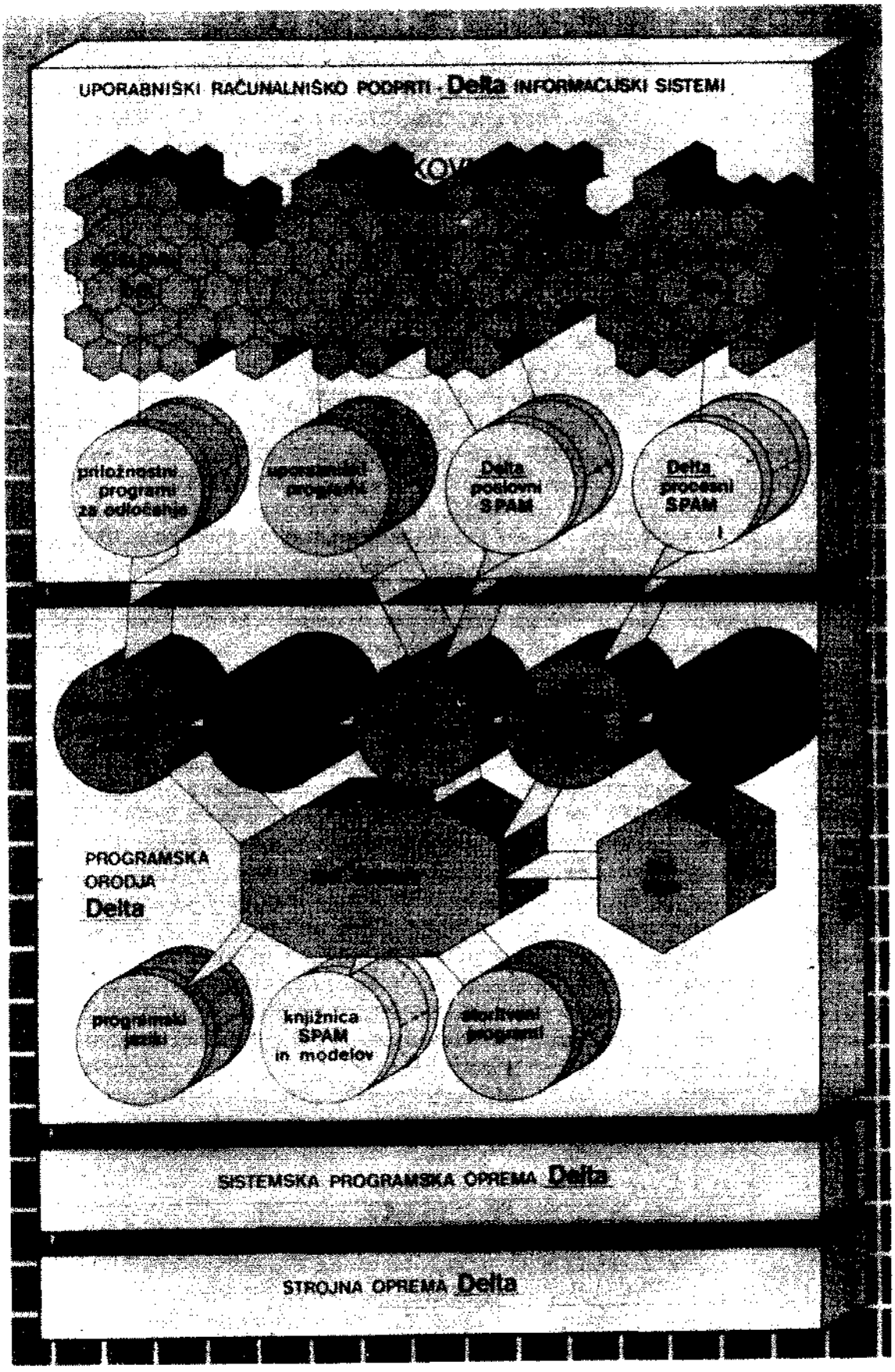
YU ISSN 0350-5596


UPORABNIŠKI RAČUNALNIŠKO PODPRTI Delta INFORMACIJSKI SISTEMI
Delta poslovni SPAM
Delta procesni SPAM
PROGRAMSKA ORODJA Delta
programski jeziki
knjižnica SPAM in modelov
SISTEMSKA PROGRAMSKA OPREMA Delta
STROJNA OPREMA Delta

informatica

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA
IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I
PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO
I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350 – 5596

LETNIK 8, 1984 – Št. 3

Časopis izdaja Slovensko društvo INFORMATIKA, 61000 Ljubljana, Parmova 41, Jugoslavija

UREDNIŠTVO IN UPRAVA: Informatica, Parmova 41, 61000 Ljubljana; telefon (061) 312-988; teleks 31366 YU Delta

LETNA NAROČNINA za delovne organizacije znaša 1900 din, za redne člane 490 din, za študente 190 din; posamezna številka 590 din.
ŽIRO RAČUN: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije.

Na podlagi mnenja Republiškega sekretariata za prosveto in kulturo št. 4210-44/79, z dne 1.2.1979, je časopis oproščen temeljnega davka od prometa proizvodov

TISK: Tiskarna Kresija, Ljubljana

GRAFIČNA OPREMA: Rasto Kirn

VSEBINA

R. Faleskini	3	Razvoj računalništva in usmerjeno izobraževanje
A.P.Železnikar	7	Programi za pisanje programov I
I. Marić	21	Algorithms for Fast B/D Conversion of Integers
R. Markovic	27	Interaktivni generator programa - SIRUP
B. Minovilović P. Kolbezen	31	Paralelno izvajanje opravil v večprocesorskem sistemu I
B. Minovilović P. Kolbezen	35	Paralelno izvajanje opravil v večprocesorskem sistemu II
M. Kukrika	38	Pristup kreiranju raspoređivača zadataka ...
M. Gams in ostali	43	Programski jezik Pascal II
B. Kastelic R. Murn D. Pečet	49	Testiranje ROM pomnilnikov
B. Džonova - Blažič J. Žerovnik	53	Uporaba programskih grafov pri ugotavljanju vzporednosti v računalniških algoritmih II
A.P.Železnikar	57	Programiranje z zbirkami v jeziku CBasic
	69	Uporabni programi
	72	Novice in zanimivosti

informatica

YOURNAL OF COMPUTING AND INFORMATICS

Published by INFORMATIKA, Slovene Society for Informatics, Parmova 41, 61000 Ljubljana, Yugoslavia

HEADQUARTERS: Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia
Phone: 61-312-988; Telex: 31366 YU DELTA

ANNUAL SUBSCRIPTION RATE: US$ 22 for companies, and US$ 10 for individuals

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

YU ISSN 0350 – 5596

VOLUME 8, 1984 – No 3

CONTENTS

R. Feleskini	3	Development of Computing and Oriented Education
A.P.Železnikar	7	Programs Writing Programs I
I. Marić	21	Algorithms for Fast B/D Conversion of Integers
R. Marković	27	SIRUP - An Interactive Program Generator
B. Mihovilović P. Kolbezen	31	Parallel Task Execution in a Multiprocessor System I
B. Mihovilović P. Kolbezen	35	Parallel Task Execution in a Multiprocessor System II
M. Kukrika	38	An Approach to Design of Task Scheduler in a Real-Time Distributed Operating System
M. Gams et All	43	Programming Language Pascal II
B. Kastelic R. Murn D. Peček	49	Testing of ROM Memory
B. Džonova - Jerman J. Žerovnik	53	Application of Program Graph Analysis for Measurment of Parallelism in Computer Algorithms II
A.P.Železnikar	57	Programming in CBasic Using Files
	69	Programming Quickies
	72	News

INFORMATICA 3/1984

RAZVOJ RAČUNALNIŠTVA IN USMERJENO IZOBRAŽEVANJE

R. FALESKINI

DO ISKRA DELTA

UDK: 378.681.3

Posvetovanje o opremljanju šol z računalniki je pokazalo, da manjkajo bistveni elementi planskega pristopa: primerjalna analiza z razvitejšimi deželami, zasotovitev virov financiranja in kvantificirane jugoslovanske potrebe. Prispevek povzema razpravo avtorja z vidiki, ki so pomembni za računalniško industrijo

Development of Computing and Oriented Education

At the meeting of school computerisation in Slovenia it was found that some important planning elements are missing: a comparative analysis of better developed countries, guaranted finance sources, and quantified Yugoslav needs. This article deals with some apects as seen from computer industry point of view.

1. Uvod

Dne 21.4.1984 je potekalo v Ljubljani celodnevno posvetovanje o uvajanju računalnikov v usmerjeno izobraževanje.

V SR Sloveniji imamo veliko število centrov usmerjenega izobraževanja in osemletk, ki bi potrebovale minimalno računalniško opremo. Ideje, kaj pravzaprav šole potrebujejo, so dokaj nejasne. Delovna skupina, ki naj bi pripravila neko standardno konfiguracijo, je ugotovila, da naj bi imela vsaka osemletka po 4 hišne računalnike s televizorjem in kaseto in da naj bi imela vsaka srednja šola en računalnik v konfiguraciji s 4 terminali, z dokaj močnim diskom in sibkim diskom. Poleg tega pa naj bi vsaka srednja šola imela vsaj še 8 hišnih računalnikov z ustreznimi kasetnimi enotami in televizorji.

Ob času posvetovanja so si udeleženci posvetovanja (bilo jih je okrog 80) ogledali tudi razstavo DO Iskra Delta v Cankarjevem domu. Razen tega je bilo v prostorih VTOZD Kemija v Vegovi ulici, kjer je posvetovanje potekalo, razstavljenih 8 prototipov različnih računalnikov. Kot proizvajalci so se tukaj s prototipi pojavili Iskra Široka potrošnja, Gorenje, Slovenijales TOZD Inženiring in Inštitut Jožef Stefan. Z računalniki se je pojavilo tudi več zasebnikov, ki imajo določene ideje, kako bi razrešili problem računalništva v usmerjenem izobraževanju.

Pokazalo se je, da obstaja več idej o prototipih in da se v bistvu lotevamo stvari na tistem koncu, kjer so še realizive, ne pa iz proizvodnega vidika. Na posvetovanju je prišlo do izraza tudi to, da ima usmerjeno izobraževanje tudi na univerzitetnem nivoju premalo računalniške opreme in da bi poleg te profesionalne računalniške opreme, če jo tako imenujemo, potrebovali nujno še opremo nižjega nivoja ali večje število terminalov ali pa hišne računalnike, na katerih bi bilo možno pripravljati kasete, ki bi jih potem obdelali na večjih računalnikih.

Ko se je govorilo o potrebah, se je posebej ugotavljalo število hišnih računalnikov, in razpravljavci so prišli do števila 2000 hišnih računalnikov. Proizvajalci smo opozarjali na probleme, ki izhajajo predvsem iz tega, da sistem usmerjenega izobraževanja

nima plana

Na posvetovanju izražena povpraševanje po računalniški opremi v bistvu ni plačilno sposobno povpraševanje in osnovni problemi obratovanja te opreme niso niti približno razrešeni (v kolikor bi ta oprema prišla na srednje šole oz. šole usmerjenega izobraževanja). Od razpravljavcev zaželena oprema predstavlja v bistvu divergentno opremo v povezavi z obstoječo opremo na šolah, kjer imamo situacijo, nastalo skozi velike napore DO Iskre Delte v preteklih obdobjih. Divergentne ideje z divergentnimi izvori financiranja pa nujno vodijo tudi v divergentno opremo in naposled v rušenje kakršnihkoli možnosti za standardizacijo.

V usmerjenem izobraževanju ni konkretnih idej, kako bi pravzaprav povezovali Sinclairje (za katere je veliko navdušenja) z opremo Iskre Delte na profesionalnem nivoju: čeprav je ta problem rešljiv z enostavnimi lokalnimi mrežami, se o tem ne razmišlja in v bistvu so prisotne izolirane divergentne ideje.

Pri nadaljnjih kontaktih industrije s šolniki bo potrebna velika mera potrpežljivosti in previdnosti tudi zaradi tega, ker ni dovolj finančnih sredstev, ker še ni niti centra, ki bi kvalificirano in odgovorno kupoval ustrezno

opremo ter skrbel za njeno obratovanje. Tako bi se lahko dogodilo, da ko bi ta oprema prispela, ne bi ustrezno delalovala in te obratovalne pomanjkljivosti bi verjetno lahko povzročile mnogo večje stroške kot bi bili prihranki pri izboru cenješe opreme oz. popusti pri nabavi te opreme.

Šolniki se v bistvu obnašajo izraziteo vsak po svoje: vsak ima svoje probleme za najpomembnejše in čeprav ne vidi celote ima ideje, ki jih zelo vehementno razlagajo; šolniki še niso pripravljeni na združevanje in na izdelavo enotnega plana nabav opreme za usmerjeno izobraževanje. Divergenčne ideje so prisotne tudi na univerzah in pozitivne izkušnje Republiškega računskega centra in Računalniškega centra univerze so v bistvu šolnikom neznane.

Za svojo nespodbudno situacijo iščejo šolniki krivca predvsem v domači industriji. Povečini so prepričani, da bi morali domači proizvajalci sami financirati celoten projekt opremljanja šol z računalniki, pri čemer ne razmišljajo niti o ekonomski moči domačih proizvajalcev, ki je za tak projekt daleč pretibka, niti o kompletnih potrebah celotnega izobraževalnega sistema.

Več kot 70 % vseh računalniških zmogljivosti v SFRJ, s katerimi razpolagajo izobraževalne organizacije, je iz domače proizvodnje podjetij Iskra-TOZD Računalniki, Elektrotehna-DO Delta, Iskra Delta oziroma iz zastopniških programov teh podjetij (CDC in DEC). Pred združitvijo v enovito DO Iskra Delta sta TOZD Računalniki in DO Delta vsaka po svojih močeh podpirali uvajanje računalništva v srednje in visoke šole. Obe organizaciji sta tudi sodelovali v pripravah učnih programov in lahko rečemo, da se je ta dejavnost nadaljevala tudi po združitvi v enovito delovno organizacijo Iskra Delta pred dvema letoma.

2. Potrebe po delavcih za informacijsko tehnologijo

Iskra Delta je kot proizvodna organizacija življenjsko zainteresirana za različne profile delavcev, ki se srečujejo z računalniško tehnologijo. Prvi profil oblikujejo delavci, ki se zaposlujejo neposredno v razvoju in proizvodnji računalniških sistemov, in sicer tako v proizvodnji aparaturne kot v proizvodnji programske opreme. Ti delavci naj bi bili usposobljeni za delo v domačih organizacijah, proizvajalkah računalniških elementov, aparaturne in programske opreme, in sistemov.

Domača računalniška industrija je zainteresirana tudi za vzgojo delavcev, ki znajo uporabljati računalniško tehnologijo iz domače proizvodnje, ki to tehnologijo implementirajo v različnih okoljih, tj. delajo v organizacijah, ki računalniško opremo uporabljajo.

Zainteresirani smo za multidisciplinarna in interdisciplinarna znanja, ki jih morajo imeti strokovnjaki vseh drugih področij, ki se z informacijsko tehnologijo srečujejo v svojih rednih delovnih procesih, tako v procesih družbenih in upravnih dejavnosti kot tudi v procesih same proizvodnje v primarnih, sekundarnih in terciarnih sektorjih.

Seveda smo zainteresirani, da so ta znanja primerljiva s svetovnimi znanji. Mislimo namreč, da uporaba računalniške tehnologije v našem okolju ne more biti bistveno drugačna kot je v razvitem svetu.

Prve (skromne) uspehe pri razširjanju znanj s področja informatike in računalništva smo že dosegli na nivoju univerz. Če pogledamo nivo srednjih šol, moramo ugotoviti, da je ta nivo zaradi mnogo večje razširjenosti, zaradi pomanjkanja računalniške opreme, zaradi pomanjkanja učiteljev, zaradi pomanjkanja materializiranih interesov dejansko mnogo manj opremljen, kot bi bilo potrebno glede na potrebe po prestrukturiranju jugoslovanske družbe. Na nivoju osnovnih šol in nižjih nivojih seveda sploh še ne moremo govoriti o kakršni koli družbeno organizirani akciji, ki bi bila po svojem obsegu tudi družbeno relevantna.

Potrebe po delavcih, ki bodo delali neposredno na razvoju, izgradnji in vzdrževanju v informacijskih sistemih, so zelo velike. Smatramo, da bi moralo delati v proizvodnji aparaturne opreme, proizvodnji programske opreme in v neposredni implementaciji informacijske tehnologije v Sloveniji leta 2000 vsaj 15000 delavcev v celotnem proizvodno-poslovnem kompleksu informacijske tehnologije. Torej bi morali na nivoju srednjih šol takoj zagotoviti ustrezno število mest za te manjkajoče kadre.

Danes lahko ocenjujemo, da dela v proizvodno-poslovnem kompleksu informacijske industrije v Sloveniji približno 2000 delavcev, če upoštevamo poleg DO Iskra Delta še druge, oziroma če definicijo postavimo zelo ohlapno, ker je v naših razmerah moramo. V tem kompleksu upoštevamo poleg Iskre Delte tudi druge Iskrine DO in pa delavce, ki so zaposleni v trgovskih delovnih organizacijah in delavce, ki delajo v organizacijah, kjer se proizvajajo samo posamezni elementi računalniške opreme.

Drug kompleks delavcev, ki jih je potrebno usposabljati, so delavci za potrebe implementacije računalniške tehnologije, kjer gre za približno trikrat večje število. To število predstavlja seveda potrebne kadre, ki jih je možno zaposliti na ta način, da upoštevamo potrebe jugoslovanskega in pa zlasti svetovnega trga po informacijski tehnologiji v prihodnjih 15 letih. Poudarimo, da je SOZD Iskra kot celota že danes zelo usmerjena na svetovni trg in da se bo ta usmerjenost še povečevala v naslednjih 15 letih. To pa pomeni, da moramo računati s kapacitetami in organizacijo izobraževalnega sistema, ki bo sposoben dati že v kratkem kritične mase kadrov za posamezna področja, ki jih bomo potrebovali za pokrivanje dejavnosti na svetovnem trgu.

Jasno je, da pri tem ne računamo samo na delavce, ki se bodo izšolali v SR Sloveniji. DO Iskra Delta je že danes jugoslovanska organizacija s svojimi 894 delavci v 20 krajih širom po državi (samo v Beogradu je več kot 100 delavcev; glej podatke o izobrazbeni strukturi in zaposlitvi po republikah v naslednjih tabelah). Poleg tega ima Iskra Delta kooperacijske odnose na zelo visokem nivoju s sedemnajstimi delovnimi organizacijami iz vse Jugoslavije.

Izobrazbena struktura zaposlenih v DO Iskra Delta 31. 12. 1983	
doktorji znanosti	7
magistri znanosti	21
visokošolska izobrazba	324
višješolska izobrazba	120
srednješolska izobrazba	372
pol- in nekvalificirani delavci	50
skupaj delavcev	894

Zaposlenost delavcev DO Iskra Delta po republikah 31. 12. 1983	
SR Slovenija	654
SR Hrvaška	65
SR Srbija	133
SR Makedonija	26
SR Bosna in Hercegovina	16
skupaj zaposlenih	894

Pri proizvodnji aparaturne in programske opreme ni mogoče zapirati rastoče informacijsko industrijo v republiške meje, zato si ne smemo delati iluzij, da bodo šole v drugih republikah čakale, če se v Sloveniji ne bomo hitro razvijali. Nasprotno, upoštevati moramo, da so v nekaterih drugih republikah sposobni združevati napore celotne družbene skupnosti okrog določenih projektov, kar so že nekajkrat dokazali (primer SAP Vojvodine in SR Hrvatske)! zato verjamemo, da bomo v Jugoslaviji prej kot slej imeli kadre za informacijsko industrijo.

Od sposobnosti izobraževalnega sistema v Sloveniji bo v veliki meri odvisna velikost segmenta Jugoslovanske informacijske industrije, ki se bo razvil v Sloveniji.

3. Vsebina znanj in izobraževalne metode

Cilj, da z računalniško tehnologijo začnemo ustvarjati večji del dohodka na svetovnem trgu, predpostavlja jasno tudi bistvene spremembe v samem izobraževalnem sistemu. Te spremembe vidimo v nekaj smereh. Ena od sprememb je v tem, da se bo morala vsebina znanj strokovnjakov, ki prihajajo iz naših šol, bistveno bolj približati znanjem, ki jih imajo strokovnjaki iz razvitejših držav. Tako se bomo morali pri svojih učnih programih bistveno približati na eni strani priporočilom UNESCA, to se pravi standardom, ki jih dobimo prek Organizacije združenih narodov, na drugi strani pa učnim programom, ki jih imajo v deželah Evropske gospodarske skupnosti, v ZDA in na Japonskem. Ne moremo si namreč predstavljati, da bi z bistveno različnimi znanji lahko konkurirali produktom industrij teh dežel, ker pri drugih faktorjih, kot so kapital, organizacija, obremenjenost gospodarstva in drugo nismo boljši od proizvajalcev v teh državah.

To velja za vsebino znanj naših strokovnjakov tako v proizvodnji aparaturne kot v proizvodnji sistemske programske opreme, aplikativne programske opreme kot v mejnih področjih, kot so npr. avtomatika, kibernetika, telematika oziroma komunikacije, organizacijska interdisciplinarna znanja itd.

Te usmeritve pa lahko pomenijo bistvene spremembe za izobraževalni sistem. Izobraževalni sistem sam oziroma družba bosta morala zagotoviti veliko bolj primerljive pogoje, kar zadeva znanja učiteljev, kar se tiče preverjanja znanja ter samih izpitnih oziroma ocenjevalnih postopkov. Bistvena pa bo tudi zagotavljanje materialnih pogojev dela tako prenovljenega izobraževalnega sistema. Smatramo, da je izobraževalna skupnost dolžna, da primerjalno analizira, na kakšen način se financirajo investicije in obratovanje računalniške opreme v izobraževalnem sistemu v različnih razvitih deželah. Pri tem je potrebno jasno ločiti problematiko pouka o računalnikih in z računalniki.

V prihodnjih letih lahko pričakujemo množično uporabo metod programiranega učenja s pomočjo računalnikov in povezovanje računalniške tehnologije z drugimi mediji (zaprta televizija, film, projekcije diapozitivov itd.).

Znanja, ki jih bodo dobili učenci v usmerjenem izobraževanju, morajo biti taka, da bodo predstavljala dobro bazo za nadaljno specializacijo v sami proizvodni organizaciji oziroma za nadaljno specializacijo v delovni organizaciji, ki uporablja računalnik. Upoštevati moramo tudi, da bodo strokovnjaki morali uporabljati različne podatke ali pa programske produkte iz svetovnih baz podatkov, da bodo morali znati uporabljati sodobno informacijsko tehnologijo tudi v tem smislu, da bodo prišli do teh baz podatkov na ustrezno računalniško podprt način, od koder bodo dobivali podatke o samih programskih produktih, o patentih, o raziskovalnih in razvojnih nalogah v tujini, o člankih, knjigah, revijah, o standardih itn. Jasno je, da bo moralo biti tem strokovnjakom jasno, kakšne so pravne oblike zaščite znanja, kaj je vsebina znanja, kaj je pravzprav programska oprema glede na to, da v današnjih pogojih tega znanja še nimamo, torej znamo programirati, redkeje pa vemo, kako bi iz svojega programa naredili tržni produkt, ki se je možno tržiti na domačem ali na svetovnem trgu.

V zvezi z novo izobraževalno opremo bodo potrebne tudi investicije v zgraditev telekomunikacijskih linij in pokrivanje stroškov prenosa podatkov.

4. Izobraževalna dejavnost kot tržišče računalniških proizvodov

Izobraževalni sistem predstavlja brez dvoma pomembno tržišče računalniških proizvodov. V razvitem svetu je izobraževalni sistem največji porabnik hišnih računalnikov, osebnih računalnikov in drugih tipov mikroračunalnikov. Tako imajo v razvitih zahodnih deželah, npr. v Veliki Britaniji produkcijo računalnikov posebej za namene uporabe v izobraževalnem sistemu v šolah. Ta trg je zanimiv tudi za domače proizvajalce. Pri tem je potrebno poudariti to! računalniški trg v izobraževanju ne more biti trg, ki se oblikuje po tržnih zakonitostih, temveč je to lahko samo planiran trg! o tem trgu je smiselno govoriti le v primeru, ko ustrezne finančne institucije planirajo ustrezna sredstva za investicije in za obratovanje računalnikov. Mislimo, da je smiselno govoriti o tem računalniškem trgu samo, v kolikor gre za plačilno sposobno povpraševanje.

Praktično danes v nobeni razviti deželi ne moremo govoriti o računalniškem trgu v izobraževalnem sistemu kot o trgu, kjer veljajo čiste tržne zakonitosti, temveč vedno le kot o trgu, kjer se striktno planira ne samo enoletno temveč srednjeročno. Interes domačih proizvajalcev je, da bi tak način planiranja zaživel tudi v Jugoslaviji. Le tako bi vedeli, od kod bodo pritekala sredstva za računalnike v usmerjenem izobraževanju. Pri tem je bistveno spoznanje, da je računalniška kultura stvar, ki zadeva celoten izobraževalni sistem, da je računalništvo tisto znanje, ki bo posojevalo uporabo katerih koli strojev v kateri koli proizvodnji, ki bo dejavno prisotno ob katerem koli družbeno informacijskem sistemu in bo torej moralo biti prisotno tudi v ustreznih smereh usmerjenega izobraževanja. Zaradi vsega tega računalništva ne kaže omejevati le na tiste segmente usmerjenega izobraževanje, ki

naj vzgojijo kadre za proizvajalce računalniške opreme in kadre za neposredno uporabo računalnikov.

Problem je torej kompletni izobraževalni sistem, specifičen problem pa je seveda usmerjeno izobraževanje v tistih šolah, ki naj vzgojijo kadre za potrebe proizvodnje in za potrebe neposredne uporabe v računalniških informacijskih sistemih. Za ta center usmerjenega izobraževanja je potrebno poleg splošne cenene opreme, ki jo lahko predstavljajo danes hišni računalniki in osebni računalniki, torej nivo mikro računalniške opreme, zasotoviti tudi opremo višjih stopenj, pri čemer mislimo na zelo zmogljive delovne postaje, torej inženirske delovne postaje z grafiko, inženirske delovne postaje, ki omogočajo priključevanje različnih instrumentov, to se pravi različnih vhodov, in pa inženirske delovne postaje, ki omogočajo kontrolo različnih procesov. Drug segment bodo splošno namenski računalniki, ki bodo omogočali vključevanje v uporabo podatkovnih baz, uporabo sodobnih programskih produktov, programskih orodij, programskih generatorjev itd., v kompletne sodobne programske arhitekture in arhitekture mrež. Torej bo zelo važna tudi uporaba teh računalnikov na mejnih področjih, kjer se računalništvo stika s področjem organizacije, s področjem tehnologije, s področjem komunikacij, s področjem same elektronske produkcije, s področjem produkcije strojne industrije, s področjem robotike, s področjem kibernetike.

Menimo torej, da morajo šole pri opremljanju z računalniki oziroma pri planiranju sredstev, upoštevati ne samo potrebe po samih računalnikih, to se pravi po strojni opremi, po programski opremi, ampak morajo zadovoljevati tudi potrebe po tem, da postane prenašanje znanj mnogo bolj sodobno. Tukaj mislimo na vlaganja v področje, kot so računalniško poučevanje, uporaba sodobnih svetovnih podatkovnih baz itn., kar pa seveda vse zahteva dodatna sredstva in napore. Kot že rečeno, moramo ta sredstva plansko zagotoviti. Vse to pa je možno samo v okvirih samoupravne interesne skupnosti za izobraževanje in s sodelovanjem poklicne organizacije, kakršno predstavlja na nivoju Univerze Edvarda Kardelja računalniški center univerze.

Menimo, da je tudi vse razpravljanje in ugibanje o tem, ali domača proizvodnja blokira uvoz teh hišnih računalnikov ali ne, ravno negacija planskega pristopa v naših razmerah in v bistvu nepoznavanje svetovnih in pa domačih razmer na tem področju. Prvi interes domače industrije je, da se informacijska kultura razširi; tu bi bila domača industrija voljna materialno podpreti potreben uvoz, v kolikor bi se pokazalo, da je to optimalna, družbeno usklajena rešitev. Današnji postopek uvoza mikroračunalnikov ni povezan z nikakršnjim dajanjem soglasij domačih proizvajalcev. Jugoslovani, ki niso na delu v tujini, ne morejo uvažati mikroračunalnikov podobno kot katerihkoli drugih predmetov (fotoaparatov, oblek), katerih vrednost presega predpisani znesek.

5. Sklep

Jugoslovanska računalniška industrija mora biti zainteresirana za trg računalnikov v usmerjenem izobraževanju pa tudi v osnovnem izobraževanju in želi, da bi se ta trg oblikoval. Možnost oblikovanja tega trga pa je seveda samo v tem, da se zasotovijo ustrezni izvori za financiranje in da se planirajo ustrezna sredstva, ki bodo omogočala prestrukturiranje. Še bolj kot za trenutno prodajo v usmerjenem izobraževanju pa je industrija zainteresirana za kvalitetne kadre za prizvodnjo in uporabo informacijske tehnologije.

INFORMATICA 3/1984

PROGRAMI ZA PISANJE PROGRAMOV I

ANTON P. ŽELEZNIKAR

DO ISKRA DELTA

UDK: 519.682.8

Članek opisuje tri generirne segmente, ki pišejo programe za vhod, izhod in navodila. Generatorji so posebni programi, napisani v jeziku CBasic, generirajo pa programe (generirance) v istem jeziku. Sestavljanje uporabniških programov z uporabo generatorjev je omogočeno na več načinov. Z generatorji lahko gradimo eno samo rezultatno zbirko (generirani uporabniški program), tako da nad njo zaporedoma uporabljamo generatorje (sproti dodajamo generirani tekst). Zgradimo pa lahko tudi ločene generirane segmente, ki jih v rezultatni program povežemo z INCLUDE ukazi in z ročno napisanimi (dodatnimi) programskimi segmenti. Seveda pa lahko uporabimo tudi mešani način sestavljanja rezultatnega programa.

Programs Writing Programs I

This article describes three generating segments for writing input, output, and instruction programs. These generators are particular programs written in CBasic and generating programs in the same language. The assembling of user programs by means of generators can be done in several ways. Using generators a single (resulting) file (user program) can be built by sequential usage of several generators (adding new program text to existing one in the file). But, one can generate isolate user program segments and put them together (by placing them into program and linking them) using INCLUDE directives and inserting program text written by hand. The third way is to use a mixed approach for program assembly (sequential adding and include procedure).

1. Uvod

Programi, ki pišejo programe, se imenujejo tudi programski generatorji (kratko generatorji). Kako lahko program napiše uporabniški program z uporabo dialoga za določeno aplikacijo?

Najpreprostejša zamisel programskega generatorja temelji na enostavnem povezovanju že izdelanih rutin (podprogramov, prevedenih v strojni jezik), ki jih je mogoče inicializirati (modificirati, parametrično prirediti) z dialogom med uporabnikom in generirnim sistemom. Določeni parametri teh osnovnih rutin so uporabniško spremenljivi (nastavljivi), tako da dobi lahko rutina opredeljeno (dovolj natančno, specializirano) funkcijo. Takšen generator proizvede povezavo uporabniško modificiranih rutin na ravnini strojnega jezika.

Zahtevnejša in semantično splošnejša zamisel programskega generatorja temelji na izvirnem razvoju programa v visokem programirnem jeziku (prenosljivost na različne sisteme z ustreznimi prevajalniki) skladno s semantiko uporabniškega algoritma. Od te splošne zamisli je mogoče sestopati na več načinov in v več korakih, tako da se še vedno dosegajo bistveni praktični učinki (v lahkotnosti, prijaznosti, hitrosti in avtomatičnosti nastajanja novega programa).

V praktičnih primerih avtomatičnega ali natančneje polavtomatičnega generiranja programov bomo lahko izbirali poti med obema skrajnostima, tako da bo tudi izdelava (razvoj) generatorja (programa, ki piše programe) potekala v sprejemljivih časovnih, stroškovnih in kadrovskih obsegih. Zamisli programskih generatorjev v tem članku bodo temeljile na generiranih izdelkih (generirancih) v visokem programirnem jeziku in v posameznih primerih bomo uporabljali jezik CBasic tako za generatorje kot za generirance. V prvi fazi bomo gradili enostavnejše (vendar netrivialne), parcialne generatorje, ki jih bomo povezovali interaktivno, ročno, s poseganjem uporabnika prek urejevalnika v generirane segmente (v generirance). Kasneje bomo enostavne generatorje dopolnjevali, jih zapletali (povečevali njihovo kompleksnost oziroma zmogljivost) in medsebojno povezovali, pri čemer bomo na ravnini jezika CBasic uporabljali tudi vključitveni mehanizem (npr. %INCLUDE ukaz).

V prvem delu članka bomo razdelili parcialne generatorje na tri osnovna programirna (tudi semantična) področja:

-- vhodni programski generator (OBVHODn.BAS bo vobče ime programa oziroma zbirke za oblikovanje oziroma generiranje vhodnega segmenta, tj. generiranca) bo oblikoval vhodne

```
A)crun2 obvhod3

CRUN VER 2.07P
Število spremenljivk ?  10
Dimenzija polj ?  50
Ime spremenljivke štev.  1  ($ za niz) :
? prvo.ime$
Tekst za spremenljivko prvo.ime$:
? Vstavi ime:
Ime spremenljivke štev.  2  ($ za niz) :
? druso.ime:$
Tekst za spremenljivko druso.ime$:
? Vstavi druso ime (če je):
Ime spremenljivke štev.  3  ($ za niz) :
? priimek$
Tekst za spremenljivko priimek$:
? Vstavi priimek:
Ime spremenljivke štev.  4  ($ za niz) :
? ulica$
Tekst za spremenljivko ulica$:
? Vstavi ime ulice s številko:
Ime spremenljivke štev.  5  ($ za niz) :
? post.stev$
Tekst za spremenljivko post.stev$:
? Vstavi poštno številko:
Ime spremenljivke štev.  6  ($ za niz) :
? kraj$
Tekst za spremenljivko kraj$:
? Vstavi ime kraja:
Ime spremenljivke štev.  7  ($ za niz) :
? drzava$
Tekst za spremenljivko drzava$:
? Vstavi ime države:
Ime spremenljivke štev.  8  ($ za niz) :
? datum$
Tekst za spremenljivko datum$:
? Vstavi datum vplačila:
Ime spremenljivke štev.  9  ($ za niz) :
? znesek
Tekst za spremenljivko znesek:
? Vstavi velikost vplačila:
Ali želis preizkus območja (d/n) ?  d
Najmanjša sprejemljiva vrednost ?  190.00
Največja sprejemljiva vrednost ?  1900.00
Ime spremenljivke štev.  10  ($ za niz) :
? status$
Tekst za spremenljivko status$:
? Vstavi status naročnika:
Indeks spremenljivke za končanje ?  3
Kakšna je vrednost končanja ?  KONEC
Številka začetne vrstice programa ?  10
Programski inkrement vrstice ?  10
```

Lista 1. Ta lista prikazuje primer vhodnega dialoga, na osnovi katerega naj bi se generiral vhodni del programa, prikazan v listi 2. V dialogu navedemo najprej število spremenljivk (polj v zapisu zbirke), ki jih zaporedoma poimenujemo. K imenu dodamo po želji še pripadajoči tekst, ki pove uporabniku, kaj in kako naj vstavlja. Pri številskih spremenljivkah (celoštevilskih in realnih) bomo imeli še preizkus vrednostnega območja (po izbiri z dialogom). Na koncu vhodnega dialoga bomo določili spremenljivko (s pripadajočim indeksom) in vrednost spremenljivke (npr. KONEC) za končanje vtavljanja podatkov v generiranem programu (npr. v listi 2, desno). Posamezni segmenti generiranega programa bodo oštevilčeni, kot dopušča to sintaksa jezika CBasic (zaradi možnih skokov in subrutinskih klicev); dialog za oštevilčevanje vrstic imamo na koncu te liste. Seveda število možnih vhodnih spremenljivk ne bo vnaprej omejeno (npr. število polj v zapisu) in spremenljivčna imena bodo lahko svobodno (ustrezno) izbrana. Po potrebi bomo lahko dialog in generirani segment tudi spremenili (s spremembo generatorja in/ali z ročno modifikacijo).

```
10
DIM  prvo.ime$(50),Ð
     druso.ime$(50),Ð
     priimek$(50),Ð
     ulica$(50),Ð
     post.stev$(50),Ð
     kraj$(50),Ð
     drzava$(50),Ð
     datum$(50),Ð
     znesek(50),Ð
     status$(50)
iX=1

20
PRINT "Vstop stevilka "; iX

30
INPUT "Vstavi ime: "; prvo.ime$(iX)

40
INPUT "Vstavi druso ime (če je): "; druso.ime$(iX)

50
INPUT "Vstavi priimek: "; priimek$(iX)
IF priimek$(iX)="KONEC" THEN GOTO 130

60
INPUT "Vstavi ime ulice s številko: "; ulica$(iX)

70
INPUT "Vstavi poštno številko: "; post.stev$(iX)

80
INPUT "Vstavi ime kraja: "; kraj$(iX)

90
INPUT "Vstavi ime države: "; drzava$(iX)

100
INPUT "Vstavi datum vplačila: "; datum$(iX)

110
INPUT "Vstavi velikost vplačila: "; znesek(iX)
IF znesek(iX)<190.00 OR Ð
   znesek(iX)>1900.00 THEN GOTO 110

120
INPUT "Vstavi status naročnika: "; status$(iX)

iX=iX+1: GOTO 20

130
Ali želiš shranitev na disk (d/n) ?  d
Ali želis novo (n) ali obstoječo (o) zbirko ? n
Vstavi ime zbirke ?  ike1.bas
Ali želis končati (d/n) ?  d
```

Lista 2. Ta lista kaže z dialogom v listi 1 (levo) generirani program in na koncu liste še dialog za shranitev tega programa v izbrano zbirko in za končanje generiranja. Znak 'Ð' ponazarja znak '\' v jeziku CBasic.

dialogne stavke, s katerimi se bodo opredeljevale poljubne vhodne spremenljivke, preizkušanje njihovih vrednostnih območij in kasneje shranjevanje njihovih vrednosti v zbirke; ta generator bo vobče zelo univerzalen in le izjemoma se bodo pojavile večje ali obsežnejše vhodne zahteve

-- izhodni programski generator (OBIZHODn.BAS bo vobče ime programa oziroma zbirke za oblikovanje oziroma generiranje izhodnega segmenta) bo oblikoval izhodne rezultatne in sporočilne stavke z upoštevanjem določenih formatov (naslovni deli, pravila tabeliranja, rezultatni formati); ta generator ne bo tako splošen kot vhodni generator, s katerim se vhodni podatki v bistvu zajemajo

```
D=============================================D
D       Program za oblikovanje vhodnega       D
D                   programa                  D
D---------------------------------------------D
D       Ime zbirke: OBVHOD3.BAS               D
D       Ime 'include' zbirke: OBDISK.BAS      D
D       marec 1984      A. P. Železnikar      D
D=============================================D

10
DIM tipX(200),v$(200),p$(200),lv$(200),hv$(200)

D---------------------------------------------D
D               Vhodni dialog                 D
D---------------------------------------------D

INPUT "Število spremenljivk ? ";nX
INPUT "Dimenzija polj ? ";mX
FOR iX=1 TO nX: tipX(iX)=0: NEXT iX
FOR iX=1 TO nX
  PRINT "Ime spremenljivke štev. ";iX;D
      " ($ za niz) : "
  INPUT v$(iX)
  IF right$(v$(iX),1)="$" THEN tipX(iX)=3
  PRINT "Tekst za spremenljivko ";v$(iX);": "
  INPUT p$(iX)
  IF tipX(iX)=3 THEN GOTO 15
  INPUT "Ali želiš preizkus območja (d/n) ? ";D
      z$
  IF z$="d" OR z$="D" THEN tipX(iX)=1D
    ELSE GOTO 15
  INPUT "Najmanjša sprejemljiva vrednost ? ";D
      lv$(iX)
  INPUT "Največja sprejemljiva vrednost ? ";D
      hv$(iX)
  tipX(iX)=2
15
NEXT iX
INPUT "Indeks spremenljivke za končanje ? ";tX
INPUT "Kakšna je vrednost končanja ? ";tv$
INPUT "Številka začetne vrstice programa ? ";fr
INPUT "Programski inkrement vrstice ? ";inc

D---------------------------------------------D
D           Oblikovanje DIM stavka            D
D---------------------------------------------D

DIM l$(10+6*nX)
plc=fr: jX=1
l$(jX)=str$(plc): jX=jX+1
a$=""
IF nX>1 THEN a$=",D"
l$(jX)="DIM  "+v$(1)+"("+str$(mX)+")"+a$
jX=jX+1
FOR iX=2 TO nX
  a$=""
  IF iX<nX THEN a$=",D"
  l$(jX)="     "+v$(iX)+"("+str$(mX)+")"+a$
  jX=jX+1
NEXT iX
plc=plc+inc

D---------------------------------------------D
D           Oblikovanje vrstice  iX=1         D
D---------------------------------------------D

l$(jX)="iX=1": jX=jX+1
loop=plc

D---------------------------------------------D
D   Oblikovanje PRINT, INPUT in IF stavkov    D
D              vhodne zanke                   D
D---------------------------------------------D

l$(jX)="": jX=jX+1
l$(jX)=str$(plc): plc=plc+inc: jX=jX+1
l$(jX)="PRINT ""Vstop stevilka "";iX: jX=jX+1
FOR iX=1 TO nX
  er=plc
  l$(jX)="": jX=jX+1
  l$(jX)=str$(plc): plc=plc+inc: jX=jX+1
  l$(jX)="INPUT """+p$(iX)+""";"+v$(iX)+"(iX)"
  jX=jX+1
  IF iX<>tX THEN GOTO 20

D       D-----------------------------------D
D       D       Izstop iz vhodne zanke      D
D       D-----------------------------------D

  dnX=jX REM          Indeks izstopnega stavka
  q$=""
  IF tipX(iX)=3 THEN q$=chr$(34)
  l$(jX)="IF "+v$(iX)+"("+iX)="+q$+D
        tv$+q$+" THEN GOTO "
  jX=jX+1

20
D       D-----------------------------------D
D       D    Oblikovanje preizkusa območja  D
D       D-----------------------------------D

  IF tipX(iX)<>2 THEN GOTO 25
  l$(jX)="IF "+v$(iX)+"("+iX)<"+lv$(iX)+" OR D"
  jX=jX+1
  l$(jX)="   "+v$(iX)+"("+iX))">"+hv$(iX)+ D
        " THEN GOTO "+str$(er)
  jX=jX+1

25
NEXT iX
l$(jX)="": jX=jX+1
l$(jX)="iX=iX+1: GOTO "+str$(loop)
jX=jX+1: l$(jX)=""

D       D-----------------------------------D
D       D     Dopolnitev izstopnega stavka  D
D       D-----------------------------------D

l$(dnX)=l$(dnX)+str$(plc)
jX=jX+1
l$(jX)=str$(plc)

XINCLUDE obdisk.bas

END
```

Lista 3. Program na tej listi (zgoraj) je vhodni generator, ki z svojim dialogom (npr. v listi 1) sestavi vhodni program v listi 2 in z uporabo končnega dialoga (npr. na koncu liste 2) shrani ta program v imenovano zbirko in konča generiranje (z dialogom). Iz programa v tej listi je razvidno, kako se npr. generira program v listi 2, ko se oblikuje začetni DIM stavek in nato spremenljivčni vhodni stavki. Na koncu liste imamo INCLUDE ukaz, ki vključe segment za zapis generiranega programa na disk in končanje generiranja (zbirka OBDISK.BAS). INCLUDE segment je prikazan v listi 4.

-- ukazni programski generator (OBUKAZn.BAS bo vobče ime programa oziroma zbirke za oblikovanje oziroma generiranje tkim. uvodnega teksta, navodil o uporabi generiranega programa) bo oblikoval zaslonu primerno obliko in format uporabnih sporočil

Opisani trije generirni segmenti lahko proizvedejo v celoti ali delno programska zaporedja, ki so značilna za večino uporabniških programov. Kasneje bomo te osnovne segmente dopolnjevali in povezovali npr. s ciljem upravljanja podatkovnih baz. V članku bomo najprej prikazali enostavno metodologijo nastajanja programskega generatorja.

2. Vhodni programski generator

Kakšen generirani vhodni programski segment pravzaprav želimo? Kakšna je njegova dovolj splošna oblika? Kakšen naj bo ustrezni generiranec? Vobče so nam potrebna polja sovisnih, povezanih, relacijskih podatkov. Polje ima tu dvojen pomen: kot masiv (array) in kot element zbirčnega zapisa (record field). Osnova je zbirčni zapis (file record), ki naj ga sestavlja poljubno število zapisnih polj (kratko polja). Če ima zapis 10 polj, bomo potrebovali 10 vhodnih spremenljivk. Generirani segment bo imel v jeziku CBasic tkim. DIM stavek na samem začetku. Velikost dimenzije bo odvisna od števila pričakovanih zapisov oziroma od količinskega pošiljanja teh zapisov v zbirko. Ta dimenzija mora biti uporabniško določljiva. Generirani segment naj bi imel tudi možnosti preizkušanja številskih obsegov za posamezne vhodne spremenljivke, ki niso niznega tipa.

V listi 1 imamo primer vhodnega dialoga za 10 spremenljivk z dimenzijo 50. Za vsako spremenljivko določamo s tem dialogom pripadajoči vhodni tekst in pri številski spremenljivki še preizkus vrednostnega območja (najmanjša in največja vrednost). Na koncu dialoga določimo še vrednost specificirane spremenljivke (njen zaporedni indeks), s katero se vstavljanje konča. Ko določimo še številko začetne vrstice programa v jeziku CBasic in vrstični inkrement, se dialog konča, nakar se na zaslonu ali tiskalniku izpiše generirani program, ki je prikazan v listi 2. Temu programu, ki se končuje s prazno vrstico, oštevilčeno v našem primeru s številko 130, sledi še kratek dialog za ustrezno shranitev generiranega segmenta na disk ali pa za ponovitev oziroma nadaljevanje postopka vhodnega generiranja.

Generirani program v listi 2 ima 10 spremenljivk, tako da bomo imeli zapise s po desetimi zapisnimi polji, in sicer:

prvo.ime$, drugo.ime$, priimek$,
ulica$, post.stev$, kraj$, drzava$,
datum$, znasek, status$

To zapisno zgradbo bomo npr. uporabili v zbirki časopisnih naročnikov, ko bomo s spremenljivko status$ lahko opredelili, ali je naročnik Študent (naročniški popust), posameznik, šlan društva ali podjetje (ime podjetja bomo vpisali kot spremenljivko priimek$). Priimek "KONEC" (pisano z velikimi črkami) bo v generiranem programu povzročil izstop iz generiranega segmenta v nadaljevalno programsko vrstico (oznaka 130 v listi 2).

