Capitolo 13

Il calcolatore

Il calcolatore è un sistema complesso con caratteristiche nuove rispetto agli schemi fin qui studiati. La sua prerogativa principale consiste nella possibilità di *memorizzare* un insieme di istruzioni, che vengono poi eseguite a una a una, in un ordine stabilito dal sistema stesso: tale insieme di istruzioni prende il nome di *programma*.

Il programma è la formalizzazione di un algoritmo, espresso in forma intellegibile dal particolare calcolatore impiegato. Tale algoritmo può essere estremamente complesso, e richiedere un considerevole lavoro di definizione e stesura: una volta redatto in forma di programma e inserito in memoria, esso è eseguito automaticamente dal calcolatore senza richiedere ulteriori interventi esterni, salvo l'eventuale specificazione dei dati su cui deve operare.

Benché non sia nostro compito spiegare come si definiscano algoritmi e si trasformino poi in programmi, è utile presentarne alcuni semplicissimi esempi per comprendere la grande importanza che riveste il "sistema calcolatore".

13.1 Istruzioni e programma

La sequenza di istruzioni che costituisce un programma è allocata in celle consecutive di memoria, a partire da una cella iniziale di indirizzo noto. Il calcolatore estrae dalla memoria le istruzioni, e le interpreta ed esegue una alla volta: rimandiamo al prossimo paragrafo lo studio di come ciò possa avvenire, esaminando qui il *formato* delle istruzioni (cioè il loro contenuto informativo e il modo in cui esso è

memorizzato) e i programmi che possono essere conseguentemente costruiti.

Un formato tipico, utilizzato nei calcolatori di dimensioni medie, è quello delle istruzioni *a un indirizzo*. Esso prevede che ogni istruzione sia allocata in una cella di memoria, e che sia divisa in due parti: la prima, detta *codice operativo*, specifica l'operazione da eseguire; la seconda, detta *indirizzo*, specifica l'indirizzo di memoria di un dato, o altra informazione pertinente all'istruzione. Ci riferiremo nel seguito a istruzioni di questo tipo.

Come semplicissimo esempio consideriamo un calcolatore con parole di 15 bit, e una memoria di 4096 celle indirizzabili con 12 bit. Le istruzioni (di 15 bit) dovranno riservare 12 bit alla parte indirizzo, e i rimanenti 3 bit alla parte codice operativo: esse potranno specificare in tal modo solo $2^3 = 8$ codici operativi distinti. Indichiamo nella tabella 13.1 un possibile *repertorio* di tali istruzioni, rappresentate con codici alfabetici anziché stringhe binarie, per facilitare la successiva lettura dei programmi. Esse fanno riferimento a un solo registro A, detto in genere *accumulatore*, e prevedono una parte operativa del tipo indicato nelle figure 12.8 e 12.9.

Consideriamo anzitutto un programma per eseguire l'addizione tra i

Tabella 13.1 Repertorio di otto istruzioni a un indirizzo

cod-op	ind	significato
LDA	X	*Load A*: carica il registro A con il contenuto della cella di memoria di indirizzo X.
STA	X	*Store A*: memorizza il contenuto del registro A nella cella di memoria di indirizzo X.
ADA	X	*Add in A*: addiziona i contenuti del registro A e della cella di memoria di indirizzo X e poni il risultato in A.
SBA	X	*Subtract in A*: sottrai il contenuto della cella di indirizzo X dal contenuto del registro A e poni il risultato in A.
JZA	X	*Jump on Zero A*: se il contenuto del registro A è 0, salta all'istruzione contenuta nella cella di indirizzo X.
RDA	X	*Read in A*: trasferisci nel registro A il dato introdotto dall'esterno nell'unità di ingresso X.
WTA	X	*Write A*: scrivi il contenuto del registro A mediante l'unità di uscita X.
HLT	–	*Halt*: arresta le operazioni.

contenuti delle celle di memoria di indirizzi 200 e 201, e porre il risultato nella cella di indirizzo 202. L'ovvia formulazione è la seguente:

```
LDA  200
ADA  201
STA  202
HTL  –
```

Questo programma occuperà quattro celle di memoria consecutive, con l'unico vincolo che esse siano diverse dalle 200, 201 e 202 destinate ai dati e al risultato.

