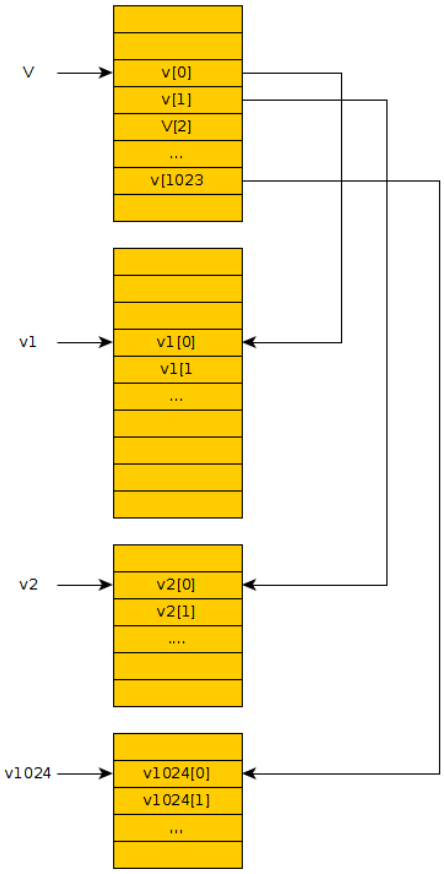
Architettura degli elaboratori—Primo Appello

A.A. 2017-18—24 gennaio 2018

Riportare in alto a destra di ciascun foglio consegnato Nome, Cognome, Corso (A o B) e numero di matricola.
I risultati saranno comunicati via web appena disponibili.

Domanda 1

Si consideri un vettore di M=1K posizioni che contiene in ciascuna posizione l'indirizzo base di un vettore di numeri interi. Ciascuno dei vettori di numeri interi contiene a sua volta N=16 elementi, tutti positivi. Si fornisca lo pseudocodice e il codice D-RISC del programma che trova l'indirizzo del vettore per cui il prodotto di tutti gli elementi è massimo (si trascuri la possibilità di overflow).

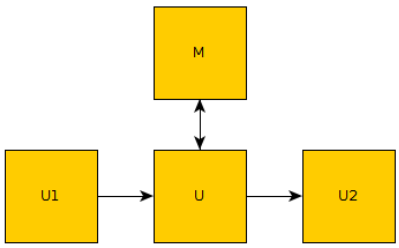


Il processore è un processore D-RISC pipeline standard, con unità EU slave da due stadi per l'esecuzione di moltiplicazione di numeri interi, con cache di primo livello nelle IM e DM, collegato direttamente alla memoria principale esterna (senza ulteriori componeni cache) che è una memoria interallacciata.

Si evidenzino le eventuali dipendenze nell'intero codice, si discutano possibili ottimizzazioni del codice D-RISC del ciclo più interno e si discutano le caratteristiche della cache in DM che minimizzano il tempo di completamento, motivando adeguatamente la risposta.

Domanda 2

Si progetti un'unità firmware U interfacciata a due unità U_1 ed U_2 e, tramite una interfaccia di memoria standard, ad una memoria esterna M da 4G parole. Eventuali errori di lettura da M vanno trattati come errori temporanei, ripetendo quindi la richiesta di operazione.



U_1 può richiedere ad U due operazioni: OP_1(IND,K) ed OP_2().

OP_1 ha come parametri l'indirizzo IND in M di un vettore di interi positivi e il numero K di posizioni nel vettore e calcola il massimo MAX del vettore. Quindi memorizza nel registro R la somma tra il valore di R e MAX.

OP_2 non ha parametri e viene eseguita inviando ad U2 il valore di R. R è inizializzato a 0.

Si implementi l'unità minimizzando il numero di microistruzioni da eseguire per eseguire ciascuna delle due operazioni esterne.

