Architettura degli elaboratori—A.A.2018-19—Appello 22 luglio 2019

Riportare in alto a destra di ogni foglio consegnato Nome, Cognome, Matricola e corso di appartenenza (A o B). La traccia di correzione, i risultati e il calendario degli orali saranno resi noti via web (didawiki e/o pagine docenti) appena disponibili.

Domanda 1

Si consideri il seguente frammento di pseudocodice, per il calcolo del prodotto di due polinomi p e q:

```
int p[N], q[N], r[2N-1];

for(int i=0; i<2N-1; i++)
  {r[i]=0}
for(int i=0; i<N; i++) {
  for(int h=0; h<N; h++) {
    r[i+h] = r[i+h] + p[i]*q[h];
  }
}
```

Si consideri un'architettura D-RISC Pipeline, con EU parallela (EU master per le aritmetico logiche corte ed EU slave da 4 stadi per la moltiplicazione e divisione fra interi). Si richiede:

- la compilazione in Assembler D-RISC secondo le regole standard;
- l'individuazione di tutte le cause di degrado delle prestazioni;
- di valutare il tempo di completamento ideale e reale del ciclo più interno del frammento del programma;
- l'individuazione di possibili ottimizzazioni;
- la valutazione del guadagno di prestazioni ottenuto
- l'individuazione del working set (trascurando l'inizializzazione del vettore R).

Domanda 2

Si progetti una unità firmware U contenente una memoria M di 1K parole da 32 bit, in cui sono presenti N dati significativi (N>1); i dati significativi sono numeri interi positivi, e devono occupare sempre le prime N posizioni di M. U interagisce con due unità, U_1 ed U_2 : U_1 richiede di eliminare da M il dato memorizzato nella parola di memoria di indirizzo IND (IND è fornito da U_1 ad U, ed è compreso tra 0 e 1023) se tale dato è un multiplo di 256, mentre U_2 richiede ad U di fornirgli il numero degli elementi presenti in memoria il cui valore è dispari. In caso di richieste contemporanee, viene data la precedenza alla richiesta di U_1. Nel caso di eliminazione di un elemento, la porzione dei dati significativi va ricompattata in modo da soddisfare il vincolo che vuole i dati significativi sempre nelle prime posizioni della memoria interna.

Si progetti l'unità U, fornendo il microprogramma utilizzato, e se ne determini la lunghezza del ciclo di clock, supponendo di avere a disposizione porte logiche ad 8 ingressi che si stabilizzano in 1 t_p, e solamente due ALU che fanno addizioni e sottrazioni. È richiesto che l'unità minimizzi il numero di microistruzioni necessarie all'esecuzione delle operazioni esterne.

Bozza di soluzione

Domanda 1

Compilazione (regole standard).

```
        CLEAR Ri
        ADD Rn, Rn, R2n
        SUB R2n, #1, R2n1
Lp1:    STORE Rbaser, Ri, R0
        INC Ri
        IF< Ri, R2n1, Lp1
        CLEAR Ri
LPi:     CLEAR Rh
LPh:    LOAD Rbasep, Ri, R1
        LOAD Rbaseq, Rh, R2
        MUL R1, R2, R3
        ADD Ri, Rh, Rih
        LOAD Rbaser, Rih, R4
        ADD R4, R3, R5
        STORE Rbaser, Rih, R5
        INC Rh
        IF< Rh, Rn, LPh
        INC Ri
        IF < Ri, Rn, LPi
Cont:   …
```

Degradi delle prestazioni

Ci concentriamo sul ciclo interno:

1. LPh: LOAD Rbasep, Ri, R1
2. LOAD Rbaseq, Rh, R2
3. MUL R1, R2, R3
4. ADD Ri, Rh, Rih
5. LOAD Rbaser, Rih, R4
6. ADD R4, R3, R5
7. STORE Rbaser, Rih, R5
8. INC Rh
9. IF< Rh, Rn, LPh

Possiamo notare dipendenze IU-EU 4->5,6->7 e 8->9, mentre abbiano una dipendenza EU-EU 3->6. Inoltre il salto alla 9 (prevalentemente preso) causerà una bolla da salto.

Tempo di completamento ideale

Il ciclo interno è composto da 9 istruzioni e dovrebbe quindi completare in 9t.