Dialog iz liste 1 in na osnovi tega dialoga generirani program in končni dialog v listi 2 je mogoče dobiti z generatorskim programom v listi 3. Ta program (generator) je ustrezno parametriziran s podatki, ki se vstavljajo s konzole. Ta vhodni generator je enostaven in bi ga bilo mogoče dopolniti še tako, da bi v generiranem segmentu povzročil

- -- nastanek ustreznih (parametriziranih) komentarjev, ki bi omogočali lažjo ročno modifikacijo generiranca
- -- nastanek segmenta za vpisovanje vhodnih podatkov (spremenljivk) v diskovne zbirke različnih tipov (zaporedni ali naključni dostop)
- -- nastanek drugih semantičnih dodatkov (po potrebi)

```
D-----------------------------------------D
D    Izpis generiranega programa na zaslon  D
D       in na disk (v izbrano zbirko)       D
D-----------------------------------------D
D       Ime zbirke: OBDISK.BAS              D
D       Ta zbirka je 'include' zbirka za    D
D         OBVHOD3.BAS, OBIZHOD3.BAS in      D
D         OBUKAZ1.BAS                       D
D       marec 1984      A. P. Železnikar    D
D-----------------------------------------D

D-----------------------------------------D
D          Izpis generiranega programa      D
D-----------------------------------------D

PRINT: PRINT
FOR kX=1 TO JX
  PRINT 1$(kX)
NEXT kX

D-----------------------------------------D
D        Shranitev na disk ali izstop ?     D
D-----------------------------------------D

INPUT "Ali želiš shranitev na disk (d/n) ? ";z$
IF z$<>"d" AND z$<>"D" THEN GOTO 90

55
D-----------------------------------------D
D     Zapis generiranega programa na disk   D
D-----------------------------------------D

PRINT "Ali želiš novo (n) ali obstoječo (o)";
PRINT " zbirko ";
INPUT z$
IF z$="n" OR z$="N" THEN GOTO 60
IF z$="o" OR z$="O" THEN GOTO 65
GOTO 55

60
  INPUT "Vstavi ime zbirke ? ";f$
  CREATE f$ AS 1
  GOSUB 100
  GOTO 80

65
  INPUT "Vstavi ime zbirke ? ";f$
  OPEN f$ AS 1
70
  IF END #1 THEN 75
  READ #1;slepo$
  GOTO 70
75
  GOSUB 100
80
  CLOSE 1

90
D-----------------------------------------D
D     Nadaljevanje ali konec generiranja ?  D
D-----------------------------------------D

INPUT "Ali želiš končati (d/n) ? ";z$
IF z$="d" OR z$="D" THEN STOP
GOTO 10

100
D-----------------------------------------D
D          Zapis generiranega programa      D
D                   na disk                 D
D-----------------------------------------D

    FOR kX=1 to JX
      PRINT USING "&";#1;1$(kX)
    NEXT kX
    RETURN
D-----------------------------------------D
```

Lista 4. Ta lista prikazuje segment (zbirko tipa include), ki izpiše generirani program na zaslon in po želji še v imenovano zbirko. Ta segment se vključuje v zbirke OBVHOD3.BAS, OBIZHOD3.BAS in OBUKAZ1.BAS (liste 3, 5 in 6).

```
Ð=============================================Ð
Ð      Program za oblikovanje izhodnega      Ð
Ð                  programa                   Ð
Ð---------------------------------------------Ð
Ð      Ime zbirke: OBIZHOD3.BAS               Ð
Ð      Ime 'include' zbirke: OBDISK.BAS       Ð
Ð      marec 1984      A. P. Železnikar       Ð
Ð=============================================Ð

10
DIM tip%(200),v$(200),w%(200),d%(200),Ð
    p1$(200),p2$(200),p3$(200)

Ð---------------------------------------------Ð
Ð              Izhodni dialog                 Ð
Ð---------------------------------------------Ð

INPUT "Število spremenljivk ? ";n%

FOR i%=1 TO n%: tip%(i%)=0: NEXT i%

FOR i%=1 TO n%
  PRINT "Ime spremenljivke štev. "; i%; Ð
      " ($ za niz) : "
  INPUT v$(i%)
  IF right$(v$(i%),1)="$" THEN tip%(i%)=3
  INPUT "Širina stolpca ? "; w%(i%)
  IF tip%(i%)=3 THEN GOTO 20
  PRINT "Število mest za decimalno vejico ";
  INPUT d%(i%)
20
  PRINT "Imamo 3 naslovne vrstice za stolpec"
  PRINT
  PRINT "Stolpna naslovna vrstica štev. 1 ";
  INPUT p1$(i%)
  IF len(p1$(i%)) > w%(i%) THEN GOTO 20
30
  PRINT "Stolpna naslovna vrstica štev. 2 ";
  INPUT p2$(i%)
  IF len(p2$(i%)) > w%(i%) THEN GOTO 30
40
  PRINT "Stolpna naslovna vrstica štev. 3 ";
  INPUT p3$(i%)
  IF len(p3$(i%)) > w%(i%) THEN GOTO 40
NEXT i%

INPUT "Število presledkov med stolpci ? "; sp%

DIM l$(200)

Ð---------------------------------------------Ð
Ð     Generiranje izhodnega programskega      Ð
Ð                segmenta                     Ð
Ð---------------------------------------------Ð

j%=1
l$(j%)="PRINT"
j%=j%+1
t%=0

FOR i%=1 TO n%
  t%=t%+w%(i%-1)+sp%
  l$(j%)="PRINT tab("+Ð
         str$(int(t%+(w%(i%)-Ð
         len(p1$(i%)))/2+1))+");"+Ð
         chr$(34)+p1$(i%)+chr$(34)+";"
  j%=j%+1
NEXT i%

l$(j%)="PRINT"
j%=j%+1
t%=0

FOR i%=1 TO n%
  t%=t%+w%(i%-1)+sp%
  l$(j%)="PRINT tab("+Ð
         str$(int(t%+(w%(i%)-Ð
         len(p2$(i%)))/2+1))+");"+Ð
         chr$(34)+p2$(i%)+chr$(34)+";"
  j%=j%+1
NEXT i%

l$(j%)="PRINT"
j%=j%+1
t%=0
FOR i%=1 TO n%
  t%=t%+w%(i%-1)+sp%
  l$(j%)="PRINT tab("+Ð
         str$(int(t%+(w%(i%)-Ð
         len(p3$(i%)))/2+1))+");"+Ð
         chr$(34)+p3$(i%)+chr$(34)+";"
  j%=j%+1
NEXT i%

l$(j%)="PRINT"
j%=j%+1
l$(j%)="FOR i%=1 TO m%"
j%=j%+1
t%=0

FOR i%=1 TO n%
  IF tip%(i%)=3 THEN Ð
    l$(j%)="  a$="+v$(i%)+"(i%)":Ð
    j%=j%+1:Ð
    GOTO 50
  l$(j%)="  a="+v$(i%)+"(i%)"
  j%=j%+1
  l$(j%)="  w%= "+str$(w%(i%))+Ð
         ": d%="+str$(d%(i%))
  j%=j%+1
  l$(j%)="  GOSUB 6000"
  j%=j%+1
50
  t%=t%+w%(i%-1)+sp%
  l$(j%)="  PRINT tab("+Ð
        str$(int(t%+w%(i%)+1))+"-len(a$)): a$;"
  j%=j%+1
NEXT i%

l$(j%)="  PRINT"
j%=j%+1
l$(j%)="NEXT i%"
j%=j%+1

%INCLUDE obdisk.bas

END
```

Lista 5. Ta lista, ki prikazuje program OBIZHOD3.BAS, kaže drugega od treh programskih generatorjev, ki jih bomo uporabljali. Znak 'Ð' se uporablja namesto znaka '\' (YU abeceda). Ta generator ima oštevilčene vrstice (od 10 do 50), vanj vključeni segment OBDISK.BAS na koncu liste pa ima oštevilčenja od 55 do 100 (glej listo 4). Na začetku generatorja imamo polja spremenljivk in v teh poljih se bodo paralelno zbirali podatki za posamezne spremenljivke (npr. tip%(i%), v$(i%), w%(i%), d%(i%), p1$(i%), p2$(i%), p3$(i%) za spremenljivko z indeksom i%). Dimenzija teh polj omogoča zajetje 200 izhodnih spremenljivk. Na začetku opredelimo število spremenljivk n% nato pa začnemo teh n% spremenljivk zajemati. Najprej zajamemo ime spremenljivke v v$(i%), potem se določi njen tip%(i%), vstavimo širino njenega stolpca w%(i%) in število mest za decimalno vejico d%(i%). Nadalje vstavimo še tri stolpne naslovne vrstice p1$(i%), p2$(i%) in p3$(i%). Zadnji podatek, ki ga vstavimo, je število presledkov med stolpci sp%. V polje l$ se bodo shranjevale vrstice generiranega programa, ki se bo oblikoval v treh zaporednih FOR zankah (glej listo). Generirni program oblikuje nato še FOR stavek za vstavljanje spremenljivčnih vrednosti v izhodno tabelo (glej kasnejši primer) z uporabo zunanje rutine, oštevilčene s 6000. To konverzijsko rutino bomo napisali ročno in ne bo generirana. Na koncu liste imamo znani segment OBDISK.BAS (lista 4) za izpis generiranega segmenta na zaslon in nato še možnega zapisa v imenovano zbirko.

3. Izhodni programski generator

Kakšen generirani izhodni programski segment pravzaprav želimo? Semantično se bodo v tem generiranem segmentu pojavile spremenljivke iz vhodnega generiranega segmenta in izhodni generirani segment bo semantično nadaljevanje vhodnega generiranega segmenta (ta segment se bo v celotnem generiranem programu pojavil za vhodnim segmentom). Kakšna bo dovolj splošna oblika izhodnega segmenta? Če je izhod povezan s podatki v zbirki in večina uporabnih programov bo to povezavo imela, potem pričakujemo v izhodnem generiranem segmentu del, ki bo zagotavljal prenos podatkov z diska; vendar tega dela zaenkrat ne bomo obravnavali.

Največkrat bomo s programom, ki ponazarja izhodni segment, želeli oblikovati nekakšno preglednico (npr. cenik, račun, seznam, tabelo). Tako bomo imeli možnost, da z dialogom oblikujemo stolpce tabele, napise (naslove) nad stolpci, njihovo širino, njihovo medsebojno razdaljo in da v te stolpce vnašamo vrednosti z dialogom določenih spremenljivk.

Ker bomo primer z dialogom za generiranje izhodnega segmenta obravnavali v enem od naslednjih poglavij, si na kratko oglejmo zgradbo izhodnega generatorja v listi 5.

Na začetku programa v listi 5 imamo definicijo ustreznega števila polj. Z INPUT stavkom določimo število izhodnih spremenljivk, ki jih nato poimenujemo. Imena shranjujemo v polje v$ in ugotavljamo, ali so spremenljivke tipa niz in če so, jim v polju tip% prirejamo vrednosti 3. Ker bo izhodna spremenljivka oblikovala stolpec v izhodni razpredelnici, se vprašujemo za širine stolpcev in te širine shranjujemo v polje w%.

Kadar je spremenljivka številska, ji določimo število mest za decimalno vejico (piko) (nič mest pri celih številih, npr. pri komadih). Ta števila shranjujemo v polju d%.

K vsaki spremenljivki določimo njen trivrstični stolpni naslov v izhodni tabeli in te tri naslovne vrstice shranjujemo v polja p1$, p2$ in p3$. Pri tem ne dopuščamo, da bi bila posamezna naslovna vrstica daljša od širine polja (w%). Ko so ti podatki zbrani, določimo še število presledkov med stolpci (oddaljenost, spremenljivka sp%).

V polju l$ se bodo shranjevale vrstice izhodnega (generiranega) programa. Prva vrstica generiranega programa je stavek PRINT. Ostali PRINT stavki v polju l$ bodo oblikovali za vsako izhodno spremenljivko trivrstični stolpni naslov (trije zaporedni FOR stavki v listi 5). Ko je tako oblikovan programski segment za izpis vseh stolpnih naslovov v izhodni tabeli, se morajo oblikovati še stavki za izpis vrednosti izhodnih spremenljivk v posamezne stolpce tabele, in sicer z generacijo ustrezne FOR zanke. Tu se program končuje z INCLUDE ukazom, ki izpiše generirani program na zaslon in omogoča zapis tega programa v imenovano zbirko (lista 4).

4. Programski generator za navodila

Namen programskega generatorja za navodila in izpis teksta (zbirka OBUKAZ1.BAS) je generiranje stavkov za pojasnila uporabniku in za izpis različnih besedil v uporabniškem programu. Ta generator mora proizvesti ustrezne PRINT stavke, dana pa mora biti tudi možnost za popravljanje vrstic besedil, vstavljenih z dialogom (vrstična urejevalna subrutina). Na koncu dialoga se generirani program izpiše na zaslon (zaradi kontrole) in po potrebi shrani v imenovano zbirko. Seveda lahko programski generator (zbirka OBUKAZ1.BAS) uporabimo večkrat in oblikujemo ločene segmente (shranitev nastalih generirancev v različne zbirke), ki jih bomo kasneje vključevali (z INCLUDE ukazi) v rezultatni uporabniški program.

Lista 6 prikazuje generator OBUKAZ1.BAS. Na začetku se pripravi zaslon in dialog se začne. Nato se vpisujejo želene vrstice, ki pojasnjujejo program, vstavljajo pa se različna besedila za kasnejši izpis. Funkcija tega generatorja je razumljiva iz liste 6 in iz kasnejših primerov.

5. Generiranje določenega primera

Doslej smo opisali tri generirne programe, in sicer

OBVHOD3.BAS (lista 3),
OBIZHOD3.BAS (lista 5) in
OBUKAZ1.BAS (lista 6)

ki jih lahko uporabimo pri avtomatičnem oblikovanju določenega primera. V tem primeru imajmo določene segmente, ki jih bomo dodali ročno (v skladu s semantiko programa), ker jih z obstoječimi generatorji ne bo moč generirati.

Uporabniški program je mogoče zgenerirati v eno samo zbirko, tako da v ustreznem vrstnem redu uporabljamo generatorje in dodajamo ročno manjkajoče segmente. Uporabniški program pa lahko generiramo tudi s sestavljanjem različnih zbirk, ko uporabimo INCLUDE ukaze. Ta način generiranja bomo prikazali v našem primeru.

Lista 7 kaže konkreten uporabniški program, ki je hkrati tudi načrt za generiranje. S tem programom želimo pisati ponudbe za različne izdelke. Ta program vsebuje pet generiranih segmentov, ki jih vključujemo z INCLUDE ukazi, in sicer:

AZUKAZ1.BAS	generiran z	OBUKAZ1.BAS
AZVHOD1.BAS	generiran z	OBVHOD3.BAS
AZUKAZ2.BAS	generiran z	OBUKAZ1.BAS
AZIZHOD1.BAS	generiran z	OBIZHOD3.BAS
AZUKAZ3.BAS	generiran z	OBUKAZ1.BAS

Rutina SUBR1.BAS je bila napisana ročno in ves ostali tekst programa OBPROG.BAS v listi 7 je bil dodan ročno. Oglejmo si nastanek posameznih INCLUDE segmentov tega programa.

5.1. Oblikovanje segmenta AZUKAZ1.BAS

Na začetku liste 7 smo najprej pripravili zaslon, nato pa smo vključili segment AZUKAZ.BAS; ta segment bomo dobili z uporabo generatorja OBUKAZ1.BAS. Na začetku želimo oblikovati navodila za uporabo programa in prav to funkcijo bo opravil prvi segment.

Lista 8 prikazuje izvajanje generatorskega programa OBUKAZ1.BAS, s katerim želimo oblikovati programski segment AZUKAZ1.BAS. Generatorju sporočimo približno število besedilnih vrstic (20). Sledi navodilo za končanje vstavljanja besedila in za odgovarjanje na vprašanja. Sedaj lahko vpišemo 20 vrstic besedila, izstopimo pa lahko tudi prej, če vstavimo s konzole znak ´CTL-u´. V 12 vrstic smo vpisali želene uporabniško navodilo in v vrstici 13 smo izstopili z znakom ´CTL-u´. Vpisano besedilo se nato izpiše v obliki, ki se bo pojavila pri izvajanju uporabniškega programa. Sedaj je mogoče vrstice še poljubno popravljati, če se za to odločimo. Če se ne, sledi na zaslonu izpis generiranega segmenta in vprašanje za shranitev na disk v ime-

```
D=============================================D
D      Program za oblikovanje uporabniških    D
D                  navodil                    D
D---------------------------------------------D
D       Ime zbirke: OBUKAZ1.BAS               D
D       Ime 'include' zbirke: OBDISK.BAS      D
D       marec 1984     A. P. Železnikar       D
D=============================================D

10
D---------------------------------------------D
D   Priprava zaslona: home, clear screen      D
D---------------------------------------------D

aa$=chr$(30)+chr$(26)
PRINT aa$

D---------------------------------------------D
D     Začetni dialog in opredelitev znakov    D
D---------------------------------------------D

INPUT "Koliko vrstic bo približno ? ";iX
DIM l$(int(iX*1.5))
cr$=chr$(13)
bs$=chr$(8)
eq$=chr$(34)
qq$=chr$(21) REM  Znak 'CTL-u'
PRINT "Vpiši 'CTL-u' za končanje"
PRINT "Na vprašanja odgovarjaj z d/n ";
PRINT "ali z D/N"
lnX=1

15
D---------------------------------------------D
D        Urejevalno vstavljanje vrstic        D
D---------------------------------------------D

PRINT
PRINT "Vpiši vrstico ";lnX
GOSUB 110

IF ch$()qq$ THEN GOTO 15
nlX=lnX-1

20
PRINT: PRINT
FOR iX=1 TO nlX
  PRINT l$(iX)
NEXT iX
PRINT
INPUT "Ali želiš spremeniti vrstico ? ";z$
IF z$()"d" AND z$()"D" THEN GOTO 30

25
INPUT "Katero vrstico ? ";lnX
IF lnX)nlX OR lnX<1 THEN GOTO 25
PRINT l$(lnX)
PRINT
INPUT "Ali je to prava vrstica ? ";z$
IF z$()"d" AND z$()"D" THEN GOTO 25
l$(lnX)=""
PRINT "Izpiši vrstico ";lnX
GOSUB 110
GOTO 20

30
D---------------------------------------------D
D    Oblikovanje pripadajočih PRINT stavkov   D
D---------------------------------------------D

FOR iX=1 TO nlX
  lX=len(l$(iX))
  FOR jX=1 TO lX
    IF lX=0 THEN l$(iX)="PRINT": GOTO 40
    IF left$(l$(iX),1)()" " THEN GOTO 35
    l$(iX)=right$(l$(iX),len(l$(iX))-1)
  NEXT jX

35
  s1$="tab(": s2$=")";": sj$=str$(jX)
  IF jX=1 THEN s1$="": s2$="": sj$=""
  l$(iX)="PRINT "+s1$+sj$+s2$+eq$+l$(iX)+eq$

40
NEXT iX
jX=nlX

XINCLUDE obdisk.bas

110
D---------------------------------------------D
D        Vrstična urejevalna subrutina        D
D---------------------------------------------D
D   Pomen krmilnih znakov:                    D
D                                             D
D        BS   (CTL-h)  pomik v levo           D
D        CR   konec vrstice                   D
D        NAK  (CTL-u)  izstop iz urejevalnika D
D---------------------------------------------D

  chX=conchar%
  ch$=chr$(chX)

115
  IF ch$=qq$ THEN RETURN
  IF ch$()cr$ AND ch$()bs$ AND ch$()qq$ THEN D
     l$(lnX)=l$(lnX)+ch$: GOTO 110
  IF ch$=bs$ AND len(l$(lnX))<=1 THEN D
    l$(lnX)="": GOTO 110
  IF ch$=bs$ THEN D
    l$(lnX)=left$(l$(lnX),len(l$(lnX))-1):D
    GOTO 110
  IF ch$=cr$ THEN lnX=lnX+1: RETURN

STOP
```

Lista 6. Program v tej listi lahko generira programske segmente za izpis različnih besedil (v uporabniškem programu); to velja tako za navodila o uporabi generiranega programa kot za izpise, ki so povezani s semantiko uporabniškega programa. Znak 'CTL-u' se uporablja za končanje generiranja, znak BS (backspace) pa pri popravljanju vpisanega besedila. Program OBUKAZ1.BAS je tako enostavni urejevalnik in generator. Tudi ta generator vključuje segment OBDISK.BAS (z INLUDE ukazom). Znak "Đ" je nadomestilo za znak "\".

novano zbirko (AZUKAZ1.BAS). Tako je ustrezni programski segment dobljen in shranjen.

5.2. Oblikovanje segmenta AZVHOD1.BAS

Naslednji segment v listi 7 je AZVHOD1.BAS, ki ga generiramo s programom OBVHOD3.BAS. S tem segmentom moramo zajeti želene vhodne spremenljivke.

Lista 9 prikazuje izvajanje generatorskega programa OBVHOD3.BAS. V ponudbi bomo imeli določeno število izdelkov, ki imajo enotenske cene. Torej bo število spremenljivk 3 (količina, izdelek, cena), ponudili pa bomo lahko do 20 različnih izdelkov. Podobno kot v listi 1 določujemo podatke, ki spadajo k posameznim spremenljivkam (ime spremenljivke, pripadajoči tekst za zajemanje, kontrola vrednostnega območja). Proti koncu dialoga določimo spremenljivko za končanje in njeno končno vrednost, tako da bomo v uporabniškem programu lahko izstopili pri manj kot 20 izdelkih. Ker potrebuje ta generirani segment oštevilčenje programskih vrstic, določimo še začetno in inkrementno število.

Generirani program se nato zapiše na zaslon, kot kaže lista 9 in se končno shrani v izbrano zbirko AZVHOD1.BAS. Tako je segment oblikovan in shranjen za vključitev.

```
╔═══════════════════════════════════════════════╗
║      Primer generiranega uporabniškega        ║
║                  programa                     ║
╠═══════════════════════════════════════════════╣
║        Ime zbirke: OBPROG.BAS                 ║
║        Imena 'include' zbirk: AZUKAZ1.BAS     ║
║                               AZVHOD1.BAS     ║
║                               AZUKAZ2.BAS     ║
║                               AZIZHOD1.BAS    ║
║                               AZUKAZ3.BAS     ║
║                               SUBR1.BAS       ║
║        april 1984          A. P. Železnikar   ║
╚═══════════════════════════════════════════════╝

aa$ = chr$(30)+chr$(26)
PRINT aa$

%INCLUDE azukaz1.bas
%INCLUDE azvhod1.bas

m% = i%-1
tt = 0
DIM cel.cena(20)
FOR n%=1 TO m%
  cel.cena(n%)=kolicina(n%)*cena(n%)
  tt=tt+cel.cena(n%)
NEXT n%

LPRINTER
%INCLUDE azukaz2.bas
%INCLUDE azizhod1.bas

a = tt
GOSUB 6000
PRINT
PRINT " Skupaj"; tab(46-len(a$))a$
PRINT

%INCLUDE azukaz3.bas
%INCLUDE subr1.bas

END
```

Lista 7. Ta lista predstavlja generirani uporabniški program, ki je nastal z avtomatičnim generiranjem segmentov AZ??????.BAS in z ročno napisanimi segmenti. Program je primer za pisanje ponudb. S stavkom LPRINTER se ponudba izpiše na vrstičnem tiskalniku. Celoten program je mogoče napisati tako, kot je prikazano v tej listi, ko so bili generirani posamezni segmenti in ko smo natanko določili semantiko programa. Znak '₩' se uporablja namesto znaka '\'.

5.3. Oblikovanje segmenta AZUKAZ2.BAS

Nadaljnji segment iz liste 7, ki ga moramo oblikovati (generirati), je zbirka AZUKAZ2.BAS. S tem segmentom dobimo izpisni program, s katerim je določen sedež pošiljatelja ponudbe, datum, številka dopisa in opis predmeta ponudbe. Lista 10 kaže najprej ustrezen dialog, ki se konča v 14 vrsticah, ko smo vstavili znak 'CTL-u'. Na koncu vpišemo dobljeni programski segmet v zbirko z imenom AZUKAZ2.BAS, ki se vključuje v celoten uporabniški program v listi 7 z ukazom %INCLUDE azukaz2.bas.

5.4. Oblikovanje segmenta AZIZHOD1.BAS

V listi 11 imamo prikazano izvajanje generatorja OBIZHOD3, ko generiramo vključitveno zbirko AZIZHOD1.BAS za listo 7. Pri tem generiranju moramo upoštevati združljivost vhodnih in izhodnih spremenljivčnih imen (vhodno semantiko). Najprej se opredeli število spremenljivk, ko želimo 4 izhodne tabelne stolpce, in sicer za spremenljivke

Lista 8. Spodnja lista prikazuje dialog in generacijo za program, ki bo izpisal navodila za uporabo programa. Ta navodila vstavljamo z dialogom vrstico za vrstico (12 vrstic), končamo pa v 13. vrstici z znakom 'CTL-u'. Vstavljeno besedilo se nato izpiše v obliki, kot jo bo izpisal uporabniški program. V tej točki je možno še popravljanje vstavljenih vrstic. Zatem se izpiše zgenerirani program na zaslon in shranimo ga lahko v imenovano zbirko.

```
A)crun2 obukaz1

CRUN VER 2.07P

Koliko vrstic bo približno ?  20
Vpiši 'CTL-u' za končanje
Na vprašanja odgovarjaj z d/n ali z D/N

Vpiši vrstico  1

Vpiši vrstico  2
      PROGRAM ZA IZPIS PONUDBE
Vpiši vrstico  3

Vpiši vrstico  4
Ta program bo napisal ponudbo ali
Vpiši vrstico  5
  naročilo za do 20 izdelkov.
Vpiši vrstico  6

Vpiši vrstico  7
Pri potrditvi na zaslonu vstavi
Vpiši vrstico  8
  opis izdelka, njegovo ceno in
Vpiši vrstico  9
  količino.
Vpiši vrstico  10

Vpiši vrstico  11
Vstavi ime izdelka 'KONEC', ko je
Vpiši vrstico  12
  seznam končan.
Vpiši vrstico  13



     PROGRAM ZA IZPIS PONUDBE

Ta program bo napisal ponudbo ali
  naročilo za do 20 izdelkov.

Pri potrditvi na zaslonu vstavi
  opis izdelka, njegovo ceno in
  količino.

Vstavi ime izdelka 'KONEC', ko je
  seznam končan.

Ali želiš spremeniti vrstico ?  n


PRINT
PRINT tab(6);"PROGRAM ZA IZPIS PONUDBE"
PRINT
PRINT "Ta program bo napisal ponudbo ali"
PRINT tab(3);"naročilo za do 20 izdelkov."
PRINT
PRINT "Pri potrditvi na zaslonu vstavi"
PRINT tab(3);"opis izdelka, njegovo ceno in"
PRINT tab(3);"količino."
PRINT
PRINT "Vstavi ime izdelka 'KONEC', ko je"
PRINT tab(3);"seznam končan."
Ali želiš shranitev na disk (d/n) ?  d
Ali želiš novo (n) ali obstoječo (o) zbirko ? n
Vstavi ime zbirke ?  azukaz1.bas
Ali želiš končati (d/n) ?  d

A)
```

```
A>crun2 obvhod3

CRUN VER 2.07P
Število spremenljivk ?  3
Dimenzija polj ?  20
Ime spremenljivke štev.  1  ($ za niz) :
? izdelek$
Tekst za spremenljivko izdelek$:
? Vstavi ime izdelka:
Ime spremenljivke štev.  2  ($ za niz) :
? cena
Tekst za spremenljivko cena:
? Vstavi ceno za enoto izdelka:
Ali želiš preizkus območja (d/n) ?  d
Najmanjša sprejemljiva vrednost ?  10
Največja sprejemljiva vrednost ?  10000
Ime spremenljivke štev.  3  ($ za niz) :
? kolicina
Tekst za spremenljivko kolicina:
? Vstavi število kosov izdelka:
Ali želiš preizkus območja (d/n) ?  d
Najmanjša sprejemljiva vrednost ?  1
Največja sprejemljiva vrednost ?  200
Indeks spremenljivke za končanje ?  1
Kakšna je vrednost končanja ?  KONEC
Številka začetne vrstice programa ?  20
Programski inkrement vrstice ?  20


20
DIM  izdelek$(20),B
     cena(20),B
     kolicina(20)
iX=1

40
PRINT "Vstop stevilka "; iX

60
INPUT "Vstavi ime izdelka:"; izdelek$(iX)
IF izdelek$(iX)="KONEC" THEN GOTO 120

80
INPUT "Vstavi ceno za enoto izdelka:"; cena(iX)
IF cena(iX)<10 OR B
   cena(iX)>10000 THEN GOTO 80

100
INPUT "Vstavi število kosov izdelka:"; kolicina(iX)
IF kolicina(iX)<1 OR B
   kolicina(iX)>200 THEN GOTO 100

iX=iX+1: GOTO 40

120
Ali želiš shranitev na disk (d/n) ?  d
Ali želis novo (n) ali obstoječo (o) zbirko ? n
Vstavi ime zbirke ?  azvhod1.bas
Ali želiš končati (d/n) ?  d

A>
```

Lista 9. Ta lista prikazuje dialog in na osnovi tega dialoga generirani vhodni segment uporabniškega programa. Za tri vhodna polja s po 20 elementi so najprej izbrana imena spremenljivk izdelek$, cena in kolicina. Za te spremenljivke se vselej vprašuje po pripadajočem tekstu, ki je navodilo za zajemanje teh podatkov uporabniku. Pri številskih spremenljivkah, kot sta cena in kolicina, se vprašuje po preizkusu njihovih vrednostnih območij. Končno se z dialogom določi indeks spremenljivke (izdelek$ ima indeks 1, cena indeks 2 in kolicina indeks 3) in njena vrednost za končanje vstavljanja podatkov v uporabniškem programu; ta vrednost je v primeru s te liste "KONEC" za indeks 1, tj. za izdelek$. Po opredelitvi številke začetne vrstice generiranega programa se določi še vrstični inkrement. Generirani program se nato izpiše na zaslon, nakar sledi odločitev o njegovi shranitvi v imenovano zbirko.

Lista 10. Spodnja lista prikazuje generiranje besedilnega segmenta za primer ponudbe, ko imamo 13 vrstic teksta. Po dialogu se končna oblika izpiše in generirani program se pokaže na zaslonu. Ta program se nato lahko vpiše v imenovano zbirko, v našem primeru v AZUKAZ2.BAS.

```
A>crun2 obukaz1

CRUN VER 2.07P

Koliko vrstic bo približno ?  30
Vpiši 'CTL-u' za končanje
Na vprašanja odsovarjaj z d/n ali z D/N

Vpiši vrstico  1
TOZD Živalska farma, DO Živalstvo
Vpiši vrstico  2
Lisičja cesta 24 - 29
Vpiši vrstico  3
61000  L j u b l j a n a
Vpiši vrstico  4

Vpiši vrstico  5
               DO Nova menažerija
Vpiši vrstico  6
               Ob potoku 63
Vpiši vrstico  7
               69113   M a č j i   d o l
Vpiši vrstico  8

Vpiši vrstico  9

Vpiši vrstico  10
Ljubljana, 22. 4. 1984      Štev. dop. 443-52KL
Vpiši vrstico  11

Vpiši vrstico  12
Predmet: Ponudba za dobavo živali
Vpiši vrstico  13

Vpiši vrstico  14


TOZD Živalska farma, DO Živalstvo
Lisičja cesta 24 - 29
61000  L j u b l j a n a

               DO Nova menažerija
               Ob potoku 63
               69113   M a č j i   d o l


Ljubljana, 22. 4. 1984      Štev. dop. 443-52KL

Predmet: Ponudba za dobavo živali


Ali želiš spremeniti vrstico ?  n


PRINT "TOZD Živalska farma, DO Živalstvo"
PRINT "Lisičja cesta 24 - 29"
PRINT "61000  L j u b l j a n a"
PRINT
PRINT tab(16);"DO Nova menažerija"
PRINT tab(16);"Ob potoku 63"
PRINT tab(16);"69113   M a č j i   d o l"
PRINT
PRINT
PRINT "Ljubljana, 22. 4. 1984      Štev. dop. 443-52KL"
PRINT
PRINT "Predmet: Ponudba za dobavo živali"
PRINT
Ali želiš shranitev na disk (d/n) ?  d
Ali želis novo (n) ali obstoječo (o) zbirko ? n
Vstavi ime zbirke ?  azukaz2.bas
Ali želis končati (d/n) ?  d

A>
```

```
A)crun2 obizhod3

CRUN VER 2.07P
Število spremenljivk ?  4
Ime spremenljivke štev.  1  ($ za niz) :
? kolicina
Širina stolpca ?  4
Število mest za decimalno vejico ? 0
Imamo 3 naslovne vrstice za stolpec:

Stolpna naslovna vrstica štev. 1 ? kom.
Stolpna naslovna vrstica štev. 2 ?
Stolpna naslovna vrstica štev. 3 ? ----
Ime spremenljivke štev.  2  ($ za niz) :
? izdelek$
Širina stolpca ?  7
Imamo 3 naslovne vrstice za stolpec:

Stolpna naslovna vrstica štev. 1 ? opis
Stolpna naslovna vrstica štev. 2 ? živali
Stolpna naslovna vrstica štev. 3 ? -------
Ime spremenljivke štev.  3  ($ za niz) :
? cena
Širina stolpca ?  8
Število mest za decimalno vejico ? 2
Imamo 3 naslovne vrstice za stolpec:

Stolpna naslovna vrstica štev. 1 ? cena
Stolpna naslovna vrstica štev. 2 ? enote
Stolpna naslovna vrstica štev. 3 ? --------
Ime spremenljivke štev.  4  ($ za niz) :
? cel.cena
Širina stolpca ?  10
Število mest za decimalno vejico ? 2
Imamo 3 naslovne vrstice za stolpec:

Stolpna naslovna vrstica štev. 1 ? skupna
Stolpna naslovna vrstica štev. 2 ? cena
Stolpna naslovna vrstica štev. 3 ? ----------
Število presledkov med stolpci ?  4


PRINT
PRINT tab(5);"kom.";
PRINT tab(14);"opis";
PRINT tab(26);"cena";
PRINT tab(38);"skupna";
PRINT
PRINT tab(7);"";
PRINT tab(13);"živali";
PRINT tab(25);"enote";
PRINT tab(39);"cena";
PRINT
PRINT tab(5);"----";
PRINT tab(13);"-------";
PRINT tab(24);"--------";
PRINT tab(36);"----------";
PRINT
FOR iX=1 TO mX
  a=kolicina(iX)
  wX= 4: dX=0
  GOSUB 6000
  PRINT tab(9-len(a$));a$;
  a$=izdelek$(iX)
  PRINT tab(20-len(a$));a$;
  a=cena(iX)
  wX= 8: dX=2
  GOSUB 6000
  PRINT tab(32-len(a$));a$;
  a=cel.cena(iX)
  wX= 10: dX=2
  GOSUB 6000
  PRINT tab(46-len(a$));a$;
  PRINT
NEXT iX

Ali želiš shranitev na disk (d/n) ?  d
Ali želiš novo (n) ali obstoječo (o) zbirko ? n
Vstavi ime zbirke ?  azizhod1.bas
Ali želiš končati (d/n) ?  d

A)
```

Lista 11. Leva lista prikazuje generiranje izhodnega uporabniškega segmenta za primer ponudbe, ko se bodo v izhodni tabeli pojavile vse tri vhodne spremenljivke (kolicina, izdelek$, cena) in še dodatna izhodna spremenljivka cel.cena, ki bo produkt količine in cene. Po vprašanju o številu spremenljivk (4) sledi za vsako spremenljivko vpraševanje o njeni širini stolpca, o številu decimalnih mest za decimalno vejico v njenih vrednostih ter o treh naslovnih vrsticah spremenljivčnih stolpcev v tabeli. Ko smo za vse 4 spremenljivke odgovorili na postavljena vprašanja, se na zaslonu izpiše generirani program. Ta program ima natanko izračunane pomike za ustrezno oblikovane tabelne stolpce, v katere se bodo vpisovale spremenljivčne vrednosti. Subrutina 6000 bo napisana ročno (glej kasneje). Na koncu dialoga sledita še standardni vprašanji o shranitvi generiranega programa in o končanju generiranja.

```
A)crun2 obukaz1

CRUN VER 2.07P

Koliko vrstic bo približno ?  10
Vpiši 'CTL-u' za končanje
Na vprašanja odgovarjaj z d/n ali z D/N

Vpiši vrstico  1

Vpiši vrstico  2

Vpiši vrstico  3
Račun je plačljiv v 15 dneh.
Vpiši vrstico  4

Vpiši vrstico  5
Hvala za Vaše zanimanje.
Vpiši vrstico  6

Vpiši vrstico  7
P. P. Premočrtnik, komercialni direktor
Vpiši vrstico  8

Vpiši vrstico  9
. . . . . .
Vpiši vrstico  10



Račun je plačljiv v 15 dneh.

Hvala za Vaše zanimanje.

P. P. Premočrtnik, komercialni direktor

. . . . . .

Ali želiš spremeniti vrstico ?  n


PRINT
PRINT
PRINT "Račun je plačljiv v 15 dneh."
PRINT
PRINT "Hvala za Vaše zanimanje."
PRINT
PRINT "P. P. Premočrtnik, komercialni direktor"
PRINT
PRINT ". . . . . ."
Ali želiš shranitev na disk (d/n) ?  d
Ali želiš novo (n) ali obstoječo (o) zbirko ? n
Vstavi ime zbirke ?  azukaz3.bas
Ali želiš končati (d/n) ?  d

A)
```

Lista 12. Gornja lista prikazuje gengriranje programa za končno besedilo ponudbenega primera, ko je bil uporabljen generator OBUKAZ1.

kolicina, izdelek$, cena, cel.cena

Za vsako od teh spremenljivk določimo z dialogom še pripadajočo širino tabelnega stolpca, pri številski spremenljivki še želeno število decimalnih mest za vejico (natančnost prikazovanja) in po tri naslovne vrstice (naslov v tabeli). Na koncu začetnega dialoga določimo še razmak med posameznimi stolpci. Ko smo ta podatek vstavili, se na zaslonu izpiše zgenerirani program, ki začenja v listi 11 s 16 PRINT stavki in se nadaljuje z generirano FOR zanko za vpisovanje vrednosti v tabelne stolpce. Tu se pojavi tudi subrutinski klic 6000; ta subrutina je opisana v listi 13. S končnim dialogom liste 11 zapišemo dobljeni program na disk in končamo generiranje segmenta AZIZHOD1.BAS.

5.5. Oblikovanje segmenta AZUKAZ3.BAS

Lista 12 prikazuje še zadnjega izmed generiranih programskih segmentov, ki se v listi 7 pojavlja kot ukaz %INCLUDE azukaz3.bas. Ta segment zagotavlja izpis zaključnega dela naše ponudbe, tako da je dopis oblikovno ustrezno sklenjen.

5.6. Subrutina 6000

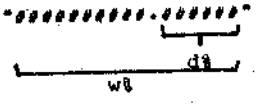
Ta subrutina pretvori številsko vrednost v 'a' v ustrezen format (skladno z vrednostjo d%, ki predstavlja število mest za decimalno vejico). Lista 13 kaže ta program. Najprej se vrednost 'a' pretvori v zaokroženo vrednost z ustreznim številom mest za decimalno vejico, potem pa se pretvorjena vrednost 'a' pretvori še v niz a$. Seveda ta metoda pretvorbe ni najboljša (predvidena je bila za uporabo v šibkejših jezikih tipa Basic). V jeziku CBasic pa bi boljši učinek dosegli z ustrezno spremembo generatorja OBIZHOD3.BAS, ko bi namesto zaporedja

```
GOSUB 6000
PRINT tab(. . . . . .); a$
```

generirali zaporedje

```
PRINT tab(. . . . . .);
PRINT USING format$; a;
```

pri čemer bi bil format$ niz



Ustrezni generacijski segment za PRINT USING stavek pa bi bil tale:

```
1$ = "  PRINT USING " + chr$(34)
FOR k%=1 TO w%(i%)-d%(i%)-1
  1$ = 1$ + "#"
NEXT k%
1$ = 1$ + "."
FOR k%=1 TO d%(i%)
  1$ = 1$ + "#"
NEXT k%
1$ = 1$ + chr$(34) + "; a;"
j% = j% + 1
```

S tem bi tudi odpadla potreba po generiranju vr-stice

```
w% = ... : d% = ...
```

v segmentu AZIZHOD1.BAS.

```
D-------------------------------------------D
D                Subrutina  6000             D
D          'Include' zbirka SUBR1.BAS         D
D-------------------------------------------D

6000
  a = int(a*10Čd%+.5)/(10Čd%)
  a$ = str$(a)
  RETURN
```

Lista 13. Ta lista kaže subrutino za pretvorbo in zaokrožitev številske vrednosti 'a' v niz a$. Zank 'Č' nadomešča znak '^' (potenciranje). Tu je 'a' vrednost številske spremenljivke, d% pa je število mest za decimalno vejico (oziroma piko).

5.7. Ročno dodani segmenti v programu OBPROG.BAS (lista 7)

Program OBPROG.BAS v listi 7 je sestavljen iz določenih (generiranih) segmentov AZ??????.BAS (liste 8 do 12) in SUBR1.BAS (lista 13) in ročno vstavljenih segmentov, ki jih moramo še pojasniti. Segment

```
aa$ = chr$(30) + chr$(26)
PRINT aa$
```

vrne kurzor na začetek zaslona in zbriše zaslon, tako da se začne dialog z uporabniškim programom na zbrisanem zaslonu. Segment

```
%INCLUDE azukaz1.bas
%INCLUDE azvhod1.bas
```

izpiše navodila uporabniškega programa na zaslon in omogoči vstavljanje vrednosti vhodnih spremenljivk. Temu segmentu sledi daljše zaporedje

```
m% = i% - 1
tt = 0
DIM cel.cena(20)
FOR n%=1 TO m%
  cel.cena(n%) = kolicina(n%) * cena(n%)
  tt = tt + cel.cena(n%)
NEXT n%
```

Tu se najprej pripravi ustrezna vrednost m%, inicializira se spremenljivka tt, ki predstavlja skupno (totalno) vsoto, uvede pa se tudi polje cel.cena z 20 elementi. Iz vhodnih vrednosti kolicina in cena se izračunavajo vrstične vrednosti cel.cena in sproti še skupna vsota tt.

Nadaljnji tekst se ne bo izpisoval več na zaslon ampak z vrstičnim tiskalnikom (na papir v obliki dopisa). Segment

```
LPRINTER
%INCLUDE azukaz2.bas
%INCLUDE azizhod1.bas
```

bo izpisal glavo dopisa in izhodno tabelo na papir, nato pa bodo s segmentom

```
a = tt
GOSUB 6000
PRINT
PRINT "  Skupaj"; tab(46-len(a$)); a$
PRINT
```

izpisane še tri vrstice, srednja od njih bo vrstica s skupno vsoto. V tab(46 - len(a$)) smo vrednost 46 vzeli iz liste 11 (zadnja vrednost za tabulacijo proti koncu liste).

```
A>cbas2 oberos
CBASIC COMPILER VER 2.07
    1: ╔═══════════════════════════════════════════════╗
    2: ╠    Primer generiranega uporabniškega         ╣
    3: ╠                 programa                      ╣
    4: ╠═══════════════════════════════════════════════╣
    5: ╠       Ime zbirke: OBPROG.BAS                 ╣
    6: ╠       Imena 'include' zbirk: AZUKAZ1.BAS     ╣
    7: ╠                                AZVHOD1.BAS    ╣
    8: ╠                                AZUKAZ2.BAS    ╣
    9: ╠                                AZIZHOD1.BAS   ╣
   10: ╠                                AZUKAZ3.BAS    ╣
   11: ╠                                SUBR1.BAS      ╣
   12: ╠       april 1984          A. P. Železnikar    ╣
   13: ╚═══════════════════════════════════════════════╝
   14:
   15: aa$ = chr$(30)+chr$(26)
   16: PRINT aa$
   17:
   18: %INCLUDE azukaz1.bas
   19: PRINT
   20= PRINT tab(6);"PROGRAM ZA IZPIS PONUDBE"
   21= PRINT
   22= PRINT "Ta program bo napisal ponudbo ali"
   23= PRINT tab(3);"naročilo za do 20 izdelkov."
   24= PRINT
   25= PRINT "Pri potrditvi na zaslonu vstavi"
   26= PRINT tab(3);"opis izdelka, njegovo ceno in"
   27= PRINT tab(3);"količino."
   28= PRINT
   29= PRINT "Vstavi ime izdelka 'KONEC', ko je"
   30= PRINT tab(3);"seznam končan."
   31: %INCLUDE azvhod1.bas
   32= 20
   33= DIM  izdelek$(20),D
   34=      cena(20),D
   35=      kolicina(20)
   36= i%=1
   37=
   38= 40
   39= PRINT "Vstop stevilka "; i%
   40=
   41= 60
   42= INPUT "Vstavi ime izdelka:"; izdelek$(i%)
   43= IF izdelek$(i%)="KONEC" THEN GOTO 120
   44=
   45= 80
   46= INPUT "Vstavi ceno za enoto izdelka:"; cena(i%)
   47= IF cena(i%)<10 OR D
   48=    cena(i%)>10000 THEN GOTO 80
   49=
   50= 100
   51= INPUT "Vstavi število kosov izdelka:"; kolicina(i%)
   52= IF kolicina(i%)<1 OR D
   53=    kolicina(i%)>200 THEN GOTO 100
   54=
   55= i%=i%+1: GOTO 40
   56=
   57= 120
   58:
   59: m% = i%-1
   60: tt = 0
   61: DIM cel.cena(20)
   62: FOR n%=1 TO m%
   63:   cel.cena(n%)=kolicina(n%)*cena(n%)
   64:   tt=tt+cel.cena(n%)
   65: NEXT n%
   66:
   67: LPRINTER
   68: %INCLUDE azukaz2.bas
   69= PRINT "TOZD Živalska farma, DO Živalstvo"
   70= PRINT "Lisičja cesta 24 - 29"
   71= PRINT "61000  L j u b l j a n a"
   72= PRINT
   73= PRINT tab(16);"DO Nova menažerija"
   74= PRINT tab(16);"Ob potoku 63"
   75= PRINT tab(16);"69113   M a č j i   d o l"
   76= PRINT
   77= PRINT
   78= PRINT "Ljubljana, 22. 4. 1984     Štev. dop. 443-52KL"
   79= PRINT
   80= PRINT "Predmet: Ponudba za dobavo živali"
   81= PRINT
   82: %INCLUDE azizhod1.bas
   83= PRINT
   84= PRINT tab(5);"kom.";
   85= PRINT tab(14);"opis";
   86= PRINT tab(26);"cena";
   87= PRINT tab(38);"skupna";
   88= PRINT
   89= PRINT tab(7);"";
   90= PRINT tab(13);"živali";
   91= PRINT tab(25);"enote";
   92= PRINT tab(39);"cena";
   93= PRINT
   94= PRINT tab(5);"-----";
   95= PRINT tab(13);"--------";
   96= PRINT tab(24);"----------";
   97= PRINT tab(36);"----------";
   98= PRINT
   99= FOR i%=1 TO m%
  100=   a=kolicina(i%)
  101=   w%= 4: d%=0
  102=   GOSUB 6000
  103=   PRINT tab(9-len(a$)); a$;
  104=   a$=izdelek$(i%)
  105=   PRINT tab(20-len(a$)); a$;
  106=   a=cena(i%)
  107=   w%= 8: d%=2
  108=   GOSUB 6000
  109=   PRINT tab(32-len(a$)); a$;
  110=   a=cel.cena(i%)
  111=   w%= 10: d%=2
  112=   GOSUB 6000
  113=   PRINT tab(46-len(a$)); a$;
  114=   PRINT
```

Lista 14. Ta lista kaže izpis programa OBPROG.BAS med prevajanjem, ko se na mestih INCLUDE ukazov z liste 7 vstavijo pripadajoče zbirke (A2??????.BAS in SUBR1.BAS). K INCLUDE zbirki pripadajoče besedilo je za vrstično številko označeno z znakom '=', ali * (pri deklaraciji in pojavitvi nizne spremenljivke). Iz liste 14 je tako mogoče razbrati z INCLUDE ukazi vstavljene segmente in segmente, ki so bili v listi 7 vstavljenkročno (vrstice z znakom ':' za urejevalno številko). Tako zbrana lista omogoča v kritičnih primerih semantičen pregled nad programom kot celoto in njegovo možno popravljanje (modificiranje).

```
  115= NEXT iX
  116=
  117:
  118: a = it
  119: GOSUB 6000
  120: PRINT
  121: PRINT "  Skupaj";tab(46-len(a$));a$
  122: PRINT
  123:
  124: XINCLUDE azukaz3.bas
  125= PRINT
  126= PRINT
  127= PRINT "Račun je plačljiv v 15 dneh."
  128= PRINT
  129= PRINT "Hvala za Vaše zanimanje."
  130= PRINT
  131= PRINT "P. P. Premočrtnik, komercialni direktor"
  132= PRINT
  133= PRINT ". . . . . ."
  134: XINCLUDE subr1.bas
  135= D-----------------------------------------------D
  136= D                 'Subrutina  6000'             D
  137= D           'Include' zbirka SUBR1.BAS          D
  138= D-----------------------------------------------D
  139=
  140= 6000
  141=   a = int(a*10^dX+.5)/(10^dX)
  142=   a$ = str$(a)
  143=   RETURN
  144:
  145: END
NO ERRORS DETECTED
CONSTANT AREA:      16
CODE SIZE:        1445
DATA STMT AREA:      0
VARIABLE AREA:     104
A>
```

Lista 14 (nadaljevanje s prejšnje strani). Na koncu te liste imamo še nekaj sporočil, ki dajejo podatke o številu napak (teh seveda v končno prevedenem programu ni), o številu zlogov v pomnilniškem območju konstant, v ukaznem območju (prevedeni ukazni kod), v območju podatkovnih stavkov in v območju spremenljivk. Iz te liste je razvidno, kako je mogoče vključevati drugačne segmente od obstoječih, npr. ko želimo spremeniti sklepni del ponudbenega dokumenta. Tako bi imeli v vrstici 124: lahko klic nekega drugega segmenta. Podobno bi lahko spremenili tudi strukturo subrutine 6000. Splošno velja, da imajo programirni jeziki z INCLUDE ukazom določeno strukturirno prednost pri generiranju, ker omogočajo enostavno zamenjavo obsežnih segmentov v obstoječih programih in seveda s tem povezano uporabo raznovrstnih generatorjev od primera do primera.

Segment

```
%INCLUDE azukaz3.bas
%INCLUDE subr1.bas
```

izpiše sklepni del ponudbe in vključuje subrutino 6000.

6. Prevod programa OBPROG.BAS

Program OBPROG.BAS z liste 7 je uporabniški program (generirani program, generiranec), ki ga moramo s prevajalnikom za jezik CBasic še prevesti. Prevod programa z liste 7 pa je mogoč pod pogojem, da se na istem disku nahajajo še vsi njegovi INCLUDE segmenti (parcialno generirane zbirke AZ??????.* in zbirka SUBR1.BAS).

Lista 14 prikazuje prevajanje (kompilacijo) programa OBPROG.BAS, in sicer sporočila na zaslon med prevajanjem. Prevedene vrstice programa so urejevalniško označene s številkami 1 do 145; to so dejanske vrstice programa, potem ko so bili vključeni vsi INCLUDE segmenti. Za ustreznim INCLUDE ukazom se pojavi ustrezen program s predznakom '=' ali '*' (nizna spremenljivka). Lista 14 prikazuje tako nazorno celoten (podroben) program OBPROG.BAS. Na koncu liste imamo še sporočilo o številu napak (ki jih seveda ni) in o številu zlogov za pomnilna območja konstant, ukazov, podatkovnih stavkov in spremenljivk.

7. Izvajanje programa OBPROG.BAS

Lista 15 kaže zaslonski, lista 16 pa tiskalni del izvajanja generiranega programa OBPROG.BAS, potem ko je bil ta program preveden. V listi 16 se na začetku pojavijo na zaslonu pojasnila in navodila za uporabnika, tem pa sledi zaslonski

```
A>crun2 obprog

CRUN VER 2.07P


     PROGRAM ZA IZPIS PONUDBE

Ta program bo napisal ponudbo ali
  naročilo za do 20 izdelkov.

Pri potrditvi na zaslonu vstavi
  opis izdelka, njegovo ceno in
  količino.

Vstavi ime izdelka 'KONEC', ko je
  seznam končan.
Vstop stevilka  1
Vstavi ime izdelka: pes
Vstavi ceno za enoto izdelka: 1250.55
Vstavi število kosov izdelka: 5
Vstop stevilka  2
Vstavi ime izdelka: maček
Vstavi ceno za enoto izdelka: 1150.25
Vstavi število kosov izdelka: 13
Vstop stevilka  3
Vstavi ime izdelka: slonček
Vstavi ceno za enoto izdelka: 999.33
Vstavi število kosov izdelka: 3
Vstop stevilka  4
Vstavi ime izdelka: riba
Vstavi ceno za enoto izdelka: 4332.14
Vstavi število kosov izdelka: 4
Vstop stevilka  5
Vstavi ime izdelka: KONEC

ERROR RG

A>
```

Lista 15. Ta lista prikazuje zaslonski del dialoga pri izvajanju programa OBPROG. Uporabnik vstavlja zahtevane podatke s pomočjo programskih navodil.

```
TOZD Živalska farma, DO Živalstvo
Lisičja cesta 24 - 29
61000  L j u b l j a n a

                DO Nova menažerija
                Ob potoku 63
                69113   M a č j i   d o l


Ljubljana, 22. 4. 1984.     Štev. dop. 443-52KL

Predmet: Ponudba za dobavo živali

    kom.      opis          cena          skupna
              živali        enote         cena
    ----      -------       --------      ----------
       5        pes         1250.55        6252.75
      13      maček         1150.25       14953.25
       3     sloníček        999.33        2997.99
       4       riba         4332.14       17328.56

  Skupaj                                  41532.55


Račun je plačljiv v 15 dneh.

Hvala za Vaše zanimanje.

P. P. Premočrtnik, komercialni direktor

. . . . . .
```

Lista 16. Ta lista prikazuje tiskalni del izvajanja programa OBPROG, potem ko so bili vstavljeni podatki v zaslonskem delu. Lista je dopis, ki bo poslan naročniku ponudbe.

dialog. Za končanje dialoga je potrebno vstaviti določeno ime izdelka, ki je v našem primeru 'KONEC' (pisano z velikimi črkami, ker je bilo tako opredeljeno in se tudi s tem razlikuje od navadnih izdelčnih imen). Z vstavitvijo tega izdelčnega imena se dialog na zaslonu prekine in program začne izpisovati ponudbo s tiskalnikom, ki jo prikazuje lista 16. Lista 16 je lično izpisan dopis. Izvajanje programa se naposled konča, ko se po izpisu pojavi na zaslonu še nebistveno sporočilo o napaki (Error 82).

Slabost programa OBPROG.BAS je njegova subrutina 6000, namesto katere bi bilo smotrneje uvesti PRINT USING stavek.

8. Sklep

V tem članku smo pokazali primere programskih generatorjev in nekatere probleme, ki se pri generiranju pojavljajo. Ti primeri generatorjev so v bistvu primitivne generativne procedure oziroma metaprocedure, kot so bile obravnavane v delih ((2, 3)).

