Architettura degli elaboratori – A.A. 2017-2018
Secondo appello straordinario – 6 aprile 2018

Riportare in alto a destra su tutti i fogli consegnati nome, cognome, matricola e corso (A o B).
I risultati verranno pubblicati su web appena disponibili, insieme al calendario degli orali.

Domanda 1

Una unità U contiene una memoria M da 1K posizioni ed implementa due operazioni esterne:

- la prima operazione ricerca un valore X all'interno della memoria M e ne sostituisce tutte le occorrenze con la somma di X e del valore letto nella posizione di memoria precedente. In caso X si trovi nella prima posizione, si assume che il valore letto nella posizione di memoria precedente (che non esiste) sia 123.
- la seconda operazione conta il numero di occorrenze di X all'interno della memoria, e ne restituisce il valore come risultato dell'operazione esterna.

L'unità U_0 richiede operazioni ad U utilizzando un protocollo a domanda risposta.

Si fornisca un'implementazione di U che minimizza la durata del ciclo di clock, motivando adeguatamente la risposta.

Domanda 2

Si implementi in assembler D-RISC l'operazione di moltiplicazione fra interi, rappresentati in modulo e segno su 16 bit, con risultato ancora in modulo e segno su 32 bit e se ne fornisca una stima delle prestazioni su un processore D-RISC pipeline.

L'algoritmo da utilizzare è quello classico: ad ogni passo si considera un bit del secondo fattore: se vale 0, si passa a considerare il bit successivo, altrimenti si somma il primo fattore, opportunamente allineato, al risultato parziale. Si ricordi che tutte le istruzioni del set D-RISC operano su interi rappresentati in complemento a due.

La procedura ha 3 parametri, passati per registro: due sono i registri che contengono nella parte bassa (16 bit meno significativi) i due interi da moltiplicare rappresentati in modulo e segno e uno conterrà il prodotto calcolato. Si forniscano le prestazioni della procedura eseguita su un processore D-RISC pipeline.

A titolo esemplificativo, riportiamo i passi necessari a moltiplicare due numero positivi su 3 bit:

```
    100 x 101         // moltiplicazione di 4 * 5

    100 x 101         // primo bit a 1, sommo il primo fattore
    ----
    100

   1000 x 101         // secondo bit a 0, ignoro il primo fattore (sommo 0)
-------               // che è stato traslato di una posizione a sn
    100
   000

  10000 x 101         // terzo bit a 1, sommo il primo fattore, che come ad ogni
-------               // passo è stato traslato a sn di una posizione
    100
   0000
  10000

  10000 x 101         // somma e ottieni il risultato finale
-------               // dal momento che la somma è associativa e commutativa
    100               // il risultato può essere ottenuto anche per somme parziali
   0000               // eseguite ad ognuno dei passi
  10000
-------
  10100               // risultato finale = 16 + 4 = 2
```

Bozza di soluzione

Domanda 1
Il microcodice potrebbe essere il seguente:

0. 0→I, 0→ COUNT, 123→PREV, 1 // inizializzazioni
1. (RDY,I0,OP=0--) nop, 1 // attesa richiesta
   // prima operazione
   (=100) M[I]+PREV → M[I] sse M[I]==X, M[I] → PREV, I+1→ I, 1
   (=110) reset RDY, set ACK, 123→ PREV, 0→ I, 1
   // seconda operazione
   (=101) COUNT+1 → COUNT sse M[I] == X, I+1→ I, 1
   (=111) reset RDY, set ACK, COUNT→ RESOUT, 0→ COUNT, 1

La minimizzazione della lunghezza del ciclo di clock deriva dal fatto che vengono utilizzate tutte variabili di condizionamento semplici ($T\omega PO = 0$) e l'ALU che calcola il bit di controllo residuo

lavora in parallelo all'ALU che comunque servirebbe per sommare M[I] e PREV o per incrementare COUNT o a quella che servirebbe per calcolare M[I]+PREV. Serve comunque un commutatore in ingresso per i registri I e COUNT. La PC è semplice e non introduce ritardi maggiori ai 2tp (3 variabili di condizionamento e 2 microistruzioni danno 4 ingressi per il livello AND; 6 frasi danno al massimo 5 ingressi per il livello OR).