Tempo di completamento

L'effetto delle dipendenze logiche sul codice del loop interno è riassunto dalla simulazione che segue:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
IM	LOAD	LOAD	MUL	ADD	LOAD			ADD	STORE			INC	IF<	XXX	LOAD			
IU		LOAD	LOAD	MUL	ADD	LOAD	LOAD	LOAD	ADD	STORE	STORE	STORE	INC	IF<	XXX	LOAD		
DM			LOAD	LOAD					LOAD				STORE				LOAD	
EUmaster				LOAD	LOAD	MUL	ADD			LOAD	ADD			INC				LOAD
EUslave							MUL	MUL	MUL	MUL								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
IM	LOAD	LOAD	MUL	ADD	LOAD			ADD	STORE			INC	IF<	XXX	LOAD			
IU		LOAD	LOAD	MUL	ADD	LOAD	LOAD	LOAD	ADD	STORE	STORE	STORE	INC	IF<	XXX	LOAD		
DM			LOAD	LOAD					LOAD				STORE				LOAD	
EUmaster				LOAD	LOAD	MUL	ADD			LOAD	ADD			INC				LOAD
EUslave							MUL	MUL	MUL	MUL								

Il ciclo interno richiede 15t invece dei 9t del tempo ideale, con un'efficienza pari quindi a 3/5.

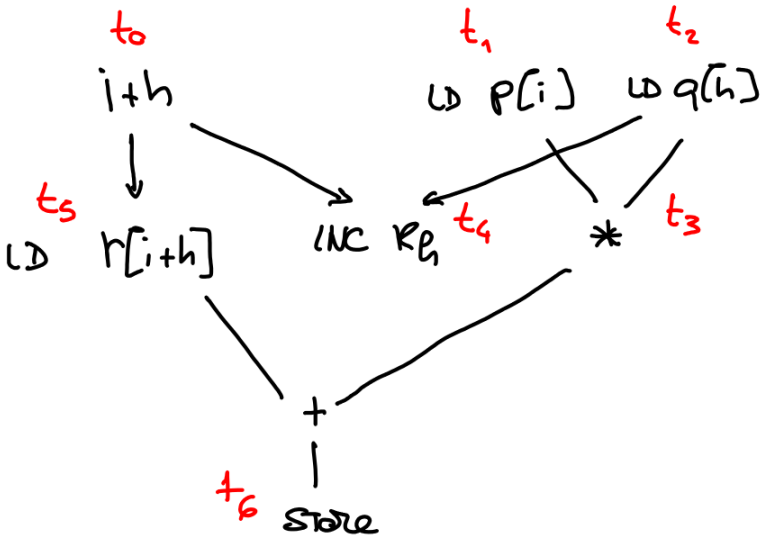
Tempo di completamento effettivo

Al tempo di completamento così calcolato vanno aggiunti i costi dei fault in cache. Gli array P e Q sono acceduti sequenzialmente. Q è soggetto a riuso, P viene utilizzato una volta sola ma i suoi elementi sono utilizzati ognuno per N iterazioni del ciclo for h. L'array R è acceduto sequenzialmente ed è soggetto a riuso. Nella prima iterazione for i si utilizzano le prime N posizioni, nella iterazione (i) successiva le N posizioni a partire dalla seconda e così via. Nell'ipotesi che la cache di primo livello sia sufficientemente ampia per contenere sia Q che N posizioni di R, abbiamo un totale di $3N/\sigma$ fault della cache di primo livello. L'esecuzione del ciclo interno prevederà quindi N/s fault alla prima iterazione per Q, 1 fault per P e N/s per R. Il tempo di trattamento dei fault va sommato al tempo di completamento precedentemente computato. Non essendo note le caratteristiche del sottosistema di memoria, possiamo considerare un termine aggiuntivo pari a (2N/s+1)*Tfault

Ottimizzazioni

Possiamo considerare di spostare alcune istruzioni nel codice e di utilizzare il salto ritardato. L'analisi del codice del ciclo più interno rileva le dipendenze in nero nel grafo qui sotto. In rosso è riportata una delle sequenze ammissibili di esecuzione, ovvero una sequenza di operazioni che rispetta tutte le dipendenze e che include tutti i nodi del grafo.

$$r[i+h] = r[i+h] + p[i] * q[h]$$



Il codice del ciclo più interno potrebbe quindi essere trasformato come segue:

LPh: ADD Ri, Rh, Rih
LOAD Rbasep, Ri, R1
LOAD Rbaseq, Rh, R2
MUL R1, R2, R3
INC Rh
LOAD Rbaser, Rih, R4
ADD R3, R4, R5
IF< Rh, Rn, LPh, delayed
STORE Rbaser, Rih, R5

Questo codice riduce il tempo richiesto per l'esecuzione della singola iterazione a 12t.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IM	ADD	LOAD	LOAD	MUL	INC	LOAD	ADD	IF<	STORE				LOAD			
IU		ADD	LOAD	LOAD	MUL	INC	LOAD	ADD	IF<	STORE	STORE	STORE	STORE	LOAD		
DM				LOAD	LOAD			LOAD							LOAD	
EUmaster			ADD		LOAD	LOAD	MUL	INC	LOAD	ADD	ADD	ADD				LOAD
EUslave								MUL	MUL	MUL	MUL					

Questo sposta l'efficienza da 3/5 a 4/5.

