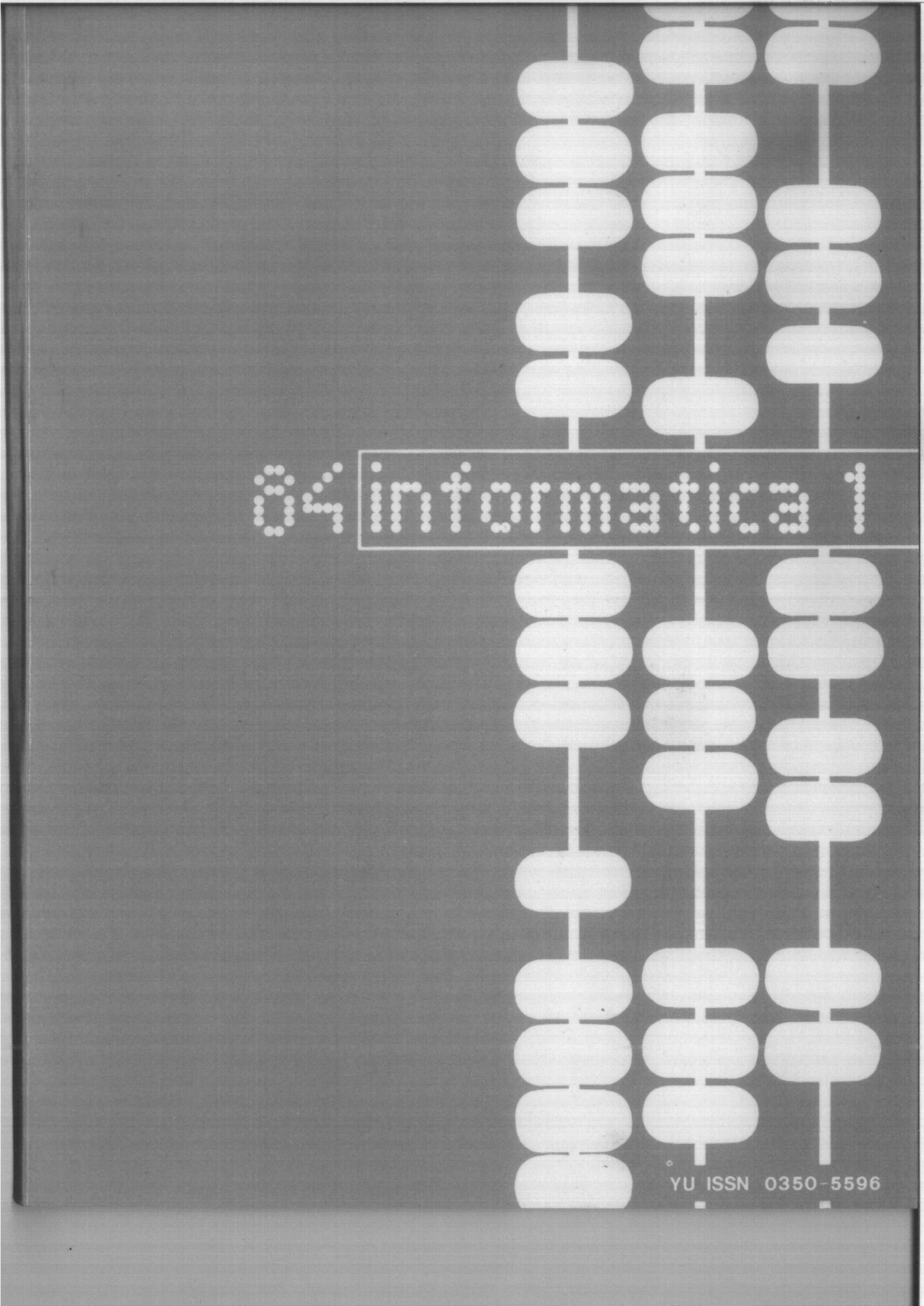
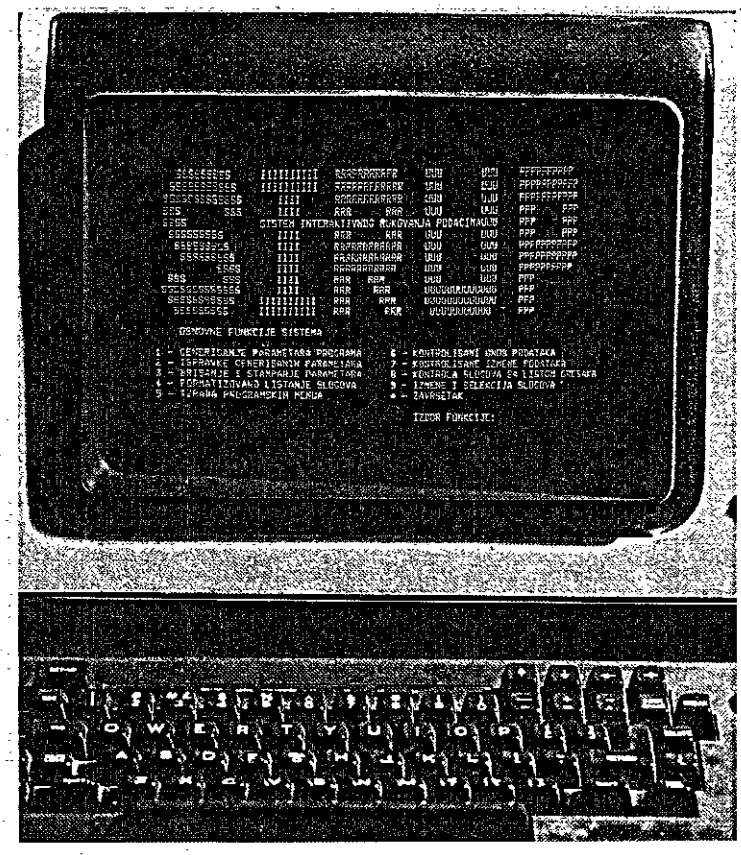


YU ISSN 0350-5596

INTERAKTIVNI GENERATOR PROGRAMOV



- SIRUP omogoča programiranje brez poznavanja programskih jezikov.
- SIRUP je niz interaktivnih programov za splošno uporabo. Njegov osnovni cilj je, da pospeši in olajša pripravo in vzdrževanje uporabniških podatkov.
- SIRUP zmanjšuje čas, potreben za izdelavo aplikacij, angažiranja računalnikov in programerjev, za 30–90 %.
- SIRUP deluje interaktivno s pomočjo parametrov, katere uporabnik sam določa in izbira. SIRUP omogoča hitre spremembe in dopolnitve programov.

Iskra Delta

Iskra Delta
proizvodnja računalniških sistemov in inženiring, p.o.
61000 Ljubljana, Parmova 41
telefon: (061) 312-988
telex: 31366 YU 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. 1

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

V S E B I N A

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 1

C O N T E N T S

RAZVOJ INFORMACIJSKE INDUSTRIJE JUGOSLAVIJE DO LETA 2000

R. FALESKINI

DO ISKRA DELTA, LJUBLJANA

UDK: 681.3(497.1)

Članek opisuje nekatere aspekte prihodnjega razvoja jugoslovanske informacijske industrije. Vsaka razvojna strategija mora upoštevati svetovni računalniški trg in njegovo rast. V procesih planiranja je potrebno študirati možnost uporabe novih lastnosti tehnologije in produktov in možnost premikov v področje z visoko rastjo. Opisan je popust model razvojne strategije, ki naj pomaga tehničnim strokovnjakom pri opredeljevanju tehnične in tržne konkurenčne pozicije produktov informacijske industrije.

Development of Yugoslav Information Industry to the Year 2000

This article deals with some aspects of future development of Yugoslav information industry. Any strategy of development must consider the world computer market and it's growth. In the planning processes the possibility to exploit new technology and product opportunities and to move into new high growth areas has to be studied. A simple model of development strategy is described to help technical experts in defining the technological and market competitive positioning of the products of information industry.

1. UVOD

Uspešnost razvoja informacijske industrije v SFRJ je pogojena z realnostjo ocene stanja in sposobnosti naše družbe, kakor tudi s pravilnim opredeljevanjem politike, strategije in posameznih razvojnih usmeritev. Delavci Iskra Delte so si postavili določene razvojne cilje, ki jih s tem člankom želimo predstaviti strokovni javnosti. Ne glede na zanimivost v članku niso podane opredelitve do čisto tehnične problematike, kot je npr. peta generacija računalnikov, zveza med telekomunikacijami in računalniki, temveč je dan poudarek nekaterim za inženirje marginalnim temam, ki pa v realni praksi lahko postanejo centralne, zlasti ko analiziramo, zakaj določen projekt ni uspel.

2. STRATEŠKI CILJI INFORMACIJSKE INDUSTRIJE

V jugoslovanskih razmerah se informacijska industrija lahko razvije samo kot odprt sistem, ki je tako na vhodni kot na izhodni strani povezan s svetovnim trgom.

Informacijska industrija mora biti sposobna proizvajati računalniške sisteme in izvajati ustrezen inženiring. Računalniški sistem opredeljujem kot materialno opremo, sistemsko programsko opremo in aplikativno programsko opremo, ki skupaj predstavljajo računalniški sistem; ta sistem pa je podsistem nekega širšega tehničnega ali tehnološkega sistema. Inženiring opredeljujem kot spreminjanje okolja, v katerega se vgrajuje računalniški sistem, zaradi medsebojne prilagojenosti (ti dve opredelitvi imata zgolj delovni značaj).

Tržno in tehnološko nujni pa so seveda tudi nižji nivoji kompleksnosti - nivo opreme, nivo posameznih materialnih in programskih modulov ter nivo storitev. Tržni in tehnološki razlogi pa nas silijo tudi v to, da nismo prisotni na vseh segmentih proizvodnje in ponudbe računalniških sistemov, ter da smo prisotni na nižjih nivojih samo tam, kjer imamo kakšne komparativne prednosti. Ob taki politiki si med drugimi lahko postavimo naslednje strateške cilje:

1. Na osnovi koncentracije kadrov, znanja, družbenega kapitala in ustrezne organiziranosti razviti novo industrijsko panogo v SFRJ - informacijsko industrijo
2. Zadovoljevati leta 1990 in kasneje večji del potreb jugoslovanskega trga po računalniških sistemih in inženiringu v industriji, velikih tehnično-tehnoloških sistemih in družbenih informacijskih sistemih
3. Ustvarjati leta 1990 in kasneje večji del dohodka na tujih trgih, ob ne samo kompletnem pokrivanju potrebnega uvoza opreme, repromaterialov in znanja z izvozom, ampak tudi ob ustvarjanju znatnih deviznih presežkov.

3. BLAGO IN STORITVE INFORMACIJSKE INDUSTRIJE

3.1. Opredelitev trga in njegove dinamike

Pri izboru tehnološke strategije je nedvomno primarno poznavanje trgov, na katere želimo priti s svojimi produkti. Informacijska industrija ponuja na svetovnem trgu računalniške sisteme, kot smo jih opredelili v naslednji točki, kakor tudi opremo, sklope in module, ki predstavljajo produkte nižjih stopenj v vertikalni tehnološki verigi.

Za produkte uporabljamo naslednjo segmentacijo trga informacijske industrije:

TABELA 1.

A. Računalniki:
- splošno namenski računalniki
- super računalniki za posebne namene
- "plug compatible" računalniki
- mali poslovni sistemi
- namizni računalniki
- komponentni sistem

B. Terminali:
- teleprinterji in neinteligentni zaslonski terminali
- inteligentni terminali
- multifunkcijske delovne postaje
- samostojni in povezani procesorji tekstov
- terminali za posebne namene (grafični, npr.)

C. Periferne naprave:
- pomnilniki vseh vrst
- tiskalniki in druge naprave za izdelavo dokumentov
- naprave za včitavanje dokumentov
- naprave za komuniciranje z glasom

D. Programska oprema:
- sistemska programska oprema
- aplikacijska programska oprema

E. Storitve:
- obdelave informacij v informacijskih centrih
- vzdrževanje strojne in programske opreme
- consulting in projektiranje in inženiring
- izobraževanje kupcev

Zaradi orientacije podajamo tabelarični pregled vrednosti trgov, vzet iz gradiva World Markets for Information Processing Products to 1992 (1)

Iz tabele 2. je razvidno, da gre na vseh segmentih svetovnega trga za zelo visoko letno rast, kar lahko prenesemo tudi v domače razmere.

Ocenjujemo, da je vrednost trga SFRJ za vseh pet naštetih segmentov za leto 1983 15.000.000.000 din ali izraženo zaradi primerjave s tabelo v USA $ blizu 200 milijonov USA $ (tečaj z dne 1.5.1983).

Jugoslovanska računalniška industrija je prisotna na vseh segmentih (A, B, C,D,E), vendar v vsakem segmentu pokriva samo nekaj podsegmentov, podanih v alineah. Ni potrebno posebej razlagati, da v SFRJ nimamo proizvodnje npr. super računalnikov ali diskovnih enot.

Skupna značilnost proizvodnih programov je, da imamo v celotnem spektru velike praznine, istočasno pa občutimo na 16 bitnih miniračunalnikih relativno veliko prekrivanje ob istočasni veliki povezanosti posameznega programa na tuje TNC.

Del programa v SFRJ še nima blagovnega značaja (programska oprema, storitve), kar predstavlja oviro za razvoj in za prodor na svetovni trg.

Večina jugoslovanskih inženirjev se sploh ne zaveda, kaj pomeni blagovni značaj programske opreme in storitev z aspektov prava industrijske lastnine, upoštevanja mednarodnih in tujih nacionalnih standardov, financiranja, dokumentiranja, marketinga itd. To velja tako za inženirje kreativce, kot za vodilne delavce v naših proizvodnih, razvojnih,izobraževalnih in raziskovalnih DO. Odprava te pomanjkljivosti in ustrezna motivacija za kreativno delo sta med drugim pogoja za širši prodor na svetovni trg blaga in storitev informacijske industrije.

3.2. Prekrivanje ali dopolnjevanje v SFRJ

3.2.1. Proizvodnja računalniških sistemov, terminalov, perifernih naprav in software

TABELA 2. VREDNOST SVETOVNIH TRGOV INFORMACIJSKIH PRODUKTOV (BREZ STORITEV) V MILIJARDAH USA $ (konstantnih za 1. 1982)

	Računal-niki	Termi-nali	Perifer. naprave	Program. oprema	SKUPAJ
		1 9 8 2			
ZDA	17,6	7,6	15,3	4,1	44,6
Zah.Evropa	10,9	2,8	9,9	1,6	25,2
Japonska	4,5	2,0	3,2	0,7	10,4
Ostali svet brez SEV	2,1	0,6	1,9	0,3	4,9
SKUPAJ	35,1	13,0	30,3	6,7	85,1
		1 9 8 7			
ZDA	28-31	9-11	22-25	11-13	68-78
Zah. Evropa	17-19	3-5	14-17	6-8	42-48
Japonska	7-9	3-5	5-7	1,5-2,5	18-22
Ostali svet brez SEV	3,5-4,5	1-1,4	3-4	1-1,5	9-11
SKUPAJ	56-62	17-22	45-52	20-25	140-155
		1 9 9 2			
ZDA	37-43	13-16	31-36	24-28	110-120
Zah.Evropa	22-25	4-7	20-24	15-18	64-72
Japonska	10-13	4-6	10-12	4-6	29-35
Ostali svet brez SEV	4-6	1,4-2	4-5	2-3	12-15
SKUPAJ	75-85	23-30	68-75	46-54	220-240

V tabeli ni zaobsežen trg storitev, ker se le-ta šele razvija.

(1) A.D. Little Service: IMPACT: January 1983

Zaradi standardizacije in doseganja velikih serij je nujno, da se jugoslovanski proizvajalci specializirajo za programe, ki so med seboj komplementarni. Vsako prekrivanje je potrebno analizirati z dveh aspektov: prvič, kateri produkt je ekonomsko bolj opravičen in drugič, kakšno zvezo ima določen produkt s tujimi transnacionalnimi družbami (TNC) ter kako to vpliva na dolgoročni jugoslovanski razvoj z vidika standardizacije, dolgoročnih ekonomskih efektov ter družbenih ekonomskih efektov, ki se kažejo v tistih delovnih procesih, kjer se informacijska tehnologija pojavlja kot infrastruktura.

Sedanje stanje je za zunanjega opazovalca bolj karakteristično po tem, da na posameznih podsegmentih trga sploh ni ponudbe in po prekrivanju programov (številni 16 bitni mini računalniki, terminali) ter po majhnih serijah, ki so tehnološko gledano neopravičene za osvajanje delovnih procesov v globino, kot po dpolnjujočih se programih. Razen na samih končnih produktih se kaže neurejenost tudi:

- v sistemu odnosov do TNC oziroma v povsem neodvisnem in zato neenakopravnem odnosu posamezne naše DO,
- v sistemu financiranja proizvodnje,
- v predimenzioniranem razvojno raziskovalnem sektorju tega področja glede na znane zakonitosti: 1:10:100 (laboratorijski prototip, industrijski prototip, serijska proizvodnja),
- v neenakopravnem položaju v primeru s tujimi TNC, ki nudijo produkte v zakup, ki so postavile norme obnašanja in norme kvalitete in ki povsem dominirajo po volumnu potreb, ki jih zadovoljujejo v SFRJ.

Večina odnosov jugoslovanskih proizvajalcev TNC je neustaljena. V primeru izvozno sposobnih projektov, ki bodo tehnološko optimalne serije plasirali predvsem na svetovni trgi, je možno, da se v Jugoslaviji razvije dvoje ali več proizvajalcev na nivoju računalniških sistemov, kar pa ne govori proti unifikaciji na nižjih nivojih v vertikalni tehnološki verigi.

3.2.2. Storitve

Ponudba storitev jugoslovanskih DO je namenjena izključno domačemu trgu. Danes praktično še ni upoštevanja vrednega plasmana tujih storitev pri nas. Ta tuj plasman je omejen na koriščenje različnih svetovnih bank podatkov in na podporo organizacijam, ki servisirajo računalnike.

Ponudba domačih storitev je prisotna na vseh naštetih štirih področjih v točki E (tabela 1), na segmentu vzdrževanje in prekrivanje. Tu gre za monopol proizvajalca oziroma zastopnika tuje TNC. Na vseh ostalih segmentih konkurenca eksistira in jo imamo za pozitivno. Eksistenca konkurence lahko tudi pripomore k temu, da se bodo razvili standardi, tehnični pogoji, pravna regulativa in normalno gospodarjenje s temi storitvami, seveda pa je to samo potreben pogoj, ne pa še zadosten. V tem segmentu je zavest o državnem intervencionizmu kljub nekaj zakonom še zelo nizka.

3.3. Primerjava s svetom

3.3.1. Ekonomska moč

Prihodek 1 milijardo (novih) dinarjev sta v letu 1982 presegla samo dva proizvajalca v SFRJ, Iskra Delta in El. Računari. Če upoštevamo povprečen tečaj US dolarja v l. 1982, preračunamo to v dolarje, bi pri povprečnem tečaju za l. 1982 - 1 USA $ = 50 din, znesla 1 milijarda din 20 milijonov US$. Za primerjavo,kaj ta prihodek predstavlja v svetovnih merilih, podajamo v tabeli 3. prihodke dvajsetih največjih TNC.

Vseh firm, ki so naštete v tabeli 3, ne moremo šteti niti za tehnološko niti za tržno vodilne. Samo za ameriške in japonske proizvajalce lahko rečemo, da so tehnološko vodilni in obenem povsem dominantni na nacionalnem trgu. Za zahodnoevropske proizvajalce to ne velja.

V deželah SEV je računalništvo in informatika na nivoju SEV koordinirana skupna panoga proizvodnje, ki pokriva dominanten del (95 %) potreb po količinah, medtem ko so zaradi tehnološkega zaostanka kvalitativne možnosti pokrivanja potreb manjše in se čuti zaostanek v kvaliteti strojev in sistemov, ki jih proizvajajo.

Industrija SEV zaposluje v produkciji približno 200.000 delavcev in njen prihodek oz. vrednost produkcije po svetovnih cenah, ocenjujemo na cca 5 milijard USA $.

3.3.2. Državni intervencionizem

Vse dežele z razvito računalniško produkcijo razen ZDA uporabljajo vse instrumente državnega intervencionizma za zaščito in razvoj nacionalnih industrij. Ti instrumenti segajo od planiranja preko budžetskega financiranja do zakonodaje in organiziranja.

TABELA 3. DVAJSET NAJVEČJIH PROIZVAJALCEV (BREZ PROIZVAJALCEV DEŽEL SEV)

1980 Rang	1981 Rang	1982 Rang	PODJETJE	Prihodki za koledarsko leto v milijardah USA $		
				1980	1981	1982
1	1	1	IBM	21,4	23,8	28,5
4	2	2	Digital Equipment	2,7	3,4	4,1
6	4	3	Burroughs	2,5	3,0	3,8
3	3	4	Control Data	2,8	3,1	3,3
2	5	5	NCR	2,8	2,9	3,2
5	6	6	Sperry Computers	2,6	2,8	2,8
7	7	7,8	Fujitsu	1,7	2,0	2,2
9	8		Hewlett-Packard	1,6	1,9	2,2
8	9	9	Honeywell	1,6	1,8	1,9
12	10,11	10	Nippon Electric	1,2	1,4	1,5
13	13	11	Hitachi	1,1	1,3	1,4
14	14		Olivetti	1,2	1,3	1,3
15	17	12,13,14	Siemens	0,8	0,9	1,3
18	15		Wang	0,7	1,0	1,3
10	10,11	15	ICL	1,4	1,4	1,2
11	12	16,17	CII-HB	1,2	1,3	1,1
19	19		Storage Technology	0,6	0,9	1,1
16	16	18,19	Nixdorf	0,8	0,9	0,9
17	18		Xerox	0,8	0,9	0,9
20	20	20	Data General	0,6	0,7	0,8

Posebej moramo opozoriti, da delujejo na področju računalništva in informatike tudi nadnacionalni mehanizmi (COCOM), ki imajo za cilj zagotoviti dominacijo ZDA in razvitega zahoda na tem področju.

V primerjavi s svetom je jugoslovanski intervencionizem na področju računalništva bistveno pomanjkljiv v sferi zakonskega urejanja odnosov domačih firm s tujimi TNC, v sferi financiranja dejavnosti, v stopnji kompleksnosti zagotavljanja pogojev za razvoj, pa tudi v sferi urejanja okolja, kamor računalniška tehnologija prihaja. Razdrobljenost velikih sistemov po republikah, regijah itd., ki je nastala v obdobju, ko še ni bilo domače industrije, se nadaljuje, ker se velik del nastajajočih jugoslovanskih proizvodenj naslanja na tuje TNC v pogledu produktov, tehnologije, repromaterialov, financ itd.

4. PROGRAMI - RAZVOJ DO LETA 2000

4.1. Dominantno področje

Glede na to, da v SFRJ ne bomo razvijali kompletnega spektra računalništva, bomo aktivni predvsem na dveh področjih:
- kompletna informatizacija industrijskih procesov (CAD/CAM sistemi)
- razvoj klasičnih informacijskih sistemov.

Medtem pa ne bomo razvijali kot prioritetnih vej, temveč samo veje, ki naj omogočijo večkraten izkoristek dominantnih
- domačih računalnikov
- informatične opreme za globalne sisteme.

Določeni proizvodi za ta področja bodo dopolnili obstoječo ponudbo, npr. home computer kot dopolnitev opreme za dom, povečini pa jih v SFRJ nihče ne bo proiciral.

V zvezi s kompleksno informatizacijo industrijskih procesov moramo opozoriti, da ne razpolagamo s projekcijami razvoja jugoslovanskih proizvajalcev strojne in procesne opreme.

Nedvomno bo ta razvoj v skladu s svetovnimi standardi in naša izvozna usmeritev bo vsiljevala svetovne standarde tudi v SFRJ k proizvajalcem strojne in procesne opreme.

Tudi omenjeni dominantni področji se ne bosta razvijali avtarktično, temveč ob specializaciji za posamezne njune komponente, ki bodo z izvozom v celoti pokrivale komplementaren uvoz. Tu omenimo zlasti storitve in programsko opremo.

4.2. Možnosti programa za vključitev v tržišče SFRJ in sveta

4.2.1. Komparativne prednosti enotnega računalniškega sistema v SRS

V primerjavi z drugimi projekti v SFRJ ima proizvodno poslovni kompleks enotnega računalnišega sistema v SRS, ki ga danes predstavljajo v organizacijskem smislu predvsem Iskra Delta, TGO Gorenje, SE in Liko, TRS Zagreb in ERA Novi Sad, v tehnološkem smislu pa spekter produktov od nivoja terminalov preko mikroračunalnikov, minijev, srednjih računalnikov, do kompletnih sistemov, ki so vsi medsebojno kompatibilni, bistveno prednost zaradi velikosti, tehnološke (komponente in postopkovne)baze, kadrovske baze, celovitega pristopa in odprtosti za vse načine povezovanja z drugimi potenciali v SFRJ.

Pristop, katerega bistvo je v nevezanosti na enega samega tujega partnerja na nivoju končnih produktov, sklopov ali komponent, omogoča fleksibilno povezovanje s tujci zaradi neobhodnega uvoza, kakor tudi povezovanje z vsemi jugoslovanskimi potenciali brez vpliva določene tuje TNC.

Povezovanje potencialov v SFRJ okrog enotnega programa bo trajno zagotovilo pritisk v popolno odprtost trga na področju informatike.

4.2.2. Možne bistvene pomanjkljivosti

Odsotnost kompleksne strategije tehnološkega razvoja v okviru strategije družbenega razvoja, odsotnost družbene verifikacije ciljev in družbene podpore za njihovo realizacijo je prvi faktor, ki lahko program zavre v razvojnih ambicijah.

Pomanjkljiva zavest o tem, da ima razviti svet tehnološko politiko celo na nadnacionalni ravni, in da jo stalno razvija v smislu podpore svojih ekonomskih in vojaško političnih interesov, lahko privede do tega, da bo informatika v svojem razvoju zapadla v totalno materialno in tudi moralno-mentalno odvisnost od TNC ob sprotni uporabi mehanizmov CoCom NATO pakta, OECD, bančnih mehanizmov, pa tudi pritiskov obveščevalnih služb itd. Poleg nevarnosti, ki jo predstavlja zahod, obstaja tudi nevarnost zaradi ambicij vzhoda. Seveda pa predstavlja največjo nevarnost naše nepoznavanje problematike in naša neorganiziranost za kompleksno pariranje pritiskom iz tujine.

Naslednja možnost - bistvena pomanjkljivost je gospodarska kriza, ki bi pomenila zmanjšano sposobnost financiranja produkcije in porabe, zmanjšanje povpraševanja v SFRJ ter povečan obseg naturalnih oblik gospodarjenja (trampa). V tem primeru bi določen del kadrov prešel k TNC, ki so prisotne v SFRJ, del pa bi odšel v tujino, tudi k TNC (nekateri bi vztrajali tudi v kriznih razmerjih, nekateri pa bi se prekvalificirali).

SHEMA 1. VSEBINA RAZVOJNE STRATEGIJE

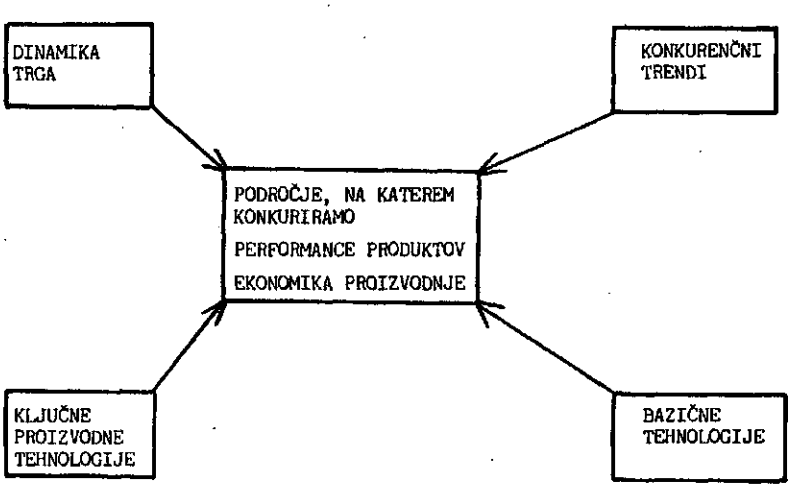
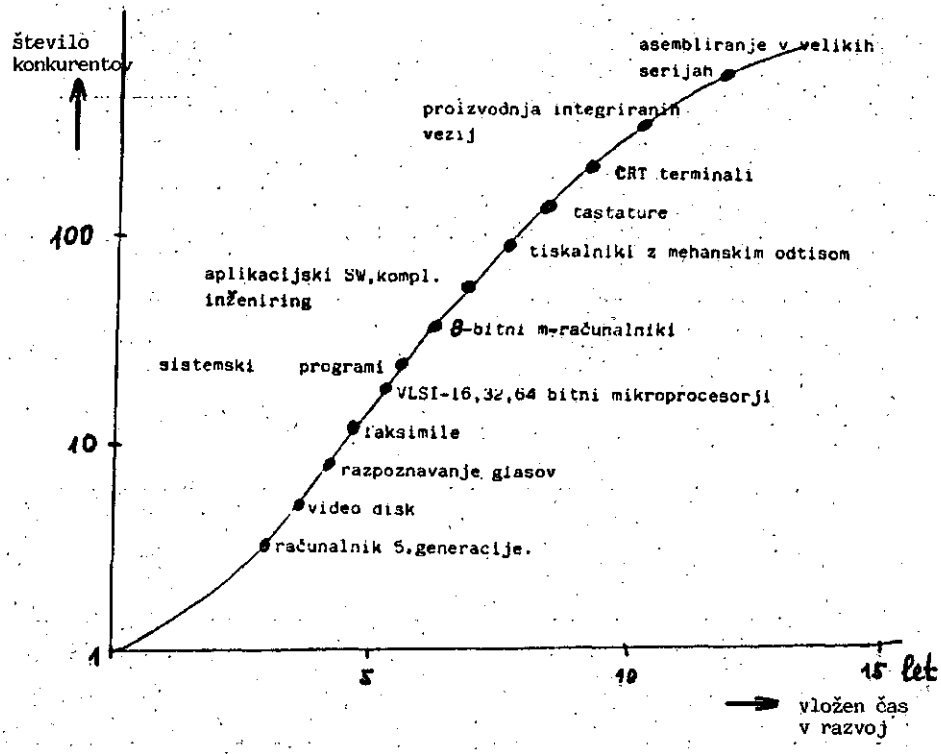


DIAGRAM 1. KONKURENCA NA RAZLIČNIH TEHNOLOŠKIH PODROČJIH



S tem bi prišlo do družbeno negativnega izkoriščanja kadrovskih potencialov v smislu povečanega odliva kapitala v tujino, ob tehnološki podpori naših delavcev v zastopniških OZD, ter v smislu koriščenja investicij na univerzah, institutih itd., za zagotavljanje cenene delovne sile tujim TNC.

4.3. Izbor lastne tehnološke strategije

Za izbor lastne tehnološke strategije je pomembno poznavanje kompleksne strategije SOZD in DO, kakor tudi strategije konkurence oz. okolja.

Najprej je potrebno opredeliti nekatere faktorje pri izboru lastne tehnološke strategije. Vsebina razvojne strategije je podana na diagramu 1.

Ekonomika proizvodnje produktov z določeno performanso nam v bistvu določa področje, na katerem bomo konkurirali. Bazične tehnologije, ki bodo determinirale razvoj, so gotovo prisotne pri največjih svetovnih proizvajalcih in v največjih raziskovalnih institutih pri nas gre za njihovo aplikacijo. Da bi sledili dinamiki trga in trendom razvoja konkurence, pa moramo vsekakor razviti ključne proizvodne tehnologije, ki bodo omogočile ekonomično proizvodnjo.

Če pogledamo stopnjo razvitosti posameznih tehnoloških področij, glede na število firm, ki delujejo na tem področju ter glede na čas, ki je bil na posameznem področju že uporabljen, vidimo, da je pri različnih tehnoloških področjih konkurenca zelo različna.

Izbor tehnološke strategije vedno diktira dva pogoja:
- tehnološka pozicija
- konkurenčna pozicija.

V informacijski industriji so tipični predstavniki tipa 1 IBM (računalniki za splošne namene), DEC (miniračunalniki), APPLE (mikroračunalniki), tipični predstavniki tipa 2 so Fujitsu (sledi IBM), Robotron (sledi IBM), tipični predstavniki tipa 3 npr. MDS (naprave za zbiranje podatkov za IBM). Jugoslovanske industrije do l. 1983 so vse spadale v tip 4.

SHEMA 2. TIPIČNE STRATEGIJE

konkurenčna pozicija \ Tehnološka pozicija	Tip razvoja	Kompleten	Selektiven
	Inovacijski	1. tehnološki voditelj	3. komplementaren razvoj poudarjen na področjih, ki jih ne pokriva voditelj
	reakcijski	2. zasledovalno - konkurenčen razvoj določenim zaostankom v času	4. tehnološki racionalizator

SHEMA 3. MOŽNE SPREMEMBE STRATEŠKIH POZICIJ

KONKURENČNA POZICIJA \ TEHNOLOŠKA POZICIJA	močna	ugodna	znosna
močna	tehnološko in tržno vodstvo	dobra tehnologija in močna tržna pozicija	zastarela tehnologija (zasledovanje) in močna tržna poz.
ugodna	tehnološko vodstvo in ugodna tržna pozicija	zastarela konkurenčna tehnologija, zasledovanje / komplementarna tehnologija	povečan obseg racionalizacije, ugodnejša tržna pozic.
znosna	komplementarna tehnologija in znosna tržna pozicija	mešana družba (Joint venture)	tehnološke racionalizacije

Spremembe tehnoloških in tržnih pozicij so možne tako glede na komplementarni pristop - razvoj nečesa, kar tehnološko in tržno vodilne dopolnjuje kot tudi na zasledovalni - konkurenčni pristop - razvoj nečesa, kar pri konkurenci že obstoji.

5. POGOJI AMBICIOZNEGA RAZVOJA

5.1. Sodelovanje z drugimi v SFRJ in v svetu

5.1.1. Sodelovanje v SFRJ

Proizvodno poslovni kompleks ERS vzpostavlja sodelovanje v SFRJ v samem proizvodnem smislu:
- upoštevanje vseh proizvajalcev komponent, sklopov in podsistemov ter namen o sovlaganjih v te proizvodnje
- standardizacija
- združevanja dela v skupen proizvod (na načelu skupnega prihodka in skupnega dohodka)

Stiki z vsemi potencialnimi kooperanti so najmanj kvartalni.

V širšem smislu se v SFRJ Iskra Delta povezuje vsemi potenciali na fakultetah, institutih, specializiranih zavodih v SFRJ z ambicijo, da se čimveč kadrov angažira na enotnem programu.

V področju financiranja poskušamo angažirati sredstva v celi SFRJ za financiranje katerekoli faze celotnega ciklusa reprodukcije.

Panoga se zaenkrat še ni uspela identificirati in zato je pri povezovanju v SFRJ potrebno odpraviti nekaj iluzij, kot so:
- "potrebno je opraviti delitev dela"
- "industrija ne želi angažirati znanstveno razvojnih potencialov"
- "informatiko je možno razvijati na majhnih osnovah in obratnih sredstvih".

Najprej moramo povedati, da v SFRJ ni problem delitev dela, ker je obseg produkcije zelo skromen, spekter potrebnih komponent izgradnje informacijske industrije zelo širok in nam danes manjka še mnogo sestavin celovitega sistema ter ker ni praktično tudi nikjer podvajanja tehnoloških postopkov razen na 16 bitnih računalnikih. Problem je združevanje tistega, kar imamo, standardizacija, ki bi omogočala kompatibilnost in povečanje serij tega, kar delamo tako, da bi en del lahko šel na OEM trg. Problem združevanja ima poleg tehnološkega tudi finančni in pravni aspekt, ki sta znana tudi iz drugih panog.

Ni res, da industrija ne želi angažirati znanstveno razvojnih potencialov na univerzah in institutih. Problem v računalništvu je podoben kot drugo. Za financiranje tega dela so potrebna določena sredstva, recimo 1 milijarda din letno. Da bi inovacije iz laboratorija prenesli na nivo industrijskih prototipov, prilagojenih na zaradi tehnoloških ali ekonomskihrazlogov razpoložljive materiale, delovne pripreve, stroje in postopke, jih prilagodili interni in eksterni standardizaciji, potrebujemo za to vsaj 10 milijard din. Da pa bi nato vzpostavili ekonomično proizvodnjo, je potrebno angažirati dodatne vire za osnovna in obratna sredstva, recimo 100 milijard din. Teh pa seveda ni. Seveda se ne zavzemam za uvajanje tujega znanja v naše produkte - največkrat da tujec poleg tehnološke dokumentacije na razpolago tudi ustrezne vire sredstev za zagon take nove proizvodnje - temveč za angažiranje dodatnih domačih virov, ki bodo omogočili bolj koristno zaposlitev raziskovalno razvojnih kapacitet.

Delovno mesto v informatiki ni poceni, zelo drago je. Potrebno je angažirati velika sredstva na delavca. V svetu velja, da je potrebno angažirati na enega delavca v informacijski industriji povprečno:
- osnovna sredstva 10.000 US$
- obratna sredstva 50.000 US$.

Od obratnih sredstev jih mora imeti velik del denarno obliko in pri nas morajo biti tudi v določeni meri v obliki deviz. Angažirana sredstva se obračajo relativno počasi, obrestne mere oz. dobički pa so zelo veliki. Tudi informacijski produkti ne bodo mogli na svetovni trg brez ustreznega finančnega inženiringa, kot se je to že pokazalo pri nekaterih drugih dobrinah (ladje). Nepoznavanje in nepriznavanje svetovnih procesov na tem področju nas lahko pega samo v objem tujih TNC.

5.1.2. Sodelovanje v svetu

Odsotnost proizvodnje določenih komponent, zagotavljanje svetovnih standardov v HW in SW, povezanost z drugimi industrijami, nas silijo na permanentno povezanost z drugimi proizvajalci v svetu. Način povezovanja, ki ga želimo doseči, naj bi bil čimbolj podoben načinom, ki vladajo v razvitem zahodnem svetu. V računalništvu je zelo razvit OEM trg, kar

pomeni, da je možno kupovati poljubne sklope, komponente ali podsisteme in jih vgrajevati v lastne produkte. Tega tipa odnosov se nameravamo posluževati tudi v prihodnosti.

Kooperacije in protidobave v smislu jugoslovanske zakonodaje bomo poskušali razvijati povsod tam, kjer bomo morali začeti na novo uvajati določene tehnologije.

Do leta 2000 še ne predvidevamo, da bi lahko prišlo do sodelovanja na bazi licenčnih pogodb na takem nivoju, kot je običajen na zahodu ali do resničnih sovlaganj, ki imajo za cilj skupen razvoj nove tehnologije in nove produkcije. Pri nas so TNC doslej šle v licenčni odnos ali sovlaganja predvsem kot v vlaganje za zagotovitev ustreznega deleža na jugoslovanskem trgu.

5.2. Odvisnost od tujega znanja

Odvisnost od tujega znanja je kompleksen pojem, ki bi ga lahko razdelili na nekaj nivojev. Poznamo naslednje oblike tujega znanja, ki pomenijo odvisnost:
- tuje znanje vsebovano v izdelkih in zaščiteno s patenti ter avtorskimi pravicami
- tuje znanje o proizvodnih postopkih (know-how)
- tuje znanje v svetovnih bankah podatkov, informacijskih sistemih itd.

Temu tujemu znanju lahko postavimo nasproti kot antipod domače znanje.

Vendar to ni povsem opravičeno zaradi same vsebine pojma znanja. Domače znanje, ki nima zveze s tujim znanjem, ne eksistira.

Vprašanje pa je odnos. Pogoje za računalniško proizvodnjo z jasnimi izvoznimi cilji je poznavanje in uporaba tujega znanja, ki pa mora biti mostrificirano predvsem preko:
- šolanja v tujini
- dela v tujini
- uporabe podatkov o patentih, raziskovalnih in razvojnih dosežkih, člankih, knjigah, revijah, uradnih listih itd.
- proučevanju in upoštevanju svetovnih (ISO) in tujih nacionalnih standardov.

Na tak način mostrificirano tuje znanje, ki mora biti z lastnim uvarjalnim naporom aplicirano v proizvodnjo, je pogoj za neomejevan lasten razvoj in za izvoz.

Mostrifikacija tujega znanja preko nakupa licenc, preko vzpostavitve kooperacij in preko skupnih vlaganj, kar obsega pojem "Višje oblike gospodarskega sodelovanja s tujino", bo seveda tudi prisotna na nivoju komponent, sklopov in podsistemov. Vendar bo povsod prisotna naša težnja, da imamo čimvečjo prostost v razvoju in v prodorih na tuje trge.

Višja oblika sodelovanja s tujino (licenca, kooperacija, joint venture) se bo morda pojavila tudi na nivoju računalniških sistemov, vendar smatramo, da bi nas tak tip odnosa v današnjem času bistveno zavrl v razvoju, ker računalniška industrija kot celota danes še nima izoblikovane svoje fiziognomije in ni sposobna ustrezno kreativno in družbeno optimalno izkoristiti takega tipa odnosov.

Danes je naša naloga predvsem v premagovanju letargičnih domačih kreativnih potencialov, ki bodo samo s tem, da bodo ustvarili določene izvozno sposobne izdelke in storitve, pridobili ustrezno komponentnost in sposobnost za kreativno uporabo tujega znanja.

6. ZAKLJUČEK

Namen tega pisanja je vzpodbuditi pri bralcih Informatike razmišljanje in diskusije o nekaterih zanje morda drobnih problemih, ki pa postajajo v določenem trenutku usodni. V Jugoslaviji se danes postavlja alternativa, ali vlagati še naprej pretežni del akumulacije v "včerajšnje proizvodne panoge", ki potrebujejo ogromno kapitala, energije in surovin, ali pa začeti pogumneje razvijati današnje in jutrišnje programe proizvodov, storitev in procesov, ki nam lahko zagotavljajo trajnejši in konkurenčnejši izvoz, zmanjšujejo energetsko odvisnost ter zagotavljajo bistveno večje možnosti zaposlovanja strokovnih kadrov. Ta alternativa se bo razreševala v odvisnosti od osebnega angažiranja prizadetih delavcev strokovnjakov in ne morda za njihovim hrbtom. Prav zato so dane v prispevku določene usmeritve, ki naj bralce polarizirajo, da bi bila rezultanta njihovih naporov usmerjena in da bo tudi njihov vpliv večji kot če bi se problemi razreševali v nepolariziranih oblikah, kjer je rezultanta naporov posameznikov pogosto zelo majhna ali nična.

Obvestilo avtorjem

Prosimo vse avtorje člankov, da nam s članki pošljejo še tele podatke: številko žiro računa, točen naslov avtorja in ime krajevne skupnosti in občine.

Uredništvo

Obaveštenje autorima

Molimo sve autore da nam uz radove pošalju još sledeće podatke: broj žiro računa, ime opštine i mesne zajednice i tačnu adresu autora.

Uredništvo

PROIZVODNJA MIKRORAČUNALNIŠKEGA SISTEMA PARTNER

ANTON P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.3

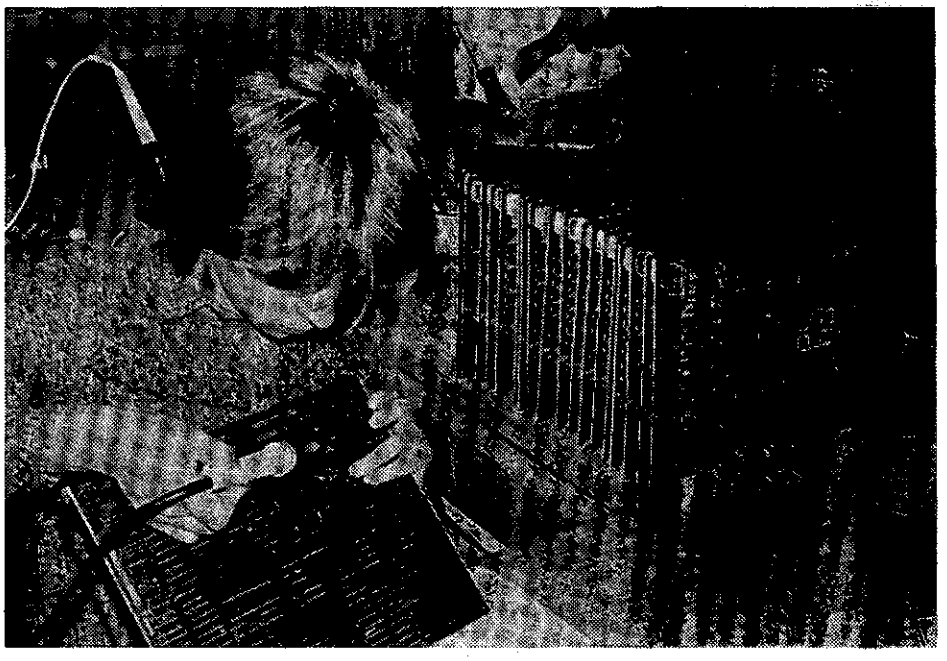
Ta prispevek opisuje v obliki slikovnega poročila proizvodnjo mikroračunalniškega sistema Partner, ki se proizvaja v Laborah pri Kranju. Sistem Partner se v proizvodnji približuje masovni reprodukciji, saj je v njegovem življenskem ciklu predvidena proizvodna količina 10000. To število je za nas novo po svoji velikosti in predstavlja nov pristop v proizvodni in tržni organiziranosti. Partner je osebni računalnik, mali poslovni sistem, sistem za razvoj programske opreme in v mrežo povezana delovna postaja.

Production of Microcomputer System Partner

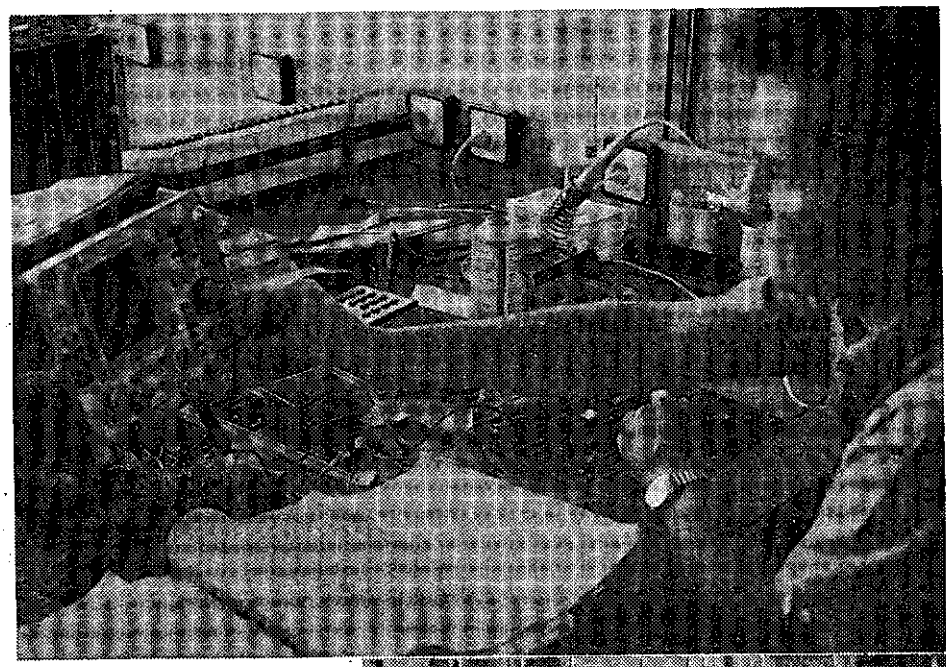
This article deals in a form of picture report with production of Partner Microcomputer System being fabricated in the Labore Factory near Kranj. This product is approaching the so called mass production and will reach a quantity of 10000 in its life cicle. This production quantity represents a new organizational and marketing experiance in Yugoslavia. Partner is a personal computer, a small bussiness system, a system for software development, and a work station in the Delta Computer Net.

Industrijska proizvodnja računalniških sistemov v Jugoslaviji -tudi v primeru mikrosistemov- je povezana z znanimi problemi industrializma na področju visoke tehnologije. Ti problemi zajemajo prizvodno zapletenost (kompleksnost), h kateri prispevajo veliko število osnovnih elementov (nekaj desettisoč v vsakem izdelku), mehanično in električno zapleteni in zmogljivi podsistemi (periferne naprave, njihovi elektronski vmesniki in krmilniki), zahtevne metode proizvodnega preizkušanja elementov, podsistemov in končnih sistemov (izdelkov). Takšen proces proizvodnje je mogoče organizirati le z intenzivnim prenosom znanja iz razvoja in z uporabo izkušenj iz prejšnjih, podobnih proizvodenj.

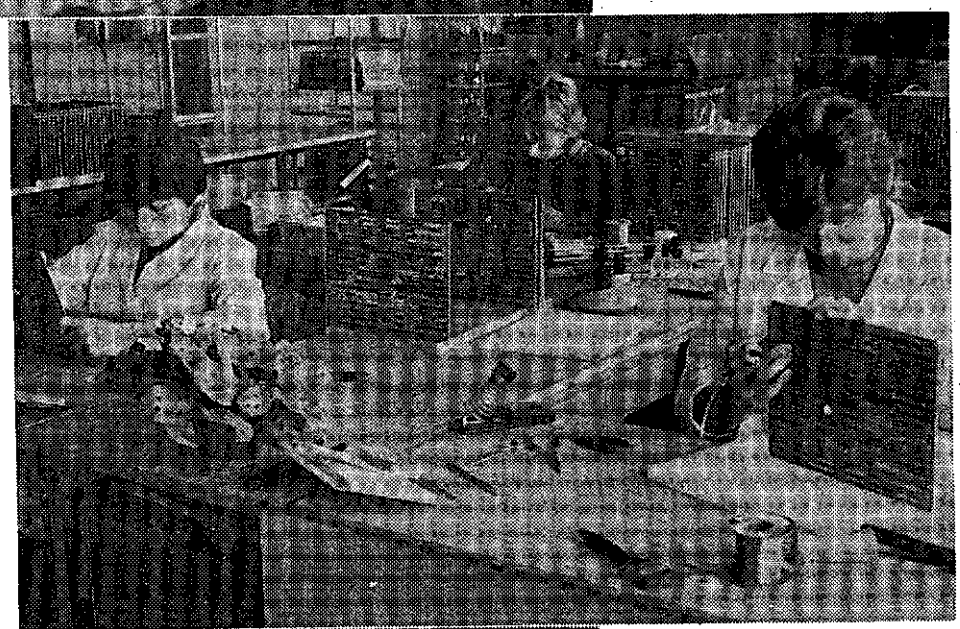
Slikovno poročilo daje splošen vpogled v delovni proces proizvodnje mikrosistema Partner. V tem zahtevnem procesu sodelujejo inženirji, tehniki in drugi delavci proizvodne enote Iskra Delta v Kranju. To poročilo prikazuje v slikah delo in napore teh delavcev.



Slika 1. Nada Mrgole je montažni mehanik. V kositrni kopeli obdelane tiskane plošče, ki jih vidimo na sliki, morajo biti še spremenjene, tj. modificirane. Plošča dobi določene preveze, ki so posledica razkoraka med načrtom, napako in dokončno zahtevo. Delo je natančno, terja pa tudi ročno spretnost in osredotočenost.



Slika 2. Gorazd Vidic je elektrotehnik, je kontrolni inženir, odogvoren za pravilno delovanje modulov na tiskani plošči. Modul je zapleteno vezje, ki ima več sto elementov (velikih, srednjih in malih integriranih vezij, diod, uporov, kondenzatorjev, tranzistorjev itd.). Kontrola signalov na osciloskopskem zaslonu zahteva znanje in osredotočenost.

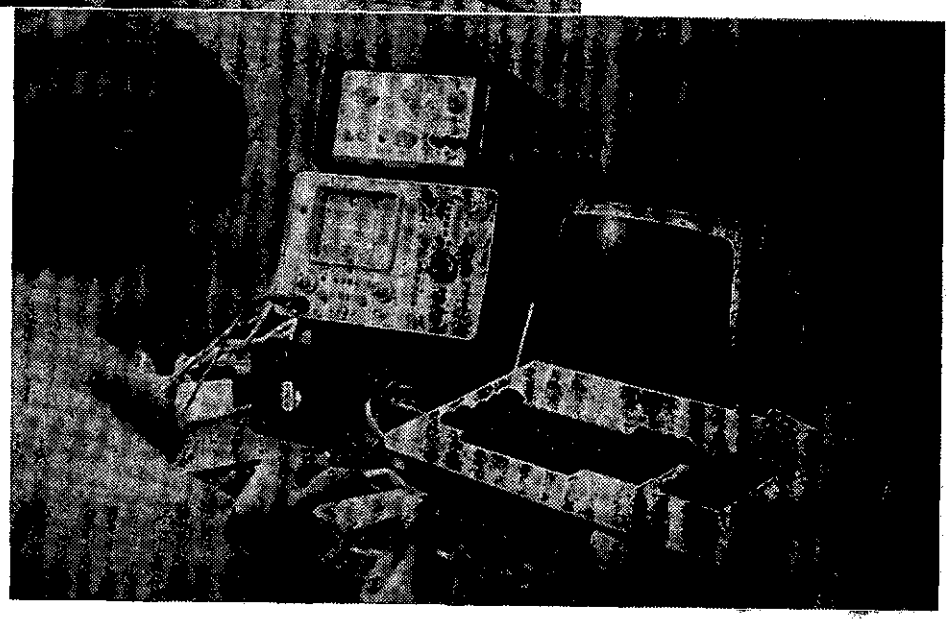


Slika 4. Modifikacija glavne tiskane plošče sistema partner je večstopenjska in delo je porazdeljeno. Na sliki vidimo delo montažnih mmehanikov Martine Rupnik, Antonije Hafner in Nade Mrgole.

