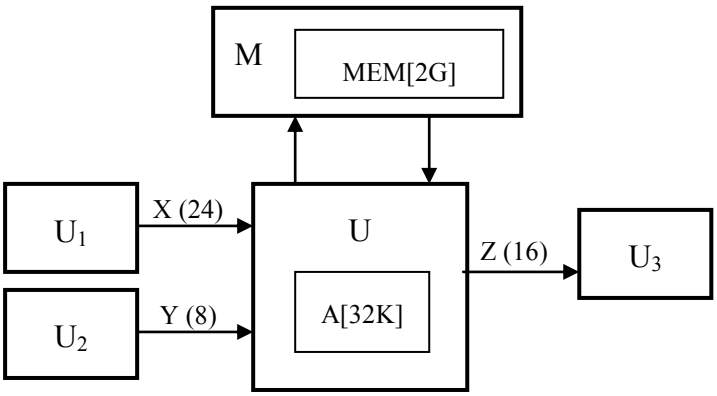
Architettura degli Elaboratori

Appello del 2 febbraio 2011

Riportare su tutti i fogli consegnati nome, cognome, numero di matricola, corso di appartenenza, e la sigla NEW (per nuovo ordinamento), oppure OLD-0 (per vecchio ordinamento, programma 2009-10), oppure OLD-1 (per vecchio ordinamento, programma 2008-09). I risultati verranno pubblicati sulle pagine web del corso/dei docenti appena disponibili.

Domanda 1 (tutti)

Si consideri il seguente sistema a livello firmware:



L'unità U contiene un componente logico memoria A di capacità 32K parole, e l'unità M un componente logico memoria MEM di capacità 2G parole. Le operazioni esterne di M sono lettura o scrittura di una parola su richiesta di U (per semplicità, senza esplicita informazione di esito). M ha ciclo di clock uguale a 11 volte quello di U. I collegamenti tra unità hanno latenza di trasmissione uguale a 4 cicli di clock di U.

Le informazioni contenute in MEM e in A sono logicamente organizzate in blocchi disgiunti, ciascuno costituito da 128 parole contigue.

Ricevendo da U_1 e U_2 le informazioni X (24 bit) e Y (8 bit) rispettivamente, U identifica l'operazione esterna come segue:

- se Y è uguale a due o tre dei byte di X: operazione esterna 0;
- se Y è uguale ad uno e solo uno dei byte di X: operazione esterna 1;
- se Y è diverso da ognuno dei tre byte di X: operazione esterna 2.

Le operazioni esterne hanno il seguente effetto:

- operazione esterna 0: copiare il blocco di MEM identificato da X nel blocco di A identificato da Y;
- operazione esterna 1: copiare il valore assoluto di ogni parola del blocco di A identificato da Y nelle corrispondenti parole del blocco di MEM identificato da X;
- operazione esterna 2: inviare a U_3 il numero di parole di A che hanno valore positivo e uguale a una potenza di 2.

I numeri sono rappresentati in complemento a due. È noto il ritardo di stabilizzazione t_p di una porta logica con al più 8 ingressi. Una ALU ha ritardo di stabilizzazione uguale a $5t_p$. Il componente logico memoria A è a singolo indirizzamento e ha tempo di accesso $4t_p$.

É richiesto il microprogramma di U con il requisito di *minimizzare il tempo medio di elaborazione dell'operazione esterna 2*, e la valutazione di tutti i tempi medi di elaborazione**.**

Prima del microprogramma deve essere riportata la spiegazione di quali scelte sono effettuate per soddisfare il suddetto requisito sui tempi di elaborazione.

Ogni rete combinatoria utilizzata, che sia diversa da una ALU o un commutatore, deve essere completamente definita e implementata.

Domanda 2 (NEW, OLD-0)

Per una macchina D-RISC con CPU pipeline, compilare con ottimizzazioni un programma corrispondente alla seguente computazione:

int A[M], B[M], C[M];

$\forall$ *i = 0 .. M-1:*

C[i] = if (A[i] > B[i])

then A[i] − B[i]

else B[i] − A[i]

con il vincolo che il calcolo del valore assoluto della differenza di due interi sia eseguito come procedura con parametri d'ingresso e di uscita passati via memoria.