Un'esecuzione alternativa poteva essere quest'altra:

LPh: ADD Ri, Rh, Rih
LOAD Rbasep, Ri, R1
LOAD Rbaseq, Rh, R2
MUL R1, R2, R3

LOAD Rbaser, Rih, R4
INC Rh
ADD R3, R4, R5
IF< Rh, Rn, LPh, delayed
STORE Rbaser, Rih, R5

Con lo stesso effetto sul tempo di completamento della singola iterazione:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IM	ADD	LOAD	LOAD	MUL	LOAD	INC	ADD	IF<		STORE			LOAD			
IU		ADD	LOAD	LOAD	MUL	LOAD	INC	ADD	IF<	IF<	STORE	STORE	STORE	LOAD		
DM				LOAD	LOAD		LOAD							STORE	LOAD	
EUmaster			ADD		LOAD	LOAD	MUL	LOAD	INC	ADD	ADD	ADD				LOAD
EUslave								MUL	MUL	MUL	MUL					

Domanda 2

L'interfaccia dell'unità U con U 1 è costituita da RDY1 e da un registro a 10 bit IND (in ingresso) e da ACK1 (in uscita), mentre quella con U 2 è costituita da RDY2 (in ingresso), e da ACK2 e da un registro a 10 bit RIS (in uscita).

Si utilizzeranno i registri a 10 bit UE, che contiene l'indirizzo in M dell'ultimo elemento significativo, I usato per scorrere la memoria, ed A, per contare il numero di dati di valore dispari. Il confronto tra due dati A e B si farà mediante il segno di A-B, e il test per sapere se un dato C è multiplo di 256 o meno si farà mediante l'OR degli 8 bit meno significativi di C (questo è 0 se e solo se C è multiplo di 256), chiamato OR8() nel seguito. Per sapere se un elemento di M è dispari o meno, basta guardare il suo bit meno significativo (che vale 1 se e solo se è dispari): nel seguito sarà indicato con UB(M[I]).

Il microcodice è il seguente:

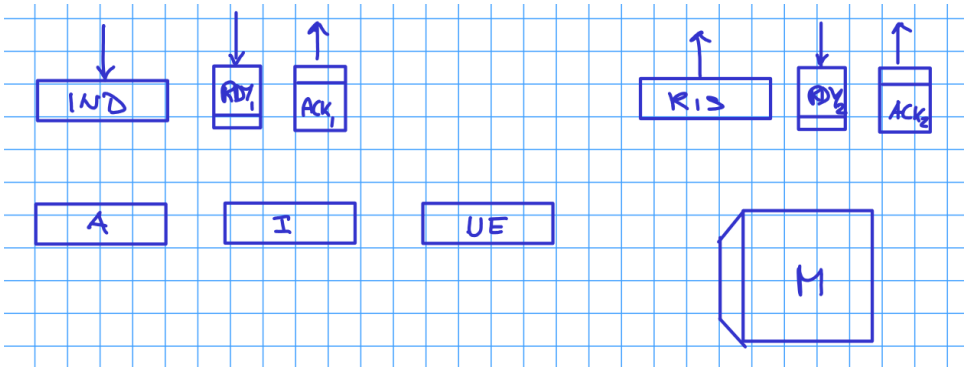
0. (RDY1,RDY2,segno(UE-IND),OR8(M[IND])=0 0 - -) NOP, 0 //non ci sono richieste
   (=1 - 1 -) reset RDY1, set ACK1, 0 // richiesta da U 1 ma dato non esiste
   (=1 - 0 1) reset RDY1, set ACK1, 0 // richiesta da U 1 ma dato non multiplo
   (=1 - 0 0) M[IND+1] →M[IND] , IND+1→I , 1 //richiesta da U 1 , inizio ciclo compattamento
   (=0 1 - -) 0 →I , 0→A , 2 //richiesta da U 2 , inizio ciclo conteggio elementi dispari
1. (segno(UE-I)=0) M[I+1] →M[I], I+1→I , 1 // esistono elementi significativi, si compattamento
   (=1) reset RDY1, set ACK1, UE-1→UE, 0 //fine elementi significativi
2. (segno(UE-I), UB(M[I])=0 0) I+1→I, 2 //elemento pari
   (=0 1) I+1→I, A+1→A, 2 // elemento dispari
   (=1 -) reset RDY2, set ACK2, A →RIS, 0 // fine operazioni, restituzione del risultato