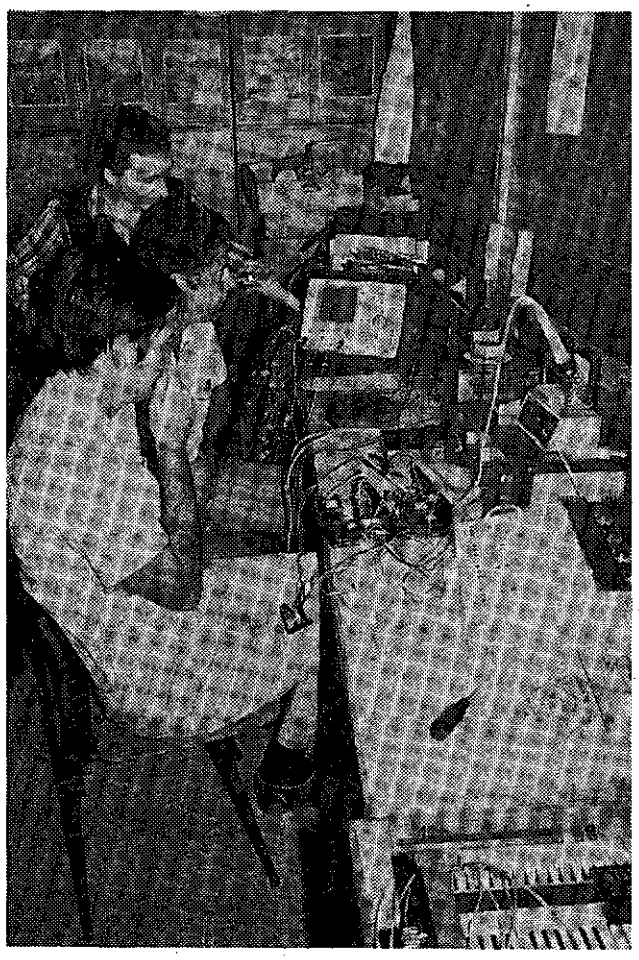
Slika 3. Marija Bajželj in Marta Grohar sta montažna mehanika. Mikroračunalnik Partner uporablja tudi sodoben preklopniški napajalnik (do 200 W), ki je narejen pretežno iz domačiih sestavnih delov (izjema so polprevodniški elementi).

Slika 5. Del razvojne skupine projekta Partner, ki je opravila tudi prenos znanja in tehnologije v proizvodnjo. Od leve proti desni: Lučka Rinkwitz (tajnica), Marko Kovačević, dipl.ing., mgr. Božo Blatnik, avtor tega prispevka in mgr. Drago Novak (vodja razvojnega projekta). Manjkajo: Saš Hadži, dipl. ing., Dušan šalehar, dipl.ing., Andrej Leskovar, dipl.ing. (razvoj preklopniškega napajalnika) in Bojan Drobež, dipl.ing..

Slika 6. Zbirna montaža plošč, diskov, tiskalnikov, ohišij, kablov itd. Sistem partner je sestavljen iz vinčestrskega diska (10M zlogov), upogljivega diska (800k zlogov), matričnega tiskalnika, zaslonskega in video dela, tiskanih plošč itd.

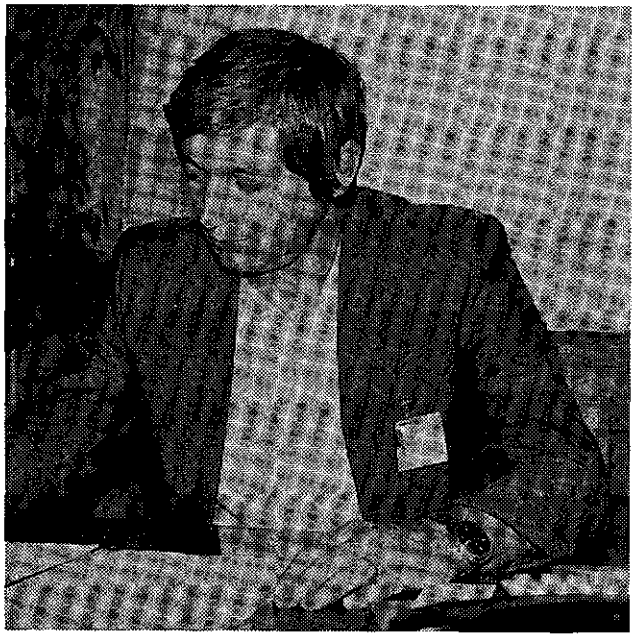


Slika 7. Tudi Tomaž Pibernik, elektrotehnik prispeva k končni kontroli tiskanih modulov.



Slika 8. Razvoj preklopniškega napajalnika in prenos ustreznega znanja v proizvodnjo je bilo delo Andreja Leskovarja, dipl.ing. (najbižje osciloskopu). Reprodukcija napajalnika je zahtevala vrsto popravkov in prilagajanje domačim elementom, seveda pa tudi posebno znanje na področju zapletenega vezja napajalnika.

Slika 9. Izdelovanje kablov oziroma njihovo spajanje s konektorji mora biti opravljeno natančno in zanesljivo: Boris Oblak, elektrotehnik na delovnem mestu.



Slika 10. Mgr. Milan Makinda, dipl.ing., vodja Območne enote Kranj (obrat Labore), kjer se proizvajajo mikroračunalniški sistemi Partner.

METODOLOGIJA SNOVANJA IZVRŠILNEGA PROCESORJA ZA 32-BITNI RAČUNALNIŠKI SISTEM

MAKSIMILJAN GERKEŠ

VISOKA TEHNIŠKA ŠOLA, VTO ELEKTROTEHNIKA, MARIBOR

UDK: 681.326

Članek podaja pregled postopkov snovanja, uporabljenih pri izdelavi prototipa 32-bitnega izvršilnega procesorja, ki je sestavni del 32-bitnega računalniškega sistema, ki podpira kompleksno instrukcijsko množico.
Izvršilni procesor je po predlaganem postopku v začetku definiran kot univerzalni trioperandni operator, ki ga postopoma razgrajujemo v vedno večje detalje. Vrednosti parametrov hrani posebna pomnilna struktura, ki je dopolnjena z delovno pomnilno strukturo in omogoča hranjenje vmesnih rezultatov zaradi interpretacije operacij, definiranih z instrukcijsko množico. Zaradi interpretacije operacij je vpeljana mikroprogramirana krmilna struktura, ki je definirana na podlagi karakterističnih lastnosti mikroprogramov.

DESIGN METHODOLOGY FOR A 32-BIT EXECUTION PROCESSOR FOR A 32-BIT COMPUTER SYSTEM: Article describes design methods used for a 32-bit execution processor prototype design. 32-bit execution proccessor is relatively high speed universal operator support for a 32-bit computer system based on a complex instruction set.
At the beginning we define execution processor as a universal three operand operator. With stepwise refinement it is then decomposed into lower level parts. In and outcomming parameter values are written into I/O register set which is completed with working register set for holding partial results, generated with interpreted operations, defined by complex instruction set. Because of operations interpretation microprogrammed control structure is defined based on the characteristic features of microprogramms.

UVOD:

Porazdelitev operacij, pri implementaciji neke računalniške arhitekture, na primerno organizirane sisteme je osnovna naloga vsakega snovanja. V preprostejših primerih so sheme, ki definirajo povezavo takih sistemov v celoto enostavne, organizacija sistemov pa je preprosta. Kadar pa je potrebno za implementacijo izdelati modele kompleksnih arhitektur, pa je takšna naloga znatno težavnejša. Pogosto imajo dominanten vpliv na organizacijo takega modela zahteve po performansah.

V našem primeru smo izdelali model organizacije izvršilnega procesorja, ki izvaja operacije določene s kompleksno instrukcijsko množico VAX arhitekture. Globalni model sistema je namreč takšen, da predvideva delitev operacij na instrukcijski in izvršilni del. Pri tem ima instrukcijski del sistema nalogo strežbe izvršilnemu procesorju, ki izvaja zahtevane operacije. Osnovna naloga instrukcijskega procesorja (strežba) definira lastnost instrukcijskega dela, ki je vhodno-izhodno intenziven. Medtem pa je izvršilni del (izvajanje operacij) operacijsko intenziven.

V nadaljevanju podajamo potek snovanja organizacije izvršilnega procesorja.

1. IZHODIŠČA ZA DEFINIRANJE ORGANIZACIJE IZVRŠILNEGA PROCESORJA

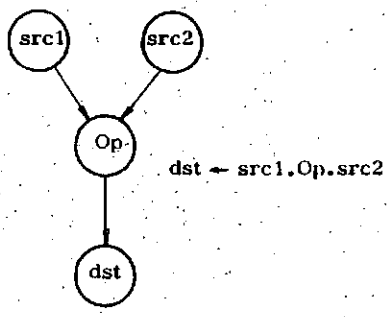
Organizacija izvršilnega procesorja je podrejena pospešenemu izvajanju operacij, ki so določene s kompleksno instrukcijsko množico. S stališča instrukcijskega procesorja, je izvršilni procesor samo razmeroma zmogljiv operator, ki mu instrukcijski procesor posreduje kodo zahtevane operacije in vrednosti parametrov (operande). izvršilni procesor pa vrne rezultat zahtevane operacije.

Analiza instrukcijske množice pokaže, da mora biti izvršilni procesor univerzalnega tipa, da prevladujejo tri operandne operacije in da je potrebna interpretacija večine operacij, ki so specificirane s kompleksno instrukcijsko množico.

Na podlagi opisanih lastnosti lahko definiramo začetne približke k organizaciji izvršilnega procesorja. Tako lahko prvi zahtevi po univerzalnosti priredimo operator univerzalnega tipa, trioperandni tip operacij pa ponazorimo po sliki 1.1.

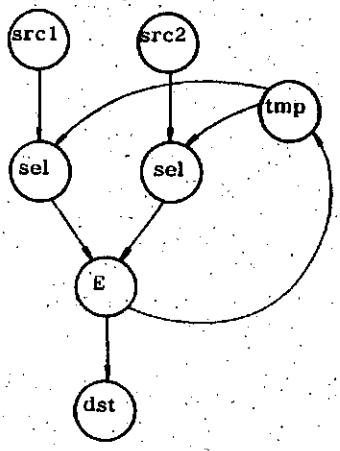
Oznake na sliki 1.1 pomenijo:

src 1 - prvi operand
src 2 - drugi operand
Op - univerzalni operator
dst - tretji operand (rezultat).



Slika 1.1: Začetni približek k organizaciji izvršilnega procesorja

Shema na sliki 1.1 izpolnjuje samo prvi dve zahtevi iz naše specifikacije. Zahteva po interpretaciji operacij $Op = \{Op_i \mid i = 1, 2, \ldots, n\}$ namreč predpostavlja, da lahko poljubno operacijo Op_i iz instrukcijske množice interpretiramo delomá ali v celoti z operacijami iz množice elementarnih operacij $E = \{e_j \mid j = 1, 2, \ldots, m\}$. Sedaj lahko shemo na sliki 1.1 dopolnimo tako, kot to podaja slika 1.2.



Slika 1.2: Dopolnjen model, ki omogoča interpretacijo operacij

Oznake na sliki 1.2 pomenijo:

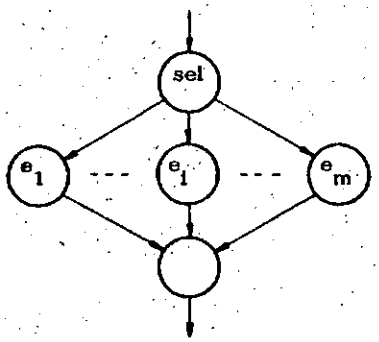
sel - selektorska operacija, ki izbere za operacijo operand iz src k ali tmp

tmp - nabor parametrov, ki omogočajo interpretacijo operacij iz Op. $tmp = \{tmp_e \mid e=1,2,\ldots,q\}$

S tem, da smo operacije iz Op nadomestili z operacijami iz E smo seveda predpostavili nek krmilni mehanizem, ki poskrbi, da se operacije iz Op interpretirajo z operacijami iz E. Po predpostavki lahko tedaj operacijo Op_i ponazorimo z eno ali več alternativnimi sekvencami, ki sestojijo iz operacij $\{e_j\}_{j=1,\ldots,m}$. Sekvencioniranje operacij iz E, tako da bodo ponazorile operacije iz Op, pa je naloga krmilnega mehanizma, ki ga bomo definirali kasneje.

Pri takšnem konceptu moramo tedaj za vsako operacijo Op_i, ki jo specificira ustrezna sekvenca sestavljena z elementi množice E, poskrbeti za selekcijo ustreznih operacij iz E. Če takšno sekvenco ponazorimo kot niz $\langle e_i \rangle$ moramo v vsakem koraku iz množice E izbrati ravno tisto operacijo, ki jo zahteva pripadajoči element operacijskega niza.

Začetni približek za takšno operacijo selekcije podaja slika 1.3.



Slika 1.3: Začetni približek k selekciji operacij iz E

Če posameznim elementom množice E priredimo selekcijske kode sel_i $i=1,2,\ldots,m$ lahko zapišemo tudi začetno krmilno enačbo selekcijske operacije

$$\text{izbrana operacija} = sel1 \wedge e_1 \vee sel_2 \wedge e_2 \vee \ldots sel_m \wedge e_m$$

kjer velja: $i \in \{1, 2, \ldots, m\}$,

$j \in \{1, 2, \ldots, m\}$,

$$(\forall i \neq j) \rightarrow sel_i \wedge sel_j \ .$$

Za izpolnjevanje performančnih zahtev takega operatorja, je potrebno grupiranje operacij iz E v tri osnovne skupine: osnovne aritmetične in logične operacije; operacije pomika, rotacije, maskiranja in ekstrakcije; operacije množenja. Šele tako lahko izpolnimo časovne zahteve, saj zahteva vsaka skupina operacij specifično organizacijo pripadajočega operatorja. Sedaj velja:

E = rot U mul U alu

kjer je:

rot - skupina operacij pomika, rotacije ...

mul - skupina operacij množenja

alu - skupina osnovnih aritmetičnih in logičnih operacij.

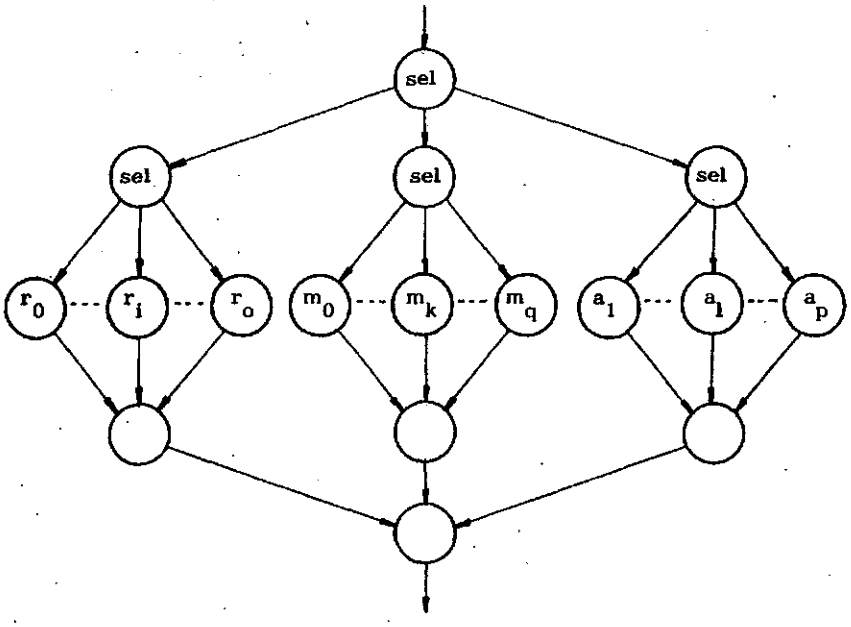
Ker se lahko v sistemski časovni enoti izvede naenkrat samo ena operacija iz izbrane skupine operacij lahko model krmiljenja že nekoliko dopolnimo po sliki 1.4.

Oznake na sliki pomenijo:

$\{r_i \mid i = 1, 2, \ldots, o\}$ = rot

$\{in_k \mid k = 1,2,\ldots,q\}$ = mul

$\{a_l \mid l = 1,2,\ldots,p\}$ = alu.



Slika 1.4: Podrobnejša krmilna shema za izbiro operacije

Glede na sliko 1.4 lahko prilagodimo tudi krmilno enačbo, saj smo vpeljali dva nivoja selekcije.

izbrana skupina operacij = sel 1. sk. $\wedge$ rot $\vee$

$\vee$sel 2. sk. $\wedge$mul $\vee$sel. 3. sk. $\wedge$alu

sel i. sk. - izbira skupine operacij

izbrana operacija 1. sk. = sel $1 \wedge r_1 \vee$ sel $2 \wedge r_2 \vee$ ....

$\ldots \vee$sel $o \wedge r_o$

izbrana operacija 2. sk. = sel $1 \wedge m_1 \vee$sel $2 \wedge m_2 \vee$ ...

$\ldots \vee$sel $q \wedge m_q$

izbrana operacija 3. sk. = sel $1 \wedge a_1 \vee$ sel $2 \wedge a_2 \vee$ ...

$\ldots \vee$sel $p \wedge a_p$

V celoti pa velja:

izbrana operacija = sel 1. sk. $\wedge$(sel $1 \wedge r_1 \vee$ ...

$\ldots \vee$sel $o \wedge r_o$)

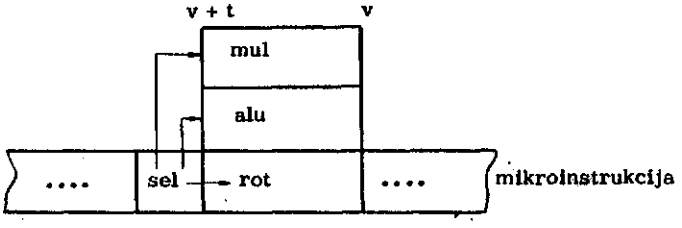
sel 2.sk. $\wedge$(sel i $m_1 \vee \ldots \vee$sel $q \wedge m_q$)

sel 3.sk. $\wedge$(sel $1 \wedge a_1 \vee \ldots \vee$sel $p \wedge a_p$)

Omenimo, da lahko takšno enačbo realiziramo z bralnimi pomnilniki. Tako definirana krmilna enačba ima tudi zanimivo lastnost s stališča mikroprogramske realizacije. V mikroinstrukcijo lahko namreč vpeljemo vertikalno funkcionalnost polj po sliki 1.5.

S tem smo tudi utemeljili enakost selekcijskih kod sel_i v operacijskih skupinah alu, mul, rot.

Čeprav razgradnja modela s tem še ni zaključena, smo



Slika 1.5: Zgradba polja mikroinstrukcije

osnovne karakteristike snovanja univerzalnega operatorja že zajeli.

2. DEFINICIJA INTERNE POMNILNE STRUKTURE IZVRŠILNEGA PROCESORJA

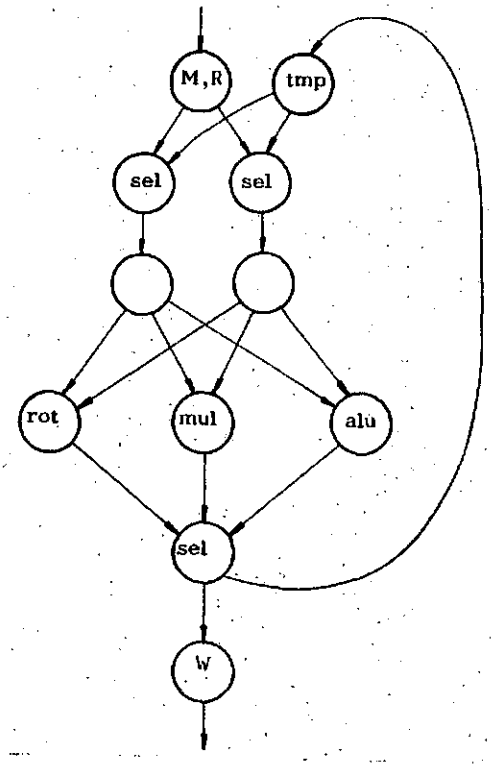
Za izpeljavo interne pomnilne strukture izvršilnega procesorja vzamemo model izvršilnega procesorja, ki ga podaja slika 2.1.

Pomen oznak je:

M, R - pomnilna struktura v katero vpisuje operande instrukcijski procesor

W - pomnilna struktura v katero vpisuje rezultate izvršilni procesor

Čeprav je shema pomnilne strukture na videz nepregledna, lahko krmilni sistem zelo jasno definiramo. Trioperandni tip operacij namreč zahteva v enem strojnem ciklu tri adrese operandov in sicer v začetku strojnega cikla čitanje do dveh izvornih operandov in ob koncu strojnega cikla destinacijsko adreso rezultata. Tako lahko pomnilni strukturi priredimo triadresni pomnilnik iz katerega lahko hkrati beremo do dva operanda, vanj pa vpišemo en operand. Operacije pomnilne strukture defi-

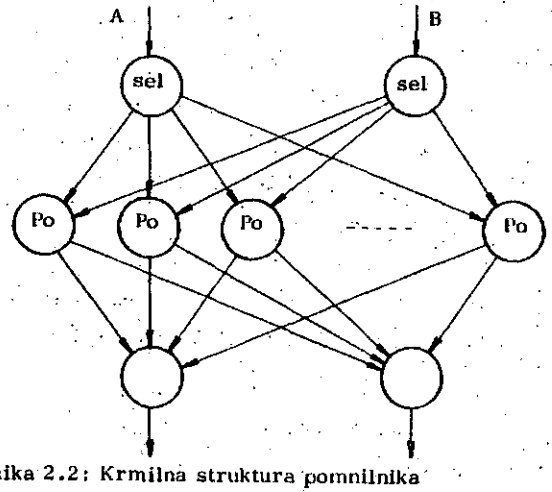


Slika 2.1: Organizacija izvršilnega procesorja

niramo takole:

read A, read B = R_{AB}
read A = R_A
read B = R_B
write B = W_B

Ker se čitanje in vpis časovno ne prekrivata, lahko preko adresne strukture B krmilimo čitanje in vpis operacij. kar nekoliko poenostavi strukturo pomnilnika. Pomnilno strukturo lahko tedaj ponazorimo kot n zaporednih lokacij. določenih z adresami. Nad vsako lokacijo lahko izvedemo operacijo čitanja ali vpisa. Zgoraj definiramo množico operacij nad lokacijami pomnilnika označimo s Po. Slika 2.2 podaja tedaj predlagano krmilno



Slika 2.2: Krmilna struktura pomnilnika

shemo interne pomnilne strukture.

Krmilni enačbi A in B strani se glasita:

Operacija A = sel 1$\wedge R_A \vee$ sel 2$\wedge R_A \vee \ldots \vee$ sel n$\wedge R_A$
$= R_A \wedge$ (sel 1$\vee$ sel 2$\vee \ldots \vee$ sel n)

Operacija B = sel 1$\wedge (R_B \vee W_B) \vee$ sel 2$\wedge (R_B \vee W_B) \vee \ldots$
$\ldots \vee$ sel n$\wedge (R_B \vee W_B)$
$= (R_B \vee W_B) \wedge$ (sel 1$\vee$ sel 2$\vee \ldots \vee$ seln).

Ker imamo opravka s tremi strukturami M, R, tmp, W definiramo nad vsako strukturo naslednje operacije:

M, R - R_A, R_B, R_{AB}
tmp - R_A, R_B, R_{AB}, W_B
W - W_B.

Da preprečimo hkraten vpis in čitanje postavimo:

WE - odobreno čitanje,
$\overline{WE}$ - odobren vpis.

Sedaj lahko krmilni enačbi zapišemo določneje:

operacija A = WE$\wedge$ sel M,$R_A \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel p)
$\vee$ WE$\wedge$ sel $tmp_A \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel q)

operacija B = WE$\wedge$ sel M, $R_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel p)
$\vee$ WE$\wedge$ sel $tmp_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel q)
$\vee \overline{WE} \wedge$ sel $tmp_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel q)
$\vee \overline{WE} \wedge$ sel $W_B \wedge$(sel1$\vee$ sel2$\vee \ldots \vee$ sel r)

Slika 2.3 podaja tako definirano krmilno strukturo pomnilnika izvršilnega procesorja.

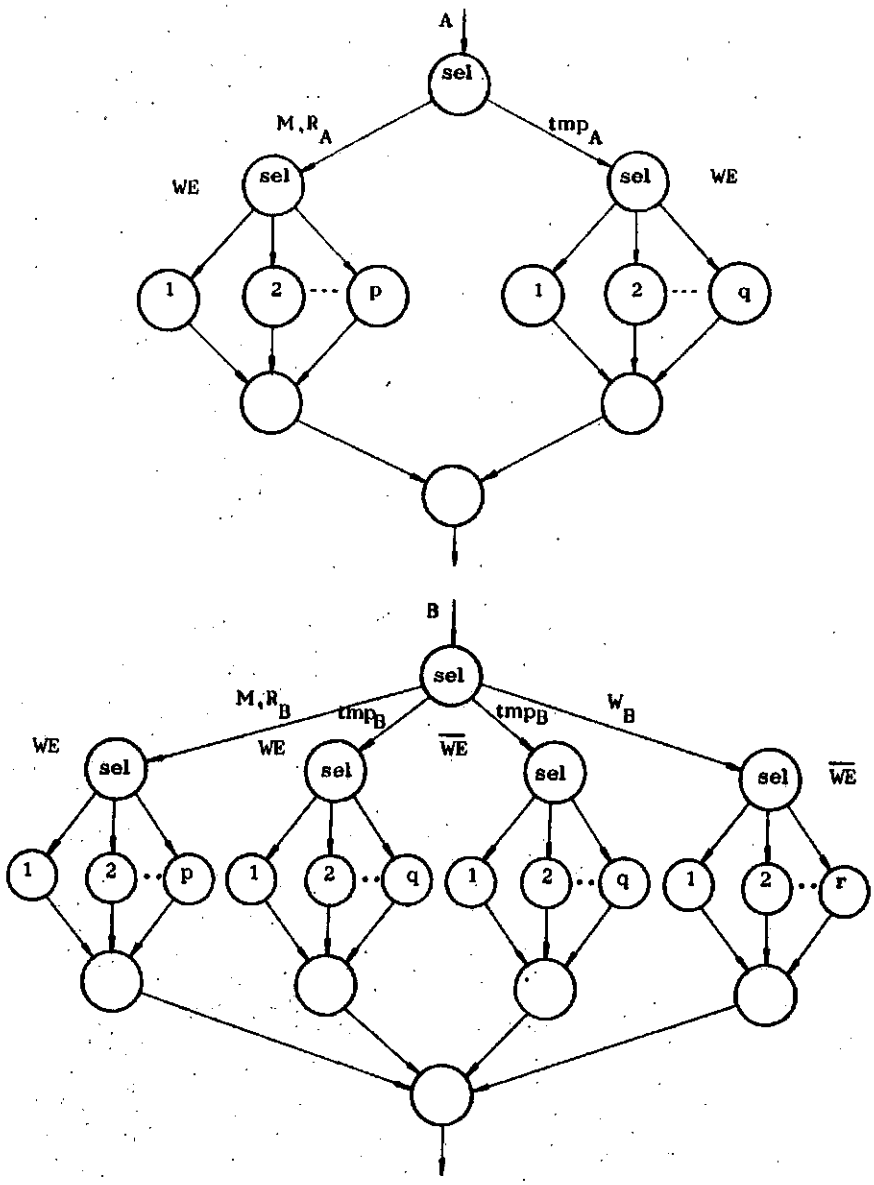
Organizacijo interne pomnilne strukture podaja slika 2.4, tako kot jo vidi izvršilni procesor.

Razgradnja interne pomnilne strukture izvršilnega procesorja s tem še ni zaključena, vendar pa že dosedanji postopek dovolj dobro ilustrira potek snovanja.

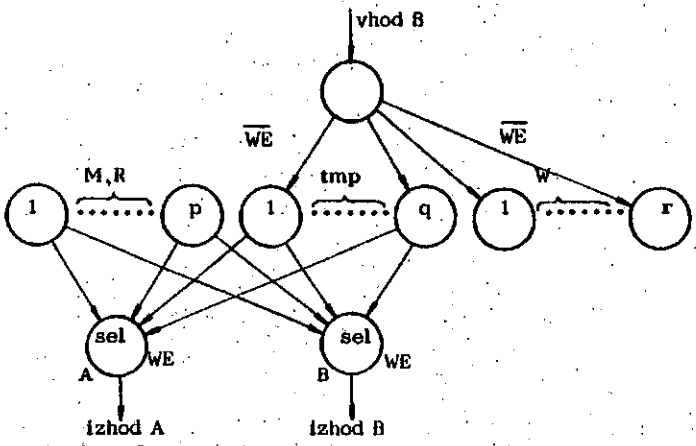
3. ORGANIZACIJA KRMILNE ENOTE IZVRŠILNEGA PROCESORJA

Organizacijo krmilne enote izvršilnega procesorja določimo s pomočjo analize lastnosti diagramov poteka, ki opisujejo operacije, specificirane z instrukcijsko množico. Diagrami poteka predstavljajo mikroprograme zapisane na registerskem nivoju v meta jeziku. Osnovna značilnost mikroprogramov je velika razvejanost. Zaradi časovnih zahtev je potrebno vejitve na eno izmed več možnih paralelnih vej izvesti v enem strojnem ciklu.

Naslednja značilnost je, da se identične skupine mikrooperacij pogosto ponavljajo v mikroprogramih sicer različnih instrukcij, kar omogoča definiranje primernih makroukazov ali podprogramov. Operacije v zankah so sicer v našem primeru manj pogoste, vendar se zaradi ča-



Slika 2.3: Krmilna struktura pomnilnika izvršilnega procesorja



Slika 2.4: Organizacija interne pomnilne strukture izvršilnega procesorja

sovnih zahtev izkaže smotrna realizacija zančnega mehanizma, ki omogoča tudi vgnezditev zanke.

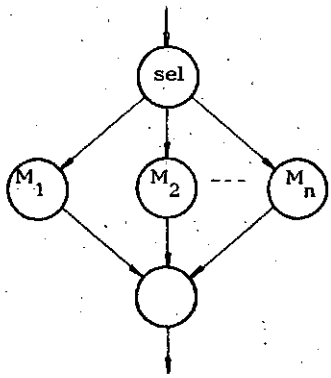
Zaradi razvejanosti mikroprogramov tedaj adresa naslednje mikroinstrukcije v splošnem ni enolično določena. Zato si prizadevamo mikroprogramski pomnilnik organizirati tako, da bi bilo možno vnaprej prebrati vse mikroinstrukcije, ki so možne naslednice mikroinstrukcije v izvajanju.

Prvi del naloge torej zahteva, da vsaki mikroinstrukciji v izvajanju določimo množico možnih naslednjih mikroinstrukcij. Drugi del naloge pa zahteva, da na podlagi statusov, ki se postavijo med izvajanjem tekoče mikroinstrukcije, izvedemo selekcijo izbrane mikroinstrukcije iz prebrane množice mikroinstrukcij. Takšno nalogo je možno rešiti s primernim asociativnim pomnilnikom.

V našem primeru smo ugotovili, da imajo vejitve na do štiri možne veje mnogo višjo frekvenco, kot vejitve na več kot štiri možne veje. Zato se odločimo, da vejitve na do štiri možne veje realiziramo s paralelnim branjem

štirih mikroinstrukcij, vejitve na več kot štiri veje pa klasično z modificiranjem naslednje adrese mikroinstrukcije z ALI funkcijo, vendar samo do štiri mikroinstrukcije natančno. Dokončna selekcija se opravi enako kot v primeru vejitve na eno izmed štirih možnih vej.

Izhodiščno krmilno shemo mikroprogramirane krmilne enote podaja slika 3.1.



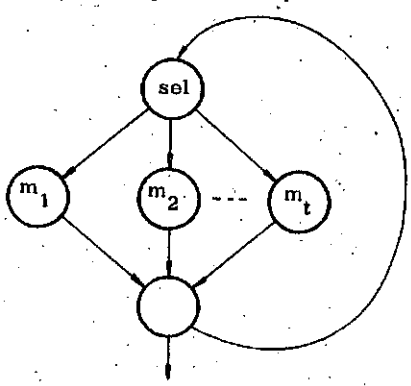
Slika 3.1: Izhodiščni model mikroprogramirane krmilne enote

Začetna predpostavka je, da je mikroprogramov toliko, kot je operacij v množici operacij Op. Na podlagi slike 3.1 lahko zapišemo enačbo za izbiro mikroprograma glede na zahtevano operacijo Op_i.

Izbira M_i = sel1 $\wedge M_1 \vee$ sel2 $\wedge M_2 \vee \ldots$
sel i $\wedge M_i \vee \ldots \vee$ sel n $\wedge M_n$

Pravila, po katerih teče izbira mikroprogramov za izvajanje, so seveda določena na nivoju instrukcijske množice. Model, s katerim lahko ponazorimo tako operacijo, je pomnilnik, razdeljen na podmnožice lokacij, kjer je na vsaki podmnožici lokacij zapisan mikroprogram.

Model krmiljenja izbranega mikroprograma M_i (selekcijo mikroinstrukcij) lahko ponazorimo po sliki 3.2.



Slika 3.2: Izbira mikroinstrukcij v mikroprogramu

Pravila, po katerih se izbirajo mikroinstrukcije, določimo sami.

Krmilna enačba za sliko 3.2 se glasi:

izbira m_i = sel 1 $\wedge m_1 \vee$ sel 2 $\wedge m_2 \vee \ldots$
sel i $\wedge m_i \vee \ldots \vee$ sel t $\wedge m_t$.

Doslej razvit krmilni model mikroprogramske krmilne enote podaja slika 3.3.

Za izbiro mikroinstrukcij znotraj mikroprograma izberemo princip "next" polja v mikroinstrukciji. Iz slike 3.3 lahko razberemo zanimivo lastnost takega načina izbire naslednje mikroinstrukcije. Z izbiro mikroprograma M_i omejimo gibanje kazalca, ki izbira mikroinstrukcije na razmeroma omejeno področje lokacij mikroprogramskega pomnilnika. Iz tega pa sledi, da je lahko velikost "next" polja znatno krajša od dolžine celotnega adresnega prostora mikroprogramskega pomnilnika. Pri tem je značilna naslednja lastnost takega pristopa: za izbiro mikroprograma lahko uporabimo identični mehanizem, kot velja za vejitveni krmilnik na nižjem abstraktnem nivoju (nivo mikroinstrukcij v izbranem mikroprogramu). Mehanizem razvijemo v poslošeni obliki.

Adreso ponazorimo kot binarni vektor:

$A = \langle a_{n-1}, a_{n-2}, \ldots, a_1, a_0 \rangle$

Bazno adreso, ki je osnova za določitev adrese A zapišemo:

$BA = \langle a_{n-1}, a_{n-2}, \ldots, a_i, \underbrace{0, 0, \ldots, 0}_{i} \rangle$

Pomik na bazno adreso ("next" polje, vejitev) izrazimo v obliki:

$PA = \langle 0, 0, \ldots, 0, a_{i-1}, a_{i-2}, \ldots, a_1, a_0 \rangle$

Adresa A je sedaj določena z relacijo:

$A = BA \,.\, ALI \,.\, PA$

$A = \langle a_{n-1}, a_{n-2}, \ldots a_i, 0, \ldots, 0 \rangle \,.\, ALI \,.$
$\langle 0, 0, \ldots, a_{i-1}, a_{i-2}, \ldots, a_1, a_0 \rangle$

Alternativni zapis zgornja relacije, ki je primeren za izvedbo v logiki s tremi stanji lahko zapišemo:

$A = 1BA' \, 1PA$, kjer je $'$ operacija konkatenacije nizov

$1BA = \langle a_{n-1}, a_{n-2}, \ldots, a_i \rangle$ in

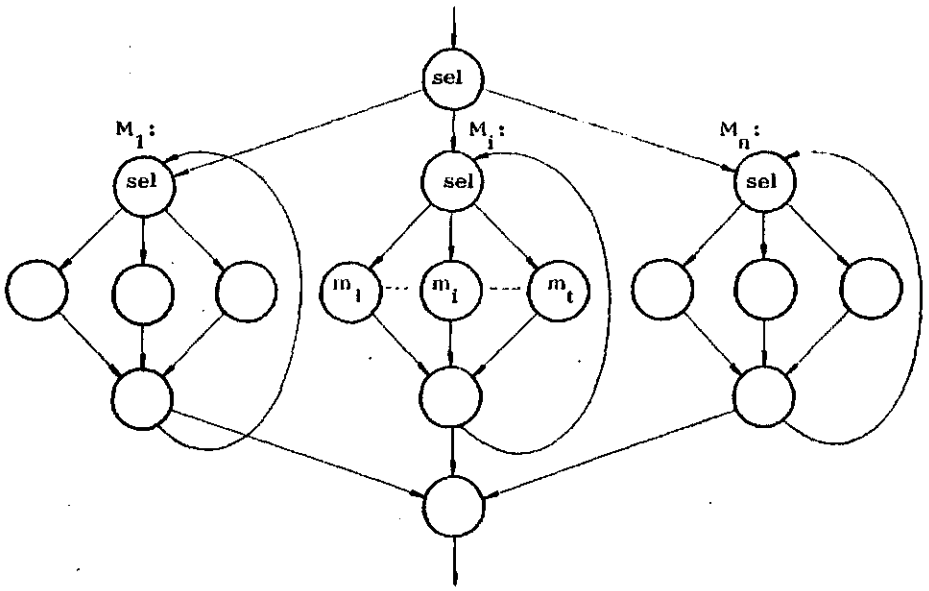
$1PA = \langle a_{i-1}, a_{i-2}, \ldots, a_1, a_0 \rangle$.

Glede na izhodiščne predpostavke tega razdelka opisani postopek tvorbe adrese za vejitveni krmilnik v našem primeru nekoliko dopolnimo. Bazna adresa ima v našem primeru naslednjo obliko:

$BA = \langle a_{15}, a_{14}, \ldots, a_6, 0, 0, \ldots 0 \rangle$.

Izbiro mikroinstrukcije, do štiri mikroinstrukcije natančno podaja prvi pomik:

$1PA = \langle 0, 0, \ldots, 0, a_5, a_4, a_3, a_2, 0, 0 \rangle$,

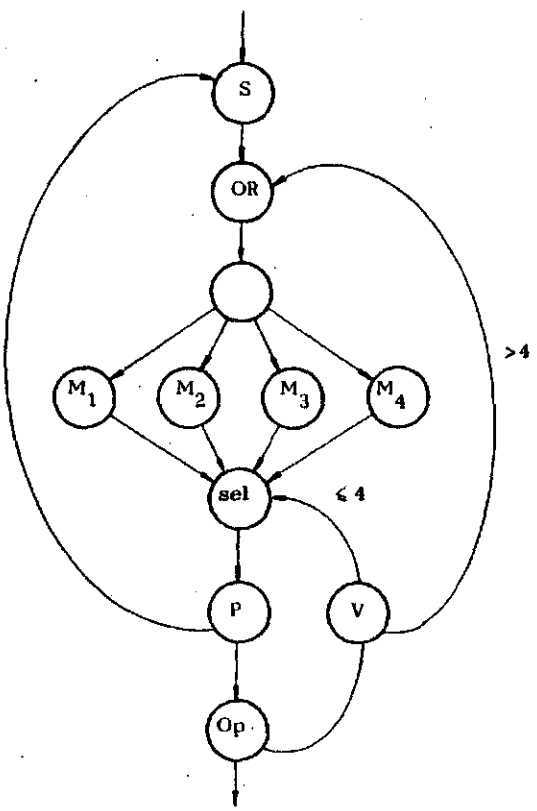


Slika 3.3: Model mikroprogramske krmilne enote izvršilnega procesorja

dokončno izbiro mikroinstrukcije pa določa drugi pomik (izbira ene izmed štirih pomnilnih matrik):

$2PA = \langle 0, 0, \ldots\ldots, 0, a_1, a_0 \rangle$.

Izvedbo tako zasnovane mikroprogramske krmilne enote podaja slika 3.4.



Slika 3.4: Organizacija mikroprogramske krmilne enote izvršilnega procesorja

Oznake na sliki 3.4 pomenijo:

S - sekvencer adres mikroinstrukcij

OR - ALI funkcija

M_i - mikroprogramski pomnilniki

sel - operacija selekcije

P - prekrivalni register

Op - univerzalni operator

V - krmilnik vejitev

Razgradnja mikroprogramske krmilne enote s tem še ni zaključena, podane pa so osnovne karakteristike za naš primer.

4. SELEKTORSKA OPERACIJA

Realizacija selekcijske operacije:

sel (sel . koda):

sel1 → operacija 1 (mikroprogram 1, mikroinstrukcija 1, ...)

sel2 → operacija 2 (.....)

.

.

.

seln → operacija n (.....)

izvedemo s pomočjo izraza:

izbira = $sel1 \wedge m_1 \vee sel2 \wedge m_2 \vee \ldots \vee seln \wedge m_n$.

V zgornjem izrazu je:

$m_i = \langle u_a, u_{a-1}, \ldots, u_1, u_o \rangle$

binarni vektor, kjer je:

$u_j \in \{0, 1\}$.

sel. koda je preslikava definirana takole:

sel. koda: $sel_i \longrightarrow \ldots$,

sel. koda: $\langle s_b, s_{b-1}, \ldots s_1, s_0 \rangle \rightarrow \wedge$,

kjer je:

$s_k \in \{0, 1\}$ in $\wedge \in \{0, 1\}$.

Selekcijsko kodo pogosto zapišemo kot logični produkt elementov s_k ob zahtevi, da je lahko pravilen en sam produkt v selektorski operaciji.

Zgled:
Izbiro mikroinstrukcije m_i v mikroprogramu M_j izvedemo takole:

sel. koda = 1 za mikroinstrukcijo m_i.

Za vse ostale mikroinstrukcije iz $M_j - \{m_i\}$ ima selekcijska koda vrednost 0. Tedaj lahko zapišemo:

izbrana $m_i = 0 \wedge m_1 \vee 0 \wedge m_2 \vee \ldots \vee 1 \wedge m_i \vee \ldots \vee 0 \wedge m_n$

izbrana $m_i = 1 \wedge m_i$

izbrana $m_i = m_i$

izbrana $m_i = \langle u_a, u_{a-1}, \ldots, u_1, u_0 \rangle_i$.

5. ZAKLJUČEK

Postopek navzdolnjega snovanja je ilustriran na primeru implementacije izvršilnega procesorja, ki izvaja operacije določene s kompleksno instrukcijsko množico. Za ponazoritev rezultatov so uporabljeni usmerjeni grafi in matematična logika.

Izhodiščni približek je bil, da lahko model procesorja specificiramo s selektorsko operacijo, ki ma toliko vej, kot je instrukcij. Vsaka veja tako realizira eno izmed operacij specificiranih s kompleksno instrukcijsko množico.

Zaradi interpretacije teh operacij z bolj elementarnimi operacijami, smo bili prisiljeni definirati lokalne parametre, katerih vrednosti hrani delovni pomnilnik. Posledica interpretacije je tudi mikroprogramska krmilna enota, ki je za programerja na nivoju kompleksne instrukcijske množice nevidna.

Ilustriran postopek snovanja je bil uporabljen pri izdelavi modela za celotni 32-bitni računalniški sistem, ki izvaja kompleksno instrukcijsko množico.

Osnovna prednost predlaganega postopka snovanja so mnogo krajši časi snovanja, pregledna struktura sistema in večja zanesljivost realizacije, saj je nevarnost neodkritih napak mnogo manjša kot pri klasičnih postopkih snovanja.

6. LITERATURA

1. M. Gerkeš, M. Družovec, M. Pernek: Aplikacija bipolarnega mikroprocesorja, poročilo o delu za leto 1983, raziskovalna naloga št. 03-2570/796-83, Visoka tehniška šola, VTO Elektrotehnika, Maribor 1983.
2. S. P. Kartashev, S. I. Kartashev, eds.: Designing and Programming Modern Computers and Systems, Vol. 1 LSI Modular Computer Systems, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1982
3. J.K.Iliffe: Advanced Computer Design, Prentice-Hall International, Inc., London, 1982

PROGRAMSKI JEZIK PASCAL I

MATJAŽ GAMS (1),
IVAN BRATKO (2,1),
VLADIMIR BATAGELJ (3),
ROBERT REINHARDI (1),
MARK MARTINEC (1),
MARJAN ŠPEGEL (1),
PETER TANCIG (1)

UDK: 519.682.8

(1) INSTITUT „JOŽEF STEFAN", LJUBLJANA
(2) FAKULTETA ZA ELEKTROTEHNIKO, UNIVERZA EDVARDA KARDELJA, LJUBLJANA
(3) FAKULTETA ZA NAROSLOVJE IN TEHNOLOGIJO, UNIVERZA EDVARDA KARDELJA, LJUBLJANA

V članku je opisano mesto programskega jezika Pascala glede na ostale višje programske jezike. Opisani so značilni predstavniki najbolj pomembnih skupin programskih jezikov in posebej algoritmični jeziki, kamor uvrščamo tudi Pascal. Sledi kratek opis zgodovine Pascala in zaključna primerjava z ostalimi jeziki. Skozi celoten članek je osnovno merilo enostavnost oz. zapletenost pri programiranju s stališča človeka - uporabnika.

P r o g r a m m i n g L a n g u a g e P a s c a l I (comparision with other programming languages). The programming language Pascal is compared to other high level programming languages. Programming languages are devided into most important groups and one representative of each group is very shortly described. Special care is devoted to "algorithmic" languages. Finally some conclusions are drawn. Throughout the paper the emphasis is on the human engineering side.

1. Uvod

Programski jezik Pascal se v svetu čedalje bolj uveljavlja, čeprav ni miljenček nobene velike računalniške firme, tako kot npr. IBM podpira PL/1. V zadnjih letih pa se v literaturi poleg hval pojavlja precej kritik Pascala. Marsikdaj so te kritike upravičene. Zato je smiselno zbrati te kritike in jih objektivno oceniti /25/. Poleg teh pogosto drobnjakarskih kritik pa je potrebno oceniti Pascal tudi v širšem kontekstu programskih jezikov. Prava uveljavitev Pascala šele prihaja /1/, njegova usoda, kot enega najboljših predstavnikov skupine "algoritmičnih" (tudi postopkovnih, proceduralnih) jezikov, pa je v daljni bodočnosti negotova.

Tudi drugi razlogi so botrovali nastanku tega članka, npr. zmotne trditve v literaturi, da je Pascal primeren predvsem za učenje, ali marsikatera neutemeljena kritika v strokovni literaturi. Po drugi strani pa v nekaterih okoljih prevladuje pretirano navdušenje glede Pascala. Pomembno je dejstvo, da Pascal (upravičeno) prevladuje v slovenskem šolstvu /2/.

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

2. Programski jeziki

Na kratko se bomo ustavili pri značilnih skupinah visokih programskih jezikov (glej sliko 1). Osnovno merilo za razdelitev je način izražanja oz. oblikovanja ideje v delujočo kodo.

2.1. Skupine visokih programskih jezikov

SKUPINA	PREDSTAVNIK	SKRAJNI DOMET(*)
algoritmični j.	FORTRAN	PL/1
	COBOL	PL/1
	ALGOL-60	ALGOL-68
	PASCAL	ADA ?
	BASIC	
	C	
matrični j.	APL	(izvedenke?)
funkcijski j.	LISP	-"-
logični j.	PROLOG	-"-
vzorčni j.	SNOBOL	-"-
objektni j.	SMALLTALK	-"-

Slika 1. Skupine programskih jezikov.

(*) Tudi nenor, izvedenka, strannost, angleško "black hole".

Pri ocenjevanju uspešnosti programskih jezikov, ki smo jih označili s "SKRAJNI DOMET", so mnenja deljena. Npr. Hoare v /3/ očita tovrstnim jezikom kopico pomanjkljivosti, nekateri drugi avtorji pa jih zagovarjajo. O tej problematiki bomo še govorili v nadaljevanju članka.

Vsako skupino jezikov smo poskusili okarakterizirati samo z enim pridevnikom, ki skuša opozoriti na stil oz. način programiranja, ki ga jezik najbolj podpira. To je bilo tudi osnovno merilo za razdelitev jezikov v skupine. Stili programiranja so opisani hkrati z opisom skupin. Skupine so v veliki meri privzete po /4/. Pri vsaki

skupini je ocenjen najbolj značilen jezik in primerjava s Pascalom. Velja naslednje:
-vsaka skupina jezikov je za določeno področje bolj primerna kot druga skupina.
-V Pascalu lahko z nekaterimi spremembami (najčešče s spremembo stila programiranja in naborom podprogramov v knjižnjicah) dokaj uspešno konkuriramo nekaterim jezikom.

2.2. Opis najbolj značilnih nealgoritmičnih jezikov

APL (A Programming Language) temelji na konverzacijski interakciji preko posebnega terminala. Nastal je okoli šestdesetih let. Je funkcijski jezik z obsežnim naborom uporabnih podprogramov. APL ima goreče nasprotnike in zagovornike. Zagovorniki trdijo, da so bistveno bolj produktivni, kadar uporabljajo ta jezik /5/. Nasprotniki trdijo, da je jezik popolnoma nepregleden. Po mnenju nekaterih strokovnjakov je mogoče podobno produktivnost doseči v večini algoritmičnih jezikov, če uporabljamo knjižnjice z ustreznim naborom podprogramov. APL na prvi pogled ločimo od ostalih jezikov po množici nenavadnih znakov. Poglejmo si primer enostavnega stavka v APLju:

```
MID ← (A + B)÷2 ◇ FM ← FUN MID
```

LISP (LISt Processor) je najbolj široko uporabljan jezik umetne inteligence. Nastajal je okoli leta 1960 /6/. Poglejmo si primer stavka v LISPu, tj. telo rekurzivne procedure, ki računa faktoriela:

```
((COND ((ZEROP N) 1)
       (T (TIMES N (FACT (DIFFERENCE (N 1))))))
)
```

LISP podpira funkcijsko programiranje /7/, ki pa ga po našem mnenju lahko v veliki meri uporabljamo tudi v Pascalu. Funkcijsko programiranje je stil oz. način programiranja tako kot npr. strukturirano ali objektno programiranje. Zelo preprosto novedano je funkcijsko programiranje tako, da prevladujejo funkcije, oziroma vrednotenje in ne prireditveni stavki kot v običajnem postopkovnem programiranju. Poglejmo si na preprostem primeru stil funkcijskega programiranja:

```
IF NaRobuDosega(letalo)
  THEN SproziOpozorilo(letalo)
  ELSE IF ZnotrajDosega(letalo) THEN
    IF Nenajavljeno(letalo)
      THEN SprozIAlarm(letalo)
      ELSE IF NeznanoLeteceVozilo(letalo)
        THEN SproziPojzvedbo(stan)
        ELSE continue;
```

Na tak način lahko programiramo precej podobno kot v funkcijskih jezikih, kar pa Pascal ni. Prav tako Pascal nima množice ugodnih lastnosti LISPa, npr. udobnega procesiranja seznamov.

PROLOG je novejši jezik umetne inteligence. Omogoča nam tudi nedeklarativno in logično programiranje /8,9/, ki je v Pascalu često težko izvedljivo. Na PROLOG lahko gledamo tudi kot na produkcijski sistem /10,11/, kar je ena izmed alternativ bodočnosti programskih jezikov /1/. V Pascalu je smiselno uporabiti principe produkcijskih sistemov zlasti pri strukturiranju podprogramov /11/. Poglejmo si primer deklarativnega in nedeklarativnega načina programiranja na podprogramih za združevanje dveh seznamov in za ugotavljanje vsebovanosti elementa v seznamu:

```
PROLOG
conc([],X,X).
conc([X|Tail],Y,[X|Z]) :- conc(Tail,Y,Z).

Pascal
procedure Conc(x,y: tipSeznam; var z:tipSeznam)
BEGIN
  IF x = nil THEN z := y
    ELSE
     BEGIN
       Conc(x↑.next, y, z);
       DodajVSeznam(x↑.vsebina, z)
     END
END;(*Conc*)

PROLOG
member(X,Z) :- conc(L1,[X|L2],Z).

Pascal
FUNCTION Member(x: tipElement; z: tipSeznam) :
         boolean;
BEGIN
  IF z = nil THEN Member := false
    ELSE IF x = z↑.vsebina THEN Member := true
      ELSE Member := Member(x,z↑.next)
END;(*Member*)
```

PROLOGov stavek "member" preberemo takole: Element "X" je član seznama "Z", kadar lahko združimo nek seznam "L1" in seznam, ki je sestavljen iz elementa "X" in seznama "L2", v seznam "Z".

V Pascalu smo dokaj podobno kot v Prologu rešili nalogo z združevanjem dveh seznamov. Pri nedeklarativni verziji iskanja vsebovanosti pa vidimo, da tak način programiranja v Pascalu ni mogoč.

SNOBOL (String Oriented Symbolic Language) je vzorčni jezik/12/. Nastajal je koncem šestdesetih let. Novejše variante SNOBOLa so SPITBOL, SITBOL in FASBOL. SNOBOL omogoča asociativno in vzorčno programiranje. Poglejmo si princip vzorčnega programiranja v SNOBOLu:

```
 U 'Z' BREAK('+,-') = 'FIRST' : S(SEM) F(TJA)
```

Zgornji stavek preberemo takole: če se v spremenljivki "U" (tipa niz) pojavi vzorec, ki se začne s črko "Z" in mu sledi poljubno število znakov do znaka "+" ali "-", celoten vzorec zamenjamo z nizom "FIRST" in izvajanje nadaljujemo na oznaki "SEM". Če vzorca v spremenljivki "U" ne najdemo, se izvajanje prenese na "TJA".

S Pascalskim prevajalnikom, ki dopušča spremenljivo dolžino polj (dinamična polja), npr. kot v novem ISO standardu/22/, lahko dokaj uspešno konkuriramo SNOBOLu.

