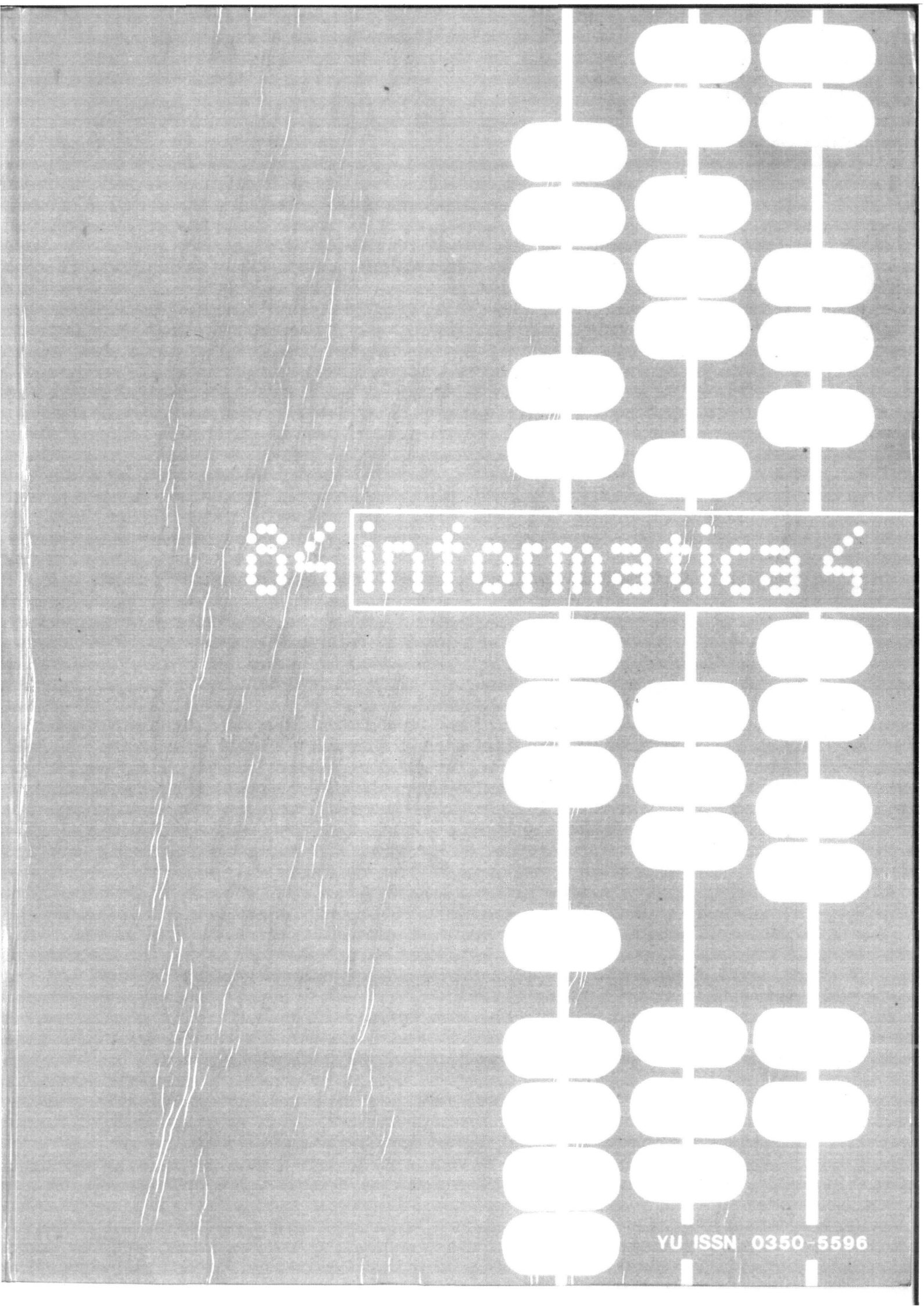
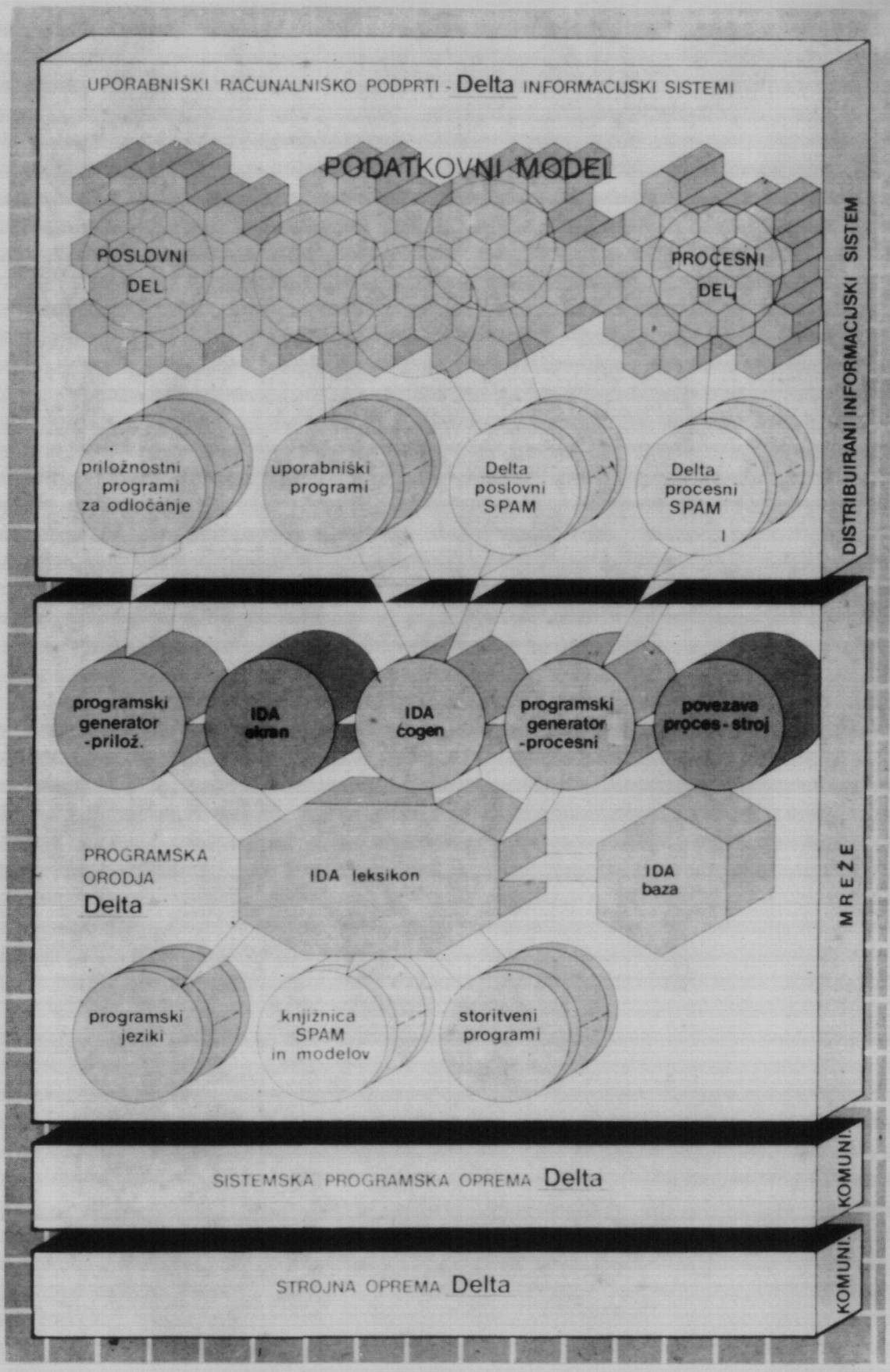


YU ISSN 0350-5596


UPORABNIŠKI RAČUNALNIŠKO PODPRTI - Delta INFORMACIJSKI SISTEMI
PODATKOVNI MODEL
POSLOVNI DEL
PROCESNI DEL
DISTRIBUIRANI INFORMACIJSKI SISTEM
priložnostni programi za odločanje
uporabniški programi
Delta poslovni SPAM
Delta procesni SPAM
programski generator -prilož.
IDA ekran
IDA čogen
programski generator -procesni
povezava proces - stroj
PROGRAMSKA ORODJA Delta
IDA leksikon
IDA baza
MREŽE
programski jeziki
knjižnica SPAM in modelov
storitveni programi
SISTEMSKA PROGRAMSKA OPREMA Delta
KOMUNI.
STROJNA OPREMA Delta
KOMUNI.

informatica

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

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 8, 1984 — No. 4

CONTENTS

H.Nežić	3	An Executive for Real Time Microcomputer Systems
D.Wechtersbach	13	Statistical Multiplexing
M.M.Miletić	17	PMP-11, 16-Bit-Microcomputer Compatible with PDP-11 Minicomputers
P.Kokol V.Žumer	20	LALR Tables Generator
A.P.Železnikar	26	Usage of Ada in Numeric Computations
D.Vukadin	39	Typewriter Interface for a Microcomputer
M.Tuta	43	Z800 - A New Microprocessor Family
S.Prešern	51	Visual Inspection of Printed Circuit Boards
A.P.Železnikar	55	Petra - An IBM-like Personal Computer I
R.Murn te All	68	Error Detection with Berger and Modified Berger Code
S.J.Djordjević	75	Keys Algebra
I.Mozetič	79	Principles of Qualitative Modeling
	86	Programming Quickies
	89	News
	98	Authors Index of the Volume

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. 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

H.Nežić	3	Primjer izvedbe izvršnog sistema za mikroračunala u realnom vremenu
D.Wechtersbach	13	Statistično multipleksiranje
M.M.Miletić	17	PMP-11, 16-bitni mikroračunar kompatibilan sa PDP-11 miniračunarima
P.Kokol V.Žumer	20	Generiranje LALR tabel
A.P.Železnikar	26	Uporaba jezika Ada v številskih izračunih
D.Vukadin	39	Priključitev pisalnega stroja na mikroračunalnik
M.Tuta	43	Nova družina mikroprocesorjev Z800
S.Prešern	51	Optična kontrola plošč tiskanega vezja
A.P.Železnikar	55	Ibmovski osebni računalnik Petra I
R.Murn in drugi	68	Odkrivanje napak z Bergovim in podobnimi kodi I
S.J.Đjordjević	75	Algebra ključeva
I.Mozetič	79	Principi kvalitetnega modeliranja
	86	Uporabni programi
	89	Novice in zanimivosti
	98	Avtorsko kazalo letnika

INFORMATICA 4/84

PRIMJER IZVEDBE IZVRŠNOG SISTEMA ZA MIKRORAČUNALA U REALNOM VREMENU

HRVOJE NEŽIĆ

ELEKTROTEHNIČKI INSTITUT „RADE KONČAR", ZAGREB

UDK: 681.519.7

U članku je opisan izvršni sistem za rad u realnom vremenu koji je razvijen sa ciljem da bude što brži, ali da ipak sadrži sve najpotrebnije funkcije kao što su aktiviranje i deaktiviranje procesa, kašnjenje, sinhronizacija procesa, obrada prekida itd. Izvršni sistem razvijen je prvenstveno za mikroračunala bazirana na mikroprocesoru Z8Ø, ali je prenosivost na druga mikroračunala moguća. U dodatku rada dani su algoritmi izvršnog sistema napisani u pseudojeziku sličnom ADI.

AN EXECUTIVE FOR REAL TIME MICROCOMPUTER SYSTEMS

The article describes a real time executive which is developed with wish to achieve high speed property while still including all necessary functions: activation, deactivation and sinchronization of processes, delay functions, interrupt handling etc. The executive is written basically for microcomputers based on Z8Ø microprocessor although portability on other systems is possible. The algorithms written in ADA-like pseudolanguage are given in the appendix.

1. UVOD

Upotreba mikroprocesora u upravljačkim i nadzornim dijelovima raznih uređaja, kao i u raznim mjerenjima, danas je vrlo rasprostranjena. U mnogim takvim primjenama programska podrška mora obrađivati razne zahtjeve iz vanjskog svijeta i reagirati na vanjske događaje. To ne predstavlja problem ako se u nekom vremenskom intervalu pojavljuje samo jedan događaj i ako je vrijeme između dva uzastopna događaja dovoljno da se izvrše sve operacije potrebne za obradu prvog događaja. Međutim, to često nije slučaj, tako da programska podrška mora obavljati mnogo raznih zadataka, koji, ako se promatra veći vremenski interval, moraju izgledati kao da se izvršavaju istovremeno iako svi oni zapravo dijele isti mikroprocesor. Zbog toga se u većini mikroprocesorskih sistema za rad u realnom vremenu javlja potreba za izvršnim sistemom koji će koordinirati rad paralelnih zadataka, no unatoč tome često se programska podrška za takve sisteme piše tako da su funkcije izvršnog sistema ugrađene u dotičnu aplikaciju što onemogućava strukturiranost, modularnost i prenosivost.

Razni zadaci koji se izvršavaju kvaziparalelno obično imaju različite funkcije, ali to ne znači da su oni potpuno nezavisni. Izvršni sistem u općem slučaju mora podržavati međusobno isključivanje zadataka kad oni rade nad zajedničkim sredstvima, međusobnu sinhronizaciju i komunikaciju zadataka, njihovo aktiviranje i deaktiviranje, kašnjenje, blokiranje zadataka do ispunjenja određenih uvjeta itd. Na taj način izvršni sistem uvodi red u sistem programske podrške aplikacije. Zadaci su oslobođeni od funkcija međusobnog koordiniranja kao i dodjele procesora i drugih sredstava, tako da svaki zadatak može biti napisan kao odjeljen sekvencijalni modul, a izvršni sistem se brine za sve ostalo ali nešto i uzima za to: određeni dio memorije i što je obično mnogo važnije, procesorsko vrijeme. Vremenski zahtjevi u sistemima realnog vremena u mnogim slučajevima su dosta strogi pa je poželjno da overhead koji unosi izvršni sistem bude što manji. Veća kompleksnost izvršnog sistema u pravilu znači veće vrijeme izvršavanja i time veći overhead. Postoji niz primjena koje imaju stroge vremenske zahtjeve, tj. zahtjevaju izvršavanje mnogih funkcija u relativno kratkim intervalima, a za koordiniranje funkcija dovoljan je relativno jednostavan izvršni sistem.

U ovom radu opisan je izvršni sistem koji je pisan upravo sa ciljem da zadovolji takve zahtjeve: posebna pažnja posvećena je nastojanju da sistem bude što brži, ali da ipak osigura sve neophodne funkcije.

Sistem je nastao prilikom razvoja jedne konkretne aplikacije tako da su u njega uglavnom uključene one funkcije koje su bile potrebne za tu aplikaciju i to u opsegu koji je bio potreban, ali je ipak većina njegovih funkcija dovoljna za upotrebu i u drugim sličnim aplikacijama; a razvijen je za mikroračunala bazirana na mikroprocesoru Z8Ø. Prenosivost na druga mikroračunala je moguća, ali su za mikroprocesore koji nemaju sekundarni skup registara potrebne izmjene u nekim rutinama jer je sistem koncipiran tako da zbog veće brzine rada koristi sekundarni skup registara mikroprocesora Z8Ø.

2. OPIS IZVRŠNOG SISTEMA

2.1. STANJA PROCESA

Paralelne zadatke programske podrške u daljnjem tekstu ćemo nazivati procesima. Zbog činjenice da dijele isti procesor, zbog prekida iz vanjskog svijeta, kao i zbog same prirode funkcija koje obavljaju, procesi tokom rada sistema u realnom vremenu, a pod kontrolom izvršnog sistema, prolaze kroz razna stanja. Svaki proces može biti u 4 globalna stanja:

- RUNNING (proces je u stanju izvršavanja, tj. ima kontrolu nad procesorom)
- INTERRUPTED (proces je prekinut i trenutno se izvršava prekidna rutina)
- READY (proces je spreman za izvođenje)
- NOTREADY (proces nije spreman za izvođenje) .

Pritom se stanja READY i NOTREADY dijele na nekoliko podstanja. Podstanja stanja NOTREADY su slijedeća:

- IDLE (proces je u besposlenom, neaktivnom stanju)
- DELAY (proces je u stanju kašnjenja - čeka da istekne određeni broj vremenskih intervala)
- WAIT (proces je u stanju čekanja);

dok stanje READY ima ova podstanja:

- START (izvršavanje procesa tek treba započeti sa njegovom prvom instrukcijom)
- FLOW (proces je već bio u toku kad je izgubio kontrolu nad procesorom - izvršavanje procesa treba se nastaviti na onom mjestu na kojem je izgubio kontrolu).

Prije nego što pređemo na detaljniji opis stanja procesa i načina prijelaza iz stanja u stanje, dati ćemo opis struktura podataka.

2.2. STRUKTURE PODATAKA

Izvršni sistem je baziran na strukturama podataka prikazanim na slici 1. Osim što sadrže prostor za spremanje registara procesora i služe kao stog za pojedine procese, strukture podataka čuvaju informacije o trenutnim stanjima procesa. Strukture podataka su organizirane tako da omogućuju što veću brzinu rada, tj. što brži pristup pojedinim podacima. Zato svi podaci koji se odnose na jedan proces nisu okupljeni na jednom mjestu, već su raspodjeljeni na više raznih polja kojima se može nezavisno pristupati. Trenutna stanja procesa opisuju polja XREADY i XSTATUS.
Polje XREADY čuva informaciju o globalnim stanjima procesa, dok polje XSTATUS sadrži informacije o podstanjima stanja NOTREADY i READY kao i dodatne informacije o stanjima procesa. Polja XREADY i XSTATUS sadrže po jednu 8-bitnu riječ za svaki proces. Svakom procesu pridjeljen je jedan fiksni 8-bitni broj koji se može shvatiti kao redni broj procesa ali ujedno označava i prioritet procesa. Raspodjela prioriteta je prema tome fiksna, tj. ne može se mijenjati u toku izvođenja. Broj koji označava prioritet procesa nigdje nije posebno spremljen, već predstavlja indeks za polja XREADY, XSTATUS, XADRDESKR i XTABSP. Proces sa najvećim prioritetom ima broj Ø, slijedeći ima broj 1, itd. Varijabla XN označava ukupni broj procesa. Pojedini član polja XREADY može sadržavati samo dvije vrijednosti: READY i NOTREADY. Stanja RUNNING i INTERRUPTED posebno se ne pamte, jer to nije potrebno. Dok je proces u nekom od tih stanja, na mjestu u polju XREADY koje odgovara tom procesu i dalje je upisano READY.

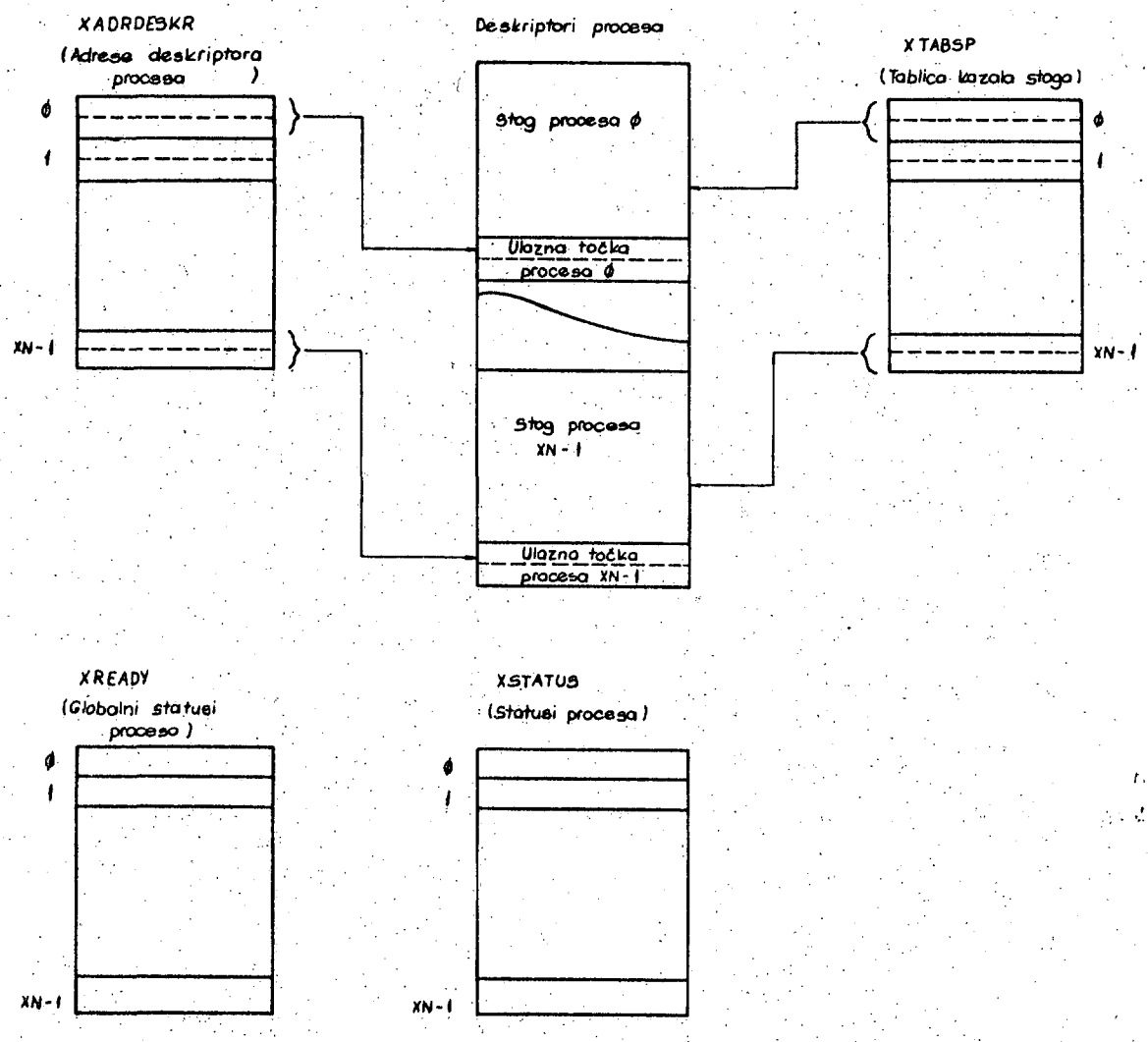
Polje XREADY koristi se samo za to da se ubrza traženje spremnog procesa sa najvećim prioritetom. Vrijednost NOTREADY jednaka je $ØC_H$ (kod instrukcije INC C), a vrijednost READY je $C9_H$ (kod instrukcije RET). Svaki put kad treba pronaći spremni proces s najvećim prioritetom izvode se instrukcije

```
LD   C,ØØ
CALL XREADY
```

Na taj način se za ispitivanje svakog nespremnog procesa troši samo 4 ciklusa, što je minimalno, a kad se dođe do prvog spremnog procesa u C registru se nalazi njegov broj. U svakoj riječi polja XSTATUS koristi se 6 bitova koji imaju slijedeća značenja: RESTART, START, RESTORE, IDLE, WAIT i DELAY.

Bitovi START, IDLE, WAIT i DELAY označavaju pojedina podstanja procesa (podstanje FLOW ne registrira se u posebnom bitu jer to nije potrebno) dok bitovi RESTART i RESTORE sadrže dodatne informacije o kojima će biti govora kasnije. Za svaki proces rezervirano je jedno memorijsko područje koje se naziva deskriptorom procesa.

ces koji je trenutno u stanju RUNNING, zamjeni neki drugi proces. Ako je proces prekinut ili prelazi u stanje čekanja tada trenutnu vrijednost kazala stoga treba spremiti da bi se kasnije kada proces bude ponovo raspoređen za izvođenje kazalo stoga moglo obnoviti. Vrijednosti kazala stoga spremaju se u polje XTABSP



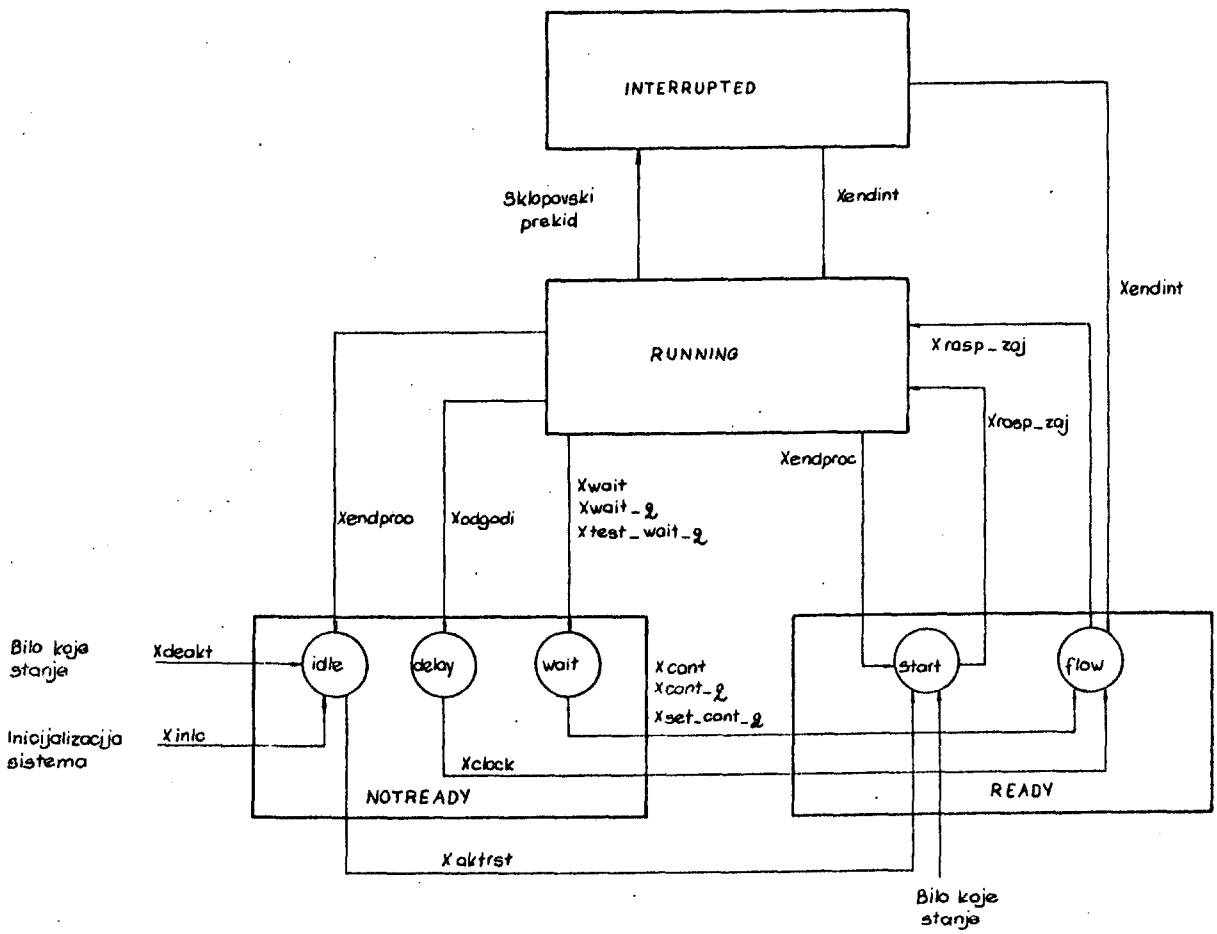
Slika 1 - Strukture podataka

Na kraju tog prostora spremljena je ulazna točka procesa, tj. kazaljka na prvu instrukciju procesa, a ostatak se koristi za spremanje registara procesa kada je proces prekinut, te kao stog koji proces može upotrebljavati na uobičajen način za pozivanje potprograma na više nivoa kao i za privremeno spremanje određenih vrijednosti. Prema tome, svaki proces ima svoj vlastiti stog, a pošto procesi u toku izvođenja mijenjaju stanja, mora se mijenjati i kazalo stoga svaki puta kada proces koji je trenutno u stanju RUNNING, zamjeni neki drugi proces. Ako je proces prekinut ili prelazi u stanje čekanja tada trenutnu vrijednost kazala stoga treba spremiti da bi se kasnije kada proces bude ponovo raspoređen za izvođenje kazalo stoga moglo obnoviti. Vrijednosti kazala stoga spremaju se u polje XTABSP koje sadrži po dvije 8-bitne riječi za svaki proces. Svaki put kad je proces raspoređen za izvođenje, a nalazi se u stanju START, kazalo stoga treba postaviti u početnu vrijednost za taj proces. Početna vrijednost kazala stoga postavlja se tako da prvi podatak koji se stavi na stog bude upisan ispred ulazne točke procesa (ona se nalazi na samom vrhu deskriptora). Početne vrijednosti kazala stoga spremljene su u polju XADRDESKR (mnemonik: adrese deskriptora) da se ne bi morale svaki puta ponovo izračunavati.

2.3. DIJAGRAM STANJA I KOMUNIKACIJA SA KORISNIČKIM PROGRAMIMA

Provođenje transformacija stanja procesa predstavlja glavni zadatak izvršnog sistema tako da dijagram stanja procesa prikazan na slici 2 ujedno predstavlja ilustraciju osnovne koncepcije izvršnog sistema.

Spomenimo na ovom mjestu da je predstavljeni izvršni sistem najprije napisan u pseudojeziku vrlo sličnom jeziku ADA (takav oblik izvršnog sistema prikazan je u dodatku), a nakon toga je translatiran u asembler. U asemblerskoj realizaciji komunikacija korisničkih programa sa izvršnim sistemom omogućena je pomoću makrodefinicija koje sadrži izvršni sistem.



Slika 2. - Dijagram stanja procesa

Na tom dijagramu prikazana su sva moguća stanja u kojima se procesi mogu nalaziti, te mogućnosti i načini prijelaza iz jednog stanja u drugo. Uz svaku strelicu koja prikazuje prijelaz iz jednog stanja u drugo napisana su imena programskih odsječaka izvršnog sistema koji provode određeni prijelaz, osim u jednom slučaju gdje promjena stanja procesa nije izazvana programski nego sklopovski - prekidom. Programski odsječci koje piše projektant programske podrške za rad u realnom vremenu (u daljnjem tekstu nazivat ćemo ih korisničkim programima) komuniciraju sa izvršnim sistemom pozivanjem njegovih potprograma ili predavanjem kontrole njegovim ulaznim točkama.

Makrodefinicije imaju ista imena kao pripadne procedure i ulazne točke napisane u ADI, a obično sadrže samo nekoliko instrukcija: zabranu prekida, punjenje određenih registara argumentima i poziv odgovarajućeg potprograma ili skok na odgovarajuću ulaznu točku izvršnog sistema. Na taj način korisnički programi komuniciraju sa izvršnim sistemom izvođenjem instrukcija makropoziva. U daljnjem tekstu, u cilju lakšeg razumijevanja rada izvršnog sistema, ponekad ćemo govoriti o makroinstrukcijama, a ponekad ćemo naglašavati da li se u konkretnom slučaju radi o potprogramu ili o ulaznoj točki. Za vezu između korisničkih programa i izvršnog sistema postoje slijedeće makroin-

strukcije (u zagradama su dani formalni argumenti):

- xinic
- xakt (proces)
- xaktrst (proces)
- xdeakt (proces)
- xwait
- xwait_q (rep)
- xtest_wait_q (uvjet)
- xcont (proces)
- xcont_q (rep)
- xset_cont_q (uvjet)
- xodgodi (broj_intervala)
- xendproc
- xendint
- xrasp

Korisnički programi mogu se podijeliti na tri kategorije s obzirom na način njihovog aktiviranja te komuniciranja s izvršnim sistemom:

- inicijalna procedura
- prekidne procedure
- procesi.

Svakoj od ovih kategorija dozvoljeno je korištenje samo određenog podskupa makroinstrukcija iz gornjeg popisa. Inicijalna rutina može koristiti makropozive

xinic, xakt, xaktrst i xrasp;

procesi koriste makropozive

xakt, xaktrst, xdeakt, xwait, xwait_q, xtest_wait_q, xcont, xcont_q, xset_cont_q, xodgodi, xendproc;

a prekidne rutine makropozive

xakt, xaktrst, xdeakt, xcont, xcont_q, xset_cont_q, xendint.

2.4. DIJELOVI IZVRŠNOG SISTEMA

2.4.1. Aktiviranje i deaktiviranje procesa

Prilikom inicijalizacije sistema, tj. svaki put kad se aktivira signal " Reset " mikroprocesora, potrebno je procese dovesti u određena početna stanja. Inicijalna rutina najprije pozivom potprograma XINIC postavlja sve procese u stanje IDLE. U stanju IDLE proces nema nikakvih zahtjeva: ne natječe se za dodjelu procesora, a isto tako ne čeka na ispunjenje bilo kakvih uvjeta - jednom rječju besposlen je. Iz tog stanja proces može doći samo u stanje START koje označava da je proces spreman za izvođenje i da će njegovo izvršenje započeti sa prvom instrukcijom kada bude raspoređen za izvođenje. Nakon što je dovela sve procese u stanje IDLE inicijalna rutina mora pozivima potprograma XAKT I XAKTRST neke procese aktivirati, tj. dovesti u stanje START. U sistemu mora postojati " dummy " proces, tj. proces koji ima najniži prioritet i nikad ne dolazi u stanje NOTREADY. Taj proces se izvršava uvijek kad svi ostali procesi nisu spremni za izvođenje. Inicijalna rutina mora ga obavezno aktivirati. Posljednja instrukcija inicijalne rutine je skok na ulaznu točku XRASP izvršnog sistema, koja pronalazi spremni proces s najvišim prioritetom i raspoređuje ga za izvođenje, tj. predaje mu kontrolu nad procesorom. Time je taj proces prešao u stanje RUNNING. Proces u izvođenju može pozivima potprograma XAKT, XAKTRST i XDEAKT aktivirati (dovesti u stanje START) ili deaktivirati (dovesti u stanje IDLE) bilo koji drugi proces.

Potprogrami XAKT i XDEAKT postavljaju specificirani proces u stanja START odnosno STOP bez obzira na to u kojem se je stanju proces prije nalazio. Za razliku od toga, potprogram XAKTRST neće odmah postaviti proces u stanje START ako je on već u toku, tj. započeo je izvršavanje, ali ga još nije dovršio. U tom slučaju procedura XAKTRST samo će postaviti RESTART bit u statusu procesa označivši na taj način da proces treba ponovo aktivirati kada on završi. Ako se proces nalazi u stanju IDLE tada će ga procedura XAKTRST odmah dovesti u stanje START.

2.4.2. Obrada prekida, raspoređivanje i završavanje procesa

Prekid iz vanjskog svijeta prebacuje proces iz stanja RUNNING u stanje INTERRUPTED. Sve prekidne rutine moraju se izvoditi potpuno neprekidivo, što znači da se ne dozvoljava prekidanje na više nivoa. Samim prihvaćanjem prekida sadržaj programskog brojila je spremljen na stog unutar deskriptora prekinutog procesa, a na početku prekidne rutine moraju se nalaziti instrukcije kojima se sadržaj primarnog skupa registara sprema u sekundarni skup (registri IX i IY ne spremaju se, tj. pretpostavlja se da ih procesi ne koriste u prekidivim dijelovima). Prekidne rutine mogu aktivirati ili deaktivirati bilo koji proces, a završavaju sa skokom na ulaznu točku XENDINT izvršnog sistema. Pošto je prekidna rutina mogla aktivirati neki proces koji ima viši prioritet od tekućeg, tj. prekinutog procesa, a mogla je i deaktivirati prekinuti proces, programski odsječak XENDINT najprije pronalazi prvi spremni proces i uspoređuje njegov broj sa brojem tekućeg procesa.

Ako je broj prvog spremnog procesa različit od broja prekinutog procesa tada će izvršni sistem rasporediti prvi spremni proces za izvođenje (programski odsječak XRASP_ZAJ) ali će prije toga (ako prekidna rutina nije aktivirala ili deaktivirala tekući proces) prebaciti sadržaj registara tekućeg procesa iz sekundarnog skupa u njegov deskriptor, postaviti RESTORE bit u statusu tekućeg procesa i spremiti kazalo stoga u polje XTABSP. Time je prekinuti proces prešao u stanje FLOW. Bit RESTORE označava da registre treba obnoviti iz deskriptora procesa kada taj proces bude ponovo raspoređen za izvođenje. U suprotnom, ako je tekući proces ujedno i prvi spremni proces, izvršni sistem obnavlja registre iz sekundarnog skupa i izvršavanjem instrukcije RET nastavlja prekinuti proces, ili (ako je prekidna rutina dovela tekući proces u stanje START) inicijalizira kazalo stoga i predaje kontrolu prvoj instrukciji tekućeg procesa.

Raspoređivanje prvog spremnog procesa za izvođenje, što izvršava programski odsječak XRASP_ZAJ sastoji se u slijedećem.Ako je proces u stanju START izvršni sistem inicijalizira kazalo stoga i predaje kontrolu prvoj instrukciji procesa, a inače obnavlja kazalo stoga i izvršavanjem RET instrukcije vraća se na prekinuto mjesto (ako je postavljen RESTORE bit, prije toga obnavlja registre iz deskriptora).

Proces završava svoje izvođenje sa skokom na ulaznu točku XENDPROC. Ako je postavljen RESTART bit u njegovom statusu proces će nakon toga prijeći u stanje START, a u suprotnom u stanje IDLE.

2.4.3. Odgađanje izvođenja procesa

U primjenama mikroračunala u realnom vremenu često je potrebno ostvariti kašnjenje, tj. odgađanje izvršavanja određenog programskog odsječka za neko vrijeme (tipičan primjer: neki proces treba upaliti LED diodu i nakon jedne sekunde je zagasiti). Naravno, zbog potrebe obavljanja drugih funkcija, proces najčešće ne može ostvariti potrebno kašnjenje vrteći se u petlji, pa je za rješenje tog problema potrebno predvidjeti druge mehanizme. Pod kontrolom ovog izvršnog sistema proces koji se trenutno izvršava može na bilo kojem mjestu odgoditi svoje izvođenje pozivom potprograma XODGODI (broj_intervala). Pritom mikroračunarski sistem mora sadržavati jedno programabilno brojilo (npr. ZILOG-ov CTC sklop) koje se koristi za ostvarivanje ovakvih kašnjenja. Za ostvarivanje kašnjenja koriste se i dvije varijable, tj. dvije memorijske lokacije, koje označavamo ovako:

XDELAY. BROJILO
XDELAY. REP .

Pošto u aplikaciji za koju je razvijen ovaj izvršni sistem nije bilo drugačijih zahtjeva, ovdje se pretpostavlja da u određenom trenutku samo jedan proces može biti u stanju kašnjenja, tj. u stanju DELAY, tako da je rep kašnjenja reduciran na samo jedno mjesto. Vjerojatno će u mnogim aplikacijama ovo biti dovoljno, no u mnogim neće, tako da će biti potrebno proširiti ovu koncepciju. Potprogram XODGODI puni varijablu XDELAY. BROJILO sa specifiranim brojem intervala, upisuje u varijablu XDELAY. REP broj procesa koji je pozvao potprogram, prebacuje proces u stanje DELAY, daje naredbu programabilnom brojilu da započne brojanje, omogućuje prekide od programabilnog brojila te raspoređuje za izvođenje prvi spremni proces. Nakon što je brojilo odbrojilo programirani broj impulsa ritma ono inicira prekid i ponovo počinje brojati impulse. Prekidna rutina XCLOCK najprije smanjuje iznos varijable XDELAY.BROJILO za 1. Ako je nakon toga varijabla XDELAY.BROJILO veća od Ø tada je rutina XCLOCK završena i brojilo će ponovo inicirati prekid nakon što odbroji programirani broj impulsa. U suprotnom, ako je XDELAY. BROJILO = Ø, to znači da je istekao željeni broj intervala pa rutina XCLOCK onemogućuje daljnje prekide iz brojila te prebacuje proces koji čeka u repu kašnjenja iz stanja DELAY u stanje FLOW (naravno, samo u slučaju da je taj proces još uvijek u stanju DELAY, jer je u međuvremenu mogao biti aktiviran ili deaktiviran).