Vediamo ora un programma la cui formulazione dipende dalla zona di memoria ove è allocato. Esso legge un dato X attraverso l'unità di ingresso 1 e lo confronta con il dato Y contenuto nella cella 300: se $X = Y$ si arresta, altrimenti stampa Y attraverso l'unità di uscita 2. Il programma (e la sua allocazione in memoria) è il seguente:

indirizzo	cella		
193	RDA	1	programma
194	SBA	300	
195	JZA	198	
196	LDA	300	
197	WTA	2	
198	HLT	–	
.			
.			
.			
300	*Y*		dati

Il confronto tra X e Y è realizzato sottraendo Y da X nel registro A (istruzione SBA 300) e saltando alla fine del programma se il risultato così ottenuto è zero (istruzione JZA 198). La necessità di specificare l'indirizzo 198 di destinazione nell'istruzione di salto, rende la formulazione del programma dipendente dalla sua allocazione in memoria.

Vediamo ora un programma che contiene un ciclo, cioè un insieme di istruzioni che devono essere eseguite più volte. Tale programma è sostanzialmente più complicato dei precedenti, poiché durante la sua esecuzione una delle istruzioni viene modificata dal programma stesso a ogni iterazione, finché si raggiunge un'opportuna condizione di terminazione. Supponiamo che la memoria contenga in celle consecutive, a partire dalla cella 200 e non oltre la 300, una successione di uno o più dati D tutti uguali, che termina con un dato T diverso da D; il pro-

gramma deve contare il numero di occorrenze consecutive di D e stamparlo attraverso l'unità di uscita 2. Per realizzare queste operazioni si utilizzeranno due celle di servizio: la cella 301, il cui contenuto, inizialmente posto uguale a 1, è via via incrementato per rappresentare il numero N di dati D incontrati; e la cella 302, che contiene costantemente il valore 1, per realizzare l'incremento di N e la trasformazione dell'istruzione che dà luogo al ciclo. Il programma occupa le celle da 303 a 314, ma la prima istruzione da eseguire è quella contenuta nella cella 309, e questo indirizzo è noto al calcolatore all'inizio delle operazioni (vedi prossimo paragrafo).

	indirizzo	cella		
	200	D		
	201	D		
	202	D		
	.			
	.	T		
	.			
	301	N		inizializzata a 1
	302	1		inizializzata a 1
	303	LDA	301	incremento di N
	304	ADA	302	
	305	STA	301	
	306	LDA	310	incremento contenuto cella 310 (trasformazione istruzione)
	307	ADA	302	
	308	STA	310	
→	309	LDA	200	confronto di D con il dato corrente
	310	SBA	201	
	311	JZA	303	
	312	LDA	301	stampa di N
	313	WTA	2	
	314	HLT	–	

La particolarità di questo programma è la sua automodifica nel corso dell'esecuzione. Le istruzioni nelle celle 306, 307 e 308 trattano il contenuto della cella 310 come se fosse un numero anziché un'istruzione (SBA 201), e vi sommano 1 trasformandolo via via nell'istruzione SBA 202, SBA 203 ecc.: ovviamente tale operazione è legittima, poiché la cella contiene una stringa di bit, interpretabile sia come istruzione sia come numero binario. La necessità di ricorrere a operazioni di questo tipo è legata al ristretto repertorio di istruzioni disponibili e

all'estrema semplicità del calcolatore considerato. Sistema più complessi consentiranno di risolvere gli stessi problemi con programmi più semplici.

Questa brevissima introduzione alla programmazione di un calcolatore elementare fa nascere moltissimi interrogativi, i più seri dei quali troveranno risposta nei prossimi paragrafi. Si noti in particolare che non abbiamo ancora studiato come si carichi la memoria con il suo contenuto iniziale di programmi e dati.

Esercizi

13.1 Posto che nella memoria vi sia una serie di celle contenenti 0, a partire dall'indirizzo 100, scrivere un programma che legge dati dall'unità di ingresso 1 e li carica in tali celle, arrestandosi quando ne incontra una con contenuto diverso da 0.

13.2 Inserita nel repertorio la nuova istruzione JNA X con significato: "se il contenuto del registro A è diverso da 0, salta all'istruzione nella cella X", scrivere il programma di un ciclo che ripeta cento volte un'operazione a piacere.

13.2 Un calcolatore elementare

Il repertorio di istruzioni della tabella 13.1 può essere associato a un calcolatore elementare, che analizzeremo allo scopo di stabilire come venga trattato un programma, cioè come le sue istruzioni siano *estratte* dalla memoria ed *eseguite* in sequenza.