SMALLTALK je objektni jezik ali bolje rečeno sistem. SMALLTALK je del sistema, ki je nastal kot raziskovalni projekt osebnih računalnikov (Dynabook) /13/. Te raziskave imajo velik vpliv na razvoj najmodernejših osebnih računalnikov. Za vse te sisteme je značilno, da so močno povezani s celotnim programskim okoljem, ki je v Pascalu nomankljivo. Kaj je to objektno programiranje? Poglejmo si najprej razlike med vrednostmi in objekti. Vrednosti so abstrakcije, zato jih ne moremo spreminjati ali jim nastaviti vsebine - so časovno nespremenljive. Objekti so časovno spremenljivi, imajo stanje, ki odgovarja realnemu svetu. Objektom lahko spreminjamo vsebino, jih kreiramo ali brišemo, ali jim damo ime. Uporabnik vedno vidi objekte le "od zunaj", vsi objekti so zani enaki (ni različnih tipov). Uporabnik pošlje objektu sporočilo in objekt izvrši opravilo sam ali

tako, da drugim objektom pošlje sporočila. Poleg aktivnosti in komunikacije je pomembna lastnost dedovanje oz. deljivost. Na primer, zemlja in pomaranče sta okrogli in imata skupni vse lastnosti, ki pripadajo lastnosti "okroglost".

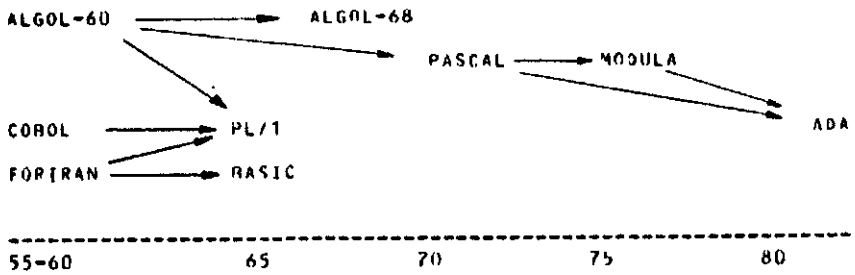
Stil objektnega programiranja je za Pascal precej neprimeren, čeprav ima velika večina jezikov vsaj nekaj objektnega. Po /14/ je programiranje objektno usmerjena matematika in matematika vrednostno usmerjeno programiranje. Pascalski zapis (record) ima nekaj lastnosti objektov, vendar ne podpira objektnega programiranja. Okviri (frame) v LISPu so veliko bližje pojmu objekta.

2.3. Opis značilnih algoritmičnih jezikov

Poglejmo si razvoj algoritmičnih jezikov:

BASIC (Beginner's ALL-purpose Symbolic Instruction Code) so razvili leta 1965 kot enostaven jezik za učenje programiranja in kot stopničko do FORTRANa in ALGOLa. BASIC ima kljub svoji relativni skromnosti pomembno vlogo v računalništvu kot jezik za pogovorno delo (interaktivnost) in v zadnjih letih zaradi svoje majhnosti in preprostosti kot jezik za domače računalnike.

PL/1 (Programming Language One) je nastajal v letih 1960 - 1965. Prvi prevajalnik je postal dostopen po letu 66. V PL/1 so upoštevane izkušnje FORTRANa, ALGOLa, COBOLa, itd., torej je jezik s širokim področjem uporabe in tudi primerno močnimi konstrukti. V PL/1 je vse, kar se da razumeti, tudi dovoljeno, zato poskuša razumeti tudi napake. PL/1 podpira IBM in to je verjetno tudi eden od vzrokov za precejšnjo razširjenost tega jezika.



Slika 2. Razvoj algoritmičnih jezikov.

Na sliki 2 vidimo razvoj algoritmičnih jezikov. Slika je prevzeta pretežno po /15/.

FORTRAN (FORmula TRANslation). FORTRAN je nastajal okoli leta 1955 kot program za prevajanje matematičnih postopkov v računalniške programe. Prvotno zelo enostavni verziji so se kmalu pridružile izpeljanke: FORTRAN II, III in IV. FORTRAN je že okoli leta 1960 postal zelo razširjen jezik. Leta 1966 sta bila sprejeta dva standarda za FORTRAN, ki približno odgovarjata FORTRANu II in FORTRANu IV. V zadnjih letih so se pojavili še FORTRAN V, FORTRAN 77 in cela vrsta "strukturiranih" FORTRANov. FORTRAN se uporablja predvsem za reševanje znanstvenih in tehničnih problemov.

ALGOL (ALGOritmic Language) je nastajal okoli leta 1960. To je bil prvi jezik, ki je bil tudi formalno opisan. Iz tega poskusa je zrasla teorija formalnih jezikov in prevajalnikov. ISO standard za ALGOL 60 je bil izdan leta 1965. ALGOL je močno vplival na nadaljni razvoj programskih jezikov, vendar je ostal pretežno evropski jezik. V Ameriki so ga izpodrivali jeziki, za katerimi je stala ameriška računalniška industrija. Kot izvedenka ALGOLa 60 se je pojavil ALGOL 68, vendar se ni uveljavil. Uspešnejše izvedenke so bile ALGOL W, SIMULA in PASCAL.

COBOL (COmmon Business Oriented Language) je nastal kot jezik za poslovne obdelave okoli leta 1960. Standardiziran je bil leta 1968. Za COBOL je značilna težnja, da bi bil čim bolj "naraven" - podoben naravnemu jeziku (angleščini). Ta lastnost naj bi omogočala precejšnjo samodokumentiranost programov in tudi preverljivost s strani neprogramerjev, npr. vodstvenih delavcev. COBOL in FORTRAN sta še vedno najbolj uporabljana programska jezika.

MODULA je nastala kot naslednik Pascala. Kmalu jo je nadomestila precej bolj uspešna MODULA-2 /16,17/. MODULA-2 je večnamenski jezik, ki je predvsem namenjen za implementacijo sistemov na mikroračunalnikih. Jezik je definiral N. Wirth, implementiran je bil leta 1980.

ADA (Ada Avgusta Lovelace - pionirka računalništva) je jezik, katerega izdelavo je naročil DOD (Department of Defense), to je ameriško obrambno ministrstvo. ADA naj bi bil "jezik vseh jezikov", torej naj ni bil zelo široko uporabljan, predvsem pri numeričnih nalogah, sistemskem programiranju in paralelnem izvrševanju programov v realnem času. Pri snovanju jezika je sodelovala množica strokovnjakov, ki so skušali v ADI združiti dobre lastnosti vrste jezikov, od Pascala do MODULE ali Concurrent PASCALa, Euclida ali CLUja. ADA ima zelo močno izražen koncept modulov. Kljub dolgoletnim naporom (od leta 75 dalje) še danes ni delujočega prevajalnika, ki bi implementiral vse opisane lastnosti jezika. Zadnja leta naj se pojavljajo podmnožice ADE, npr. JANUS /18,19/.

2.4. Ocena algoritmičnih jezikov

FORTRAN, COBOL in BASIC so splošno razširjeni jeziki in ne kaže, da bodo v bližnji bodočnosti v večji meri odstopili mesto drugim, čeprav boljšim jezikom. Ta prehod bo kvečjemu postopen. FORTRAN je primeren zlasti za numerične naloge za znanstvene raziskave. COBOL se uporablja predvsem za poslovne aplikacije, BASIC pa za učenje in enostavne poslovne aplikacije. Za vse tri jezike je značilno, da se pojavljajo čedalje novejše izpeljanke teh jezikov, ki v veliki meri privzemajo konstrukte, ki podpirajo

strukturirano programiranje kot v Pascalu. Ti jeziki so precej enakovredni Pascalu, čeprav večina programerjev, ki so dalj časa programirali v katerem koli omenjenih treh jezikov in Pascalu, raiši programira v Pascalu. Pascal nima tako obsežnega nabora standardnih podprogramov kot FORTRAN, vendar je v standardu določena možnost klicanja podprogramov v FORTRANu. COBOL ima pred Pascalom to prednost, da ima datoteke z direktnim dosegom, indeksno sekvencialne datoteke in lepe možnosti formatiranja zapisov (I/O). Glavna prednost BASICA je njegova poenostavljenost in interaktivna usmerjenost. Zato navedenih pomanikljivosti je Pascal brez dodatkov glede na omenjene jezike manj primeren za nekatere poslovne aplikacije in za manj vešče uporabnike.

Prvi jezik, ki je bil narejen kot močnejši brat sorodnega jezika, pa je doživel dokaj žalostno usodo, je bil ALGOL-68. Danes se malo uporablja, čeprav se pojavljajo znaki oživljanja. Ta jezik je nastal kot izpeljanka ALGOLa-60 tako, da je bil bistveno sposobnejši in svobodnejši v izražanju. To pa je vodilo v zelo obsežne in počasne prevajalnike, v nepreglednost in težko čitljivost programov. Odkrivanje napak je bilo otežkočeno in produktivnost je padla.

PL/1 je nastal kot naslednik FORTRANa, COBOLa in ALGOLa 60 tako, da je pobral večino konstruktov iz teh jezikov in še kaj. Je močnejši od Pascala. Računalniški strokovnjaki ga v precejšnji meri ocenjujejo kot stranpot. Težko je v nekaj besedah povedati, kje so snovalci tega jezika pogrešili. Morda je razlog preobširnost jezika, ki se kaže kot obsežna množica med seboj dostikrat tujih konstruktov in prevelika svoboda izražanja. Mogoče so pogrešili implementatorji jezika, ki so napisali zelo obsežen in pogosto neučinkovit prevajalnik. Kljub vsemu se PL/1 precej uporablja.

Podobno situacijo srečujemo tudi pri ADI. ADA ima relativno malo popolnoma novih konceptov glede na ostale algoritmične jezike. Predvsem MODULa-2 ji je močno podobna, sem na lahko prištejemo še USCD Pascal za mikroračunalnike, Pascal PLUS za diskretno simulacijo in Concurrent Pascal za aplikacije v realnem času. Te verzije Pascala so usmerjene na podobna področja kot ADA, standardni Pascal pa lahko obogatimo s knjižnjicami podprogramov v zbirnem jeziku. ADA je vseeno močnejša od omenjenih jezikov, vendar je ta njena moč tudi dvorezen meč, saj je v tako močnih jezikih pogosto neprijetno programirati in se jih je težja naučiti. Poleg tega je prevajalnik vedno obsežen in ga je težko implementirati. Debate o ADI so dokaj ostre in deljene, pravo sodbo pa bo dal šele čas. Pascal je po svojih sposobnostih podmnožica ADE. Podobna razmerja smo srečali npr. že v odnosu COBOL - PL/1, vendar nas zgodovina uči, da močnejši jezik redkokdaj zasenči svoje predhodnike /3,15,20/, še zlasti, kadar nima veliko novih konceptov. Primerna podmnožica ADE bi bila podobna MODULI-2 in bi prav gotovo pomenila korak naprej.

3. Zgodovina in kratek opis Pascala

Pascal je algoritmični programski jezik algolskega tipa. Med neposredne predhodnike Pascala lahko štejemo Algol 60 in Algol W. Pascal je v celoti razvil prof. N. Wirth, tako idejno kot implementacijsko. Prvi prevajalnik je nastal leta 1969 /21/. Izšlo je že nekaj predlogov za standarizacijo jezika, vendar brez končnega rezultata /22/. Naštejmo nekaj dobrih lastnosti Pascala /15,23/:

- majhen in prenosljiv prevajalnik, zato se lahko uporablja na mini in zmogljivejših mikro računalnikih
- dobra, enostavna in razumljiva literatura o Pascalu
- majhen in udoben jezik z malo rezerviranimi besedami, malim številom sintaktičnih in semantičnih pravil in z malo izjemami, zatorej čista in učinkovita krmilna struktura, ki daje občutek zanesljivosti, uporabnik se lahko nauči in obdrži v spominu vse značilnosti jezika
- dobre metode strukturiranja podatkov
- dovolj močni kontrolni in podatkovni konstrukti, da omogočajo udobno in nitro programiranje
- omogoča dobro preglednost in razne stile programiranja (npr. s strukturiranjem ali brez, itd.)

Nameni načrtovalcev Pascala /15,23/:

- velika učinkovitost naj bi bila dosežena s čimveč kontrolami v času prevajanja in samo nujno potrebnimi v času izvajanja
- Pascal naj bi omogočal sistematično učenje programiranja
- dokazali bi radi, da se da jezik z bogatimi kontrolnimi strukturami in fleksibilnimi podatkovnimi tipi implementirati z učinkovitim in majhnim prevajalnikom
- tako jezik kot prevajalnik naj bi bila lahko čitljiva, učinkovita in zanesljiva
- Pascal naj bi bil splošno namenski jezik, torej naj bi omogočal udobno programiranje in aplikacije na čimveč področjih.

4. Zaključna ocena

Pascal (to velja tudi za njemu sorodne jezike, npr. za MODULO-2 ali nekatere inačice ADE) je po svojih lastnostih verjetno eden najboljših predstavnikov algoritmičnih splošno namenskih jezikov. Nealgoritmični jeziki so težje primerljivi s Pascalom, vendar se tudi tu Pascal dokaj dobro odreže. Skoraj na vsakem ožjem področju lahko najdemo jezike, ki so boljši kot Pascal in vsaka značilna skupina jezikov ima svoje načine izražanja, ki so v Pascalu neprijetni. Kljub temu pa lahko velik del za določena problemska okolja značilnih nalog v Pascalu rešimo na skoraj enakovreden način, pa naj primerjamo COBOL pri poslovnih aplikacijah ali SNOBOL pri procesiranju tekstov. Poleg tega je učinkovitost novejših Pascalskih prevajalnikov tako glede dolžine generirane kode programa kot hitrosti izvajanja kode taka kot pri novejših Fortranskih prevajalnikih /24/, torej običajno precej boljša kot pri BASICu, COBOLu ali PL/1 in še bistveno boljša kot pri LISPu, SNOBOLu ali PROLOGu. Velika slabost Pascala ostaja slabo programersko okolje. Ravno to pa je področje, kjer lahko v naslednjih letih pričakujemo največje korake naprej. 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 razvijanje programov, kot pa jeziki brez teh lastnosti. Primerjajmo recimo udobnost PROLOGovega debuggerja ali LISPovo programsko okolje ali BASICovo interaktivnost in videli bomo, da ima tu Pascal še veliko neizkoriščenih možnosti. Daljna bodočnost pa bo verjetno prinesla kaj novega, verjetno nekaj izven skupine algoritmičnih jezikov. Mogoče bo ta jezik v marsičem podoben PROLOGu, temeljnemu jeziku japonske pete generacije računalnikov.

5. Literatura

1. A.I.Wasserman,S.Gutz : The Future of Programming, CACM, Vol. 25, Num. 3, str. 196 - 206, maj 1982
2. R.Reinhardt,V.Rajković,I.Lajovic,J.Vrečko : Kriteriji za izbiro programirnega jezika za pouk računalništva na srednji šoli, simpozij INFORMATICA 77, 7 107, 5 str., Bled, 1977
3. C.A.R.Hoare : The Emperor's Old Clothes, CACM, Vol. 24, Num. 2, str. 75 - 83, februar 1981
4. Editorial, ACM SIGPLAN Notices, Vol. 19, Num. 9, september 1982
5. On APL and Productivity, CACM Forum, CACM, Vol. 24, Num. 7, str. 478 - 479, julij 1981
6. J.McCarthy, etc.: LISP 1.5 Programmer's Manual, The M.I.T. Press, 1962
7. R.Herman : The Function of Faster Programming, New Scientist 25, str. 512 - 515, november 1982
8. I.Bratko,M.Gams : PROLOG : osnove in principi strukturiranja podatkov, Informatica 4, str. 40 - 47, 1980
9. R.Kowalski : Algorithm = Logic + Control, CACM, Vol. 22, Num. 7, str. 572 - 595, 1797
10. D.A.Waterman,F.Hayes-Roth : An Overview of Pattern-Directed Inference Systems, Academic Press, 1978
11. M.Gams : Pomen in vloga znanja v sistemih za interakcijo z uporabnikom, magisterska delo, junij 1982
12. W.D.Maurer : The Programmer's Introduction to SNOBOL, American Elsevier Publish. Comp., 1976
13. B.J.MacLennan : Values and Objects in Programming Languages, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 70 - 80, december 1982
14. T.Rentsch : Object Oriented Programming, ACM SIGPLAN Notices , Vol. 17, Num. 9, str. 51 - 58, september 1982
15. R.Cailliau : How to Avoid Getting SCHLONKED by Pascal, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 31 - 41, december 1982
16. 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
17. R.Cailliau : A Letter to Editor, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 10 - 11, december 1982
18. A.P.Železnikar : Programiranje v ADI I, Informatica 3, str. 10 - 23, 1982
19. A.P.Železnikar : Programiranje v ADI II, Informatica 4, str. 19 - 30, 1982
20. H.F.Leddgard,A.Singer : Scaling Down Ada (Or Towards A Standard Ada Subset), CACM, Vol. 25, Num. 2, str. 121 - 125, februar 1982
21. N.Wirth : The Design of a Pascal Compiler, Software Practice and Experience, 1, str. 309 - 333, 1971
22. Second draft proposal ISO/DP 7185 - Specification for the Computer Programming Language - Pascal, Pascal News, num. 20, december 1980
23. K.Jensen,N.Wirth : Pascal,User Manual and Report, Springer Verlag, 1978
24. Benchmark test na Quicksortu, Special Software Limited, Informatica 3, str. 77, 1982
25. M.Gams,I.Bratko,V.Batagelj,R.Reinhardt,M.Martinec,M.Spegel,P.Tancig : Programski jezik Pascal II (podrobnejša analiza pomanjkljivosti Pascala), v pripravi

ALGOL 60 ZA SISTEM CP/M II

A. P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.06 Algol 60: 519.682

Članek opisuje vhodne/izhodne procedure, izbiro vhoda/izhoda, zapiranje in brisanje zbirk, vhodne/izhodne podporne rutine, neposreden vhod/izhod v pomnilnik in iz pomnilnika, naključni zbirčni dostop, knjižnične procedure in vključevanje zunanjih algolskih (knjižničnih) zbirk v algolski tekst (program). Kot programirni primeri so opisani programi za kvadraturno integriranje in izračunavanje vrednosti hermitskih, čebiševskih, laguerrskih in legendrskih polinomov.

A CP/M System Algol 60 Language II

This article deals with input/output procedures, input/output selection, closing and deleting files, input/output support routines, input/output directly to or from memory, random access files, library procedures and library inserts. Several programming examples are presented for quadrature integration and value calculation of Hermitian, Chebishevian, Laguerrian, and Legendrian polynomials.

3.25. Vhodne/izhodne procedure

Za jezik Algol 60 ni bila strogo določena vhodna/izhodna oblika; ta je prepuščena proizvajalcu prevajalnika, bila pa naj bi dovolj vsestranska in zmogljiva. Vhodne/izhodne funkcije so zastopane z vrsto procedur, vgrajenih v sistem časa izvajanja. Vhod/izhod mora biti periferno neodvisen in povezan s številkami podatkovnih tokov oziroma naprav. V primerih nezbirčno strukturiranih naprav (konzola, tiskalnik) je vhodna/izhodna operacija določena z ustrezno številko podatkovnega toka. Te številke so pojasnjene v Dodatku 5. V primeru diskovne zbirke je potreben določen dialog, s katerim odpremo ali oblikujemo imenovano zbirko. Tu se ustrezne tokovne številke priredijo dinamično s sistemom. Vhodne/izhodne procedure bomo opisali v nadaljnih odstavkih, zbrane pa so v Dodatku 2 tega članka ((12)). V diskovne zbirke bomo lahko dostopali zaporedno in naključno.

V vseh preddeklariranih V/I procedurah je prvi parameter tokovna številka, označena z 'dev'. Ime 'val' označuje realno spremenljivko in 'ival' celoštevilsko. Ker je formalne parametre mogoče klicati z vrednostjo, so dejanski parametri lahko tudi izrazi. Pretvorba med realno in celoštevilsko vrednostjo je avtomatična. Imamo:

```
PROCEDURE skip(dev);
```

Izdata se znaka CR in LF v napravo dev.

```
INTEGER PROCEDURE chin(dev);
```

Bere se naslednji znak iz naprave dev. Rezultat te procedure je vrednost znaka. Pri diskovnem vhodu imamo na koncu zbirke znak CTRL-Z.

```
REAL PROCEDURE read(dev);
REAL PROCEDURE read(dev,label);
```

Bere se število s plavajočo vejico ali celo število iz naprave dev. Število ima prost format in se konča z znakom, ki ni del števila. Decimalni eksponent začenja s črko E. Ne upoštevajo se znaka SPACE, TAB in prazne vrstice pred številom, drugi znaki pa povzročijo napako. Presledek končuje število z izjemo med E in eksponentnim poljem. Cela števila se berejo brez zaokraževalnih napak, brez decimalne pike ali eksponentnega dela. Pri branju zbirke z neznano dolžino se lahko uporabi druga (spodnja) oblika: na koncu zbirke se krmiljenje prenese na označitev 'label'. Prevajalnik ve, da je drugi parameter označitev ali označitveni izraz. Na koncu zbirke je navaden omejevalnik in skok se ne izvrši, dokler branje ni opravljeno. Če označitev ni dana v programu, se v času izvajanja pojavi napaka, ko je dosežen konec zbirke. Primeri veljavnih številskih formatov so

```
0.123     +1.23E -3     -123
```

Bralna rutina sprejme tudi formate, ki se nikdar ne pojavijo v izhodnih rutinah:

```
E -3      .123      -123.
```

Lahko se pojavi potreba za branje izvirnih podatkov, ki vsebujejo komentarsko besedilo. Bralna rutina je lahko obveščena, da ne upošteva znaka pred številom. Tu se uporabi poziv

```
ioc(18);
```

Posledica tega poziva je, da se števili

E -6 in .45

bereta kot -6 in 45 (brez predhodnih znakov). Stanje, v katerem komentarji niso dovoljeni, dosežemo s pozivom

ioc(19);

(to je prvotno stanje).

PROCEDURE text(dev,"niz");
Ta procedura izda niz na napravo dev. Tu se lahko uporabi vrsta krmilnih znakov (glej odstavek 3.9). Niz je lahko tudi nizni parameter procedure, ko imamo npr.

```
PROCEDURE sporočilo(s); STRING s;
  BEGIN text(1,s); ...
```

PROCEDURE chout(dev,ival);
Pošlje se en sam znak v napravo dev. Da bi se poslal znak, mora biti uporabljena njegova ASCII vrednost. Pri tem se lahko uporabi tudi

chout(1,&X);

ko se znak 'X' odtisne na terminalu.

PROCEDURE write(dev,ival);
PROCEDURE write(dev,ival,osnova);
Natisne se ival kot celo število na napravi dev. Manjkajoče osnova je decimalna. Nebistveni znaki se ne natisnejo. Za formatirano tiskanje se lahko uporabi funkcija rwrite. Tiskanje v oktalni in heksadecimalni obliki je mogoče z uporabo tretjega parametra:

osnova := 0	decimalno
osnova := 1	oktalno
osnova := 2	heksadecimalno

Druge vrednosti osnove povzročijo napako v času izvajanja.

PROCEDURE rwrite(dev,val,a,b);
PROCEDURE rwrite(dev,val);
Na izhod dev se izda vrednost val v plavajoči vejici v tem formatu: 'a' je celotno število znakov s predznakom in decimalno vejico, 'b' pa je število znakov za decimalno vejico. Pri a = 0 se uporabi eksponenčni format z b decimalnimi številkami. Če je b = 0, imamo formatirani celoštevilski izhod. Če sta a = 0 in b = 0, ali če sta izpuščena, imamo navadni 6-mestni eksponenčni format.

Vobče imamo tri skupine ioc pozivov. Prva skupina povzroči operacije pri prevelikih vrednostih izhoda. Imamo tole:

ioc(6);
Ta rutina poskuša prilagoditi število tako, da pomika decimalno vejico. Če to ni mogoče, uporabi eksponenčni format, ko dolžina polja to dopušča, sicer pa izda vrstico zvezdic "****", ki označuje preseganje območja števila.

ioc(7);
Formatne spremembe niso dovoljene. Če števila ni mogoče prilagoditi, se natisne vrstica zvezdic.

ioc(8);
Tiskanje napake ni dovoljeno. Pri izvajanju tega poziva se vrstica zvezdic nikdar ne natisne. Formatne spremembe so dovoljene; če je potrebno, se uporabi eksponenčni format neglede na obseg polja.

Druga skupina ioc pozivov je povezana s predstavitvijo presledkov v okviru izhodnega formata:

ioc(9);
Ta poziv postavi "manjkajoči znak presledka" na ničlo (ASCII 0). Vodeče ničle se zadušijo, število je levo poravnano. (Ničelni znak je le past za rutino in se ne pošlje v izhodni tok.)

Tretja skupina ioc pozivov ureja predstavitev pozitivnih števil:

ioc(11);
Uporabi se trenutni manjkajoči znak presledka (glej prejšnjo skupino ioc pozivov), ko se pričakuje pozitivni predznak. V začetku je manjkajoči znak presledka presledek. Če se ioc(11); uporabi za pozivom ioc(10);, je rezultat zadušitev znakovne sledi, ki je bila rezervirana za označitev pozitivnega rezultata.

ioc(12);
Za označitev pozitivnega števila se natisne znak '+'.

Opomba: Poziva rwrite in write se končata s tiskanjem "manjkajočega znaka presledka" (glej prejšnjo skupino 2 ioc pozivov). Ta znak je v začetku postavljen kot presledek in rabi kot terminator za ločevanje izhoda, tako da ga je mogoče ponovno brati z bralno rutino.

3.26. Vhodna/izhodna izbira

RML Algol omogoča uporabniku izbiro vhodnih/izhodnih zbirk in naprav iz programov ali s konzole tastature. Obstaja vmesnik, v katerega se vstavi V/I izbira z uporabo tokovne številke 7 ali z ioc pozivi. Zaporedje dogodkov je takole sestavljeno:

1. Vstavi niz V/I izbire v vmesnik.
2. Pokliči interpret ukaznega niza za branje vsebine vmesnika in kopiraj niz ustrezno v "vhodno listo" in/ali v "izhodno listo".
3. Poziv preddeklarirane procedure "vhod" ali "izhod" prebere naslednji vstop v "vhodno listo" ali "izhodno listo" in vrne programu ustrezno tokovno številko, ko je odprl ali oblikoval potrebne zbirke.

Vhod iz toka 7 se shrani v vmesnik in je za program dosegljiv, ko se je pojavil znak pomika valja. Nepravilni znaki se lahko zbrišejo z uporabo posebne tipke (rubout). S tokom 7 sta povezana dva kazalca, eden za vhod in drugi za izhod. Pri vmesniškem branju ali pisanju znakov se vrednost ustreznega kazalca povečuje za 1. Ta kazalca se lahko zbrišeta (resetirata) z uporabo teh pozivov:

ioc(0);
Zbriše se vsebina vhodnega kazalca. Poziv chin(7) bo sedaj vrnil vrednost prvega znaka v vmesniku.

ioc(1);
Zbriše se vsebina izhodnega kazalca in vpiše se nizni terminator na prvo pozicijo vmesnika. Poziv chout(7,znak); bo postavil vrednost znaka na prvo pozicijo vmesnika in pomaknil nizni terminator za eno mesto naprej. Pri uporabi naslednjih pozivov ioc(2); ioc(3); ioc(4); in ioc(5); naj bi programer pred branjem iz toka 7 uporabil ioc(0); in ioc(1); za brisanje vsebin kazalcev.

Naslednji ioc pozivi omogočajo vstop vhodnih in izhodnih list:

ioc(2);
Ta poziv povzroči na konzoli sporočilo OUT =

IN? Uporabnik vstavi ukazni niz splošne oblike

outputlist = inputlist(cr)

Ko se pojavi znak 'cr', ki končuje ukazno vrstico, se pokliče interpret ukaznega niza. Vsak znak do enačaja (ali do znaka 'cr', če enačaja ni) se kopira in shrani kot trenutna "izhodna lista", vse za enačajem pa kot tenutna "vhodna lista". Po en kazalec imamo za vsako od teh list in ko se pojavi nova vhodna ali izhodna lista, se kazalec postavi na začetek liste.

ioc(3);
Ta poziv je podoben ioc(2);, le da se besedilo vzame kot vsebina iz vmesnika brez sporočila uporabniku. Značilno pozivno zaporedje za oblikovanje vhodne/izhodne liste je lahko

```
ioc(1);text(7,"outputlist=inputlist");ioc(3);
```

ioc(4);
Ta poziv sporoči na konzolo

INPUT =

Uporabnik vstavi ukazni niz splošne oblike

inputlist(cr)

Ta niz postane tako trenutna "vhodna lista", izhod pa je ostal nespremenjen.

ioc(5);
Ta poziv je podoben ioc(4);, le da se besedilo vzame kot vsebina iz vmesnika brez sporočila na konzolo. Značilno pozivno zaporedje je lahko

```
ioc(1); text(7,"inputlist"); ioc(5);
```

Opomba: Poziv ioc(3); ali ioc(5); pusti vsebino vmesnika nespremenjeno. Isti niz se tako lahko analizira dvakrat pri enakih imenih vhodnih in izhodnih zbirk. To operacijo opravi prevajalnik pri izbiranju vhoda in izhoda.

Splošna oblika vhodnih in izhodnih list je sestavljena iz zaporedja ene ali več naprav ali zbirčnih specifikacij, ločenih z vejicami. Npr.

```
CON:, A:OUT1, , LST:=DATA B , RDR:
```

V tem primeru smo specificirali 4 izhodne kanale in 2 vhodna kanala. Poziv vhoda ali izhoda predddeklariranih procedur bo poiskal ustrezno listo od trenutnega položaja do naslednje pojavitve vejice ali konca liste; vrnjena bo številka toka, ki ustreza najdenemu vstopu.

Seznam mnemonike naprav s pripadajočimi številkami tokov je dan v Dodatku 5.

Splošna oblika specifikacije CP/M zbirke je

DISK: IMEZBIRKE.RAZŠIRITEV

V imenu zbirke so lahko črke, številke, znaka '$' in '?'; vprašaj naj bi bil rezerviran za specifikacijo dvoumnih zbirčnih imen. Male črke se pretvorijo v velike in vsi zanki, manjši od presledka, se v V/I listah ne upoštevajo.

IMEZBIRKE je sestavljeno iz enega do osem znakov. Zbirčna RAZŠIRITEV ima 1 do 3 znake. Kadar se razširitev ne navede, se upošteva njena manjkajoča vrednost (npr. pri imenu quad se vzame quad.alg); na začetku ima razširitev 3 presledke. Lahko pa zahtevamo obvezno navajanje manjkajoče razširitve z uporabo poziva

ioc(20);

Prvotno stanje (nenavajanje razširitve) dosežemo s pozivom

ioc(21);

Za DISK: lahko uporabimo A:, B:, C:, D: ali pa diskovne enote ne navedemo (trenutno aktivna enota).

Uporabijo se lahko tudi stikalne opcije za V/I naprave in zbirke; te opcije so zaporedje do 12 znakov v oglatih oklepajih. Npr. pri vhodni zbirki DATA.DAT[B] povzroči stikalo [B] odprtje zbirke za naključni dostop. Vrsta stikal se lahko uporabi tudi v programih, saj obstaja možnost branja stikalne liste iz programa.

Pojav dveh zaporednih vejic v V/I listi pomeni specifikacijo "ničte" V/I naprave NL: (tok 0).

Z opisanimi ioc pozivi se oblikujejo V/I liste, ki jih lahko uporabimo za prirejanje zbirk ali naprav skozi vhode in izhode predddeklariranih procedur:

dev:=input;
S tem stavkom se prebere naslednji vstop iz vhodne liste. Ko je vstop najden in je naprava, se napravi priredi vrednost, ki ustreza imenu naprave. Če je bilo specificirano ime diskovne zbirke, se ta zbirka odpre in dodeli se vmesniško območje za zbirčni krmilni blok in za sektorski vmesnik (pri zaporednem dostopu). Vrnjena tokovna številka je od 64 navzgor in označuje dejansko vrednost vmesnika, dodeljenega zbirki.

Negativna vrednost za 'dev' označuje napako, tj. ali napačno sintakso ali manjkanje vstopa v vhodni listi ali manjkanje navedenega zbirčnega imena.

dev:=output;
Podobno kot za vhodne velja tudi za izhodne zbirke in naprave. Imamo vrsto opcij glede na akcije, ki se izbirajo z ioc pozivi.

ioc(13);
Ni preizkušanja. Druge zbirke z enakim imenom se oblikujejo.

ioc(14);
Obstoječa zbirka z navedenim imenom se zbriše pred oblikovanjem nove zbirke.

ioc(15);
Če se najde zbirčno ime, ki že obstaja, bo ta poziv vrnil tokovno številko -100. Nova zbirka se ne oblikuje.

3.27. Zapiranje in brisanje zbirk

Ko je uporaba zbirke končana, jo moramo zapreti s pozivom predddeklarirane procedure:

close(dev);
Ta poziv zapre zbirko toka 'dev' predhodnim vhodnim ali izhodnim pozivom. Če 'dev' ni zbirčno ime, se ničesar ne zgodi.

Opomba: Če izhodne zbirke ne zapremo, se njena vsebina zgubi. Tudi vhodne zbirke naj se zapirajo, ker se z zapiranjem sprošča vmesnik in zbirčni krmilni blok.

delete(dev);
Ta poziv zbriše zbirko 'dev' s predhodnim vhodnim ali izhodnim pozivom in sprosti vmesnik in zbirčni krmilni blok.

3.28. Vhodne/izhodne podporne rutine

V času izvajanja programa razpozna sistem še dodatne procedure; prevajalnik vključuje z dodatnimi procedurami besedila iz knjižnice ALIB.ALG . Imamo:

```
rewind(dev);
```
Zaporedna vhodna ali izhodna zbirka, povezana z ´dev´, se (najprej v primeru izhodne zbirke zapre in) previje za branje od začetka.

```
dev:=findinput("niz");
```
Ta poziv odpre zbirko ali napravo, opredeljeno z "niz" za vhod na tok ´dev´. Če je prvi znak v "niz" vprašaj ´?´, se zgodi tole: ostanek niza se tiska na konzolo kot sporočilo uporabniku, da vstavi želeno ime vhodne zbirke ali naprave. Npr.:

```
dev:=findinput("?Izvirna zbirka =");
```

izda na konzolo sporočilo

```
Izvirna zbirka =
```

in uporabnik vstavi želeno ime. Prireditev

```
dev:=findinput("DATA.DAT");
```

odpre zbirko DATA.DAT na trenutno aktivni diskovni enoti.

Vhodna specifikacija je lahko sestavljena iz "vhodne liste" in njen prvi vstop se uporabi za prireditev ´dev´. Uporaba te procedure izniči prejšnje vhodne specifikacije, ki čakajo v vhodni listi.

```
dev:=findoutput("niz");
```
Ta prireditev je podobna prireditvi findinput. Izhodna specifikacija je lahko posplošena na popolno vhodno/izhodno listo, kot je bilo opisano za ioc(2); in ioc(3);.

```
i:=rename;
```
S tem pozivom se zbirka preimenuje. Staro ime zbirke in enote se vzame kot naslednji vstop v "vhodno listo". Novo zbirčno ime se vzame iz naslednjega vstopa v "izhodni listi". Npr. zaporedje

```
ioc(1);
text(7,"PETRA.MUC=B:MICIKA.MIS");
ioc(3);
i:=rename;
```

bo preimenovalo zbirko MICIKA.MIS na enoti B: v PETRA.MUC. Pri izstopu bo

i=-1	pomenilo napako, ko zbirka ni bila najdena ali zaradi slabe sintakse;
i=255	bo pomenilo odgovor sistema CP/M neglede na uspeh ali napako

Kadar zbirčna razširitev ni navedena, se uporablja manjkajoča, če pa je ioc(20); aktivno, je razširitev treba navesti. Kadar imeni nove in stare zbirke sovpadata, velja tole:

ioc(13);	ni preizkušanja
ioc(14);	zbriše se prejšnja (stara) zbirka
ioc(15);	vrne se vrednost -100 (v i)

```
i:=newext(j,"XYZ");
```
Zbirka, ki je povezana s tokom j prek prejšnjega vhodnega ali izhodnega poziva, se zapre, njena zbirčna razširitev pa se spremeni v triznakovni niz, dan z drugim parametrom. Ta niz postane manjkajoča zbirčna razširitev. Npr. zaporedje

```
j := findinput("PETRA.VRT");
i := newext(j,"DOM");
```

preimenuje zbirko PETRA.VRT v PETRA.DOM. Pri tem ni preizkusa, ali taka zbirka že obstaja. Negativna vrednost v ´i´ pomeni napako, 255 pa je pričakovani odgovor.

```
a:=bios(n,bc);
```
Ta procedura povzroči neposreden poziv skozi skočni vektor BIOSa (sistem CP/M), ko je n = vstop v skočni tabeli (0 do 14), bc = vsebina registra BC ob vstopu in a= vsebina v registru A ob izstopu (poglej v priročnik CP/M System Alternation Guide).

```
a:=cpm(c,de);
```
Ta procedura povzroči neposreden poziv v sistem CP/M, ko je

c = vsebina registra C ob vstopu (0 do 27)
de = vsebina registra DE ob vstopu
a = rezultat v registru A ob izstopu

Podrobnosti so navedene v priročniku CP/M Interface Guide.

```
i:=fcblock(dev);
```
V ´i´ se vrne naslov zbirčnega krmilnega bloka, povezanega z zbirčnim tokom ´dev´. To je lahko uporabljivo pri neposredni manipulaciji CP/M lastnosti.

```
i:=exflt(a,t);
```
Razširi se lista zbirčnega krmilnega bloka. Sistem RML Algol dovoljuje v začetku odprtje in dostop v 4 zaporedne in v 2 naključni zbirki. Če uporabnik potrebuje večje število zbirk, se s to proceduro razširi lista zbirčnih krmilnih blokov. Vsak poziv razširi dolžino liste za 1. Negativna vrednost v ´i´ pomeni, da je bila presežena maksimalna dolžina 16 vstopov. Parametri rutine exflt so:

a = naslov uporabljenega vmesnika
t = zbirčni tip

Če je t = 0, imamo zbirko z zaporednim, sicer z naključnim dostopom. Uporabljeni vmesniki so uporabnjiško deklarirana polja, njihove naslove pa dobimo z uporabo

```
BEGIN BYTE ARRAY buf[0:160];
  i := exflt(location(buf[0]),0);
```

Potrebni obsegi vmesnikov za serijske zbirke so 161 zlogov (33 za zbirčni krmilni blok in 128 za sektorski vmesnik), za naključne pa 33 zlogov. Polje mora biti dovolj obsežno, da vsebuje vmesnike in da se ti medseboj ne prekrivajo.

3.29. Neposreden V/I iz ali v pomnilnik

Branje in pisanje je mogoče tudi neposredno iz pomnilnika in v pomnilnik. Takšen V/I je povezan s tokovno številko 10. Pri tem je potrebna nastavitev kazalcev, ki se ustrezno povečujejo (inkrementirajo). Procedure, ki manipulirajo s temi kazalci, so v zbirki ALIB.ALG .

```
seti(a);
```
Vhodni kazalec se nastavi na naslov a.

```
seto(a);
```
Izhodni kazalec se nastavi na naslov a.

```
i:=ipoint;
```
V ´i´ se vrne trenutni naslov vhodnega kazalca.

```
i:=opoint;
```
V ´i´ se vrne trenutni naslov izhodnega ka-

zalca.

Tipično zaporedje je:

```
BEGIN BYTE ARRAY buf[0:1000];
  seto(location(buf[0]));
  seti(location(buf[0]));
  rwrite(10,x,0,6); ...
  i := opoint; ...
  x := read(10);
```

Takšen V/I mora ostati v mejah deklariranega polja.

i:=swlist;
V ´i´ se vrne naslov preklopniške (switch) liste. Uporabnik lahko preveri, ali je bila specificirana preklopniška opcija za vhodnim ali izhodnim pozivom, tako da prebere vsebino preklopniške liste. Ti preklopniki (največ 12 znakov) se lahko preberejo z uporabo vhodnega toka 10. Značilno zaporedje je

```
seti(swlist);
i:=chin(10);
```

Prvo stikalo je ´1´. Lista je zaključena z ničto vrednostjo. Preklopniška lista vsebuje vselej informacijo o najbolj pogostnem pozivu V/I procedur.

Manjkajoča razširitev se shrani v 3 zloge, ki sledijo preklopniški listi. Trije znaki se vpišejo v ustrezni vmesnik z uporabo izhoda toka 10, in sicer:

```
seto(swlist+13);
text(10,"XYZ");
```

To zaporedje postavi manjkajočo razširitev na vrednost XYZ. Ob vstopu se manjkajoča razširitev postavi na ničlo, tj. na tri presledke.

Ta metoda se lahko uporabi tudi kot način branja majhnih količin podatkov; ta način je tako podoben DATA stavku v jeziku BASIC. Npr.

```
seti(sloc("1.32  99.6  ... "));
FOR i:=1 STEP 1 UNTIL 20 DO
  x[i] :=read(10);
```

Procedura sloc je opisana v okviru knjižničnih procedur.

3.30. Zbirke z naključnim dostopom

Zbirko lahko odpremo za branje prej z naključnim kot z zaporednim dostopom. Zbirka se odpre kot vhodna zbirka s stikalom [B]. Če se zbirka popravlja, imamo stikalo [BM], kjer M označuje modifikacijo. Na ta način lahko odpremo obstoječe zbirke za naključni dostop. Primer vhodne specifikacije je

```
DATA1.DAT[B], DATA2.DAT[BM]
```

Prva zbirka se odpre za branje z naključnim dostopom, druga pa za branje/pisanje.

i:=rblock(dev,a,b,n);
Prebere se ´n´ blokov z diskovne zbirke, povezane s tokom ´dev´, začenši pri bločni številki ´b´, ko se vsebina vpiše v pomnilnik pri naslovu ´a´. Dolžina prenosa je 128*n zlogov. Prvi blok zbirke ima številko 0. Naslov je vobče odvisen od dela polja, oblikovanega s pomočjo procedure location:

```
i:=rblock(dev,location(buf[0],c,)0);
```

Pri izstopu bomo imeli za ´i´ tele vrednosti:

i=0	uspešno branje
i=1	branje po koncu zbirke
i=2	branje nenapisanih podatkov
i=3	aparaturna napaka

Deklarirano polje mora biti dovolj veliko za sprejetje prenosa. Prenos za konec zbirke bo izničen.

i:=wblock(dev,a,b,n);
Zapiše se ´n´ blokov na disk. Parametri so enaki kot pri rblock. Pri izstopu ima ´i´ tele vrednosti:

i=0	uspešen zapis
i=1	napaka v razširitvi zbirke
i=2	konec diskovne zbirke
i=3	aparaturna napaka
i=255	imeniški prostor je izčrpan

3.3.1. Knjižnične procedure

Opisane procedure so vgrajene v sistem časa izvajanja in so znane prevajalniku ob prisotnosti zbirke ALIB.ALG . Imamo:

i:=location(x);
Ta rutina vrne v ´i´ naslov spremenljivke x. x je lahko realno, celoštevilsko ali indeksno tipa realno, celoštevilsko ali zlogovno. V primeru realnega ali celoštevilskega argumenta se vrne naslov prostora, predvidenega za to spremenljivko v času izvajanja. Vsak tak prostor obsega 4 zloge in je v primeru celoštevilske spremenljivke enak le zgornji polovici. Pri elementih polja kot argumentih se vselej vrne pravilen naslov. Procedura izda vselej izračunani naslov spremenljivke; spremenljivka je klicana z vrednostjo in prevajalnik sprejme tudi izraz kot dejanski parameter. Podobno lahko zgradimo tudi proceduro za lociranje boolovske spremenljivke, le da pri enakem telesu kličemo parameter boolovskega tipa z vrednostjo.

i:=fspace;
Procedura vrne število prostih zlogov (npr. pri kontroli sklada). Na velikih sistemih je rezultat lahko večji od 32k zlogov in se tako pojavi negativna vrednost (dvojiški komplement).

blmove(s,f,len);
Opravi se pomik bloka dolžine ´len´ zlogov pri naslovu ´s´ na naslov ´f´. Procedura location se lahko uporabi za ugotovitev naslova, npr.

```
blmove(location(a[0]),location(b[0]),100);
```

Uporabnik sam odgovarja za pomikanje blokov v okviru mej deklariranih polj. Ta procedura dopušča tudi prekrivanje dveh blokov.

i:=peek(a);
Procedura vrne vrednost zloga na naslovu ´a´

poke(a,i);
Procedura nastavi vsebino naslova ´a´ na vrednost ´i´, upoštevaje 8 najvišjih bitov v ´i´.

a:=in(c);
To je vhod iz vrat; ta procedura izvrši ukaz IN A,(C).

out(c,a);
To je izhod na vrata; ta procedura izvrši ukaz OUT (C),A.

b:=parity(i);
Ta boolovska procedura vrne TRUE, če je vre-

dnost parnosti znaka za ´i´ (8 najvišjih bitov) soda (parna), sicer pa vrednost FALSE.

Pomiki in zasuki

V naslednjih procedurah bo ´v´ vrednost (tipa INTEGER) in ´n´ število mest za pomik ali zasuk. Vselej se uporabijo samo 4 zgornji biti za ´n´, tako da je vrednost v območju (0,15). Imamo:

```
i:=shl(v,n);    pomik v levo

i:=lsr(v,n);    logični pomik v desno

i:=asr(v,n);    aritmetični pomik v desno

i:=rotr(v,n);   zasuk v desno
```

Aritmetični pomik v desno razširi bit predznaka, logični pomik v desno pa vstavi ničle.

```
x:=random;
```
Procedura vrne psevdonakljucno število v intervalu (0,1).

```
clarr(a,len);
```
Zbriši območje polja dolžine ´len´ zlogov, začenši pri naslovu ´a´.

```
dpb(u,t,s,a);
```
Nastavi diskovne parametre. ´u´ je številka enote (0 do 3), ´t´ je steza, ´s´ sektor in ´a´ DMA naslov.

```
i:=rdisk;
```
Prebere disk neposredno z uporabo pred tem nastavljenih parametrov v ´dpb´. Rezultat CP/M poziva bo v ´i´.

```
i:=wdisk;
```
Vpiše na disk neposredno z uporabo pred tem nastavljenih parametrov v ´dpb´. Rezultat CP/M poziva bo v ´i´.

```
i:=sloc("niz");
```
V ´i´ se vrne naslov začetka niza. Dejanski parameter je lahko tudi nizni parameter procedure. Npr.:

```
PROCEDURE x(s); STRING s;
BEGIN INTEGER i;
   i:=sloc(s);
   i:=sloc("XYZ");
```

Nizi so sestavljeni iz zaporedja znakov, shranjenih kot zaporedje zlogov in zaključenih z ničlo.

```
atext(dev,s);
```
Ta poziv je podoben preddefinirani proceduri text, toda drugi parameter je naslov niza. Npr.:

```
text(dev,"XYZ");   je ekvivalentno
     atext(dev,sloc("XYZ"));
```

```
i:=tlen(s);
```
Vrne se dolžina niza, katerega naslov je ´s´. Npr.:

```
i:=tlen(sloc("XYZ"));
```

vrne vrednost 3.

```
i:=smatch(long,short);
```
Ta procedura primerja dva niza in išče prvo ujemanje kratkega niza short v dolgem nizu long. Parametra sta naslova obeh nizov. Če se pojavi ujemanje, se ´i´ nastavi na naslov v dolgem nizu, kjer se ujemanje začne. Če ujemanja ni, se ´i´ nastavi na vrednost 0. Lahko nastopi tudi več ujemanj, če se povečuje naslov od trenutnega ujemanja.

3.32. Knjižnične vključitve

Obstaja mehanizem, s katerim je mogoče vključiti "knjižnične" izvirne zbirke v programsko telo, in sicer v času prevajanja. Npr.

```
LIBRARY "B:ALIB.ALG";
```

Učinek tega poziva je, da prevajalnik poišče zbirko, navedeno v narekovajih. Ta zbirka se odpre in njena vsebina se vključi v izvirni program v točki poziva. V našem primeru se zbirka ALIB.ALG na enoti B: prebere v programsko besedilo. Ta poziv omogoča uporabo procedurnih knjižnic brez večkratnega pisanja procedur v programska telesa. Če zbirčne razširitve ne navedemo, se vzame razširitev ´.ALG´ .

Oglejmo si primer:

```
BEGIN INTEGER i,j,k;
  LIBRARY "ALIB";
  LIBRARY "IOLIB";
  LIBRARY "STATLIB";
  PROCEDURE abc; ...
```

V tem primeru se v tekst vključijo vsebine treh knjižničnih zbirk v času prevajanja. Tudi te zbirke lahko vsebujejo LIBRARY pozive. Omejite je postavljena s številom vhodnih in izhodnih zbirk, ki so lahko istočasno odprte. V RML prevajalniku je ta omejitev 5.

Slovstvo k drugemu delu

((12)) A.P.Železnikar: Algol 60 za sistem CP/M I. Informatica 7(1983), št.4, str. 41 - 54.

((13)) A.P.Eršov, S.S.Lavrova, M.R.Šura-Bura: Algoritmičeskij jazyk ALGOL-60. Mir, Moskva 1965.

((14)) Ja.S.Dymarskij, N.N.Lozinskij, A.T.Makuškin, V.Ja.Rozenberg, V.R.Erglis: Spravočnik programista. Tom pervyj. Gos. sojuz.izd.sudostr.prom., Leningrad 1963 (str. 107 - 117).

((15)) L.A.Ljusternik, O.A.Červonenkis, A.R. Janpol´skij: Matematičeskij analiz: Vyčislenie elementarnih funkcij. Fizmatgiz, Moskva 1963.

((16)) E.D.But: Čislennye metody. Fizmatgiz, Moskva 1959.

((17)) S.M.Nikol´skij: Kvadraturnye formuly. Fizmatgiz, Moskva 1958.

((18)) V.L.Zaguskin: Spravočnik po čislennym metodam rešenija uravnenij. Fizmatgiz, Moskva 1960.

DODATEK 4. PRIMERI KVADRATURNEGA INTEGRIRANJA IN IZRAČUNA VREDNOSTI ORTOGONALNIH POLINOMOV

V tem dodatku bomo obravnavali nekaj primerov s področij kvadrature integralov in izračuna vrednosti polinomov.

4/1. Kvadraturno integriranje z Gaussovo metodo

Težišče našega preizkusnega programa na listi 4/1 bo procedura

```
quad(a,b,m,n,p,c,u,f,sum);
```

ki je primerna zlasti za istočasno integriranje več funkcij v enakih območjih (integracijskih mejah) in pri enakih vozliščnih točkah. Integracijski interval (a,b) bo razdeljen na m enakih podintervalov za n-točkovno kvadraturno integriranje. Prameter p procedure quad bo predstavljal število funkcij, ki jih nameravamo integrirati. Nadalje bomo imeli vnaprej dani polji konstant c in u, kjer bo c[k] normiranje uteži in u[k] normiranje abscis pri k = 1, 2, 3, ... , n. Procedura f(t,j) bo dana programsko in bo izračunavala j-to funkcijo argumenta t. V polju sum bomo shranjevali rezultate integracije, in sicer v sum[j] rezultat integracije funkcije f(t,j).

Procedura quad je v listi 4/1 prva in je bila sestavljena z upoštevanjem Gaussove formule

$$S = \sum_{i=0}^{m-1} \sum_{k=1}^{n} c_k f(a+h(i+u_k)) = \sum_{i=1}^{m} \sum_{k=1}^{n} c_k f(a+h(i-1+u_k))$$

kjer je h = (b-a)/m, koeficienti c_k in u_k pa so dani v listi 4/1 za 3-, 8- in 12-točkovno kvadraturno integriranje, sicer pa v literaturi ((14)) z večjo natančnosto in za 1- do 48-točkovno integriranje.

V listi 4/1 je f(t,j) v proceduri quad j-ta podintegralska funkcija, j pa je v intervalu (1,p); tako lahko integriramo v enem izvajanju procedure quad p različnih podintegralskih funkcij. Procedura func(x,j) v listi 4/1 vsebuje 3 funkcije, in sicer x*x, x*(1+x*x) in 1, ki jih bomo integrirali v intervalu (2.0, 5.0).

VALUE specifikacija v procedurah omogoča, da pokličemo proceduro tudi z vrednostnim (številskim) dejanskim parametrom, npr.

```
quad(2.0,5.0,m,8,p,c8, ...);
```

kjer so 2.0, 5.0 in 8 številski parametri. Nadalje si velja zapomniti, da moramo v enoprehodnem prevajalniku za RML Algol procedurni dejanski parameter navesti z

```
REAL PROCEDURE func
```

pri pozivu quad, kot je razvidno iz liste 4/1 v treh primerih.

Nadalje smo v listi 4/1 parametrizirali vse procedure za izpis s spremenljivko dev, ki smo jo fiksirali z

```
dev := 6;
```

tako da se izpisi pošiljajo na tiskalnik (LST:); pri dev := 1; bi dobili izpis na konzolo (CON:).

Lista 4/2 kaže rezultate izvajanja programa iz liste 4/1. Za tri integrale bi dobili idealne vrednosti

$$\int_2^5 x^2 dx = 39, \quad \int_2^5 x(1+x^2) dx = 162{,}75,$$

$$\int_2^5 dx = 3$$

V listi 4/2 imamo tudi seštete elemente polja c, ki so bili v programu zaokroženi. Idealna vrednost vsakokratne vsote je enaka 1.00. V zvezi z Gaussovo kvadraturno formulo so namreč izpolnjeni tile pogoji:

$$u[n+1-k] := 1-u[k] \qquad (1)$$

ko so abscise u[k] simetrično razporejene glede na točko u = 1/2.

$$c[n+1-k] := c[k] \qquad (2)$$

kjer so simetrični koeficienti enaki in

$$\sum_{k=1}^{n} c[k] = 1 \qquad (3)$$

Elementi polj c in u v listi 4/1 očitno izpolnjujejo te tri pogoje.

```
 n   Vsota vseh c[i]
---------------------
 3     9.99999E-01
 8     9.99999E-01
12     9.99000E-01

 n     Integral 1      Integral 2      Integral 3
-------------------------------------------------
 3     3.89998E 01     1.62749E 02     2.99999E 00
 8     3.89998E 01     1.62749E 02     2.99999E 00
12     3.89608E 01     1.62586E 02     2.99699E 00
```

Lista 4/2. Ta lista kaže izvajanje programa z liste 4/1. Najprej se kontrolirajo vrednosti koeficientov s pogojem (3), nato pa so izračunane vrednosti treh določenih integralov, in sicer za 3-, 8- in 12-točkovno integriranje.

