(17)

CENTRO STUDI CALCOLATRICI ELETTRONICHE

Universita' di Pisa

E. Fabri

Appunti delle lezioni di :

INTRODUZIONE ALLA PROGRAMMAZIONE DI UNA CALCOLATRICE ELETTRONICA

Raccolte e redatte da:

Luigina Bosman Fabri

Lezione 1ª

Scopo di queste lezioni è di illustrare i procedimenti che si seguono nel compilare i programmi per le C. E., cioè per tradurre nel linguaggio delle C.E. i calcoli che bisogna far fare alla macchina stessa. E' ovvio che per far ciò non si può prescindere completamente dall'organizzazione di una C.E., descriveremo quindi molto brevemente come una calcolatrice è costituita, mettendo in risalto le caratteristiche fondamentali che interessano per la sua utilizzazione esterna.

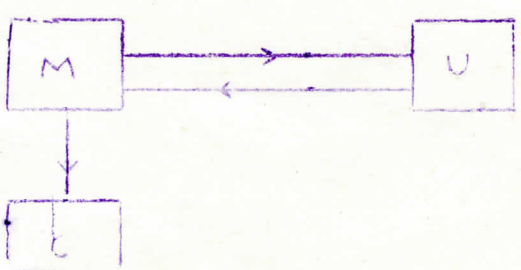
Una calcolatrice elettronica è costituita da tre complessi fondamentali: memoria, unità aritmetica, controllo, e precisamente:

Memoria: è un organo che ha la funzione di conservare numeri (dati e risultati di calcoli) ed istruzioni.

Unità aritmetica: è un organo la cui funzione principale è quella di eseguire operazioni aritmetiche.

Controllo: è un organo che determina che cosa ogni parte della macchina deve fare e la successione temporale delle varie operazioni, cioè coordina le fasi di lavoro della macchina.

La connessione tra gli organi detti può essere rappresentata molto schematicamente nel modo seguente:



dove il collegamento bidirezionale tra M ed U indica simbolicamente il fatto che nell'unità aritmetica vengono eseguite operazioni aritmetiche su dati numerici prelevati dalla memoria, e che nella memoria vengono inviati i risultati di tali operazioni. Il collegamento unidirezionale tra M e C, indica che nel controllo vengono interpretate le istruzioni contenute nella memoria; rimangono quindi esclusi comandi e segnali tra controllo e memoria ed unità aritmetica, perché in questo corso interessa mettere in rilievo la struttura funzionale della macchina. Passiamo ora a descrivere con maggior dettaglio M, U, C.

La memoria è suddivisa in celle ognuna delle quali è capace di conservare una determinata informazione; ogni cella è costituita da un gruppo di elementi ciascuno dei quali può stare in due stati stabili condizione che caratterizza le macchine "binarie". Una cella dunque contiene tante untità di informazione quanti sono gli elementi che la compongono e se, come nella macchina descritta questi elementi sono 18, esprimeremo ciò dicendo che ogni cella contiene una parola, ed ogni parola è di 18 bit (bit : unità di informazione). Abbiamo chiamato parola il contenuto di una cella perché nella memoria si conservano le rappresentazioni di numeri e di istruzioni, e dalla semplice osservazione del contenuto di una cella non si può dire se si tratta di un numero o di una istruzione, la distinzione si può fare solo in base al programma.

Le celle della memoria conservano il loro contenuto anche dopo che l'informazione è stata utilizzata, o come si suol dire, dopo che la parola è stata "letta"; quindi volendo scrivere in una cella una informazione bisogna cancellare il contenuto precedente. Per distinguere l'una dal-

l'altra le celle della memoria le si mette in corrispondenza biunivoca con altrettanti numeri interi; chiameremo "indirizzo" di una cella l'intero ad essa associato e supponendo che la memoria abbia 1024 celle (come nel nostro caso), sceglieremo per indirizzi gli interi da 0 a 1023, inoltre poichó useremo il sistema di numerazione binario gli indirizzi saranno rappresentati ciascuno da dieci cifre binarie ($1024 = 2^{10}$). Passiamo ora alla rappresentazione delle istruzioni; esse sono costituite da due parti, una contenente il simbolo di operazione τ l'altra l'indirizzo x della cella della memoria a cui si riferisce l'istruzione. Es. l'istruzione $T\,x$ significa trasferire il contenuto del registro A (di cui parleremo più avanti) nella cella di indirizzo x. Per rappresentare una istruzione abbiamo dunque a disposizione 18 bit, e poiché per rappresentare un indirizzo occorrono 10 bit mentre per il simbolo di operazione ne bastano 5, perché per il momento useremo meno di 32 istruzioni, potremo rappresentare le istruzioni con una certa larghezza utilizzando i primi 6 bit della parola per il simbolo di operazione e gli altri 12 per l'indirizzo. Per quanto riguarda la rappresentazione dei numeri osserviamo che essendo le parole di 18 bit, si potranno rappresentare solo 2^{18} numeri compresi in un dato intervallo. Per ragioni che qui non possono essere spiegate, per rappresentare i numeri sceglieremo l'intervallo $-1 \longleftrightarrow 1$ [1]. Rimane ora da mostrare come la macchina distingue un numero da una istruzione, dato che entrambi vengono rappresentati nello stesso modo; per far

---

(1) Cfr.: "Aritmetica, logica, organizzazione delle C.E." Lez. 4ª.

ciò dobbiamo esaminare come la macchina stessa opera. Il controllo delle istruzioni, cioè quella parte del controllo adibito all'interpretazione delle istruzioni stesse, è costituito da due elementi fondamentali: il registro delle istruzioni R di 18 bit, ed il numeratore N di 12 bit, che hanno la proprietà, come le celle della memoria, di conservare inalterate le informazioni; vedremo subito la loro funzione. Se ad un dato momento del funzionamento della macchina viene letta una cella della memoria ed il suo contenuto viene scritto in R , questa parola viene interpretata come rappresentazione di una istruzione: i suoi primi 6 bit designano perciò il simbolo di operazione dell'istruzione che verrà eseguita, gli altri 12 l'indirizzo. Se invece il contenuto di una cella della memoria viene inviata nell'unità aritmetica, tale parola viene interpretata come numero e su di esso vengono eseguite le operazioni aritmetiche. La distinzione tra numero ed istruzione viene dunque fatta in base al programma che stabilisce in quali celle della memoria sono contenuti i numeri ed in quali le istruzioni, ed a seconda che tali parole si presentino in R o nell'unità aritmentica verranno interpretate dalla macchina come istruzioni o come numeri su cui operare. Per chiarire meglio questo punto descriveremo il comportamento della macchina: supponiamo di partire da un certo istante in cui in N sia scritto il numero 000001011111, il controllo allora fa leggere la cella della memoria di cui il numero detto è l'indirizzo ed il suo contenuto passa in R (istruzione), qui viene interpretata l'istruzione, e precisamente la parte che indica l'operazione da eseguire passa nel controllo che fa disporre l'unità aritme

---tica in modo da poter eseguire l'operazione contenuta nel l'istruzione in questione, mentre viene letta la cella della memoria corrispondente all'indirizzo dell'istruzione detta e il suo contenuto passa in A, che è il registro della U, detto perciò anche registro aritmetico.

Rimane ora da chiarire come la macchina esegue l'una dopo l'altra le varie istruzioni, cioè la successione temporale di queste. Abbiamo già parlato del numeratore N ed abbiamo già visto che esso è tale da contenere l'indirizzo della istruzione che di volta in volta viene eseguita; dunque la successione temporale delle varie istruzioni di indirizzi consecutivi avviene nel modo seguente: appena letto il contenuto della cella della memoria il cui indirizzo è scritto nel numeratore, il contenuto di N viene aumentato di una unità e la macchina è pronta per eseguire l'istruzione successiva. L'aumento di una unità del contenuto del numeratore non avviene sempre altrimenti non sarebbe possibile interrompere in un punto voluto la successione delle istruzioni; esiste dunque la possibilità di modificare completamente il contenuto di N facendo in modo che ad una istruzione di dato indirizzo ne segua un'altra di indirizzo non consecutivo al precedente.

