Soluzioni degli esercizi del Capitolo 2

Questo documento contiene le soluzioni ad un numero selezionato di esercizi del Capitolo 2 del libro "Calcolatori Elettronici - Architettura e organizzazione", Mc-Graw Hill 2017.
Coloro che avessero sviluppato soluzioni alternative a quelle qui proposte, o soluzioni a esercizi non compresi tra quelli qui trattati, sono invitati a trasmetterle all'indirizzo sotto riportato. Serviranno a migliorare e tenere aggiornati i contenuti di questo sito.
L'autore sarà grato nei confronti di coloro che segnaleranno errori di qualunque genere, sia nella parte che segue sia nel libro menzionato.
giacomo.bucci@unifi.it

Aggiornato il giorno 19 aprile 2017

2.5 La logica richiesta è mostrata in Figura 2.1.

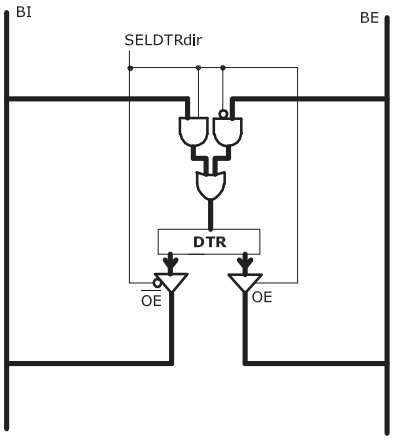

Figura 2.1 (Esercizio 2.5) Logica per far attraversare il registro DTR da flussi opposti. BI e BE rappresentano rispettivamente il bus interno e quello esterno.

2.8 I passi previsti dalla fase di esecuzione dell'operazione di `SWP MEM1,MEM2` sono riportati di seguito. Si fa l'ipotesi che lettura e scrittura di un dato in memoria richiedano due cicli di clock.

Per quanto si riferisce alla notazione usata, assumendo che i registri siano a 32 bit, `IR[x:y]` sta a indicare il campo dal bit x al bit y (y meno significativo) di IR; 0^x sta a indicare un campo di x bit posti tutti a zero; infine A||B indica l'accostamento (concatenamento) del campo B al campo A. Viene adottata la convenzione little endian (Paragrafo 4.5.4 del testo), ovvero il bit 0 è il meno significativo.

$\mathrm{T1} : 0^{19} \parallel \mathtt{IR}[12:0]_{\mathtt{out}}, \mathtt{MAR}_{\mathtt{in}}$;MAR<-indirizzo di MEM2
$\mathrm{T2} : \mathtt{M}_{\mathtt{read}}$
$\mathrm{T3} : \mathtt{M}_{\mathtt{read}}, \mathtt{DTR}_{\mathtt{in}}$;DTR<- MEM2

T4 : $\texttt{DTR}_{\texttt{out}}, \texttt{TEMP}_{\texttt{in}}$;TEMP<- MEM2
T5 : $0^{19} \parallel \texttt{IR}[25:13]_{\texttt{out}}, \texttt{MAR}_{\texttt{in}}$;MAR<-indirizzo di MEM1
T6 : $\texttt{M}_{\texttt{read}}$
T7 : $\texttt{M}_{\texttt{read}}, \texttt{DTR}_{\texttt{in}}$;DTR<- MEM1
T8 : $0^{19} \parallel \texttt{IR}[12:0]_{\texttt{out}}, \texttt{MAR}_{\texttt{in}}$;MAR<-indirizzo di MEM2
T9 : $\texttt{SELDTRdir}, \texttt{DTR}_{\texttt{out}}, \texttt{M}_{\texttt{write}}$;MEM2<- MEM1
T10: $\texttt{SELDTRdir}, \texttt{DTR}_{\texttt{out}}, \texttt{M}_{\texttt{write}}$
T11: $0^{19} \parallel \texttt{IR}[25:13]_{\texttt{out}}, \texttt{MAR}_{\texttt{in}}$;MAR<-indirizzo di MEM1
T12: $\texttt{SELDTRdir}, \texttt{TEMP}_{\texttt{out}}, \texttt{DTR}_{\texttt{in}}$;DTR<- MEM2
T13: $\texttt{SELDTRdir}, \texttt{DTR}_{\texttt{out}}, \texttt{M}_{\texttt{write}}$;MEM1<- MEM2
T14: $\texttt{SELDTRdir}, \texttt{DTR}_{\texttt{out}}, \texttt{M}_{\texttt{write}}$

L' esecuzione dell'operazione ha richiesto 14 cicli di clock, che sommati ai 4 cicli di fetch porta ad 18 i cicli necessari a eseguire l'istruzione `SWP MEM1,MEM2`.