→

Lista 4/1. Program na naslednji strani izračunava vrednosti integralov, uporabljajoč Gaussovo metodo za kvadraturno integriranje. Osrednja procedura tega programa je 'quad', ostali del programa je namenjen le preizkusu te procedure. Elementi polj u in c so določeni le z natančnostjo jezika RML Algol in so zaokroženi. Pomen ostalih procedur v programu je neposredno razviden iz programske liste.

```
BEGIN
COMMENT  Poskusno  kvadraturno  integriranje  z
         Gaussovo metodo

-----------------------------------------------
Naslednja procedura je predmet nase  pozornosti
***********************************************;

   PROCEDURE quad(a,b,m,n,p,c,u,f,sum);

      VALUE a,b,n; REAL PROCEDURE f;
      ARRAY sum,u,c;
      REAL a,b; INTEGER m,n,p;

     BEGIN
       REAL h,t,r; INTEGER i,j,k;
       h:=(b-a)/m;
       FOR j:=1 STEP 1 UNTIL p DO
         BEGIN
           sum[j]:=0;r:=a-h;
           FOR i:=1 STEP 1 UNTIL m DO
             BEGIN
               r:=r+h;
               FOR k:=1 STEP 1 UNTIL n DO
                 BEGIN
                   t:=r+u[k]*h;
                   sum[j]:=sum[j]+c[k]*f(t,j)
                 END k
             END i;
           sum[j]:=h*sum[j]
         END j
     END quad

***********************************************;

   REAL PROCEDURE func(x,j);
     VALUE x,j;
     REAL x; INTEGER j;
   BEGIN
     IF j=1 THEN
        func:=x*x
     ELSE
       IF j=2 THEN
         func:=x*(1+x*x)
       ELSE
         func:=1;
   END func
-----------------------------------------------;

   PROCEDURE arg(dev,n);
     VALUE dev,n; INTEGER dev,n;
   BEGIN
     skip(dev);
     IF n<10 THEN text(dev," ");
     write(dev,n); text(dev,"   ");
   END arg
-----------------------------------------------;

   PROCEDURE vred(dev,sum,p);
     VALUE dev,p;
     ARRAY sum; INTEGER dev,p;
   BEGIN
     INTEGER k;
     FOR k:=1 STEP 1 UNTIL p DO
       BEGIN
         rwrite(dev,sum[k]);
         text(dev,"   ");
       END
   END vred
-----------------------------------------------;

   PROCEDURE csum(dev,n,c);
     VALUE dev,n;
     ARRAY c; INTEGER dev,n;
   BEGIN
     INTEGER i; REAL a;
     a:=0;
     FOR i:=1 STEP 1 UNTIL n DO
       a:=a+c[i];
     skip(dev);
     IF n<10 THEN text(dev," ");
     write(dev,n); text(dev,"   ");
     rwrite(dev,a);
   END sum
-----------------------------------------------;

   PROCEDURE glava(dev,p);
     VALUE dev,p; INTEGER dev,p;
     BEGIN
       INTEGER i;
       skip(dev); text(dev," n ");
       FOR i:=1 STEP 1 UNTIL p DO
         BEGIN
           text(dev,"    Integral  ");
           write(dev,i)
         END;
       text(dev,"*N--");
       FOR i:=1 STEP 1 UNTIL p DO
         text(dev,"----------------");
     END glava
-----------------------------------------------;

   INTEGER dev,m,p;

   ARRAY vsota,u3,c3[1:3],u8,c8[1:8],
         u12,c12[1:12];

COMMENT  Glavni preizkusni program

-------  Inicializacija spremenljivk  -------;

   dev:=6; p:=3; m:=10;

   u3[1] :=.112702;        u3[2] :=.5;
   u3[3] :=1-u3[1];
   c3[1] :=.277778;        c3[2]:=.444444;
   c3[3] :=c3[1];

   u8[1] :=.198551e-01;    u8[2] :=.101667;
   u8[3] :=.237238;        u8[4] :=.408283;
   u8[5] :=1-u8[4];        u8[6] :=1-u8[3];
   u8[7] :=1-u8[2];        u8[8] :=1-u8[1];
   c8[1] :=.506143e-01;    c8[2] :=.111191;
   c8[3] :=.156853;        c8[4] :=.181342;
   c8[5] :=c8[4];          c8[6] :=c8[3];
   c8[7] :=c8[2];          c8[8] :=c8[1];

   u12[1] :=.921968e-02;   u12[2] :=.479414e-01;
   u12[3] :=.115049;       u12[4] :=.206341;
   u12[5] :=.316085;       u12[6] :=.437383;
   u12[7] :=1-u12[6];      u12[8] :=1-u12[5];
   u12[9] :=1-u12[4];      u12[10]:=1-u12[3];
   u12[11]:=1-u12[2];      u12[12]:=1-u12[1];
   c12[1] :=.230877e-01;   c12[2] :=.534695e-01;
   c12[3] :=.800392e-01;   c12[4] :=.101584;
   c12[5] :=.116746;       c12[6] :=.124574;
   c12[7] :=c12[6];        c12[8] :=c12[5];
   c12[9] :=c12[4];        c12[10]:=c12[3];
   c12[11]:=c12[2];        c12[12]:=c12[1];

   COMMENT Kontrola vrednosti elementov polj c3,
           c8 in c12;

   text(dev,"*N n   Vsota vseh  c[i]");
   text(dev,"*N---------------------");
   csum(dev,3,c3); csum(dev,8,c8);
   csum(dev,12,c12); skip(dev);

   COMMENT Izracun vrednosti integralov;

   glava(dev,p);
   arg(dev,3);
   quad(2.0,5.0,m,3,p,c3,u3,
                        REAL PROCEDURE func,vsota);
   vred(dev,vsota,3);

   arg(dev,8);
   quad(2.0,5.0,m,8,p,c8,u8,
                        REAL PROCEDURE func,vsota);
   vred(dev,vsota,3);

   arg(dev,12);
   quad(2.0,5.0,m,12,p,c12,u12,
                        REAL PROCEDURE func,vsota);
   vred(dev,vsota,3); skip(dev)

END quad
FINISH
```

4/2. Izračun vrednosti ortogonalnih polinomov

V tem poglavju bomo obravnavali algoritme in algolske programe za izračun hermitskih, čebiševskih, laguerrskih in legendrskih polinimov. Te polinome je mogoče izraziti tudi z rekurzivnimi formulami, kar omogoča njihovo izračunavanje brez uporabe nekaterih standardnih (vgrajenih, predefiniranih) prevajalniških procedur (izjema so le aritmetične operacije). Izračunane vrednosti bomo primerjali tako, da bomo izračune opravili z različnimi formulami. Neposredna kontrola je mogoča tudi z uporabo neposredne polinomske izražave in seveda z uporabo tabel. Tako imamo npr. za polinome Hermita, Čebiševa, Laguerra in Legendra tele neposredne izražave:

$H_0(x)=1$
$H_1(x)=2x$
$H_2(x)=4x^2-2$
$H_3(x)=8x^3-12x$
$H_4(x)=16x^4-48x^2+12$
$H_5(x)=32x^5-160x^3+120x$
$H_6(x)=64x^6-480x^4+720x^2-120$
$H_7(x)=128x^7-1344x^5+3360x^3-1680x$
. . .
$T_0(x)=1$
$T_1(x)=x$
$T_2(x)=2x^2-1$
$T_3(x)=4x^3-3x$
$T_4(x)=8x^4-8x^2+1$
$T_5(x)=16x^5-20x^3+5x$
$T_6(x)=32x^6-48x^4+18x^2-1$
$T_7(x)=64x^7-112x^5+56x^3-7x$
. . .
$L_0(x)=1$
$L_1(x)=-x+1$
$L_2(x)=x^2-4x+2$
$L_3(x)=-x^3+9x^2-18x+6$
$L_4(x)=x^4-16x^3+72x^2-96x+24$
$L_5(x)=-x^5+25x^4-200x^3+600x^2-600x+120$
$L_6(x)=x^6-36x^5+450x^4-2400x^3+5400x^2-4320x+720$
$L_7(x)=-x^7+49x^6-882x^5+7350x^4-29400x^3+52920x^2-35280x+5040$
. . .
$P_0(x)=1$
$P_1(x)=x$
$P_2(x)=1/2(3x^2-1)$
$P_3(x)=1/2(5x^3-3x)$
$P_4(x)=1/8(35x^4-30x^2+3)$
$P_5(x)=1/8(63x^5-70x^3+15x)$
$P_6(x)=1/16(231x^6-315x^4+105x^2-5)$
$P_7(x)=1/16(429x^7-693x^5+315x^3-35x)$
. . .

4/2.1. Hermitski polinomi

Procedura herm(n,x) v listi 4/3 izračunava vrednosti hermitskega polinoma

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

za poljuben realen argument x in za poljubno (celoštevilsko) stopnjo n z rekurzivno formulo

herm(0,x) = 1; herm(1,x) = 2*x;
herm(n+1,x) = 2*x*herm(n,x) - 2*n*herm(n-1,x)

Procedura herm2(n,x) v listi 4/3 izračunava vrednosti hermitskega polinoma

x	n = 1	n = 3	n = 4	n = 6
1.00000E-01	2.00000E-01	-1.19200E 00	1.15216E 01	-1.12848E 02
2.00000E-01	4.00000E-01	-2.33600E 00	1.01056E 01	-9.19639E 01
3.00000E-01	6.00000E-01	-3.38400E 00	7.80960E 00	-5.90414E 01
4.00000E-01	8.00000E-01	-4.28800E 00	4.72960E 00	-1.68259E 01
5.00000E-01	1.00000E 00	-5.00000E 00	1.00000E 00	3.09999E 01

x	n = 1	n = 3	n = 4	n = 6
1.00000E-01	1.00000E-01	-2.99000E-01	2.94010E 00	-1.45515E 01
2.00000E-01	2.00000E-01	-5.92000E-01	2.76160E 00	-1.32239E 01
3.00000E-01	3.00000E-01	-8.73000E-01	2.46810E 00	-1.10708E 01
4.00000E-01	4.00000E-01	-1.13600E 00	2.06560E 00	-8.17991E 00
5.00000E-01	5.00000E-01	-1.37500E 00	1.56250E 00	-4.67188E 00

x	n = 1	n = 3	n = 4	n = 6
1.00000E-01	1.00000E-01	-2.99000E-01	2.94010E 00	-1.45515E 01
2.00000E-01	2.00000E-01	-5.91999E-01	2.76160E 00	-1.32239E 01
3.00000E-01	3.00000E-01	-8.72998E-01	2.46810E 00	-1.10708E 01
4.00000E-01	4.00000E-01	-1.13600E 00	2.06560E 00	-8.17990E 00
5.00000E-01	5.00000E-01	-1.37500E 00	1.56250E 00	-4.67188E 00

Lista 4/4. Ta lista prikazuje rezultate izvajanja programa na listi 4/3; tu so izračunane vrednosti hermitskega polinoma, in sicer za H(n,x) (prva tretjina razpredelnice) in za h(n,x) (druga in tretja tretjina razpredelnice). Druga tretjina razpredelnice je izračun h(n,x) iz H(n,x), tretja tretjina razpredelnice pa izračun h(n,x) z rekurzijo. Razlike med drugo in tretjo tretjino razpredelnice so minimalne; te vrednosti se dobro ujemajo s tabelo v priročniku ((15)).

```
BEGIN
COMMENT Poskusni izracun vrednosti Hermitovega
        polinoma

Naslednja procedura je predmet nase pozornosti
*********************************************;
  REAL PROCEDURE herm(n,x);
    VALUE n,x;
    REAL x; INTEGER n;
  BEGIN
    REAL a,b,c; INTEGER i;
    a:=1; b:=2*x;
    IF n=0 THEN
       c:=a
    ELSE
      IF n=1 THEN
         c:=b
      ELSE
        FOR i:=1 STEP 1 UNTIL n-1 DO
          BEGIN
            c:=2*x*b-2*i*a;
            a:=b; b:=c
          END;
    herm:=c
  END herm
*********************************************

Tudi ta procedura je predmet nase pozornosti
*********************************************;

  REAL PROCEDURE herm1(n,x);
    VALUE n,x;
    REAL x; INTEGER n;
  BEGIN
    REAL a,b;
    a:=1/2^(n/2);
    b:=x/sqrt(2);
    herm1:=a*herm(n,b)
  END herm1
*********************************************

Se ta procedura bo predmet nase pozornosti
*********************************************;

  REAL PROCEDURE herm2(n,x);
    VALUE n,x;
    REAL x; INTEGER n;
  BEGIN
    REAL a,b,c; INTEGER i;
    a:=1; b:=x;
    IF n=0 THEN
       c:=a
    ELSE
      IF n=1 THEN
         c:=b
      ELSE
        FOR i:=1 STEP 1 UNTIL n-1 DO
          BEGIN
            c:=x*b-i*a;
            a:=b; b:=c
          END;
    herm2:=c;
  END herm2
*********************************************;

  PROCEDURE tiskaj(dev,x1,n1,herm);
    VALUE dev; REAL PROCEDURE herm;
    INTEGER ARRAY n1; ARRAY x1;
    INTEGER dev;
  BEGIN
    INTEGER i,j;
    glava(dev);
    FOR i:=1 STEP 1 UNTIL 5 DO
    BEGIN
      skip(dev);
      rwrite(dev,x1[i]);
      FOR j:=1 STEP 1 UNTIL 4 DO
      BEGIN
        text(dev,"    ");
        rwrite(dev,herm(n1[j],x1[i]));
      END
    END
    skip(dev); skip(dev);
  END tiskaj
-----------------------------------------------;

  PROCEDURE glava(dev);
    VALUE dev; INTEGER dev;
    BEGIN
      text(dev,"*N       x                n = 1
       n = 3             n = 4
           n = 6");
      text(dev,"*N ----------------------
----------------------------
----------------------------");
    END glava
===============================================

======== Zacetek preizkusnega programa ========;

  INTEGER ARRAY n1[1:4];
  REAL ARRAY x1[1:5];
  INTEGER dev;

  dev:=6;

COMMENT Inicializacija polj n1 in x1;
  n1[1] := 1; n1[2] := 3; n1[3] := 4;
  n1[4] := 6;
  x1[1] := 0.1; x1[2] := 0.2; x1[3] := 0.3;
  x1[4] := 0.4; x1[5] := 0.5;

  tiskaj(dev,x1,n1,REAL PROCEDURE herm);
  tiskaj(dev,x1,n1,REAL PROCEDURE herm2);
  tiskaj(dev,x1,n1,REAL PROCEDURE herm1)

END
FINISH
```

Lista 4/3. Program na tej listi sestavljajo med drugim tri bistvene procedure, in sicer herm(n,x), herm1(n,x) in herm2(n,x), zaprte med zvezdične vrstce. Proceduri herm(n,x) in herm2(n,x) izračunavata dva tipa hermitskih polinomov z uporabo rekurzivnih formul

$$H(n+1,x) = 2xH(n,x) - 2nH(n-1,x)$$
$$h(n+1,x) = xh(n,x) - nh(n-1,x)$$

tako da je med njima določena odvisnost, in sicer

$$h(n,x) = 2^{**}-(n/2) * H(n,x/\sqrt{2})$$

kjer je '**' operator potenciranja, in '*' operator množenja. Procedura herm1(n,x) izračunava vrednost h(n,x) z uporabo te formule, tako da je mogoča primerjava izračunov v listi 4/4.

Proceduri glava(dev) in tiskaj(dev,x1,n1,herm) sta namenjeni izpisovanju izračunanih podatkov. Procedura tiskaj pokliče proceduro glava iz zunanjega bloka, dočim pokliče proceduro herm parametrično. V glavnem programu se pojavljajo trije klici procedure tiskaj (na koncu programa) z različnimi procedurnimi parametri (herm, herm2, herm1). Za izpis na vrstični tiskalnik imamo prireditev dev := 6, pri dev := 1 pa bi dobili izpis na konzolo.

$$h_n(x) = (-1)^n e^{\frac{x^2}{2}} \frac{d^n}{dx^n} (e^{-\frac{x^2}{2}})$$

za poljuben realen argument x in za poljubno (celoštevilsko) stopnjo n z rekurzivno formulo

herm2(0,x) = 1; herm2(1,x) = x;
herm2(n+1,x) = x*herm2(n,x) - n*herm2(n-1,x)

Med obema tipoma hermitskih polinomov obstaja odvisnost

$$h_n(x) = \frac{1}{2^{\frac{n}{2}}} H_n(x/\sqrt{2})$$

in ta odvisnost je upoštevana s proceduro herml(n,x) v listi 4/3. V listi 4/4 je mogoča primerjava dobljenih rezultatov.

4/2.2. Čebiševski polinomi

V listi 4/7 imamo štiri procedure za izračunavanje vrednosti čebiševskih polinomov, kot je opisano v besedilu liste 4/7. Procedura cheb(n,x) izračunava vrednost čebiševskega polinoma skladno s formulo

T(n,x) = cos(n*arccos x)

ki je realizirana tudi s proceduro chebco(n,x), ki pa uporablja standardne algolske funkcije sqrt(t), arctan(t) in cos(t). V listi 4/6 so vrednosti za obe proceduri zbrane v prvih dveh podrazpredelnicah.

Procedura chebc(n,x) izračunava vrednost čebiševskega polinoma skladno s formulo

C(n,x) = 2*cos(n*arccos x/2)

izračunane vrednosti pa so zbrane v tretji podrazpredelnici liste 4/6. Procedura chebl(n,x) upošteva odvisnost

C(n,x) = 2*T(n,x/2)

tako da je četrto podrazpredelnico liste 4/6 mogoče primerjati s tretjo podrazpredelnico.

Procedura tisk(dev,xl,cheb) natisne posamezne podtabele liste 4/6, pri tem pokliče še proceduro glava(dev) neparametrično. V stavku

```
FOR j:=2,4,5,7 DO
  BEGIN ...
```

so vrednosti za 'n' fiksirane z 2, 4, 5, 7, kot je razvidno tudi iz tabele na listi 4/6. Podatkovni tok 'dev' je fiksiran s številko 6, tako da imamo izpis na tiskalnik, vse izpisne rutine pa so parametrizirane z 'dev'.

Pozivi procedur cheb, chebco, chebc in chebl kot procedurnih parametrov v proceduri tisk imajo obliko

REAL PROCEDURE cheb

itd. zaradi zahteve enoprehodnega prevajalnika.

x	n = 2	n = 4	n = 5	n = 7
1.00000E-01	-9.80000E-01	9.20800E-01	4.80160E-01	-6.45113E-01
2.00000E-01	-9.20000E-01	6.92800E-01	8.45120E-01	-9.87021E-01
3.00000E-01	-8.20000E-01	3.44800E-01	9.98880E-01	-8.46164E-01
6.60000E-01	-1.28801E-01	-9.66821E-01	-4.46187E-01	9.44955E-01
8.80000E-01	5.48799E-01	-3.97641E-01	-7.85732E-01	-9.48303E-01

x	n = 2	n = 4	n = 5	n = 7
1.00000E-01	-9.80000E-01	9.20798E-01	4.80163E-01	-6.45116E-01
2.00000E-01	-9.19998E-01	6.92799E-01	8.45120E-01	-9.87021E-01
3.00000E-01	-8.19999E-01	3.44799E-01	9.98880E-01	-8.46162E-01
6.60000E-01	-1.28800E-01	-9.66820E-01	-4.46187E-01	9.44953E-01
8.80000E-01	5.48800E-01	-3.97638E-01	-7.85730E-01	-9.48303E-01

x	n = 2	n = 4	n = 5	n = 7
1.00000E-01	-1.99000E 00	1.96010E 00	4.95010E-01	-6.86070E-01
2.00000E-01	-1.96000E 00	1.84160E 00	9.60320E-01	-1.29023E 00
3.00000E-01	-1.91000E 00	1.64810E 00	1.36743E 00	-1.73879E 00
6.60000E-01	-1.56440E 00	4.47348E-01	1.98775E 00	-1.41714E 00
8.80000E-01	-1.22560E 00	-4.97903E-01	1.52037E 00	9.51569E-02

x	n = 2	n = 4	n = 5	n = 7
1.00000E-01	-1.99000E 00	1.96010E 00	4.95010E-01	-6.86070E-01
2.00000E-01	-1.96000E 00	1.84160E 00	9.60320E-01	-1.29023E 00
3.00000E-01	-1.91000E 00	1.64810E 00	1.36743E 00	-1.73879E 00
6.60000E-01	-1.56440E 00	4.47348E-01	1.98775E 00	-1.41714E 00
8.80000E-01	-1.22560E 00	-4.97903E-01	1.52037E 00	9.51569E-02

Lista 4/6. Ta lista (razpredelnica) je rezultat štirih izračunov vrednosti in sicer za čebiševske procedure cheb(n,x), chebco(n,x), chebc(n,x) in chebl(n,x), ki so zapisane v programski listi 4/5. Prvi dve podrazpredelnici vsebujeta rezultate izračuna vrednosti čebiševskega polinoma tipa T in sta tako primerljivi; ostali dve podrazpredelnici vsebujeta rezultate izračuna vrednosti čebiševskega polinoma tipa C in sta medseboj primerljivi.

```
BEGIN
COMMENT Poskusni izracun vrednosti Cebisevovega
        polinoma

Naslednja procedura je predmet nase pozornosti
*********************************************;
  REAL PROCEDURE cheb(n,x);
    VALUE n,x;
    INTEGER n; REAL x;
    BEGIN
      REAL a,b,c; INTEGER i;
      a:=1; b:=x;
      IF n=0 THEN
         c:=a
      ELSE
        IF n=1 THEN
          c:=b
        ELSE
          FOR i:=2 STEP 1 UNTIL n DO
            BEGIN
              c:=2*x*b-a;
              a:=b; b:=c
            END;
          cheb:=c
    END cheb
*********************************************

Tudi ta procedura je predmet nase pozornosti
*********************************************;

  REAL PROCEDURE chebc(n,x);
    VALUE n,x;
    INTEGER n; REAL x;
    BEGIN
      REAL a,b,c; INTEGER i;
      a:=2; b:=x;
      IF n=0 THEN
         c:=a
      ELSE
        IF n=1 THEN
          c:=b
        ELSE
          FOR i:=2 STEP 1 UNTIL n DO
            BEGIN
              c:=x*b-a;
              a:=b; b:=c
            END;
          chebc:=c
    END chebc
*********************************************

Tudi ta procedura je predmet nase pozornosti
*********************************************;

  REAL PROCEDURE chebco(n,x);
    VALUE n,x;
    INTEGER n; REAL x;
  BEGIN
    REAL arccos;
    arccos:=arctan(sqrt(1-x*x)/x);
    chebco:=cos(n*arccos);
  END chebco
*********************************************

Tudi ta procedura je predmet nase pozornosti
*********************************************;

  REAL PROCEDURE cheb1(n,x);
    VALUE n,x;
    INTEGER n; REAL x;
  BEGIN
    x:=x/2;
    cheb1:=2*cheb(n,x);
  END cheb1
*********************************************

---------------------------------------------;
  PROCEDURE glava(dev);
    VALUE dev; INTEGER dev;
    BEGIN
      text(dev,"*N       x                n = 2
       n = 4           n = 5
           n = 7");
      text(dev,"*N ---------------------
-------------------------
-------------------------");
    END glava

=============================================;
    PROCEDURE tisk(dev,x1,cheb);
      VALUE dev; REAL PROCEDURE cheb;
      ARRAY x1; INTEGER dev;
    BEGIN
      INTEGER i,j;
      glava(dev);
      FOR i:=1 STEP 1 UNTIL 5 DO
      BEGIN
        skip(dev);
        rwrite(dev,x1[i]);
        FOR j:=2,4,5,7 DO
          BEGIN
            text(dev,"   ");
            rwrite(dev,cheb(j,x1[i]));
          END
      END
      skip(dev); skip(dev);
    END tisk
=============================================

======== Zacetek preizkusnega programa ======;

  REAL ARRAY x1[1:5];
  INTEGER dev;

  dev:=6;

COMMENT Inicializacija polja x1;
  x1[1] := 0.1; x1[2] := 0.2; x1[3] := 0.3;
  x1[4] := 0.66; x1[5] := 0.88;

  tisk(dev,x1,REAL PROCEDURE cheb);
  tisk(dev,x1,REAL PROCEDURE chebco);
  tisk(dev,x1,REAL PROCEDURE chebc);
  tisk(dev,x1,REAL PROCEDURE cheb1);

END
FINISH
```

Lista 4/7. Ta programska lista vsebuje štiri bistvene procedure za izračun vrednosti čebiševskih polinomov, in sicer cheb(n,x), chebc(n,x), chebco(n,x) in cheb1(n,x). Procedura cheb(n,x) izračunava vrednost polinoma s klasično formulo (tip T) oblike

$$\text{cheb}(0,x) = 1;\ \text{cheb}(1,x) = x;\ \text{cheb}(n+1,x) = 2*x*\text{cheb}(n,x) - \text{cheb}(n-1,x)$$

Procedura chebco(n,x) (s katero opravimo primerjalni izračun s cheb(n,x)) uporablja klasično formulo (tip T)

$$\text{chebco}(n,x) = \cos(n*\text{arctg}(\sqrt{1 - x**2}\ /x))$$

Procedura chebc(n,x) vzame za osnovo čebiševsko formulo (tip C)

$$\text{chebc}(0,x) = 2;\ \text{chebc}(1,x) = x;\ \text{chebc}(n+1,x) = x*\text{chebc}(n,x) - \text{chebc}(n-1,x)$$

Procedura cheb1(n,x) upošteva odvisnost med tipoma C in T, ko velja enakost

$$\text{cheb1}(n,x) = 2*\text{cheb}(n,x/2)$$

4/2.3. Laguerrski polinomi

Procedura lag(n,x) v listi 4/8 izračunava vrednosti laguerrskega polinoma

$$L_n(x) = e^x \frac{d^n}{dx^n} (x^n e^{-x})$$

za poljuben realen argument x in za poljubno celoštevilsko stopnjo 'n' z rekurzivno formulo

lag(0,x) = 1; lag(1,x) = 1 - x;
lag(n+1,x) = (2n+1-x)*lag(n,x) -n*n*lag(n-1,x);

Procedura lagn(n,x) izračunava normirane vrednosti (deljene z n!) laguerrskih polinomov, ki jih lahko primerjamo tabelarično ((15)).

Lista 4/8. Ta lista vsebuje tri bistvene procedure, in sicer lag(n,x), lagn(n,x) in fact(n); pomen programa je razumljiv iz liste same.

```
BEGIN
COMMENT Poskusni izracun vrednosti Laguerrovega
        polinoma

Naslednja procedura je predmet nase.pozornosti
*********************************************;
  REAL PROCEDURE lag(n,x);
    VALUE n,x;
    REAL x; INTEGER n;
  BEGIN
    REAL a,b,c; INTEGER i;
    a:=1; b:=1-x;
    IF n=0 THEN
       c:=a
    ELSE
      IF n=1 THEN
        c:=b
      ELSE
        FOR i:=1 STEP 1 UNTIL n-1 DO
          BEGIN
            c:=(1+2*i-x)*b-i^2*a;
            a:=b; b:=c
          END;
    lag:=c
  END lag
*********************************************;

  REAL PROCEDURE lagn(n,x);
    VALUE n,x; INTEGER n; REAL x;
    lagn:=lag(n,x)/fact(n);

  REAL PROCEDURE fact(n);
    VALUE n; INTEGER n;
    fact:=IF n=0 THEN 1 ELSE n*fact(n-1);

  PROCEDURE glava(dev);
    VALUE dev; INTEGER dev;
    BEGIN
      text(dev,"*N       x              n = 2
       n = 3            n = 4
           n = 6");
      text(dev,"*N ----------------------
----------------------------");
    END glava
-----------------------------------------------;

  PROCEDURE tisk(dev,x1,n1,lag);
    VALUE dev; REAL PROCEDURE lag;
    ARRAY x1; INTEGER ARRAY n1; INTEGER dev;
  BEGIN
    INTEGER i,j;
    glava(dev);
    FOR i:=1 STEP 1 UNTIL 5 DO
    BEGIN
      skip(dev);
      rwrite(dev,x1[i]);
      FOR j:=1 STEP 1 UNTIL 4 DO
      BEGIN
        text(dev,"    ");
        rwrite(dev, lag(n1[j],x1[i]));
      END
    END
    skip(dev); skip(dev)
  END tisk
=================================================

======= Zacetek preizkusnega programa ========;

  INTEGER ARRAY n1[1:4];
  REAL ARRAY x1[1:5];
  INTEGER dev;

  dev:=6;

COMMENT Inicializacija polj n1 in x1;
  n1[1] := 2; n1[2] := 3; n1[3] := 4;
  n1[4] := 6;
  x1[1] := 2.5; x1[2] := 3.6; x1[3] := 4.7;
  x1[4] := 5.8; x1[5] := 6.9;

  tisk(dev,x1,n1,REAL PROCEDURE lag);
  tisk(dev,x1,n1,REAL PROCEDURE lagn);

END
FINISH
```

x	n = 2	n = 3	n = 4	n = 6
2.50000E 00	-1.75000E 00	1.62500E 00	2.30625E 01	4.76641E 02
3.60000E 00	5.59998E-01	1.11840E 01	3.29856E 01	-8.30720E 02
4.70000E 00	5.28999E 00	1.63870E 01	-9.91983E 00	-1.67254E 03
5.80000E 00	1.24400E 01	9.24802E 00	-1.00862E 02	7.37729E 01
6.90000E 00	2.20100E 01	-1.82189E 01	-1.99912E 02	4.47171E 03

x	n = 2	n = 3	n = 4	n = 6
2.50000E 00	-8.75000E-01	2.70833E-01	9.60937E-01	6.62001E-01
3.60000E 00	2.79999E-01	1.86400E 00	1.37440E 00	-1.15378E 00
4.70000E 00	2.64500E 00	2.73117E 00	-4.13326E-01	-2.32297E 00
5.80000E 00	6.22000E 00	1.54134E 00	-4.20259E 00	1.02462E-01
6.90000E 00	1.10050E 01	-3.03648E 00	-8.32965E 00	6.21071E 00

Lista 4/9. Tabela vrednosti z laguerrske polinome: prva podrazoredelnica kaže nenormirane vrednosti, druga podrazpredelnica pa normirane vrednosti (deljene z n!).

4/2.4. Legendrski polinomi

Procedura leg(n,x) v listi 4/10 izračunava legendrski polinom

$$P_n(x) = \frac{1}{2^n n!}\,(x^2-1)^n$$

za poljuben realen argument x in za poljubno stopnjo 'n' z uporabo rekurzivne formule.

DODATEK 5

Naslednji razpredelnici opredeljujeta tok ali številko naprave, prirejene V/I kanalu. Te številke kažejo na vstop v liste V/I naprav, kjer se nahajajo naslovi ustreznih programskih perifernih vmesnikov. Prazen prostor je namenjen uporabniku za njegove lastne periferne rutine. V primeru diskovnih zbirk se tokovne številke dodeljujejo dinamično s sistemom (od 64 navzgor). Številke v oklepajih označujejo ekvivalentne tokovne številke za čiste CP/M verzije.

VHODNI TOKOVI

TOK		IME	NAPRAVA
0		NL:	Slepi vhod, ki vrne vselej znak konca zbirke (CTRL-Z)
1	(4)	TI:	Tastaturni vhod za po en znak
2*	(5)	TTY:	Teleprinter
4		CON:	CP/M konzola
5		RDR:	CP/M bralnik
7		TIB:	Tastaturni vhod z vmesnikom, ki se uporablja za V/I zbirčne ukazne vrstice
10		---	Vhod iz pomnilnika (glej seti/seto knjižnični proceduri)
11*	(0)	---	Vhod iz grafičnega zaslona
13		---	Vrne zadnjo številko napake v času izvajanja

IZHODNI TOKOVI

TOK		IME	NAPRAVA
0		NL:	Slep izhod
1	(4)	VT:	Zaslon
2*	(5)	TTY:	Teleprinter
3*	(6)	LP:	Linijski tiskalnik
4		CON:	CP/M konzola
5		PUN:	CP/M luknjalnik
6		LST:	CP/M naprava za listanje
7		---	Izhod v V/I zbirčni ukazni vmesnik
10		---	Izhod v pomnilnik (glej seti/seto knjižnični proceduri)
11*	(0)	---	Izhod na grafičen zaslon

* : velja za posebno implementacijo
Razpoznani diski so: A:, B:, C: in D:

Preklopne možnosti so tele:
[B] Bločni V/I (naključni dostop)
[M] Modificiran dostop (pisanje pri naključnem dostopu)

```
BEGIN
COMMENT Poskusni izracun vrednosti Legendrovega
        polinoma

Naslednja procedura je predmet nase pozornosti
*********************************************;
  REAL PROCEDURE leg(n,x);
    VALUE n,x;
    REAL x; INTEGER n;
  BEGIN
    REAL a,b,c; INTEGER i;
    a:=1; b:=x;
    IF n=0 THEN
       c:=a
    ELSE
      IF n=1 THEN
         c:=b
      ELSE
        FOR i:=1 STEP 1 UNTIL n-1 DO
          BEGIN
            c:=b*x+(i/(i+1))*(b*x-a);
            a:=b; b:=c
          END;
    leg:=c
  END leg
*********************************************;

  PROCEDURE glava(dev);
    VALUE dev; INTEGER dev;
    BEGIN
      text(dev,"*N       x              n = 2
       n = 3          n = 4
           n = 6");
      text(dev,"*N ----------------------
----------------------------
----------------------------");
    END glava;

COMMENT Zacetek preizkusnega programa;

  INTEGER ARRAY ni[1:4];
  REAL ARRAY x1[1:5];
  INTEGER i, j, dev;

  dev:=6;

COMMENT Inicializacija polj ni in x1;
  ni[1] := 2; ni[2] := 3; ni[3] := 4;
  ni[4] := 6;
  x1[1] := 0.0; x1[2] := 0.35; x1[3] := 0.69;
  x1[4] := 0.7; x1[5] := 0.99;

  glava(dev);
  FOR i:=1 STEP 1 UNTIL 5 DO
  BEGIN
    skip(dev);
    rwrite(dev,x1[i]);
    FOR j:=1 STEP 1 UNTIL 4 DO
    BEGIN
      text(dev,"   ");
      rwrite(dev,leg(ni[j],x1[i]));
    END
  END
END
FINISH
```

Lista 4/10. Ta lista kaže med drugim proceduro leg(n,x) za izračun vrednosti legendrskega polinoma z uporabo rekurzije:

$$leg(0,x) = 1;\ leg(1,x) = x;$$
$$leg(n+1,x) = leg(n,x)*x + (n/(n+1))*(leg(n,x)*x - leg(n-1,x));$$

x	n = 2	n = 3	n = 4	n = 6
0.00000E 00	-5.00000E-01	0.00000E 00	3.75000E-01	-3.12500E-01
3.50000E-01	-3.16250E-01	-4.17812E-01	-1.87223E-02	2.22510E-01
6.90000E-01	2.14150E-01	-2.13728E-01	-4.18688E-01	-9.26139E-02
7.00000E-01	2.35000E-01	-1.92500E-01	-4.12062E-01	-1.25286E-01
9.90000E-01	9.70149E-01	9.40746E-01	9.02230E-01	8.00287E-01

Lista 4/11. Tabela kaže vrednosti izračune legendrskih polinomov s programom na listi 4/10.

INFORMATICA 1/1984

ESTIMATION OF THE AVERAGE TRANSFER TIME OF RANDOMLY CHOSEN BLOCKS FROM A DISC CYLINDER

MIROSLAV B. JOCKOVIĆ

INSTITUT FOR NUCLEAR SCIENCES – VINČA

UDK: 681.327.63

The paper deals with a part of the research of efficient use of magnetic discs and contains the analysis of the transfer time of a group of blocks on a single disc cylinder. The optimum strategy of accessing sectors was considered in order to minimize the transfer time of selected blocks. The probability distribution function has been defined which describes the distribution of blocks, 1 sector in length, on a magnetic disc cylinder and a mathematical model for data transfer time has been given. The model has been proven on a computer system. The paper also deals with aspects of the model application.

Rad pretstavlja deo istraživanja efikasnog korišćenja magnetnih diskova i sadrži analizu vremena prenosa grupe blokova sa disk cilindra. Razmatrana je optimalna strategija pristupa sektorima sa ciljem da se minimizira vreme prenosa izabranih blokova. Definisana je funkcija distribucije verovatnoća koja opisuje distribucije blokova dužine 1 sektora na magnetnom disk cilindru i dat je matematički model vremena prenosa. Model je potvrdjen na računarskom sistemu. Rad takodje razmatra aspekte primene modela.

1. INTRODUCTION

The main restriction in online processing is the time it takes to transfer records between store and discs. The delays which result from these transfers can be a very critical factor in system performance as a whole. A number of software and hardware parameters can affect the disc transfer time. This paper is only a part of the total problem and contains the analysis of the transfer time of blocks distributed on a single disc cylinder.

Suppose that K blocks are requested from the cylinder with I tracks and N sectors per track. Each block is one sector in lenght. Let the distribution of requested blocks, shown in Fig. 1, be one of the possible distributions of K = 13 blocks on the cylinder, with N = 15 and I = 20.

I \ N	1	2	3	4	5	6	7 . . .	15
1	1	2	3	4	5			
2		17	18	19	20			
3				34	35	36		
.				49				
.								
.								
20								

Fig. 1. Disc cylinder

Number 1,2,3,4,5,17,18,19,20,34,35,36,49 represent the block sequence numbers on the cylinder. If we define a sector column (denoted by the shaded area) as a set of I sectors located at the same angle with respect to the reference mark, the distribution of blocks per sector columns is 1224310...0. This means that one block is in sector column 1, two blocks in sector column 2, etc. The minimum transfer time can be realised if we first access a sector column with the maximum number of blocks. So, if we access sector column 4 first,we shall traverse 48 sectors in order to transfer 13 blocks. Generally, we wish to investigate the timing aspect of the data transfer and to produce an analytical expression for estimating the average transfer time of a group of randomly distributed blocks held on a disc cylinder.

Little work has been published in the above problem area (for example, see the references 1,2, 3,4), although a great deal of related work has been published on the analysis of disc performance for different file structures. Weingarten /1/ first discussed the techniques of sector scheduling. He simplified the drum model, using conservative approximations, to the extent necessary for the application of well-known results from the theory of queues applied to analogous queueing models. Denning /2/ extended Weingarten's results and studied the significant increases in the performance of both disc and drum systems brought about by scheduling their usage. By scheduling is meant the sequencing of existing demands on the system according to the state of the system and characteristics of input/ output devices. Theory and Pinkerton /3/ improved father the scheduling technics. Fuller /4/ suggested certain considerations in reducing the rotational time. Further investigation in this area are aimed at discovering new aspects for effecient disc utilisation.

2. MATHEMATICAL MODEL

Let $d = (N_1^{C_1}, N_2^{C_2}, \ldots, N_J^{C_J})$ represents a distribution of requested blocks on a disc cylinder with C_1 sector columns each including N_1 blocks, C_2 sector columns with N_2 blocks etc, and finally C_J sector columns, each with N_J blocks, whereby

$$I \geq N_1 > N_2 > \ldots > N_j \geq 1 \qquad (1)$$

and

$$N_1C_1+N_2C_2+\ldots+N_JC_J = K \qquad (2)$$

According to distribution d we can say

$$\bar{T}(N,I,K) = 1 + \frac{N}{2} + \sum_d P_d L_d \qquad (3)$$

In order to read (or write) and transfer the first block, 1+N/2 sectors must be traversed in average. L_d is the number of sectors traversed during the transfer of the K-1 blocks upon the first block has already been transferred from one of the sector columns from group C_1. P_d is the probability distribution function for different distributions of K blocks in N sector columns. Here, we shall derive a probability distribution function which gives correct results in the interval $1 \leq K \leq N \times I$.

First we define the total number of possible methods of selecting K blocks from N sector columns, i.e.,U.

Let $m_1, m_2, \ldots, m_N$ be integers such that

$$m_1+m_2+\ldots+m_N = K, \quad I \geq m_i \geq 0 \qquad (4)$$

The number of ways in which a population of K elements can be divided into N ordered parts (partitioned into N subpopulations and ordered by sector columns) of which the first contains m_1 elements, the second m_2 elements, etc., is

$$\binom{K}{m_1}\binom{K-m_1}{m_2}\binom{K-m_1-m_2}{m_3}\ldots\binom{K-m_1-m_2\cdots-m_{N-2}}{m_{N-1}} \qquad (5)$$

Note that the order of subpopulations is essential in the sense that, for example (m_1 = =3, m_2=5,...) and (m_1=5,m_2=3,...) represent different partitions; however, the order within m_i is not relevant. If we want to introduce the order of the subpopulations, we must multiply the amount (5) by

$$N!/\prod_{\ell=0}^{I} \alpha_\ell!$$

(α_ℓ is the number of elements in the distribution of m_i's which have the value ℓ). Summing all different populations, the expression for U becomes:

$$U = \sum_{\omega}\binom{K}{m_1}\binom{K-m_1}{m_2}\binom{K-m_1-m_2}{m_3}\ldots \binom{K-m_1-m_2-\ldots-m_{N-2}}{m_{N-1}} \frac{N!}{\prod_{\ell=0}^{I} \alpha_\ell!} \qquad (6)$$

where ω is the total number of populations which differ one another by the value at least of one element of m_i (i = 1,...,N).

Eq. (6) can be easily reduced to:

$$U = \sum_{\omega} \frac{K!N!}{m_1!m_2!\ldots m_N! \prod_{\ell=0}^{I} \alpha_\ell!} \qquad (7)$$

The number of ways in which K requested blocks can be distributed among $C_1+C_2+\ldots+C_J$ sector columns represents the favourable cases in the sense of the distribution d. Let denote this number by V. In order to calculate the probality distribution function P_d, given by P_d = = V/U, we must find V. The evaluation of V is as follows.

There are $\binom{N}{C_1}\binom{K}{C_1N_1}\frac{(C_1N_1)!}{(N_1!)^{C_1}}$ (8)

different selection methods of C_1 sector columns with the maximum number of N_1 blocks per sector column, where

$\binom{N}{C_1}$ = different selections of C_1 sector columns from a total of N sector columns,

$\binom{K}{C_1N_1}$=different selections of C_1N_1 blocks from a total of K blocks,

$\frac{(C_1N_1)!}{(N_1!)^{C_1}}$=permutation with repeating C_1N_1 blocks stored in C_1 sector columns, whereby the sequence within each sector column is unimportant from the aspects of the distribution d.

There are $\binom{N-C_1}{C_2}\binom{K-N_1C_1}{N_2C_2}\frac{(C_2N_2)!}{(N_2!)^{C_2}}$ (9)

different methods of selecting C_2 sector columns with N_2 blocks per sector column, etc., and finally, for the J-th group of C_J sector columns with N_J blocks per sector column there are

$$\binom{N-C_1-C_2-\ldots-C_{J-1}}{C_J}\binom{K-N_1C_1-N_2C_2-\ldots-N_{J-1}C_{J-1}}{N_JC_J}\ldots \frac{(N_JC_J)!}{(N_J!)^{C_J}} \qquad (10)$$

different methods of selection.

As the events (i = 1,2,...,J) are simultaneous, the number of favourable cases V is obtained from the product

$$V = \binom{N}{C_1}\binom{K}{N_1C_1}\frac{(N_1C_1)!}{(N_1!)^{C_1}}\binom{N-C_1}{C_2}\binom{K-N_1C_1}{N_2C_2}\frac{(N_2C_2)!}{(N_2!)^{C_2}}\cdots$$

$$\binom{N-C_1-C_2-\cdots-C_{J-1}}{C_J}\binom{K-N_1C_1-N_2C_2-\cdots-N_{J-1}C_{J-1}}{N_JC_J}\cdot\frac{(N_JC_J)!}{(N_J!)^{C_J}} \qquad (11)$$

Eq. (11) can be readily reduced to:

$$V = \frac{K!N(S)}{\prod_{i=1}^{J}(N_i!)^{C_i}.C_i!} \qquad (12)$$

where $N(S)=N(N-1)(N-2)\ldots(N-S+1)$ (13)

and $S = \sum_{i=1}^{J} C_i$ (14)

Finally, the expression for the probability distribution function becomes

$$p_d=\frac{V}{U}=\frac{N(S)}{N!\left(\prod_{i=1}^{J}N_i!^{C_i}.C_i!\right)\sum_{\omega}\frac{1}{m_1!m_2!..m_N!\prod_{\ell=0}^{I}a_\ell}} \qquad (15)$$

Now we need to explain the procedure for getting L_d. Additional N_1-1 revolutions, or $N(N_1-1)$ sector times (one sector time is the fime for reading or writing 1 block) are required in order to transfer N_1-1 blocks from each of the sector columns from group C_1. During this period all blocks from groups of sector columns $C_2,C_3,\ldots,C_J$ will have been read as they have fewer blocks per sector column. However, we need to add $N(1-\frac{1}{C_1})$ sector times for transferring the last blocks from the remaining C_1-1 sector columns. This increment in time due to last blocks in group C_1 is calculated using the mathematical theorem given and proven in APPENDIX . If we denote this increment by M(C), its value is readily obtained by substituting $C = C_1-1$ in M(C). The total time for this distribution, together with the additional 1+N/2 sector times for reading the first block, amounts to

$$L_d=N(N_1-1)+N(1-\frac{1}{C_1})=N(N_1-\frac{1}{C_1}) \qquad (16)$$

There is an insignificant reduction in the transfer time when adjacent sector columns contain the maximum number of blocks.

3. VERIFICATION OF THE MODEL

In order to verify the model an experiment was conducted on an ICL 2903 computer system operating under a manual executive. Processing was carried out on PERI level (ICL's Assembler) without the use of Housekeeping package (sets of subroutines which enable the programmer to access data in direct access files at the record level). The system consisted of a main program in FORTRAN with a PLAN subroutine. The subroutine measured the access and transfer times of the contents of a group of blocks into the operating store for each individual distribution. The main program identified subsets of blocks in sector columns for each distribution, starting and ending blocks, etc.

To provide the handling of experimental data, it is necessary to establish the relation between the measured quantity $t_{iS}(K)$ and the parameters of the experiment N, I and K ($t_{iS}(K)$ is the time required to transfer K blocks from S sector columns in i-th distribution). Firstly, we must determine the number of ways in which K blocks are distributed over S sector columns for an arbitrary distribution i, D_{iS}. Suppose now that we have $n_{i1}+n_{i2}+\ldots+n_{iN} = K$, where n_{ij} means the number of blocks to be transferred from the j-th sector column in i-th distribution. The number of n_{ij}'s different from zero must be equal to S. It comes out that exactly N-S of n_{ij}'s must have zero value. The number of distributions of n_{ij} blocks over I tracks of j-th sector column is $\binom{I}{n_{ij}}$. Accordingly, D_{iS} is given by

$$\binom{I}{n_{i1}}\binom{I}{n_{i2}}\ldots\binom{I}{n_{iN}}.$$

Multiplying D_{iS} by $t_{iS}(K)$ gives the time to transfer all D_{iS} combinations of K blocks from S sector columns in a given distribution i, T_{iS}. The number of distributions i is the number of distinguishable distributions in which none of S sector columns remains empty and amounts to $R_{SK} = \binom{K-1}{S-1}$ /ref. 5/.

Evidently, T_{iS} must be summed over all R_{SK} distributions to obtain T_S. It should be noted that T_S is obtained only for one of $\binom{N}{S}$ possible combinations of sector columns. To include all $\binom{N}{S}$ combinations of sector columns in our statistical computations it is necessary to

multiply T_S by $\binom{N}{S}$ to obtain $T_{N,S}$. Now, $T_{N,S}$ should be summed over all values which S can take. Evidently, the lower value of S is $S_p=\lceil\frac{K}{I}\rceil$, while the upper value is $S_q = N$ for $K/N \geq 1$ and $S_q = K$ for $K/N < 1$. Thus, we obtain $T_{S_p,S_q} = \sum_{S_p}^{S_q} T_{N,S}$. The experimental mean transfer time, $\bar{t}_e(N,I,K)$, is evaluated by dividing T_{S_p,S_q} by the total number of all distributions of K blocks over N sector columns with I tracks, N_{total}. The normalized experimental mean number of traversed sectors during the transfer of K blocks, $\bar{T}_e(N,I,K)$, is obtained by dividing $\bar{t}_e(N,I,K)$ by the time required to transfer one block, t_b. Finally, our relation for statistical computation looks like:

$$\bar{T}_e(N,I,K) = \frac{\sum_{S_p}^{S_q} T_{N,S}}{t_b \cdot N_{total}} = \frac{\sum_{S=S_p}^{S=S_q} \binom{N}{S} \sum_{i=1}^{R_{SK}} \binom{I}{n_{i1}}\binom{I}{n_{i2}}\cdots\binom{I}{n_{iN}} t_{iS}(K)}{t_b \sum_{S=S_p}^{S=S_q} \binom{N}{S} \sum_{i=1}^{R_{SK}} \binom{I}{n_{i1}}\binom{I}{n_{i2}}\cdots\binom{I}{n_{iN}}} \qquad (17)$$

The input to the experiment are N,I and K. All other values, intermediate results and necessary combinations are calculated in the main program. Using this approach, we exclude a great amount of computer processing time, but not reducing the quality of the experiment.

The results shown in Fig. 2 are obtained from: a) the model, Eq.3, ($\bar{T}$), b) the experiment, Eq. 17, ($\bar{T}_e$), and c) the approximation using the fitting polynomial $P_4(x)=a_4 \cdot x^4+a_3x^3+a_2x^2+a_1x+a_o$ (T_f). The coefficients found through fitting are $a_4=-9.69\text{x}10^{-8}$, $a_3=5.0\text{x}10^{-5}$, $a_2=-7.82\text{x}10^{-3}$, $a_1=1,4$ and $a_o=20.47$.

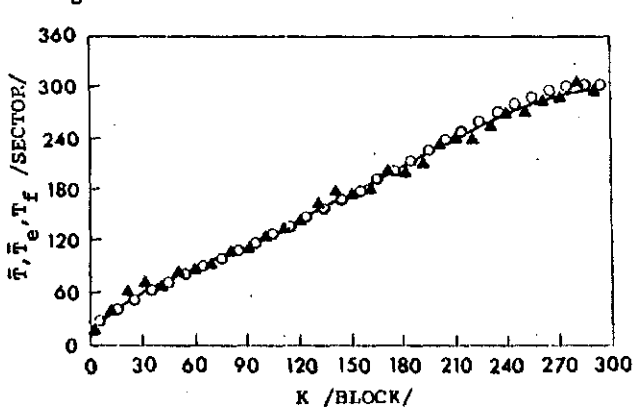


Fig.2. Results; '-' model, 'v' experiment, 'o' approximation

As it can be seen from Fig.2, the results of the analytical and experimental approach are very close. Slight deviations in the results are due to insufficient statistics when timing input operations in relation to the timer resolution and the operating system overhead. Because the effect of the channel queueing has not been considered the experiment was carried out in monoprogramming conditions.

Technical characteristics of ICL exchangeable disc store, model 2815, which are relevant in this paper, are: N=15, I=20 and T_r=25 msec (time for one revolution).

4. APPLICATION ESPECTS OF THE MODEL

In general case, a cylinder may contain multiblock records, many files or their segments, with competing access by many users. The time aspects of the magnetic disc use could be significantly improved by further development of the operating system in the light of model described. The operating system would therefore have to support the queueing lists in front of each sector column of the cylinder, to establish the correspondance between records and the associated blocks and to control the transfer between the system and user buffers. A modification of the SATF (Shortest Access Time First) strategy would be used for accessing the data on a cylinder.

The results presented can also be used when implementing a simulation package for modelling certain situations which can occur during processing of disc files.

5. CONCLUSION

The paper demonstrated that the minimum data transfer time from a magnetic disc cylinder is achieved if the sector column with the maximum number of blocks to be transferred is accessed first. It is assumed that the device remains busy until the end of transfer of the group of K blocks, irrespective from block positions on the cylinder. A correct probability distribution function defining the probability of each random selection of a group of K blocks from a set of NxI sectors is derived. Although the paper considers only the physical data organisation, a suggestion is made to include the model described into the operating system in order to improve the system performance.

APPENDIX

Theorem:

If C of different integers are selected randomly (with no replacement) from a group of integers 1,2,...,N, the expected value of the maximum integer among C numbers selected is

$$M(C) = \left(1 - \frac{1}{1+C}\right)N$$

Let us give one of the possible proofs for this theorem.

