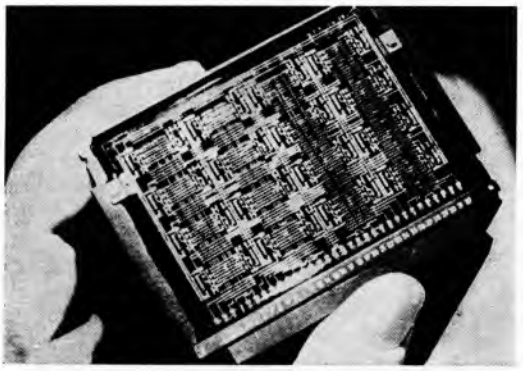
sporije memorije, i to se u računskim jedinicama i svagdje gdje je to još potrebno upotrebljavaju brze memorije, a na drugim mjestima sporije i jeftinije. Tendencija je da se razviju što brže, a ipak jeftinije memorije. Tako su magnetske jezgrice s vremenom pristupa od 2 do 20 mikrosekunda istisnule neke ranije brze memorije. Iz istog razloga dobivaju sve veće značenje magnetski diskovi koji se mogu mijenjati kao gramofonske ploče u džuk-boksima. Takve ploče imaju vrijeme pristupa od 2 do 400 mikrosekundi (srednja vrednost 150 mikrosekundi). Tek odnedavna su u upotrebi tankoslojne magnetske memorije naparene na staklenu ili metalnu foliju, kojima je vrijeme pristupa samo 0,06 mikrosekundi. U najnovije vrijeme se radi i na razvoju dijelova koji rade na principu supravodljivosti a sastavljeni su od kriotrona. Oni služe kao bistabilni multivibratori ili kao ćelije za memoriju. Primjena supravodljivih dijelova u tankoslojnoj tehnici još je u stanju razvoja.

Minijaturizacija dijelova našla je upravo u računskim strojevima svoju najveću primjenu. Štampani krugovi, integrirani sklopovi, planarne i epitaksijalne silicijumske konstrukcije brzo su prihvaćene od konstruktera računala. Istovremenom izradom



Sl. 41. Minijaturni integrirani poluvodički elektronski sklop

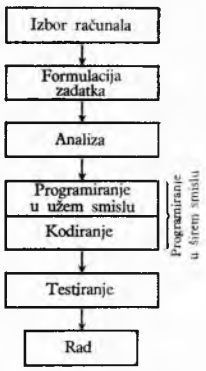
dioda, tranzistora, otpora i kondenzatora na zajedničkoj podlozi nevjerojatno je smanjena veličina, a naročito zapremina pojedinih logičkih i memorijskih organa.

Paralelno s razvojem računala i njegovih organa razvijaju se i ulazne i izlazne jedinice. Pri tom je osnovni cilj brzina rada. Primjenjuju se brži čitači koji rade na čisto optičkim principima a i brži pisači na elektro-optičkim osnovama koji postižu sa specijalnim papirom i brzinu od 3000 do 5000 redaka u minuti. Njihova cijena je zasada još visoka. U razvoju su pisači koji rade s elektrostatičkom fotografijom. S njima se postižu još znatno veće brzine.

LIT.: *W. Meyer zur Capellen,* Mathematische Instrumente, Leipzig 1949. — *D. R. Hartree,* Calculating instruments and machines, Cambridge 1950. — *H. Rutishauser, A. Speiser, E. Stiefel,* Programmgesteuerte digitale Rechengeräte, Basel 1951. — *A. D. Booth, K. H. V. Booth,* Automatic digital calculators, London 1956. — *F. Alt,* Electronic digital computers, New York 1958. — *H. A. Архангельский, В. И. Зайцев,* Автоматические цифровые машины, Москва 1958. — *C. V. L. Smith,* Electronic digital computers, New York 1959. — Handbook of automation, computation and control, vol. 2: Computers and data processing, New York 1959. — *T. C. Bartee,* Digital computers' fundamentals, New York 1960. — *В. А. Батраков, В. И. Богатырев,* Электронные цифровые машины, Москва-Ленинград 1961. — *А. И. Китов, Н. А. Криницкий,* Электронные цифровые машины и программирование, Москва ²1961. — *И. С. Евдокимов, Г. П. Евстигнеев, В. Н. Крушин,* Цифровые вычислительные машины, Москва ³1961. — *В. А. Зимин,* Электронные вычислительные машины, Москва 1962. — *В. И. Лоскутов,* Управляющие математические машины, Москва 1962. — *Л. И. Гутенмахер,* Электронные информационно-логические машины, Москва ²1962 — *M. Greenberger,* Computers and the world of the future, Cambridge, Mass. 1962. — *J. S. Murphy,* Elektronische Ziffernrechner (prevod s engleskog), Berlin 1965. — *A. P. Speiser,* Digitale Rechenanlagen, Berlin-Heidelberg-New York ²1965. — *H. W. Gschwind,* An introduction to the design of digital computers, Wien-New York 1967. — *G. Schubert,* Digitale Kleinrechner, Berlin 1968. — *T. R. H. Sizer,* ed., The digital differential analyzer: an incremental computer, London 1968.

A. P. Železnikar

DIGITALNA RAČUNALA, PROGRAMIRANJE. Programiranje ili sastavljanje programa sastoji se u određivanju niza postupaka prema kojima treba digitalno računalo zadane podatke da pretvori u tražene rezultate. Da bi se dakle neki problem mogao obraditi s pomoću automatskog digitalnog računala, moraju se pripremiti podaci, a pored toga treba za određeni zadatak izraditi program rada računala. U prvoj fazi izrade programa određuje se vrsta i redosljed operacija koje računalo treba obaviti (programiranje u užem smislu), u drugoj se fazi tako sastavljeni program ispisuje u takvom obliku da ga računalo može primiti i prema njemu računati (kodiranje).



Sl. 1. Pregled radnih faza pripreme zadatka

Put od problema do rezultata ne obuhvaća samo programiranje već i širu pripremu zadatka, u koju idu: izbor računala, formuliranje zadatka, analiza, programiranje s kodiranjem, testiranje i računanje (sl. 1). Velik rad uložen u pripremu svakog većeg programa ipak je obično ekonomski opravdan jer se računanje s istim programom ali s različitim zadanim podacima obavlja obično mnogo puta.

PRIPREMA ZADATKA

U istom gradu ili čak u istom računskom centru može programeru biti na raspolaganju nekoliko računala. Još prije nego je neki problem precizno formuliran, pristupa se *izboru računala,* vodeći računa o ekonomičnosti izvođenja zadatka i o brzini dobivanja rezultata. Općenito je jeftinije računanje na bržem računalu, ali na izbor računala utječu i druga svojstva pojedinih računala, npr. veličina glavne memorije, veličina i vrsta vanjskih memorija, brzina davanja rezultata, mogućnost upotrebe viših jezika za programiranje, raspoloživo vrijeme za rad na računalu.

Stručnjaci za problem prelaze zatim na *formuliranje zadatka,* što se vrši obično pomoću matematičkih izraza, jednadžbi i nejednadžbi, ili pomoću jasnih i jednoznačnih rečenica. Kad su zadaci složeniji, može se već prilikom formulacije problema nacrtati tzv. *blok-dijagram,* kojim se zadatak prikazuje rastavljen na nekoliko dijelova. Tako formuliran zadatak predaju stručnjaci za pripremu programerima, tj. stručnjacima koji treba da pripreme zadatak za obradu u računalu. Da prilikom predaje zadatka ne dođe do nesporazuma, mora zadatak biti precizno i detaljno formuliran.

Slijedeći dio pripreme, *analiza zadatka,* obuhvaća ispitivanje broja rješenja i određivanje približnih postupaka. Najprije se ispituje ima li zadatak rješenje, i ako ga ima, postoji li samo jedno rješenje ili ih ima više. Ako prema formulaciji zadatka postoji više rješenja, a traži se samo jedno, treba formulaciju problema upotpuniti. Slično, ako formulirani zadatak nema rješenja, a prema prirodi problema rješenje mora postojati, treba formulaciju ispraviti.

Poslije ispitivanja mogućnosti rješenja analiziraju se pogreške. Kod jednostavnih zadataka takva analiza nije potrebna, ali ako se beskonačni procesi zamjenjuju konačnim, mora se istražiti u kojoj mjeri takva aproksimacija utječe na rezultat. Strogo uzevši gotovo uvijek nastaju takve pogreške, jer se beskonačni razlomci zaokružuju na konačne i jer se brojevi zaokružuju na određen broj decimala. Međutim, u većini tehničkih proračuna ovo se ne odražuje na rezultat.

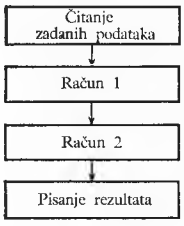
Nakon analize pristupa se *programiranju u užem smislu.* Pri tom se određuje vrsta i slijed operacija. Broj tih postupaka može biti velik, tako da se lako gubi pregled nad njima. Radi lakšeg snalaženja obično se slijed operacija i kriterija prikazuje *dijagramom toka.*

Dijelovi pripreme koji su do sada opisani ne ovise gotovo nikako o svojstvima računala.

Slijedeći dio pripreme, *kodiranje,* postupak je u kome se ispisuju instrukcije na jeziku računala ili na nekom višem jeziku koji računalo može prevesti na svoj jezik. Kodiranje je detaljnije opisano u ovom članku u poglavlju Osnove programiranja i kodiranja. Kodiranje je vezano za svojstva računala.

Po završenom kodiranju, a prije prelaska na računanje, treba program *testirati,* tj. provjeriti u samom računalu. Pri tom se pogreške u kodiranju traže tako da se stroju zadaju ulazni podaci za koje je rezultat unaprijed poznat. Testiranje s traženjem i ispravljanjem pogrešaka često je mučan i dugotrajan posao. Kod zamršenih programa nema vremena da se provjere svi dijelovi i sve mogućnosti u programu. Zato se nakon važnijih testiranja prelazi pažljivo na upotrebu programa za rješavanje novih zadataka

naknadno se kroz neko vrijeme ispravljaju eventualne pogreške koje su u toku rada otkrivene. Nakon tog »uhodavanja« prelazi se na rutinsko iskorištavanje programa.

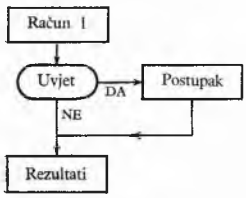


Sl. 2. Blok-dijagram programa s podjelom

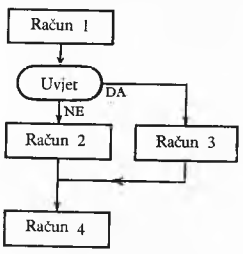
Blok-dijagram. U blok-dijagramu se problem prikazuje shematski, rastavljen u nekoliko dijelova prikazanih »blokovima«. Time se postiže pregled nad čitavim problemom, a svaki dio problema može se razmatrati odvojeno i cijelu pažnju koncentrirati samo na njega.

Jednostavan blok-dijagram prikazan je na sl. 2. U njemu je račun podijeljen na dva dijela.