Bozza di soluzione

Domanda 1

Pseudo Codice

```
pointer indvec = 0;
float maxprod = 0;
for(int i=0; i<N; i++) {
  float prod = 1;
  for(int j=0; j<N; j++) {
     prod *= (v[i])[j];       // v[i] is the Rbase and j the Rindex
  }
  if(prod > maxprod) {
    maxprod = prod;
    indvec = v[i];
  }
}
```

Assembler D-RISC (ciclo interno non ottimizzato)

```
     CLEAR Rindvec                  // azzera indirizzo vettore
     CLEAR Rmaxprod                 // azzera prodotto
     CLEAR Ri                       // inizilalizza var iterazione
loopi: ADD R0,#1,R0                 // inizializza prodotto
parziale
     LOAD RbaseV,Ri,RbaseVi         // calcola base vettore i-esimo
     CLEAR Rj
loopj: LOAD RbaseVi, Rj, Rtemp      // carica elemento j-esimo
     MUL Rprod, Rtemp, Rprod        // moltiplica per il parziale
     INC Rj                         // incremento var iterazione
     IF< Rj, RN, loopj              // controllo fine ciclo
     IF<= Rprod, Rmaxprod,cont      // secondo statement ciclo i
     ADD Rprod, R0, Rmaxprod        // se > aggiorna maxsum
     ADD RbaseVi, R0, Rindvec       // se > aggiorna base vett
cont:INC Ri                         // altrimenti incrementa i
     IF< Ri, R1024, loopi           // controllo fine ciclo
     END
```

Il costo del calcolo di una iterazione del ciclo interno è 7t:

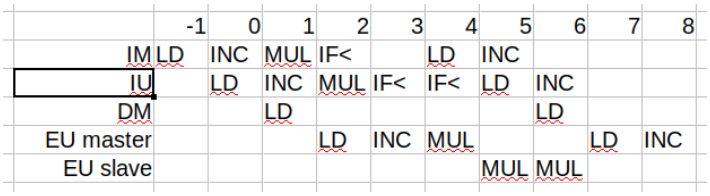
	0	1	2	3	4	5	6	7	8	9	10
IM	LD	MUL	INC	IF<			IF<=	LD			
IU		LD	MUL	INC	IF<	IF<	IF<	IF<=	LD		
DM			LD							LD	
EU master				LD	MUL	INC					LD
EU slave						MUL	MUL				

Ottimizzazione ciclo interno

Possiamo anticipare la INC quanto vogliamo e, utilizzando il salto ritardato, mettere la LOAD nello slot del salto di fine ciclo, saltando direttamente alla INC:

```
          LOAD RbaseVi, Rj, Rtemp
loopj:    INC Rj
          MUL Rprod, Rtemp, Rprod
          IF< Rj, RN, delayed
          LOAD RbaseVi, Rj, Rtemp
          IF<= Rprod …
```

Questo permette di ridurre la latenza della singola iterazione a 5t:



	-1	0	1	2	3	4	5	6	7	8
IM	LD	INC	MUL	IF<		LD	INC			
IU		LD	INC	MUL	IF<	IF<	LD	INC		
DM			LD					LD		
EU master				LD	INC	MUL			LD	INC
EU slave							MUL	MUL		

Caratteristiche minimali cache DM

Il working set del programma è costituito dal codice (acceduto con località e riuso), da una linea del vettore V e da una linea del vettore Vi. Il codice viene acceduto in IM e dunque non vi è influenza sulla cache DM. La cache in DM deve permettere l'accesso simultaneo delle due linee con I dati di V e Vi. Qualunque cache set aassociativa ad almeno due vie o completamente associativa garantisce che non vi saranno fault spuri. Una cache DM ad indirizzamento diretto potrebbe verificare un conflitto fra gli accessi al vettore degli indirizzi e l'accesso alla prima linea del vettore dati corrente. Questo comporterebbe un aumento dei fault dovuti al vettore degli indirizzi.