Prima di dare la lista delle istruzioni introdurremo la nomenclatura ed il simbolismo usato per rappresentare i programmi. Indicheremo con lettere maiuscole (latine) tutti gli organi della macchina dotati di memoria e precisamente i registri A, R, N e le celle della memoria; sia ad esempio Q una di queste celle, introdurremo ora l'operatore δ che applicato al simbolo della cella ci fornisce l'indirizzo della cella stessa, quindi se q è tale indirizzo avremo :

$$\delta Q = q$$

espressione che si legge: l'indirizzo della cella di nome Q è q. Accanto a δ introduciamo l'operatore inverso Δ tale che:

$$Q = \Delta q$$

cioè Q è il nome della cella il cui indirizzo è q. Inoltre per indicare il contenuto della cella Q introduciamo l'operatore γ che dal nome della cella fa passare al suo contenuto:

$$\gamma Q = a$$

cioè a è il contenuto della cella di nome Q, ed inversamente :

$$Q = \Gamma a$$

Q è il nome della cella il cui contenuto è a. Ancora indicando con p l'intero rappresentato dalle ultime 12 cifre della parola contenuta nella cella di nome Q, chiamiamo β l'operatore che dal nome della cella fa passare all'indirizzo dell'istruzione in essa contenuta; per cui avremo:

$$\beta Q = p$$

ed inversamente:

$$Q = \mathcal{B} p$$

($\mathcal{B}$ indica β maiuscolo per evitare di confondere con la corrispondente lettera latina), che esprime che Q è il nome della cella in cui il gruppo delle ultime 12 cifre del suo contenuto è p. Per completare queste notazioni useremo il simbolo $\longrightarrow$ per indicare che la parola a

sinistra del segno viene scritta nella cella o registro il cui nome si trova indicato a destra. Es. la scrittura :

$$\gamma A \rightarrow \Delta x$$

significa: il contenuto (numero del registro A va (viene scritto) nella cella il cui indirizzo è x .

Lezione 2ª

E' stata data nella lezione precedente una descrizione molto sommaria degli organi fondamentali della macchina per illustrare l'utilizzazione esterna della calcolatrice, passeremo ora a specificare il codice della macchina stessa, cioè la lista delle istruzioni. Avremo dunque:

simbolo di operazione	indirizzo	descrizione simbolica
A+	x	$\gamma A + \gamma \Delta x \rightarrow A$ $(\beta N+1 \rightarrow N)$
A-	x	$\gamma A - \gamma \Delta x \rightarrow A$ (")
C+	x	$\gamma \Delta x \rightarrow A$ (")
C-	x	$-\gamma \Delta x \rightarrow A$ (")
T	x	$\gamma A \rightarrow \Delta x$ (")
F	x	ALT $(x \rightarrow N)$
Z+	x	$\gamma A \geq 0$: $x \rightarrow N$; $\gamma A < 0$: $\beta N+1 \rightarrow N$
Z	x	$x \rightarrow N$

e precisamente:

A+ x significa sommare al contenuto di A il contenuto della cella di indirizzo x.

A- x sottrarre dal contenuto di A il contenuto della cella di indirizzo x.

C+ x scrivere in A il contenuto della cella di indirizzo x.

C- x scrivere in A il contenuto della cella di indirizzo x cambiato di segno.

T x trasferire il contenuto di A nella cella di indirizzo x.

F x fermarsi disponendosi a ripartire dall'istruzione di indirizzo x.

Z+ x saltare all'istruzione di indirizzo x se il contenuto di A è positivo.

Z x saltare all'istruzione di indirizzo x.

Le istruzioni A+ A- C+ C- T sono istruzioni normali in quanto in corrispondenza di esse vengono eseguite delle operazioni effettive; inoltre esse vengono eseguite l'una dopo l'altra nel senso che appena terminata una istruzione la cui rappresentazione era contenuta in una cella di dato indirizzo si passa all'istruzione contenuta nella cella di indirizzo successivo. Questo però non può accadere sempre, altrimenti la macchina non potrebbe fare altro che esplorare tutte le celle della memoria dalla prima all'ultima senza fermarsi. Passiamo quindi alla funzione delle istruzioni indicate nella lista data con Z e Z+; dell'istruzione Z, che prende nome di "salto incondizionato" si comprende abbastanza l'utilità, quando

si pensi che le celle che contengono le istruzioni ad un dato calcolo possono non essere tutti consecutivi, per cui eseguita una istruzione, la successiva sarà contenuta in una cella della memoria di indirizzo non consecutivo a quello dell'istruzione precedente; il salto incondizionato consente appunto di eseguire questo passaggio tra istruzioni contenute in celle non consecutive. L'istruzione Z+, che prende nome "salto condizionato", è assai utile quando la macchina debba eseguire un corso di operazioni piuttosto che un altro a seconda dei risultati ottenuti precedentemente. Questo si realizza facilmente se si riesce a subordinare la decisione sul corso da far prendere al programma successivo al segno di un numero ottenuto come risultato dei calcoli precedenti: infatti il tratto di programma che verrà eseguito dopo l'istruzione Z+ sarà quello che comincia con l'istruzione contenuta nella cella di indirizzo x se il numero di cui sopra (che supponiamo si trovi contenuto in A) è maggiore o uguale a zero; sarà invece quello che comincia con l'istruzione contenuta nella cella successiva a quella in cui è contenuta l'istruzione Z+ se il numero detto è minore di zero. Infine F è l'istruzione di arresto che ovviamente serve a fermare la macchina; dal codice si vede che essa è scritta F x per indicare che l'arresto della macchina è accompagnato dalla trascrizione dell'indirizzo x in N, essendo x l'indirizzo della prima istruzione del programma successivo.

Vediamo ora alcuni esempi di programmi, e cominceremo da un caso molto semplice, ma tuttavia abbastanza istruttivo.

Dati tre numeri c_1, c_2, c_3 scritti nelle celle della memoria C_1 C_2 C_3, trovare la loro somma ed inviar

la nella cella di nome D. Avremo allora secondo il simbolismo esposto :

$$c_1 = \gamma C_1 \qquad c_2 = \gamma C_2 \qquad c_3 = \gamma C_3$$

ed il programma è:

C+	δC_1	$\gamma C_1 \rightarrow A$	$(\beta N+1 \rightarrow N)$
A+	δC_2	$\gamma A + \gamma C_2 \rightarrow A$	$(\beta N+1 \rightarrow N)$
A+	δC_3	$\gamma A + \gamma C_3 \rightarrow A$	"
T	δD	$\gamma A \rightarrow D$	"
F	α		$(\alpha \rightarrow N)$

Dal programma scritto risulta che per sommare i numeri contenuti nelle celle C_1 C_2 C_3 ed inviarne il risultato in D sono necessarie cinque istruzioni; se i numeri da sommare fossero cinque basterebbe aggiungere nel programma altre due istruzioni del tipo A+ x ; si potrebbe allora pensare di generalizzare il procedimento in modo che il programma relativo ad n numeri sia costituito di n+2 istruzioni; ma se la macchina per ripetere n volte le stesse operazioni ha bisogno di un programma in cui le istruzioni relative siano ripetute n volte, essa è di ben poca utilità, nel senso che differisce da una calcolatrice elettromeccanica solamente per essere più veloce; in realtà una calcolatrice elettronica ha delle prestazioni qualitativamente diverse ed è in grado di eseguire un calcolo in cui si debbano ripetere le operazioni n volte, con un programma in cui oltre alle istruzioni concernenti l'insieme di operazioni da ese

guire, siano contenute le istruzioni relative alla ripetizione delle operazioni stesse. Vedremo in seguito come ciò può farsi.

A proposito del programma visto osserviamo che esso è costruito in modo indipendente dai numeri effettivi, figurano infatti solo i contenuti delle celle e gli indirizzi, questo perché la macchina opera su delle variabili e non su numeri.

