(T)->M(IND(T)), set RDY(T), reset ACK(T), X(T)->XOUT, IND(T)->INDOUT, OP(T)->OPOUT, T->UNITA, set RDY3, reset ACK3, 0
   // seconda operazione
   (=011) M(IND(T))->ESITO(T), set RDY(T), reset ACK(T), M(IND(T))->XOUT, IND(T)->INDOUT, OP(T)->OPOUT, T->UNITA, set RDY3, reset ACK3, 0

// terza operazione, si può partire anche senza ACK da U3
(=1--) 1->I, X(T) -> XC, uguale(M[0], X(T))->U, 2

3. (U,I0,ACK3=00-) I+1->I, uguale(M[I], XC)->U, 2 // non trovato, si continua
   // trovato, si comunica e si ritorna alla 0.
   (=101,111) I-1->ESITO(T), set RDY(T), reset ACK(T), I-1->INDOUT, XC -> XOUT, T->UNITA, OP(T)->OPOUT, set RDY3, reset ACK3, 0
   // se non c'è ACK da U3 si aspetta
   (=100) nop, 2
   // si arriva in fondo senza trovare l'elemento
   (=101) -1->ESITO(T), set RDY(T), reset ACK(T), -1->INDOUT, XC -> XOUT, T->UNITA, OP(T)->OPOUT, set RDY3, reset ACK3, 0

In questo caso, tutti gli accessi in lettura tipo X(T) sono realizzati utilizzando l'uscita di un commutatore che ha come ingressi X_1 e X_2 e come ingresso di controllo il registro T (costo $t_k = 2t_p$). Gli accessi in scrittura sono effettuati inviando il β di scrittura dalla PC a X_1 o X_2 mediante un selettore comandato da T.

La parte controllo ha un massimo di tre variabili di condizionamento testate contemporaneamente, che con i 2 bit del registro di stato (3 m-istruzioni) fanno 5 ingressi per i livelli AND di $T_{\omega PC}$ e $T_{\sigma PC}$. Il μ-codice ha 13 frasi, quindi anche gli ingressi del livello OR possono essere ridotti a otto o meno. Questo comporta un ritardo per entrambe le $T_{\omega PC}$ e $T_{\sigma PC}$ di $2t_p$.

Il $T_{\omega PO}$ è non nullo nella 1. (t_k) dove il massimo $T_{\sigma PO}$, per X(T)->M(IND(T)), è t_k (per X(T) e IND(T) in parallelo) + t_k + t_a (per ingresso M e scrittura), dunque

$$\tau = T_{\omega PO} + \max\{T_{\sigma PC}, T_{\omega PC} + T_{\sigma PO}\} + \delta = 2t_k + \max\{2t_p, 2t_p + 2t_k + t_a\} + t_p = 17t_p$$

Nelle altre microistruzioni il $T_{\omega PO}$ è nullo. La massima $T_{\sigma PO}$ è nella prima frase della 2. dove dobbiamo considerare il massimo fra il tempo richiesto per calcolare I+1->I e uguale(M[I], XC)->U ovvero fra $t_{alu}+t_k$ ($=17t_p$) e $t_k + t_a + t_k + t_{uguale}$ ($=14t_p$), cioè $17t_p$. Dunque in questo caso:

$$\tau = T_{\omega PO} + \max\{T_{\sigma PC}, T_{\omega PC} + T_{\sigma PO}\} + \delta = 0 + \max\{2t_p, 2t_p + 17t_p\} + t_p = 20t_p$$

Il ciclo di clock dovrà essere scelto come il massimo fra i due valori, cioè pari a

$$\tau = 20t_p$$

Da notare

La 0. e la 1. Devono essere separate perchè nella 1. vogliamo utilizzare il valore del registro T per il controllo residuo. Tuttavia, qualunque funzione dello stato interno è valida come valore da utilizzare per il controllo residuo nella produzione di variabili di condizionamento (ovviamente il caso critico è OP(T) nelle variabili di condizionamento della 1.). Dunque si potrebbe pensare di realizzare un componente che calcoli T a partire da RDY1, RDY2 e T, cioè che produca i valori scritti in T nella 0. La funzione avrà la tabella di verità:

