Una unità firmware U contiene una memoria M da 1K parole di 32 bit, in cui è memorizzata una lista di elementi a due campi: il primo (memorizzato in indirizzi pari) contiene un numero intero, il secondo (memorizzato all'indirizzo successivo, quindi dispari), contiene il puntatore all'elemento successivo della lista (se è l'ultimo elemento, allora contiene -1). U implementa due operazioni esterne, "sposta" ed "elimina", che vengono richieste da una unità esterna W secondo un protocollo a domanda-risposta.
Per la prima operazione ("sposta") U riceve un valore X da W e ricerca nella lista un elemento il cui primo campo è uguale ad X. Se lo trova, lo sposta in fondo alla lista e restituisce a W "trovato", altrimenti restituisce "non trovato". Per la seconda operazione ("elimina") U riceve da W la richiesta di eliminazione: se la lista non è vuota, elimina il primo elemento della lista e restituisce "eliminato", altrimenti restituisce "non eliminato". Si consideri l'esistenza di due registri (START e END) che indicano inizialmente l'indirizzo del primo e dell'ultimo elemento della lista. I registri vanno mantenuti aggiornati durante lo svolgimento delle operazioni esterne.
Si fornisca un'implementazione di U che minimizza la durata del ciclo di clock, motivando adeguatamente la risposta. Si hanno a disposizione porte logiche a 8 ingressi ed ALU che operano in un tempo talu = 6 tp.

Algoritmo e risorse

L'interfaccia della unità U verso W comprende:

• in **ingresso**: un **RDY**, un registro **OP** da un bit **e** un registro da 32 bit **X**

• in **uscita**: un **ESITO** da 1 bit (0 -> esito negativo, 1 -> esito positivo), un **ACK** .

Utilizziamo un registro **START** e un registro **END**, come suggerito dal testo, per mantenere un puntatore alla testa ed alla coda della lista.

Per minimizzare il ciclo di clock, cerchiamo di **non utilizzare variabili di condizionamento complesse** e di **non utilizzare micro-operazioni che richiedano risorse in cascata** (per esempio due ALU in cascata).

Per la **prima operazione** dovremmo innanzitutto controllare se la lista è **vuota o** se l'elemento cercato è il **primo** della lista. Entrambi i casi richiedono un **trattamento particolare**, che consiste semplicemente nell'**invio dell'esito** (negativo nel primo caso e positivo nel secondo). Per la **seconda** operazione l'unico caso particolare sarà quello di **lista vuota**, che richiede un **esito negativo**.

Per **l'accesso** ad un elemento della lista dovremmo indirizzare **due posizioni consecutive della memoria** (per esempio START e START+1). **Evitiamo** l'utilizzo di una **ALU per** calcolare la **START+1** utilizzando, per tale valore, i **31 bit più significativi di START concatenati con un singolo bit a 1**, facendo leva sul fatto che sappiamo che l'inizio dell'elemento della lista è sempre a una posizione pari.

Consideriamo di utilizzare una memoria a **doppia porta** (un selettore per la scrittura e due commutatori per la lettura, quindi in grado di **leggere due** posizioni distinte e **scrivere una** delle due nello stesso ciclo di clock). Questa caratteristica è sfruttata nella seconda frase della terza microistruzione (leggiamo la posizione START+1 e scriviamo la posizione END+1), nella quinta microistruzione (leggiamo le posizioni CURR e CURR+1) e nella terza frase della microistruzione 6 (leggiamo la posizione CURR+1 e scriviamo PREV+1). Occorre dunque anche un **commutatore** sugli **ingressi degli indirizzi**, che preveda come ingressi START, START+1, END+1, CURR, CURR+1, PREV, TEMP, distribuiti fra i due ingressi indirizzo disponibili (**ingresso 1**, che comanda il **selettore per il** β e **uno dei due commutatori di lettura**, e **ingresso 2** che comanda **solo uno** dei due commutatori di **lettura**) in modo da permettere come **operazioni concorrenti (simultanee)**:

• M[START+1]-> … e … -> M[END+1] (**lettura e scrittura**)
• M[CURR] -> … e M[CURR+1] -> … (**doppia lettura**)
• M[CURR+1] -> … e … -> M[PREV+1], (**lettura e scrittura**)

Il **tempo del commutatore** va ad **aggiungersi** al tempo di **accesso** della memoria (che per 1K posizioni sarebbe 6tp) **portando** la lettura (o scrittura) della memoria **ta = 8tp**.