Passiamo ora ad un altro programma: dati due numeri f e g contenuti rispettivamente nelle celle F e G, inviare nella cella di nome H il maggiore dei due, o se sono uguali f, cioè:

$$\max\{f, g\} \longrightarrow H$$

ove

$$\max\{f, g\} \begin{cases} f & f \geq g \\ g & f < g \end{cases}$$

si tratta dunque di far decidere alla macchina quale dei due numeri sia il maggiore; per far ciò basterà fare la differenza $f - g$ ed a seconda che il risultato sia maggiore o uguale a zero, o minore di zero inviare in H f o g. Il programma allora è :

p	C+	F	$\gamma A = f$
p+1	A-	G	$\gamma A = f-g$
p+2	Z+	q	$\begin{cases} \gamma A \geq 0 & q \to N \\ \gamma A < 0 & P+3 \to N \end{cases}$

p+3	G+	δ G	γA=g
p+4	T	δ H	γH=g
p+5	F	p+q	alt.
q	C+	δ F	γA=f
q+1	T	δ H	γH=f
q+2	F	p+q	alt

L'indirizzo q può essere qualsiasi, possiamo quindi porre q= p+6 in modo da avere tutti gli indirizzi espressi in funzione della sola p. Chiameremo p parametro di posizione perché i simboli (i = 1, 2,.....8) rappresentano la posizione delle celle della memoria che contengono le istruzioni del programma dato, cioè gli indirizzi delle istruzioni, e che sono noti una volta fissato p, cioè non appena si sia deciso da quale cella il programma deve iniziare. Esaminando il programma scritto si vede che esso contiene le due istruzioni:

T δ H

F p+q

rispettivamente nelle celle di indirizzi p+4, p+5 e p+7, p+8, perché tali istruzioni costituiscono il tratto finale del programma sia che si verifichi $f - g > 0$ sia $f - g < 0$, e precisamente nel primo caso queste

due istruzioni verranno eseguite dopo quella di indirizzo p+6 , nell'altro caso dopo quella di indirizzo p+3. Si può allora pensare di evitare di ripetere due volte le stesse istruzioni sostituendo alle istruzioni di indirizzi p+4, p+5 l'istruzione di salto condizionato Z p+6, essendo ora p+6 l'indirizzo dell'istruzione T δ H, ed a seconda che ad essa si proviene dall'istruzione di indirizzo p+3 o p+5 verrà scritto nella cella H rispettivamente g o f . Avremo allora che il programma diviene :

p	C+	δ F
p+1	A-	δ G
p+2	Z+	p+5
p+3	C+	δ G
p+4	Z	p+6
p+5	C+	δ F
p+6	T	δ H
p+7	F	p+8

Abbiamo ottenuto così un programma che rispetto al precedente contiene una istruzione in meno, il che ovviamente permette di risparmiare una cella della memoria. La possibilità di risparmiare celle della memoria, che in questo caso porta un vantaggio non apprezzabile, è molto im

portante quando il numero delle celle della memoria che si possono risparmiare è maggiore di uno.
Per illustrare graficamente la struttura topologica dei due programmi visti possiamo servirci dei seguenti diagrammi.

(a) (b)

Di questi a rappresenta il primo programma in cui dopo l'istruzione di salto condizionato i due tratti di programmi successivi proseguono separatamente, mentre b rappresenta il secondo dove, come si è visto, dopo l'alternativa le due diramazioni confluiscono nel comune tratto finale del programma.

Lezione 3ª

Abbiamo visto precedentemente due esempi di programma di cui il primo molto semplice ed il secondo un po' più complesso; vogliamo ora affrontare una ulteriore complicazione generalizzando il programma del secondo esempio e precisamente vogliamo trovare il maggiore tra n numeri dati ed inviarlo in una determinata cella della memoria.
Abbiamo parlato di generalizzazione del programma già visto perché tale programma può essere eseguito nel modo seguente: confrontati i primi due numeri, come nel caso già visto, e trovato il maggiore, si confronta quest'ultimo con il suc

cessivo numero da confrontare, trovato il maggiore si passa al quarto numero e così via. In tal modo però sarebbero necessarie molte istruzioni, ad es. 800 nel caso di 100 numeri, il che naturalmente sarebbe poco pratico; ancora il programma risulterebbe costituito da n parti uguali, avendosi per ogni confronto sempre le stesse istruzioni con indirizzi diversi. Viene allora l'idea di procedere nel modo seguente: eseguito una volta il confronto tra due numeri dati, ed inviato il maggiore nella cella voluta, si modificano le istruzioni in modo da avere pronto il passo successivo, in modo cioè che dopo il primo confronto le stesse istruzioni si riferiscano all'indirizzo della cella che contiene il successivo numero da confrontare, e questo fino ad avere esaurito tutti i numeri dati. Questo è possibile perché nella macchina le istruzioni ed i numeri sono rappresentati nello stesso modo (la distinzione avviene in base al programma) e, sappiamo che il contenuto di una cella della memoria è un numero o una istruzione a seconda che tale contenuto passa nell'unità aritmetica o nel registro R delle istruzioni; ma la rappresentazione di una istruzione può essere considerata rappresentazione di un numero su cui si possono quindi eseguire operazioni aritmetiche e si ottiene così la modifica delle istruzioni. Dunque dovendo ripetere molte volte le stesse operazioni su numeri diversi, si ricorre alla modifica delle istruzioni istituendo quello che si dice un "programma ciclico"; l'esempio scelto si presta bene ad illustrare le caratteristiche fondamentali comuni ai programmi ciclici.

La situazione è dunque la seguente: si hanno n celle della memoria $G_1 \ldots G_n$ contenenti gli n numeri da

confrontare:

$$G_1 \quad G_2 \qquad\qquad G_n$$

$$g_1 \quad g_2 \qquad\qquad g_n$$

e

$$g_i = \gamma\, G_i$$

supponiamo poi che gli indirizzi delle n non siano disposti a caso ma che esista una legge che faccia passare da uno all'altro, ad es. siano consecutivi, in modo che:

$$\delta G_\ell = \delta G_1 + (\ell - 1)$$

Vogliamo determinare il $\max(g_i)$ cioè trovare $h = \max(g_1 \cdots g_n)$ ed il suo indirizzo, e mandare h nella cella H ed il suo indirizzo in K.
Indicando con h_ℓ il massimo fra i primi ℓ numeri, cioè:

$$h_\ell = \max\{g_1 \cdots g_\ell\} = g_{k_\ell}$$

dove g_{k_ℓ} indica che h_ℓ è uno dei g_i $(i = 1 \cdots \ell)$ precisamente quello di posto k_ℓ ed ancora che esso dipende dall'indice ℓ ; si vede che valgono le relazioni:

$$h_\ell = \max\{h_{\ell-1}, g_\ell\}$$

cioè:

$$h_\ell = \begin{cases} h_{\ell-1} & g_\ell < h_{\ell-1} \\ g_\ell & g_\ell \geq h_{\ell-1} \end{cases}$$

parallelamente si ha:

$$K_\ell = \begin{cases} K_{\ell-1} & g_\ell < h_{\ell-1} \\ \ell & g_\ell \geq h_{\ell-1} \end{cases}$$

relazioni che esprimono che si ha a che fare con un proce dimento ricorrente. Vediamo ora in dettaglio il tratto di programma fondamentale. Supponiamo di partire dalla situa zione seguente:

$$\gamma H = h_{\ell-1} \qquad \gamma K = \delta G_{K_{\ell-1}}$$

per sostituire ad $h_{\ell-1}$ il nuovo massimo si ha il se- guente programma:

P_1	C+	δG_ℓ	$\gamma A = g_\ell$	$(\beta N+1 \to N)$
P_1+1	A-	δH	$\gamma A = g_\ell - h_{\ell-1}$	(")
P_1+2	Z+	P_2	$\gamma A \geq 0$	$P_2 \to N$
			$\gamma A < 0$	$\beta N+1 \to N$

se allora $g_\ell - h_{\ell-1} \geq 0$ essendo g_ℓ il nuovo massimo dovremo mandarlo nella cella H ed il suo indirizzo in K, avremo allora:

P_2	A+	δH	$\gamma A = g_\ell$
P_2+1	T	δH	$\gamma H = g_\ell = h_\ell$
P_2+2	C+	P_1	$\gamma A = \rho(C+, \delta G_\ell)$
P_2+3	A-	δR	$\gamma A = \rho(2^{-17}, \delta G_\ell)$
P_2+4	T	δK	$\gamma K = 2^{-17}\, \delta G_\ell$

in cui la rappresentazione dell'istruzione di indirizzo p_1 è stata trattata come rappresentazione di un numero da cui è stato sottratto il contenuto della cella R che è ancora una istruzione, ma relativa all'indirizzo zero, $\gamma R = \varphi(C+, 0)$. La funzione di tale operazione è quella di separare dal simbolo di operazione l'indirizzo di G_ℓ che figura infatti solo nell'istruzione di indirizzo p_1 ; si ottiene così:

$$\begin{array}{ll} C+ & \delta G_\ell \; - \\ C+ & 0 \; = \\ \hline 0 & \delta G_\ell \end{array}$$

Dopo ciò potremmo aver finito se i numeri da confrontare sono finiti, ma per essere sicuri di ciò dobbiamo sapere se $\ell = m$, in modo da uscire dal ciclo o ripeterlo in caso contrario; per fare ciò costruiamo un tratto di programma che operando sugli indici ℓ ed m a seconda del valore della differenza $\ell - m$ decida se si debba o no uscire dal ciclo; occorrerà allo scopo avere in qualche cella una parola che contenga l'indice ℓ , e poiché si trova in queste condizioni la cella di indirizzo p_1 si ha:

$$\begin{array}{llll} p_3 - & C+ & p_1 & \gamma A = \varphi(C+, \delta G_\ell) \\ p_{3+1} & A- & \delta S & \gamma A = 2^{-17}(\ell - n) \\ p_{3+2} & Z+ & p_5 & \begin{cases} \ell - n = 0 & p_5 \to N \\ \ell - n < 0 & \beta N+1 \to N \end{cases} \end{array}$$

dove S è una cella della memoria tale che: $\gamma S = \varphi(C+, \delta G_n)$.

