Architettura degli elaboratori – A. A. 2014-2015
Terzo appello – 17 luglio 2015

Indicare Nome Cognome Matricola e Corso (A/B) su tutti i fogli consegnati.
Risultati, bozza di correzione e calendario degli orali su web appena disponibili

Domanda 1

Una unità firmware U_s implementa uno *stack* di 128 posizioni per l'interconnessione di tre altre unità U_a, U_b ed U_c. U_a ed U_b possono eseguire operazioni di *push* di una parola, mentre U_c può eseguire operazioni di *pop* di una parola.

Si richiede la realizzazione di un arbitraggio fair fra U_a e U_b evitando comunque attese inutili di una unità e, al contempo, di dare priorità alle operazioni di *pop*.

Dell'unità si fornisca microcodice e tempo di servizio, facendo tutte le debite assunzioni su tempi di eventuali ALU e memorie utilizzate nella Parte Operativa.

Domanda 2

Per il seguente codice assembler, operante sui vettori A e P, ciascuno da N=8K posizioni, di interi:

1. ADD R_0, #1, R_{prefix}
2. ADD R_0, R_0, R_i
3. loop: LOAD R_{baseA}, R_i, R_{ai}
4. MUL R_{ai}, R_{prefix}, R_{prefix}
5. STORE R_{baseP}, R_i, R_{prefix}
6. INC R_i
7. IF< R_i, R_N, loop
8. END

si forniscano

- la traccia degli accessi in memoria principale per le prime iterazioni del ciclo, assumendo che il vettore P, il vettore A ed il codice siano allocati in memoria virtuale consecutivamente a partire dall'indirizzo 0 e che R_{baseA}, R_{baseP} ed R_N siano inizializzati ai valori corretti
- il tempo di completamento su processore D-RISC, dotato di cache di primo livello associativa di insiemi (1K insieme da 4 linee di σ=16 parole ciascuna) e di memoria principale dotata di 1M pagine da 4K, con unità di esecuzione per moltiplicazione/divisione fra interi a 4 stadi,
- una possibile ottimizzazione del codice, con valutazione quantitativa del guadagno in termini di tempo di completamento.

Domanda 3

Si indichi il costo temporale della gestione di un'interruzione sul processore D-RISC monolitico come descritto nella prima parte del corso.

Traccia di soluzione

Domanda 1

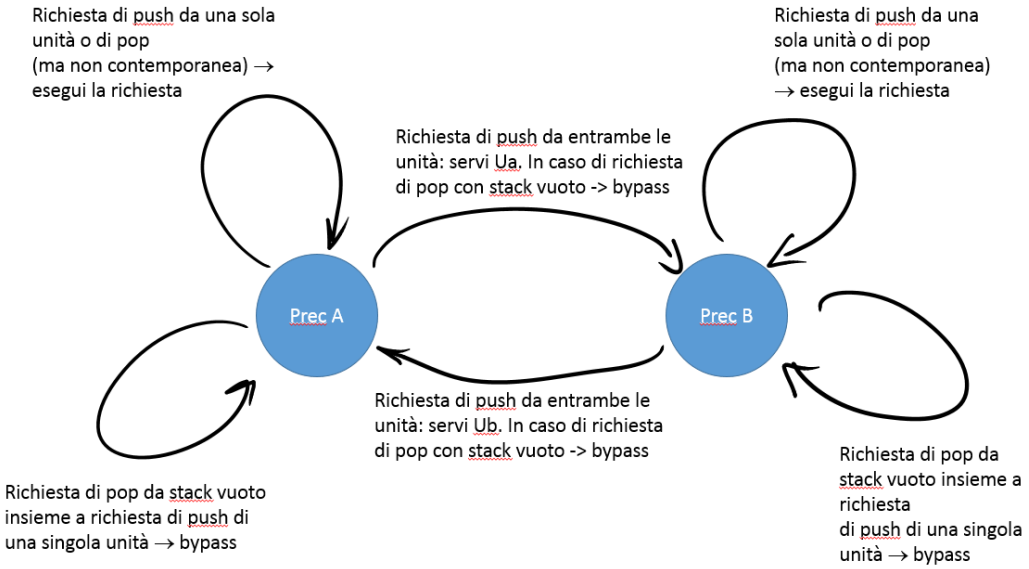
Utilizziamo un registro SP da 8 bit come stack pointer, ovvero come indicatore della prima posizione libera sullo stack. SP=0 determina la condizione di stack vuoto. SP_0=1 determina la condizione di stack pieno. Lo stack è implementato con un modulo di memoria da 128 parole. L'interfaccia verso U_a e U_b contiene:

RDY_x, VAL_x (1 parola) in ingresso, ACK_x in uscita ($x \in [a,b]$)

L'interfaccia verso U_c contiene

RDY_c in ingresso, VAL_c (1 parola) e ACK_c in uscita.

Per implementare una politica fair per i push immaginiamo di avere due stati: uno in cui in caso di conflitto si privilegia Ua e uno in cui si privilegia Ub. In caso di conflitto si transita nell'altro stato:



Il microcodice potrebbe dunque essere scritto come segue:

0. // precedenza alle push da Ua
   (RDYa, RDYb, RDYc,or(SP),SP0=000--) nop, 0
   (=100-0) VALa →M[SP], SP+1→SP, set ACKa, reset RDYa, 0
   (=100-1) nop,0
   (=010-0) VALb →M[SP], SP+1→SP, set ACKb, reset RDYb, 0
   (=010-1) nop, 0
   (=--111) M[SP-1] → VALc, SP-1→SP, set ACKc, reset RDYc, 0
   (=1010-) VALa → VALc, set ACKa, reset RDYa, set ACKc, reset RDYc, 0
   (=0110-) VALb → VALc, set ACKb, reset RDYb, set ACKc, reset RDYc, 0
   // arbitraggio
   (=1110-) VALa → VALc, set ACKa, reset RDYa, set ACKc, reset RDYc, 1
   (=110-0) VALa →M[SP], SP+1→SP, set ACKa, reset RDYa,1
   (=110-1) nop, 0
1. // precedenza alle push da Ub
   (RDYa, RDYb, RDYc,or(SP),SP0=000--) nop, 1
   (=100-0) VALa →M[SP], SP+1→SP, set ACKa, reset RDYa, 1
   (=100-1) nop,1
   (=010-0) VALb →M[SP], SP+1→SP, set ACKb, reset RDYb, 1
   (=010-1) nop, 1
   (=--111) M[SP-1] → VALc, SP-1→SP, set ACKc, reset RDYc, 1

(=1010-) VALa → VALc, set ACKa, reset RDYa, set ACKc, reset RDYc, 1
(=0110-) VALb → VALc, set ACKb, reset RDYb, set ACKc, reset RDYc, 1
(=--10-) nop, 1
// arbitraggio
(=1110-) VALb → VALc, set ACKb, reset RDYb, set ACKc, reset RDYc, 0
(=110-0) VALb →M[SP], SP+1→SP, set ACKb, reset RDYb,0
(=110-1) nop, 1

Realizzazione PC

Abbiamo 2 stati, 5 variabili di condizionamento (ingressi), 22 frasi. Dunque sia per la ωPC che per la σPC basta un livello AND ma occorre un doppio livello di porte OR. Dunque entrambe le reti hanno un ritardo di 3tp.

Realizzazione PO

Tre classi di registri:

- SP, con ingresso dalla ALU
- M, con ingressi dalle due interfacce verso Ua e Ub e indirizzi da SP o dalla ALU
- VALc, con ingresso dalla memoria M o da VALa/VALb

Serve una ALU per SP, assumiamo che abbia un ritardo di 5tp e un α per poter fare l'operazione +1 o -1.

Il calcolo delle variabili di condizionamento richiede al più un tp (or di 7 bit o lettura da registri e indicatori a transizione di livello).

La σPO ha come operazione più lunga la lettura da M[SP-1]: talu + tk + ta. Assumendo che la memoria si acceda in 10tp, e che la alu si stabilizzi in 5tp, questo comporta un tempo di 17tp.

Dunque avremo un τ di

$$tp + \max\{3tp, 3tp+17tp\} + tp = 22\ tp$$

Il tempo di servizio sarà pari a 1τ, visto che tutte le operazioni si concludono con l'esecuzione di una sola microistruzione.

L'unità si poteva anche realizzare derivandola da un microprogramma in cui si utilizza una variabile turno per rendere fair il protocollo in caso di richieste contemporanee di push.

Domanda 2

La traccia degli indirizzi in memoria principale interessa 5 pagine (fisiche): due per P, due per A e una per il codice. Le pagine hanno IPL da 0 a 5, rispettivamente e saranno tradotte mediante TABRIL in cinque IPF (rispettivamente IPF0, IPF1, IPF2, IPF3 eIPF4). Dunque la traccia degli indirizzi sarà:

1. IPF4 (fetch prima istruzione)
2. IPF4+1 (seconda ADD)
3. IPF4+2 (fetch LOAD)
4. IPF2 (load A[0])
5. IPF4+3 (fetch MUL)
6. IPF4+4 (fetch STORE)
7. IPF0 (store P[0])
8. IPF4+5 (fetch INC)
9. IPF4+6 (fetch IF<)
10. IPF4+7 (fetch END, scartata da IU)

11. IPF4+2 (fetch LOAD)
12. IPF2+1 (load A[1])
13. …

In assenza di ottimizzazioni, l'esecuzione di una iterazione impiega 13t:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IM	ADD	ADD	LOAD		MUL	STORE							INC	IF<		END	LOAD				
IU					LOAD	MUL	STORE						STORE	INC	IF<	IF<	END	LOAD			
DM		ADD	ADD			LOAD								STORE					LOAD		
Eum			ADD	ADD			LOAD	MUL							INC					LOAD	
EU*/									MUL	MUL	MUL	MUL									

Il codice si può ottimizzare anticipando la INC e post ponendo la STORE nel delay slot della IF< (avendo cura di utilizzare un registro baseP decrementato di 1):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IM	ADD	ADD	LOAD	MUL	INC	IF<			STORE	LOAD							
IU		ADD	ADD	LOAD	MUL	INC	IF<		IF<	STORE		STORE	LOAD				
DM					LOAD								STORE	LOAD			
Eum			ADD	ADD		LOAD	MUL	INC							LOAD		
EU*/								MUL	MUL	MUL	MUL						

Che riduce il tempo per la singola iterazione a 9t.

In entrambi i casi, vanno aggiunte 2 (o 3) t iniziali per le istruzioni che inizializzano i registri. Dunque il tempo di completamento per il codice ottimizzato sarà Tc = 2t + 4k (9t) + Tfault. Senza considerare l'utilizzo del prefetching, abbiamo 1 fault per il codice e $4K/\sigma$ fault per ciascuno dei due vettori per un totale di

$$Tc = 2t + 4K\ (9t) + (1+4K/\sigma)\ Ttrasf = (36K+1)t + (256+1)\ Ttransf$$

Il Ttransf andrà valutato facendo le opportune considerazioni sul costo del fault di cache.

Domanda 3

Vedi libro: va sommato il costo della fase firmware + quello della fase assembler (reperimento dell'indirizzo dell'handler e call dell'handler + costo dell'esecuzione dell'handler stesso).