Programski generator bi lahko imel nekatere lastnosti metaprocedure ((3)), njegova sintaksna definicija pa bi seveda morala ostati v okviru možnosti izražave v določenem programirnem jeziku. Zanimiva bi bila zgradba metageneratorja, ki bi generiral različne generatorje, ti pa raznovrstne uporabniške programe. Seveda je izdelava metageneratorja brez nadaljnega mogoča in tudi višje metagenerativne ravnine so dosegljive.

Če ugotovimo npr., da je pisanje uporabniških programov z generatorji desetkrat cenejše od pisanja teh programov v visokem programirnem jeziku, potem velja ta pospeševalni princip tudi za pisanje generatorjev: z metageneratorjem napišemo generator desetkrat ceneje kot z neposrednim programiranjem generatorja v visokem jeziku itd. itd. Ko imamo v določenem trenutku na voljo dovolj obsežno knjižnico generatorjev in generiranih programov (generatorskih in uporabniških), lahko postane razvoj novih uporabniških programov in generatorjev cenen, hiter in zanesljiv: tako se izognemu tudi velikim programirnim skupinam (kadrovski problem) in dolgim razvojnim časom (projektni problem).

Slovstvo

((1)) J. R. Jacobs: Generating Programs Automatically. Byte, December 1981, str. 352 - 362.

((2)) A. P. Železnikar: Uvod v formalne informacijske sisteme. Automatika (1975), št. 1-2, str. 3 - 7.

((3)) A. P. Železnikar: On the Definition of Metaprocedure. Informatica 75 (Zbornik del). Bled, oktober 1975, str. 4.2/1 - 4.

INFORMATICA 3/1984

ALGORITHMS FOR FAST B/D CONVERSION OF INTEGERS

I. MARIĆ

UDK: 681.3.05:510.5

INSTITUT „RUDER BOŠKOVIĆ"

The process of converting binary integers with variable lengths into decimal BCD coded integers is considered. Four different conversion algorithms, like successive divisions by ten, successive divisions by powers of ten and conversions by means of look-up tables containing the binary values of the corresponding decimal digits or BCD digit weighting factors, are compared and analysed. The algorithms are realized on the PDP 11/03-L microcomputer and the perameters like algorithms execution times and program lengths are measured and analysed. The results of the measurements are presented.

ALGORITMI ZA BRZU B/D KONVERZIJU CIJELIH BROJEVA - U radu se razmatra postupak pretvorbe binarnih cijelih brojeva različite duljine u dekadske BCD kodirane cijele brojeve. Uspoređivana su i analizirana četiri različita algoritma pretvorbe: uzastopno dijeljenje s deset, uzastopno dijeljenje s potencijama od deset, te pretvorba pomoću tablica preslikavanja koje sadrže ili binarne vrijednosti odgovarajuće dekadske znamenke ili vrijednosti težinskih faktora BCD znamenke. Mjerene su i analizirane brzine izvođenja algoritama i duljine programa te prikazani rezultati mjerenja. Algoritmi su realizirani na laboratorijskom mikroračunalu PDP 11/03-L.

INTRODUCTION

Binary to decimal (B/D) and decimal to binary (D/B) conversions are present in the most of digital data processings on computers as special cases of the more general base conversion process. The well known fact is that $\log_b a$ as many digits is required in representing an integer number after conversion from base a into base b. The bases 2 and 10 are incommensurable bases i.e. $10^i \neq 2^j$, where i and j are nonzero integers. Thus, conversion of binary into decimal number and vice versa is not possible by means of series of shiftings, but the corresponding arithmetic operations are necessary instead. In cases where the large amount of input or output data are needed, the conversion time might take a great deal of total processing time. In this paper some different software realizations of the conversion algorithms are presented, and some advantages and disadvantages with regard to the program lengths and algorithm execution times are discussed and analysed.

We have restricted our attention to the pure software realizations only. Certain conclusions might also be applicable to the hardware considerations. There are several different ways of the B/D conversion on the computers, but the most favourable one fairly depends upon particular computer instruction set and its application. Generally, two different computers do not necessarily have the same optimum conversion algorithm.

CONVERSION ALGORITHMS

Two different approaches to the conversion process are considered: conversion by means of look-up tables and conversion by means of successive divisions. The conversions by means of the look-up tables, as composition of shifting and subtraction operations, are particularly suited for hardware solutions, because of the simplicity and efficiency of the realizations. As it will be shown, this does not mean to be true in the software considerations, especially on

the computers having a hardware implemented division instruction.

a. B/D conversions by means of look-up tables

The decimal integer can be represented as:

$$D = \sum_{i=0}^{n-1} d_i \cdot 10^i \qquad (1)$$

where n represents the number of decimal digits, d_i is the value of the i-th decimal digit and 10^i is the corresponding weighting factor of the i-th digit. The values 10^i in binary representation, where i=1,...n-1, are stored in the look-up table.

B/D conversion is performed by successive subtractions of the current number 10^i from the binary number to be converted, beginning from the largest value 10^{n-1} toward the smallest number 10, each time incrementing the value of the current decimal digit until the result of the subtraction becomes negative. One step before the negative result of the subtraction is obtained, the calculation of the current digit is over. Then the next less significant decimal digit is calculated. The least significant decimal digit is the final result of the series of shiftings and subtractions and it is attached to the previously calculated digits. This conversion requires m-1 binary values of 10^i to be stored in the look-up table where

$$m = \left\lceil N \cdot \log_{10} 2 \right\rceil \qquad (2)$$

and N is the integer denoting the binary number length in bits.

If the decimal digits are represented in the BCD code, the decimal number can be written in the form:

$$D = \sum_{i=0}^{m-1} \sum_{j=0}^{3} k \cdot 2^j \cdot 10^i \qquad (3)$$

where m is the total number of decimal digits after conversion, 2^j represents the weighting factors (1,2,4, or 8) of the corresponding BCD digit and k can be either 0 or 1 depending upon the presence of the corresponding weighting factor in the digit. The look-up table contains the values $2^j \cdot 10^i$ (j=0,...,3, i=1,...,m-1) in the binary representation. The conversion is performed by means of successive subtractions of the stored numbers from the binary number which has to be converted. If the result after subtraction is positive, the subtraction operation will be carried on or halted if negative. In the case of positive result, showing the presence of the corresponding weighting factor in the decimal digit, one is attached to the resulting number at the position of the least significant bit and zero if negative. If the calculation is not over the resulting number is then shifted left once. At the end of calculation the resulting number is shifted left four times and the least significant digit is added to the result. The conversion program needs (m-1) binary values of $2^j \cdot 10^i$ stored in the look-up table where

$$m = \left\lceil 4(N \cdot \log_{10} 2 - 1) \right\rceil \qquad (4)$$

and N denotes the binary length in bits.

b. B/D conversions by successive divisions

If the integer division is hardware realized by the instruction of reasonable execution time, it is possible to construct the B/D conversion with comparable or even better execution times than those using look-up tables. The result of an integer division is integer quotient and integer reminder. If the division by ten is performed, the reminder represents decimal digit. The decimal number can be written in the following way:

$$D = d_0 + 10(d_1 + 10(d_2 + \ldots + 10 \cdot d_{m-1})\ldots) \qquad (5)$$

where $d_0,\ldots,d_{m-1}$ are corresponding decimal digits. Obviously, the reminder after the first integer division by 10 is the least significant digit d_0. The next more significant digit d_1 is the reminder after the second division, etc. The software realization of the conversion algorithm is very simple. The use of the unsigned integer division instruction will be by far more advantageous for the algorithm execution time, than the use of the signed division instruction. In order to avoid overflows at signed divisions by ten, for example, the number must be divided by twenty at first, then the quotient multiplied by two. If the reminder is <10, the division is ended, otherwise 10 is subtracted from the reminder and the quotient is incremented. The unsigned division instruction is particularly suited for such applications, because no overflow can occur at successive divisions (the reminder is always smaller than the divisor). In this way the conversion by means of successive divisions would be considerably faster if realized by unsigned division instruction.

The conversion process can be accelerated if the divisions by powers of ten are combined. The conversion of the binary into decimal number having at most four decimal digits might be performed by one division by hundred and by two divisions by ten. This is a very favourable manner of B/D conversion, particularly for the large number lengths where the total number of basic divisions, as the time consuming operation, can be significantly reduced. The basic division represents an execution of single instruction. According to that, at first we can divide the binary number by 10^4 in order

to obtain the reminders being $<10^4$, then divide the reminders by 10^2 to form subreminders being $<10^2$ and finally divide the subreminders by 10, where the quotient and the reminder represent two succeeding BCD digits of the resulting decimal number.

In order to emphasize the benefits of the division instruction application in the B/D conversions, the same algorithms were realized by means of the software implemented division instruction.

RESULTS OF MEASUREMENTS - PRESENTATION AND ANYLYSIS

The conversion of variable length binary numbers into the corresponding BCD coded decimal numbers have been realized. Binary number length ranged from 8 to 64 bits in the 8-bit increments, therefore there are 8 measuring points over the entire interval. In order to obtain as much as possible true results of the measurements, a separate program has been realized for each number length rather than a general one. This resulted in different program lengths as well as execution times. Special program for the execution time measurements using the real time line clock has been written. For this reason real execution times are slightly shorter then the presented ones, but this is of no importance in the mutual comparison of the algorithms. The results of the measurement are presented in Table 1.

In designation of the conversion programs "BDxyy" the letters BD denote binary to decimal conversion, the most significant digit x denotes the conversion type and two least significant digits yy show the binary number length in bits. Throughout the text we shall refer to the different conversions as follows:

- BD1 - B/D conversion by means of look-up tables (values 10^i, i=1,...,m-1, where m satisfies (2))
- BD2 - B/D conversion by means of look-up tables (values $2^j \cdot 10^i$, i=1,...,m-1, j=0,...,3 where m satisfies (4)
- BD3 - B/D conversion by means of successive (hardware) divisions by ten
- BD4 - B/D conversion by means of successive (hardware) divisions by powers of ten (10000, 100 and 10)
- BD5 - B/D conversion by means of successive (software) divisions by ten
- BD6 - B/D conversion by means of successive (software) divisions by powers of ten (10000, 100 and 10)

Figure 1 graphically illustrates the conversion program lengths versus binary number length.

From the presented results of the measurements it can be seen that the conversion algorithms have quite different characteristics considering the program length.

B/D conversion type	The longest conversion time /mms/	Average conversion time /mms/	Program length /words/
BD108	355	266.9	27
BD208	275	248.1	30
BD308	245	243.3	16
BD408	243	242.1	15
BD508	445	429.2	35
BD608	325	313.9	44
BD116	1 077	721.0	34
BD216	665	588.9	40
BD316	767	765.5	17
BD416	593	589.8	30
BD516	2 405	2 358.7	30
BD616	1 725	1 682.5	56
BD124	2 341	1 534.0	48
BD224	1 577	1 340.0	84
BD324	2 129	2 094.0	37
BD424	1 065	1 059.0	35
BD524	7 065	6 995.2	64
BD624	3 265	3 185.6	84
BD132	5 018	3 608.0	70
BD232	3 012	2 580.0	111
BD332	2 872	2 827.5	38
BD432	2 105	2 091.8	66
BD532	9 565	9 279.3	61
BD632	5 065	4 990.1	121
BD140	8 775	6 107.4	95
BD240	5 375	4 524.4	191
BD340	5 355	5 300.1	49
BD440	2 865	2 851.5	70
BD540	18 865	18 471.4	75
BD640	7 865	7 777.7	111
BD148	12 521	8 711.0	110
BD248	6 297	5 302.5	216
BD348	6 297	6 167.5	49
BD448	3 961	3 935.4	73
BD548	22 198	21 759.2	71
BD648	10 698	10 555.5	112
BD156	16 725	11 629.0	138
BD256	9 765	8 077.0	322
BD356	9 485	9 388.8	67
BD456	4 605	4 546.3	76
BD556	34 065	33 280.9	90
BD656	12 865	12 703.1	117
BD164	23 305	16 002.1	163
BD264	10 990	9 085.0	360
BD364	11 225	11 150.7	67
BD464	6 315	6 265.0	79
BD564	40 665	39 687.2	88
BD664	17 065	16 826.9	118

Table 1. B/D Conversion times and program lengths

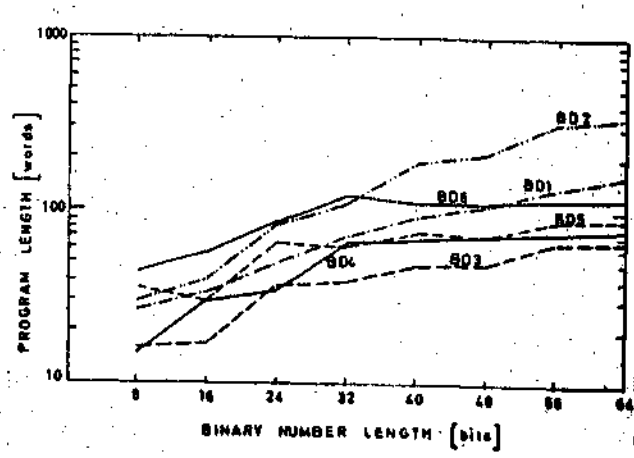


Figure 1. Conversion program lengths

For the very large binary number lengths the BD2 conversion generally needs more program memory than any other conversion because of the lengthy look-up table. According to (4) it takes $\lceil 4(N \cdot \log_{10} 2-1) \rceil$ binary numbers stored in the look-up table.

The BD1 conversion requires about four times shorter look-up table than BD2 conversion (see equation (2)), and, in accordance with that, considerably less program memory. For the very large number lengths it tends to be four times shorter than the BD2 conversion program. The memory used to store the look-up numbers in the BD1 and BD2 conversions, when expressed in the terms of bytes, is $\lceil N \cdot \log_{10} 2-1 \rceil \cdot \lceil \frac{N}{8} \rceil$ and $\lceil 4(N \cdot \log_{10} 2-1 \rceil \cdot \lceil \frac{N}{8} \rceil$, respectively. It is evident that the necessary memory space for the look-up table grows with the power of 2 in both conversions with an increase of the binary number length. Compared to other conversions, the BD3 conversion takes the least amount of program memory space because of the simple realization of the repetitive division by 10. The BD4 conversion includes three consecutive successive divisions by 10000, 100 an 10 and it is slightly more complicated than the BD3 conversion and needs correspondingly more program memory. The BD3 and BD4 conversion programs could also be made a little shorter when realized by the unsigned division instruction.

For a small binary number lengths, the BD6 conversion needs more program memory than any other, because of the software realized divisions by powers of ten. This initial defect disappears in the case of larger number lengths. The BD5 conversion needs considerably less program memory because of the single software realizations of the division by ten. Figure 2 graphically illustrates the average conversion times versus binary number length.

If we suppose the uniform distribution of the decimal digits, then the average number of subtractions of the stored values per each calculated digit is 5.5. Furthermore, one addition per each stored value needs to be done in order to restore the positive value of the number. Therefore the average number of subtractions per converted m-digit BCD number is 5.5(m-1) plus m-1 additions. Additions and subtractions are time consuming operations particularly when applied to lengthy numbers.

The BD2 conversion has 4 compare operations per calculated digit and average number of 1.5 subtractions per calculated digit (1.5 is average number of ones in the BCD digit). In this way the average number of compare nad subtraction operations per converted m-digit BCD number is 4(m-1) and 1.5(m-1), respectively. Generally, compare is faster than the subtraction or addition operation particularly for the lengthy numbers because the result of the compare operation is often known after the single or few most significant words have been compared. That is the reason why the BD2 conversion is considerably faster than the BD1 conversion.

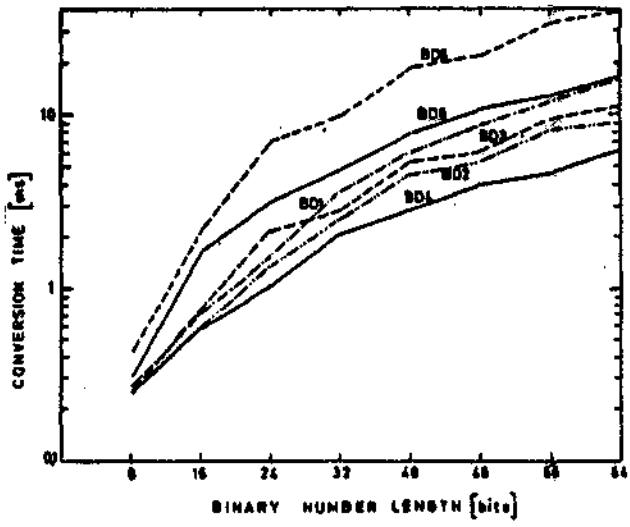


Figure 2. Average conversion times

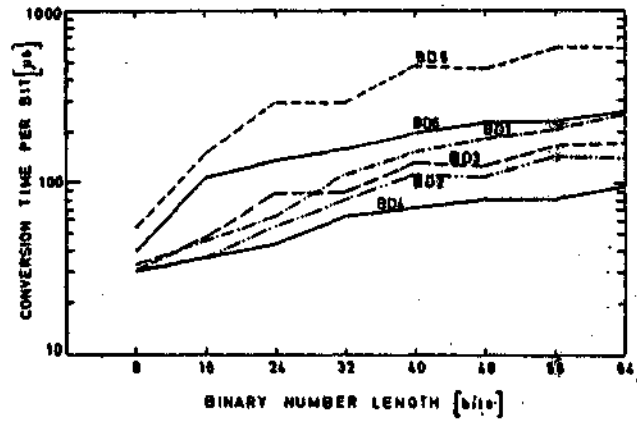


Figure 3. Conversion time per bit

Figure 3 shows conversion time per bit. It can be seen that the conversion time per bit increases when extending the binary number length. The conversion by means of stored powers of ten (BD1) has the largest and the conversion by means of successive divisions by powers of ten (BD4) the smallest increase of the conversion time per bit when increasing the binary number length. From Figure 4 it can

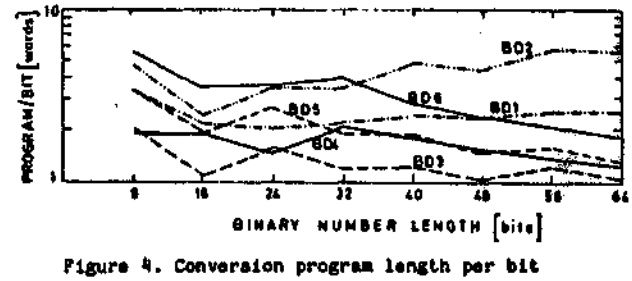


Figure 4. Conversion program length per bit

be seen that the program length per bit in the conversion by means of stored BCD weighting factors (BD2) increases significantly with the increase of the binary number length. Conversion by means of stored powers of ten (BD1) has approximately the same program length per bit regardless of the binary number length while the program

lengths per bit in the conversions by means of successive divisions (BD3) and BD4) are decreasing when increasing the binary number length.

The conversion by means of successive divisions by powers of 10 (BD4) has the shortest conversion time over the entire range. Compared with the conversions by means of successive divisions by ten (BD3), the number of basic divisions is considerably reduced. Let us suppose binary number having N-bit length and the corresponding decimal number having maximum m BCD coded digits, than the total number of the basic divisions by ten in order to convert binary into decimal number (BD3 conversion) is $(m-1)*\left\lfloor\frac{N+15}{16}\right\rfloor$. The average number of the basic divisions by 10000, 100 and 10 in the BD4 conversion are $\left\lfloor\frac{N+15}{16}\right\rfloor$, $\left\lfloor\frac{m-1}{4}\right\rfloor$, $\left\lfloor\frac{m+1}{4}\right\rfloor$ and $\left\lfloor\frac{m}{2}\right\rfloor$, respectively. The average number of the basic divisions in the BD3 and BD4 for the corresponding binary number lengths are shown in Table 2.

Binary number length /bits/	Total number of basic divisions		Maximum number of BCD digits
	BD3	BD4	
8	2	2	3
16	4	4	5
24	14	8	8
32	18	11	10
40	33	15	12
48	42	20	15
56	64	28	17
64	76	31	20

Table 2. Average number of basic divisions in BD3 and BD4 conversion programs

The average number of basic divisions in the BD3 conversion tends to be with the factor of 4 bigger than in the BD4 conversion for the very large number lengths. Owing to the smaller number of the necessary basic divisions, as the time consuming operations, the BD4 conversion has accordingly shorter conversion time than the BD3 conversion. As evident from Table 1 and Table 2, the conversion times are not directly proportional to the number of the basic divisions, because of the influence on the conversion time of the other instruction in the conversion program, and the suitability to write a conversion program for the specified number length. The average numbers of the basic divisions presented in Table 2 are valuable in the general iterative conversion process and can be reduced if separate programs are written for the particular number lengths. During the conversion by means of successive divisions, the number length decreases after each division and the necessary number of the basic division per each succeding division decreases, too. It is possible to write such a general conversion program but it would be sometimes more complicated and usually slower program than the separate programs written for the particular number lengths.

The conversion times in the BD1 and BD2 conversions are fairly dependent upon the values of the binary numbers. The basic division takes approximately the same division time regardless of the converted numbers. Therefore, unlike the BD1 and BD2, the BD3 and BD4 conversions have very small dispersion of execution times. As it can be seen from the presented results of the measurement BD5 and BD6 conversions have the program lengths comparable with other conversions and even better than BD1 and BD2 conversions for very large number lengths, but the worst conversion times over the entire range.

CONCLUSIONS

Since base conversion is frequently used in a digital data computer processings, it is often necessary to increase the conversion speed as much as possible particularly in the real time applications or in the applications where the large amounts of data need to be converted. This paper deals with software implementations of the fast binary to decimal conversion algorithms and presents its advantages and disadvantages taking into consideration the conversion speed and the program length.

It was shown that the conversion by means of successive hardware divisions by ten (BD3) would be the most suitable conversion in the applications where the shortest possible conversion program is needed, regardless of the conversion speed. The conversion by means of successive divisions by powers of ten (BD4), as the most favourable conversion, is particularly suited for the applications where either the short program or the highest possible conversion speed are needed. The conversion by means of look-up tables always take more program memory space than conversions by means of successive hardware divisions. It should be noted that the linear increase of the binary number length increases by power of two the memory used to store the look-up numbers. The conversion by means of stored BCD weighting factors is faster than the conversions by means of stored powers of ten and by successive divisions by ten but they are all slower than the conversion by means of successive hardware divisions by powers of ten. Conversions by means of successive software divisions by ten (BD5) and by powers of ten (BD6) were realized to emphasize the benefits of the hardware divisions only, and are considerably slower than any other conversion presented.

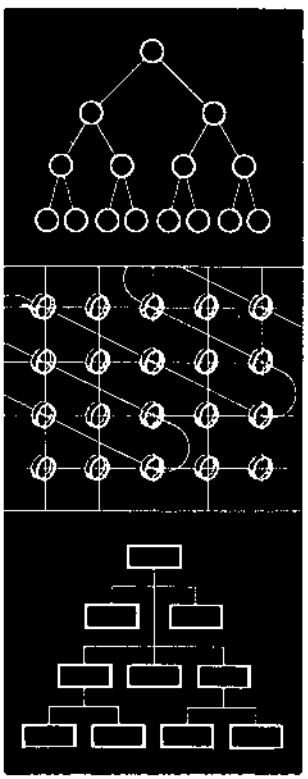
Therefore, the B/D conversion algorithm by means of successive divisions by powers of ten has the best performance when realized on the computers having a hardware division instruction.

REFERENCES

1. David W. Matula: "A Formalization of Floating-Point Numeric Base Conversion", IEEE Transactions on Computers, Vol. C-19, No.8, August 1970, pp. 681-692.
2. I. Marić, L. Cucančić: "On the Possibilities of the BCD Code Application in the Floating-Point Arithmetic Algorithms", Proceedings of the ISMM "MIMI 82" Paris 1982, pp. 8-11
3. Edward L. Braun: "Digital Computer Design", Academic Press Inc., New York 1963.
4. LSI 11 Microcomputer Handbook, Digital Equipment Corporation, Maynard Massachusetts.

ARTIFICIAL INTELLIGENCE

International Summer Seminar
August 27 — September 1, 1984
Dubrovnik, Yugoslávia



PROGRAM AND TOPICS OF THE SEMINAR

MONDAY, August 27,

8.00— 9.00	Registration	
9.00—11.00	Learning	P. H. WINSTON
16.00—19.30	Robotics and assembly-oriented reasoning	T. LOZANO-PEREZ

TUESDAY, August 28

9.00—12.30	Spatial planning	T. LOZANO-PEREZ
16.00—19.30	Learning	P. H. WINSTON

WEDNESDAY, August 29,

9.00—12.30	Machine Learning	R. MICHALSKI
16.00—19.30	Automatic synthesis of expert knowledge in medical applications	I. BRATKO

THURSDAY, August 30,

9.00—12.30	Automatic synthesis based on: a) learning b) qualitative modeling	I. BRATKO
9.00—12.30	Automatic Learning	D. MICHIE

FRIDAY, August 31,

9.00—12.30	Practical aspects of machine learning	D. MICHIE
16.00—19.30	Machine Learning	R. MICHALSKI

SATURDAY, September 1,

9.00—12.30	Automatic Understanding of Natural Language	E. HAICHOVA
13.00	Closing session	

ACCOMMODATION

Due to heavy demand for hotel accommodation in August in Dubrovnik, participants are advised to make hotel reservations in advance. To secure accommodation, the enclosed form should be submitted, together with the indicated deposit, directly to Palace Hotel, Dubrovnik, so as to arrive before July 1, 1984. After this date accommodation cannot be guaranteed.

SPONSORED BY

— "Ivo Lola Ribar", Machine Industry, Belgrade
— The Institute for Nuclear Sciences, "Boris Kidrič", Vinča

TIME AND PLACE

The Seminar will be held from Monday, August 27 through Saturday, September 1, 1984, at Palace Hotel, Dubrovnik, Yugoslavia.

ORGANIZED BY

Center for Advanced Studies, Kneza Miloša 9, 11001 Belgrade, Yugoslavia, and ETAN/SVI — Yugoslav Association for AI.

REGISTRATION

The Seminar is open to all interested on completion of the application form and payment of the registration fee of US $ 250. The fee should be made payable to:

BEOBANKA—BEOGRAD—YUGOSLAVIA
60811-620-16-151-25733-421-01062
YUGOSLAV SOCIETY FOR ETAN

With a note: For Seminar Artificial Intelligence

For Yugoslav participants, the amount of 1500 Yugoslav dinars should be transfered to acc No: 60803-678-12115.

INFORMATICA 3/1984

INTERAKTIVNI GENERATOR PROGRAMA – SIRUP

R. MARKOVIĆ

DO ISKRA DELTA

UDK: 378.681.3

Kada je reč o generatorima programa, prvo treba rešiti dilemu:

Da li treba korisniku omogućiti proširenje predviđenih mogućnosti ili se zadržati u "zatvorenom" sistemu generisanja.

Prva varijanta podrazumeva generisanje izvornog koda, zahteva prevođenje programa i po pravilu traži da korisnik poznaje programiranje.

Kod primene druge varijante odmah se dobijaju izvršni programi i korisnik ne mora da poznaje programiranje.

Pri izradi SIRUP-a opredelili smo se za drugu varijantu iz sledećih razloga:

- brže se dolazi do izvršnog programa
- mnogi korisnici obrade podataka ne raspolažu dovoljnim brojem profesionalno obučenih programerskih kadrova
- greške su svedene na minimum, pošto se radi sa već gotovim i proverenim programima
- paket pokriva veći deo opštih potreba u obradi koje se najčešće pojavljuju
- vecina su retki generatori programa koji ne zahtevaju profesionalno znanje programiranja ili poznavanje operacionog sistema.

Šta je to SIRUP?

Naziv SIRUP je skraćenica od System of Interactive Record Updating and Processing.

SIRUP je skup interaktivnih programa za rukovanje podacima i za generisanje izvršnih programa opšte namene.

Kome je namenjen SIRUP?

SIRUP je namenjen za opštu upotrebu pri rukovanju podacima. Mogu ga koristiti osobe čije osnovno znanje nije vezano za rad sa računarima, ali imaju potrebe za obradom podataka. Zato je koncepcija rada sa SIRUP-om zasnovana na interaktivnom komuniciranju između korisnika i programa. Poruke i pitanja koja se u toku rada pojavljuju na ekranu precizno ukazuju na moguću reakciju korisnika i usmeravaju njegov rad. Ipak, korisnik mora da ima osnovna znanja o podacima u računaru i strukturi tih podataka, ukoliko želi da SIRUP-om pripremi programe. Ukoliko korisnik treba da radi samo sa već pripremljenim programima, onda je dovoljno poznavanje rada sa terminalom i zahteva aplikacije na kojoj radi.

Druga, mnogo češća, grupa korisnika SIRUP-a su osobe koje već rade na poslovima programiranja. Za povećanje produktivnosti u programiranju (čak i preko 10 puta!) mogu se koristiti programi SIRUP-a. Na taj način programerima ostaje više vremena za programiranje specifičnih delova aplikacije čije rešenje nije obuhvaćeno mogućnostima SIRUP-a. Dodatna pogodnost je što se postiže tipizacija u delovima aplikacija i standardizacija kako u postupku razvoja same aplikacije, tako i u postupku korišćenja iste. Održavanje aplikacija u ovom delu koji je obuhvaćen SIRUP-om postaje jednostavnije i smanjuje se potreban broj izvršilaca.

Koje ciljeve ima SIRUP?

SIRUP ima sledeće osnovne ciljeve:

1. Da obezbedi korisniku prilikom rada sa podacima one mogućnosti koje nisu obuhvaćene već postojećim sistemskim pomoćnim programima.

2. Da omogući brzu izradu korisničkih programa bez programiranja pa čak bez poznavanja bilo kog programskog jezika.

3. Da korisniku pruži što veći komfor pri radu, pri čemu je osim već navedenih ciljeva dat naglasak na potrebnim kontrolama u radu i potpunim obaveštavanjem na ekranu, tako da je veći deo rada pravilno usmeravan od strane računara.

Šta može SIRUP?

Prilikom rada na obradi podataka veći deo vremena posvećen je pripremi podataka za konačnu obradu. SIRUP tu pruža korisniku sledeće mogućnosti:

1. Formatizovani i kontrolisani unos podataka u sekvencijalne i indeksne datoteke,
2. Formatizovano listanje slogova na ekranu i štampaču.
3. Pretraživanje datoteka; pronalaženje slogova, izmenu podataka i selekciju slogova prema slobodno izabranim kriterijumima.
4. Logičku kontrolu sadržine datoteke sa listom grešaka.
5. Izradu programskih menu-a za formiranje aplikativnih paketa.
6. Izdavanje prateće dokumentacije.
7. Pomoć pri rukovanju sa datotekama.
8. Obradu podataka pomoću korisnikovih formula u kojima mogu da se koriste osnovne računske operacije (+,-,*,/).

Iskorišćenje ovih mogućnosti SIRUP-a realizuje se kroz brzu izradu izvršnih programa odredivanjem parametara za posebne uslove svake aplikacije.

Pri realizaciji nije potrebno da izvršioci raspolažu znanjem programskih jezika, već je dovoljno poznavanje materijala za obradu i zahteva same obrade.

Pri tome je vreme potrebno za početak eksploatacije ovih programa svedeno na najmanju moguću meru (30-60 min. u odnosu na klasično programiranje sa prevodenjem programa koje može trajati nekoliko dana). Sa upotrebom SIRUP-a se smanjuje angažovanje programera i računara i do 90%.

Postupak generisanja izvršnog (binarnog) programa koji zamenjuje prevodenje izvornog koda programa angažuje rad računara manje od 30 sekundi.

Kako radi SIRUP?

Za raznorodne aplikativne namene koriste se isti programi. Specifičnost namene odreduje se parametrima koje korisnik sam zadaje. Parametri se unose u parametarske datoteke ili direktno u izvrčne programe.

Prilikom generisanja korisničkih izvršnih programa parametri se ameštaju u programe i na taj način nastaje program nezavisan od parametarske datoteke i od SIRUP-a.

Programi SIRUP-a su modularno gradeni tako da je zauzimanje memorije optimalno, a nema ograničenja u broju terminala koji ih koriste istovremeno.

Kod rada sa funkcijama koje koriste štampač formiraju se izlazne datoteke koje su jednoznačno imenovane korišćenjem terminalskog broja, tako da nema kolizije kada se sa više terminala istovrmeno koristi štampač.

Prilikom unošenja parametara korisnik privremeno formira parametarsku datoteku iz koje se u fazi generisanja parametri prenose u izvršni binarni program. Parametarske datoteke ne moraju se čuvati posle generisanja, ali kod eventualnih izmena ili dopuna zahteva potrebno je izvršiti korekcije u njima.

Dokumentacija o karaketristikama izradenih programa dobija se u potrebnoj formi iz sadržine parametarskih datoteka. Na taj način dobija se detaljan opis programa sa slikama upotrebljenih ekrana.

Datoteke formirane SIRUP-om i podaci u njima su standardni za odredeni tip računara tako da se mogu koristiti i sa svim programima koji su pisani na nekom od programskih jezika. Takode se SIRUP-ovim programima mogu koristiti svi podaci u datotekama bez obzira na njihovo poreklo.

Pre početka rada sa sistemom SIRUP potrebno je pripremiti uobičajene podatke koji se inače pripremaju pre početka programiranja.

Faza unošenja parametara koja zamenjuje pisanje izvornog koda programa usmerena je kontrolom uzajamnih veza između parametara i objašnjenjima ispisanim na ekranu.

Kako se koristi SIRUP

Da bi se realizovale sve mogućnosti SIRUP-a njegovo korišćenje je organizovano kroz devet osnovnih funkcija:

1. Unos podataka
2. Ispravke podataka
3. Izveštaji
4. Izrada MENU-a
5. Pomoćni programi
6. Kontrola slogova
7. Selekcija slogova
8. Generisanje parametara
9. Izmene parametara

1. Unos podataka

Koristi se za unos podataka u sekvencijalne i indeksne datoteke, te za obradu (računske operacije, formule). Kod unosa se primenjuju sve kontrole odredene parametrima generisanim od strane korisnika.

Unos u istu indeksnu datoteku moguć je sa više terminala istovremeno a u toku rada moguće je duplicirati polja, kao i prenositi podatke iz već postojećih datoteka u sistemu.

Funkcija 1 omogućuje generisanje korisničkog izvršnog programa za unos i obradu podataka. Specifičnosti programa su odredene parametrima iz odgovarajuće parametarske datoteke. Generisani programi su nezavisni od parametarske datoteke i od SIRUP-a.

2. Ispravke podataka

Ova funkcija omogućuje ispravke na slogovima sekvencijalnih i indeksnih datoteka preko maske ekrana uz primenu svih kontrola odredenih parametrima kao u funiciji za unos podataka

Pristup odredenom slogu moguć je preko vrednosti ključa (za indeksne datoteke) ili prema položaju sloga u datoteci (za sekvencijalne datoteke).

Funkcija 2 omogućuje generisanje korisničkih izvršnih programa za izmenu podataka. Njihove specifičnosti su odredene parametrima. Generisani programi su nezavisni od parametarske datoteke i od SIRUP-a.

3. Izveštaji

Funkcija omogućava formatizovano listanje slogova na ekranu i na štampaču. Mogu se prikazivati delovi slogova po izboru, a format prikaza se odreduje parametrima. Omogućen je rad sa ekranom do 132 znaka u redu, a izmedu polja su dozvoljene računske operacije. Za izveštaje na štampaču omogućena je izrada vertikalnih zbirova.

Zaglavlja tabela dobijaju se slobodnim crtanjem na ekranu i mogu zauzimati do 10 redova.

Pristup slogovima u sekvencijalnim datotekama je po položaju, a u indeksnim po ključu.

Pojedina polja mogu se preneti iz 5 dodatnih datoteka u sistemu.

Funkcija 3 omogućuje generisanje korisničkih programa za izradu izveštaja. Njihove specifičnosti su odredena parametrima: Generisani programi su nezavisni od parametara i od SIRUP-a.

4. Izrada MENU-a

Omogućuje generisanje MENU-a pomoću dve maske ekrana. Preko prve maske se slobodno crta izgled ekrana MENU-a a preko druge maske se unose veze MENU-a. Parametri za izgled i veze se automatski unose u parametarsku datoteku MENU-a.

Generisani MENU-i su izvršni programi koji povezuju do 36 različitih programa i procedura dobivenih sa SIRUP-om ili na bilo koji drugi način.

Menu interaktivno preko maske ekrana koju je korisnih kreirao omogućuje izbor pojedinačnog programa ili procedure. Na takav način su programi i procedure povezani u jedinstveni paket.

Broj nivoa za povezivanje različitih MENU-a nije ograničen.

5. Pomoćni programi

Ova funkcija omogućuje štampanje parametara programa i parametara MENU-a.

Pored toga su u njoj obuhvaćene mogućnosti koje inače pruža operativni sistem (rukovanje sa datotekama). SIRUP omogućuje uspešno rukovanje sa datotekama i bez poznavanja operativnog sistema.

Za rad sa datotekama postoje sledeće mogućnosti:

- Definisanje datoteka
- Brisanje datoteka
- Kopiranje datoteka
- Pregled svih datoteka
- Pregled parametarskih datoteka (programa, MENU-a i izveštaja).

6. Kontrola slogova

Ova funkcija služi za pronalaženje onih slogova u sekvencijalnim i indeksnim datotekama, koji ne zadovoljavaju neki od uslova traženih parametrima. Na listi grešaka se dobija podatak o broju sloga u datoteci ili o vrednosti primarnoga ključa, sadržaj polja koje ne zadovoljava kontrole i uslov kontrole koje to polje ne zadovoljava. Time se dobija osnov za rad sa funkcijom 2.

Ova funkcija omogućuje generisanje korisničkih izvršnih programa za kontrolu slogova cije su specifičnosti određene parametrima. Generisani programi su nezavisni od parametarske datoteke i od SIRUP-a.

7. Selekcija slogova

Služi za pretraživanje datoteka, pronalaženje slogova, izmene na slogovima i selekciju određenih slogova sa izdvajanjem u druge datoteke. Kriterijumi za definisanje karakteristika traženih slogova su slobodno izabrani ključevi u području sloga. Ovih ključeva može biti do 12 i dozvoljena su sva preklapanja menju njima.

Na ekranu se dobija tipska formatizovana slika sloga i moguće je pristupiti izmeni do nivoa pojedinačnog znaka. Primena je moguća na indeksne i sekvencijalne datoteke.

8. Generisanje parametara

Koristi se za generisanje parametarskih datoteka potrebnih za rad funkcija 1, 2, 3, i 6. Generisanje parametara se vrši preko pet maski ekrana:

1. Preko prvog ekrana se određuju osnovne karakteristike radne datoteke i datoteka za kontrolu. Rad je usmeravan sa strane računara.

2. Pomoću drugog ekrana se kreira izgled ekrana i to sa slobodnim "crtanjem".

3. Preko trećeg ekrana se unose granične vrednosti intervala na koje se može polje kontrolisati.

4. Preko četvrtog ekrana se određuju karakteristike pojedinačnih polja.

5. Preko petog ekrana se unose formule pomoću kojih će se vršiti određene obrade.

Ukoliko se rad sa ovom funkcijom ne završi regularno ili se iz nekih razloga prekida, moguće je nastaviti generisanje iste parametarske datoteke korišćenjem funkcije 9. U generisanoj parametarskoj datoteci se zapisuje sistemski datum.

9. Ispravke parametara

Koristi se za izmenu i dopunu parametara u parametarskim datotekama preiranim funkcijom 8.

Rad sa ovom funkcijom je sličan radu sa funkcijom broj 8, sve kontrole su sprovedene na isti način. U datoteci se upisuje datum izvršene izmene.

Dijalog između korisnika i računara usmerava rad na taj način što su sprovedene sve potrebne kontrole, a na ekranu su data uputstva kako treba postupati. Kod zahteva gde je odgovor složeniji, korisnik ima na raspolaganju pomoćnu funkciju gde se na ekranu ispisuju detaljna uputstva.

Svi dijalozi su predvideni na više jezika (srpskohrvatski, slovenački, makedonski i engleski), a prelaz sa jednog jezika na drugi je veoma jednostavan i brz (oko 15 sekundi!).

SIRUP radi na sledećim sistemima:

- na računarima iz proizvodnog programa ISKRA DELTA pod operacionim sistemima DELTA/M i DELTA/V
- na svim računarima iz familije PDP 11 sa operacionim sistemom RSX
- na računarima ISKRADATA C 18 i C 19 pod operacionim sistemom ITOS.

Nije potrebno prisustvo ni jednog programskog prevodioca na računaru.

Paket SIRUP je do sada instaliran kod 15 korisnika ISKRA DELTE.

INFORMATICA 3/1984

PARALELNO IZVAJANJE OPRAVIL V VEČPROCESORSKEM SISTEMU I.

B. MIHOLOVILOVIĆ,
P. KOLBEZEN

UDK: 681.519.7

INSTITUT JOŽEF STEFAN

V članku so podane karakteristike večprocesorskih sistemov, ki pomenijo praktično edino rešitev za zadovoljitev zahtev po vse večji procesni moči računalnikov. Istočasno pa se kaže velika potreba za načrtovanje učinkovitih paralelnih algoritmov. Razvoj računalniške tehnologije pogojuje hitro upadanje cene komponentam MP sistemov, s tem pa možnost načrtovanja paralelnih računalnikov z več kot 1000 procesorji.

PARALLEL EXECUTION OF TASKS IN MULTIPROCESSOR SYSTEMS, 1.PART.- In this article the multiprocessor systems are described. MPS offer a natural solution to the ever-increasing demand for computing power. At the same time, their evolution has brought about the need for the development of efficient parallel algorithms. The advances in computer technology have drastically reduced the cost of components, and it is quite conceivable that parallel computers composed of 1000 or more processors.

UVOD

Zahtevane velike procesne moči današnjih računalnikov lahko pridobimo na sistemskem nivoju tako, da povečamo zmogljivost procesorja in omogočimo hkratno izvajanje velikega števila opravil. Pri uvajanju paralelizma je potrebno procese segmentirati na manjša opravila ter poskrbeti za časovno dodeljevanje le-teh več procesorjem. Ne bistveno drugačno, a bolj ceneno rešitev dobimo z uporabo tehnologije mikroprocesorjev; vendar je pri načrtovanju računalniškega sistema težko izvesti direktno ekstrapolacijo od majhnega k večjemu sistemu. Na primer majhne programe lahko hitro in enostavno napišejo tudi neizkušeni programerji. Večji programski sistemi pa zahtevajo mnogo več znanja in izkušenj.

Enoprocesorski sistem lahko razširimo na več procesorjev, katerim dodamo posebne preklopne strukture in pomnilniške elemente, spremenimo operacijski sistem in na ta način zgradimo večprocesorski sistem z dvema ali največ štirimi procesorji.Direktna ekstrapolacija na sistem z bistveno več procesorji ni možna. Načrtovanje teh sistemov je zdaleč zahtevnejše, saj zahteva od načrtovalca ogromno predhodno teoretično osnovo in kasnejše zahtevno preizkušanje in testiranje zgrajenega sistema.

V tem in naslednjih sestavkih bi na kratko odgovorili na naslednja vprašanja: Kaj je večprocesorski sistem, kako ga načrtujemo in gradimo ter kakšne so njegove prednosti pred ostalimi sistemi. Vemo, da VP sistemi sodijo v razred paralelnih računalnikov, paralelno procesiranje pa predstavlja osnovo tehnologije pete generacije računalnikov. Osnovna Von Neumannova struktura v strukturi VP računalnikov predstavlja omejitev, ker ne more zadovoljiti zahtevam računalnikov 5. generacije; med drugim bomo v naslednjem poskušali odgovoriti tudi na dvoje vprašanj ZAKAJ in KDAJ načrtujemo in gradimo VP sisteme.

Kaj je večprocesorski sistem?

Ena od možnih definicij pravi, da je večprocesorski sistem množica neodvisnih procesorjev, ki medseboj komunicirajo preko skupnega pomnilnika, ki je dostopen vsem procesorjem. Ko govorimo o večprocesorskih sistemih moramo strogo ločiti logično in fizikalno strukturo takšnih sistemov. Logična struktura določa, kako je upravljanje sistema razporejeno med posamezne elemente sistema. Ločimo vertikalne in horizontalne VP sisteme.Vertikalni sistem je navadno hierahično grajen, medtem pa ni nujno, da so elementi sistema logično enaki. Pri horizontalnem večprocesorskem sistemu pa zasledimo logično enakost med elementi. Pri vertikalnem sistemu z logično različnimi elementi je vsakemu elementu sistema posebej omogočeno, da deluje v vlogi 'master' enote. Medprocesorske komunikacije vselej potekajo preko master enote, ki poleg koordinacije komuniciranja opravlja tudi inicializacijo celotnega VP sistema. Za koordinacijo med posameznimi enotami v horizontalni organizaciji skrbi takoimenovani plavajoči krmilnik (vsak procesor posebej lahko prevzame vlogo krmilnika medprocesorskih komunikacij). Če primerjamo obe organizaciji, lahko ugotovimo, da je horizontalna organizacija mnogo bolj fleksibilna od vertikalne. Omogoča nareč dinamično razporejanje obremenitev celotnega sistema med posamezne procesorje v sistemu.

Ko govorimo o fizikalni strukturi VP sistemov, mislimo predvsem na medprocesorske komunikacije i topologijo povezav med

procesorji. Prenos podatkov med procesorji lahko poteka preko skupnega pomnilnika ali preko posebne strukture skupnega vodila. V prvem primeru noben procesor nima direktnega dostopa do kateregakoli drugega procesorja; medtem, ko se pri drugi strukturi s programskim preoblikovanjem logične pa tudi fizikalne strukture sistema ustvari komunikacijski kanal med elementi VP sistema. Pri procesiranju velike količine podatkov učinkovitost opisanih fizikalnih struktur močno pade in to predvsem zaradi ozkega grla v Von Neumannovi strukturi procesnih elemenov v VP sistemu.

Načinov, kako lahko medseboj povežemo N elementov VP sistema je veliko. Povezujemo jih tako, da med drugim dosežemo tudi čim večjo ZANESLJIVOST in RAZŠIRLJIVOST celotnega VP sistema. Zanesljiv sistem je grajen tako, da omogoča tvorbo več direktnih poti med dvema elementoma sistema, s tem pa se zmanjša možnost izpada sistema zaradi pojava napak v delovanju posameznih procesnih enot. V razširljivem sistemu lahko osnovni konfiguraciji dodamo več elementov, ne da bi bistveno vplivali na delovanje obstoječe strukture VP sistema. Omenimo naj še dva načina razvrščanja VP sistema; NAČIN VZAJEMNEGA DELOVANJA in NAČIN PROCESIRANJA. Med VP sisteme, ki vzajemno delujejo sodijo sistemi, ki imajo takoimenovane RAHLE odnosno ČVRSTE povezave med procesnimi elementi. Tipični predstavnik rahle povezave je računalniška mreža, predstavnik čvrste povezave pa je večprocesorski sistem.

Predno naštejemo možne načine procesiranja, si oglejmo nekaj lastnosti VP sistemov:

- Skupni pomnilnik je dostopen vsem procesorjem v sistemu; vsakemu procesorju posebaj pa pripada njegov lastni pomnilnik.

- Skupni operacijski sistem upravlja in koordinira vzajemno delovanje med procesorji.

- Vhodno/izhodni in ostali resorsi se dodeljujejo procesorjem, pri čemer nekateri resorsi pripadajo samo nekaterim procesorjem.

- Vsi procesorji v sistemu so lahko simetrično razvrščeni in imajo medsebo enake procesne moči.

- Procesne obremenitve se dinamično razporejajo med procesorje tako, da so vsi procesorji približno enako obremenjeni.

- Vsa pomembnejša opravila se izvajajo samo na enem procesorju.

- Paralelizem in koordinacija izvajanja opravil v VP sistemu. Ta lastnost je najpomembnejša lastnost VP sistemov. Razlikujemo predvsem tri oblike paralelizmov, ki razvrščajo VP sisteme po načinu procesiranja. M. J. Flynn navaja naslednje štiri načine;

1. SISD (Single Instruction Single Data). Sem sodijo klasični računalniki

2. MISD (Multiple Instruction Single Data). Karakteristika teh računalnikov je takoimenovano cevičenje (pipelining); Prednost takšnih računalnikov je v tem, da so povsem programsko transparentni. Njihovi procesorji so precej zahtevni, sami pa nimajo modularne strukture in zahtevajo precej visoko zanesljivost delovanja.

3. SIMD (Single Instruction Multiple Data). Sem sodijo vsi vektorski, array in asociativni procesorji. Značilnost teh sistemov je ta, da imajo eno centralno procesno enoto. Ta dekodira tok in instrukcij in generira množico ukazov, ki jih dodeljuje procesnim elementom tako, da je možna paralelna obdelava podatkov.

4. MIMD (Multiple Instruction Multiple Data) sistem sestavljajo enake ali različne procesne enote, od katerih že vsaka posebej predstavlja računalnik.

Zgoraj naštete lastnosti VP sistema zagotavljajo določene prednosti le-tega pred ostalimi sistemi, ima pa tudi določene slabosti.Med največje slabosti štejemo možnost, da se lahko v sistemu pojavljajo konfliktne situacije. Te nastopajo v primeru, ko več procesorjev posega v skupni pomnilnik. Zaradi konfliktnih situacij je število procesorjev v sistemu omejno.Z uvajanjem posebnih preklopnih struktur med procesorjem in pomnilnikom pa je možno verjetnost pojava takšnih nezaželjenih situacij občutno zmanjšati. Teoretično lahko zgradimo VP sistem poljubnih mikroprocesorjev, ki enako uspešno opravlja delo računalnika z enim hitrim procesorjem. Mikroprocesorji so lahko celo različni med seboj. Ozko grlo med procesorjem in pomnilnikom v osnovnih modulih VP sistema dopušča gradnjo zgolj takšnega VP sistema, v katerem si večje število medsebojno podobnih, dokaj neodvisnih procesov izmenjuje majhno število podatkov.

Rahla in čvrsta povezanost med procesnimi elementi v VP sistemu sta značilni za dva skrajna razreda organizacij VP sistemov. Seveda pa so možne organizacije, v katerih so prisotne tudi kombinacije obeh povezav. Omembe vredna je vsekakor organizacija DIMS. Kratica pomeni "Distributed Intelligence Microcomputer Systems". Je torej multimikroračunalniški sistem, v katerem je delo razdeljeno na specifična opravila. Ta se izvajajo popolnoma neodvisno na različnih procesnih elementih sistema; medtem, ko eden od procesorjev opravlja končno zbiranje, procesiranje in prezentacijo rezultatov predhodno procesiranih opravil.

Kratko naštejmo še nekaj pomembnejših karakteristik multiračunalniških sistemov:

-Avtonomna zgradba elementov. Vsak element zase vsebuje CPE, pomnilnik z lokalnim programom, podatkovni pomnilnik; Element lahko krmili tudi računalniško periferijo.

- Materialna in programska oprema vsakega procesnega elementa je načrtovana za opravljanje specifičnih opravil.

- Medprocesorska komunikacija poteka na nivoju podatkov, ki predstavljajo ukaze in zahteve.