2.4.4. Sinhronizacija procesa

Sinhronizacija procesa ostvaruje se pomoću potprograma XWAIT, XWAIT_Q (Rep) i XTEST_WAIT_Q (Uvjet) , pomoću kojih proces u izvođenju može sam sebe prebaciti u stanje WAIT, te pomoću potprograma XCONT (proces), XCONT_Q (Rep) i XSET_CONT_Q (Uvjet) koji mogu neki proces transformirati iz stanja WAIT u stanje FLOW. Potprogram XWAIT prebacuje proces iz stanja RUNNING u stanje WAIT i raspoređuje za izvođenje prvi spremni proces. Bilo koji proces ili prekidna rutina može nastaviti proces koji se nalazi u stanju WAIT, tj. dovesti ga iz stanja WAIT

u stanje FLOW izvođenjem potprograma XCONT (proces) . Potprogram XWAIT_Q (Rep) također prebacuje proces u izvođenju u stanje WAIT ali ga osim toga stavlja i u rep čekanja specificiran u pozivu potprograma. Repovi čekanja isti su kao i rep kašnjenja, tj. imaju samo jedno mjesto jer se pretpostavlja da u jednom trenutku najviše jedan proces može čekati u repu. Potprogram XWAIT_Q upisuje u specificiranu varijablu Rep broj tekućeg procesa. Pritom broj FF_H označava da je rep trenutno prazan. Potprogram XCONT_Q (Rep) prebacuje proces koji čeka u specificiranom repu iz stanja WAIT u stanje FLOW (naravno, to se izvršava samo ako rep nije prazan i ako se proces zaista nalazi u stanju WAIT). Argument potprograma XTEST_WAIT_Q (Uvjet) i XSET_CONT_Q (Uvjet) sastoji se od varijabli UVJET.VRIJEDNOST i UVJET.REP i donekle je sličan strukturi semafora, ali se i znatno razlikuje od njega. Varijabla UVJET.VRIJEDNOST je logička varijabla koja označava ispunjenje određenog uvjeta (npr. da je uređaj kojim upravlja mikroračunarski sistem došao u određeno stanje). Kad proces na nekom mjestu ne može nastaviti izvođenje sve dok se ne ispuni određeni uvjet on poziva potprogram XTEST_WAIT_Q (Uvjet) koji će ga, u slučaju da uvjet nije ispunjen, postaviti u stanje WAIT i u rep za taj uvjet (tj. upisat će njegov broj u varijablu UVJET.REP) te rasporediti prvi spremni proces, dok u suprotnom slučaju proces odmah nastavlja izvođenje. Bilo koji proces ili prekidna rutina može označiti da je uvjet ispunjen pozivom potprograma XSET_CONT_Q (Uvjet) koji upisuje u varijablu UVJET.VRIJEDNOST vrijednost " true " i ako neki proces čeka u repu UVJET.REP prebacuje ga iz stanja WAIT u stanje FLOW. Razni procesi i prekidne rutine mogu na uobičajen način (ne koristeći procedure izvršnog sistema) ispitivati, te postavljati u stanje " false " sve logičke varijable koje se koriste kao argumenti u pozivima potprograma XTEST_WAIT_Q i XSET_CONT_Q. Procedure za međusobno isključivanje procesa nisu realizirane u ovom izvršnom sistemu. Ako kritične sekcije nisu dugačke one se mogu izvoditi neprekidivo pa se međusobno isključivanje može ostvariti na taj način.

3. ZAKLJUČAK

Predstavljeni izvršni sistem uključuje većinu funkcija koje su obično potrebne za koordinaciju paralelnih zadataka u mikroračunarskim sistemima u realnom vremenu: aktiviranje i deaktiviranje procesa, obradu prekida, kašnjenje, sinhronizaciju procesa i drugo. Funkcije su ostvarene u onom opsegu koji je zahtjevala aplikacija za koju je izvršni sistem razvijen, tako da su npr. svi repovi ostvareni krajnje reducirano dok funkcije za međusobno isključivanje uopće nisu uključene. Međutim, opisani izvršni sistem može se relativno jednostavno proširiti sa takvim funkcijama, kao i sa potpunom realizacijom repova. Neke funkcije izvršnog sistema (npr. funkcija XAKTRST za aktiviranje, te funkcije za sinhronizaciju procesa) izvedene su na pomalo neuobičajen način, no takva koncepcija pokazala se je potrebnom i korisnom u spomenutoj aplikaciji.

LITERATURA

1. Wicklund T.:"MINI-EXEC: A Portable Executive for 8-Bit Microcomputers", Communications of the ACM, vol.25, number 11, November 1982.
2. Salčić Z., Štrkić G.:" Projektovanje izvršnih sistema mikroračunara za rad u realnom vremenu korištenjem jezika visokog nivoa za sekvencijalno programiranje", Informatica, 1/1982.
3. Holter J.:" Add Multiple Tasks to Your Communication and Control Program", BYTE, September 1983.
4. Salčić Z.:" Mikroračunarski sistemi: Arhitektura-programiranje-primjene", Svjetlost, Sarajevo, 1982.
5. Kvaternik R.:" Uvod u operativne sisteme", Informator, Zagreb, 1983.

DODATAK

U dodatku je priložena lista algoritama izvršnog sistema napisanih u pseudojeziku sličnom ADI. Lista je dana u obliku paketa koji sadrži niz procedura i blokova (što znači da nisu striktno slijedena pravila strukture programa u ADI) pri čemu blokovi predstavljaju one module izvršnog sistema kojima korisnički programi predaju kontrolu korištenjem JP instrukcija, za razliku od procedura koje korisnički programi pozivaju korištenjem CALL instrukcija. Osim toga, pošto su algoritmi pisani u pseudojeziku, a ne u čistom programskom jeziku, svi detalji nisu uvijek specificirani programskim konstrukcijama, već su umjesto njih upotrebljene rečenice govornog jezika. Tako npr. deskriptori procesa nisu uopće deklarirani jer naredbe u algoritmima koji se odnose na njih nisu programske naredbe nego obične rečenice. Uopće, prilikom pisanja algoritama naglasak je bio na jednostavnosti i preglednosti a ne na rigoroznosti. U deklaracijama se koriste tipovi " byte " i " word " koji označavaju jednu, odnosno dvije 8-bitne riječi, a koji inače nisu definirani u ADI. Zbog jednostavnosti u deklaracijama nije korištena specifikacija " access " koja je mogla zamjeniti specifikaciju " word ".

```
Package body IZVRŠNI_SISTEM is
   XN: constant byte := .....;
   prazan: constant byte := 16#FF ;
   Xadrdeskr: constant array ( Ø..XN-1 ) of
              word := ( ...., ......, .... );
   Xtabsp: array ( Ø..XN-1 ) of word ;
   Xready: array ( Ø..XN-1 ) of ( ready,
                                  notready );
   Xstatus: array ( Ø..XN-1 ) of
              record
                 restart, start, restore,
                 idle, wait, delay: boolean;
              end record ;
   Xdelay: record
              brojilo, rep : byte ;
           end record ;
   tekproc, proces : byte ;
   type two_byte_record is
      record
         vrijednost : boolean ;
         rep : byte ;
      end record ;

   Procedure XAKT (proces : byte) is
   Begin
      Xready (proces) := ready ;
      Xstatus (proces) := (start => true,
                           idle => false,
                           restart => false,
                           restore => false,
                           wait => false,
                           delay => false ) ;
   End ;

   Procedure XAKTRST (proces : byte) is
   Begin
      If not Xstatus. start (proces) then
         If Xstatus. idle (proces) then
            Xready (proces) := ready ;
            Xstatus (proces) :=
                        (start => true,
                         idle => false,
                         restart => false,
                         restore => false,
                         wait => false,
                         delay => false) ;
         else
            Xstatus.restart (proces):=true;
         end if ;
      end if ;
   End ;

   Procedure XDEAKT (proces : byte) is
   Begin
      Xready (proces) := notready ;
      Xstatus (proces) := (start => false,
                           idle => true,
                           restart => false,
                           restore => false,
                           wait => false,
                           delay => false) ;
   End ;

   << XENDINT >>
   Begin
      Pronadi prvi spremni proces;
      If proces = tekproc then
         If Xstatus. start (tekproc) then
            Xstatus.start (tekproc):=false;
            SP := Xadrdeskr (proces);
            EI ;
            goto Ulazna točka procesa;
         else Obnovi registre iz sekundar-
              nog skupa ; EI ;
```

```
        Vrati se na prekinuto mjesto procesa;
        end if ;
    else
        If not ( Xstatus. idle ( tekproc )
            or Xstatus. start ( tekproc ))
            then
            Prebaci registre iz sekundarnog
            skupa u deskriptor tekućeg
            procesa ;
            Xtabsp ( tekproc ) := SP ;
            Xstatus. restore ( tekproc )
            := true ;
        end if ;
        goto Xrasp_zaj ;
    end if ;
End ;

<< XRASP >>
   Pronadi prvi spremni proces ;
<< XRASP_ZAJ >>
Begin
    tekproc := proces ;
    If Xstatus. start ( proces ) then
        Xstatus. start (proces) := false;
        SP := Xadrdeskr ( proces ) ;
        EI ;
        goto Ulazna točka procesa ;
    elsif Xstatus. restore (proces) then
        Xstatus. restore (proces):= false;
        SP := Xtabsp ( proces ) ;
        Obnovi registre iz deskriptora
        procesa ;
        EI ;
        Vrati se na prekinuto mjesto procesa;
    else
        SP := Xtabsp ( proces ) ;
        EI ;
        Vrati se na prekinuto mjesto procesa;
    end if ;
End ;

<< XENDPROC >>
Begin
    If Xstatus. restart ( tekproc ) then
        Xstatus. restart (tekproc):=false;
        Xstatus. start (tekproc):= true;
    else
        Xstatus. idle ( tekproc ) := true;
        Xready ( tekproc ) := notready;
    end if ;
    goto Xrasp;  -- proces je mogao akti-
                 -- virati neki proces
                 -- većeg prioriteta ko-
                 -- jeg sada treba raspo-
                 -- rediti
End ;
```

```
Procedure XODGODI (broj_intervala:byte) is
Begin
    Xdelay. brojilo := broj_intervala ;
    Xdelay. rep := tekproc ;
    Xready (tekproc) := notready ;
    Xstatus. delay (tekproc) := true ;
    Xtabsp (tekproc) := SP ;
    Naredba CTC sklopu da započne brojanje;
    goto Xrasp ;
End ;

Interrupt CTC
Procedure XCLOCK is
Begin
    Xdelay. brojilo := Xdelay. brojilo - 1;
    If Xdelay. brojilo = Ø then
        Onemogući prekide iz CTC sklopa ;
        If Xstatus.delay (Xdelay.rep) then
            Xstatus. delay (Xdelay. rep)
                                := false ;
            Xready (Xdelay. rep) := ready;
        end if ;
    end if ;
    goto Xendint ;
End ;

Procedure XWAIT is
Begin
    Xready (tekproc) := notready ;
    Xstatus. wait (tekproc) := true ;
    Xtabsp (tekproc) := SP ;
    goto Xrasp ;
End ;

Procedure XWAIT_Q (Rep : byte) is
Begin
    Rep := tekproc ;
    Xready (tekproc) := notready ;
    Xstatus. wait (tekproc) := true ;
    Xtabsp (tekproc) := SP ;
    goto Xrasp ;
End ;

Procedure XTEST_WAIT_Q (Uvjet :
                    two_byte_record) is
Begin
    If not Uvjet. vrijednost then
        Uvjet. rep := tekproc ;
        Xready (tekproc) := notready ;
        Xstatus. wait (tekproc) := true ;
        Xtabsp (tekproc) := SP ;
        goto Xrasp ;
    end if ;
End ;
```

```
    Procedure XCONT (proces : byte) is
    Begin
        If Xstatus. wait (proces) then
            Xstatus. wait (proces) := false ;
            Xready (proces) := ready ;
        End if ;
    End ;

    Procedure XCONT_Q (Rep : byte) is
    Begin
        If Rep /= prazan then
            If Xstatus. wait (Rep) then
                Xstatus. wait (Rep) := false ;
                Xready (Rep) := ready ;
            End if ;
            Rep := prazan ;
        End if ;
    End ;

    Procedure XSET_CONT_Q (Uvjet :
                           two_byte_record) is
    Begin
        Uvjet. vrijednost := true ;
        If Uvjet. rep /= prazan then
            If Xstatus. wait (Uvjet. rep) then
                Xstatus. wait (Uvjet. rep) :=
                                          false;
                Xready (Uvjet. rep) := ready ;
            End if ;
            Uvjet. rep := prazan ;
        End if ;
    End ;
End Izvršni_sistem ;
```

STATISTIČNO MULTIPLEKSIRANJE

D. WECHTERSBACH

DO ISKRA DELTA

UDK: 681.3.519

V zadnjem času se terminali vedno bolj oddaljujejo od sostiteljskega računalnika k uporabnikom, povezava pa poteka prek telefonskih linij. Kadar je več terminalov na isti oddajni lokaciji, se v računalniških sistemih lahko uporabi statistično multipleksiranje. Ta članek opisuje koncepte statističnega multipleksiranja in izvedbo multipleksorja STDM Delta-MUX.

Statistical Multiplexing

In the last time, computer terminals are moving away from host computers to user workins places and the connections are realized by telephone transmission lines. If several terminals are located on one place the statistical multiplexing can be used. This article describes the notions of statistical multiplexing and the multiplexing equipment STDM Delta-MUX.

1. Uvod

V mladih letih računalništva je moral priti uporabnik k računalniku, da ga je lahko uporabljal, terminali pa so bili blizu racunalnika. Pozneje se je računalnik približeval uporabniku na ta način, da se so terminali preselili k posameznemu uporabniku. Povezavo je omogočal par modemov ter linija. Za več terminalov na isti oddaljeni lokaciji je bilo potrebno pač več modemov ter več linij, kar pa ni ravno ekonomično. V primeru, da je bil terminal teleprinter se je lahko uprabljal klasični PTT teleprinterski multipleksor. Vendar to ni zadovoljiva rešitev. Zato so se pojavili disitalni multipleksorji, prvi na principu TDM, ter kasneje okrog leta 1977 tudi statistični.

2. Multipleksor, TDM, STDM

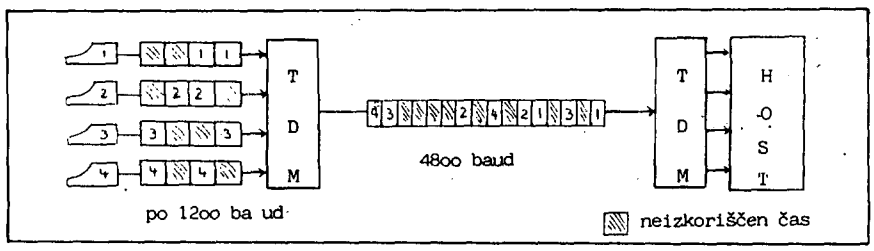
Vsak multipleksor združuje več vhodnih tokov v enega izhodnega. Obraten proces je demultipleksiranje. Obe napravi sta običajno združeni v eno, ki se kratko imenuje multipleksor. V računalniških komunikacijah združujemo disitalne znake - bite, ki so lahko organizirani v zloge ali besede z določenim pomenom (ASCII, strojna koda). Linije, ki jih združujemo, imenujemo vhodne linije (čeprav pri demultiplksiranju delujejo kot izhod), skupno linijo pa modemska. Multipleksor se lahko imenuje lokalni ali oddaljeni, odvisno pač od tega na kateri strani linije se nahajamo.

Prvotno se je v računalniških komunikacijah uporabljalo multipleksiranje z dodeljevanjem časa (časovni multipleks, TDM - time division multiplexing). Pri tem načinu se modemska linija časovno razdeli na več intervalov. Prvi interval se dodeli znaku (bit, zlog, beseda) iz prve vhodne linije, drugi iz druge, ter n-ti iz n-te vhodne linije, n+1 pa zopet znak iz prve vhodne linije (slika 1). TDM potrebuje modemsko linijo tako hitro, kot je vsota hitrosti vseh vhodnih linij.

Iz slike 1 se vidi, da ima TDM slabo lastnost, ker ne izkorišča modemske linije, če kaka vhodna linija nima znaka za prenesti.

V računalništvu pa nastopa še en problem: uporabnik namreč zelo neenakomerno zaseda linijo proti računalniku. V določenem trenutku uporabnik želi čim hitreje dobiti karakterje (presle-

Slika 1. Osnovna shema za TDM

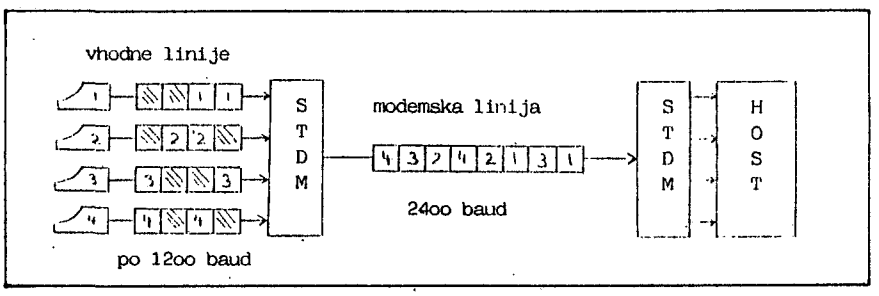


duje datoteko), nato pa nekaj časa linije sploh ne zaseda (usotavlja, kje je napaka). Oba problema, oni iz TDM ter slednji, sta komplementarna in se rešita v statističnem multipleksiranju (STDM, demand multiplexing, statistical multiplexing).

STDM je v osnovi še vedno TDM, le da se modemska linija dodeljuje tisti vhodni liniji, ki ima kaj za prenesti (slika 2). Tudi posamezni intervali sedaj niso več nujno enako dolgi. Modemska linija je sedaj polno izkoriščena do trenutka ko nobena od vhodnih linij nima več kaj za prenesti.

3. Značilnosti in možnosti

Jasno je, da inteligenco STDM omogoča mikroprocesor. Uporabljajo se zlasti 6800, 6809, 8085 ter Z80 pa tudi novejši 16-bitni. Njih število varira od 1 pa do 70 (Infotron) in več. Protokol med obema multipleksorjema je običajno sinhroni HDLC ali modificirani HDLC, v novejšem času pa že podpirajo 3 nivoji protokola X.25.



Slika 2. Osnovna shema za STDM z vhodnimi linijami, modemsko linijo in sostiteljem

Ti protokoli zlasti omogočajo skoraj 100% prenos brez napak, ker imajo priključen 16-bitni CRC na koncu vskega paketa. Tu se uporabljajo vezja 6854, 8273 ali Z80 - SIO. Količina RAM pomnilnika sega tja do 4k zlogov na vhodno linijo.

3.1 Kontrola pretoka

Prav pomnilnik STDM je pomemben za njegovo delovanje, potrebuje pa vsaj par vmesnikov na vhodno linijo. V določenem trenutku se lahko dosodi, da dve ali več vhodnih linij želi poseči po modemski liniji. Takrat mikroprocesor na podlagi prioritetnega algoritma odloči, kateri bo dal prednost. znak od druge pa se začasno shrani v vhodni vmesnik za kasnejšo oddajo.

Drugi vmesnik, ki tudi zlasti služi kontroli pretoka, pa je izhodni. Vanj se shranjujejo znaki, ki pridejo iz modemske linije, pa jih vhodna linija ne more oziroma noče (tiskalnik) prevzeti.

Vhodni vmesnik (statične dolžine ali pa dinamično raztegljiv) se lahko prenapolni v primeru, ko je modemska linija zelo obremenjena. Tedaj se vhodni liniji prepove pošiljanje novih znakov. Uporabljata se dva načina. Po vhodni liniji lahko odpošljemo ASCII znak CTRL/S (13H), pri drugem načinu pa spustimo EIA RS-232C CTS linijo. Običajno STDM podpira oba načina.

Po določenem času se vhodni vmesnik izprazni, takrat pa izvedemo obratno dejanje, torej ali odpošljemo CTRL/Q (11H) oziroma dvignemo CTS linijo. Vstavljanje ter ponoven pogon vhodne linije mora vsebovati določeno histerezo, če se želimo izogniti nezaželjenim oscilacijam.

Seveda se lahko prenapolni tudi izhodni vmesnik. Takrat se po kratkem internem pogovoru med parom multipleksorjev prepove oddaja znakov za to vhodno linijo v modemsko linijo ter se začne polniti vhodni vmesnik te vhodne linije. Ustavitveni pogoj (CTRL/S ali pa DTR) za polnjenje izhodnega vmesnika pa je lahko tudi transparenten. Npr., če v lokalni multipleksor v vhodno linijo odpošljemo CTRL/S potem se ta znak takoj prenese na isto vhodno linijo oddaljenega multipleksorja. Tudi pri izhodnem vmesniku je potrebno paziti na določeno histerezo.

3.2. Kode, hitrosti, zakasnilni čas, učinkovitost

Mikroprocesor ter bitno naravnani modemski protokol med multipleksorjema omogočata različne kode na vhodnih linijah (na istem multipleksorju) z različnim številom bitov za znak ter tudi z različnimi paritetami ter hitrostmi. Agregatna hitrost vedno presega hitrost modemske linije. Zelo važen je zakasnilni čas, ki ga povzroči multipleksor pri prehodu znaka. Podatki se preko modemske linije prenašajo v paketih in dokler ni sprejet cel paket, se ne ve, če je prenos v redu. Zato si želimo čim krajše pakete. Seveda ima vsak paket nekaj odvečnih znakov, ki jih zahteva protokol. Želimo si čim daljši paket, da je izkoristek paketa procentualno večji. Število zlogov na paket se tako siblje do 120 na paket, pa tudi več, zlasti pri satelitskih prenosih. Zakasnilni časi se sibljejo tako od dveh karakternih časov do 100ms in več.

Pri vseh teh primerih pa precej pomemben algoritem določevanja prioritete (katera vhodna linija lahko zasede modemsko linijo). Implementacija tega algoritma je jasno skrbno varovana, važni parametri pa so pretekla zasedenost modemske linije, zasedanje modemske linije, čas, polnost vhodnega vmesnika itd.

Učinkovitost se lahko poveča z kompresiranjem. Najbolj enostavno kompresiranje se dosodi že, ko asinhroni znak prenašamo kot sinhroni brez start in stop bita (20% pri 8 - bitni kodi). Nadalje se ponavljajoči znaki (pri tabelah) oddajo enkrat s ponavljalnim faktorjem. Najve-

čji prihranek se doseže s pomočjo Huffmanove kode, kjer se znakom, ki se večkrat ponavljajo, dodeli krajša koda, redkejšim pa daljša. Proizvajalec STDM trdi, da je s tem že dosegel kompresijo 1:2, kar pri kvaliteti telefonskih linij ni zanemarljivo. Jasno, da to zahteva vsaj primerjalne tabele in obilico procesorskega časa. Prednost je še v tem, da prinaša dodatno zaščito brez eksplicitne enkripcije, ki je tudi eden izmed problemov v komunikacijah.

3.3. Ekonomičnost

Mislim, da se iz opisanega vidijo predosti STDM nad TDM, zlasti v računalniških komunikacijah. Jasno je, da se klasični TDM lahko uporablja tam, kjer ne potrebujemo prednosti STDM (npr. stalno enak pretok podatkov v vhodnih linijah). STDM je popolnoma izrinil FDM, skoraj v celoti pa TDM. Gola ekonomika pa je približno vidna iz naslednje enačbe:

$$t <= (2*X-2*(n-1)*M)/(1*(n-1))$$

kjer je

- t - čas v letih, ko se nakup izplača,
- X - cena multipleksorja,
- M - cena modema,
- l - cena modemske linije na leto in
- n - št. vh. linij na multipleksorju

Pri tem bi še pripomnil, da pri nas primanjkuje telefonskih linij, in da je uporaba multipleksorja koristna tudi zaradi tega.

4. Delta - MUX

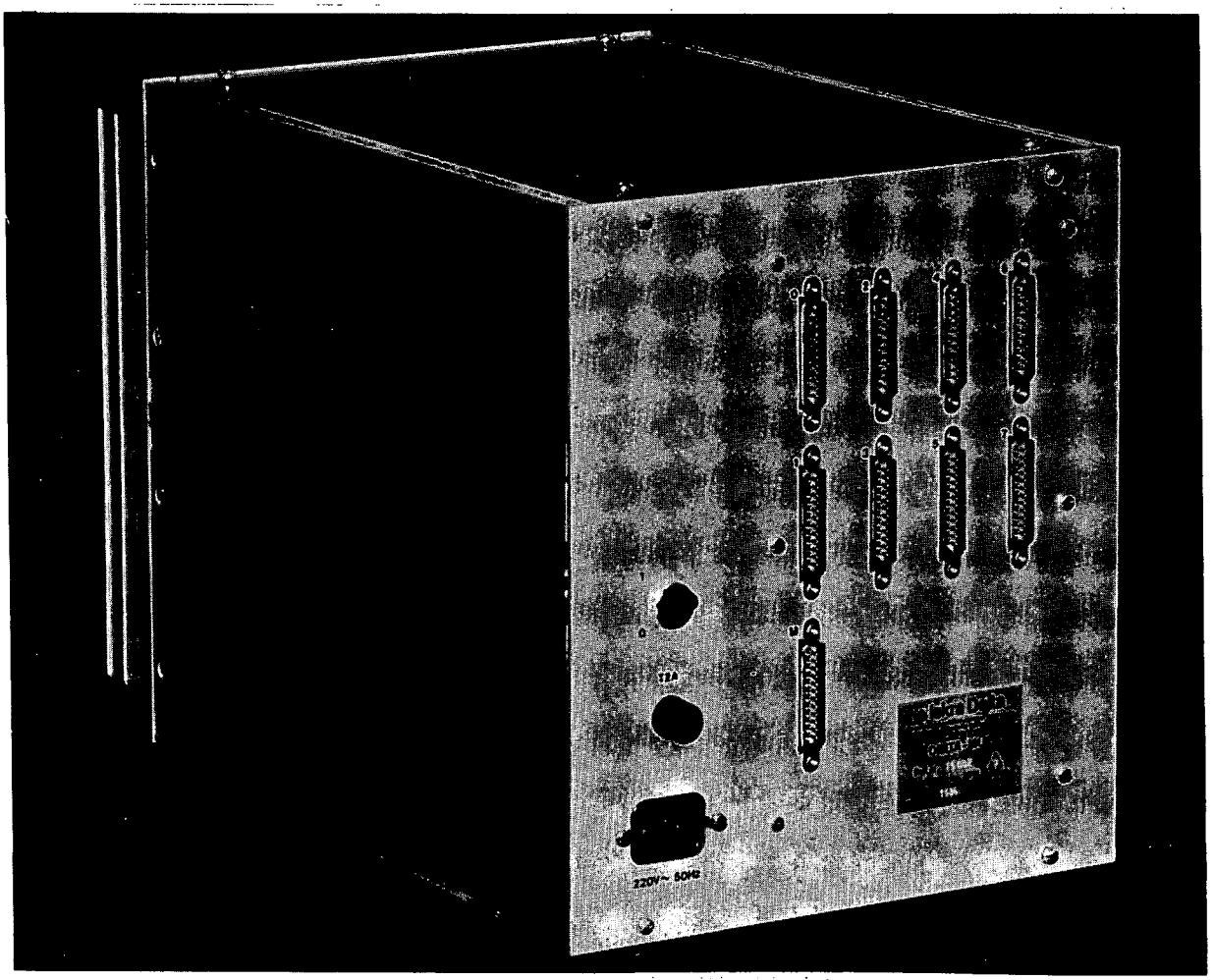
4.1. Splošno

V Iskri Delti se v redni proizvodnji nahaja STDM, ki je v celoti plod domačega znanja in izkušenj. Optimiziran je na interaktivno delo s terminali in ima zato majhen zakasnilni čas. Podpira 8 vhodnih linij (slika 3). Te so asinhrone, z ASCII kodo ter lahko delujejo z eno izmed 15 standardnih hitrostjo, a največ z 9600 baudov. Agregatna vhodna hitrost je torej 76,8k baudov in jo DELTA - MUX izdrži cca. 2 sekundi.

Modemska linija je sinhrona in deluje v modificiranem (zmanjšana je redundanca) protokolu X.25, nivo II, kar izboljšuje efektivnost. Največja hitrost je tudi tu 9600 baudov. Protokol podpira ARQ (Automatic request for repetition) in ima okno 7.

Od EIA signalov podpirajo vhodne linije TxD, RxD,RTS, CTS, DTR signale, modemska linija pa TxD, RxD, RTS, CTS, DTR, RxC, TxC (DTE) in TxC

Slika 3. Izvedba modula Delta-MUX (pogled na zadnjo ploščo)



(DCE) signale.

Vhodna linija št. 0 lahko postane kontrolna. V ukaznem načinu dela lahko uporabnik presleduje in spreminja parametre in statistične podatke. Za posamezno vhodno linijo lahko spreminja hitrost, velikost vhodnega vmesnika ter stalno prioriteto, ki sodeluje v algoritmu za določevanje dinamične prioritete. S slednjima se fleksibilno določa zmogljivost glede na priključeno periferijo.

Med statističnimi podatki naj zlasti omenim število karakterjev, ki bi jih Delta - MUX lahko prenesel, število vseh prenešenih karakterjev ter število koristnih karakterjev, ki so bili prenešeni. Iz tega lahko uporabnik hitro ugotovi kakšna je kvaliteta prenosa (ali je hitrost modemske linije prevelika oz. premajhna). Tu je še sedem vrst napak za modemsko linijo in kazalec napak za vsako vhodno linijo posebej.

4.2. Aparaturna oprema

Delta - MUX je sestavljen iz treh tiskanin. Prva je usmernik, ki je enak onemu v terminalu Paka 2000. Na drugi se nahajajo EIA konektorji ter vezja za pretvorbo EIA/TTL ter obratno. Tretja plošča pa je logična. Okoli 6808 mikroprocesorja je razvrščenih osem 6850 UARTjev ter 8k zlogov RAM in 4k zlogov ROM pomnilnika.

4.3. Programska oprema

Program je razdeljen v štiri dele. Prvi del servisira vhodne linije ter modemsko linijo. Karakterji se le shranjujejo v vmesnike ali jemljejo iz njih, tako da je izvajanje hitrejše. Nasploh je ta del programa časovno najbolj kritičen in je časovno minimiziran.

Drugi del programa ureduje posamezne vmesnike ter presleduje pakete. Na skrbi ima celotno kontrolo pretoka. Vodi vso potrebno statistiko in določuje dinamično prioriteto.

Tretji del je namenjen uporabniku, če se nahaja v ukaznem načinu dela. V tem delu se nahaja tudi nekaj rutin, namenjenih lažjemu servisiranju.

Zadnji del programa presleduje, če se celotni program izvršuje pravilno.

5. Sklep

STDM se hitro uveljavlja v računalniških komunikacijah. Izpodriva TDM, sam pa postaja zaradi čedalje močnejših mikroprocesorjev bolj in bolj inteligenten. Novejši STDM naj bi obvladali kompresijo, na modemski liniji pa naj bi podpirali protokol X.25. Nekateri posegajo tudi na trg PBX, kajti podpirajo tudi preklopne funkcije.

Delta - MUX lahko trenutno pokriva potrebe jugoslovanskega trga pri asinhronih terminalih. Ker sam popravlja napake (ARQ), je ob kvaliteti naših linij njegova uporaba celo zaželjena. Doseženi rezultati na terenu so bili dovolj dobri.

Literatura

1. Data Communications, razni članki,

2. Chu W. W., Proc. 1st and 2nd ACM symposium, Design Considerations of Statistical Multiplexers,

3. Davies D. W.,et all, Computer Networks and Their Protocols, John Wiley and Sons, 1979, str. 43-45.

PMP – 11, 16 – BITNI MIKRORAČUNAR KOMPATIBILAN SA PDP–11 MINIRAČUNAROMA

MARIJAN M. MILETIĆ

INSTITUT „JOŽEF STEFAN", LJUBLJANA

UDK: 681.3.06 PMP-11

Sadržaj: U članku je opisan prvi Jugoslovenski 16-bitni mikroračunar opšte namene.Korišćen je DEC mikroprocesor T-11 sa instrukcijskim setom kompatibilnim sa PDP-11 miniračunarima.Sistem ima 64 kB dinamičke memorije,2 kB ROM,dva serijska asinhrona kanala,dve diskete i časovnik realnog vremena. Operativni sistem je preraden RT-11,verzija 5.Glavna magistrala je 8-bitna, te je broj integriranih kola manji od 30.Performanse procesora su u klasi PDP-11/23 ili PDP-11/34 miniračunara.

PMP-11,16-BIT MICROCOMPUTER COMPATIBLE WITH PDP-11 MINICOMPUTERS is described in this article.It uses DEC T-11 microprocesor which has instruction set compatible with PDP-11 minicomputers.System has 64 kB of dynamic RAM, 2 kB of ROM,two serial asynchronous chanels,two diskettes and real-time clock.Operating system is modified RT-11,version 5.Bus is 8-bit wide,which reduces number of integrated circuits bellow 30.Processor performance is in class of PDP-11/23 or PDP-11/34 minicomputers.

1. UVOD

Prva generacija osmobitnih mikroračunara omogućila je masovnost upotrebe koja je prevazišla sva očekivanja.To je dovelo do pada cene diskova te razvoja kompleksnih aplikacionih programa.Tu su se iskazale slabosti bajtovske arhitekture u radu na bazama podataka,komunikacijama i grafici.

Nije slučajno da je IBM PC koncipiran sa 16-bitnom arhitekturom.Značajno smanjenje cene materijalne opreme postiže se osmobitnom magistralom.

U Jugoslaviji su razvijeni 16-bitni sistemi posebne namene sa INTEL 8086 i MOTOROLA 68000 mikroprocesorima.Sistemska programska oprema ovih sistema je relativno skromna.

Mi smo pošli drugim putem i izbrali DEC T-11 mikroprocesor koji ima instrukcijski set kompatibilan sa PDP-11 miniračunarima.Na taj način imamo na raspolaganju 15 godina programa te široku bazu korisnika po celoj Jugoslaviji. Na našem Institutu imamo programske prevodioce za sve važnije visoke jezike : COBOL,FORTRAN, PASCAL,LISP,PROLOG itd. Jednostavni komunikacioni programi omogućavaju prenos podataka i na 32-bitni superminiračunar VAX-11.Koristimo operacioni sistem za rad u realnom vremenu RT-11 koji podržava izvršavanje dva programa.

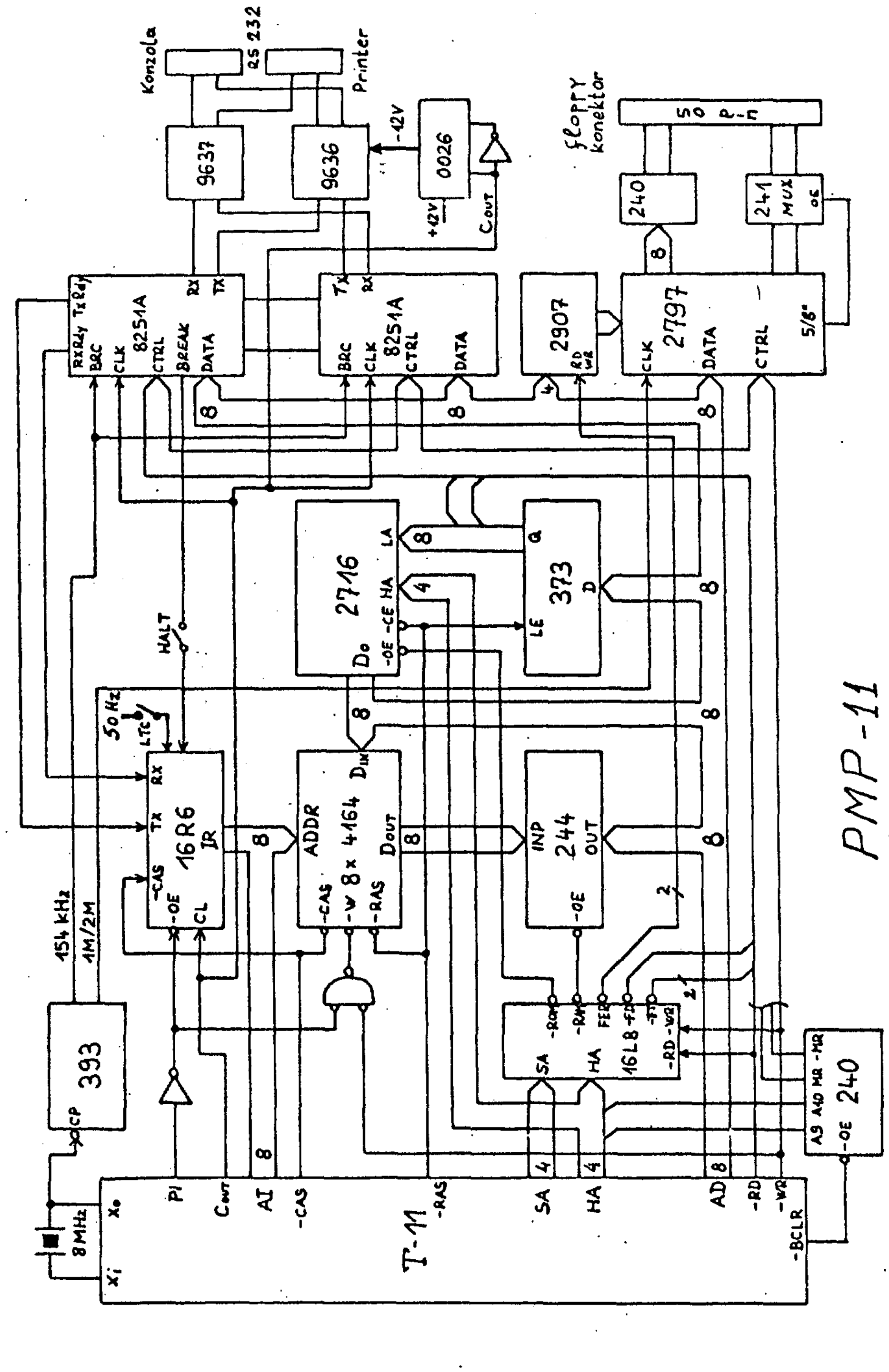
PMP-11 je prvenstveno zamišljen kao profesionalni,personalni 16-bitni mikroračunar za naučni rad i školovanje.Sistem ima 64 kilobajta dinamičke memorije,dve minidiskete,dva serijska standardna komunikaciona kanala te časovnik realnog vremena sa mrežnom učestanošću.Time je omogućen rad sa širokim spektrom videoterminala i štampača.Radi prenosa programa sa PDP-11 miniračunara moguće je priključenje i 8" diskete.U razvoju su kontroleri za tvrdi disk od 10 Mb,video izlaz za kolor TV,paralelni TTL kanali,instrumentaciona magistrala te priključak za tastaturu.

T-11 ima elektronsku podršku osvežavanja dinamičkih memorija.Izborom 8-bitne magistrale broj integriranih kola je manji od 30.Štampana pločica se montira iznad minidiskete tako da je ceo sistem moguće ugraditi u standardnu 19" kutiju.Troši manje od 40 W što omogućava napajanje i iz 12 V akumulatora.

2. MATERIJALNA OPREMA

Blok dijagram mikroračunara PMP-11 dat je na slici 1.

Ustanovili smo da T-11 pouzdano radi na 8 MHz što je iznad specifikacija. Časovnik delimo sa 16 za potrebe disketnog kontrolera te sa 13


PMP-11
T-11
8 MHz
393
154 kHz
1M/2M
16R6
8 x 4164
2716
373
244
16L8
240
8251A
8251A
2907
2797
9637
9636
0026
241 MUX
Konzola
RS 232
Printer
floppy konektor
50 pin
HALT
50 Hz
+12V
-12V

za serijski prenos sa standardnim brzinama.Negativni napon za RS-232 predajnik generiramo na učestanosti internog časovnika mikroprocesora.