Domanda 2

Interfacce

Assumiamo di avere tre interfacce:

- con U1, in ingresso RDY1, OP, IND e K, in uscita ACK1
- con U2, in ingresso ACK2, in uscita RDY2 e OUTR
- con M, in uscita RDYm, OPM, INDM, DATAOUTM e in ingresso ACKm, DATAINM, ESITO

Microcodice

```
0.    (RDY1,ACK2,OP=0--) nop, 0
      // prima operazione, inzializzo I registri
      // e chiedo la lettura del primo elemento
      (=1-0) -1 → MAX, 1→ I,  IND → INDM, “read”→ OPM, set RDYm, 1.
      // seconda operazione, attendo l’ack a trasmettere per U2 e mando R
      (=101) nop, 0
      (=111) R → OUTR, set RDY2, reset ACK2, 0

1.    // ciclo che implementa la prima operazione: attendo esito lettura in M
      (ACKm, OR(ESITO), zero(I-K), segno(MAX-DATAINM)=0---  ) nop, 1
      // esito negativo, ritento la stessa lettura
      (=11--) reset ACKm, set RDYm, 1
      // lettura di un valore più grande del MAX corrente
      (=1001) DATAINM→ MAX, I+1→ I, IND+I→ INDM, “read”→ OPM,
      reset ACKm, set RDYm, 1
      // lettura dell’ultimo valore, che risulta il più grande,
      // memorizzo e torno alla 0
      (=1011) DATAINM + R→R, reset RDY1, set ACK1, reset ACKm, 0
      // lettura di un valore minore del valore corrente
      (=1000) I+1 → I, IND+I→INDM, “read”→ OPM, reset ACKm, set RDYm, 1
      // lettura dell’ultimo valore, minore del massimo corrente
      (=1010) MAX + R → R, reset RDY1, set ACK1, reset ACKm, 0
```

Per eseguire la prima operazione eseguiamo la 0. una volta (richiedendo la lettura di V[IND]) e la 1 (K) volte, a meno di errori e processando ogni volta il valore letto mentre si ordina la lettura del nuovo valore in M.

Per implementare la prima operazione esterna sono necessarie almeno K iterazioni, visto che dobbiamo leggere e confrontare tutti i valori dell’array, che sono appunto K. Almeno per la prima iterazione, va prevista una microistruzione per l’ordine della lettura in M ed una per l’attesa del risultato. Per le successive, l’attesa del risultato può inglobare anche la preparazione della nuova richiesta di lettura. Dunque la prima iterazione costa 2 istruzioni (al minimo) e le successive costano 1 singola micro istruzione, per un totale di K+1 micro istruzioni.
Il numero minimo di istruzioni per implementare la seconda operazione esterna è 1. Dunque il microprogramma è ottimizzato come richiesto.

La PC può essere realizzata semplicemente con un ritardo di $2t_p$ sia per ω che per σ, visto che abbiamo 1 bit di stato e massimo 4 variabili di condizionamento testate contemporaneamente.

Per la PO seguendo il procedimento standard non vi sono particolari accorgimenti, se non che le ALU per il test della variabili di condizionamento sono ALU dedicate.

In alternativa, potremmo utilizzare un po' di controllo residuo per calcolare il massimo dei due valori. Questo permetterebbe di ridurre il numero delle variabili di condizionamento e delle frasi della microistruzione 1. che potrebbe diventare:

```
1.    // ciclo che implementa la prima operazione: attendo esito lettura in M
      (ACKm, OR(ESITO), zero(I-K) = 0--) nop, 1
      // esito negativo, ritento la stessa lettura
      (=11-) reset ACKm, set RDYm, 1
      // lettura di un valore
      (=100) max(DATAIN,MAX)→ MAX, I+1→ I, IND+I→ INDM, "read"→ OPM,
      reset ACKm, set RDYm, 1
      // lettura dell'ultimo valore
      (=101) max(DATAINM,MAX) + R→R, reset RDY1, set ACK1, reset ACKm, 0
```

In questo caso max(X,Y) viene implementato con un commutatore con ingressi X e Y comandato (α_k) dal bit segno di una ALU che calcola X-Y. Benchè il codice sia più compatto, tutto questo non porta vantaggi dal punto di visto implementativo. I ritardi della PC erano già minimi e non variano. La stabilizzazione della ALU per il calcolo del massimo avviene in parallelo con la stabilizzazione di quella che serve per il calcolo di zero(I-K) e il commutatore lavora mentre si calcola il resto delle operazioni nella frase scelta.