Architetture degli Elaboratori–Prima prova di verifica intermedia
bozza di soluzione

19 dicembre 2012

1 Domanda 1

Si possono considerare due soluzioni alternative:

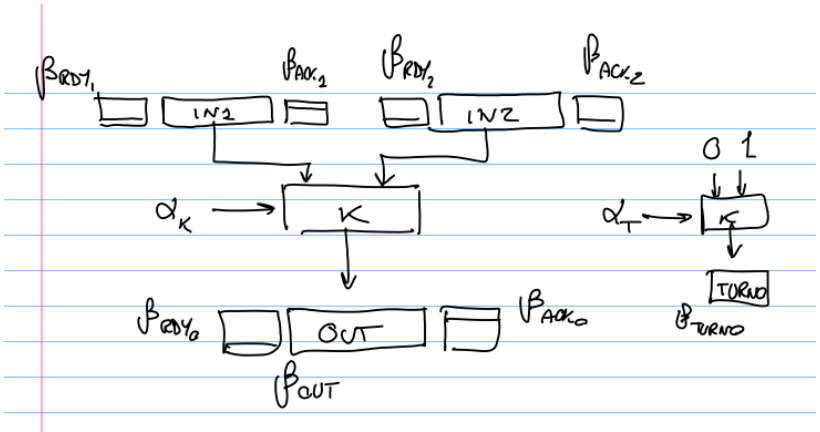
- utilizzare un registro da 1 bit TURNO per gestire la priorità, oppure
- utilizzare due microistruzioni diverse, una che dà priorità all'unità U_1 e l'altra che la dà invece ad U_2 quando siano presenti dati in ingresso da entrambe le unità.

1.1 Prima soluzione: variabile TURNO

Nel primo caso, il microprogramma potrebbe essere scritto come segue:

0. $(RDY_1, RDY_2, ACK_{out}, TURNO = 00--)nop, 0$
   $(= 101-)IN_1 \rightarrow OUT, reset\ RDY_1, set\ ACK_1, reset\ ACK_{out}, set\ RDY_{out}, TURNO = 1, 0$
   $(= 011-)IN_2 \rightarrow OUT, reset\ RDY_2, set\ ACK_2, reset\ ACK_{out}, set\ RDY_{out}, TURNO = 0, 0$
   $(= 1110)IN_1 \rightarrow OUT, reset\ RDY_1, set\ ACK_1, reset\ ACK_{out}, set\ RDY_{out}, TURNO = 1, 0$
   $(= 1111)IN_2 \rightarrow OUT, reset\ RDY_2, set\ ACK_2, reset\ ACK_{out}, set\ RDY_{out}, TURNO = 0, 0$
   $(= --0-)nop, 0$

In questo caso, possiamo realizzare l'unità come un'unica rete sequenziale. La struttura dalla parte operativa sarà la seguente: i vari β e α saranno calcolati a partire da $RDY_1, RDY_2, ACK_{out}, TURNO$ direttamente nella PO.



Tenendo conto del fatto che quando si effettua un invio verso U_3 occorre sempre avere

$$\beta_{out} = \beta_{ACK_{out}} = \beta_{RDY_{out}} = 1$$

e che quando si invia il dato ricevuto da U_i occorre avere

$$\beta_{RDY_i} = \beta_{ACK_i} = 1$$

possiamo assumere di avere una rete che calcola

$$\beta_{out} = \overline{\overline{RDY_1 RDY_2} + \overline{ACK_{out}}}$$

e una rete che calcola:

$$\alpha_K = \overline{RDY_1 RDY_2} ACK_{out} + RDY_1 RDY_2 ACK_{out} TURNO$$

che può essere quindi utilizzata anche per i β della prima e della seconda interfaccia di ingresso:

$$\beta_{setRDY_1} = \beta_{resetACK_1} = \overline{\alpha_K}$$

$$\beta_{setRDY_2} = \beta_{resetACK_2} = \alpha_K$$

Il ciclo di clock in questo caso deve essere calcolato come il tempo necessario per scrivere il valore corretto in OUT, quindi come tempo necessario per calcolare α_K ($2t_p$) e per stabilizzare K (altri $2t_P$), per un totale di

$$\tau = 4t_p + \delta = 5t_p$$

Di fatto questo costituisce il tempo necessario per il calcolo del prossimo stato interno (oltre al tempo necessario per l'impulso di clock). Non va considerato il tempo per il calcolo delle uscite verso la PC dal momento che non c'è una PC in questo caso.