Pri uključenju napajanja iniciraju se kontroleri serijskih linija i kontroler diskete. Istovremeno se aktivira registar načina rada koji specificira da T-11 radi na 8-bitnoj magistrali sa 64 kilobita dinamičkim memorijskim kolima.

Ciklusi osvežena su automatski insertirani po izvođenju svake instrukcije.

Adrese su dekodirana sa programiranim logičkim nizom i vrše selekciju primarne memorije od 000000 - 172000,startnog ROM-a od 172000 - 176000 te perifernih kontrolera u preostalih 10000 oktalnih adresa.

Dinamičke memorije su adresirane sa AI linijama koje strobiramo sa RAS i CAS.

Signal za upisivanje je zakašnjen sa PI da bi imali podatke spremne na ulazu. Izlazi iz memorijskih kola su stoga odvojeni od skupne magistrale.

Dva serijska kontrolera su izvedena sa standardnim 8251 kolima.Brzina prenosa se reguliše izborom faktora deljenja sa kratkospojnicima i programski.Obično se uzima frekvencija od 154 KHz sa internim deljenjem sa 18 za 9600 boda.

Zahtevi za prekid se upisuju sa svakim impulsom časovnika T-11 u 6-bitni registar programirane logike.Kad se pojavi CAS,registrirani zahtevi se sortiraju po prioritetu internih vektora prekida te signalom PI proslede na AI linije.Pomoću napona mreže generiraju se časovni prekidi svakih 20 ms. PMP-11 je moguće zaustaviti prekidačem ili pritiskom na tipku BREAK konzole. U oba primera se generišu prekidi koji se programski analiziraju.Jedan PAL zamenjuje najmanje 4 MSI TTL kola i za ovo rešenje je zahtevan patent.

Logika za kontrolu napajanja koristi monostabilni multivibrator okidan mrežnom učestanošću.Nestanak dva mrežna ciklusa prouzročava nemaskiran prekid na standardnoj adresi 24 i sprečava dalji,nepouzdan rad sa disketama.

Disketni kontroler je potpuno integrisan u kolu WD2797.Četvorobitni registar koristimo za selekciju dva pogona te parametara kontrolera. Jedan konektor sa 50 nosica se koristi za obe dimenzije diskete,a specifični signali se multi- pleksiraju sa dva 4-bitna odvajača.Osmobitni invertor se koristi za rad sa disketnom magistralom impedanse 150 Oma.Prenos podataka je pod procesorskom kontrolom bez korišćenja prekida.

3. PROGRAMSKA OPREMA

Operativni sistem RT-11 verzija 5 je modificiran zbog specifičnih rešenja na PMP-11.

Razlike su u rutinama za konzolu zbog drugačijeg rasporeda kontrolnih i komandnih bitova 8251 kontrolera.Pri inicijalizaciji ovo kolo zahteva prvo komandu načina rada,a zatim se dozvoljava rad prijemnika i predajnika.

T-11 se razlikuje od ostalih PDP-11 po reakciji na zamke /trap/.Tad se izvođenje programa nastavlja na adresi koja je za 4 veća od startne, a to je u našem slučaju 172004.Potrebna je programska analiza uzroka pada u zamku i ta je potpuno odvisna od organizacije mikroračunara.

Napisan je nov program za upravljanje inteligentnim kontrolerom diskete. Sva pomeranja glava se verificiraju na željenom tragu i po potrebi ponavljaju u obe smerove zapisa.Greške u čitanju se popravljaju bez pomeranja glava. Optimiziran kod za upravljanje disketama smešten je u ROM jer se obavezno koristi pri startu sistema.

Razvijena su dva programa za formatiranje disketa koja rade sa IBM standardom koji je istovremeno i DEC na 8" disketama sa jednostrukom gustinom zapisa.Na taj način je omogućen prenos programa između PMP-11 i PDP-11 sistema.

4. ZAKLJUČAK

U slučaju da je vreme izvršenja instrukcija veće bar dva puta od vremena pripreme,gubi se manje od 30 % na brzini izvršavanja instrukcija zbog dvojnog pristupa po osmobitnoj magistrali radi kompletiranja 16-bitne reči mikroprocesora.

Jednostavan test sa petljom od milion instrukcija u Pascalu pokazuje da PMP-11 procesor radi sa istom brzinom kao PDP-11 modeli 23 ili 34.

Iako je PMP-11 ograničen na 64 kilobajta memorije izvedbom T-11,to nije veliki nedostatak jer inherentna PDP-11 arhitektura podržava samo 16 adresnih bitova te nijedan pojedinačan program nemože biti veći od 64 KB. Za multiprogramski rad potrebna je jedinica za upravljanje memorijom koja vrši segmentaciju do 4 MB.

DEC prodaje i J-11 mikroprocesor mnogo većih mogućnosti no mi smatramo ekonomski neopravdanim korišćenje ovog mnogo skupljeg kola u personalnom mikroračunaru PMP-11 klase.

Na kraju bih želeo da zahvalim Branku Jevtiću za modifikaciju sistemskih programa te Iskri-Delti za materijalnu pomoć.

GENERIRANJE LALR TABEL

PETER KOKOL
VILJEM ŽUMER

UDK: 681.3.06:808.61

VISOKA TEHNIŠKA ŠOLA MARIBOR, VTO ELEKTROTEHNA MARIBOR

V članku na kratko opisujemo postopek za generacijo LALR(1) tabel, dodajamo osnovne algoritme in procedure. Generiranje LALR(1) tabel smo preverili za različne jezike s tem, da je potrebno kot vhodni podatek vnesti gramatiko ustreznega jezika v BNF meta jeziku. Na koncu dodajamo krajši primer.

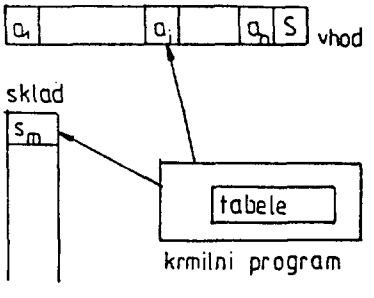
LALR TABLES GENERATOR. Algorithms and procedures for LALR(1) tables generator are given. To produce the LALR(1) tables for some language we must input the grammar of this language in BNF meta language. The described generator was tested for different languages. A short example is given at the end of this paper.

1. UVOD

Eden prvih korakov pri implementaciji jezika je sintaksna analiza oz. razpoznavanje jezika. V zadnjem času se vse več uporabljajo LR razpoznavniki, ki imajo več prednosti pred ostalimi. LR razpoznavnik je zelo preprost program in splošen za vse jezike, pač pa zahteva pri svojem delu obsežne tabele. Te LR tabele vsebujejo lastnosti posameznega jezika in jih je potrebno generirati za vsak jezik. Glede na večino današnjih programskih jezikov lahko obsežne LR tabele skrčimo na LALR(1) tabele, ki zavzamejo mnogo manj prostora. Generiranje teh tabel je zamudno opravilo, ki se ga da mehanizirati. Osnovna naloga razpoznavnika je učinkovito odkrivanje in javljanje sintaksnih napak. Zato morajo LALR(1) tabele vsebovati podatke za odpravljanje napak.

2. LR razpoznavniki

V splošnem je razpoznavnik sestavljen iz dveh delov: iz tabel in programa, ki ga te tabele krmilijo. Za različne gramatike se spreminjajo le tabele. Zgradba razpoznavnika je prikazana na sliki 1.



Slika 1: Zgradba razpoznavnika

Razpoznavnik ima sklad, vhod in tabelo. Vhod se bere iz leve proti desni, simbol za simbolom. Sklad vsebuje niz stanj $s_0\ s_1\ s_2\ s_3 \cdots s_m$, kjer je s_m na vrhu sklada in vsebuje informacije o skladu.

Tabela je sestavljena iz dveh delov, iz funkcij ACTION in GOTO. Funkcija ACTION (s_m, a_i) ima lahko naslednje vrednosti:

1. pomakni s
2. reduciraj $A \rightarrow \gamma$
3. sprejmi
4. napaka

Funkcija GOTO (s_m, a_i) generira novo stanje s in predstavlja tabelo prehajanja stanj determinističnega končnega avtomata, katerega vhodna abeceda so simboli gramatike.

Konfiguracija LR razpoznavnika je dvojica:

$(s_0\ s_1\ s_2 \cdots s_m, a_i\ a_{i+1} \cdots a_n\ \$)$

Prva komponenta je vsebina sklada, druga komponenta pa še ne prebranidel vhodnega teksta. Naslednja akcija razpoznavnika je odvisna od trenutnega vhodnega simbola a_i in stanja na vrhu sklada s_m. Konfiguracija po tej akciji je glede na štiri vrednosti funkcije ACTION naslednja:

1. če je funkcija ACTION (s_m, a_i) = pomakni s, razpoznavnik izvede pomik in dobimo naslednjo konfiguracijo

   $(s_0\ s_1\ s_2 \cdots s_m\ \ s, a_{i+1} \cdots a_n\ \$)$

2. Če je funkcija ACTION (s_m, a_i) = reduciraj $A \rightarrow \gamma$, razpoznavnik reducira niz v neterminal A in dobimo naslednjo konfiguracijo:

$(s_0\ s_1\ s_2 \dots s_{m-r}\ s,\ a_i\ a_{i+1} \dots a_n\ \$)$

Stanje s dobimo iz funkcije GOTO (s_{m-r}, A), kjer je r dolžina niza γ, to je desne strani produkcije.

3. Če je funkcija ACTION (s_m, a_i) = sprejmi, je razpoznavanje končano.

4. Če je funkcijo ACTION (s_m, a_i) = napaka, je razpoznavnik odkril napako in se pokličejo ustrezni podprogrami.

2.1. Postopek za generiranje LALR tabel

Prva naloga pri gradnji tabel je, da skonstruiramo množico LR(1) postavk za gramatiko G, ki opisuje programski jezik. Najprej G razširimo s produkcijo $S \rightarrow S'$, kjer je S startni simbol gramatike G. S to dodatno produkcijo dodamo informacijo, kdaj naj se razpoznavnik ustavi. Generira se stanje "sprejmi" v ACTION delu tabele.

LR(1) postavka je urejen par oblike $(A \rightarrow \alpha . \gamma, a)$, kjer je prva komponenta jedro, druga pa množica terminalov s končnim simbolom $.

Prvi del postavke-jedro nam pokaže, kolikšen del produkcije je bil že razpoznan pri razpoznavanju. Iz zgornje produkcije dobimo naslednje postavke:

1. $(A \rightarrow . \alpha\gamma, a)$
2. $(A \rightarrow \alpha . \gamma, a)$
3. $(A \rightarrow \alpha\gamma ., a)$

Prva postavka pove, da na vhodu pričakujemo niz, ki je izpeljiv iz $\alpha\gamma$, druga pa, da pričakujemo niz izpeljiv iz γ. Tretje postavka se nanaša na drugo komponento in pove, da je potrebna redukcija $A \rightarrow \alpha\gamma$, če je na vhodu simbol a. Formalno pravimo, da je postavka $(A \rightarrow \alpha . \gamma, a)$ veljavna za prefiks β, če obstaja izpeljava $S \rightarrow^* \delta A w \rightarrow \delta\alpha\gamma w$, kjer velja

1. $\beta = \delta\alpha$ in
2. a je prvi simbol w ali w je ε in a je $.

Algoritem za konstrukcijo LR (1) postavk

Vhod: Razširjena gramatika G

Izhod: Množica LR (1) postavk

Metoda: Procedure CLOSURE, GOTO in izračunaj - postavke

```
procedure CLOSURE(P)
begin
  repeat
    for  za vsako postavko (A--->b.Bg,a) v P,vsako produkcijo B--->d in
         vsak terminal b v FIRST(ga) tako da B--->.d se ni v P
    do   dodaj (B--->.d,b) k P;
    endfor;
  until nobena postavka ne more vec biti dodana k P;
  return P
end


procedure GOTO(P,X);
begin
  naj bo J mnozica postavk (A--->aX.g,a) tako da je (A--->a.Xg,a) v P;
  return CLOSURE(J)
end


program IZRACUNAJ_POSTAVKE;
  begin
    C:=$CLOSURE({S'--->.S,$})$
    repeat
      for za vsako mnozico postavk P v C in vsak simbol gramatike X,
          tako da GOTO(P,X) ni prazna mnozica in da se ni v C
      do  dodaj GOTO(P,X) k C
      endfor
    until nobene mnozice postavk ne moremo vec dodati k C
end.
```

Naslednji korak pri konstrukciji tabel je pretvorba množic LR (1) postavk v množice LALR (1) postavk. To dosežemo tako, da združimo vse množice postavk, ki imajo enaka jedra, se pravi enake prve komponente.

Algoritem za konstrukcijo LALR (1) postavk

Vhod: Razširjena gramatika G
Izhod: Množica LALR (1) postavk
Metoda:

1. Konstruiraj $C = P_0, P_1 \dots P_n$, množice LR (1) postavk

2. Za vsako jedro v LR (1) množici postavk najdi vse množice z enakim jedrom in jih združi v unijo.

Zadnji korak v konstrukciji tabele je pretvorba množice LALR (1) postavk v LALR (1) tabelo. Če v tabeli ni večkratno definiranih vstopov (konfliktov) pravimo, da je tabela LALR (1) tabela.

Poznamo dve vrsti konfliktov:

1. Reduciraj - reduciraj konflikt
2. Pomakni - reduciraj konflikt

Algoritem za konstrukcijo LALR (1) tabele

Vhod: Razširjena gramatika G

Izhod: LALR (1) tabela v primeru, da ni konfliktov

Metoda:

1. Skonstruiraj množico LALR (1) postavk C za G, kjer je $C = P_0, P_1 \ldots P_n$

2. Stanje i za razpoznavnika se skonstruira iz P_i. Funkcija ACTION je določena takole:
   a) Če je postavka $(A \rightarrow \alpha . a \beta, b)$ v P_i, GOTO $(P_i, a) = P_j$ postavi ACTION (i, a) na "pomakni j"
   b) Če je postavka $(A \rightarrow \alpha ., a)$ v P_i, potem postavi ACTION (i, a) na "reduciraj" $A \rightarrow \alpha$.
   c) Če je postavka $(S' \rightarrow S., \$)$ v P_i, potem postavi ACTION (i, $) na "sprejmi"

3. GOTO funkcija za stanje P_i se določi na naslednji način: Če je GOTO $(P_i, A) = P_j$ potem je GOTO (i, A)=j, (A je neterminal.)

4. Vsi vstopi, ki niso definirani z 2 ali 3 so postavljeni na "napako"

5. Začetno stanje je stanje skonstruirano iz postavk $(S' \rightarrow S ., \$)$

3. Opis programa

Program za generiranje LALR (1) tabel zahteva kot vhodne podatke gramatiko ustreznega jezika v BNF meta jeziku. Ta ima naslednja pravila:

GRAMATIKA: : = BLOK $

BLOK : : = PRODUKCIJA {';'PRODUKCIJA} *

PRODUKCIJA : : = LEVI DEL ':' : = 'DESNI DEL

LEVI DEL : : = NETERMINAL

DESNI DEL : : IZRAZ {'/'IZRAZ}

IZRAZ : : = {NETERMINAL / TERMINAL}

NETERMINAL : : = ČRKA {ČRKA / ŠTEVILKA} *

TERMINAL : : = '.' {ZNAK} * '.'

DELIMITER : : = '='/'/'/';'/'$'/'.'

Program najprej pretvori vhodne podatke v vmesni tekst, iz tega vmesnega teksta izračuna postavke, nato konstruira tabelo, jo linearizira in izpiše. Glavni program z imenom GENERIRAJ je:

```
program GENERIRAJ(VHODNI_PODATKI)
(* PROG so produkcije ki določajo gramatiko jezika *)
(* F10 je kretnica za izpis postavk                *)
(* MATR je vmesna matrika                          *)
(* IZRACUNAJ_POSTAVKE_MODIF  procedura             *)
(* KONSTRUIRAJ_TABELO        procedura             *)
(* LINEARIZIRAJ              procedura             *)
(* RAZPOZNAJ                 procedura             *)

begin
  preberi VHODNE_PODATKE;RAZPOZNAJ;
  if ni napak then IZRACUNAJ_POSTAVKE_MODIF;
    if F10 do izpisi mnozico postavk end;
    KONSTRUIRAJ-TABELO;
    LINEARIZIRAJ;
    izpisi linearizirano tabelo;
    if odkriti konflikti do izpisi konflikte end
  else
    javi slovnicne napake
  endif
end.
```

Procedura RAZPOZNAJ opravi leksikalno in sintaksno analizo vhodnih podatkov in jih preslika v vmesni tekst z namenom, da se poenostavi nadaljnje generiranje in da je program kar se da modularen.

```
procedure RAZPOZNAJ;
begin
  opravi leksikalno analizo;
  opravi sintaksno analizo;
  if ni napak then pretvori v MATR endif
end
```

Za konstrukcijo postavk smo nekoliko spremenili algoritma in razdelka 2, ker smo tako prihranili precej pomnilniškega prostora.

Glavne razlike so:

1. Shranimo le glave množic postavk. Glava je tisti del množice postavk, ki jih skonstruiramo pri GOTO funkciji preden izračunamo CLOSURE (množica J).

2. Ne skonstruiramo celotne množice LR (1) postavk, ampak zahtevane unije računamo kar sproti. Zaradi tega postane postopek iterativen, torej se poveča čas obdelave.

Prvo komponento postavke predstavimo v računalnik z dvema celima številoma (N, I), kjer je N številka produkcije in I mesto pike v postavki.

Drugo komponento predstavimo z logičnim poljem z n biti, kjer je n število atomov v jeziku.

Izračun LALR postavk je prikazan v proceduri.

```
procedure IZRACUNAJ_POSTAVKE_MODIF;
(* N,L sta stevca                                    *)
(* I(N) je mnozica postavk                           *)
(* I je mnozica vseh mnozic postavk                  *)
(* P1,P delovni pomnilnik                            *)
(* GOTO,CLOSURE sta proceduri definirani prej *)
begin N:=0;L:=0;I(N):=(S'--)S,$)
(* izracun prve iteracije *)
  while niso obdelane vse I(N) do P:=CLOSURE(N);X:=prvi simbol;
    while X se ni obsel vseh simbolov do
      I(L):=GOTO(P,X);L:=L+1;
      if I(K)=I(L-1) do
        I(K):=I(K) unija I(L-1);          (* 0<=K<=L *)
        L:=L-1                            (* brisi I(L-1) iz I *)
      end;
      X:=naslednji simbol;
    end;
    N:=N+1;
  end;
(* izracun naslednjih iteracij *)
  repeat N:=0;
    while niso obdelani vsi I(N) iz I do P:=CLOSURE(I(N));X:=prvi simbol;
      while X se ni obsel vseh simbolov do
        P1:=GOTO(P,X)
        if I(K)=P1 do I(K):=I(K) unija P1 end     (* 0<=K<=L *)
      end
    end
  until ni sprememb v I
end
```

Preslikavo iz LALR postavk v LALR tabelo opravimo kot je opisano v razdelku 2. 1. in je prikazana v proceduri KONSTRUIRAJ - TABELO

```
procedure KONSTRUIRAJ-TABELO
(* TAB je LALR tabela organizirana kot matrika m x n,
   m je stevilo simbolov *)
   n je stevilo stanj *)
begin N:=0
  while niso obdelane se vsi I(N) iz I do P:=CLOSURE(I(N));
    while niso obdelane vse postavke v P do
      if (S'--)S,,$) v P do  TAB(N,$):=(0,"A")
      else (A--)α.,a) v P do TAB(N,a):=(A,"R")
      else (A--)α.bβ,a v P do
        P1:=GOTO(I(N),b);
        najdi v I takšen I(K) da je I(K)=P1;
        if b je terminal do TAB(n,b)=(K,"S")
        else b je neterminal do TAB(n,b):=(K,"G") end
      end
    end
  end
end
```

Kot izhod iz procedure KONSTRUIRAJ - TABELO dobimo matriko m x n, kjer je m število atomov jezika, za katerega konstruiramo tabele, n pa število stanj končnega avtomata za razpoznavnik.

Zadnji korak pri konstrukciji LALR tabele je linearizacija matrike. Linearizacija pomeni, da tabelo, ki je organizirana kot matrika preslikamo v vektor, v katerega ne vpisujemo praznih členov matrike.

```
(* I,K,L,N,J ... so stevci *)
(* TAB1 je linearizirana LALR tabela *)
(* TAB2 je tabela indeksov *)
begin
procedure LINEARIZIRAJ;
  K:=1;TAB2(1):=1;                    (* inicializacija *)
  for N:=1 to n do
    begin L:=0
      for M:=1 to m do
        begin
          if TAB(N,M) ni 0 do TAB1(K):=(K,n,v,M) end;
          L:=L+1;K:=K+1;
        end;
      TAB2(N+1):=TAB2(N)+L
    end
end
```

4. ZAKLJUČEK

Naveden generator LALR tabel je zelo koristen pripomoček in omogoča razpoznavanje kakršnega koli jezika, ki se ga da izvajati v kontekstno svobodni obliki. Torej ga lahko uporabimo pri implementaciji večine današnjih programirnih jezikov, kakor tudi za komandne (ukazne) jezike. Generator smo preizkusili za proceduralni jezik, za aplikativni jezik in za komandni jezik simulacijskega svežnja. Da smo lahko pri tem odpravljali napake, je bilo potrebno generirati poleg osnovne LALR(1) tabele še tabelo napak. V bodoče nameravamo spremeniti generator tabel tako, da bo sprejel kot vhodne podatke tudi gramatike jezikov v razširjeni BNF obliki.

LITERATURA

1. W. A. Barrett, J. D. Couch, Compiler Construction: Theory and Practice, SRA, 1978.
2. A. V. A Ho, J. D. Ullman, Principles of Compiler Design, Addison-Wesley, 1977.
3. V. Žumer, idr., Raziskava mikroprogramirane arhitekture za implementacijo visokega programskega jezika, VTŠ Maribor 1982.

5. PRIMER

```
K O N S T R U K C I J A   L A L R   T A B E L E

VHODNI TEKST

   1    EXP::=EXP ADDOP TERM / TERM ;
   2    TERM::=TERM MULTOP FACTOR / FACTOR ;
   3    FACTOR::=.(. EXP .). / .IDENT. ;
   4    ADDOP::=.+. / .-. ;
   5    MULTOP::=./. / .*. $

 0. NAPAK ODKRITIH

            SIMBOLI

 KODA     IME        TIP

    1     EXP        NETERMINAL
    2     ADDOP      NETERMINAL
    3     TERM       NETERMINAL
    4     MULTOP     NETERMINAL
    5     FACTOR     NETERMINAL
    6     (          TERMINAL
    7     )          TERMINAL
    8     IDENT      TERMINAL
    9     +          TERMINAL
   10     -          TERMINAL
   11     /          TERMINAL
   12     *          TERMINAL
   13     $          KONCNI SIMBOL

          PRODUKCIJE

  1     EXP     -->  EXP     ADDOP   TERM
  2     EXP     -->  TERM
  3     TERM    -->  TERM    MULTOP  FACTOR
  4     TERM    -->  FACTOR
  5     FACTOR  -->  (       EXP     )
  6     FACTOR  -->  IDENT
  7     ADDOP   -->  +
  8     ADDOP   -->  -
  9     MULTOP  -->  /
 10     MULTOP  -->  *
```

LINEARIZIRANA LALR TABELA

N	STANJE	TIP	J
1	2	G	1
2	3	G	3
3	4	G	5
4	5	S	6
5	6	S	8
6	7	G	2
7	8	S	9
8	9	S	10
9	0	A	13
10	10	G	4
11	3	R	7
12	3	R	9
13	3	R	10
14	11	S	11
15	12	S	12
16	3	R	13
17	5	R	7
18	5	R	9
19	5	R	10
20	5	R	11
21	5	R	12
22	5	R	13
23	13	G	1
24	3	G	3
25	4	G	5
26	5	S	6
27	6	S	8
28	7	R	7
29	7	R	9
30	7	R	10
31	7	R	11
32	7	R	12
33	7	R	13
34	14	G	3
35	4	G	5
36	5	G	6
37	6	S	8
38	8	R	6
39	8	R	8
40	9	R	6
41	9	R	8
42	15	G	5
43	5	S	6
44	6	S	8
45	10	R	6
46	10	R	8
47	11	R	6
48	11	R	8
49	7	G	2
50	16	S	7
51	8	S	9
52	9	S	10
53	10	G	4
54	2	R	7
55	2	R	9
56	2	R	10
57	11	S	11
58	12	S	12
59	2	R	13
60	4	R	7
61	4	R	9
62	4	R	10
63	4	R	11
64	4	R	12
65	4	R	13
66	6	R	7
67	6	R	9
68	6	R	10
69	6	R	11
70	6	R	12
71	6	R	13

TABELA REDUKCIJ

PRODUKCIJA	DOLZINA	LEVA STRAN
1	3	1
2	1	1
3	3	3
4	1	3
5	3	5
6	1	5
7	1	2
8	1	2
9	1	4
10	1	4

TABELA INDEKSOV

I	INDEX
1	1
2	6
3	10
4	17
5	23
6	28
7	34
8	38
9	40
10	42
11	45
12	47
13	49
14	53
15	60
16	66
17	72

UPORABA JEZIKA ADA V ŠTEVILSKIH IZRAČUNIH

A. P. ŽELEZNIKAR

UDK: 681.3.06 ADA:519.682

DO ISKRA DELTA, LJUBLJANA

Ta članek opisuje uporabo prevajalnika Janus/Ada (izvedenka 1.5.0) pri številskih izračunih z aritmetiko plavajoče vejice enojne in dvojne dolžine. Prikazano je dinamično območje aritmetičnih operacij v plavajoči vejici in njihova natančnost. Članek obravnava primere s področja elementarnih operacij (korenjenje), numerične sistemske pakete, numerično integriranje, izračun vrednosti determinante in obrnitev matrike. V članku je nakazana tudi možnost reševanja drugih numeričnih problemov v jeziku Ada.

Usage of Ada in Numeric Computations

This article describes the usage of Janus/Ada Compiler (Version 1.5.0) in numeric computations with floating point arithmetics of single and double precision. The dynamic area of floating point operations and their precision is shown. The article deals with examples of elemntary operations (square rooting), numeric system packages, numeric integration, calculation of the determinant value and inversion of a matrix. This article presents posibilities of solving several numeric problems in Ada language.

1. Uvod

Nova izpeljanka prevajalnika Janus/Ada (1.5.0) podpira podatkovne tipe v plavajoči vejici z enojno in dvojno dolžino (standard 8087). Operacije s pomično vejico se vključujejo v programske pakete z uporabo člena WITH modula 'sfloatop' ali 'floatops'.

Operacije s pomično vejico so usklajene s standardom matematičnega procesorja 8087 podjetja Intel (standardni format IEEE), ki zagotavlja

enojno natančnost 6,5 decimalnih mest
na intervalu
(8.43E-37, 3.37E+38)

in

dvojno natančnost 15 decimalnih mest
na intervalu
(4.14E-307, 1.67E+308)

Vgrajena tipa 'float' in 'long_float' ustrezata enojni in dvojni natančnosti. Rutine pomične vejice pretvorijo vse operande v enoten format in omogočajo uporabo celih števil in različnih tipov s pomično vejico v izrazih. Obstajata dve knjižnici plavajoče vejice, in sicer:

-- sfloatop: vsi izračuni se opravijo v aritmetiki enojne natančnosti; ta modul je hitrejši kot modul floatops, vendar ga ne smemo uporabljati pri izračunih z dvojno natančnostjo

-- floatops: vsi izračuni se opravijo z dvojno natančnostjo; modul se lahko uporablja pri izračunih z enojno in dvojno natančnostjo; ta modul je počasnejši od modula sfloatop, vendar so z njim tudi izračuni z enojno natančnostjo bolj točni, ker je zmanjšana možnost prestopa (overflow, underflow); če hitrost ni bistvena, naj bi se uporabljal ta modul

Le enega od obeh modulov lahko uporabimo v programski enoti. Pojavijo pa se lahko tele napake oziroma njihova sporočila:

-- prestop (preveliko število)
-- deljenje z ničlo
-- slab operand (neinicializirano ali poškodovano število) in
-- izraz je prezapleten

PRAGMA arithcheck(off) se upoablja za zadušitev teh sporočil.

Napake prestopa se ne javljajo in vrednost se zaokroži na ničlo. Različni IEEE formati so opisani v priročniku koprocesorja 8087.

Za izpis števil v plavajoči vejici imamo stavek

put(realna_spremenljivka, pred, po, eksponent);

kjer je

realna_spremenljivka
    realno število,
pred
    je celo število, ki označuje število mest pred decimalno vejico (piko) v mantisi,

po
    je celo število, ki označuje število mest za decimalno vejico v mantisi in
eksponent
    je celo število, ki označuje dolžino eksponenta; če je eksponent = 0, se eksponentno polje ne natisne.

V tem članku bomo obravnavali nekatere značilne primere številskih izračunov v jeziku Ada. Vrsto primerov bomo programirali v Adi neposredno iz znanih algolskih programov ((1,2,3)). Pri tem bomo večkrat pokazali, kako je jezik Ada v primerjavi z jezikom Algol omejen, tako da določene omejitve prispevajo k varnosti oziroma zanesljivosti dobljenega Ada programa. Vobče lahko pričakujemo, da bodo Ada programi daljši, manj splošni in matematično manj lepi od "ekvivalentnih" algolskih programov.

Nekatere vire in snov za spoznavanje jezika Ada najdemo tudi v časopisu Informatica ((4,5,6)).

2. Preizkus prevajalnika Janus/Ada

Programi za številske izračune so občutljivi na natančnost in na dinamično območje operacij v plavajoči vejici ((7)). Algoritem se lahko konča, ko je vrednost določenega člena manjša od relativne tolerance, npr.

člen < toleranca * vsota

Vrednost tolerance ne sme biti manjša od natančnosti operacij v plavajoči vejici, ker se sicer računalni postopek (nikoli) ne konča. Če se operacije v plavajoči vejici izvajajo z natančnostjo 15 številskih mest, mora biti toleranca večja od 10**-15.

Lista 1 prikazuje program za preizkus natančnosti in dinamičnega območja prevajalnika Janus/Ada. Začetna vrednost x je določena z deljenjem vrednosti 1 z vrednostjo 3. Korakoma se izračunavajo manjše in manjše vrednosti za x, ki se prikazujejo na zaslonu, kot kaže lista 2 (v izsekih zaradi dolžine liste), ko se prejšnja vrednost x pomnoži z vrednostjo 0,1. Postopek se nadaljuje, dokler sa ročno ne ustavimo zaradi podstopa (premajhne vrednosti, ki se zaokroži v vrednost 0). Kot vidimo, je dinamično območje pri eksponentu -309 še resularno, potem pa se žačne vrednost mantise neresularno spreminjati, ko se po eksponentu -323 zaokroži v ničlo.

```
-------------------------------------------------
--  Osnovni preizkus aritmetičnih operacij s   --
--     pomično vejico                          --
-------------------------------------------------
WITH floatio, floatops, util;

PACKAGE BODY preizkus IS
  SUBTYPE real IS long_float;
  i: integer; x: real;
  BEGIN
    new_line;
    x := 1.0/3.0;
    FOR i IN 1 .. 200 LOOP
      put(" x = "); floatio.put(x); x := 0.1*x;
      put("   x = "); floatio.put(x);
      new_line; x := 0.1*x;
    END LOOP;
  END preizkus;
```

Lista 1. Preizkus osnovnih aritmetičnih operacij v plavajoči vejici (long_float in float)

Lista 2. Preizkus natančnosti in dinamičnega območja prevajalniške aritmetike s programom iz liste 1 kaže, da je izračun do potence -307 regularen. Vrednost mantise, ki se je povečevala, se začne pri eksponentu -310 zmanjševati in preide po eksponentu -323 v ničlo.

```
13A>preizkus

 x = 3.33333333333333E-1    x = 3.33333333333342E-296
 x = 3.33333333333333E-3    x = 3.33333333333342E-298
 x = 3.33333333333333E-5    x = 3.33333333333342E-300
 x = 3.33333333333333E-7    x = 3.33333333333342E-302
 x = 3.33333333333334E-9    x = 3.33333333333342E-304
 x = 3.33333333333334E-11   x = 3.33333333333342E-306
 x = 3.33333333333334E-13   x = 3.33333333333342E-308
 x = 3.33333333333334E-15   x = 3.33333333333317E-310
 x = 3.33333333333334E-17   x = 3.33333333333317E-312
 x = 3.33333333333334E-19   x = 3.33333332883357E-314
 x = 3.33333333333342E-295  x = 3.33333329533459E-316
 x = 3.33333333333342E-297  x = 3.33333245542299E-318
 x = 3.33333333333342E-299  x = 3.33296684684506E-320
 x = 3.33333333333342E-301  x = 3.31023982713637E-322
 x = 3.33333333333342E-303  x = 0.00000000000000E+0
 x = 3.33333333333342E-305
 x = 3.33333333333342E-307
 x = 3.33333333333342E-309
 x = 3.33333333333317E-311
 x = 3.33333333332823E-313
 x = 3.33333329919916E-315
 x = 3.33331269279715E-319
 x = 3.33000245297002E-321
 x = 2.96439387504749E-323
```

Lista 3. Preizkus natančnosti in dinamičnega območja za "kratko" aritmetiko

```
13A>preizkus

 x = 3.33333E-1    x = 3.33333E-2
 x = 3.33333E-3    x = 3.33333E-4
 x = 3.33333E-5    x = 3.33333E-6
 x = 3.33333E-7    x = 3.33333E-8
 x = 3.33333E-9    x = 3.33333E-10
 x = 3.33333E-11   x = 3.33333E-12
 x = 3.33333E-13   x = 3.33333E-14
 x = 3.33333E-15   x = 3.33333E-16
 x = 3.33333E-17   x = 3.33333E-18
 x = 3.33333E-19   x = 3.33333E-20
 x = 3.33333E-21   x = 3.33333E-22
 x = 3.33333E-23   x = 3.33333E-24
 x = 3.33333E-25   x = 3.33333E-26
 x = 3.33333E-27   x = 3.33333E-28
 x = 3.33333E-29   x = 3.33333E-30
 x = 3.33333E-31   x = 3.33333E-32
 x = 3.33333E-33   x = 3.33333E-34
 x = 3.33333E-35   x = 3.33333E-36
 x = 3.33333E-37   x = 3.33333E-38
 x = 3.33333E-39   x = 3.33332E-40
 x = 3.33327E-41   x = 3.33229E-42
 x = 3.32108E-43   x = 3.22299E-44
 x = 2.80260E-45   x = 0.00000E+0
```

Primer iz liste 1 oziroma liste 2 kaže, da je dinamično območje prevajalniške aritmetike enako prevajalniški aritmetični natančnosti, torej precej boljše od "dinamike" nekaterih pscalskih prevajalnikov ((7)). Podobno velja tudi za natančnost pri enojni dolžini, kot kaže lista 3 (tu je long_float zamenjan s tipom float).

```
PACKAGE floatio IS

  -- Floating Point I/O Package

  PRAGMA arithcheck(Off);
  PRAGMA rangecheck(Off);
Ž PRAGMA arithcheck(On); PRAGMA rangecheck(On);

  PROCEDURE set(file : IN file;
                value : OUT float);
  PROCEDURE set(value : OUT float);
        -- Gets a value from the file

  PROCEDURE put(file : IN file;
                value : IN float);
  PROCEDURE put(value : IN float);
  PROCEDURE put(file : IN file;
                value : IN float;
                fore : IN integer);
  PROCEDURE put(value : IN float;
                fore : IN integer);
  PROCEDURE put(file:IN file; value : IN float;
                fore,aft : IN integer);
  PROCEDURE put(value : IN float;
                fore,aft : IN integer);
  PROCEDURE put(value : IN float;
                fore,aft,exp : IN integer);
  PROCEDURE put(file : IN file;
                      value : IN float;
                   fore,aft,exp : IN integer);
        -- Puts formatted value to file

  FUNCTION float_to_string(value : IN float)
                                  RETURN string;
  FUNCTION float_to_string(value : IN float;
                       fore,aft,exp : integer)
                                  RETURN string;
        -- Formats value into a string

  PROCEDURE set(file : IN file;
                value : OUT long_float);
  PROCEDURE set(value : OUT long_float);
        -- Gets a value from the file

  PROCEDURE put(file : IN file;
                value : IN long_float);
  PROCEDURE put(value : IN long_float);
  PROCEDURE put(file : IN file;
                value : IN long_float;
                fore : IN integer);
  PROCEDURE put(value : IN long_float;
                fore : IN integer);
  PROCEDURE put(file:IN file;
                value : IN long_float;
                fore,aft : IN integer);
  PROCEDURE put(value : IN long_float;
                fore,aft : IN integer);
  PROCEDURE put(value : IN long_float;
                fore,aft,exp : IN integer);
  PROCEDURE put(file : IN file;
                value : IN long_float;
                fore,aft,exp : IN integer);
        -- Puts formatted value into file

  FUNCTION float_to_string(value :
                  IN long_float) RETURN string;
  FUNCTION float_to_string(value :
         IN long_float; fore,aft,exp : integer)
                                  RETURN string;
        -- Formats value into a string

END floatio;
```

Lista 4. Ta lista prikazuje zgradbo paketa floatio, ki sa sestavljajo procedure tipa set in put in funkcije tipa float_to_string. Iz liste je razvidno, na katere možne načine lahko uporabimo te procedure in funkcije (s kakšnimi parametri oziroma s kakšnimi učinki).

```
PACKAGE mathlib IS
  -- Double Precision Mathematic Functions
  --  Library


  pi : CONSTANT := 3.14159_26535_89793_23846;
  e : CONSTANT := 2.71828_18284_59045_23536;
  log10_e :
        CONSTANT := 0.43429_44819_03251_82765;
        -- log10(e)

  FUNCTION sqrt (val : IN long_float) RETURN
           long_float;
        -- Returns the sqrt of val

  FUNCTION round (val : IN long_float) RETURN
           long_float;
        -- Rounds val to an integer value, and
        --   returns it as a float

  FUNCTION trunc (val : IN long_float) RETURN
           long_float;
        -- Truncate val to its integer part,
        --   and returns it as a float

  FUNCTION exp (val : IN long_float) RETURN
           long_float;
        -- Returns e ** val
        -- To get 10 ** val, divide by log10_e.

  FUNCTION log (val : IN long_float) RETURN
           long_float;
        -- Returns the natural logarithm of val
        --    val must be > 0.
        -- To get Log10, multiply by log10_e.

  FUNCTION power (val,exp : IN long_float)
           RETURN long_float;
        -- Returns val ** exp

  -- All angles are in radians!

  FUNCTION sin (angle : IN long_float) RETURN
           long_float;
        -- Returns the sine of the angle

  FUNCTION cos (angle : IN long_float) RETURN
           long_float;
        -- Returns the cosine of the angle

  FUNCTION tan (angle : IN long_float) RETURN
           long_float;
        -- Returns the tangent of the angle

  FUNCTION arctan (val : IN long_float) RETURN
           long_float;
        -- Returns the arctangent of the value

  FUNCTION arccos (val : IN long_float) RETURN
           long_float;
        -- Returns the arccosine of the value

  FUNCTION arcsin (val : IN long_float) RETURN
           long_float;
        -- Returns the arcsine of the value

  FUNCTION arctan2(X,Y : IN long_float) RETURN
           long_float;
        -- Returns the arctangent of X / Y

  FUNCTION deg_to_rad (angle : IN long_float)
           RETURN long_float;
        -- Converts the angle in degrees to the
        --   same angle in Radians

  FUNCTION rad_to_deg (angle : IN long_float)
           RETURN long_float;
        -- Converts the angle in radians to the
        --   same angle in degrees

END mathlib;
```

Lista 5. Knjižnica matematičnih funkcij z dvojno natančnostjo

3. Paket "floatio" in knjižnica "mathlib"

Knjižnica "floatio" je zbirka funkcij in procedur za sprejemanje (get) in izdajanje (put) vrednosti in za pretvarjanje vrednosti, kot kaže lista 4. Ta paket je lep primer podprogramskega prekrivanja (overloading of subprograms). Isto podprogramsko ime se lahko uporablja v več različnih podprogramskih specifikacijah; rečemo, da se ime prekriva (da je prekrivajoče).