Ci riferiremo al semplice calcolatore di figura 13.1, che costituisce un'immediata estensione del sistema delle figure 12.8 e 12.9, cui sono stati aggiunti due nuovi registri PC e IND di 12 bit con le seguenti funzioni. Il registro contatore PC (per *Program Counter*) contiene inizialmente l'indirizzo ove è memorizzata la prima istruzione del programma, e, durante l'esecuzione di questo, contiene in ogni istante l'indirizzo della prossima istruzione da eseguire. Il registro composto da IR e IND (indicato con IR|IND) contiene in ogni istante l'istruzione in esecuzione; più precisamente IR contiene la parte codice operativo (3 bit), e IND contiene la parte indirizzo (12 bit). A è il registro accumulatore cui fanno riferimento le istruzioni; C è un'altro registro operativo. I segnali di condizione $\beta_0 = \mathrm{OR}(A)$ e $\beta_1 = \mathrm{OR}(\text{pronto})$ sono utilizzati indipendentemente nelle istruzioni JZA di salto e RDA di ingresso,

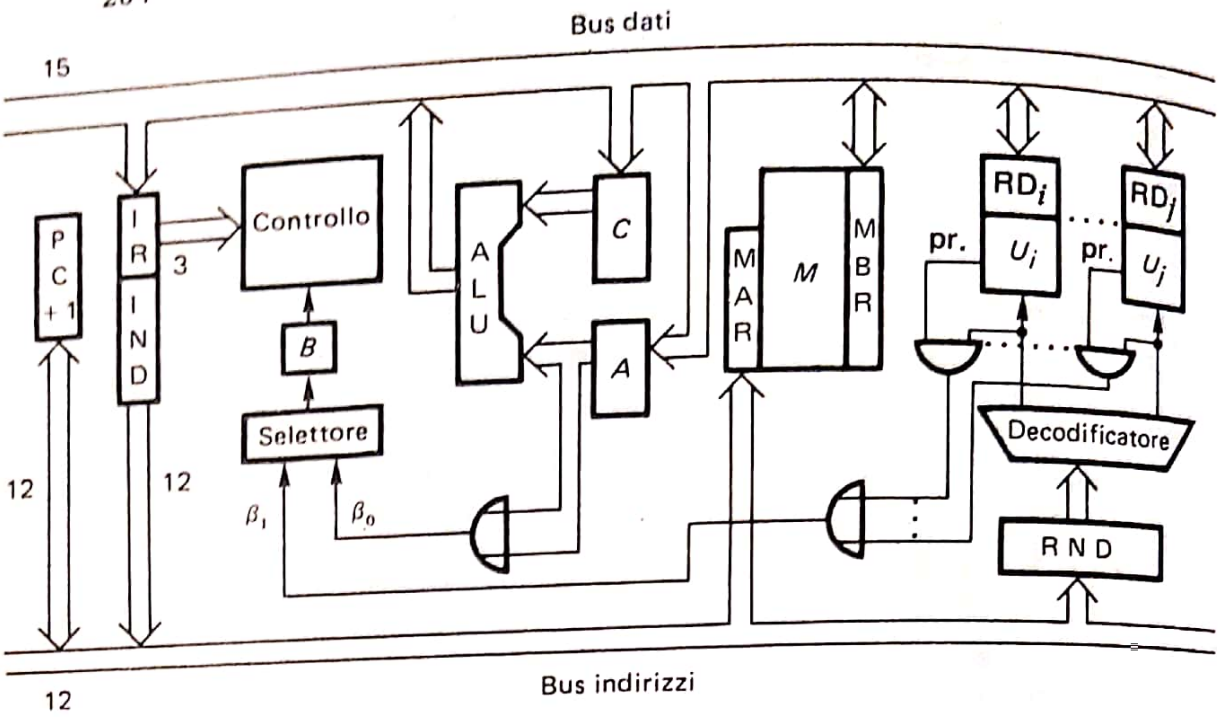

Figura 13.1
Un calcolatore elementare.

e in corrispondenza a queste sono caricati nel flip-flop *B* connesso alla parte controllo.

Il funzionamento della parte operativa è descritto da una μsequenza (uguale per tutte le istruzioni) per l'estrazione dalla memoria dell'istruzione da eseguire e il suo trasferimento nel registro IR|IND; e dalle successive μsequenze per l'esecuzione delle singole istruzioni.