- Vsak procesni element izvaja tako vhodno/izhodne funkcije, kot funkcije sistemske komunikacije. Funkciji sta lahko porazdeljeni na posamezne procesne elemente.

-Ker so funkcije procesorjev vnaprej določene ni možno izvesti dinamičnega razporejanja opravil.

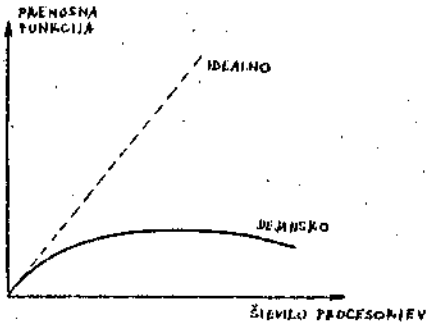
Zakaj večprocesorski sistem?

Že pri načrtovanju in kasneje pri delovanju VP sistema opazujemo tri osnovne sistemske parametre, ki jih tudi merimo. Ti parametri so: procesna sposobnost, zanesljivost, načrtovanje in razvoj sistema. Našteti parametri so med seboj tesno povezani.

Procesno sposobnost sistema ocenjujemo na osnovi meritev treh parametrov: razmerja cena/učinkovitost, prenosne funkcije, dodeljevanja resorsov.

Vemo, da učinkovitost procesorja raste s ceno. Pri enoprocesorskem sistemu velja Grosch-ovo pravilo, ki pravi, da je učinkovitost procesorja proporcionalna kvadratu cene. Torej je ekonomsko povsem neopravičljivo graditi MPS s procesorji velike procesne moči in pri tem pričakovati, da bomo pridobili veliko procesno sposobnost celotnega sistema. Veliko boljše rezultate dobimo z uporabo mikroprocesorske tehnologije. Teoretično velja, da dosežemo optimalno učinkovitost celotnega sistema kar s seštevanjem učinkovitosti posameznih procesorjev. No, dejansko je tako pridobljena učinkovitost mnogo manjša, saj smo pri ocenjevanju učinkovitosti pozabili na visoko ceno programske opreme in na dodatne elemente materialne opreme (konektorji, tiskanine, itd.) katerih cena ne pada tako hitro kot pada cena procesorskim in pomnilniškim elementom.

Prenosna funkcija sistema je definirana kot recipročna vrednost časa, ki je potreben, da se izvrši dana množica algoritmov. Prenosna funkcija je tudi najboljše merilo učinkovitosti sistema. Idealno je, da prenosna funkcija narašča proporcionalno številu dodatnih procesorjev VPS sistema. Dejansko obliko prenosne funkcije pa prikazuje slika 1.



slika 1

Na upognjenost funkcije upliva pojav nasičenja, ki si ga razložimo na ta način, da se z naraščanjem števila procesorjev povečuje tudi število resorsov, ki se delijo med procesorje. Gostota informacijskega pretoka v medprocesorskih komunikacijah na ta način močno naraste. Prav zato želimo, da je prenosna funkcija čim bližja linearni obliki, delo sistema (job) pa razdeljeno v množico manjših čimbolj neodvisnih opravil (tasks), katerih izvajanje zahteva manj medprocesorskih komunikacij.

Zanesljivost sistema. Znana je definicija, po kateri je zanesljivost v delovanju sistema enaka pogojni verjetnosti, pravilnega delovanja sistema v času 0→t, če je le-ta pravilno deloval že v času t=0. Seveda je v primeru večprocesorskega sistema taka definicija zanesljivosti preveč posplošena. Pri izvajanju enega opravila navadno sodeluje več procesorjev, vsi pa delujejo pod različnimi pogoji. Poleg tega ločimo redundantne in neredundantne sisteme. Pri redundantnih sistemih podvojenost delujočih komponent omogoča zanesljivejše delovanje sistema kot celote. Zanesljivost delovanja redundantnega sistema (ki je v bistvu večprocesorski sistem) določata zanesljivost delovanja posameznih modulov in izbira modela takoimenovane Fault- tolerant sheme.

FT sistemi so sposobni, da odklanjajo vse morebitne napake v delovanju materialne in/ali programske opreme brez posredovanja človeka. Da takšen sistem deluje, mora vsebovati ali sistemsko ali selektivno redundanco. V sistemu s sistemsko redundanco delujejo vse enote hkrati in stem varujejo celoten sistem pred morebitnim izpadom; če se pojavi v eni od enot napaka, je z ispadom le-te sistem vselej zaščiten z drugo delujočo enoto. V sistemu s selektivno redundanco posebna procedura v realnem času odkriva nedelujočo enoto in preostalemu delujočemu delu sistema priklopi novo delujočo enoto. Večprocesorski sistemi sodijo v razred sistemov manj občutljivih na napake (fault tolerant systems). Stopnjo neobčutljivosti na napake ocenjujemo pri VP sistemih s takoimenovano zmožnostjo prenosa odgovornosti za izvajanje opravil iz enega elementa na drug element sistema, ne da bi se spremenila učinkovitost celotnega sistema. Če je VP sistem takšen, da ima omejene procesne zmožnosti, pravimo da je sistem napakovno mehak sistem.

Namesto zaključka

V svetu je zgrajenih razmeroma malo VP sistemov in še ti so grajeni namensko za reševanje točno določenih problemov. Kot primer naj navedemo tipični multiračunalniški sistem, ki je zgrajen na C. M. univerzi C.mmp. Namenjen je reševanju problemov na področju umetne inteligence (računalniški govor in vid) in različnim numeričnim problemom. Operacijski sistem Hydra omogoča, da je računalnik uporabljiv tudi kot več uporabniški sistem.

Upravičenost načrtovanja VP sistemov iščemo v neglede na povečano zanesljivost takšnega sistema vsekakor v paralelizmih, ki so pogojeni že z samo aplikacijo. Omenjeni paralelizmi se kažejo na tri načine:

- Več neodvisnih uporabnikov sistema z veliko bazo podatkov kot so sistemi rezervacij, bančni sistemi in podobno.

- Potencialni paralelizmi. V programih, ki so pisani za enoprocesorske sisteme, so često možni paralelizmi; če je njihovo število omejeno (ne več kot 10), je smiselno, da takšni programi tečejo na VP sistemu.

- Paralelna dekompozicija. Veliko je računalniških aplikacij, katerih programi so dekomponirani zaradi bolj učinkovitega izvrševanja celotnega programa; kot primer vzemimo probleme na področju umetne inteligence, ki so polni različnih paralelizmov. Bolj je primerno, da takšni programi tečejo na VP računalniku namesto, da oblikujemo serijo opravil, ki bi povsem neučinkovito potekala na enoprocesorskem sistemu. Rekli smo že, da je v kompleksnejših aplikacijah smiselno večje računalniško delo razdeliti v manjša opravila in le-ta dodeliti procesorjem tako, da je med njimi kar najmanj komunikacij, ki potekajo na nivoju podatkov.

Določitev razmerja cena/učinkovitost je pri VP sistemih precej zapletena. Kljub temu pa vemo, da na velikost tega razmerja uplivajo trije parametri: Razvoj sistema, modularnost, podaljšanje življenske dobe sistema. Čas, ki je potreben za razvoj VP sistema je vsekakor daljši od časa, ki je potreben za uvajanje sistema v katerikoli aplikacijo.

Tako lahko sistemu, ki že deluje, razmeroma preprosto dodajamo procesne enote z namenom, da dosežemo maksimalno sposobnost sistema, kar pa je možno le, če je celoten sistem modularno grajen iz enostavnejših materialnih in programskih modulov. Moduli so optimizirani za specifične opravila, ki jih opravljajo v sistemu. Največjo zmogljivost enoprocesorskega sistema kaj hitro dosežemo za razmeroma visoko ceno, medtem ko VP sistem s svojo modularno zgradbo omogoča nadgradnjo sistema in s tem večjo zmogljivost za bistveno manjšo ceno.Hkrati pa modularna nadgradnja pogojuje tudi daljšo življenjsko dobo sistema.

Literatura

1. Eli T.Fathi, Moshe Krieger,
Multiple Microprocessor Systems: What, Why, and When; Coeputer, March 1983

2. Arvind, R. Iannucci,
A Critique of Multiprocessing von Neumann Style; Communications of ACM, April 1983

3. G. Gardarin
Design of a Multiprocessor relational Database System; Information processing 83, IFIP, 1983

POPRAVEK IZ PREJŠNJE ŠTEVILKE
ČASOPISA INFORMATICA

Uredništvo in tiskarna časopisa
se upravičujeta za napako, ki je nastala
pri zamenjavi naslovov
člankov avtorja M. Kukrika
na straneh 24 in 60,
Informatica 8, 1984 - št. 2.

V kazalu številke sta naslova pravilna!

INFORMATICA 4/1984

PARALELNO IZVAJANJE OPRAVIL V VEČPROCESORSKEM SISTEMU II.

B. MIHOLIVILOVIĆ,
P. KOLBEZEN

UDK: 681.519.7

INSTITUT JOŽEF STEFAN

V članku so podane bistvene smernice za načrtovanje paralelnih algoritmov. Obravnavane so lastnosti paralelnih algoritmov in relacije med temi algoritmi in paralelnimi računalniškimi strukturami. Posebej so opisane lastnosti in nekateri paralelni algoritmi za MIMD multiprocesorske sisteme.

PARALLEL EXECUTION OF TASKS IN MULTIPROCESSOR SYSTEMS, 2.PART.-The purpose of this article is to create a general framework for the study of parallel algorithms. A taxonomy of parallel algorithms, based on their relations to parallel computer architectures, is introduced. Examples of parallel algorithms for MIMD multiprocessors are given.

Uvod

Po načinu procesiranja, kot je Flynn razdelil vse večprocesorske sisteme, sodijo takoimenovani asinhroni večprocesorski sistemi (VPS) med MIMD računalnike. Takšen VPS večkrat imenujemo tudi multiračunalniški sistem. Osnovni elementi takšnih sistemov so največkrat standardni računalniki (LSI 11, PDP 11 in pod.). V članku ne bomo govorili o podrobni zgradbi nekega asinhronega multiračunalniškega sistema (AMRS), temveč se bomo oprli na definicijo asinhronih VP sistemov, ki smo jo obravnavali že v prvem delu istoimenskega članka. VP sistem je množica neodvisnih procesorjev, vsak procesor izvaja lasten program, med seboj pa komunicirajo največkrat preko skupnega pomnilnika. V primeru, da so procesne enote med seboj različne, poseganje v skupni pomnilnik ni uniformno. To pa je vzrok, da je poleg asinhronega delovanja prisotna še dodatna možnost pojava kaotičnega stanja v delovanju VP sistema. V svetu obstaja več primerov asinhronih multiračunalniških sistemov, medtem ko je načrtovanje paralelnih algoritmov, ki bi bili učinkoviti v takšnih sistemih dandanes še slabo raziskano.

Namesto, da načrtujemo vedno nove algoritme, lahko paralelne algoritme načrtujemo s pomočjo že preizkušenih sekvenčnih algoritmov. V tem primeru govorimo o takoimenovani transformaciji sekvenčnih algoritmov, iskanju notranjih paralelizmov in neodvisnih opravil znotraj programa ter medsebojne odvisnosti med opravili. Na ta način zgrajen paralelni program lahko izvaja različna opravila po predhodno zadanem grafu relacij med opravili. Poleg tega morajo biti paralelni algoritmi, ki so namenjeni AMRS takšni, da je čas, ki je potreben za komunikacijo med posameznimi računalniki manjši od časa, ki ga potrebuje posamezen računalnik za izvršitev opravila.

Načrtovanje algoritma

Pravimo, da je paralelni algoritem množica sodelujočih asinhronih procesov, ki komunicirajo med seboj preko globalnih spremenljivk [1]. Algoritmi, ki so namenjeni SIMD računalnikom so podobni algoritmom grajenim za MIMD računalnike. V obeh primerih je prisotna dekompozicija problema v opravila, ki se izvajajo sočasno. Pri SIMD računalnikih srečujemo takoimenovane sinhrone koračne algoritme, ki zahtevajo centralni nadzor, pri MIMD računalnikih pa asinhrone algoritme, ki so, kot pravimo močno "razdrobljeni".

V definiciji paralelnega algoritma je podčrtana beseda proces. Proces je osnovna enota dela, ki ga nedeljivo opravi nek proces v večračunalniškem sistemu. Določen je z segmentom programa, ki se lahko razporeja med procesorje. Paralelni algoritem skrbi za neposredni nadzor nad procesi. Delovanje procesorjev nadzira operacijski sistem, ki s pomočjo paralelnega algoritma nadzira izvajanje in dodeljevanje procesov enotam znotraj sistema. Na splošno so lahko procesorji med seboj različni. Program se lahko izvaja na več procesorjih hkrati. Pri tem se lahko eneki procesi odvijajo sočasno na posameznih računalniških enotah sistema. Komunikacije med procesi potekajo preko globalnih spremenljivk (preoblikovanje podatkov v skupnem pomnilniku) tako, da se rezultati procesiranja oblikujejo brez zakasnitev in so odvisni od sprotnih vrednosti globalnih spremenljivk.

Pri študiju odnosno načrtovanju paralelnih algoritmov nas zanimata predvsem dve merili za ocenjevanje algoritmov. To sta pravilnost in učinkovitost algoritmov. Od vsakega algoritma zahtevamo, da je pravilen, vendar so različni postopki preverjanja pravilnosti zelo zahtevni. Primer enega od načinov testiranja

pravilnosti je ta, da v programu verificiramo takšne globalne spremenljivke, ki po izvajanju programa dobijo pričakovano vrednost.

Učinkovitost je pomembna karakteristika paralelnih algoritmov. V njej se odraža večja hitrost izvajanja paralelnih algoritmov napram sekvenčnim algoritmom pri enaki zanesljivosti pravilnega izvajanja in rabi prostora.

Sočasni sistem osnove podatkov

S tem izrazom poimenujemo takšno osnovo podatkov, v katero sočasno posegajo številni procesi. Eden od pomembnih kriterijev, ki določajo sočasno izvajanje procesov, predstavlja takoimenovana strnjenost podatkov. Rečemo, da so posegi v osnovo podatkov pravilni, če se po vsakem posegu strnjenost podatkov ohranja. Pri sočasnih procesih obstaja namreč možnost, da se nekateri podatki nezaželjeno spremene, s čemer ni več zajamčeno pravilno izvajanje programa. Strnjenost podatkov je tako porušena. Mehanizem, ki v sočasnem sistemu osnove podatkov ohranja strnjenost podatkov imenujemo sočasni nadzor (concurrency control).

Do danes sta v veljavi nekako dva pristopa k reševanju problema ohranjanja strnjenosti podatkov. Prvi sloni na metodi zaporednosti, ki ne zahteva poznavanje strnjenosti v osnovi podatkov, temveč poznavanje sintakse posameznih posegov v osnovo podatkov. Drug pristop sloni na vnaprejšnjem poznavanju strnjenosti podatkov z namenom, da se zgradi učinkovit sočasni sistema osnove podatkov.

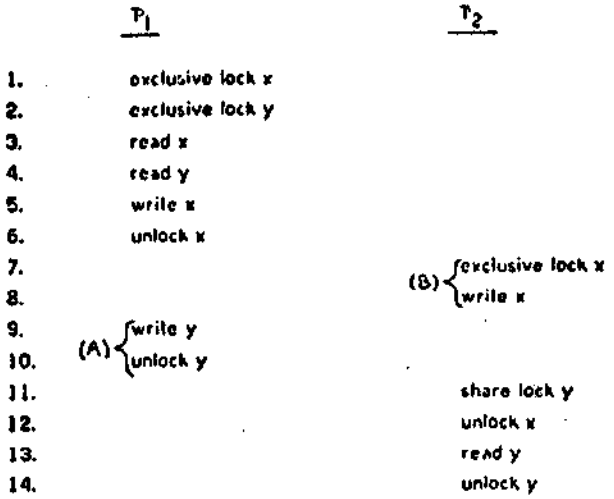
1. Metoda zaporednosti

Metoda zaporednosti, temelji na predpostavki, da je sočasno procesiranje osnove podatkov enako učinkovito, kot zaporedno procesiranje. Pri tem se mora strnjenost podatkov ohranjati. Poseben sinhronizacijski protokol dopušča le pravilne posege v osnovo podatkov in omogoča posebna stanja, in sicer blokiranje odnosno sprostitev osnove podatkov nad katerimi se izvaja določen proces. Govorimo tudi o dvofaznosti procesov. Rečemo, da je proces dvofazen, če ne posega v osnovo podatkov brez posebnega predhodnega posega. Vsak poseg v osnovo podatkov vsebuje torej dve fazi: daljšo fazo, v kateri se omogoči dostop in zajemanje podatkov, in krajšo fazo, v kateri se dostop do podatkov blokira, podatki pa obdelajo .

Proces razvrščanja (schedule) v množici sočasnih procesov se izvede že pred izvajanjem programa. Razvrstitev je lahko popolna ali le delna. Delna urejenost se lahko kasneje modificira glede na VP sistem, na katerem se procesi odvijajo. Serijsko razvrščanje pomeni v bistvu serijsko poseganje v osnovo podatkov.

V splošnem pravimo, da je proces razvrščanja legalen, če ne vsebuje posegov, ki bi lahko blokirali tiste podatke, po katerih sega tudi drug proces. Na ta način lahko pride sistem v konfliktno stanje. Na sliki 1 je prikazan primer legalnega razvrščanja dveh dvofaznih procesov P1 in P2.

Na sliki 1 vidimo, da sta akciji A in B v procesu P1 oz. P2 povsem neodvisni, saj uporabljata neodvisni spremenljivki. Akcija A se lahko izvaja pred akcijo B ali obratno. To



Slika 1

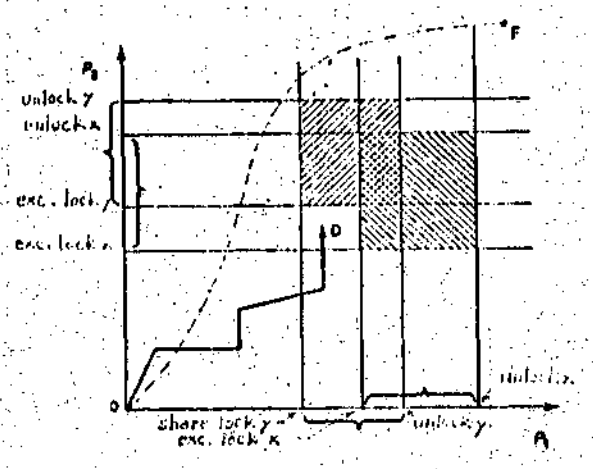
pomeni, da se razvrščanje sočasnih procesov v ničemer ne razlikuje od serijskega razvrščanja, po katerem bi se izvedel najprej proces P1 nato pa proces P2. Iz tega sledi naslednji izrek:
Katerokoli legalno razvrščanje dvofaznih procesov je ekvivalentno serijskemu razvrščanju le-teh.

Predno podrobneje proučimo ta izrek, si oglejmo nekaj novih pojmov. Graf soodvisnosti razvrščanj je direkten graf, katerega vozlišča označujejo procese (posegi v osnovo podatkov), povezave med vozlišči pa označujejo medsebojno odvisnost procesov. Tak graf opisuje stanje osnove podatkov pri vseh procesih, ki so se izvajali po grafu razvrščanj. Pravimo, da sta dve razvrščanji ekvivalentni, če imata enak graf. Vsekakor lahko sočasno izvajanje dvofaznih procesov privede sistem v dead-lock stanje. V tem primeru je potrebno (potem, ko je dead-lock stanje detektirano) proces, ki je povzročil konfliktno stanje ponoviti. Ker zgornji izrek zagotavlja veljavnost dvofaznih procesov, ponovitev enega ali drugega procesa ne povzroča izgube podatkov.

Bolj nazorno predstavo o stanju sistema v primeru dveh sočasnih procesov (posegov v osnovo podatkov) dobimo, če uporabimo enostavne grafične metode. Na sliki 2 je prikazana grafična ponazoritev primera iz slike 1.

Katerakoli trajektorija v tem "dvo-dimenzijskem prostoru" ponazarja sočasno izvajanje dveh procesov. S točko F je označeno stanje, ko sta oba procesa zaključena. Krivuljo med točkama O in F imenujemo tudi "progresijska krivulja", ki loči celotno ravnino v dva dela. V nekem trenutku se lahko zgodi, da progresijska krivulja vstopi v okvirjeno območje (D), kjer obstaja velika verjetnost pojava dead-lock stanja. Posebno točko predstavlja vstop trajektorije v območje D, kar pomeni, da bosta procesa zanesljivo prišla v konfliktno situacijo. Torej je mogoče doseči preprečitev dead-lock stanja z preusmeritvijo teka trajektorije nekaj korakov pred območjem D. Detekcijske algoritme, ki bi na ta način reševali problem konfliktnih situacij je veliko. Vsem takšnim rešitvam je skupno to, da zahtevajo poznavanje večih ali vsaj enega parametra, ki "kaže v prihodnost".

Pri znanem BANKER algoritmu je ta parameter maksimalna velikost vsakega vira, v katerega bo proces posegal v času procesiranja. Maksimalna velikost vira označuje tudi



Slika 2

upravičenost procesa do vira. Praktično je definicija tega parametra povsem realna, saj vnaprejšnja določitev maksimalne vrednosti virov sodi v zadetno specifikacijo procesov.

Algoritem oblikuje poseg procesa v določen vir samo pod pogojem, da:

- je vsota vseh zahtev po virih, vključno s tekočimi posegi, manjša kot je upravičenost procesa do virov, in da

- obstaja niz procesov, ki se bodo zanesljivo izvedli pod pogojem, da posegajo po maksimalni upravičenosti virov.

V skrajnem primeru se lahko procesi izvajajo posamično drug za drugim, to pa pomeni, da takšno zaporedno izvajanje algoritma zahteva precej časa. Če imamo npr. sistem z m viri in n procesi, potem algoritem "pregleduje", če je VS sistem varen pred dead-lockom. Algoritem pregleduje upravičenost m virov dokler ne najde proces, ki se bo lahko izvedel do konca. Pri tem pregleda vseh n procesov.

2. Semantika procesov

Opisana metoda zaporednosti ne zahteva poznavanja semantike procesov, odnosno posegov v osnovo podatkov. Zato je ta metoda ostala edini možni pristop k reševanju problemov ohranjanja strnjenosti osnove podatkov v sočasnih sistemih. Če pa sta že vnaprej poznani celotna strnjenost podatkov in posegi v osnovo podatkov, lahko gradimo sočasne sisteme ali algoritme z največjo stopnjo sočasnega izvajanja procesov. Vendar tudi v takšnem primeru predstavlja osnovo gradnje "sočasnega sistema" ugotavljanje pravilnosti paralelnega algoritma. Če je ugotavljanje pravilnosti izvzeto iz posameznih procesov, pravimo temu potrjevanje algoritmov.

Paralelne algoritme potrjujemo s pomočjo trditev, ki veljajo na določenih mestih algoritma. Vsak poseg v osnovo podatkov predstavimo z diagramom poteka. Vozlišča označujejo stavke, povezave med vozlišči pa ponazarjajo trditve. Trditve so zapisane v obliki predikatnih izrazov, (ki poleg logičnih izrazov vsebujejo še kvantifikatorje univerzalnosti in eksistence). Pravilnost algoritma dokazujemo tako, da vsakemu stavku pripišemo dva predikatna izraza. Prvi izraz, imenovan vzrok, stoji pred stavkom, drugi, imenovan posledica pa takoj za stavkom. Posledica vpliva na vzrok naslednjega stavka, kar povezuje med seboj trditve vseh posegov v osnovo podatkov. Pravimo, da je posamezen poseg pravilen, če se za vsak stavek trditve ohranjajo.

Strnjenost osnove podatkov pri sočasnem izvajanju procesov pa zagotavlja veljavnost naslednjega pravila razdeljevanja [2]: Vsak naslednji korak katerega koli posega v osnovo podatkov se izvaja le pod pogojem, da poseg ne razveljavi trditve, veljavne za stavek, ki istočasno pripada tudi drugim posegom.

Lahko nastopi primer, da noben od posegov ne omogoča izvajanja naslednjega koraka. To pomeni obstoj dead-lock stanja, čemur enostavno sledi ponavljanje posegov v osnovo podatkov.

Zaključek

V članku smo obravnavali paralelne algoritme za asinhrone večprocesorske sisteme. Težo problema predstavlja potrjevanje pravilnosti in učinkovitosti algoritmov zaradi asinhronosti procesov. V večprocesorskih sistemih, kjer so nam znane zgolj sintaktične lastnosti osnov podatkov, lahko zagotovimo pravilen algoritem le z metodo zaporednosti. Če pa je v osnovah podatkov podana tudi semantika podatkov, ki obsega strnjenosti podatkov in informacijo o posegih procesov v osnovo podatkov, se ponuja možnost za načrtovanje veliko bolj učinkovitih algoritmov. Le-ti podpirajo kar največjo stopnjo sočasnega izvajanja.

Analiza učinkovitosti paralelnih algoritmov za večprocesorske sisteme je precej zahtevna, saj čas izvajanja posameznih procesov (bolje opravil) ni konstanta, temveč naključna spremenljivka. Zato je v analizi učinkovitosti potrebna raba modelov, ki slonijo na statističnih metodah.

Na koncu ponovno poudarimo, da so algoritmi, ki so grajeni za asinhrone večprocesorske sisteme modularni. Večja številčnost modulov pa občutno zmanjšuje nepotrebno čakanje v medprocesorski komunikaciji.

Literatura

[1] Kuck D.,
The structure of Computers and Computations, John Wiley and Sons, New York, 1987.

[2] Lamport L.,
Proving the correctness of Multiprocess Programs IEEE Transactions on Software Engineering, March 1977.

[3] Gerard M. Baudet
The Design and Analysis of Algorithms for Asynchronous Multiprocessors
Carnegie Mellon u. Report, April 1978.

INFORMATICA 3/1984

PRISTUP KREIRANJU RASPOREDJIVAČA ZADATAKA U DISTRIBUIRANOM IZVRŠNOM SISTEMU SA RADOM U STVARNOM VREMENU

M. KUKRIKA

ELEKTROTEHNIČKI FAKULTET BANJALUKA

UDK: 681.3.06

SAŽETAK - Problematika izbora optimalne arhitekture raspodijeljenog sistema nije dovoljno istražena, a mnoga važna pitanja su ostala bez odgovora. Kada je ovaj problem riješen trebalo bi definirati pristup raspodjeli zadataka po računalima, te redoslijed njihovog izvodjenja u svakom od računala. Ako se postavlja zahtjev da se u stvarnom vremenu izvede n zadataka na m procesora pri čemu je n mnogo veće od m (n >> m), tokom izvodjenja bi često trebalo odgovoriti na pitanje kojem od pripravnih zadataka pridijeliti procesor. Tu zadaću, uz ostale upravljačke aktivnosti preuzima jezgro raspodijeljenog izvršnog sistema. U ovom radu rasporedjivač je definiran kao skup identičnih algoritama, od kojih se svaki izvodi na vlastitom računalu. Pretpostavlja se da paralelni procesi tokom svog izvodjenja često izmjenjuju poruke, te stoga mijenjaju i svoja stanja iz pripravan u blokiran i obrnuto.

ABSTRACT - AN APPROACH TO DESIGN OF TASK SCHEDULER IN A REAL-TIME DISTRIBUTED OPERATING SYSTEM. The research problems associated with the optimal architectures of highly parallel structures are difficult and less well understood. However, when the system's architecture has been determined, other significant problems are allocation of tasks and scheduling of processors. If it is requiered to execute in real time a n number of processes on a m number of processors (n >> m), a frequent question which ready process is to be assigned to processor is apparent and to be answered while computing. It is the task of the operating system kernel to provide the necessary management operations. Scheduler described in this paper consists of a set of replicated algorithms, so each computer can run its own local scheduler. It is assumed that the parallel processes reflect a strong cooperation: message exchange with other processes is quite frequent, and as a consequence the processes will continously change the execution state (ready <-> blocked).

1. UVOD

Zahtjevi za ostvarivanjem stvarnovremenog rada dovode nas do problema podjele vremena i mjesta korištenja sredstava za izvodjenje pojedinih zadataka, jer nam je zbog povećanja brzine odgovora, cilj da se što više zadataka izvodi uporedo. Procesor sekvencijalno sprovodi obradu definiranu programom, pa se izvodjenje skupa zadataka može povjeriti ili moćnom procesoru, čija će brzina ostaviti dojam paralelnog izvodjenja, ili pak zadatke razdijeliti na više računala. O podjeli vremena dakle, govorimo ako se izvodjenje skupa zadataka povjeri moćnom procesoru, a o podjeli mjesta ako zadatke raspodijelimo na više računala.

U uvjetima kada se traži što hitniji odgovor na pobude, multiprogramiranje omogućava brže vrijeme odgovora, jer se izvodjenje zadatka može prekinuti nakon bilo kojeg koraka, te nastaviti kasnije s jednoznačnim konačnim rezultatom uz jedini uvjet da se u medjuvremenu sačuva prethodno stanje prekinutog zadatka. Na taj način se u računalu može paralelno odvijati nesmetano više zadataka, ako se osigura da oni jedan drugom ne mijenjaju stanja, tj. da se svaki od njih odvija na vlastitom skupu registara i memorijskih lokacija. Ovakvu razdvojenost najlakše je osigurati višeprocesorskom gradjom, te se višeprocesorski i raspodijeljeni sistemi mogu smatrati proširenjem monoprocesorskih multiprogramskih sistema kod kojih virtualna mašina postaje stvarna.

Činjenica da se svaki zadatak izvodi na vlastitom računalu trebalo bi da omogući znatno ubrzanje rada i na prvi pogled se čini da su svi problemi time riješeni. Medjutim, oni tek započinju:

Mnogi zadaci koje bi raspodijeljeni sistem trebalo da rješava ne mogu se pretvoriti u čisto paralelne algoritme. Potpunu uporedivost u izvodjenju zadataka je veoma teško postići ako su djelovanja u sistemu medjuovisna, a to je upravo karakteristika sistema koji nas primarno zanimaju. Problematika analiziranja i specificiranja svojstava paralelizma u skupu zadataka razmatrana je detaljnije u (1).

Drugi problem je odredjivanje topologije raspodijeljenog sistema. Izbor načina povezivanja računala diktiran je zahtjevima koji se na sistem postavljaju (vrijeme odziva, fleksibilnost itd...), a zasniva se na kompromisu zahtjeva pouzdanosti i cijene. Optimalna je ona topologija koja će postavljene zahtjeve ostvariti uz najmanju cijenu (2).

Raspodijeljeni sistemu u odnosu na jednoprocesorske posjeduju još jednu klasu sredstava koje rasporedjuje izvršni sistem, a to su procesori. Odluka o rasporedjivanju zadataka po računalima, kao i odredjivanje redoslijeda njihovog izvršavanja u svakom od računala može bitno utjecati na performanse raspodijeljenog sistema. Tu zadaću, uz ostale upravljačke aktivnosti preuzima raspodijeljeni izvršni sistem.

2. RASPODIJELJENI IZVRŠNI SISTEMI

Osnovna zadaća pri kreiranju sistemske programske podrške je ostvarivanje najbolje moguće raspodjele postojećih sredstava, kako bi se što više zadataka moglo uporedo izvoditi. Uzme li se u obzir da u raspodijeljenim sistemima postoji mnogostruki izbor da li se upravljački zadaci odvijaju uporedo ili jedan za drugim, tada postaje razumljivo koliko su kompliciranl problemi i moguće konstelacije sa kojima se suočava planiranje vremenske strukture raspodijeljenih izvršnih sistema.

U literaturi postoji više primjera realiziranih izvršnih sistema za višeračunarske strukture kao što su izvršni sistem Roscoe za Aracne system (9), Medusa za Cm*(10), Staros za Cm* (11), Mike za EDLON (12) itd.

Prema (5,6,7) raspodijeljeni izvršni sistem može se realizirati kao nadgradnja na standardni, konvencionalni izvršni sistem, ili razradom koncepcije jezgra. U radu (8) je predstavljen prvi pristup - realiziran je Cocanet Unix - mrežni operativni sistem za računala tipa VAX i PDP/11 povezana u prstenastu mrežu.

Postojeće izvršne sisteme je često teško prilagoditi, pa se njima donekle mora podrediti organizacija cijelog sistema. Stoga se izlaz može potražiti u razradi koncepcije jezgra (5,6).

Koncepcija jezgra sastoji se u tome da se u nepreki divom načinu rada obave samo najnužnije akcije, i da se, što je moguće prije dozvoli prekidanje. Minimiziranje dijela sistemske programske podrške koji ne smije biti prekinut je od velikog značaja za rad u stvarnom vremenu.

Kod sistema sa radom u stvarnom vremenu pojedine aktivnosti (zadaci) odvijaju se pod uticajem okoline i njihov redoslijed se ne može u potpunosti unaprijed predskazati. Zadaci u sistemu moraju stoga biti povezani na takav način da ispravno suradjuju u obavljanju predviđenog posla u svim dinamičkim uvjetima koje nameće okolina. Stoga je za optimizaciju rada raspodijeljenih sistema od najveće važnosti pristup raspodjeli zadataka.

Da bi se učinio pristupačnijim problem rasporedjivanja zadataka se obično razmatra na nekoliko nivoa apstrakcije. Ovdje izloženo gledište o rasporedjivanju razlikuje dva glavna nivoa:

- na nižem nivou, koji se može nazvati lokalnim rasporedjivanjem, odlučuje se koji zadatak iz repa pripravnih zadataka u pojedinom računalu nastaviti;

- na višem nivou, koji se može nazvati globalnim rasporedjivanjem, odlučuje se koje od računala će preuzeti na sebe kreiranje, iniciranje i izvršavanje najnovijeg zadatka kojeg bi hitno trebalo izvesti.

Izvoljenje zadatka koji vrši odredjena izračunavanja preuzeće u kriznim situacijama ono računalo koje je sa sistemskog stanovišta u tom trenutku najpodobnije. Time se po principu spojenih posuda ukupno opterećenje rasporedjuje jednoliko na sve učesnike, a jednoliko opterećen sistem je i sistem maksimalne pouzdanosti i raspoloživosti.

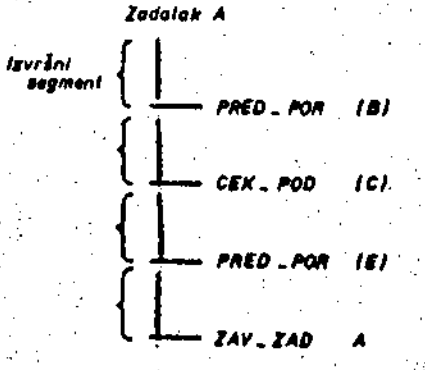
Svako od računala u sistemu posjeduje vlastitu kopiju sistema zadataka, te je sposobno, na osnovu odluke globalnog rasporedjivača preuzeti dio zadataka drugog opterećenog računala.

U (3) i (4) su predstavljena dva deterministička algoritma za rasporedjivanje zadataka po računalima, a u ovom radu je definiran algoritam za odredjivanje redoslijeda izvodjenja pridijeljenih zadataka u svakom od računala Za razumijevanje ovog algoritma potrebno je poznavati model zadatka:

3. MODEL ZADATKA

Raspodijeljeni programi su programi koji se sastoje od dva ili više zadataka koji medjusobno komuniciraju isključivo izmjenom poruka (14,15,16). Zadaci unutar raspodijeljenog programa ne dijele nikakve varijable, svaka varijabla pripada odredjenom (lokalnom) zadatku. Upravljanje u raspodijeljenom programu je takodjer raspodijeljeno, nema centralnog zadatka koji bi upravljao akcijama pojedinih zadataka ili usmjeravao poruke. Pri tome bi trebalo težiti da se što više zadataka izvršava uporedo, te da se medjuzavisnim zadacima omogući da izmjenjuju poruke.

Prema slici 1 strukturu zadatka predstavljamo kao slijed izvršnih segmenata izmedju kojih su umetnuti pozivi jezgru za predajom (PRED-POR) i prijemom (CEK-POD) poruka, te za završenjem zadatka (ZAV-ZAD). Nakon toga zadatak se završava.



Sl.1. Redoslijed izvodjenja zadatka

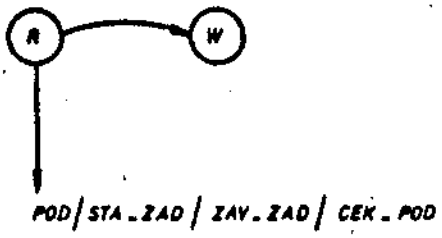
Zadatak koji je aktiviran u nekom od računala može se naći u slijedećim stanjima:

- R - izvodi se, ili je pripravan za izvodjenje;
- W - čeka na podatke koje će mu poslati ostali zadaci (tj. na akcije koje su potpuno neovisne o njemu),

ili na potvrdu o primitku poruke koju je poslao drugim zadacima.

Zadatak prelazi iz stanja R u stanje W u trenutku kada struktura programa diktira prijem ili predaju poruke. Prijemom poruke ili potvrde za predanu poruku zadatak iz stanja W prelazi u stanje R i uvrštava se u rep pripravnih zadataka.

Na slici 2. je prikazano u kojim slučajevima zadatak prelazi iz stanja R u stanje W, a na slici 3. u kojim slučajevima iz stanja W u stanje R.

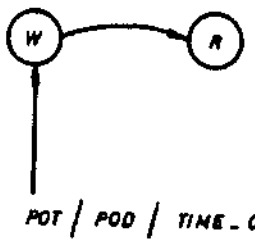


Sl.2. Prelazak zadatka iz stanja R u stanje W

Pri tome skraćenice imaju slijedeća značenja:

U slučaju prelaza R - > W :

POD - zadatak je predao poruku i čekaće na potvrdu
STA-ZAD - trebalo bi startati zadatak višeg prioriteta
ZAV-ZAD - završen je neki drugi zadatak
CEK-POD - zadatak se ne može nastaviti dok ne primi odgovarajuće podatke



Sl.3. Prelazak zadatka iz stanja W u stanje R.

U slučaju prelaza W - > R

POT - stigla je potvrda predane poruke
POD - stigli su podaci na osnovu poruke CEK-POD
TIME-OUT - isteklo je vrijeme predvidjeno za odgovor i postupak se ponavlja

4. ODREĐIVANJE REDOSLIJEDA IZVOĐENJA ZADATAKA

Algoritam za raspoređivanje je odgovoran za donošenje odluke kojem od zadataka iz repa pripravnih zadataka bi, u suglasnosti sa zahtjevima rada u stvarnom vremenu, trebalo dodijeliti procesor. Primjena poznatih strategija raspoređivanja, kao što su FCFS, RR i druge, (a što se čini u većini postojećih raspodijeljenih sistema), mogla bi negativno utjecati na performanse sistema. Naime, ovi pristupi ne uzimaju eksplicitno u obzir međuovisnosti zadataka koji se izvode na različitim mjestima u sistemu.

Raspoređivač, u skladu sa usvojenim komunikacijskim i sinhronizacijskim mehanizmima manipulira sa aktivnim zadacima, ažurirajući tri liste:

- listu pripravnih zadataka;
- listu zadataka koji čekaju na prijem poruka;
- listu zadataka koji čekaju na prijem potvrde o predanoj poruci.

Prema komunikacijskim i sinhronizacijskim mehanizmima koji su izloženi u (16) svaki od zadataka mijenja svoje stanje iz W u R i obrnuto, u zavisnosto o broju poruka koje bi trebalo izmijeniti sa zadacima sa kojima komunicira. U isto vrijeme računalu se dodjeljuju novi zadaci na osnovu odluka globalnog raspoređivača. Zadaci koji se izvode u radnom računalu se prema prethodnom stanju u kojem su se nalazili mogu podijeliti u tri skupine:

1) Pripravni za izvođenje sa prethodnim stanjem čekaj - podatke

2) Pripravni za izvođenje sa prethodnim stanjem čekaj - potvrdu

3) Pripravni za izvođenje sa prethodnim stanjem nije ni počeo sa izvođenjem.

Za sve zadatke koji su prešli u stanje Pripravni za izvođenje na osnovu primljenih podataka egzistiraju komunikacijski partneri koji su u stanju čekaj na potvrdu, iako bi možda trenutno opterećenje njihovog računala objektivno omogućavalo njihovo izvođenje. Stoga se preporuča da se najveći prioritet izvođenja dodijeli zadacima iz prve skupine, jer se time potiče da se njihovi komunikacijski partneri uzmu u obzir pri dodjeli procesora čim prije je to moguće. Sa ovakvim pristupom postiglo bi se zadovoljavajuće vrijeme odziva cjelokupnog sistema.

S druge strane ovakav pristup daje znatnu prednost zadacima koji su već izvođeni, u odnosu na novopridružene zadatke sa čijim izvođenjem bi tek trebalo početi. Ekstremni slučaj je kada se procesor dodjeljuje zadatku iz druge skupine tek kada više nema zadataka iz prve skupine u repu pripravnih zadataka, te zadacima iz treće skupine ako nema pripravnih zadataka ni u prvoj ni u drugoj skupini.

U predloženom rješenju, koje je u nastavku algoritamski predstavljen, teži se ka izbjegavanju diskriminacije nekih od zadataka u repu. To se postiže smanjivanjem vrijednosti parametara prioriteta pridruženog uz svaku skupinu zadataka, kad god se izvodi neki od zadataka iz

te skupine.

Zbog zahtjeva rada u stvarnom vremenu redoslijed zadataka u svakoj skupini se odredjuje prema hitnosti njihovog izvodjenja. Nadalje se unutar jednakopravnih zadataka izbor može izvršiti na osnovu:

- najdužeg vremena čekanja;
- najvećeg broja poruka koje bi trebalo izmijeniti;
- najkraćeg vremena izvršavanja;
- nekog drugog pristupa.

Pri izboru zadatka kojem će biti dodijeljen procesor mora se voditi računa i o hitnosti izvodjenja zadataka unutar odredjene skupine. Uvjeti rada u stvarnom vremenu zahtijevaju da se zadatak izvede (dovrši) unutar roka koji je pridružen svakom od zadataka. Time se ujedno nameće i potreba da se ispitivanje hitnosti izvodjenja zadataka provede prije bilo koje druge odluke.

Pretraživanje počinje u skupini sa najnižim prioritetom, (tj. sa zadacima sa čijim izvodjenjem se još nije počelo), nastavlja se sa drugom skupinom i dovršava se prvom. Hitnost zadatka se procjenjuje u ovisnosti o roku izvodjenja pridruženom svakom zadatku, te procjeni vremena potrebnog za dovršenje zadatka. Zadatak iz prve skupine može postati pripravan za izvodjenje ili stoga što njegovo vrijeme čekanja ističe, ili pak zbog toga što je njegov komunikacijski partner hitan zadatak. Informacija o hitnosti dovršenja predajnog zadatka može se dostaviti uz podatke koje PRE_ZAD šalje PRI_ZAD-u. Ako to situacija zahtijeva poruka može sadržati i više informacija o predajnom zadatku kao što su:

- stupanj hitnosti;
- stanje računala i vrijeme čekanja zadatka u sistemu;
- vrijeme izvodjenja zadatka;
- faktor opterećenja predajnog računala;
- ostale informacije.

Unošenje ovakvih informacija unutar poruke može smanjiti broj upravljačkih poruka koje se izmjenjuju medju računalima.

U nastavku je opisani postupak algoritamski predstavljen.

```
Procedure Dodjela _ procesora ()
(* algoritam za lokalno rasporedjivanje zadataka *)

  (* potraga za hitnim zadacima unutar repova *)
    rep (1) = Pripravni_za_izvodjenje sa prethodnim sta-
              njem čekaj_podatke
    rep (2) = Pripravni_za_izvodjenje sa prethodnim sta-
              njem čekaj_potvrdu
    rep (3) = Pripravni_za_izvodjenje sa prethodnim sta-
              njem nije_ni_počeo_sa_izvodjenjem

(* Pronalaženje pripravnih zadataka unutar repova ČEK_POD
   i ČEK_POT kao i koordiniranje ovih repova sa ulaznim
   prihvatnicima za potvrde i poruke *)

FOR I = 1 TO 3 DO
 BEGIN
 (* potraga za hitnim zadacima unutar repova *)
  X : = bilo_koji_hitan (rep(i))
     IF (NOT (X=0)) THEN
         RETURN (START (X));
 END;

(* izbor zadatka kojem će se dodijeliti procesor *)
 (* dodjela prioriteta  repovima zadataka  *)
  (*  k < l < m     *)
   rep (1) : = k;
   rep (2) : = l;
   rep (3) : = m ;

 (* prioriteti se smanjuju svaki put kada se izabere
    zadatak iz pripadnog repa *)

IF star_zad = star_zad - 1 THEN K : = K-1 ;
IF pot_zad  = pot_zad - 1  THEN L : = L-1 ;
IF por_zad  = por_zad - 1  THEN M : = M-1;

 (* potraga za zadacima u repu Pripravni_za_izvodje-
    nje sa prethodnim stanjem čekaj_na_poruku *)

IF (por_zad < 0) THEN
  BEGIN
   IF (NOT (prazan (rep(3))) THEN
     BEGIN
     por_zad = k ;
     RETURN (START (prvi_u_repu (3)));
     (* izabran je najhitniji zadatak jer je rep
        organiziran prema hitnosti izvodjenja *)
     (* Neke od alternativa pri ovom pristupu su:
        START (najduže_vrijeme_čekanja)
        START (najveći_broj_poruka)
        START (najkraće_vrijeme_dovršenja) *)

 (* potraga za zadacima u repu Pripravni_za_izvodje-
    nje sa prethodnim stanjem čekaj_na_potvrdu *)
    IF (pot_zad < 0) THEN
     BEGIN
      IF (NOT (prazan(rep(2))) THEN
       BEGIN
        pot_zad = l;
```

```
      RETURN (START (prvi_u_repu (2)));
(* potraga za zadacima u repu Pripravni_za_izvodjenje
   sa prethodnim stanjem izvodjenje_nije_ni_počelo *)
   If (star_zad < 0) THEN
    BEGIN
     IF (NOT (prazan(rep(1))) THEN
       BEGIN
        star_zad = m ;
          RETURN (START (prvi_u_repu (1))) ;
    END;
   END;
END.
```

4. ZAKLJUČAK

Glavne prednosti raspodijeljenih sistema u odnosu na druge računarske strukture, koji motivišu daljnja istraživanja u oblasti njihovog projektiranja i izgradnje su modularnost, pouzdanost, propusnost sistema, brzina, efikasnost rada, mogućnosti proširenja, raspoloživost, prihvatljiva cijena i druge. To su i osnovni ciljevi koje bi trebalo postići izgradnjom takvog sistema.

Medjutim, prednosti koje uvodi uporedno izvršavanje zadataka u raspodijeljenim sistemima su ograničene složenošću realizacije upravljačkih mehanizama, te definiranjem pristupa komunikaciji i sinhronizaciji medjuzavisnih zadataka koji se izvode na različitim mjestima u sistemu. U sistemima kod kojih je komunikacija zasnovana na izmjeni poruka znatno opterećenje komunikacijske mreže može dovesti do veoma loših osobina sistema.

Od pristupa problemu rasporedjivanja zadataka po računalima ovisiće i da li će se raspodijeljeni sistemi pokazati boljima od multiprogramskih i višeprocesorskih sistema.

Ako se postavlja zahtjev da se u stvarnom vremenu izvode n zadataka na m procesora pri čemu je n mnogo veće od m (n >> m), tokom izvodjenja bi često trebalo odgovoriti na pitanje "Kojem od pripravnih zadataka pridijeliti procesor?". Strategija prema kojoj se donosi ova odluka naziva se algoritam rasporedjivanja. Primjena poznatih strategija rasporedjivanja, kao što su FCFS, RR i druge, (a što se čini u većini postojećih raspodijeljenih sistema), mogla bi negativno utjecati na performanse sistema. Naime, ovi pristupi ne uzimaju eksplicitno u obzir medjuovisnosti zadataka koji se izvode na različitim mjestima u sistemu. Produženo vrijeme odgovora pojedinog računala može sprečavati druga računala da rade sa maksimalnim učinkom. Zbog toga predloženi algoritam za osnovni kriterij pri odlučivanju uzima smanjivanje komunikacijskog opterećenja sistema. Neproduktivno vrijeme potrebno da se donese odluka je veoma maleno i ne ovisi o informacijama koje se dobijaju od ostalih računala. Predloženo rješenje bi stoga trebalo potpomoći ubrzavanje izvršavanja zadataka u drugim računalima, te utjecati na smanjenje ukupnog vremena odziva sistema.

LITERATURA

1. Kukrika,M.: "O mogućnostima uporednog izvršavanja zadataka u višeračunarskim sistemima", I Simpozij o procesnom upravljanju (JUREMA 1984).
2. Kukrika, M.: "Pristup organiziranju lokalnih mreža mikroračunala" V medjunarodni simpozij "Kompjuter na sveučilištu", Cavtat (1983).
3. Kukrika, M.: "Pristup dinamičkom rasporedjivanju zadataka u prstenastoj mreži računala", Informatica 2 (1984).
4. Kukrika, M.: "Neke mogućnosti ravnomjernog korištenja računala u višeračunarskim sistemima sa radom u realnom vremenu", VIII bosanskohercegovački simpozij iz informatike, Jahorina (1984).
5. Clark, D. et al.: "Design of distributed systems supporting local autonomy", Proc. 1980. IEEE COMPCON, (feb. 1980).
6. Preebles, R.: "Adapt: A guest system", Proc.1980 IEEE COMPCON, (feb. 1980).
7. Tanenbaum, A.: "Computer Networks", Prentice-Hall (1981).
8. Rowe, L et al. "A local network based on the UNIX operating system, IEEE trans. soft. eng., vol. SE-8, no. 2. (march 1982).
9. Solomon, M et al.: "The Roscoe distributed operating system", Proceedings of the 7th symposium on operating systems principles, (dec. 1979).
10. Ousterhout, J et al.: "Medusa: An experiment in distributed operating system structure" Comm. ACM vol.23, no. 2, (feb. 1980).
11. Jones, A.: "The object model: A conceptual tool for structuring software", Lecture notes in computer science vol. 60, Springer Verlag (1978).
12. Liu, M.: "Design of a network operating system for the distributed double-loop computer network (DDLCN) Computer networks, North-holland publishing company (1982).
13. Kukrika, M.: "Primjer realizacije jezgra za rad u stvarnom vremenu", VI medjunarodni simpozij "Kompjuter na sveučilištu", Cavtat (1984).
14. CHANDY,K. et al.,: "Distributed simulation: A case study in design and verification of distributed programs, IEEE trans. on Software Engineering, vol. se-5, no. 5, september 1979.
15. Hansen, P. B.: "Network: A Multiprocessor Program", IEEE Tran. on Soft. Eng., May 1978.
16. Kukrika, M.: "Primjer sinhronizacije medjuzavisnih zadataka u raspodijeljenim sistemima", Etan (1984).

INFORMATICA 3/1984

PROGRAMSKI JEZIK PASCAL II.

