Architettura degli Elaboratori – A.A. 2013—2014
Quarto Appello—14 gennaio 2015

Riportare su tutti i fogli consegnati Nome, Cognome, Numero di matricola e Corso di appartenenza.
Risultati e il calendario degli orali verranno pubblicati via web appena disponibili.

Domanda 1

Si consideri un'unità firmware U dotata di una memoria interna A da 64K posizioni che riceve da un'altra unità U_{in} richieste per l'esecuzione delle operazioni esterne:

- memorizzazione nella posizione J della memoria A del valore X;
- lettura del valore alla posizione J della memoria A;
- ricerca del valore X nella memoria e restituzione dell'indice della prima occorrenza di X in A o del valore (-1) in caso X non sia presente in A;
- sostituzione del valore della posizione J della memoria col massimo fra il vecchio valore ed X e restituzione del minimo fra i due valori.

I valori J (16 bit) e X (32 bit) sono passati da U_{in}. Il valore dell'indice di X in A (della posizione J della memoria) viene restituito mediante il registro JOUT (XOUT). L'interazione fra U_{in} e U è a domanda risposta.

Si fornisca il microprogramma della unità U nei due casi:

- utilizzando variabili di condizionamento complesse
- senza l'utilizzo di variabili di condizionamento complesse

Avendo cura di minimizzare il numero di micro istruzioni eseguite per implementare le operazioni esterne, si fornisca un raffronto del tempo medio di esecuzione delle operazioni esterne (da considerarsi equiprobabili) nei due casi e si discuta l'impatto dell'eventuale utilizzo di una memoria a doppia porta.

Domanda 2

Si compili il seguente pseudo codice in assembler D-RISC:

```
for(i=0;i<N;i++) {
   x[i] = a[i]*b[i]; y[i] = a[i]+1; b[i] = b[i]-1;
}
```

e se ne fornisca un'analisi delle cause di degrado delle prestazioni su un processore pipeline con unità di esecuzione slave per le moltiplicazioni intere da 4 stadi. Si fornisca quindi un codice che ottimizza l'efficienza.

Domanda 3

Stimare la differenza del tempo di esecuzione di un trasferimento di un blocco di dati da una unità di I/O alla memoria centrale quando

- la periferica supporti memory mapped I/O ma non DMA;
- la periferica supporti sia memory mapped I/O che DMA.

Domanda 1

La prima versione del codice fa uso di variabili di condizionamento complesse per la realizzazione della terza e della quarta operazione esterna:

0. *// attendi una richiesta di operazione*
   (RDY, OP0, OP1, segno(X – A[J]),zero(X-A[J])=0--_) nop, 0;
   // prima operazione esterna: memorizza il valore in ingresso
   (=100--) X→A[J], reset RDY set ACK, 0;
   // seconda operazione esterna, leggi il valore richiesto
   (=101--) A[J]→XOUT, reset RDY, set ACK, 0;
   // terza operazione: se non è il primo elemento, salta al ciclo di ricerca in [1,63]
   (=110-0) 1→C, 1;
   // se è il primo elemento restituisci l'indice 0 e termina operazione esterna
   (=110-1) 0 → OUTJ, reset RDY, set ACK, 0;
   // quarta operazione: si conclude immediatamente
   // se l'elemento in ingresso è il maggiore memorizzalo e restituisci il vecchio valore
   (=1110-) X→A[J], A[J]→XOUT, reset RDY, set ACK, 0;
   // altrimenti restituisci il valore in ingresso e lascia immutata la memoria
   (=1111-) X→XOUT, reset RDY, set ACK, 0
1. *// ciclo di confronto: se trovo l'elemento alla posizione corrente, ne restituisco l'indice*
   (C0,zero(X-A[C])=01) C →JOUT, reset RDY, set ACK, 0;
   // se non lo trovo, proseguo
   (=00) C+1 → C, 1;
   // se sono arrivato in fondo senza trovarlo, restituisco -1
   (=1-) (-1) → JOUT, reset RDY, set ACK, 0

Tutte le operazioni esterne, tranne la terza, vengono concluse con la sola micro istruzione 0. Per la terza operazione esterna è necessario eseguire un ciclo che scorre tutte le posizioni della memoria, se il valore cercato non c'è, o la parte di memoria fino alla sua prima occorrenza.