Risorse di memoria:

- I con commutatore in ingresso che sceglie fra 0 e l'uscita di una ALU che calcola +1
- COUNT con commutatore in ingresso che sceglie fra 0 e l'uscita di una ALU che calcola +1
- PREV con commutatore in ingresso che sceglie fra 123 e l'uscita della memoria M[I]
- X, OP registri in ingresso dall'unità W
- RDY, ACK indicatori a transizione di livello in ingresso e uscita da e verso W
- RESOUT registro in uscita verso W

Risorse di calcolo:

- una ALU che calcola +1 per I e una per COUNT
- una ALU che calcola X-M[I] e produce il falg Z che viene utilizzato come beta di M[I] e per il controllo della scrittura di COUNT. Poichè COUNT è scritto nella prima microistruzione (inizializzazione) e nella seconda (incremento) il beta sarà il risultato dell'OR fra il beta di scrittura della prima micro-istruzione (generato dalla PC) e quello preso dal flag Z della ALU (generato dalla PO).
- Una ALU che calcola M[I]+PREV
- commutatori a due vie per gli ingressi di K, COUNT, PREV

Il ciclo di clock può essere calcolato come:

$$T\omega PO + T\omega PC + T\sigma PO + \delta$$

ovvero

$0 + 2tp + (ta + talu) + tp$

assumendo che ta > tk e talu > tk. La $T\sigma PO$ deriva dalla frase:

M[I]+PREV → M[I] sse M[I]==X, M[I] → PREV, I+1→ I, 1

Dobbiamo leggere M[I] calcolare la variabile controllo residuo (M[I]==X) e la somma M[I]+PREV e poi scrivere il risultato in M[I]. M[I] → PREV richiede ugualmente l'accesso a M e un tk per la scrittura, così come la I+1→ I.

Domanda 2

Lo pseudo codice potrebbe essere il seguente:

```
int mul(int a, int b) {
  int res = 0;          // risultato assoluto

  memorizza in sa ed sb i segni di a e b (bit 15);
  elimina i 17 bit più significativi di a e b (16 bit alti e segno)

  for(count=0; count<16; count++) {    // algoritmo di moltiplicazione
        if(bit0(b) == 1)
                res = res + a;
        shl(res,1);
        shr(b,1);
  }

  if(sa != sb)          // se segni non concordi risultato positivo
    res = -res;         // altrimenti negativo

  return res;
}
```

Il compilato sarà dunque (primo fattore in R1, secondo in R2, risultato in Rres, indirizzo di ritorno in Rret):

```
mul:    AND R1, RmaskA, R1      ; cancella primi 15 bit
        AND R2, RmaskA, R2      ; cancella primi 15 bit
        SHR R1, 15, Rsa         ; segno di R1 nel primo bit di Rsa
        SHR R2, 15, Rsb         ; segno di R2 nel primo bit di Rsb
        AND R1, RmaskB, R1      ; cancella bit di segno
        AND R2, RmaskB, R2      ; cancella bit di segno
        CLEAR Rres              ; prepara risultato = 0
        CLEAR Rcount            ; contatore ciclo for
for:    AND R2, #1, Rfadd       ; controlla se bit 0 è 1
        IF=0 Rfadd, skip        ; se è 0 non sommare
        ADD Rres, R1, Rres      ; aggiungi al risultato il primo fattore
skip:   SHR R2, #1, R2          ; scorri a destra il secondo operando
        SHL R1, #1, R1          ; scorri a sn il primo operando
        INC Rcount              ; nuova iterazione, incrementa var
        IF< Rcount, #15, for    ; prossima iterazione se non finito
cont:   IF= Rsa, Rsb, fine      ; se i segni sono concordi
neg:    OR Rres, RmaskC, Rres   ; altrimenti negativo
fine:   goto Rret               ; ritorno dalla procedura
```

considerando li seguenti valori per i registri con le costanti:

```
RmaskA      0000 0000 0000 0000 1111 1111 1111 1111
RmaskB      0000 0000 0000 0000 0111 1111 1111 1111
RmaskC      1000 0000 0000 0000 0000 0000 0000 0000
```

Per l'esecuzione abbiamo due dipendenze (AND R2, #1, Rfadd su IF=0 successiva e INC Rcount su IF< successiva) che generano entrambe una bolla da 1t (non ci sono LOAD nelle sequenze di istruzioni che portano all'istruzione su cui è indotta la dipendenza). Inoltre, c'è un'ulteriore bolla da

1t per il salto condizionato di fine ciclo for. Dunque abbiamo 8t per la parte iniziale, 3t per la parte finale (la OR o viene eseguita o viene scartata. Nel primo caso ho 3t utili, nel secono 2t utili e uno “bolla da salto”) e 10t per ognuna delle iterazioni (tranne l’ultima che conta 9t, visto che manca la bolla da salto per il ritorno alla testa del ciclo). In totale fanno

$$8t + 15*10t + 9t + 3t = 170t$$