2.9 All'inizio della fase di fetch, il μPC punta alla locazione della ROM da cui inizia la sequenza di CW la cui interpretazione dà luogo alla fase di fetch. La fase di fetch inizia con il prelievo dell'istruzione e si conclude quando questa viene scritta in IR per essere codificata. I passi non differiscono da quelli del paragrafo 2.6.1 del testo:

T1: PC_{out}, MAR_{in}, SEL4, ADD, TO_{in}
T2: M_{read}, DTR_{in}, TO_{out}, PC_{in}
T3: DTR_{out}, IR_{in}

L'effetto di `ADD R1,R2,R3`, è di trasferire in R1 il risultato della somma del contenuto dei registri R2 e R3. Poichè si assume una macchina con singolo bus interno, come in Figura 2.11 del testo, seguono questi passi

T1: PC_{out}, MAR_{in}, SEL4, ADD, TO_{in}
T2: M_{read}, DTR_{in}, TO_{out}, PC_{in}
T3: DTR_{out}, IR_{in}
T4: R2_{out}, TI_{in}
T5: R3_{out}, ADD, TO_{in}
T6: TO_{out}, R1_{in}

Ragionando secondo lo schema della codifica orizzontale, la parola di controllo deve contenere tanti bit quanti sono i comandi individuati.
La sequenza sopra riportata prevede 16 comandi, da cui conseguirebbe – per la specifica istruzione `ADD R1,R2,R3` – una parola di controllo di 16 bit del tipo di quella mostrata in Tabella 2.1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PC out	PC in	MAR in	SEL4	ADD ALU	T0 in	T0 out	M read	DTR in	DTR out	R1 in	R2 out	R3 out	IR in	TI in	TI out

Tabella 2.1 Parola di controllo specifica della soluzione dell' Esercizio 2.9. Si noti che relativamente ai registri, si è assunto che esista uno specifico bit per comandare, per ognuno di essi, sia l'ingresso che l'uscita.

BIT:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T1:	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
T2:	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0
T3:	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
T4:	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
T5:	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	0
T6:	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0

Tabella 2.2 (Esercizio 2.9) Sequenza di controllo riferita alle parole aventi formato come in Tabella 2.1.

Risulterebbe la sequenza di CW di Tabella 2.2.
Come indicato della didascalia della Tabella 2.1, la parola di controllo ivi mostrata contiene i soli bit che si richiedono nella specifica istruzione, e i bit relativi agli specifici registri. Seguendo questo schema, con una macchina a 32 registri occorrerebbero ben 96 bit per la selezione dei registri stessi. Conviene invece decodificare a parte i campi Rd, Rs1 e Rs2 dell'istruzione prevedendo che i bit 10, 11 e 12 della parola di controllo abbiano il significato Rd_{in}, $\mathrm{Rs1}_{out}$ e $\mathrm{RS2}_{out}$ e prevedere che i segnali di comando dei singoli registri siano ottenuti come nello schema di Figura 2.2, tracciato in riferimento al registro di destinazione. In questo modo la parola di controllo mantiene la stessa dimensione di Tabella 2.2, a scapito della presenza del demultiplexer.

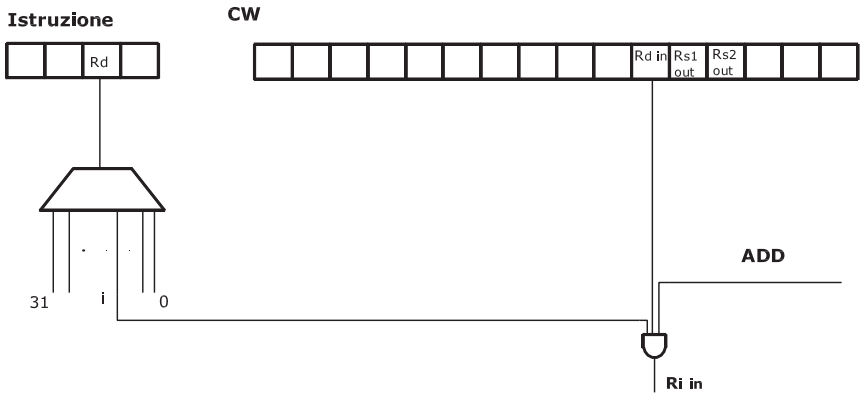

Figura 2.2 (Esercizio 2.9) Miglioramento rispetto alla codifica di Tabella 2.1. I bit 10,11 e 13 della parola di controllo assumono il significato riportato in figura.

2.12 Si riporta la Tabella 2.3 in cui, rispetto a quella del testo, sono stati aggiunti i valori del numero totale di cicli per ciascuna istruzione (*Ncicli*), la percentuale x_i nel mix ($x_i = X_i / \sum X_i$), e il contributo dato dall'istruzione a C_{PI} (ovvero $x_i \times c_{pi}$).