La minimalità del numero di istruzioni eseguite è determinata dal fatto che i controlli necessari e le inizializzazioni dei cicli sono fatte tutte nelle frasi della 1. Si testano variabili di condizionamento complesse che potevano essere eliminate solo al prezzo di introdurre un'altra microistruzione per l'esecuzione delle operazioni esterne.

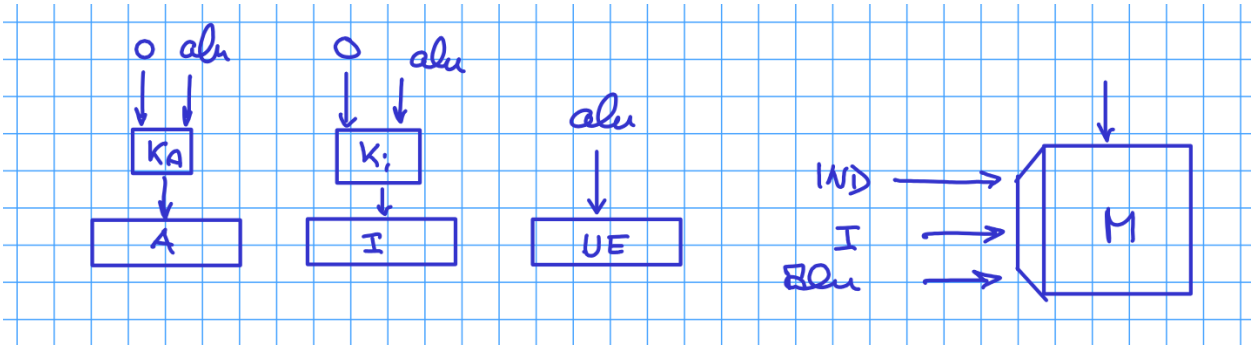
La memoria M deve essere una memoria a "doppia porta". Servono (internamente alla memoria M) tre commutatori per la lettura di M[IND] e M[I] senza commutatori esterni (condizione di correttezza per le

variabili di condizionamento nella 0. e nella 2.) e un commutatore per leggere M[I+1], oltre al selettore comandato dall'ingresso su cui si trova I per le scritture.

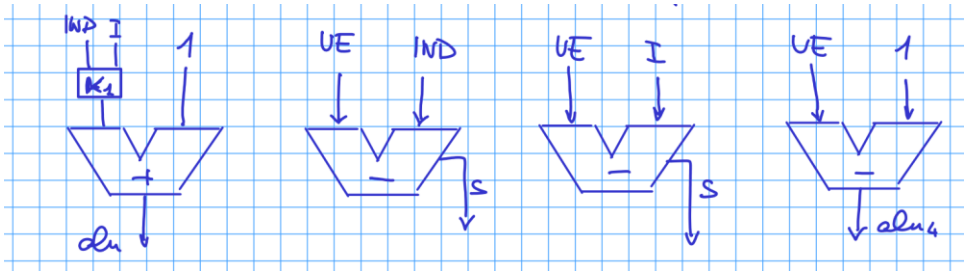
Risorse di stato



Ingressi delle risorse



Risorse di calcolo



Parte operativa:

- TsigmaPO: tempo della microistruzione più lenta (la 0, ad esempio): $t_M + t_{ALU} + t_K$ (cioè tempo di accesso alla memoria+tempo della alu+tempo del commutatore).
- TomegaPO: segno richiede t ALU mentre OR8 si fa con una porta OR e quindi 1 t_p ; le altre variabili di condizionamento richiedono tempo 0.

Parte controllo:

- abbiamo 3 microistruzioni e quindi 2 bit di stato. Inoltre, ci sono 10 frasi e 5 variabili di condizionamento, di cui al massimo 4 testate contemporaneamente. Quindi abbiamo un solo livello di porte AND ed un solo livello di porte OR (se abbiamo più di 5 valori ad 1, codifichiamo gli "0" e complementiamo). Quindi TsigmaPC = TomegaPC = $2t_p$;

Pertanto:

Tau = t_{ALU} + max {$2t_p$; t_M +t_{ALU} +t_K +2tp } + delta = t_M +$2t_{ALU}$ + t_K +$3t_p$