In taking C numbers out of N possible numbers, the maximum numbers can be, C, C+1, C+2,..., N-1,N. The remaining C-1 numbers can be drown in the following number of ways

$$\binom{C-1}{C-1}, \binom{C}{C-1}, \binom{C+1}{C-1}, \ldots, \binom{N-2}{C-1}, \binom{N-1}{C-1} \qquad (18)$$

The total number of possible combinations in drawing C numbers amounts to $\binom{N}{C}$. Therefore, the mathematical expectation of the maximum value of the integer among C selected is:

$$M(C) = \frac{C\binom{C-1}{C-1}+(C+1)\binom{C}{C-1}+(C+2)\binom{C+1}{C-1}+\ldots+N\binom{N-1}{C-1}}{\binom{N}{C}}$$

$$= \frac{\sum_{i=C}^{N} i\binom{i-1}{C-1}}{\binom{N}{C}} = \frac{C}{\binom{N}{C}} \cdot \sum_{i=C}^{N}\binom{i}{C} \qquad (19)$$

For the calculation of the sum $\sum_{i=C}^{N}\binom{i}{C}$ the following mathematical relation will be used:

$$\sum_{k=o}^{m}\binom{n+k}{n} = \binom{n+m+1}{n+1} \qquad (20)$$

From Eq. (20) and (19) follows that:

$$\sum_{i=C}^{N}\binom{i}{C} = \sum_{k=o}^{N-C}\binom{C+k}{C} = \binom{N+1}{C+1} \qquad (21)$$

By substitution Eq.(21) into Eq.(19) one obtains

$$M(C) = \frac{C}{\binom{N}{C}}\binom{N+1}{C+1} = \frac{C}{C+1}(N+1) \qquad (22)$$

and for a large N (N >> 1)

$$M(C) = \left(1 - \frac{1}{1+C}\right)N \qquad (23)$$

REFERENCES

/1/ Weingarten A., The Eschenbach drum cheme, Comm. ACM 9,7 (July 1966)

/2/ Denning P., Effects of scheduling on the file memory operations, Proc. AFIPS SJCC, Vol. 30,AFIPS Press, Montvale N.Y.,1967

/3/ Theory T.J. and Pinkerton T.B., A comparative analysis of disc scheduling policies, CACM, Vol. 15, No 3, 1972.

/4/ Fuller S., Analysis of Drum and Disc Storage Units, Springer-Verlag, Berlin-Heidelberg-New York, 1975

/5/ Feller W., An Introduction to Probability Theory and its Applications, Vol. 1, John Wiley & Sons, New York, 1968

/6/ Coffman E. and Denning P., Operating Systems Theory, Prentice Hall, New Jersey, 1973

/7/ ICL (1974), Direct Access, Technical Publication 4385

/8/ ICL (1972), Plan Reference Manual, Technical Publication 4322

MIKRORAČUNALNIŠKI SISTEM ZA NADZOR IN VODENJE DNEVNIH KOPOV

MILOŠ JENKO,
BOŠTJAN DELAK

UDK: 681.519.7

ISKRA DO AVTOMATIKA, TOZD PROJEKTIRANJE IN GRADNJA SISTEMOV

Članek podaja multimikroračunalniški sistem zasnovan na serijskih vodilih in možnosti njegove uporabe v nadzoru in vodenju segmentov in dnevnega kopa v celoti.

Sistem je sestavljen iz aparaturne opreme družine TI-30 in programske opreme, katere centralni del je programski paket MOSPIK (mikrooperacijski sistem protokoliranja in krmiljenja).

Prikazana je modularnost celotnega sistema in povdarjena je njegova uporabnost v aplikacijah različnega namena in obsega.

Programski paket MOSPIK omogoča zajem do 240 signalizacij z ločljivostjo 10 ms in 55 analognih meritev v času ene sekunde. Vsebuje vse standardne obdelave nad vhodnimi informacijami in nudi možnost prikaza informacij na alfanumeričnem zaslonu ali pisalniku.

V članku je podrobneje obdelana aplikacija paketa na projektu REK Bitola, kjer je izpeljan nadzor delovanja štirih bagrov, dveh odlagalcev, drobilnice, transformatorske postaje in sedemnajstih transporterjev.

The article describes the microcomputer system, based on serial buses and possibility to be applied for supervision and control of segments and surface mining on the whole.

The system consists of hardware from family TI-30 and software the central part of which is the programming pachage MOSPIK (microoperational system for registering and control).

The article represents the modularity of the entire system and points out its usage in applications of varions purposes and scopes.

Programming pachage MOSPIK encompasses up to 240 signallings with resolution 10 ms and 55 analoge measurings in time of one second. It deals with standard processes of input information and indicates it on alfanumerical display or printer.

The article shows in all details the application of package on the project REK Bitola where the system supervises four dredgers, two unloaders, crushing units, transformer stations and 17 conveyers.

Procesiranje informacij v projektu REK Bitola se sestoji iz treh delov:
- zajem informacij
- obdelava informacij
- izdaja informacij

ZAJEM INFORMACIJ

V projektu REK Bitola zajemamo 574 enobitnih digitalnih informacij z ločljivostjo ene sekunde. Dobimo jih preko telemehanskega sistema TM-15 in digitalnega aparaturnega modula DIV-D.

Naloga programskega paketa MOSPIK pri zajemu je, da določi spremembe digitalnega vhoda. Pod to adreso je vsebovana skupina osmih enobitnih digitalnih signalizacij, ki vsebujejo tudi nastalo spremembo enega ali več digitalnih vhodov. Informacija, ki vsebuje naslednje podatke:

- byte, ki označuje skupino digitalnih informacij, kjer je nastala sprememba,
- adresa skupin, kjer so nastale spremembe,

se shrani v krožni pomnilnik, ki predstavlja vhodni vmesnik za nadaljno obdelavo vhodnih informacij.

Modul, ki ima nižjo prioriteto izvede, iz prej omenjenega vhodnega vmesnika naslednje operacije:

- iz byta, ki označuje skupino sprememb, ugotovi število blokov, ki jih bo treba formirati za vpis v naslednji vhodni vmesnik modula, ki bo izvedel nadaljne obdelave,

- vsak formirani blok vsebuje podatke:
  - dolžina bloka;
  - adresa skupine vhodov, kjer je nastala sprememba;
  - digitalno stanje na vhodih in realni čas nastanka spremembe (ura, minuta, sekunda in stotinka sekunde)

OBDELAVA INFORMACIJ

S pomočjo statičnih list in vhodnih podatkov poišče v ranžirni listi (programska preslikava enobitnih digitalnih fizičnih vhodov) ustrezno podlisto, ki vsebuje zadnje stanje vseh digitalnih vhodov v eni skupini in pripadajoče tehnološke adrese. S primerjanjem vhodnega byta stanj in byta stanj v ranžirni podlisti, poišče spremembe stanj digitalnih vhodov v skupini. S pomočjo tehnološke adrese vpiše novo stanje v banko digitalnih signalizacij.

Pri izračunu časa delovanja posameznega objekta se pripravi vhodni vmesnik za modul za izračun časa. Ta vsebuje podatke:
- dolžina bloka,
- tehnološka adresa digitalnih signalizacij,
- realni čas nastanka spremembe.

Ko je sprememba obdelana, modul pripravi blok podatkov za izpis obratovalnega stanja informacije. Ta blok vsebuje:
- dolžino,
- številko izhodne enote (CRT, printer),
- podatke o realnem času spremembe,
- pripadajoče tehnološke adrese,
- stanje digitalne informacije.

Vpiše se v vhodni vmesnik modula za izpis obratovalnega protokola.

Modul za izračun časa delavanja objektov določi čas delovanja s pomočjo vhodnega vmesnika in statičnih list. Ta čas delovanja v zadnjem intervalu se prišteje k skupnem času delovanja objekta.

IZDAJA.

Izdaj informacij je sestavljena iz dveh delov:

- spontani izpis informacije
- ciklični izpis informacije

Celotni izpis pri projektu REK BITOLA se vrši na enem vhodnem kanalu - na printerju. Kljub zahtevi po izpisu spontanega protokola se lahko ciklični protokol neprekinjemo izpiše. Ta se v tem primeru izpiše po zaključku cikličnega protokola.

Sestavljena vrstica v obliki ASII kode se s pomočjo modula za izpis prenese v izhodni vmesnik. Vrstica se na izhodno enoto (printer) izpisuje znak po znak s pomočjo monitorskega dela programskega paketa MOSPIK.

Modul za spontani izpis vzame iz pripadajočega vhodnega vmesnika podatke. Po vključenem algoritmu s pomočjo statičnih list določi kodo digitalnega vhoda in pripadajoče delne tekste. Vrstico spontanega protokola sestavljajo:
- koda digitalnega vhoda,
- tekst z opisom digitalnega vhoda,
- stanje vhoda in realni čas nastopa spremembe na digitalnem vhodu.

Modul za ciklični izpis s pomočjo dinamičnih in statičnih list sestavi 25 vrstic teksta za 25 objektov. Te vrstice prenese v izhodni vmesnik za izpis na printer. Vrstica cikličnega izpisa vsebuje kodo objekta, tekstni opis objekta in čas delovanja za cikel zadnjih osmih ur.

INFORMATICA 1/1984

FUTURE DEVELOPMENTS IN COMPUTER ARCHITECTURE

J. V. KNOP,
K. SZYMANSKI,
N. TRINAJSTIĆ

COMPUTER CENTRE, UNIVERSITY OF DUSSELDORF
4000 DUSSELDORF, FEDERAL REPUBLIC OF GERMANY

UDK: 681.324

Abstract

An overview is given of the development of computer architecture. It is shown which approaches provide more computational power without changing the well-known hardware technology of silicon-based chips by developing new computer architectures. Multiprocessors, dispersed networks, local networks, data-flow systems and highly parallel systems are discussed. Their impact on mathematics and natural sciences is expected to be enormous, especially in the area of non-numerical aspects of computing, far surpassing the present achievements.

Povzetek

Podan je pregled poti razvoja računalniške arhitekture. Obravnavani so večprocesorski sistemi, porazdeljene in lokalne mreže in sodobni paralelni sistemi. Povdarjen je vpliv tega razvoja na naravoslovne vede v okviru nenumeričnega računanja.

1. Introduction

Considering the development of digital computers one finds that the speed of computation was increasing by a factor of 10 every 3 years throughout the sixties and seventies. This tendency was described by the "Grosch-Rule" (Figure 1) which states that the computational power is a function of the square of the costs (effort) (1); if one pays twice the price of its predecessor, one obtains a computer which has four times the computational power. And this computer had the advantage of being compatible with the predecessor. The reason of this development has to be seen in the rapidly evolving technology of microelectronic components and integrated circuits. This progress has historically gone from circuits containing gates, to MSI (Medium Scale Integration) chips for byte-sliced processors, memory and input-output functions. The current plateau of microminiaturization, namely Very-Large-Scale-Integration (VLSI), is enabling the introduction of integrated circuits (ICs) with cost performance characteristics (Figure 2) that are orders of magnitude superior to circuits available only a few years ago. By the mid-80s, million-bit memory components should be in production. By 1990 single-chip microcomputers and custom ICs can be expected to contain a million active devices (e.g. transistor elements) (2).

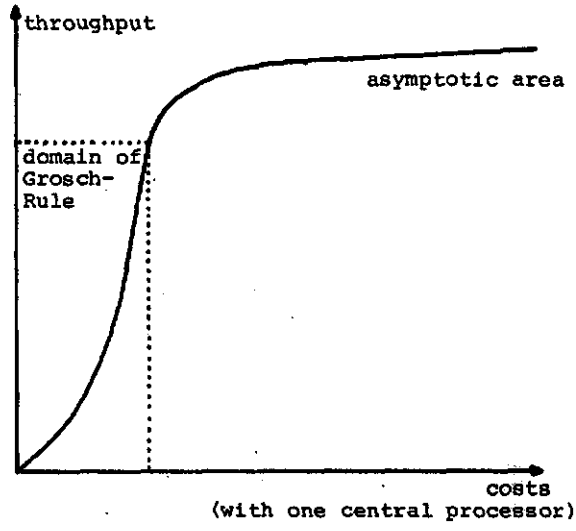
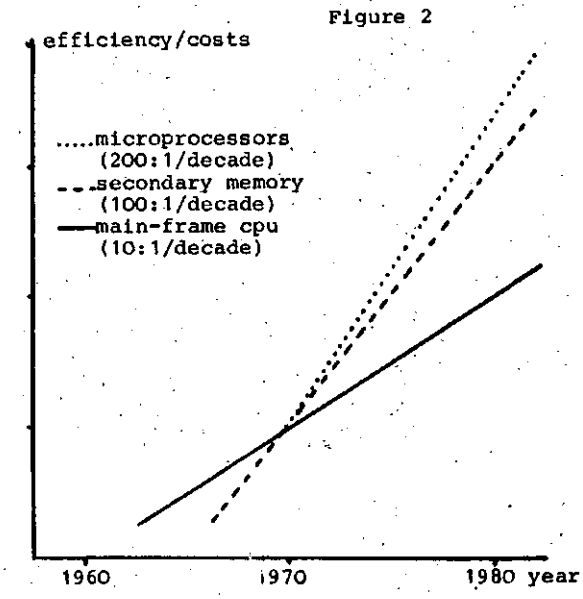


Figure 1: Grosch-Rule

But this progress in microminiaturization has to come to an end as physical boundaries are reached. We only point to limitations present in the fabrication of the chips (wave length of light), the problems of

Figure 2



several physical effects on the chip such as
- power dissipation
- usable area of silicon,
- cosmic-radiation faults,
- thermal and electrical noise, etc.

Of course extensive research and development in the following areas will extend these boundaries
- process and material technologies (especially in regard to improved material purity),
- lithography techniques (improvements in photolithography, UV, x-ray and electron-beam lithography),
- design and simulation techniques,
- packaging (for example, effective heat dissipation, reliability and pin-out limitations),
- testing (adequate fault detections, cost efficencies, selftesting, etc.)

In this area the combined efforts of molecular biologists, chemists, physicists and mathematicians may introduce quite a new technology based on the idea of the molecular switch. Thefore, the research in the area of bioelectronics should be viewed with optimism and hope that new kinds of chips (biochips) will emerge with superior properties in comparison with the chips presently in use.

What we can see at this point is that the validity of the Grosch-Rule is bounded so far as computers of the normal "von Neumann" -type are concerned. The Grosch function will lead to an asymptotic behaviour. By a von Neumann architecture we understand a computer consisting of a memory, a control unit and an arithmetic/logic unit. Instructions and data are stored in the memory and are processed in a "natural" sequence of instructions and addresses (Figure 3).

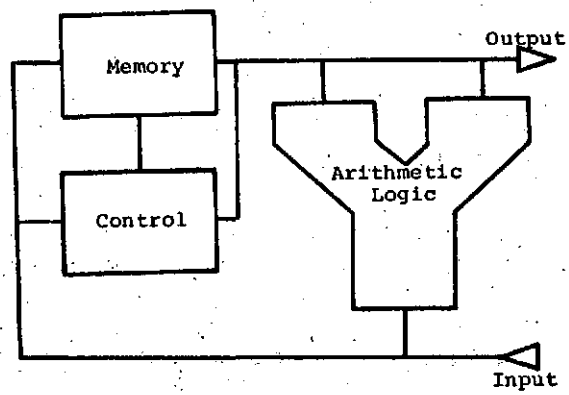


Figure 3: von Neumann computer

A new approach to get more computational power has to be made. This approach uses well-known hardware and tries to find a way to get more computational power by combining many computers into a network of computers, into a multicomputer system. Several types of multicomputer systems have been developed:

1. multiprocessors,
2. dispersed networks,
3. local networks and distributed systems,
4. data-flow systems,
5. highly parallel and internally concurrent systems.

We will consider these five types of multicomputer systems and discuss some of the advantages and disadvantages of these types of architecture.

2. Multiprocessors

One of the first ideas of getting more

computational power was the simple multiplication of the structure which led to multiprocessor structures. In 1971 Minsky made the conjecture that you can only expect an increase of the throughput of logarithmus dualis p (ld p)(Figure 4), where p is the number of processors.

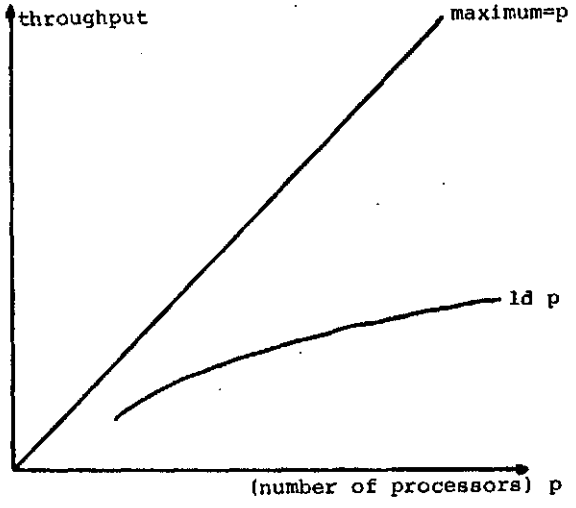
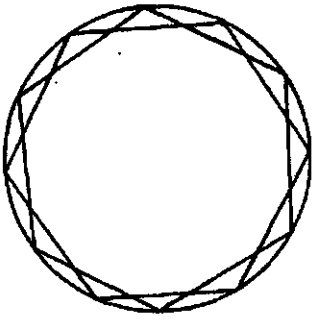


Figure 4: Minsky's Conjecture

This conjecture is nowadays assumed to have been hindering the research on the field of multiprocessors, because of the conditions for Minsky's statement was that only one job is processed. One finds other conditions if one processes more than one program or programs which have the ability to be processed in parallel. One of the greatest difficulties in constructing multiprocessors is the "degree of complexity" resulting from the necessity of combining all processors with each other, while the local complexity (that is the number of processors which are linked to one processor) is only growing linearly. In the graphs in scheme I and II the processors are denoted by the vertices and the connections between them by the edges. It is easily seen that the local complexity - the degree of the vertex - is equal to n-1 if each processor is connected with each other processor (n is the number of processors), and the global complexity is equal to the number of connections, i.e. edges of the graph, $k = n(n-1)/2$.

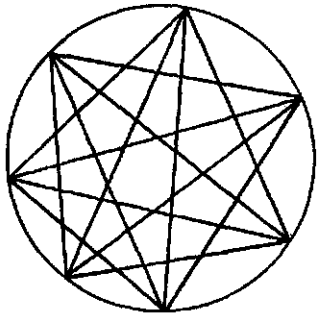
SCHEME I



The limited local complexity of multicomputer systems

So while constructing multiprocessors one has to reduce the global complexity, for instance by connecting one processor not with all the other processors but only with his four neighbours. Hereby you can assure that the local complexity will be constant and the global complexity grows only linearily.

SCHEME II



n = 7

Global Complexity $k = \frac{n(n-1)}{2} = 21$

Local Complexity $k = n-1 = 6$

In the times of expensive hardware a considerable effort was spent on the construction of systems which made sure that the processor at any time finds a program which can be processed.

This led to the principle of time sharing. Now in the times of diminishing hardware costs one finds a tendency that a program (or better, task) will stay in a processor for a longer time, processes data and moves data to the next processor which is then responsible for further processing of this data. One gets a flow of data through a field of processors working in parallel. To organize the data flow through the processors and distribute the tasks to the processors the operating system may be in a special processor or special layer of processors (if the operating system is divided in parallel working tasks, too). The architecture of this multiprocessor will look like a pyramid of processors linked horizontally and vertically together (Figure 5). At this point we have to mention that all processors are of the von Neumann-type.

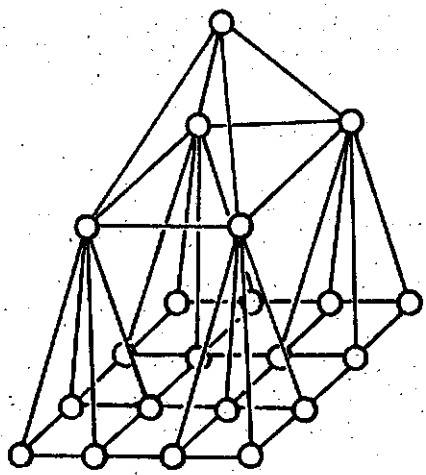


Figure 5: Multiprocessor architecture with constant local complexity

It is useful to construct programs being processed on such a multiprocessor in a way that segments of the program which are processed concurrently are formulated in special tasks. We give as an example of this way of processing the "Many-Body-Problem" in quantum mechanics (Figure 6). Another example which can be solved easily in this way is ab initio computations in quantum chemistry (3).

3. Dispersed Networks

3.1 Logical Structure

As communication reliability continues to increase and long distance communication costs continue to diminish, it becomes increasingly feasible to develop computer-communication networks spanning wide geographical areas. The demand for reliable, controllable communications in distributed systems, and to some extent, the desire for verifiable protocols has led to the definition and standardization of various levels of intercommunication abstractions in such networks. The current decomposition is the 7-Layer-Model (4) (Figure 7) such as

Layer	Examples
1. Physical	ISO 2593 (International Standardization Organisation)
2. Link	HDLC (High-Level Data Link Control), SDLC (Synchronous Data Link Control)
3. Network	X.25
4. Transport	TCP (Transport Control Protocol)
5. Session	SNA (System Network Architecture)
6. Presentation	VT (Virtual Terminal)
7. Application	IPC (Interprocess Comunication Protocol), FTP (File Transport Protocol)

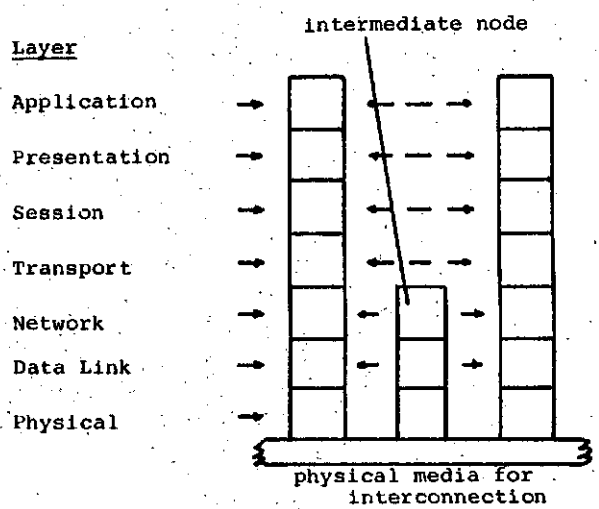
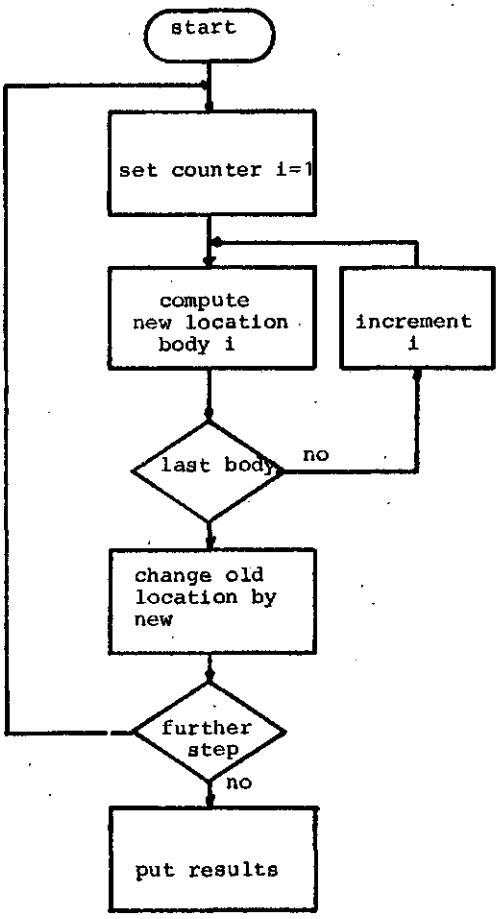


Figure 7: Layers of the reference-model



Figures 6: Many-Body-Problem (sequential)

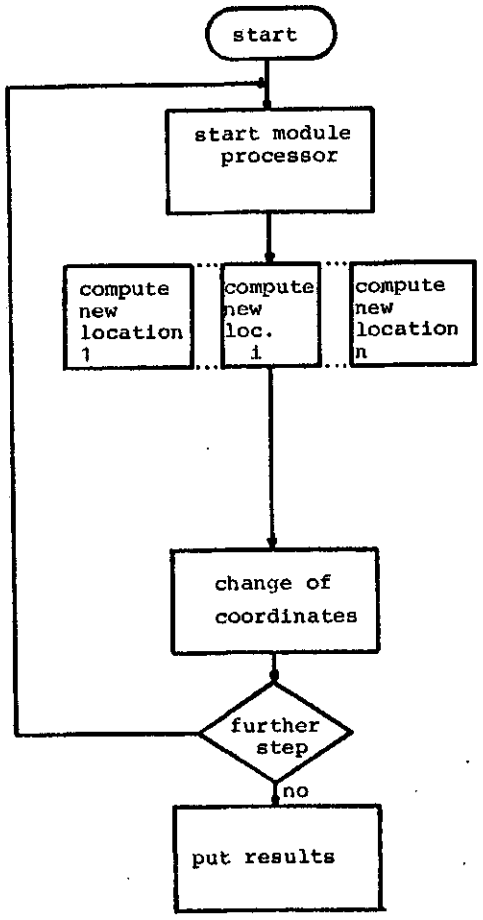


Figure 6: Many-Body-Problem (parallel)

Standardization of the lower four levels in this hierarchy provides the basis for flexible interconnectability. The physical level is concerned with the actual sequence of bits being transmitted on a communication channel between systems. The link level is concerned with the establishment amd maintenance of internode links. The network level is concerned with maintenance of the network and protocols for system intercommunication, including routing. The transport level is concerned with the transmission of packets, particularly end-to-end control of their reliable delivery.
The upper three levels are more application dependent and consequently less defined. It is hoped by the standardization groups that the establishment of such standards will have a stabilizing influence on computer proliferation, i.e. by increasing the ability of computers to be linked together. This standardization is also important to permit the construction of multicomputer structures of different hardware-components.

3.2 Examples

A good example for a dispersed network is the ARPANET, although its protocols should be considered as prototypes rather than as standards. A new network has just been established in the U.S. as a project of the National Science Foundation: CSNET (The Computer Science Research Network) (5). It is a logical network (Figure 8) utilizing the services of several physical networks, namely ARPANET, Phone Net, Telenet. There are three service-functions implemented in CSNET

- network mail (available to all users)
- file transfer, to copy files from one host to another, remote login, which is the ability to log into one's local CSNET host and use the network to get to another CSNET host and log in there.

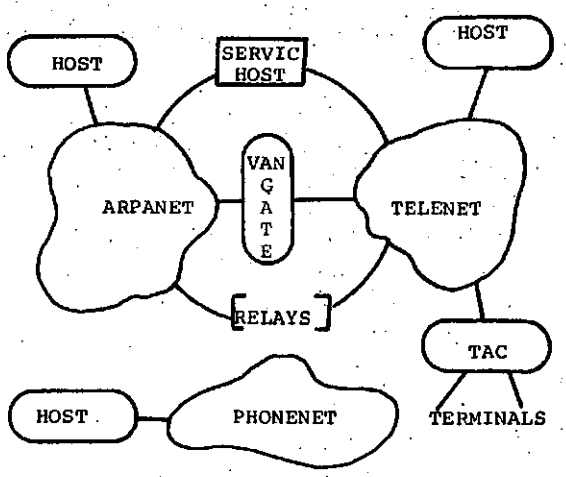


Figure 8: Logical Structure of CSNET (5)

3.4 Workstations

In close connection with the development of dispersed networks we have to see the development of workstations. Due to the sinking costs of hardware, personal computers are gaining more importance. They may be used in offices (public and business administration), schools, universities (6), research laboratories (6), computing centers, etc. Portable workstations help to allow one to become independent of a certain working place.
In contrast to the use of terminals of mainframes, single-user-computers guarantee independence and are always available. The user profile may be designed to the knowledge and the special interest of the user.
However, in many cases workstations alone will not be sufficient. A user may need facilities which are expensive to buy or which any user does not use continously. A high-speed printer, a good quality printer, high-speed disk storage, archiving facilities and service programs are examples for such facilities. In addition, there are problems which cannot be solved on a small personal computer, but must be done on large or very large computers.

Second, the normal office worker or scientist or any other person using workstations does not work in a vacuum and needs to communicate in various ways with the outside world.
To do this there is a need for access to specialised services and databases, information services, libraries, patents information, dat networks and electronic mail services. These should be accessible from the user's desk. For example, automated office systems generally based on interactive workstations connected to a communication network.
In conclusion one can say each workstation has some degree of functionality: text processing, communication and personal applications, which include in many cases graphic functions (7).

Workstations are typically single user (but multi-tasking) computers with local processing power, a keyboard, a visual display screen (which can eventually be replaced by a TV), local magnetic storage (floppy or Winchester drives) and a quality printer. The keyboard should be suitable for text handling. In many cases good support of program and data input is also useful. The keyboard may be replaced or supplemented by a mouse or another easy input system. The high resolution screen may have graphic facilities, for both line art graphics display and for the display of facsimile transmissions and other digitised images. For graphics a non-impact image printer is required. For economic reasons this will probably need to be a shared central facility, but in the longer term it may be possible for this to be provided in the workstation. Ergonomic considerations are important not only from the point of view of convenience, comfort and safety of the operators, but also for the need to minimize errors by bad human interfaces. As workstatons come into use in offices and other normal working places things like visual screen, keyboard, etc., have to be designed to ergonomic aspects. Environmental factors as heat and noise, etc., have to be considered (8). Other general important points of workstations are reliability, simplicity, flexibility and adaptibility.

In almost any case text processing facilities are required. Taylored to the different roles of workstation there may exist different software modules: decision support systems, computer aided graphics and design tools, compilers, etc..
To allow the communication functions, workstations must have access facilities to Local Area Networks and to Public Networks.

The following should be available, for example:

- electronic message systems for person-to-person communication
- teleconferencing systems
- Teletex
- Facsimile: electronic transmission of images which typically are not text, such as pictures, drawings, annotated documents, etc.
- remote data access: file transfer, access to remote (distributed) data bases
- remote access to central computing facilities
- terminal access to other computers
- remote job entry.

4. Local Networks and Distributed Systems

The technology developed for distributed networks has also influenced the design of local networks (4,9). From a functional point of view, a local computer network may be thought of as inhabiting a region between multiprocessing systems and dispersed networks for purpose of resource sharing. Characteristic of a local network is the existence of a high-speed transmission medium (up to 10 MBits/s) for data transmission over a "limited" (i.e. 0.1 - 10 km) distance. The nature of the transmission medium and the topology of the network are left unspecified. Several network adapters attached to this transmission medium serve as line interfaces for computing equipment. The adapters transmit data on the transmission medium. Computing system components can be attached to the adapter. Much effort has been devoted in recent years to developing technologies for local networks. Two of the basic networking technologies that have been developed are the ring and bus network. One of today's most widely emulated local network technologies is that of the Ethernet (2). Digital Equipment Corporation, Intel Corporation, and Xerox Corporation are jointly developing a local area network based on Xerox Ethernet. This network consists of a coaxial cable as transmission medium and transceivers to link different kinds of computers, peripherals and data terminals. The network is designed to operate at 10 million bits per second. A typical configuration consists of one or more segments of coaxial cable each of which can be up to 500 meters in length. The current design is limited to a maximum distance of 2.5 kilometers. One hundered terminals can be supported per segment, and overall maximum is at present 1000.
The Ethernet, as it is most often used, acts merely as a communication medium. Computational resources of a particular maschine are not made available to any other maschine in the local network.
Xerox has been pursuing a very interesting experiment in which programs can span maschine boundaries. Such programs are called worm programs. A worm is composed of program segments, each of which is capable of running on any other maschine. The worm mechanism finds free machines as needed.

5. Highly-Parallel and Internally Concurrent Systems

The computer architecture introduced with
- dispered networks and
- local networks

should increase the computational power in the sense of getting more throughput for a community of users, to get additional functions on remote hosts or to find adequate computational power for solving special kinds of problems. These architectures do not help one to get high speed for extensive computation for solving a single problem. For these computations for instance on the field of computer tomography, pattern recognition, or quantum chemistry, one needs the so called supersystems (10,13). By a supersystem we understand systems with throughput in excess of one billion instructions per second and with general purpose computational flexibilty. These systems, of course, do not exist today).
At this point of discussion we have to mention highly parallel and internally concurrent systems such as ILLIAC IV and vectormachines as CDC Star, Cray 1 and CDC Cyber 205. The systems are often thought of as multicomputers but they can be thought of at a very high level of abstraction as systems that logically have only a single processor and a single instruction stream which is able to operate on a field of data. These architectures do not lead to a supersystem because of lack of general purpose computational flexibility.

6. Data-Flow Architecture

One of the most exciting new directions in multicomputer architecture is unquestionably the data-flow concept (2). In traditional computing systems (von Neumann machines), a processor addresses a piece of data by its location. An instruction is executed with whatever data happen to be in that location and the processor then goes to the next instruction. Thus the data come to the processor in a sequentially preprogrammed way. In data-flow architecture there is no explicit concept of a program counter or of control flow. An instruction is ready to be executed whenever all its operants are in place. Thus execution of many instructions may happen concurrently, in an order that need not be determined in advance. As a consequence this architecture offers a great potential for the future of very large computations, especially those that permit extensive instrinsic concurrency. The only disadvantage of such a system is that it is not developed until now. The data-flow architecture is an architecture which is far away from the von Neumann architecture.

7. Some Application to Natural Sciences

7.1 Portable Soft-ware

In order to increase the usefulness of personal computers, the exchangeability of programs is a necessary contition. Therefore, one may expect the growth of the availability of portable programs in the future.

7.2 Network Communication

For many results observed in natural sciences and mathematics, the essential quality is the speed and the cost of one dissemination of information. Under the speed of dissemination of information we consider, for example, transmitting simultaneously the information to say 200 places in less than 30 minutes. The transmitted information may be a new experimental data, or a correction of some previously published data, etc. The cost of transmitting information in this way is expected to be reduced in comparison with the cost of classical systems (secretary's typing time, preparing addresses in some way, postage, time spent while information travels from sender to receiver,etc.). A reduction in cost is expected because the network communication becomes a one-man operation. Besides, in science it becomes a scientist-to-scientist communication without going through any structure in between them. In addition, it is a 24-hour a day, every day of the week, open system. This becomes even more important in connection with the prediction of many that a great number of the scientific journals will gradually change their mode of printing and distribution. It is exspected that they will be electronically prepared and distributed through the network communicating units.

7.3 Knowledge Centers

Many computations in natural sciences, especially in physics and chemistry are routine work. However, the results are not available for all concerned. This is so because many do not possess either the computing devices or the necessary soft-ware or the time for doing the desired computations. The lack of the first two factors may be avoides by establishing the national and international knowledge centers for each faculty of science which would provide the necessary to know-how, computations, and even the interpretations. The time factor is related to the predictivity power of the given theory and the related soft-ware, because in this ever competitive world the standard theoretical calculations should not last longer than the related experimental work (14).
Finally, the justification for establishing the knowledge centers is also the fact that we cannot expect that the time will ever come when everybody would possess the time and the necessary specialised hard-ware and soft-ware.

In order to summarize this section we point out that the use of computers in natural sciences and mathematics will be influenced by two major factors. The first one is of the organisational nature and the other one is of technical nature. Only improvements and new developments in both aspects may bring forth the rational use of computers in chemistry.

8. Conclusions

In the future we see the development of microcomputers to such an extent that they will overshadow the traditional main-frame hard-ware. We see black days for mainframes, because they failed to produce the techniques for computer-computer communications, like local network and is not yet possible to carry out the automatic decomposition of large numerical and non-numerical problems to loosely coupled part-problems. Therefore, the Amdahl- Cray-Cyber line is probably a dead end line because it does not permit the development of the novel concepts in computer technology.

Bibliography

1. G. Regenspurg (Ed.),
   GMD - Rechnerstruktur - Workshop,
   Bericht Nr. 128,
   (R. Oldenburg Verlag, München-Wien, 1980).

2. P. G. Neumann, D. H. Brandin,
   J. A. Goguen, J. Goldberg, and
   M. A. Placko, Directions for Future
   Research and Development in Multicomputer Systems, SRI Project 1826.

3. R. Seege, J. Comput. Chem. 2, 168 (1981).

4. Carl Troppe,
   Local Computer Network Technologies,
   (Academic Press, New York, 1981).

5. CSNET Project Status Report 7/8/82.

6. Deutsches Forschungsnetz - DFN,
   Bericht der Arbeitsgruppen, Karlsruhe,
   1982

7. M. H. Olson and H. C. Lucas Jr.,
   Comm. ACM, 25, S. 838 (1982)

8. S. G. Price,
   Introducing the Electronic Office,
   (NCC Publications, Manchester, 1979).

9. K. C. E. Gee,
   Local Area Networks,
   (NCC Publications, Manchester, 1982).

10. R. G. Arnold, R. O. Berg and W. Thomas,
    IEEE Trans. Comput., C 31, 385, (1982).

11. E. E. Swartzlander, B. K. Gilbert,
    IEEE Trans. Comp., C 31, 399 (1982).

12. J. P. Ignazio, D. F. Palmer, and
    C. M. Murphy,
    IEEE Trans. Comput. C 31, 410 (1982).

13. K. B. Irani and N. G. Khabbaz,
    IEEE Trans. Comp. C 31, 419 (1982).

14. M. J. S. Dewar, Science 190, 531
    (1975).

O IMPLEMENTACIJI PREVODIOCA LIPSKIT LISP – JEZIKA NA JEZIK SECD – MAŠINE IZVRŠENOJ NA FORTRAN – JEZIKU

IVAN STOJMENOVIĆ,
VOJISLAV STOJKOVIĆ,
LJUBOMIR JERINIĆ,
JULIJANA MIRČEVSKI

INSTITUT ZA MATEMATIKU, NOVI SAD

UDK: 681.3.068

Programski jezik LISPKIT LISP je jedan dijalekat programskog jezika LISP.
U radu je detaljno opisana implementacija prevodioca LISPKIT LISP-jezika na jezik SECD-mašine. Implementacija je izvršena na FORTRAN-jeziku i prosleđena je na računaru DELTA 11/34Ø Instituta za matematiku Prirodnomatematičkog fakulteta u Novom Sadu.
Navedeni prevodilac je sastavni deo interpretatora LISPKIT LISP-jezika. Koliko je autorima poznato, ovo je prva implementacija LISPKIT LISP-interpretatora izvršena u našoj zemlji.

An implementation of LISPKIT LISP-translator.
The programming language LISPKIT LISP is the one dialekt of the programming language LISP.
In this paper was described the one implementation of the translator of LISPKIT LISP-language to SECD-machine language. The implementation is completed in FORTRAN-language and is tested on the DELTA 11/34Ø computer system of the Institue of mathematics University of Novi Sad.
The translator is the part of LISPKIT LISP-language interpreter. The authors don't know any similar implementation of LISPKIT LISP-interpreter in Yugoslavia.

1. Uvod

Programski jezik LISP (McCarthy [3]) je najpoznatiji jezik veštačke inteligencije. Koristi se za implementaciju sistema i visokih programskih jezika veštačke inteligencije.

Od 1958. godine do današnjih dana izvršeno je niz programskih i mašinskih implementacija LISP-jezika i njegovih dijalekata. Najpoznatije su sledeće implementacije(Boley [1]):

Godina	Verzija	Autor	Mesto-Ustanova
58	LISP 1	McCarty	MIT
62	LISP 1.5	McCarty	MIT
68	LISP A	Sandewall	Uppsala
7o	MACLISP	Moon	MIT
73	LISP70	Tesler	Stanford
73	MLISP2	Smith	Stanford
73	QLISP	Wilber	SRI
74	LISP MACHINELISP	Weinreb	MIT
75	MAGMA LISP	Montangero	Pisa
76	VLISP	Greussay	Paris
79	CONCURRENT LISP	Sugimoto	Kyoto
80	LOGLISP	Robinson	Syracuse
82	COMMON LISP	Stule	C-MV
82	GLISP	Novak	Stanford

U ovom radu je detaljno opisana implementacija prevodioca LISPKIT LISP-jezika na jezik SECD-mašine. Implementacija je izvršena na FORTRAN-jeziku i prosleđena je na računaru DELTA 11/34Ø.

Navedeni prevodilac je jedan od tri sastavna dela interpretatora LISPKIT LISP-jezika napisanih na višem programskom jeziku:

1) LISP-prevodilac.FORTRAN,
2) LISP-prevodilac.LISP (Henderson [2]),
3) Simulator SECD-mašine.FORTRAN(Stojković [6]).

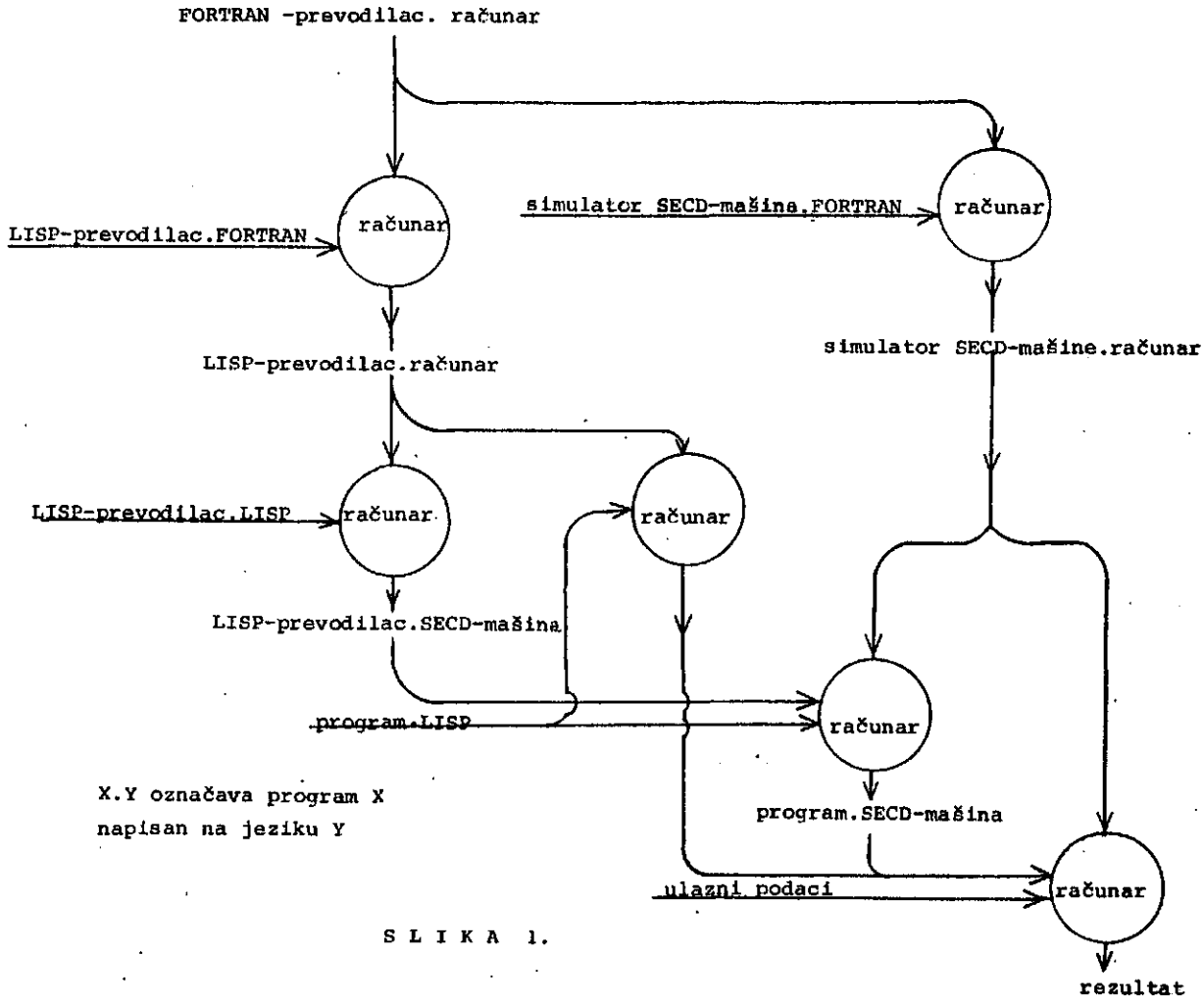
(Videti sliku 1)

Za izvršavanje (rad) LISP-interpretatora neophodan je jedan od sledeća dva para programa:

A.1. LISP-prevodilac.računar
A.2. Simulator SECD-mašine.računar ili
B.1. LISP-prevodilac.SECD-mašina
A.2. Simulator SECD-mašine.računar.

Programi A.1 i B.1 su semantički ekvivalentni. Međutim:

a) program B.1. dobija se automatskim prevođenjem programa LISP-prevodilac.LISP pomoću programa A.1. ili ručnim prevođenjem programa



S L I K A 1.

LISP-prevodilac.LISP što se inače zbog grešaka ne preporučuje.

b) izvršavanje programa B.1. simulira se pomoću programa A.2., što nije slučaj sa izvršavanjem programa A.1.

c) program B.1. se sporije izvršava od programa A.1.

d) zapis programa B.1. je kraći i čitljiviji od zapisa programa A.1.

e) program B.1. se jednostavnije modifikuje od programa A.1.

f) jednostavno se dokazuje korektnost programa B.1., što nije slučaj za program A.1.

Programi A.1. i B.1. izvršavaju prevođenje programa sa LISPKIT LISP-jezika na jezik SECD-mašine.

Program A.2. izvršava na realnom(postojećem) računaru simulaciju procesa izvršavanja datog programa napisanog na mašinskom jeziku SECD-mašine za date ulazne podatke na SECD-mašini i po završetku procesa izdaje dobijeni rezultat.

2. LISPKIT LISP-jezik

Programski jezik LISPKIT LISP je čist funkcionalni programski jezik i predstavlja jedan dijalekt(verziju) programskog jezika LISP.

Atributi čist i funkcionalni imaju sledeće značenje:

- ne postoje bočni efekti,
- LISPKIT LISP-programi su funkcionalnog oblika.

Simbolički ili skraćeno S-izrazi su osnovna struktura podataka i funkcija LISP-jezika i njegovih dijalekata.

Sintaksa S-izraza se može pomoću proširene Backusove notacije definisati na sledeći način(Stojković [7]):

⟨S-izraz⟩ ::= [⟨razdvoj⟩] ⟨član⟩ [⟨razdvoj⟩] [control/z]

⟨razdvoj⟩ ::= ⟨razdvajač⟩ {⟨razdvajač⟩}

⟨razdvajač⟩ ::= ␣ | tab |)

⟨član⟩ ::= ⟨atom⟩ | ⟨molekul⟩

⟨atom⟩ ::= ⟨simbol⟩ | ⟨ceo broj⟩

⟨simbol⟩ ::= ⟨slovo ili podvlaka⟩ {⟨slovo ili podvlaka⟩ | ⟨cifra⟩}$_0^{79}$

⟨slovo ili podvlaka⟩ ::= A|B|C|...|X|Y|Z|_

⟨cifra⟩ ::= 0|1|2|3|4|5|6|7|8|9

⟨ceo broj⟩ ::= [⟨znak⟩] ⟨ceo broj bez znaka⟩

⟨znak⟩ ::= +|-

⟨ceo broj bez znaka⟩ ::= ⟨cifra⟩ {⟨cifra⟩}

⟨molekul⟩::=([⟨razdvoj⟩] ⟨S-izraz lista⟩ [⟨razdvoj⟩])

⟨S-izraz lista⟩::=⟨simbol⟩ ⟨S-izraz simbol⟩ | ⟨ceo broj⟩ ⟨S-izraz ceo broj⟩ | ⟨molekul⟩ ⟨S-izraz molekul⟩

⟨S-izraz simbol⟩::=⟨razdvoj⟩ ⟨simbol⟩ ⟨S-izraz simbol⟩ | ⟨razdvoj⟩ ⟨ceo broj bez znaka⟩ ⟨S-izraz ceo broj⟩ | [⟨razdvoj⟩] ⟨znak⟩ ⟨ceo broj bez znaka⟩ ⟨S-izraz ceo broj⟩ | [⟨razdvoj⟩] ⟨molekul⟩ ⟨S-izraz molekul⟩ | ε

⟨S-izraz ceo broj⟩::=[⟨razdvoj⟩] ⟨simbol⟩ ⟨S-izraz simbol⟩ | ⟨razdvoj⟩ ⟨ceo broj bez znaka⟩ ⟨S-izraz ceo broj⟩ | [⟨razdvoj⟩] ⟨znak⟩ ⟨ceo broj bez znaka⟩ ⟨S-izraz ceo broj⟩ | [⟨razdvoj⟩] ⟨molekul⟩ ⟨S-izraz molekul⟩ | ε

⟨S-izraz molekul⟩::=[⟨razdvoj⟩] ⟨S-izraz lista⟩ | ε

⟨control/z⟩::=↑z

```
LISP1                                          (1)
-2                                             (2)
((DR I DJ 4)(21000 NOVI SAD))                  (3)
((DR.(I.(DJ.(4)))).((21000.(NOVI.(SAD)))))    (4)
```

su primeri S-izraza. Prema članu kojeg sadrže, S-izraze možemo podeliti na (simboličke i numeričke) atome i molekule. (1) je primer simboličkog, a (2) numeričkog atoma. Po konvenciji o izostavljanju tačke(.) izrazi (3) i (4) predstavljaju isti molekul.

Prazan molekul () se označava sa NIL. Po definiciji je NIL simbolički atom.

Simbolički atomi T i F označavaju redom istinitu i lažnu logičku vrednost.

LISPKIT LISP jezik se sastoji iz sledećih klasa funkcija:

a) primitivnih funkcija

-(CAR X) -vrednost funkcije je prvi element vrednosti molekula X.

-(CDR X) - vrednost funkcije je molekul(može i prazan) dobijen izostavljanjem prvog elementa molekula X ili drugi element para X=(A.B).

-(CONS X Y) -vrednost funkcije je par (X.Y).

-(QUOTE X) -vrednost funkcije je jednaka X.

b) aritmetičkih funkcija

-(ADD X Y) -vrednost je X+Y (tj. zbir vrednosti argumenata X i Y).

-(SUB X Y) -vrednost je X-Y.

-(MUL X Y) -vrednost je X*Y (proizvod vrednosti X i Y).

-(DIV X Y) -vrednost je celobrojni količnik X div Y.

-(REM X Y) -vrednost je ostatak X mod Y celobrojnog deljenja X sa Y.

-(EXPT X Y) -vrednost je X^Y (X** Y) (Y≥0).

c) logičkih funkcija (predikata)

-(ATOM X) -vrednost funkcije je T ako je vrednost argumenta X atom.

-(EQ X Y) -vrednost funkcije je T ako su vrednosti argumenata X i Y međusobno jednaki atomi.

-(NE X Y) -vrednost funkcije je T ako su vrednosti argumenata X i Y međusobno različiti atomi.

-(LE X Y) -vrednost funkcije je T ako je vrednost numeričkog atoma X manja ili jednaka vrednosti numeričkog atoma Y.

-(GE X Y) -vrednost funkcije je T ako je vrednost numeričkog atoma X veća ili jednaka vrednosti numeričkog atoma Y.

-(LT X Y) -vrednost funkcije je T ako je vrednost atoma X manja od vrednosti Y.

-(GT X Y) -vrednost funkcije je T ako je vrednost X veća od vrednosti Y.

d) uslovne (if) funkcije (uslovnog izraza)

-(IF X Y Z) -ako je vrednost argumenta X jednaka T vrednost uslovne funkcije jednaka je vrednosti S-izraza Y, u suprotnom vrednost uslovne funkcije jednaka je vrednosti S-izraza Z.

e) λ-izraza (ili definicionog izraza funkcije)

-(LAMBDA (X_1 ... X_m) Y) -$X_1,\ldots,X_m$ su simbolički atomi i predstavljaju promenljive od kojih zavisi funkcija koja se definiše. Y je S-izraz obrazovan od navedenih promenljivih i služi za izračunavanje vrednosti funkcije, nakon što se izvrši povezivanje promenljivih sa odgovarajućim vrednostima.

f) poziva funkcije

-(F X_1 ... X_m) - F je simbolički atom i predstavlja naziv funkcije. $X_1,\ldots,X_m$ su argumenti funkcije.

g) blok nerekurzivnih, odnosno rekurzivnih lokalnih definicija

-(LET Y (X_1.Y_1)...(X_m.Y_m)) ,

-(LETREC Y (X_1.Y_1)...(X_m.Y_m)) -$X_1,\ldots,X_m$ su simbolički atomi i predstavljaju lokalno definisane promenljive. $Y_1,\ldots,Y_m$ su S-izrazi i redom definišu lokalne promenljive. (X_1.Y_1),..., (X_m.Y_m) su lokalne definicije i označavaju pridodeljivanje atomima $X_1,\ldots,X_m$ vrednosti S-izraza $Y_1,\ldots,Y_m$.

```
(LETREC APPEND
      (APPEND LAMBDA ( X Y )
            (IF (EQ X (QUOTE NIL)) Y
                (CONS(CAR X)(APPEND(CDR X)Y))
            )
      )
)
```

je primer LISPKIT LISP-programa. Navedeni program definiše funkciju append. Vrednost funkcije je molekul jednak konkatenaciji molekula koji redom odgovaraju prvom i drugom argumentu funkcije.

Za argumente (1 2 3) i (1 (2)3) koji se zadaju kao ulazni podatak ((1 2 3)(1 (2)3)) dobija se rezultat (1 2 3 1 (2) 3).

Može se formalno pisati:

lisp(append,((1 2 3)(1(2)3)))=(1 2 3 1(2)3)

ili u opštem slučaju

lisp(fn,args)=result,

što znači da lisp-interpretator za funkciju fn i ulazni podatak args daje za rezultat result.

3. Opis SECD-mašine

SECD-mašina se može formalno definisati kao binarna funkcija execute(fn,args).

Prvi argument funkcije execute fn je program napisan na mašinskom jeziku SECD-mašine. Drugi argument funkcije execute- args je ulazni podatak programa.

Vrednost funkcije execute- result je rezultat dobijen izvršavanjem programa fn za ulazni podatak args na SECD-mašini.

Program fn, ulazni podatak args i rezultat result su S-izrazi.

SECD-mašina se sastoji iz 4 glavna, više pomoćnih registara i memorije. Glavni registri su:

s(STACK)- koristi se za čuvanje međurezultata dobijenih u procesu izračunavanja vrednosti funkcija- izvršavanja programa,

e(ENVIRONMENT)- koristi se za čuvanje tekućih vrednosti promenljivih,

c(CONTROL)- koristi se za čuvanje programa ili dela programa koji se trenutno izvršava,

d(DUMP)- koristi se za privremeno čuvanje vrednosti registara s, e i c za vreme izvršavanja neke funkcije.