La μsequenza di *estrazione* è la seguente:

μsequenza	μprogramma
PC→MAR;	μ_1
M[MAR]→MBR;	μ_2
MBR→IR|IND, INCR(PC)→PC	μ_3

Si noti che la μsequenza, contemporaneamente al trasferimento dell'istruzione nel registro IR|IND, comanda l'incremento di PC, che indica così la locazione della prossima istruzione da estrarre. I segnali α di controllo che compongono le μistruzioni μ_1, μ_2, μ_3 si deducono immediatamente dallo schema di figura 13.1, e non sono qui riportati.

Le μsequenze di esecuzione per le istruzioni LDA, STA, ADA, SBA, JZA e HLT sono elencate qui di seguito, e non richiedono particolari spiegazioni; si noti comunque che nell'istruzione JZA l'indirizzo di destinazione del salto viene trasferito nel registro PC, che conterrà così

la locazione dell'istruzione da cui il programma deve proseguire (diversa da quella sequenzialmente calcolata nella precedente fase di estrazione).

istruzione	μsequenza	μprogramma
LDA	IND → MAR;	μ_4
	M[MAR] → MBR;	μ_2
	MBR → A	μ_5
STA	A → (via: ALU-bus dati) → MBR, IND → MAR;	μ_6
	MBR → M[MAR]	μ_7
ADA	IND → MAR;	μ_4
	M[MAR] → MBR;	μ_2
	MBR → C;	μ_8
	$A + C \rightarrow A$	μ_9
SBA	IND → MAR;	μ_4
	M[MAR] → MBR;	μ_2
	MBR → C;	μ_8
	$A - C \rightarrow A$	μ_{10}
JZA	$\beta_0 \rightarrow B$;	μ_{11}
	if B = '0' **then** IND → PC	μ_{12}
	else ϕ **fi**	μ_{13}
HLT	ϕ	μ_{13}

Indichiamo infine qui di seguito le μsequenze di esecuzione delle due istruzioni RDA e WTA. In entrambi i casi il numero distintivo dell'unità di ingresso o uscita viene trasferito nel registro RND, per la selezione dell'unità attraverso il decodificatore. Nell'istruzione RDA la lettura può aver luogo solo quando l'unità selezionata ha acquisito il dato dall'esterno, ed emette in conseguenza un segnale di pronto uguale a uno, che è inviato all'unità di controllo attraverso il segnale β_1 e il flip-flop B; finché ciò non si verifica ($B=0$), l'istruzione mantiene il calcolatore in un ciclo di attesa (attraverso la parte **else** della μoperazione 2). Si noti la rete utilizzata nella figura 13.1, ove i segnali di interruzione provenienti da tutte le unità abilitate all'ingresso di dati sono collegati a β_1, ma solo quello proveniente dall'unità selezionata può attraversare lo sbarramento di blocchi AND.

istruzione	μsequenza	μprogramma
RDA	IND → RND;	μ_{14}
	$\beta_1 \rightarrow B$;	μ_{15}
	2: **if** B = '1' **then** $RD_{RND} \rightarrow A$	μ_{16}
	else $\beta_1 \rightarrow B$, **goto** 2 **fi**	μ_{15}
WTA	IND → RND;	μ_{14}
	A → (via: ALU-bus dati) → RD_{RND};	μ_{17}
	stampa con U_{RND}	μ_{18}

Durante lo svolgimento di un programma, l'esecuzione di ogni istruzione deve essere seguita dall'estrazione dell'istruzione successiva. Perché ciò possa avvenire, le μsequenze di esecuzione sopra riportate devono essere completate da un comando di salto all'inizio della μsequenza di estrazione; questo meccanismo sarà ora completamente definito attraverso il progetto della parte controllo, costruita secondo i principi esposti nel capitolo 12.

Applicando lo schema generale di figura 12.2, accederemo alla ROM del controllo attraverso i tre registri di istruzione IR, di condizione B e di stato Y, indicati nella figura 13.2. Come già sappiamo, IR contiene tre bit I_2, I_1, I_0 per la rappresentazione degli otto codici operativi, e B contiene il bit di condizione. Inoltre, poiché le μsequenze sono costituite al massimo da quattro μoperazioni, il registro Y conterrà due bit y_1, y_0. I registri IR, B e Y sono dotati dei comandi di azzeramento Z_{IR}, Z_B, Z_Y per portare il sistema in uno stato di quiete: a questo scopo nella cella della ROM di indirizzo $I_2 I_1 I_0 B y_1 y_0 = 000000$ sarà memorizzata la μistruzione per HLT.