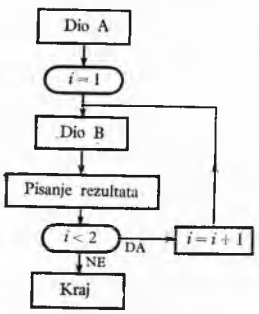
Blok-dijagramom mogu se prikazati i različni uvjeti. Budući da digitalno računalo može prema rezultatima »odlučiti« o tome koji će postupak slijediti, može se programirati da se neki postupak *preskoči* (sl. 3) ili da se bira između dva postupka (*alternativa*, sl. 4). Odluka o izboru postupka ovisi o nekom uvjetu. U dijagramu se takav uvjet stavlja u oval ili u romb (sl. 3 i 5) od koga vode dvije crte označene sa DA i NE, za razliku od ostalih blokova koji su označeni pravokutnicima i od kojih vodi samo jedna crta. Ovakvim



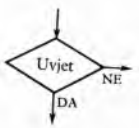
Sl. 3. Preskakivanje postupka ako uvjet nije ispunjen



Sl. 4. Alternativa



Sl. 6. Ponavljanje



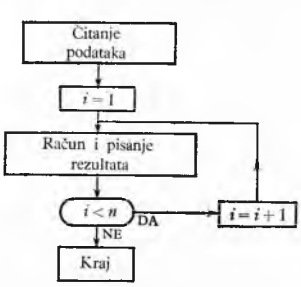
Sl. 5. Drugi način prikazivanja uvjeta

uvjetima može se postići i *ponavljanje* dijelova (sl. 6). Ako treba, na primjer, poslije dijela A dio B i pisanje rezultata jedanput ponoviti, tj. dva puta izvesti, prije prvog izvođenja dijela B stavlja se $i=1$. Varijablom i će se brojati koliko puta se B izvršava tokom računa. Nakon izvođenja dijela B i pisanja rezultata ispituje se je li $i < 2$. Prvi put je ta nejednadžba ispunjena jer je $i = 1$, dakle $i < 2$, pa se i povećava za 1, tj. na $i = 2$, a dio B će se ponovo izvesti.

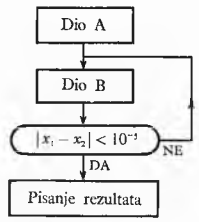
Povećanje varijable i za 1 u dijagramima se može prikazati na različite načine. Pomoću strelice označava se to tako da se ranija vrijednost varijable nalazi s lijeve strane strelice, a nova vrijednost s desne strane: $i + 1 \rightarrow i$. Pomoću znakova = ili := se označuje: $i = i + 1$ ili $i := i + 1$. Ovdje je stara vrijednost od i s desne strane, a nova s lijeve strane.

Kad su rezultati prema sl. 6 ispisani drugi put, uz $i = 2$ uvjet $i<2$ nije ispunjen, pa se prelazi na kraj.

Na sličan se način postiže da se neki dio ponavlja n puta (sl. 7), gdje je n neki broj, zadan kao ulazni ili izračunani podatak. Također se može postići da se neki dio ponavlja neodređeno mnogo puta, dok



Sl. 7. Izvođenje bloka n puta za redom

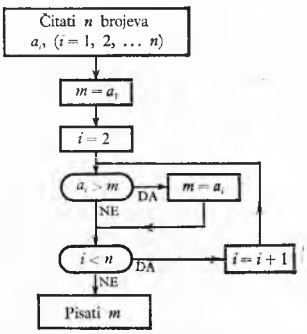


Sl. 8. Ponavljanje bloka dok uvjet nije ispunjen

neki uvjet ne bude ispunjen. Na slici 8 je prikazano ponavljanje dijela B tako dugo dok se x_1 i x_2 ne razlikuju za manje od 0,00001. Kod takvih se postupaka mora paziti da se ponavljanjem računa taj uvjet zadovolji, to jest da se ponavljanje ne protegne u beskonačnost.

Dijagram toka. Kad je problem analiziran, sastavlja se detaljan slijed operacija. Zbog preglednosti se taj slijed prikazuje u dijagramu toka koji se razlikuje od blok-dijagrama samo u tome što je detaljan.

Dijagram toka ćemo prikazati na zadatku: »Ispisati, tj. odabrati i kao rezultat napisati, najveći između n brojeva« (sl. 9). Najprije se čitaju, tj. unesu u računalo, brojevi (njih n), a_i ($i = 1, 2 \ldots n$). Zatim se kao broj za uspoređivanje stavlja $m = a_1$. Ovo se može označiti i sa $m := a_1$ ili $a_1 \rightarrow m$. Nakon toga se za sve ostale brojeve a_i ($i = 2, 3, \ldots, n$) ispituje nije li $a_i > m$. Ako jest, stavlja se $m = a_i$. Ako nije $a_i > m$, nastavlja se s uspoređivanjem sve dok i posljednji a_i, tj. a_n, nije uzet u obzir. Nakon toga se ispisuje m, koji je jednak najvećem od svih a_i.



Sl. 9. Ispisivanje najvećeg od n brojeva

Pošto je sastavljen dijagram toka, može se prema njemu kodirati ne misleći više na sadržaj pojedinih postupaka. Time se postiže da se programer može više koncentrirati na samo kodiranje.

PROGRAMIRANJE I KODIRANJE

Računalo samo po sebi nije kadro izvršiti niz operacija koje su potrebne za rješenje nekog zadatka. Da bi se izveo bilo kakav niz suvislih operacija, treba da postoji poseban program operacija i da taj program bude pohranjen u memoriji računala. Tek nakon toga računalo postaje sposobno za rad. Programom se propisuje računalu slijed djelovanja njegovih izvršnih organa (upravljačke jedinice, akumulatora računske jedinice, jedinicâ za ulaz i izlaz podataka i drugih). Npr., ako treba zbrojiti dva broja, treba programom predvidjeti da se oba broja unesu u memoriju računala, zatim da se prvi od njih stavi u akumulator, da se drugi pribroji sadržaju akumulatora i, konačno, da se sadržaj akumulatora (rezultat) ispiše preko izlazne jedinice. Za rješenje ovakvog jednostavnog računskog zadatka potrebno je u računalu izvršiti pet jednostavnijih programiranih operacija. Pri tome se, međutim, u računskoj jedinici računala izvršava mnogo više elementarnih operacija. Iz ovog se primjera vidi da sastavljanje programa nije jednostavan postupak. Usprkos tome mogu se računalima rješavati i veoma zamršeni zadaci.

Svaki se program sastavlja i piše pomoću slova, brojaka i uobičajenih znakova; pored toga se ispisuje i niz uputa. Zatim treba program pretvoriti (prepisati, odn. kodirati) u takav oblik da ga računalo može pomoću svoje ulazne jedinice *čitati* i registrirati. Ovo se pisanje obavlja npr. s pomoću bušilice kartica ili s pomoću pisača, teleprintera ili perforatora koji buši papirnatu traku. Pojedinim slovima, brojkama i znakovima programa odgovara određen broj na određen način raspoređenih rupica. Računalo s pomoću svoje ulazne jedinice »čita« program s izbušenih kartica ili s izbušene papirnate trake i prenosi ga u memoriju, gdje ga pohranjuje kao niz diskretnih stanja.

Instrukcija je najmanja uputa za rad računala. Ona se sastoji od dva dijela: od operacionog i adresnog dijela. Dužina i značenje tih dijelova zavise od konstrukcije računala. Operacioni dio određuje operaciju koja će se obaviti; adresni dio označava adrese ćelija u kojima se nalaze podaci (*operandi*) s kojima treba izvršiti operaciju.

Operacioni dijelovi instrukcija pišu se na različite načine, npr. s pomoću kratica engleskih riječi: ADD (pribroji), CLA (clear

and add: poništi i dodaj), SUB (subtract: oduzmi), MULT (multiply: pomnoži), DIV (divide: podijeli), STO (store: spremi) itd. Ove kratice mogu biti različite za različna računala, a upotrebljavaju se i brojčani simboli za pojedine operacije.

Instrukcija se dakle isprva piše s pomoću znakova koji su za ljude uobičajeni, ali se preko bušilice ili teleprintera i preko ulazne jedinice računala pretvara u memoriji u niz diskretnih stanja koje poprime pojedini najmanji dijelovi memorije. Tako može npr. magnetska jezgrica memorije biti magnetizirana u jednom ili u drugom smjeru; stanje najmanjeg dijela memorije može se stoga prikazati bitovima 0 ili 1, a instrukcija u računalu kao niz bitova. (Za pojam »bit« vidi str. 315.) U ovom obliku je instrukcija čovjeku nerazumljiva ako nema posebnih uputa o sastavu instrukcija.

U trenutku kada računalo treba izvršiti neku instrukciju, niz bitova koji predstavlja tu instrukciju prenosi se u upravljačku jedinicu. Ovdje se taj niz bitova »dekodira«, i računalo pristupa izvođenju predviđene operacije. To se vrši na taj način da upravljačka jedinica na osnovu primljene instrukcije šalje nizove električkih impulsa na pojedina mjesta u računalu i time aktivira odgovarajuće izvršne organe računala koji onda djeluju onako kako to instrukcija propisuje.

Kao primjer razmotrimo instrukciju »ADD 1026« koja znači: »Dodaj broj spremljen u ćeliji memorije s dekadskom adresom 1026 sadržaju akumulatora«. Ako je u računalu, koje brojeve predstavlja binarno, predviđena duljina instrukcija od 16 bitova, onda će npr. 0001 010000000010 u memoriji prikazivati ADD 1026. Prva 4 bita ovog niza označuju dio ADD (zbroji), tj. operaciju. Ostalih 12 bitova prikazuju binarno napisan broj 1026. Razmak između tih bitova je napisan samo zbog preglednosti. Kad za vrijeme rada računala ovu instrukciju treba izvršiti, upravljačka jedinica je poziva iz memorije, odvaja prva 4 bita i tumači ih kao operaciju »pribroji sadržaju akumulatora«, a ostale bitove interpretira kao adresu ćelije u kojoj se nalazi broj koji treba pribrojiti. Nakon toga upravljački uređaj šalje u pojedine sklopove računala impulse koji su potrebni da se instrukcija izvrši.

Kodiranje na jeziku računala. Pisanje programa na način na koji će se predati ulaznoj jedinici zove se kodiranje. Ako programer piše instrukcije jednu po jednu onim redom kojim će se smjestiti u računalu, kaže se da se program *kodira na jeziku računala*. Ovaj je način kodiranja težak i nepregledan, ali se ipak upotrebljava kad se želi da program zauzme što manji dio memorije ili ako treba da je trajanje računanja što kraće.

Adrese se prilikom kodiranja na jeziku računala mogu pisati u instrukcijama na više načina. O *direktnim* ili *apsolutnim adresama* govorimo kada se u instrukciji prilikom kodiranja upisuje stvarna adresa ćelije memorije, npr. 1026 u instrukciji ADD 1026. U tom slučaju valja prilikom programiranja voditi popis adresa ćelija koje su već zauzete, kako ne bi došlo do zabune. Prilikom *relativnog adresiranja* ne pišu se više direktne adrese, već se upotrebljavaju adrese relativno prema nekoj drugoj određenoj adresi. To znatno olakšava kodiranje i pojednostavnjuje izmjene u programu. Kod *indirektnog adresiranja* adresni dio sadrži adresu ćelije u kojoj se nalazi adresa operanda.