Imenovani pomoćni registri su NIL, T i F koji se redom koriste za čuvanje vrednosti atoma: NIL, T i F.

Memorija je lista(molekul) slobodnih elemenata.

Sadržaj svakog od navedenih registara je S-izraz ili pri implementaciji pokazivač na strukturu podataka koja predstavlja S-izraz.

Stanje SECD-mašine može se formalno opisati navođenjem sadržaja 4 glavna registra- otud i naziv SECD.

Rad SECD-mašine je definisan programom fn i ulaznim podatkom args.

Na početku rada SECD-mašina se nalazi u početnom stanju. Početno stanje SECD-mašine je: s=(args), e=NIL, c=fn, d=NIL .

Za vreme rada, SECD-mašina prelazi iz jednog stanja u drugo stanje. Prelazak iz jednog stanja u drugo stanje je definisano naredbom programa, koji se trenutno izvršava.

Po završetku rada,SECD-mašina se nalazi u završnom stanju. Prvi element molekula, čiji je pokazivač sadržaj s-registra završnog stanja SECD-mašine je rezultat izvršavanja programa.

Naredbe mašinskog jezika SECD-mašine definišu se navođenjem stanja SECD-mašine pre i posle izvršavanja naredbe, što se simboliči piše: $s\ e\ c\ d \xrightarrow{\text{naredba}} s'\ e'\ c'\ d'$ i naziva prelaz SECD-mašine.

dekadni i mnemo kod operacije	prelaz SECD-mašine
1 LD	s e (LD i.c) d → (x.s) e c d ; x=locate(i,e)
2 LDC	s e (LDC x.c) d → (x.s) e c d
3 LDF	s e (LDF c′.c) d → ((c′.e).s) e c d
4 AP	((c′.e′) v.s) e (AP.c) d → NIL (v.e′) c′ (s e c.d)
5 RTN	(x) e′ (RTN) (s e c.d) → (x.s) e c d
6 DUM	s e (DUM.c) d → s (NIL.e) c d
7 RAP	((c′.e′) v.s) (NIL.e) (RAP.c) d → NIL rplaca(e′,v) c′ (s e c.d)
8 SEL	(x.s) e (SEL c_T c_F.c) d → s e c_x (c.d) ; x=T ili x=F
9 JOIN	s e (JOIN) (c.d) → s e c d
10 CAR	((a.b).s) e (CAR.c) d → (a.s) e c d
11 CDR	((a.b).s) e (CDR.c) d → (b.s) e c d
12 ATOM	(a.s) e (ATOM.c) d → (t.s) e c d ; t=T ako je a atom, t=F u suprotnom
13 CONS	(a b.s) e (CONS.c) d → ((a.b).s) e c d
14 EQ	(a b.s) e (EQ.c) d → (b=a.s) e c d
15 ADD	(a b.s) e (ADD.c) d → (b+a.s) e c d
16 SUB	(a b.s) e (SUB.c) d → (b-a.s) e c d
17 MUL	(a b.s) e (MUL.c) d → (b*a.s) e c d
18 DIV	(a b.s) e (DIV.c) d → (b/a.s) e c d
19 REM	(a b.s) e (REM.c) d → (b mod a.s) e c d
20 LE	(a b.s) e (LE.c) d → (b≤a.s) e c d
21 EXPT	(a b.s) e (EXPT.c) d → (b^a.s) e c d
22 NE	(a b.s) e (NE.c) d → (b≠a.s) e c d
23 GE	(a b.s) e (GE.c) d → (b≥a.s) e c d
24 LT	(a b.s) e (LT.c) d → (b<a.s) e c d
25 GT	(a b.s) e (GT.c) d → (b>a.s) e c d
26 STOP	s e (STOP) d → s e (STOP) d

locate((a.b),e) je član sa rednim brojem b u podmolekulu sa rednim brojem a molekula e. Podmolekuli datog molekula, kao i njihovi čla-

novi su numerisani brojevima Ø,1,2...

rplaca(e′,v) je S-izraz e′u kome je izvršena zamena (CAR e′)=v.

Funkcija STOP štampa (CAR s)- rezultat rada SECD-mašine.

Sledi primer rada SECD-mašine.

```
s e(LDC 1 LDC 2 LDC 3 MUL SUB LDC 4 EQ STOP) d
(1.s) e (LDC 2 LDC 3 MUL SUB LDC 4 EQ STOP)  d
(2 1.s) e (LDC 3 MUL SUB LDC 4 EQ STOP)      d
(3 2 1.s) e (MUL SUB LDC 4 EQ STOP)          d
(6 1.s)     e (SUB LDC 4 EQ STOP)            d
(-5.s)        e (LDC 4 EQ STOP)              d
(4 -5.s)        e (EQ STOP)                  d
(F.s)             e (STOP)                   d
```

Rezultat rada SECD-mašine je F. U primeru je proverena tačnost relacije 4=1-3*2.

4. Prevodilac LISPKIT LISP-jezika na jezik SECD-mašine

Označimo sa compile prevodilac LISPKIT LISP-jezika na jezik SECD-mašine, sa fn program na LISPKIT LISP-jeziku i sa fn^* odgovarajući program na mašinskom jeziku SECD-mašine. Očigledno je da važi:

$$compile(fn)=fn^*.$$

Ako se na program fn i ulazni podatak args primeni funkcija execute dobija se kao rezultat result. To dalje znači da važi:

$$lisp(fn,args)=execute(fn^*,args)=result\ .$$

Za svaku funkciju (program) LISPKIT LISP-jezika postoji pravilo na osnovu kojeg se vrši prevođenje u odgovarajuću funkciju jezika SECD-mašine. Prevod S-izraza e u odnosu na molekul promenljivih n označimo sa e*n.

Važe sledeća pravila prevođenja:

- $x*n=(LD\ i)$; $i=location(x,n)$

Ovo pravilo se primenjuje ako je x atom.

$i=(a.b)=location(x,n)$ označava da je x b-ti element u a-tom podmolekulu molekula n. Na primer,

$location(APPEND,((A\ B)(D\ E\ 3)(APPEND\ 1))=(2.0)$

- $(QUOTE\ s)*n = (LDC\ s)$
- $(ADD\ e_1\ e_2)*n=e_1*n|e_2*n|(ADD)$
- $(CONS\ e_1\ e_2)*n = e_2*n|e_1*n|(CONS)$
- $(IF\ e_1\ e_2\ e_3)*n = e_1*n|(SEL\ e_2*n|(JOIN)\ e_3*n|(JOIN))$
- $(LAMBDA\ (X_1\ \ldots\ X_k)\ e)*n = (LDF\ e*((X_1\ \ldots\ .X_k).n)|(RTN))$

a|b=append(a,b) označava konkatenaciju.

Za sve aritmetičke funkcije i binarne logičke funkcije pravila prevođenja su analogna pravilu prevođenja za ADD. Zato se može pisati:

- $(BIN\ e_1\ e_2)*n = e_1*n|e_2*n|(BIN)$; $BIN\in\{ADD, SUB,MUL,DIV,REM,EXPT,EQ,NE,LE,GE,LT,GT\}$
- $(UN\ e)*n = e*n|(UN)$; $UN\in\{CAR,CDR,ATOM\}$
- $(LET\ e\ (X_1.e_1)\ \ldots\ (X_k.e_k))*n = (LDC\ NIL)|e_k*n|(CONS)|\ldots|e_1*n|(CONS)|(LDF\ e*m|(RTN)AP)$; $m=((X_1\ \ldots\ X_k).n)$
- $(LETREC\ e\ (X_1.e_1)\ \ldots\ (X_k.e_k))*n = (DUM\ LDC\ NIL)|e_k*m|(CONS)|\ldots|e_1*m|(CONS)|(LDF\ e*m|(RTN)\ RAP)$; $m=((X_1\ \ldots\ X_k).n)$
- $(e\ e_1\ \ldots\ e_k)*n = (LDC\ NIL)|(e_k*n|(CONS)|\ldots|e_1*n|(CONS)|e*n|(AP)$

U poslednjem pravilu e je poziv funkcije.

Označimo sa comp(e,n,c) funkciju comp(e,n,c) =e*n|c. Prevod funkcije LISPKIT LISP-jezika na jezik SECD-mašine može se napisati u obliku:

$$fn^*=compile(fn)=fn*NIL|(AP\ STOP)=comp(fn,NIL,(AP\ STOP))\ .$$

Prevedimo, na primer, funkciju

f=(LAMBDA(X Y)(ADD(MUL X(CAR Y))(QUOTE 1)))

koja izračunava proizvod vrednosti X i prvog elementa Y uvećan za 1.

f^*=(LAMBDA(X Y)(ADD(MUL X(CAR Y))(QUOTE 1)))* NIL|(AP STOP)= (LDF(ADD(MUL X(CAR Y))(QUOTE 1))*((X Y).NIL)|(RTN))|(AP STOP)= (LDF(MUL X(CAR Y))*((X Y))|(QUOTE 1)*((X Y))|(ADD)|(RTN))|(AP STOP)= (LDF X*((X Y))|(CAR Y)*((X Y))|(MUL)|(LDC 1)|(ADD)|(RTN))|(AP STOP)= (LDF(LD(0.0))|Y*((X Y))|(CAR)|(MUL)|(LDC 1)|(ADD)|(RTN))|(AP STOP)= (LDF(LD(0.0))|(LD(0.1))|(CAR)|(MUL)|(LDC 1)|(ADD)|(RTN)|(AP STOP)= (LDF(LD(0.0)LD(0.1)CAR MUL LDC 1 ADD RTN) AP STOP)

Prevod zapisan pomoću dekadnog koda operacija glasi:

f^*=(3(1(0.0)1(0.1)10 17 2 1 15 5)4 26) .

Za ulazni podatak (2(3 4)) LISP-interpretator daje rezultat

$lisp(f,(2(3\ 4))=execute(f^*,(2(3\ 4))=2*3+1=7.$

Promenljivoj X je dodeljena vrednost 2, a promenljivoj Y vrednost (3 4).

Koristeći navedena pravila svaka funkcija se može ručno prevesti sa LISPKIT LISP-jezika na jezik SECD-mašine. Da bi se prevođenje izvršilo automatski potrebno je programski implementirati funkciju compile, odnosno comp.

Obrada S-izraza u FORTRAN-jeziku je detaljno opisana u [5]. Najveći problem je nepostojanje rekurzije u FORTRAN-jeziku i rešen je upotrebom steka ([4]).

S-izrazi se u FORTRAN-jeziku obrazuju od slogova. Slog se sastoji iz sledeća 4 polja: IS, VALUE, CAR, CDR, koja su članovi odgovarajućih nizova. Po prirodi problema polje IS je logičkog tipa, a polja VALUE, CAR i CDR celobrojnog tipa.

Polje IS određuje da li je u pitanju molekul, simbolički ili numerički atom(odgovarajuće vrednosti su 'C', 'S', 'N'). Vrednost polja IS

se postavlja pozivom konstruktorskih funkcija CONS, SYMBOL ili NUMBER zavisno od tipa S-izraza. Utvrđivanje tipa S-izraza se vrši pomoću funkcija ISCONS, ISSYMBOL i ISNUMBER.

Ako je S-izraz numerički atom, njegova vrednost se upisuje u polje VALUE. Ako je S-izraz simbolički atom, u polje VALUE upisuje se redni broj reda tablice simboličkih atoma koji sadrži dati simbolički atom.

Ako je S-izraz molekul oblika s1.s2,u polje CAR upisuje se vrednost pokazivača strukture podataka koja reprezentuje S-izraz s1 (tj. redni broj odgovarajućeg člana niza), a u polje CDR upisuje se vrednost pokazivača strukture podataka koja reprezentuje drugi S-izraz s2.

Polja VALUE i CAR mogu se zameniti jednim poljem.

Zbog kasnijeg dinamičkog upravljanja memorijom ([4]), podesno je sav dodeljeni memorijski prostor organizovati u obliku liste(molekula) slogova nedefinisanog tipa, tj. liste slobodnog memorijskog prostora. Obrazovanje i inicijalizaciju takve liste vrši podprogram INITSTORE.

Funkcija COMPILE može se definisati na sledeći način:

```
FUNCTION COMPILE(E)
:
APSTOP=CONSN(26,NIL)
APSTOP=CONSN(4,APSTOP)
COMPILE=COMP(E,NIL,APSTOP)
RETURN
END
```

Promenljiva APSTOP u funkciji dobija vrednost (4 26). Funkcija CONSN(I,J) na prvo mesto molekula J ubacuje numerički atom I.

```
FUNCTION CONSN(I,J)
:
CONSN=CONS()
CAR(CONSN)=NUMBER()
VALUE(CAR(CONSN))=I
CDR(CONSN)=J
RETURN
END
```

U zavisnosti od vrednosti e i (CAR e) funkcija COMP(E,N,C) je podeljena na 7 delova.

Prvi deo je prevod ako je e atom i može se zapisati u obliku:

```
70    IF(IS(E).EQ.SLOVOC) GOTO 1
      COMP=CONS()
      CAR(COMP)=LOCATION(E,N)
      CDR(COMP)=C
      COMP=CONSN(1,COMP)
      GOTO 100
```

Potprogrami STORE i LOAD redom vrše upis i čitanje simboličkog atoma iz tablice atoma.

Drugi deo je prevod funkcije (QUOTE E) i može se zapisati u obliku:

```
2     COMP=CONS()
      CAR(COMP)=CAR(CDR(E))
      CDR(COMP)=C
      COMP=CONSN(2,COMP)
      GOTO 100
```

U preostalim delovima odgovarajući prevodi koriste rekurziju. Zbog toga je uveden stek u obliku celobrojne matrice STACK(I,L) ($1 \le I \le 4$). Upis vrednosti u stek vrši podprogram UPISUSTEK:

```
      SUBROUTINE UPISUSTEK(K)
      :
      :
      L=L+1
      STACK(1,L)=E
      STACK(2,L)=N
      STACK(3,L)=C
      STACK(4,L)=K
      RETURN
      END
```

L-ta kolona steka sadrži informacije o trenutnom stanju S-izraza E,N,C pre početka prevođenja nekog dela početne funkcije, i informaciju o tome kako nastaviti rad programa po završetku prevoda posmatranog dela. STACK(1,L) STACK(2,L) i STACK(3,L) imaju redom vrednost E,N,C, a STACK(4,L) ima vrednost koja određuje obeležje naredbe od koje program nastavlja rad

Po završetku prevoda nekog dela funkcije iz steka se uzima poslednja (L-ta) kolona i iz nje vrednosti E, N, C i informacija o obeležju naredbe kojom se nastavlja rad programa. Uzimanjem L-te kolone iz steka ona se briše iz steka smanjenjem L za jedan. Odgovarajući deo u programu izgleda ovako:

```
100   IF(L.EQ.0) RETURN
      E=STACK(1,L)
      N=STACK(2,L)
      C=STACK(3,L)
      L=L-1
      GOTO(11,100,33,99,55,69,77,88,99,88,119,
           129,11) STACK(4,L+1)
```

Program završava rad kad se iz steka ne mogu uzeti odgovarajuće vrednosti, tj. kada je stek prazan (L=0).

Treći deo programa je prevod aritmetičkih i logičkih funkcija i primitivnih funkcija CAR, CDR, CONS. Dokadni kod funkcije koja se prevodi je označen sa KOLIKO. U slučaju funkcije ADD rezultat rada je:

```
comp((CAR(CDR E)),N,comp((CAR(CDR(CDR E))),
     N,(CONS(QUOTE 15) C)))  .
```

Slično vredi za sve binarne aritmetičke i logičke funkcije.

Prevod funkcije CONS se jedino razlikuje u tome što u gornjem izrazu prvi argumenti funkcija COMP zamenjuju mesta.

U slučaju funkcija CAR, CDR i ATOM rezultat rada prevodioca je:

```
comp((CAR(CDR E)),N,(CONS(QUOTE X)C)),
     X∈{10,11,12}  .
```

Odgovarajući program je:

```
3     CALL UPISUSTEK(1)
      IF(KOLIKO.LE.12) STACK(4,L)=2
      IF(KOLIKO.EQ.13) STACK(4,L)=13
      IF(KOLIKO.LE.13)E=CAR(CDR(E))
      IF(KOLIKO.GT.13)E=CAR(CDR(CDR(E)))
      C=CONSN(KOLIKO,C)
      GOTO 70
11    K=STACK(4,L+1)
```

```
      CALL UPISUSTEK(2)
      IF(K.EQ.13) E=CAR(CDR(CDR(E)))
      IF(K.NE.13) E=CAR(CDR(E))
      C=COMP
      GOTO 70
```

70 je obeležje naredbe kojom počinje izvršavanje rada funkcije COMP(E,N,C).

Četvrti deo programa je prevod funkcije IF. Rezultat rada je:

```
comp((CAR(CDR E)),N,(CONS(QUOTE 8)(CONS(comp
     (CAR(CDR(CDR E))),N,(CONS(QUOTE 9)NIL))
     (CONS(comp(CAR(CDR(CDR(CDR E)))),N,
     (CONS(QUOTE 9)NIL)) C)))) .
```

Odgovarajući deo programa može se napisati u obliku:

```
7     CALL UPISUSTEK(12)
      E=CAR(CDR(CDR(CDR(E))))
      C=CONSN(9,NIL)
      GOTO 70
129   ELSEPT=COMP
      CALL UPISUSTEK(6)
      E=CAR(CDR(CDR(E)))
      C=CONSN(9,NIL)
      GOTO 70
69    CC=CONS()
      CAR(CC)=COMP
      CDR(CC)=CONS()
      CAR(CDR(CC))=ELSEPT
      CDR(CDR(CC))=C
      CALL UPISUSTEK(2)
      E=CAR(CDR(E))
      C=CONSN(8,CC)
      GOTO 70
```

U preostalim delovima programa koristi se pomoćna funkcija

$$complis(e,n,c)=(LDC\ NIL)|e_k*n|(CONS)|\ldots\ldots|e_1*n|(CONS)|c$$

koja je realizovana u okviru funkcije comp bez posebnog potprograma. complis je realizovan na sledeći način:

```
71    IF(E.NE.NIL) GOTO 10
      COMPLIS=CONS()
      CAR(COMPLIS)=NIL
      CDR(COMPLIS)=C
      COMPLIS=CONSN(2,COMPLIS)
      GOTO 100
10    CALL UPISUSTEK(5)
      E=CAR(E)
      C=CONSN(13,C)
      GOTO 70
55    CALL UPISUSTEK(2)
      E=CDR(E)
      C=COMP
      GOTO 71
```

Peti deo programa je prevod funkcije LAMBDA, i može se realizovati na sledeći način:

```
4     CC=CONS()
      CAR(CC)=CAR(CDR(E))
      CDR(CC)=N
      CALL UPISUSTEK(7)
      E=CAR(CDR(CDR(E)))
      N=CC
      C=CONSN(5,NIL)
      GOTO 70
77    CC=CONS()
      CAR(CC)=COMP
      CDR(CC)=C
      COMP=CONSN(3,CC)
      GOTO 100
```

Rezultat rada ovog dela programa je prevod funkcije LAMBDA:

```
(CONS(QUOTE 3)(CONS(comp(CAR(CDR(CDR E)) ,
 (CONS(CAR(CDR E))N),(CONS(QUOTE 5)NIL)))C)) .
```

Šesti deo programa je prevod funkcija LET i LETREC. U slučaju funkcije LET program treba da izvrši:

```
complis(exprs((CAR(CDR(CDR E)))),N,(CONS
 (QUOTE 3)(CONS comp((CAR(CDR E)),(CONS
 vars(CDR(CDR E)),N),(CONS(QUOTE 5)NIL))
 (CONS(QUOTE 4) C))))  .
```

U slučaju funkcije LETREC program treba da izvrši:

```
(CONS(QUOTE 6) complis(exprs(CDR(CDR E)),
 (CONS vars(CDR(CDR E)) N),(CONS(QUOTE 3)
 (CONS comp((CAR(CDR E)),(CONS vars(CDR(CDR E))
 N),(CONS(QUOTE 5)NIL))(CONS(QUOTE 7)C))))) .
```

Funkcije vars i exprs izdvajaju promenljive i izraze iz definicija, tj. $vars((X_1.e_1)\ldots(X_k.e_k))=(X_1\ \ldots\ X_k)$, $exprs((X_1.e_1)\ldots(X_k.e_k))=(e_1\ \ldots\ e_k)$. Obe funkcije su realizovane pomoću podprograma VARS(S,EXPRS) za koji važi: vars(S)=VARS(S,.FALSE.) i exprs(S)=VARS(S,.TRUE.) .

Program koji realizuje navedene prevode može se napisati u obliku:

```
5     M=CONS()
      CAR(M)=VARS(CDR(CDR(E)),.FALSE.)
      CDR(M)=N
      IF(LET) CALL UPISUSTEK(8)
      IF(LETREC) CALL UPISUSTEK(10)
      E=CAR(CDR(E))
      N=M
      C=CONSN(5,NIL)
      GOTO 70
88    CALL UPISUSTEK(STACK(4,L+1)+1)
      CC=CONSN(4,C)
      IF(STACK(4,L).EQ.11) CC=CONSN(7,C)
      C=CONS()
      CAR(C)=COMP
      CDR(C)=CC
      C=CONSN(3,C)
      IF(STACK(4,L).EQ.11)N=M
      E=VARS(CDR(CDR(E)),.TRUE.)
      GOTO 71
99    COMP=COMPLIS
      GOTO 100
119   COMP=CONSN(6,COMPLIS)
      GOTO 100
```

U sedmom delu programa se obrađuje poziv funkcije. Program treba da izvrši:

```
complis((CDR E),N, comp((CAR E),N,(CONS
 (QUOTE 4) C))) .
```

Sledeći deo programa to izvršava:

```
40    CALL UPISUSTEK(3)
      E=CAR(E)
      C=CONSN(4,C)
      GOTO 70
33    CALL UPISUSTEK(4)
      E=CDR(E)
      C=COMP
      GOTO 71
```

5. Pomoćni podprogrami koje prevodilac koristi

Opišimo podprograme koji nisu dosad pominjani.

Potprogram GETEXP(X) obrazuje strukturu podataka- S-izraz koji odgovara datom programu napisanom na mašinskom jeziku SECD-mašine, odnosno datim ulaznim podacima. X je pokazivač

na obrazovani S-izraz. Podprogram GETEXP koristi, pored nekih već pomenutih podprograma, podprograme GETCHAR(čita tekući karakter iz teke), GETTOKEN (obrazuje atom,.,(,) ili kraj teke), TOSTRING i TOINTEGER (transformiše brojevne podatke u azbučne i obratno), CONC (dopisuje znak u atom koji se formira), ASS (određuje ulaznu i izlaznu jedinicu programa i ulaznog podatka), ISLETTER, ISDIGIT, ISCHARACTER, ISDELIMITER koji utvrđuju tip datog znaka.

Podprogram PUTEXP(Y) izdaje na terminalu i/ili nekoj drugoj izlaznoj jedinici S-izraz na osnovu njegove unutrašnje reprezentacije. Pored nekih već navedenih podprograma, PUTEXP koristi i podprograme PUTTOKEN (upisuje atom, .,(,) ili kraj teke u izlazni red), PUTCHAR (upisuje znak u izlazni red) i PUTLINE (izdaje poslednji- nepotpunu red S-izraza).

Podprogrami COLGAR, COLLECT i MARK vrše skupljanje otpadaka, tj. obrazovanje liste slobodnog memorijskog prostora od slogova koji su slobodni za ponovno korišćenje. Skupljanje otpadaka se vrši tokom rada podprograma COMPILE i EXECUTE, kada je lista slobodnog memorijskog prostora iscrpljena ([4]). U toku rada podprograma GETEXP skupljanje otpadaka nije dozvoljeno.

Podprogram LOCATION koristi potprogram MEMBER(X,S) (ispituje da li je X član molekula S i ako je član određuje mu poziciju).

6. Primer interpretacije

Glavni program LISP-prevodioca može se napisati u obliku:

```
      PROGRAM LISP
      .
      CALL INITSTORE
      .
      CALL GETEXP(ARGS)
      .
      FN=COMPILE(ARGS)
      .
      CALL PUTEXP(FN)
      .
      CALL INITSTORE
      .
      CALL GETEXP(FN)
      .
      CALL GETEXP(ARGS)
      .
      CALL EXECUTE(FN,ARGS,RESULT)
      .
      CALL PUTEXP(RESULT)
      END
```

Neka se funkcija append nalazi u datoteci APPEND.LSP. Program se izvršava na sledeći način:

```
>RUN LISP
INPUT FILE NAME = APPEND.LSP ↵
( 6 2 NIL 3 ( 1 ( 0 . 0 ) 2 NIL 14 8 ( 1 ( 0
. 1 ) 9 ) ( 2 NIL 1 ( 0 . 1 ) 13 1 ( 0 . 0 )
11 13 1 ( 1 . 0 ) 4 1 ( 0 . 0 ) 10 13 9 ) 5 )
13 3 ( 1 ( 0 . 0 ) 5 ) 7 4 21 )
INPUT FILE NAME = ↵
?
(( 1 2 3)(1 (2) 3))   ↑Z
OUTPUT FILE NAME = ↵
( 1 2 3 1 ( 2 ) 3 )
```

Poruke programera LISP-interpretatoru su podvučene.

7. Zaključak

Programski komplet se sastoji iz glavnog programa i 35 pomoćnih podprograma- sve skupa oko 1ØØØ naredbi FORTRAN-jezika.

Zahvaljujući mašinskoj orjentaciji FORTRAN-jezika izvršna(task) verzija programa efikasno koristi tehničke mogućnosti računara.

Zbog široke rasprostranjenosti FORTRAN-jezika programski komplet je dostupan velikom broju korisnika.

Prevođenje programa sa LISPKIT LISP-jezika na jezik SECD-mašine pomoću prevodioca implementiranog na FORTRAN-jeziku se izvršava brže nego pomoću prevodioca na jeziku SECD-mašine.

L i t e r a t u r a

[1] Boley H., Artificial Intelligence, Language and Machine, Bericht Nr. 94, Universität Hamburg, 1982.

[2] Henderson P., Functional Programming; Application and Implementation, Prentice-Hall, London, 1980.

[3] McCarthy J., Recursive Functions of Symbolic Expressions and Their Computation by Machine, Comm. ACM, 3(4), 1960,184-195.

[4] Stojković V., Stojmenović I., Jerinić Lj., Mirčevski J., Kulaš M., Dinamičko upravljanje memorijom sa gledišta korišćenja i implementacije programskih jezika, V Međunarodni simpozij "Kompjuter na sveučilištu", Cavtat, 1983, str 135-142.

[5] Stojković V., Jerinić Lj., Stojmenović I., Mirčevski J., Kulaš M., Implementacija NP-tehnike obrade S-izraza u FORTRAN-jeziku, V Međunarodni simpozij "Kompjuter na sveučilištu",Cavtat, 1983, str 379-386.

[6] Stojković V., Stojmenović I., Mirčevski J., Jerinić Lj., Kulaš M., Programska implementacija simulatora SECD-mašine, XXVII Konferencija ETAN-a, Struga, 1983.

[7] Stojković V., Jerinić Lj., Mirčevski J., Stojmenović I., Budimac Z., Putnik Z., O programskom jeziku za obradu lista LISPKIT LISP i njegovoj primeni, Bilten Pokrajinskog saveta za informatiku, Novi Sad (u pripremi).

INFORMATICA 1/1984

ANALIZA MULTIPROCESORSKIH SISTEMA S LOKALNOM CACHE MEMORIJOM

BEKIĆ ZORAN

SVEUČILIŠNI RAČUNSKI CENTAR, ZAGREB

UDK: 681.324:681.327.67

U radu se govori o osnovnim parametrima arhitekture i organizacije multiprocesorskih sistema s lokalnom cache memorijom. Ukazuje se na neke važne probleme koji postoje u analizi takvih računarskih sistema. Posebno se govori o utjecaju ponašanja programa na efektivne performanse takvih računarskih sistema i o mogućnostima modeliranja ponašanja programa. Ukazuje se i na problem koherentnosti informacija u multiprocesorskim sistemima s lokalnom cache memorijom.

AMALYSIS OF MULTIPROCESSORS WITH PRIVATE CACHE MEMORIES. Major parameters of architecture of multiprocessor system with private cache memories are considered. Some problems of such multiprocessor architecture analysis are discussed. Program behavior influence on effective performances of such computer systems and possibilities of its modeling are especially considered. Cache coherency problem in multiprocessor systems with private cache memories is also mentioned.

UVOD

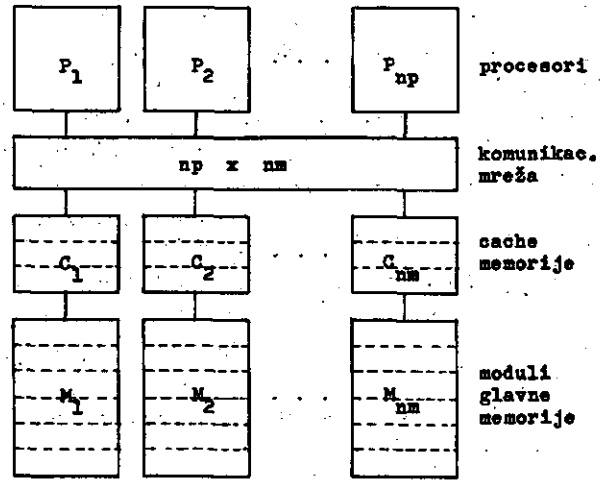
Paralelan rad nekolicine procesorskih jedinica osnovna je, dakako, karakteristika multiprocesorskih sistema. Uvođenjem većeg broja procesorskih jedinica povećava se nominalna procesorska moć računarskog sistema. Efektivne performanse multiprocesorskog sistema najčešće se, međutim, razlikuju od nominalnih, pošto ne zavise samo od broja procesorskih jedinica u sistemu, nego i od drugih arhitekturnih osobina a također i od tipa programa koji se rješavaju na sistemu. Tako se brzina rasta efektivnih performansi mijenja od logaritamske zavisnosti od broja procesorskih jedinica (odnosno od nominalnih performansi) pa do rijetkih pojedinačnih slučajeva kada su efektivne performanse veće od nominalnih.

Jedan od osnovnih uzroka degradacije efektivnih performansi multiprocesorskih sistema je neadekvatna arhitektura memorije. Naime, povećanje procesorske moći sistema dovodi do povećanih zahtjeva na memoriju i njene karakteristike.

Osim povećanja apsolutnog broja dohvata iz memorije (što i nije glavni problem, a pogotovo nije specifičan problem samo za multiprocesorsku arhitekturu memorije) pojavljuju se i posebni problemi, kao na primjer, mnogobrojna istovremena obraćanja u memoriju, izazvana sinhronizacijom rada procesora ili postojanjem zajedničkog područja podataka. Istovremeno obraćanje u memoriju većeg broja procesora obično izaziva zaustavljanje nekolicine njih, što očigledno dovodi do degradacije efektivnih performansi sistema.

Da bi se poboljšale karakteristike arhitekture memorije u multiprocesorskim sistemima realiziraju se mnoge ideje, koje se primjenjuju i u klasičnim računarskim sistemima. Memorija se dijeli na nezavisne module, što omogućava da kao cjelina istovremeno odgovara na nekoliko zahtjeva za dohvat iz memorije (dakako pod uslovom da ti zahtjevi budu upućeni različitim modulima memorije). Realizira se i ideja raslojavanja memorije na nivoe, koji se razlikuju srednjom brzinom odgovora na zahtjev procesora i najčešće svojim kapacitetom.

Uvođenje cache memorije u multiprocesorske sisteme moguće je na dva principijelno različita načina.

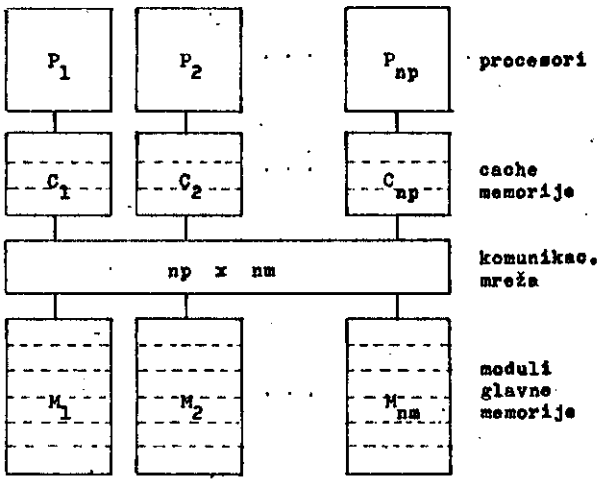


Slika 1. Konfiguracija sa zajedničkim cache memorij.

Vezivanjem cache memorije za određenu oblast glavne memorije, tj. najčešće za pojedini modul glavne memorije, nastaju sistemi sa zajedničkim cache memorijama. U takvoj konfiguraciji cache memorija ima uglavnom istu ulogu kao u klasičnim jednoprocesorskim sistemima i

praktički se može razmatrati kao primarni ili nulti nivo memorijske hijerarhije.

Alternativna mogućnost uvođenja cache memorije sastoji se u pridruživanju cache memorije svakom procesoru sistema, čime nastaju multiprocesorski sistemi s lokalnom cache memorijom. U ovom slučaju, neki autori nisu skloni razmatranju cache memorije kao običnog nivoa memorijske hijerarhije. Između ostalog, objašnjavaju to i time, što se pretpostavlja da je dinamika izmjene porcija informacije u cache memoriji u takvoj konfiguraciji znatno veće nego u ostalim nivoima memorijske hijerarhije. Također se pretpostavlja da je količina informacije koja se predaje u toku jedne izmjene među glavnom i cache memorijom znatno manja nego što je to obično među susjednim nivoima memorijske hijerahije.



Slika 1. Konfiguracija s lokalnim cache memorijama

U svakom slučaju uvođenje lokalne cache memorije osim funkcije buferizacije toka informacije među procesorima i modulima glavne memorije ima cilj iskorištavanja tendencije lokalizma u sekvencijalnom nizu obraćanja pojedinog procesora u memoriju. Ta je tendencija u većoj ili manjoj mjeri uvijek prisutna u svakom programskom kodu, a proizlazi iz slijedećih činjenica:

1. Dio obraćanja u memoriju je čitanje instrukcija programskog koda, koje su sekvencijalno razmještene u glavnoj memoriji. Na taj način obraćanje u memoriju zbog čitanja instrukcija je u toku nekog vremenskog intervala lokalizirano na ograničeno potprostranstvo adresa glavne memorije.
2. Kod mnogih grupa algoritama proces obrade je unutar nekog vremenskog intervala koncentriran na ograničenu grupu podataka. Ako je ta grupa podataka smještena unutar nekog zatvorenog potprostranstva adresa, tada će i niz obraćanja u memoriju pokazivati znakove lokalizma.

Najveći efekt od uvođenja cache memorije u multiprocesorski sistem postiže se u teoretski idealnoj situaciji kad u svakom trenutku svaki od procesora ima trenutačno potrebne podatke u svojoj cache memoriji i znači ne biva blokiran zbog situacija istovremenog obraćanja u glavnu memoriju. Koliko će se rad realnog multiprocesorskog sistema s lokalnom cache memorijom približiti toj teoretski idealnoj situaciji zavisi prije svega od algoritma problema koji se na njemu rješava, ali i od određenih parametara arhitekture i organizacije računarskog sistema.

PARAMETRI ARHITEKTURE MULTIPROCESORSKIH SISTEMA S LOKALNOM CACHE MEMORIJOM

Blokom memorije zvat ćemo porciju informacije (količinu mašinskih riječi) koja se predaje u toku jedne izmjene između glavne i cache memorije.

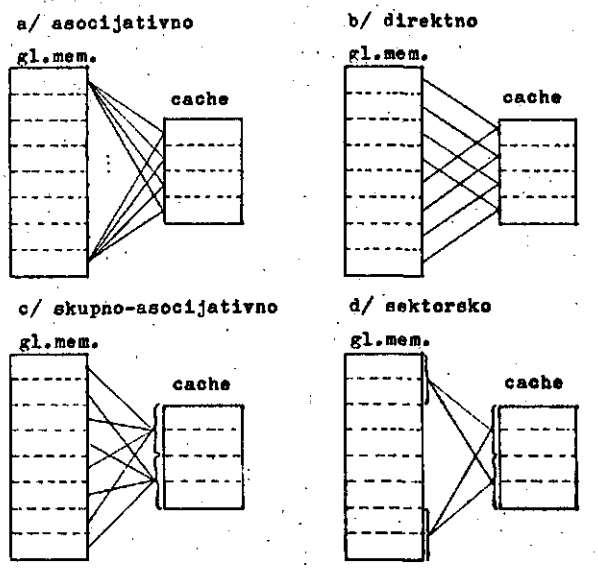
Potpuna analiza utjecaja lokalne cache memorije na efektivne performanse multiprocesorskog sistema treba osim osnovnih parametara multiprocesorskih sistema (broj procesora, broj, veličina i raslojavanje modula glavne memorije) razmatrati i takve specifične parametre arhitekture i organizacije kao što su:

1. Organizacija mapiranja, odnosno algoritam preslikavanja glavne memorije u cache memoriju. Problem mapiranja pojavljuje se uvijek kad je potreban prenos informacije između dva memorijska prostranstva (fizička ili virtualna) različitih kapaciteta. Postoji nekoliko standardnih algoritama preslikavanja glavne memorije u cache memoriju:

   a) asocijativni, koji dozvoljava da svaki blok glavne memorije bude smješten u proizvoljnom bloku cache memorije.
   b) direktni, koji određenom podskupu blokova glavne memorije pridružuje određeni blok cache memorije i predvida da u tom bloku cache memorije može biti istovremeno samo jedan blok pridruženog podskupa glavne memorije.
   c) skupno-asocijativni, koji podskupu blokova glavne memorije pridružuje podskup blokova cache memorije, s tim da svaki blok iz podskupa glavne memorije može zauzeti bilo koji blok pridruženog podskupa blokova cache memorije.
   d) sektorski, koji predvida organizaciju transfera između glavne i cache memorije sektorima (grupama blokova) s tim da raspored blokova unutar sektora bude unaprijed fiksiran.

Asocijativno preslikavanje daje najveću slobodu izbora mjesta u cache memoriji za novi blok glavne memorije, a time smanjuje broj povratnih transfera blokova iz cache memorije u glavnu memoriju. Međutim, takva organizacija zahtijeva kod svakog obraćanja u memoriju asocijativno (istovremeno) pretraživanje cache memorije, što, zavisno od njene veličine, može oduzeti mnogo vremena, a

svakako povećava cijenu njene sklopovske izvedbe.



Sl. 3. Načini preslikavanja glavne memorije u cache

Suprotno asocijativnom, direktno preslikavanje minimizira vrijeme provjere prisutnosti bloka u cache memoriji, pošto je fiksno predodređeno mjesto gdje se blok može nalaziti. Nedostatak je ovog algoritma što predodređeno vezivanje blokova glavne memorije za blokove cache memorije može dovesti do neracionalnog iskorištavanja pojedinih dijelova cache memorije.

Skupno-asocijativni algoritam preslikavanja ujedinjuje prednosti prvih dvaju algoritama, dajući relativnu slobodu izbora mjesta u cache memoriji, istovremeno ograničavajući vrijeme pretraživanja. Sektorsko preslikavanje također teži istom cilju kao i skupno-asocijativno, omogućavajući izbor smještaja sektora u cache memoriji, ali istovremeno ograničavajući mogući broj varijanti fiksiranim rasporedom blokova unutar sektora.

2. Organizacija zamjenjivanja, odnosno algoritam izbora bloka u cache memoriji koji treba biti izbačen iz nje (eventualno vraćen u glavnu memoriju) u trenutku kad procesor zahtijeva transfer novog bloka iz glavne u cache memoriju. Najčešće se upotrebljavaju slijedeće strategije:

a) LRU (last-recently used) - iz cache memorije se izbacuje blok, koji najduže nije bio korišten,
b) FIFO (first-in-first-out) - iz cache memorije se izbacuje blok, koji se najduže u njoj nalazi,
c) slučajni izbor bloka, koji treba biti izbačen iz cache memorije.

Utjecaj izbora algoritma zamjenjivanja na efektivne performanse računarskog sistema zavisi od izbora organizacije mapiranja. Međutim, analize su pokazale da je taj utjecaj uglavnom neznatan. To se može objasniti relativno malom predviđenom veličinom cache memorije, pa sam izbor bloka ne daje veći broj mogućnosti.

3. Organizacija obnavljanja sadržaja glavne memorije, kojom se određuje u kojim se slučajevima sadržaj bloka glavne memorije korigira sadržajem odgovarajućeg bloka u cache memoriji. Postoje dva principijelno različita pristupa:

a) direktno pisanje (WT-write-trough), koje predvida da se istovremeno s pisanjem u blok u cache memoriji, korigira i sadržaj istog bloka u glavnoj memoriji.
b) Razmjena blokova (FS-flag-swaping), koja predvida da se sadržaj bloka u glavnoj memoriji korigira tek pošto blok u cache memoriji bude izabran za zamjenjivanje, odnosno vraćanje u glavnu memoriju.

Iako FS organizacija djeluje povoljnije, pošto u principu manje opterećuje glavnu memoriju, optimalnost prve ili druge strategije potrebno je ocijeniti u cjelini s ostalim parametrima izabrane arhitekture. Naime, FS organizacija pridonosi kompliciranju ionako teškog problema koherentnosti u multiprocesorskim sistemima s lokalnom cache memorijom. S druge strane, činjenica da operacije pisanja čine tek 10-30% svih dohvata iz memorije donekle objašnjava neka opažanja da u određenim situacijama WT organizacija pokazuje bolje rezultate od FS organizacija obnavljanja sadržaja glavne memorije. Tako Yen i Fu (7) analitički pokazuju da FS organizacija daje znatno bolje rezultate ako je vjerojatnost uspjeha kod obraćanja u cache memoriju (hit ratio) veća od 0.85, dok kod vjerojatnosti manje od 0.5 nešto bolje rezultate pokazuje WT organizacija (autori su dakako u svojoj analizi imali i neka druga ograničenja-pretpostavke o ponašanju programa).

4. Organizacija dohvata bloka iz glavne memorije u cache memoriju, koja određuje kada i kakva informacija se prenosi iz glavne u cache memoriju.

Strategija direktnog punjenja cache memorije (load-through), predvida da se u cache memoriju prije svega transportira onaj dio bloka, koji sadrži riječ, zbog koje je i došlo do obraćanja u glavnu memoriju. Na taj se način omogućuje da procesor nastavi rad paralelno s transferom ostatka bloka iz glavne u cache memoriju.

Ako se koristi WT organizacija obnavljanja sadržaja glavne memorije, moguće je koristiti strategiju dohvata koja ignorira operacije pisanja, tj. ne inicijalizira transfer iz glavne u cache memoriju, ako je neuspješno obraćanje u memoriju izazvala operacija pisanja. Implementiranje takve strategije u WT organizaciji obnavljanja doprinosi smanjenju ukupnog prometa među glavnom i cache memorijom.

5. Homogenost cache memorije. Moguće je uvođenje funkcionalne specijalizacije područja unutar cache memorije. Tako se može uvesti podjela cache memorije na prostranstvo programskog koda i prostranstvo podataka ili na prostranstvo sistemskog koda i prostranstvo korisničkog koda. Podjela na prostranstvo koda i prostranstvo podataka osigurava da blokovi s programskim kodom ne budu prerano izbacivani iz cache memorije zbog vrlo dinamičkog korištenja podataka. Takva podjela međutim može uzrokovati i neracionalno iskorištavanje cache memorije.

Kaplan i Winder (3) su pokazali da podjela na prostranstva sistemskog i korisničkog koda ima veliko značenje kod izvršavanja korisničkih programa organiziranih kao niz kratkih jednostavnih aktivnosti, kad često dolazi do prekljućivanja procesora s jedne aktivnosti na drugu.

6. Veličina cache memorije svakako je jedan od osnovnih parametara analize multiprocesorskih sistema s lokalnom cache memorijom. Obično se veličina cache memorije kreće od 1-16 K. Njenu veličinu potrebno je u svakom konkretnom slučaju pažljivo odrediti kao kompromis među obično visokom cijenom cache memorije i težnjom da se minimizira srednje vrijeme dohvata iz memorije.

7. Veličina bloka memorije. Kako pokazuju mnoge analize (3, 7, 8) veličina bloka memorije jedan je od najvažnijih parametara arhitekture multiprocesorskih sistema s lokalnom cache memorijom. Izbor veličine bloka memorije (kod fiksirane izabrane veličine cache memorije) mora biti izvršen pažljivim balansiranjem suprotnih tendencija izmjene efektivnih performansi sistema u zavisnosti od veličine bloka. Naime, povećanje veličine bloka u početku može znatno poboljšati performanse sistema, pošto se povećava količina informacije koja se prenosi iz glavne u cache memoriju u toku jednog transfera. Međutim, nakon neke kritične točke povećanje veličine bloka ili ne doprinosi poboljšanju performansi ili utječe na njih negativno. To se može objasniti mnogim razlozima: opadanjem korisnosti informacije u bloku s rastom njegove veličine, rastom količine informacije koja se istiskuje iz cache memorije kod zamjene blokova, rastom vremena potrebnog za transfer bloka iz glavne u cache memoriju, što dovodi do povećanog opterećenja glavne memorije i komunikacionih veza među glavnom i cache memorijama. Pravilan izbor veličine bloka zbog toga je vrlo važan za konačnu veličinu efektivnih performansi. Analize (7) pokazuju da je optimalna veličina bloka obično između 16 i 256 bytaea.

8. Karakteristika linija povezivanja glavne i cache memorije. Očigledno je da brzina transfera blokova među glavnom i cache memorijom znatno utječe na performanse memorijskog sistema u cjelini. Ta brzina osim od mogućnosti same glavne memorije zavisi i od sklopovskih performansi linija povezivanja, konfiguracije tih linija, kao i od njihove širine, odnosno količine informacije, koja se može istovremeno slati kroz jednu liniju. Poboljšanje navedenih karakteristika memorijskog sistema može bitno utjecati na efektivne performanse računarskog sistema, ali i na njegovu cijenu, što znači da kod projektiranja arhitekture treba tražiti optimalni kompromis.

PONAŠANJE PROGRAMA I NJEGOVO MODELIRANJE

Velik broj autora (2, 5, 6, 7, 8) naglašava prvostepeni utjecaj ponašanja programa na efektivne performanse računarskog sistema s lokalnom cache memorijom. U analizi takvih računarskih sistema pod terminom "ponašanje programa" podrazumjeva se (kao i kod analiziranja stranične segmentske organizacije memorije) karakteristika niza zahtjeva za dohvat iz memorije u dinamici izvršavanja programskog koda. To ponašanje ima tri slijedeća aspekta:

1. vremenski, tj. vremensku učestalost zahtjeva za dohvat iz memorije,
2. redosljedni, tj. redoslijed kojim se pojedine lokacije (blokovi, moduli) glavne memorije koriste,
3. logički, tj. logičku zavisnost među pojedinim zahtjevima za dohvat iz memorije.

Jasno je da ponašanje realnih programa zavisi od mnogobrojnih faktora: počevši od algoritma problema, koji se rješava tim programom, pa do osobina upotrijebljenih kompajlera ili čak iskustva programera.

Vremenski aspekt ponašanja programa u računarskom sistemu s režimom multiprogramiranja može se mijenjati od jednog izvršavanja do drugog u zavisnosti od stanja i opterećenja sistema. U multiprocesorskim sistemima opterećenost i stanje sistema utječe, osim na vremenski, i na logički aspekt ponašanja svakog programa.

Sve navedeno govori da je ponašanje programa ne samo jedan od najvažnijih objekata analize multiprocesorskih sistema s lokalnom cache memorijom, nego i objekt koji je nemoguće formalno parametrizirati u općem slučaju. Pošto, međutim, nije moguća nikakva analiza efektivnih performansi bez nekih osnovnih pretpostavki o ponašanju programa, a to znači i o stupnju lokalizma u dohvatima memorije, razrađeni su neki modeli, koji (ovisno o cilju) analiziranja) više ili manje uspješno simuliraju ponašanje realnih programa.

Kod modeliranja frekventnosti zahtjeva za dohvat iz memorije obično se koristi jedna od dvije moguće varijante:

a) procesor u toku jediničnog vremenskog intervala inicijalizira novi dohvat iz memorije s nekom zadanom vjerojatnošću ili po nekom zadanom statističkom zakonu.
b) procesor inicijalizira novi dohvat, svaki put kad nije blokiran nekom drugom, prethodno započetom, aktivnošću.

Redoslijed dohvata iz memorije najinteresantniji je aspekt ponašanja programa, pošto direktno iskazuje lokalizam u programskom kodu. Jasno je da je zbog toga baš za

taj aspekt ponašanja programa razraden veći broj modela, koji mogu imitirati redoslijed dohvata u realnim programima. Među njima su i slijedeći:

a) IRM (independent reference model) koji predvida da svaki blok memorije ima fiksiranu (možda ne uniformnu) vjerojatnost dohvata. U nekim slučajevima koristi se čak i RIRM (random IRM) model, koji predvida jednaku vjerojatnost dohvata iz svakog bloka.

b) WSM (working set model) koji predvida postojanje diskretne vjerojatnostne funkcije vremena f(I), gdje je f(I) - dio već izvršenih instrukcija, za koje je dohvat iste memorijske lokacije (u našem slučaju bloka) bio prije I jedinica vremena. Drugim riječima, ako je zadnji dohvat iz nekog bloka bio u vremenu t_1 tada je vjerojatnost dohvata iz tog bloka u vremenu t_2 jednaka $f(t_2-t_1)$.

c) LRUSM (last recently used stack model) je osnovan na ideji da se blokovi nalaze u stogu, s tim da se na vrhu stoga nalazi zadnji upotrebljeni blok, a na dnu blok koji najduže nije bio korišten. Svaki put kad se vrši dohvat iz nekog bloka, taj blok dolazi na vrh stoga. LRUSM model pretpostavlja da je vjerojatnost dohvata iz nekog bloka jednaka p(I), gdje je I-trenutačno mjesto bloka u stogu. Vjerojatnostna funkcija p(I) nezavisna je od svih ostalih faktora osim pozicije bloka u stogu.

Osim navedenih modela ponekad se koriste i drugi, koji se međusobno razlikuju, uostalom kao i navedena tri modela, kako stupnjem apstraktnosti, detaljizacije i točnosti imitiranja ponašanja realnih programa, tako i potrebnim vremenom i statističkim podacima za njihovo implementiranje.

Logička zavisnost među pojedinim zahtjevima za dohvat iz memorije postaje predmetom razmatranja u analizama multiprocesorskih sistema, koje dopuštaju da procesori imaju istovremeno nekoliko nedovršenih zahtjeva za dohvat iz memorije.

Raznovrsnost ali istovremeno i nepotpunost i parcijalnost pojedinih metoda modeliranja ponašanja programa potvrđuju da još ne postoji zadovoljavajući pristup tom važnom aspektu analize multiprocesorskih sistema s lokalnom cache memorijom. Svaki od postojećih modela je zasnovan na određenim pretpostavkama koje olakšavaju analizu performansi sistema, ali koje se ponekad i znatno udaljuju od realnih situacija (kao što je, na primjer, slučaj s RIRM modelom (6)).

S druge strane, praktički sve analize koje bar donekle uključuju i utjecaj ponašanja programa, pokazuju da ponašanje programa ima presudan utjecaj na efektivne performanse računarskog sistema. Sve to govori da bi kod analize efektivnih performansi računarskih sistema s lokalnom cache memorijom bilo potrebno puno pažljivije razmatrati taj problem.

PROBLEM KOHERENTNOSTI U MULTILROCESORSKIM SISTEMIMA S LOKALNOM CACHE MEMORIJOM

Govoreći o analizi multiprocesorskih sistema s lokalnom cache memorijom nemoguće je ne spomenuti jedan važan problem, specifičan za arhitekturu s lokalnim memorijama. Radi se o problemu koherentnosti, odnosno jednoznačnosti informacije. U multiprocesorskim sistemima taj se problem pojavljuje zbog slijedećih razloga:

a) postojanje zajedničkih područja podataka za nekolicinu procesora, odnosno različitih kopija tog područja u cache memorijama različitih procesora,

b) migracije procesa s jednog procesora na drugi, dakle mogućnosti da kod aktivizacije nekog procesa sadržaj pripadajuće cache memorije ne odgovara stanju u trenutku blokiranja tog procesa.

Moguća su dva pristupa u rješavanju problema koherentnosti informacije.

Statičko rješenje predlaže diferencijaciju blokova memorije na one koji mogu biti transportirani u lokalne cache memorije i one koji to ne mogu biti. Među posljednje odnose se i blokovi sa zajedničkim podacima. Na taj način statičko rješenje onemogućava postojanje različitih kopija kritičnih blokova. Ako je, međutim, broj kritičnih blokova velik ili se oni često koriste tada glavno opterećenje opet prelazi s cache na glavnu memoriju.

Dinamički pristup predvida međusobno spajanje cache memorije i primjenu direktnog pisanja (WT) u organizaciji obnavljanja sadržaja glavne memorije. Osim glavne memorije obnavlja se i sadržaj izmjenjenog bloka u svim cache memorijama, u kojima je on u tom trenutku prisutan. Glavni nedostatak takvog pristupa je u kompliciranosti njegove sklopovske izvedbe, koja je ili prespora (spoj preko zajedničkog kanala) ili preglomazna (međusobno spajanje svih cache memorija jednosmjernim kanalima). Osim toga, procesor završava operaciju pisanja tek nakon što dobije povratni signal od svih ostalih procesora (cache memorije) da je korekcija bloka izvršena.