Il **massimo TσPO** è un **ta+talu+tk** (considerando che gli indirizzi dispari siano ottenuti concatenando un bit 1 al posto del bit meno significativo dell'indirizzo pari). Possiamo **diminuire ancora** questo TσPO **spezzando gli accessi in memoria e il calcolo con la ALU**.

E quindi il **microcodice** definitivo diventa il seguente:

1. (RDY, OP = 0-) nop, 1. // se non ho richieste attendo
(=10) eq(START, -1) -> V, M[START]->MSTART, 2.
// prima op, controllo se vuota o se = 1o elem
(=11) eq(START, -1) -> V, 3 // seconda op, controllo se lista vuota
2. eq(MSTART,X) -> T, 4 // solo per evitare ta+talu in cascata
3. (V = 0) M[START+1] -> START, 1->ESITO, reset RDY, set ACK, 1.
// non vuota, elimino primo elem
(=1) 0 -> ESITO, reset RDY, set ACK, 1. // vuota, esito negativo
4. (V, T = 1-) 0 -> ESITO, reset RDY, set ACK, 1. // lista vuota, non trovato
(= 01) M[START+1] -> START, START +1 -> TEMP, START -> M[END+1], 5
// non vuota, è il primo elemento, aggiorno START a next il primo
//elemento corrente diventa l'ultimo, ricordati l'ultimo elmento (evito 2 write contemporanee in M)
(= 00) START -> PREV, M[START+1] -> CURR, 6.
// lista non vuota è il primo elemento
5. -1 -> M[TEMP], 1 -> ESITO, reset RDY, set ACK, 1.
// finisco di sistemare l'ultimo elem della lista
6. M[CURR+1] -> MCURR1, M[CURR] -> MCURR, 7
7. eq(MCURR1, -1) -> V, eq(MCURR, X) -> T, 8.
// controllo se trovato o fine lista
8. (V, T = 1 0) 0 -> ESITO, reset RDY, set ACK, 1. // non trovato, lista finita
(=00) CURR -> PREV, M[CURR+1] -> CURR, 6.
// non trovato, considero il prossimo elem

(=01) M[CURR+1] -> M[PREV+1], 9. // trovato, lo metto in fondo

9. CURR -> M[END+1], CURR+1 -> TEMP, 5.
// e finisco di sistemare la fine della lista

In questo modo, per ognuna delle frasi abbiamo al massimo

TωPO (cioè 0) + max { TωPC + max { talu+tk, ta+tk } , TσPC } + δ

Abbiamo **9 microistruzioni**, **4 variabili di condizionamento** (2 testate contemporaneamente) e **16 frasi**. **4 bit di stato e 2 bit** testati come variabili di condizionamento fanno sì che il numero di livelli di porte **AND** nella ωPC e σPC sia **uno** solo. Le **16** frasi fanno sì che avremo un **massimo di 8 "1"** (altrimenti codifichiamo gli **zeri e neghiamo** l'uscita) e quindi avremo anche **un** solo livello di porte **OR**. Il ritardo della **ωPC e σPC sarà 2 tp.**

Il ciclo di clock sarà dunque pari a

(2 + max {max{6+2, 8+2}, 2} +1) tp = 13tp.