Simbolički skupljački program (engl. symbolic assembly program) pridonio je daljnjem olakšanju kodiranja. Ako je za neko računalo sastavljen takav program, prilikom kodiranja mogu se upotrijebiti simboli umjesto direktnih adresa. Te simbole nazivamo *simboličkim adresama*; npr. umjesto adrese 1026 može se napisati simbolička adresa ALFA. U skupljačkom programu, koji je spremljen u memoriji računala, nalaze se simboličke i pripadajuće direktne (stvarne) adrese. Kad ulazna jedinica prima simbolički kodiran program, računalo će, čim pročita nakon operacionog dijela instrukcije simboličku adresu, npr. ALFA, potražiti u popisu simboličkih adresa skupljačkog programa pripadajuću direktnu adresu, npr. 1026. Zatim će sastaviti operacioni dio s pripadnom direktnom adresom i takvu kompletnu instrukciju spremiti u određenu ćeliju memorije.

Za program s relativnim, indirektnim ili simboličkim adresama također se kaže da je napisan na jeziku računala jer se svaka posebno kodirana instrukcija pojavljuje i kao posebna instrukcija u memoriji računala.

Automatsko programiranje sastoji se u tome da računalo u znatnoj mjeri učestvuje u sastavljanju i stvaranju svog programa; time je kodiranje, dakako, znatno olakšano. U tom se slučaju program ne kodira više na jeziku računala, nego na nekom drugom, tzv. višem jeziku. Da računalo bude u mogućnosti sudjelovati pri automatskom programiranju, treba da u memoriji računala postoji odgovarajući (vrlo zamršeni i obimni) program, koji se isporučuje zajedno s računalom. Program koji se sastavlja za rješavanje pojedinog zadatka piše se pri automatskom programiranju na način koji je pregledan i bliz uobičajenom načinu pisanja, a sadržava, osim formula koje su napisane u određenom algebarskom obliku, različne stilizirane upute koje su obično riječi na engleskom jeziku. Pojedine cjeline tako kodiranog programa za čovjeka su lakše razumljive nego kad su kodirane na jeziku računala. Najmanje cjeline takva programa sada su npr. formule. Računalo prilikom čitanja programa zamjenjuje te formule nizom instrukcija, koje smješta u memoriju. Za razliku od instrukcija na jeziku računala, ove cjeline, koje programer unosi u računalo npr. s pomoću izbušenih kartica ili trake, nazivaju se i pseudoinstrukcijama.

Postoje dvije vrste automatskog programiranja. U prvoj se vrsti pseudoinstrukcije pohranjuju u memoriji računala i za vrijeme rada one se »interpretiraju« s pomoću posebnog programa, tzv. *interpretatora* (engl. interpreter). On sadrži niz dijelova, potprograma, prema kojima se odvija rad računala u ovisnosti o pseudoinstrukcijama. Ova se metoda rjeđe upotrebljava. Druga se vrsta automatskog programiranja služi programom nazvatim *kompilatorom* (kompajler, engl.: compiler). Dok računalo čita program, svaka se pseudoinstrukcija pomoću kompilatora odmah prevodi. Umjesto pseudoinstrukcije nastaje čitav niz internih instrukcija, pa se na taj način sastavlja konačni program u memoriji računala. Prema tome se kod ovog načina automatskog programiranja razlikuju tri vrste programa: a) *kompilator*, program koji je sastavljen za pojedino računalo i stavljen u memoriju na početku rada računala; b) *izvorni program* koji sastavlja programer za pojedini zadatak, a sastoji se od pseudoinstrukcija; c) *konačni, objektivni program* koji se nalazi u memoriji računala nakon prevođenja izvornog programa, a prema kojem računalo radi. Računalo koje rješava neki zadatak prema izvornom programu programera radi zapravo na osnovi objektivnog programa koji je sámo sastavilo uz pomoć svog kompilatora. Kompilatori su vrlo zamršeni i opširni programi, pa neki od njih sadrže mnogo više od 10 000 instrukcija.

Pri automatskom programiranju programer se ne mora više toliko brinuti o oblicima pojedinih instrukcija niti mora voditi računa o akumulatoru, indeks-registru ili adresama, već se može koncentrirati na druge probleme u vezi s programiranjem. Time se vrijeme koje je potrebno za sastavljanje nekog programa bitno skraćuje.

Postoji mnogo (∼ 200) viših jezika za automatsko programiranje. Oni su većinom prilagođeni određenim vrstama zadataka, npr. rješavanju matematičkih, tehničkih, komercijalnih statističkih, prevodilačkih i drugih problema. Među najpoznatije takve više jezike ubrajaju se FORTRAN, ALGOL, COBOL. Pokušaj univerzalnog programskog jezika predstavlja PL/I. Od svih tih jezika najviše se primjenjuju različne varijante FORTRANa.

Program koji je sastavljen na jednom od jezika za automatsko programiranje nije više vezan za jedan tip računala, nego je upotrebljiv, uz eventualne manje izmjene, za svako računalo koje posjeduje prevodilački program za dotični jezik.

Svaki se viši jezik sastoji od potpuno određenih simbola i znakova (GO TO, if, then, SUM, =, + itd.), naziva (imena varijabli ili dijelova programa: ALFA, N1 itd.) i brojeva. Kao što postoje određena pravila govornih jezika, tako se i kod viših jezika za programiranje govori o sintaksi (o pravilnom nizanju simbola, znakova i naziva), o semantici (o interpretaciji upotrijebljenih simbola itd.) i o pragmatici (o svrsishodnosti jezika za određenu namjenu).

Najvažniji programski jezici obrađeni su detaljnije u nastavku ovog članka.

Kodiranje na jeziku računala

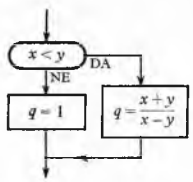
Kao što je naprijed rečeno, o kodiranju na jeziku računala govorimo kad svakoj pojedinoj instrukciji programa odgovara jedna instrukcija u računalu, tj. kad se instrukcije programa ispi-

suju u onom obliku i redoslijedu u kojem će biti kasnije smještene u memoriji računala. Mada se program pohranjuje u memoriji kao niz diskretnih stanja, instrukcije se ipak ne kodiraju izravno s pomoću bitova, jer bi to bilo teško, zamorno i nepregledno, nego se instrukcija ispisuje slovima i brojkama. Ovi se znakovi prilikom upisivanja i čitanja u ulaznim organima automatski pretvaraju u oblik koji odgovara računalu.

Svaka instrukcija programa, kao što je već iznijeto, sastoji se od operacionog i adresnog dijela. Međutim, oblik i dužina same instrukcije znatno ovisi o konstrukciji računala. Dužina operacionog dijela ovisi o broju operacija koje neko određeno računalo može izvesti, a koji npr. iznosi oko 150. Za operacioni dio treba prema tome predvidjeti toliko bitova da se može izraziti naziv operacije. Broj adresa u instrukciji kreće se kod raznih računala između 1 i 3, pa prema tome govorimo o jedno-, dvo- i troadresnim računalima. S obzirom na to da treba i za te adrese predvidjeti dovoljno mjesta, duljina instrukcije današnjih računala iznosi između 16 i 40 bitova.

Adrese se mogu pri kodiranju na jeziku računala izražavati kao direktne, indirektne ili relativne adrese, ili simbolički uz upotrebu skupljačkog programa.

Jednoadresno kodiranje s direktnim adresama bit će objašnjeno na osnovu primjera koji predstavlja dio većeg programa, a prikazan je dijagramom toka na sl. 10. Izračunati treba vrijednost izraza $q = \frac{x+y}{x-y}$, ako je $x < y$. Ako je $x \geqq y$, treba staviti $q = 1$. Kod objašnjavanja pojedinih instrukcija obilježit će se adresni dio sa m, a sadržaj ćelije s adresom m sa $\langle m \rangle$. Kod programiranja se mora uzeti u obzir i akumulator, tj. ćelija koja je vezana s računskom jedinicom, a koja je objašnjena u članku *Digitalna računala*. Akumulator će se označiti sa ak, a njegov sadržaj sa $\langle$ ak $\rangle$. Prema tome je $\langle$ ak $\rangle$ broj koji se nalazi u akumulatoru. Instrukcije koje će se upotrijebiti u ovom primjeru imaju operacione dijelove označene kraticama engleskih riječi kao što je to prikazano u tablici 1.



Sl. 10. Dijagram toka za primjer iz tabl. 3

Tablica 1
KRATICE ZA NEKE NAJVAŽNIJE INSTRUKCIJE PRI KODIRANJU U JEZIKU RAČUNALA

Instrukcija	*Značenje instrukcije*
CLA m	Sadržaj akumulatora postaje jednak sadržaju ćelije m. Sadržaj ćelije m ostaje nepromijenjen. Kraće: $\langle m \rangle \rightarrow$ ak
ADD m	Sadržaju $\langle$ak$\rangle$ akumulatora dodaje se sadržaj m ćelije m. Sadržaj ćelije m ostaje nepromijenjen. Kraće: $\langle ak \rangle + \langle m \rangle \rightarrow ak$
SUB m	Od sadržaja akumulatora odbija se sadržaj ćelije m, a sadržaj ćelije m ostaje nepromijenjen: $\langle ak \rangle - \langle m \rangle \rightarrow ak$
STO m	Sadržaj ćelije m postaje jednak sadržaju akumulatora a sadržaj akumulatora ostaje nepromijenjen: $\langle ak \rangle \rightarrow m$
DIV m	Sadržaj akumulatora se dijeli sadržajem ćelije m, a sadržaj ćelije m ostaje nepromijenjen: $\langle ak \rangle : \langle m \rangle \rightarrow ak$
TRA m	Instrukcija za bezuvjetni skok: računalo prelazi na izvršavanje instrukcije koja se nalazi u ćeliji m.
TMI m	Instrukcija za uvjetni skok: računalo prelazi na izvršenje instrukcije koja se nalazi u ćeliji m samo ako je sadržaj akumulatora manji od nule u času kad je izvršena instrukcija TMI m. Ako sadržaj akumulatora nije negativan, onda računalo prelazi na izvršenje instrukcije koja slijedi iza TMI m.

Ako nije kodiran skok, računalo obavlja operacije redom kako su napisane, dakle prelazi nakon instrukcije iz ćelije s adresom n na izvršenje instrukcije iz ćelije s adresom $n + 1$. Budući da tokom računanja dolazi do skokova, treba voditi popis adresa ćelija u kojima su smještene instrukcije i popis adresa brojeva (tablice 2 i 3).

Tablica 2
POPIS ADRESA
brojeva (varijabla i konstanti) za programiranje u jeziku računala za zadatak $q = \frac{x+y}{x-y}$, ako je $x < y$, $q = 1$ ako nije $x < y$

Adrese brojeva	*Sadržaj ćelije*	*Objašnjenje*
500	x	
501	y	
502	$x - y$	međurezultat
503	1	konstanta
504	q	rezultat

Tablica 3
POPIS INSTRUKCIJA
za rješenje zadatka $q = \frac{x+y}{x-y}$ ako je $x < y$, $q = 1$ ako nije $x < y$

Adresa instrukcije	*Instrukcija*	*Primjedbe*
200	CLA 500	x se prenosi u akumulator: $x \rightarrow$ ak
201	SUB 501	$\langle ak \rangle - y \rightarrow ak$, dakle $ak = x - y$
202	TMI 205	ako je $x < y$, onda je $x - y < 0$, $\langle ak \rangle < 0$, pa se skače na instrukciju u ćeliji 205
203	CLA 503	$1 \rightarrow$ ak
204	TRA 209	skok na instrukciju u ćeliji 209
205	STO 502	$x - y \rightarrow 502$
206	CLA 500	$x \rightarrow$ ak — račun $\frac{x+y}{x-y}$
207	ADD 501	$x + y \rightarrow$ ak
208	DIV 502	$(x + y) : (x - y) \rightarrow$ ak
209	STO 504	sadržaj akumulatora jednak je traženoj veličini q, pa se sadržaj akumulatora premješta u ćeliju 504.

