Architettura degli elaboratori – A. A. 2019-20 – 2o appello
Corso A (vecchio programma) e Corso B – 6 febbraio 2020

Riportare su tutti i fogli consegnati in alto a destra Nome, Cognome, Matricola e Corso di appartenenza (A/B).
I risultati saranno resi pubblici via WEB appena disponibili assieme al calendario degli orali.

Domanda 1

In riferimento al seguente frammento di pseudocodice, dove **k<M<N,**

```
i=0;
found=false;
while (!found && i<N) {
   x=0;
   for(j=0; j<M; j++) {
      if big[i+j]==small[j] {  x++; }
   } // end for
   if (x<k) {  i++; } else { found = true; }
} // end while
```

si forniscano (giustificando adeguatamente ogni risposta):

- compilazione in assembler D-RISC secondo le regole di compilazione standard;
- prestazioni (tempo di servizio) del ciclo for, su un processore D-RISC pipeline;
- cause di degrado delle prestazioni;
- eventuali ottimizzazioni del ciclo while, fornendo il codice ottimizzato e quantificando il guadagno in termini di tempo di servizio (il guadagno in termini di tempo di servizio va quantificato solo per il ciclo for).

Domanda 2

Una unità U_a accede mediante una interfaccia di memoria standard ad una memoria esterna **M** e comunica con una unità U_b. La memoria esterna, indirizzata alla parola, contiene un vettore **Y** di record di tipo **struct {int cod; int tot}** con **cod** e **tot** da 32 bit, memorizzati in parole contigue. La posizione I-esima del vettore contiene in **cod** una chiave e in **tot** un intero. U_a interagisce con un'unità U_b che le manda una serie di richieste di operazione. Per ogni richiesta, U_a riceve da U_b un intero **I**, che utilizza come indice per accedere al vettore. *Successivamente*, riceve una chiave **K**. Se la chiave ricevuta **K** coincide con la chiave alla posizione **I** del vettore, U_a invia un riscontro positivo ad U_b e attende un terzo valore **V** da U_b, altrimenti segnala un errore e la comunicazione termina. Il terzo valore viene confrontato con **Y[I].tot** e si restituisce un **1** o uno **0** ad U_b a seconda se il valore di **Y[i].tot** è diverso o uguale al terzo valore ricevuto. L'interfaccia di U_a verso U_b è costituita dai soli registri in ingresso **IN** (32 bit) e in uscita **OUT** (1 bit), oltre agli indicatori a transizione di livello necessari all'implementazione del protocollo di comunicazione. Si forniscano:

- una implementazione della unità Ua, che minimizzi il numero di cicli di clock necessari ad eseguire l'operazione esterna che essa implementa;
- la lunghezza del ciclo di clock, nell'ipotesi di avere a disposizione ALU che operano in 6 t_p e porte logiche a 4 ingressi;
- il tempo medio di elaborazione, nell'ipotesi che la segnalazione dell'errore avvenga nel 25% dei casi.

Bozza di soluzione

Domanda 1

Compilazione del codice secondo le regole di compilazione standard:

```
        clear Ri                    i=0
        clear Rfound                found=false
loopw:  if>= Ri,RN,fine             i<N?
        if!= R0,Rfound,fine         found=false?
        clear Rx                    x=0
        clear Rj                    j=0
loopf:  add Ri,Rj,Rij               i+j
        load Rbasebig,Rij,Rbij      leggi big[i+j]
        load Rbasesmall,Rj,Rsj      leggi small[j]
        if!= Rbij,Rsj,skip          big[i+j]=small[j]?
        inc Rx                      x++
skip:   inc Rj                      j++
        if< Rj,Rm, loopf            j<M?
        if< Rx,Rk,inci              x<k?
        addi Rfound,#1,rfound       ramo else: found=true
        goto loopw                  salta all'inizio
inci:   inc Ri ramo                 then: i++
        goto loopw                  salta all'inizio
fine:   xxx                         fine
```

La simulazione mostra che occorrono 12t per completare le 7 istruzioni del corpo del ciclo for, sia nel caso che la condizione dell'if sia vera che falsa (vedi le due simulazioni riportate di seguito per i due casi).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IM	ADD	LOAD		LOAD	IF !=			INC	INC	IF<		XXX	ADD		
IU		ADD	LOAD	LOAD	LOAD	IF !=	IF !=	IF !=	INC	INC	IF<	IF<	XXX	ADD	
DM					LOAD	LOAD									
EU			ADD			LOAD	LOAD			INC	INC				ADD

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IM	ADD	LOAD		LOAD	IF !=			INC	INC	IF<		XXX	ADD		
IU		ADD	LOAD	LOAD	LOAD	IF !=	IF !=	IF !=	~~INC~~	INC	IF<	IF<	~~XXX~~	ADD	
DM					LOAD	LOAD									
EU			ADD			LOAD	LOAD				INC				ADD

Questo fà sì che il tempo di servizio sia pari a 12t/6.5 (se la condizione è falsa si esegue una istruzione in meno, 6 invece che 7, quindi mediamente sono (6+7)/2 = 6.5 istruzioni per iterazione). Il degrado delle prestazioni è legato alla dipendenza indotta dalla ADD sulla prima LOAD, da quella indotta dalla seconda LOAD sulla IF!= e infine quella indotta dalla INC sulla IF<. Inoltre il salto induce un ulteriore degrado delle prestazioni.