Za naše primere bo zlasti bistvena uporaba procedure put za enojno (float) in dvojno dolžino (long_float). Imamo 8 različnih procedur put za enojno in dvojno dolžino parametrov. Lista 4 nazorno pojasnjuje posamezne primere. Razen prikazanih procedur put v paketu floatio pa obstajajo še druge procedure put v drugih paketih (standardio), ko imamo npr. še

```
PROCEDURE put(value : IN string);
PROCEDURE put(value : IN integer;
              fore : IN integer);
```

ki jih tudi pogosto uporabljamo. Pri procedurah put iz paketa floatio se nanje sklicujemo s procedurnim imenom

floatio.put

tako da prevajalniku olajšamo delo.

Za številsko računanje pa so zlasti pomembne nekatere matematične funkcije, ki so zbrane v knjižnici mathlib.lib (ime zbirke). Te funkcije se prikazane v paketu liste 5 za dvojno natančnost. Te funkcije so:

```
sqrt(x), round(x),   trunc(x),  exp(x),
log(x),  power(x,y), sin(x),    cos(x),
tan(x),  arctan(x),  arccos(x), arcsin(x),
arctan2(x,y), deg_to_rad(x), rad_to_deg(x)
```

Med zanimivimi funkcijami sta vsekakor tudi exp(e) in power(x,y), ki omogočata konstrukcijo novih uporabniških funkcij (v novih knjižnicah). Te knjižnice vključujemo in povezujemo v uporabniške pakete s členoma WITH in USE, kot bomo pokazali v naših primerih.

4. Primer funkcije kvadratnega korena

Napišimo v jeziku Ada program za funkcijo kvadratnega korena, ko uporabimo Newtonovo formulo. Računalniki, na katerih se operacija deljenja izvaja dovolj hitro, dajo dobre rezultate z uporabo Newtonove iteracijske formule ((8, 9))

$$y(i+1) = 1/2 * (y(i) + z/y(i)) \quad (i = 0,1,\ldots)$$

$$\lim y(i) = \sqrt{z}$$

Za število x v plavajoči vejici

$$x = (2^{**}p) * z,$$

kjer velja $1/2 \le z < 1$

dobimo

$$\sqrt{x} = (2^{**}(p/2)) * \sqrt{z}$$

Pri lihem p izračunamo 2**((p-1)/2) in pomnožimo ta faktor še s

$$\sqrt{2} \quad \text{ali} \quad 1/\sqrt{2}$$

v odvisnosti od predznaka p. Normalizacija vre-

```
WITH floatio, floatops, util;
PACKAGE BODY testkor IS
  SUBTYPE real IS long_float;
  x,y: real; i,p: integer;
  sq2: CONSTANT := 1.414_213_562_373_096;
-------------------------------------------------
  PROCEDURE normal (x: IN real;
                    y: OUT real;
                    p: OUT integer) IS
    BEGIN
      IF x < 0.0 THEN new_line;
      put("Napaka: koren negativnega števila");
                 new_line; y := 1.0; p := 0;
      ELSIF x > 1.0 THEN  y := 2.0; p := 1;
        WHILE x > y LOOP
          y := y*2.0; p := p+1; END LOOP;
      ELSIF x > 0.0 AND x < 0.5 THEN
        y := 0.5; p := -1;
        WHILE x < y LOOP
          y := y*0.5; p := p-1; END LOOP;
        y := y*2.0; p := p+1;
      ELSE y := 1.0; p := 0;
      END IF;
      y := x/y;
    END normal;
-------------------------------------------------
  FUNCTION sqr_z (z: IN real) RETURN real IS
    x, y: real; j: integer;
    BEGIN
      x := abs(z);
      IF z = 1.0 THEN RETURN z;
      ELSE
        y := 0.5+0.5*x;
        FOR j IN 1 .. 6 LOOP
          y := 0.5*(y+x/y);
        END LOOP;
        RETURN y;
      END IF;
    END sqr_z;
-------------------------------------------------
  FUNCTION sign (i: IN integer)
                              RETURN integer IS
    BEGIN
      IF i < 0 THEN RETURN -1;
      ELSE RETURN 1; END IF;
    END sign;
-------------------------------------------------
  FUNCTION sqrt (x: IN real) RETURN real IS
    y: real; p: integer;
    BEGIN
      normal(x,y,p);
      IF p mod 2 = 0 THEN
        RETURN (2.0**(p/2))*sqr_z(y);
      ELSE RETURN
       (2.0**((p-1)/2))*sqr_z(y)*(sq2**sign(p));
      END IF;
    END sqrt;
-------------------------------------------------

  BEGIN
    x := 2.0;
    FOR i IN 1 .. 15 LOOP
      new_line;
      floatio.put(x); put("    ");
      floatio.put(sqrt(x)); put("    ");
      floatio.put(sqrt(x)*sqrt(x));
      x := x+1.0;
    END LOOP;
END testkor;
```

Lista 6. Ta lista prikazuje proceduro in funkcije, ki se uporabljajo pri korenjenju, in sicer za normalizacijo vrednosti na interval (1/2, 1) (normal), za izračun korena normaliziranе vrednosti (sqr_z), za določitev predznaka celega števila (sign) in za izračun korena vrednosti x. Pri tem niso uporabljene funkcije iz knjižnice "mathlib".

```
13A>korada
 2.00000000000000E+0    1.41421356237310E+0    2.00000000000000E+0
 3.00000000000000E+0    1.73205080756888E+0    3.00000000000000E+0
 4.00000000000000E+0    2.00000000000000E+0    3.99999999999999E+0
 5.00000000000000E+0    2.23606797749979E+0    4.99999999999999E+0
 6.00000000000000E+0    2.44948974278318E+0    5.99999999999999E+0
 7.00000000000000E+0    2.64575131106459E+0    6.99999999999998E+0
 8.00000000000000E+0    2.82842712474619E+0    8.00000000000000E+0
 9.00000000000000E+0    3.00000000000000E+0    9.00000000000000E+0
 1.00000000000000E+1    3.16227766016838E+0    1.00000000000000E+1
 1.10000000000000E+1    3.31662479035540E+0    1.10000000000000E+1
 1.20000000000000E+1    3.46410161513775E+0    1.20000000000000E+1
 1.30000000000000E+1    3.60555127546399E+0    1.30000000000000E+1
 1.40000000000000E+1    3.74165738677394E+0    1.40000000000000E+1
 1.50000000000000E+1    3.87298334620742E+0    1.50000000000000E+1
 1.60000000000000E+1    4.*:*:*:*:*:*:*:*E+0   1.60000000000000E+1
 1.70000000000000E+1    4.12310562561766E+0    1.70000000000000E+1

13A>testkor

 2.00000000000000E+0    1.41421356237310E+0    2.00000000000000E+0
 3.00000000000000E+0    1.73205080756888E+0    3.00000000000000E+0
 4.00000000000000E+0    2.00000000000000E+0    4.00000000000000E+0
 5.00000000000000E+0    2.23606797749979E+0    5.00000000000001E+0
 6.00000000000000E+0    2.44948974278318E+0    6.00000000000001E+0
 7.00000000000000E+0    2.64575131106459E+0    7.00000000000001E+0
 8.00000000000000E+0    2.82842712474619E+0    8.00000000000001E+0
 9.00000000000000E+0    3.00000000000000E+0    9.00000000000000E+0
 1.00000000000000E+1    3.16227766016838E+0    1.00000000000000E+1
 1.10000000000000E+1    3.31662479035540E+0    1.10000000000000E+1
 1.20000000000000E+1    3.46410161513775E+0    1.20000000000000E+1
 1.30000000000000E+1    3.60555127546399E+0    1.30000000000000E+1
 1.40000000000000E+1    3.74165738677394E+0    1.40000000000000E+1
 1.50000000000000E+1    3.87298334620742E+0    1.50000000000000E+1
 1.60000000000000E+1    4.00000000000000E+0    1.60000000000000E+1
```

Lista 7. Ta lista prikazuje dva izpisa, in sicer za paketno telo korada (lista 8) in testkor (lista 6). Programska enota korada daje pri kvadriranju nekaterih vrednosti manjše, enota testkor pa večje vrednosti od nominalnih (tretji stolpec). Iz tega sklepamo, da sta funkciji sqrt iz knjižnice "mathlib" in iz testkor praktično enako natančni. V prvem stolpcu imamo x, v drugem sqrt(x), v tretjem pa sqrt(x) * sqrt(x).

dnosti z v območje (1/2, 1) je potrebna zaradi slabe iteracijske konvergence Newtonove formule pri vrednostih, ki se približujejo ničli.

Začetna vrednost za y(0) je linearni približek

$$y(0) = 0.5903*z + 0.4173$$

ki zagotavlja že pri dveh iteracijah posrešek manjši od 2**-31. Kasneje bomo uvideli, da dobimo s tremi iteracijami rezultate, ki se z nadaljnimi iteracijami ne spreminjajo več (v območju 15-mestnih mantis).

Lista 6 prikazuje paketno telo testkor za preizkušanje funkcije sqrt, ko dobimo izpis v drugi polovici liste 7. Ta funkcija sqrt je dovolj natančna, kar dokazujemo tako, da vrednost korena kvadriramo in dobimo zopet vrednost argumenta.

```
WITH floatio, floatops, util, mathlib;
PACKAGE BODY korada IS
  USE mathlib;
  SUBTYPE real IS long_float;
  x, y: real; i: integer;
  BEGIN
    x := 2.0;
    FOR i IN 1 .. 16 LOOP
      floatio.put(x); put("  ");
      floatio.put(sqrt(x)); put("  ");
      floatio.put(sqrt(x)*sqrt(x)); new_line;
      x := x+1.0;
    END LOOP;
  END korada;
```

Lista 8. Ta lista prikazuje programsko enoto za evaluacijo natančnosti vgrajene funkcije sqrt (v paketu "mathlib"). Rezultati so izpisani v zgornji polovici liste 7.

```
PRAGMA condcomp(on);
WITH floatio, floatops, util;
PACKAGE BODY testkor1 IS
  SUBTYPE real IS long_float;
  x,y,z: real; i,p: integer;
  sq2: CONSTANT := 1.414_213_562_373_096;
-------------------------------------------------
  FUNCTION sqr_z (z: IN real) RETURN real IS
    x, y: real; j: integer;
    BEGIN
      x := abs(z);
      IF z = 1.0 THEN RETURN z;
      ELSE
        y := 0.5903*x+0.4173;
Ż       put("y0 = ");floatio.put(y); new_line;
        FOR j IN 1 .. 6 LOOP
          y := 0.5*(y+x/y);
Ż       put("y"); put(j,1); put(" = ");
Ż       floatio.put(y); put("    ");
Ż       IF j mod 2 = 0 THEN new_line; END IF;
        END LOOP;
        RETURN y;
      END IF;
    END sqr_z;
-------------------------------------------------
  BEGIN
    x := 3.0;
    FOR i IN 1 .. 15 LOOP
      new_line;
      put("x  = "); floatio.put(x); new_line;
      z := sqrt(x);
      put("Koren je "); floatio.put(z);
      new_line; put("Kvadrat korena je ");
      z := z*z; floatio.put(z); new_line;
      x := x+2.0;
    END LOOP;
  END testkor1;
```

Lista 9. Modifikacija programa s liste 6, ki omogoča opazovanje natančnosti v posameznih iteracijskih korakih (y0, y1, ... , y6)

```
13A)testkori

 x  =  3.00000000000000E+0
 y0 =  8.60025000000000E-1   y2 =  8.66025404037408E-1
 y1 =  8.66046336225691E-1   y4 =  8.66025403784439E-1
 y3 =  8.66025403784439E-1   y6 =  8.66025403784439E-1
 y5 =  8.66025403784439E-1
 Koren je  1.73205080756888E+0
 Kvadrat korena je  3.00000000000000E+0

 x  =  5.00000000000000E+0   y2 =  7.90569415132164E-1
 y0 =  7.86237500000000E-1   y4 =  7.90569415042095E-1
 y1 =  7.90581348769456E-1   y6 =  7.90569415042095E-1
 y3 =  7.90569415042095E-1
 y5 =  7.90569415042095E-1
 Koren je  2.23606797749979E+0
 Kvadrat korena je  5.00000000000001E+0

 x  =  7.00000000000000E+0   y2 =  9.35414346694494E-1
 y0 =  9.33812500000000E-1   y4 =  9.35414346693485E-1
 y1 =  9.35415720584298E-1   y6 =  9.35414346693485E-1
 y3 =  9.35414346693485E-1
 y5 =  9.35414346693485E-1
 Koren je  2.64575131106459E+0
 Kvadrat korena je  7.00000000000001E+0
```

Lista 10. Ta lista prikazuje rezultate programa z liste 9. Po tretji iteraciji praktično ni več sprememb v rezultatih, tako da velja praktično y3 = y4 = y5 = y6.

Lista 8 kaže za primerjavo še programsko enoto korada z "vgrajeno" funkcijo sqrt (iz matematične knjižnice "mathlib"). Ta funkcija je za uporabnika dosegljiva z ukazome WITH mathlib; in USE mathlib;, kot je razvidno iz liste 8. Rezultati tega programa so prikazani v prvi polovici liste 7 (korada). Minimalne razlike med obema funkcijama sqrt (iz mathlib in iz testkor) so vidne v tretjem stolpcu liste 7.

V listi 9 imamo modifikacijo programa z liste 6, ko nas zanima natančnost rezultatov v posameznih iteracijskih korakih. Lista kaže le spremembe, in sicer stavek PRAGMA condcomp(on); na začetku, vrinjene stavke v funkciji sqr_z (označene z 'Ž') in spremenjeni glavni program. Rezultati tako modificiranega programa so delno prikazani v listi 10. Od tu je vidno, da tri iteracije v Newtonovi formuli povsem zadoščajo za 15-mestno natančnost.

5. Primeri številskih izračunov vrednosti določenih integralov

V tem poglavju bomo pokazali primere izračunov vrednosti določenih integralov po Simpsonu, po Rombergu in po Gaussu. V prvem primeru bo podintegralska funkcija lahko "eksperimentalna", podana z merjenimi vrednostmi na določenem intervalu s konstantnim korakom. V ostalih dveh primerih pa bo podintegralska funkcija podana "analitično". Seveda pa imamo v vseh treh primerih možnosti za ustrezno programsko predstavitev takih ali drugačnih podintegralskih funkcij (npr. z dodatno uporabo interpolacije).

```
--  Preizkus funkcije za izračun integrala  --
--              po Simpsonu                 --

WITH floatio, floatops, util;
PACKAGE BODY testsim IS
  SUBTYPE real IS long_float;
  TYPE polje IS ARRAY (0 .. 100) OF real;
  n,j: integer;
  yy1,yy2,yy3,yy4: polje;

-- Ta funkcija je predmet naše pozornosti:
--*****************************************

  FUNCTION sim ( n: IN integer; a, b : IN real;
                 y: IN polje)
                 RETURN real IS
    s: real; i: integer;
    BEGIN
      s := (y(0)-y(n))/2.0;
      i := 1;
      WHILE i <= n-1 LOOP
        s := s + 2.0*y(i) + y(i+1);
        i := i+2;
      END LOOP;
      RETURN 2.0*(b-a)*s / (3.0*real(n));
    END sim;
--*****************************************

  FUNCTION func ( x: IN real; j: IN integer)
                  RETURN real IS
    BEGIN
      IF     j=1 THEN RETURN x*x;
      ELSIF j=2 THEN RETURN x*(1.0+x*x);
      ELSIF j=3 THEN RETURN
                     0.43429_44819_03251_82765/x;
      ELSE  RETURN 1.0;
      END IF;
    END func;
-----------------------------------------

  PROCEDURE fill ( a,b: IN real;
                   j,n: IN integer;
                   yy:  OUT polje) IS
    h: real; i: integer;
    BEGIN
      h := (b-a)/real(n);
      FOR i IN 0 .. n LOOP
        yy(i) := func(a+real(i)*h,j);
      END LOOP;
    END fill;
-----------------------------------------

---------  Glavni preizkusni program  ---------

  BEGIN
    --  Polnitev funkcijskih podatkovnih polj:
    fill (2.0, 5.0, 1,  90, yy1);
    fill (2.0, 5.0, 2, 100, yy2);
    fill (1.0, 5.0, 3,  70, yy3);
    fill (2.0, 5.0, 4,  10, yy4);
    --  Izračun integralov:
    new_line; put("Integral 1 = ");
        floatio.put(sim(90, 2.0, 5.0, yy1));
    new_line; put("Integral 2 = ");
        floatio.put(sim(100, 2.0, 5.0, yy2));
    new_line; put("Integral 3 = ");
        floatio.put(sim(70, 1.0, 5.0, yy3));
    new_line; put("Integral 4 = ");
        floatio.put(sim(10, 2.0, 5.0, yy4));
  END testsim;
```

Lista 11. Ta lista prikazuje funkcijo sim in njeno uporabo v določenih primerih. Funkcija sim izračunava vrednost določenega integrala na intervalu (a, b), ko imamo n+1 funkcijskih vrednosti pri konstantnem koraku podanih v polju y. S proceduro fill in funkcijo func generiramo določena poskusna funkcijska polja v glavnem programu te liste.

Lista 12. Ta lista prikazuje rezultate programske enote z liste 11, ko se izračunajo vrednosti določenih integralov. Te vrednosti se ujemajo z idealnimi vrednostmi.

```
13A)testsim

Integral 1 =  3.90000000000000E+1
Integral 2 =  1.62750000000000E+2
Integral 3 =  6.98970157257503E-1
Integral 4 =  3.00000000000000E+0
```

5.1. Izračun integrala po Simpsonu

Ta program (funkcija sim) omogoča izračun vrednosti določenega integrala podintegralske funkcije, ki je dana eksperimentalno oziroma točkovno v enakomernem koraku. Funkcija sim v listi 11 izračuna približno vrednost določenega integrala zvezne funkcije f(x) na intervalu (a, b). Ta funkcija je dana tabelarično z vrednostmi y(0), y(1), ... , y(n) in s konstantnim korakom, kjer je

n sodo število,
y(0) = f(a) in y(n) = f(b)

Izračun se izvrši po Simpsonovi formuli

$$\int_a^b f(x)dx = (b-a)/(3*n) * (y(0) + 4*y(1) + 2*y(2) + \ldots + \ldots + 4*y(n-1) + y(n))$$

V listi 12 imamo rezultate programa z liste 11; te vrednosti se ujemajo z idealnimi vrednostmi določenih integralov.

Pri izračunu integrala po Simpsonu moramo upoštevati, da smo "umetno" oblikovali tabele vrednosti podintegralskih funkcij iz danih analitičnih funkcij. Tu se pokaže, da se nekateri izračuni res ujemajo z idealnimi vrednostmi (pri podintegralskih funkcijah x*x, x*(1.0 + x*x) in 1), pri podintegralski funkciji tipa 1.0/x, ko imamo pri mejah (1, b) integral log b, pa vrednostno ujemanje ni tako dobro (primerjaj Integral 3 v listi 12 z vrednostjo desetiškega logaritma log 5.0).

5.2. Izračun integrala po Rombergu

Funkcija rombint v listi 13 izračuna vrednost integrala

$$\int_a^b f(x)dx$$

s pogreškom stopnje

$$2*k + 2 (k)= 0)$$

Čas izračuna se približno podvoji pri povečanju vrednosti k za 1. V listi 13 uporabimo zopet slobalno podintegralsko funkcijo f s CASE stavkom za posamezne primere; ta oblika funkcije je priročna za preizkušanje integracijskih postopkov z raznimi algoritmi. V štirih primerih preizkušanja v listi 13 smo izbrali k = 6, 6, 8, 3. Rezultati programske enote z liste 13 so prikazani v listi 14. Z izjemo vrednosti integrala 3 so vrednosti enake idealnim.

Lista 13. Izračun integrala po Rombergu

```
-------------------------------------------------
--   Preirkus funkcije za izračun integrala   --
--              po Rombergu                    --
-------------------------------------------------
WITH floatio, floatops, util;
PACKAGE BODY testromb IS
  SUBTYPE real IS long_float;
  TYPE polje IS ARRAY (1 .. 257) OF real;

-------------------------------------------------

  FUNCTION f (x: IN real; q: IN integer)
              RETURN real IS
    BEGIN
      CASE q IS
        WHEN 1 => RETURN x*x;
        WHEN 2 => RETURN x*(1.0+x*x);
        WHEN 3 => RETURN
                    0.43429_44819_03251_82765/x;
        WHEN OTHERS => RETURN 1.0;
      END CASE;
    END f;
-------------------------------------------------

-- Ta funkcija je predmet naše pozornosti:
--*********************************************

  FUNCTION rombint ( a,b: IN real;
                     k,q: IN integer )
                     RETURN real IS
    d,s,h: real; m,i,j,n: integer;
    t: polje;
    BEGIN
      d := b-a;
      t(1) := (f(a,q)+f(b,q))/2.0;
      n := 1;
      FOR i IN 1 .. k LOOP
        s := 0.0; n := 2*n; h := d/real(n);
        j := 1;
        WHILE j <= n LOOP
          s := s + f(a+real(j)*h,q);
          j := j+2;
        END LOOP;
        t(i+1) := (2.0*s/real(n) + t(i)) / 2.0;
        m := 1;
        FOR j IN REVERSE 1 .. i LOOP
          m := 4*m;
          t(j):=t(j+1)+(t(j+1)-t(j))/real(m-1);
        END LOOP;
      END LOOP;
      RETURN t(1)*d;
    END rombint;
--*********************************************

-------------- Glavni program ---------------

  BEGIN
    new_line; put("Integral 1 = ");
        floatio.put(rombint(2.0, 5.0, 6, 1));
    new_line; put("Integral 2 = ");
        floatio.put(rombint(2.0, 5.0, 6, 2));
    new_line; put("Integral 3 = ");
        floatio.put(rombint(1.0, 5.0, 8, 3));
    new_line; put("Integral 4 = ");
        floatio.put(rombint(2.0, 5.0, 3, 4));
  END testromb;
```

Lista 14. Ta lista prikazuje rezultate izračuna vrednosti integralov s programom iz liste 13. Vrednosti integralov 1, 2 in 4 se ujemajo z idealnimi, vrednost integrala 3 pa se dobro približa idealni vrednosti (izjema so le zadnja tri mesta).

```
13A)testromb

Integral 1 =  3.90000000000000E+1
Integral 2 =  1.62750000000000E+2
Integral 3 =  6.98970004336892E-1
Integral 4 =  3.00000000000000E+0
```

```
WITH floatio, floatops, util;

PACKAGE BODY quad1 IS

  SUBTYPE real IS long_float;
  TYPE polje IS ARRAY (1 .. 20) OF real;
  TYPE sum1 IS ARRAY (1 .. 4) OF real;
--  Zgornja meja tipa sum1 je p!
  vsota: sum1;
  u1,c1,u2,c2,u3,c3,u5,c5,u8,c8: polje;
  u13,c13,u20,c20: polje;
-------------------------------------------------

  FUNCTION f (x: IN real; j: IN integer)
                                  RETURN real IS
    BEGIN
      CASE j IS
        WHEN 1 => RETURN x*x;
        WHEN 2 => RETURN x*(1.0+x*x);
        WHEN 3 => RETURN 1.0/x;
        WHEN OTHERS => RETURN 1.0;
      END CASE;
    END f;
-------------------------------------------------

-- Naslednja procedura je predmet nase pozorno-
--   sti
--*************************************************

  PROCEDURE quad (
              a,b: IN real;
              m,n,p: IN integer;
              c,u: IN polje;
              sum: OUT sum1) IS
    h,t,r: real; i,j,k: integer;
    BEGIN
      h := (b-a)/real(m);
      FOR j IN 1 .. p LOOP
        sum(j) := 0.0; r := a-h;
        FOR i IN 1 .. m LOOP
          r := r+h;
          FOR k IN 1 .. n LOOP
            t := r+u(k)*h;
            sum(j) := sum(j)+c(k)*f(t,j);
          END LOOP;
        END LOOP;
        sum(j) := h*sum(j);
      END LOOP;
    END quad;
--*************************************************

  PROCEDURE ars (n: IN integer) IS
    BEGIN
      new_line;
      put(n,2); put("   ");
    END ars;
-------------------------------------------------

  PROCEDURE vred (p: IN integer;
                  sum: IN sum1) IS
    k: integer;
    BEGIN
      FOR k IN 1 .. p LOOP
        floatio.put(sum(k)); put("  ");
      END LOOP;
    END vred;
-------------------------------------------------

  PROCEDURE csum (n: IN integer;
                  c: IN polje) IS
    i: integer; aa: real;
    BEGIN
      aa := 0.0;
      FOR i IN 1 .. n LOOP
        aa := aa+c(i);
      END LOOP;
      new_line;
      put(n,2); put("   "); floatio.put(aa);
    END csum;
-------------------------------------------------

  PROCEDURE line (p: IN integer) IS
    i: integer;
    BEGIN
      new_line; put("---");
      FOR i IN 1 .. p LOOP
        put("-------------------------");
      END LOOP;
    END line;
-------------------------------------------------

  PROCEDURE glava (m,p: IN integer) IS
    i: integer;
    BEGIN
      line(p); new_line;
      put("   m = "); put(m,2);
      line(p);
      new_line; put(" n ");
      FOR i IN 1 .. p LOOP
        put("        Integral"); put(i,2);
        put("      ");
      END LOOP;
      line(p);
    END glava;
-------------------------------------------------

  PROCEDURE inv_koef (n: IN integer;
                      u,c: IN OUT polje) IS
    i,j: integer;
    BEGIN
      i := n; j := 1;
      WHILE  i>j  LOOP
        u(i) := 1.0-u(j); c(i) := c(j);
        i := i-1; j := j+1;
      END LOOP;
    END inv_koef;
-------------------------------------------------

  PROCEDURE izpis (a,b: IN real;
                   m,n,p: IN integer;
                   c,u: polje) IS
    BEGIN
      ars(n);
      quad(a,b,m,n,p,c,u,vsota);
      vred(p,vsota);
    END izpis;
-------------------------------------------------

  PROCEDURE cel_izpis (a,b: IN real;
                       m,p: IN integer) IS
    BEGIN
      glava(m,p);
      izpis(a,b,m, 1,p, c1, u1);
      izpis(a,b,m, 2,p, c2, u2);
      izpis(a,b,m, 3,p, c3, u3);
      izpis(a,b,m, 5,p, c5, u5);
      izpis(a,b,m, 8,p, c8, u8);
      izpis(a,b,m,13,p,c13,u13);
      izpis(a,b,m,20,p,c20,u20);
      line(p);
      new_line;
    END cel_izpis;
-------------------------------------------------

--          Glavni preizkusni program

--------   Inicializacija spremenljivk   --------

BEGIN
  u1(1)   := 0.5;
                  c1(1)   := 1.0;

  u2(1)   := 0.211_324_865_4052;
                  c2(1)   := 0.5;
  inv_koef(2,u2,c2);

  u3(1)   := 0.112_701_665_3792;
  u3(2)   := 0.5;
                  c3(1)   := 0.277_777_777_7778;
                  c3(2)   := 0.444_444_444_4444;
  inv_koef(3,u3,c3);

  u5(1)   := 0.046_910_077_0307;
  u5(2)   := 0.230_765_344_9472;
  u5(3)   := 0.5;
```

```
                    c5(1)   := 0.118_463_442_5281;
                    c5(2)   := 0.239_314_335_2497;
                    c5(3)   := 0.284_444_444_4444;
  inv_koef(5,u5,c5);

  u8(1)   := 0.019_855_071_7512;
  u8(2)   := 0.101_666_761_2931;
  u8(3)   := 0.237_233_795_0418;
  u8(4)   := 0.408_282_678_7522;
                    c8(1)   := 0.050_614_268_1452;
                    c8(2)   := 0.111_190_517_2267;
                    c8(3)   := 0.156_853_322_9389;
                    c8(4)   := 0.181_341_891_6892;
  inv_koef(8,u8,c8);

  u13(1)  := 0.007_908_472_6407;
  u13(2)  := 0.041_200_800_3885;
  u13(3)  := 0.099_210_954_6333;
  u13(4)  := 0.178_825_330_2798;
  u13(5)  := 0.275_753_624_4818;
  u13(6)  := 0.384_770_842_0224;
  u13(7)  := 0.5;
                    c13(1)  := 0.020_242_002_3827;
                    c13(2)  := 0.046_060_749_9189;
                    c13(3)  := 0.069_436_755_1099;
                    c13(4)  := 0.089_572_990_3810;
                    c13(5)  := 0.103_908_023_7684;
                    c13(6)  := 0.113_141_590_1314;
                    c13(7)  := 0.116_275_776_6154;
  inv_koef(13,u13,c13);

  u20(1)  := 0.003_435_700_4075;
  u20(2)  := 0.018_014_036_3610;
  u20(3)  := 0.043_882_785_8743;
  u20(4)  := 0.080_441_514_0889;
  u20(5)  := 0.126_834_046_7699;
  u20(6)  := 0.181_973_159_6367;
  u20(7)  := 0.244_566_499_0246;
  u20(8)  := 0.313_146_955_6423;
  u20(9)  := 0.386_107_074_4292;
  u20(10) := 0.461_736_739_4333;
                    c20(1)  := 0.008_807_003_5696;
                    c20(2)  := 0.020_300_714_9002;
                    c20(3)  := 0.031_336_024_1671;
                    c20(4)  := 0.041_638_370_7884;
                    c20(5)  := 0.050_965_059_9086;
                    c20(6)  := 0.059_097_265_9808;
                    c20(7)  := 0.065_844_319_2246;
                    c20(8)  := 0.071_048_054_6592;
                    c20(9)  := 0.074_586_493_2363;
                    c20(10) := 0.076_376_693_5654;
  inv_koef(20,u20,c20);

--  Kontrola  vrednosti  elementov polj c1, c2,
--     c3, c5, c8, c13 in c20

  new_line;
  put(" n      Vsota vseh  c(i)");
  new_line;
  put("----------------------------");
  csum(1,c1); csum(2,c2); csum(3,c3);
  csum(5,c5); csum(8,c8); csum(13,c13);
  csum(20,c20);
  new_line;

--------  Izracun vrednosti integralov  ---------

  cel_izpis(2.0,5.0,10,4);
  cel_izpis(1.0,4.0, 8,4);

END quad1;
```

Lista 15. Lista na prejšnji strani in zgoraj prikazuje paketno telo quad1 s proceduro quad; ta procedura izračuna vrednost določenega integrala po Gaussu. Abscisne in utežne norme so dane 13-mestno ((10)), procedura inv_koef pa izračuna še preostale normne vrednosti (simetrija). Funkcija f v proceduri quad je slobalna (Ada ne omogoča uporabe funkcijskih parametrov). Proceduro quad in paketno telo quad1 lahko primerjamo z algolsko proceduro quad in z obdajajočim algolskim programom v literaturi ((11, str. 34)).

5.3. Izračun integrala po Gaussu

Vrednosti integralov lahko izračunavamo z uporabo kvadraturnega integriranja po Gaussu, kot je bilo opisano v ((11, str. 33-34)). Osrednja procedura paketnega telesa quad1 v listi 15 je procedura

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

ki je primerna za hkratno integriranje več funkcij pri enakih integralskih mejah in pri enakih vozliščnih točkah. Integracijski interval (a, b) je razdeljen na m enakih podintervalov za n-točkovno kvadraturno integriranje. Parameter p procedure quad predstavlja število funkcij, ki jih nameravamo integrirati. Nadalje sta dani polji konstant c in u, kjer je c(k) utežno in u(k) abscisno normiranje pri k = 1, 2, 3, ... , n. Funkcija f(x,j) v proceduri quad je slobalna (Ada ne omogoča parametričnega klica funkcije) in izračuna j-to funkcijo argumenta x. V polju sum se shranjujejo integracijski rezultati in sum(j) je rezultat integracije funkcije f(x,j).

Procedura quad je v listi 15 druga (prva je funkcija f, ki je v quad slobalna in mora biti deklarirana pred njo) in je sestavljena skladno z Gaussovo formulo

$$\text{vsota} = \sum_{i=0}^{m} \sum_{k=1}^{n} c(k)*f(a+h*(i-1+u(k)))$$

kjer je h = (b-a)/m, koeficienti c(k) in abscise u(k) pa so dani v listi 15 za 1-, 2-, 3-, 5-, 8-, 13- in 20-točkovno kvadraturno integriranje ((10)).

V listi 15 je f(x,j) v proceduri quad j-ta podintegralska funkcija, j pa je v intervalu (1, p); tako lahko integriramo v enem izvajanju procedure p različnih podintegralskih funkcij. Funkcija f(x,j) v listi 15 vsebuje 4 funkcije, in sicer x*x, x*(1.0+x*x), 1.0/x in 1, ki bodo integrirane na intervalih (2.0, 5.0) in (1.0, 4.0) (glej predzadnji vrstici liste 15).

Procedura inv_koef v listi 15 izračuna simetrične abscisne in utežne norme, in sicer

```
u(n+1-k) := 1.0 - u(k);
c(n+1-k) := c(k);
```

Procedura csum pa nato izračuna (za primerjavo)

$$\sum_{k=1}^{n} c(k)$$

katere idealna vrednost naj bi bila 1.

Lista 16 prikazuje rezultate programske enote z liste 15 za vse 4 funkcije (Integral 1, ... , Integral 4) in za n-točkovno integracijo na m podintervalih (druga in tretja rezultatna skupina na tej listi). Na vrhu liste 16 imamo kontrolne vsote utežnih norm.

5.4. Kratka ocena natančnosti integrirnih algoritmov

Iz obravnavanih primerov (Simpson, Rombers, Gauss) je razvidno, da dobimo pri določenih pogojih dovolj natančne rezultate (idealne vred-

```
13A)quad1

 n      Vsota vseh  c(i)
-----------------------------
 1   1.00000000000000E+0
 2   1.00000000000000E+0
 3   1.00000000000000E+0
 5   1.00000000000000E+0
 8   1.00000000000000E+0
13   1.00100000000000E+0
20   1.00000000000040E+0
```

Lista 16. Ta lista prikazuje rezultate izvajanja paketnega telesa quad1 z liste 15. Najprej so izračunane vsote uteži c(i) za posamezne n-točkovne integracijske postopke. Idealna vrednost teh vsot je 1. Pri 13-točkovnem integriranju bodo uteži c(i) najbolj odstopale, saj je vsota 1,001. To odstopanje bo vplivalo tudi na natančnost rezultatov v drugi in tretji rezultatni skupini pri n = 13. Nekoliko manjše odstopanje imamo pri vsoti vseh c(i) za n = 20. Nekatere vrednosti integralov v drugi in tretji rezultatni skupini so idealne. Že sama delitev na podintervale vpliva na natančnost izračunov (npr. pri nižjetočkovni podintervalski integraciji).

m = 10

n	Integral 1	Integral 2	Integral 3	Integral 4
1	3.89775000000000E+1	1.62513750000000E+2	9.15506786189153E-1	3.00000000000000E+0
2	3.90000000000000E+1	1.62750000000000E+2	9.16290056510541E-1	3.00000000000000E+0
3	3.90000000000000E+1	1.62750000000000E+2	9.16290731219806E-1	3.00000000000000E+0
5	3.90000000000000E+1	1.62750000000000E+2	9.16290731874155E-1	3.00000000000000E+0
8	3.90000000000000E+1	1.62750000000000E+2	9.16290731874155E-1	3.00000000000000E+0
13	3.90390535135555E+1	1.62912806189233E+2	9.17207208470252E-1	3.00300000000000E+0
20	3.90000000000156E+1	1.62750000000065E+2	9.16290731874522E-1	3.00000000000120E+0

m = 8

n	Integral 1	Integral 2	Integral 3	Integral 4
1	2.09648437500000E+1	7.09863281250000E+1	1.38093545108453E+0	3.00000000000000E+0
2	2.10000000000000E+1	7.12500000000000E+1	1.38626914320686E+0	3.00000000000000E+0
3	2.10000000000000E+1	7.12500000000001E+1	1.38629422218607E+0	3.00000000000000E+0
5	2.10000000000000E+1	7.12500000000000E+1	1.38629436111486E+0	3.00000000000000E+0
8	2.10000000000000E+1	7.12500000000001E+1	1.38629436111989E+0	3.00000000000000E+0
13	2.10210083614930E+1	7.13213127111973E+1	1.38768190801016E+0	3.00300000000000E+0
20	2.10000000000084E+1	7.12500000000286E+1	1.38629436112045E+0	3.00000000000120E+0

nosti) pri podintegralskih funkcijah x*x, x*(1 + x*x) in 1. Nekoliko manj natančni pa so rezultati pri podintegralski funkciji tipa 1.0/x, katere integral je naravni logaritem log x. Tu velja vobče

$$\int_a^b \frac{dx}{x} = \log b - \log a, \qquad \int_{1.0}^{b} \frac{dx}{x} = \log b$$

Rezultate integriranja lahko tako primerjamo z rezultati "vgrajene" funkcije log v paketu "mathlib". Vzemimo program testlog in njegove rezultate v listi 17. Na osnovi teh in prejšnjih rezultatov lahko sestavimo tabelo 1.

Tabela 1.

testsim	n= 70	I3 =	0,698970157257503
testsim	n=200	I3 =	0,698970006646348
testromb	k= 6	I3 =	0,698970004807672
testromb	k= 7	I3 =	0,698970004336892
testromb	k= 8	I3 =	0,698970004336892
testlog			0,698970004336019
quad1	m= 10 n=5	I3 =	0,916290731874155
testlog			0,916290731874155
quad1	m= 8 n=5	I3 =	1,38629436111486
quad1	m= 8 n=8	I3 =	1,38629436111989
testlog			1,38629436111939

Če vzamemo log x (testlog) kot referenco, lahko iz tabele 1 usotovimo tole:

-- s povečevanjem števila točk (testsim, n = 200) se približamo vrednosti decimalnega logaritma log 5.0 le na 8 decimalnih mest (preostalih 7 mest se ne ujema)

-- testromb se pri k = 7 približa vrednosti decimalnega logaritma log 5.0 na 12 decimalnih mest (preostala 3 mesta se ne ujemajo)

-- quad1 se pri m = 10, n = 5 natanko ujema z razliko naravnih logaritmov log 5.0 -log 2.0

-- quad1 se pri m = 8, n = 8 natanko ujema z vrednostjo naravnega logaritma log 4.0

```
WITH floatio, floatops, util, mathlib;
PACKAGE BODY testlog IS
  USE mathlib;
  BEGIN
    new_line; put("Integral 3 (Simpson, ");
        put("Romberg) = "); new_line;
        put("                          ");
    floatio.put(0.434294481903251827655*
                log(5.0));
    new_line; put("Integral 3 (Gauss, ");
        put("1. primer) = "); new_line;
        put("                          ");
    floatio.put(log(5.0)-log(2.0));
    new_line; put("Integral 3 (Gauss, ");
        put("2. primer) = "); new_line;
        put("                          ");
    floatio.put(log(4.0));
  END testlog;
```

```
13A)testlog

Integral 3 (Simpson, Romberg) =
                        6.98970004336019E-1
Integral 3 (Gauss, 1. primer) =
                        9.16290731874155E-1
Integral 3 (Gauss, 2. primer) =
                        1.38629436111989E+0
```

Lista 17. Izračun vrednosti logaritmov za primerjavo z rezultati integrirnih algoritmov

-- splošno opažamo, da nekateri algoritmi ne izboljšajo rezultatov s povečevanjem števila točk (npr. testsim, testromb) in ne dosežejo 15-mestne natančnosti; pri drugih algoritmih (quad1, Gaussova metoda) pa je pri ustrezni izbiri točk dosegljiva 15-mestna natančnost (sledi na funkcijo 'log')

6. Izračun determinante

Procedura det v listi 18 izračuna determinanto matrike a razsežnosti n*n z metodo triangularizacije. Lista 18 vsebuje razen procedure det še proceduro matizpis (matrični izpis in izračun njene determinante) in glavni program, ki obsega inicializacijo posameznih matrik bb, cc, dd, ee (primerov za naše ocene) in štiri klice procedure matizpis za izpis štirih matrik in njihovih determinant, kot prikazuje lista 19.