U programu se pišu nule prekrižene radi razlikovanja od slova O.

Prikazani način kodiranja ima prave adrese, pa se kaže da su adrese direktne ili apsolutne. Velika teškoća kod ovog načina kodiranja s direktnim adresama je u tome što se mora voditi popis adresa instrukcija i popis adresa brojeva. Ovo je naročito nezgodno kod ispravaka tokom testiranja kao i kod kasnijih izmjena programa.

Iste se poteškoće javljaju i kod nekih drugih načina kodiranja. Npr., kod *indirektnog adresiranja* adresni dio instrukcije sadržava adresu ćelije u kojoj je smještena adresa operanda. *Relativno adresiranje* (zapravo indiciranje) prema sadržaju specijalne ćelije, tzv. *indeksregistra*, upotrebljava se naročito kod modernih računala. Instrukcije imaju različitu dužinu, a po sadržaju su prilično zamršene. Na primjer dvoadresna instrukcija za aritmetičku operaciju ima u jednom određenom računalu sedam dijelova:

1) operacioni dio (8 bitova),
2) dio koji određuje duljinu prvog operanda u memoriji i
3) dio koji određuje duljinu drugog operanda u memoriji (po 4 bita),
4) redni broj indeksregistra prvog operanda (4 bita),
5) relativnu adresu prvog operanda (12 bitova),
6) redni broj indeksregistra drugog operanda (4 bita),
7) relativnu adresu drugog operanda (12 bitova)

Ukupno ima ta instrukcija dakle 48 bitova.

Skupljači. Kodiranje na jeziku računala može biti vrlo zamršeno. Da se ono olakša, sastavljeni su raznovrsni programi za pojedina računala koji omogućuju da se kodira na način koji je bliži uobičajenom označavanju. Skupljači (engl. assembly program) su razmjerno jednostavni takvi programi; oni dodjeljuju varijablama adrese i određuju adrese simbolički napisanim adresnim dijelovima instrukcija. Tako se omogućuje da programer mjesto direktne adrese napiše simbol adrese, tj. ime, a skupljač će automatski uvesti pravu adresu. Mjesto CLA 500 u prijašnjem primjeru može se napisati CLA (X). Osim adresa ćelija za brojeve treba u tom slučaju simbolički označiti i adrese ćelija za instrukcije, dakako ne za svaku instrukciju. Instrukcija dobiva svoje ime, nazvat ćemo ga *marka* (engl. label, npr. ALFA). Direktno

kodirani primjer tabl. 3 može se simbolički kodirati npr. kako je prikazano u tabl. 4.

Tablica 4
POPIS INSTRUKCIJA
za zadatak iz tabl. 3, sa simboličkim adresnim dijelovima

Instrukcija	*Objašnjenje:*
CLA (X)	$x \rightarrow$ ak
SUB (Y)	$x - y \rightarrow$ ak
TMI (ALFA)	ako je $x < y$, dakle $x - y < 0$, skače se na instrukciju ALFA, dakle se preskaču slijedeće instrukcije;
CLA (BROJ)	smatra se da je prije određeno da je BROJ = 1;
TRA (BETA)	skače se na instrukciju BETA;
ALFA: STO (XMY)	$x - y$ se naziva XMY;
CLA (X)	$x \rightarrow$ ak
ADD (Y)	$x + y \rightarrow$ ak
DIV (XMY)	$(x + y)/(x - y) \rightarrow$ ak
BETA: STO (Q)	varijabla Q postaje jednaka sadržaju akumulatora.

U ovakvom se kodu čovjek lakše snalazi, a različne promjene se mogu uvesti brže, jer se ne mora misliti na stvarne adrese.

Zbog jasnoće prikaza programiranja u primjerima se pretpostavilo da instrukcije i brojevi zauzimaju po jednu ćeliju. Kod mnogih računala to nije tako, nego npr. instrukcije imaju dva puta manju duljinu nego brojevi. U jednu ćeliju mogu stati tada po dvije informacije, a to zahtijeva da instrukcija za skok na prvu instrukciju u ćeliji bude različita od instrukcije za skok na drugu instrukciju u ćeliji. Postoje i računala koja imaju instrukcije različitih duljina, npr. 16, 32 i 48 bitova.

Prikazivanje brojeva

Kod većine računala se brojevi prikazuju na dva bitno različita načina: »u nepomičnom zarezu«, tj. u običnom obliku, ili »u pomičnom zarezu«, tj. u polulogaritamskom ili eksponencijalnom obliku. U nepomičnom zarezu se prikazuju cijeli brojevi, a oni su potrebni npr. kao indeksi za indicirane varijable. U pomičnom se zarezu mogu prikazati i razlomci, a najveći broj koji se može prikazati u pomičnom zarezu znatno je veći od najvećeg broja u nepomičnom zarezu. Slično se i u običnom životu upotrebljavaju dva načina pisanja brojeva: možemo ih pisati sa svim znamenkama, npr. 11 000 000, ili, u analogiji s pomičnim zarezom, s pomoću potencije: $0{,}11 \cdot 10^8$.

Kod *brojeva u nepomičnom zarezu* se većinom smatra da se zarez nalazi iza posljednje desne znamenke, naročito u binarnom prikazivanju brojeva. Ako se npr. brojevi prikazuju binarno sa 16 bitova, onda je 2 u nepomičnom zarezu jednak 0000000000000010.

Brojevi u pomičnom zarezu zapravo su sastavljeni od dva broja: »mantise« m i eksponenta a. Oba broja mogu biti i pozitivna i negativna, a oni zajedno određuju broj $N = m \cdot b^a$, gdje je b baza (dakle $b = 2$ u binarnom, a $b = 10$ u dekadskom sistemu). Uz zadanu bazu b brojevi m i a određuju jednoznačno broj N. Međutim, postoje različiti parovi brojeva m i a koji određuju isti broj, npr. $N = m \cdot b^a = (m \cdot b) \cdot b^{a-1}$ itd. Zato se može propisati pravilo kojim se određuje veličina mantise m i kojim se utvrđuje samo jedan od mnogih mogućih načina prikazivanja nekog broja. Uzima se da mantisa treba da zadovolji nejednadžbe

$$1 > |m| \geqq \frac{1}{b}.$$

Kod binarnog računala ($b = 2$) se prema tome npr. broj 2 u pomičnom zarezu prikazuje kao $0{,}5 \cdot 2^2$, dakle je $m = 0{,}5$; $a = 2$. Neka kod riječi od 16 bitova desnih 5 bitova prikazuje eksponent s predznakom, a lijevih 11 bitova mantisu s predznakom. Kod toga neka znak 1 kao 5. bit zdesna označuje da je eksponent pozitivan, a 0 kao 16. bit zdesna, odnosno kao 1. bit slijeva, označuje da je mantisa pozitivna. Tada se broj 2 tj. $0{,}5 \cdot 2^2$ prikazuje u pomičnom zarezu ovako: 0100000000010010. (Znak 1 kao drugi bit slijeva znači 0,5.)

Kod programiranja se mora voditi računa o tome jesu li pojedini brojevi ili varijable zadani u pomičnom ili nepomičnom zarezu. Npr. kod nekih viših jezika za programiranje se sa 2 označava broj u nepomičnom zarezu, a sa 2. broj u pomičnom zarezu.

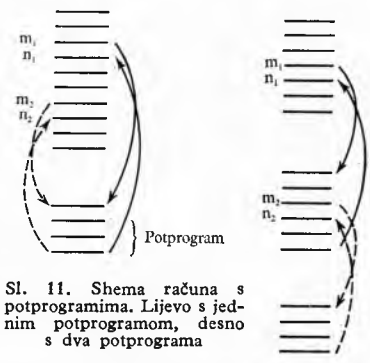
Računanje s brojevima u pomičnom zarezu traje dulje od računanja s brojevima u nepomičnom zarezu jer se operacije obavljaju posebno s eksponentima i posebno s mantisama. Računske operacije u pomičnom zarezu mogu biti posebno programirane ili se postižu odgovarajućim električnim spojevima u računskoj jedinici. U drugom je slučaju trajanje operacije višestruko kraće.

Kod mnogih računala broj bitova, odn. riječi, što ga zauzimaju brojevi različit je za brojeve u pomičnom zarezu i u nepomičnom zarezu.

Potprogrami

Često se u toku programiranja pojavljuje potreba da se isti postupak u programu ponovi više puta. Ako takav postupak sačinjava jednu cjelinu i samostalni dio programa, onda se jednom kodirane instrukcije mogu upotrijebiti i više puta. Takav se dio programa zove potprogram (engl. subroutine, subprogram).

Na sl. 11 je shematski prikazan tok računanja za slučaj da se takav potprogram upotrebljava dva puta. Kad je račun prema potprogramu dovršen, računalo treba da se vrati na instrukciju iza odskoka na potprogram. Instrukcija u ćeliji s adresom m_1 sadrži poseban nalog za skok na potprogram. Kad se ta instrukcija izvršava, potrebno je u posebni registar staviti adresu n_1 ($n_1 = m_1 + 1$) koja omogućuje da se računalo po završenom potprogramu vrati na instrukciju u ćeliji n_1. Slično, kad se poslije instrukcije u ćeliji m_2 skoči na potprogram, stavlja se u isti registar kao prije adresu n_2 ($n_2 = m_2 + 1$).



Sl. 11. Shema računa s potprogramima. Lijevo s jednim potprogramom, desno s dva potprograma

Stvarno je postupak programiranja najčešće nešto zamršeniji, jer treba omogućiti da se iz jednog potprograma pređe na drugi potprogram, pa kad je drugi potprogram završen, završiti prvi potprogram i vratiti se na glavni dio programa. Ovo se omogućuje na razne načine. Potprogrami mogu biti dijelovi programa koji se u njemu na različnim mjestima više puta ponavljaju, a kodiraju se baš za taj program. Time se pri programiranju štedi na pisanju, a za isti je rad i smještaj programa potreban manji broj ćelija. Nešto je drugačiji slučaj ako se nađe već gotov potprogram iz nekog drugog programa. Tada se takav potprogram može jednostavno prepisati u novi program. Potprogrami se najčešće upotrebljavaju za različne jednostavne česte operacije i postupke. Tako se na primjer kod mnogih računala aritmetičke operacije u pomičnom zarezu ne izvršavaju s pomoću električkih spojeva i ne obavljaju s pomoću jedne jedine instrukcije u memoriji, nego postoje za te operacije potprogrami. To vrijedi u prvom redu za zbroj, razliku, produkt, kvocijent i drugi korijen. Često se uz računalo isporučuju i gotovi standardni programi u kojima su sadržani i potprogrami za najčešće funkcije, npr. za sinus, tangens, arkus sinus, arkus tangens, za eksponencijalnu i logaritamsku funkciju, za apsolutnu vrijednost i druge.