La realizzazione della PO richiede necessariamente l'utilizzo di una memoria a doppia porta (un indirizzo da J e un indirizzo da C). Si potrebbe implementare il programma utilizzando una memoria con un solo indirizzo (memoria standard), a patto di trasferire, nella prima micro istruzione J in C e utilizzare poi una seconda micro istruzione per completare le operazioni esterne uno, due e tre. Questo non avrebbe però soddisfatto il vincolo di utilizzare il minimo numero di micro istruzioni per l'implementazione delle operazioni esterne (2 invece di 1).

PC

Il micro programma ha due micro istruzioni (stato interno della PC da 1 bit) e 6 variabili di condizionamento (6 ingressi per la PC). Dunque $T_{\omega PC}$ e $T_{\sigma PC}$ hanno al più 7 ingressi (1 livello AND). Nel

micro programma ci sono 10 frasi, quindi ciascuna delle variabili di controllo (α e β) e la variabile che indica il prossimo stato interno, avranno al massimo 10 "uno" nella corrispondente colonna della tabella di verità $T_{\omega PC}$. Tuttavia, se assumiamo che colonne con più di 8 "uno" possono essere codificate come negazione della codifica degli "zero", possiamo concludere che anche il livello OR di $T_{\omega PC}$ e $T_{\sigma PC}$ sarà costituito da un singolo livello di porte. Dunque $T_{\omega PC}$ e $T_{\sigma PC}$ costeranno $2t_p$.

PO

Per $T_{\omega PO}$ dobbiamo considerare $t_a + t_{alu}$. Occorrono due ALU distinte per la realizzazione delle variabili di condizionamento della prima micro istruzione (che prendono in ingresso l'uscita di indirizzo J di A) e per la variabile di condizionamento della seconda micro istruzione (che prende in ingresso l'uscita di inidirizzo C di A).

Per $T_{\sigma PO}$ dobbiamo invece considerare i tempi spesi nelle diverse frasi del micro codice:

- Nella prima micro istruzione, la frase A[J]→XOUT prevede una lettura in memoria seguita da una scrittura in un registro con un commutatore all'ingresso ($t_a + t_k$).
- Nelle altre frasi, o si scrive l'uscita di una ALU in un registro passando da un commutatore ($t_{alu}+t_k$) o si scrive un valore in memoria (senza passare da commutatori, dunque in t_a).

Dunque possiamo concludere che il massimo $T_{\sigma PO}$ vale ($t_a + t_k$) e è relativo a frasi con variabili di condizionamento complesso, dunque:

$\tau = t_a + t_{alu} + \max\{ 2t_p, 2t_p + t_a + t_k \} + t_p = 2t_a + t_{alu} + t_k + 3t_p$

In caso non volessimo utilizzare variabili di condizionamento complesse, il codice dovrebbe essere leggeremente diverso: per la terza operazione esterna, in particolare, dovremmo produrre il valore di zero(X-A[J]) in un registro da testare poi in una micro istruzione successiva e per la quarta dovremmo fare lo stesso per il segno(X-A[J]). Il microcodice potrebbe essere il seguente:

0. (RDY, OP0, OP1 = 0--) nop, 0;
   (=100) X → A[J], reset RDY, set ACK 0;
   (=101)A[J] → XOUT, reset RDY, set ACK, 0;
   (=110) 0 → C, zero(X-A[0])→Z, 1;
   (=111) segno(X-A[J]) → S, 2
1. (C0,Z=00) C+1 → C, zero(X-A[C])→Z, 1;
   (=01) C →JOUT, reset RDY, set ACK, 0;
   (=1-) (-1) →JOUT, reset RDY, set ACK, 0;
2. (S=0) X→A[J], A[J]→XOUT, reset RDY, set ACK, 0;
   (=1) X→XOUT, reset RDY, set ACK, 0

Per la PC valgono considerazioni analoghe al caso precedente e possiamo senz'altro dire che il ritardo di $T_{\omega PC}$ e $T_{\sigma PC}$ sarà di $2t_p$.