Količina nepotrebnih razmjena signala između cache memorija može se bitno smanjiti uvođenjem tablice blokova glavne memorije i lokalnih tablica stanja cache memorija. Takav pristup rješavanju problema koherentnosti čini se zasad najrazumnijim, iako je jasno da i on ima određenih nedostataka (2):

1. iako znatno smanjen, ostaje problem povišenog intenziteta komunikacija između procesora (odnosno njihovih cache memorija),
2. tablica stanja blokova glavne memorije postaju kritičnim resursom sistema, koji se neprekidno intenzivno koristi, pa kao takva može postati značajnim "uskim grlom" sistema. Zbog toga bi bilo nužno smjestiti ih u posebno brzu memoriju ili pokušati riješiti

taj problem kako to predlažu Dubois i Briggs: raspoređivanjem tablice stanja blokova po modulima glavne memorije, tako da se podaci koji se odnose na blokove nekog modula nalaze u dijelu tablice smještenom u tom istom modulu.

3. broj radnji potrebnih za izvršenje operacije dohvata iz memorije postaje vrlo velik (sl. 4) i znači zahtijeva iznimne mogućnosti kontrolne jedinice sistema i kontrolnih jedinica memorijskih sklopova.

Analizirajući utjecaj problema koherentnosti na efektivne performanse multiprocesorskog sistema s lokalnom cache memorijom Dubois i Briggs dolaze do zaključka da je kod izvršavanja procesa, od kojih svaki ima bar 10% dohvata iz zajedničkog područja podataka, problem koherentnosti glavni razlog degradacije efektivnih performansi sistema. Iako su autori u svojoj analizi krenuli od nekih restriktivnih pretpostavki, što oduzima značaj opće zakonomjernosti dobivenim rezultatima, ta analiza potvrđuje da problemu koherentnosti treba posvetiti meksimalnu pažnju.

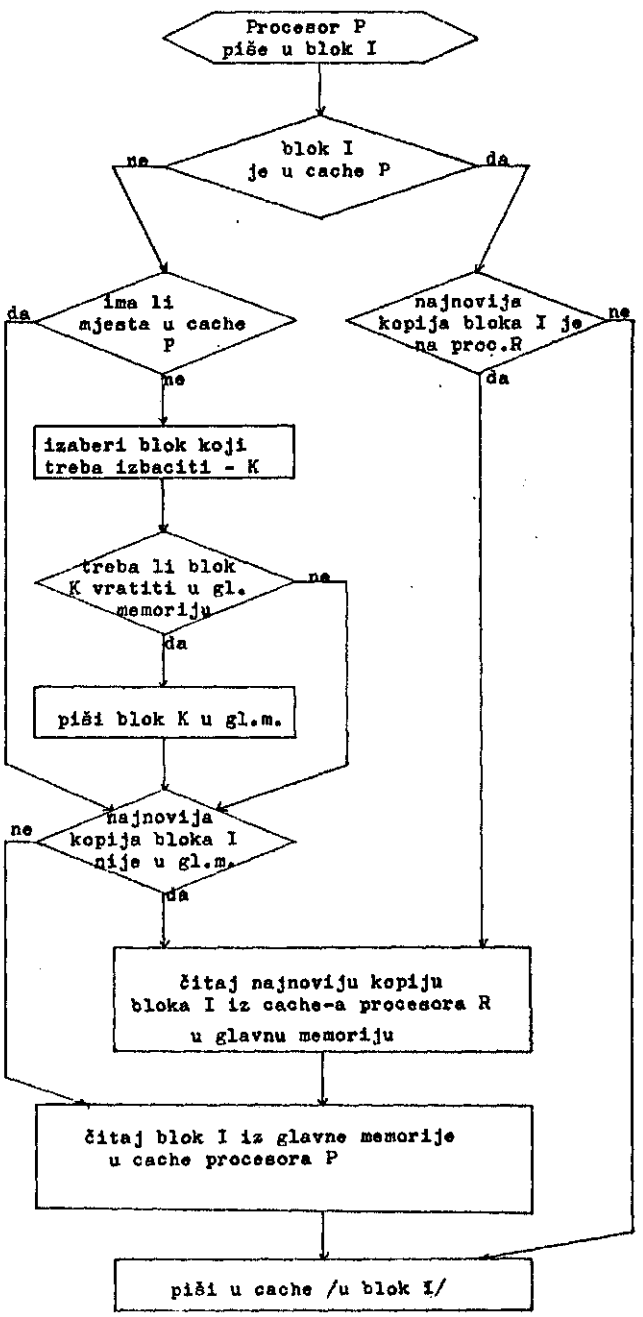
ZAKLJUČAK

Objekt analize multiprocesorskih sistema s lokalnom cache memorijom vrlo je složen i kompleksan. Uvođenje lokalne cache memorije povećava broj organizacionih i arhitekturnih mogućnosti kod projektiranja računarskog sistema. To, jasno, dovodi do povećanja broja parametara koji se razmatraju, što neminovno komplicira analizu predložene arhitekture i efektivnih performansi sistema.

Pošto su mjerenja realnih sistema danas još praktički nemoguća, jedini mogući pristup analizi i evaluaciji efektivnih performansi je modeliranje računarskih sistema s lokalnom cache memorijom. Bez obzira radi li se o analitičkim ili simulacionim modelima, u takve analize potrebno je uključiti sve specifične parametre. Posebnu pažnju potrebno je kod modeliranja i analize posvetiti problemu koherentnosti informacija i utjecaju ponašanja programa (kao i mogućnostima njegovog točnog modeliranja) na efektivne performanse sistema. Nema razloga sumnjati u korisnost uvođenja lokalnih cache memorija u multiprocesorske sisteme, a zadatak je analize njihove moguće arhitekture i organizacije da sugerira optimalne vrijednosti parametara te arhitekture, pri kojima će se postići najpovoljnije efektivne performanse.

LITERATURA:

1. D. P. Bhandarkar, "Analysis of Memory Interference in Multiprocessors"; IEEE Trans. Comput., Vol. C-24, pp. 897-908, Sept., 1975.
2. M. Dubois, A. Briggs, "Effects of Cache Coherency in Multiprocessors", IEEE Trans. Comput., Vol. C-31, pp. 1083-1099, November, 1982.
3. K. R. Kaplan, R. O. Winder, "Cache-Based Computer Systems", IEEE Computer, Vol. 6, March, 1973.
4. J. H. Patel, "Analysis of Multiprocessors with Private Cache Memories" IEEE Trans. Comput., Vol. C-31, pp. 296-304, April, 1982.
5. G. S. Rao, "Performance Analysis of Cache Memories", Journal of the ACM, Vol. 25, No 3, pp. 378-395, July, 1978.
6. B. Ramakrishna Rau, "Program Behavior and the Performance of Interleaved Memories", IEEE Trans. Comput., Vol. C-28, pp. 191-199, March, 79.
7. W. C. Yen, K. S. Fu, "Analysis of Multiprocessor cache Organization with Alternative Main Memory Update Policies", IEEE report, 1981.
8. З. Бекич, "Анализ мультипроцессорных систем с локальной памятью", Московский Государственный Университет, Сборник статьей, 1983 (u štampi).



Slika 4. Jedna moguća varijanta operacije pisanja

Sprememba programa PIP (CP/M 2.2) za večkratno zbirčno kopiranje

```
**********************************
* Informatica UP 11              *
* PIP for Multiple File Transfers *
* september 1983                 *
* modificiral A. P. Železnikar   *
* sistem CP/M 2.2                *
**********************************
```

1. Področje uporabe

PIP (Peripheral Interchange Program) je prehodni ukaz (program) sistema CP/M 2.2 za kopiranje ene ali več zbirk iz enega diska na drugega ali na periferne naprave. PIP ne dovoljuje s preprostim ukazom kopiranja zbirk na več diskov: vselej zahteva po kopiranju izstop iz PIPa, vstavitev novega diska in ponoven klic programa PIP. To je seveda zamuden postopek pri kopiranju zbirk na več diskov.

Postavimo si tole nalogo: modificirajmo PIP tako, da bomo z ukazom v eni vrstici lahko tega ponovili poljubno mnogokrat brez ponovnega pisanja ukaza pri poljubni izmenjavi diskov oziroma disket in seveda brez izstopanja iz PIP in toplega zagona.

Prikazali bomo dve identični modifikaciji PIPa, in sicer z uporabo zbirnika ASM (8080) in ukazov DDT in SAVE ter z uporabo zbirnika ZASM (Z80), povezovalnika LINK in ukazov DDT in SAVE. Tako bomo hkrati spoznali še nekatere mehanizme programiranja v zbirnih jezikih.

2. Kratek opis obeh programov

Za modifikacijo programa PIP moramo poznati določena mesta modifikacije. Modifikacija programa ne bo naslovno strnjena: imeli bomo tri ločene segmente. Z zbirnikom ASM (8080) bo ta problem segmentiranja rešljiv enostavno, ker ta zbirnik generira prevedeno heksadecimalno zbirko, ki natanko upošteva psevdoukaze ORG in tako že natanko segmentira celotno modifikacijo. Oglejmo si najprej ta modifikacijski program na listi 1.

Iz liste 2 je razvidno, da je heksadecimalni prevod natanko to, kar smo želeli. Trije segmenti so določeni s psevdoukazi ORG 100H, ORG 10AH in ORG 96FH v listi 1 (in seveda z psevdoukazom END na koncu programa). Program je dokumentiran, tako da dodatna pojasnila niso potrebna.

V listi 3 je enak program napisan v zbirnem jeziku procesorja Z80, in sicer v treh absolutnih modulih, ker je ZASM premeščevalni zbirnik (ne absolutni kot ASM). Te trije moduli se povežejo v en sam heksadecimalni kos s povezovalnikom LINK, ko imamo npr. operacijo

```
LINK rpip1,rpip2,rpip3
```

Posledica te operacije je heksadecimalna zbirka rpip1.hex, ki jo vidimo na listi 4. Tu je razvidno, da je ta zbirka natanko to, kar smo želeli. Tako bo nadaljna modifikacija programa PIP enostavno izvedljiva z uporabo prehodnega ukaza DDT za oba naša primera.

Ko imamo tako obe heksadecimalni zbirki (PAPIP1.HEX za 8080 in RPIP1.HEX za Z80), lahko izvedemo modifikacijo programa PIP.COM, in sicer tekole:

```
A>ddt pip.com (cr)      ;Pokliče se DDT, ki naloži
                        ; PIP.COM v pomnilnik
DDT VERS 2.2            ;DDT izda sporočilo
NEXT  PC
1E00 0100               ;PIP.COM uporablja 29
                        ; strani pomnilnika
-ipapip1.hex (cr)       ;Priprava za včitanje mo-
                        ; difikacijske zbirke
-r (cr)                 ;Včitanje PAPIP.HEX, ki
                        ; prekrije dele zbirke
                        ; PIP.COM
NEXT  PC
1E00 0000
-^C                     ;S CTRL-C izstopimo iz DDT

A>save 29 rpip.com (cr)
                        ;Iz 29 strani v pomnilniku
                        ; oblikujemo novo zbirko
                        ; RPIP.COM
```

Podoben postopek imamo tudi za zbirko RPIP1.HEX, ki smo jo imeli za procesor Z80. Sedaj nam preostane samo še preizkus dobljenih COM zbirk rpip.com (8080) in zipip.com (Z80).

3. Izvajanje dobljenih programov

Večkratno kopiranje zbirk brez vračanja v CP/M lahko preizkusimo z ukazoma rpip in zipip, in sicer predvsem oba nova ukaza R (resetiranje diskete v stanje R/W) in H (hitra ponovitev kopirnega ukaza z vstavitvijo nove diskete). Pri tem ostanejo v veljavi seveda vsi dosedanji ukazi programa PIP. Naš primer pa izgleda tako, kot je prikazano na listi 5.

Tako smo si pripravili novo, privlačno orodje za množično kopiranje disket na sistemu z operacijskim sistemom CP/M 2.2. Nov sistem CP/M Plus ima te možnosti že vgrajene v osnovni sistem, tako da jih ni potrebno posebej dodajati.

```
:03010000C30A012E
:10010A003A8000B7117A01CC7401C3CE0421CB1E08
:10011A003680EB0E0ACD05003ACC1EFE01C26D01F7
:10012A003ACD1EE65FFE48C256012AF30122CC1ED2
:10013A0011DB01CD740121CC1E4E0600230936 24A1
:10014A0011CD1ECD74010E0DCD0500C9FE52C26D32
:10015A000111B401CD74010E0DCD0500CD2E08E1BB
:10016A00C33C052ACC1E22F301C90E09CD0500C9DC
:10017A000D0A50495020312E35207A207265736558
:10018A00746972616E6A656D20285229206 96E2031
:10019A007320686974726F20706F6E6F76697476F7
:1001AA00696A6F202848290D0A240D0A5265736569
:1001BA00746972616E6A652076736568206 4697312
:1001CA006B6F762076207374616E6A6520522F57A2
:1001DA00240D0A506F6E6F7669746576206B6F70A6
:0B01EA006972616E6A613A2024000017
:03096F00C31701AA
:0000000000
```

Lista 2. Heksadecimalni kod (Intelov format) za program z liste 1 ima tri ločene naslovne segmente (100H, 10AH in 96FH); to pa smo tudi želeli. S tako strukturiranim kodom smemo neposredno prekriti obstoječi kod (PIP.COM) v pomnilniku (z naložitvijo v okviru ukaza DDT).

```
A>C:PIP
*CON:=B:PAPIP1.PRN

                ; PRILAGODITEV PROGRAMA PIP ZA UPORABO PRI VEC-
                ;   KRATNEM PRENOSU ZBIRK: FUNKCIJI R (RESETI-
                ;   RANJE DISKOV) IN H (HITRA PONOVITEV)
                ; DISKETE LAHKO ZAMENJUJEMO BREZ UPORABE TOPLE-
                ;   GA ZAGONA IN IZSTOPANJA IZ PROGRAMA PIP
                ;*********************************************
0005 =          BDOS    EQU  05H        ;VSTOPNA TOCKA ZA BDOS
0080 =          VMES    EQU  080H       ;DISKOVNI/UKAZNI VMESNIK
                ;                      CP/M FUNKCIJE:
0009 =          TISPOR  EQU  9          ;FUNKCIJA TISK. SPOROCILA
000A =          BRAKON  EQU  10         ;FUNKCIJA BRANJA KONZOLE
000D =          REDISK  EQU  13         ;FUNK. RESETIRANJA DISKA
                ;
04CE =          ZAPIP   EQU  04CEH      ;NORMALNI ZACETEK PIPA
1ECB =          KOVMES  EQU  1ECBH      ;INTERNI KONZOLNI VMESNIK
082E =          CRLF    EQU  082EH      ;INTERNA CR/LF RUTINA
053C =          ANALIZ  EQU  053CH      ;VSTOP.NASL. ANALIZATORJA
096F =          VSTAVI  EQU  096FH      ;VSTAVITVENI NASLOV
                ;
000A =          LF      EQU  0AH        ;POMIK VRSTICE
000D =          CR      EQU  0DH        ;POMIK VALJA
                ;
0100                    ORG  100H       ;ZACETEK SPREMEMBE PIPA
                ;
0100 C30A01             JMP ZACETK      ;SKOK PREK V/I VEKTORJEV
                ;
010A                    ORG  10AH
                ;
010A 3A8000     ZACETK: LDA  VMES       ;IME ZBIRKE SPECIFICIRANO?
010D B7                 ORA A           ; =0, CE NI IMENA
010E 117A01             LXI D,SPOR1     ;ZACETNO SPOROCILO
0111 CC7401             CZ  TISKAJ      ;NATISNI SPORO"ILO
0114 C3CE04             JMP ZAPIP       ;SKOCI NA ZACETEK PIPA
                ;
0117 21CB1E     DODANO: LXI H,KOVMES    ;KAZALEC NA KONZOLNI VMES
011A 3680               MVI M,128       ; Z DOLZINO 128 ZLOGOV
011C EB                 XCHG            ;KAZALEC ZA CP/M JE V DE
011D 0E0A               MVI C,BRAKON    ;PREBERI KONZOLNI VMESNIK
011F CD0500             CALL BDOS       ; Z UPORABO CP/M
0122 3ACC1E             LDA KOVMES+1    ;STEVILO VTIPKANIH ZNAKOV
0125 FE01               CPI 1           ; ALI JE SAMO EDEN?
0127 C26D01             JNZ RESI        ; CE NI, GA RESI ITD.
012A 3ACD1E             LDA KOVMES+2    ;VZEMI ENOZNAKOVNI UKAZ
012D E65F               ANI 05FH        ; KOT VELIKO CRKO
012F FE48               CPI 'H'         ;ALI JE HITRO PONAVLJANJE
0131 C25601             JNZ RESET       ; NI! RESETIRANJE?
0134 2AF301             LHLD STEVZN     ; JE! STEVILO ZNAKOV:
0137 22CC1E             SHLD KOVMES+1
013A 11D501             LXI D,SPOR3     ;SPOROCILO O PONOVITVI
013D CD7401             CALL TISKAJ     ; POSTOPKA
0140 21CC1E             LXI H,KOVMES+1  ;KAZALEC K ZADNJEMU UKAZU
0143 4E                 MOV C,M         ;DOLZINA UKAZA
0144 0600               MVI B,0         ;IZRACUN ODMIKA
0146 23                 INX H
0147 09                 DAD B
0148 3624               MVI M,'$'       ;KONEC SPOROCILA
014A 11CD1E             LXI D,KOVMES+2 ;TISKANJE SPORO"ILA
014D CD7401             CALL TISKAJ
0150 0E0D               MVI C,REDISK    ;IZVAJANJE FUNKCIJE RESE-
0152 CD0500             CALL BDOS       ; TIRANJA DISKA
0155 C9                 RET
                ;
0156 FE52       RESET:  CPI 'R'         ;RESETIRANJE DISKA?
0158 C26D01             JNZ RESI        ; "E NI, SHRANI ITD.
015B 11B401             LXI D,SPOR2     ;TISKAJ R/W SPORO"ILO
015E CD7401             CALL TISKAJ
0161 0E0D               MVI C,REDISK    ;IZVAJANJE FUNKCIJE RESE-
0163 CD0500             CALL BDOS       ; TIRANJA DISKA
0166 CD2E08             CALL CRLF       ;TISKAJ CRLF
0169 E1                 POP H           ;OCISTI SKLAD
016A C33C05             JMP ANALIZ      ;IZDAJ ZNAK '*'
                ;
016D 2ACC1E     RESI:   LHLD KOVMES+1   ;STEVILO ZNAKOV + ZNAK
0170 22F301             SHLD STEVZN
0173 C9                 RET
                ;
0174 0E09       TISKAJ: MVI C,TISPOR    ;FUNKCIJA TISKANJA SPOR.
0176 CD0500             CALL BDOS
0179 C9                 RET
                ;
017A 0D0A504950 SPOR1   DB CR,LF,'PIP 1.5 Z RESETIRANJEM (R) '
0197 696E207320         DB 'IN S HITRO PONOVITVIJO (H)',CR,LF,'$'
                ;
01B4 0D0A526573 SPOR2   DB CR,LF,'RESETIRANJE VSEH DISKOV V '
01D0 7374616E6A         DB 'STANJE R/W$'
                ;
01DB 0D0A506F6E SPOR3   DB CR,LF,'PONOVITEV KOPIRANJA: $'
                ;
01F3 0000       STEVZN  DW 0            ;STEVILO ZNAKOV V KON VM
                ;
096F                    ORG VSTAVI      ;VSTAVITEV SPREMEMBE SE
                ;                         SE ZACNE TUKAJ
096F C31701             JMP DODANO      ;DODANA UKAZA R IN H
                ;
                ;*********************************************
0972                    END
```

Lista 1. Ta lista prikazuje program v zbirnem jeziku procesorja 8080, s katerim bomo prekrili obstoječi kod programa PIP.COM; to operacijo bomo imenovali modifikacija programa PIP.COM (angleško patching). Program v zbirnem jeziku je bil preveden s standardnim CP/M zbirnikom ASM.COM, ki generira absolutni računalniški kod. V prvem segmentu programa (100H) spremenimo začetni skok, tako da PIP začne najprej izvajati programski segment, ki začenja pri naslovu 10AH. Ta drugi segment natisne ustrezna sporočila na konzolo, s katerimi pouči uporabnika, kako lahko dodatno uporablja modificirani PIP (dodana sta dva nova ukaza R - resetiranje v stanje R/W in H - hitra ponovitev prejšnje operacije pri novovstavljeni disketi v diskovno enoto). Uporabniški ukaz za določeno sestavljeno PIP operacijo se vstavi v poseben vmesnik, tako da je mogoče ta ukaz ponoviti poljubno mnogokrat pri novih disketah z uporabo ukaza H. Tretji segment je skok na lokaciji 96FH na naslov DODANO. Sporočili S in L o napaki (lista) nista bistveni!

```
*CON:=C:RPIP1.PRN
                                        SD SYSTEMS Z80 ASSEMBLER PAGE
 ADDR  CODE      STMT SOURCE STATEMENT

               0001 ; PRVI MODUL PROGRAMA RPIP S POJASNITVIJO V MO-
               0002 ;   DULU RPIP2.ASM
               0003 ;*********************************************
               0004        PSECT  ABS
               0005        GLOBAL ZACETK
 >0100         0006        ORG 100H
 0100  C3FFFF  0007        JP ZACETK         ;SKOK V BISTVENI DEL
               0008        END               ; DRUGEGA MODULA

ERRORS=0000
```

```
*CON:=C:RPIP2.PRN
                                        SD SYSTEMS Z80 ASSEMBLER PAGE
 ADDR  CODE      STMT SOURCE STATEMENT

               0001 ; PRILAGODITEV PROGRAMA PIP ZA UPORABO PRI VEC-
               0002 ;   KRATNEM PRENOSU ZBIRK: FUNKCIJI R (RESETI-
               0003 ;   RANJE DISKOV) IN H (HITRA PONOVITEV)
               0004 ; DISKETE LAHKO ZAMENJUJEMO BREZ UPORABE TOPLE-
               0005 ;   GA ZAGONA IN IZSTOPANJA IZ PROGRAMA PIP
               0006 ;*********************************************
               0007        PSECT  ABS
               0008        GLOBAL ZACETK
               0009        GLOBAL DODANO
               0010 ;
 >0005         0011 BDOS   EQU  05H          ;VSTOPNA TOCKA ZA BDOS
 >0080         0012 VMES   EQU  80H          ;DISKOVNI/UKAZNI VMESNIK
               0013 ;                      CP/M FUNKCIJE:
 >0009         0014 TISPOR EQU  9            ;FUNKCIJA TISK. SPOROCILA
 >000A         0015 BRAKON EQU  10           ;FUNKCIJA BRANJA KONZOLE
 >000D         0016 REDISK EQU  13           ;FUNK. RESETIRANJA DISKA
               0017 ;
 >04CE         0018 ZAPIP  EQU  4CEH         ;NORMALNI ZACETEK PIPA
 >1ECB         0019 KOVMES EQU  1ECBH        ;INTERNI KONZOLNI VMESNIK
 >082E         0020 CRLF   EQU  82EH         ;INTERNA CR/LF RUTINA
 >053C         0021 ANALIZ EQU  53CH         ;VSTOP.NASL. ANALIZATORJA
               0022 ;
 >010A         0023        ORG 10AH
               0024 ;
 010A  3A8000  0025 ZACETK:LD A,(VMES)       ;IME ZBIRKE SPECIFICIRANO?
 010D  B7      0026        OR A              ; =0, CE NI IMENA
 010E  117701  0027        LD DE,SPOR1       ;ZACETNO SPOROCILO
 0111  CC7101  0028        CALL Z,TISKAJ     ;NATISNI SPORO^ILO
 0114  C3CE04  0029        JP ZAPIP          ;SKOCI NA ZACETEK PIPA
               0030 ;
 0117  21CB1E  0031 DODANO:LD HL,KOVMES      ;KAZALEC NA KONZOLNI VMES
 011A  3680    0032        LD (HL),128       ; Z DOLZINO 128 ZLOGOV
 011C  EB      0033        EX DE,HL          ;KAZALEC ZA CP/M JE V DE
 011D  0E0A    0034        LD C,BRAKON       ;PREBERI KONZOLNI VMESNIK
 011F  CD0500  0035        CALL BDOS         ; Z UPORABO CP/M
 0122  3ACC1E  0036        LD A,(KOVMES+1)   ;STEVILO VTIPKANIH ZNAKOV
 0125  FE01    0037        CP 1              ; ALI JE SAMO EDEN?
 0127  2041    0038        JR NZ,RESI-$      ; CE NI, GA RESI ITD.
 0129  3ACD1E  0039        LD A,(KOVMES+2)   ;VZEMI ENOZNAKOVNI UKAZ
 012C  CBAF    0040        RES 5,A           ; KOT VELIKO CRKO
 012E  FE48    0041        CP 'H'            ;ALI JE HITRO PONAVLJANJE
 0130  2022    0042        JR NZ,RDISK-$     ; NI! RESETIRANJE?
 0132  2AF001  0043        LD HL,(STZN)      ; JE! STEVILO ZNAKOV:
 0135  22CC1E  0044        LD (KOVMES+1),HL
 0138  11D801  0045        LD DE,SPOR3       ;SPOROCILO O PONOVITVI
 013B  CD7101  0046        CALL TISKAJ       ; POSTOPKA
 013E  21CC1E  0047        LD HL,KOVMES+1    ;KAZALEC K ZADNJEMU UKAZU
 0141  4E      0048        LD C,(HL)         ;DOLZINA UKAZA
 0142  0600    0049        LD B,0            ;IZRACUN ODMIKA
 0144  23      0050        INC HL
 0145  09      0051        ADD HL,BC
 0146  3624    0052        LD (HL),'$'       ;KONEC SPOROCILA
 0148  11CD1E  0053        LD DE,KOVMES+2    ;TISKANJE SPORO^ILA
 014B  CD7101  0054        CALL TISKAJ
 014E  0E0D    0055        LD C,REDISK       ;IZVAJANJE FUNKCIJE RESE-
 0150  CD0500  0056        CALL BDOS         ; TIRANJA DISKA
 0153  C9      0057        RET
               0058 ;
 0154  FE52    0059 RDISK: CP 'R'            ;RESETIRANJE DISKA?
 0156  2012    0060        JR NZ,RESI-$      ; ^E NI, SHRANI ITD.
 0158  11B101  0061        LD DE,SPOR2       ;TISKAJ R/W SPORO^ILO
 015B  CD7101  0062        CALL TISKAJ
 015E  0E0D    0063        LD C,REDISK       ;IZVAJANJE FUNKCIJE RESE-
 0160  CD0500  0064        CALL BDOS         ; TIRANJA DISKA
 0163  CD2E08  0065        CALL CRLF         ;TISKAJ CRLF
 0166  E1      0066        POP HL            ;OCISTI SKLAD
 0167  C33C05  0067        JP ANALIZ         ;IZDAJ ZNAK '*'
               0068 ;
 016A  2ACC1E  0069 RESI:  LD HL,(KOVMES+1)  ;STEVILO ZNAKOV + ZNAK
 016D  22F001  0070        LD (STZN),HL
 0170  C9      0071        RET
               0072 ;
 0171  0E09    0073 TISKAJ:LD C,TISPOR       ;FUNKCIJA TISKANJA SPOR.
 0173  CD0500  0074        CALL BDOS
 0176  C9      0075        RET
               0076 ;
 0177  0D0A    0077 SPOR1  DEFW 0A0DH
 0179  50495020 0078        DEFM 'PIP 1.5 Z RESETIRANJEM (R)
       312E3520
       7A207265
       73657469
       72616E6A
       656D2028
       522920
 019A  696E2073 0079        DEFM 'IN S HITRO PONOVITVIJO (H)'
       20686974
       726F2070
       6F6E6F76
       69747669
       6A6F2028
       4829
 01AE  0D0A    0080        DEFW 0A0DH
 01B0  24      0081        DEFM '$'
               0082 ;
 01B1  0D0A    0083 SPOR2  DEFW 0A0DH
 01B3  52657365 0084       DEFM 'RESETIRANJE VSEH DISKOV V '
       74697261
       6E6A6520
```

```
       76736568
       20646973
       6B6F7620
       7620
01CD   7374616E   0085          DEFM 'STANJE R/W$'
       6A652052
       2F5724
                  0086 ;
01D8   0D0A       0087 SPOR3    DEFW 0A0DH
01DA   506F6E6F   0088          DEFM 'PONOVITEV KOPIRANJA: $'
       76697465
       76206B6F
       70697261
       6E6A613A
       2024
                  0089 ;
01F0   0000       0090 STZN     DEFW 0              ;STEVILO ZNAKOV V KON VM
                  0091 ;
                  0092          END

ERRORS=0000
```

```
*CON:=C:RPIP3.PRN
                                        SD SYSTEMS Z80 ASSEMBLER PAGE
 ADDR  CODE       STMT SOURCE STATEMENT

                  0001 ; DODATEK K GLAVNEMU MODULU RPIP2.ASM
                  0002 ;*************************************************
                  0003          PSECT ABS
                  0004          GLOBAL DODANO
 >096F            0005 VSTAVI   EQU 096FH           ;VSTAVITVENI NASLOV
 >096F            0006          ORG VSTAVI
 096F  C3FFFF     0007          JP DODANO
                  0008          END

ERRORS=0000
```

Lista 3. Programska lista na tej in na prejšnji strani prikazuje program, ki je ekvivalenten programu v listi 1, le da je zapisan v zbirnem jeziku procesorja Z80 in da so posamezni segmenti programa (ti začenjajo pri naslovih 100H, 10AH in 96FH) razdeljeni na module. Zbirnik ZASM, s katerim te programe prevedemo, generira premestljivi in povezljivi kod (kasnejša uporaba programa LINK za povezovanje modulov). Ti moduli morajo biti zaradi natančne njihove vstavitve v program PIP.COM absolutni; zato začenja vsak od teh modulov s posebnim zbirniškim psevdoukazom PSECT ABS. Po povezovanju moramo dobiti heksadecimalni kod, ki je sestavljen iz treh naslovnih segmentov, kot kaže lista 4. Program na tej listi je lažje čitljiv in bolje razumljiv zaradi smotrnejšega zbirnega jezika procesorja Z80 v primerjavi z zbirnim jezikom procesorja 8080. V zbirni listi so oštevilčene tudi vrstice, kar olajšuje iskanje in sklicevanje na določene ukaze oziroma programske segmente. Iz modula v modul lahko skačemo le z JP (absolutnimi) in ne z JR (relativnimi) ukazi.


```
:0301000C30A012E
:20010A003A8000B7117701CC7101C3CE0421CB1E3680EB0E0ACD050D3ACC1EFE012041 3AB5
:20012A00CD1ECBAFFE48202822AF00122CC1E11D801CD710121CC1E4E0600230936241 1CDB5
:20014AC01ECD71010E0DCD0500C9FE5220121 1B101CD71010E0DCD0500CD2E08E1 33C0529
:20016A002ACC1E22F001C90E09CD0500C90D0A5049502031 2E35207A207265736574697267
:20018A00616E6A656D20285229206962974726F20706F6E6F7669747669 6A6FD0
:2001AA0020284829 0D0A2400DA52657365746972616E6A65207673656820646973656F7628
:2001CA0020762073746 16E6A6520522F57240D0A506F6E6F766974657620 6F706972137
:0801EA006E6A613A2024000056
:03096F00C31701AA
:00010001FE
```

Lista 4. Zlogov heksadecimalni format na tej listi se razlikuje od Intelovega formata na listi 2. Tu je maksimalno število zlogov v vrstici enako 32 (20H) za razliko od 16 (10H) pri Intelovem formatu. Nekatere vrstice so seveda lahko tudi krajše (npr. prva, deveta, deseta in enajsta). Prikazani format očitno zadovoljuje našo zahtevo po prekrivanju programa PIP.COM v pomnilniku, ko uporabljamo interne ukaze prehodnega programa DDT.COM. Razlika med zlogovim in Intelovim heksadecimalnim formatom je tudi v zadnji vrstici. Intelova zadnja vrstica je sestavljena iz samih ničel (glej listo 2), v zlogovi zadnji vrstici pa se oblikuje med drugim tudi vsota parnosti (zadnji zlog = zadnja dva znaka = FEH). Izkaže se, da ta razlika v zadnji vrstici ne povzroča stranskih učinkov in da smemo zlogove heksadecimalne zbirke brez nadaljnega uporabljati v okviru ukaza DDT. Še pojasnilo: v drugi vrstici pomeni ':' začetek nove naslovne vrstice z dolžino 20H zlogov pri naslovu 010AH, ko imamo tip zapisa 00 (zadnja vrstica ima tip 01); nato sledi 32 zlogov. Na koncu vrstice je vselej zlog vsote parnosti. Ta format je nastal v razdobju uporabe ASCII teleprinterjev in zadnji zlog se je uporabljal za verifikacijo pri čitanju z luknjanega papirnega traku.

```
A>C:ZIPIP ———————————————————————— Pokličemo modificiran program ZIPIP.COM,
                                    ki se nahaja na diskovni enoti C:
PIP 1.5 Z RESETIRANJEM (R) IN S HITRO PONOVITVIJO (H) — Program se javi s sporočilom
*R ——————————————————————————————— Uporabimo nov ukaz R, ki povzroči reseti-
                                    ranje zbirk v stanje R/W
RESETIRANJE VSEH DISKOV V STANJE R/W
*D:=C:*.?SC ——————————————————————— Vtipkamo PIP ukaz, s katerim se bodo vse
                                    zbirke, ki končujejo v razširitvi s
                                    podnizom 'SC' kopirale z diskete enote
COPYING -                           C: na disketo enote :D
AAAA.ASC
BBBB.ASC
CCCC.ASC
DDDD.BSC
*H ——————————————————————————————— Vstavimo novo disketo v enoto :D in pono-
PONOVITEV KOPIRANJA: D:=C:*.?SC     vimo postopek kopiranja

COPYING -
AAAA.ASC
BBBB.ASC
CCCC.ASC
DDDD.BSC
*H ——————————————————————————————— Vstavimo novo disketo v enoto :D in pono-
PONOVITEV KOPIRANJA: D:=C:*.?SC     vimo postopek kopiranja

COPYING -
AAAA.ASC
BBBB.ASC
CCCC.ASC
DDDD.BSC
* ————————————————————————————————— Po trikratnem kopiranju izstopimo iz pro-
                                    grama ZIPIP.COM v operacijski sistem
A>                                  CP/M na začetno diskovno enoto :A
```

Lista 5. Primer na tej listi kaže, kako je mogoče z modificiranim programom PIP večkrat kopirati zbirke iz ene diskete na drugo brez vmesnega vračanja v sistem CP/M. Pri tem smemo diskete v diskovnih enotah poljubno zamenjevati z novimi in uporabljati vse ukaze programa PIP vključno z novima dvema ukazoma R in H. Podobno listo bi dobili z uporabo programa RPIP, ki je bil napisan za procesor 8080.

****
* Modula-2 za CP/M sistem *
****

Prevajalnik za jezik Modula-2 na CP/M sistemu zahteva 60k zlogov pomnilnika. Modula-2 je napreden programirni jezik (o njenu smo pisali tudi v časopisu Informatica), ki se je razvil iz jezika Pascal. Njegov poudarek temelji na konceptu modula, kar omogoča razvoj obsežnih programov, ki so sestavljeni iz modulov; ti se shranjujejo v programski knjižnici. Ker je vsak modul razdeljen v tkim. definicijski in implementacijski del, je mogoče vsak modul modificirati brez vpliva na njegovo povezavo z ostalim delom programa. Razen tega ima Modula-2 lastnost tipskega preizkušanja med ločeno prevedenimi programskimi segmenti.

Sintaksa Module-2 je podobna pascalski sintaksi, je pa bolj sistematična in lažje priučljiva. Modula-2 ima tudi nizkoravninske pripomočke, ki omogočijo neposreden dostop v računalniško opremo in dovoljujejo strogo tipsko preizkušanje. Ta relativno nov jezik ima tudi multiprogramirne možnosti, kot so signali, monitorji in procesno oblikovanje. Koncept procedurnih spremenljivk prinaša nove možnosti za programsko krmiljenje.

Modulo-2 je definiral N. Wirth, ki je oblikoval jezik Pascal v letu 1970. Cena paketa je $ 100. Naslov proizvajalca: JRT Systems, 45 Camino Alto, Mill Valley, CA 94941, USA.

A.P.Železnikar

****
* Mikrovojna filmska zanimivost *
****

V novem filmu podjetja MGM/UA, ki ima naslov "Vojne igre", uporablja 100 vojaških strategov dva mikrosistema podjetja CompuPro. Dva sistema poganjata vrsto zaslonskih računalnikov z zanimivimi barvnimi učinki; programe za te efekte je izdelal filmski računalniški konzultant Steve Grumette.

Za veliko vojno dvorano je Grumette sprogramiral 120 monitorjev, ki oddajajo na stotine različnih bojnih planov z bliskovito hitrostjo - vse z uporabo sistemov CompuPro. Pri tem je odločilna hitrost za realno dogajanje na sceni. Glavna prednost je namreč v tkim. mikromodulu M-Drive, ki simulira winchestrski disk z veliko hitrostjo. Druga zahteva je bila zanesljivost mikrosistema.

A.P.Železnikar

****
* Novice iz podjetja Digital Research *
****

Podjetje Digital Research je objavilo nov cenik svojih programskih izdelkov, in sicer:

Opis izdelka	Format 3740 disk 8" 8-bitni	Format 3740 disk 8" 8086
Operacijski sistemi in pripomočki		
CP/M	$165	$275
CP/M Plus	$385	
MP/M	$495	$715
CP/Net	$220	
Despool	$ 55	
Tex	$110	
Jeziki in programirni pripomočki		
CBASIC	$165	$358
CBASIC Compiler	$550	$660
Personal BASIC		$165
C		$660
Pascal/MT+	$385	$660
Speed Programming Package	$220	$275
Pascal/MT+ with Speed Programming Package	$550	$880
PL/1	$605	$825
CIS Cobol	$935	$935
Level II Cobol		$1760
Animator		$880
Forms II		$220
Native Code Generator		$880
Run Time System		$440
DR-Logo	$ 99,95	$149,95
Programirne storitve		
Programmers Utilities	$220	$220
BT-80	$220	
XTL86	$165	
SID	$110	$165
Access Manager	$330	$440
Display Manager	$440	$550
Grafika		
GSX	$ 82,50	$ 99
DR-Kernel	$550	
DR-Plot	$550	
DR-Graph	$440	
DR-Draw	$440	

Cene v tej tabeli veljajo za en sam kos. Seveda prodaja Digital Research še druge izdelke za IBMov PC, Apple II, Rainbow (DEC) in za TI PC. Cena CP/M-86 za IBMov PC znaša le $66.

A.P.Železnikar

****
* Vinčestrski krmilnik za amaterje *
****

Z novimi integriranimi vezji si lahko zgradite krmilnik za uporabo vinčestrskega diska tudi sami. Podjetje Western Digital proizvaja vrsto združljivih integriranih vezij za te namene, in sicer:

-- WD1010 vinčestrski krmilnik, ki oblikuje krmilno vezje za standardne vinčestrske diskovne enote (ST500/SA1000 itd.), je povezljiv z večino 8- in 16-bitnih mikroprocesorskih vodil in dosega hitrost prenosa podatkov do 5Mbit/s

-- WD1011 je ustrezni separator digitalnih podatkov, ki se uporablja pri branju podatkov iz vinčestrskega diska

-- WD1012 rabi za pisalno prekompenzacijo

-- WD1014 je krmilnik za popravljanje napak

-- WD1015 je vmesniški upravljalnik

-- WD1510 je LIFO/FIFO, ki se lahko uporabi kot zunanji sektorski vmesnik in

-- WD279X je pripadajoči krmilnik za upogljivi disk (disketa)

Naslov proizvajalca: Western Digital, Components Group, 2445 McCabe Way, Irvine, CA 92714, USA.

A.P.Železnikar

****
* Ameriškojaponski mikrosistem *
****

Na evropskem tržišču se je pojavil mikroračunalnik Seiko Series 8600, ki ima šest (beri 6) različnih operacijskih sistemov in sedem visokih programirnih jezikov. Ta sistem se proizvaja v skupnem ameriškojaponskem podjetju Sci-Com. Ta računalnik je večuporabniški mali poslovni sistem, ki uporablja procesor 8086 (5 MHz), 512k RAM, 10M vinčestrski disk in do dva upogljiva diska (5 in 1/4"). Za komunikacijo je predviden dupleksni RS232C port (50 do 19,2k baud), 8-bitni paralelni V/I za Centronics' in dodatni CCITT-X.21 standardni HDLC vmesnik ter do štiri prikazovalnopisalne enote s procesorji 8085.

Operacijski sistemi za Seikov mikroračunalnik so MS-DOS, CP/M-86, MBOS, OASIS-16, UNIDOL in MP/M-86. MBOS (Science Management Corp) je multioperacijski sistem, ki je podprt z Basic Four poslovnim jezikom. UNIDOL (isti proizvajalec) je eno- in večuporabniški primerek sistema UNIX III. Dodatni jeziki so še Fortran, Cobol, Pascal, C, MBasic in CBasic.

Osrednji procesor ima pri 5MHz taktu 4 DMA kanale, tri intervalne časovnike, uro realnega časa in 8 vektorskih prekinitev. Osnovni sistem ima 128k 200ns RAM, razširljiv do 512k. Inicializacijske rutine so samodiagnostične. Cena minimalne večuporabniške konfiguracije z enim 640k pogonom za upogljivi disk in z 10M vinčestrskim diskom, z enim terminalom in večuporabniškim operacijskim sistemom znaša v Veliki Britaniji približno 7000 funtov.

A.P.Železnikar

****
* Unix System 5 naj bi postal standard *
****

Kako naj bi dosegli zadovoljivo standarizacijo operacijskega sistema Unix, ki se prodaja v

različnih izvedbah? Bell Laboratories, ki je oblikoval sistem Unix, je prodal licence štirim vodilnim proizvajalcem integriranih vezij, in sicer podjetjem Intel, Motorola, National Semiconductors in Zilog, pod pogojem, da zgradijo splošen (generičen) operacijski sistem Unix 5 za njihove sedanje in prihodnje 16-bitne mikroprocesorje. Cilj takega dela naj bi bila skupna aplikativna programska oprema, ki je portabilna (prenosljiva) iz enega računalniškega sistema na drugega brez spreminjanja koda ali drugih popravkov.

Generični primerek operacijskega sistema bo pripravljen za prodajo v začetku leta 1984, in sicer za procesorje iAPX 286, 68000, 16032 in Z8000. Western Electric bo dobavil tem proizvajalcem Unix System 5 v izvirnem kodu za VAX 11/750 in proizvajalci bodo na osnovi tega razvili procesorsko odvisne dele sistema. Po izdelavi teh delov bodo ti vrnjeni Western Electric in bodo ocenjeni, ali res izpolnjujejo pogoje sistema 5.

A.P.Železnikar

****
*** Potrditev prevajalnikov jezika Ada ***
****

DOD (Department of Defense) je doslej potrdil dve izvedbi prevajalnika za Ado, in sicer podjetij Rolm Corp (Santa Clara, CA) in univerzi v New Yorku za Ada/Ed prevajalnik. Tretjo potrditev je dobilo podjetje Western Digital. Verifikacija je sestavljena iz 1800 poskusov s 100000 vrsticami koda. Ta preizkus je bil npr. opravljen na računalniku Eclipse MV/8000 (Data General) za prevajalnik podjetja Rolm; preizkus je trajal 36 ur. Iz tega je razvidno, kako strog in zapleten je tak preizkus. IBM si zelo prizadeva, da bi dobil ta prevajalnik za uporabo na svojih sistemih.

A.P.Železnikar

****
*** Intelov Multibus za 32-bitne procesorje ***
****

Na projektu razvoja 32-bitnega vodila podjetja Intel dela 13 podjetij, med njimi tudi nemški Siemens in Nixdorf. Zgradba tega vodila (Multibus ii) bo omogočala podobno delovanje procesorjev, kot je omogočeno pri dosedanjem 16-bitnem multibusu (to vodilo je sprejelo 150 najpomembnejših podjetij). Modularna in procesno neodvisna zgradba vodila naj bi povečala sistemsko zmogljivost, zanesljivost in multiprocesiranje. Na 32-bitnem vodilu bi lahko obratovali tudi prejšnji 8- in 16-bitni moduli, ki upoštevajo standard multibusa. Novo vodilo naj bi imelo poseben protokol, ki omogoča realizacijo večvodilnosti (večkratno vodilo), višje arhitekturne ravnine; vse to naj bi omogočilo načrtovanje zmogljivih multiprocesorskih sistemov.

A.P.Železnikar

****
*** MagicPrint ***
****

MagicPrint je dodatek k procesorju in urejevalniku besedil z imenom WordStar in je predviden za uporabo tiskalnikov Diablo, Qume in NEC; ti tiskalniki zmorejo tkim. proporcionalno tiskanje znakov, ki je po videzu lepše od enakomernega tiskanja (navadni tiskalniki). MagicPrint je samostojen tiskalni program. Besedilne zbirke se pripravijo kot navadno z WordStarom z vgnezdenimi novimi ukazi v zbirki; pri tem se ne upoštevajo ukazi s piko na začetku vrstic, ki pripadajo WordStaru. Normalno pa delujejo ukazi WordStara za polkrepko, podčrtano in drugače poudarjeno besedilo. Magic Print pa uvaja tudi druge lastnosti, kot je avtomatično pisanje opomb (footnoting), večvrstične glave in podnožja besedil in še druge lastnosti. Ko je zbirka dokončno urejena, se uporabi MagicPrint za njeno tiskanje in normalne rutine WordStara se pri tem ne uporabljajo.

Krmilni znaki MagicPrinta so zbrani v tabeli 1. Ti znaki so vgnezdeni (vnešeni) v besedilo in izginejo (se zbrišejo) na koncu odstavka.

CTRL-An	Spremenljivi razmak med črkami (pitch), kjer je n = 0, ... ,9
CTRL-B	Polkrepko tiskanje
CTRL-C	Ustavitev, ki omogoča zamenjavo tiskalne glave
CTRL-D	Razprto tiskanje
CTRL-E	Začetek in konec opombe /komentarja
CTRL-F	Natisne znak centa
CTRL-G	Natisne tkim. "neznak"
CTRL-H	Akcentiranje
CTRL-L	Nova stran
CTRL-N	Ukaz resetiranja
CTRL-O	Neprekinitveni razmak
CTRL-R	Poravnava ene same vrstice
CTRL-S	Podčrtavanje
CTRL-T	Tiskanje potence (superscript)
CTRL-V	Tiskanje indeksa (subscript)
CTRL-X	Prečrtavanje
CTRL-Y	Sprememba barve traku
CTRL-_	Mehka delilna črtica

Tabela 1. Vgnezdeni ukazi za MagicPrint

MagicPrint pozna dva načina podčrtavanja: podčrtavanje znakov (in ne presledkov) in strnjeno podčrtavanje. Wordstar dovoljuje le dva načina vnaprej opredeljenega razmikanja (pitch); Magic Print ima deset inkrementov, ki se uporabljajo za tkim. lokalno tipografsko nameščanje (kerning). Popolnoma nov ukaz je CTRL-R, s katerim se poravnavajo posamezne vrstice in CTRL-L, ki se uporablja namesto ukaza s piko ´.pa´ v Word Staru (brezpogojni konec strani) ter seveda ukaz CTRL-E za pisanje opomb (komentarjev).

MagicPrint uporablja vejične ukaze (ukazi, ki začenjajo z vejico v nasprotju z onimi, ki začenjajo s piko v WordStaru), ki so zbrani v tabeli 2.

Podobno kot pri WordStaru se morajo vejični ukazi postaviti,začeti v prvem stolpcu levo. MagicPrint pozna dva razreda vejičnih ukazov, in sicer stabilne ukaze, na katere ne vpliva pomik valja (navadni ukazi) in prehodne ukaze, ki se s pomikom valja resetirajo. Iz tabele 2 je razvidna raznovrstnost ukazov pa tudi, kaj je v bistvu k WordStaru dodanega.

MagicPrint deluje zanesljivo in natanko tako, kot je opredeljeno. Sporočila o napakah se pojavljajo v zvezi s problemi v zbirkah. Tiskanje teksta je neprimerno bolj kvalitetno kot pri WordStaru, seveda z uporabo proporcionalnega tiskalnika. Slabost MagicPrinta je v slabosti vsakega tipičnega procesorja besedila, kjer oblikovanega besedila ne vidimo prej na zaslonu; zato je včasih potrebno ponovno tiskanje besedila, če želimo doseči natanko zamišljeno obliko besedila.

Cena MagicPrinta znaša $195. Znakovne glave (marjetice) za proporcionalni tisk so kovinske (ne plastične) in za slovanske jezike še niso na voljo. Kljub tem pomanjkljivostim pa pomeni

B	Pomik za vrstico nazaj
C	Centriraj vrstice
C	Centriraj eno vrstico
C+	Vključi centriranje
C-	Izključi centriranje
C0	Enako kot C-
Cx	Centriraj x vrstic
D	Začetne vrednosti ponovno veljajo
E	Pravilna poravnava posamezne vrstice
E-	na straneh s sodo številko
E+	na straneh z liho številko
F	Krmiljenje ustavitve na koncu strani
F0	Tiskanje brez ustavitve
F1	Ustavitev na koncu strani
G	Nastavitev dolžine strani s številom vrstic
GG	Nastavitev trajne dolžine strani s številom vrstic
H	Sprememba gostote znakov
I	Umaknitev vrstice na levi strani
J	Poravnava vrstic na desnem robu
J0	prava proporcionalna neporavnava
J1	prava proporcionalna poravnava
J2	stara proporcionalna neporavnava
J3	stara proporcionalna poravnava
L	Nastavitev dolžine vrstice
M	Nastavitev levega roba
N0	Enako kot CTRL-L
N+x	Pomaknitev na novo stran, če je v trenutnem odstavku več kot x vrstic
N-x	Pomaknitev na novo stran, če je v trenutnem odstavku manj kot x vrstic
O	Izmaknitev prve vrstice odstavka
P	Nastavitev števila strani
Px	Začetek oštevilčevanja strani z x
P0	Prestavitev oštevilčevanja strani na naslednjo stran
P0x	Začetek oštevilčevanja strani na naslednji strani
P-	Izključi oštevilčevanje strani
P+	Vključi oštevilčevanje strani
R	Pravilna poravnava vrstice
S	Nastavitev vrstičnega razmaka Znak '+' doda polovični razmak
U	Prekinjeno ali neprekinjeno podčrtavanje
U0	prekinjeno podčrtavanje (presledki
U1	neprekinjeno podčrtavanje
V	Nastavitev razdalje med vrsticami
W	Razdalja pri skrajšani vrstici na desni (desna umaknitev)
RETURN	Ustavitev tiskanja
	Simbol formata številke strani
T	Številka strani na vrhu strani
B	Številka strani na dnu strani
T0 ali B0	Odloži oštevilčevanje strani na naslednjo stran
)	Konec formatnega bloka oštevilčevanja strani
*	Formatni simbol naslova strani
*T	Naslov strani na njenem vrhu
*B	Naslov strani na njenem dnu
*T0 ali *B0	Začetek naslavljanja na naslednji strani
)	Konec formatnega bloka naslavljastrani

Tabela 2. Vejični ukazi MagicPrinta

MagicPrint kvalitetén dodatek na področju tiskanja besedil, še posebej, če bi ga bilo moč uporabiti z visokotočkovnim mozaičnim proprorcionalnim tiskalnikom, kjer bi odpadel problem kovinskih marjetic. Prodajalec MagicPrinta je Lifeboat Associates, 1651 Third Avenue, New York, NY 10028, USA.

A.P.Železnikar

Kako je z upravljavskim informacijskim sistemom danes ?

Pred nekako desetimi leti se je pojavila strezнitev o možnostih upravljanskega informacijskega sistema (MIS), ko ni bilo več pričakovati rešitev s praktično vrednostjo. Mikroelektronika in razvoj programskih sistemov pa sta v tem razdobju doživela popolnoma nov razmah in tako se je ponovno pojavilo staro vprašanje: ali je v današnjem času ob pomoči mikroelektronike, telekomunikacij in programske metodologije mogoče zgraditi MIS in to še posebej v majhnih in srednjevelikih podjetjih? Kakšen je tedaj odgovor na to vprašanje danes?

Pred desetimi leti je bilo znanih več zaprek za postavitev MIS, ki jih je mogoče strniti v tele predpostavke:

Predpostavka 1. Centralizirana kontrola in integracija vseh informacijskih sistemov podjetja je nesmiselna in učinkuje kvečjemu zaviralno.

V tistem času je bilo treba upoštevati težave, ki so se pojavile pri velikih računalnikih poznih 60-ih let: vrsta programskih prolagoditev z ročnimi metodami, pomanjkanje prijaznejših programirnih pripomočkov, pomanjkanje primerne obdelave besedil in obstoj tehnokracije, ki je prisegala na velike sisteme in na centralizirane rešitve.

Danes je mogoče z uporabo internih podatkovnih mrež, programirnih terminalov in pisarniških računalnikov rešiti določene probleme optimalno z uporabo centralnih in necentralnih elementov, kar zagotavlja ustrezno koordinacijo in integracijo vseh informacijskih sistemov, z ohranitvijo njihove gibljivosti in lokalne samostojnosti. Ti sistemi nudijo računalno moč na delovnem mestu s prednostmi visoko zmogljivih centralnih naprav.

Predpostavka 2. Masivna uporaba računalnikov ne prispeva bistveno k izboljšanju informacijske kakovosti.

Masivna uporaba računalnikov je v tistih časih pomenila masovno obdelavo, slabo strukturiran output brez komentarjev in nezadostno prilagoditev uporabniškim potrebam.

Danes se že približujemo integraciji podatkov, besedil in slik v enem sistemu; ta integracija je podprta z uporabniško prijaznostjo, dialogičnostjo in visokimi programirnimi jeziki in generatorji. Informacijo je tako mogoče predstaviti v obliki, ki je skladna z uporabniškimi potrebami glede na njeno koncentracijo in obliko, podprto z dialogom na zaslonu ali popolnoma komentirano in optimalno grafično oblikovano.