Determinante matrik v listi 19 se ujemajo z idealnimi vrednostmi. Determinanta 1 je natančna vsaj na sedem mest ((12)).

Lista 18. Ta lista prikazuje paket za izračun determinante. Paket sestavljata dve proceduri, in sicer za izračun determinante (det) in za izpis matrike in vrednosti determinante (slej listo 19) in glavni preizkusni program; v njem imamo inicializacijo štirih matrik (bb, cc, dd, ee) in štirikratni klic procedure matizpis. Rezultati tega paketa so prikazani v listi 19.

```
--          Izračun   determinante

WITH floatio, floatops, lonsops, util;
PACKAGE BODY deter IS

  n: CONSTANT := 4;
  SUBTYPE real IS long_float;
  SUBTYPE dim IS integer RANGE 1 .. n;
  TYPE stolp IS ARRAY (dim) OF real;
--  Tip polje je matrika dimenzije  n X n
  TYPE polje IS ARRAY (dim) OF stolp;

--  Naslednja procedura je predmet naše  pozor-
--    nosti
--*************************************************

  PROCEDURE det (a: IN OUT polje;
                 n: IN integer;
                 detr :OUT real) IS
    i,j,k: integer; t,d,max: real;
    BEGIN
      d := 1.0;
      FOR k IN 1 .. n LOOP
        max := 0.0;
        FOR i IN k .. n LOOP
          t := a(i)(k);
          IF abs(t) > abs(max) THEN
            max := t; j := i; END IF;
        END LOOP;
        IF max = 0.0 THEN
          d := 0.0; GOTO fin; END IF;
        IF j /= k THEN
          d := -d;
          FOR i IN 1 .. n LOOP
            t := a(j)(i); a(j)(i) := a(k)(i);
            a(k)(i) := t;
          END LOOP;
        END IF;
        FOR i IN k+1 .. n LOOP
          t := a(i)(k)/max;
          FOR j IN k+1 .. n LOOP
            a(i)(j) := a(i)(j) - t*a(k)(j);
          END LOOP;
        END LOOP;
<<fin>>
        d := d*a(k)(k);
      END LOOP;
      detr := d;
    END det;
--*************************************************

  PROCEDURE matizpis (
                      i,pred,po,eks: IN integer;
                      aa: IN OUT polje;
                      n: IN integer) IS
    k,p: integer; determ: real;
    BEGIN
      new_line;
      put("Matrika"); put(i,2); put(" = ");
      new_line; put("----------");
      new_line;
      FOR k IN 1 .. n LOOP
        FOR p IN 1 .. n LOOP
          floatio.put(aa(k)(p),pred,po,eks);
          put("  ");
          IF p = n THEN new_line; END IF;
        END LOOP;
      END LOOP;
      put("      Determinanta"); put(i,2);
      put(" = ");
      det(aa,n,determ);
      floatio.put(determ);
      new_line;
    END matizpis;
-------------------------------------------------

  bb, cc, dd, ee: polje;

--------    Glavni preizkusni program   ----------

  BEGIN
--   Inicializacija matrik

    bb(1)(1) := 10.96597; bb(1)(2) := 35.10765;
    bb(1)(3) := 96.72356;
    bb(2)(1) :=  2.35765; bb(2)(2) :=-84.11256;
    bb(2)(3) :=  0.87932;
    bb(3)(1) := 18.24689; bb(3)(2) := 22.13579;
    bb(3)(3) :=  1.11123;

    cc(1)(1) :=  1.0;     cc(1)(2) :=  3.0;
    cc(1)(3) :=  3.0;     cc(1)(4) :=  1.0;
    cc(2)(1) :=  1.0;     cc(2)(2) :=  4.0;
    cc(2)(3) :=  6.0;     cc(2)(4) :=  4.0;
    cc(3)(1) :=  1.0;     cc(3)(2) :=  5.0;
    cc(3)(3) := 10.0;     cc(3)(4) := 10.0;
    cc(4)(1) :=  1.0;     cc(4)(2) :=  6.0;
    cc(4)(3) := 15.0;     cc(4)(4) := 20.0;

    dd(1)(1) := 0.0;      dd(1)(2) := 0.0;
    dd(1)(3) := 0.0;
    dd(2)(1) := 5.0;      dd(2)(2) := 9.0;
    dd(2)(3) := 2.0;
    dd(3)(1) := 7.0;      dd(3)(2) := 5.0;
    dd(3)(3) := 4.0;

    ee(1)(1) := 0.0;      ee(1)(2) := 0.0;
    ee(1)(3) := 0.0;      ee(1)(4) := 1.0;
    ee(2)(1) := 0.0;      ee(2)(2) := 0.0;
    ee(2)(3) := 1.0;      ee(2)(4) := 0.0;
    ee(3)(1) := 0.0;      ee(3)(2) := 1.0;
    ee(3)(3) := 0.0;      ee(3)(4) := 0.0;
    ee(4)(1) := 1.0;      ee(4)(2) := 0.0;
    ee(4)(3) := 0.0;      ee(4)(4) := 0.0;
-------------------------------------------------

    matizpis(1,3,5,0,bb,3);
    matizpis(2,2,1,0,cc,4);
    matizpis(3,1,1,0,dd,3);
    matizpis(4,1,0,0,ee,4);

  END deter;
```

```
i3A)deter

Matrika 1 =
-----------
 10.96597   35.10765   96.72356
  2.35765  -84.11256    0.87932
 18.24689   22.13579    1.11123
     Determinanta 1 =  1.52731360015724E+5

Matrika 2 =
-----------
 1.0   3.0   3.0   1.0
 1.0   4.0   6.0   4.0
 1.0   5.0  10.0  10.0
 1.0   6.0  15.0  20.0
     Determinanta 2 =  1.00000000000000E+0

Matrika 3 =
-----------
0.0  0.0  0.0
5.0  9.0  2.0
7.0  5.0  4.0
     Determinanta 3 =  0.00000000000000E+0

Matrika 4 =
-----------
0.  0.  0.  1.
0.  0.  1.  0.
0.  1.  0.  0.
1.  0.  0.  0.
     Determinanta 4 =  1.00000000000000E+0
```

Lista 19. Ta lista kaže izpise posameznih matrik in pripadajoče vrednosti determinant za programski paket iz liste 18.

7. Obrnitev matrike

Procedura invert v listi 20 obrne kvadratno matriko stopnje n z uporabo več elementarnih operacij nad vrsticami matrike matr, razširjene z dopolnitvijo njene enotine matrike. Primer izrojene matrike je indiciran z vrednostjo s = 1.

Program v listi 20 obrne matriko bb (sled njeno inicializacijo v listi), rezultat te obrnitve pa je prav končni segment matrike Hilberta, in sicer matrika

$$\begin{Vmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{Vmatrix}$$

(to je matrika 2 v listi 21). Obrat te matrike da zopet prvotno matriko, tako da velja

matrika 1 = matrika 3

(primerjaj v listi 21). Matrika 1 je idealno obrnjena matrika hilbertovske matrike (matrika 2). V programu liste 20 smo upoštevali lastnost

obrnitev(obrnitev(M)) = M

kjer je M dana začetna matrika.

Lista 20. Programski paket invmat rabi za preizkus obračanja kvadratnih matrik (lista se nadaljuje na naslednji strani).

```
------------    Inverzija matrike    ------------

WITH floatio, floatops, util;

PACKAGE BODY invmat IS

  n: CONSTANT := 4;
  SUBTYPE real IS long_float;
  SUBTYPE d1 IS integer RANGE 1 .. n;
  SUBTYPE d2 IS integer RANGE 1 .. 2*n;
  TYPE stolp IS ARRAY (d2) OF real;
  TYPE polje IS ARRAY (d1) OF stolp;

--  Naslednja procedura je predmet naše pozor-
--    nosi
--*********************************************

  PROCEDURE invert (
                matr: IN polje;
                matr1: OUT polje;
                s: OUT integer) IS
    --  n je globalna spremenljivka procedure
    t: real;
    i,j,k,m: integer;
    a: polje;

    BEGIN
      m := 2*n; s := 0;
      FOR i IN 1 .. n LOOP
        FOR j IN 1 .. m LOOP
          IF j <= n THEN
            a(i)(j) := matr(i)(j);
          ELSIF j = n+i THEN
            a(i)(j) := 1.0;
          ELSE
            a(i)(j) := 0.0;
          END IF;
        END LOOP;
      END LOOP;

----------      Začetek obračanja      ---------
      FOR i IN 1 .. n LOOP
        k := i;
<<test0>>
        IF a(k)(i) = 0.0 THEN
          s := 1;
          IF k < n THEN k := k+1;
          ELSE GOTO fin; END IF;
        GOTO test0;
        END IF;
        IF s = 1 THEN
          FOR j IN 1 .. m LOOP
            t := a(k)(j);
            a(k)(j) := a(i)(j);
            a(i)(j) := t;
          END LOOP;
        END IF;
        FOR j IN REVERSE i .. m LOOP
          a(i)(j) := a(i)(j)/a(i)(i);
        END LOOP;
        FOR k IN 1 .. n LOOP
          IF k /= i THEN
            FOR j IN REVERSE i .. m LOOP
              a(k)(j) := a(k)(j) -
                                a(i)(j)*a(k)(i);
            END LOOP;
          END IF;
        END LOOP;
      END LOOP;

      FOR i IN 1 .. n LOOP
        FOR j IN 1 .. n LOOP
          matr1(i)(j) := a(i)(j+n);
        END LOOP;
      END LOOP;
      s := 0;
<<fin>> null;
    END invert;
--*********************************************

  PROCEDURE matizp (
                      i,pred,po,eks: IN integer;
                      aa: IN OUT polje;
                      n: integer ) IS
```

```
    k, p: integer;
    BEGIN
      new_line;
      put("Matrika "); put(i,2); put(" = ");
      new_line; put("-----------");
      new_line;
      FOR k IN 1 .. n LOOP
        FOR p IN 1 .. n LOOP
          floatio.put(aa(k)(p), pred, po, eks);
          put("  ");
          IF p = n THEN new_line; END IF;
        END LOOP;
      END LOOP;
      new_line;
    END matizp;
-------------------------------------------------------------

  bb, cc, dd: polje;
  s: integer;

--------    Glavni preizkusni program    --------

  BEGIN
    --     Inicializacija matrike

    bb(1)(1) :=    16.0;   bb(1)(2) :=  -120.0;
    bb(1)(3) :=   240.0;   bb(1)(4) :=  -140.0;
    bb(2)(1) :=  -120.0;   bb(2)(2) :=  1200.0;
    bb(2)(3) := -2700.0;   bb(2)(4) :=  1680.0;
    bb(3)(1) :=   240.0;   bb(3)(2) := -2700.0;
    bb(3)(3) :=  6480.0;   bb(3)(4) := -4200.0;
    bb(4)(1) :=  -140.0;   bb(4)(2) :=  1680.0;
    bb(4)(3) := -4200.0;   bb(4)(4) :=  2800.0;
    -------------------------------------------------------

    matizp(1,5,1,0,bb,n);

    invert(bb,cc,s);
    IF s = 0 THEN matizp(2,1,5,0,cc,n); END IF;

    invert(cc,dd,s);
    IF s = 0 THEN matizp(3,5,1,0,dd,n); END IF;

  END invmat;
```

Lista 20 (nadaljevanje s prejšnje strani). Ta paket je sestavljen iz dveh procedur (invert in matizp) in iz glavnega preizkusnega programa. Inicializira se začetna matrika bb, matriki cc in dd pa sta dobljeni z dvema zaporednima obrnitvama, tako da je bb = dd.

```
13A)invmat

Matrika 1 =
-----------
   16.0   -120.0    240.0   -140.0
 -120.0   1200.0  -2700.0   1680.0
  240.0  -2700.0   6480.0  -4200.0
 -140.0   1680.0  -4200.0   2800.0


Matrika 2 =
-----------
1.00000  0.50000  0.33333  0.25000
0.50000  0.33333  0.25000  0.20000
0.33333  0.25000  0.20000  0.16667
0.25000  0.20000  0.16667  0.14286


Matrika 3 =
-----------
   16.0   -120.0    240.0   -140.0
 -120.0   1200.0  -2700.0   1680.0
  240.0  -2700.0   6480.0  -4200.0
 -140.0   1680.0  -4200.0   2800.0
```

Lista 21. Ta lista prikazuje rezultate programskega paketa z liste 20, ko imamo prikazane tri matrike. Matrika 2 je hilbertovska, matriki 1 in 3 pa sta njeni obrnitvi, in sicer tako, da je matrika 2 obrnitev matrike 1, matrika 3 pa obrnitev matrike 2.

8. Sklep

V tem članku smo pokazali primernost jezika Janus/Ada za programiranje številskih problemov. Aritmetika prevajalnika je zgrajena na standardu IEEE (procesor 8087) in njena natančnost je zadovoljiva (petnajstmestna). Tako je mogoče tudi z osebnim računalnikom (kot je v tem primeru Iskra-Delta Partner) dosegati zadovoljive laboratorijske, konstrukcijske in matematične rezultate. Vrsta algoritmov v tem članku je bila privzetih iz ((12)), tako da so bili algolski programi prepisani v adovske. Pri tem so se pojavljale le manjše modifikacije, ki so bile potrebne zaradi jezikovnih razlik. Vobče velja seveda usotovitev, da je Ada manj elegantna kot njen predhodnik Algol, ima pa druge prednosti, ki podpirajo zanesljivejše načine programiranja. V Adi ne moremo uporabljati procedurnih in funkcijskih imen parametrično, uporabljati jih moramo globalno. Meje adovskih polj morajo biti določene v času prevajanja: zopet lahko te meje definiramo globalno, tako da v določenem deklaracijskem stavku določimo ustrezno vrednost. Seveda pa imamo še druge omejitve, ki podaljšujejo zapis ekvivalentnega adovskega programa glede na algolski program.

Slovstvo

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

((2)) A.P.Železnikar: Algol 60 za sistem CP/M II. Informatica 8(1984), št.1, str.27-40.

((3)) A.P.Železnikar: Algol 60 za sistem CP/M III. Informatika 8(1984), št.2, str.31-41.

((4)) A.P.Železnikar: Programiranje v Adi I. Informatica 6(1982), št.3, str.10-22.

((5)) A.P.Železnikar: Programiranje v Adi II. Informatica 6(1982), št.4, str.19-29.

((6)) A.P.Železnikar: Programiranje v Adi III. Informatica 7(1983), št.1, str.28-37.

((7)) A.R.Miller: Pascal Programs for Scientists and Engineers. Sybex, Berkeley (Ca), 1981.

((8)) V.S.Linskij: Vyčislenie elementarnych funkcij na avtomatičeskih cifrovyh mašinah. Vyčislitel'naja matematika. Sbornik 2, Izd. AN SSSR, Moskva 1957, str. 94-95.

((9)) L.A.Ljusternik, O.A.Červonenkis, A.R.Janpol'skij: Matematičeskij analiz (Vyčislenie elementarnyh funkcij). Fizmatgiz, Moskva 1963.

((10)) Ja.S.Dymarskij, N.N.Lozinskij, A.T.Makuškin, V.Ja.Rozenberg, V.R.Erglis: Spravočnik programista. Tom pervyj. Sudpromizdat, Leningrad 1963 (str.107-117).

((11)) je enako kot ((2))

((12)) M.I.Ageev, V.P.Alik, R.M.Galis: Algoritmy (1-50). Vyčislitel'nyj centr AN SSSR, Moskva 1966.

PRIKLJUČITEV PISALNEGA STROJA NA MIKRORAČUNALNIK

DUŠAN VUKADIN

DO ISKRA DELTA

UDK: 681.3.06

V članku je opisana konkretna priključitev elektronskega pisalnega stroja na terminal ali mikroračunalnik. Opisana rešitev je bila realizirana že pred dvemi leti in deluje uspešno. Možna je prilagoditev vmesniške naprave za raznovrstne elektronske pisalne stroje. Namen tega članka je, da spodbudi bralce k reševanju sicer manjših toda večkrat perečih nalog, s katerimi se vsakodnevno srečujemo pri nas na področju računalništva.

Typewriter Interface for a Microcomputer

This article deals with a TTL interface between an electronic typewriter and a computer terminal or microcomputer. The described solution was realized two years ago and since that time is beeing in operation. The interface can be easily modified for various electronic typewriters. The aim of this article is to encourage readers for solving similar problems.

1. Uvod

Pomankanje kakovostnih lepopisnih tiskalnikov in potrebe po njih so narekovale, da najdemo rešitev s obstoječo opremo in priročnim materialom. Ta pristop je narekoval čim enostavnejšo rešitev, pomanjkanje kompletne dokumentacije in univerzalnost uporabe pa pristop, da se izvrši emulacia tipkovnice pisalnega stroja .

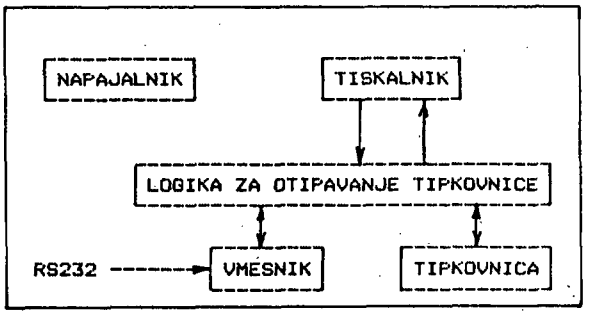
Vsi sodobni pisalni stroji (Olivetti, Olympia, mikroprocesorji in pri vseh imamo tri bistvene dele :

- tiskalnik,
- tipkovnico in
- napajalnik

Po vgraditvi vmesnika mora pisalni stroj obdržati svojo osnovno funkcijo pisalnega stroja, pridobiti pa mora še možnost, da se njegov tiskalniški del lahko uporablja kot izhodna serijska enota miroračunalniškega sistema. Hitrost tiskanja takih pisalnih strojev se giblje med 100 do 250 znaki na minuto, kar je sicer zadosti za ročni izpis vendar in bistveno prepočasi za strojni izpis. Poleg emulacije same tipkovnice prevzame vmesnik še sinhronizacijo izpisa skladno z mehanskimi zmožnostmi tiskalnika. Priključitev mikroračunalnika je izvedena s standardno povezavo RS-232C.

2. Opis rešitve

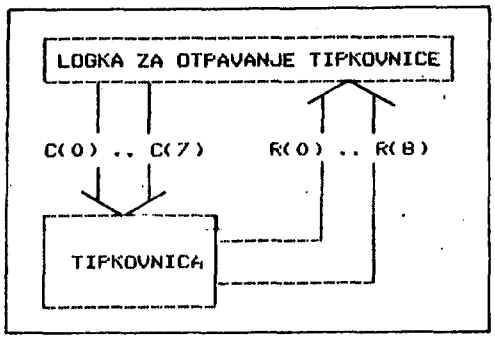
Vmesnik za priklop pisalnega stroja je v bistvu emulacija tipkovnice pisalnega stroja in je povezan paralelno k tipkovnici. Povezavo kaže slika 1.



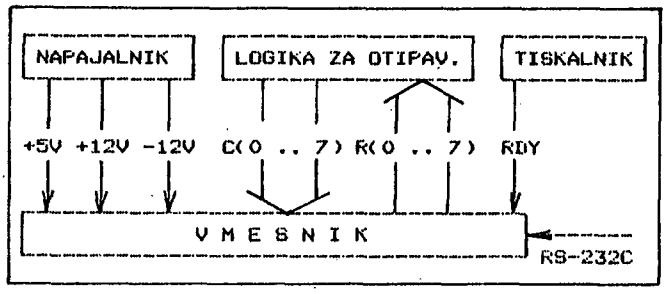
Slika 1. Povezava vmesnika z logiko za otipavanje tipkovnice pisalnega stroja

Logika za otipavanje tipkovnice presleduje matrično tipkovnico, tako da v izbranem stolpcu C(i) ugotavlja, katera vrstica R(i) ima vrednost 0. Iz tega ugotovi, katera tipka je bila pritisnjena . Ta podatek se posreduje tiskalniku , ki izpiše ustreznen znak ali izvrši ustrezen ukaz. Slika 2 prikazuje povezavo med tipkovnico in logiko za otipavanje tipkovnice.

Poleg signalov C(i) in R(i) je na vmesnik pripeljan še signal, ki dopove vmesniku, da je tiskalnik prost. Napajalna napetost za vmesnik je vzeta iz samega pisalnega stroja. Na sliki 3 je prikazana povezava vmesnika in pisalnega stroja.



Slika 2. Povezava med tipkovnico in logiko za otipavanja tipkovnice v pisalnem stroju.



Slika 3. Povezava vmesnika z elementi pisalnega stroja.

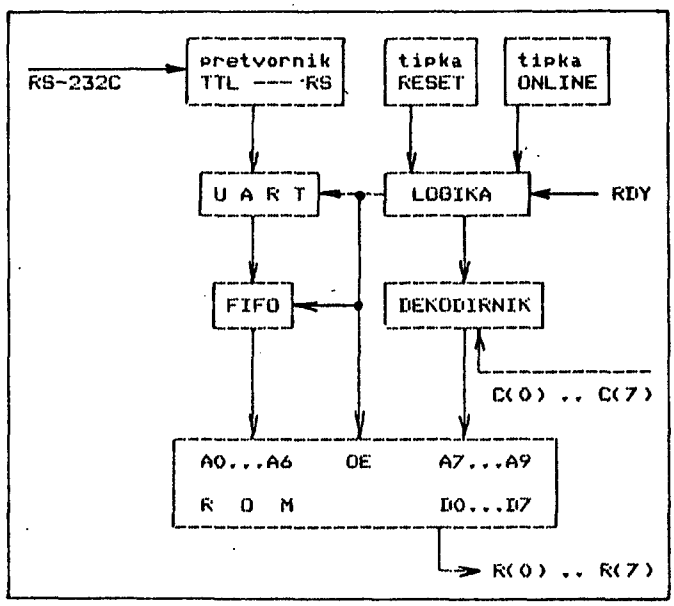
2.1. Princip delovanja vmesnika

Serijski vhodni signal sprejme UART in sa pretvori v paralelni signal in posreduje naprej v FIFO pomnilnik. Pomnilnik FIFO odda sprejeto kodo s pomočjo logike v ROM kot prvi del naslova. Drugi del naslova posreduje dekodirnik iz tipalnih linij C(0) do C(7). Sočasnost kode in položaja tipke znaka, ki sa koda predstavlja prek preslikave, ki je vpisana v ROMu, povzroči, da se v vrsticah R(0) do R(7) pojavi ena sama ničla , ki predstavlja pritisnjeno tipko. To ničlo posreduje logika za otipavanje tipkovnice tiskalniku, ki izvrši ustrezen ukaz. Logika nadzoruje poleg sinhronizacije izpisa še napolnjenost FIFO pomnilnika, sinhronizira sprejem s posredovanjem RTS signala ali pošiljanjem DC1 in DC3 kod, daje takt za hitrost prenosa in omogoča pravilno časovno zaporedje signalov proti pisalnem stroju . Od zunajnih ukazov ima vmesnik dve tipki, in sicer RESET in ONLINE/OFFLINE . Svetleči indikator gori, ko je vmesnik v položaju ONLINE. Slika 4 ponazarja princip delovanja vmesnika.

Princip delovanja vmesnika je v bistvu isti za priključitev vseh pisalnih strojev . Razlika je lahko samo v izbiri vira (izvora) za RDY signal. Poleg pisalnih strojev se na vmesnik z manjšimi predelavami lahko priključi tudi paralelni tiskalnik.

3. Izvedba vmesnika

Vmesnik je izveden z MOS in TTL integriranimi vezji. Slika 5 je vezalni načrt vmesnika za priključitev pisalnega stroja OLIVETTI ET221.



Slika 4. Blokovna shema vmesnika

3.1. Opis vezalnega načrta

Integrirani vezji E4 in E23 opravljata RS/TTL pretvorbo; vezje E19 je UART, vezji E18 in E21 oblikujeta 8-bitni FIFO, vezje E7 je dekodirnik, vezje E22 pa ROM, ki preslikava ASCII kode na tipke (znake pisalnega stroja). Vezje za resetiranje je E13, preklopnik za ON-LINE/OFFLINE pa polovica vezja E8. Vsaka sprememba signala RDY povzroči pošiljanje DC1 ali DC3 kode, kar omogoča vezje, ki sa oblikujejo elementi E16, E4, E23, E17, E1 in polovica elementa E8. Preostala integrirana vezja oblikujejo kontrolno logiko.

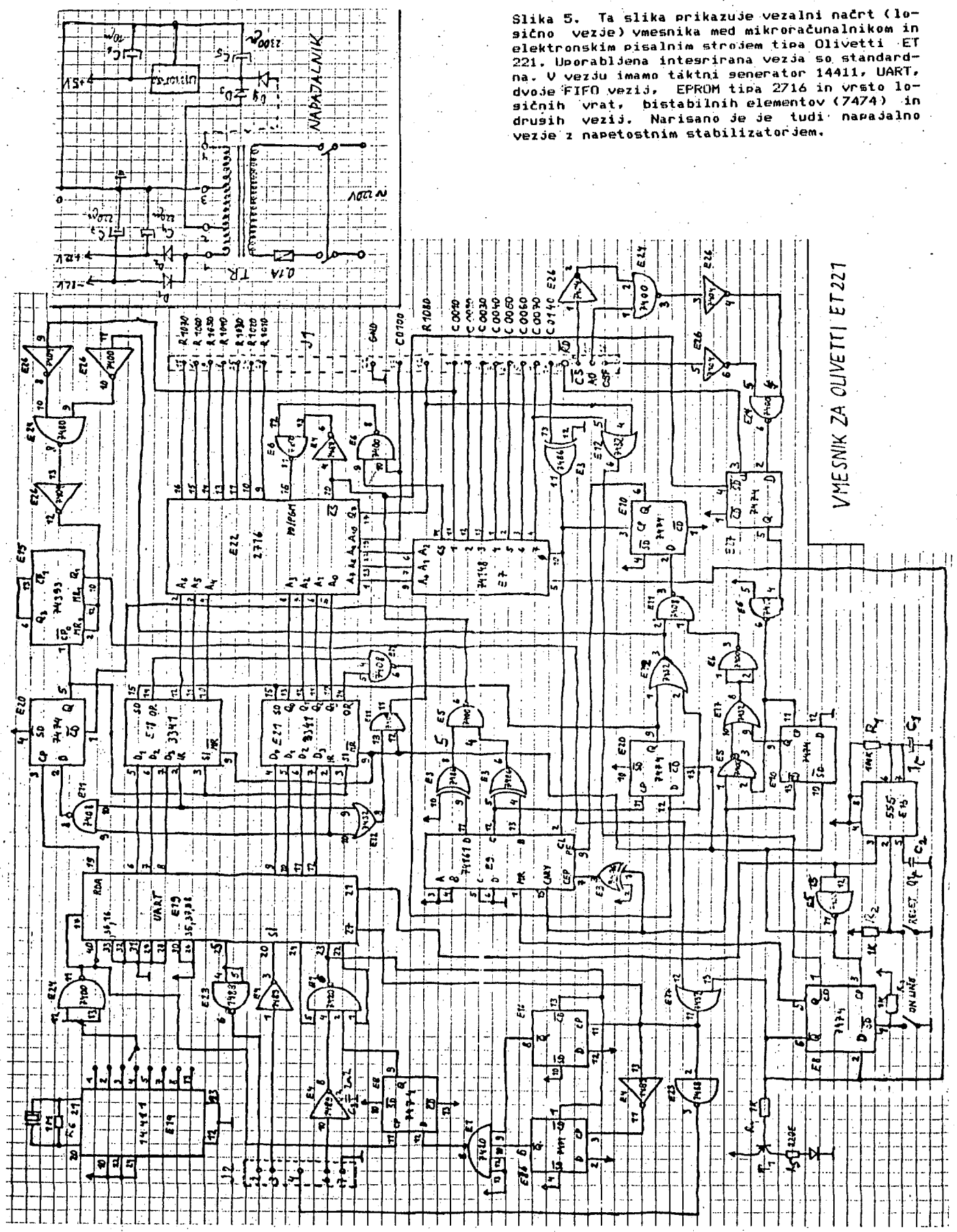
Specifičnost povezave na pisalni stroj ET221 je vezje, ki sa oblikujejo elementi E27, E24 in E26. To vezje zazna zahtevo pisalnega stroja za podatek iz tipkovnice in generira RDY signal, ki sa logika uporabi za sinhronizacijo izpisa. To vezje omogoča uporabo internega vmesnega pomnilnika. Drugi pisalni stroji pa posredujejo RDY signal direktno na vezje E6, nožica 4. Priključitev pisalnega stroja na vmesnik je izvedena prek priključnice DB-25-S; povezava tipa RS-232C pa usteza standardu.

Tabela 1 kaže kodno tabelo za pisalni stroj Olivetti ET221.

4. Sklep

Vmesnik je bil narejen z namenom, da se na terminalu in malih računalniških sistemih omogoči izpis dopisov in krajših tekstov. Vmesnik je bil v ožičeni obliki izdelan že pred dvema letoma. Izkušnje so pokazale, da je naprava zelo uporabna in da ustreza namenu, s katerim je bila narejena. Za preizkusno serijo so bila izdelana tiskana vezja, toda zaradi pomanjkanja pisalnih strojev na trgu naprava ni doživela serijske proizvodnje. Hitrost izpisa je

Slika 5. Ta slika prikazuje vezalni načrt (logično vezje) vmesnika med mikroračunalnikom in elektronskim pisalnim strojem tipa Olivetti ET 221. Uporabljena integrirana vezja so standardna. V vezju imamo taktni generator 14411, UART, dvoje FIFO vezij, EPROM tipa 2716 in vrsto logičnih vrat, bistabilnih elementov (7474) in drugih vezij. Narisano je je tudi napajalno vezje z napetostnim stabilizatorjem.



Znakom na tipkovnici pisalnega stroja ustrezajo naslednje ASCII heksadecimalne kode, ki jih pošiljata računalnik ali terminal:

1 31	# 20	o 6F	O 4F	b 62
2 32	" 22	p 70	P 50	B 42
3 33	% 25	ž 78	Š 58	n 6E
4 34	/ 2F	d 7C	D 5C	N 4E
5 35	! 21	a 61	A 41	m 6D
6 36	(28	s 73	S 53	M 4D
7 37	) 29	d 64	D 44	, 2C
8 38	_ 5F	f 66	F 46	. 2E
9 39	: 3A	g 67	G 47	? 3F
- 2D	; 3B	h 68	H 48	č 26
¨ 20	´ 20	j 6A	J 4A	ž 60
^ 20	` 20	k 6B	K 4B	Ž 40
q 71	Q 51	l 6C	L 4C	BS 08
w 77	W 57	č 7E	Č 5E	SP 20
e 65	E 45	ć 7D	Ć 5D	TAB 09
r 72	R 52	0 30	= 3D	DEC TAB 0B
t 74	T 54	y 79	Y 59	CR 0A
z 7A	Z 5A	x 78	X 58	CTRL L 0C
u 75	U 55	c 63	C 43	≠ 20
i 69	I 49	v 76	V 56	

nekoliko nižja od hitrosti, ki jo deklarira proizvajalec pisalnega stroja. To naj bi razbremenilo mehaniko in motor v pisalnem stroju .

Tabela 1. Gornja tabela prikazuje razpoložljive znake pisalnega stroja Olivetti ET 221 in pripadajoče ASCII kode. Ta pisalni stroj ima marjetice (različne pisave) za YU abecedo.

NOVA DRUŽINA MIKROPROCESORJEV Z 800

MAKS TUTA

ISKRA – AVTOELEKTRIKA, TOZD AET TOLMIN

UDK: 681.3.06 Z800

An advanced mikroprocessor family adds on-chip cache and memory management yet retains software compatibility with its predcessor – Z80. Increased speed, aditional instructions and an addressing space that extends the available memory give greater flexibility.

Nova družina mikroprocesorjev ima na vezju "cache" spomin in spominsko upravljalno enoto, a kljub temu ostaja programsko kompatibilna s svojim predhodnikom – Z80. Povečana hitrost, dodatni ukazi in shema za naslavljanje, ki razširi spominski prostor, omogočajo večjo fleksibilnost.

1. SPLOSNO

Splošen trend je uporaba visokih programskih jezikov na mikroračunalniških sistemih. Konfiguracija postaja vedno bolj kompleksna zaradi potreb po večjem centralnem spominu, večji izvajalni hitrosti in enostavnejšem dostopu do programskih knjižnic – na splošno je to zahteva po bolj zahtevni arhitekturi procesorja. Že leta snovalci niso mogli izrabiti vseh možnosti, ki so jih nudila hitra RAM vezja – drugče rečeno procesor je zaviral razvoj mikroračunalnikov.

Ta situcija pa se bo spremenila z uvedbo nove družine 8- in 16-bitnih procesorjev. Nasledniki popularnega mikroprocesorja Z80 bodo lahko delovali s frekvenco 25M Hz in bodo lahko uporabljali blokovni način prenosa (burst mode). Toda to še ni vse.

Družina Z800 je izdelana na osnovi nove NMOS tehnologije in ima na vezju tudi "cache" spomin, spominsko upravljalno enoto, števnike-časovnike, DMA kontroler in serijski V/I kontroler. Dodane so nove instrukcije, ki olajšajo razvoj programov in programer ima možnost virtualnega naslavljanja spomina.

Družina je sestavljena iz štirih članov, dva procesorja imata 8-bitni vmesnik, ki je kompatibilen s procesorjem Z80 in dva s 16-bitnim vmesnikom za Z-vodilo (družina Z8000). Vsi člani družine so popolnoma programsko kompatibilni z mikroprocesorjem Z80. Nove instrukcije, skupaj z novostmi na vezju in visoko časovno frekvenco, razširijo sposobnosti na 5-miljonov-instrukcij/s, kar je bilo simulirano s Pascalskim prevajalnikom.

Družina Z800 je sestavljena iz 8-bitnih procesorjev Z8108 in Z8208 in iz 16-bitnih procesorjev Z8116 in Z8216 (glej tabelo 1). Seveda imata samo Z8208 in Z8216 na vezju vso periferijo in popoln 16M zložni naslovni prostor. Da se zmanjša poraba prostora na tiskanih ploščah, sta ta dva procesorija, ki imata 64 nožic, vgrajena v vezja, ki imajo zmanjšano razdaljo med nožicami, tako da zavzamejo prostor kot normalno vezje z 48-imi nožicami (slika 2).

Z vmesnikom za Z-vodilo nudi procesor dvakrat toliko možnosti kot z vodilo za 8-bitne periferne naprave; lahko uporablja vse prednosti perifernih naprav za Z-vodilo, ki so že dostopne za družino 16-bitnih mikroprocesorjev Z8000.

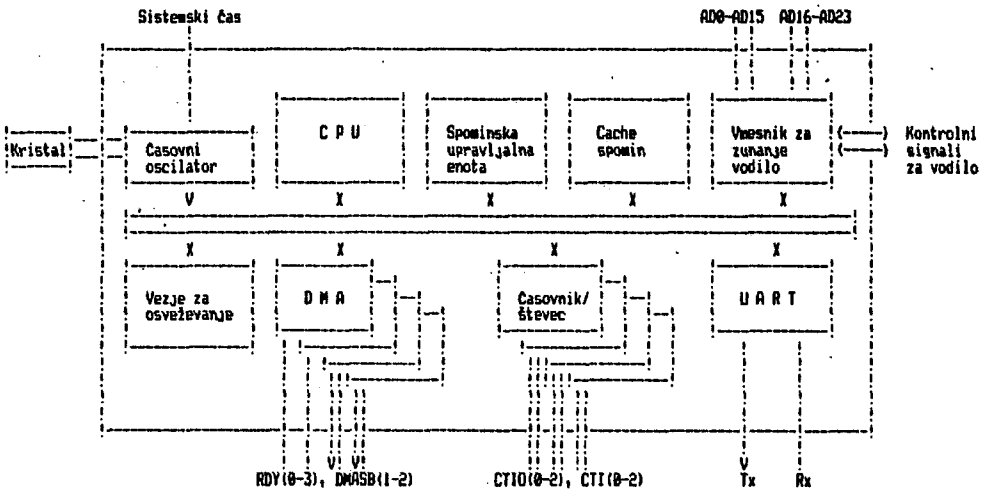
2. ARHITEKTURA PROCESORJA

V tem poglaviju si bomo ogledali vse bistvene novosti glede na procesor Z80, ki je predhodnik družine procesorjev Z800. V nadaljnjih poglavjih pa si bomo podrobneje ogledali posamezne novosti.

Arhitektura jedra procesorja Z800 je enaka kot pri Z80, seveda z dodatkom nekaj registrov, ki

	Z8108	Z8116	Z8208	Z8216
Podnožije (št. nožic)	48	48	64	64
Podatkovno vodilo (bitov)	8	16	8	16
Periferija na veziju	-štirije 16-bitni števci/časovniki (samo notranji)		-štirije 16-bitni števci/časovniki (eden notranji) -štirije DMA kanali -Serisko komunikacija	
Skupne lastnosti	- Spominska upravljalna enota - "chace" spomin - generator za osveževanje spomina - časovni oscilator			

Tabela 1: Različni predstavniki procesorske družine Z800 imajo različno konfiguracijo



Slika 1: Najbolj sposobni vezji iz družine Z800, to sta Z8216 in Z8208, imata na vezju vse sestavne dele potrebne za mikroračunalnik s spominsko upravljalno enoto, "cache" spomin, štiri DMA kanale, štiri števnike/časovnike in seriska vrata.

omogočajo večjo fleksibilnost. Ker je binarna koda procesorjev Z800 kompatibilna z Z80, ima vse registre kot Z80; dve 8-bitni banki registrov A-L in A'-L', dva 16-bitna indeksna registra IX in IY, dva 16-bitna skladovna kazalnika ter programski števec. Z800 ima dodatni glavni statusni register, ki vsebuje zastavice za indikacijo procesorjevega statusa. Poleg tega ima še števec za prekinitve in za notranje softwarske prekinitve (trap-vektor) ter register za V/I stran.

Kot del sprememb v arhitekturi lahko smatramo tudi možnost, da procesor lahko postavimo v sistemski način delovanja ali pa v uporabnikov način. Zato imamo tudi dva skladovna kazalca, enega za sistemske programe (vštevši prekinitve in notranje softwarske prekinitve - traps) in drugega za uporabnikov program. V sistemskem načinu lahko izvajamo vse ukaze in so nam dostopni vsi registri CPU-ja. Ta način naj bi se uporabljal za programe, ki izvajajo funkcije operacijskega sistema, lahko pa izvaja tudi emulacijo programov Z80.

V uporabnikovem načinu ne moremo izvajati nekaterih ukazov in nekateri registri nam niso dostopni. Uporabniški način je kot nekakšna podmnožica Z80 ukazov, ker so nekateri Z80 ukazi, kot je na primer "Stop", v Z800 priviligirani in jih lahko izvajamo samo v sistemskem načinu. Programi, pisani za Z80, so popolnoma pravilno izvedljivi na Z800, ker se procesor postavi v sistemski način ob vklopu napajanja.

Z800 vsebuje nekatere nedokumentirane instrukcije procesorja Z80 (kot so direkten dostop do polovice indeksnih registrov), z namenom, da naredi obstoječe registre bolj splošno uporabne. Štirje novi načini naslavljanja omogočajo večjo fleksibilnost obstoječih ukazov in enostavnejšo generacijo kode za višje programske jezike. Poleg tega ima Z800 ukaz 'Test and Set', ki omogoča sinhronizacijo v večprocesorskih sistemih in tudi ukaze za 8-bitno in 16-bitno množenje in deljenje, kar izboljša hitrost v računskih aplikacijah.