Per la PO, $T_{\omega PO}$ è 0 e $T_{\sigma PO}$ sarà dato determinato dal massimo fra il tempo richiesto per la A[J] → XOUT ($t_k + t_a + t_k$, il primo commutatore è per gli indirizzi, il secondo per la scrittura nel registro) e per la segno(X-A[J]) ($t_k + t_a + t_{alu}$) dunque presumibilmente ($t_k + t_a + t_{alu}$). Il calcolo del ciclo di clock porterà dunque a

$\tau = 0 + \max \{2\ t_p, 2t_p + t_k + t_a + t_{alu}\} + t_p = t_a + t_{alu} + t_k + 3t_p$

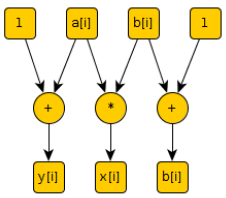
Che risulta leggermente inferiore a quello calcolato nel caso di utilizzo delle variabili di condizionamento complesse. Tuttavia, in questo caso il tempo medio di esecuzione conta 2τ per la quarta operazione esterna invece che 1τ. Tutte le altre operazioni esterne richiedono invece lo stesso numero di micro istruzioni. Quindi nel computo del tempo medio di esecuzione il caso di utilizzo di variabili di condizionamento complesse "guadagna" $1\tau/4$ (1 ciclo in un quarto dei casi corrispondenti alla quarta operazione esterna, pari a $2t_a + t_{alu} + t_k + 3t_p$) ma "perde" un ta per ognuno dei cicli richiesti per le altre operazioni esterne per un totale di $(¼ + ¼ + ¼(64K))t_a = t_a/2 + 16K\ t_a$. Questo ci porta a concludere che la seconda soluzione è senz'altro più efficiente.

Domanda 2

La compilazione del codice secondo le regole viste a lezione produce il codice (non ottimizzato):

```
 1          CLEAR Ri
 2  loop:   LOAD RbaseA, Ri, Rai
 3          LOAD RbaseB, Ri, Rbi
 4          MUL Rai, Rbi, Rtemp
 5          STORE RbaseX, Ri, Rtemp
 6          ADD Rai, #1, Rai
 7          STORE RbaseY, Ri, Rai
 8          SUB Rbi, #1, Rbi
 9          STORE RbaseB, Ri, Rbi
10          INC Ri
11          IF< Ri, RN, loop
12  end:    END
```

Vi sono dipendenze logiche fra la 4 e la 5, fra la 6 e la 7, fra la 8 e la 9 e infine fra la 10 e la 11, tutte di distanza 1. Tuttavia, non vi è alcuna dipendenza fra i risultati, come si evince dal grafo (data flow) delle dipendenze:



Di conseguenza le operazioni si possono eseguire in un ordine arbitrario, purchè i calcoli avvengano prima delle rispettive STORE.

L'esecuzione del codice (non ottimizzato sul processore pipeline può essere schematizzata come segue:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
IM	CLEA	LD		LD	MUL	ST	ADD							ST	SUB		ST	INC		IF	END		LD			
IU		CLEA	LD	LD	LD	MUL	ST						ST	AD	ST	ST	SUB	ST	ST	INC	IF	IF	~~END~~	LD		
DM					LD	LD								ST			ST			ST					LD	
Eum			CLEAR			LD	LD	MUL							ADD			SUB			INC					LD
EU*									MUL	MUL	MUL	MUL														

da cui si evince che un T = 20t/10 = 2t (al netto della prima bolla legata all'inizializzazione del registro indice) e quindi un'efficienza pari al 50%.