C_{PI} viene calcolato come $\sum (x_i \times c_{pi})$, oppure come rapporto tra il totale del numero di cicli e il totale delle istruzioni ($\frac{\sum Ncicli}{\sum X_i} = 600/78$). Si ottiene $C_{PI} = 7,69$.

Tipo di istruzione	X_i	c_{pi}	*Ncicli*	x_i (%)	$x_i \times c_{pi}$
Addizioni	20 mil.	4	80 mil.	25,64	1,03
Moltiplicazioni	6 mil.	10	60 mil.	7,69	0,77
Divisioni	2 mil.	40	80 mil.	2,56	1,02
Memoria	40 mil.	8	320 mil.	51,28	4,10
Salti	10 mil.	6	60 mil.	12,82	0,77
Totale	78 mil.		600 mil.	100,00	7,69

Tabella 2.3 (Esercizio 2.12) Calcolo del C_{PI}.

2.13 Si ricorda che l'accelerazione è definita con la seguente formula:

$$a = \frac{t_v}{t_n}$$

dove t_v è il vecchio tempo di esecuzione e t_n è il nuovo tempo di esecuzione.

$\sum X_i$ è il numero totale di istruzioni eseguite, dunque $t_v = C_{PIv} \times \sum X_i = 7,69 \times 78 \times 10^6$. I tempi nuovi richiedono il calcolo dei nuovi C_{PI}.

Una accelerazione pari a 4 nelle addizioni comporta questo cambiamento nella prima riga di Tabella 2.3.

Tipo di istruzione	X_i	c_{pi}	*Ncicli*	x_i (%)	$x_i \times c_{pi}$
Addizioni	20 mil.	1	20 mil.	25,64	0,25

Conseguentemente l'ultima riga di Tabella 2.3 diventa:

Tipo di istruzione	X_i	c_{pi}	*Ncicli*	x_i (%)	$x_i \times c_{pi}$
...	...	...	...	...	...
Totale	78 mil.		540 mil.	100,00	6,92

Dunque il tempo nuovo è pari a $t_v = 6,92 \times 78 \times 10^6$; si ha quindi una accelerazione globale pari a: $\frac{7,69 \times 78 \times 10^6}{6,92 \times 78 \times 10^6} = 7,69/6,92 = 1,11$.

Una accelerazione pari a 4 per la memoria comporta questo cambiamento nella quarta riga di Tabella 2.3.

Tipo di istruzione	X_i	c_{pi}	*Ncicli*	x_i (%)	$x_i \times c_{pi}$
Memoria	40 mil.	2	80 mil.	51,28	1,02

Conseguentemente l'ultima riga di Tabella 2.3 diventa:

Tipo di istruzione	X_i	c_{pi}	*Ncicli*	x_i (%)	$x_i \times c_{pi}$
...	...	...	...	...	...
Totale	78 mil.		360 mil.	100,00	4,61

Dunque si ha una accelerazione globale pari a: $7,69/4,61 = 1,66$. Ne consegue la convenienza di apportare il miglioramento alla memoria.

2.14 Valgono le considerazioni seguenti.

- Imponendo una accelerazione pari a 4 sia alle operazioni di memoria che alle addizioni si ottiene la situazione riassunta in Tabella 2.4. Ricordando dalla Tabella 2.3 che il C_{PI} del sistema non modificato vale 7,69 si può scrivere l'accelerazione globale pari a $\frac{7,69}{3,83} = 2,00$.
- L'accelerazione globale relativa deve essere calcolata considerando al numeratore il C_{PI} proprio del sistema già accelerato nelle operazioni di memoria. Recuperando questo dato dalle Tabelle dell'Esercizio 2.12, si ha che l'accelerazione globale relativa è pari a: $\frac{4,61}{3,83} = 1,20$.
- Come per il caso precedente l'accelerazione globale relativa deve essere calcolata considerando che al numeratore si ha il C_{PI} proprio del sistema già accelerato nelle addizioni. Recuperando questo dato dalle Tabelle dell'Esercizio 2.12 si ottiene un'accelerazione globale relativa pari a: $\frac{6,92}{3,83} = 1,80$.

Tipo di istruzione	X_i	c_{pi}	*Ncicli*	x_i (%)	$x_i \times c_{pi}$
Addizioni	20 mil.	1	20 mil.	25,64	0,25
Moltiplicazioni	6 mil.	10	60 mil.	7,69	0,77
Divisioni	2 mil.	40	80 mil.	2,56	1,02
Memoria	40 mil.	2	80 mil.	51,28	1,02
Salti	10 mil.	6	60 mil.	12,82	0,77
Totale	78 mil.		300 mil.	100,00	3,83

Tabella 2.4 (Esercizio 2.14) Calcolo del nuovo C_{PI}.