Si assuma uguale a 0,5 la probabilità che A[i] > B[i].

Valutare il tempo di completamento per una architettura con cache primaria operante su domanda e blocco di 8 parole. La cache secondaria è on-chip, ha tempo di accesso di 2 cicli di clock, e si assume che abbia probabilità di fault trascurabile.

Domanda 2 (OLD-1)

Si consideri un programma parallelo LC contenente i processi A e B.

Il processo A è definito dalle seguente sequenza di azioni: esegue una lista di comandi F1; invia un valore intero N a B; invia un intero M a B; esegue una lista di comandi F2.

Il processo B è definito dalle seguente sequenza di azioni: esegue una lista di comandi F3; riceve da A un valore intero R; riceve da A un valore intero S; esegue una lista di comandi F4.

Supponendo che il processo A vada in esecuzione prima del processo B, si dica qual'é la sequenza di schedulazione dei due processi, nel caso il canale fra A e B sia di tipo sincrono oppure asincrono con grado di asincronia *k*, e che influenza ha il valore di *k* sulla schedulazione.

Si evidenzino quindi le differenze (a livello di pseudo-codice) nell'implementazione delle *send* e *receive* tra il caso in cui il canale è sincrono e quello in cui è asincrono con $k = 1$.

Soluzione

Domanda 1 (tutti)

Occorre realizzare, mediante una rete combinatoria, una funzione che,operando su ogni coppia di valori *X* e *Y* ricevuti da U_1 e U_2, identifichi l'operazione esterna da eseguire:

$$OP = F\ (X.byte_0,\ X.byte_1,\ X.byte_2,\ Y)$$

con *OP* di due bit (OP_0, OP_1), tali che:

$OP_0\ OP_1 = 0\ 0$ *:* Y è uguale a due o tre dei byte di X: *operazione esterna 0*

$OP_0\ OP_1 = 0\ 1$ *:* Y è uguale ad uno e solo uno dei byte di X: *operazione esterna 1*

$OP_0\ OP_1 = 1\ -$ *:* Y è diverso da ognuno dei tre byte di X: *operazione esterna 2*

La funzione *F* è definita nel seguente modo:

- vengono confrontati contemporaneamente i tre byte di *X* con il byte *Y*, e gli 8 bit di ogni singolo risultato messi in OR. Siano W_0, W_1, W_2 le tre variabili booleane risultanti:

$$i = 0, 1, 2:\ W_i = OR\ (X.byte_i \oplus Y)$$

- dalla specifica delle tre operazioni esterne si ricava una tabella di verità con variabili d'ingresso W_0, W_1, W_2 e variabili di uscita OP_0, OP_1. Da essa si ottengono le espressioni logiche (del tutto intuitive applicando direttamente le specifiche):

$$OP_0 = W_0\ W_1\ W_2$$

$$OP_1 = \overline{W_0}\ W_1\ W_2 + W_0\ \overline{W_1}\ W_2 + W_0\ W_1\ \overline{W_2}$$

La realizzazione di *F* è una rete combinatoria con ritardo di stabilizzazione

$$T_F = 2t_p \text{ (confrontatori)} + t_p \text{ (porta OR a 8 ingressi)} + 2t_p = 5t_p$$

Effettuando la decodifica del codice operativo nella prima microistruzione con variabili di condizionamento complesse *F(X.byte0, X.byte1, X.byte2, Y)*, nella valutazione del ciclo di clock avremmo almeno $T_{\omega PO} = 5t_p$.

Le operazioni esterne 0 e 1 non hanno particolari problemi dal punto di vista della scrittura del microprogramma ottimo, in entrambi i casi trattandosi di un loop ripetuto 128 volte con variabili di condizionamento semplici. Nell'operazione 1 il calcolo del valore assoluto viene effettuato con una rete combinatoria:

abs(num) = if (bit_segno(num) = 0) then num else negazione_complemento_a_due(num)

così da non introdurre inutilmente una variabile di condizionamento complessa. La funzione è realizzabile con una ALU controllata dal bit del segno dello stesso operando *num*.