Automatsko programiranje i jezici za programiranje

Automatsko programiranje je postupak u kome računalo sámo automatski učestvuje u sastavljanju svog internog programa. Simboličko programiranje koje je ranije prikazano oslanja se na instrukcije; svakoj pojedinoj napisanoj instrukciji odgovara jedna instrukcija u računalu. Slijedeći viši stupanj programiranja omogućuje da se pišu jednostavne eksplicitne formule jednom operacijom, npr.:

$$A = B + C.$$

I ovakvo kodiranje zove se simboličko. Na jeziku računala bi

se ta operacija kodirala ovako:

```
CLA 5ØØ
ADD 5Ø1
STO 5Ø2
```

ako su za A, B i C rezervirane ćelije s adresama 502, 500 i 501. Uspoređujući oba načina kodiranja čini se kao da je uputstvo za isti postupak napisano na dva različita »jezika«. Prema tome, ako računalo čita $A = B + C$, onda ono treba da taj izraz »prevede« s jezika programa na jezik računala, tj. u više internih instrukcija. Za to prevođenje potrebno je da računalo ima u memoriji odgovarajući program koji se naziva *kompilator* (engl. compiler) ili *prevodilac* (engl. translator). Pomoću njega je omogućeno automatsko programiranje, to jest automatsko prevođenje izraza koji su ako lshvatljivi čovjeku u niz odgovarajućih instrukcija. Pri prevođenju računalo čita pojedine male cjeline kodiranog programa. Takve su cjeline, npr., jedna kartica ili jedan red. S pomoću kompilatora računalo stvara instrukcije s odgovarajućim adresama koje prikazuju tu cjelinu programa na jeziku računala. Kad su se te instrukcije smjestile u memoriju, računalo čita slijedeću cjelinu.

Racionalno programiranje na različnim jezicima za automatsko programiranje omogućeno je velikom brzinom rada modernih računala, tako da prevođenje nekog programa (kompiliranje) na interni jezik traje obično samo nekoliko minuta.

Jezici za automatsko programiranje. Kod viših jezika za programiranje treba definirati veći broj pravila za izraze koji su dopušteni (sintaksa). Nabrojat ćemo u nastavku neka od njih.

Označivanje varijabli. Varijable imaju imena koja se sastoje od jednog slova ili od više slova i brojeva, ali tako da je prvi znak slovo. Upotrebljava se 26 slova engleske abecede. Razlikuju se cjelobrojne varijable i realne varijable. Varijable se mogu indicirati; indeksi su cijeli brojevi ili cjelobrojne varijable, a pišu se u okruglim ili uglatim zagradama, npr. ALFA (2) označuje a_2. Ako imena varijabli mogu sadržavati samo jedno slovo, indeks se može pisati i bez zagrade; npr. AI označuje varijablu a_i.

Označivanje operacija. Aritmetičke se operacije obično označuju na engleski način: znak + znači zbrajanje, znak — odbijanje, znak × množenje i znak / dijeljenje. Mjesto znaka × se većinom upotrebljava * da ne bi došlo do zamjene sa slovom x. Brojevi se obično označavaju na engleski način decimalnom tačkom, npr. 3.14159, a ne zarezom.

Standardne funkcije. Kompajleri sadržavaju potprograme za sve češće funkcije: za drugi korijen, sinus, kosinus, arkus tangens, za eksponencijalnu i logaritamsku funkciju, za odvajanje cijelog dijela nekog broja i sl.

Pridjeljivanje vrijednosti varijabli. Varijabla kojoj se pridjeljuje vrijednost izražena je eksplicitno s lijeve strane znaka = ili znaka :=, npr. A = B + C ili A := B + C;. Završetak takva izraza se označuje prelazom na nov redak ili posebnim znakom (npr. sa ;).

Skokovi. Najvažnije su tri vrste skokova: bezuvjetni skok, uvjetni skok i skok na potprogram. Bezuvjetni skokovi odgovaraju instrukciji TRA u jeziku računala (v. tablicu 1). Izraz na koji se prelazi označava se markom tog izraza. Marka može biti broj ili ime koje je slično građeno kao ime varijable, a stavlja se ispred izraza koji se označuje.

Uvjetni skokovi su upravljani pomoću iznosa neke varijable, analogno instrukciji TMI u jeziku računala (v. tablicu).

Skok na potprogram mora omogućiti vraćanje na glavni dio programa.

Potprogrami. Kod jednostavnijih jezika varijable koje su označene istim imenom unutar potprograma i izvan njega uvijek imaju isto značenje: one su identične. Kod viših jezika varijabla unutar potprograma može biti uvedena tako da je različita od varijable izvan potprograma koja je označena istim imenom. Time je omogućeno da se prije sastavljen potprogram može upotrijebiti u nekom novom programu bez ikakve izmjene.

Petlje predstavljaju ponavljanje pojedinih manjih dijelova programa više puta uzastopce. »Petlje« za ponavljanje koje se češće javljaju jednostavno se programiraju.

Iz ovog pregleda je jasno da je sastavljanje kompilatora golem posao. Kompilator za pojedini jezik ima obično mnogo tisuća instrukcija, a za njegovo sastavljanje potrebno je više jedinica rada godina-čovjek.

Jezici koji su blizi uobičajenom načinu pisanja formula i drugih izraza uveliko ubrzavaju programiranje. Napisani su programi vrlo pregledni, a eventualne izmjene se lako unose. Treba ipak napomenuti da je automatsko prevođenje šablonsko, pa stoga program zauzima veći dio memorije nego što bi je zauzeo da je kodiran na jeziku računala običnim instrukcijama.

U slijedećim poglavljima su opisani najznačajniji jezici za programiranje: ALGOL, FORTRAN i COBOL, kao i univerzalni jezik za programiranje PL/I.

Jezik za programiranje: ALGOL. Algol je kovanica, sastavljena od dijelova engl. riječi ALGOrithmic Language, koja služi kao naziv međunarodno prihvaćenog jezika za programiranje. Jezik Algol sastavila je (1960) i revidirala (1962) komisija stručnjaka. Taj je jezik namijenjen programiranju zadataka u kojima se pojavljuje mnogo formula i logičkih odluka. Postoji niz pravila tog jezika koja propisuju kako treba sastavljati i pisati pojedine probleme u jasnom i standardnom obliku. Na osnovu tih definicija i pravila sastavljaju se i kompilatori za pojedina računala. Kod toga se odustaje obično samo od malog broja pravila koja su predviđena u ALGOLu iz g. 1960 (kratica ALGOL 60).

U ALGOLu se mogu upotrebljavati velika i mala slova, ali mnogi kompilatori dopuštaju samo primjenu velikih slova. Broj znakova za imena varijabli je ograničen propisima za pojedini kompilator. Npr. izraz

$$\sin\left(y_1 \cdot \frac{a+B}{\xi}\right) - \pi r$$

u ALGOLu se može napisati ovako:

```
sin (yl * (a + B)/KSI) — 3.14159 * r
```

Pridjeljivanje vrijednosti nekoj varijabli piše se prema primjeru A := B + C; time se propisuje da varijabla A dobiva vrijednost koja je jednaka zbroju vrijednosti varijabli B i C. Razmaci ili novi reci ne prekidaju cjelinu.

Da se omogući upotreba neke varijable, ona se mora »deklarirati«. Varijable bez indeksa se deklariraju tako da se navedu iza simbola real, integer ili Boolean. Iza real se uvode realne varijable u pomičnom zarezu, iza integer cijele varijable u nepomičnom zarezu, a iza Boolean Booleove varijable koje mogu imati samo vrijednosti true (istina) ili false (neistina). U ovom uvodu nećemo više navoditi Booleove varijable (v. str. 323).

Čitav program počinje simbolom begin, a završava simbolom end. Ove i slične napisane riječi predstavljaju za kompilator posebne simbole, kao znak + ili), pa se stoga pišu na poseban način, npr. podcrtano ili među apostrofima, već kako to propisuju pojedini kompilatori ALGOLa.

Preporučuje se da se čitanje podataka u pomičnom zarezu kodira sa inreal (n, v), gdje n znači redni broj ulazne jedinice, a v ime varijable. Slično se s integer (2, j) kodira čitanje cijelog broja na ulaznoj jedinici 2 i pridjeljivanje tog broja cjelobrojnoj varijabli j. Ispisivanje vrijednosti na izlaznoj jedinici 4 se analogno kodira sa outreal (4, ksi) odnosno outinteger (4, j).

Tablica 5
PROGRAM ZA IZRAČUNAVANJE ZADATKA $y = a x^3 + b x + c$ U ALGOLU
(*Primjer br. 1*)

Program	*Objašnjenje*
begin real a, b, c, x, y;	Početak programa. Deklariraju se varijable a, b, c, x, y u pomičnom zarezu. Time se rezerviraju određene ćelije za te varijable.
inreal (1, a), inreal (1, b), inreal (1, c); inreal (1, x);	Četiri se broja čitaju i stavljaju redom u ćelije rezervirane za a, b, c, x. Brojevi se čitaju na ulaznoj jedinici br. 1, npr. na čitaču kartica.
y := (a * x +b) * x + c;	Mjesto znaka × se ovdje upotrijebila zvjezdica * . Ovim se izrazom izračunava $(a x + b) x + c$, a dobivena se vrijednost stavlja u ćeliju koja je rezervirana za varijablu y.
outreal (4, y)	Iznos ćelije koja je rezervirana za varijablu y se ispisuje na izlaznoj jedinici br. 4, npr. na štampaču.
end	Kraj programa.

U primjeru br. 1 (tabl. 5) prikazan je program za izračunavanje zadatka: $y = a\,x^2 + b\,x + c$, ako su zadane vrijednosti: a, b, c i x.

Varijable s indeksima deklariraju se iza simbola real array ili integer array tako da se u uglatim zagradama navede najmanja i najveća vrijednost indeksa, npr.

real array v [1 : 3], a [1 : n, 1 : 3];

Time su uvedene varijable koje se obično pišu v_i, ($i = 1, 2, 3$ i $a_{r,s}$, ($r = 1, 2, \ldots, n$, $s = 1, 2, 3$) a za odgovarajuće se podatke rezervira mjesto u memoriji.

Ako neki dio programa treba predstavljati manju cjelinu, stavlja se između simbola begin i end. Takav se dio zove blok, ako sadrži neku deklaraciju. Primjer programa br. 2 (tabl. 6) sadržava dva manja bloka.

Tablica 6
PROGRAM U ALGOLU KOJI SADRŽAVA DVA MANJA BLOKA
(*Primjer br. 2*)

Program	*Objašnjenja*
begin real array a [1 : 40];	Početak programa. Deklaracijom se rezerviraju ćelije za 40 indiciranih varijabli u pomičnom zarezu: a_1, a_2, a_3, ... a_{40}.
. . .	
begin real array b [1 : 40];	Početak prvog manjeg bloka. U njemu se deklariraju indicirane varijable b_1, b_2 ... b_{40}.
b [1] := a [1] + 1;	Varijabla b_1 postaje jednaka $a_1 + 1$.
. . .	Ako je neki blok obuhvaćen većim, onda se u manjem može upotrijebiti varijabla koja je deklarirana u većem bloku.
end;	Kraj prvog manjeg bloka.
begin real array b1 [1 : 20], b2 [1 : 20];	Početak drugog manjeg bloka. U njemu se mogu pojaviti varijable a_i, $b1_i$, $b2_i$, ali se ne smije pojaviti b_i.
. . .	
end;	Kraj drugog manjeg bloka.
end	Kraj programa.

Memorija za b [1 : 40] se zauzima samo za vrijeme računanja u prvom manjem bloku. Nakon napuštanja tog bloka postaje odgovarajući dio memorije slobodan.