Se allora $l = m$ il ciclo è terminato e come risulta dall'istruzione $p_3 + 2$ si passerà all'istruzione di indirizzo p_5 in cui sarà contenuta l'istruzione di arresto; se invece $l < m$ si deve rientrare nel ciclo ed eseguire le operazioni precedenti sul numero g_{l+1} ma per far ciò bisognerà modificare le istruzioni, cioè trasformare il contenuto di Δp_1 da $C+ \delta G_l$ a $C+ \delta G_{l+1}$; a ciò serve il seguente programma:

$$p_4 \quad C+ \quad p_1 \quad \gamma A = \varphi(C+, \delta G_l)$$

$$p_4+1 \quad A+ \quad \delta T \quad \gamma A = \varphi(C+, \delta G_{l+1})$$

$$p_4+2 \quad T \quad p_1 \quad \gamma \Delta p_1 = \varphi(C+, \delta G_{l+1})$$

dove $\gamma T = \varphi(2^{-17})$

Il ciclo allora ricomincia e si ripetono tutte le fasi descritte: confronto, determinazione del maggiore, invio di questo e del suo indirizzo in H e K, decisione di uscire dal ciclo, modifica delle istruzioni e di nuovo confronto dell'ultimo massimo trovato con il successivo numero.

Resta da vedere ora come si entra nel ciclo, dato che il programma va bene per tutti gli indici tranne che per $l = 1$ in tal caso infatti non conoscendo il contenuto della cella H, non potremmo far seguire alla istruzione $C+ \delta G_1$ l'altra $A- \delta H$; il programma iniziale allora deve essere diverso e precisamente:

P_6	C+	δG_1	$\gamma^A = \delta G_1$
P_6+1	T	δH	$\gamma^H = \delta G_1$
P_6+2	C+	P_1	$\gamma^A = \varphi(C+, \delta G_1)$
P_6+3	A-	δT	$\gamma^A = \delta G_1$
P_6+4	T	δK	$\gamma^K = \delta G_1$

Per terminare vediamo ora come si esce dal ciclo, ricordando che il tratto di programma che inizia con l'istruzione P_3 conteneva alla fine (dopo la decisione) la istruzione $Z+ \; P_5$ avremo che P_5 è l'indirizzo della cella che contiene l'istruzione di arresto, e poiché il programma è terminato potremo utilizzare le celle di indirizzo successivo a P_5 per individuare le celle R, S, T. Avremo quindi:

P_5	F	P_3	
P_5+1	C+	0	(R)
P_5+2	C+	δG_m	(S)
P_5+3	0	$\varphi(2^{-17})$	(T)

Lezione 4ª

Per completare il programma visto bisogna montare le varie parti cioè mettere insieme i vari tratti di programma P_i (i=1....6) di cui il programma stesso ri

sulta costituito. Abbiamo infatti i seguenti gruppi di istruzioni :

P_1

P_1	C+	δG_ℓ
P_1+1	A-	δH
P_1+2	Z+	P_2

P_2

P_2	A+	δH
P_2+1	T	δH
P_2+2	C+	P_1
P_2+3	A-	δR
P_2+4	T	δH

P_3

P_3	C+	P_1
P_3+1	A-	δS
P_3+2	Z+	P_5

P_4

P_4	C+	P_1
P_4+1	A+	δT
P_4+2	T	P_1

P_5

P_5	F	P_7
P_5+1	C+	0
P_5+2	C+	δG_m
P_5+3	/	1

P_6

P_6	C+	δG_1
P_6+1	T	δH
P_6+2	C+	P_1
P_6+3	A-	δR
P_6+4	T	δK

di cui:

P_1 contiene le operazioni relative al confronto tra g_ℓ ed $h_{\ell-1}$ che è il massimo tra i primi $\ell-1$ g_i e termina con la decisione sul come si debba proseguire a seconda di quale sia il maggiore.

P_2 ha la funzione di scrivere in H il maggiore tra g_ℓ ed $h_{\ell-1}$ ed in K l'indirizzo; operazioni che vanno eseguite solo nel caso che g_ℓ sia maggiore del massimo precedente.

P_3 decide se si debba ripetere il ciclo oppure no a seconda che dal confronto tra gli indici n ed ℓ risul

ti $\ell < n$ o $\ell = n$; in altre parole controlla se sono stati esaminati tutti i numeri da confrontare.

P_4 ha il compito di modificare il tratto di programma relativo al confronto tra il massimo precedente ed il numero successivo, prima che questo venga eseguito di nuovo.

P_5 contiene l'istruzione di arresto con preparazione a ripartire dall'istruzione di indirizzo P_7 , e le rappresentazioni delle costanti contenute nelle celle R, S, T i cui indirizzi figurano in P_2, P_3, P_4.

P_6 serve per iniziare il programma perché, come abbiamo visto, il tratto di programma fondamentale non va bene per $\ell = 1$.

Per quanto riguarda le connessioni cominciamo col fare i primi ritocchi, cioè a sostituire δR con $p_5 + 1$, δS con $p_5 + 2$ e δT con $p_5 + 3$; ancora a δG_ℓ nel tratto P_1 va sostituito δG_2 in quanto questo tratto di programma viene eseguito per la prima volta per $\ell = 2$ e, poiché le celle in cui sono contenuti i numeri sono consecutive avremo $\delta G_2 = \delta G_1 + 1$ ed ancora sarà: $\delta G_n = \delta G_1 + n - 1$.

Mancano ora gli indirizzi di G_1, H, K che insieme ad n che determina il numero di ripetizione del ciclo sono considerati i parametri di lavoro e che restano determinati una volta deciso da quale cella della memoria deve iniziare il programma, la posizione di H e K ed il numero dei g_ℓ da confrontare. Riassumendo abbiamo dunque per i valori P_i le seguenti funzioni:

P_1 salta a P_2 se $g_\ell \geq h_{\ell-1}$ $(\ell \geq 2)$ e passa a P_3 se $g_\ell < h_{\ell-1}$

P_2 pone h_ℓ in H e k_ℓ in K.

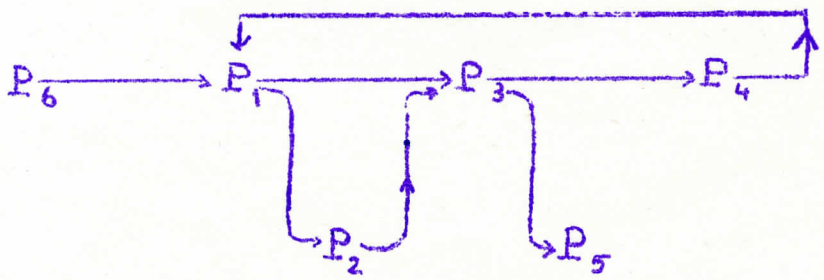
P_3 salta a P_5 se $\ell = n$ e passa a P_4 se $\ell < n$

P_4 pone $\ell + 1$ in luogo di ℓ e passa a P_1.

P_5 arresto (e celle per le costanti).

P_6 entrata ($\ell = 1$).