L'operazione esterna 2 consta di un loop ripetuto 32K volte. Per minimizzarne il tempo medio di elaborazione, occorre anzitutto fare uso di una rete combinatoria che realizzi la funzione booleana:

$$P = G(a) = if\ (a > 0)\ and\ (a = 2^h)\ |\ 0 \le h \le 30$$

con *a* contenuto della generica locazione di *A*.

Dalla proprietà delle potenze di 2 si ottiene quindi la funzione G come OR di 31 termini AND, ognuno di 32 variabili (i bit di *a*), tutti i termini AND aventi a_{31} negato, ed ogni termine AND avente una sola variabile affermata e tutte le altre negate:

$$P = \overline{a_{31}}\ a_{30}\ \overline{a_{29}}\ \ldots\ \overline{a_0} + \overline{a_{31}}\ \overline{a_{30}}\ a_{29}\ \ldots\ \overline{a_0} + \ \ldots + \overline{a_{31}}\ \overline{a_{30}}\ \overline{a_{29}}\ \ldots\ a_0$$

La rete combinatoria corrispondente ha ritardo $2t_p$ (livello AND) + $2t_p$ (livello OR), a cui va sommato il tempo di accesso nella memoria *A*: complessivamente *almeno* $T_G = 8t_p$ ("almeno": a seconda che esistano altri ritardi significativi, ad esempio per l'indirizzamento di *A*).

Per minimizzare il tempo medio di elaborazione dell'operazione 2, occorre inoltre che il corpo del loop consti di una sola microistruzione. Questo si può ottenere

i) con una variabile di condizionamento complessa, nella quale si valuti la funzione *G*,

ii) con una variabile di condizionamento semplice, scrivendo il valore di *G* in un registro di un bit *P*, ed anticipandone opportunamente l'esecuzione di un ciclo di clock.

La prima soluzione comporta che, nella valutazione del ciclo di clock, sarebbe almeno $T_{\omega PO} = 8t_p$, che è ancora maggiore del tempo di stabilizzazione della funzione *F*.

In conclusione, la soluzione che minimizza il tempo di elaborazione dell'operazione esterna 2 è la *ii)*, a condizione che

- *nella valutazione del ciclo di clock sia* $T_{\omega PO} = 0$.

Questo si ottiene impiegando *due cicli di clock per la ricezione di X, Y e la decodifica del codice operativo*, utilizzando il primo ciclo per scrivere il valore della funzione *F* in un registro *OP* di 2 bit. Il ciclo di clock speso in più non ha alcun effetto sui tempi medi di elaborazione, *anzi*: è fondamentale per minimizzare il tempo medio di elaborazione dell'operazione esterna 2, e minimizza anche i tempi di elaborazione delle operazioni 0 e 1 in quanto permette di minimizzare il ciclo di clock (come detto, il loop nelle operazioni 0 e 1 impiega solo variabili di condizionamento semplici).

Il microprogramma che ne deriva è il seguente:

****** *Attesa X e Y, decodifica e inizializzazione. L'interfaccia verso la memoria usa il registro IND per l'indirizzamento. Il registro I è di 8 bit, per controllare il loop delle operazioni 0 e 1. Il registro J è di 16 bit, per controllare il loop dell'operazione 2. C contiene il risultato dell'operazione 2.*

0. (RDY_1, RDY_2 = 0 0, 0 1, 1 0) nop, 0;

   (= 1 1) reset RDY_1, set ACK_1, reset RDY_2, set ACK_2, *F(X, Y)* $\rightarrow$ *OP*, X ° *settezeri* $\rightarrow$ INDM,

   Y ° *settezeri* $\rightarrow$ INDA, 1

1. (OP_0, OP_1 = 0 0) INDM $\rightarrow$ IND, 'read' $\rightarrow$ OPM, set RDYOUTM, 1 $\rightarrow$ I, INDM + 1 $\rightarrow$ INDM, 2;