Il codice può essere ottimizzato:

- Scambiando l'ordine delle LOAD
- Anticipando l'incremento di Rj prima dell'IF!= sul risultato delle LOAD

	0	1	2	3	4	5	6	7	8	9	10	11
IM	ADD	LOAD	LOAD	INC Rj	IF!=		INC Rx	IF<	XXX	ADD		
IU		ADD	LOAD	LOAD	INC Rj	IF!=	IF!=	INC Rx	IF<	XXX	ADD	
DM				LOAD	LOAD							
EU			ADD		LOAD	LOAD1	INC Rj		INC Rx			ADD

In questo caso si riesce a ridurre il tempo di servizio a 9t/7, visto che la dipendenza indotta sulla IF!= ha un effetto ridotto, e la dipendenza ADD -> LOAD è annullata, così come quella INC -> IF< (e dovremmo sempre considerare la media con il caso in cui la condizione dell'IF non comporti l'esecuzione del ramo THEN, che tralasciamo).

Per quanto riguarda il resto del codice del ciclo while, si può ottimizzare modificando le istruzioni fuori dal ciclo for e alla fine del while in questo modo (le modifiche sono in grassetto), essendo inutile ritornare in ciclo quando si assegna found=true:

```
        if>= Rx,Rk,else
        ~~goto loopw~~              salta all'inizio viene eliminata
inci:   inc Ri                      ramo then: i++
        goto loopw                  salta all'inizio
else:   addi Rfound,#1,rfound       ramo else: found=true
fine:   xxx                         prima istruzione dopo il while
```

Domanda 2

Per minimizzare il numero di cicli di clock del microcodice di controllo si cerca di raggruppare le micro-operazioni necessarie quanto più possibile all'interno delle stesse microistruzioni. Le operazioni di

comunicazione richiedono comunque microistruzioni dedicate. Dovendo nel migliore dei casi ricevere tre valori da Ub e due da M, sembrerebbe che occorrano almeno 5 microistruzioni. In realtà, la seconda comunicazione da Ub può sovrapporsi alla prima lettura in memoria, così come la terza comunicazione da Ub può sovrapporsi alla seconda lettura. Dunque, dovremmo considerare 3 microistruzioni come minimo numero di microistruzioni da eseguire.

Microcodice:

0. (RDYb = 0) nop, 0
   (=1) IN -> IND, IND+1 -> TEMP, "read"->OP, set RDYoutM, reset RDYb, set ACKb, 1
1. (RDYb, RDYinM, zero(DATAIN-IN),OR(ESITO) = 0---, -0--) nop, 1
   (=1110) 1-> OUT, reset RDYm, set ACKm, reset RDYinM, TEMP-> IND, set RDYoutM, 2
   (=1100) reset RDYm, 0 -> OUT, set ACKm, reset RDYinM, 0
   (=11-1) reset RDYm, 0->OUT, set ACKm, reset RDYinM, 0
2. (RDYm, RDYinM, OR(ESITO) =0--, -0-) nop 2,
   (=110) segno(DATAIN – IN) -> OUT, reset RDYm, set ACKm, reset RDYinM, 0
   (=111) 0 -> OUT, reset RDYm, set ACKm, reset RDYinM, 0

Si noti che in caso di errore nella lettura dalla memoria esterna diamo comunque una risposta all'altra unità:

- Nel caso sia fallita la lettura della chiave, diamo una risposta 0 come se il confronto fosse comunque fallito
- Nel caso sia fallita la lettura del valore da confrontare col secondo intero inviato dall'altra unità, scegliamo di inviare una risposta qualunque (minore, in questo caso). Sarebbe stato più corretto prevedere l'invio di un bit di validità della risposta, che però non era previsto nell'interfaccia fra le due unità così come proposta nel testo.

Calcolo del ciclo di clock:
Le micro-operazioni costano al più Talu+Tk (0.) Il calcolo delle variabili di condizionamento costa Talu. I due valori sono però relativi a due microistruzioni diverse. La 0. costa (TomegaPO + TsigmaPO) 0 + Talu + Tk (commutatore sull'ingresso IND. La 1. costa Talu + Tk. La 2. costa 0 + Talu. Dunque il ciclo di clock dovrà essere almeno Talu + Tk + TomegaPC + delta. La parte controllo ha in ingresso 3 variabili di condizionamento. Lo stato interno è da 2 bit. Le frasi sono 9. Dunque, sono sufficienti due livelli di logica per implementare sia sigmaPC che omegaPC (per il livello OR, con 9 frasi abbiamo comunque al massimo 8 uni per ognuna delle colonne beta e alpha). Questo porta il ciclo di clock a

$$Talu + Tk + 2tp + tp = Talu + 5tp.$$

Il tempo medio di elaborazione sarà dato dal tempo medio nei due casi (codice uguale => 2 microistruzioni) e codice diverso (=> 3 microistruzioni) per le rispettive probabilità. Dunque

$$T = 1* 2 \text{ tau} / 4 + 3*3 \text{ tau} / 4 = 11 \text{ tau}/ 4$$