Predpostavka 3. Sposoben in strokovno kompententen izvedenec, ki bi lahko zgradil MIS, se doslej še ni rodil.

Zaradi naraščajoše sistemske združljivosti in povezovalnih zmogljivosti in z obstojem zmogljivih mrež za podatke in besedila je povezava informacijskih sistemov danes bistveno lažja in ne predstavlja več nerešljivih problemov.

Predpostavka 4. Zaradi razumevanja (znanja) in ekzagtnosti, ki bi bila potrebna za obvladovanje MIS, bi bila večina direktorjev, upravljavcev in njihovih sodelavcev preveč obremenjenih.

Danes je uporabnik voden z dialogom ali z uporabo menujev, tako da lahko v celoti obvlada določen problem na sistemu; komunikacija s sistemom je lahkotna, prijetna in poučna in njena težavnost dejansko ne presega težavnosti telefonskega razgovora.

Predpostavka 5. Visoko učinkovite znanstvene metode, ki so potrebne za visoko zmogljivost MIS, kot so kibernetika, hevristika, odločitveni modeli, izvedenski sistemi itd., v svoji današnji obliki še niso praktično dozoreli za področje podatkovne obdelave.

Pripomočki za vodenje, upravljanje, poslovanje itd. so danes realizirani v programskih paketih že za namizne in osebne računalnike, podprti so z grafiko, izvedenskimi sistemi in s kompleksno komunikacijsko povezavo in z neposrednimi pripomočki za uporabnike.

Predpostavka 6. MIS je mogoče realizirati teoretično, ni pa mogoče predvideti dela, naporov, stroškov in časa za njegovo prektično realizacijo.

Izhodišča so danes bistveno drugačna - izboljšana. Teoretične osnove MIS, izdelane v začetku 70-ih let, so zaradi novih tehnoloških možnosti še vedno uporabne. Medtem se je tudi zmogljivost materialne opreme pri padajočih cenah povečala za več kot velikostni razred; tudi programska oprema nudi bistveno nove možnosti.

Predpostavka 7. Programirna tehnika, standarizacija in sistemska tehnika še niso dovolj zrele za premagovanje težav, povezanih z MIS.

Na treh omenjenih področjih je bil dosežen znaten napredek tako v njihovi teoretični zgradbi kot v praktični uporabi. Še vedno pa se razvija in je viden tudi napredek v integraciji podatkov, besedil in slik in v njihovi obdelavi, shranjevanju in komunikaciji.

Predpostavka 8. Možnosti enostavnih in cenenih delnih informacijskih sistemov in razvoj majhnih in najmanjših računalnikov z veliko zmogljivostjo in možnosti decentralizacije so močno znižali interes za uvedbo MIS.

Ta težnja je v vrsti primerov povzročila medsebojno izolirane rešitve in tehnično delitev podatkovnih in besedilnih obdelav. Kot nasprotna usmeritev se kaže v poslednjih letih integracija, ki kaže v smeri MIS, čeprav je povezana z decentralizirano računalniško zmogljivostjo na delovnem mestu.

Predpostavka 9. Tehnične zmogljivosti računalnih in pomnilnih enot današnje računalniške generacije so za MIS nezadostne.

V tem času so se računalne in pomnilne zmogljivosti ob padajočih stroških povečale za več velikostnih razredov. Danes so namizni računalniki zmogljivejši od velikih računalnikov 70-ih let.

Predpostavka 10. Informacijska baza za smiselno uporabo v MIS je v večini podjetij prerevna in premalo zaščitena. Predvsem manjkajo uporabni prognostični podatki, brez katerih bi bil MIS le draga tehnična igrača.

S to predpostavko so težave žal tudi danes. Vendar so se ovire bistveno zmanjšale pri zajemanju operativnih podatkov z uporabo zmogljivih statističnih programov.

Predpostavka 11. Praktični problemi totalne reorganizacije, velikega števila strokovnjakov in investicijskih finančnih zahtev bi lahko večino podjetij preveč obremenjevali.

Od takratnega prepričanja, da se morata človek in organizacija prilagoditi računalniku, ni ostalo ničesar več. Uporabniška prijaznost naprav in programov je ta problem izničila, cene pa so pri tem skokovito padale.

Iz napisanega izhaja, da so danes tehnične in gospodarske predpostavke za realizacijo upravljavskega informacijskega sistema (MIS) neprimerno boljše kot pred desetimi leti in da je tako praktičen izničen nekrolog enajstih predpostavk, ki je veljal za MIS pred desetimi leti.

Teorija vodenja podjetij je doživela v poslednjih desetih letih bistvene spremembe: metodično usmerjeno mišljenje je bilo nadomeščeno z mišljenjem o nalogah; pri tem niso več toliko bistvene naprave in sistemi kot je bistvena informacijska potreba upravljavske strukture. Pri tem je bistveno vprašanje, kako je mogoče z uporabo MIS zadostiti tem potrebam.

Informacijska potreba na področju poslovanja je izrazito nalogovno usmerjena in jo je glede na njeno podrobno opredelitev, težišče, priročnost in zgradbo le težko predvideti ali jo celo programirati. Z veliko količino števil se tu ne more kaj bistvenega doseči. Direktor uporabi v tem težavnem odločitvenem postopku svoje izkušnje, svoj smisel za bistveno in določeno intuicijo. Seveda pa mora pri tem stati na tleh in potrebni so mu zanesljivi podatki o likvidnosti, kapitalni strukturi, produktivnosti itd. in o stanju gospodarskih in tržnih perspektiv, o novostih in temu podobno. Razen tega pa se mora direktor kratkoročno ukvarjati tudi s podrobnostmi, kot so gospodarske posledice ustavitve določenega produkta, mej cenovnih popustov, poznati mora konkretne možnosti financiranja določene racionalizacije ali učinek napredovanja pomembne stranke na likvidnost itd. Pri vsem tem ne gre za množico podatkov, potrebna pa je kvalitetna in natančno usmerjena informacija. Praktično izgleda to takole:

- -- direktor mora dobiti hitro in neposredno prave in potrebne informacije
- -- posredovane informacije morajo opisovati najnovejše stanje
- -- informacije morajo biti zgoščene, pregledne in razumljive
- -- v težavnih položajih mora biti dana možnost povratnega vpraševanja in zbiranja podrobnosti

Tudi računalniško podprt sistem bo le težko ustregel vsem tem zahtevam. Zahtevam, kot so neposreden dostop, dnevna veljavnost podatkov, nalogovno usmerjeno posredovanje in hitro produciranje podrobnosti, bo možno ugoditi le s pomočjo računalnikov na delovnih mestih, tako pri vodilnih delavcih kot v okolici teh. K vse-

mu temu pa je potrebna še tista integracija podatkov, besedil in slik, ki jo lahko nudi le smotrno organiziran MIS.

Seje in zasedanja so največji požiralci časa, če niso vodeni učinkovito. Ta učinkovitost se bistveno zmanjšuje takrat, ko udeleženci sej govoričijo o svojih pridržkih namesto o sklepnih argumentih ali ko je potrebno predlagani sklep odložiti zaradi pomanjkanja informacij. Če je v sejni dvorani nameščen terminal z barvno grafiko, se lahko razprava seje usmeri na konkretne in veljavne informacije, kot so brutoprodukt, rentabilnost, likvidnost itd. tako da teh podatkov ni potrebno iskati v aktih in v sejnem materialu.

Pogosto se direktor sooča z banalnimi in časovno zahtevnimi informacijskimi problemi. Oglejmo si nekaj primerov:

- oblikovanje dokumenta o trenutno ne preveč razjasnjenem problemu
- pozaba terminov, povezanih z določenim dnevnim redom
- časovno zahtevno iskanje po določenem postpku v arhivu ali v registraturi
- zadrega zaradi neznatnih, vendar v tem trenutku pomembnih takojšnjih informacij
- poprava besedila pod hudim časovnim pritiskom
- nezadovoljiva informacija kljub goram papirja
- nepopolna informacijska učinkovitost zaradi prekomerno centraliziranih naprav, kot so npr. tipkarske storitve in kopiranje

Pred desetimi leti bi pri teh problemih odpovedal tudi zaslon MIS. Današnje naprave pa omogočajo oblikovanje in obdelavo besedil, podatkov in grafik na delovnem mestu, zaradi njihove lokalne inteligence jih je moč uporabiti za osebne storitve, vključene pa so tudi v komunikacijske mreže informacijskih bank, drugih delovnih mest in visoko zmogljivih naprav za obdelavo podatkov.

Čeprav je ideja o izgradnji MIS pred desetimi leti zamrla pa obstajajo danes vse možnosti, da se upravljavski informacijski sistem realizira na zadovoljivi ravni - še zlasti v manjših podjetjih. Tudi domači mikrosistemi (npr. Partner) lahko v tej smeri prispevajo bistveni delež.

A.P.Železnikar

****
* Najmočnejši v obračunskem letu 1982 *
****

Dohodek 100 ameriških podjetij računalniške industrije je znašal v letu 1982 $ 79,4 milijarde. Najbolj so napredovala v primerjavi s prejšnjim letom nekatera manjša podjetja, kot so npr. Tandon (151%), Commodore (99,4%), Apple (65,5%) in Tandy (57,6%). Vsa ta podjetja proizvajajo za področje mikroračunalnikov. Osebni računalniki so se končno tržno uveljavili, kar kaže tudi tabela 1.

Podjetje	PC dohodek 1982 ($ milij.)	PC dohodek 1981 ($ milij.)	Rast %
1. Apple	664,0	401,1	65,5
2. IBM	500,0	n.p.	n.p.
3. Tandy	466,2	293,0	59,1
4. Commodore	367,8	184,4	99,4
5. Hew.-Packard	235,2	195,0	20,6
6. Texas Instr.	233,0	144,3	61,4
7. DEC	200,0	n.p.	n.p.

(n.p. = ni podatkov)
Tabela 1. Povprečna rast 61,2%

Velika povprečna rast je bila dosežena tudi na področju pisarniških sistemov, kot je razvidno iz tabele 2.

Podjetje	P. dohodek 1982 ($ milij.)	P. dohodek 1981 ($ milij.)	Rast %
1. IBM	1800,0	1600,0	12,5
2. Wang Labs	584,8	456,3	28,1
3. Motorola	274,9	232,0	18,4
4. Lanier	241,2	228,0	5,7
5. Burroughs	200,0	50,0	300,0
6. Xerox	200,0	132,0	51,5
12. DEC	100,0	n.p.	n.p.

(n.p. = ni podatkov)
Tabela 2. Povprečna rast 52,1%

Na področju proizvodnje velikih sistemov (mainframes) je bil dosežen vrstni red: IBM, Sperry, Burroughs, NCR, CDC, Honeywell, Amdahl itd. s povprečno rastjo borih 6,84%.

Miniračunalniško področje kaže določene spremembe v primerjavi s prejšnjim letom. IBM se je tu pojavil na prvem mestu in odvzel prvenstvo DECu, ki je celo nazadoval, kot je razvidno iz tabele 3.

Podjetje	Mi dohodek 1982 ($ milij.)	Mi dohodek 1981 ($ milij.)	Rast %
1. IBM	3000,0		n.p.
2. DEC	1680,0	2068,1	-18,7
3. Burroughs	800,0	575,0	39,1
4. Data General	603,8	573,2	5,3
5. Hew.-Packard	588,0	429,4	36,9
6. Wang Labs	584,7	456,4	28,1
7. Prime Compu.	351,0	309,0	13,5
8. Honeywell	330,0	300,0	10,0

Tabela 3. Povprečna rast 14,18%

Nazadnje si oglejmo še vrstni red, ki upošteva podatke o totalnem dohodku in dohodku na področju informatike in njihovo rast. Tako dobimo tabelo 4.

Podjetje	To dohodek 1982 ($ milij.)	In dohodek 1982 ($ milij.)	Rast %
1. IBM	34364,0	31500,0	19,5
2. DEC	4018,8	4018,8	12,0
3. Burroughs	4186,3	3848,0	24,0
4. CDC	4292,0	3301,0	5,8
5. NCR	3526,2	3173,4	3,3
6. Sperry	5242,7	2800,5	0,7
7. Hew.-Packard	4335,0	2164,8	17,8
8. Honeywell	5490,4	1684,7	-5,0
9. Wang Labs	1321,5	1321,5	31,0
0. Xerox	8455,6	1300,0	18,1

Tabela 4. Deset najmočnejših v ZDA
A.P.Železnikar

****
* Evropska mikroračunalniška industrija *
****

Evropsko mikroračunalniško tržišče ima največjo stopnjo rasti na svetu. V zadnjem letu se je britansko tržišče podvojilo, zapadnonemško povečalo za 130%, francosko pa kar za 250%. Prvi

mikroračunalnik je nastal v Evropi, in sicer v letu 1973 (Micral francoskega podjetja R2E), danes pa se Evropa bori za svoj delež na samih evropskih tleh.

Veliki evropski proizvajalci so se relativno pozno vključili v mikroračunalniško proizvodnjo. Siemens je šele letos (1983) na Hannoverskem sejmu najavil mikroračunalniške proizvode. Thomson (Francija) prodaja lastne in cenene mikroračunalnike (TO7 in Fortune 32), ICL (V. Britanija) pa svoj Rair Black Box (osebni računalnik). Nizozemski Philips prodaja večterminalski sistem P3500, Olivetti (Italija) pa svojo družino Linea Uno. Iskra (Jugoslavija) je začela z "masovno" proizvodnjo poslovnega, razvojnega, osebnega in večterminalskega mikroračunalnika Partner šele v letu 1983.

Mikroračunalniško prodajno projekcijo imamo v tabeli 1; ta kaže na možnosti vzpona evropske mikroračunalniške industrije. Na evropsko tržišče pa prodira tudi japonska in ameriška mikroračunalniška industrija; obe se zavedata hitrega vzpona evropskega tržišča.

Prodajna projekcija mikroračunalnikov s ceno pod $12000 (1982 - 1990)

	Gospodarstvo Vzgoja	Dom	Skupaj
Francija			
1982/85	494 920	359 960	854 880
1985/90	1 860 100	1 193 690	3 053 790
Skupaj	2 355 020	1 553 650	3 908 670
Zapadna Nemčija			
1982/85	914 320	522 980	1 437 300
1985/90	2 730 300	1 719 900	4 450 200
Skupaj	3 644 620	2 242 880	5 887 500
Italija			
1982/85	303 950	205 360	509 310
1985/90	1 244 500	642 390	1 886 890
Skupaj	1 548 450	847 750	2 396 200
Združeno kraljestvo			
1982/85	882 685	1 394 325	2 277 010
1985/90	2 255 200	948 890	3 204 090
Skupaj	3 137 885	2 343 215	5 481 100
SFRJ			
1982/85	20 000	75 000	95 000
1985/90	200 000	150 000	350 000
Skupaj	220 000	225 000	445 000

Tabela 1. Število mikroračunalnikov

Jugoslovansko tržišče bo pri omejenem uvozu dokaj ugodno za domačo mikroračunalniško industrijo, vendar bo zaostanek v primerjavi z državami EGS še vedno očiten.

A.P.Železnikar

****
* Britanski uspešnež *
****

Z osebnim računom prek $ 18 milijonov je mikroračunalniški inovator Clive Sinclair eden najuspešnejših novih multimilijonarjev. Sinclair spreminja zloge (bytes) dobesedno v milijone dolarjev na množičnem tržišču mikroračunalniške revolucije. Sinclair je osebnost nasprotij z izbrušenim poslovnim ukusom, ki uživa v poeziji, klasični glazbi in v radikalnih novelah. Šoli je obrnil hrbet že pri sedemnajstih letih, ogibal se je vsake akademske izobrazbe in je cenil predvsem življenje na svoj način. Vendar je pri svojih 42 letih dosegel akademsko stopnjo na Cambridgu in je predsednik britanske Mense - kluba ljudi z visokim IQ (inteligenčnim količnikom).

Njegovo podjetje Sinclair Research je prodalo več kot milijon cenenih domačih računalnikov. Že pred tem uspehom je izdeloval kalkulatorje in propadel z digitalnimi urami, želi pa proizvajati električne avtomobile in miniaturne televizorje. Njegov uspeh temelji na tehnološkem realizmu, čeprav je hkrati tudi družbeni sanjač, ki vidi prihodnost brez potrebe po sindikatih, ko bodo ljudje svobodno razpolagali s svojim delovnim časom. Njegovo delovno načelo je, da lahko za 10 penijev naredi to, kar zmorejo nesposobni narediti za en funt. Denar mu je le sredstvo za dosega končnega cilja, ne sredstvo moči; denar omogoča izdelavo uporabnih izdelkov, ob katerih ljudje uživajo. Razen svojih $ 18 milijonov ima Sinclair v svojem podjetju še 85% delež, vrednost podjetja pa je ocenjena na prek $ 200 milijonov.

Svoje prvo podjetje Sinclair Radionics je ustanovil že pri 22 letih, pred tem pa je bil novinar časopisa Practical Wireless. To podjetje je izdelovalo radijske sprejemnike in sestavljanke, ki so se pošiljale po pošti; tu je spoznaval metodologijo marketinga, ki je bila pomembna za njegov kasnejši poslovni uspeh. V začetku sedemdesetih let je Sinclair vstopil na tržišče potrošniške elektronike, ko je začel proizvajati nezanesljive kalkulatorje in digitalne ure. Tu se je naučil, kaj pomeni proizvodnja in pod kakšnimi pogoji jo lahko zaupa drugemu podjetju. Problemi dobave in kvalitete, upravljavske pomanjkljivosti in japonska konkurenca so povzročili krizo njegovega podjetja. Tu mu je pomagala britanska vlada z denarno injekcijo, tako da je Sinclair lahko prišel na tržišče z žepnimi televizorji. Vendar je Sinclair leta 1979 izstopil iz podjetja, ki ga je vodil odbor za upravljanje vladne investicije in ustanovil novo podjetje Sinclair Research, ki je začelo proizvajati prvi domači računalnik ZX80 za ceno pod $ 160.

Ta računalnik je bil za Sinclairja le vmesni začasni model v razvojnem ciklu, vendar je s hitrostjo prihoda na tržišče prehitel inozemsko konkurenco. V letu 1981 je že prodal 100000 računalnikov ZX80, pripravil pa je tudi že močnejšega naslednika ZX81, katerega cena je bila le še $75. Temu je sledil še ZX Spectrum, ki se je lahko uspešno kosal z računalnikom Commodore VIC 20. Sinclair je oddal proizvodnjo svojih računalnikov multinacionalnemu podjetju Timex, ki je kupilo tudi licenco za prodajo Sinclairovih računalnikov na ameriškem tržišču. V Združenem kraljestvu se je tako oblikovalo množično tržišče za domače računalnike, ki je močno dvignilo računalniško pismenost britanske populacije (na prvo mesto na svetu). Ob podjetju Sinclair Research je zrasla še vrsta drugih podjetij, ki so jih s pomočjo Sinclaira ustanavljali njegovi uslužbenci in novinci. Tako je Sinclair sprožil pravo poslovno in tehnološko inovacijsko verigo za proizvodnjo domačih računalnikov v Veliki Britaniji.

Tudi Sinclair je imel in ima vrsto zavistnih nasprotnikov v Veliki Britaniji. Ti so npr. poskušali izločiti njegov ZX Spectrum iz tkim. šolskega programa. Sinclair je takoj reagiral in znižal ceno za šole na 50%. Tako se je ta njegov daleč najcenejši domači sistem pojavil tedi na priporočilni listi vlade (za šolske računalnike). Medtem je Sinclair že začel investirati svoj zasebni kapital v razvoj električnega avtomobila in v razvoj novega, visokointegriranega mikroračunalniškega sistema, v katerem naj bi našel uporabo tudi njegov miniaturni televizijski sistem.

A.P.Železnikar

****
* Nekatere nove knjige *
****

L. Hancock, M. Krieger: The C Primer, Osborne/ McGraw-Hill, 1982, 235 strani, $ 14.95.

Ta knjiga je namenjena začetnikom v programiranju, ki tudi nič ne vedo o jeziku C. Ta jezik predpostavlja okolje operacijskega sistema Unix, vendar je knjiga bogata na primerih in pojasnilih, tako da je mogoča uporaba reduciranega jezika C v okolju Unix na računalniku VAX. Posamezna poglavja obravnavajo polja, strukture in kazalce in problematiko, ki je za začetnika nova. Pojasnjuje se rekurzija z uporabo preglednih primerov. Poglavje o vhodu/izhodu in knjižničnih funkcijah pokaže obliovanje nizov in števil.

T. Plum: Learning to Program in C, Plum Hall, 1 Spruce Ave, Cardiff, NJ 08232; 1983, 372 strani, $ 25.

Podjetje Plum Hall Inc. je specializirano za poučevanje in konzultacijo uporabe jezika C in operacijskega sistema Unix. Knjiga je učbenik za tečaje o jeziku C. Nekaj znanja z računalniškega področja je potrebno, vendar ne s področja programirnih jezikov. Poudarek knjige je na prenosljivosti programov v jeziku C in na primernosti tega jezika za sisteme realnega časa. Knjiga prinaša vrsto primerov in problemov za študenta in napotke za oblikovanje programske opreme. Odgovori na vprašanja se nahajajo v dodatku. Knjiga temelji na večletnih izkušnjah poučevanja uporabe programirnih jezikov.

J. Purdum: C Programming Guide, Que Corporation, 7960 Castleway Drive, Indianopolis, IN 46250, 1983, 250 strani, $ 17,95.

Ta knjiga je namenjena bralcu, ki ima nekaj znanja iz programirnih jezikov (iz Basica). Funkcije se programirajo za primerjavo v jeziku Basic in v C. Primeri so pisani za ceneni Small C prevajalnik (cena $ 19,95) in za dražji jezik C (Software Toolworks, cena $ 50). V dodatku je navedenih 13 C prevajalnikov za CP/M in za IBM PC. Ta knjiga obravnava tudi več primerov z operacijami nad diskovnimi zbirkami, še posebej v CP/M okolju.

D. E. Cortesi: Inside CP/M, with CP/M-86 and MP/M2, Rhinehart & Winston, 1982, 371 strani, $ 25,95.

To je eno najboljših del in priročnikov o sistemu CP/M doslej. Med vrsto storitev sistema CP/M obravanava tudi makrozbirnik MAC (Digital Research), oblikovanje in uporabo makrojev, njihovo vstavljanje v makrojsko knjižnico. Knjiga ima tudi natančno in podrobno informacijo o diskovnih V/I operacijah in načinu shranjevanja podatkov na diskih. Ima seznam funkcijskih pozivov in njihovo pojasnitev za sistema CP/M in MP/M.

A. R. Miller: Mastering CP/M, Sybex,Inc., 2344 Sixth Street, Berkeley, CA 94710, 1983, 398 strani, $ 16,95.

To je še eden, izčrpen programerski priročnik za CP/M. Uporablja se zbirni jezik in makroji pri predstavitvi različnih CP/M operacij in pri graditvi uporabne makrojske knjižnice. Potrebno pa je določeno znanje iz zbirnega jezika.

M. Waite, R. Lafore: Soul of CP/M, H. W. Sams & Co., Inc., 4300 West 62nd St., Indianopolis, IN 46268, 1983, 391 strani, $ 18,95.

Knjiga je namenjena programerjem v jeziku Basic in v drugih visokih jezikih, ki potrebujejo pozivanje zbirniških subrutin. Te subrutine so namenjene obdelavi diskovnih zapisov na način, ki ni dostopen visokim programirnim jezikom; tako se lahko poveča tudi hitrost določenih programov. Primeri so predvideni za uvrstitev v rutinsko knjižnico. Knjiga je dobro napisana in privlačna, ima vrsto diagramov, slik in programskih shem.

K. Barbier: CP/M Assembly Language Programming, Prentice Hall Inc., Englewood Cliffs, NJ 07632, 1983, 226 strani, $ 19,95.

Ta knjiga ne zahteva predhodnih izkušenj iz uporabe CP/M in zbirnega jezika. Avtor poučuje osnove programiranja v zbirnem jeziku CP/M okolja z uporabo zbirnika ASM.

W. Lowen: Dichtomies of the Mind, John Wiley & Sons, 1982, 314 strani.

Ta knjiga obravnava avtorjeve zamisli mišljenja, kot jih je mogoče razpoznati s stališča možganov, podobno kot materialna računalniška oprema razpoznava programsko (informacijsko) opremo. Avtor sicer razpravlja o računalnikih in o oblikovanju sistemov, vendar to ni njegova izhodiščna težnja. Avtorjev pristop, v kolikor sploh zadeva računalnike, je namenjen naravi oblikovanja konceptualnih procesov. Avtor uvaja 16-polni osebnostni model miselnih akcijskih procesov, s pomočjo katerih človek misli, ukrepa, sodeluje in inventira (domišlja). Vrsta bralcev bo pri branju knjige ugotovila: "To sem jaz. Želim, da bi jaz to izpovedal." Drugi pa bodo ugotavljali: "Sem zelo zapletena osebnost in se ne morem uvrščati v avtorjeve premočrtne modele." Vendar je mogoče v delu najti tudi lastno domišljanje, del osebne resnice skozi avtorjevo gledanje, ogledalo mišljenja samega. Z nadaljevanjem računalniške prispodobe pravi avtor: "Računalnik mora biti vodilo (ključ) do načina delovanja mišljenja, ne zato, ker je računalnik podoben mišljenju, marveč ker mišljenje gleda v svojo lastno strukturo, ko konceptualizira." Knjiga je zapleteno delo o snovi, o kateri smo včasih sposobni razmišljati, ki pa jo tudi večkrat odlagamo, ker je preveč zapletena. Avtor poskuša z vso resnostjo izravnavati skrivnosti nas samih.

H. M. Deitel: An Introduction to Operating Systems, Addison-Wesley P.C., 1983, 673 strani.

Knjiga je koristen priročnik in berilo za študenta in praktika s področja operacijskih sistemov. Je dobro organizirana, podrobna in sodobna, tako konceptualno kot implemetacijsko. Posamezna poglavja obravnavajo procesno upravljanje, pomnilniško upravljanje, procesorsko upravljanje, pomožno pomnilniško upravljanje in zmogljivost. Avtor ponazarja posamezne koncepte s podrobnimi študijami primerov skozi CP/M, Unix, Vax, MVS in VM.

A.P.Železnikar

Miro Germ, dipl. ing.
Sektor aplikacij
Tehnična informatika
ISKRA DELTA LJUB

PONATIS

sistemi delta
GLASILO UPORABNIKOV IN PROIZVAJALCEV

ŠTEVILKA: 2, LETO: 1
LJUBLJANA, 15. OKTOBER 1982

RAČUNALNIŠKO PROJEKTIRANJE

Povzetek:

Članek vsebuje kratek zgodovinski pregled razvoja metode končnih elementov, opis najbolj znanih programov za analizo modela s končnimi elementi in grafičnih paketov v CAD/CAM ter spisek proizvajalcev in tipov grafične opreme.

1. Metoda končnih elementov

V zadnjih letih je metoda končnih elementov postala zelo razširjena pri računalniških obdelavah kompleksnih problemov v inženirstvu. Metoda je posplošitev že prej uporabljenih tehnik analize, kjer so konstrukcijo predstavljali diskretni palični in gredni elementi. Uporabljeni so isti postopki za izračun elementov, le da namesto paličnih in grednih elementov uporabimo končne elemente za opis ravninskih stanj napetosti in deformacij, obnašanja plošč, lupin in kontinuuma.

V začetku razvoja metode je bil poudarek na razvoju učinkovitih elementov za reševanje konkretnih problemov iz prakse. Z razvojem digitalnih računalnikov se je kmalu pokazala uporabnost metode. Reševali so vedno večje ter kompleksnejše probleme. To je spodbudilo iskanje še uspešnejših numeričnih algoritmov za reševanje ravnotežnih enačb. Rezultat je kompleten sistem algoritmov, ki jih uporabljajo računalniki pri metodi končnih elementov. Numerični proces obsega tvorbo matrik, numerično integracijo za izračun matrik elementov, združitev matrik elementov v matriko konstrukcije in numerično reševanje sistema algebraičnih enačb. Raziskave tako na področju mehanike kot v matematiki so mnogo prispevale k razvoju in uporabi metode končnih elementov. Med leti 1850 in 1860 so bile zaokrožene teorije o torziji in upogibu gredi. Postavljeni so bili temelji področju strukturne analize.

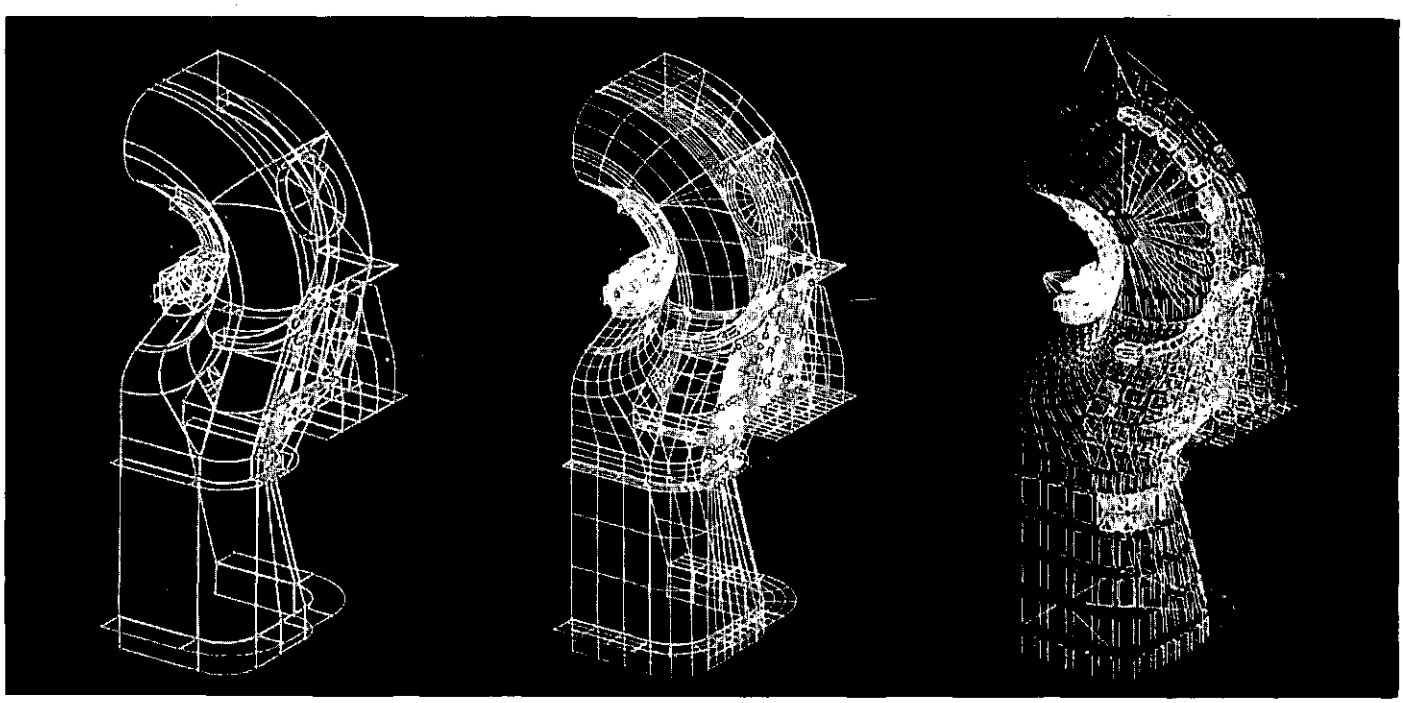
Nadaljnjih sto let je bil študij omejen na sisteme enodimenzionalnih paličnih in grednih elementov. Sredi petdesetih let našega stoletja so razvili dvodimenzionalne elemente za uporabo v letalski industriji, da bi tako izboljšali modeliranje tenkih membranskih elementov, ki so povezovali tradicionalne enodimenzionalne elemente. Leta 1960 je Clough prvi uvedel terminologijo te metode v članku »The Finite Element Method in Plane Stress Anallysis«. Posplošil je metodo strukturne analize na reševanje problemov v mehaniki kontinuumov.

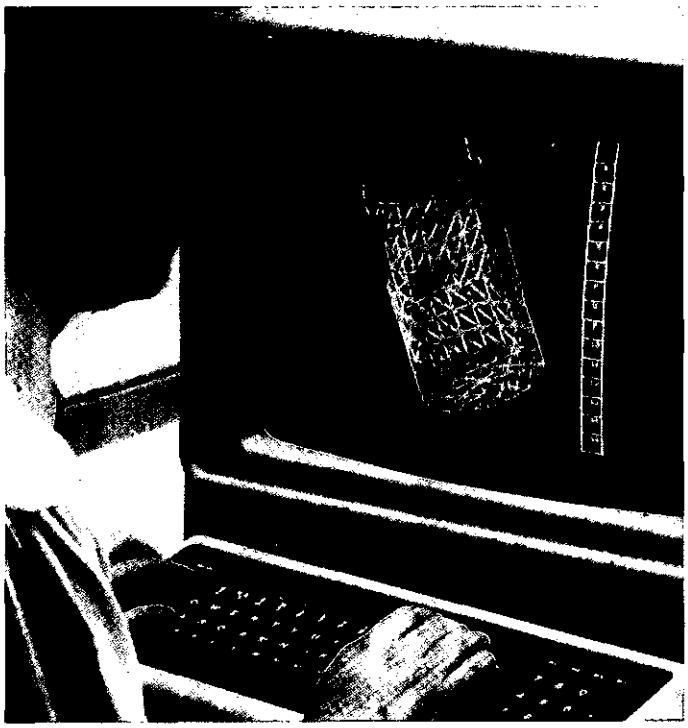
Že prej, leta 1909, je Ritz razvil zelo močno metodo aproksimacij polj v mehaniki kontinuuma. Aproksimiral je funkcional potenciala s testnimi funkcijami neznanih velikosti. Minimum funkcionala glede na vsako neznanko je sistem enačb neznanih velikosti. Velika omejitev metode je, da so funkcije morale zadoščati robnim pogojem problema. Courant je leta 1943 izboljšal Ritzovo metodo tako, da je uvedel ločene linearne funkcije na trikotnih območjih. Metodo je uporabil pri reševanju problema torzije.

Neznanke so bile vrednosti funkcij v stičiščih trikotnih območij. Tako je odpravil omejitev Ritzovih globalnih funkcij pri zadoščanju robnega pogoja, saj je tu potrebno zadostiti pogoju le v končnem številu točk na robu. Ritzova metoda, kot jo je uporabil Courant, je ekvivalentna metodi, ki jo je kasneje neodvisno razvil Clough. Seveda pa je razlog takojšnjega uspeha metode računalnik, saj je lahko opravil veliko število operacij, kar pa Courantu leta 1943 ni bilo dano.

Metodo so kmalu posplošili na tri dimenzije, na probleme nelinearnosti tako materiala kot geometrije ter na dinamiko. Razvoj je šel tudi preko meja strukturne analize na mehaniko fluidov, prenos toplote in na analizo magnetnih polj.

Danes je metoda polno uveljavljena v inženirski praksi.





2. Programi za analizo modela s končnimi elementi

Do nedavnega so le največji računalniki omogočali strukturno analizo, kinematični design, reševanje problemov polj, geometrijsko modeliranje, programiranje NC, mehanično testiranje in ostale obsežne operacije v CAD/CAM (Computer Aided Design/ Computer Aided Manufacturing).

Miniračunalniki sredi 70 let so lahko opravljali le določene specializirane inženirske izračune. 16-bitna beseda, omejen spomin in majhna hitrost računanja niso zadoščali množici podatkov v inženirski analizi.

Pojava takoimenovanih supermini računalnikov, njihova 32-bitna arhitektura, velik spomin in hitrosti procesorja enake največjim sistemom, je omogočila uporabo te metode tudi na manjših računalnikih. Ti sistemi imajo poleg občutno nižje cene in manjšega števila vzdrževalcev tudi druge prednosti. Omogočajo decentralizacijo računalniških zmogljivosti, tako da ni nevarnosti prekinitve dela zaradi okvare enega samega sistema. Hitrost analize se poveča, ker jo lahko izvajamo vzporedno na več sistemih; na enem teče priprava ali korekcija podatkov, na drugih pa računanje.

Danes uporabljajo za največje probleme mrežo superminijev, ki si delijo isto bazo podatkov. Vsak od njih pa obdeluje del problema iz področja designa, proizvodnje ali testiranja. Zaradi zgornjih razlogov raste število superminijev v CAD/CAM.

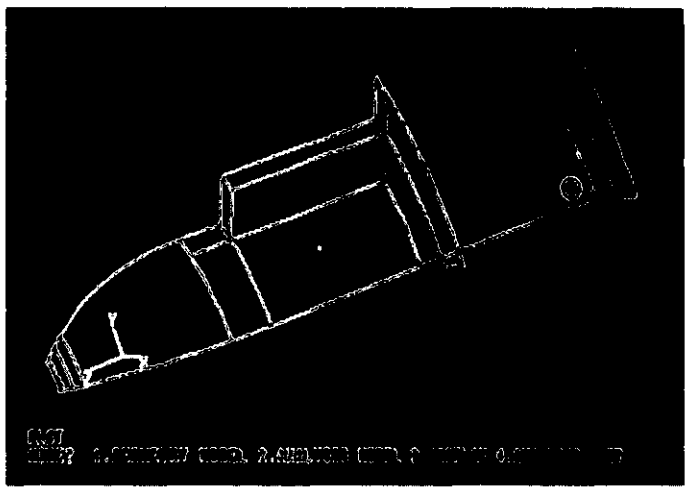
Oglejmo si nekaj v svetu najbolj uporabljenih programov, ki delajo na sistemih VAX in tudi na Delta 4780.

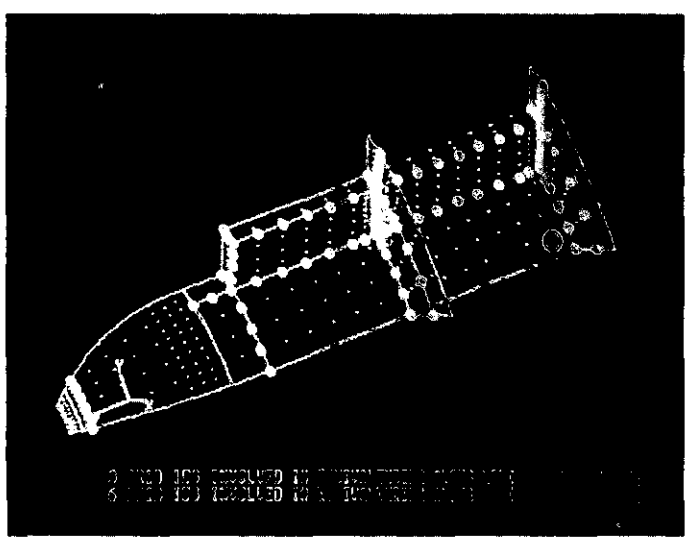
Najobsežnejši program s tega področja je NASTRAN. Razvit je bil pod sponzorstvom NASA za potrebe letalske in vesoljske industrije. Program uporabljajo za reševanje raznih problemov statične in dinamične analize, prenosa toplote, aerodinamike, akustike, elektromagnetizma, hidroelastičnosti ter ostalih polj. Vsebuje poseben macro jezik za nadziranje izvajanja. V komercialni uporabi je od začetka 70-tih let. Leta 1978 ga je firma MacNeal-Schwendler Corp. predelala za uporabo na računalnikih VAX.

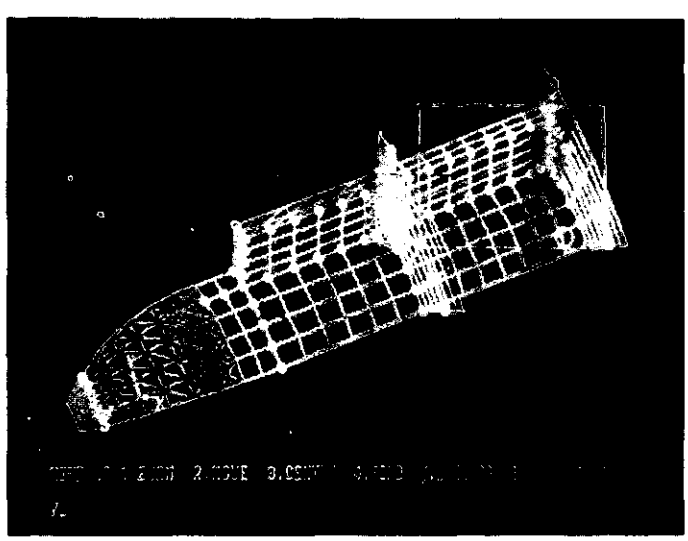
Drugi znani program s tega področja je ANSYS (Swanson Analysis Systems). To je splošni program za inženirsko analizo in je komercialno zelo razširjen. Obsega okrog 60 končnih elementov za analizo problemov strukturne analize, za linearne in nelinearne probleme statike in dinamike, prenos toplote, termoelektričnost, itd. Dopolnili so ga za reševanje posebnih nalog pri konstruiranju naftnih vrtalnih ploščadi. Izšla bo nova verzija programa na sistemih VAX z upoštevanjem vseh njegovih možnosti; obljubljajo 70 % hitrejše izvajanje.

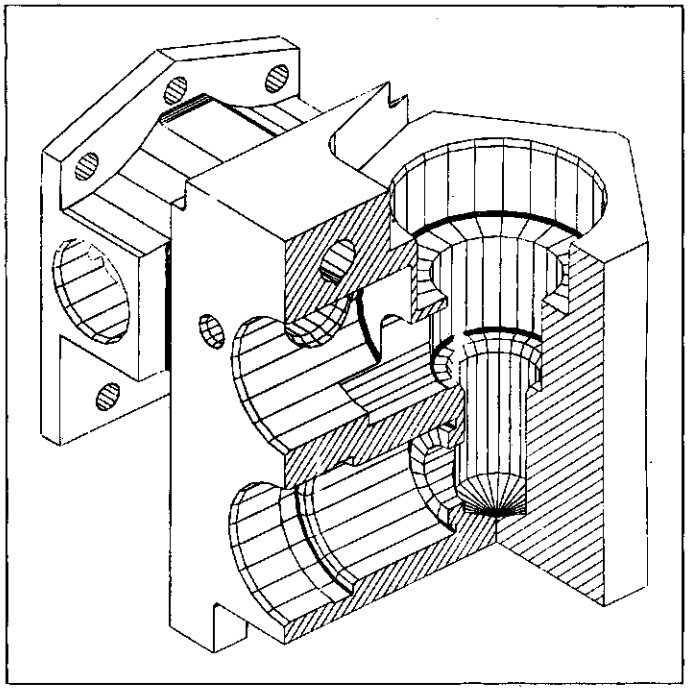
Eden najbolj znanih in razširjenih programov je STRUDL (Structural Design Language). Razvit je bil v letih 1960–70 v MIT za računalnike IBM. GTSTRUDL (GT pomeni Geogia Tech) je bil predelan v letih 1975–77 iz originalne IBM verzije kot del GTCES (Georgia Tec Integrated Computer Engineering System). Leta 1977 so ga začeli uporabljati na sistemih CDC CYBER, od 1981 pa je dostopen na sistemih VAX. Vsebuje preko 30 elementov, uporabljajo ga predvsem v gradbeništvu.

V svetu obstajajo tudi številni drugi programi. V našem prostoru sta najbolj znana SAP 4 in STRESS ter BERSAFE, kot del paketa za računalniško konstruiranje in projektiranje CETES-a (Center za tehnični software na Visoki tehnični šoli v Mariboru).









3. Priprava podatkov in prikaz rezultatov analize modela (CAD – Computer Aided Design)

Pri analizi modelov so ugotovili, da največ časa porabimo za pripravo podatkov, včasih tudi do 75 % časa celotne analize. Tu je tudi glavni vir napak. Uporaba grafične opreme in ustrezne programske opreme omogoča enostavno in hitro pripravo podatkov, barvni vizualni prikaz rezultatov ter tako zmanjša možnost napak, projektantu pa omogoči ustvarjalnejše delo.
Poglejmo si korake in možnosti, ki jih nudijo ti grafični procesorji pri modeliranju:

- Programska oprema omogoča enostavno komunikacijo med človekom in računalnikom preko izbirnega prikaza programov (menu). Uporabnik lahko poveča vsak del modela, lahko ga zavrti okoli poljubne osi modela ali zaslona, ki je lahko razdeljen na več samostojnih področij. Vsako področje je neodvisno in lahko prikazuje model iz različnih smeri, različno orientiran in različno povečan.
- Enostavno oblikovanje geometrije z grafičnimi ukazi ali preko digitalizatorja. Opisati telo je tako enostavno, kot opisati premico. Npr. z rotacijo točke dobimo krivuljo; z rotacijo krivulje definiramo ploskev; z rotacijo ploskve dobimo telo. Enako enostavno dobimo preseke med telesi, definiramo obtežbo in lastnosti materiala ter porazdelitev temperature po telesu.
- Poleg standardnih materialov lahko definiramo območja, kjer se lastnosti spreminjajo, podamo lahko sestavljene materiale, itd.
- Modele lahko spravimo v bazo podatkov za kasnejšo uporabo ali izboljšavo.
- Predprocesorji omogočajo avtomatično generiranje enakomernih in neenakomernih mrež z različnimi tipi elementov. Pri spreminjanju gostote mrež ni potrebno ponovno podati lastnosti materiala in obtežbe.
- Predprocesorji transformirajo te podatke v kartične slike raznih programov strukturne analize.
- Predprocesorji nudijo možnost raznih optimizacij numeričnih postopkov.
- Poprocesorji omogočajo prikaz različnih rezultatov; lahko si pogledamo deformacijo konstrukcije, napetosti, itd. Če uporabimo dinamične terminale, si lahko pogledamo tudi časovni potek deformacije. Barve uporabimo za prikaz določenih aspektov rezultatov. Programi uporabljajo barvni spekter za prikaz vrednosti rezultatov, lahko pa tudi sami izberemo barve za prikaz določenih vrednosti, itd.

Dva najbolj znana sistema na računalnikih DEC sta PATRAN-G (Prototype Development Associates, PDA) in GRAPHICS SYSTEM (Structural Dynamics Research Corporation, SDRC).

4. Projektiranje in priprava proizvodnje modela (CAM – Computer Aided Manufacturing)

Drugo veliko področje, kjer se uporablja grafika, je projektiranje, dokumentiranje in proizvodnja raznih strojnih in drugih delov. Programska oprema omogoča modeliranje, kotiranje, kreiranje ostale tehnične dokumentacije, arhiviranje, generiranje kode za stroje NC. Model, ki ga kreiramo na grafičnem zaslonu, lahko numerično vodeni stroji takoj začno proizvajati. Paketi nam po potrebi lahko izračunajo volumne, površine, momente, težišče modelov,itd.
Zaenkrat v svetu ne obstaja paket, ki bi enako dobro omogočal oboje, CAD in CAM; to je pripravo podatkov za analizo modela (CAD) in tudi kotiranje, izris presekov, pripravo ostale dokumentacije ter generiranje kode NC (CAM).
Na sistemih VAX uporabljajo pakete kot so EUCLID (Matra Datavision), ANVIL-4000 (Manufacturing and Consulting Services), SYSTRID (Baltele Research Centres), PALETTE (McLean Computer Consultants Pty, Ltd),itd.

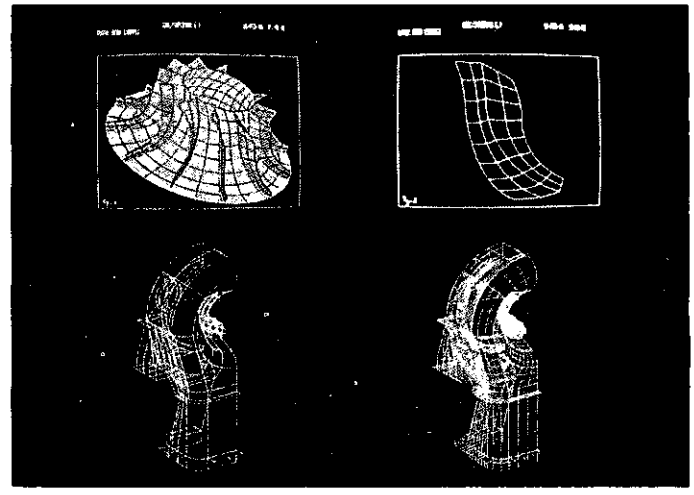
5. Grafična oprema

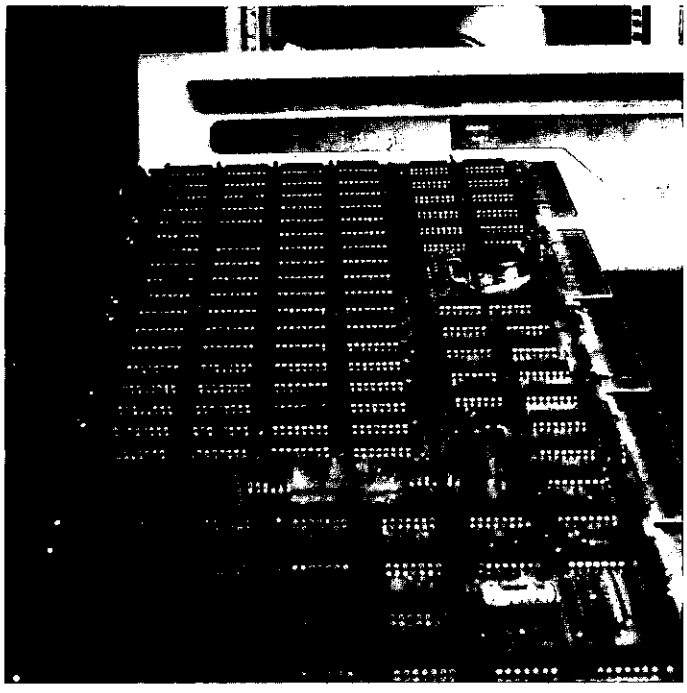
Najvažnejši del grafične opreme so terminali. Poglejmo si glavne proizvajalce in tipe barvnih terminalov:

— RAMTEK	serija 6000
	(6200 A najbolj razširjen terminal)
	serija 9000
— MEGATEK	serija 7000
	serija 7200
— AED	512
— LEXIDATA	3400
— DIGITAL	VS 11
— TEKTRONIX	4027
— EVANS & SUTHERLAND	PS-2
	PS-300

Terminai so vsi izdelani v tehnologiji »raster scan«, ki je znana že vrsto let, saj so jo uporabljali pri televizijskih sprejemnikih. Šele razvoj v tehnologiji chipov je omogočil uporabo te tehnologije tudi v računalniški grafiki.
Tovrsten grafični terminal vsebuje precejšen lasten spomin, kjer shrani informacije o sliki v točkasti matrični obliki. Elektronski žarek potem riše to sliko vrstico za vrstico v enakomernih intervalih 50-krat/s, ravno tako kot pri televizijskem sprejemniku.

SDRC – »GRAPHICS SYSTEM« (zgoraj)
PDA – »PATRAN-G« (spodaj)





Druga tehnologija je tehnologija »stroke« (črnobeli terminali TEKTRONIX), kjer so grafični ukazi shranjeni v spominu računalnika in se zaporedno zapišejo v krmilnik terminala (Direct Memory Access). Tam se pretvorijo v analogne napetosti. Te se potem uporabijo v katodni cevi za kontrolo elektronskega žarka, ki riše sliko. Ta tehnologija ima več pomanjkljivosti, kot so npr.: počasni odziv, slaba kvaliteta slike, obremenitve komunikacijskega kanala, itd. Naštejmo še nekaj proizvajalcev risalnikov. To so CALCOMP, VERSATEC, TEKTRONIX in tiskalnik/risalnik TRILOG ter PRINTRONIX.

6. Zaključek

To je bil kratek pregled možnosti, ki jih nudijo programski paketi in grafična oprema. Pri reševanju čedalje obsežnejših in vse bolj zapletenih problemov potrebuje konstrukter sodobno grafično opremo in ustrezne programske pakete. Naša družba bo morala, če hočemo postati konkurenčni tujini, projektantom tako računalniško opremo tudi zagotoviti. Neizdelan projekt, ki je danes predvsem posledica pomanjkljive strojne in programske opreme, ne more dati dobrega izdelka.

Literatura:

1. R. H. Gallagher: Finite Element Analysis, Prentice-Hall, 1975.
2. K. J. Bathe, E. L. Wilson: Numerical Methods in Finite Element Analysis, Prentice-Hall, 1976.
3. J. Cokonis: Minicomputers Tackle CAD/CAM, Machine Design, 1981.
4. H. Hamilton, L. M. Crain, E. L. Stanton: PDA/PATRAN-G, PDA, 1981.
5. Digital: Vax and Patran-G Engineering Tools for the 80's, 1981.
6. Digital: SDRC CAE on Vax, 1981.
7. Matra Datavision: Euclid, 1981.
8. Digital: MSC/NASTRAN, ANSYS, GTSTRUDL, PATRAN-G, AD-2000, PALETTE, SYSTRID, EUCLID, Engineering Systems bulletin.
9. Digital: Sales Update.
10. Swenson Analysis Systems Inc: ANSYS Analytic Capabilities, 1981.

IZDAJATELJ:
ISKRA DELTA
Služba tržnega komuniciranja
Parmova 41, 61000 Ljubljana

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 7, 1983 – Št. 4

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

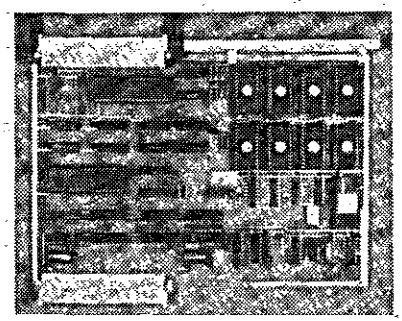
V S E B I N A

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 nasilno 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.