Ricaviamo quindi le connessioni logiche:



Questo diagramma permette di stabilire le relazioni tra i vari P_i , cioè di ordinare la successione degli indirizzi. Infatti potremo disporre di seguito gli indirizzi delle istruzioni dei gruppi P_6, P_1, P_3, P_4 i quali contengono operazioni che vanno eseguite successivamente nel tempo se non ci sono salti, per cui si avrà:

$$P_1 = P_6 + 5$$

$$P_3 = P_1 + 3$$

$$P_4 = P_3 + 3$$

Al gruppo P_4 bisogna poi aggiungere l'istruzione:

$$P_4 + 3 \quad \geq \quad P_1$$

perché da P_4 si deve rientrare in P_1 . Quanto alle istruzioni dei gruppi P_2 e P_5 che rappresentano le altre due alternative con cui terminano rispettivamente P_1 e P_3 , poiché ad esse si passa nel caso di salto, potranno avere indirizzi a piacere per cui potremo decidere di

disporle in ordine dopo P_4 . Ancora aggiungeremo al gruppo P_2 l'istruzione:

$$P_2 + 5 \quad Z \quad P_3$$

perché dopo le istruzioni di P_2 si deve passare a quelle del gruppo P_3 . Infine potremo disporre di seguito al gruppo P_5 l'indirizzo che figura nell'istruzione di arresto. Avremo allora nell'ordine:

$$P_2 = P_4 + 4$$

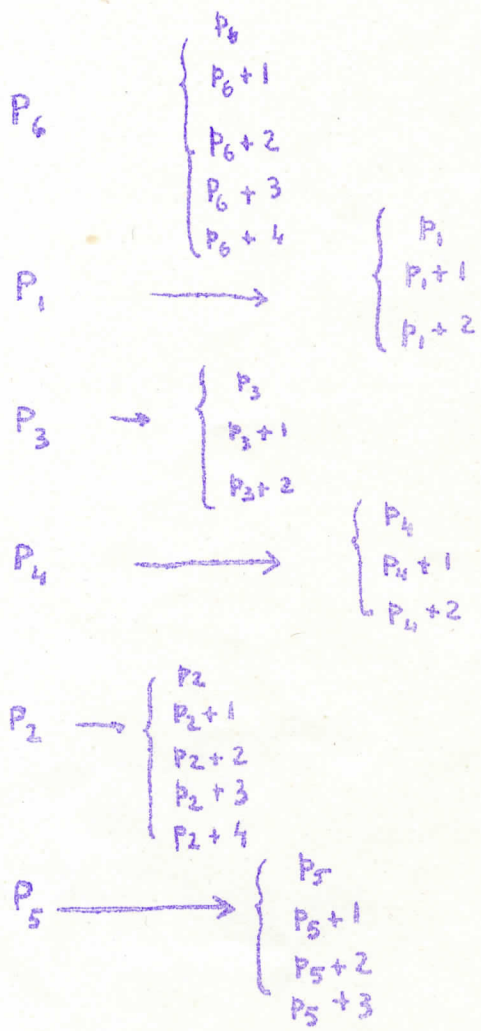
$$P_5 = P_2 + 6$$

$$P_7 = P_5 + 4$$

Possiamo ora esprimere tutti gli indirizzi delle istruzioni in funzione di un solo parametro, ad es. p_6 che caratterizza il tratto di entrata del programma, avremo così:

$$P_1 = p_6 + 5$$
$$P_3 = P_6 + 8$$
$$P_4 = P_6 + 11$$
$$P_2 = P_6 + 15$$
$$P_5 = P_6 + 21$$
$$P_7 = P_6 + 25$$

e posto p in luogo di p_6 (trattandosi di un solo parametro non è più necessario conservare l'indice) il programma, finalmente montato può essere scritto nel modo seguente :

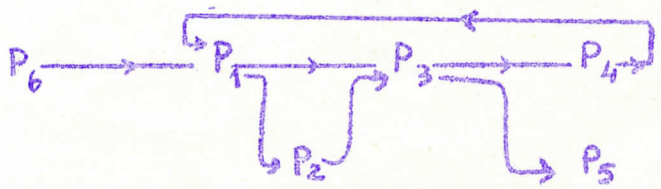


di cui rimane indeterminato solo (insieme ad n, δH, δK), che viene fissato appena deciso da quale cella deve iniziare il programma.

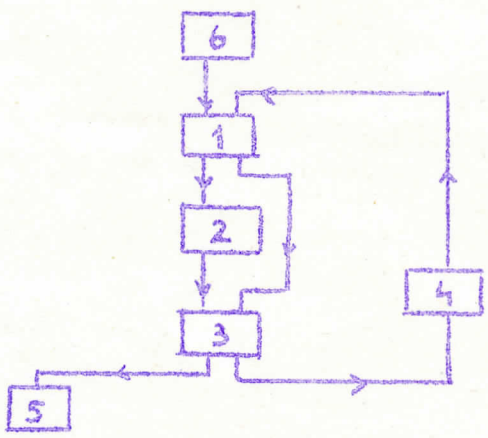
Lezione 5ª

Ci occuperemo in questa lezione del "diagramma dinamico" (flow-diagram) che consiste essenzialmente nel dare del programma uno schema a blocchi che corrispondono ai vari gruppi di istruzioni, di cui il programma stesso risulta costituito, e ne rappresentano la relativa funzio

ne prescindendo completamente da come le operazioni dei vari gruppi vengono eseguite e dalla disposizione delle istruzioni nelle celle della memoria. Si tratta dunque in sostanza di una rappresentazione funzionale delle varie parti che compongono il programma. Per mostrare ciò partiremo dal programma visto nelle lezioni scorse; di esso abbiamo una rappresentazione grafica mediante il diagramma:



che possiamo modificare nella maniera seguente:



Le varie parti di tale diagramma hanno, come si è visto, funzioni diverse, e precisamente:

[6] ha la funzione di trasferire q_i in h ed il suo indirizzo in k, e sebbene questa non sia una operazione aritmetica può essere considerata tale in quanto equivale a calcolare il primo massimo $(h_1 = q_i)$. Questo blocco quindi determina un numero.

[1] confronta q_i con h_{i-1}, e anche se questo gruppo contiene delle operazioni aritmetiche ha una funzione logi

ca in quanto le operazioni servono a decidere tra due alternative. Chiameremo questo "blocco di decisione".

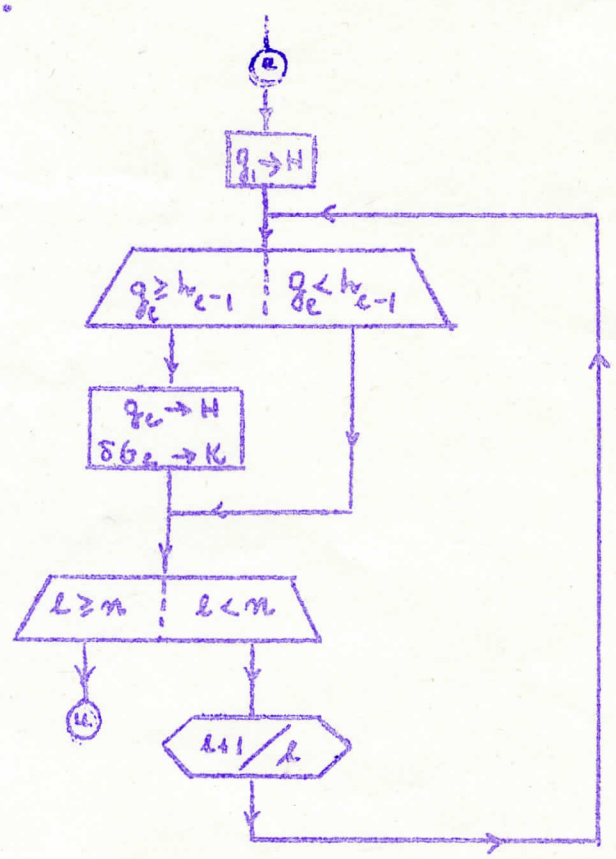
[2] a cui si passa nel caso che $q_\ell = h_\ell$ fornisce il nuovo massimo ed il suo indirizzo, cioè è "un"blocco di operazioni".

[3] è ancora un "blocco di decisione".

[4] pone $\ell + 1$ in luogo di ℓ , e lo chiameremo "blocco di alterazione".

5 blocco di uscita.

Avremo così il seguente schema a blocchi, in cui le varie parti sono rappresentate in forme diverse a seconda della funzione.



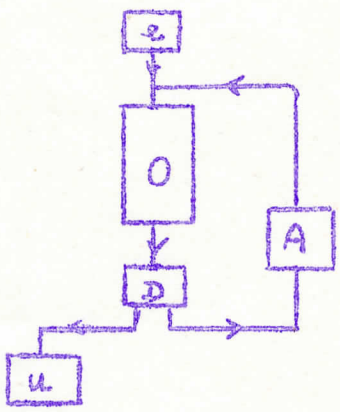
Abbiamo così ottenuto il diagramma dinamico del programma visto; in esso come si vede non sono contenute indicazioni circa la successione degli indirizzi delle celle della

memoria che contengono le istruzioni, né circa il modo in cui le operazioni vengono eseguite. Tuttavia il diagramma dinamico contiene qualche cosa di più del programma come successione di istruzioni, nel senso che ne mette in risalto l'aspetto funzionale dei vari tratti di cui è costituito; ciò nonostante non conviene usare il d.d.dopo aver già scritto il programma, esso infatti è molto più utile come primo passo nell'elaborazione del programma perché può essere costruito prescindendo dalla conoscenza della realizzazione dei vari blocchi; quindi in realtà il diagramma dinamico poteva essere costruito prima di scrivere il programma e la ragione del fatto che si è proceduto diversamente è che nell'affrontare per la prima volta i problemi relativi alla programmazione conviene seguire il procedimento che sembra più spontaneo. Infatti l'avere scritto prima il programma ci ha permesso di parlare di gruppi di istruzioni, che corrispondono ai blocchi, cioè ha facilitato la comprensione del significato e della funzione del diagramma dinamico. Per quanto riguarda l'utilizzazione dei d.d. la questione è controversa, c'è infatti chi ritiene che il d.d. costituisce il primo passo nella compilazione di un programma, c'è invece chi ritiene che nell'elaborazione di un programma non è affatto necessario il d.d.; probabilmente il procedimento più corretto è quello di usare il d.d., ma accompagnandolo ad una idea sulla struttura dei vari blocchi, cioè le due cose vanno insieme.