Il problema maggiore è legato alla MUL e nell'ottimizzare il codice vi si porrà particolare attenzione. Possiamo anticipare le operazioni che producono i risultati da memorizzare con le store prima di effetture le store. Inoltre, visto che la MUL determina un bolla lunga, possiamo utilizzare lo slot del salto ritardato per la fine del ciclo per eseguire questa STORE. Possiamo anche anticipare la Ri, in modo da eliminare l'effetto dalla dipendenza della INC sulla IF, avendo cura di utilizzare per le STORE (che a questo punto risultano successive alla INC Ri, registri base decrementati di 1. Tuttavia, l'anticipo della INC Ri determina dipendenze EU-IU fra la INC Ri anticipata e le STORE. Per minimizzarne l'effetto possiamo anticipare la INC rispetto alle altre operazioni aritmetiche. Questo però re-introduce una piccola bolla perché al momento di eseguire la prima STORE, il risultato da memorizzare non è ancora pronto sulla EU. Il codice risultante sarà quindi:

```
 1          CLEAR Ri
 2  loop:   LOAD RbaseA, Ri, Rai
 3          LOAD RbaseB, Ri, Rbi
 4          INC Ri
 5          MUL Rai, Rbi, Rtemp
 6          ADD Rai, #1, Rai
 7          SUB Rbi, #1, Rbi
 8          STORE RbaseY', Ri, Rai
 9          STORE RbaseB', Ri, Rbi
10          IF< Ri, RN, loop, delayed
11          STORE RbaseX', Ri, Rtemp
12  end:    END
```

per il quale abbiamo:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IM	LD	LD	INC	MUL	ADD	SUB	ST	ST		IF	ST	LD			
IU		LD	LD	INC	MUL	ADD	SUB	ST	ST	ST	IF	ST	LD		
DM			LD	LD						ST	ST		ST	LD	
Eum				LD	LD	INC	MUL	ADD	SUB						LD
EU*								MUL	MUL	MUL	MUL				

T = 11t / 10 = t ed efficienza pari al 91%.

E' da tener presente che stiamo eseguendo un programma che effettua tre operazioni di store in 10 cicli. L'architettura del sottosistema di memoria non è specificata, dunque dovremmo anche dire che, in presenza di politica write-through il sottosistema di memoria deve permettere l'esecuzione di una STORE ogni 10t/3 senza rallentamenti.

Domanda 3

L'esecuzione di una copia fra unità di I/O e memoria centrale quando sia o non sia presente il supporto DMA differisce sostanzialmente nel tempo speso, sul processore, per realizzare il trasferimento:

- se non vi è supporto DMA, il processore esegue un ciclo di LOAD+STORE il cui effetto è quello di leggere in un registro una locazione della memoria interna della periferica (utilizzando il MM I/O) e di portare tale valore in memoria. Il costo per ognuno dei trasferimenti è quindi pari al fetch + esecuzione di almeno due istruzioni (senza considerare il controllo delle iterazioni) in aggiunta al tempo di trasferimento dalla memoria interna della periferica alla memoria principale.
- se è presente il supporto per il DMA, il tempo speso a livello di processore è nullo, ma ovviamente si deve comunque attendere un trasferimento da memoria interna della unità di I/O a memoria principale per ognuna delle parole del blocco.

Va notato che, qualora il trasferimento avvenisse sotto il controllo del processore, la scrittura avviene in cache (ed eventualmente in memoria principale se la cache è gestita mediante politiche *write-through*), mentre utilizzando il DMA normalmente la scrittura avviene in memoria principale e una successiva lettura dei valori causa fault in cache.