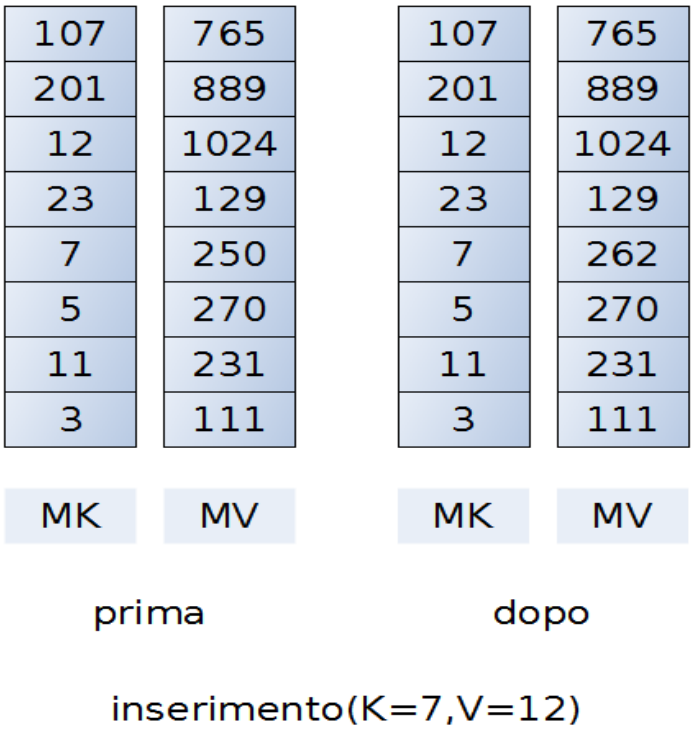
Meno di questo è **impossibile**, visto che servono **almeno 2tp** per la **ωPC** (e questo rappresenta il minimo ritardo possibile per ωPC e σPC), **8 tp** per l'accesso in **memoria** e **2tp** per il **commutatore** per scrivere nei registri, che hanno input diversi, oltre al **tp** per il δ.

Alternativamente, avremmo potuto considerare che il **test** per controllare se un valore è **-1** potrebbe semplicemente **controllare il bit più significativo**: se è 1 allora il numero è negativo. Siccome i puntatori o sono indirizzi della memoria M (numeri positivi) o –1 (unico valore ammesso negativo), il bit ci dice se il valore è effettivamente –1.

Valgono le stesse considerazioni fatte per il codice precedente. Per esempio, la 3 si può spezzare in una microistruzione che mette M[CURR]->TEMP e una che fa eq(TEMP,X)->T, M[CURR+1]->NEXT in modo da non sommare il tempo di accesso alla memoria al tempo della ALU che calcola eq. Similmente possiamo spezzare la 4a frase della 1. Con 7 microistruzioni (3 bit di stato) e max 2 variabili di condizionamento testate contemporaneamente abbiamo un livello AND nella TωPC e nella TσPC. Il numero di livello OR dipende dal numero delle frasi, che sarebbero 16. Per le stesse considerazioni fatte precedentemente si può assumere che basti un solo livello or nella TωPC e nella TσPC. Dunque TωPC = TσPC = 2tp. Per la σPO

la frase più lunga sarà della stessa durata di quella che avevamo nel codice precedente. **Valgono quindi anche le stesse considerazioni sulla minimalità del τ.**

Una unità U implementa una memoria di coppie *<chiave, valore>*. Una unità U1 può richiedere l'inserimento di una coppia *<K, V>*. U cerca fra le proprie coppie e, quando trova la coppia con chiave pari a *K*, aggiorna il valore *Va* presente per quella chiave, con *Va+V*. Si assume che non possano essere effettuate richieste per chiavi non esistenti. Due unità U2 ed U3 possono richiedere la lettura del valore relativo alla chiave *K*. La chiave può essere o non essere presente su U, e la risposta diretta all'unità richiedente l'operazione, deve indicare se la chiave è stata trovata (e con quale valore, in caso di lettura) o no. Tutte le unità interagiscono a domanda risposta. La priorità va sempre data alla unità di indice minore. La struttura dati che contiene le coppie è implementata utilizzando due moduli di memoria distinti: MK contiene alla posizione i la chiave della i-esima coppia ed MV contiene alla posizione i il valore della i-esima coppia, entrambi di 1K parole da 32 bit.
Si progetti l'unità U e se ne fornisca il tempo medio di elaborazione in funzione di tp, sapendo che il 50% delle operazioni sono inserimenti e il 50% sono letture, che la memoria per le coppie ha una capacità di 1K coppie, che le porte logiche che stabilizzano in 1tp hanno al massimo 8 ingressi e che sono disponibili solo ALU che fanno somme e incrementi/decrementi di una unità con un ritardo di stabilizzazione pari a 5tp.