Tornando ora al diagramma dinamico del nostro programma, vogliamo esaminare una caratteristica particolare di questo programma ma che caratterizza la struttura di una intera classe di programmi (cicli), questo nostro ciclo risulta dunque costituito da tre blocchi fondamenta

li, oltre all'entrata ed uscita, e precisamente "blocco di operazioni", di "decisione" e di "alterazione" (per blocco di operazioni intendiamo ora l'insieme dei gruppi di istruzioni 1 e 2 perché, come abbiamo visto, essi hanno complessivamente la funzione di calcolare $h_e = \max\{h_{e-1}, g_e\}$ e, come già si è detto, qui si vuole prescindere dalla organizzazione interna del blocco stesso, cioè dal tipo di istruzioni usate per realizzare i vari tratti di programma).

Avremo allora il seguente schema:



Questo è il d.d. caratteristico di quel che si chiama programma ciclico o brevemente: ciclo. Da un punto di vista funzionale un ciclo è un tratto di programma costruito con lo scopo di eseguire più volte una stessa operazione (o insieme di operazioni). Con la "stessa operazione" si intende dire o che la stessa operazione viene eseguita su contenuti di celle progressivamente diversi, o anche che viene eseguita sempre sulle stesse celle, ma che i contenuti di esse vanno progressivamente cambiando nel corso delle successive ripetizioni (come nel caso iterativo). Es. si pensi al calcolo della potenza ennesima (n = = 2,4,8) di un numero a. Dal punto di vista del numero di ripetizione distinguiamo tra due tipi di cicli e precisa

mente il numero di ripetizione può essere prefissato (come nel caso in cui si debbano ripetere le stesse operazioni su un numero assegnato di elementi diversi), o è condizionato al verificarsi di una certa situazione (come nel caso di uno sviluppo in serie che debba venir arrestato quando il resto è divenuto minore di una quantità prefissata e non si sa a priori quando ciò accada). Questa differenza fa sì che il ciclo venga organizzato diversamente a seconda che si verifichi il primo caso o il secondo e il blocco di decisione avrà nei due casi rispettivamente la funzione di contare il numero di ripetizione assegnato, o constatare il verificarsi della situazione in base alla quale si decide di uscire o no dal ciclo.

Un'altra distinzione tra cicli si ha per quel che riguarda il blocco di operazione; infatti il gruppo [O] può venir ripetuto in modo identico ogni volta che viene eseguito (come nel caso del calcolo di $a^{z^{m}}$), cioè l'operazione non contiene istruzioni i cui indirizzi debbano essere modificati ad ogni ripetizione; oppure il gruppo [O] è tale da contenere indirizzi da modificare progressivamente (come nel caso della determinazione del massimo tra numeri dati). Abbiamo dunque in tutto quattro possibilità:

a) ciclo con numero di ripetizione prefissato e blocco fisso.

b) ciclo con numero di ripetizione prefissato e blocco da modificare.

c) ciclo con numero di ripetizione non prefissato e blocco fisso.

d) ciclo con numero di ripetizione non prefissato e blocco da modificare.

Lezione 6ª

Riprendiamo la discussione dei vari tipi di cicli visti nella lezione passata. A proposito del blocco di decisione la distinzione è fatta in base al fatto che il numero di ripetizioni sia prefissato o no; possiamo precisare questo punto introducendo un "indice di ripetizione", che indica istante per istante il numero di volte che il ciclo è stato già eseguito, e aumenta di 1 a ogni nuova ripetizione. Se il numero di ripetizioni è prefissato, la funzione del blocco D è di contare l'indice di ripetizione, determinando l'uscita quando tale indice i=n. In questo caso i deve dunque venire esplicitamente calcolato passo passo.

La seconda possibilità è che la decisione dipenda da una condizione che non implica direttamente i . Un esempio chiarirà meglio la situazione. Si debba calcolare la radice quadrata di un numero dato: esiste un procedimento iterativo, il cui limite fornisce il risultato cercato; siano z_0, z_1, ... z_i, ... i successivi risultati. Poiché il risultato finale interessa solo con una certa approssimazione, imporremo una condizione per il termine del calcolo, che potrà essere del tipo:

(1) $$| z_i - z_{i-1} | < \varepsilon$$

E' chiaro che la verifica della (1) non richiede la conoscenza esplicita di i, ma solo quella dei risultati dell'iterazione: l'indice di ripetizione viene qui introdotto per ragioni descrittive, ma non ha una funzione effettiva nel calcolo.

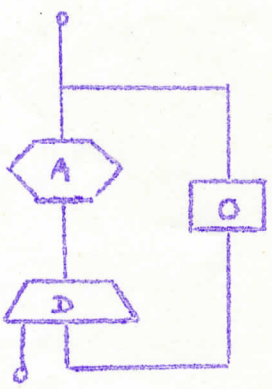
Per quanto riguarda il blocco di operazione si è fatta una distinzione a seconda che occorresse una modifi

ca delle istruzioni del blocco a ogni passo, oppure no. Vediamo ora da che cosa dipenda una tale necessità. Se le istruzioni del blocco O debbono essere modificate, ciò vuol dire che le celle da cui si prelevano i dati, o quelle in cui si inviano i risultati, non sono sempre le stesse, ma cambiano a ogni passo. Una tale circostanza non si presenta nell'esempio ora citato del calcolo della radice quadrata, in quanto il dato è unico, e anche il risultato finale è unico: si può dunque conservare tutti i risultati parziali nella stessa cella, sostituendo a ogni passo il risultato vecchio col nuovo, senza perdere nessuna informazione utile. Lo stesso accade nella ricerca del massimo fra i numeri: qui però la modifica delle istruzioni è necessaria perché varia a ogni passo la cella da cui si preleva il numero da confrontare con il massimo parziale già trovato.

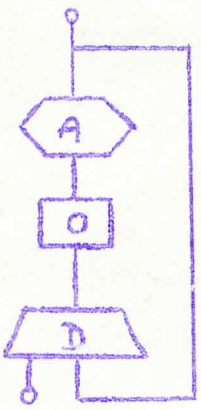
Riassumendo: la natura della decisione può richiedere o no la conoscenza esplicita (e quindi il calcolo) dell'indice di ripetizione; lo stesso si ha per il blocco di operazione, le cui istruzioni, se devono essere modificate, risultano funzioni di i, e ne richiedono quindi la conoscenza esplicita. Ciò non accade se non è necessaria modifica. In tutti i casi in cui i è esplicitamente necessario, esso deve essere calcolato e conservato; è possibile in particolare, come è stato fatto nel nostro programma, conservare l'indice di ripetizione proprio attraverso la forma che assumono via via le istruzioni di O (v. il modo come è stata realizzata la decisione).

Passiamo ora a un altro problema: quello della disposizione dei vari blocchi di istruzione nella memoria, in rapporto alle loro connessioni indicate nel

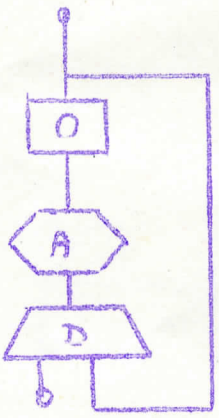
diagramma dinamico. Vedremo che per tale problema non esiste una soluzione unica, e la scelta può dipendere da quello che più interessa raggiungere caso per caso (risparmio di tempo, di spazio di memoria, ecc.). Ci limitiamo ancora a un ciclo semplice, per cui dovremo studiare la disposizione reciproca dei tre blocchi O, A, D. Osserviamo in primo luogo che esistono sei modi di disegnare il d.d. del ciclo, quando si tenga conto della possibilità di entrare nel ciclo su uno qualsiasi dei tre blocchi. In figura si trovano i sei diagrammi dinamici, che ora discuteremo.