M. GAMS (1),
I. BRATKO (2,1),
V. BATAGELJ (3),
R. REINHARDT (1),
M. MARTINEC (1),
M. ŠPEGEL (1),
P. TANCIG (1)

UDK: 519.682.8

(1) Institut „JOŽEF STEFAN",
(2) FAKULTETA ZA ELEKTROTEHNIKO, UNIVERZA E. KARDELJA
(3) FAKULTETA ZA NARAVOSLOVJE IN TEHNOLOGIJO

V članku so najprej opisana pravila lepega programiranja v Pascalu. Večji del članka je posvečen slabostim Pascala in oceni kritik v literaturi. Sledi opis primernih in neprimernih področij uporabe, zaključna ocena pa vsebuje celotno analizo.

Programming Language Pascal II (detailed analysis of Pascal's shortcomings). First we very shortly describe what is "good" programming in Pascal and than we devote special care to Pascals shortcomings and critics in literature. Finally we describe which problem domains are suitable for Pascal and which not.

1. Uvod

Vsakemu jeziku lahko očitamo kopico pomanjkljivosti. Tudi Pascal ni izjema. Nekaterim slabostim se da izogniti z doslednim programiranjem, nekatere slabosti pa so neprijetne za uporabnika. Zato smo najprej navedli pravila lepega programiranja v Pascalu. Šele nato smo skušali zbrati vse smiselne pripombe in jih oceniti. Na koncu so zbrani rezultati ankete o Pascalu. Vse skupaj naj da uporabnikom dodatno znanje o tem, kdaj uporabiti Pascal in kdaj ne, snovalce novih jezikov pa naj opozori na nekatere napake, ki naj jih pri današnjem stanju računalniške znanosti ne bi smeli ponavljati. Vse to razkrivanje napak pa naj ne vrže slabe luči na Pascal, saj si ta izvrstni jezik zasluži več pohval kot kritik in najbrž enega prvih mest med algoritmičnimi jeziki /13/.

Radi bi se zahvalili vsem, ki so sodelovali pri popravljanju članka, predvsem pa: Janezu Žerovniku, Marku Bohancu, Henriku Krnecu, Damjanu Bojadžijevu in Igorju Mozetiču.

2. Pravila lepega programiranja v Pascalu

Vsak jezik bolj podpira (in vsiljuje) nekatere stile programiranja. Z uporabo človeku bolj naravnega stila programiranja se poveča produktivnost programerja. Program večkrat beremo, kot pa ga pišemo. Vsak jezik, še posebej pa splošno namenski jezik, je mogoče "zlorabiti", to je uporabiti na neprimeren način. Npr. programerji, ki s Fortrana preidejo na Pascal, pogosto brez potrebe uporabljajo "goto" stavek. Podobno lahko zlorabimo Pascal z nepremišljeno uporabo globalnih spremenljivk /1/.

2.1. Strukturiranje

Pascal omogoča učinkovito strukturiranje, tj. pravilno strukturo zapisa algoritma. Osnovni so trije principi strukturirane gradnje programov:

a) od zgoraj navzdol (top-down)

b) od spodaj navzgor (bottom-up)

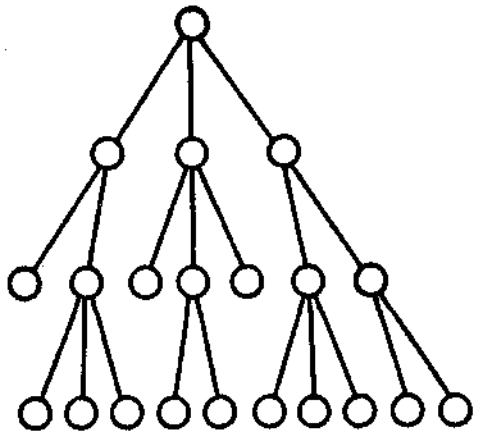
c) kombinacija a in b.

Princip "od zgoraj navzdol" gradnje programov je v tem, da obsežne heterogene kose delimo na manjše in bolj kompaktne dele. Manjše dele lažje obvladamo, preglednost raste.

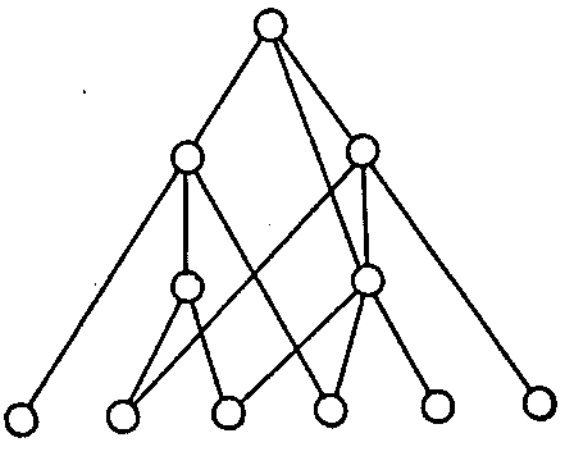
Princip "od spodaj navzgor" gradnje programov je v tem, da se nekatera osnovna opravila - atomarne funkcije - pogosto ponovijo. Te atomarne operacije najprej zakodiramo, nato pa čimveč ostalih podprogramov gradimo s čimveč klici že kodiranih podprogramov. Tako lahko realiziramo programe z manj kode, "moč" kode naraste.

V praksi se najbolj pogosto srečujemo z uporabo obeh stilov, tako da sproti razbijamo večje kose v manjše, kadar pa naletimo na osnoven podprogram, ga takoj zakodiramo. Med kodiranjem vedno preverjamo, ali lahko kakšno nalogo rešimo tako, da uporabimo že zakodiran (malce popravljen) podprogram. Podprogram naj bo dolg okoli ene strani in naj vsebuje največ 7 podprogramov. Podprogram mora imeti omejen pretok informacij preko čimmanj parametrov. Za globalne spremenljivke glej pravilo številka 2.2. Kadar bo program obsežen, moramo pred kodiranjem vedeti, kaj vse bo sestavljalo en modul (za overlaye ali za vsebinsko povsem ločene module), drugače pa pod-

programov rajši ne gnezdimo, ampak jih gradimo "od spodaj navzgor" ali v obliki produkcijskih sistemov /2,3/.



Slika 1: Strukturiranje "od zgoraj navzdol": večje module razbijamo v manjše tako, da med moduli ni povezav. "Idealna" kontrolna struktura je drevo brez prekrivanja listov, gradimo ga od zgoraj navzdol.



Slika 2: Strukturiranje "od spodaj navzgor": osnovne podprograme čimprej zakodiramo in ostale podprograme gradimo s pomočjo že kodiranih podprogramov, ki so dostopni vsem podprogramom. Kontrolna struktura je neplanaren usmerjen graf, prekrivanja je precej.

Vsak izmed načinov kodiranja ima svoje prednosti in slabosti. Bolj pogosto srečamo kodiranje od zgoraj navzdol, ker lažje razbijamo velik problem na lažje podprobleme in zato je preglednost običajno zelo dobra. Strukturiranje od spodaj navzgor omogoča veliko izkoriščenost kode, vendar ni primerno za vsa področja. Kljub temu je po mnenju nekaterih ob pazljivi uporabi to izredno učinkovita metoda.

Poglejmo si rezultate strukturiranja na primeru iz literature/4/: Ko so obstoječi program v jeziku PL/I preoblikovali iz nestrukturirane oblike v strukturirano, so testirane osebe za razumevanje nestrukturiranega programa potrebovale dvakrat več časa kot za razumevanje strukturirane verzije. Tudi drugi faktorji kot ciklomatična kompleksnost (ustreza intuitivnemu pojmu zapletenosti ciklov) so se bistveno izboljšali pri praktično isti hitrosti izvajanja. Pač pa se je bistveno (za polovico) podaljšalo število vrstic programa, največ na račun deklaracij procedur.

2.2. Ne uporabljaj goto stavkov in globalnih spremenljivk, če to ni nujno potrebno.

2.2.1. Pogosta uporaba konstrukta "goto" zelo poslabša čitljivost kode /4/, zato ga uporabljaj le v izjemnih primerih, npr. pri nenadni prekinitvi izvajanja ali pri skoku iz zanke.

2.2.2. Globalne spremenljivke uporabljaj le takrat, kadar so pogosto uporabljane v večjem številu podprogramov, ali kadar so podatkovne strukture obsežne. V komentarju ob klicu podprograma in v glavi komentarja navedi, katere globalne spremenljivke nastopajo v podprogramu in kdaj (kako) spremenijo vrednost znotraj podprograma.

2.3. Zamikaj

Stile zamikanja si lahko ogledamo v literaturi. Pri zamikanju ne obstaja en splošno veljaven princip in vsak programer ima svojo inačico oblikovanja programa. Smisel zamikanja je v tem, da z obliko nakažemo strukturo programa, oziroma algoritma. Lepo zamikanje močno poveča čitljivost programa.

2.4. Uporabljaj mnemonična imena za spremenljivke, tipe, imena podprogramov, itd.

Tako se močno poveča čitljivost in samodokumentiranost programa. Mnemoničnih imen ne uporabljamo le za podatke, ki nimajo semantičnega pomena, npr. tabela "TAB" ali "A" nastopa v podprogramu za sortiranje, ali za pogosto rabljene spremenljivke kot npr. "i" za števec v zanki. Le izjemoma - ali sploh ne - krajšamo mnemonična imena npr. "število vrstic" v "stvr". Samostojne besede združujemo v eno ime tako, da prvo črko samostojne besede pišemo z veliko začetnico, npr. "številoVrstic" ali "ŠteviloVrstic".

2.5. Komentiraj, dokumentiraj

Obvezno opiši vlogo vsakega podprograma, kaj dela, kakšen vhod in izhod ima in vse posebnosti. S komentarjem loči podprograme ali obsežne stavke, npr. tako, da ima prvi "begin" in pripadajoči "end" pripisano ime podprograma ali ime konstrukta. Vsak podprogram naj ima vsaj en stavek komentarja.

3. Analiza kritik Pascala

3.1. Uvod

V strokovni literaturi za vsak programski jezik najdemo vsaj nekaj kritičnih člankov. Bistven problem snovalcev vsakega programskega jezika je najti pravo ravnotežje med sposobnostjo in preprostostjo. Tudi to je eden od razlogov, da noben jezik ne more biti primeren za vsa področja in za vse stile programiranja. Kritika je upravičena zlasti takrat, kadar opozori na očitno slabost ali kadar predlaga boljšo rešitev.

Programski jezik je običajno definiran na štirih nivojih:

a) abstraktna sintaksa (za Pascal ena A4 stran)
b) konkretna sintaksa (za Pascal štiri A4 strani sintaktičnih diagramov)
c) priročniki in ostala literatura o standardnem jeziku (Pascal User Manual and Report ima 167 strani)
d) prevajalnik za standardni jezik (okoli 150 - 200 strani listinga).

Običajno tudi velja, da se kontekstno svobodna gramatika konča z b) in v c) opišemo kontekstno občutljivo gramatiko.

Z nivojem narašča količina informacij in določenost jezika. Iz abstraktne sintakse je npr. že razvidna možnost stranskih efektov pri funkcijah. S konkretno sintakso so določene stvari kot hierarhija operatorjev, oblika kontrolnih konstruktov, itd. Izjeme, ki jih s konkretno sintakso ni mogoče lepo opisati, so opisane v priročnikih. Tak primer je IF - THEN - IF - THEN - ELSE konstrukt, za katerega iz sintaktičnih diagramov ni jasno, ali zadnji ELSE pripada prvemu ali drugemu IF-u. Zaželjeno je, da je takih izjem čim manj. Slabo je, kadar ostanejo take izjeme neopisane ali prepuščene implementatorju. V Pascalu je to primer za evaluacijo (ovrednotenje) Boolovih izrazov, ko ni jasno, ali se v logičnem izrazu "p1 and p2 and p3" vedno ovrednotijo vsi trije pogoji ali celoten izraz dobi vrednost "false", kadar npr. "p1" dobi vrednost "false". Kot posledica tega imajo nekateri prevajalniki en in drugi drug način evaluacije, to pa poslabša prenosljivost programov.

Večina kritik v literaturi je povzeta v nadaljevanju članka. Poleg kritik je narejena analiza in ocena upravičenosti kritike. Kritike so v veliki meri privzete po /5/, predvsem pa se skicujemo še na /6,7,8,9/ in /10/. Pri tem analiziramo iste slabosti Pascala, le da se ocene včasih razlikujejo.

3.2. Problemi na nivoju kodiranja

3.2.1. Nezaključen komentar. Pri komentiranju pogosto naredimo napako, da pozabimo zaključiti komentar, Recimo:

```
i := 1 - 1; (* nek komentar pozabimo zaključiti ??)
j := 1 ;    (* drug komentar *)
```

To napaka je pogosto neprijetna, saj je prevajalnik ne odkrije in lahko se zgodi, da program dela pravilno, razen za posebno kombinacijo vhodnih podatkov.

Komentar: Predlagana rešitev z vrstičnimi komentarji kot v Fortranu je slabša kot naslednji dve možnosti:

- dober listing programa nam pokaže, kateri stavki so komentarski
- prevajalniku bi lahko dodali opozorilo, kadar bi znotraj komentarja naleteli na simbol "(*" .

Obe rešitvi je mogoče enostavno realizirati.

3.2.2. Fiksiran red deklaracij - CONST, TYPE, VAR. Nekateri avtorji trdijo, da je bolj smiselno dodati možnost poljubnega ponavljanja teh treh deklaracij. Komentar: Smiselno bi bilo dodati poljuben vrstni red deklaracij. To zahteva minimalne popravke prevajalnika, olajša pa probleme pri ločenem prevajanju ali delu s knjižnicami.

3.2.3. Napake zaradi "default-a". Neprijetna napaka je npr. pozabljeni "var" pri deklariranju parameterov ali to, da pozabimo deklarirati spremenljivko znotraj podprograma in postane globalna. Primer:

```
procedure increment ((*var*) i: integer);
(*var i : integer; *)
begin
   i := i + 1
end ;
```

Komentar: Mogoče bi bilo smiselno izločiti "default" princip. Tako bi za globalne spremenljivke vedno deklarirali npr. "global" in za lokalne parametre npr. "value".

3.3. Problemi s tipi

3.3.1. Vrednost funkcije je lahko le nekaj strogo določenih tipov. Komentar: Brez večjih naporov bi lahko spremenili prevajalnik tako, da bi imele funkcije poljuben tip in nekatere izvedenke Pascala to že imajo. Omejitev vrednosti funkcij pa ni zelo neprijetna, saj si lahko pomagamo s klicem procedure.

3.3.2. Množice. Komentar: Množice so zelo uporaben podatkovni tip. Na žalost pa so vezane na dolžino besede in največja možna dolžina množic je tako najpogosteje vezana na konkreten računalnik, kar zelo otežkoča prenosljivost programov. Dokaz za boljšo implementacijo množic najdemo npr. v izpeljanki Pascala "P+" /5/. Možna rešitev bi bila, če bi v standardnem Pascalu dovolili deklaracijo dolžine množice na

način kot pri dinamičnih tabelah. Druga slabost množic je, da so v Pascalu lahko elementi množice samo statični objekti, torej se njihova vsebina s časom ne more spreminjati. Zaradi tega so množice učinkovito implementirane, žal pa to dostikrat onemogoča udobno programiranje. Zato bi bilo mogoče smiselno dodati poseben konstrukt, imenujmo ga "LIST", ki bi omogočal tudi konstrukte tipa "množica zapisov" (set of record). Tak konstrukt pa je razumljivo implementacijsko precej počasnejši od standardnih množic.

3.4. Manjkajoči konstrukti

3.4.1. Manjkajoči inverzni konstrukt "ORDu". Razen za znake (chr) inverzni konstrukt "ORDu" ne obstaja. Komentar: Smiselno bi bilo dodati inverzni konstrukt "ORDju", saj ne zahteva velike popravkov prevajalnika. Tako bi za vsak enostaven tip T = (v0, v1, v2, ..., vn) imeli funkcijo ORD: i := ORD(vi) in funkcijo za inverzno transformacijo: vi := Create(T,i). Brez te inverzne funkcije npr. ne moremo preprosto poiskati srednjega elementa za neceloštevilčne indekse:

```
srednjiElement :=
  Create(T,(ORD(leviElement) +
  + ORD(desniElement)) div 2));
```

Tak konstrukt je enostavno dodati v prevajalnik.

3.4.2. Manjkajoči "LOOP" in "EXIT" konstrukt. Nekateri avtorji predlagajo "LOOP" konstrukt, podobno kot v COBOLU, in s prekinitvijo zanke z "EXIT"<name> konstruktom (ta obstaja v nekaterih verzijah Pascala). Komentar: V Pascalu je zadovoljiv konstrukt

```
WHILE true DO
BEGIN
  ----
  IF alarm THEN EXIT (*ali izjemama 'GOTO 111'*);
  ----
END;
111:
```

"LOOP" konstrukt je po mnenju večine nepotreben.

3.4.3. Manjkajoči "ELSE" v "CASE" stavku. Komentar: Smiselno bi bilo dodati "ELSE" ali "OTHERWISE" konstrukt v "CASE" stavek, saj zahteva majenkosten popravek prevajalnika. Novejši prevajalniki ga večinoma imajo.

3.5. Slabosti

3.5.1. Vrstni red evaluacije Boolovih izrazov. V poglavju 3.1. smo omenili, da ni jasno, ali se ovrednotijo vsi podizrazi v Boolovem izrazu tipa "p1 and p2", ali se vrednotenje prekine čim prevajalnik ugotovi, da je "p1" false. V naslednjih treh primerih vidimo prednosti evaluacije s prekinitvijo:

```
WHILE (i <= MAX) AND (a[i] <> 0) DO ...
WHILE (p <> nil) AND (p^.vsebina <> iskanaVsebina) DO...
WHILE NOT eof(f) AND NOT (ft = '*') DO ...
```

Komentar: V priročniku bi morali določiti način ovrednotenja Boolovih izrazov. Precej boljše je ovrednotenje s prekinitvijo.

3.5.2. Datoteke. Standardni Pascal nima definiranega povezovanja logičnih datotek z datotekami v operacijskem sistemu. Nima sintakse za branje in pisanje po datotekah z direktnim dostopom. Nima indeksno sekvencialnih datotek. Okence v datotekah je vedno inicializirano na tekoče mesto v datoteki, kar otežkoča interaktivno delo. Pri branju numeričnih podatkov je otežkočano testiranje konca datoteke. Komentar: Opis odpiranja datotek ali branja po datotekah z direktnim dostopom je močno vezan na operacijski sistem, zato se sintaktično močno razlikuje. Večina teh problemov je zadovoljivo rešena na konkretnem prevajalniku. Hujša pomanjkljivost je pomanjkanje indeksno sekvencialnih datotek na večini prevajalnikov. Le redke verzije Pascala (Pascal R) jih imajo. Na splošno pa so datoteke šibka točka Pascala.

3.5.3. Prirejanje začetnih vrednosti. V Pascalu ne moremo uporabiti naslednjih izrazov:

```
CONST
    n = 10;
    m = 20;
    i = n * m;
VAR
    a: array[1..n*m] of integer;
BEGIN a := (3, 4, 5, ...)
```

Prav tako ne moremo deklarirati začetnih vrednosti spremenljivkam pri deklaraciji, ne obstaja ekvivalent "DATA" stavku iz Fortrana. Komentar: Prirejanje začetnih vrednosti bi lahko zelo enostavno dodali v prevajalnik. Nekateri prevajalniki imajo dano to možnost.

3.5.4. Nezmožnost direktnega dosega računalniških enot. Pascal nima posebnih konstruktov za sistemsko delo kot npr. Modula ali C. Komentar: Te naloge lahko realiziraš z zbirnikom, kar pa še vedno ni enakovredno jezikom za sistemsko delo.

3.5.5. Pomanjkanje "lastnih" spremenljivk. Pascal nima lastnih spremenljivk, torej podprogram ne more imeti svoje spremenljivke, ki bi obdržala vrednost do ponovnega klica podprograma, ne da bi bila vidna vsem ostalim podprogramom kot npr. globalne spremenljivke. Komentar: Zaradi velikega pome-

na skritih spremenljivk za ločeno prevajanje in knjižnice bi bilo smiselno dodati lastne spremenljivke. To ne bi zahtevalo veliko popravkov prevajalnika in pogosto najdemo tovrstne možnosti v novejših prevajalnikih.

3.5.6. Neučinkovitost knjižnic. Knjižnice so neučinkovite zaradi stroge kontrole prenosa parametrov (type checking) in pomanjkanja skritih spremenljivk. Zato je v večini obstoječih prevajalnikov npr. nemogoče napisati funkcijo, ki bi izračunala dolžino poljubnega niza (packed array (MINX .. MAXX) of char). Novi ISO standard /9/ to sicer omogoča, vendar samo za tabele. Problemi z dinamičnimi strukturami (seznami, drevesa) pa ostajajo nerešeni. Komentar: Pri ločenem prevajanju in pri klicanju podprogramov v zbirniku ni kontrole tipov, vendar to ne omogoča splošno uporabnih podprogramov npr. za procesiranje dreves. To je precejšnja slabost, ki pa je ni mogoče enostavno rešiti. Nekateri prevajalniki kot Pascal 2 pa omogočajo prenose tudi brez kontrole tipov.

3.5.7. Nezmožnost ločenega prevajanja in procesiranja v realnem času. V standardnem Pascalu ni govora o ločenem prevajanju ali o konstruktih za nadzorovanje procesov v realnem času. Komentar: Večina prevajalnikov omogoča ločeno prevajanje, vendar vsak na svoj način. Nezmožnost ločenega prevajanja je ena večjih slabosti Pascala. S pomočjo podprogramov v zbirniku lahko na večini računalnikov procesiramo v realnem času tako, da podprogrami v zbirniku kličejo podprograme v Pascalu. Ta možnost pa zahteva globlje poznavanje zbirnika, pascalskega prevajalnika in operacijskega sistema.

4. Anketa o Pascalu

V reviji Sigplan Notices so objavili rezultate ankete o Pascalu /11/. Tu številke pomenijo število pozitivnih odgovorov.

Prednosti Pascala:

30 kontrolne strukture
26 podatkovni tipi
20 stroga kontrola tipov
14 preprostost
8 prenosljivost
5 dobra čitljivost
4 rekurzija
4 kontrola mej

Slabosti Pascala:

7 nezmožnost ločenega prevajanja
6 omejeno branje/čitanje
5 stroga kontrola tipov
3 slabe možnosti procesiranja nizov

Zakaj uporabljate Pascal

13 ker je prenosljiv
12 ker je enostaven za uporabo
10 ker omogoča dobro strukturiranje
6 ker omogoča strogo kontrolo tipov
8 ne uporabljajo Pascala, ker obstaja boljši jezik (najpogosteje omenjeni je C)
(opomba - verjetno so vprašani mislili na uporabo jezika na sistemskem področju. C je verjetno bolj uporaben za sistemsko delo, manj pa za ostala področja).

5. Primerna in neprimerna področja uporabe

Standardni Pascal brez dodatkov je primeren za:
- pisanje prevajalnikov
- procesiranje tekstov
- pisanje uporabniških programov, npr. editorjev
- procesiranje nenumeričnih podatkov
- procesiranje dreves, seznamov in drugih kompleksnih podatkovnih tipov
- nekatere matematične probleme
- za učenje
- za pisanje splošno prenosljivih programov.

Brez dodatkov je manj primeren za:
- sistemsko programiranje
- aplikacije v realnem času in kontrolo procesov
- paralelno procesiranje
- konstrukcijo velikih programov
- numerično analizo
- aplikacije, ki zahtevajo indeksno sekvencialne datoteke
- nekatere poslovne aplikacije.

Velja poudariti, da lahko za večino manj primernih področij učinkovito izboljšamo lastnosti Pascala, tako da uporabimo podprograme v zbirniku ali Fortranu. Poleg tega lahko uporabimo bolj specializirano usmerjene verzije kot USCD Pascal za mikroračunalnike, Pascal PLUS za diskretne simulacije in Concurrent Pascal za aplikacije v realnem času. Novejši Pascali uspešno rešujejo večino tu omenjenih problemov.

6. Zaključna ocena

Nekaj slabosti Pascala bi lahko z minimalnim trudom odpravili, tako da bi imel prevajalnik malo popravkov, funkcijsko pa bi bil Pascal precej močnejši. In najbrž bi moralo vsakemu Pascalskemu prevajalniku dodati indeksno sekvencialne datoteke. Precej kritik je nemogoče razrešiti, ne da bi se prevajalnik in jezik pretirano razširila. Večino omenjenih problemov imajo novejši Pascali običajno dokaj elegantno rešenih. Kljub vsemu pa je Pascal po svojih lastnostih verjetno eden najboljših pred-

stavnikov algoritmičnih splošno namenskih jezikov. Jeziki kot FORTRAN, COBOL, BASIC ali PL/1 so v splošnem objektivno nekaj slabši, čeprav so primernejši za določena področja. Velja naslednje: Pascal (in podobni jeziki kot MODULA-2 ali inačice ADE) je verjetno eden najboljših splošno namenskih jezikov, čeprav skoraj na vsakem ožjem področju lahko najdemo jezike, ki so boljši kot Pascal. Učinkovitost novejših Pascalskih prevajalnikov tako glede dolžine kode generiranega programa in hitrosti izvajanja kode je približno taka kot pri novejših Fortranskih prevajalnikih /12/, torej običajno neprimerno boljša kot pri BASICu, COBOLu ali PL/1. Velika slabost Pascala ostaja slabo programersko okolje. Programer veliko časa porabi za testiranje programov. Jeziki (bolje rečeno programsko okolje), ki omogočajo ločeno prevajanje, uspešno testiranje (debugger) in imajo interpreter in prevajalnik, ter vse to integrirano in istočasno dostopno, so bistveno boljše orodje za testiranje, kot pa npr. Pascal. Tudi Pascalske knjižnice s splošno uporabnimi podprogrami so le redkokdaj komercialno dosegljive in jih mora uporabnik pisati sam, še zlasti kadar bi rad odpravil kakšno pomanjkljivost Pascala s podprogramom v zbirnem jeziku. Ravno to pa je področje, kjer lahko v naslednjih letih pričakujemo največje korake naprej.

7. Literatura

1. E.B. Levy: The Case Againt Pascal as a Teaching Tool, ACM SIGPLAN Notices, Vol. 17, Num. 11, str. 39-42, november 1982
2. D.A. Waterman, F. Hayes-Roth: An Overview of Pattern-Directed Inference Systems, Academic Press, 1978
3. M. Gams: Pomen in vloga znanja v sistemih za interakcijo z uporabnikom, magistrsko delo, junij 1982
4. J.L. Elshof, M. Marcotty: Improving Computer Program Readibility to Aid Modification, CACM, Vol. 25, Num. 8, str. 512-521, avgust 1982
5. R. Cailliau: How to Avoid Getting SCHLONKED by Pascal, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 31 - 41, december 1983
6. R.E. Sumner, R.E. Gleaves: Modula-2 -- A Solution to Pascal's Problems, ACM SIGPLAN Notices, Vol. 17, Num. 9, str. 28 - 34, september 1982
7. R. Cailliau: A Letter to Editor, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 10 - 11, december 1982
8. K. Jensen, N. Wirth: Pascal, User Manual and Report, Springer Verlag, 1978
9. Second draft proposal ISO/DP 7185 - Specification for the Computer Programming Language - Pascal, Pascal News, Num. 20, december 1980
10. N. Wirth: The Design of a Pascal Compiler, Software Practice and Experience, 1, str. 309 - 333, 1971
11. K. Mogel: A Report on a PASCAL Questionaire, ACM SIGPLAN Notices, Vol. 17, Num. 10, str. 23 - 33, oktober 1982
12. Benchmark test na Quicksortu, Special Software Limited, Informatica 3, str. 77, 1982
13. M. Gams, I. Bratko, V. Batagelj, R. Reinhardt, M. Martinec, M. Špegel, P. Tancig: PASCAL I (primerjava z ostalimi jeziki), Informatica 1, str. 22 - 26, 1984

Novice iz Instituta "Jožef Stefan"

Domač enokartični mikroračunalnik tipa DEC

Pod vodstvom M.M.Miletića je bil razvit enokartični mikroračunalnik MMA-11 z DECovim 16-bitnim mikroprocesorjem T-11, ki uporablja operacijski sistem RT-11. Ta računalnik ima 64k-zložni hitri pomnilnik, dvoje serijskih in ena paralelna vrata, krmilnik za upogljivi disk itd. Predviden je tudi krmilnik za trdni disk. Podobnost z mikroračunalnikom VT-150 je očitna.

Program za povezovanje na tiskanih vezjih

Na IJS je bil razvit program za iskanje povezav na tiskanih vezjih (M. Gams). Problem 500 povezav je npr. ta program razrešil v manj kot 4 urah (računalnik Delta 4850). Program je v paketu z rutinami za izrisovanje shem, filmov in izdelavo trakov za numerično krmiljeni vrtalni stroj.

A. P. Železnikar

INFORMATICA 3/1984

TESTIRANJE ROM POMNILNIKOV

B. KASTELIC,
R. MURN,
D. PEČEK

UDK: 681.3.325.6.08

INSTITUT „JOŽEF STEFAN"

V članku opisujemo testni postopek za testiranje delovanja in verifikacijo vsebine ROM pomnilnika. Postopek je zasnovan na programskem računanju ciklične kode nad testiranim področjem pomnilnika in verifikacijo le-te. Na koncu članka je priložen izpis podprograma za računanje CRC za mikroračunalnik s procesorjem M6800.

TESTING ROM MEMORY. In the paper the dynamic and static test procedures for testing and verifying ROM - look like memories are described. The basic elements of procedures are cyclic redundancy codes. At the end of the paper the 6800 processor software procedure for CRC generation is shown.

1. UVOD

V mikroračunalniških sistemih vedno srečamo poleg pomnilnika z naključnim dostopom (RAM) še pomnilnik s stalno vsebino, ki jo lahko samo čitamo - ROM pomnilnik. Ta je večje ali manjše kapacitete, kar zavisi od konfiguracije sistema. V sistemih, ki imajo na razpolago zunanje pomnilne enote, je v ROM pomnilniku običajno zapisana le osnovna sistemska programska oprema, ki nam omogoči nalaganje programov v delovni pomnilnik. Obstajajo pa tudi manjši mikroračunalniški sistemi, ki imajo vso sistemsko programsko opremo shranjeno v ROM pomnilniku. Tudi ROM pomnilnik je potrebno občasno, predvsem pa ob zaznanih neregularnostih delovanja sistema, preizkusiti in verificirati njegovo vsebino.

ROM pomnilnik je funkcionalen, če lahko v vsakem trenutku pravilno čitamo vsebino vsake celice.

V ROM pomnilniku se lahko napake pojavijo na področju pomnilniških celic, v dekodirni logiki ali v čitalni logiki. Vse napake na področju pomnilniških celic se ob predpostavki, da je bil ROM pomnilnik pravilno zapisan, odražajo kot sprememba vsebine ene ali več pomnilnih celic. Za napake v dekodirni in čitalni logiki zadošča model napak za področje pomnilniških celic.

V ROM pomnilnik ne moremo zapisati znane vsebine, zato je za odkrivanje napak potrebno poznavanje že zapisane vsebine. Najenostavnejši postopek za odkrivanje napak bi bila verifikacija vsebine z ustrezno referenčno vsebino, ki bi bila lahko zapisana na zunanji pomilni enoti. Tak postopek je zelo dolgotrajen, poleg tega pa obstajajo tudi sistemi brez zunanjih pomnilnih enot. Prav ti sistemi pa imajo ROM pomnilnik največje kapacitete. Nekateri uporabljajo za verifikacijo vsebine ROM pomnilnika kontrolno vsoto, ki pa je preveč nezanesljiva za odkrivanje napak v pomnilnikih večje kapacitete. Mi smo izbrali testni postopek, ki za posamezno pomilno vezje izračuna ciklični kod. Izračunani ciklični kod primerjamo z referenčnim kodom in, če se ujemata, je vsebina ustreznega pomnilnega vezja pravilna.

Naloga je bila realizirana v okviru diagnostičnega paketa za mikroračunalnik TK6800, ki

ga izdeluje za potrebe telefonije ISKRA - Telematika.

2. VERIFIKACIJA PODATKOV S CIKLIČNIMI KODI

Ciklični kodi se uporabljajo zlasti pri masovnih magnetnih pomnilnikih in pri prenosih podatkov. Štejemo jih med kodirne kontrolne postopke.

Algebrajske kode lahko spremljamo s polinomom z neodvisno spremenljivko x. Tako lahko informacijo

10011010001...

predstavimo kot polinom

$$G(x) = 1x^0+0x^1+0x^2+1x^3+1x^4+... = 1+x^3+x^4+x^6+x^{10}+...$$

Nad polinomom G(x) lahko izvajamo različne operacije z vednostjo, da velja seštevanje po modulu 2

$$1x^n+1x^n = 0x^n$$
$$1x^n+0x^n = 1x^n$$
$$0x^n+0x^n = 0x^n$$
$$-1x^n = 1x^n$$

Informacijski polinom označimo z G(x) in je stopnje < K (dolžina informacije je K bitov). Polinom, s katerim delimo informacijski polinom, se imenuje generatorski polinom in ga označimo s P(x). Celo sporočilo se imenuje kodirani polinom, označimo ga z F(x). Število bitov v celem sporočilu je N. Na začetku je K bitov informacije in na koncu kodiranega polinoma je ostanek R(x) dolžine N-K bitov. Ostanek izračunamo, ko pomnožimo informacijski polinom G(x) z

$$x^{N-K}$$

in delimo s P(x).

$$x^{N-K}G(x) = P(x)Q(x)+R(x)$$

Dolžina ostanka R(x) je N-K bitov, torej je stopnje < N-K. Q(x) je kvocient (rezultat delitve), ki pa nas ne zanima. Odposlani kodirani polinom je tako:

$$F(x) = G(x)x^{N-K}+R(x) = P(x)Q(x)$$

F(x) je tako zgrajen, da bo vedno deljiv z generatorskim polinomom. Ravno to lastnost pa uporabimo pri verifikaciji. Na sprejemni strani delimo kodirani polinom F(x) prav tako z generatorskim polinomom P(x), in če ostanek R(x) ni enak 0 vemo, da so v sporočilu napake.

Primer računanja ostanka:

Vzemimo generatorski polinom

$$P(x) = 1+x^2+x^4+x^5 = 101011$$

in sporočilo 101010001. To sporočilo najprej predstavimo kot polinom

$$G(x) = 1+x^2+x^6+x^7$$

Ker je N-K = 5, pomnožimo G(x) z x^5 in delimo s P(x):

```
                1110001111   - kvocient
(110101)  100010010100000    - deljenec
delitelj  110101
           101110
           110101
            110111
            110101
                100100
                110101
                 100010
                 110101
                  101110
                  110101
                   110110
                   110101
                       11    - ostanek
```

Dobimo naslednji kodirani polinom:

$$F(x) = x^5G(x)+(1+x)$$
$$= 1+x+x^5+x^7+x^{10}+x^{11}$$
$$= 100010010100011$$

Če sedaj F(x) ponovno delimo z generatorskim polinomom P(x), dobimo ostanek 0. Poglejmo si še to delitev.

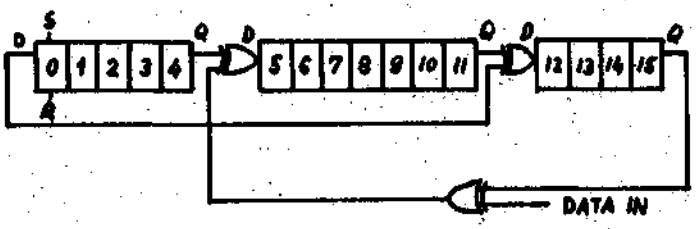
```
                1110001111   - kvocient
(110101)  100010010100011    - deljenec
delitelj  110101
           101110
           110101
            110111
            110101
                100100
                110101
                 100010
                 110101
                  101111
                  110101
                   110101
                   110101
                       0     - ostanek
```

Iz gornjega računa vidimo, da je način deljenja v bistvu zelo preprost. V vsakem koraku odštejemo 110101 na tistem mestu, kjer se prva enica ujema z najvišjo enico v ostanku. Ta postopek je zelo enostavno realizirati z vezji, kot bomo videli, pa tudi programsko.

Standard CRC-CCITT, ki se največ uporablja pri prenosu podatkov in zapisu podatkov na magnetne medije, ima generatorski polinom

$$P(x) = 1+x^5+x^{12}+x^{16}$$

Ustreza mu registreska ureditev na sliki 1.



SLIKA 1: Realizacija CRC-CCITT standarda

V registrih se po vsakem bitu ohranja ostanek delitve. Pri oddajanju se na koncu vsebina registrov odda za informacijskim delom sporočila. Na sprejemni strani, se v enakem vezju deli kodirano sporočilo. Na koncu delitve je ostanek v registrih ob pravilnem sprejemu 0.

Iz teorije cikličnih kod (1) izhaja, da ima generatorski polinom

$$P(x) = 1+x^5+x^{12}+x^{16}$$

naslednje lastnosti:

a) odkriva vse enojne napake pri poljubnem zaporedju bitov;
b) odkriva vsako liho število napak;
c) odkriva vse dvojne napake, če je dolžina N celega sporočila manjša ali enaka 32767 bitov;
d) odkriva vsako skupinsko napako dolžine 16 bitov ali manj;
e) odkriva 99,997% skupinskih napak dolžine 17 bitov pri poljubnem zaporedju;
f) odkriva 99,998% skupinskih napak večje dolžine kot 17 bitov pri poljubnem zaporedju.

3. TESTNI POSTOPEK

Za odkrivanje napak in verifikacijo vsebine ROM pomnilnika smo uporabili lastnosti cikličnih kod. Predpostavili smo namreč, da nimamo na razpolago referenčne vsebine ROM pomnilnika, zato bomo za vsebino posameznega vezja izračunali ciklično kodo oziroma ostanek po standardu CRC - CCITT.

Pri diagnostiki ROM pomnilnikov velikoserijskih sistemov imamo lahko v okviru diagnostičnega programa v posebni tabeli shranjene ostanke deljenja R(x). V tem primeru dodamo vsebini ROM pomnilnika še ustrezni ostanek deljenja in vse skupaj delimo z generatorskim polinomom. Če je ostanek deljenja nič pomeni, da v vezju ni napak ali, da jih generatorski polinom ne odkrije. Kot smo videli v prejšnjem poglavju je verjetnost, da generatorski polinom v vezju, ki ima manjšo kapaciteto od 32767 bitov, ne odkrije obstoječe napake, zelo zelo majhna.

V maloserijskih sistemih, ali v sistemih, ki nimajo sandardne vsebine ROM pomnilnika, ne moremo uporabiti te ugodne lastnosti cikličnih kod, saj običajno nimamo v okviru diagnostičnega programa ustrezne tabele ostankov. Zato se pri teh sistemih zadovoljimo z računanjem ostanka deljenja informacijskega polinoma z generatorskim polinomom. Ostanek deljenja na koncu preveri operater z referenčnim ostankom. Zaradi hitrejše verifikacije izračunamo poleg ostankov za posamezno vezje še skupni ostanek deljenja za celotni ROM pomnilnik. Če se skupni ostanek ne ujema, lahko operater na podlagi ostankov posameznih vezij določi vezje z napako.

Testni postopek obsega programsko računanje ostanka delitve. Programsko pretvorimo vsak zlog v zaporedje osmih bitov. Tako dobimo serijsko zaporedje bitov dolžine nx8, kjer je n kapaciteta vezja ROM pomnilnika v zlogih. To zaporedje bitov smatramo kot serijski izvor podatkov, oziroma informacijski polinom, ki ga delimo z generatorskim polinomom. V posebnih registrih se nam tvori ostanek, ki ga na koncu ovrednotimo.

Na koncu članka je priložen izpis podprograma za računanje CRC nad definiranim pomnilniškim področjem za mikroračunalnike s procesorjem M6800. Omenjeni procesor potrebuje pri

```
PAGE  001   ROM    TEST

00001                          NAM    ROM
00002                          TTL    TEST


00004                    ***********************************************
00005                    *                                             *
00006                    *    GENERATE CRC FOR ROM AREA                *
00007                    *                                             *
00008                    *   ENTRY: BEGADR = BEGIN ADDRESS             *
00009                    *          ENDADR = END ADDRESS + 1           *
00010                    *                                             *
00011                    *   EXIT:  CHECKW = CHECK WORD (CRC)          *
00012                    *          A&B    = CHECK WORD (CRC)          *
00013                    *                                             *
00014                    ***********************************************


00016                          XDEF   CRCGEN,BEGADR,ENDADR,CHECKW


0001BD 0000                    DSCT

00020D 0000     0002  A BEGADR RMB    2
00021D 0002     0002  A ENDADR RMB    2
00022D 0004     0002  A POLYNM RMB    2
00023D 0006     0002  A CHECKW RMB    2
00024D 0008     0001  A DATA   RMB    1
00025D 0009     0001  A COUNT  RMB    1


00027P 0000                    PSCT

00029P 0000 CE 0810  A CRCGEN LDX    #$0810   SET POLYNOMIAL
00030P 0003 FF 0004  D        STX     POLYNM
00031P 0006 CE FFFF  A        LDX    #$FFFF   SET CHECKWORD
00032P 0009 FF 0006  D        STX     CHECKW

00034P 000C FE 0000  D        LDX     BEGADR
00035P 000F A6 00    A CRCGE1 LDAA    X        READ DATA

00037                    * GENERATE CRC FOR BYTE

00039P 0011 B7 0008  D        STAA    DATA     SAVE DATA BYTE
00040P 0014 86 08    A        LDAA   #8
00041P 0016 B7 0009  D        STAA    COUNT    COUNTER = 8

00043P 0019 B6 0006  D CRCGE2 LDAA    CHECKW   EOR MSB OF PARTIAL CHECKWO
00044P 001C B8 0008  D        EORA    DATA     AND MSB OF DATA
00045P 001F 49                ROLA            PUT RESULT IN CARRY BIT

00047P 0020 B6 0006  D        LDAA    CHECKW
00048P 0023 F6 0007  D        LDAB    CHECKW+1
00049P 0026 24 06 002E        BCC     CRCGE3   IF C=0 THEN CRCGE2
00050P 0028 B8 0004  D        EORA    POLYNM   ELSE EOR CHECKWORD
00051P 002B FB 0005  D        EORB    POLYNM+1 AND POLYNOMIALORD
```

```
PAGE  002   ROM    TEST

00053P 002E 59          CRCGE3 ROLB            SHIFT CHECKWORD LEFT AND
00054P 002F 49                 ROLA            PUT CARRY BIT IN LSB

00056P 0030 B7 0006  D         STAA    CHECKW  STORE PARTIAL CHECKWORD
00057P 0033 F7 0007  D         STAB    CHECKW+1

00059P 0036 78 0008  D         ASL     DATA    SHIFT DATA LEFT
00060P 0039 7A 0009  D         DEC     COUNT   IF COUNT  0
00061P 003C 26 DB 0019         BNE     CRCGE2  THEN REPEAT

00063P 003E 08                 INX
00064P 003F BC 0002  D         CPX     ENDADR
00065P 0042 26 CB 000F         BNE     CRCGE1
00066P 0044 39                 RTS

00068                          END
TOTAL ERRORS 00000
```

sistemski uri 1MHz za izvajanje CRC algoritma nad pomilniškim področjem velikosti 1k zlogov približno 0,5 sekunde.

4. LITERATURA

1. R. Murn, D. Pečekt Verifikacija in zanesljivost prenosa podatkov pri sodobnih pomilnikih, IJB DP-175, december 1979

2. R. Murn, D. Pečekt Zanesljivost in verifikacija napak pri serijskih prenosih podatkov, Zbornik radova JUREMA 24 (1979), 2 svezak

3. J. Virant: Zanesljivost računalniških sistemov, Fakulteta za elektrotehniko, Ljubljana, 1981

4. B. Kastelic, R. Murn, D. Pečekt Funkcionalno testiranje mikroračunalniških enot, Informatica 2/3, 1983

INFORMATICA 3/1984

UPORABA PROGRAMSKIH GRAFOV PRI UGOTAVLJANJU VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH II.

B. DŽONOVA–JERMAN,
J. ŽEROVNIK

UDK: 519.698

INSTITUT JOŽEF STEFAN

POVZETEK. Prikazane so metode za merjenje vzporednosti v računalniških programih, ki za analizo algoritmov uporabljajo programske grafe. Časovne enačbe za vzporedno in zaporedno izvajanje računalniških ukazov so konstruirane za dva modela računalniške arhitekture krmiljene s pretokom podatkov. Razmerje med enačbami je uporabljeno kot merilo in ocena o obstoječih vzporednostih v računalniškem algoritmu. V kratkem so podane osnovne značilnosti računalnikov krmiljenih s pretokom podatkov.

ABSTRACT. "Application of program graph analysis for measurement of parallelism in computer algorithms".
Techniques of program graph analysis are used to measure the parallelism in computer programs. For a given semantic model of architectural support, characteristic timing equations are first constructed from the high level program to describe the sequential and parallel execution times. The ratio of these equations is then used as a measure of the inherent parallelism in the program. Graph analysis techniques are illustrated using two data flow models of architectural support. The basic feature of data flow computers are described very briefly.

5.3. Uporaba modela 1 pri vrednotenju konstruktov višjega nivoja

Model 1 in metoda časovnega vrednotenja programskega grafa omogočata vrednotenje konstruktov višjega nivoja kot so zanke in pogojni stavki. Pri tem je najpomembnejše, da so ti konstrukti primerno strukturirani. Časovne enačbe za določene operacije so ponazorjene skupaj z vozlišči, ki te operacije predstavljajo v usmerjenem grafu na sliki 2. Aciklično strukturo grafov zgradimo tako, da vozlišča, v katerih prihaja do zaprtih krogov, transformiramo v tranzitivno obliko (4). Oblika karakteristične enačbe je enako ne glede na to ali gre za vzporedno ali zaporedno izvajanje programa. Oznaka t se nanaša na čas potreben za izvajanje strojnega ukaza. Konstrukti s slike 2 omogočajo izračun karakteristične časovne enačbe zaporednega izvajanja za celotni program z enostavnim seštevanjem časov posameznih vozlišč. Karakteristično enačbo vzporednega izvajanja konstruiramo s pomočjo konstruktov iz slike 2 in sledečega pravila: če so vozlišča n1, n2, ..., nk podatkovno neodvisna potem je njihov zbirni vzporedni čas izvajanja enak max (tpar1, tpar2, ..., tpark).

Vsa vozlišča iz slike 2, ki so označena z B so sestavljena iz različnih konstruktov in omogočajo konstrukcijo časovne enačbe programa z vrha navzdol. Vozlišča označena s P pa označujejo predikate. Vse operacije v vozliščih B, ki sledijo vozliščem P čakajo na ovrednotenje predikata. Tak potek izvajanja je v skladu z osnovno predpostavko, da uporabljamo procesno enoto s povratno zanko v podatkovnem toku, tako kot je ponazorjeno na slikah 2c, 2d, 2e in 2f. Zakasnitve vseh operacij, ki so odvisne od parametrov, katere posreduje "while-do" zanko po zaključenem izvajanju so v skladu z načinom, po katerem deluje procesna enota s povratno zanko. Podobne zakasnitve, ki nastajajo v "repeat-until" zankah niso v skladu z delovanjem procesorja kot je bil slučaj s predhodnimi zankami, ker ponekod prihaja do prekrivanja med iteracijami. To je lahko vir resnih napak pri uporabi modela za računanje časovnih enačb programskih grafov. Do manj resnih napak lahko pride pri vrednotenju zakasnitev v operacijah, ki so odvisne od izhodnih vrednosti v pogojnih stavkih. Tak slučaj je ponazorjen v vozlišču OR na slikah 2c in 2d.

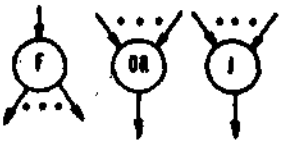
Sliki 2e in 2f ilustrirata tranzitivno predstavitev zank v programu. Časovna enačba za nerekurzivne procedure na sliki 2g vključuje čas za oblikovanje strukture argumenta oziroma izvajanje procedure B in predelavo strukture argumenta. Sliki 2h in 2i kažeta časovne enačbe za branje (ali zapis) elementarnih vrednosti in čas potreben za izvajanje operacij "readedit", "write" in "writedit" ter za ukaze "select".

V predhodnih poglavjih smo s pomočjo časovno ovrednotenega usmerjenega grafa G' = (N', A', T, B) opredelili merilo za oceno vzporednosti v računalniških programih. Merilo vzporednosti nam daje razmerje med časom vzporednega in zaporednega izvajanja algoritma. V ilustracijo metode smo na sliki 3. ponazorili analizo programa napisanega v visokem programskem jeziku. Poleg programa, slika 3, kaže še časovno ovrednoteni usmerjeni graf programa in karakteristične časovne enačbe. Čas izvajanja v posameznih vozliščih je ovrednoten s pomočjo preslikav, ki so ponazorjene na sliki 4. Kako vrednotenje poteka, bomo pokazali z izračunom časa za vozlišče 3. Vozlišče 3 ponazarja stavek v visokem programskem jeziku. Za izvajanje tega stavka sta potrebna dva ukaza "select" (2 časovni enoti), ukaz "multiply" (6 časovnih enot) in ukaz "append" (1 časovna enota), skupno 9 časovnih enot za vsako od N1 iteracij. Dejansko potrebuje vozlišče 3 le 8 časovnih enot, saj se ukaz "select" lahko izvaja v paraleli.