   (= 0 1) INDM $\rightarrow$ IND, *abs*(A[INDA]) $\rightarrow$ DATAOUT, 'write' $\rightarrow$ OPM, set RDYOUTM, 1 $\rightarrow$ I,

   INDM + 1 $\rightarrow$ INDM, 3;

   (= 1 –) *G(A[0])* $\rightarrow$ *P*, 1 $\rightarrow$ INDA, 1 $\rightarrow$ J, 0 $\rightarrow$ C, 4

****** *Operazione esterna 0*

2. (I_0, RDYINM = – 0) nop, 2;

   (= 0 1) reset RDYINM, DATAIN $\rightarrow$ A[INDA], INDM $\rightarrow$ IND, 'read' $\rightarrow$ OPM, set RDYOUTM,

   I + 1 $\rightarrow$ I, INDM + 1 $\rightarrow$ INDM, INDA + 1 $\rightarrow$ INDA, 2;

   (= 1 1) reset RDYINM, DATAIN $\rightarrow$ A[INDA], 0

****** *Operazione esterna 1*

3. (I_0, RDYINM = – 0) nop, 3;

   (= 0 1) reset RDYINM, INDM $\rightarrow$ IND, *abs*(A[INDA]) $\rightarrow$ DATAOUT, 'write' $\rightarrow$ OPM,

   set RDYOUTM, I + 1 $\rightarrow$ I, INDM + 1 $\rightarrow$ INDM, INDA + 1 $\rightarrow$ INDA, 3;

   (= 1 1) reset RDYINM, 0

****** *Operazione esterna 2*

4. (J_0, P, ACK_3 = 0 0 –) J + 1 → J, INDA + 1 → INDA, *G(A[INDA])* → *P*, 4;

   (= 0 1 –) J + 1 → J, INDA + 1 → INDA, *G(A[INDA])* → *P*, C + 1 → C, 4;

   (= 1 – 0) nop, 4;

   (= 1 0 1) reset ACK_3, set RDY_3, C → OUT3, 0;

   (= 1 1 1) reset ACK_3, set RDY_3, C + 1 → OUT3, 0;

Trascurando i cicli di clock spesi nelle microistruzioni 0 e 1, i tempi medi di elaborazione valgono:

$$T_0 = T_1 = 128\ (\tau + t_a) = 128\ (\tau + 2T_{tr} + \tau_M) = 2560\ \tau$$

$$T_2 = 32K\ \tau$$

Per la valutazione del ciclo di clock, abbiamo $T_{\omega PO} = 0$. Una volta appurato facilmente che $T_{\omega PC} = T_{\sigma PC} = 2t_p$, valutiamo $T_{\sigma PO}$: il numero di ALU necessarie per il parallelismo delle micro operazioni è 3, usando tutte ALU a 32 bit e completando con opportuni zeri i numeri rappresentati con meno di 32 bit; tali ALU hanno quindi commutatori in ingresso. L'operazione più costosa è *abs(A[INDA])*, che comporta i seguenti ritardi in serie: attraversamento di un commutatore per l'indirizzamento di *A* (con la costante *0* oppure con il contenuto di *INDA*), lettura di *A*, attraversamento del commutatore di una ALU e ritardo di tale ALU. Quindi,