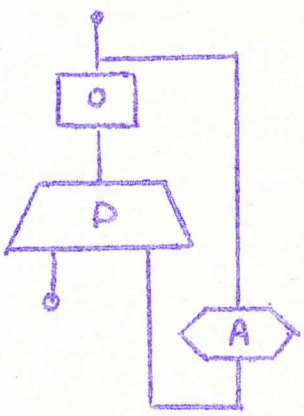
1



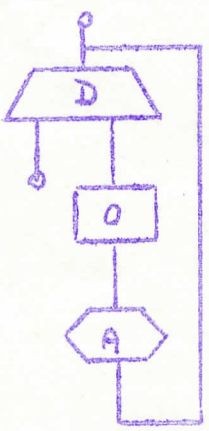
2



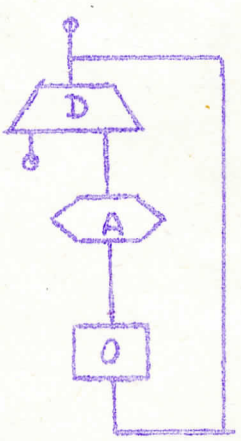
3



4



5



6

In primo luogo si può vedere se i sei casi sono completamente equivalenti dal punto di vista della funzione. Si vede che la sola differenza è che i diagrammi 1, 4, 5, consentono di uscire dal ciclo senza avere attraversato 0, cosa che può riuscire utile quando il numero di ripetizioni non sia prefissato, ma possa anche assumere il valore zero. Che possa essere utile eseguire zero volte un ciclo lo si può vedere dal seguente esempio: si voglia ridurre un numero minore di 1 ad avere la prima cifra significativa subito dopo la virgola. Si procederà allora a spostare la virgola a destra di un posto per volta, finché non si sarà raggiunta la condizione voluta (non è qui possibile spiegare l'interesse pratico di tale problema). Il procedimento è ciclico, e il numero di ripetizioni coincide con quello degli zeri dopo la virgola. Ne segue che se il numero è già nella condizione voluta non si deve fare nessuno spostamento. Se si vuole scrivere un programma che possa eseguire la trasformazione descritta su qualunque numero minore di 1, sarà necessario tener conto della possibilità che il numero di ripetizione assuma il valore 0: questo può essere realizzato solo con i diagrammi 1, 5, 6. Fra questi poi il primo comporta uno spreco, che risulta chiaro se si osserva che nel caso in cui il ciclo non va percorso affatto, viene eseguita un'alterazione che non servirà, ma che comporta una perdita di tempo.

Lezione 7ª

Riprendiamo l'argomento della lezione scorsa costruendo una tabella, dalla quale risulti in corrispondenza di ogni diagramma il numero di volte che vengono ripetu

ti i blocchi A e D per un numero prefissato di ripezioni di O. La costruzione della tabella è immediata:

	1	2	3	4	5	6
A	$n+1$	n	n	$n-1$	n	n
D	$n+1$	n	n	n	$n+1$	$n+1$

passiamo ora a discuterla.

Si vede subito che se si vuole che n possa anche assumere il valore zero, si dovrà limitare alle soluzioni 1, 5, 6; infatti nei casi 2, 3, 4 per n=O si avrebbe un numero di decisioni pure nullo, il che è chiaramente assurdo. Fra le tre soluzioni restanti è evidente la convenienza di 5, 6 su 1, che esegue una volta di più il blocco A. Fra 5, 6 la scelta è invece pressoché indifferente, sebbene si possa preferire 5 che esegue l'operazione prima dell'alterazione.

Se non c'è la possibilità che n valga zero, la soluzione 4 è certo la più economica, poi vengono a parità 2 e 3; vedremo poi come da un altro punto di vista si possa preferire 2 o 3 a 4.

Per comprendere questo nuovo punto di vista occorre fare qualche premessa. Ricordiamo che abbiamo a che fare con una macchina a un indirizzo, nella quale le istruzioni vengono eseguite nello stesso ordine secondo cui sono conservate nella memoria; fanno eccezione i salti, con i quali si può passare a un'istruzione contenuta in una cella qualsiasi. Supporremo che ogni blocco abbia in sé una struttura lineare, cioè non contenga salti, cosic-

ché le sue istruzioni dovranno essere conservate in celle consecutive della memoria, per potere venire eseguite nel corretto ordine. Lo stesso problema, dell'ordine temporale secondo il quale devono venire eseguiti, si presenta però anche fra i vari blocchi: la struttura ciclica impedisce che a tale problema si possa dare soluzione semplicemente disponendo i vari blocchi consecutivamente nell'ordine voluto. Inoltre l'esistenza di una decisione, cioè di un salto condizionato, lascia una certa libertà che occorre sfruttare nel modo migliore.

In linea generale si potrebbe risolvere il problema facendo terminare tutti i blocchi da una istruzione di salto, e determinando poi i vari indirizzi di questi salti in modo da realizzare la connessione che interessa. Può darsi però il caso che non si vogliano introdurre istruzioni superflue, sia per risparmiare spazio di memoria, sia per economia di tempo: si deve allora trovare la soluzione più economica, cioè quella che richiede il minor numero di salti. La ricerca di tale soluzione è subordinata alle seguenti condizioni:

a) si è già stabilito quale delle 6 disposizioni viste dovrà essere adottata;
b) esistono prescrizioni definite sulle modalità dell'entrata nel ciclo e dell'uscita da questo.

Quanto al punto b) occorre precisare di che prescrizioni si tratta. L'entrata può darsi che debba essere diretta, cioè nel ciclo si deve entrare direttamente, senza salto, da un tratto di programma precedente: è chiaro che in tal caso il blocco al quale avviene l'entrata deve figurare in testa nella disposizione del programma. Se tale condizione manca, cioè se si può avere un'entrata

con salto, la disposizione è chiaramente molto più libera. Per quanto riguarda l'uscita, la situazione è simile: se si deve avere l'uscita diretta il blocco finale dovrà essere necessariamente in coda; altrimenti (uscita con salto) la disposizione può essere qualsiasi.

Per comprendere tutti i termini del problema occorre ancora riflettere che il rientro nel ciclo richiederà necessariamente un salto, per il quale si può cercare di sfruttare il salto condizionato del blocco D: ciò sarà naturalmente possibile se non contraddice le esigenze già enunciate; altrimenti si dovrà adottare una soluzione diversa.

Possiamo ora riassumere le soluzioni possibili in una tabella, dove sono indicate le 6 disposizioni base, le 4 condizioni possibili per l'entrata e l'uscita (EDUD significa: entrata diretta, uscita diretta; EDUS: entrata diretta, uscita con salto; ecc.), e in corrispondenza l'ordine da adottare per i vari blocchi allo scopo di evitare, se possibile, l'introduzione di salti addizionali all'interno del ciclo.

	EDUD	EDUS	ESUD	ESUS
1	—	—	OAD	OADZ
2	AOD	AODZ	AOD	AODZ
3	OAD	OADZ	OAD	OADZ
4	—	—	AOD	AODZ
5	—	—	AOD	AODZ
6	—	—	OAD	OADZ

Come si vede nella prima colonna vi sono due sole soluzioni possibili: ciò accade perché l'uscita diretta impone che D si trovi in coda, e restano quindi le due possibilità di entrata su A o su O. La seconda colonna non differisce dalla precedente, in quanto il modo migliore per ottenere un'uscita con salto è di avere un'uscita diretta con un salto finale (che resta così fuori dal ciclo). Per la terza e quarta colonna si hanno sempre soluzioni possibili, poiché c'è la sola restrizione di porre D in coda, seguito o no da un salto (Z).

Dall'insieme delle due tabelle che abbiamo discusso si può ricavare per ogni situazione quale soluzione adottare: è bene però notare che queste considerazioni hanno un interesse molto variabile caso per caso, in quanto l'importanza di sprecare delle celle di memoria dipende dalla capacità di questa: la tendenza attuale essendo verso macchine con memoria sempre più grande, il problema perde sempre più di importanza. Così pure il risparmio di tempo è molto importante se la macchina è lenta, o se il calcolo è molto complesso; può esserlo meno in condizioni diverse. Inoltre sul tempo totale necessario per l'esecuzione di un calcolo influiscono talora in modo determinante altri fattori che non è qui possibile esporre: tutto questo induce a non fare un uso troppo rigido delle regole che abbiamo indicate.