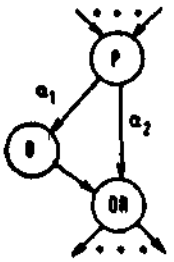
Iz slike 3 je razvidno, da je ocena vzporednosti (Vz) 13/38 dobljena pod predpostavko, da se Ni približuje neskončnosti in da najbolj globoko gnezdena zanka dominiro nad potekom obdelave

Postopek izračunavanja notranje (inherent) vzporednosti v algoritmu lahko avtomatiziramo tako, da izdelamo simulator. Za potrebe simulatorja predstavimo programski graf v obliki, ki ponazarja izvajanje na strojnem nivoju (9). Časovne vrednosti operacij podamo v celoštevilnih spominskih časovnih ciklov. Tako na primer za seštevanje lahko ugotovimo, da zahteva to opera-

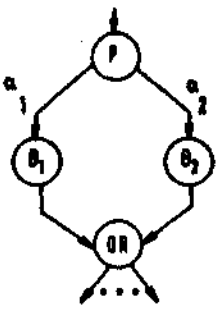
(a) assignment
$\tau_s(AS)$ = sum of machine level operations
$\tau_p(AS)$ = height of parse tree



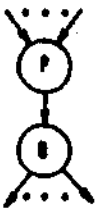
(b) fork, join and OR operators
$\tau(F) = \tau(J) = \tau(OR) = 0$



(c) If P then B
$\tau(COND) = \tau(P) + {}_1\tau(B)$

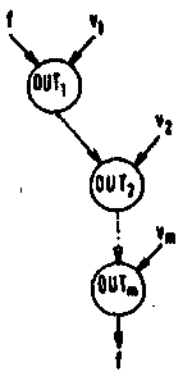


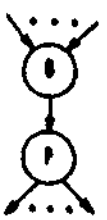
(d) if P then B_1 else B_2
$\tau(COND) = \tau(P) + a_1\tau(B_1) + a_2\tau(B_2)$



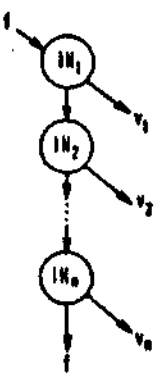
(e) while P do B
$\tau(LOOP) = (N_L + 1)\tau(P) + N_L(B)$

(g) nonrecursive procedure call
A(in $(I_1, \ldots, I_m)$, out $(J_1, \ldots, J_k)$)
$\tau_s(CALL) = m\ t(append) + t(apply) + \tau_s(A) + k\ t(select)$
$\tau_p(CALL) = m\ t(append) + t(apply) + \tau_p(A) + t(select)$





(f) repeat B until P
$\tau(LOOP) = N_L(\tau(B) + \tau(P))$

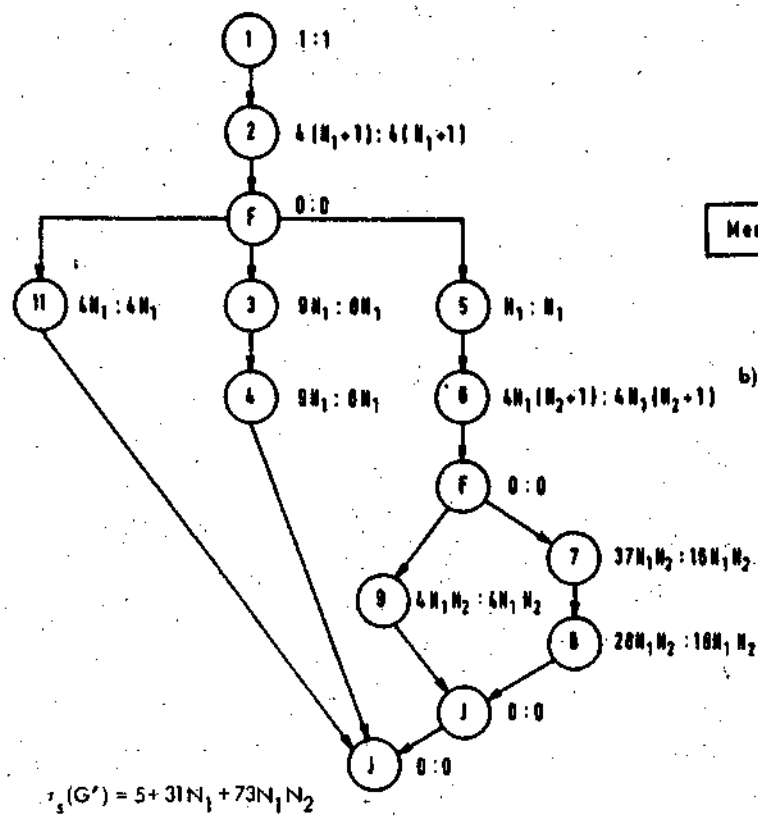


(h) output $v_1, \ldots, v_m$ file = f
format = $(f_1^1, \ldots, f_{k_1}^1, \ldots, f_1^m, \ldots, f_{k_m}^m)$
$\tau(output) = \sum_{i=1}^{m} (OUT_i)$
$\tau(OUT_i) = (k_i - 1)\ t(writeedit) + t(write)$

(i) input $v_1, \ldots, v_m$, file = f
format = $(f_1^1, \ldots, f_{k_1}^1, \ldots, f_1^m, \ldots,$
$\tau(input) = \sum_{i=1}^{n} \tau(IN_i)$
$\tau_s(IN_i) = (k_i - 1)\ t(readedit) + t(read) + 2\ t(select)$
$\tau_p(IN_i) = (k_i - 1)\ t(readedit) + t(read) + t(select)$

Slika 2. Predstavitev časovnega vrednotenja višjih konstruktov

```
1.  I:=1 ;
2.  while I < N do
3.        V(I):=A(I) * B(I) ;
4.        Q(I):=V(I) * C(I) ;
5.        J:=1 ;
6.        while J < N do
7.              E(I,J):=F(I,J) + K(I,J) ;
8.              D(I,J):=E(I,J) * 10 ;
9.              J:= J+1 ;
10.       end ;
11.       I:= I + 1 ;
12. end ;
```



$$\tau_s(G') = 5 + 31N_1 + 73N_1N_2$$

$$\tau_p(G') = 5 + 4N_1 + \max\{4N_1 + 16N_1,\ 5N_1 + 4N_1N_2\} + \max\{4N_1N_2,\ 34N_1N_2\} = 5 + 9N_1 + 38N_1N_2$$

$$\lim V_z(G') = 73/38 \text{ as } N_i \to \infty$$

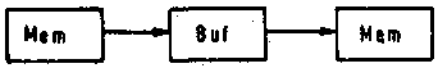
Slika 3. Analiza programa pisanega v visokonivojskem jeziku

Čas izvajanja	Operacije
1	select, append, constant
3	negate, not, readedit
4	+, −, relationals, or, and, writedit
5	abs, arc tan, $\log_e$, sqrt
6	*,/
8	tanh, cosh, sinh, cos, sin, tan
10	read, write
12	arc cos, arc sin, **
14	apply

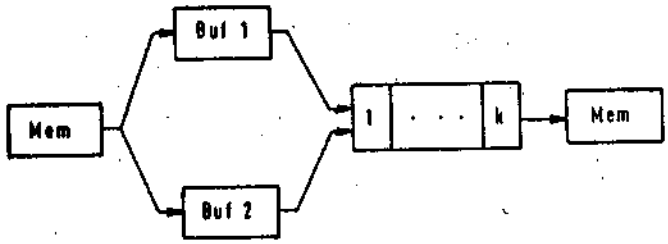
Slika 4. Časi izvajanja za osnovne strojne operacije

cija 4 spominske cikle, enega za izvajanje ukaza "Store", enega za ukaz "Addition" in dva za izvajanje ukazov "Fetch". Za oceno časa kompleksnih operatorjev se lahko usposobijo formule, ki jih je predlagal DeZugish (10).

Rezultati dobljeni z uporabo predlagane metode (12) kažejo na obstoječo vzporednost v programih pisanih v visokem programskem jeziku, in jih kot take lahko uporabljamo, kljub temu, da dobljene vrednosti v časovnem razmerju med vzporednim in zaporednim izvajanjem niso absolutne.



a) Prireditev vrednosti vektorja $t_s^{os} = 1$, $t_w^{os} = 2$



b) Operacija z vektorji $t_s^v = 1+1+k+1 = k+3$, $t_w^v = 2$

Slika 5. Model 2

6. UPORABA MODELA 2

Model 2 predstavlja nadgradnjo modela 1. Za razliko od modela 1, omogoča sprejem in obdelavo vektorsko predstavljenih in zapisanih podatkov. Vsaka vektorska operacija zahteva pred izvajanjem hkratni dostop do vseh operandov.

Pri implementaciji koncepta računalnika krmiljenega s pretokom podatkov operacije z vektorji povzročajo določene težave zaradi problemov pri dodeljevanju spomina in zaradi večkratnega kopiranja podatkov (v primerih ko več vozlišč potrebuje isti podatek).

Model 2, za razliko od modela 1, elemente polj pošilja skozi funkcionalno enoto tako kot kaže slika 5.

Ukaz na strojnem nivoju, določa začetno lokacijo vektorskih operandov v spominu s podatkovnimi strukturami, razširitev operandov ter vrednost inkrementa. Kontrola zanke, ki usmerja pretok vrednosti med spominom in med funkcionalnimi enotami je del logike vgrajene v funkcionalni enoti. Pretok vektorskih elementov iz spomina v posamezne funkcionalne enote in nazaj poteka skozi ena vrata v spominu. Pri tem zanemarjamo vse možne konflikte, ki lahko nastanejo pri vzporednem delovanju več vektorskih funkcionalnih enot.

Čas izvajanja vektorskih operacij je izračunal Ramamoorthy (11):

$$t_s^v + (N-1)t_w^v = t \quad ,$$

kjer je t_s^v čas potreben za pripravo enote, N je dolžina vektorja in t_w^v je enako recipročni vrednosti kapacitete cevasto organiziranih enot. Tako je na primer na sliki 5b t enako:

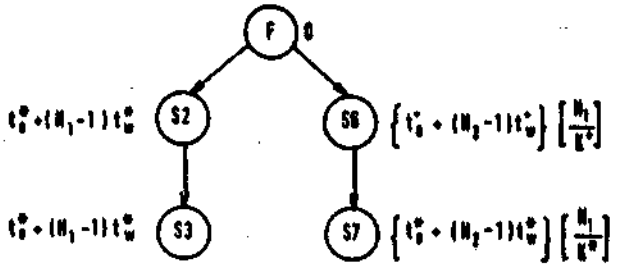
$$t_s^v = 3 + k \quad \text{in} \quad t_w^v = 2 \quad .$$

k označuje številko segmentov v enoti, številka 3 se nanaša na potrebne spominske cikle za dosego operandov in shranjevanje rezultata. Slika 5a kaže poenostavljeno strukturo delovanja in potrebni čas pri prireditvi vrednosti vektorju.

Slika 6 kaže del programa v visokem programskem jeziku (vzporedna inačica programa iz slike 3) in časovno ovrednoteni graf.

```
DO PAR I:=1 TO N1
       V(I):=A(I)*B(I);
       Q(I):=V(I)*C(I);
END PAR;
DO PAR I:=1 TO N1 , J:=1 TO N2
       E(I,J):=F(I,J)+K(I,J);
       D(I,J):=E(I,J)*10;
END PAR;
```

Vzporedna oblika programa iz slike 3



$$\tau_s(G') = 6 + 3lN_1 + 73N_1N_2$$

$$\tau_p(G'') = 2\left\{t_s^* + (N_1-1)t_w^*\right\} + \left[\frac{N_1}{k^+}\right]\left\{t_s^+ + (N_2-1)t_w^+\right\} + \left[\frac{N_1}{k^*}\right]\left\{t_s^* + (N_2-1)t_w^*\right\}$$

b) Programski graf in karakteristične časovne enačbe

Slika 6. Program v visokonivojskem jeziku in njegov programski graf v računalniškem modelu 2

V časovnih enačbah iz slike 6 je upoštevan dodatni dejavnik. Če vozlišče opravi N vektorskih operacij, potem v časovno enačbo za to vozlišče vključimo dejavnik g:

$$g = \left\lceil \frac{N}{K^v} \right\rceil ,$$

kjer K označuje število razpoložljivih oziroma dostopnih funkcionalnih vektorskih enot tipa v. Vrednost g upoštevamo le v primeru uporabe modela 2, oziroma vektorskih procesnih enot. Pri takšni računalniški arhitekturi nastopajo problemi razvrščanja uporabe procesorskih kapacitet s strani neodvisnih podatkovnih poti. Dejavnika g uporabljamo zaradi serijsko zastavljenega načina izvajanja operacij v neodvisnih vozliščih (slika 5b). Če primerjamo skalarni in vektorski model, lahko ugotovimo, da pri vektorskem modelu pridobimo na vzporednosti. Na sliki 6 je ocena te vzporednosti podana z enačbo:

$$V_z(G') = t_{sec}(G')/t_{par}(G'') = 14 \quad \text{za } N1 = N2 = 10$$

in

$$k^+ = k^* = 1 \qquad t_s^+ = t_s^* = 7 \qquad t_w^+ = t_w^* = 2 .$$

Poizkusi, da prevajalnik obdela program, tako kot, da bo obdelava potekala na vektorskem modelu so ostali omejeni na dekompozicijo zank (12).
Vsako zanko, ki vključuje računanje s poljem prevajalnik analizira posebej, in ugotovi ali obstaja neodvisnost v iteracijah. Če taka neodvisnost obstaja, potem se ta transformira v paralelni konstrukt. Če neodvisnosti ni, potem se uporabljajo dodatne tehnike, kot je substitucija, uvajanje začasnih imen polj in konverzija skalarjev v vektorsko obliko in podobno, s ciljem zagotoviti podatkovno neodvisnost. V kolikor poizkus za generiranje podatkovne neodvisnosti uspe, se zanka razdeli na neodvisne dele in vsak del se posebej analizira.

7. ZAKLJUČEK

Programske grafe in karakteristične časovne enačbe smo uporabili za oceno zaporednega in vzporednega časa izvajanja računalniških algoritmov. Pri tem so bila uporabljena dva modela računalniške arhitekture. Prvi model dovoljuje le uporabo skalarnih veličin. Pri oceni zaporednega časa izvajanja je bila upoštevana predpostavka, da je ta čas izvajanja sorazmeren z dolžino najdaljše poti v grafu in z največjo globino gnezdenja zank na tej poti.

8. LITERATURA

1. S. Ribarič, "Računari upravljani tokom podataka", Informatica, 6, No 4, 3 (1982).
2. T. Agervala, M. Arvind, Data flow systems computer, 15, 2, 1982, 15.
3. A. Davis, R. Keller, Data flow program graphs, computer, 15, 2, 1982, 26.
4. D. Martin, G. Estrin, Models of computational systems, IEEE Trans. on Computers, 16, 1, 1967, 70.
5. D. Martin, G. Estrin, Path length computations on graph models of computation, IEEE Trans. on Comp. 18, 6, 1969, 530.
6. R. Russel, The Cray-1 Computer system, CACM, 21, 1, 1978, 63.
7. V. Aho, J. Hopcroft, D. Ullman, The design and analysis of computer algorithms, Add. Wesley, P.C. 1975.
8. J.B. Denis, First version of data-flow procedure language, MAC TM 61, May 1975.
9. A.E. Oldehoeft, Translation of high level programs to data flow and their simulated execution on a feedback interpreter, Comp. Sci., TR, 78-2, Iowa State University, 1978.
10. B. De Lugish, A class of algorithms for automatic evaluation of certain elementary functions in a binary computer, Tec. R. 399, University of Illinois, 1970.
11. C. Ramamoorthy, H. Li, Pipelined architecture, ACM computing surveys, 9, 1, 1977, 61.
12. A. Oldehoeft, R. Zingg, C. Retnadhas, Measurement of parallelism in computer programs through analysis of program graphs, T.R. 78, Iowa State University, 1978.

INFORMATICA 3/1984

PROGRAMIRANJE Z ZBIRKAMI V JEZIKU CBASIC

A. P. ŽELEZNIKAR

UDK: 681.06. CBasic: 519.682

DO ISKRA DELTA

Članek opisuje sintakso in primere stavkov za manipulacijo zbirk v jeziku CBasic (izvedenka jezika Basic za komercialno uporabo). V preprostejših izvedbah jezika Basic se dostikrat pišejo programi, ki se uporabe podatkovnih zbirk ogibajo. Jezik Basic za komercialne namene (CBasic) pe predvideva izdatno uporabo programiranih podatkovnih in drugih zbirk (v primerih generiranja bodo generirani programi oblikovali programske zbirke). Jezik CBasic2 je bil uporabljen v številnih aplikacijah malih poslovnih sistemov, zaradi dovolj velike natančnosti pa je primeren tudi za tehnične aplikacije, ima lastnosti strukturiranih jezikov, racionalno upravljanje pomnilnika, ne potrebuje doslednega oštevilčevanja vrstic, dopušča uporabo niznih in številskih spremenljivk in ima možnost obdelave zbirk z zaporednim in naključnim dostopom. Članek opisuje sintakso in primere stavkov tipa OPEN, CLOSE, CREATE, DELETE, IF END, FILE, READ in PRINT. Dalje so opisani formati zbirčnih PRINT stavkov (s primeri).

Programming in CBasic Using Files

This article deals with CBasic statements and examples of file manipulation. Several file maitaining statements are described (OPEN, DELETE, CLOSE, CREATE, IF END, FILE, READ and PRINT).

1. Uvod

Zbirka (file) je osrednji pojem operacijskega sistema (npr. diskovnega operacijskega sistema CP/M), hkrati pa tudi uporabniški pripomoček (shramba raznovrstnih podatkov) zlasti v programih z množičmo podatkovno obdelavo. Različne izvedbe jezikov tipa Basic uporabljajo zbirčne rutine dostopanja za shranjevanje in razpoznavanje podatkov. V tem članku bomo obravnavali osnove in primere zbirčne organizacije v okviru jezika CBasic2 (s prevajalnikom na sistemu CP/M).

Sistem CP/M vzdržuje vobče imenik zbirčnih krmilnih blokov (FCB za File Control Block) na disku (vinčestrskem) in/ali disketi. FCB vsebuje zbirčno ime, število zbirčnih zapisov in navedbe fizičnih lokacij, ki jih zasedajo zbirčni podatki na disku ([1]). Sistem CP/M je povezan z diskovnimi enotami prek osnovnih sistemskih rutin (BIOS), ki jih uporabljajo tkim. prehodni programi (vključno programi napisani v jeziku CBasic2), ko dostopajo v zbirke. Z osnovnimi rutinami oziroma posebnimi stavki jezika CBasic2 lahko zbirke oblikujemo (CREATE stavek), odpiramo (OPEN stavek), zapiramo (CLOSE stavek), beremo (READ stavek) in v nje zapisujemo (PRINT stavek). Podatki se obdelujejo v segmentih, ki so mnogokratniki 128 zlogov. Nekateri jeziki tipa Basic vzdržujejo vse potrebne kazalce in podatkovne vmesnike (npr. CBasic2), tako da uporabnik ni omejen na zapise (npr. 128 zlogov). Pri zbirčnih dostopih se uporabljajo klici CP/M sistema.

V naslednjih odstavkih si bomo ogledali stavke jezika CBasic2 za zbirčne dostope. Za aktiviranje in deaktiviranje se uporabljajo trije stavki, in sicer CREATE, OPEN in CLOSE. Ko je bila zbirka odprta, je mogoče z READ in PRINT stavkom zbirke brati in v nje zapisovati. Aktivno zbirko je mogoče deaktivirati s stavkoma CLOSE in DELETE.

Jezik CBasic2 je bil uporabljen v številnih aplikacijah malih poslovnih sistemov, primeren pa je tudi za tehnične aplikacije zaradi dvojne dolžine aritmetičnih operandov (14-mestna aritmetika). Obsežna literatura za uporabo jezika CBasic2 opisuje področja glavne knjige ([2]), računovodskih operacij ([3]), plačilnih seznamov z obračunavanjem stroškov ([4]), različnih drugih aplikacij ([5]) itd. Tudi v uporabniških skupinah (CPMUG in SIG/M) se je nabralo veliko uporabniških programov, med drugim pa so bile izdane tudi diskete s celotno programsko opremo knjig ([2, 3, 4]) za sistemo tipa CP/M.

Prevajalnik CBasic2 je izdelek podjetja Compiler Systems Inc., P.O.Box 145, Sierra Madre, CA 91024. Jezik CBasic ima lastnosti strukturiranih jezikov, kot so Pascal, Ada in PL/I. Ima racionalno upravljanje pomnilnika, ne potrebuje oštevilčevanja vrstic kot drugi jeziki tipa Basic, daje dovolj natančne matematične rezultate, dopušča uporabo niznih in številskih spremenljivčnih imen do dolžine 31 znakov (vsi ti znaki so bistveni). Najpomembnejša lastnost jezika CBasic pa je možnost obdelave zbirk z zaporednim in naključnim dostopom. V tem članku bomo opisali raznovrstne možnosti obdelave zbirk v okviru jezika CBasic2.

2. OPEN stavek

OPEN stavek odpre (aktivira) že obstoječo (ne novo) zbirko za branje in pisanje (popravljanje, dodajanje). Splošna oblika tega stavka je

```
[št_stavka] OPEN ime_zbirke
                        [RECL dolžina_zapisa]
AS številka_zbirke [BUFF število_sektorjev
                        RECS število_zlogov ]
{, ime_zbirke [RECL dolžina_zapisa]
AS številka_zbirke [BUFF število_sektorjev
                        RECS število_zlogov]}
```

Imamo tale pomen znakov in sintaksnih kategorij:

[ ]
oglati oklepaj in zaklepaj v paru nakazujeta opcijo (možnost) kategorije med njima, in sicer njeno enkratno ali nobenkratno pojavitev

{ }
zaviti oklepaj in zaklepaj v paru nakazujeta iteracijo (ponovitev) kategorije med njima, in sicer njeno poljubnomnogokratno ali nobenkratno pojavitev

_
znak '_' je kategorijski vezaj

št_stavka
je številka stavka (pred stavkom) v Basic programu

ime_zbirke
je nizni izraz za ime obstoječe zbirke na disku, ko lahko navedemo tudi diskovno enoto, npr. "C:POSKUS.DAT"; ime_zbirke je nizni izraz, ki upošteva CP/M format

dolžina_zapisa
je RECL izraz (RECL je okrajšava za RECord Length), ki določa f i k s n o dolžino zapisov; vrednost tega izraza mora biti pozitivna (sicer napaka v času izvajanja); zbirka je z uporabo RECL pridevka dostopna naključno (neposredno, direktno) ali zaporedno, brez uporabe tega pridevka pa samo zaporedno; RECL izraz mora biti številski (nenizen) in realne vrednosti se pretvorijo v celoštevilske

številka_zbirke
je AS izraz z vrednostjo katerega se priredi odpirani zbirki njena razpoznavna številka; ta vrednost (številka) se uporablja v nadaljnih programskih navedbah zbirke (v drugih zbirčnih stavkih); vsaka aktivna (odprta) zbirka mora imeti enolično prirejeno številko v intervalu (1,20); če vrednost AS izraza ni v tem intervalu, se v času izvajanja pojavi napaka; AS izraz, tj. številka_zbirke mora biti številski (nenizen), realna vrednost pa se pretvori v celoštevilsko

število_sektorjev
je BUFF izraz (BUFF je okrajšava za buffer, tj. vmesnik), katerega vrednost je število diskovnih sektorjev dane zbirke, ki se bodo obravnavali v pomnilniku naenkrat (skupaj); iz definicije je razvidno, da morata biti BUFF in RECS izraz oba prisotna, če pa ju ni, je vzeta vrednost 1 za BUFF izraz; pri n a k l j u č n e m dostopu v zbirko mora biti vrednost izraza število_sektorjev enaka 1, če je bil BUFF izraz uporabljen, sicer se pojavi napaka v času izvajanja; izraza število_sektorjev in število_zlogov sta vselej numerična

število_zlogov
je RECS izraz (RECS je okrajšava za records) in ko se je pojavil BUFF izraz, se mora pojaviti tudi RECS izraz, vendar se njegova vrednost ne upošteva; za prihodnjo uporabo naj bi bila vrednost RECS izraza (tj. število_zlogov) predvidena za opredelitev števila zlogov v sektorju (128, 256 zlogov itd.)

Največ dvajset zbirk je lahko hkrati aktivnih (odprtih). Vmesniški prostor za zbirke se dodeljuje dinamično; pomnilniški prostor se zaseda pri odpiranju in sprošča pri zapiranju zbirk.

Primeri OPEN stavkov so tile:

```
1234 OPEN "B:IKE.TXT" AS 17

     OPEN ime.zbirke$ AS stev.zbirke% \
                  BUFF 26 RECS 128

     OPEN ime.delovne.zbirke$ (tren.zbirka%)\
       RECL del.dolžina% AS tren.zbirka% \
       BUFF vmesnik% RECS 128
```

3. CLOSE stavek

S CLOSE stavkom se odprta zbirka deaktivira, tako da ni več dostopna za vhodne ali izhodne operacije. Splošna oblika tega stavka je

```
[št_stavka] CLOSE številka_zbirke
                        {, številka_zbirke }
```

Pri zaprtju zbirke se sprosti njen vmesniški prostor v hitrem pomnilniku. Če se na zaprto zbirko nanaša IF END stavek (glej kasneje), ta stavek nima učinka. Vse aktivne zbirke se avtomatično zaprejo s STOP stavkom, ali če se vstavi znak 'CTRL-z' z INPUT stavkom. Zbirke se ne zaprejo, če se vstavi znak 'CTRL-c' s konzole, ali če se pojavi napaka v izvajalnem času.

številka_zbirke
je številski izraz v intervalu (1,20) in njegova realna vrednost se pretvori v celoštevilsko

Primeri so tile:

```
926 CLOSE st.zbirke%

    CLOSE nova.glavna.zbirka%, stara.zbirka%,\
          poprava.1983%

    FOR x% = 1 TO stev.zbirke%
      CLOSE x%
    NEXT x%
```

4. CREATE stavek

CREATE stavek je podoben OPEN stavku, vendar se s tem stavkom odpre nova (neobstoječa) zbirka na izbrani diskovni enoti. Splošna oblika tega stavka je

```
[št_stavka] CREATE ime_zbirke
                        [RECL dolžina_zapisa]
AS številka_zbirke [BUFF število_sektorjev
                        RECS število_zlogov]
{, ime_zbirke [RECL dolžina_zapisa]
AS številka_zbirke [BUFF število_sektorjev
                        RECS število_zlogov]}
```

Pomen posameznih izrazov v tej definiciji (št_stavka, ime_zbirke, dolžina_zapisa, število_sektorjev, število_zlogov) je enak pomenu teh izrazov v OPEN stavku.

Če s CREATE stavkom odprta zbirka že obstaja, se njena vsebina zbriše in se začne oblikovati nova zbirka.

Imamo tele primere CREATE stavkov:

```
1532 CREATE "NOVA.ZBI" AS 13 BUFF 4 RECS 256

     CREATE glav.mojster$ RECL g.zap.dol% \
       AS g.zbir.stev%

     CREATE "B:"+ ime$ + left$(str$(t.d%),3)\
       AS t.d%
```

V zadnjem primeru se ime_zbirke pojavlja kot sestavljen nizni izraz.

5. DELETE stavek

DELETE stavek zbriše navedeno zbirko (njeno ime, povezano s številko zbirke) iz zbirčnega imenika na disku. Splošna oblika tega stavka je

```
[št_stavka] DELETE številka_zbirke
                        {, številka_zbirke}
```

številka_zbirke
: je zopet izraz v vrednostnem intervalu (1,20); če v danem trenutku ni prirejena številka aktivni zbirki, se pojavi napaka v času izvajanja; izraz številka_zbirke je številski in pri realni vrednosti nastopi pretvorba v celoštevilsko vrednost; če se številka zbrisane zbirke pojavlja v IF END stavku, nima ta stavek nobenega učinka

Primeri so tile:

```
9200 DELETE 1, 12

     DELETE st.zbirke%, st.izh.zbirke%

     i% = 0
     WHILE i% < st.del.zbirke%
       i% = i% + 1
       DELETE i%
     WEND
```

6. IF END stavek

IF END stavek omogoča obdelavo pogoja konca aktivne zbirke. Splošna oblika tega stavka je

```
[št_stavka] IF END # številka_zbirke
                          THEN št_stavka
```

Ko je bil zaznan konec zbirke, se lahko zgodi dvoje: če je IF END stavek za določeno zbirko bil izveden (to pomeni, da je zbirka aktivna), se prenese programsko krmiljenje na označeni stavek, ki je oštevilčen s št_stavka za rezervirano besedo THEN; če pa IF END stavek ni bil izvršen, se pojavi napaka v času izvajanja.

IF END stavek mora biti sam v programski vrstici. Več IF END stavkov se lahko pojavi v programu za dano zbirko. Z uporabo DELETE ali CLOSE stavka za dano zbirko zgubijo ustrezni IF END stavki svoj učinek.

Za izraz številka_zbirke veljajo podobna pojasnila kot v prejšnjih stavkih.

Če obstaja pogoj, ki povzroči prenos programskega krmiljenja na označeni stavek, se vsebina sklada popravi (obnovi) na pogoj, ki je obstajal pred stavkom, ki je povzročil IF END aktiviranje. Če je bil stavek, ki je rezultiral v prenos krmiljenja, v subrutini, se mora izvršiti RETURN po obdelavi pogoja konca zbirke. Imamo tale primera:

```
     IF END #7 THEN 500

     IF END #st.zbirke% THEN 100.1
```

IF END stavek se lahko izvrši (uporabi) pred prireditvijo imenu zbirke ustrezne številke zbirke (1, 20). Naslednji OPEN stavek za zbirko, ki ne obstaja, bo povzročil učinek, kot da je bil dosežen konec zbirke.

V naslednjem primeru se krmiljenje prenese na stavek s številko 500.5, če zbirke mojster.pod ni na diskovni enoti B: . Po uspešnem odprtju zbirke (OPEN stavek) bo konec zbirke med branjem povzročil nadaljevanje programa s stavkom številka 500. Torej imamo:

```
IF END #stev.mojst.zbirke% THEN 500.5
OPEN "b:mojster.pod" AS stev.mojst.zbirke% \
                                  BUFF 6 RECS 128
IF END #stev.mojst.zbirke% THEN 500
```

IF END stavek se lahko uporabi tudi pri zapisovanju v zbirko. V tem primeru se programsko krmiljenje prenese na stavek, povezan z IF END stavkom, ko se pojavi poskus zapisovanja v zbirko, vendar na disku ni več dovolj prostora. Del zapisa, ki je bil oblikovan, se bo lahko zapisal v zbirko. Pri zbirkah s fiksno organizacijo se zadnji zapis lahko prepiše, ko se sprosti dodatni prostor.

7. FILE stavek

Ta stavek ima obliko

```
[št_stavka] FILE ime_zbirke [(dolž_zapisa)]
                  {, ime_zbirke [(dolž_zapisa)]}
```

ime_zbirke
: je nizna spremenljivka (ne sme biti nizni izraz), ki se ji številka_zbirke priredi kot naslednja prosta številka (začenši z 1); če je vseh 20 številk že prirejenih, se pojavi napaka; spremenljivka ime_zbirke ne sme biti indeksirana, mora biti tipa niz, ne sme biti literal ali izraz

dolž_zapisa
: v oklepajih je številski izraz

Imamo tale primera:

```
     FILE ime$

     FILE ime.zbirke$(dolzina.zapisa%)
```

8. READ stavek

Obstajajo štiri oblike READ stavka za dostopanje v diskovne zbirke. Prva dva tipa READ stavka imata zbirčni dostop, ki je podoben INPUT stavku za podatke s konzole. Druga dva tipa sta podobna INPUT LINE stavku. Splošna oblika za zaporedno branje je tale:

```
[št_stavka] READ #številka_zbirke;
              spremenljivka {, spremenljivka}
```

S tem READ stavkom se bere zaporedno iz navedene zbirke (njej prirejena številka). Zbirka se bere zaporedno polje za poljem v spremenljivke, dokler vsaki od spremenljivk ni bila prirejena vrednost. Polja so lahko celoštevilska, realna in/ali nizna in so ločena z vejicami. Izraz številka_zbirke je številski, njegova realna vrednost se pretvori v celoštevilsko. Vrednost se mora nanašati na aktivno zbirko, sicer se

pojavi napaka v izvajalnem času.

Imamo tale primera:

```
READ #7; nizƒ, stevilo

READ #mojst.zbirka%; imeƒ, naslovƒ,\
          mestoƒ, drzavaƒ, tel.stevilkaƒ
```

Splošna oblika drugega tipa READ stavka je

```
[št_stavka] READ #številka_zbirke,
                    številka_zapisa;
          spremenljivka {, spremenljivka}
```

številka_zapisa
je izraz, s katerim se izbere zapis za branje; n a k l j u č n i zapis, določen s številka_zapisa, se prebere z navedene diskovne zbirke, določene s številka_zbirke; polja zapisa so prirejena spremenljivkam v listi spremenljivk; napaka se pojavi, če je spremenljivk več kot polj v zapisu; izraz številka_zapisa je numeričen in realna vrednost se pretvori v celoštevilsko, vendar ta vrednost ne sme biti enaka 0, je pa lahko v intervalu (1, 65535)

Pri uporabi te bralne oblike je morala biti zbirka aktivirana (odprta) z uporabo RECL člena.

Naključno branje brez določenih spremenljivk postavi dostop v zbirko na izbrani zapis in nadaljnje zaporedno branje se izvrši z dostopom v izbrani zapis. Imamo tale primer:

```
READ #stev.zbirke%, stev.zapisa%;\
     imeƒ, dohodek, ure, prispevki, dopustƒ
```

Naslednji obliki READ stavka obravnavata zbirke kot vrstice besedila. Splošna oblika za zaporedno branje je:

```
[št_stavka] READ #številka_zbirke;
                          LINE spremenljivka
```

Ta stavek prebere zaporedno vse podatke iz navedene zbirke, dokler se ne pojavi znak pomika valja (CR), ki mu sledi znak pomika na naslednjo vrstico (LF). Podatki se preberejo do teh znakov (ne vključno s CR in LF) v LINE nizno spremenljivko. Če ta spremenljivka ni nizna, se pojavi napaka.

Naključna oblika READ LINE stavka je:

```
[št_stavka] READ #številka_zbirke,
          številka_zapisa; LINE spremenljivka
```

Ta stavek prebere zapis, ki je določen s številka_zapisa iz zbirke številka_zbirke. Podatki zapisa se priredijo nizni spremenljivki LINE člena.

READ LINE stavek omogoča dostop do zapisov, ki vsebujejo ASCII podatke v poljubnem formatu, vendar na osnovi vrstica-za-vrstico. Npr. poljubna zbirka, oblikovana s CP/M urejevalnikom, se lahko bere po vrsticah. V primeru

```
READ #13; LINE v.nizƒ
```

se preberejo vsi znaki naslednjega zapisa, dokler se ne pojavita CR in LF. Dodatna primera sta:

```
READ #13; LINE nasl.vrst.besedilaƒ

READ #vhod.zbirka%, zapis%; LINE naslednjiƒ
```

9. PRINT stavek

Štiri vrste PRINT stavkov so predvidene za izdajanje podatkov v diskovne zbirke. Zapisovati je mogoče zaporedno in naključno (neposredno). Prva in druga oblika PRINT stavkov sta:

```
[št_stavka] PRINT #številka_zbirke;
          spremenljivka {, spremenljivka}

[št_stavka] PRINT #številka_zbirke,
                    številka_zapisa;
          spremenljivka {, spremenljivka}
```

Tu je št_stavka številska konstanta, izrazi pa so številka_zbirke, številka_zapisa in spremenljivka (glej prejšnje definicije).

Prva oblika PRINT stavka izda podatke v naslednji zaporedni zapis zbirke, navedeno s številka_zbirke. V zbirko se vpišejo vrednosti spremenljivk kot polja zapisa in te vrednosti so v zbirki omejene z dvojnimi narekovaji in ločene z vejicami; zadnje polje v zapisu je omejeno z znakoma CR in LF.

Izraz številka_zbirke za znakom je številski, pri čemer se realna vrednost pretvori v celoštevilsko. Vrednost izraza številka_zbirke mora navajati aktivno (odprto) zbirko, sicer se v izvajalnem času pojavi napaka.

Druga oblika PRINT stavka izda naključni (neposredni) zapis, katerega lokacija v zbirki številka_zbirke je določena z vrednostjo izraza številka_zapisa. Pomen formata je enak pomenu za zapis v zaporedno zbirko. Zbirka za ta PRINT stavek je morala biti odprta s f i k s n o dolžino zapisa. Napaka se pojavi, če v zapisu ni dovolj prostora za vse podatke. Izraz številka_zapisa mora biti številski in realne vrednosti se pretvorijo v celoštevilske. Vrednost izraza številka_zapisa ne sme biti enaka 0, ker se sicer pojavi napaka v izvajalnem času. Vrednosti številka_zapisa so v intervalu (1, 65535).

Imejmo primere:

```
PRINT #4; "Petra, Micika"

PRINT #stev.zbirke%; imeƒ, dohodek, naslovƒ

PRINT #placa%, stevilka.zaposlenega%;\
          razred(stevilka.zaposlenega%),\
            ure(stevilka.zaposlenega%)

PRINT #13, 61; datum
```

Obravnavani obliki PRINT stavkov oblikujeta zbirke, ki jih je mogoče brati z uporabo READ stavka (glej poglavje 8). Vse vrednosti, ki se pošiljajo v zbirko, so omejene z vejicami ali pa z dvojicami CR in LF. Vsi nizi so zaprti v dvojne narekovaje (").

Kadar želimo izhodne podatke posebej oblikovati (npr. v poslovnih poročilih), se lahko uporabi PRINT USING stavek za diskovne zbirke. Imamo dve splošni obliki:

```
[št_stavka] PRINT USING format;
          #številka_zbirke;
               spremenljivka {, spremenljivka}

[št_stavka] PRINT USING format;
          #številka_zbirke, številka_zapisa;
               spremenljivka {, spremenljivka}
```

S tema stavkoma se vpisujejo podatki v zbirke z uporabo tiskalnih formatnih možnosti (izraz format za rezervirano besedo USING). Formatne možnosti so enake onim za konzolni izpis. Prva stavčna oblika velja za zaporedni zbirčni dostop k podatkom, druga pa za naključni (neposredni) podatkovni dostop. Zapisi so omejeni z dvojico CR in LF.

Izraz 'format' za besedo USING je tipa niz in pojavi se napaka, če se vstavi številski izraz. Če je formatni niz ničeln, se pojavi napaka v izvajalnem času. Pomene ostalih izrazov pa že poznamo iz prejšnjih definicij (številka_zbirke, številka_zapisa, spremenljivka).

Oglejmo si tale primer:

```
STEVILKA.ZBIRKE% = 7
IZDELEK$ = "VRTALNI STROJ"
CENA = 12345.67
CREATE "D:CENIK.DIN" AS STEVILKA.ZBIRKE%
PARE.ZELENE% = 1
FORM1$ = "DIN ##,###.##"
FORM2$ = "DIN ##,###"
111 IF PARE.ZELENE% THEN FORMAT$ = FORM1$ \
   ELSE FORMAT$ = FORM2$
PRINT USING "CENA IZDELKA ""&"" JE "+ FORMAT$;\
    #STEVILKA.ZBIRKE%; IZDELEK$, CENA
IF PARE.ZELENE% THEN PARE.ZELENE% =0: GOTO 111\
   ELSE CLOSE STEVILKA.ZBIRKE%
END
```

Po izvršitvi te procedure dobimo rezultat v dani zbirki (izpis z TYPE ukazom):

```
A>TYPE D:CENIK.DIN
CENA IZDELKA "VRTALNI STROJ" JE DIN 12,345.67
CENA IZDELKA "VRTALNI STROJ" JE DIN 12,346

A>
```

Uporaba dveh zaporednih dvojnih narekovajev v nizu povzroči izdajo enega dvojnega narekovaja v zbirko.

Oglejmo si možnosti uporabe različnih formatov.

9.1. Polje niznega znaka

Enoznakovno nizno podatkovno polje je določeno s klicajem (znakom '!'). Prvi znak vrednosti naslednje spremenljivke v PRINT stavku se izda kot izhod. Npr. za

```
prvo.ime$ = "Anton": drugo.ime = "Pavel"
priimek$ = "Železnikar"
PRINT USING "!. !. &"; #3;\
  prvo.ime$, drugo.ime$, priimek$
```

se izda v odprto zbirko številka 3 zapis

A. P. Železnikar

V tem primeru se znak '.' v formatnem določilu obravnava kot literalni podatek. Iz primera spoznamo pomen oziroma učinek določila. V listi 1 so zbrani vsi primeri (program) poglavja 9, lista 2 pa kaže zaporedje zapisov (vrstice) za te programske primere.

9.2. Nizna polja s fiksno dolžino

Nizno podatkovno polje fiksne dolžine z več kot enim mestom se določi z dvojico ulomkovih črt (znak '/'), ki sta ločeni z nič ali več znaki. Dolžina polja je enaka številu znakov med ulomkovima črtama plus dve. Med črtama je lahko poljuben znak in polnilni znaki se ne upoštevajo (upošteva se le njihovo število).

Nizni izraz (spremenljivka) iz tiskalnega seznama je levo poravnan v fiksnem polju in na desni strani polja dopolnjen s presledki (razvidno iz liste 2). Niz, ki je daljši od podatkovnega polja, se na desni strani odreže. Npr. program

```
format1$ = "Potreben del je /..5....0......7/"
sest.del$ = "avtomobilski zaganjač"
PRINT USING format1$; #17; sest.del$
```

pošlje v aktivno zbirko številka 17 zapis

Potreben del je avtomobilski zaga

(glej drugi zapis liste 2).

Uporaba pik in številk med ulomkovima črtama omogoča verifikacijo dolžine polja (15 + 2 = 17); ti znaki se ne pojavijo v izhodu.

9.3. Nizna polja spremenljive dolžine

Nizno polje spremenljive dolžine se v formatu določa z znakom '&' (glej predprejšnji primer); to ima za posledico, da se izda niz natanko tako, kot je zapisan. Npr. programsko zaporedje

```
podjetje$ = "DO Iskra Delta"
PRINT USING "& &"; #9;\
  "To sporočilo je namenjeno "; podjetje$
```

izda v zbirko s prirejeno številko 9 zapis

To sporočilo je namenjeno DO Iskra Delta

(glej zapis 3 v listi 2).

Niz je lahko desno poravnan v okviru fiksnega polja z uporabo spremenljivega niznega polja. Naslednja rutina kaže primer

```
dolz.polja% = 20
presl$ = "                    "
tel.st$ = "061-211-635"
PRINT USING "# &"; #11;\
  right$(presl$+tel.st$, dolz.polja%)
```

ko se v aktivno zbirko številka 11 pošlje zapis

```
#          061-211-635
 ----------
```

(znaki '-' označujejo presledke v zapisu in seveda niso del zapisa). Ker je v tem tiskalnem seznamu ena sama spremenljivka, se '#' uporabi kot literalni znak. Sicer pa znak '#' označuje številsko podatkovno polje (glej naslednji odstavek). Ta primer je pokazan tudi z zapisom 4 v listi 2.

9.4. Številska podatkovna polja

Številsko podatkovno polje je določeno z znakom

```
D-------------------------------------------------D
B          Preizkus formatov na disku           B
D                 iz poglavja 9                 D
D-------------------------------------------------D

INPUT "Vstavi ime zbirke: "; f$
CREATE f$ AS 1

D--------  Primer iz podpoglavja 9.1 ----------D

prvo.ime$ = "Anton": drugo.ime$ = "Pavel"
priimek$ = "Železnikar"
PRINT USING "!. !. &"; #1; D
  prvo.ime$, drugo.ime$, priimek$
D                                    Zapis  01  D

D--------  Primer iz podpoglavja 9.2 ----------D

format1$ = "Potreben del je /..5....0......7/"
sest.del$ = "avtomobilski zasanjač"
PRINT USING format1$; #1; sest.del$
D                                    Zapis  02  D

D--------  1. primer iz podpoglavja 9.3 --------D

podjetje$ = "DO Iskra Delta"
PRINT USING "& &"; #1; D
  "To sporočilo je namenjeno ", podjetje$
D                                    Zapis  03  D

D--------  2. primer iz podpoglavja 9.3 --------D

dolz.poljaX = 20
presl$ = "
tel.st$ = "061-211-635"
PRINT USING "#&"; #1; D
  right$(presl$+tel.st$, dolz.poljaX)
D                                    Zapis  04  D

D--------  1. primer iz podpoglavja 9.4 --------D

x = 123.7546
y = -21.0
format$ = "####.####   ###.#   ###"
PRINT USING format$; #1; x, x, x
D                                    Zapis  05  D
PRINT USING format$; #1; y, y, y
D                                    Zapis  06  D

D--------  2. primer iz podpoglavja 9.4 --------D

x = 12.345
PRINT USING "#.###ĆĆ    "; #1; x, -x
D                                    Zapis  07  D
PRINT USING "###.##ĆĆĆ"; #1; 17.987
D                                    Zapis  08  D

D--------  3. primer iz podpoglavja 9.4 --------D

PRINT USING "##.###    "; #1; 100, 1000, 10000
D                                    Zapis  09  D

D--------  4. primer iz podpoglavja 9.4 --------D

cena = 8765432.01
PRINT USING "****,******.##    "; #1; cena,-cena
D                                    Zapis  10  D
PRINT USING "####,######.##    "; #1; cena,-cena
D                                    Zapis  11  D

D--------  5. primer iz podpoglavja 9.4 --------D

PRINT USING "###- ###ĆĆĆĆ- "; #1; 10,10,-10,-10
D                                    Zapis  12  D

D--------  6. primer iz podpoglavja 9.4 --------D

PRINT USING "-#####    "; #1; 10,-10
D                                    Zapis  13  D
x = 132.71
PRINT USING "##.#  ###.#"; #1; x, x
D                                    Zapis  14  D

D---------  Primer iz podpoglavja 9.5 ---------D

stev.izdelkaX = 37
PRINT USING "Številka izdelka je D###"; #1; D
  stev.izdelkaX
D                                    Zapis  15  D

CLOSE 1
END
```

Lista 1. V tej listi so zbrani primeri iz podpoglavij 9.1 do 9.5 v obliki programa, katerega izvajanje je prikazano z rezultati v listi 2. Znak ´D´ je nadomestilo za znak ´\´ (lista je bila izpisana z yu-tiskalnikom). Nadalje je znak ´Č´ nadomestilo za znak ´^´ (ostali znaki so regularni). V programu liste 1 se vseskozi uporablja zbirčna številka 1 namesto raznih drugih zbirčnih številk v primerih podpoglavij 9.1 do 9.5 (zaradi enostavnosti programa).

```
A. P. Železnikar
Potreben del je avtomobilski zasa
To sporočilo je namenjeno  DO Iskra Delta
#          061-211-635
 123.7546    123.8    124
 -21.0000    -21.0    -21
1.235E 01    -.123E 02
179.87E-01
   100      1,000    10,000
**8,765,432.01    *-8,765,432.01
 $8,765,432.01     -8,765,432.01
 10  100E-01    10- 100E-01-
    10    -    10
% 132.71    132.7
Številka izdelka je #37
```

Lista 2. Ta lista kaže rezultate izvajanja programa z liste 1. Prikazanih je 15 zapisov, ki izvirajo iz primerov, opisanih v podpoglavjih 9.1 do 9.5. Rezultati so izpisani z yu-tiskalnikom.

´#´, ki označuje vsako številko v rezultatnem številu. Polje lahko vsebuje tudi eno decimalno piko. Vrednosti se zaokrožajo tako, da so prilagojene podatkovnemu polju. Vodeče ničle se nadomestijo s presledki. Če je število negativno, se vpiše pred število znak ´-´. Ena sama ničla se zapiše pred decimalno piko, ko je število manjše od 1. Imamo tale primer:

```
x = 123.7546
y = -21.0
format$ = "####.####   ###.#   ###"
PRINT USING format$; #5; x, x, x
PRINT USING format$; #5; y, y, y
```

Izvajanje tega segmenta povzroči v zbirki s številko 5 dva zapisa, in sicer

```
 123.7546    123.8    124
 -21.0000    -21.0    -21
```

(glej zapis 5 in zapis 6 v listi 2).

Števila se lahko zapisujejo tudi v eksponenčnem formatu s pridružitvijo enega ali več znakov ´^´ na koncu številskega podatkovnega polja. Tako izda segment

```
x = 12.345
PRINT USING "#.###^^^^  ";#12; x, -x
```

v aktivno zbirko številka 12 zapis

```
1.235E 01  -.123E 02
```

(glej zapis 7 v listi 2). Eksponent se nastavi tako, da so vsa zahtevana mesta (znak '#') uporabljena. Npr.

```
PRINT USING "###.##^^^^";#12; 17.987
```

povzroči zapis (zapis 8 v listi 2)

```
179.87E-01
```

Štiri mesta so predvidena za eksponent neglede na število znakov '^', uporabljenih v formatu.

Če se pojavi ena ali več vejic, ki so vgnezdene v numerično podatkovno polje, se število zapiše z vejicami med skupine treh številk pred decimalno piko. Npr. pri

```
PRINT USING "##,###  ";#12;100,1000,10000
```

se pošlje v zbirko s številko 12 zapis (zapis 9 v listi 2)

```
100   1,000   10,000
```

Vsaka vejica, ki se pojavi v podatkovnem polju, se vračuna v dolžino polja. Dejansko je potrebna ena sama vejica, da se vgnezdene vejice pojavijo v zapisu; vendar je pregledneje, če se vejice vstavljajo v podatkovno polje tako, kot naj bi se pojavile v zapisu. Npr. formata

```
#,##########
#,###,###,###
```

povzročita enako obliko zapisa, le da prvi format omogoča izdajo 9, drugi pa 10 številk.

Če se uporabi eksponenčna oblika, se vejice ne zapisujejo; vejice se tu obravnavajo kot znaki '#'.

Z dvema zvezdicama (znak '*') v formatu se lahko dopolni začetek podatkovnega polja. Z dvema dolarskima znakoma (znak '$') se lahko dobi plavajoči dolarski znak. Ekaponenčni format se ne sme uporabljati z zvezdičnim polnilnim ali plavajočim dolarskim znakom. Dvojica zvezdičnih ali dolarskih znakov je vključena v seštevek razpoložljivih mest polja in znaka se v izhodu pojavita le, če je dovolj prostora za število, zvezdični ali dolarski znak. Izdaja dolarskega znaka se zaduši pri negativni vrednosti. Npr. pri segmentu

```
cena = 8765432.01
PRINT USING "**$#,######.##  ";#3; cena,-cena
PRINT USING "$$##,######.##  ";#3; cena,-cena
```

se izdata v zbirko številka 3 zapisa

```
**$8,765,432.01   *-$8,765,432.01
 $8,765,432.01    -$8,765,432.01
```

(glej zapisa 10 in 11 v listi 2).

Število se lahko zapiše s sledilnim namesto z vodilnim predznakom pri negativnem številu; imamo primer

```
PRINT USING "###- ###^^^^- ";#6;\
      10, 10, -10, -10
```

ko se zapiše (zapis 12 v listi 2)

```
10  100E-01   10- 100E-01-
```

Če je minus prvi znak v številskem podatkovnem polju, je položaj predznaka fiksiran z naslednjim zapisnim položajem. Pri pozitivnem številu se natisne presledek na mestu predznaka, sicer pa minus. V primeru

```
PRINT USING "-#####  ";#6;10,-10
```

se zapiše v zbirko številka 6 (zapis 13 v listi 2)

```
10  -10
```

Če število ni prikrojeno številskemu podatkovnemu polju, se natisne znak '%', ki mu sledi število v standardnem formatu. Imamo npr.

```
x = 132.71
PRINT USING "##.#  ###.#";#6; x, x
```

ko se zapiše (zapis 14 v listi 2)

```
% 132.71   132.7
```

9.5. Znaki pobega

Večkrat se pojavi potreba za vključitev znaka kot literalnega podatka, ki bo del podatkovnega polja. Vključitev dosežemo s "pobegom" znaka. Obrnjena poševna črta (znak '\') pred znakom povzroči, da se njej sledeči znak obravnava kot literalni znak. Imamo primer

```
stev.izdelka% = 37
PRINT USING "Stevilka izdelja je \###";\
      6; stev.izdelka%
```

ko se zapiše v zbirko s številko 6 (zapis 15 lista 2)

```
Stevilka izdelka je #37
```

V primeru '\\' se zapiše znak '\'. Če je '\' zadnji znak pobega v formatu, se pojavi napaka v izvajalnem času.

10. Programiranje z zbirkami

Možnosti za dostopanje v zbirke so v jeziku CBasic dokaj raznovrstne, saj je omogočena uporaba različnih zbirčnih organizacij in dostopnostnih metod.