Per comandare l'inizio dell'esecuzione di un programma (già contenuto in memoria) il registro PC è connesso all'esterno tramite un selettore, per poter essere inizializzato con l'indirizzo della prima istruzione del programma, e il flip-flop y_0 è dotato di un comando di *start* per la memorizzazione di un 1. La sequenza dei comandi di avvio introdotti dall'esterno è la seguente:

$Z_{IR} Z_B Z_Y = 111$ (azzera IR, B, Y);

sel = 1, $A_{PC} = 1$ (indirizzo prima istruzione → PC);

start = 1 ($1 \rightarrow y_0$).

Tale sequenza fa partire l'unità di controllo dall'indirizzo della ROM: $I_2 I_1 I_0 B y_1 y_0 = 000001$, ove inizia la μsequenza di estrazione delle istruzioni.

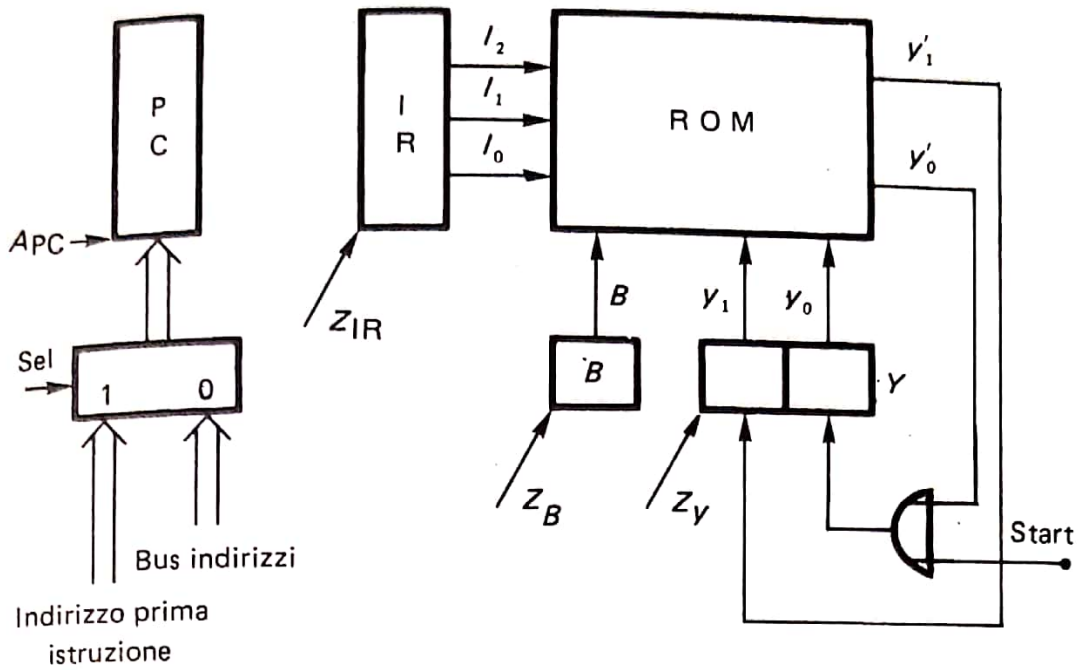

Figura 13.2
Il controllo per lo schema di figura 13.1.

Siamo ora in grado di definire compiutamente il contenuto della ROM del controllo: un attento esame di tale contenuto consentirà al lettore di comprendere fino in fondo il funzionamento del nostro calcolatore. Rappresentati i codici operativi delle istruzioni con le configurazioni binarie:

HLT=000, LDA=001, STA=010, ADA=011,
SBA=100, JZA=101, RDA=110, WTA=111,

i μprogrammi sono memorizzati a partire dall'indirizzo della ROM per cui $I_2I_1I_0$ =(codice operativo), $By_1y_0=000$, e sono specificati come è mostrato nella figura 13.3.

Ovvio è il significato del μprogramma di HLT, che corrisponde alla μsequenza vuota ϕ (μ_{13}); il registro Y conserva il suo contenuto ($y'_1y'_0=00$), mantenendo l'unità di controllo ferma all'indirizzo 000000. L'invio di un segnale di start fa ripartire il controllo dall'indirizzo 000001 del μprogramma di estrazione.

Il μprogramma di estrazione percorre i tre passi della corrispondente μsequenza, caricando il codice operativo dell'istruzione da eseguire in $I_2I_1I_0$; esso termina ponendo $y'_1y'_0=00$, e trasferisce in tal modo il controllo all'inizio del μprogramma della nuova istruzione. Poniamo per esempio che questa sia la LDA, ed esaminiamone a sua volta l'evoluzione.