Risorse utilizzate

Utilizziamo due memorie MV e MK per contenere, alla posizione i, il valore e la chiave della coppia i-esima. “uguale” è una rete combinatoria che ci dice se due registri sono identici. È costituita da n confrontatori in parallelo le cui uscite sono messe in OR. Il risultato dell’OR è negato per avere 1 (vero) quando tutti i bit corrispondenti nei due registri sono uguali. Il tempo di stabilizzazione della rete è 4 tp (2tp per i confrontatori, che lavorano tutti in parallelo, e 2tp per l’albero di due livelli di porte OR che calcolano l’OR dei 32 bit risultato dei confrontatori).

L’interfaccia con l’unità che richiede l’inserimento è composta, oltre che ai due indicatori a transizione di livello in ingresso ed in uscita, da un registro in ingresso K, che contiene la chiave per la quale si vuole effettuare l’inserimento, un registro in ingresso V, che contiene il valore da utilizzare per l’aggiornamento della coppia con chiave K e un registro in uscita ESITO, che verrà utilizzato per comunicare se la chiave è stata trovata (e quindi l’aggiornamento è avvenuto) o no.

L’interfaccia con le unità che richiedono la lettura, è simile a quella della unità che richiede l’inserimento, tranne per il fatto che non è presente il registro V.

Il **microcodice** potrebbe essere scritto come segue:

1. // nessuna richiesta, ne attendo una
   (RDY1, RDY2, RDY3 = 0 0 0) nop, 1
   // richiesta di inserimento dalla unità a priorità più alta,
   // inizializzo i contatori per la ricerca della chiave
   (=1 - -) 0 → I, uguale (MK [0], K1)→ TROVATO, 2.
   // idem per la unità con la seconda priorità più alta
   (=0 1 -) 0 → I, uguale (MK [0], K2) → TROVATO, 3.
   // richiesta di lettura, dall'ultima unità,
   // inizializzo i contatori per la ricerca della chiave
   (=0 0 1) 0 → I, uguale (MK [0], K3) → TROVATO, 4.
2. // fine memoria, chiave non trovata
   (I0, TROVATO = 1 0) 0 → ESITO1, set ACK1, reset RDY1, 1
   // chiave trovata, aggiorna il valore
   (=0 1) MV[I] + V → MV[I], 1→ ESITO1, set ACK1, reset RDY1, 1
   // chiave non trovata, non sono alla fine, continua alla prossima coppia
   (=0 0) I+1 → I, uguale (MK [I+1], K1) → TROVATO, 2
3. // fine memoria, chiave non trovata
   (I0, TROVATO = 1 0) 0→ESITO2, set ACK2, reset RDY2, 1
   // chiave trovata, aggiorna il valore
   (=0 1) MV[I] → OUT2, 1→ ESITO2, set ACK2, reset RDY2, 1
   // chiave non trovata, non sono alla fine, continua alla prossima coppia
   (= 0 0) I+1 → I, uguale (MK [I+1], K2) → TROVATO, 3
4. // fine memoria, chiave non trovata, esito negativo
   (I0, TROVATO = 1 0) 0→ESITO3, set ACK3, reset RDY3, 1
   // chiave trovata, comunica il valore, esito positivo
   (=0 1) MV[I] → OUT3, 1→ ESITO3, set ACK3, reset RDY3, 1
   // chiave non trovata, non sono alla fine, continua alla prossima coppia
   (= 0 0) I+1 → I, uguale (MK [I+1], K3) → TROVATO, 4

Abbiamo cinque variabili di condizionamento, che sono tutte uscite di registri o di indicatori in ingresso a transizione di livello, La condizione di correttezza è rispettata. Al massimo vengono testate 3 variabili (nella prima microistruzione) contemporaneamente. TωPO è pari a 0.