Svaki se program može shvatiti kao blok u kome su deklarirane sve varijable koje se u njemu pojavljuju.

Varijable s indeksima se čitaju s inarray, a ispisuju s outarray. Računanje s indiciranim varijablama se često kodira pomoću petlje, npr.:

```
for i := 1 step 1 until 3 do
        begin
          .
          .
          .
        end;
```

Ovdje se za $i = 1$, $i = 2$, $i = 3$ ponavljaju operacije u manjem dijelu programa koji je sadržan između begin i end.

Primjer br. 3 pokazuje program za nalaženje najvećeg od n brojeva prema dijagramu toka na sl. 9 (tabl. 7).

Prikazat ćemo nadalje shematski *kodiranje alternative* prema sl. 10. Posebno pisanje uvjetnih skokova je izbjegnuto preglednim simbolima < (manje od), then (onda) i else (inače):

```
if x < y then q := (x + y)/(x − y)
        else q := 1;
```

Slično se kodira alternativa u primjeru na sl. 8 uz primjenu marke »b« na početku dijela B:

```
          .
          .
          .
        end;                    } dio A
b: begin
          .
          .
          .
        end;                    } dio B
if abs (x1 − x2) ≧ 0.00001 then go to b
        else begin
          .
          .
          .
        end;                    } pisanje rezultata
```

U ovom se primjeru pojavljuje standardna funkcija abs, koja je jednaka apsolutnoj vrijednosti argumenta: abs (x) ≡ |x|. Simbol go to je oznaka za bezuvjetan skok.

Potprogrami u ALGOLu sadržavaju varijable koje imaju posebno značenje, neovisno o eventualno jednako označenim varijablama izvan potprograma. Veza između potprograma i ostalog dijela programa se postiže preko tzv. formalnih parametara, a to su varijable potprograma koje su posebno navedene u zagradama iza imena potprograma. Potprogram se primjenjuje tako da se navede ime, a formalni se parametri zamjenjuju varijablama koje su deklarirane izvan potprograma ili izrazima koji sadrže te varijable. Kaže se da se iza imena potprograma mjesto formalnih parametara pišu »aktualni parametri«.

Jezik za programiranje: FORTRAN. Najrasprostranjeniji su, što se tiče upotrebe, jezici koji nose zajednički naziv FOR-

Tablica 7
PROGRAM U ALGOLU ZA NALAŽENJE NAJVEĆEG OD n BROJEVA
(*Primjer br. 3*)

Program	*Objašnjenje*
begin comment nalaženje najvećeg od n brojeva;	Početak programa. Tekst iza comment se ne prevodi i ne stavlja u memoriju.
integer n, i;	Cjelobrojne varijable n, i se deklariraju, pa se za njih rezerviraju ćelije u memoriji.
real m;	Deklarira se varijabla m u pomičnom zarezu.
real array a [1 : 1000];	Deklariraju se indicirane varijable a_i, ($i = 1, 2, \ldots, 1000$).
ininteger (1, n);	Na ulaznoj jedinici za čitanje br. 1 (npr. čitačem kartica) čita se cijeli broj n.
for i := 1 step 1 until n do inreal (1, a [i]);	Čitaju se brojevi a_1, a_2, ..., a_n. To je programirano pomoću petlje: čita se nov broj a_i tako dugo dok je $i \leqq n$. Nakon svakog čitanja se i povećava za 1, pa se ispituje je li $i > n$. Ako nije, računalo ponavlja postupak čitanja. Ako je $i > n$, petlja se napušta.
m := a [1];	Najveći broj je možda jednak a_1. Ako nije, broj m će se povećati u petlji koja slijedi.
for i := 2 step 1 until n do if a [i] m then m := a [i];	Za $i = 2, 3, \ldots, n$ se ispituje je li $m < a_i$. Ako jest, m se povećava tako da bude $m < a_i$. Ako nije $m < a_i$, onda m ostaje nepromijenjen.
outreal 2, m	Izlazna jedinica br. 2 ispisuje broj m.
end	Kraj pograma.

TRAN (kratica od FORmula TRANslation). Različni tipovi strojeva imaju različite gotove kompilatore za programiranje u FORTRANu. Od različitih varijanti najčešći su FORTRAN II i FORTRAN IV.

Između programa koji su napisani za istu operaciju u ALGOLu i FORTRANu postoje ove razlike: program u FORTRANu se ne dijeli u blokove; marke u FORTRANu ne mogu biti varijable, nego samo brojevi; varijable bez indeksa se u FORTRANu obično ne deklariraju posebno; varijable s indeksom se u mnogim varijantama FORTRANa mogu deklarirati samo s unaprijed brojčano zadanim granicama indeksa; mjesto znaka := upotrebljava se u FORTRANu znak = za pridjeljivanje vrijednosti nekoj varijabli.

Tablica 8
PROGRAM U FORTRANU ZA NALAŽENJE NAJVEĆEG OD n BROJEVA
(*Primjer br. 4*)

Program
C NALAZENJE NAJVECEG OD N REALNIH BROJEVA DIMENSION A (1000) READ 1, N 1 FORMAT (14) DO 2 I = 1.N 2 READ 3, A (I) 3 FORMAT (E14.4) EM = A (1) DO 4 I = 2, N IF (A(I)−EM) 4,4, 6 EM = A (I) 4 CONTINUE PUNCH 3, EM STOP END

Prvi red počinje s oznakom C, što je kratica za comment (komentar). Čitav taj red ima značenje obavijesti o napisanom programu, te se ne prevodi. U drugom retku se rezervira mjesto za 1000 indiciranih varijabli A(1), A(2), ..., A(1000). Ove mogu biti npr. elementi neke matrice ili mnogo mjernih podataka iste fizikalne veličine.

Treći i četvrti red daju uputu za čitanje jedne cjelobrojne varijable N, na primjer s bušene kartice. Redak: READ 1, N znači da treba čitati varijablu N prema obliku koji je dan u retku s markom 1, tj. u retku: 1 FORMAT (I 4). Općenito FORMAT daje uputu za čitanje i ispisivanje ili izdavanje. Oznaka I4 znači da se neka instrukcija odnosi na cjelobrojnu varijablu (I je kratica od »integer«, cijeli broj) koja je napisana s najviše četiri znaka. Prema tome se može ispravno čitati broj 1000, ali ne i broj 10000, ni broj −1000, jer imaju 5 znakova.

Slijedi petlja za učitavanje n brojeva. Petlja završava u retku s markom 2, jer je iza DO napisan 2. U petlji se varira indeks I od minimalne vrijednosti 1 do maksimalne vrijednosti N. Ako korak (prirast varijable) nije napisan, podrazumijeva se da je jednak 1 kao što je u gornjem primjeru. Ako korak nije jedan, treba ga navesti na kraju, pa npr. DO 50 I = 1, N, 2 označuje da I prima vrijednosti 1,3,5 itd. do N (ako je N neparan) ili do $N-1$ (ako je N paran). Prema tome DO 2 I = 1, N odgovara u ALGOLU duljoj uputi: for I := 1 step 1 until N do. S druge strane, u ALGOLu je prišteđena marka tako da se dio koji se ponavlja stavlja između begin i end.

Brojevi a_i se učitavaju prema uputi s markom 3: FORMAT (E14.4). To znači da su brojevi napisani s najviše 14 znakova u »eksponencijalnom« obliku, na primjer sa 10 znakova:

$$-24.58E+11$$

Ovaj broj ima značenje $-24{,}58\cdot 10^{11}$. Kad u dijelu −2458 ispred E ne bi bilo decimalne tačke, onda bi prema uputi FORMAT računalo automatski odabralo 4 decimalna mjesta. Navedeni broj bi se dakle mogao napisati i ovako:

$$-2458E+13$$

što bi značilo $-0{,}2458\cdot 10^{13}$.

Nakon čitanja stavlja se $m = a_1$, ili u programu napisano EM = A(I). Mjesto m se ne može bez daljnjega upotrijebiti M, jer M znači cjelobrojnu varijablu. Zato se dodaje ispred M npr. E, da se dobije realna varijabla.

Slijedi petlja koja završava na retku: 4 CONTINUE, jer iza DO slijedi broj 4. Ovaj redak ne označuje nikakvu računsku operaciju, ali je potreban zato što se unutar petlje u slučaju $a_i \leq m$ mora prijeći na kraj petlje.

Unutar petlje se upotrebljava uvjetni skok IF (A(I)−EM) 4,4,6. Skače se na red s markom 4 ako je izraz u zagradi negativan ili jednak nuli, a na red s markom 6 ako je taj izraz pozitivan. Kad su svi brojevi a_i uspoređeni, varijabla EM ima vrijednost najvećeg broja a_i. Taj broj se npr. buši na karticu: PUNCH, i to prema uputi iz retka s markom 3. Prema tome bit će izbušen broj sa 4 decimalna mjesta u »eksponencijalnom« obliku, npr.:

$$-0.2458E+13$$

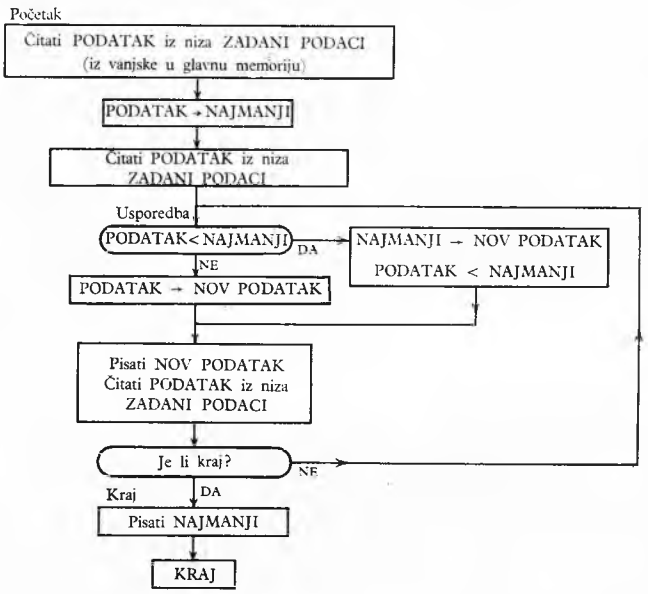
Među različnim vrstama programiranja u FORTRANu postoje velike razlike. Kod nekih vrsta nisu mogući izrazi koji sadržavaju i realne varijable u pomičnom zarezu i cijele varijable u nepomičnom zarezu, dok FORTRAN IV dopušta računanje s kompleksnim brojevima čak u izrazima s realnim i cijelim brojevima. Implicitno se podrazumijeva da varijable kojima imena počinju slovima I, J, K, L, M, N prikazuju cijele brojeve, a ostale varijable prikazuju realne brojeve. Kod nekih vrsta FORTRANA se ta podjela može promijeniti prema volji programera. U jeziku FORTRAN II nije moguća upotreba Booleovih varijabli i izraza, dok FORTRAN IV dopušta njihovu upotrebu.

Programiranje u FORTRANu prikazano je na primjeru br. 4, prema dijagramu toka na sl. 9, u tabl. 8; objašnjenje pojedinih dijelova programa dano je u tekstu uz tablicu.

Potprogrami su mogući u FORTRANu slično kao i u ALGOLu, osim u najjednostavnijim varijantama FORTRANa.

