AE corso B – A.A. 2019-2020 – 2ª prova di verifica intermedia
Tipo **B**

*Riportare nella **prima facciata** in **alto a destra** di ciascuno dei fogli consegnati **Nome**, **Cognome**, numero di **matricola** e **tipo** di compito **(B** in questo caso**)**.*

Si consideri il seguente programma in pseudolinguaggio (supponendo che N<2049):

```
For (i=0, i<N, i++)
        {if (B[i]<N/2)&(B[i]>0)
                {
                   C[i]=(A[i] *N)MOD B[i]+16*B[i];
                }
        }
```

Per tale programma:

1) Si fornisca la compilazione in assembler D-RISC (a cui è aggiunta, oltre alle istruzioni dell'assembler D-RISC descritto nelle note distribuite a lezione, l'istruzione *aritmetica lunga* **MOD** che calcola il resto della divisione intera), utilizzando le regole standard.
2) *a)* Si indichi il numero di parole di ogni linea della cache (σ), e
   b) si fornisca il numero di cache fault (sia con che senza prefetch) per una cache associativa su insiemi di 64K parole, con 512 insiemi di 8 linee ciascuno.
3) Si forniscano le dipendenze tra istruzioni e le prestazioni (tempo di completamento) dell'esecuzione di un ciclo (caso in cui si esegue l'assegnazione a C[i]), su un processore D-RISC pipeline con unità *EU slave pipeline a **2** stadi* per l'esecuzione delle operazioni aritmetiche lunghe, e il tempo di completamento ideale.
4) Si individuino le cause di degrado delle prestazioni dovute a bolle.
5) Si fornisca il codice con eventuali ottimizzazioni per ridurre l'effetto delle bolle, quantificando il guadagno in termini di tempo di servizio; giustificare tale guadagno.

Bozza di soluzione

1) Il programma derivato dalla compilazione in assembler D-RISC secondo le regole standard può essere (a sinistra una numerazione delle istruzioni):

```
        ADD     R0,R0,RI        //i=0
1   LOOP: LOAD  RBB,RI,RBI      //leggi b(i)
2       SHR     RN,#2,RN2       //calcola N/2
3       IF>=0   RBI,RN2,IND     // test b(i)<N/2
4       IF<=0   RBI,R0,IND      // test b(i)>0
5       LOAD    RBA,RI,RAI      //leggi a(i)
6       MUL     RAI,RN,RT1      //calcola a(i)*N
7       MOD     RT1,RBI,RT2     //calcola (a(i)*N)MODb(i)
8       SHL     RBI,#4,RT3      //calcola 16*b(i)
9       ADD     RT3,RT2,RT1     //somma i due termini sopra
10      STORE   RBC,RI,RT1      //memorizza il risultato
11  IND: ADDI   RI,#1,RI        //incrementa l'indice
12      IF<     RI, RN,LOOP     //fine ciclo
13      END
```

2) Cache

a) 64K/512=$2^{16}/2^9=2^7$ parole per ciascun insieme; $2^7/2^3=2^4$=16 parole per ciascuna linea. Quindi, **σ=16**;

b) Premessa: la cache è grande a sufficienza da poter contenere sia tutto il codice che tutti e tre i vettori.

***con** prefetch*: **1** fault per il codice (14 istruzioni=14 parole che stanno tutte in una linea);

1 fault per a(.) e **1** fault per b(.); dato che il vettore c entra tutto nella cache, la scrittura avviene alla fine della esecuzione del processo, quando si carica il nuovo processo, e quindi viene effettuata dal sistema operativo e non è a carico di questo processo.