10.1. Zbirčna organizacija

Zbirčna organizacija predpisuje način predstavitve zbirke na pomnilnem mediju (disku, disketi). Vsi v jeziku CBasic zapisani zbirčni podatki imajo znakovni format, ko se uporablja ASCII znakovni kod. Vsebine niznih in številskih spremenljivk se predstavljajo z ASCII znaki in ne kot binarni podatki. Ta način predstavitve omogoča uporabo tako rezidentnih kot prehodnih CP/M programov (ukazov) v povezavi s podatkovnimi zbirkami, nastalimi z uporabo programov v jeziku CBasic.

Z n a k i so v okviru CBasic podatkovnih zbirk hierarhično organizirani. Najnižja ravnina zbirke je p o l j e (field). Skupine polj oblikujejo z a p i s e (records) in z b i r k a (file) je sestavljena iz enega ali več zapisov.

Polje lahko vsebuje nizni ali številski podatek. N i z n o polje je zaprto v narekovaja (znak '"'). Š t e v i l s k o polje ni zaprto v narekovaja in lahko vsebuje poljubno veljavno število. Polja so v zapisu ločena z

vejicami ali na koncu zapisa z znakoma CR in LF.

Jezik CBasic omogoča dve vrsti zbirčne organizacije, in sicer tokovno (stream) in fiksirano (fixed), ki nudita programerju ustrezno (standardno) prožnost.

10.2. Tokovna organizacija

Kadar se pojavi potreba za zaporednim shranjevanjem podatkov, ko predmet (postavka) sledi predmetu (postavki), se uporablja tokovna organizacija. Dostopanje k podatkom temelji v tem primeru na strogi metodi dostopa polja za poljem. Tu ni omejitve vrednosti ali dolžine podatka, ki se zapiše v polje: vsak podatkovni predmet (polje) zavzame le toliko prostora, kot je zanj in za njegove omejevalnike nujno potrebno. Z drugimi besedami: pri tej zbirčni organizaciji nimamo polnjenja praznih mest (s presledki) oziroma nimamo praznih mest.

Del tokovne zbirke, ki vsebuje le nizna polja, lahko ima npr. tole obliko:

```
"prvo polje","drugo polje","tretjé" CR LF
"četrto polje","","123.45" CR LF
"xxx123yyy" CR LF
```

Tu imamo 7 polj in peto polje je prazen niz. V naslednjem primeru imamo številske in nizne podatke:

```
"Petra",100500,"Domžale" CR LF
"Tony Ike",12.34,"Perth" CR LF
```

CBasic omogoča branje zbirk, v katerih nizi niso zaprti v narekovaje. V tem primeru so omejevalniki vejice. Tako se vejice ne smejo pojaviti v nizih, narekovaj, ki je vgnezden v nizu, pa se obravnava kot nizni znak. Nizi, ki se v okviru jezika CBasic vpisujejo v zbirke, so vselej zaprti v narekovaje. Poskus zapisa niza, ki vsebuje narekovaj, ima za posledica napako v izvajalnem času.

PRINT USING stavek ne vstavlja omejevalnikov med polja (to se vidi iz prejšnjih primerov); vsak zapis končuje z znakoma CR in LF.

10.3. Fiksna organizacija

Fiksna (ali fiksirana) organizacija povzroči logično strukturiranje podatkov, povezanih s specifično aplikacijo.

Zbirka ima fiksno organizacijo, če se uporabi možnost določitve zapisne dolžine (dolžine zapisa) v stavkih CREATE, OPEN ali FILE. Vsak (posamezen) podatkovni predmet v fiksni zbirki je bil zapisan kot enostavno polje, ločeno z vejico kot pri tokovni organizaciji, vendar s konceptom fiksne dolžine zapisa. Zapis je vselej omejen z znakoma CR in LF.

Pri vsaki izvršitvi PRINT stavka se v fiksno zbirko zapiše en zapis. Vsak zapis ima natanko tisto število zlogov, ki je bilo določeno z RECL (RECord Length) parametrom neglede na število in obseg zapis sestavljajočih polj. Medtem ko lahko ima polje poljubno dolžino, mora biti vsota dolžin vseh polj v zapisu za dve manjša od dolžine zapisa, tako da je mogoča še shranitev znakov CR in LF na koncu zapisa. Zadnjemu polju zapisa ne sledi vejica.

Imejmo tale primer:

```
CREATE ime.zbirke$ RECL 25 AS st.zbirke%
a$ = "ena"
b$ = "zapis štev ena"
c$ = "3"
d$ = ""
e$ = "pet"
f$ = "abc123def"
PRINT #st.zbirke%; a$, b$
PRINT #st.zbirke%; c$, d$, e$
PRINT #st.zbirke%; f$
```

ko se oblikuje zbirka, v kateri imamo zapise z natanko 25 zlogi (znaki) v zapisu (vrstici), upoštevajoč narekovaje, vejice, polnilne presledke in znaka CR in LF:

```
"ena","zapis štev ena" CR LF
"3","","pet"           CR LF
"abc123def"            CR LF
```

Tako ima prvi zapis en polnilni presledek, drugi jih ima 11 in tretji 12. Zapisni omejevalniki (narekovaji, vejice, znaka CR in LF) zasedejo mesta v zapisu in morajo biti upoštevani v določitvi zapisne dolžine (RECL parameter). V zgornjem primeru je znak LF vselej na 25. mestu zapisa. Nezasedena mesta v zapisu so popolnjena z znaki presledkov.

READ stavek za fiksno zbirko ima vsakič dostop do novega zapisa. Npr.:

```
        IF END #st.zbirke% THEN 100
        WHILE pravilno%
          READ #st.zbirke%; polje$
          PRINT polje$
        WEND
100     STOP
```

Z uporabo podatkov iz prejšnjega primera bi dobili na zaslonu izpis:

```
ena
3
abcdef
```

Fiksna zbirčna organizacija predpostavlja dobro definirano zgradbo podatkov, ki bodo dostopani. Program odloča o pomenu posameznega polja z relativnim položajem polja v zapisu; pri tem je lahko vsebina polja poljubna (tudi pomensko napačna). Ta način zagotavlja prihranek v obdelovalnem času in v programirnem naporu.

Fiksno organizirane zbirke omogočajo hiter in lahek dostop k posameznim poljem v okviru zapisov, saj je vsa polja zapisa mogoče hkrati včitati (brati). Fiksne zbirke se lahko reorganizirajo s sortiranjem po ključu v okviru zapisa. Fiksne zbirke omogočajo uporabo naključnega ali neposrednega (direktnega) dostopanja, kot bo opisano.

Ker se v okviru jezika CBasic bere poljuben zapis na osnovi polja za poljem, je priporočljivo, da imajo vsi zapisi dane zbirke enako število polj. Kadar ne obstaja podatek, ki bi ga vstavili v specificirano polje, lahko vanj zapišemo ničle ali prazen niz. To omogoča npr., da je zasedeno samo peto polje nekega zapisa, prva štiri polja pa se v dani transakciji ne uporabljajo.

Večkrat se zahteva, da začenja določeno polje na stalnem relativnem položaju v okviru zapisa. Navadno obstaja nekaj polj s fiksno in nekaj polj s spremenljivo dolžino polja. Številska polja bodo navadno spremenljiva, čeprav je številski obseg omejen. Nizna polja pa lahko imajo fiksno dolžino, saj se lahko popolnjujejo s presledki.

Imejmo primer

```
niz$ = left$(niz$ + "                    ",20)
```

Ta stavek bo vselej povzročil polje z dolžino

20 znakov. Z uporabo funkcije str$ se lahko števila pretvorijo v nize in dopolnijo s presledki tako, da imajo tudi številski podatki fiksno dolžino.

10.4. Metode zbirčnega dostopanja

Metoda d o s t o p a n j a (dostopni način) opisuje (predpisuje) vrstni red, s katerim se podatki berejo iz zbirke ali zapisujejo v zbirko. Jezik CBasic podpira dva dostopna načina, in sicer zaporednega in naključnega. Oba načina se lahko uporabljata pri fiksno organiziranih zbirkah. Pri tokovno organiziranih zbirkah se lahko uporablja le zaporedni dostop.

10.5. Zaporedni dostop

V zbirkah z zaporednim dostopom je bistveno tkim. n a s l e d n j e polje. S programom se ni mogoče vračati ali preskakovati polja; napredovanje je mogoče le od enega do drugega sosednjega polja.

Opišimo proceduro za zaporedni zbirčni dostop in za izpis (branje) zbirke na zaslon. Zbirka naj ima tele zapise:

```
"prvo polje","drugo polje","tretje" CR LF
"","5","xxx123yyy" CR LF
```

Program naj bo tale:

```
OPEN ime.zbirke$ AS st.zbirke%
WHILE pravilno%
  READ #st.zbirke%; polje$
  PRINT polje$
WEND
```

Na zaslonu se bo pojavilo tole:

```
prvo polje
drugo polje
tretje

5
xxx123yyy
```

Četrta vrstica na zaslonu bo prazna, ker je prvo polje drugega zapisa prazen niz.

Pri zaporednem branju podatkov iz zbirke ugotovi READ stavek konec polja, ko se pojavi vejica ali znak CR. V niznem polju, omejenim z narekovajema, se sme pojaviti poljuben znak z izjemo narekovaja.

Pri dostopu v tokovno zbirko se vsako njeno polje prebere enkrat in nobeno se ne preskoči. Brati je mogoče v več poljih z enim samim READ stavkom. Npr. programski segment

```
WHILE pravilno%
  READ #st.zbirke%; polje.a$, polje.b$
  PRINT polje.a$, polje.b$
WEND
```

izpiše na zaslon z uporabo zbirke iz prejšnjega primera tole:

```
prvo polje          drugo polje
tretje
5                   xxx123yyy
```

Enaka poljska organizacija se uporablja pri zapisovanju v tokovno zbirko. Vsaka spremenljivka, navedena v PRINT stavku, producira eno polje v zbirki. Če se izda več kot ena spremenljivka z enim samim PRINT stavkom, so polja omejena z vejicami, le zadnje polje je omejeno z znakoma CR in LF namesto z vejico.

Imejmo primer

```
a$ = "številka ena"
b$ = "dve"
c$ = "3"
d$ = ""
e$ = "pet"
f$ = "spremenljivka šest"
PRINT #st.zbirke%; a$, b$
PRINT #st.zbirke%; c$
PRINT #st.zbirke%; d$, e$, f$
```

ko se v zbirko st.zbirke% zapiše tole:

```
"številka ena","dve" CR LF
"3" CR LF
"","pet","spremenljivka šest" CR LF
```

Pri zbirkah, ki se berejo ali v katere se zapisuje s tokovno organizacijo, ni bistveno, kateri poljski omejevalnik se uporablja. Znaka CR in LF sta bistvena pri dostopanju s fiksno organizacijo ali pri uporabi READ LINE stavka.

Pri uporabi TYPE ukaza sistema CP/M za prikaz CBasic zbirke se znaka CR in LF pojavita v izhodu in tako se vsebina vsakega PRINT stavka zapiše kot posebna vrstica.

10.6. Naključni dostop

Pri naključnem dostopanju ni programske omejitve za dostop k naslednjemu zapisu ali polju. Vsak zapis zbirke je enako dobro dostopen. Zapis ali njegov položaj se tu navaja z relativno zapisno številko. Vsak zapis lahko vsebuje več polj.

Naključno dostopane zbirke morajo uporabljati f i k s n o organizacijo. CBasic vstavi (namesti) zapis v naključno dostopano zbirko z uporabo relativne zapisne številke, določene s programom, in sicer tako, da odšteje enico od številke zapisa in pomnoži to razliko z dolžino zapisa. Rezultat je zlogovna razdalja zapisa od začetka zbirke. Pri zapisih spremenljivih dolžin te razdalje ne bi bilo mogoče izračunati s takim načinom.

Navadno se naključno dostopane zbirke oblikujejo zaporedno (z zaporednim dostopom) in se nato berejo (prebirajo) in popravljajo z uporabo naključnega dostopa. Primer te vrste obdelave je lahko zbirka uslužbencev v malem poslovnem sistemu. Če ima seznam 20 uslužbencev, se vsakemu uslužbencu priredi paroma različna številka med 1 in 20. Vsak uslužbenec ima svoj zbirčni zapis s polji, ki vsebujejo ime, zavarovalno številko in osebni dohodek. Teh dvajset zapisov se shrani v zbirko v zaporedju uslužbenskih številk z uporabo zaporednega dostopa pri fiksni organizaciji. Kadar aplikativni program potrebuje podatke uslužbenca s številko 12, se lahko uporabi naključno branje zapisa z relativno številko 12. Naslednji program dostopa v zbirko na opisani način:

```
pravilno% = -1
OPEN "usluzben.mst" RECL 50 AS 3
IF END #3 THEN 500.1
WHILE pravilno% REM zanka do pojavitve EOF
  INPUT "Vstavi številko uslužbenca "; st.usl%
  READ #3, st.usl%; ime$, zav.st$, dohodek
  PRINT USING "& ima dohodek  ######.## din";\
    ime$, dohodek
WEND
500.1
 STOP
```

Poudarimo, da READ stavek, uporabljen pri tokovni organizaciji, vselej dostopa v naslednje tekoče polje zbirke neglede na dolžino polja

ali uporabljeni omejevalnik polja. V fiksno organizirani zbirki dostopa READ stavek k naslednjemu zapisu. Zapis je omejen z znakoma CR in LF. PRINT stavki delujejo (učinkujejo) na podoben način.

10.7. Posebne možnosti

PRINT USING stavek se lahko uporablja tako za zapisovanje podatkov v zbirke kot za zapisovanje na zaslon ali s tiskalnikom. Njegova uporaba in njegovi izhodni formati imajo enake učinke pri izpisih v zbirke ali na zaslon. Če je zbirka fiksna, se posamezno zapisno polje pri vsaki izvršitvi PRINT USING stavka dopolni s presledki do specificirane dolžine zapisa. PRINT USING stavek je primeren zlasti za uporabo pri obdelavi besedil.

Naslednji primeri kažejo uporabnost PRINT USING stavkov pri zbirkah:

```
PRINT USING "&";#stev.zbirke%; vrstica.tekst$

PRINT USING "Hitrost = ****.### km/h;\
    izh.zbirka%, cas; hitrost(cas)

ed1$ = "&"
ed2$ = " **.***.**"
PRINT USING ed1$ + ed2$ + ed1$ + ed2$; #17,\
  stev.trans; "Glavnica:", glav, "Obresti:",obr

PRINT USING "&";#tisk.zbirka; " "
  REM  Prazna vrstica

PRINT USING "/2345/"; #del.zbirka,\
  rel.stev.zapisa; sort.kljuc$

in$ = "X"
WHILE in$ <> ""
  INPUT "Vstavi podatek"; LINE in$
  PRINT USING "/...5....0....5../"; #4; in$
WEND
 CLOSE #zacas.zbirka
```

READ LINE stavek omogoča zbirčni dostop po zapisih (kot da je v zapisu eno zapisno polje). Vejice in narekovaji se berejo kot deli podatka. Omejevalnik je le zaporedje znakov CR in LF. Pri dostopu z READ LINE stavkom se zbirka obnaša tako, kot da nima strukture zapisnih polj.

Imejmo primer zbirke

```
"prvo polje","drugo","3","","četrto" CR LF
"peto","šesto" CR LF
```

in uporabimo stavka

```
READ #stev.zbirke%; LINE niz$
PRINT niz$
```

Na zaslonu se bo pojavilo tole:

```
"prvo polje","drugo","3","","četrto"
```

Tu je možna primerjava z učinkom stavkov

```
READ #stev.zbirke%; niz$
PRINT niz$
```

ko dobimo na zaslonu izpis

```
prvo polje
```

Narekovaji in vejice so deli podatkov, znaka CR in LF pa nista.

11. Sklep

Jeziki tipa CBasic (kot je npr. tudi CB-80) so najzmogljivejši Basic jeziki. Njihova uporaba je namenjena prvenstveno malim poslovnim sistemom. Pogosto se srečujemo s CBasic programi v različnih strokovnih časopisih pa tudi v obsežni literaturi, npr. v ((2, 3, 4)). Manipulacije z zbirkami predstavljajo osnovo sodobne mikroračunalniške obdelave podatkov. Poznavanje teh manipulacij je bistveno, ker se jih sicer programerji izogibajo, uporabljajoč princip programiranja po najlažji poti. V dodatku članka je "narisana" še sintaksa zbirčnih programskih stavkov jezika CBasic, ki naj bi začetniku znatno olajšala njihovo uporabo.

Slovstvo

((1)) A.P.Železnikar: Program, ki izlista CP/M dodeljevalne skupine na 8" disketi. Uporabni programi (UP 10). Informatica 7 (1983), št.1, 77 - 78.

((2)) L.Poole, M.Borchers, M.McNiff, R.Thomson: General Ledger - CBasic. Osborne/McGraw-Hill, Berkeley, California, 1979.

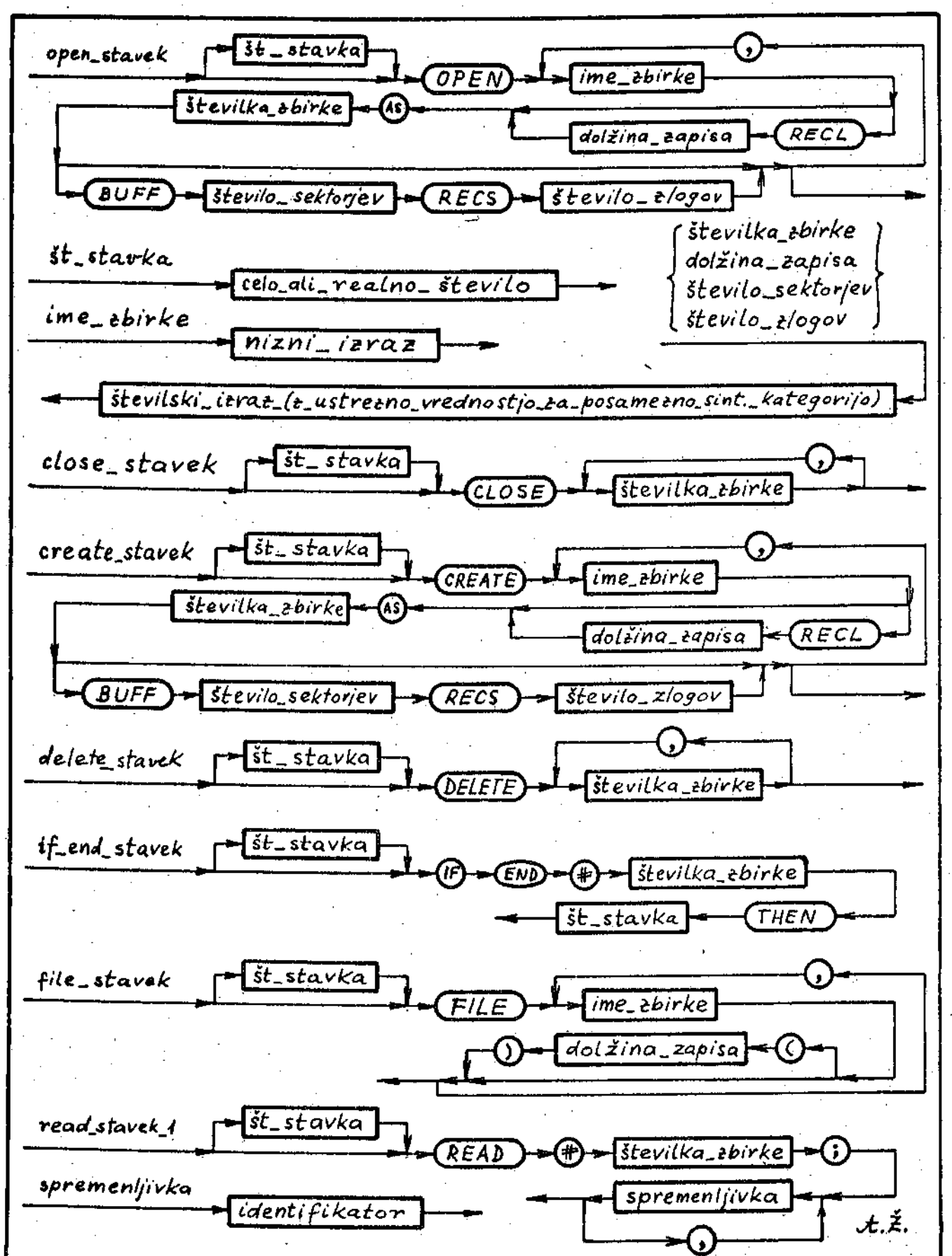
((3)) L.Poole, M.Borchers, M.McNiff, R.Thomson: Accounts Payable & Accounts Receivable - CBasic. Osborne/McGraw-Hill, Berkeley, California, 1979.

((4)) L.Poole, M.Borchers, M.McNiff, R.Thomson: Payroll with Cost Accounting - CBasic. Osborne/McGraw-Hill, Berkeley, California, 1979.

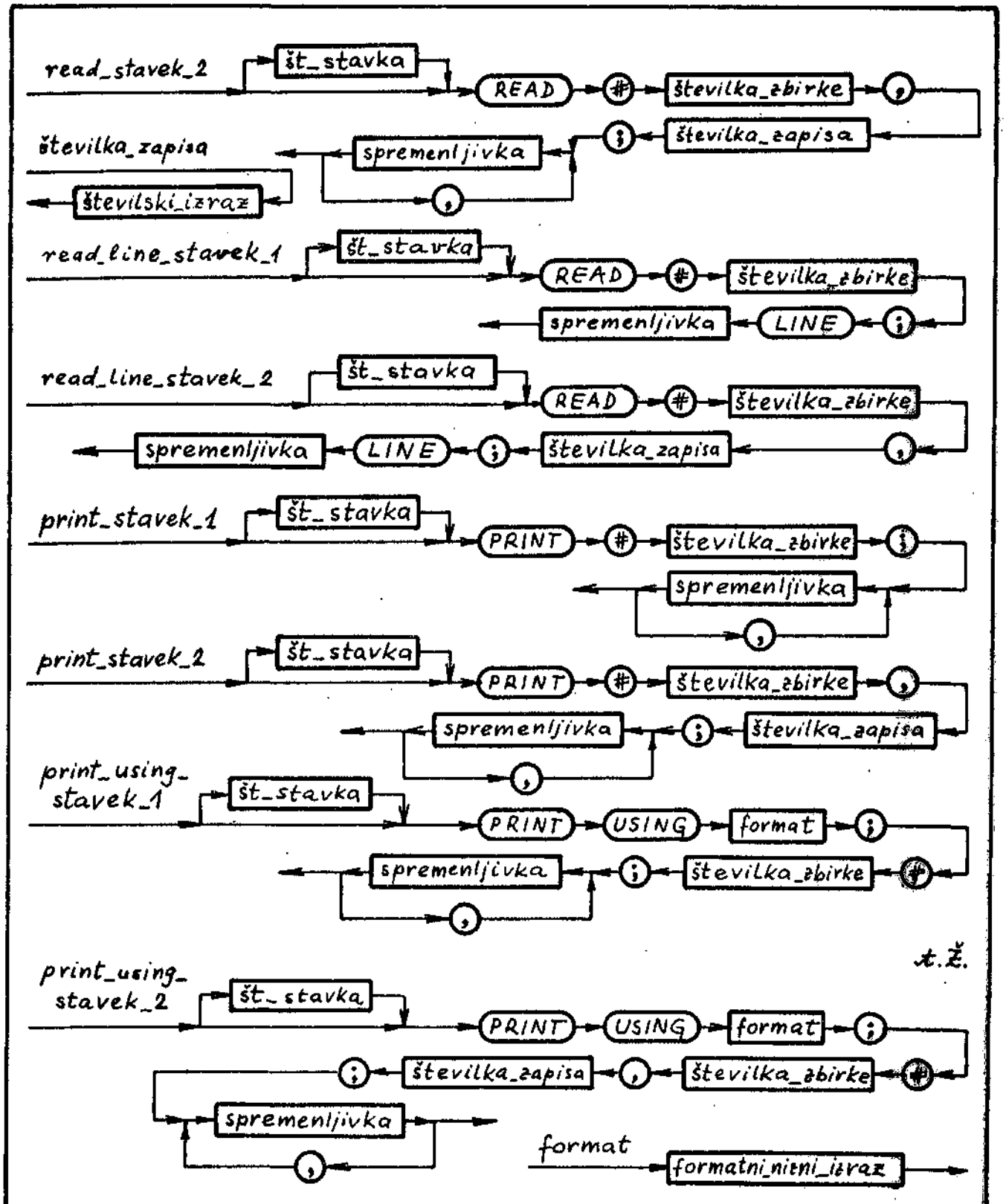
((5)) L.Poole (Ed.), S.Cook, M.McNiff, R.Thomson, R.E.Beckwith, S.H.Westerman: Practical Basic Programs. Osborne/McGraw-Hill, Berkeley, California, 1980.

((6)) J.A.Libertine: The CBasic Clinic I. Creative Computing, November 1983, 214-218.

((7)) T.G.Lewis: How to Profit from Your Personal Computer. Hayden Book Co., NJ, 1978.



DODATEK. Grafi na tej in na naslednji strani prikazujejo sintakso stavkov za odpiranje, oblikovanje, zapiranje, brisanje, branje in pisanje zbirk (sledi nadaljevanje na naslednji strani)



DODATEK (nadaljevanje s prejšnje strani). Sintaksni grafi opisujejo možno zgradbo stavkov za manipulacijo zbirk v jeziku CBasic, in sicer tako, kot je prikazano (v enakem zaporedju) v besedilu članka. Ti grafi so priporočljivi za uporabo pri programiranju zlasti začetnikom, ki se želé navajajo na uporabo teh stavkov jezika CBasic. Grafi opisujejo zaporedje open_stavek, close_stavek, create_stavek, delete_stavek, if_end_stavek, file_stavek, read_stavek_1, read_stavek_2, read_line_stavek_1, read_line_stavek_2, print_stavek_1, print_stavek_2, print_using_stavek_1 in print_using_stavek_2. Pojasnjene so tudi sestavljajoče stavčne kategorije, kot so št_stavka, ime_zbirke, številka_zbirke, dolžina_zapisa, število_sektorjev, število_zlogov, spremenljivka, številka_zapisa in format. Sintaksni grafi kažejo dobro strukturiranost jezika CBasic.

UPORABNI PROGRAMI

```
*                                                    *
*     Generiranje praštevil za vrednotenje           *
*           prevajalnika Janus/Ada                   *
*                                                    *
```

```
#####################################################
#  Informatica UP 14                                #
#  Benchmark Evaluation by Generating Primes        #
#  maj 1984                                         #
#  priredil A. P. Železniker                        #
#  sistem CP/M Plus, Delta Partner                  #
#  prevajalnik Janus/Ada 1.5                        #
#####################################################
```

```
PRAGMA condcomp(on);
PRAGMA rangecheck(off); PRAGMA debug(off);
PRAGMA arithcheck(off);
PACKAGE BODY prime1 IS
--   Iz časopisa Byte Sep 81 Benchmark eval
     size : CONSTANT := 8190;

     flags : ARRAY(0..size) OF boolean;
     count,k,prime1,m : integer;

Ž PROCEDURE tiskpra IS
Ž BEGIN
Ž    m := m + 1;
Ž    IF m = 7 THEN
Ž       m := 0; new_line;
Ž    ELSE put('  ');
Ž    END IF;
Ž    CASE prime1 IS
Ž       WHEN 0..9 => put('   ');
Ž       WHEN 10..99 => put('  ');
Ž       WHEN 100..999 => put(' ');
Ž       WHEN 1000..9999 => put(' ');
Ž       WHEN OTHERS => NULL;
Ž    END CASE;
Ž    put(prime1);
Ž END tiskpra;

BEGIN
Ž       m := 6;
        put('10 iteracij'); new_line; new_line;
deset :   FOR iter IN 1..10 LOOP
        -- Zacetne oznake so vstavljene zaradi
        --    nazornosti
            count := 0;
brisi :     FOR i IN 0..size LOOP
                flags(i) := true;
            END LOOP brisi;
prast :     FOR i IN 0 .. size LOOP
                IF flags(i) THEN
                    prime1 := i + i + 3;
                    k := i + prime1;
                    WHILE k <= size LOOP
                        flags(k) := false;
                        k := k + prime1;
                    END LOOP;
                    count := count + 1;
Ž                   tiskpra;
                END IF;
            END LOOP prast;
        END LOOP deset;
        put(count); put(' praštevil'); new_line;
END prime1;
```

Lista 1. Ta lista prikazuje ocenjevalni program za generiranje praštevil. Program je napisan v jeziku Janus/Ada in uporablja algoritem z Eratostenovim sitom. Z znakom 'Ž' označene vrstice (tj. nadomestek za znak '@') se prevedejo posebno (če je PRAGMA condcomp(on);).

1. Področje uporabe

Generiranje praštevil (njihovih zaporedij) se uporablja pri vrednotenju prevajalnikov oziroma pri vrednotenju kakovosti prevajalniške generiranega strojnega koda, tj. hitrosti izvajanja generiranega programa.

Programi za generiranje praštevilskih zaporedij v različnih programirnih jezikih so bili objavljeni v časopisu Byte ((1)). Podoben program za Ado je bil objavljen v ((2)).

2. Opis programa

Program v listi 1 se drži zamisli izračunavanja desetih zaporedij praštevil z uporabo Eratostenovega sita. Algoritem za ta izračun je bil ob-

Lista 2. Ta lista prikazuje izvajanji dveh izvedb programa PRIME1.PKG. V prvem primeru je uporabljeno PRAGMA condcomp(off);, v drugem pa PRAGMA condcomp(on);. Program generira v desetih ponovitvah po 1899 praštevil (prvi primer). V drugem primeru imamo izpis praštevilske tabele (procedura tiskpra iz liste 1), kjer vidimo začetek tabele in njeno nadaljevanje med prvo in drugo ponovitvijo.

```
13A)b:prime1
10 iteracij

1899 praštevil

13A)
```

```
13A)b:prime1
10 iteracij

     3     5     7    11    13    17    19
    23    29    31    37    41    43    47
    53    59    61    67    71    73    79
    83    89    97   101   103   107   109
   113   127   131   137   139   149   151
   157   163   167   173   179   181   191
   193   197   199   211   223   227   229
   233   239   241   251   257   263   269
   271   277   281   283   293   307   311
   313   317   331   337   347   349   353

 16073 16087 16091 16097 16103 16111 16127
 16139 16141 16183 16187 16189 16193 16217
 16223 16229 16231 16249 16253 16267 16273
 16301 16319 16333 16339 16349 16361 16363
 16369 16381     3     5     7    11    13
    17    19    23    29    31    37    41
    43    47    53    59    61    67    71
    73    79    83    89    97   101   103
   107   109   113   127   131   137   139
   149   151   157   163   167   173   179
   181   191   193   197   199   211   223
   227   229   233   239   241   251   257
   263   269   271   277   281   283   293
   307   311   313   317   331   337   347
```

javljen v ((1)). Na začetku programa v listi 1 imamo stavek PRAGMA condcomp(on)#, ki povzroči, da se v prevajalni postopek vključijo tudi vrstice programa, ki začenjajo z znakom '$' (ta znak je nadomestilo za znak '@'). Vrstice, označene z znakom '$' so povezane z izpisom praštevilske tabele, prikazane v listi 2. Zaradi tega je vsaka vrstica proceduren deklaracije tiskpra tudi označena z znakom '$'.

3. Izvajanje programa

Program iz liste 1 se prevede z uporabo ukaznih vrstic

```
janus primel
jlink primel
```

kjer sta primel zbirki ustreznih tipov. Nova izvedenka prevajalnika Janus/Ada je namreč sestavljena iz prevajalnika in povezovalnika in ima tudi aritmetiko s plavajočo vejico (do 15 decimalnih mest). Lista 2 prikazuje dva rezultata uporabe programa primel.pka, in sicer s PRAGMA condcomp(off) in s PRAGMA condcomp(on), ko se izpisuje (zelo za ilustracijo) tabela praštevil.

Slovstvo

((1)) J.Gilbreath: A High-Level Language Benchmark, Byte, Sep 1981, 180-198.

((2)) A.P.Železnikar: Programiranje v Adi I. Informatica 6(1982), št.3, 10-22.

Množenje matrik za vrednotenje prevajalnika Janus/Ada

```
**********************************************
*  Informatica UP 15                         *
*  Floating Point Matrix Multiply Benchmark  *
*  maj 1984                                  *
*  priredil A. P. Železnikar                 *
*  sistem CP/M Plus, Delta Partner           *
*  prevajalnik Janus/Ada 1.5                 *
**********************************************
```

1. Področje uporabe

Množenje dveh matrik se uporablja pri vrednotenju prevajalnikov, in sicer z dveh vidikov: z vidika hitrosti izvajanja *generiranega programa* in z vidika obsežnosti generiranega koda. Podatki za nekatere prevajalnike (MBasic, CBasic, Pascal/MT+, Pascal-M in Bascom) so navedeni v slovstvu ((1)). Podoben ocenjevalni program lahko napišemo tudi v jeziku Janus/Ada in ocenjujemo pripadajoči prevajalnik.

2. Opis programa

Program v listi 1 začenja s PRAGMA condcomp(on) le v posebnem primeru, ko želimo imeti na zaslonu več informacije. V normalnem primeru vrednotenja imamo PRAGMA condcomp(off). Na začetku navedemo z WITH stavkom knjižnice, ki bodo potrebne paketnemu telesu, bistveno novi sta tu knjižnici floatio in floatops, ki zasotavljata izvajanje aritmetičnih operacij s plavajočo vejico. V primeru večje želene natančnosti (do 15 decimalnih mest) bi navedli knjižnici lonsio in lonsops, kasneje pa bi deklarirali (glej listo 1)

```
SUBTYPE real IS long_float;
```

namesto SUBTYPE real IS float.

Ker prevajalnik za Janus/Ada še nima implemen-

```
13A)bimatmul
Matrix Multiply Benchmark
 a je napolnjeno.
 b je napolnjeno.
 c je napolnjeno.
Matriki sta zmnozeni.
Vsota je :  4.65880E+5
Kontrolna vsota je :  4.65880E+5

13A)
```

Lista 2. Ta lista prikazuje izvajanje prevedenega programa z liste 2, ko se meri čas med dvema sporočiloma. Podatek o obsegu generiranega programskega koda se dobi na zaslonu pri uporabi programa JLINK (povezovalnik).

tiranih večrazsežnostnih polj, lahko definiramo dvorazsežnostno polje kot polje enorazsežnostnih polj. Tako imamo vobče namesto regularne adovske deklaracije

```
TYPE aa IS ARRAY (dim1,dim2) OF real;
```

deklaracijsko zaporedje

```
TYPE stolpi IS ARRAY (dim2) OF real;
TYPE aa IS ARRAY (dim1) OF stolpi;
```

in namesto navedbe elementa

```
aa(i,j)    navedbo    aa(i)(j)
```

Program v listi 1 sestavljajo še procedure za polnjenje matrik (tako da nimamo periferenega vhoda), procedura za množenje in seštevanje. Glavni program je po teh deklaracijah enostaven in razumljiv.

3. Izvajanje programa

Lista 2 prikazuje izvajanje prevedenega programa. Čas izvajanja merimo med sporočiloma "Matrix Multiply Benchmark" in "Matriki sta zmnoženi" na zaslonu. V tem primeru uporabimo seveda PRAGMA condcomp(off) v listi 1. Na koncu liste 2 imamo še kontrolni vsoti, ki se udemata z vrednostjo idealne vsote. Podatke o obsegu generiranega koda dobimo pri uporabi povezovalnika (tj. realni obseg zbirke tipa COM).

```
PRAGMA condcomp(on);
WITH floatio, floatops, util;
PACKAGE BODY matmul1 IS
  -- Floating Point Benchmark
  -- Byte, October 1982, stran 254 - 270
  -- Prevedeno iz Pascala v Janus/Ada

  maxsize : CONSTANT := 45;
  m : CONSTANT := 20;
  n : CONSTANT := 20;

  SUBTYPE real IS float;
  -- Za primerjavo lahko uporabimo long_float
  SUBTYPE dim1 IS integer RANGE 1..m;
  SUBTYPE dim2 IS integer RANGE 1..n;

  -- Ker Janus/Ada se nima implemetiranih vec-
  --  razseznostnih polj, uvedemo tole:
  TYPE col1 IS ARRAY (dim2) OF real;
  TYPE col2 IS ARRAY (dim1) OF real;
  TYPE mat1 IS ARRAY (dim1) OF col1;
  TYPE mat2 IS ARRAY (dim2) OF col2;
  TYPE mat3 IS ARRAY (dim1) OF col2;

  -- Ucinek tega je kot:
  -- TYPE mat1 IS ARRAY (dim1,dim2) OF real;
  -- TYPE mat2 IS ARRAY (dim2,dim1) OF real;
  -- TYPE mat3 IS ARRAY (dim1,dim1) OF real;

  a : mat1;
  b : mat2;
  c : mat3;

  summ : real;

  PROCEDURE fill_a IS
  BEGIN
    FOR i IN dim1 LOOP
      FOR j IN dim2 LOOP
        a(i)(j) := real(i + j);
      END LOOP;
    END LOOP;
  END fill_a;

  PROCEDURE fill_b IS
  BEGIN
    FOR i IN dim2 LOOP
      FOR j IN dim1 LOOP
        b(i)(j) := real((i + j) / j);
      END LOOP;
    END LOOP;
  END fill_b;

  PROCEDURE fill_c IS
  BEGIN
    FOR i IN dim1 LOOP
      FOR j IN dim1 LOOP
        c(i)(j) := 0.0;
      END LOOP;
    END LOOP;
  END fill_c;

  PROCEDURE matrix_multiply IS
  BEGIN
    FOR i IN dim1 LOOP
      FOR j IN dim2 LOOP
        FOR k IN dim1 LOOP
          c(i)(k) := c(i)(k) + a(i)(j)*b(j)(k);
        END LOOP;
      END LOOP;
    END LOOP;
  END matrix_multiply;

  PROCEDURE summit IS
  BEGIN
    FOR i IN dim1 LOOP
      FOR j IN dim1 LOOP
        summ := summ + c(i)(j);
      END LOOP;
    END LOOP;
  END summit;
```

```
--       G l a v n i    p r o g r a m

BEGIN
  summ := 0.0;
  put("Matrix Multiply Benchmark"); new_line;
  fill_a;
  put(" a je napolnjena. "); new_line;
  fill_b;
  put(" b je napolnjena. "); new_line;
  fill_c;
  put(" c je napolnjena. "); new_line;
  matrix_multiply;
  put("Matriki sta zmnozeni."); new_line;
  summit;
  put("Vsota je : "); floatio.put(summ);
    new_line;
  put("Kontrolna vsota je : ");
  floatio.put(summ); new_line;
END matmul1;
```

Lista 1. Ta lista prikazuje program množenja dveh matrik. Ta program se uporablja za ocenjevanje kakovosti prevajalnika jezika Janus/Ada, in sicer hitrosti izvajanja prevedenega programa in obsega generiranega koda. Ta program uporablja aritmetično knjižnico za operacije s plavajočo vejico (enojne ali dvojne dolžine).

Slovstvo

((1)) J.Pournelle: A Basic and Pascal Benchmark, Elegance, Apologies, and Forth, Byte 1982, Oct, str. 254-288.

NOVICE IN ZANIMIVOSTI

Novice s področja mikroprogramske opreme

V letu 1984 je v ZDA predvidena prodaja programske opreme v vrednosti 9,7 milijard dolarjev v primerjavi s 7,4 milijardami v letu 1983. Največja rast te prodaje je značilna za področje osebnih računalnikov, ko imamo:

Prodaja programske opreme (v milijonih dolarjev)			
računalniki	1983	1984	rast
kabinetni sistemi	$ 5200	$ 6400	23 %
miniračunalniki	$ 1400	$ 1900	36 %
osebni računalniki	$ 800	$ 1400	75 %
skupaj	$ 7400	$ 9700	31 %

Največja rast prodaje programske opreme v letu 1984 se pričakuje na področjih

-- elektronske pošte,
-- sistemov za podporo odločanja in
-- s proizvodnjo povezanih funkcij

Za naštete področja bo znašalo povečanje prodaje pri osebnih računalnikih kar krepkih 135 %. Prodaja programske opreme za prenos zbirk, programov in sporočil med kabinetnimi sistemi in osebnimi računalniki se bo podvojila.

Oglejmo si še nekatere razporedelnice s podatki!

OSEBNI RAČUNALNIKI V UPORABI

	procenti računalnikov		
proizvajalec	1982	1983	1984
Apple	27,9	24,7	6,0
Commodore	3,0	1,3	0,4
DEC	4,9	4,6	4,5
Hewlett-Packard	2,8	5,4	1,8
IBM	14,2	48,3	78,9
Radio SHAck	12,9	4,4	1,1
ostali	34,3	11,3	7,3

GLAVNI PROIZVAJALCI PROGRAMSKE OPREME ZA MIKRORAČUNALNIŠKE SISTEME

Kategorija: Razporedelnične pole

procenti instalacij					
1982		1983		1984	
Visicorp	49,2	Visicorp	35,3	Lotus	20,6
Sorcim	14,4	Lotus	19,3	Microsoft	14,0
Radio Sh.	4,5	Microsoft	11,8	Visicorp	13,2
IBM	4,2	Sorcim	9,9	IBM	10,7
Apple	3,5	IBM	3,9	nedoloč.	9,5
ostali	24,2	ostali	19,8	ostali	32,0

Kategorija: Večfunkcijski paketi

procenti instalacij					
1982		1983		1984	
ni podatkov		Lotus	52,5	Lotus	43,6
		IBM	5,9	IBM	7,4
		Visicorp	5,1	nedoloč.	7,4
		Microsoft	4,5	Visicorp	6,4
		ostali	27,8	Microsoft	5,9
				Context	5,3
				ostali	24,0

Kategorija: Računi, ki so plačljivi (r/p)

procenti instalacij					
1982		1983		1984	
ni podatkov		Peachtree	21,3	nedoloč.	25,0
		BPI Syst.	10,6	Peachtree	20,8
		IBM	8,5	State Of The ART	12,6
		Radio Sh.	8,5	BPI Syst	8,3
		Realworld	6,4	IBM	6,3
		TCS Sw	6,4	ostali	25,0

Kategorija: Računi, ki so spejemljivi (r/s)

procenti instalacij					
1982		1983		1984	
ni podatkov		Peachtree	22,0	Peachtree	25,9
		BPI Syst	12,5	nedoloč.	25,9
		IBM	10,0	BPI Syst	14,8
		Radio Sh	10,0	IBM	7,5
		TCS Sw	5,0	ostali	25,9
		ostali	40,0		

Kategorija: Glavna knjiga (g/k)

procenti instalacij					
1982		1983		1984	
ni podatkov		Peachtree	22,0	nedoloč.	28,5
		BPI Syst	12,0	Peachtree	17,8
		IBM	10,0	BPI Syst	14,3
		Radio Sh	8,0	IBM	14,3
		Realworld	6,0	ostali	25,0
		TCS Sw	6,0		
		ostali	36,0		

Kategorija: Integrirani g/k, r/s, r/p

procenti instalacij					
1982		1983		1984	
Peachtree	24,1	Peachtree	26,6	Peachtree	26,2
BPI Syst	12,1	BPI Syst	16,7	BPI Syst	13,0
Radio Sh	8,6	IBM	10,6	nedoloč.	13,0
Realworld	8,6	nedoloč.	6,7	IBM	8,7
ostali	46,6	Realworld	6,7	State Of The Art	8,7
		ostali	33,3	ostali	30,4

Kategorija: Plačilni seznami/kadri

procenti instalacij					
1982		1983		1984	
Peachtree	29,0	Peachtree	20,9	nedoloč.	27,3
Apple	9,7	IBM	12,6	IBM	13,6
BPI Syst	6,5	BPI Syst	8,3	Peachtree	9,1
Radio Sh	6,5	Realworld	8,3	Sw Publ	9,1
ostali	48,3	Sw Publ	8,3	ostali	40,9
		TSC Sw	8,3		
		ostali	33,3		

Kategorija: Poslovna grafika

procenti instalacij					
1982		1983		1984	
Hew-Pack	18,8	Lotus	21,1	nedoloč.	26,0
Apple	17,3	Visicorp	15,2	Lotus	15,1
Visicorp	13,5	Hew-Pack	10,3	IBM	9,6
Tektronix	12,8	Apple	8,8	Hew-Pack	5,5
Radio Sh	6,0	IBM	6,9	Visicorp	5,5
ostali	31,6	ostali	37,7	Wang	5,5
				ostali	32,8

Kategorija: Povezava kabinetnih sistemov z osebnimi računalniki

procenti instalacij					
1982		1983		1984	
		IBM	25,8	nedoloč.	31,5
ni		Microstuf	8,7	IBM	28,0
podatkov		DEC	7,9	Wang	3,5
		Headlands	4,8	Hayes	2,5
		Apple	4,4	ostali	32,0
		TAC	4,4		
		ostali	44,0		

Kategorija: Obdelava besedil

procenti instalacij					
1982		1983		1984	
MicroPro	26,7	MicroPro	28,3	IBM	15,1
IBM	11,8	IBM	12,9	nedoloč.	14,6
Apple	9,7	Apple	6,7	MicroPro	14,2
Radio Sh	8,1	Softword	4,9	Wang	5,9
Wang	5,9	Wang	4,7	Microsoft	5,0
ostali	37,8	ostali	42,5	ostali	45,2

Kategorija: Pravopisni preizkus

procenti instalacij					
1982		1983		1984	
		MicroPro	25,5	nedoloč.	20,4
ni		IBM	16,5	MicroPro	12,5
podatkov		Wang	5,7	IBM	11,4
		Peachtree	4,7	Hew-Pack	4,5
		Radio Sh	4,2	Wang	4,5
		ostali	43,4	ostali	46,7

Kategorija: Elektronska pošta

procenti instalacij					
1982		1983		1984	
DEC	15,4	IBM	15,9	nedoloč.	39,4
Apple	7,7	Wang	10,1	IBM	18,2
IBM	7,7	DEC	5,8	DEC	4,5
MicroPro	7,7	MicroPro	5,8	Sperry Un	4,5
Radio Sh	7,7	Apple	4,3	Xerox	4,5
ostali	53,8	Microstuf	4,3	ostali	28,9
		ostali	53,8		

Kategorija: Elektronsko zbiranje

procenti instalacij					
1982		1983		1984	
Radio Sh	15,6	Ashton-Tate	17,2	nedoloč.	17,2
Apple	13,3	IBM	9,0	IBM	15,6
MicroPro	13,3	SW Publ	9,0	SW Publ	11,0
Visicorp	13,3	Apple	6,7	Ashton-Tate	9,5
SW Publ	8,9	DEC	6,7	Condor	6,3
ostali	35,6	ostali	51,4	DEC	6,3
				ostali	34,1

Kategorija: DBMS

procenti instalacij					
1982		1983		1984	
Ashton-Tate	40,0	Ashton-Tate	53,9	Ashton-Tate	36,4
Stoneware	7,3	Condor	7,2	nedoloč.	17,5
Condor	6,4	IBM	3,4	Lotus	5,8
Apple	5,5	Lotus	3,4	Condor	5,2
Visicorp	4,5	Radio Sh	2,8	IBM	5,2
ostali	36,3	ostali	29,3	Microrim	5,2
				ostali	24,7

Kategorija: Operacijski sistemi

procenti instalacij					
1982		1983		1984	
		IBM	25,9	Microsoft	22,6
ni		Microsoft	21,1	IBM	17,4
podatkov		Disital Research	17,9	nedoloč.	15,7
		Apple	9,9	Digital Research	13,0
		DEC	7,2	DEC	7,8
		ostali	18,0	ostali	23,5

Vzroki za centralizirano prodajo mikroračunalniške aparaturne in programske opreme

	odgovori
standarizacija	76 %
količinski popusti	70 %
boljši nadzor prodaje	50 %
boljša obveščenost	49 %

Kriteriji za izbiro mikroračunalniške programske opreme

	odgovori
dokumentacija	74 %
uporabna enostavnost	72 %
zmogljivosti	61 %
podpora, servis	52 %
cena	19 %
ugled proizvajalca	15 %

Kje kupujejo podjetja mikroračunalniško programsko opremo

	odgovori
proizvajalec programske opreme	67 %
računalniški dobavitelj	54 %
prodaja na drobno	48 %
sistemska hiša	23 %
naročilo po pošti	18 %

Povezljivost osebnih računalnikov na kabinetne sisteme

instalirane enote v letu 1983	44 %
planirane enote v letu 1984	67 %

Rast tržišča programske opreme v letu 1984

aplikativna programska oprema	
kabinetni sistemi	28 %
miniračunalniki	46 %
mikroračunalniki	60 %
sistemska programska oprema	
kabinetni sistemi	17 %
miniračunalniki	28 %
mikroračunalniki	45 %

Rast uporabniške baze v letu 1984 pri uporabi kabinetnih sistemov

elektronska pošta	56 %
podpora odločanju	42 %
MRP	41 %
prodajno upravljanje	39 %
načrtovanje proizvodnje	38 %

Rast uporabniške baze v letu 1984 pri uporabi miniračunalnikov

srednje načrtovanje proizvodnje	83 %
MRP	79 %
podpora odločanju	77 %
prodajno upravljanje	67 %
elektronska pošta	65 %

Rast uporabniške baze v letu 1984 pri uporabi (osebnih) mikroračunalnikov

prodajno upravljanje	190 %
elektronska pošta	130 %
načrtovanje proizvodnje	125 %
MRP	120 %
nadzor inventarja	113 %

Prodaja programske opreme v milijardah dolarjev v razdobju 1983 do 1988 za kabinetne sisteme (ks), miniračunalnike (mm) in osebne računalnike (or) v ZDA

	ks	mm	or
1983	5,1	1,4	0,8
1984	6,3	1,9	1,2
1985	7,9	2,6	2,4
1986	10,2	3,2	3,8
1987	12,6	4,2	5,6
1988	15,0	5,0	8,0

A. P. Železnikar

Mikroprocesor z vgrajenim operacijskim sistemom CP/M

Podjetje American Microsystems je razvilo mikroprocesor z operacijskim sistemom (v enem integriranem vezju), tako je dodalo 8k-zložni ROM in krmilno logiko k mikroprocesorju. Prva izvedba integriranega vezja ima oznako S83. ROM v tem integriranem vezju pa vsebuje operacijski sistem CP/M (podjetja Digital Research). Natančna oznaka tega operacijskega sistema je Personal CP/M in za razliko od navadnega sistema CP/M uporablja ta sistem ukaze v angleščini, ima pa tudi izboljšano uporabniško prijaznost. Procesor tega novega integriranega vezja je Z80.

Vezje S83 je obdržalo vse krmilne, naslovne in podatkovne signale in ukaze procesorja Z80, tako da je mogoča uporaba ostalih integriranih vezij družine Z80. Dodatna logika novega procesorja pa omogoča neposredno priključitev 64k-bitnih dinamičnih pomnilnikov.

K 7-bitnemu osveževalnemu naslovnemu števniku je bil dodan še osmi bit. Tako je omogočeno neposredno osveževanje 64k-bitnih dinamičnih pomnilnih vezij z multipleksiranjem osmih naslovnih bitov. Novo vezje generira tudi koincidenčne signale vrstičnih in stolpnih naslovov (RAS in CAS signali). Notranji ROM se lahko vključuje ali izključuje programsko.