Lezione 8ª

Nelle lezioni precedenti abbiamo descritto molto sommariamente, e su un esempio molto semplificato di macchina, alcuni problemi connessi con l'organizzazione dei pro-

grammi per le macchine a un indirizzo. In questa ultima lezione vogliamo occuparci, nelle grandi linee, di due aspetti del lavoro di programmazione che sono finora rimasti da parte.

In primo luogo consideriamo la questione della entrata nella macchina. Fin qui abbiamo supposto sempre che istruzioni, dati e risultati si trovassero nella memoria, ma non ci siamo mai occupati di come si faccia ad introdurre i dati iniziali e il programma nella memoria, all'inizio di un calcolo. A questo scopo serve l'apparato di entrata, che può essere molto diverso a seconda della macchina, e il cui uso presenta quindi caso per caso caratteristiche molto particolari. Per questo motivo è opportuno limitarsi qui a un rapido cenno, considerando il sistema forse più semplice attualmente in uso: la entrata a nastro perforato.

Si tratta di un nastro di carta, del normale tipo per telescrivente, che presenta, in corrispondenza di sezioni trasversali poste a distanza costante tra loto, degli insiemi di fori disposti in modo vario da sezione a sezione, ma sempre in modo da occupare (o no) cinque posizioni fisse. Ponendo in corrispondenza con ciascuna posizione possibile per i fori una cifra binaria, ogni sezione corrisponderà a un gruppo di 5 cifre binarie: cui daremo il valore 1 se il foro è presente, 0 se il foro manca. E' chiaro che su ogni sezione si possono avere $2^5=32$ combinazioni possibili di fori, ciascuna delle quali rappresenta un carattere alfabetico o numerico, o di altro tipo. Limitandoci qui alla sola rappresentazione delle istruzioni, è chiaro che tre caratteri saranno sufficienti per rappresentarne una: tutto è dunque ricondotto a vedere come si introduce nella

macchina l'informazione rappresentata dalle perforazioni del nastro.

A questo serve, dal punto di vista costruttivo, un organo apposito che trasforma le perforazioni del nastro in segnali elettrici: di questo non possiamo però occuparci; dal punto di vista logico basta invece un'istruzione di entrata: E x, che legge tre caratteri consecutivi dal nastro e trasferisce la parola corrispondente nella cella di indirizzo x.

Può sembrare che a questo punto il problema dell'entrata dei dati e dei programmi nella macchina sia risolto: basterà preparare un nastro perforato recante tutte le informazioni che si vogliono introdurre nella macchina, e farlo leggere a questa. Ma la macchina, per leggere il nastro d'entrata, ed effettuare il trasferimento delle informazioni in esso contenute in celle determinate, ha bisogno di un opportuno programma, che deve già essere contenuto nella memoria: il programma di lettura. Non solo: sarà necessario mettere in moto la macchina, che sarà stata inizialmente ferma; si dovrà precisare che la prima cosa da fare è leggere il nastro, o viceversa eseguire qualche altro programma eventualmente già presente nella memoria, ecc. Per tutte queste funzioni è chiaramente necessario disporre di un quadro di comando manuale, sul quale non possiamo fornire qui maggiori dettagli, e la cui struttura è comunque sempre molto particolare di ogni singola macchina, data la grande arbitrarietà esistente. Il quadro di comando manuale verrà impiegato ad es. per introdurre nella macchina il programma di lettura, senza del quale il nastro perforato non può essere usato: può sembrare che questo debba accadere ogni volta che si mette in funzione la macchina dopo un periodo in cui è stata

"spenta", ma vedremo fra poco che ciò fortunatamente non accade. L'introduzione di dati dal quadro di comando è comunque necessaria in caso di guasti, per mettere in funzione una macchina nuova, ecc.

Per chiarire la situazione quanto alla conservazione di programmi già introdotti nella macchina, è bene ricordare la distinzione fra memorie labili e memorie persistenti. Sono del primo tipo tutti quei dispositivi dotati di memoria che dipendono in modo essenziale per il loro funzionamento come memorie dalla presenza dell'alimentazione elettrica: esempio tipico sono quei classici circuiti bistabili a tubi termoionici che prendono il nome di "flip-flop". Una memoria labile non può chiaramente conservare informazione se la macchina viene "spenta". Memorie persistenti sono invece quelle memorie la cui funzione di memoria rimane inalterata anche in assenza di alimentazione elettrica: si tratta in genere di memorie che sfruttano i fenomeni di magnetizzazione residua delle sostanze ferromagnetiche. Una memoria persistente conserva un programma che ci sia stato scritto anche a macchina spenta. In pratica tutte le macchine moderne sono dotate di qualche tipo di memoria persistente, che si presta quindi a conservare indefinitamente quei programmi fondamentali che, come il programma di lettura, risultano di uso continuo e di cui è molto utile disporre quando si avvia la macchina.

Il secondo aspetto della programmazione di cui ci occuperemo ora riguarda l'uso e il significato dei "sottoprogrammi". Come abbiamo già accennato sopra a proposito del programma di lettura, nell'impiego pratico di una calcolatrice elettronica capita molto spesso di trovarsi nella situazione di costruire un programma di cui alcune par-

ti sono comuni ad altri programmi già preparati, o risultano comunque di uso frequente in programmi anche notevolmente diversi tra loro. Poiché una delle caratteristiche più importanti di una calcolatrice elettronica è la sua automaticità di funzionamento, sarebbe una cattiva utilizzazione della macchina quella in cui non fosse possibile accumulare l'esperienza di programmazione precedente, per servirsene nell'impiego futuro della macchina stessa. Proprio allo scopo di semplificare il lavoro di programmazione, e di sfruttare l'automaticità della macchina, si introducono nell'impiego pratico i sottoprogrammi, cioè dei programmi particolari che per la loro struttura e per la loro funzione si prestano ad essere impiegati, con lievi modifiche, in programmi di carattere qualsiasi.

Un esempio caratteristico di sottoprogramma può essere il calcolo di una funzione trigonometrica: si tratta di un calcolo di una certa complessità, che può avere frequente applicazione, e che quindi non conviene programmare ogni volta che serve; sarà invece opportuno elaborarne il programma in modo che possa essere usato in tutti i casi di applicazione concreta, ma una volta per tutte.

Il problema principale nell'uso di un sottoprogramma sono le modifiche che spesso è necessario apportarvi nelle sue particolari utilizzazioni: la nostra funzione trigonometrica dovrà ad es. venire calcolata con approssimazione variabile da caso a caso, o il numero che rappresenta l'argomento della funzione figurerà in celle diverse della memoria nei vari casi, ecc. L'adattamento di un sottoprogramma al caso specifico può essere più o meno semplice a seconda della macchina, e si può senz'altro asserire che un carattere essenziale di una buona macchina è la fa-

cilità di impiego dei sottoprogrammi. Non essendo possibile entrare in dettagli su questo argomento fondamentale, accenniamo qui solo un aspetto della complessa questione.

Una differenza fondamentale consiste nel modo come sono conservati i sottoprogrammi. Quando la macchina non disponga di una memoria sufficientemente grande, o quando il sottoprogramma non sia di uso frequentissimo, si preferisce conservare il sottoprogramma nella forma di un pezzo di nastro perforato. L'impiego del sottoprogramma in un programma completo potrà allora avvenire attraverso la preparazione di un'unico tratto di nastro, in cui si saldano i vari sottoprogrammi impiegati col "programma principale"; l'adattamento del sottoprogramma al caso particolare potrà allora essere fatto vantaggiosamente all'atto della preparazione del nastro unico. Questo sistema presenta l'inconveniente di essere poco automatico, e di richiedere un forte impegno del sistema di entrata, che essendo di solito piuttosto lento, rallenta a volte sensibilmente la velocità della macchina.

Per i sottoprogrammi di uso molto frequente, e per macchine la cui memoria sia sufficientemente capace, si preferisce conservare i sottoprogrammi stessi nella memoria; in questo caso l'adattamento al programma non può essere fatto per così dire a mano, ma deve essere programmato, come un elemento essenziale del programma. Questo procedimento, apparentemente più complesso, è però suscettibile di essere fortemente automatizzato, con grande economia nel lavoro di programmazione.

Concludendo, queste lezioni dovrebbero avere fornito un'idea dei problemi principali che si presentano nel

la programmazione di una calcolatrice elettronica: non è forse male insistere sul fatto che la preparazione di un programma non è mai un processo lineare e unitario, ma si scinde in vari tempi e in vari livelli, legati strettamente sia alla particolare struttura della macchina in esame, sia alle sue proprietà generali di flessibilità, complessità e automaticità.

--==ooOoo==--