S programiranim zunanjim časovnim signalom se povečuje učinkovitost sistema. Eden od kontrolnih bitov dovoljuje zmanjšanje notranjega časovnega signala za zunanje vodilo in avtomatično se dodajajo čakalna (wait) stanja za pristop k zunanjemu vodilu: Uporabnik lahko izbere zelo veliko hitrost za izvajanje instrukcij (s tem izboljša performance sistema), ni pa nujno, da ima zelo hiter spomin in V/I naprave.

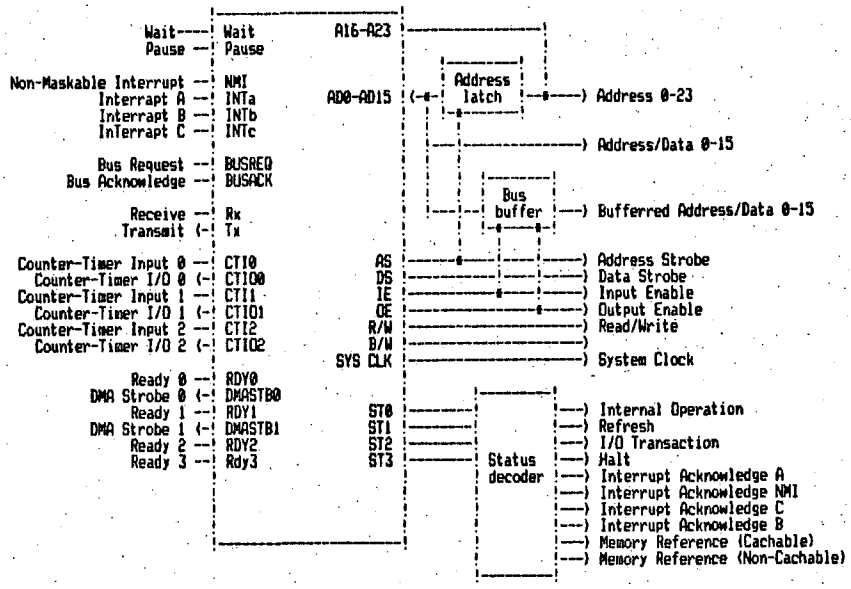
Prekinitvene možnosti procesorja Z80 so pri Z800 razširjene z uvedbo notranjih softwarskih prekinitev (traps) za posebnosti in pogojne napake ter za pospešen servisni-prekinitveni nčin. Ta novi način omogoča avtomatsko postavljanje vektorjev za vsako prekinitev in vsako programsko prekinitev ter omogoča procesiranje vgnezdenih prekinitvenih struktur.

Z dodatkom verige potrditve prekinitvenih zakasnitev lahko vsebino kontrolnega registra uporabimo za izbiro dodatnega števila čakalnih (wait) stanj, ki se bodo dodala k ciklom, ki zahtevajo prekinitev. Tako lahko uporabljamo počasno periferijo in dolge prekinitvene verige.

Naslovni prostor Z80 za vhode/izhode je pri Z800 povečan z dodatkom registra za V/I strani, kar omogoča izbiro enega od blokov V/I lokacji. Sprememba tega registra je privilegirana operacija, kar preprečuje vsak nezakonit pristop k temu registru.

Z800 vsebuje na vezju tudi kontroler za osveževanje dinamičnih spominskih vezji. Osveževanje lahko omogočimo ali onemogočimo s programom, izberemo pa lahko tudi frekvenco osveževanja. Z800 generira ločene cikle za osveževanje, kar olajša zahteve pri spominskem dostopnem času, kar ni primer pri Z80. Osveževalni cikli, ki se izgubljajo zaradi aktivnosti DMA-ja ali zaradi čakalnega (wait) stanja, se štejejo in se avtomatično generirajo, ko dobi CPU zopet kontrolo nad vodilom. Z800 števec za osveževanje generira 10-bitne oseveževalne naslove, kar dopušča uporabo zelo velikih dinamičnih RAM vezij. Ta generator osveževalnih naslovov je 10-bitni, kar omogoča uporabo vezij do 1M-bit brez zunanje logike za osveževanje.

Z800 vsebuje na vezju tudi oscilator-generator takta, kar poenostavi sistem, ker ne potrebujemo zunanjih MOS vezij za generacijo časovnega signala. Kristal lahko priključimo direktno na procesor, lahko pa uporabimo tudi



Slika 2: Priključki mikroprocesorja Z8216, z potrebnimi vmesniki in dekoderji za maksimalno konfiguracijo

zunanji TTL-komparibilen časovnih signal. Iz tega signala procesor generira notranji časovni signal, katerega frekvenca je enaka polovici vhodne.

Z800 ima na vezju spominsko upravljalno enoto (memory managment unit - MMU), ki jo lahko uporablja na različne načine. MMU preslika sistemske in uporabniške programe in ločeno komandni in podatkovni del ter enostavno premešča strani v spominu na različna fizična področja spomina, s tem omogoča lahek dostop do fizično zelo velikih spominskih področij. Direkten dostop do MMU-evega hardwara je mogoč samo s sistemskim programom.

Z800 ima na vezju tudi "cache" spomin, to je zelo hiter spomin, ki poveča storilnost procesorja. Procesor ima v tem spominu shranjene dele programa ali podatke, ki jih je nazadnje uporabil.

Na vezje procesorja Z800 je vgrajeno tudi nekaj periferije, to so : štirje popolni DMA kanali, štirje števci/časovniki in serijska V/I komunikacija.

3. DODATNI NAČINI NASLAVLJANJA

Poleg razširitve nabora instrukcij procesorja Z80 s štirimi novimi načini naslavljanja, ima Z800 razširjene tudi obstoječe načine naslavljanja (kot na primer naslavljanje Indeksnih registerov) na druge ukaze. Novi načini naslavljanja so: Indeksni s 16-bitnim odmikom, Relativno glede na kazalec sklada, Relativno glede na programski števec in Indeksno glede na bazo.

Indeksni način naslavljanja s 16-bitnim odmikom je razširitev Indeksnega načina pri Z80 in uporablja dvozlogovni odmik namesto enozlogovnega. Ta metoda dovoljuje dostop do večjih dinamičnih podatkovnih struktur s kazalcem ali doseg v polja, katerih osnovni naslov je znan - spreminja se le indeksna vrednost.

Relativno naslavljanje glede na kazalec sklada je uporabno za aplikacije višjih programskih jezikov, kjer so parametri podprogramov in lokalne spremenljvke shranjene na skladu. Naslovi teh spremenljivk so enako odmaknjeni od tekočega vrha sklada (določa ga kazalec sklada) in postanejo dostopni direktno z uporabo relativnega načina naslavljanja glede na kazalec sklada.

Z relativnim naslavljanjem glede na programski števec tvorimo lahko programe, ki imajo kodo neodvisno od pozicije - to pomeni, da program uporablja samo naslove relativno na programski števec ne pa absolutnih naslovov. To je ugodno pri standardnih ROM programih in knjižničnih podprogramih, ki jih nalagamo na različne lokacije v spomin pri različnih aplikacijah, poleg tega pa skrajša čas, potreben za povezovanje velikih programov. Z80 ima nekaj relativnih ukazov glede na programski števec (vsi so skoki), toda Z800 ukazi, glede na programski števec, obsegajo vse pogojne skoke in klice, kot tudi 8-bitne in 16-bitne ukaze naloži, shrani in aritmetične ukaze.

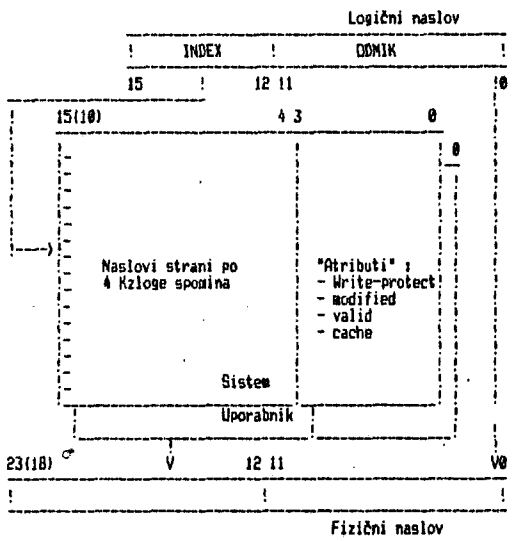
Bazni indeksni način naslavljanja uporablja dva registra za naslovitev operanda (uporabimo lahko katero koli kombinacijo HL, IX in IY). Vsebina obeh registrov se sešteje, da dobimo pravi naslov. Na ta način oba naslova (osnovni naslov strukture in indeks ali odmik) izračunamo v izvajalnem času (kar je zahteva za dinamične podatkovne strukture). Poleg tega lahko bazni indeksni način naslavljanja kombiniramo z drugimi naslovnimi načini, z uporabo ukaza LDA (Load Address) lahko gradimo poljuben kompleksen način naslavljanja, ki vsebuje poljubno kombinacijo indeksnega in indirektnega naslavljanja.

4. NOVI UKAZI

Poleg novih načinov naslavljanja so tudi stari načini naslavljanja uporabljeni za več ukazov - na primer 16-bitno nalaganje in shranjevanje uporablja indirektni registerski način in kratki indeksni način, 16 -bitno seštevanje dovoljuje takojšnji operand, PUSH uporablja direktno vrednost, PUSH in POP pa uporabljata

direktni dostop do spomina. Te razširitve dajejo procesorju Z800 moč in fleksibilnost pri programiranju tako v visko-nivojskih jezikih kot tudi pri programiranju na nivoju strojnega programa.

Z800 ima nove ukaze za množenje in deljenje. Ukazi za množenje imajo več variant, ki vključujejo množenje 8- krat 8-bitov s 16-bitnim rezultatom in 16- krat 16-bitov z 32-bitnim rezultatom, pri čemer operande naslavljamo na poljuben način. Podobno tudi deljenje obsega deljenje 16-bitov z 8-biti v 8-bitni kvocijent in ostanek in deljenje 32-bitnov s 16 -biti v 16-bitni kvocijent in ostanek. Ukazi za deljenje preverjajo prekoračitev kvocijenta in opozarjajo na deljenje z ničlo; ti pogoji povzročijo notranjo softwarsko prekinitev in prisilijo operaciski sistem, da izpiše opozorilno sporočilo, ali pa



Slika 3: Relacije med logičnimi in fizičnimi naslovi

prekinejo uporabniški program.

Ukaz "Test and Set" je vključen zaradi multiprocesiranja. Ta primitivni ukaz se pogosto uporablja kot signal med dvemi ali več sodelujočimi programi za zagotovitev ekskluzivnega dostopa do posameznih enot na vodilu.

Poleg 16-bitnega množenja in deljenja arhitektura Z800 vključuje tudi druge 16-bitne aritmetične operacije, ki jih Z80 ne vključuje. Te instrukcije vključujejo 8-bitno in 16-bitno "Prištevanje akumulatorja k naslovnemu registru" (Add Accumulator to Addressing Register), 16-bitno primerjanje, povečevanje ali zmanjševanje 16-bitne vrednosti v spominu, 16-bitno negiranje in popolno 16-bitno seštevanje in odštevanje. Vsi ti ukazi uporabljajo HL registerski par kot 16-bitni akumulator.

Obstoječi registri so pri Z800 bolje izkoriščeni. Registra IX in IY sta dostopna kot 16-bitna ali vsak kot dva 8-bitna (uporabljamo lahko vse 8-bitne ukaze: shrani, naloži in aritmetični ukazi). Ta možnost izenači IX in IY registra z ostalimi splošnimi registrskimi pari BC, DE in HL.

Z800 vključuje novo grupo ukazov za kontrolo CPU-ja, ti ukazi omogočajo dostop do novih registrov (register za V/I strani in glavni statusni register) in omogočajo uporabo sistemskega in uporabniškega načina. Ukaz LDCTL (Load Control) naloži podatke ali premakne in shrani podatke iz posebnih CPU registrov. Samo v sistemskem načinu je mogoče inicijalizirati register za V/I stran, prekinitve in tabelo kazalcev za notranje programske prekinitve (traps).

Privilegirane ukaze lahko izvajamo samo v programih, ki se izvajajo v sistemskem načinu. Ti ukazi imajo nadzor nad registri in procesorskimi stanji, in lahko bi rekli, da so del operacijskega sistema. Privilegirani ukazi so: "Stop", "Omogoči prekinitev", "Onemogoči prekinitv", "Izberi prekinitveni način", "Naloži kontrolne registre CPU-ja" ter "Vrni se iz prekinitve".

Ukaz SC (System Call) omogoča povezavo med programi v uporabnikovem načinu in operacijskim sistemom, ki se izvaja v sistemskem načinu. Ukaz SC shrani procesorjev status (vrednost programskega števca in glavnega statusnega registra) na sklad, shrani 16-bitno sistemsko klicno številko SC ukaza na sklad, in potem izvrši notranjo programsko prekinitev (trap). Operacijski sistem, po naslovitvi primerne servisne rutine za notranjo programsko prekinitev, normalno uporabi sistemsko klicno številko kot indeks v tabelo podprogramskih naslovov za različne sistemske funkcije. Ta kontrolni mehanizem omogoča uporabnikovim programom zahtevati privilegirane usluge, (kot so upravljanje s spominom, itd.) ne da bi bilo potrebno zaradi tega preiti v delovanje v sistemskem načinu - s tem izgubimo zaščito operacijskega sistema.

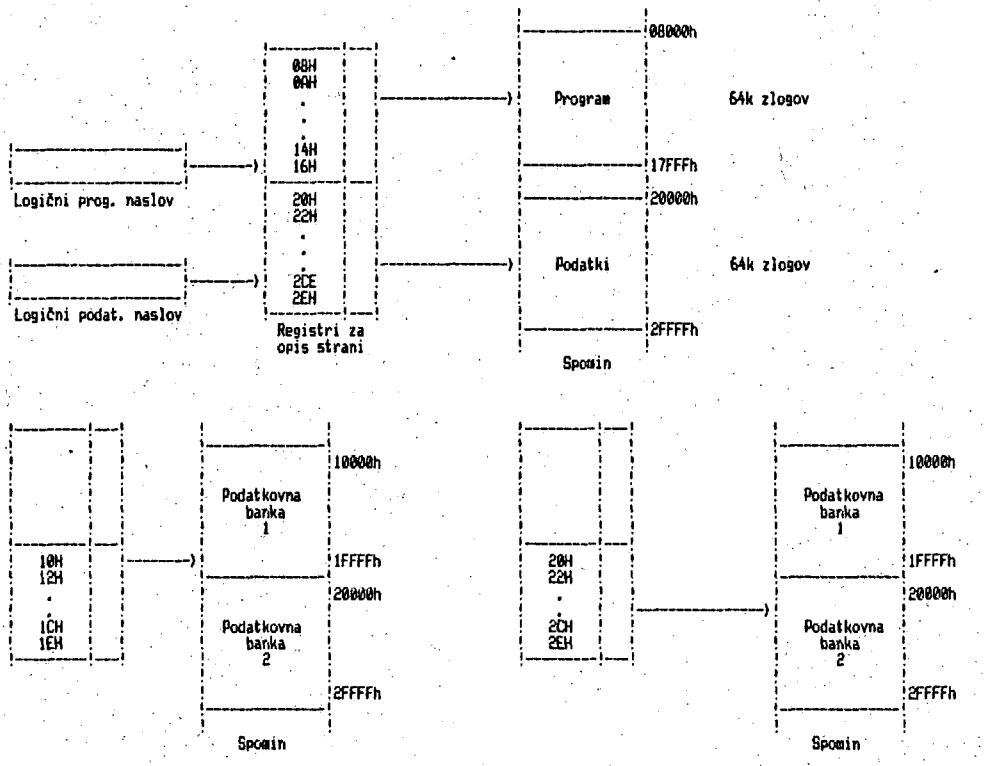
Nekateri ukazi umogočajo tudi sodelovanje z dodatnimi procesnimi enotami, kot je recimo prihajajoči Z8070 - matematični procesor za računanje s plavajočo vejico.

5. SPOMINSKA UPRAVLJALNA ENOTA

Eden od perečih problemov današnjih mikroprocesorskih sistemov je upravljanje z velikimi programskimi in/ali podatkovnimi spominskimi prostori. Ta problem se je reševalo na različne načine, kot na primer z dodajanjem zunanjih vezij za preslikavo spomina (to poveča porabo prostora na plošči in kompleksnost) ali pa je potrebno popolnoma spremeniti konfiguracijo in uporabiti 16-bitni procesor (s tem izgubimo kompatibilnost z obstoječo kodo in poveča se razvojni čas novega izdelka).

Mikroprocesor Z800 rešuje problem z uporabo spominske upravljalne enote - MMU, ki dovoljuje preslikavo spominskih strani in omogoča zaščito brez kakršnekoli zunanje logike. CPU sam loči sistemski prostor od uporabnikovega, poleg tega pa loči še programsko kodo od podatkovne v obeh spominskih prostorih - torej lahko govorimo o štirikrat večjem dostopnem spominskem prostoru brez spremembe programa ali dodajanjem zunanjega hardwara. Mehanizem za transformacijo naslovov, ki se imenuje "dinamično relociranje strani", je uporabljen za preslikavo teh logičnih naslovov v fizični naslovni prostor. Logični naslovi, ki jih generira CPU, gredo preko MMU-ja, ki jih prevede v fizične naslove in pošlje na naslovne linje, ki pridejo iz vezja.

Ta MMU omogoča pri Z8208 in Z8216 upravljanje z 16-Mzlogov spomina brez izgube hitrosti pri pretvorbi naslovov. Pri Z8108 in Z8116 omogoča 19 naslovnih linij dostop do 512 Kzlogov



Slika 4: V sistemskem in podatkovnem načinu imamo možnost takole naslavljati spomiski prostor. Spominski prostor lahko naslavljami z registri za opis strani in sicer skupaj ali ločeno programski in podatkovni del.

spomina. Za pretvorbo med logičnimi in fizičnimi naslovnimi prostori MMU uprablja dva nabora po 16 registrov za opis strani. En nabor je za sistemski način in en za uporabnikov način delovanja.

Enostavno, 16-bitni logični naslov procesorja Z800 se deli v dve polji, v 12-bitni odmik in v 4-bitni indeks (slika 3). Odmik gre dalje v fizični naslov nespremenjen, indeks pa izbere enega od registrov za opis strani. Ta indeksni register vsebuje zgornje bite fizičnega naslova in množico takoimenovanih atributov za to izbrano stran.

Vsak 16-bitni register za opis strani vsebuje 12 bitov naslovnih informacij in 4 bite za atribute.

Naslovi se prevedejo, tako da se spodnjih 12 ali 13 bitov logičnih naslovov (odvisno ali je izbran/ni-izbran programski/podatkovni način) združi z naslovnimi informacijami v ustreznem registru za opis strani spomina (slika 3). Register je izbran glede na najbolj pomembne bite v logičnem naslovu.

Atributni biti nadzorujejo dostop in zagotavljajo statusne informacije za vsako spominsko stran. Imamo sledeče atribute: "Valid bit", ki pove ali je opis strani veljaven za uporabo; "Write Protect bit" zščiti spominsko stran, da je z nje mogoče samo čitati; "Modified bit" kaže, če je bila stran v spominu vpisana; in "Cachable bit", ki pove, ali se ta spominska stran lahko naloži v "cache" spomin. Kot je prikazano, vsebuje MMU dva nabora registrov za opis spominskih strani z ločenimi zaznamki, ki jih omogočajo, en nabor registrov je za sistemske naslove, drugi pa za uporabnikove naslove. Izbrani nabor je odvisen od postavitve zaznamka za sistemski/uporabniški način v glavnem statusnem registu. Sistemski in uporabnikovi programi so lahko neodvisno preslikani ali nepreslikani, ali pa so preslikani v različna področja fizičnega spomina. Poleg tega lahko omogočimo, ločeno za vsak način, tudi delitev na programski in podatkovni del. Če je delitev omogočena, se trenutna množica registrov deli na polovico in je polovica registrov dostopnih programskemu delu, druga polovica pa za podatkovni del. V tem primeru so samo trije biti logičnega naslova uporabljeni za izbiranje strani. Spodnjih 13 bitov logičnega naslova preide MMU brez spremembe.

Z8108 ima 512K zlogov fizičnega naslovnega prostora. Fizični naslov - 19-bitni je zgrajen iz 12 ali 13 bitov logičnega naslova in iz 6 ali 7 bitov, ki jih da register za opis strani. To se pretvori v 128 strani po 4K zlogov, če sta programski in podatkovni del združena ali v 64 strani po 8K zlogov pri ločenih naslovnih prostorih za program in podatke.

Procesor ima mehanizem, s katerim sistemski programi lahko pridejo do tabele za preslikavo v uporabnikovem načinu. Z uporabo LDUD (Load in User Data Space) in LDUP (Load in User Program Space) ukazov , lahko sistemske rutine uporabljajo parametre iz uporabnikovih programov ali vrnejo vrednost uporabnikovi podatkovni strukturi.

Uporaba lastnosti MMU-ja je relativno enostavna. Ker je MMU del vezja, ni potrebna nobena zunanja logika; vezje ima torej velik naslovni prostor za zunanji svet. Enostavni Z80 programi, ki delujejo na Z800, ne skrbijo za delovanije MMU-ja. Ko priključimo napajanje, se MMU postavi v "pass-through" način, kar pomeni da so logični naslovi spuščeni direktno na naslovne linje brez transformacije.

Programi, ki so pisani posebej za Z800, ali pa programi za Z80, ki želijo izkoriščati večji naslovni prostor, pa uporabijo prednosti, ki

jih ponuja MMU enota na različne načine. Prva tehnika je ločevanje aplikacijskih programov od operacijskega sistema. Tako se oba programa (aplikacijski, ki deluje v uporabniškem načinu in operacijski sistem, ki deluje v sistemskem načinu) nahajata v različnh področjih fizičnega spomina, če uporabljata različne skupine registrov za preslikavo spomina. Drugič, MMU lahko ločeno preslika programski del in podatkovni del in dovoljuje do 64K zlogov programske kode in do 64K zlogov podatkovne kode.

Če te tehnike ne omogočajo dovolj spominskega prostora, imamo na voljo tehniko, ki preklaplja spominska področja - "Bank-switching" (slika 4). V taki shemi je program (ali pa podatki) razdeljen v sekcije po 64K zlogov. Dokler program deluje znotraj območja 64K zlogov, je uporabljeno normalno naslavljanje. Ko potrebujemo podatek iz drugega območja, moramo najprej izvesti klic v operacijski sistem (z uporabo SC ukaza), ki spremeni vsebino registrov za opis strani. Lahko spremenimo samo eno stran ali pa celoten naslovni prostor (64K zlogov).

Še ena tehnika, ki uporablja prednosti MMU pri Z800, pa je "navidezen disk". Ta uporablja veliko območje spomina (običajno 256K zlogov ali več) za simulacijo celega ali dela diska. Kadarkoli bi morali čitati blok z diska v del spomina, sedaj enostavno preslikamo ta del spomina na ustrezen del področja "navideznega diska". Če ta spomin - "navidezen disk" - vsebuje celo datoteko, lahko vse dostope naredimo v spominu, namesto da pristopamo disku, s tem izločimo pristopni čas do diska, ki pa je bistveno daljši kot dostop do spomina.

Lahko zaključimo, da sedaj programi lahko delajo na velikih podatkovnih bazah v spominu brez uporabe začasnih datotek na disku. Programe, daljše kot 64K zlogov, izvajamo z uporabo MMU-ja tako, da preslikamo različne dele fizičnega spomina v logični naslovni prostor. Sodelujoči programi, ki se izvajajo v sistemih z več hkratnimi posli, imajo lahko skupne dele podatkovnega spomina, poleg tega pa ima vsak del spomina, ki ni dostopen drugim programom. Vse te aplikacije kažejo, kako se poenostavi sistem in poveča fleksibilnost, če je MMU del vezja.

6. "CACHE" SPOMIN

Za izboljšanje dostopnega časa pri pogosto uporabljenih ali pri časovno kritičnih programskih delih, je na vezje vključen "cache" spomin, ki obsega 256 zlogov. "Cache" spomin je mogoče oblikovati tako, da vsebuje samo ukaze, samo podatke ali pa oboje. Ker je ta spomin na vezju, ni izgube časa pri dostopu do shranjenih podatkov.

Deluje na principu, da imajo pravkar uporabljeni ukazi ali podatki veliko verjetnost, da bodo ponovno uporabljeni - "cache" hrani zadnje uporabljene kode in tako omogoča zelo hitro izvajanje ponovljivih sekvenc. Vsakokrat, ko procesor zahteva podatek ali ukaz, najprej pregleda "cache" spomin, da vidi, če je iskan podatek tu že prisoten. Če je prisoten, ga procesor uporabi in ne izvede pristopa na zunanje vodilo. Očitno je, da se z uporabo "cache" spomina poveča hitrost, saj Z800 izvede Z80 kodo od dva do osemkrat hitreje.

Ko je konfiguriran kot "cache", je spomin organiziran v 16 vrstic s po 16-imi zlogi (tabela 2). Poleg tega ima vsaka vrstica še dve

	20 bitov	16 bitov	16 + 8 bitov
Vrstica 0	Zaznamek 0	Veljavnostni biti	Cache podatki
Vrstica 1	Zaznamek 1	Veljavnostni biti	Cache podatki
.	.	.	.
.	.	.	.
.	.	.	.
Vrstica 15	Zaznamek 15	Veljavnostni biti	Cache podatki

Tabela 2: Organizacija "Cache" spomina

polji - 20-bitni "fizični naslovni zaznamek" in 16-bitno "veljavno polje". Naslovni zaznamek se primerja z 20-imi najbolj pomembnimi biti vsakega fizičnega naslova, ki ga generirata CPU in spominska upravljalna enota. Vkolikor je primerjava uspešna na katerem od 16-ih zaznamkovnih naslovih, uporabimo spodnje štiri bite fizičnega naslova za izbor zahtevanega zloga ali besede v primerjani vrstici. "Veljavno polje" vsebuje po en "Veljavni bit" za vsak zlog v vrstici.

Če je ustrezen "Veljavi bit" za iskan zlog v primerjani vrstici postavljen, procesor uporabi zlog. Če "Veljavi bit" ni postavljen, pošlje procesor naslov v zunanji spomin, da se tam poišče podatek. Ta podatek potem procesor uporabi in se poleg tega še vpiše v "cache" spomin, kar povzroči, da se postavi "valid bit" za ta zlog vpisan v "cache". Če ni nobeden od 16-ih zaznamkovnih naslovov enak 20-bitnemu naslovu, potem procesor napolni tisto vrstico, ki najdlje ni bila uporabljena - briše vse "Veljavne bite" , da onemogoči vse zloge - in 20-bitni naslov postane nov zaznamkovni naslov. Sedaj se ustrezen zlog ali zlogi prenesejo iz zunanjega spomina.

Če ne potrebujemo "cache" spomina, onemogočimo vezje in spomin se rekonfigurira kot 256 zlogov fiksno naslovljivega RAM-a. Ta "lokalni" spomin lahko uporabljamo pri sistemih, ki vsebujejo samo ROM spomin, ali pa lahko vsebuje tiste dele programa, ki potrebujejo hitrost tega spomina, kot so na primer prekinitvene rutine. Pri fiksnem načinu naslavljanja, zaznamkovni naslov identificira posamezne vrstice, postavitve "Veljavnih bitov" pa nimajo nobenega pomena. Zaznamkovne naslove postavi programer in ostanejo nespremenjeni, kar omogoča pristop k temu spominu.

7. PREKINITVE

Poleg treh prekinitvenih načinov, ki jih ima mikroprocesor Z80, ima družina Z800 še dodatni, četrti način prekinitev, ki ga uporabljajo tudi notranje prekinitve (traps).

Pri tem novem prekinitvenem načinu uporabljamo register, ki kaže v tabelo vektorjev na polje novih statusnih vrednosti. Za notranje prekinitve in nevektorske prekinitve je vsako polje sestavljeno iz nove vrednosti za programski števec in nove vrednosti za glavni statusni register. Pri vektorskih prekinitvah pa polje novih vrednosti obsega samo novo vrednost programskega števca.

Pri vektorski prekinitvi se stara vrednost programskega števca in stara vrednost glavnega statusnega registra shrani na sistemski sklad, prekinitveni vektor pa se prebere iz prekinitvene naprave. Ta vrednost se potem shrani na sistemski sklad in uporabi za iskanje

nove vsebine programskega števca iz notranje prekinitvene tabele. Tak način omogoča posluževanje prekinitvene rutine na katerem koli mestu v spominu in dovoljuje tudi gnezdenje prekinitev, ker je predhodno stanje shranjeno na sklad in ne samo v začasni zaznamkovni regiser, kot je to pri Z80.

Procesor omogoča tako maskirane kot tudi nemaskirane prekinitve. Maskirane prekinitve so omogočene z enim od bitov v glavnem statusnem registru in procesor jih sprejme samo, če je ta bit postavljen. Nemaskirane prekinitve ne moremo onemogočiti, procesor jih vedno sprejme. Procesor pregleduje stanje zunanjih prekinitev po koncu vsake instrukcije in izvede servisno rutino pred izvajanjem naslednje programske instrukcije. Maskirane prekinitve so lahko vektorske ali nevektorske. Če so vektorske, jih procesor izvaja tako, kot smo sedaj opisali, če pa so prekinitve nevektorske (v prekinitvenem načinu 3), se uporabi posebno nevektorska prekinitvena tabela, ki določi servisno rutino.

8. PERIFERIJA NA VEZJU

S periferijo, ki je že na vezju, potrebuje mikroprocesor Z800 minimalno število komponent in povezav za izgradnjo sistema. Štirje DMA kanali pri Z8208 in Z8216 omogočajo neodvisen, hiter prenos podatkov; serijska vrata omogočajo popolni-dvosmerni asinhroni način prenosa, ki deluje s hitrostjo do 2M Hz/s pri časovnem signalu 10M Hz-ov. Vsak DMA kanal je lahko programiran za prenos podatkov iz spomina v spomin, iz spomina v V/I napravo (ali obratno) ali pa iz ene V/I naprave v drugo V/I napravo. Poleg tega podatke lahko prenašamo v teh načinih : prenos posameznega zloga, blokovni prenos ali kontinuiran prenos.

Pri posameznem prenosu DMA sekcija realizira vodilo med CPU-jem in še enim DMA kanalom za vsak prenos zloga ali besede. Blokovni način dovoli DMA sekciji prenos podatkov tako dolgo, dokler jih je periferija pripravljena sprejemati. Kontinuirani način dovoljuje DMA vezju prenos celega bloka brez sprostitve vodila. Poleg tega vsak kanal kontrolerja lahko deluje v načinu "ni prenosa" in se obnaša kot števec.

Vsak DMA kanal je sestavljen iz 24-bitnega izvornega naslovnega registra, 24-bitnega ciljnega naslovnega registra, 16-bitnega števnega registra in 16-bitnega registra za opis prenosa. Vsi ti registri so v V/I prostoru CPU-ja in so dostopni z I/O ukazi preko internega CPU-jevega vodila. DMA kanali uporabljajo naslovne, podatkovne in kontrolne linje procesorja za prenos podatkov izven procesorja. Vsak kanal ima vhod preko katerega lahko zunanja naprava zahteva prenos.

Vse štiri DMA kanale nadzoruje "glavi DMA kontrolni register", ki lahko ukaže kanalom, da se povežejo med sabo ali na serijski V/I kanal. Ko so DMA kanali povezani, se eden obnaša kot služabnik, ki nalaga glavni DMA kanal z novim naslovom, številom in opisnimi informacijami. Glavni kanal prenaša bloke podatkov k ciljnemu kanalu in potem čaka, dokler podrejeni kanal ne poveča svojih registrov glede na informacije, prenešene iz spomina. S to strukturo je prenos različnih tipov in prenos na različne lokacije mogoč brez posredovanja CPU-ja.

Poleg tega ima procesor na vezju še štiri števce/časovnike, samo Z8208 in Z8216 pa imata linije za tri števce/časovnike povezane navzven; četrti pa je samo notranji na vseh štirih verzijah. Trije zunanje dostopni števci/časovniki so popolni 16-bitni odštevalniki in so neodvisno programabilni za štetje zunanjih dogodkov (števni način) ali notranjih časovnih signalov (časovni način). Dva 16-bitna števca lahko povežemo tako, da formiramo 32-bitni števec.

Pri uporabi vsak števec naložimo z začetno vrednostjo, ki je poleg tega shranjena v register za časovno konstanto, ki ga tudi ima vsak števec. Ko vrednost števca doseže nič, števec povzroči enega od naslednjih dveh dogodkov: generira prekinitev ali pa se števec naloži z vrednostjo registra za časovno konstanto in ponovno prične odštevanje. Eden od komandnih bitov specificira, kateri od teh dveh načinov je uporabljen. Poleg tega je lahko vsak števec prožen z zunanjim signalom ali pa programsko.

Serijska komunikacija običajno zahteva pomoč enega od števcev, ki služi za generator prenosne hitrosti, ali pa je potreben zunanji izvor časovnega signala. Serijska komunikacija lahko pošilja ali sprejema podatke istočasno, potrebno je le povezati dva DMA kanala z oddajno in sprejemno sekcijo in s tem dobimo zelo hiter prenos. Kot večina univerzalnih asinhronih sprejemnikov-oddajnikov tudi ta komunikacija poteka s pošiljanjem podatkov v običajni obliki: startni bit, pet do osem podatkovnih bitov, paritetni bit in eden ali dva stop bita.

Ta serijska komunikacija se lahko uporablja tudi za nalaganje podatkov ali prenos programov, če naprava s procesorjem Z800 služi kot služabnik večjemu, nadrejenemu računalniku. Ta sposobnost prenašanja-nalaganja deluje v zvezi s samonalagalnim načinom, ki se izbere ob priključitvi napajanja. Ko je izbran ta način, se DMA kanal avtomatično poveže s sprejemno sekcijo serijske komunikacije, programira se vnaprej predvideni ciljni naslov (000000H) v DMA kanal, postavi se format serijske komunikacije in prične se nalagati 256 zlogov podatkov v spomin preko serijskega kanala. To omogoča procesorju Z800, da služi kot služabnik (brez ROM-a) glavnemu sistemu, in ta mu določa celotno obnašanje.

9. MULTIPROCESORSKI SISTEMI

Poleg tega, da lahko služi kot služabnik, lahko Z800 deluje tudi v multiprocesorskih sistemih. Le Z8208 in Z8216 imata na vezju vgrajene sestavine, ki dopuščajo enostavno vključitev v multiprocesorske sisteme.

Zgradimo lahko sistem, ki vsebuje dva ali več procesorjev, vsak z lokalnim vodilom, ki povezuje lokalni spomin in lokalne V/I enote, ter komunicira z glavnim sistemskim vodilom preko spominskega bloka. Taka zgradba sistema zahteva uporabo dodatne logike, ki skrbi za prireditev glavnega sistemskega vodila in s tem sistemskih enot (spominskih, I/O, ...) posameznemu procesorju.

V to shemo bi bil lahko vključen tudi glavni procesor za kontrolo sistemskega vodila in za dodeljevanje nalog služabniškim procesorjem Z800.

10. GRADNJA APLIKACIJ

Zaradi minimizacije porabljenega prostora je najboljša rešitev Z8216. Vsebuje mnogo funkcij, ki jih potrebuje snovalec mikroračunalniškega

sistema na CPU plošči. Potrebno je dodati le vmesniško logiko in vmesnike za sistemska vodila, kot so IEEE-696 ali IEEE-796.

Za rokovanje s prekinitvami in za paralelno komunikacijo (za printer) lahko dodamo procesorju dva Z8036 števca/časovnika in vezje za paralelne V/I linije. Za dodatne serijske V/I linije pa lahko na vodilo dodamo Z8030 dvokanalni kontroler za serijsko komunikacijo. S tem smo zgradili soliden mikroračunalnik.

Posebni statusni in kontrolni signali, ki so dostopni, poenostavljajo zunanjo logiko za generiranje kontrolnih signalov za vodilo in vmesnike. Prvih 10 statusnih izhodov je uporabljenih v sistemih, ki nimajo dodatne procesne enote.

11. ZAKLJUCEK

Ta družina procesorjev Z800 bo nadomestila enega najbolj razširjenih procesorjev - Z80. Pomembno je, da se bodo lahko vsi programi, napisani za Z80, lahko izvajali brez spremembe. Po priključitvi na napajanje ali po "resetu", se Z800 obnaša kot Z80. To pomeni, da je MMU onemogočen, zaznamek za sistemski/uporabiški način je postavljen za sistemski način, omogočen je sistemski kazalec sklada, register za V/I je brisan in izbran je prekinitveni način 0. Vsi ukazi procesorja Z80 delujejo identično na Z800, le da so dva do osemkrat hitrejši.

Literatura:

1. On-chip memory managemant comes to 8-bit-uP. Electronic Design, October 14, 1982.
2. 8- and 16-bit procesor family keeps pace with fast RAMs. Electronic Design, April 28, 1983.
3. "Super Z80". Elektronik 13, 1.7.83.
4. Zilogs Z800. MC no.2/84

OPTIČNA KONTROLA PLOŠČ TISKANEGA VEZJA

SAŠA PREŠEREN

UDK: 681.3:681.52

INSTITUT JOŽEF STEFAN, LJUBLJANA
ODSEK ZA RAČUNALNIŠTVO IN INFORMATIKO

Avtomatska optična kontrola plošč tiskanega vezja je ena najpomembnejših aplikacij metode razpoznavanja vzorcev, saj omogoča kontrolo kvalitete plošč in boljšo produktivnost na področju avtomatike, elektronike, računalništva in komunikacij. V članku si bomo ogledali osnovne principe delovanja optičnih senzorjev in njihovo vključitev v sistem optične kontrole. Podali bomo metode vizualne kontrole tiskanin, ocenili prednosti posamezne metode in strnili razmišljanje z ugotovitvijo o perspektivah, ki jih ima optična kontrola v industrijskih aplikacijah.

VISUAL INSPECTION OF PRINTED CICRUIT BOARDS: Automatic visual inspection of printed circuit boards has been one of key issuses in the application of pattern recognition because it enables reliable inspection and better productivity. This paper discusses the basic principles of optical sensors and their integration in a system for visual inspection. A broad spectrum of methods for visual inspection of PCB is presented, evaluation of each method is given, and a future of visual inspection systems in industrial applications is discussed.

1. UVOD

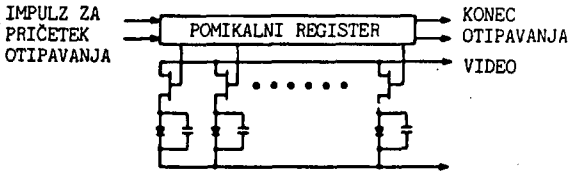
Kontrola plošč tiskanega vezja zajema odkrivanje naslednjih napak: kratki stiki, prekinitve prevodnih povezav, manjkajoče izvrtine, napačne označbe, nazobčanost ali deformacija prevodnih linij in druge lepotne napake. Najpogosteje še vedno opravljajo optično kontrolo dobro izkušeni delavci, ki odkrijejo večje napake. Majhne napake, katerih dimenzije so manjše kot 2 mm, pa je težko odkriti brez avtomatskega sistema za optično kontrolo.

Vsi algoritmi za optično kontrolo plošč tiskanega vezja uporabljajo predhodno znanje (PRE82). To znanje omogoča razvoj strategije in določitev standardov za proces kontrole. Specifikacija različnih metod kontrole je odvisna od znanja o plošči tiskanega vezja, kot je na primer geometrijska struktura in dimenzije, ali pa tudi različni predpisi o tolerancah napak.