Jezik za programiranje: COBOL. Jezik COBOL (COmmon Business Oriented Language) je sastavljen za programiranje zadataka u kojima se pojavljuje obrada velikog broja podataka s jednostavnim operacijama, kao što su npr. poslovne obrade podataka. S gledišta programiranja ne pretpostavlja se gotovo nikakvo matematičko predznanje. Zbog svega toga se COBO mnogo razlikuje od ALGOLa i FORTRANa.

Program napisan u jeziku COBOL ima četiri dijela: dio identifikacije (identification division), dio okoline (environment division), dio podataka (data division), dio postupka (procedure division). Dio identifikacije sadržava bitne podatke o programu, a pri prevođenju programa na interni jezik računala nema djelovanja na program. Prema tome se taj dio samo nalazi napisan na karticama, papirnotoj traci ili na protokolu (tj. na papiru na kojem se ispisuju sadržaji kartica ili papirnate trake). Ovaj dio



Sl. 12. Dijagram toka za primjer br. 5

služi za prepoznavanje programa, pa sadrži ime programa, programera i računala, datum i slično. Dio okoline sadržava opis potrebnih vanjskih memorija, odnosno ulazno-izlaznih organa, i pridjeljuje pojedine skupine podataka tim memorijama. Dio

Tablica 9
PROGRAM U COBOLU
za rješenje zadatka prema sl. 12
(*Primjer br. 5*)

```
PROCEDURE DIVISION.
POCETAK.
    OPEN INPUT ZADANI-PODACI, OUTPUT DOBIVENI-PODACI,
    NAJMANJI-PODATAK.
    READ ZADANI-PODACI RECORD; AT END GO TO KRAJ.
    MOVE PODATAK TO NAJMANJI. READ ZADANI-PODACI
    RECORD; AT END GO TO KRAJ.
USPOREDBA.
    IF PODATAK IS LESS THAN NAJMANJI THEN GO TO IZMJE
    NA. MOVE PODATAK TO NOV-PODATAK. GO TO PRIJENOS.
IZMJENA.
    MOVE NAJMANJI TO NOV-PODATAK. MOVE PODATAK TO
    NAJMANJI.
PRIJENOS.
    WRITE NOV-PODATAK. READ ZADANI-PODACI; AT END
    GO TO KRAJ.
    GO TO USPOREDBA.
KRAJ.
    WRITE NAJMANJI. CLOSE ZADANI-PODACI, NAJMANJI
    -PODATAK, DOBIVENI-PODACI.
```

podataka opisuje detaljno pojedine nizove podataka. Dio postupka sadržava opis djelovanja računala.

Kao primjer programa u jeziku COBOL dan je u tabl. 9 dio postupka iz programa za rješenje ovog zadatka: iz jednostavnog niza različitih brojeva ispisati najmanji i sastaviti nov niz brojeva koji ne sadrži najmanji. Dijagram toka za taj zadatak prikazan je na sl. 12. Iz tog se primjera vidi da se, uz poznavanje engleskog jezika, smisao tog programa razmjerno lako shvaća. Međutim, pisanje programa je teže, jer treba strogo paziti da se upotrebljavaju samo riječi koje su propisane, da se stave propisani znakovi itd.

Jezik za programiranje: PL/I. Programski jezici ALGOL i FORTRAN su sastavljeni tako da budu prikladni za programiranje naučno-tehničkih zadataka, dakle zadataka u kojima se pojavljuje mnogo računanja s razmjerno malo podataka. Jezik COBOL je sastavljen za programiranje zadataka u kojima se pojavljuje mnogo podataka, a računanja ima malo. Takvi su npr. komercijalni zadaci.

U novije vrijeme se razlika između tih dviju vrsta zadataka smanjila: pojavljuju se naučno-tehnički zadaci s mnogo podataka i komercijalni zadaci s mnogo računanja. To je uzrok da je sastavljen jezik PL/I (skraćenica od engl. Programming Language/One) koji ima univerzalno područje primjene. Iako je PL/I jedinstven jezik, ne mora pojedini korisnik poznavati čitav jezik, nego samo jedan njegov odgovarajući dio, a da zbog toga ne načini pogrešku u programiranju.

Jezik PL/I sadrži sve prednosti FORTRANa, ALGOLa i COBOLa. Kod običnih formula PL/I liči na FORTRAN:

$$A = A + B$$

znači da se vrijednost varijable A povećava za B.

Uvjetni skokovi se u PL/I pišu slično kao u ALGOLu:

IF X < O THEN Y = O;
ELSE Y = X + 1;

znači da Y dobiva vrijednost 0 ako je $X < 0$. Ako je $X \geqq 0$, Y postaje jednak $X + 1$.

Indicirane se varijable označuju tako da se indeks navede u okruglim zagradama. Za razliku od FORTRANa, kod operacija s indiciranim varijablama često se može skratiti pisanje. Ako su npr. A i B imena matrica sa 10 redaka i 10 stupaca pa te dvije matrice treba zbrojiti tako da se dobije matrica C, piše se jednostavno

DCL (A, B, C) (10, 10);
C = A + B;

Ovdje je DCL kratica za DECLARE, a označuje dogovor o podacima. U primjeru se deklariraju matrice A, B, C sa 10 redaka i 10 stupaca, dakle se rezerviraju ćelije za 3 odgovarajuća polja. Zbrajanje elemenata se ponavlja za sve vrijednosti indeksa. Ako samo jedan indeks treba proći sve vrijednosti, on se označuje zvjezdicom. Ako npr. matrice A, B i C imaju najveću vrijednost drugog indeksa 10, piše se na PL/I:

$$C(I, *) = A(J, *) + B(K, *),$$

što znači:

$$c_{in} = a_{jn} + b_{kn}, \ (n = 1,2, \ldots, 10).$$

KARAKTERISTIČNI PROBLEMI RJEŠAVANI DIGITALNIM RAČUNALIMA

Od matematički formuliranih zadataka koji se rješavaju digitalnim računalima najvažniji su ovi: *aproksimacija realnih funkcija jedne realne varijable* (osim poznatih metoda aproksimacije parcijalnom sumom MacLaurinova ili Taylorova reda upotrebljavaju se približne formule koje s manje računskih operacija daju rezultate iste tačnosti, kao npr. Padéova aproksimacija razlomljenom racionalnom funkcijom), *rješavanje sistema linearnih jednadžbi* (obično Gaussovom metodom eliminacije), *inverzija matrica* (može se svesti na rješavanje sistema linearnih jednadžbi), *numerička integracija* (Simpsonovom ili sličnom formulom, ili metodom Monte Carlo), *rješavanje običnih diferencijalnih jednadžbi* uz zadane početne uvjete (različitim metodama se traže vrijednosti funkcije u diobenim tačkama intervala nezavisne varijable razdijeljenog u jednake dijelove; metoda Runge-Kutta ima prednost da je jedinstvena za čitav interval nezavisne varijable; nove metode brzo izračunavaju tražene funkcije), *rješavanje linearnih parcijalnih diferencijalnih jednadžbi* (obično zamjenom derivacija kvocijentima diferencija), *rješavanje polinomijalnih jednadžbi višeg stupnja* (iterativnim metodama), *rješavanje problema iz statistike, Fourierova analiza i sinteza, linearno programiranje* (traženje ekstrema linearne funkcije uz uvjet da je ispunjen sistem linearnih nejednadžbi).

Za rješavanje problema u proizvodnji postoje osim linearnog programiranja i druge metode. Na primjer metoda PERT (engl. Program Evaluating and Review Technique) služi za određivanje roka dovršenja proizvoda koji su sastavljeni od mnogo dijelova. Potrebni podaci o proizvodu treba da stanu u glavnu memoriju. Ako proizvodi imaju vrlo mnogo sastavnih dijelova, javlja se pitanje najvećeg broja dijelova koji se tom metodom može obraditi na nekom određenom računalu.

Od ostalih programa u vezi s proizvodnjom spominjemo upravljanje (kontrolu) skladišta, provjeru (kontrolu) kvaliteta, upravljanje proizvodnjom u nekom poduzeću (naročito velik zadatak koji zahtijeva veliku i brzu vanjsku memoriju) itd.

Sasvim drugog tipa su *upravljački programi* koji upotpunjuju rad samog računala tako da se oni gotovo mogu smatrati dijelom računala. Takvi »supervajzorski« programi (engl.: supervisory program) upravljaju radom računala i dijelova koji su s računalom u vezi, dakle upravljaju slijedom zadataka koji se izvršavaju bez prekidanja rada. Pomoću supervajzorskog programa započinju se ulazne i izlazne operacije, uspostavlja se veza između operatora i stroja, obrađuju signali o pogreškama koje mogu nastati za vrijeme rada itd. On omogućuje da se za vrijeme rada sporih dijelova čitavog digitalnog sistema u glavnoj memoriji računa prema nekom drugom programu. Tako je moguće da se simultano obrađuje više programa. Moguće je da se u memoriji nalazi više programa različnih prioriteta, pa kad se jedan program prekine, npr. zbog izdavanja rezultata, automatski se traži program slijedećeg nižeg prioriteta. Rad prema drugom programu se nastavlja dok se i on ne prekine zbog prijelaza na upotrebu sporih dijelova digitalnog sistema, ili zbog toga što je sve spremno da se nastavi rad prema nekom programu višeg prioriteta. Ovakav se postupak zove multiprogramiranje (USA: multiprogramming, Engl.: time sharing).

Može se očekivati da će se budući napredak u programima odnositi na ove probleme: veza čovjek-računalo, npr. primanje govorenih naloga; razvoj viših jezika za programiranje; automatsko prevođenje s jednog jezika računala na drugi; automatsko prevođenje ljudskih jezika; umjetna inteligencija itd.

RAČUNSKI CENTRI

Da neki računski centar može uspješno djelovati, on mora osim računala i pripadnih uređaja, tj. osim »digitalnog sistema«, imati u svojoj arhivi i dvije vrste programa. Prvu vrstu sačinjavaju programi koji omogućuju rad računala, to su npr. programi za čitanje i pisanje, za automatsko upravljanje radom računala i različni kompilatori. Druga vrsta programa služi rješavanju određenih zadataka kao što su npr. inverzija matrica, linearno programiranje, programi za različne tehničke proračune i slično. U vezi s time se često u stranoj literaturi upotrebljavaju dva pojma: »hardware« označava digitalni sistem, tj. skup uređaja, za razliku od programa i uputa za rad, »software« označuje programe i upute kojima se omogućuje rad računala i pripadnih uređaja. Ova se riječ upotrebljava također u drugom značenju, naime kao skup programa koji se mogu nabaviti za neko računalo od njegova proizvođača.

S obzirom na sastavljanje programa, rad računskog centra može biti organiziran na dva načina. U centru »zatvorena tipa« programe sastavljaju samo programeri u centru, dok u centru »otvorena tipa« vrše programiranje korisnici, tj. oni koji postavljaju problem i kojima trebaju rezultati. Zatvoreni tip centra je prikladan za programiranje jednostavnijih programa, ali on zakazuje kod većih programa, za koje je znatno teže napisati sve postupke sa svim detaljima nego sastaviti sam program na nekom višem jeziku. Kod većih zadataka gube naime stručnjaci za problem previše vremena i rada dok zadatak formuliraju, tj. detaljno opišu tako da više nema nikakvih nejasnoća. U takvom je slučaju bolja uska suradnja između stručnjaka za problem i programera i prema tome povoljniji otvoreni tip računskog centra.