$$T_{\sigma PO} = 13\ t_p$$

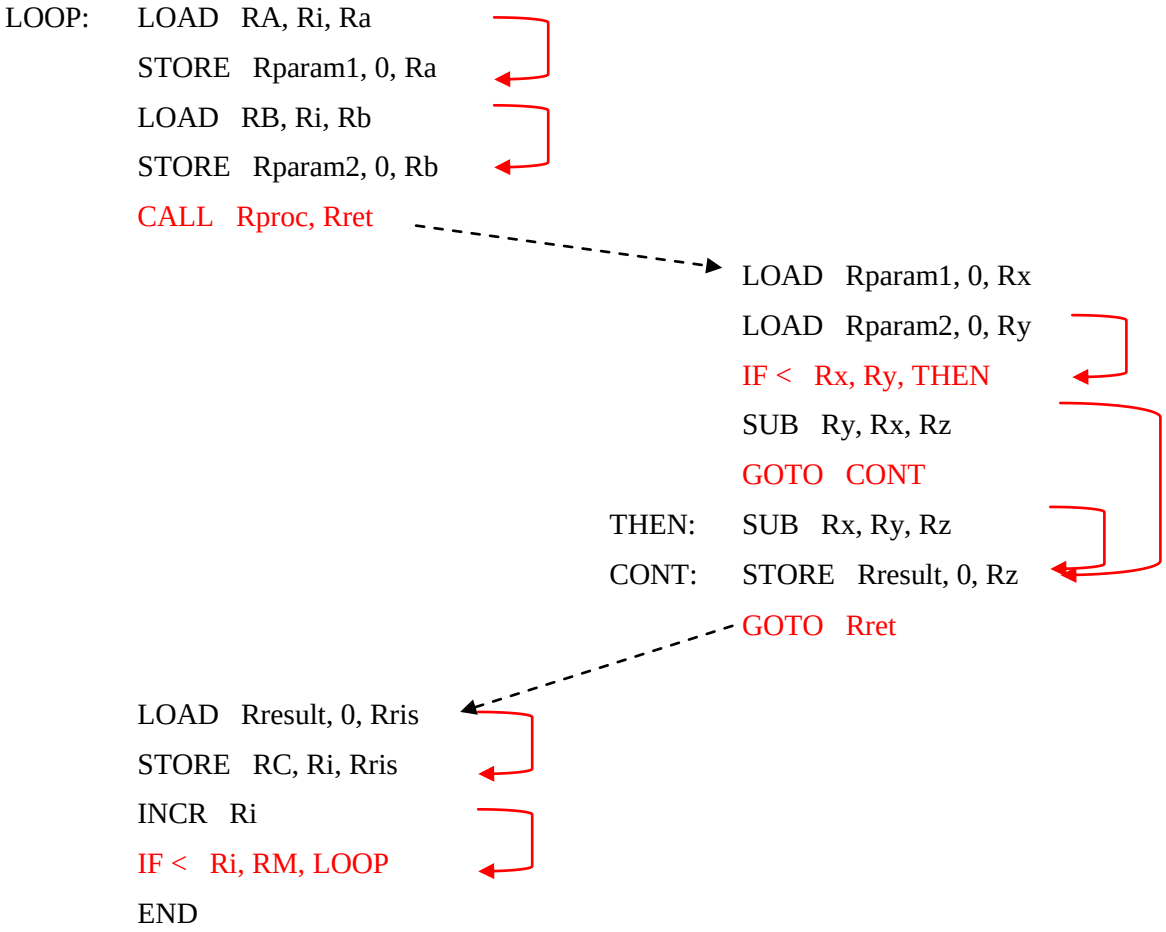
da cui

$$\tau = 16\ t_p$$

Domanda 2 (OLD-0)

Vediamo prima la compilazione non ottimizzata. La procedura si aspetta i valori interi dei due parametri d'ingresso nelle locazioni di memoria virtuale i cui indirizzi sono nei registri generali **Rparam1** e **Rparam2**, e scrive il valore intero del risultato nella locazione di memoria virtuale il cui indirizzo è nel registro **Rresult**.

Con le usuali notazioni e inizializzazioni dei registri:



Sono evidenziate in rosso tutte le numerose situazioni di dipendenze logiche (quasi tutte di distanza 1, quasi tutte con NQ = 2) e di salti.