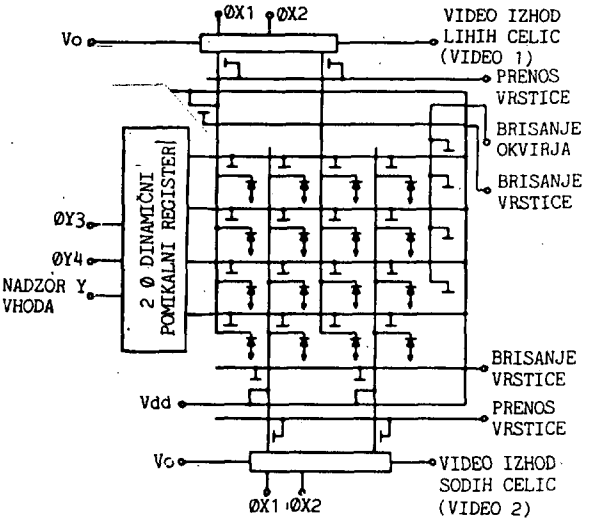
2. OKO SISTEMA

Primarna komponenta pri sistemih za optično kontrolo je pretvornik (senzor), ki pretvori svetlobni signal v električnega. Fotodioda je najboljši razpoložljiv senzor za industrijsko uporabo. Na tržišču poznamo linearne ali enodimenzionalne optične senzorje (Slika 1.) ter matrične ali dvodimenzionalne optične senzorje (Slika 2.). Vsi ti optični senzorji so sestavljeni iz vrste ali ravnine med seboj povezanih svetlobnih pretvornikov, ki jih imenujemo piksli. Tako ima lahko linearni optični senzor od 128 do 1064 zaporednih celic ali piksov, matrični optični senzorji pa imajo razpon od 12x12 piksov do 512x256 piksov ali tudi več. Ustrezni optični senzor izberemo glede na potrebe natančnosti pri optični kontroli.

Občutljivost in spektralno območje za te senzorje je ekvivalentno visokokvalitetni posamezni silikonski fotodiodi, ki jo ponavadi uporabljamo v spektralnem območju 400 do 1100 nanometrov. To pokriva celoten vidni spekter svetlobe in še del infrardečega dela spektra ter so zato ti senzorji primerni za industrijsko kontrolo.



Slika 1.: Funkcionalna slika linearnega optičnega senzorja.



Slika 2.: Funkcionalni prikaz matričnega optičnega senzorja.

Celotna zgradba sistema za optično kontrolo je prikazana na Sliki 3. Vse logične in aditivne operacije so prepuščene posebnemu video procesorju. Prav tako je video procesor tisti, ki s hitro strojno opremo izvaja izločanje značilnosti. Če je sistem za optično kontrolo tak, da omogoča več stopenj sivine ali celo barvno sliko, potem je seveda video procesor tisti ključni element video sistema, ki omogoča to vrsto obdelave. Ker je struktura video procesorja za izločanje značilnosti največkrat modularna, lahko dodatne izločevalce značilnosti brez težave dodamo. Moduli, ki so povezani z nadaljnimi stopnjami obdelave, kot na primer izločevalec polarnega profila, lahko uporabljajo rezultate modulov iz predhodnjih stopenj. Rezultate shranimo v pomnilniku med otipavanjem nove slike in jih mikroračunalnik kasneje naprej obdela.

Informacije o poziciji, orientaciji in geometrijskih značilnostih psevdo ravninskih industrijskih predmetov je mogoče dobiti z dokaj enostavnimi in direktnimi procedurami, ki vsebujejo le logične operacije in akumulacijo signalov iz senzorja ter vmesnih rezultatov. Momente ničtega, prvega in drugega reda lahko zbiramo pri vsakem pikslu posebej po x in y koordinatah. Te vsote so zbrane že ob koncu prenosa senzorskih signalov v pomnilnik in jih lahko kasneje v mikroračunalniku obdelamo med novim otipavanjem slike, ter izločimo površino, težišče, naklon glavnih osi vztrajnosti in vrednost glavnega momenta vztrajnosti. Ker so te količine popolnoma aditivne, je jasno, da ta vrsta postopkov obdelave slik ni odvisna od vrstnega reda obdelave informacij posameznih pikslov.

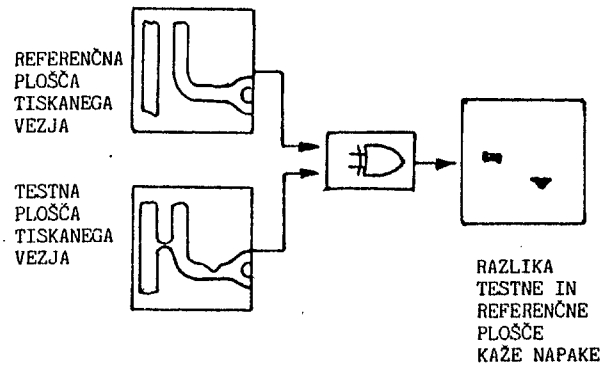
Podoben princip lahko uporabimo za izračun ostalih video sinhronih algoritmov za izločanje značilnosti. Poglejmo kakšne metode uporabljamo pri optični kontroli plošč tiskanega vezja.

3. METODE VIZUALNE KONTROLE TISKANIN

Pri optični kontroli plošč tiskanega vezja si s preprostimi operacijami računanja težišča in vztrajnostnih momentov ne moremo pomagati. Odkriti je treba napake na tiskanini in v ta namen uporabiti metode in algoritme, ki to zmorejo. V praksi so se uveljavile predvsem štiri metode: odštevanje slik, ujemanje značilnosti, dimenzijska kontrola in sintaktična analiza. Vsako od teh metod bomo opisali in razjasnili na primerih.

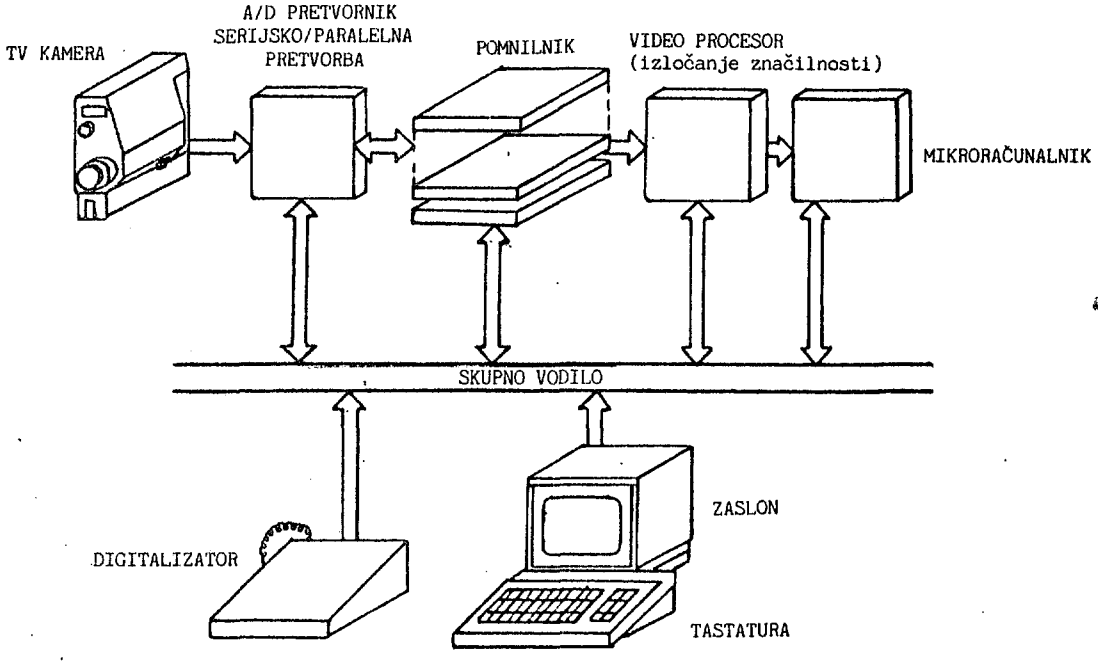
3.1. Odštevalnje slik

Odštevanje slik je najpogostejši pristop pri optični kontroli plošč tiskanega vezja. Pri tem pristopu ploščo, ki jo kontroliramo s pomočjo senzorja, digitaliziramo ter shranimo digitalizirano sliko plošče v računalniški pomnilnik. To sliko primerjamo s sliko plošče tiskanega vezja, ki nima napak ter točko po točko slike testne plošče odštevamo od slike referenčne plošče. Odšteto sliko, ki kaže napake, lahko prikažemo na ekranu in analiziramo. Slika 4. kaže odšteto sliko.



Slika 4.: Odšteta slika.

Pri vizualni kontroli z metodo odštevanja slik



Slika 3.: Zgradba sistema za optično kontrolo.

največkrat uporabljamo dve kameri, ki otipavata (skanirata) tiskanino, ki jo kontroliramo in referenčno tiskanino hkrati. Kontrolo lahko zelo pospešimo, če referenčno sliko shranimo v analogni obliki in ne v digitalni obliki.

Metoda odštevanja slik ima več pomanjkljivosti:

- potrebujemo velik pomnilnik za shranjevanje slike plošče brez napak;

- zahtevamo precizno pozicioniranje testne in referenčne tiskanine;

- metoda je občutljiva na pogoje razsvetljave in pogoje senzorja.

Dodatna pomanjkljivost pri metodi odštevanja slik je dejstvo, da se veliko in mogoče celo večina tiskanin zaradi skrčitve ali raztezanja plošč ne ujema točko po točko. V tej situaciji je nujna uporaba korekcijskih postopkov, ki kompenzirajo taka odstopanja. Tako pa lahko postanejo sistemi optične kontrole, ki temeljijo na odštevanju slik, nepraktični, to je preobsežni in zato prepočasni ali pa premalo natančni.

3.2. Ujemanje značilnosti

Ujemanje značilnosti je drugi pristop k optični kontroli tiskanin. Pri tej metodi sliko tiskanine shranimo in izločimo potrebno značilnost. Potem pa, namesto, da bi primerjali sliko testne in referenčne tiskanine direktno, primerjamo le značilnosti, ki smo jih izločili iz slike testne tiskanine z značilnostmi iz modela. Če so te značilnosti enake, je plošča dobra.

Prednosti metode ujemanja značilnosti so:

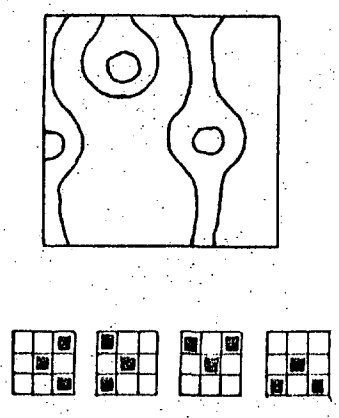
- ta strategija zelo komprimira podatke iz slike in s tem omogoča uporabo manjšega pomnilnika;

- manjša občutljivost na intenziteto vhodnih signalov;

- v nekaterih primerih (JAR80) ne zahteva preciznega pozicioniranja tiskanin;

- v nekaterih primerih (EJI73) ne zahteva referenčne slike.

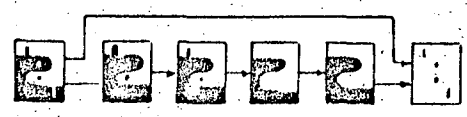
Slika 5. kaže izločene značilnosti - robove linij na tiskanini in vzorce štirih možnih konfiguracij vdolbine ali izbokline s širino enega piksla vzdolž izločenih robov.



Slika 5.: Izločene značilnosti - robovi linij na tiskanini in štiri možne konfiguracije vdolbine ali izbokline s širino enega piksla.

Jarvis (JAR80) je predlagal dvostopenjsko strategijo optične kontrole tiskanin. Metoda sestoji iz izbire skupine lokalnih binarnih vzorcev dimenzije 5x5 pikslov, ki opisujejo mejo brez napak med prevodnikom in podlago, dobljeno iz brezhibne tiskanine. Prva faza izvrši pregled vsakega mejnega vzorca iz tiskanine in ga primerja z vzorci iz pripravljene liste možnih vzorcev. Tisti vzorci, ki jih ni najti v tej listi, so prepuščeni drugi stopnji procesiranja, ki izvaja več testov, ter ugotovi, če neznani vzorec res pomeni napako. Ti dodatni testi vključujejo računajnje prevodne površine, razdalje med mejami prevodnikov, razmerje površine in dolžine prevodnikov ter drugo. Na primerih je bilo ugotovljeno, da lahko 99% možnih vzorcev dimenzije 5x5 pikslov, ki se pojavijo na brezhibni tiskanini, pripravimo z manj kot nekaj sto binarnih vzorcev, pri tem pa je le nekaj odstotkov vzorcev bilo prepuščeno drugostopenjski kontroli. Treba je poudariti, da ta metoda ne zahteva preciznega pozicioniranja testirane plošče, zahteva pa pazljivo generiranje binarnih vzorcev.

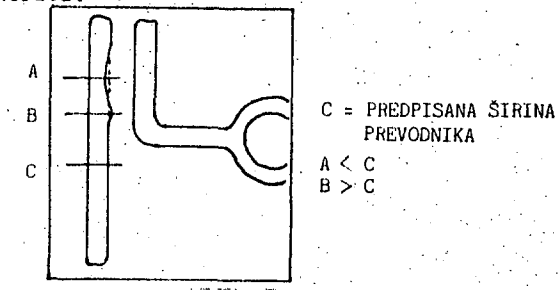
Ejiri in drugi (EJI73) so razvili tri tipe lokalnega procesiranja slik tiskanine. Napake, ki jih njihova metoda odkrije, imajo dimenzijo, ki je manjša kot je dimenzija prevodnih linij na tiskanini. Pomembno je, da ta metoda ne zahteva referenčne slike tiskanine brez napak (Slika 6). Metoda sestoji iz treh stopenj. V prvi stopnji površine prevodnikov na sliki testne tiskanine najprej povečamo v vse smeri. Ta povečani vzorec potem za enak faktor skrčimo na prvotno velikost. Ta vzorec pa primerjamo z originalnim vzorcem in z odštevanjem slik izločimo napake.



Slika 6.: Tristopenjska metoda odkrivanja napak na testni tiskanini brez primerjanja z referenčno tiskanino.

3.3. Dimenzijska kontrola

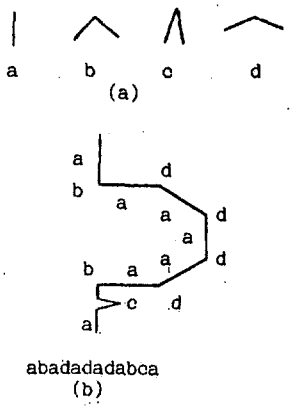
Velikokrat je izločanje značilnosti in metoda ujemanja vzorcev poenostavljena na dimenzijsko verifikacijo. Naloga sistema za optično kontrolo je za vsako meritev ugotoviti ali spada v prej postavljene standarde. Kriterij te metode lahko vključuje dolžino in širino prevodnikov, premere lukenj in skoznikov in razmike med sosednjimi prevodniki. Meritve dobimo z merjenjem razdalje med dvema prevodnima robovoma. Z drugimi besedami, razdalja med dvema robovoma je primarna značinost te metode optične kontrole. Slika 7 kaže preprosto idejo dimenzijske kontrole.



Slika 7.: Osnovna ideja dimenzijske verifikacije pri dimenzijski optični kontroli tiskanin.

3.4. Sintaktična analiza

Drugačen pristop za optično kontolo tiskanin je sintaktična analiza in razpoznavanje. Sintaktičen pristop k optični kontroli omogoča zapis velike množice kompleksnih oblik z uporabe male množice primitivnih vzorcev in strukturnih pravil. Določene slikovne vzorce, kot so tudi slike tiskanin, lahko zapišemo z nizom znakov. Tako zakodirana slika testne plošče se mora ujemati z modelnim nizom, če na testni plošči ni napak. Lokalne značilnosti lahko razdružimo v relativno majhno število različnih elementov imenovanih primitivni elementi, kot na primer koti ali linije. Določimo lahko strukturalni opis primitivnih elementov in zvezo med njimi ter tvorimo gramatiko nizov. Slika 8 ilustrira primer.



Slika 8.: Primer opisa meje prevodnika z gramatiko nizov. A) primitivni elementi gramatike; B) meja prevodnika na tiskanini in njen opis z nizom.

Če podamo niz primitivnih elementov, ki opisujejo običajne deformacije, lahko uporabimo avtomatski postopek za njihovo ločiranje s tem, da preiščemo niz, ki opisuje testirano tiskanino in iščemo vse pojave vzorca oz. primitivnega niza, ki opisuje napako.

4. ZAKLJUČEK

Nadaljne izboljšave na področju robotike in avtomatizacije so tesno povezane z razvojem procesiranja slik in razpoznavanja vzorcev. Vizualna percepcija omogoča robotom, da se obnašajo v skladu s specifično situacijo. Ta sposobnost ne samo poveča fleksibilnost robotov, ampak tudi poenostavi podajanje komponent in pritrjevanje. Še več, vizualno povratno zanko lahko uporabimo za avtomatsko sestavljanje, kot sposobno orodje, ki omogoča uporabo robota za večje število opravil. To je predvsem posledica globalnega vidika vizualne informacije, ki opisuje kompletno sceno, ne pa le eno ali nekaj točk.

Istočasno je jasno, da ogromna gostota podatkov, ki je združena v video signalu zahteva intenzivno procesiranje. Uporaba splošnega računalnika brez hitrega predprocesorja, ki zmanjša količino informacij, bi povečala čas obdelave na nekaj sekund ali več, celo za relativno enostavne algoritme, ki procesirajo slike. Zaradi omejitev stabilnosti sistema je praktična zgornja meja zakasnitev nekaj sto milisekund za vizualni modul ali za robotski sistem.

Da bi dosegli te zahteve glede hitrosti, tečejo raziskave v smeri novih arhitektur procesorjev tako, da bi procesirali slikovno informacijo paralelno. Teoretična omejitev za izvajalno hitrost je določena s ponavljalno frekvenco video senzorja. Tej omejitvi se bo mod približati, če bo možno celo sliko procesirati med tipanjem slike (video scan-om), to je sinhrona video obdelava. Ta pristop narekuje specifične omejitve na vrsto algoritmov za procesiranje slik, kot je to na primer vrstni red pikslov.

Sistemi za vizualno kontrolo se množično uveljavljajo v industrijskih aplikacijah v razvitem svetu. Ti sistemi ne samo osvobajajo človeka od izjemno napornega in monotonega dela, ampak tudi zagotavljajo boljšo kvaliteto proizvodov in s tem konkurenčnost, ki je pogoj za ohranitev industrije.

5. LITERATURA

(EJI73) M. Ejiri et. al.: A process for detecting defects in complicated patterns, Comput. Graphics Image Processing, vol.2., 1973.

(JAR80) J.F. Jarvis: A method for automating the visula inspection of printed wiring boards, IEEE Trans. Pattern Anal. Machine Intell., vol.PAMI-2,pp.77-82, jan. 1980.

(LON81) A.H. Longford: Intelligent vision for industrial control, Sensor Review, januar 1981.

(MCI83) W.E. McIntosh: Automating the Inspection of Printed Circuit Boards, Robotics Today, junuj 1983.

(PRE82) Saša Prešern: Object recognition in robotics, Second world conference on mathematics at the service of man, Las Palmas, 1982.

(VAN82) L. Vanderheydt et al.: Application of pattern recognition and image processing, Sensor Review, julij 1982.

(VIL83) P. Villers, Recent Proliferation of industrial artificial vision applications, 13th Symposium on Industrila Robots 83, 1983.

IBMOVSKI OSEBNI RAČUNALNIK PETRA I.

ANTON P. ŽELEZNIKAR

UDK: 681.3.06 Petra

Članek opisuje materialno in programsko zgradbo ibmovskega osebnega računalnika Petra. Projekt Petra se izvaja sproti (med pisanjem člankov o njemu) in je namenjen naprednim razvijalcem in amaterjem. V okviru projekta bo obdelana materialna (logična) zgradba računalnika, kritične signalne slike, nekateri osnovni programi za preizkušanje posameznih delov računalnika in njihovo oživljanje, zgradba osnovnih V/I sistemov (BIOS) za uporabljene operacijske sisteme (CP/M-86 in PC-DOS) in kratek opis njegovih operacijskih sistemov. Prvi del prinaša splošen opis materialne zgradbe računalnika Petra in prvi del opisa logičnih shem.

Petra - An IBM-like Personal Computer I

This article deals with hardware and software of IBM-like personal computer, named Petra. Petra Project is runing simultaneously with writing this and the following articles about it and is dedicated to advanced developers and to amateurs (computer hams). In the project framework the computer hardware (logical circuits), critical signal paths, some basic programs for testing of particular computer parts and for its revival, structure of basic I/O systems (BIOS) for used operating systems (CP/M-86 and PC-DOS) and a short description of computers operating systems will be presented. The first part deals with general scheme of Petra computer hardware and describes the first part of computer logical circuits.

1. Uvod

V zaporedju člankov bomo opisali gradnjo amaterskega osebnega računalnika Petra, ki bo lahko izvajal programe IBMovih osebnih računalnikov (IBM PC in IBM PC-XT). Za ta računalnik bomo razvili in opisali njegovo materialno in programsko opremo (z izjemo operacijskih sistemov, ki se bodo lahko na njem izvajali, in sicer IBM PC-DOS, CP/M-86, CP/M-Concurrent in MS-DOS). Programska oprema bo zajemala opis začetnega nalagalnega dela in BIOS pakete (Basic I/O System) za različne operacijske sisteme.

Ta projekt časopisa Informatica je namenjen prvenstveno vzgoji računalniških razvojnih strokovnjakov v naših podjetjih in seveda že dovolj usposobljenim amaterjem. Projekt je dovolj težaven, obsežen, napreden in skupinski, obravnava pa načrtovanje ibmovskega osebnega računalnika, njegovo laboratorijsko in postopno oživljanje, metodološke delovne načine, programske zamisli in seveda značilno problematičnost konkretnega projekta. Takšen projekt mora biti ob razvijalski sposobnosti podprt tudi z aparaturnim in programskim orodjem, kot so osciloskop, logični analizator (po potrebi), zbirnik, obratni zbirnik, prevajalniki, rutine oživljanja z značilnimi signali itn.

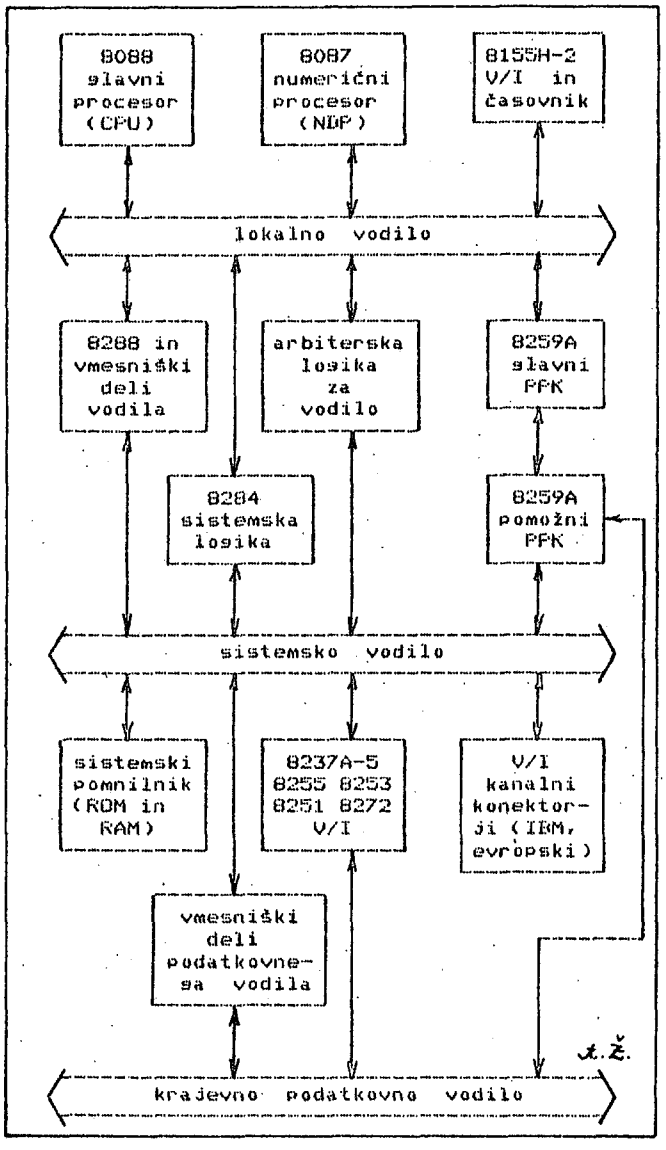
Zaradi pomanjkanja raznovrstnega orodja bo osebni računalnik Petra pomagal razvijati tudi samega sebe (imel bo samorealizacijsko komponento). Način oživljanja Petre se bo izvajal tipično paralelno. Tudi Petrin starejši brat Iskra Delta Partner bo pomagal pri razvoju programov (prečna programska oprema), pri izdelavi dokumentacije in tudi pri pisanju zaporedja člankov o Petri.

Snovalci projekta Petra bodo seveda uporabljali izkušnje iz svojih prejšnjih uspešnih projektov, objavljenih v časopisu Informatica ([1, 2, 3, 4, 5, 6, 7]). Dokaj obsežen in zapleten projekt Petra pa ne bo predmet administrativne raziskovalne naloge niti podeželsko zadrţanega, okostenelega okolja: bo izziv snovalcev za njih same in za tiste posnemovalce, ki želijo tehnološko tekmovati in sodelovati v težavnih razmerah rasti, razvoja in oblikovanja določene sposobnosti na mikroračunalniškem področju. Projekt Petra je namenjen naprednim razvijalcem s ciljem, da se enakomerno obvladajo problemi njegove materialne in programske opreme. Projekt Petra naj bi bil poživitev, ki je v razdobju pomanjkanja integriranih komponent in sodobnejše sistemske in aplikativne mikroračunalniške programske opreme še kako zaželena, prijetna in osvežujoča.

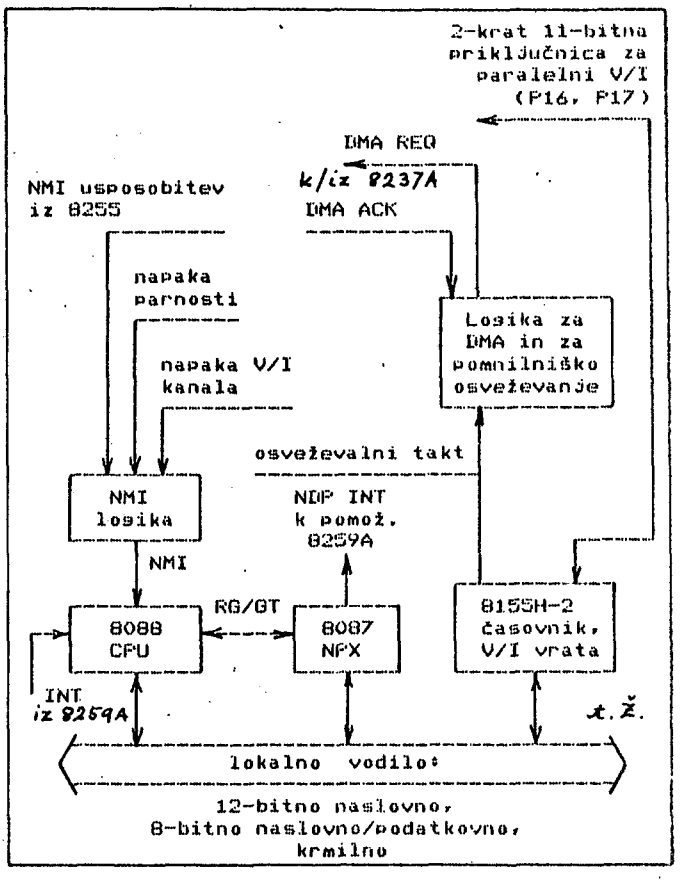
Sistem Petra, ki sa bomo razvijali, je ibmovski pri določenih pogojih, kar pomeni, da se na njemu pri določeni konfiguraciji (ustrezna tipkovnica in vtična video plošča) lahko izva-

jajo programi, napisani za IBM PC. Sistem Petra bo odprt, modularen sistem. Na posebno vodilo, ki je notranje vodilo računalnika IBM PC, se bodo lahko zatikale (dodajale) dodatne plošče, tj. vse tiskane plošče, ki se izdelujejo kot dodatki (pomnilnik, ura realnega časa, V/I kanali, barvna grafika, zvočni vmesniki, A/D in D/A pretvorniki, računalni pospeševalniki, krmilniki za trdni disk itd.) za računalnik IBM PC ali IBM PC-XT. Razen tega bomo lahko zatikali na to vodilo tudi domače tiskane ali ožičene plošče prek električno (mehansko) reducirane 64-polne evropske priključnice. Tako bomo lahko dodajali sistemu Petra plošče normalnega dvojnega ali enojnega evropskega formata. Takšen delovni način bo omogočal izredno raznovrstno preizkušanje domačih razvojnih dosežkov (plošč in programske opreme).

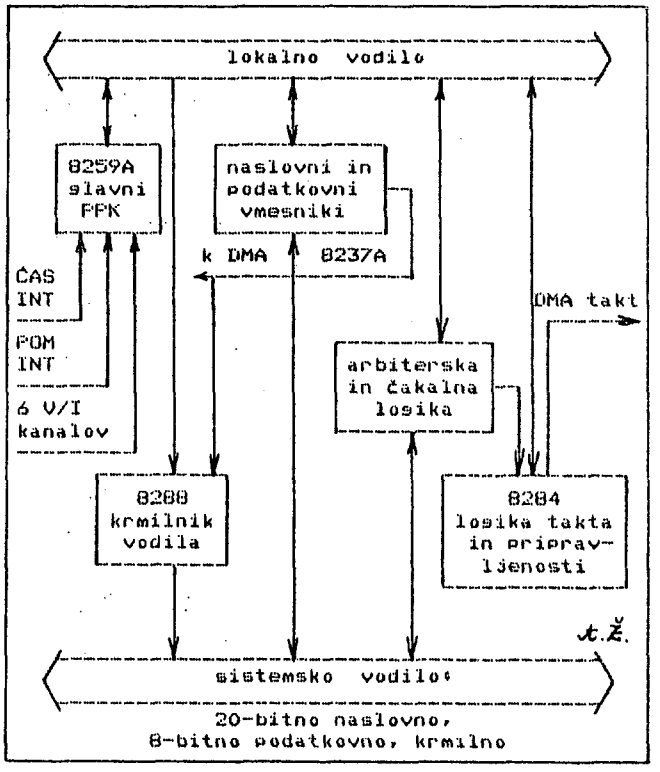
Za sistem Petra bomo imeli tudi vrsto storitvenih programov (v okviru uporabljanih operacijskih sistemov CP/M-86, PC-DOS itd.), prevajalnikov (programirni jeziki C, Logo, Basic, Ada, Pascal itd.) in aplikativnih programov (Word-



Slika 1. Poenostavljeno (grobo) vezje ibmovskega mikroračunalniškega sistema Petra z možnostjo dodajanja V/I kanalnih modulov (na evropskih in ibmovskih ploščah). PPK je okrajšava za programirani prekinitveni krmilnik.



Slika 1.1. Podrobnejši prikaz logike diagrama iz slike 1 nad lokalnim vodilom

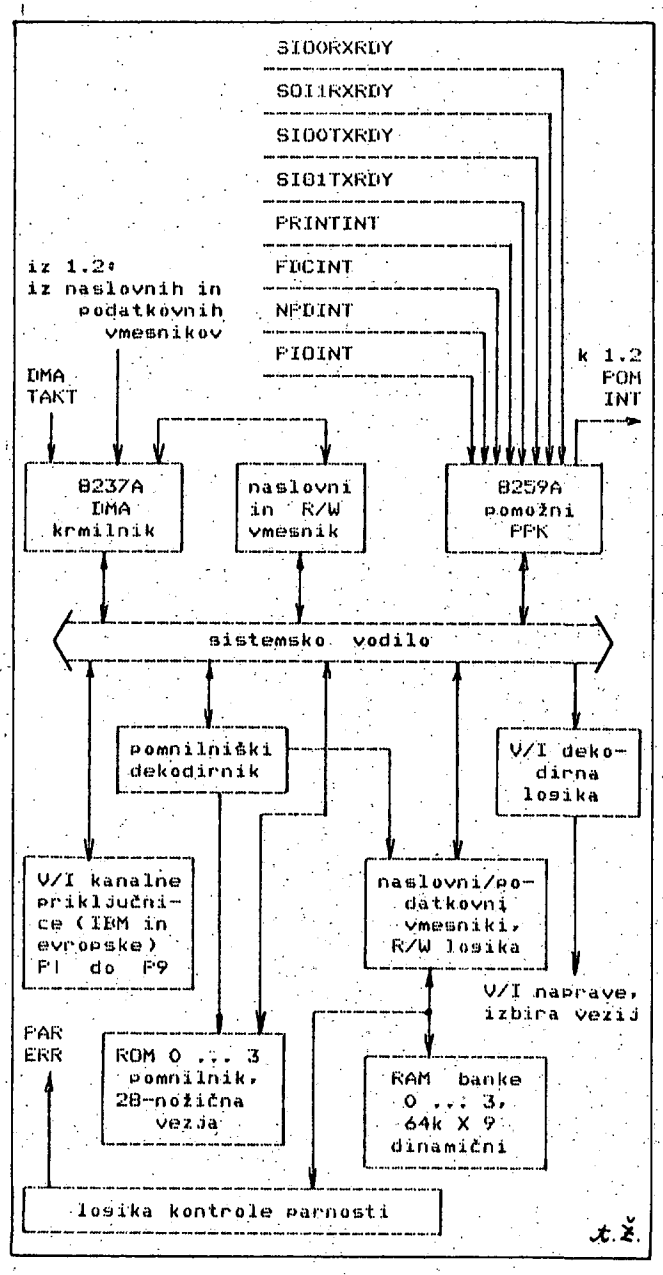


Slika 1.2. Podrobnejši prikaz logike diagrama iz slike 1 med lokalnim in sistemskim vodilom

star, Lotus itd.). Sčasoma se bo za ta sistem nabralo obilo programske opreme, tudi take, ki se uporablja na konstrukcijskih in proizvodnih področjih (CAD, CAM itd.). Po tem obetajočem uvodu nam ne preostane kaj drugega, kot da se resno in z vso močjo posvetimo uresničenju projekta Petra.

2. Pregled projekta Petra

Projekt Petra bo upošteval razvojne izkušnje domačih in tujih avtorjev ((1, 2, 3, 8, 9, 10)) na 8- in 16-bitnem mikroprocesorskem področju. Slika 1 prikazuje osnovni diagram računalnika Petra s tremi, ločenimi vodili (sistemskim, lokalnim in vhodnim/izhodnim vodilom). Na naslednjih nekaj slikah vidimo podrobnejšo razčlenitev diagrama s slike 1.



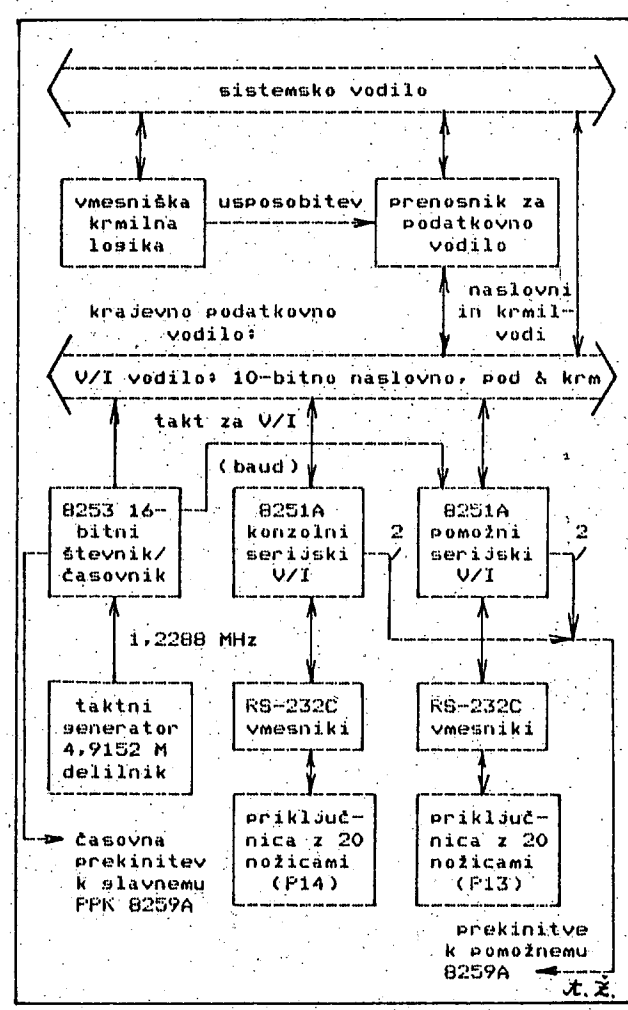
Slika 1.3. Podrobnejši prikaz logike diagrama iz slike 1 pod sistemskim vodilom (sistemski pomnilnik, V/I, DMA in pomožni PPK)

Slika 1.1 prikazuje podrobneje zgornji del (nad lokalnim vodilom) diagrama s slike 1. Ta del vezja vsebuje mikroprocesor 8088, matematični soprocesor 8087, V/I in časovniško vezje 8155 in logičnopovezovalna vezja za NMI (nemaskine prekinitve), za DMA (neposredni pomnilniški dostop) in za pomnilniško osveževanje.

Slika 1.2 vsebuje vezje med lokalnim in sistemskim vodilom diagrama s slike 1. Tu se nahaja glavni prekinitveni krmilnik 8259, krmilnik za sistemsko vodilo 8288, naslovni in podatkovni vmesniki DMA krmilnika, arbiterska in čakalna logika (vodilo) in taktno vezje 8284.

Slika 1.3 kaže vezje pod sistemskim vodilom diagrama s slike 1. Tu imamo nekatera ključna vezja, kot so DMA krmilnik 8237, pomožni prekinitveni krmilnik 8259, naslovni in bralno/pisalni vmesnik (med DMA krmilnikom in sistemskim vodilom), pomnilniški dekodirnik za ROM in RAM, kontrola parnosti za pomnilnna vezja, V/I dekodirna logika in pomnilniški vmesniki. Na tej sliki so narisane tudi vtične priključnice za ibmovske in evropske vtične plošče (dodatne module). Te plošče se tako priključujejo na sistemsko vodilo!

Slika 1.4 opisuje logiko med sistemskim in krajevno podatkovnim (V/I) vodilom s slike 1 in



Slika 1.4. Podrobnejši prikaz logike med sistemskim in krajevno podatkovnim vodilom s slike 1 in del V/I krmilnikov in vmesnikov, ki so povezani na V/I vodilo (krajevno podatkovno).

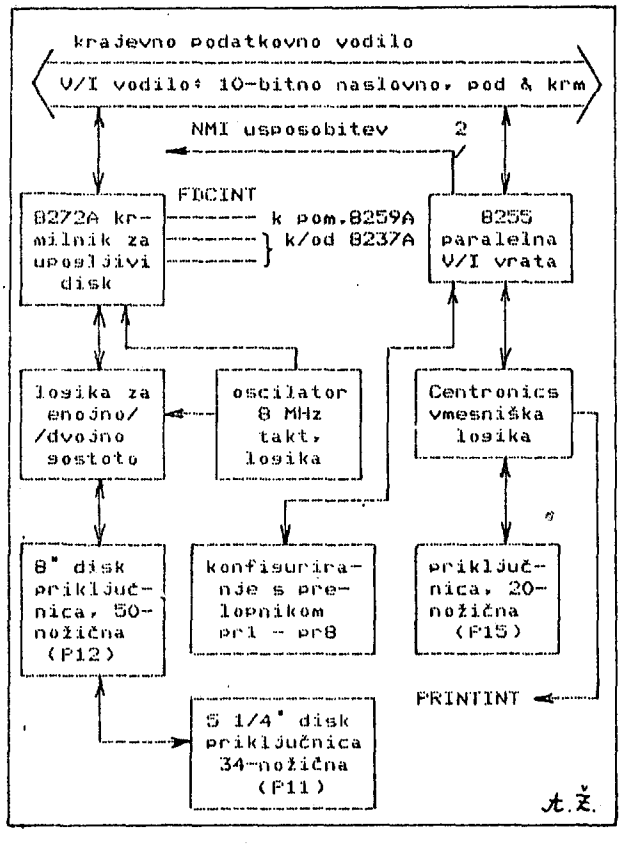
prikazuje del V/I krmilnikov (preostali del se nahaja na sliki 1.5), vmesnikov in taktnik, ki so povezani na V/I vodilo. Na tej sliki se nahajajo tudi ustrezni priključnici za serijska kanala (RS-232C).