Le risorse di stato sono le seguenti:
• MK (in sola lettura) e MV (scritta con l'uscita di una ALU)
• TROVATO, registro da 1 bit scritto mediante l'uscita della rete combinatoria UGUALE
• ESITO1, ESITO2, ESITO3, scritti con costanti (0 o 1)
• OUT1 e OUT2, scritti con l'uscita dalla memoria MV

La memoria MK viene letta con indirizzi 0 e I+1 (uscita di una ALU) dunque necessita di un commutatore sull'ingresso indirizzi. La memoria MV viene letta e scritta con il solo indirizzo I e non necessita di commutatori né sull'ingresso indirizzi (sempre I) né su quello dati (sempre uscita della ALU che somma il valore in ingresso al contenuto precedente della cella di memoria).

Le risorse di calcolo comprendono:
• La rete combinatoria UGUALE, che risponde 1 quando i due registri in ingresso sono uguali, come descritto precedentemente
• Una ALU per il calcolo del nuovo valore da inserire nella coppia con chiave K e per effettuare gli incrementi del contatore I. Tale ALU avrà dei commutatori sugli ingressi per poter effettuare, in frasi diverse, le operazioni I+1 e MV[I] + V

La parte controllo ha cinque ingressi (variabili di condizionamento) e 2 bit di stato (4 microistruzioni). Il livello AND delle σPC e ωPC sarà quindi uno solo, avendo al massimo 5 ingressi (2 bit di stato + 3 (max) variabili di condizionamento testate contemporaneamente). Essendo presenti 13 frasi, il livello OR sarà costituito da 2 livelli di porte. Il ritardo complessivo della PC sarà quindi TσPC = TωPC = 3tp.

Per TσPO consideriamo la micro operazione più lunga, che potrebbe essere o la uguale (MK [I+1], K) → TROVATO o la MV[I] + V → MV[I]. La prima delle due operazioni richiede 4tp per il calcolo di UGUALE oltre ad un ta per l'accesso in memoria e un talu per calcolare I+1. La seconda, richiede ta + talu. Il tempo di accesso alla memoria richiede la stabilizzazione di un commutatore a 1024 ingressi: 10 + 1 ingressi per il livello AND (i bit di indirizzo e il bit dell'ingresso relativo, gli altri sono

don’t care (”-”) => due livelli di porte AND) e 1024 termini per il livello OR (i 1024 “1” corrispondenti alle 1024 righe della tabella di verità => parte intera superiore di log8 (1024) = 4 livelli di porte OR), dunque 6tp. In totale la MV[I] + V → MV[I] richiede quindi 11tp (il selettore di scrittura stabilizza durante la stabilizzazione del commutatore di lettura).
Similmente la uguale(MK[I+1], K) richiede 5tp (ALU) + 6tp (M) + 4tp (uguale) = 15tp
Quindi per la seconda frase delle microistruzioni 2. e 3. occorreranno per σPO 15tp. Questo porta ad aver bisogno di un ciclo di clock pari a

$$\tau = 0 + \max\{3tp, 3tp+15tp\} + tp = 19tp$$

Supponiamo che l’operazione di inserimento o estrazione trovi la chiave in media dopo aver controllato 1K/2 = 512 posizioni. Se non la trova, occorrerà invece scorrere tutta la memoria, quindi 1024 posizioni. Nel primo caso occorre considerare l’esecuzione:

- Della microistruzione 1. (1τ)
- Di 512 esecuzioni della microistruzione 2. 3. o 4. (512τ)

Per un totale di 513τ.

Visto che le due operazioni esterne richiedono lo stesso numero di microistruzioni, questo sarà anche il tempo medio di elaborazione, indipendentemente dalle percentuali di operazioni dei due tipi effettivamente richieste.