1.2 Seconda soluzione: microistruzioni diverse per priorità diverse

La soluzione alternativa invece portava ad un microcodice con 2 microistruzioni:

0. $(RDY_1, RDY_2, ACK_{out} = 00-)nop, 0$
   $(= 1-0, 010)nop, 0$
   $(= 1-1)IN_1 \rightarrow OUT, reset\ RDY_1, set\ ACK_1, reset\ ACK_{out}, set\ RDY_{out}, 1$
   $(= 011)IN_2 \rightarrow OUT, reset\ RDY_2, set\ ACK_2, reset\ ACK_{out}, set\ RDY_{out}, 0$
1. $(RDY_1, RDY_2, ACK_{out} = 00-, -10, 100)nop, 1,$
   $(= -11)IN_2 \rightarrow OUT, reset\ RDY_2, set\ ACK_2, reset\ ACK_{out}, set\ RDY_{out}, 0$
   $(= 101)IN_1 \rightarrow OUT, reset\ RDY_1, set\ ACK_1, reset\ ACK_{out}, set\ RDY_{out}, 1$

In questo caso, la prima microistruzione dà priorità ai dati dalla unità U_1 e la seconda ai dati dalla U_2. La parte operativa è la stessa del caso precedente (tranne il registro TURNO) e avremo $T_{\omega PO} = 0$ (le variabili di condzionamento sono solo sincronizzatori) e $T_{\sigma PO} = 2t_p$ (tempo di stabilizzazione di un commutatore da 2 ingressi).

Per la parte controllo abbiamo 2 stati interni (1 bit), 3 variabili di condizionamento (e quindi un livello AND con porte da al più 4 ingressi) e 7 frasi (quindi livello OR con porte da al più 7 ingressi) il che permette di concludere che $T_{\omega PC} = T_{\sigma PC} = 2t_p$.

Il ciclo di clock quindi vale

$$\tau = 0 + max\{2t_p, 2t_p + 2t_p\} + t_p = 5t_p$$

2 Domanda 2

Si assume che i parametri vengano passati mediante un'area di memoria di tre celle consecutive, all'indirizzo IND, tali che:

- $MV[IND]$ contenga il valore X da ricercare nell'array
- $MV[IND+1]$ contenga l'indirizzo della prima posizione dell'array
- $MV[IND+2]$ contenga il numero di posizioni dell'array

Assumiamo anche che l'indirizzo dei parametri in memoria venga passato tramite `Rparam`, che il risultato sia restituito in Rres, e che per l'indirizzo di ritorno della procedura si usi Rret.

Lo pseudo codice della procedura è il seguente:

```
res = -1;
for(i=0; i<N; i++)
  if(A[i]==X) res=i;
return(res);
```

che corrisponde al codice D-RISC:

```
cerca: SUB R0, #1, Rres        ; prepara il valore di ritorno non trovato
       CLEAR Ri                ; registro indice per scorrere l'array
       LOAD Rparam, #0, Rx     ; accesso ai parametri della procedura
       LOAD Rparam, #1, Rbasea
       LOAD Rparam, #2, Rn
loop:  LOAD Rbasea, Ri, Rai    ; ciclo sugli elementi dell'array
       IF!= Rai, Rx, cont
       ADD  Ri, R0, Rres       ; compilazione ramo then
cont:  INC Ri                  ; compilazione fine ciclo
       IF< Ri, Rn, loop
       GOTO Rret               ; ritorno al chiamante
```

Si potrebbe ovviamente interrompere il ciclo appena trovato il valore:

```
for(...)
  if(  ) {
    res = i;
    break;
  }
```

In questo caso è sufficiente modificare la compilazione del ramo then come segue:

```
cerca: SUB R0, #1, Rres
       CLEAR Ri
       LOAD Rparam, #0, Rx
       LOAD Rparam, #1, Rbasea
       LOAD Rparam, #2, Rn
loop:  LOAD Rbasea, Ri, Rai
       IF!= Rai, Rx, cont
       ADD  Ri, R0, Rres
       GOTO Rret
cont:  INC Ri
       IF< Ri, Rn, loop
       GOTO Rret
```