Novi procesor se proizvaja v 48-ničnem ohišju zaradi dodatnih pomnilniških krmilnih signalov, v NMOS tehnologiji in s 5V napajanjem. To vezje je prvo iz družine, ki ima operacijski sistem vgrajen v lastnem ROMu. Vzorci tega procesorja so že dobavljivi, proizvodnja pa se bo začela v drugi polovici letos. Okvirna cena bo $32 za OEM proizvajalce. Naslov proizvajalca: American Microsystems Inc., 3800 Homestead Road, Santa Clara, CA 95051 (Tom Dugan).

A. P. Železnikar

Honeywell bo uporabil procesor NCR/32 v novi računalniški generaciji

V svoji novi računalniški generaciji bo podjetje Honeywell uporabilo mikroprogramirani 32-bitni mikroprocesor NCR/32 (sled novice v prejšnji številki časopisa Informatica). Ta mikroprocesor (NCR/32) naj bi postal osnova Honeywellovih srednjevelikih kabinetnih sistemov, ki se bodo pojavili na tržišču v letu 1986. Razvoj na izpopolnitvah tega procesorja bo skupen za obe podjetji (NCR in Honeywell).

Gre za prvi primer tovrstnega sodelovanja dveh proizvajalcev kabinetnih sistemov, ki se povezujeta pri razvoju in proizvodnji komponentne tehnologije. Pričakuje se, da bodo tudi drugi proizvajalci uporabili procesor NCR/32 zaradi mikroprogramirnih možnosti. Ta procesor sestavljajo štiri integrirana vezja, njegova proizvodnja pa se je začela že pred skoraj dvemi leti.

Za razliko od 32-bitnih procesorjev, kot sta M68020 in 80386, je pri NCR/32 omogočeno zunanje mikroprogramiranje in s tem emuliranje obstoječih računalniških sistemov; prav to je

možnost, zaradi katere se je Honeywell odločil za procesor NCR/32. Procesor NCR/32 bo podprt tudi z ostalimi integriranimi vezji in njegova zmogljivost bo dosegla 1 MIPS.

A. P. Železnikar

Jeziki in prevajalniki za umetno inteligenco

Raziskave v umetni inteligenci (UI) so namenjene razvoju programov, ki bi naredili računalnike pametnejše. Iščejo se računalniški pripomočki oziroma pristopi za njihovo inteligentno obnašanje. Reševanje teh nalog zahteva najboljšo (vrhunsko) programsko opremo in seveda programirna orodja. Ta orodja so nastajala vzporedno s potrebami UI. Programi za UI so navadno simbolični procesi, za katere ne obstajajo algoritmične rešitve: potrebno je iskanje. UI opisuje tako tipe reševanja problemov in odločanja, ki se pojavljajo v človekovem miselnem svetu. Ta oblika reševanja problemov se bistveno razlikuje od znanstvenih in tehničnih izračunov, ki so pretežno številski in za katere so rešitve znane.

V naslednjih nekaj letih se bo uporaba UI pri obdelavi naravnih jezikov, računalniškega videnja, reševanja problemov in izvedenskih sistemov že lahko premaknila iz laboratorijev. Uporabniki teh bistveno različnih programov bodo potrebovali temeljitejše razumevanje jezikov, orodij in računalnikov pri razvoju inteligentnih aplikacij.

Programirne potrebe UI

Programi za UI se navadno razvijajo iterativno (ponavljajoče) in inkrementno (po korakih). Njihovo oblikovanje zahteva interaktivno okolje z vsrajenimi pripomočki, kot je dinamično dodeljevanje računalniškega pomnilnika, ko programi naraščajo. Tudi nepredvidljive vmesne podatkovne oblike (ko programi naraščajo) vplivajo na obliko programirnih jezikov in na pomnilniško upravljanje. Drugi vidik programiranja v UI je rekurzivna izražava funkcij (ki so definirane v odvisnosti njih samih), ki znatno poenostavi pisanje programov. Jeziki za UI podpirajo rekurzivno obdelavo pri simbolni manipulaciji.

Barr in Feigenbaum ((1)) ugotavljata, da so programi za UI med največjimi in najbolj zapletenimi računalniškimi programi, ki so bili kadarkoli razviti in povzročajo velikanske oblikovalne (načrtovalne) in implementacijske probleme. V UI se je razvil poseben interaktivni programirni način v okoljih z izdatno podporo, kot so urejevalniki, zasledovalniki izvajanja in iskalniki napak, pripomočki za razvoj velikih zapletenih sistemov itd.

Tem potrebam UI zadoščata danes dva programirna jezika, in sicer

Lisp in Prolog

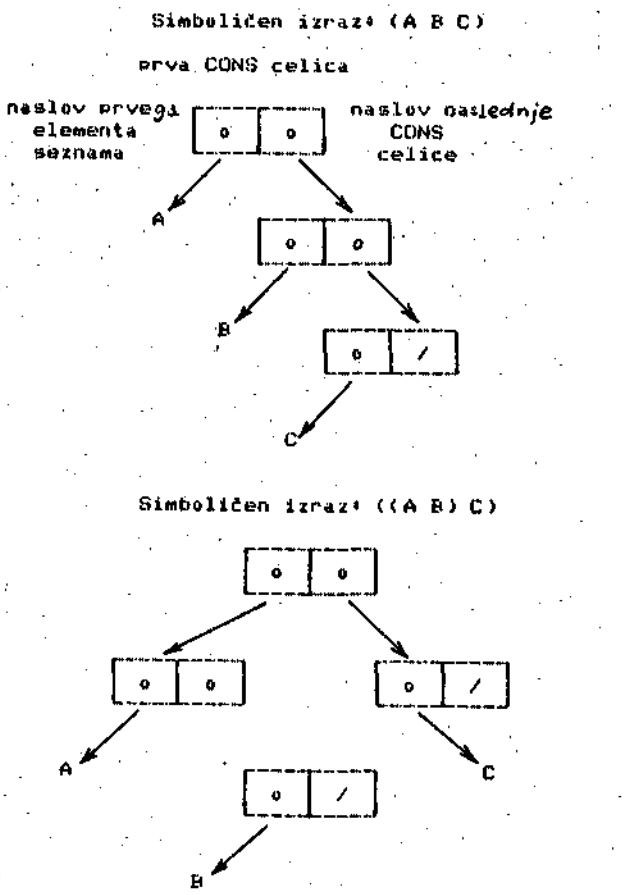
Lisp je bil osnovni programirni jezik UI. Prolog se je kot logično osnovan jezik pojavil pred kratkim in se je usidral predvsem v Evropi in na Japonskem. Obstaja vrsta izpeljank in načrtij jezika Lisp. Posebni visoki programirni jeziki za namene, kot so predstavitev znanja in oblikovanje izvedenskih sistemov, so se razvili kot nadaljevanja jezika Lisp.

V preteklih letih so bili skoraj vsi programi za UI razviti na računalnikih System-10 in System-20 podjetja DEC. V zadnjem času se ti programi prenašajo na računalnike tipa VAX (DEC) in na nove osebne računalnike za UI.

Predstavitev seznamov

Obdelava seznamov (angleško 'list') je bila prvotno uvedena v programirnem jeziku IPL ((2)) za namene simbolne manipulacije. Seznami oblikujejo simbolne združitve, ki omogočajo računalniškim programom, da oblikujejo podatkovne strukture nepredvidljivih oblik in obsegov. Za obdelavo takih nepredvidljivo oblikovanih podatkovnih struktur so se v jeziku IPL uporabljali primitivni podatkovni elementi (imenovani celice).

Podobna ideja se uporablja v jeziku Lisp v obliki tkim. CONS celic. Vsaka CONS celica je naslov (računalniška beseda), ki vsebuje dvojico kazalcev k drugim lokacijam v računalniškem pomnilniku. (Tako se lahko predpostavi, da je osnovni podatkovni element v jeziku Lisp kazalec s seznamom). Levi del celice kaže na prvi element (tkim. 'CAR') seznama, desni del celice pa kaže na drugo CONS celico, ki predstavlja ostanek (tkim. 'CDR') seznama. Iz slike 1 je razvidno, kako je mogoče predstaviti zaporedje besed ali simbolov z uporabo binarne drevesne strukture pomnilniških celic. Problem nepredvidenega obsega podatkovnih struktur je bil rešen s svobodnim seznamom pomnilniških celic, ki se dodeljujejo dinamično po potrebi.

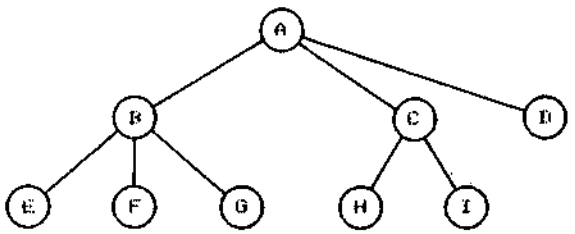


Slika 1. Predstavitev seznamskih struktur v pomnilniku

Seznam je zaporedje nič ali več elementov, zaprtih v oklepaje, kjer je element ali atom (nedeljiv element) ali seznam. Seznami se lahko uporabljajo za predstavljanje poljubnih podatkovnih tipov. Seznami so tako zlasti uporabni na tistih področjih UI, ki obravnavajo jezikovno razumevanje, računalniško videnje, reševanje problemov in načrtovanje.

Tri strukture (ki se uporabljajo za predstavitev iskalnih prostorov) so v UI vselej navzoče (ubikvitarne). Predstavitev seznama za tri strukture je prikazana na sliki 2. Rezultatna predstavitev je seznam (kot je določeno z oklepaji), sestavljen iz elementov, od katerih so nekateri tudi seznami. Venezdene strukture so za seznamsko predstavitev običajne.

Simboličen izraz: (A (B E F G) (C H I) D)



(B E F G), (C H I) in D so poddrevesa korenskega vozlišča A

Slika 2. Seznamska predstavitev iskalnega drevesa

Tudi izrazi predikatne logike, kot je izraz

IN(x,A) OR IN(x,B),

ki pomeni, x je v A ali v B, se lahko opišejo z uporabo prefiksne notacije v seznamski obliki, in sicer

(OR (IN x A) (IN x B))

Jezik Lisp

Okoli leta 1960 je John McCarthy na MIT razvil jezik LISp za praktično obdelavo seznamov, z rekurzivnimi možnostmi opisovanja procesov in problemov. Vsi lispovski programi in podatki imajo obliko simbolnih izrazov (tkim. S-izrazov), ki se shranjujejo v seznamskih strukturah. Lisp pozna dve vrsti objektov: atome in sezname. Atomi so simboli (konstante ali spremenljivke), ki se uporabljajo kot identifikatorji za poimenovanje objektov, ki so lahko številski ali neštevilski (ljudje, reči, ideje, roboti itd.). Seznam je zaporedje nič ali več elementov, zaprtih v oklepaje, kjer je vsak element ali atom ali seznam.

Graham ([3]) je ugotovil, da je Lisp sistem stroj za izračun funkcij. Uporabnik tega sistema vstavi funkcijo in njene argumente. Seštevanje se opravi takole:

```
Uporabniški vhod: (PLUS 6 2)
Lisp odgovor:     8
```

Pri manipulaciji seznamov se v jeziku Lisp uporabljajo tri osnovne funkcije (ki so v relaciji s pomnilno celično strukturo za sezname):

CONS za pridružitev novega prvega člena k seznamu

CAR za razpoznavanje prvega člena v seznamu

CDR (izgovarjava 'kood-er') za razpoznavanje seznama, v katerem so vsi členi seznama razen prvega

Imejmo:

```
Uporabnik: (CONS 'Z '(C D E))
Lisp:      (Z C D E)
```

Enojni narekovaj (') se uporablja za indikacijo, da naslednji element nima izračunanih svojih členov. Lisp namreč vstavi vrednosti za vse člene v izrazu, začenši z najbolj notranjimi oklepaji in šele potem izvede bolj zunanje operacije. Npr.

```
Uporabnik: (CAR '(Janez Micka X Y))
Lisp:      Janez
Uporabnik: (CDR '(Janez Micka X Y))
Lisp:      (Micka X Y)
```

S p r e m e n l j i v k e . SET funkcija priredi vrednost spremenljivki v jeziku Lisp. Imejmo:

```
Uporabnik: (SET 'Z Petra)
Lisp:      Petra
Uporabnik: Z
Lisp:      Petra
```

Atomi se v Lispu uporabljajo za spremenljivke. Atom, ki ima pred seboj enojni narekovaj, predstavlja samega sebe; sicer pa Lisp avtomatično vstavi njegovo vrednost.

D e f i n i r a n j e n o v i h f u n k c i j . Programiranje v Lispu je povezano z definicijo novih funkcij. Tako lahko definiramo DRUGI (kot drugi atom v seznamu) z

```
Uporabnik: (DEFUN DRUGI (Y) (CAR (CDR Y)))
           kjer je Y formalna spremenljivka
Lisp:      DRUGI
Uporabnik: (DRUGI '(Ivan, France, Petra, Jana))
Lisp:      France
```

P r e d i k a t i . Predikat je funkcija, ki vrne ali NIL (nepravilno) ali T (pravilno). Poljubna ne-NIL vrednost je praviloma pravilna. NIL je dejansko ime za prazen seznam. Tako predikat GREATERP vrne T, če so členi v zaporedju razvrščeni v padajočem vrstnem redu:

```
Uporabnik: (GREATERP 6 5 2)
Lisp:      T
```

P o g o j n a v e j i t e v . Uporaba pogojne vejitve je večkrat potrebna. Npr., če je nekaj pravilno, potem opravi X; če ni pravilno, opravi Y; če ni pravilno, opravi Z. V Lispu ima to lastnost COND funkcija in njena oblika je:

```
(COND (pogoj 1   izraz 1)
      (pogoj 2   izraz 2)
      . . . . . .
      (pogoj m   izraz m))
```

kjer je vsak pogoj izraz, ki se bo izračunal na vrednost NIL ali kakšno drugo vrednost. COND funkcija izračunava pogoje po vrstnem redu, dokler se za edega od njih ne izračuna vrednost NIL. V tej točki se vrne vrednost ustreznega (pripadajočega) izraza.

R e k u r z i v n e f u n k c i j e . Večkrat je lažje definirati funkcijo rekurzivno (v odvisnosti od nje same) kot pa eksplicitno z ustreznimi koraki. Ta rekurzivna lastnost je pomembna karakteristika jezika Lisp. Funkcija za faktorial je lep primer:

$$N! = \begin{cases} 1 & \text{če je } N = 1 \\ N * (N-1)! & \text{če je } N > 1 \end{cases}$$

To funkcijo lahko v Lispu definiramo kot seznam

```
(DEFUN FACTORIAL (N)
  (COND ((EQUAL N 1) 1)
        (T (TIMES N
                  (FACTORIAL (DIFFERENCE N 1))))))
```

P r e g l e d p r o g r a m i r n i h l a s t n o s t i . V Lispu imajo programi in podatki enako obliko (so seznami). Prav to omogoča programom, da oblikujejo in modificirajo programe (programsko generiranje)! to je pomembna lastnost za tkim. inteligentne aplikacije. Prav tako je mogoče napisati programirne pripomočke za popravljanje in urejenje s polno interaktivnostjo za programerja (to je seveda mogoče tudi v drugih programirnih jezikih). Nekateri pomembnejši vidiki jezika Lisp pa so tile!

-- Ker je Lisp interaktiven, interpretiven jezik, je relativno počasen. (Seveda so možni tudi prevodi in ne samo interpretacija.)
-- Pomnilniško dodeljevanje je avtomatično
-- Izrazi jezika Lisp so zelo enostavni in regularni. Izrazi so sestavljeni iz atomov in iz atomskih kompozicij
-- Krmiljenje je normalno aplikativno. Potek krmiljenja je voden z uporabo funkcij do argumentov (to ni navadna zaporedna krmilna struktura)
-- Spremenljivke so vidne dinamično. Navadno se nelokalni spremenljivki priredi vrednost lokalno z izračunavajočo funkcijo, če ji ni bila prirejena vrednost s funkcijo, ki kliče izračunavajočo funkcijo
-- Za delovanje v realnem času je potreben zmogljiv 'sistem za zbiranje pomnilniških odpadkov', da se sproščajo neuporabljane pomnilniške lokacije
-- Ker predstavlja Lisp ogromen programski paket, so priporočljivi osebni Lisp računalniki in vse lastnosti jezika so lahko implementirane le na velikih računalnikih
-- Uporaba vsnezdenih oklepajev je lahko nepregledna, zato je preporočljivo poimenovanje izrazov in uporaba teh imen v izrazih

T r e n u t n i i z v e d e n k i . Danes se uporabljata dve glavni Lisp narečji! Maclisp, ki je bil razvit na MIT in Interlisp, ki sta ga razvili podjetji Bolt, Beranek and Newman Inc. in Xerox Palo Alto Research Center (PARC). Obe izvedenki nudita podobno programirno okolje z urejevalnimi in popravljalnimi pripomočki in obe imata vrsto Lisp funkcij in izbirnih možnosti. Poudarek v Interlispu je na najboljšem možnem programirnem okolju na račun hitrosti in pomnilniškega prostora. Pri Maclispu je poudarek na učinkovitosti, ohranjanju naslovnega prostora in na prožnosti pri gradnji pripomočkov in vsnezdenih jezikov. Interlisp ima še dobro podprto izpeljanko s popolno dokumentacijo za vrsto uporabnikov! izvaja se pod operacijskimi sistemi podjetij DEC in Xerox.

Potreba po standardizaciji različnih narečij Maclispa je povzročila nastanek jezikov Common Lisp in Lisp Machine Lisp. Common Lisp se uporablja na večini novih osebnih računalnikih in operacijskih sistemih za UI. Ta jezik je učinkovit, prenosen in stabilen. Obstaja pa še vrsta lokalnih izpeljank na različnih univerzah, kot je npr. Franz Lisp (University of California, Berkeley).

Dobro berilo za programiranje z jezikom Lisp sta napisala Winston in Horn ((4)).

Jezik Prolog

Prolog (PROgramming in LOGic) je logično usmerjen jezik, ki je bil razvit v letu 1973 na univerzi v Marseillu (A. Colmerauer in P. Roussel). Dodatno delo na tem jeziku je bilo opravljeno na univerzi v Edinburghu. Ta jezik je bil prvotno predviden za obdelave naravnih jezikov, vendar se je kasneje uveljavil na vseh področjih UI. Razvoj jezika Prolog v Franciji se je nadaljeval do današnjih dni, tako da je nastal dokumentiran sistem, ki ga je mogoče uporabljati na vrsti računalnikov ((5)).

Prolog je sistem za dokazovanje izrekov. Programi so sestavljeni iz aksiomov, izraženih s predikatno logiko prve stopnje skupaj s ciljem (z izrekom, ki ga je potrebno dokazati). Aksiomi so omejeni na implikacije, na leve in desne strani, zapisane v obliki tkim. Hornovih stavčnih členov (klavzul). Hornova klavzula je sestavljena iz množice stavkov, združenih z logičnimi operacijami 'in'. Oblika značilnega prologovskega aksioma je npr.

A 'in' B 'in' C 'in' X 'implicira' Y 'in' Z

To pomeni, da A in B in C in X skupaj implicira Y in Z, ko se stavek bere deklarativno. Stavek je mogoče brati tudi proceduralno kot! Da bi se dokazalo Y in Z, poskusi dokazati A in B in C in X. Če sledamo s tega drugega vidika, je prologovski program sestavljen iz skupine procedur, kjer je leva procedurna stran vzorec, ki je primeren (najdeni primer, ki zadovoljuje pogoje) za dosego ciljev desne procedurne strani, ko velja

procedura! vzorec --> cilji

Obstaja podobnost teh modularnih pravil s pravili (IF, THEN produkcijami) pri konstruiranju izvedenskih sistemov. Ta modularnost zagotavlja jasno, natančno in hitro programiranje, kar je glavni vzrok razširjenosti jezika Prolog.

Prolog je v bistvu razširitev jezika Lisp, ki je povezana z vraševalnim jezikom relacijske podatkovne baze (npr. v obliki hornovskih klavzul za izražavo osnovnih podatkov), ki uporablja virtualne relacije (implicitne relacije, opredeljene s pravili). Kot Lisp je tudi Prolog interaktiven in uporablja dinamično dodeljevanje pomnilnika.

Prolog je veliko manjši program od Lispa in je bil implementiran na vrsti računalnikov (tudi mikroračunalnikov, npr. za sistem CP/M). Izvajanje Prologa je presenetljivo učinkovito in njegova kompilacijska izvedenka je hitrejša od lispovske. Prolog se je razširil predvsem v Evropi, na Japonskem pa je bil predviden kot jezik pri razvoju pete računalniške generacije. Primeren je za paralelno iskanje in za izražavo prihodnjih paralelnih procesov. Določene lastnosti Prologa se v zadnjem času implementirajo tudi v Lispu (zlasti v ZDA). Prolog naj bi bil primeren za globinsko iskanje v zapletenih problemih, pri katerih bi se lahko pojavila kombinatorna eksplozija obsega iskalnega prostora.

Drugi jeziki za UI

Več jezikov za UI je bilo razvitih kot razširitve, izboljšave in alternative k jeziku Lisp. Ti primeri so tile ((1))!

-- sistemski programirni jeziki (Lisp ravnina)
   -- Sail (Stanford AI Language, 1969)

-- QA4 in Qlisp (SRI 1968, 1972)
-- POP-2 (univerza v Edinburghu, 1967)
-- jeziki za dedukcijo/dokazovanje izrekov
-- Planner in Microplanner (MIT, 1971)
-- Conniver (MIT, 1972)
-- Popler (univerza v Edinburghu, 1972)
-- Prolog (univerza v Marseillu, 1973)
-- Amord (MIT, 1977)

Lisp in POP-2 sta primerna za enostavno oblikovanje novih jezikov v okviru njiju samih ali nad njima. Tako je Qlisp vgnezden v Interlispu in Popler v POP-2. POP-2 se je razširil le v Angliji. Večina teh jezikov ni več vzdrževanih in padajo v pozabo; vendar so prispevali k razvoju sodobnejših jezikov UI, kot so narečja Lispa in Prologa. Prologovski programirni stil je podoben stilu v QA3 in v Plannerju.

Drugi posebni jeziki so bili zgrajeni za predstavitev znanja, za upravljanje baz znanja, za pisanje sistemov s pravili (kot so npr. izvedenski sistemi) in za posebna področja uporabe.

Stroji za UI

Računalniki, ki so se uporabljali pri raziskavah UI v preteklih letih, so bili prvenstveno DECovi sistemi -10 in -20 (stroji s časovnim dodeljevanjem). V poslednjem času so bili ti sistemi zamenjani z bolj gospodarnim VAXom in z novimi osebnimi računalniki za UI. Brown ((6)) ugotavlja, da novejšim strojem manjka programska oprema, in sicer predvsem bogate programske knjižnice DECovih sistemov -10 in -20. Novejši stroji imajo sicer 32-bitne naslovne besede, ki pa komajda zadoščajo za obširen naslovni prostor, ki je potreben programom UI.

Fahlman in Steele ((7)) ugotavljata, da je VAX z operacijskim sistemom Berkeley Unix še vedno najboljši stroj z dodeljevanjem časa za delo v UI. Več lispovskih narečij obstaja za sistem VAX/Unix in VAX se uporablja na številnih univerzah.

Nekateri novi osebni, enouporabniški računalniki za UI so 3600 (Symbolics), Lambda (Lisp Machines), Perq2 (Perq Systems) in Xerox 1100 Series (Xerox). Njihova cena doseza vrednosti $ 800 000. Prodajajo se tudi osebni računalniki manjših zmogljivosti za UI. Na MIT so razvili osebni stroj, ki je posebej mikrokodiran za Lisp. Ta stroj prodajata podjetji Symbolics in Lisp Machines (LM).

Novi osebni stroji imajo zelo zmogljivo okolje za interaktivno delo in raziskovalno programiranje, kjer se sistemsko načrtovanje in program razvijata skupaj ((8)). To je v nasprotju s klasičnim strukturirnim programiranjem, pri katerem se programske specifikacije opredelijo najprej, razvoj programa pa se striktno ravna po teh specifikacijah. Jezik Zetalisp, ki je izvedenka Maclispa, je integrirano programsko okolje za razvoj in izvajanje programov na stroju Symbolics 3600. Zetalisp ima približno 10 000 prevedenih funkcij. Podobne zmogljivosti ima stroj LM Lambda. Interlisp-D se uporablja na stroju Xerox 1100 in ima posebno močno podporo za razvoj izvedenskih in drugih sistemov, temelječih na obdelavi znanja.

Pri naštetih strojih je viden razvoj v smeri raziskovalnega programiranja, uporabniški prijaznosti objektno usmerjenih programirnih jezikov itd. Objekt (kot je npr. letalo ali okno na računalniškem zaslonu) se kodira kot informacijski paket s pripadajočimi opisi procedur za informacijsko manipulacijo. Objekti komunicirajo s pošiljanjem in sprejemanjem sporočil, ta pa aktivirajo procedure. Razred je opis enega ali več podobnih objektov. Objekt je primer iz razreda in ima značilnosti razreda. Programer razvija nov sistem z oblikovanjem razredov, ki opisujejo sistemske objekte. Programer implementira sistem z opisom sporočil, ki bodo odposiljana. Uporaba objektno usmerjenega programiranja znižuje zapletenost velikih sistemov. Zamisel razreda zagotavlja enovit okvir za opredeljevanje sistemskih objektov in podpira modularno, hierarhično (navzdolnjo) programsko strukturo ((9)). "Smalltalk" je objektno usmerjen jezik na Xeroxovih strojih. "Flavors" se uporablja na lispovskih strojih MIT. "Loops", ki je bil razvit na Xeroxovem PARC, je razširitev sistema Smalltalk. "Ross" je objektno usmerjeni programirni jezik podjetja Rank Corp. za simbolično simulacijo aktivnosti.

Napoved

Pričakuje se, da bodo cene računalnikov UI, ki uporabljajo jezik Lisp, hitro padle pod $ 50 000. Ti stroji bodo standardni za uporabo v UI v naslednjih letih. Lispova narečja za osebne računalnike bodo prevladujoča. Programska prenosljivost se bo izboljšala. Prolog in njegove izpeljanke bodo vključene v lispovske sisteme.

Prihodnji UI stroji bodo imeli paralelno arhitekture. To je še posebej pomembno za Prolog, ki uporablja paralelno iskanje. Japonci nameravajo zgraditi osebni računalnik za zaporedni Prolog z 10 000 logičnimi sklepi na sekundo v letu 1985. V letu 1990 bodo imeli Japonci izredno zmogljivi stroj za UI, ki bo uporabljal Prolog z milijon logičnimi sklepi na sekundo (to je približno 10 000-krat več, kot zmore DEC-10). Računalniki s paralelno obdelavo bodo sčasoma zamenjali današnje osebne stroje za UI.

Hitro naraščajoča sposobnost razvoja največjih integriranih vezij bo povečevala računalno moč za UI tudi izven laboratorijskih okolij. Naraščala bo tudi uporaba objektno usmerjenega programiranja pri oblikovanju velikih raziskovalnih programov. Uporaba objektov je primerna tudi pri programiranju dinamičnih simbolnih simulacij, ki so pomembne pri iskanju globjega znanja in povečane zanesljivosti na znanju osnovanih sistemov. Objektno usmerjeno programiranje je obetavno tudi za porazdeljeno obdelavo, ker je mogoče posamezen objekt implementirati na posebnem procesorju v procesorski mreži. Pričakuje se tudi, da bodo raziskovalne metode programskega razvoja sčasoma izpodrinile današnje načine programiranja.

Slovstvo

((1)) The Handbook of Artificial Intelligence, Vol. II, A.Barr, E.A.Feigenbaum, Eds., Los Altos, Cal., William Kaufman, 1982.

((2)) A.Newell, J.C.Shaw, H.A.Simon: Programming the Logic Theory Machine, Proceedings, Western Joint Computer Conference, 1957, pp.230-240.

((3)) N.Graham: Artificial Intelligence, Blue Ridge Summit, Pa., Tab Books, 1979.

((4)) P.H.Winston, B.K.P.Horn: LISP, Reading, Mass., Addison-Wesley, 1981.

((5)) A.Colmerauer, H.Kanoui, M.Van Caneghem: Last Steps Toward an Ultimate PROLOG, Proceedings, International Joint Conference on Artificial Intelligence, Vancouver, B.C., Canada, Aug. 1981, pp.947-948.

((6)) D.R.Brown: Recommendations for an AI Research Facility at NASA/GFSC. SRI Project 2203. SRI International, Menlo Park, Cal. Oct. 1981.

((7)) S.E.Fahlman, G.L.Steele: Tutorial on AI Programming Technology: Language and Machines. Proceedings, National Conference on AI, Spon. Amer. Assoc. for AI, Pittsburgh, Pa., Aug. 1982.

((8)) B.Sheil: Power Tools for Programmers. Datamation, Feb. 1983, pp.131-144.

((9)) D.Robinson: Object-Oriented Software Systems. Byte, Aug. 1981, pp.74-86.

A. P. Železnikar

Iz naših dnevnih časopisov

V zadnjih mesecih opažamo določnejše opredeljevanje družbenih teles in posameznikov v podporo (verbalni) računalniški tehnologiji in računalniškemu izobraževanju. To pozitivno opredeljevanje je značilno tudi za slovensko dnevno časopisje, ki odmerja vzgoji in poročanju s področja računalništva in informatike več prostora. Počasi se vendarle oblikuje javna in politična zavest o širši pomembnosti računalništva v našem tehnološkem napredku in pri vzgoji in izobraževanju. Ta zavest je predvsem posledica določenega stanja - v razvitih državah in pri nas doma. Tako lahko zvemo, da je samo v SR Sloveniji 10 000 do 12 000 "hišnih" računalnikov, ki so bili seveda uvoženi z zasebno iznajdljivostjo. Tudi mladina se na lastno pobudo izobražuje in želi izobraževati z uporabo računalnikov. Šolniki se le počasi budijo iz utečenega sna in pristopajo k bolj organiziranemu prizadevanju za izboljšanje stanja - posodabljanju učnih načrtov in nabavi opreme.

V ljubljanskem Dnevniku lahko 11. maja 1984 na strani 3 prečitamo izjavo predsednika MK ZKS Jožeta Smoleta, ki pravi: "In ne nazadnje, truditi se moramo, da bi mladi ljudje lahko kupovali računalnike in osebne računalnike, kar je sploh izrednega pomena za pripravo novih rodov za prehod v postindustrijsko družbo informatike."

V ljubljanskem Dnevniku najdemo 12. maja 1984 na strani 2 poročilo o zasedanju Tiskovnega sveta pri RK SZDL, v katerem je rečeno tole: "V razpravi je bila v zvezi s to tematiko med drugim močno poudarjena potreba po ustrezni enotni obliki usposabljanja in izobraževanja kadrov za računalništvo. Predsednik republiškega komiteja za informiranje Marjan Šiftar je navzoče obvestil, da je izvršni svet za pospešitev proizvodnje domačih računalnikov, nadalje za prodajo računalnikov v konsignacijah, da je potrebno to področje urediti sistemsko, kajti samo po približnih ocenah je zdaj doma v Sloveniji kakih 10.000 do 12.000 osebnih računalnikov."

A. P. Železnikar

KAKŠNE SO MOŽNOSTI ZA RAČUNALNIŠKO OPREMLJANJE NAŠIH SREDNJIH ŠOL

V šolskem letu 1981/82 se je, sočasno z začetkom uvajanja srednjega usmerjenega izobraževanja v SRS ter poleg že uveljavljenega pouka predmeta Računalništvo v 4-letnih srednjih šolah, začelo tudi izvajanje srednješolskega programa računalništva. Vsebine omenjenega vzgojnoizobraževalnega programa, ki med drugim določajo: učni načrt, predmetnik, pogoje za vpis učencev, nadalje pogoje, ki jih mora izpolnjevati šola za izvajanje izobraževanja ter vrsto in stopnjo izobrazbe učiteljev, so sprejeli uporabniki in izvajalci na skupščini Izobraževalne skupnosti za elektrotehniško in računalniško usmeritev v mesecu februarju 1981. Vseh določil omenjenega programa, zlasti v zvezi z materialnimi pogoji, pa šole kljub dobri volji in angažiranosti le ne bodo mogle same izpolniti. V primerjavi z drugimi (zahodnimi) državami, kjer s pomočjo vladnega programa (Francija) z računalniškimi sistemi načrtno enotno opremljajo srednje šole, ponekod so to storili pred leti in danes že opremljajo osnovne šole (Velika Britanija) in celo otroške vrtce (Japonska), ko morajo ponekod celo zakonsko preprečiti (ZDA), da šolam ne bi darovalo opremo več firm, pri nas ni sistemsko rešeno niti vprašanje financiranja šolskih mikroračunalnikov. V prejšnjem srednjeročnem obdobju je sicer Izobraževalna skupnost Slovenije v Samoupravnem sporazumu o osnovah plana vzgoje in usmerjenega izobraževanja v SRS za obdobje 1976-1980 opredelila sredstva za razvoj računalniških sistemov visokih in srednjih šol, vendar so v ta namen zbrana sredstva zadoščala le za sofinanciranje 1. faze računalniške mreže univerze. V planskem obdobju 1981 - 1985 pa je postal sistem financiranja drugačen in je vreča, iz katere naj bi zajemali tudi za računalniško opremljanje naših srednjih šol, ostala prazna. Tako bo tudi prvi korak, ki ga je napravila Iskra Delta, ko je s 25%-nim popustom v odprtem pismu ponudila vsem šolam računalniška sistema DELTA 400 M in DELTA 400 B oz. večnamensko terminalsko postajo, verjetno ostal brez želenega odmeva, saj šole same za to nimajo sredstev.

In kakšno rešitev skušajo šole, zlasti tiste v računalniški usmeritvi, najti za odpravo omenjenega problema? Praktično edino možnost vidijo v neposrednem sodelovanju z delovnimi organizacijami - uporabniki kadrov, ki se v šolah izobražujejo. Tako srednje šole računalniške usmeritve, te so v naši republiki štiri: v Kranju, Ljubljani, Mariboru in Titovem Velenju, iščejo ustrezne botre, ki bi jim bili pripravljeni pomagati pri zagotavljanju materialne opreme. Šola v Titovem Velenju, ki deluje v okviru Centra srednjih šol, je tako s Tovarno gospodinjske opreme Gorenje sklenila Samoupravni sporazum o poslovno-tehničnem sodelovanju za zagotavljanje pogojev za izvajanje izobraževanja v računalniški usmeritvi. Namen sodelovanja je v skupnem zagotavljanju materialnih in kadrovskih pogojev, potrebnih pri izvajanju vzgojnoizobraževalnega programa računalništva. Šola je združila devizna sredstva (ki jih je pridobila v prejšnjih letih z izobraževanjem nigerijskih in libijskih učencev), Gorenje pa je dobavilo in instaliralo računalniško opremo. To po končani 1. fazi tvori: 10 mikroračunalniških sistemov z upogljivimi diski (5 1/4 ") in video monitorji. Naslednja faza bo omogočila povečanje funkcionalnosti in zmogljivosti obstoječe opreme (poleg DOS še CP/M operacijski sistem, instaliranje RAM floppy-ev, diskov z 1,2 M zlogi, RS 232 in IEEE 488 vmesnikov in 80-kolonskih kartic), priključek osmih šolskih video terminalov na računalniški sistem Gorenja ter opremo učilnice za procesne sisteme, kjer bodo instalirani prosto programirljivi krmilni sistemi. Čeprav bo omenjena oprema prioritetno namenjena pouku strokovnih predmetov računalniške usmeritve, pa jo bo možno uspešno vključiti tudi kot važen vzvod pri razvoju izobraževanja v elektrotehniški, kovinarski, naravoslovno-matematični, rudarski in družboslovni usmeritvi, torej vseh programih, ki potekajo znotraj Centra srednjih šol, pravtako pa jo že uporabljajo pri svojem delu mladi raziskovalci in člani računalniškega kluba.

Anton Gams

ŠESTI MEĐUNARODNI SIMPOZIJ PROJEKTIRANJE I PRAĆENJE PROIZVODNJE RAČUNAROM

PPPR

ZAGREB — JUGOSLAVIJA

10—11. LISTOPADA 1984.

U OKVIRU IZLOŽBE »INTERBIRO« NA ZAGREBAČKOM VELESAJMU

Šesti međunarodni simpozij »Projektiranje i praćenje proizvodnje računalom« (PPPR 1984.) nastavlja tradiciju prijašnjih skupova koji se održavaju u Zagrebu od 1979. godine.

Održavanje simpozija PPPR 1984. ponovno omogućuje kroz razgovore, diskusije i susrete razmjenu znanja i iskustava svim inženjerima, liječnicima, istraživačima i znanstvenicima koji se bave teorijom i primjenom postupaka projektiranja i praćenja proizvodnje računalom ili kuriste računalo u istraživanjima i obrazovanju. Teme simpozija nabrojene su u ovoj obavijesti, što ne znači da organizacioni i programski odbor neće uzeti u obzir i radove s drugim temama koje bi mogle biti interesantne učesnicima simpozija.

Prošlo je oko 15 godina od kada je prvi puta upotrebljen izraz CAD (Computer Aided Design) da bi se označio postupak projektiranja primjenom računala. Danas je CAD svakodnevna praksa koja se dinamično razvija, koja zahtjeva velika ulaganja u adekvatni hardver i u čiju primjenu je uključen veliki broj vrhunskih stručnjaka. Također je jasno da je CAD postao jezgra integralnog procesa praćenja proizvodnje računalom — CAM (Computer Aided Manufacturing), što se dosadašnjim može već uočiti u mnogim industrijskim granama. Na simpoziju će se razmatrati i problematika povezivanja postupaka projektiranja pomoću računala i procesa praćenja proizvodnje računalom — CAD/CAM.

TEME SIMPOZIJA

A. STRUKTURA CAD/CAM SUSTAVA

A1. JEZICI

Za pisanje CAD/CAM programa od velike su važnosti jezici. S jedne strane to je upravljački jezik pomoću kojeg projektant upravlja programom i programski jezik u kojem je korisnički program napisan. Značaj jezika je velik: jezik omogućuje prijenos drugim programskim pomagalima uključujući definicije upravljačkog jezika, grafiku i informacijske strukture. Stvaranje upravljačkog jezika posebno je značajno za interaktivne programe s grafičkim prikazom.

A2. OPERACIONI SISTEMI I ORGANIZACIJA RAČUNALA

Operacioni sistem mora se prilagoditi računalu na kojem se CAD/CAM program izvodi. Razvoj CAD CAMa moguć je samo uz dobar operacioni sistem, a naročito zbog trivijalnih ili netrivijalnih interakcija kod mnogih primjena CAD/CAMa. Kvalitetu operacionog sistema određuju fleksibilnost ulazno-izlaznih jedinica, sistemi datoteka, kompaktnost, pouzdanost, brzina odziva, mogućnost višestrukog batch pristupa, editori kao i drugi programska pomagala.

Kod računala organiziranih za batch obradu postignut je već relativno visok stupanj razvoja. Terminalski ulaz bio je važan faktor koji je batch obradu učinio efikasnijom. Puno veća raznolikost organizacije računala postoji kod sistema s interaktivnim radom: 1. Multiprogramski sistem s vremenskim dodjeljivanjem resursa koji se poslužuje s više terminala, 2. Jednokorisnički sistem (s interaktivnom grafikom), 3. Distribuirani računarski sistem (s višestrukim procesorima), 4. Inteligentni grafički terminali, 5. Mreža računala.

A3. RAČUNARSKA GRAFIKA I NAČIN PREDSTAVLJANJA OBJEKATA

Izlaz preko plotera ili uređaja za pohranu na mikrofilmove ponekad se naziva pasivnom grafikom ili neinteraktivnom grafikom. Primjena interaktivne grafike mogla bi se podijeliti u dvije kategorije: 1. vizuelno skaniranje podataka i 2. unos projektnih podataka. Što se tiče grafičkog softvera potrebno je istaknuti dva svojstva: strukturu grafičkih podataka i brzina grafičkog odziva.

Prikaz objekata se može podijeliti u slijedeće skupine: funkcionalni, topološki i geometrijski (blok geometrija, slobodne plohe, 2,5 dimenzionalni prikaz i dr.).

A4. POSTUPCI ANALIZE I SINTEZE

Karakteristično je da danas za svaku primjenu u CADu postoje posebni postupci analize i sinteze kao što su: 1. simulacije, 2. metode konačnih i graničnih elemenata, 3. optimizacija (postoje tri glavna pristupa: pretraživanje eliminacijom, linearno programiranje i nelinearno programiranje), 4. neposredna sinteza (posebno u teoriji upravljanja).

Prednost računala u mnogim CAD primjenama danas je očita: u području konačnih elemenata i analizi naprezanja, rješavanju jednadžbi polja metodom konačnih diferencija, numeričkom rješavanju kompleksnih diferencijalnih jednadžbi, modeliranju višekomponentnih procesa destilacije, simulaciji procesa kod projektiranja visokih peći i izmjenjivača topline, naprezanjima u cijevovodima, analizi seizmičkih ispitivanja, bioanalizi, dijagnostici u medicini i drugdje.

A5. INFORMACIONE STRUKTURE

Upotreba informacionih struktura u CAD/CAMu je višestruka. Glavna primjena pohrane informacija i njihovog pretraživanja je u konceptijskoj fazi projektiranja, stvaranju kataloga i lista dijelova, stvaranju datoteka informacija za CAD/CAM programe koji međusobno komuniciraju i upotreba postupaka asocijativnog pretraživanja i sortiranja za vrijeme analize i ispisa.

Informacione strukture u CAD/CAM sistemima treba promatrati u dva nivoa: 1. logički nivo, 2. smještajni nivo. Za obradu velikih struktura postoji više metoda koje se većinom baziraju na dodjeljivanju stranica što obuhvaća listovne strukture, pridružene listovne strukture i postupke virtualnih memorija. S povećanjem mogućnosti sistema datoteka, njihova uporaba kao sistema pohrane za CAD/CAM sistemе je u porastu.

B. PODRUČJE PRIMJENE CAD/CAMa

B1. ARHITEKTURA I GRAĐEVINARSTVO

B2. BRODOGRADNJA I STROJARSTVO

B3. ELEKTROENERGETIKA

B4. ELEKTROSTROJARSTVO I OPREMA

B5. ELEKTRONIKA

B6. MEDICINA

B7. METEOROLOGIJA, OCEANOLOGIJA, SEIZMOLOGIJA

CAD/CAM se odnosi na korištenje digitalnih računala kod projektiranja i proizvodnje složenih objekata kao što su automobilske karoserije, avioni, brodovi, mostovi, tlačne posude i dr., ili elemenata kao što su radilice, električni motori, transformatori, elektronički krugovi, umjetno srce i dr. Cjelokupno projektiranje može se izvesti pomoću računala, bez obzira radi li se o cijelama ili brodovima, modeliranju dijelova ljudskog tijela ili ponašanju mora i oceana. Budući da su matematički i numerički postupci u raznim primjenama jako slični, projektanti će moći koristiti isti hardver i sličan softver. CAD/CAM sistemi povećava produktivnost i efikasnost u više, međusobno povezanih faza tehnike i medicine, te proizvodnje kao što su projektiranje, analiza, priprema tehničke dokumentacije, konstrukcija alata i programiranje numeričkih upravljanim strojevima.

ROKOVI PRIJAVE SUDJELOVANJA S RADOM

Ispunjenu prijavu sa sažetkom do 500 riječi, koji će što jasnije sadržaj i svrhu rada, poslati do 15. travnja 1984. godine.

Odabrane radove, napisane prema uputama koje će autori dobiti uz obavijest o uvjetnom prihvatu rada na osnovu sažetka, treba poslati do 1. lipnja 1984. godine.

Jedan autor može prijaviti najviše dva rada, bilo kao autor ili koautor.

Obavezni, sastavni dio prijave rada je anketa o radu tiskana na poleđini prijave.

Original rada pogodan za fotokopiranje i dvije kopije, pisan strojem s jednostrukim proredom, formata A4, ukupnog opsega do 6 stranica treba pisati prema uputi koju će autori dobiti uz obavijest o prihvatu rada. Radovi većeg opsega neće se prihvatati. Reklamentima će prilikom izlaganja biti na raspolaganju diaprojektor i grafoskop.

ROK PRIJAVE PRISUSTVOVANJA BEZ RADA

Prijave prisustvovanja simpoziju i uplatu kotizacije treba izvršiti do 1. rujna 1984., godine.

KOTIZACIJA

Kotizacija iznosi 5.000,00 din i uključuje Zbornik radova, kao i učešće slušaoca na svim sjednicama simpozija. Kotizacija se uplaćuje na žiro-račun Elektrotehničkog fakulteta u Zagrebu broj 30101-603-405 uz naznaku svrhe uplate: Kotizacija za PPPR, ime i prezime.

Autori trebaju platiti kotizaciju po primitku obavijesti o konačnom prihvatu rada, a najkasnije do 1. rujna 1984. godine. Plaćanje kotizacije je uvjet za tiskanje rada u Zborniku simpozija.

MJESTO ODRŽAVANJA

Simpoziji će se održati u kongresnim dvoranama Zagrebačkog velesajma, Avenija Borisa Kidriča 2.

IZLOŽBE

Simpozij će se održati u okviru specijalizirane priredbe Zagrebačkog velesajma INTERBIRO, 8—11. listopada 1984. godine za koju će sudionici imati besplatan permanentni ulaz. Na Interbirou svjetski i domaći proizvođači izlažu svoja dostignuća na području računarske tehnike i CAD CAM opreme.

Za sudionike simpozija predviđena je posebna demonstracija rada CAD/CAM sistema i druge kompjuterske opreme.

HOTELSKI SMJEŠTAJ

Za sudionike simpozija organizator je dogovorio smještaj u hotelima ESPLANADE-Intercontinental (de Luxe), PALACE (A), BEOGRAD (B), INTERNACIONAL (B), DUBROVNIK (B).

Sudionici zainteresirani za hotelski smještaj trebaju poslati ispunjenu prijavnicu do 15. travnja 1984. nakon čega će dobiti cjenik smještaja i moći izvršiti rezervaciju.

JEZICI

Zvanični jezici simpozija su jezici jugoslavenskih naroda i engleski jezik.

ORGANIZACIONI I PROGRAMSKI ODBOR

Predsjednik

Dr Zijad Haznadar, redovni profesor Teoretske elektrotehnike Elektrotehničkog fakulteta Sveučilišta u Zagrebu

Potpredsjednik

Dr Vesna Jurčec, savjetnik Republičkog hidrometeorološkog zavoda SR Hrvatske

Tajnik

Mr Nikola Žanić, znanstveni asistent Elektrotehničkog fakulteta Sveučilišta u Zagrebu

Članovi

Dr Ibrahim Aganović, redovni profesor Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu

Dipl. ing. Mirna Bratulić-Rejkuba, »Intertrade« — predstavništvo Zagreb

Dr Janko Hančević, redovni profesor Medicinskog fakulteta Sveučilišta u Zagrebu

Dr Stjepan Jecić, redovni profesor Fakulteta strojarstva i brodogradnje Sveučilišta u Zagrebu

Rudjer Jeny, publicista iz Zagreba

Dr Ivan Mandić, rukovodilac odjela Elektrotehničkog instituta »Rade Končar« iz Zagreba

Dr Bogdan Zelenko, redovni profesor Fakulteta građevinskih znanosti Sveučilišta u Zagrebu

Dipl. ing. Zoran Žic, asistent na katedri za Teoretsku elektrotehniku Elektrotehničkog fakulteta Sveučilišta u Zagrebu

POKROVITELJI SIMPOZIJA

Jugoslavenska akademija znanosti i umjetnosti u Zagrebu

Samoupravne interesne zajednice za znanstveni rad SIZ I—VI SR Hrvatske

ORGANIZATORI SIMPOZIJA

Elektrotehnički fakultet Sveučilišta u Zagrebu

OOUR Elektrotehnički institut »Rade Končar« Zagreb

Društvo za mehaniku SR Hrvatske

Građevinski institut Zagreb

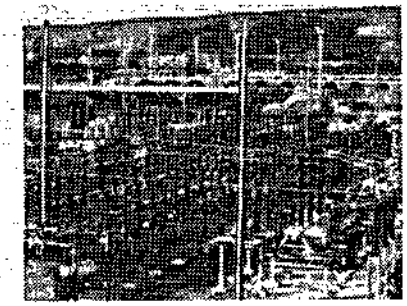
BI »Uljanik« Pula

ADRESA TAJNIŠTVA SIMPOZIJA

ELEKTROTEHNIČKI FAKULTET ZAGREB
Zavod za OEEM (za PPPR)
Unska 3, 41000 ZAGREB, YUGOSLAVIA
telefon: (041) 515-411/253
telex: 21234 ETF ZG YU

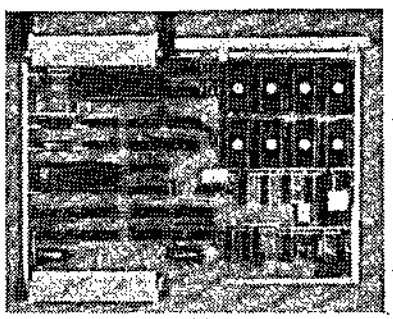
SISTEMI ZA ENERGETIKO

Ljubljana, Tržaška c. 2



DALJINSKO IN LOKALNO PROCESNO VODENJE Z RAČUNALNIKI MINIRAČUNALNIKI IN MIKRORAČUNALNIKI V NAŠIH DOMAČIH SISTEMIH DIPS-11 IN DIPS-85

RAZISKAVE, RAZVOJ, PROIZVODNJA, INSTALACIJA,VZDRŽEVANJE SPECIALISTIČNO ŠOLANJE KUPČEVIH STROKOVNJAKOV ELEKTROENERGETIKA, PLINOVODI, NAFTOVODI, VODOVODI, INDUSTRIJA



SODOBNA TEHNOLOGIJA - NAŠ TEMELJ PRI RAZVOJNEM DELU
RAČUNALNIKI - NAŠI SOPOTNIKI NA POTI NAPREDKA
OBIŠČITE NAS IN SE PREPRIČAJTE

Že veliko let se ukvarjamo z raziskavami, razvojem in proizvodnjo sistemov za daljinsko in lokalno procesno vodenje. Temeljno vodilo našega delovanja na tem področju je slediti napredku v svetu in ga presajati na naša domača tla. Vedno smo zavračali nosilno licenčno povezovanje s tujimi firmami povsod tam, kjer smo jasno videli, da vodi v dolgoročno odvisnost in tehnično nazadovanje. Verjeli pa smo v moč lastnega marljivega dela in v ustvarjalnost naših delavcev ter z vstrajnim delom dosegli uspehe, katere nam lahko zavidajo neprimerno večji in bogatejši tekmeci.

Prav zaradi lastne poti in lastnega znanja smo s svojim razvojnim delom ves čas uspeli slediti najnovejšim tehnološkim dosežkom v svetu. V praktično življenje (računalniški nadzor v elektroenergetiki) smo vpeljali najsodobnejše mikroračunalnike.

Tako smo od prvih računalniških korakov pred več kot petnajstimi leti dospeli do sedanjih kompleksnih sistemov za procesno vodenje.