Istruzione	Indirizzo $I_5 I_4 I_3 S_1 S_0$	Contenuto $S_1 S_0$	δ	
HLT	0 0 0 0 0 0	0 0	μ_{13}	
Estrazione	0 0 0 0 0 1	1 0	μ_1	← Start
	0 0 0 0 1 0	1 1	μ_2	
	0 0 0 0 1 1	0 0	μ_3	
LDA	0 0 1 0 0 0	0 1	μ_4	
	0 0 1 0 0 1	1 0	μ_5	
	0 0 1 0 1 0	0 1	μ_6, 0→IR	
STA	0 1 0 0 0 0	0 1	μ_7	
	0 1 0 0 0 1	0 1	μ_8, 0→IR	
ADA	0 1 1 0 0 0	0 1	μ_4	
	0 1 1 0 0 1	1 0	μ_2	
	0 1 1 0 1 0	1 1	μ_9	
	0 1 1 0 1 1	0 1	μ_{10}, 0→IR	
SBA	1 0 0 0 0 0	0 1	μ_4	
	1 0 0 0 0 1	1 0	μ_2	
	1 0 0 0 1 0	1 1	μ_9	
	1 0 0 0 1 1	0 1	μ_{10}, 0→IR	
JZA	1 0 1 0 0 0	0 1	μ_{11}	
	1 0 1 0 0 1	0 1	μ_{12}, 0→IR	
	.			
	.			
	.			
	1 0 1 1 0 1	0 1	μ_{13}, 0→IR, 0→B	
RDA	1 1 0 0 0 0	0 1	μ_{14}	
	1 1 0 0 0 1	1 0	μ_{15}	
	1 1 0 0 1 0	1 0	μ_{15}	
	.			
	.			
	.			
	1 1 0 1 1 0	0 1	μ_{16}, 0→IR, 0→B	
WTA	1 1 1 0 0 0	0 1	μ_{14}	
	1 1 1 0 0 1	1 0	μ_{17}	
	1 1 1 0 1 0	0 1	μ_{18}, 0→IR	

Figura 13.3
Contenuto della ROM.

Il μprogramma di LDA parte dalla cella di indirizzo 001000, e ripropone i tre passi della corrispondente μsequenza, cui è aggiunto l'azzeramento di $I_2 I_1 I_0 (0 \rightarrow IR)$ nell'ultima μistruzione. Tale azzeramento consente, assieme alla specificazione $y'_1 y'_0 = 01$ del prossimo stato, di trasferire il controllo all'indirizzo 000001, da cui riparte il μprogramma di estrazione. Si noti che nei tre μprogrammi fin qui esaminati il flip-flop B è stato mantenuto a 0.

I μprogrammi per le istruzioni STA, ADA e SBA derivano anch'essi immediatamente dalle relative μsequenze, completate dall'azzeramento di IR. Esaminiamo ora il μprogramma per l'istruzione JZA: esso carica al primo passo il flip-flop B e, in relazione al contenuto di questo, trasferisce il controllo all'indirizzo 101001 oppure 101101; in entrambi i casi il μprogramma termina con l'azzeramento di IR e, nel caso che B contenga 1 (indirizzo 101101), anche con l'azzeramento di B, affinché il controllo riparta sempre dall'indirizzo 000001 del μprogramma di estrazione.

Una situazione nuova, che è opportuno esaminare con attenzione, si presenta nell'istruzione RDA. Come abbiamo già visto studiandone la μsequenza, tale istruzione contiene un ciclo, in cui si carica ripetutamente il flip-flop B, e da cui si esce quando il contenuto di B diviene 1. Tale ciclo è realizzato dalla μistruzione di indirizzo 110010 che, specificando il prossimo stato $y'_1 y'_0 = 10$, mantiene il controllo su tale indirizzo finché B rimane 0, e lo trasferisce all'indirizzo 110110 quando B diviene 1; in questo caso il μprogramma termina con l'azzeramento dei registri IR e B.

Nessun particolare commento richiede l'ultimo μprogramma per WTA.

Esercizi

13.3 Descrivere come deve essere modificato lo schema del calcolatore, le μsequenze e il contenuto della ROM, se l'istruzione SBA viene sostituita con la JOW X (*Jump on Overflow*) con significato: "salta all'istruzione nella cella X, se vi è stato supero di capacità nella ALU". (Nota: il segnale di supero che si vuole controllare, generato ovviamente da un'operazione precedente la JOW, deve essere caricato in un apposito flip-flop.)