Postoje i centri otvorena tipa s računalom koje ima više pristupa (Multiaccess computer). Više korisnika, npr. 50 programera, mogu biti stalno u vezi s računalom preko posebnih pisača. Računalo redom prima podatke od različnih učesnika, obrađuje ih i daje rezultate tako brzo da korisnik praktički ne osjeća kašnjenje rezultata zbog rada računala s ostalim korisnicima.

Korisnik može biti udaljen od računskog centra i više stotina kilometara i biti povezan s centrom preko teleprinterske veze. Ovakav je način povoljan za male organizacije za koje bi bilo preskupo da nabave vlastito računalo. Slično telefonskim i telex-mrežama postoje u nekim zemljama i specijalne mreže za priključak na centralno računalo. Korisnici, koji komuniciraju s računalom s pomoću specijalnih primo-predajnih pisača, pretplaćeni su na rad računala s više pristupa.

LIT.: *А. П. Ершов, Г. И. Кожухин, Ю. М. Волошин*, Входной язык системы автоматического программирования, Москва 1961. — *B. A. Galler*, The language of computers, New York 1962. — *R. Ledley*, Programming and utilising digital computers, New York 1962. — *E. W. Dijkstra*, A primer of ALGOL 60 programming, London 1962. — *Н. А. Криницкий, Г. А. Миронов, Г. Д. Флоров*, Программирование, Москва 1963. — *Б. В. Гнеденко, В. С. Королюк, Е. Л. Ющенко*, Элементы программирования, Москва ²1963. — *J. Hawgood*, Numerical methods in ALGOL, Maidenhead, Berksh., 1965. — *K. Nicol*, Elementary programming and ALGOL, Maidenhead, Berksh., 1965. — *M. Žokalj* (po McCrackenu), FORTRAN programiranje, Beograd 1966. — *G. Hintz*, Fundamentals of digital machine computing, Berlin-Heidelberg-New York 1966. — *W. Knödel*, Programmieren von Ziffernrechenanlagen, Wien-New York 1966. — *R. K. Richards*, Electronic digital systems, New York 1966. — *A. Ralston, H. Wilf*, Mathematical methods for digital computers, 2 vol., New York 1965/67. — *J. Laborde*, Cours pratique de langage ALGOL, Paris 1967.

B. Zelenko

DIJALIZA, fizički proces i tehnička operacija u kojoj se ostvaruje odvajanje tvari sadržanih (pored otapala) u nekoj otopini, na osnovu različite brzine kojom difundiraju kroz pogodnu čvrstu pregradu (membranu) pod djelovanjem razlike između kemijskih potencijala tih tvari s obje strane membrane (v. *Difuzija*). Dijalizu u praksi redovito prate drugi membranski transportni procesi, naročito osmoza (difuzija otapala kroz membranu). Dijalizom naziva se tehnička operacija kad joj je svrha razdvajanje otopljenih tvari na osnovu razlike kemijskog potencijala (tom se operacijom bavi ovaj članak); operacije u kojima se iskorištava osmoza za odvajanje otapala od otopljene tvari i operacije u kojima se difuzija kroz membranu zbiva pod djelovanjem razlike drugih potencijala (električkog potencijala, pritiska, temperature) obrađene su u drugim člancima ove enciklopedije (v. *Elektrodijaliza, Elektrokinetičke operacije, Ultrafiltracija*).

Dijalizu je prvi upotrijebio Thomas Graham (1861) za odvajanje tvari male molekulske mase (»kristaloida«) od koloida u otopini. On je kao dijalizator upotrijebio široku vertikalno postavljenu cilindričnu cijev (ili bocu bez dna), kojoj je donji otvor zatvorio membranom od životinjskog mjehura. Tako stvorenu posudu s polupropusnim dnom, napunjenu koloidnom otopinom koju je trebalo očistiti od kristaloida, uronio je u čistu vodu. Kroz polupropusnu membranu dijalizatora mogle su difundirati samo male molekule kristaloida, dok su velike molekule koloida ostale u dijalizatoru. Dijalizatori su kasnije dotjerani time što im je oblik polupropusne membrane izmijenjen tako da joj je povećana površina (dat joj je npr. oblik vreće) i što su kao membrane upotrijebljeni drugi materijali: pergament, pergament-papir, celofan i dr.; upotrebljavali su se u laboratoriju i industriji uglavnom za istu svrhu za koju ih je upotrebljavao Graham, tj. za odvajanje visokomolekularnih od niskomolekularnih tvari u otopinama i disperzijama. U novije vrijeme, uslijed napretka u teoriji polupropusnih membrana i u tehnologiji njihove izrade, difuzija je mogla biti primijenjena i na razdvajanje otopljenih tvari kojima je razlika u veličini molekula mnogo manja nego što je razlika između veličine molekula koloida i »kristaloida«.

Dijaliza je danas još u industrijskom mjerilu našla u većem obimu tek mali broj primjena, ali zbog toga što obrađeni materijal pri dijalizi nije podvrgnut nikakvim drastičnim djelovanjima i što pogon dijalize ne zahtijeva ni mnogo energije ni radne snage, može se očekivati da će se industrijska primjena dijalize znatno proširiti kad se ta operacija u svojim teorijskim osnovama bolje upozna, kad se na osnovu toga i dosadašnjih iskustava razvije tehnologija njezine provedbe i kad se izrade metode proračuna potrebne aparature.

Membrane. Priroda membrane, razumljivo, od prvenstvenog je značenja za odvijanje dijalize. U tom pogledu postoje velike razlike. Neke membrane imaju tako široke pore da one samo sprečavaju konvektivno strujanje tekućine s jedne strane membrane na drugu, a molekularna se difuzija otopljenih tvari odvija kao da membrane nema; većinom, međutim, pore su membrana tako male da bitno utječu na brzinu difuzije nekih ili svih molekula kroz membranu. Kroz dugo se vrijeme dijalizna membrana smatrala naprosto sitom koje male molekule propušta a velike zadržava; danas se zna da je djelovanje membrane znatno zamršenije i dobrim dijelom još neobjašnjeno. Molekule nekih membrana djeluju određenim silama i na molekule otopljenih tvari koje se nalaze u izvjesnoj udaljenosti; neke membrane adsorbiraju jedne vrste molekula, i te adsorbirane molekule onda djeluju na druge vrste molekula koje difundiraju; u ioniziranim sistemima nastaju uslijed difuzije jednih molekula potencijalni gradijenti koji djeluju na druge molekule, itd.

Donedavna su se kao membrane za dijalizu upotrebljavali gotovo isključivo celulozni materijali (celofan, pergament-papir, denitrirana nitroceluloza, modificirane celuloze), razmjerno nedavno počele su se praviti membrane od umjetnih smola. Pri tom se pokazalo da postoje beskrajne mogućnosti poboljšanja, modifikacije i prilagođavanja takvih membrana, te se već danas raspolaže velikim izborom membrana od različitih vrsta materijala i različitog stupnja polimerizacije i poprečne povezanosti lančanih molekula; te membrane, prema tome, imaju širok dijapazon mehaničkih, kemijskih i električkih svojstava, prilagođenih različitim svrhama. O tome pobliže vidi članak *Membrane*. O difuziji kroz membrane v. i *Difuzija*.

Kinetika dijalize. Na formuliranju teorije dijalize počelo se raditi tek posljednjih godina i ta teorija danas nije dovoljno potpuna da bi mogla biti od izravne koristi u proračunu dijalizatora. Za te se svrhe obično upotrebljavaju jednadžbe koje se mogu teorijski obrazložiti, a formalno su analogne jednadžbama konvektivnog prenosa topline kroz čvrsti zid:

$$N_i = k_{i1}(c_{i1b} - c_{i12}) = k_{i2}(c_{i12} - c_{i23}) = k_{i3}(c_{i23} - c_{i3b}) \quad (1)$$

$$N_i = K_i(c_{i1b} - c_{i3b}) \quad (2)$$

$$K_i = 1/(k_{i1} + k_{i2} + k_{i3}) \quad (3)$$

U tim jednadžbama N_i je fluks mase otopljene tvari (masa prenijeta kroz jedinicu površine u jedinici vremena), k_1 i k_3 su koeficijenti tekućih slojeva (film coefficients) analogni koeficijentima prelaza topline α; k_2 je koeficijent membrane analogan toplinskoj vodljivosti λ/l, K koeficijent ukupnog prolaza mase; indeks b i brojevi-indeksi odnose se: b na glavne mase tekućine s obje strane membrane, 1 i 3 na granične slojeve s uzvodne odn. nizvodne strane membrane, 2 na membranu; dvostruki indeksi odnose se na razdjelne plohe između b, 1, 2 i 3.

Pretpostavlja se, dakle, da se dijaliza, analogno konvektivnom prenosu topline, zbiva difuzijom kroz tri sloja u seriji, dakle protiv tri otpora $1/k_1$, $1/k_2$ i $1/k_3$, čiji je zbroj ukupni otpor $1/K$. Određivanje brzine dijalize, a prema tome i dimenzioniranje dijalizatora, svodi se time uglavnom na određivanje koeficijenata k.

Koeficijenti tekućih slojeva. U svim modernim dijalizatorima ravne membrane smještene su paralelno na malom razmaku jedna od druge, a tekućina koja se dijalizira (*dijalizat*) struji kontinuirano kroz jedne prostore među membranama, paralelno s ravninama membrana, a kroz susjedne međuprostore, s druge strane membrana, struji u suprotnom smjeru tekućina koja prima tvar što difundira kroz membranu (*difuzat*). Koeficijenti k_1 i k_2 navedeni u gornjim formulama su »tačkasti«, tj. odnose se na jednu tačku na membrani, odn. na tačke pravca okomitog na ravninu membrane u određenoj tački. Međutim, koncentracije (a prema tome i koeficijenti k) prema upravo rečenom mijenjaju se ne samo po pravcima okomitim na ravninu membrane nego i po pravcima paralelnim s tom ravninom, u smjeru strujanja tekućine, uslijed toga što struja difuzata na svom putu preko membrane gubi sastojke koji difundiraju kroz membranu. U tačkama gdje dijalizat odn. difuzat ulazi u dijalizator, k_{i1} odn. k_{i3} je beskonačno velik jer je tamo još $c_{i1b} = c_{i12}$, odn. $c_{i23} = c_{i3b}$, te u jedn. (1) razlika tih koncentracija iščezava, a N_i je prema jedn. (2) konačan. Kako tekućine napreduju na svom putu kroz dijalizator, granični slojevi uz membranu (u odsutnosti sile koja miješa tekućinu u smjeru okomitom na smjer strujanja) postaju sve siromašniji (odn. bogatiji) sastojkom koji difundira kroz membranu, razlike koncentracija u jedn. (1) postaju sve veće, a koeficijenti k_1, k_3 i fluks N sve manji. Sl. 1 prikazuje, na osnovu provedenog računa, kako se mijenja koeficijent k (dat u bezdimenzijskom obliku, kao Sherwoodov broj, analog Nusseltova broja; v. *Difuzija* i *Dimenzijska analiza*) u zavisnosti od (bezdimenzijske) udaljenosti z od ulaza tekućine. (D je koeficijent difuzije, l razmak među membranama, v brzina strujanja; te su veličine za date tekućine, dat aparat i dat kapacitet konstante.) Zanimljivo je da se k asimptotski približava minimalnoj vrijednosti $4D/l$, tj. najveći mogući