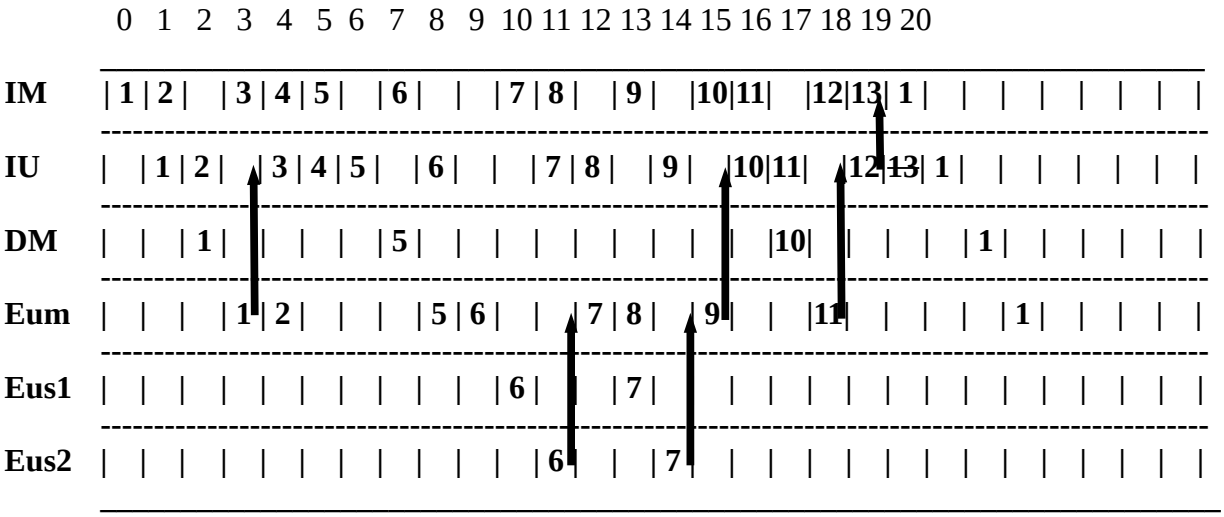
***Senza** prefetch*: **1** fault per il codice (14 istruzioni=14 parole che stanno tutte in una linea);

C(N/16) fault per a(.) e altrettanti per b(.), dove C(y)=intero superiore di y. Per il vettore c, vale la considerazione precedente.

3) Tempo di completamento

Le dipendenze tra le istruzioni sono dovute o al fatto che una istruzione utilizza il risultato di un'altra istruzione (ad esempio, la 3 usa il risultato della 1), oppure al fatto che una istruzione può essere eseguita o meno a seconda dell'esito di un'altra istruzione (che vale tipicamente per le istruzioni in un ramo delle IF, ad esempio, la 5 viene eseguita solo se la 3 e la 4 risultano in un salto non preso).
Nel nostro caso, le **dipendenze** sono: **1→ 3, 1→ 4, 1→ 7 ,1→ 8, 3 e 4 su tutte le istruzioni dalla 5 alla 10 (che altrimenti non verrebbero eseguite), 5→ 6, 6→ 7, 7→ 9, 8→ 9, 9→ 10, 11→ 12**.

Il tempo di completamento **ideale** del ciclo è uguale al numero di istruzioni del ciclo, cioè **12t**.
Per il tempo di completamento del codice si veda il seguente schema (**si fa l'ipotesi che esista il collegamento diretto tra EUslave2 ed IU, oltre a quello tra EUslave2 ed EUmaster**, altrimenti il tempo aumenterebbe di 1t); la prima riga di EUs rappresenta la EU slave 1 e la seconda riga la EU slave 2. Per maggior chiarezza e semplicità, le istruzioni sono indicate in una unità solo nello slot in cui tale unità termina la loro elaborazione. Ad esempio, IU termina l'elaborazione della istruzione 7 nello slot 11 (e la passa alla EU), anche se l'ha ricevuta prima, IM termina l'elaborazione della istruzione 2 (e la passa ad IU che la preleva) nello slot 1. le frecce rappresentano il passaggio di un valore tra l'unità che lo produce e quella che è in attesa per poterlo utilizzare. Ad esempio, la freccia tra EUmaster ed IU alla fine dello slot 15 rappresenta il passaggio del nuovo valore di RT1 ad IU, che aggiorna il valore contenuto nella sua copia di tale registro, per poi completare la 10 (nello slot 16) e passarla a DM.



Il tempo di completamento è di **20t**, con una efficienza di **12/20=60%**.