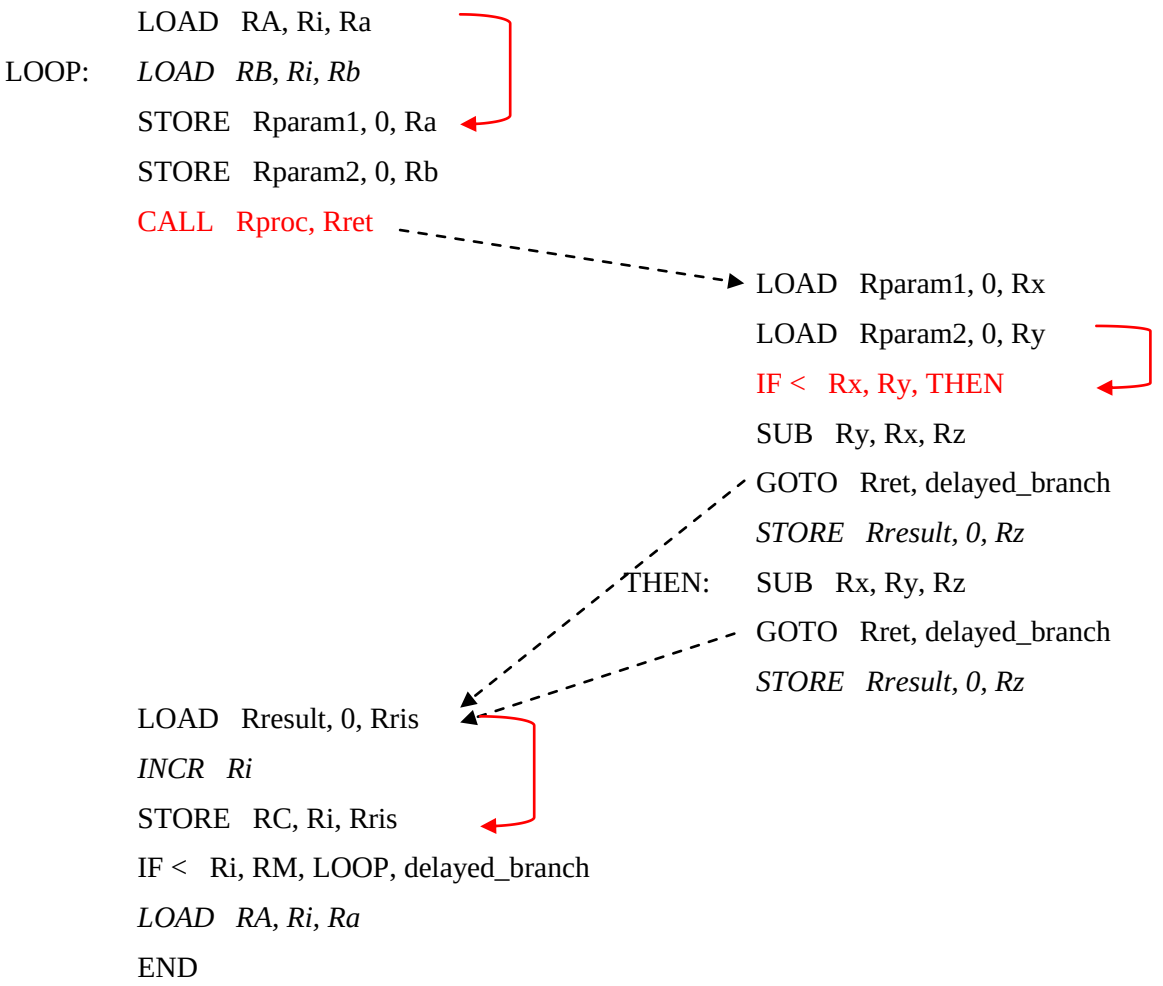
Le ottimizzazioni possono essere le seguenti. Nel programma principale:

- inframmezzare le prime LOAD e STORE per il passaggio dei parametri;
- anticipare la INCR Ri prima della STORE RC, inizializzando la base di C al valore immediatamente precedente;
- applicare il delayed branch della IF < Ri, spostandovi dopo la LOAD RA.

Nella procedura:

- eliminare l'istruzione GOTO CONT, replicando codice (STORE Rresult, GOTO Rret) nei rami *then* e *else*;
- applicare il delayed branch a GOTO Rret, spostandovi dopo la STORE Rresult.

La compilazione diviene la seguente:



Non sono indicate le dipendenze logiche intrecciate o annidate che non hanno effetto, così come non hanno effetto le dipendenze delle SUB sulle STORE Rresult nella procedura (distanza 2 con NQ = 1).