RDY1	RDY2	T		T'

1	0	-		0
0	1	-		1
1	1	0		1
1	1	1		0
0	0	0		0
0	0	1		1

e dunque avremo T' = f(RDY$_1$, RDY$_2$, T) = not(RDY$_1$) RDY$_2$ + RDY$_1$ RDY$_2$ not(T) + not(RDY$_1$) not(RDY$_2$) T = not(RDY$_1$) T + not(RDY$_2$) T . Il ritardo introdotto dal calcolo della f è pari a 2t$_p$.

Il µ-codice lo possiamo dunque riscrivere come segue:

1. (OP(f(RDY1,RDY2,T)),ACK3=0-0) nop // aspetta se non puoi mandare il report a U3
   // prima operazione
   (=001) X(f(RDY1,RDY2,T))->M(IND(f(RDY1,RDY2,T))), set RDY(T f(RDY1,RDY2,T)), reset ACK(f(RDY1,RDY2,T)), X(f(RDY1,RDY2,T))->XOUT, IND(f(RDY1,RDY2,T))->INDOUT, OP(f(RDY1,RDY2,T))->OPOUT, f(RDY1,RDY2,T)->UNITA, f(RDY1,RDY2,T)->T, set RDY3, reset ACK3, 0
   // seconda operazione
   (=011) M(IND(f(RDY1,RDY2,T)))->ESITO(f(RDY1,RDY2,T)), set RDY(f(RDY1,RDY2,T)), reset ACK(f(RDY1,RDY2,T)), M(IND(f(RDY1,RDY2,T)))->XOUT, IND(f(RDY1,RDY2,T))->INDOUT, OP(f(RDY1,RDY2,T))->OPOUT, f(RDY1,RDY2,T)->UNITA, f(RDY1,RDY2,T)->T, set RDY3, reset ACK3, 0
   // terza operazione, si può partire anche senza ACK da U3
   (=1--) 1->I, X(f(RDY1,RDY2,T)) -> XC, uguale(M[0], X(f(RDY1,RDY2,T)))->U, f(RDY1,RDY2,T)->T, 1
2. (U,I0,ACK3=00-) I+1->I, uguale(M[I], XC)->U, 2 // non trovato, si continua
   // trovato, si comunica e si ritorna alla 0.
   (=101,111) I-1->ESITO(T), set RDY(T), reset ACK(T), I-1->INDOUT, XC -> XOUT, T->UNITA, OP(T)->OPOUT, set RDY3, reset ACK3, 0
   // se non c'è ACK da U3 si aspetta
   (=100) nop, 2
   // si arriva in fondo senza trovare l'elemento
   (=101) -1->ESITO(T), set RDY(T), reset ACK(T), -1->INDOUT, XC -> XOUT, T->UNITA, OP(T)->OPOUT, set RDY3, reset ACK3, 0

Adesso testare OP(f(RDY1,RDY2,T)) costerà 2t$_p$ per il calcolo di f piuttosto che per il t$_k$ del commutatore pilotato da T, ma il punto è che il costo non cambia. Per contro, la PC avrà un solo bit nel registro di stato. Anche se questo non diminuisce il tempo di stabilizzazione delle $T_{\omega PC}$ e $T_{\sigma PC}$, riduce comunque la complessità di tali reti, che si trovano ad avere un bit di ingresso in meno rispetto alla versione precedente.

Osservazioni

Fanno tutte riferimento alla prima versione del microcodice.

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". Con l'utilizzo della rete combinatoria ad hoc per il calcolo di uguale(X,Y) si minimizza in $T_{\omega PO}$.

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. Per esempio, si sarebbero potute dividere le frasi che contengono la uguale(..., ...) in due istruzioni:

da: 0. ....
(=10--10--,=11-110--) 1 -> I, uguale(M[0],X1) -> U, 1

a: 0. (=10--10--,=11-10--) 1 -> I, M[0]->MEMOUT, 5
5. uguale(MEMOUT,X1) -> U, 1

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

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