2.15 In soli termini di incremento di frequenza, le prestazioni del secondo processore risultano $3400/5 = 680$ volte superiori.
L'8086 aveva un C_{PI} (numero medio di clock a istruzione) pari a 15, mentre per l'i7 è vicino a 1. Assumendo cautelativamente che il numero medio di clock per istruzione dell'i7 Ivy Bridge sia pari a 1.4, si ottiene un ulteriore incremento di $15/1.4 = 10,71$ volte, che porta 7.286 il rapporto tra le prestazioni dei due.
Se si applica il fattore migliorativo legato all'ampiezza del bus esterno, pari 4 (64 contro 16 bit), il rapporto tra le prestazioni sale a 29.143.
Infine, assumendo che il guadagno medio che deriva dai 4 core dell'i7 sia pari a 3 (l'aumento di prestazioni non è direttamente proporzionale al numero di core, a causa di contese per le risorse a comune e a causa della maggior complessità del software) si ottiene un guadagno complessivo pari a $29.143 \times 3 = 87.429$.

2.16 Per quanto si riferisce al punto (1), nella Tabella 2.5 a sinistra e a destra vengono riportati i clock richiesti dalla fase di fetch e dalla fase di esecuzione per i diversi tipi di istruzione. Essi derivano dalle specifiche del testo. Per assunzione, l'8086 legge sempre parole allineate di 16 bit, mentre l'8088 legge parole di 8 bit, quindi esegue un doppio ciclo di lettura in memoria per i dati a 16 bit.

Fetch	*Execute*		
	60%	40%	
		30%	70%
8	3	3+4	3+4

Fetch	*Execute*		
	60%	40%	
		30%	70%
12	3	3+4	3+4+4

Tabella 2.5 (Esercizio 2.16) Tabelle riassuntive del numero di clock relativi al caso (1) dell'Esercizio 2.16. A sinistra per l'8086 a destra per l'8088.

Nella fase di fetch, poiché le istruzioni sono mediamente di 3 byte, l'8086 deve fare mediamente 2 letture (8 cicli di clock), mentre l'8088 farà esattamente 3 letture (12 cicli di clock).

I C_{PI} valgono:

$$C_{PI}8086 = \sum x_i c_{pi} = 8 + 0,6 \times 3 + 0,4 \times 0,3 \times 7 + 0,4 \times 0,7 \times 7 = 12,6$$

$$C_{PI}8088 = 12 + 0,6 \times 3 + 0,4 \times 0,3 \times 7 + 0,4 \times 0,7 \times 11 = 17,72$$

Si nota che il rapporto tra le prestazioni vale $\frac{P_{8086}}{P_{8088}} = \frac{C_{PI_{88}}}{C_{PI_{86}}} = \frac{17,72}{12,6} = 1,41$. Cioè l'8086 è il 41% circa più efficiente dell'8088.

Per quanto si riferisce al punto (2) la Tabella 2.6 riporta i clock richiesti per l'8086. Si osservi che, rispetto alla precedente tabella, i salti non comportano aumento di cicli in fase di fetch, in quanto nella Tabella 2.5 si sono considerate istruzioni di 3 byte a quindi sempre non completamente allineate, ne consegue che l'effetto dei salti è assorbito dal (mediamente sempre presente) disallineamento delle istruzioni. Quindi per l'8086, la differenza è l'aggiunta dei 4 clock in più sul 30% del 70% delle istruzioni che leggono/scivono un dato a 16 bit.

Per l'8088 le cose risultano invariate.

Fetch	*Execute*			
	60%	40%		
		30%	70%	
			70%	30%
8	3	3+4	3+4	3+4+4

Tabella 2.6 (Esercizio 2.16) Tabella riassuntiva dei clock relativi al punto (2) per l'8086.

L'8086 ha dunque questo indice di prestazione:

$$C_{PI}8086 = 8 + 0,6 \times 3 + 0,4 \times 0,3 \times 7 + 0,4 \times 0,7 \times 0,7 \times 7 + 0,4 \times 0,7 \times 0,3 \times 11 = 12,94$$

Il rapporto tra le prestazioni vale $\frac{P_{8086}}{P_{8088}} = \frac{C_{PI_{88}}}{C_{PI_{86}}} = \frac{17,72}{12,94} = 1,37$ (l'8088 ha mantenuto invariate le sue prestazioni). Poiché l'effetto dei salti non ha influenza sulla fase di fetch, il peggioramento delle prestazioni dell'8086 è dovuto al solo 30% del 70% del 40% delle istruzioni. Ciò causa una leggera diminuzione del vantaggio dell'8086 sull'8088, in quanto dal 41% di efficenza in più si è scesi al 37%.