4)Degrado delle prestazioni dovuto a bolle

Le bolle sulla IU sono dovute alle dipendenze: **1→ 3** (bolla lunga 1t)**, 6→7** (bolla lunga 2t)**, 7→ 9** (bolla lunga 1t), **9→ 10** (bolla lunga 1t)**, 11→ 12** (bolla lunga 1t), oltre alla bolla da salto (lunga 1t) **12→ 1**

5)Ottimizzazione del codice

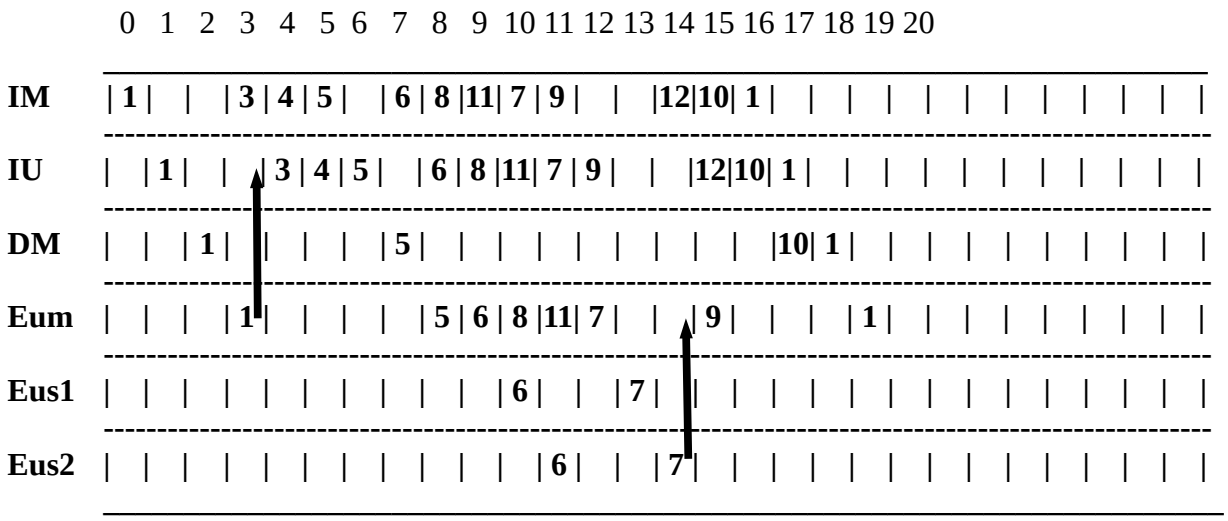
Il codice può essere ottimizzato eseguendo le seguenti modifiche:

- spostare l'istruzione 2 fuori dal ciclo (è invariante); questo spostamento però non comporta nessuna riduzione del tempo di calcolo;
- spostare le istruzioni 11 e 8 tra la 6 e la 7
- usare RBC *anticipato* (cioè *RBC'= RBC-1*) – per compensare l'anticipo della istruzione 11. Questo permette di spostare la 11 dove vogliamo;
- usare la 12 *delayed* e spostare la 10 dopo la 12, **se si vuole ottimizzare il ciclo solo nell'ipotesi descritta al punto 3 del testo, cioè se i salti dei primi due IF non vengono <u>mai</u> presi. Altrimenti, questa modifica non si può fare, e quindi il tempo di completamento risulta più lungo di 1t perché la bolla del salto finale non si può eliminare**. **Si possono però fare altre modifiche, da valutare di caso in caso per vedere se migliorano o meno il tempo totale di esecuzione del codice, tenendo conto che i salti possono essere presi.**

Questo porta alla seguente versione del codice:

```
        ADD     R0,R0,RI        //i=0
2       SHR     RN,#2,RN2       //calcola N/2
1   LOOP: LOAD  RBB,RI,RBI      //leggi b(i)
3       IF>=0   RBI,RN2,IND     // test b(i)<N/2
4       IF<=0   RBI,R0,IND      // test b(i)>0
5       LOAD    RBA,RI,RAI      //leggi a(i)
6       MUL     RAI,RN,RT1      //calcola a(i)*N
8       SHL     RBI,#4,RT3      //calcola 16*b(i)
11      ADDI    RI,#1,RI        //incrementa l'indice
7       MOD     RT1,RBI,RT2     //calcola (a(i)*N)modb(i)
9       ADD     RT3,RT2,RT1     //somma i due termini sopra
12  IND: IF<    RI, RN,LOOP, delayed    //fine ciclo
10      STORE   RBC',RI,RT1     //memorizza il risultato
13      END
```

da cui abbiamo il seguente schema temporale, con le ipotesi dello schema precedente:



e pertanto il tempo di completamento del programma ottimizzato è di **16t**, con un risparmio di 4t rispetto al codice precedente. Ora risulta una efficienza di **12/16=75%.**