Rimangono due dipendenze logiche di distanza 2 con NQ = 2 (LOAD RA su STORE Rparam1, e LOAD Rresult su STORE RC), e una di distanza 1 con NQ = 2 (nella procedura LOAD Rparam2 su IF <).

Rimane il salto della IF < nella procedura e quello della CALL.

Qualunque sia l'esito della IF <, le istruzioni eseguite per ogni iterazione sono complessivamente:

$$N_{iter} = 16$$

La probabilità che la IF < provochi un salto è uguale a (1/16)∗(1/2), quella della CALL è 1/16, quindi:

$$\lambda = (1/16) * (3/2)$$

La probabilità delle dipendenze logiche di distanza 2, con $NQ_2 = 2$, è uguale a

$$d_2 = 2/16$$

quella delle dipendenze logiche di distanza 1, con $NQ_1 = 2$, è:

$$d_1 = 1/16$$

Applicando il modello dei costi del tempo di servizio per istruzione:

$$T = t\,(1 + \lambda) + t\,d_1\,(NQ_1 + 1 - 1) + t\,d_2\,(NQ_2 + 1 - 2) = t\ 43/32$$

$\varepsilon' = 32/43 = 0{,}74$: efficienza relativa che tiene conto del solo funzionamento pipeline

Il tempo di completamento, *in assenza di fault di cache*, è espresso da:

$$T_{c\text{-}pipe} = 16\,M\,T = 43\,M\,\tau$$

La *valutazione dell'effetto dei fault di cache primaria* è semplice. L'insieme di lavoro è rappresentato dal blocco corrente di A, di B e di C. Poiché C è in sola scrittura, i fault su C non provocano trasferimento di blocco. Trascurando i pochissimi fault di cache per le istruzioni, si ha:

$$T_{fault} = N_{fault} * T_{trasf} = 2\frac{M}{\sigma} * 2\,\sigma\,\tau = 4\,M\,\tau$$

Il tempo di completamento effettivo è quindi:

$$T_c = T_{c\text{-}pipe} + T_{c\text{-}fault} = 47\,M\,\tau$$

con una efficienza relativa complessiva, che tiene conto sia del funzionamento pipeline che del funzionamento della cache, data da:

$$\varepsilon = \frac{T_{c-id}}{T_c} = \frac{32\,M\,\tau}{47\,M\,\tau} = 0{,}68$$

Domanda 2 (OLD-1)

Esiste un unico canale CH di tipo intero da A verso B, sul quale avvengono quindi entrambe le comunicazioni:

parallel A, B;

A :: **channel out** CH; …; **int** N, M; …

{ F1; **send** (CH, N); **send** (CH, M); F2 }

B :: **channel in** CH (*k*); …; **int** R, S; …

{ F3; **receive** (CH, R); **receive** (CH, S); F4 }

Caso del canale sincrono: k = 0

Quando A, in esecuzione per primo, esegue la *send* su CH e si blocca, poiché B non è ancora andato in esecuzione. Quando B andrà in esecuzione, eseguirà la prima *receive* senza bloccarsi e svegliando A. B si bloccherà sulla seconda *receive*, dal momento che A, sbloccato ma ancora presumibilmente nella coda ready, non avrà ancora invocato la seconda *send*. Quindi passerà in esecuzione A e quando eseguirà la seconda *send* non si bloccherà e provocherà il passaggio di B dallo stato di attesa allo stato di pronto.

Caso del canale con grado di asincronia k ≥ 1

Se *k > 1*, entrambe le *send* avvengono senza provocare il blocco di A.

Con *k = 1* la prima *send* non provocherà blocco, mentre sulla seconda A si bloccherà. A questo punto andrà in esecuzione B e riuscirà a completare le due *receive* senza bloccarsi. L'esecuzione della seconda *receive* provocherà il passaggio del processo A dallo stato di attesa allo stato di pronto.

Per le differenze a livello di pseudo-codice nelle implementazioni di *send* e *receive* fra il caso del canale sincrono e quello del canale asincrono con *k* = 1, si veda il libro di testo.