Un esempio di chiamata della procedura per cercare Y in B è il seguente:

```
...
STORE Rparam, R0, Ry
STORE Rparam, #1, Rbaseb
STORE Rparam, #2, Rlenb
CALL  Rcerca, Rret
...
```

La procedura esegue:

- 2 aritmetico-logiche corte e 3 LOAD nell'inizializzazione
- 1 LOAD, una aritmetico-logica corta e due IF per ognuna delle iterazioni con test falso
- 1 LOAD, 1 IF, una aritmetico logica corta e una GOTO per l'iterazione che trova il valore cercato, oppure
- 1 LOAD, 2 IF, una aritmetico logica corta e un GOTO per la fine ciclo se il valore non è presente.

Assumendo che il valore sia presente nell'array e che si trovi (mediamente) a metà array, possiamo stimare il numero di istruzioni eseguite come:

- 2 ALC + 3 LOAD per l'inizializzazione
- (1 LOAD + 1 ALC + 2 IF) * N/2 iterazioni con test falso
- 1 LOAD, 1 IF, 1 ALC e 1 GOTO per la fine ciclo

per un totale di

- 3 ALC + 4 LOAD + 1 IF + 1 GOTO + N/2 * (1 LOAD + 1 ALC + 2 IF)

Fattorizzando le ch0. e ch1. e ricordando i tempi impiegati per le esecuzioni delle varie classi di istruzioni:

$$T_C = (9 + \frac{N}{2}4)(T_{ch0} + T_{ch1}) + \frac{N}{2}(T_{execLOAD} + T_{execADD} + 2T_{execIF})$$

$$T_C = (9 + 2N)(2\tau + t_a) + \frac{N}{2}(\tau + t_a + \tau + 2\tau) = (9 + 2N)(2\tau + t_a) + \frac{N}{2}(4\tau + t_a)$$

Per N grande possiamo semplificare il tempo di esecuzione della procedura con

$$T_C = 2N(2\tau + t_a) + \frac{N}{2}(4\tau + t_a) = N(6\tau + \frac{5}{2}t_a)$$

3 Domanda 3

La parte controllo è una rete di Mealy, quindi sia σ_{PC} che ω_{PC} hanno in ingresso sia le variabili di condizionamento che lo stato interno. Quindi entrambe le reti hanno

$$n = \lceil log_2(m) \rceil + k$$

ingressi. Questo implica porte AND con n ingressi, e quindi la necessità, qualora $n > 8$ di

$$\lceil log_8(n) \rceil = \lceil log_8(\lceil log_2(m) \rceil + k) \rceil$$

livelli di porte da max 8 ingressi. Qualunque funzione booleana di n ingressi può essere definita mediante una tabella di verità di 2^n righe. Poichè l'espressione boolena della funzione contiene un termine in OR per

ognuno degli 1 presenti nella colonna che definisce l'uscita, avremo un massimo di 2^n termini in OR. Dunque il livello OR potrebbe essere realizzato con porte da max 8 ingressi con un albero di profondità

$$\lceil log_8(2^n) \rceil$$

Trasformando il logaritmo in base 8 in base 2 avremmo quindi che il numero di livelli dell'albero OR è pari a

$$\lceil log_8(2^n) \rceil = \lceil \frac{log_2(2^n)}{log_2(8)} \rceil = \lceil \frac{log_2(2^n)}{log_2(2^3)} \rceil = \lceil \frac{n}{3} \rceil$$

Nel nostro caso abbiamo che $n = \lceil log_2(m) \rceil + k$ e dunque il numero dei livelli di porte OR sarà

$$\lceil \frac{\lceil log_2(m) \rceil + k}{3} \rceil$$

Complessivamente, dunque, potremmo dire che per la stabilizzazione di ω_{PC} e σ_{PC} occorrono

$$(\lceil \frac{\lceil log_2(m) \rceil + k}{3} \rceil + \lceil log_8(n) \rceil) t_p$$

Nel caso in cui si sappia che le variabili di condizionamento vengono testate al massimo 4 alla volta, possiamo dedurre che fino a un massimo di 2^4 microistruzioni (quindi 4 bit per la codifica dello stato interno), basta un livello AND, invece dei $\lceil \frac{\lceil log_2(m) \rceil + k}{3} \rceil$ di prima.