Slika 1.5 podrobneje opisuje drugi del V/I krmilnikov, vmesnikov in taktnika, ki so povezani na V/I vodilo. Tu imamo krmilnik za upogljive diske (8272A) in njegov oscilator (8 MHz), ki prek ustrezne logike omogoča izbiro enojne ali dvojne gostote zapisa na diskete. Predvideni sta tudi priključnici za 8" in 5 1/4" disketne enote, skladno s standardom (50-nožična in 34-nožična priključnica). Prek paralelnih vrat 8255 je mogoče računalnik Petra tudi ustrezno konfigurirati (podobno kot IBM PC) z uporabo konfiguracijskega preklopnika (pr1 - pr8). Drugi paralelna vrata vezja 8255 se uporabljajo za priključitev V/I naprave tipa Centronics.

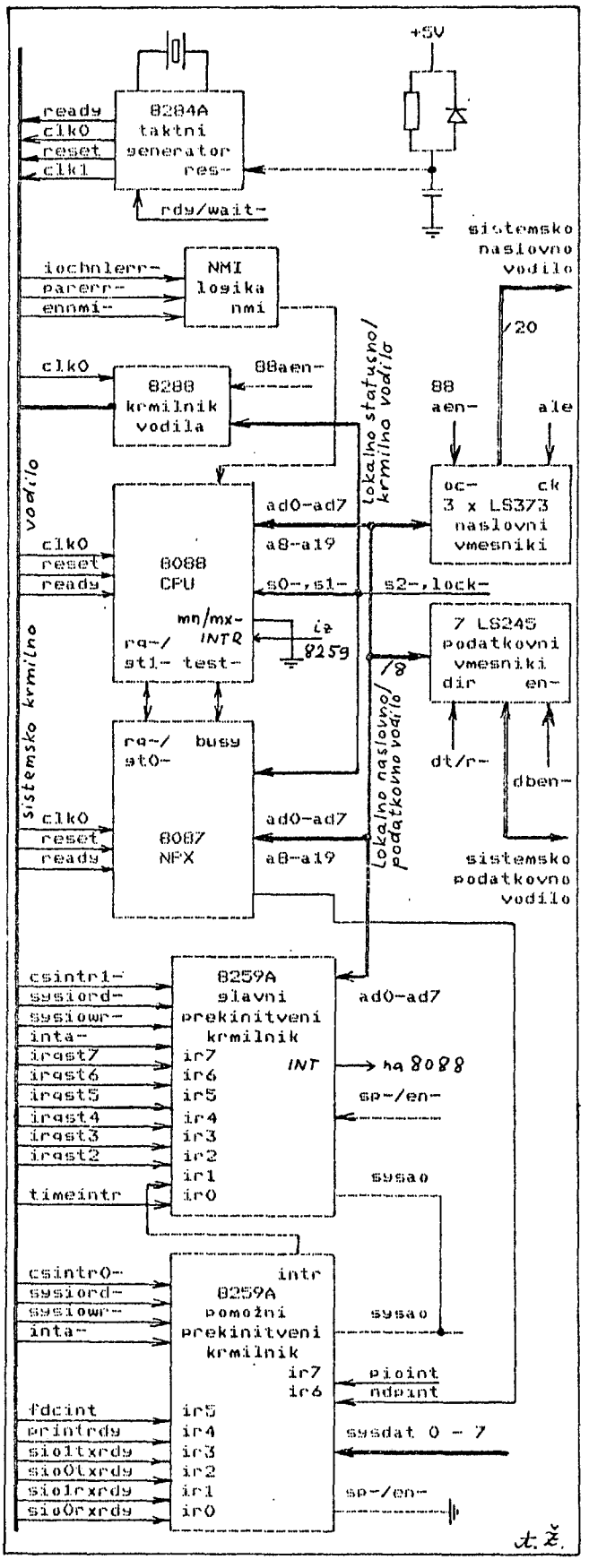
S slikami 1 in 1.1 do 1.5 je shematično opisana zgradba računalnika Petra. V naslednjem poglavju si bomo na kratko ogledali podrobno logično shemo prvega dela računalnika Petra z vsemi integriranimi vezji, upori, kondenzatorji, diodami, vodili itd.

3. Logična shema na listu 1

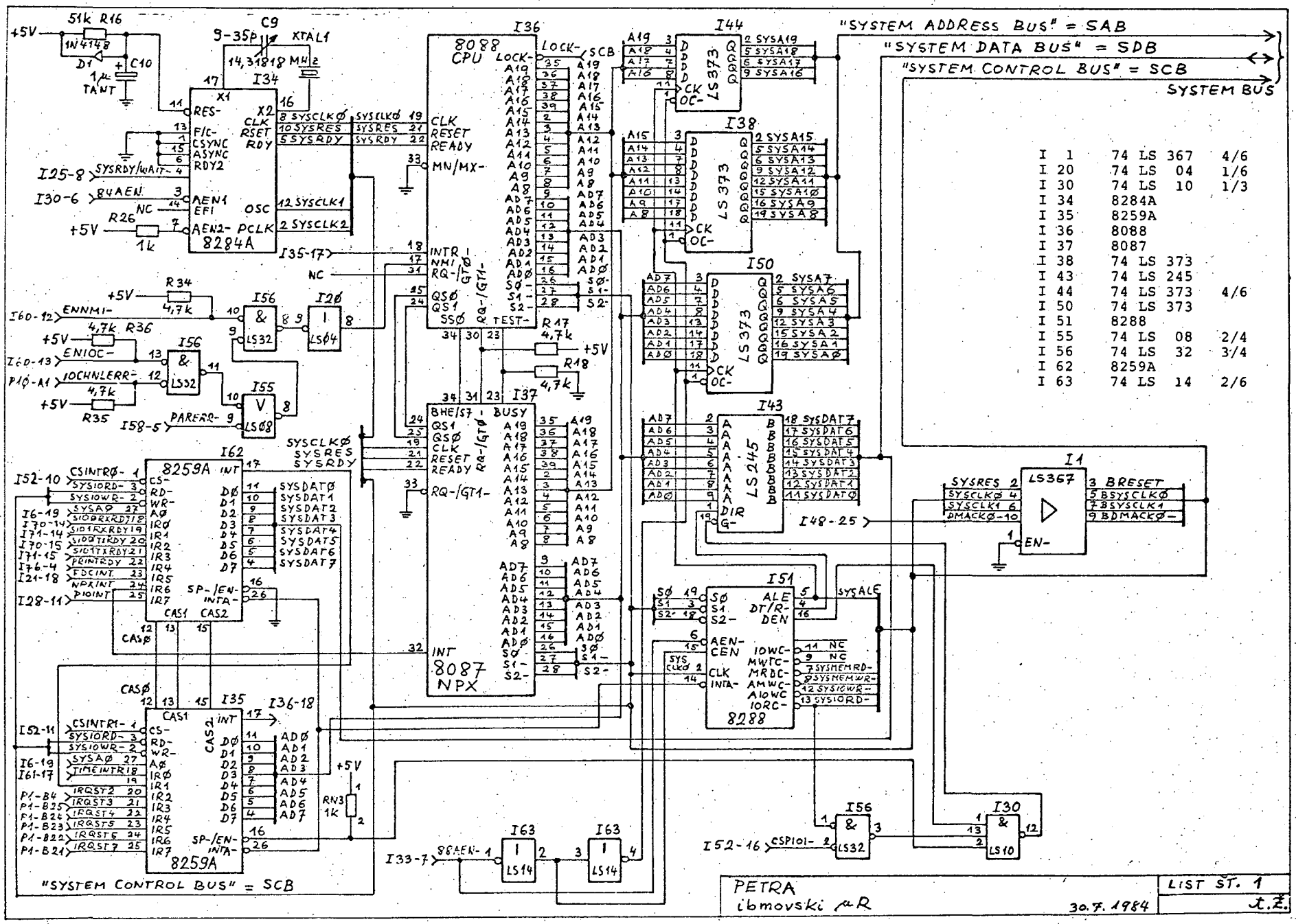
Slika 2 kaže pregled logične sheme na listu 1. Ta list ima tudi seznam integriranih vezij, uporabljenih na tem listu; ta seznam je urejen po zaporednih številkah (I nn), tako da lažje najdemo povezave med posameznimi oznakami v shemi.



Slika 1.5. Podrobnejši prikaz drugega dela V/I krmilnikov, vmesnikov in taktnika, povezanih na V/I vodilo.



Slika 2. Okvirni prikaz logičnega vezja na listu 1 (Petra mikroračunalnik, naslednja stran)


"SYSTEM ADDRESS BUS" = SAB
"SYSTEM DATA BUS" = SDB
"SYSTEM CONTROL BUS" = SCB
SYSTEM BUS
I 1 74 LS 367 4/6
I 20 74 LS 04 1/6
I 30 74 LS 10 1/3
I 34 8284A
I 35 8259A
I 36 8088
I 37 8087
I 38 74 LS 373
I 43 74 LS 245
I 44 74 LS 373 4/6
I 50 74 LS 373
I 51 8288
I 55 74 LS 08 2/4
I 56 74 LS 32 3/4
I 62 8259A
I 63 74 LS 14 2/6
PETRA
ibmovski & R
30.7.1984
LIST ST. 1
t.z.

Taktni generator z vezjem 8284A generira taktne signale (clk0, clk1, ...), signal sistemskega resetiranja reset in ready (sysrdy). Optimalni taktni signal je potreben za zanesljivo delovanje osrednjega (8088) in numeričnega procesorja (8087) (glej list 1). Taktna frekvenca procesorja 8088 mora biti med 2 in 5 MHz. Iz osnovne frekvence 14,31818 MHz se dobi z deljenjem frekvenca 4,77 MHz, ki je potrebna za barvno ibmovsko grafiko osebnega računalnika (dodatna vtična plošča). Kondenzator C9 (list 1) se uporablja za natančno nastavljanje taktne frekvence.

Vezje 8284 deli osnovno frekvenco s faktorjem 3 in generira signal sysclk0, ki sa uporablja več vezij (8088, 8087, 8288, V/I kanali), iz njega nastane tudi ustrezen signal dmaclk za DMA krmilnik 8237A-5 (vezje I28, nožica 3, list 3). Ta signal je potreben za boljšo sinhronizacijo med lokalnima mojstroma 8088 in 8087 in mojstrom sistemskega vodila 8237A-5. Signal sysclk1 je predviden za uporabo na vtičnih enotah (polovična taktna frekvenca na vtičnicah P1, P2, ... , P10; list 2).

Periferni takt sysclk2 se uporablja v vezju 8155H-2 (I47) za generiranje periodične osveževalne zahteve dinamičnih pomnilnikov z uporabo DMA krmilnika.

Pri vključitvi računalniškega napajanja se prek sponke res- vezja 8284 sproži generiranje resetirnega signala sysres, katerega minimalno trajanje 50 mikrosekund je povzročeno s časovno konstanto R16 in C10. Prek diode D1 se sprazni C10 pri izključitvi napajanja.

Nemaskirna prekinitev (NMI) procesorja 8088 se uporablja za obdelavo napake parnosti, kadar se ta pojavi v dinamičnem pomnilniku (RAM) in pri napakah V/I kanalov. Signal enmi- se uporablja za odločitev, ali naj nmi signal pride do procesorja 8088 ali ne. Z enioc- signalom (list 1) se odloči, ali bo signal iochnlerr- iz V/I kanalov povzročil prekinitev ali ne.

Procesor 8088 je povezan z numeričnim procesorjem 8087, ki je visokozmogljiv, standariziran matematični procesor. 8087 ima dodatnih 68 ukazov nad osmimi 80-bitnimi registri s plavajočo vejico. Tu je predviden 1 bit za predznak, 15 bitov za eksponent in 64 bitov za mantiso. Procesor 8087 izvaja aritmetične, trigonometrične, eksponentne in logaritmične ukaze skladno z IEEE standardom in tako povečuje zmogljivost sistema skupaj s procesorjem 8088 za 100-krat na področju številskih operacij. Posebna zahtevna/dodelitvena linija (rq-/gt0-) se uporablja za dodeljevanje lokalnega vodila med numeričnim in centralnim procesorjem. Tu gre za razmerje mojstra (8088) in njegovega pomočnika (8087) v računalniškem sistemu.

Iz slike 2 in iz kasnejših slik je razvidno, da obstajata dve kategoriji vodil: lokalna in globalna. Lokalna vodila povezujejo določene lokalne elemente, npr. procesorje 8088, 8087 in 8259A. Sistemsko vodilo je lahko krmiljeno iz lokalnega vodila (npr. krmilnik 8288) ali prek DMA krmilnika 8237A-5. Lokalno vodilo je multipleksirano (prekrivanje dela naslovnih s podatkovnimi linijami), sistemsko vodilo pa je demultipleksirano (20 naslovnih bitov, 8 dvosmernih podatkovnih bitov in več krmilnih bitov).

Krmiljenje sistemskega vodila je odvisno od arbitrske (presojevalne) sistemske logike. Vodilo je krmiljeno z enim od dveh koprocesorjev prek lokalnega vodila in krmilnika 8288 (s signalom 88aen-) ali z DMA krmilnikom 8237A-5 (s signalom aendma-). Enostavno arbitrsko vezje loči lokalno vodilo od sistemskega vodila tedaj, ko se sistemsko vodilo dodeli DMA krmilniku za neposreden dostop periferne naprave v hitri pomnilnik. Signal lock- se uporablja za signalno koordinacijo med koprocesorjema (8088 in 8087) in DMA krmilnikom.

Na sliki 2 imamo še dva prekinitvena krmilnika (mojstra in njegovega pomočnika). Sposobnost sistema se poveča z možnostjo prekinjanja njegovih procesov z zunanjimi dogodki, tako da ni potreben stalen nadzor zunanjih elementov. Iz slike 2 je razvidno, da obstaja 16 ravnin prekinitvenih prednosti pri uporabi zunanje (periferne) okolice. Z uporabo prekinitev se povečuje pretočnost sistema pri obravnavi sistemskih V/I funkcij. Programski vmesniki za obdelavo posameznih prekinitev so navadno bolj kompaktni in učinkoviti kot vmesniki za nadzor in čakanje pogojev za prenos podatkov.

Prekinitvena struktura s procesorjem 8088 ima tabelo prekinitvenih vektorjev, ki je locirana v spodnjem delu pomnilnika med naslovoma 00000H in 003ffH. Vsak prekinitveni vektor v tabeli je sestavljen iz 4 zlogov in kaže na naslov prekinitvene storitvene rutine. Uporabiti je mogoče 256 prekinitvenih vektorjev, ki so oštevilčeni od 0 do 255; naslovi storitvenih rutin so lahko kjerkoli v 1M-zložnem naslovnem prostoru. Vsakemu prekinitvenemu vektorju je prirejeno prekinitveni tipsko število, ki kaže na njegovo lokacijo v prekinitveni tabeli. Tipsko število pomnoženo s 4 je enako odmiku vektorja od lokacije 00000H.

Prekinitev z najvišjo prednostjo je NMI (8088, nožica 17, I36). Dodatno maskiranje tega vhoda je omogočeno s posebnim vezjem, tako da je omogočena tudi programska kontrola te prekinitve. NMI vhod se uporablja za signalizacijo pomnilniške napake parnosti in napak, ki se lahko pojavijo na vtičnih ploščah V/I kanalov.

Naslednjih 15 prekinitvenih ravnin je implementiranih z dvema programirljivima prekinitvenima krmilnikoma 8259A (I35 in I62). Pri tem deluje krmilnik I35 kot mojster, ki se nahaja na multipleksiranem lokalnem vodilu skupaj s procesorjema 8088 in 8087. Krmilnik IC62 je pomočnik krmilnika IC35 in se nahaja na sistemskem vodilu. Ta mojstrska/pomočniška konfiguracija se inicializira s posebnim programskim segmentom.

Vse zunanje naprave na glavni plošči (računalniški sistem Petra) imajo svojo prekinitveno linijo. Prekinitvene zahteve iz prekinitvenih krmilnikov dosežejo vhod intr procesorja 8088, ki je notranje maskiran s programskim ukazom. Pri pojavitvi prekinitvenega signala na vhodu intr vstopi procesor v potrditveni cikel, v katerem se določi prekinitveni tip. Najprej shrani procesor stanje, ki je bilo tik pred prekinitvijo v sklad (zastavni register, kodnosegmentni register, ukazni števnik). Pri tem se zbriše prekinitvena zastavica in onemogočijo se nadaljnje prekinitve, dokler procesor ni pripravljen za njihov sprejem.

V naslednjem koraku izda krmilnik vodila 8288 signal prekinitvene potrditve na linijo inta- (glej list 1). Prvi inta- impulz pove krmilnikoma 8259A, da je bila prekinitvena zahteva potrjena. Pri drugem inta- impulzu se na podatkovno vodilo izda 8-bitni prekinitveni tip. Vrednost prekinitvenega tipa se pomnoži s 4 in s tem se določi prekinitveni vektor. Programsko krmiljenje se nato prenese na naslov, ki je vsebovan v 4-zložnem prekinitvenem vektorju. Prva dva zloga vektorja oblikujeta novi ukazni kazalec (spodnjih 16 bitov naslova), drugi dva zloga pa se uporabljata za oblikovanje nove vsebine kodnosegmentnega registra (zgornjih 16 bitov). Ko je prekinitvena storitvena rutina opravila svojo funkcijo, se programsko krmiljenje vrne glavnemu programu prek ukaza IRET, ki obnovi prvotno stanje, tako da se lahko nada-

Tabela 1

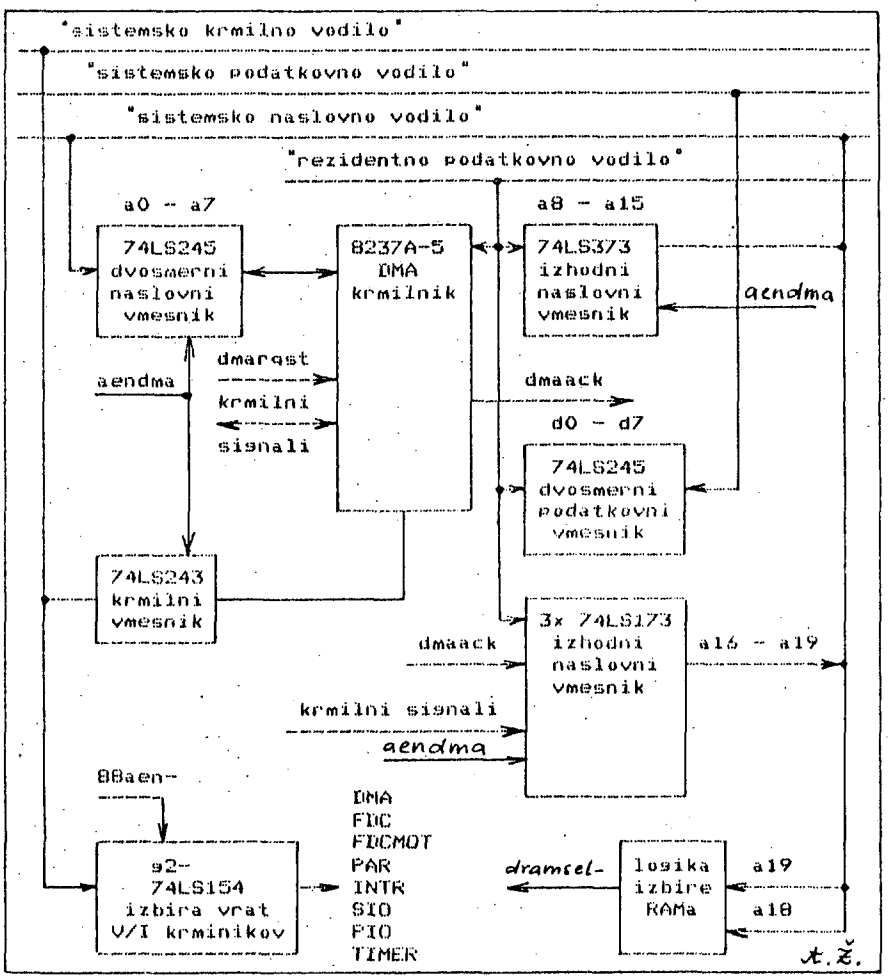
prednostna ravnina	vir	signalno ime	opis
0	NMI	parerr, iocnhnlerr	pomnilniška parnost ali napake V/I kanalov
1	mojster	timeintr	ura realnega časa
2	pomočnik	sio0rxrdy	pripravljenost za sprejem serijskega kanala A
3	pomočnik	sio1rxrdy	pripravljenost za sprejem serijskega kanala B
4	pomočnik	sio0txrdy	pripravljenost za oddajo serijskega kanala A
5	pomočnik	sio1txrdy	pripravljenost za oddajo serijskega kanala B
6	pomočnik	printrdy	pripravljenost tiskalnih vrat
7	pomočnik	fdcint	prekinitev krmilnika za upogljivi disk
8	pomočnik	npxint	prekinitev numeričnega procesorja (8087)
9	pomočnik	pioint	prekinitev paralelnih V/I vrat
10	mojster	irqst2	prekinitev V/I kanala
11	mojster	irqst3	prekinitev V/I kanala
12	mojster	irqst4	prekinitev V/I kanala
13	mojster	irqst5	prekinitev V/I kanala
14	mojster	irqst6	prekinitev V/I kanala
15	mojster	irqst7	prekinitev V/I kanala

ljuje izvajanje glavnega programa.

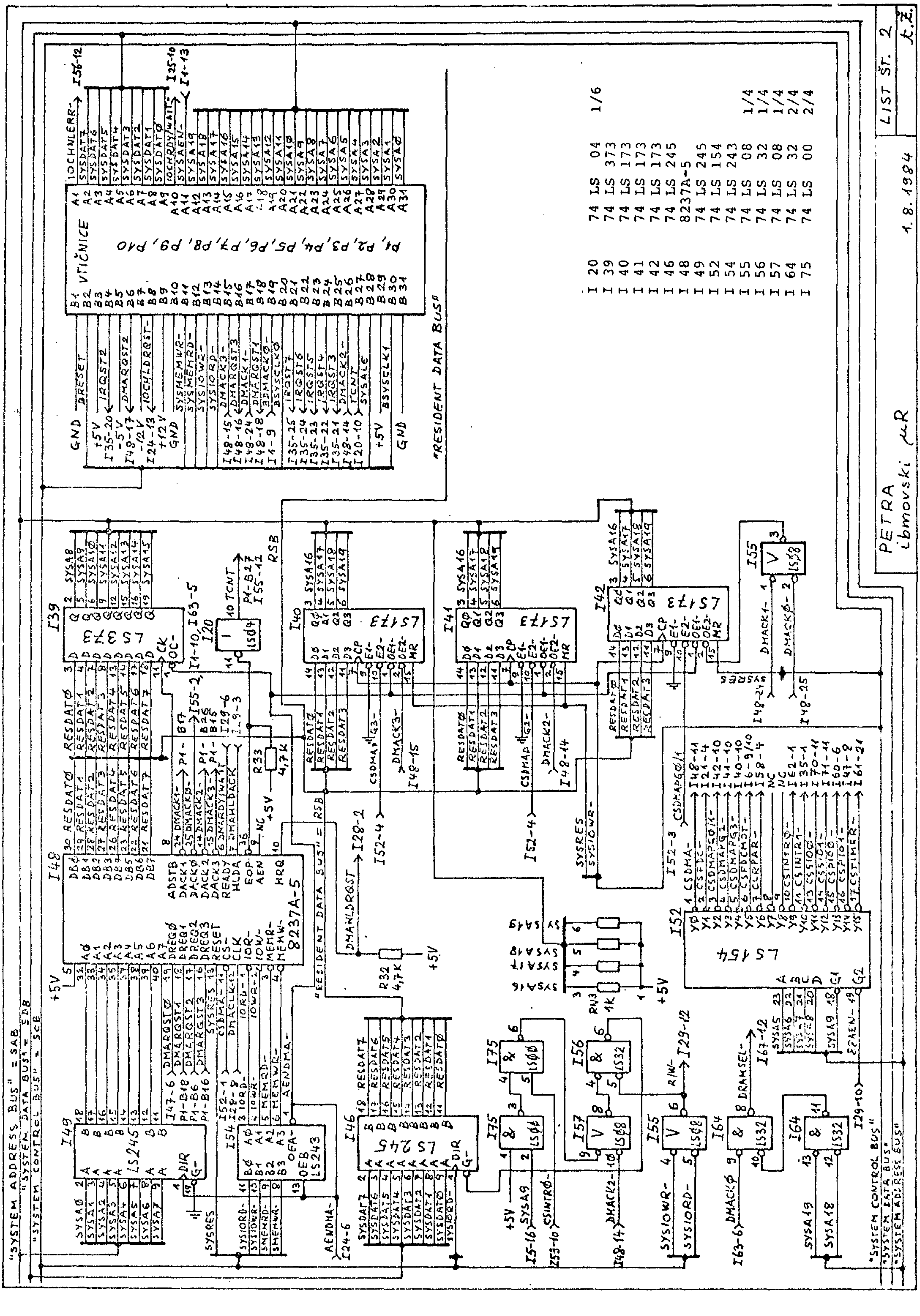
Tabela 1 kaže shemo prekinitvenih prednosti. Najvišjo prekinitveno prednost ima NMI, ki je povzročen z napako pomnilniške parnosti ali z napako V/I kanala. Najvišjo prednost maskirne prekinitve ima vhod ir0 mojstra 8259A, na katerega je povezan signal timeintr. Tabela 1 kaže prednostno zaporedje preostalih signalov. Prekinitveni mehanizem se lahko uporablja še za vrsto drugih namenov, kot so odkrivanje napak pri začetnem oživljanju sistema itd.

4. Logična shema lista 2

Na sliki se nahaja blokovna shema vezja na listu 2. Tri enote oblikujejo to shemo: DMA krmilnik z vmesniki, V/I izbiralnik (74LS154) in logika za izbiro pomnilniških segmentov (enote po 256k zlogov). Na listu 2 se nahaja še shema signalnih priključkov na vtičnicah P1, ... , P10, ki se uporabljajo za dodajanje zunanjih tiskanih plošč (ibmovske vtične plošče).



Slika 3. Ta slika prikazuje okvirno logično shemo na listu 2. Osrednje integrirano vezje te sheme je DMA krmilnik 8237A-5, ki opravlja neposreden prenos podatkov med hitrim pomnilnikom in perifernimi krmilnimi vezji. DMA krmilnik posreduje med sistemskim vodilom (podatkovnim, naslovnim) in rezidentnim (krajevnim) podatkovnim vodilom. 16 bitov za sistemsko naslovno vodilo se oblikuje prek dveh 8-bitnih vmesnikov (I49 in I39, list 2), manjkajoči 4 biti (a16 do a19) pa nastanejo v odvisnosti od ustrezne DMA zahteve v različnih pomnilnih segmentih. Na listu 2 se nahaja še shema povezave signalov na vtičnice za dodatne tiskane plošče (npr. krmilniki, barvna grafika, modemi, A/D moduli itd.). Naslovi vrat posameznih perifernih krmilnikov so določeni z vezjem 74LS154. Na tem listu se nahaja še osnovno vezje za izbiro 4 segmentov po 256k zlogov. List 2 ima desno spodaj še seznam vseh uporabljenih vezij (po vrstnem redu), tako da je olajšano iskanje označenih povezav (to velja tudi za ostale liste).


PETRA
ibmovski μR
LIST ŠT. 2
1.8.1984
"RESIDENT DATA BUS"
"SYSTEM ADDRESS BUS" = SAB
"SYSTEM DATA BUS" = SDB
"SYSTEM CONTROL BUS" = SCB
8237A-5
VTIČNICE

Neposreden pomnilniški dostop (DMA) povečuje zmogljivost sistema, ker se z njim poveča hitrost prenosa podatkov med periferijo in hitrim pomnilnikom (uporaba DMA krmilnika 8237A-5). Pri tem se podpirajo štiri neodvisni kanali 20-bitnonaslovnega neposrednega pomnilniškega dostopa. Dva DMA kanala sta predvidena za razširitveno (sistemsko) vodilo (vtičnice P1, ... , P10) za hiter podatkovni prenos med perifernimi napravami in RAM pomnilnikom. Tretji kanal uporablja krmilnik za upogljive diske (skladno s standardom za IBM PC). Četrti DMA kanal se uporablja za periodično osveževanje hitrega pomnilnika na osnovni plošči in na dodatnih vtičnih ploščah, ko mora biti dostopna vsaka naslovna vrstica dinamičnih pomnilnih integriranih vezij.

Pri inicializaciji sistema se ustrezno nastavi vezje 8155H-2 oziroma njegov signal timer out, tako da se opravlja navidezni DMA prenos približno vsakih 15 mikrosekund. DMA kanal je programiran za pomnilni bralni cikel, ko se avtomatično povečuje vrstični naslovni števnik za pomnilnik po vsakem osveževalnem ciklu.

Kadar ni DMA zahtev, je DMA krmilnik v mirovnem stanju (s1) in sa je mogoče programirati prek centralne procesne enote. Kadar DMA kanal zahteva storitev za zunanjo napravo in je bil ta kanal pripravljen (usposobljen) s sistemsko programsko opremo, odpošlje DMA krmilnik signal dmahldrqst (DMA hold request) v vezje za sistemsko presojo (arbitriranje) in preide nato v stanje s0. Krmilnik 8237A-5 ostane v stanju s0, dokler ne sprejme signala dmahldack (DMA hold acknowledge) iz vezja za arbitriranje vodila; to je znak, da je bilo krmiljenje podeljeno sistemskemu vodilu.

V tem razdobju loči arbitrsko vezje za sistem in vodilo lokalno vodilo od sistemskega vodila z izdajo krmilnega signala 88aen-. Ko ta signal zopet preneha, postavi krmilnik vodila 8288 (I51) vmesnike sistemskih krmilnih linij v stanje visoke impendance in izključi podatkovni prenosnik 74LS245 (I43, list 1). Signal 88aen- pa postavi naslovne vmesnike I50, I38 in I44 (list 1) v stanje visoke impendance, tako da mojster lokalnega vodila lahko krmili lokalno vodilo med DMA ciklom brez vpliva na operacije sistemskega vodila.

Po enem sistemskem taktnem ciklu za signalom dmahldack se s krmilnim signalom aendma- iz vezja I24 aktivirajo povezave iz DMA krmilnika na vodilo. Vezje I43 (74LS373, list 1) zagotovi sistemske naslovne signale sysa8 do sysa15, vezje I49 (74LS245, list 2) pa naslovne signale sysa0 do sysa7. Smer prenosa podatkov se krmili s signalom aendma- na vezju I49 med DMA prenosom.

Med procesorskim pomnilnim prenosom je signal aendma- visok (neaktiven) in naslovna informacija se pretaka iz sistemskega naslovnega vodila skozi vezje I49 v krmilnik 8237A-5. Štirje najvišji naslovni biti sysa16 do sysa19 se pošiljajo skozi vmesnike I40, I41 in I42. Ti vmesniki (bistabilna vezja) se naložijo prek operacijskega sistema ali z uporabniškim programom in omogočijo DMA kanalom, da uporabljajo ločene pomnilniške segmente obsega 64k zlogov. Ker se DMA kanal 0 uporablja za pomnilniško osveževanje in je bistvenih le osem nižjih naslovnih bitov (linij), se uporablja vmesnik DMA kanala 1 za sornje štiri naslovne bite tako za kanal 0 kot za kanal 1 (I42). Trije naslovni vmesniki se aktivirajo, ko je signal aendma- nizek (aktiven) in se je hkrati pojavil ustrezen signal dmack- (I40, I41, I42).

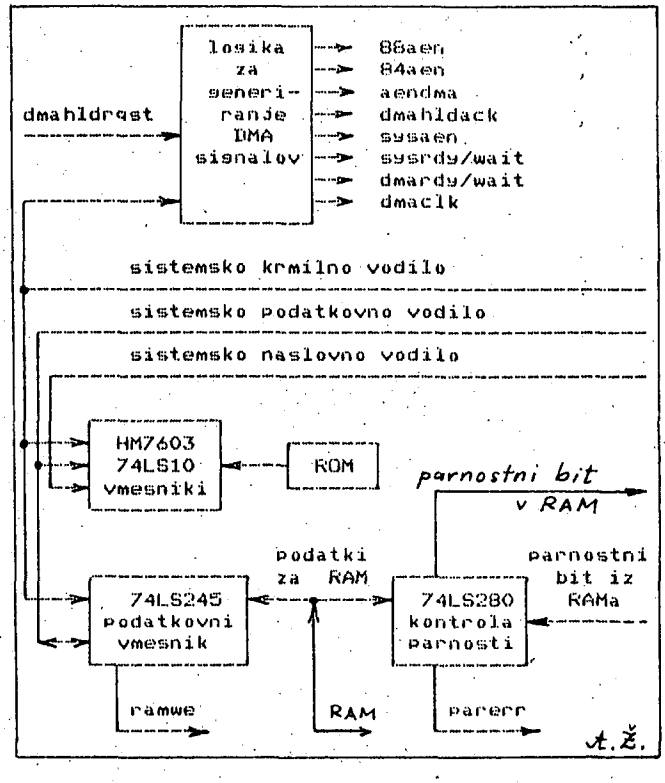
Ko se je končal prenos enega zloga, ukine DMA krmilnik signal dmahldrqst in s tem preneha obstajati tudi signal dmahldack. V naslednjem taktnem ciklu se sistemske povezovalne komponente in krmilnik 8288 zopet aktivirajo z nizko vrednostjo signala 88aen-, DMA povezovalne komponente pa se deaktivirajo z visoko vrednostjo signala aendma-. Ko začne signal 88aen- delovati, se krmilnik 8288 aktivira šele po 105 do 275 ns; to zakasnitev povzroče vezja I30 (list 1) in verige bistabilnih vezij (I32, I33, list 3).

Signal dmahldrqst preneha po prenosu vsakega zloga; tako se pojavi vsaj en strojni cikel med dvema zaporednima DMA prenosoma.

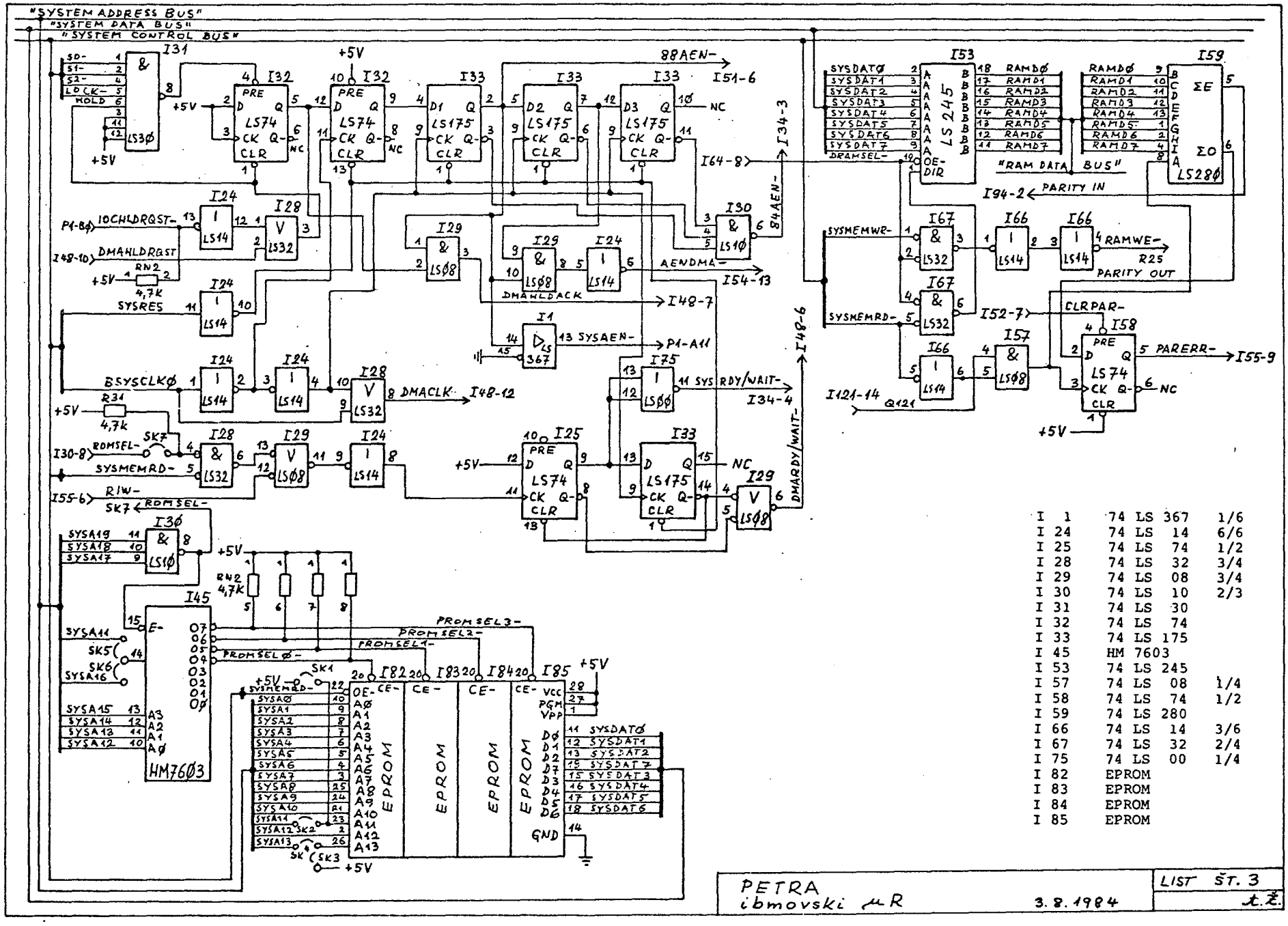
5. Logična shema lista 3

Na sliki 4 je prikazana blokovna shema logičnega vezja na listu 3. To shemo sestavljajo trije logični segmenti: logično vezje za generiranje DMA signalov (88aen-, 84aen-, aendma-, dmahldack, sysaen-, sysrdy/wait-, dmardy/wait-, dmaclk), vmesnik za izbiro ROM pomnilnika s samim ROM pomnilnikom in logično vezje za kontrolo parnosti za RAM pomnilnik.

Nastajanje bistvenih DMA signalov in njihova odvisnost sta bila delno opisana že v prejšnjem poglavju. Na listu 3 imamo sekvenčno logično vezje, ki generira DMA signale, navedene v gornjem delu slike 4. Logiko za generiranje DMA signalov na listu 3 sestavljajo integrirana vezja I31 (in-vrata), I25, I32, I33 (bistabilni elementi), I24, I28, I29 in I30 (logična vrata). Signali na sliki 4 nastajajo v ustreznem



Slika 4. Okvirni prikaz logičnega vezja na listu 3, ki vsebuje logiko za generiranje DMA signalov, vmesnike za pomnilnik tipa ROM z ROM pomnilnikom, podatkovni vmesnik med sistemskim podatkovnim vodilom in RAM pomnilnikom in logiko za kontrolo parnosti (I59, I58, I57, I66, list 3).



I 1	74 LS 367	1/6
I 24	74 LS 14	6/6
I 25	74 LS 74	1/2
I 28	74 LS 32	3/4
I 29	74 LS 08	3/4
I 30	74 LS 10	2/3
I 31	74 LS 30	
I 32	74 LS 74	
I 33	74 LS 175	
I 45	HM 7603	
I 53	74 LS 245	
I 57	74 LS 08	1/4
I 58	74 LS 74	1/2
I 59	74 LS 280	
I 66	74 LS 14	3/6
I 67	74 LS 32	2/4
I 75	74 LS 00	1/4
I 82	EPROM	
I 83	EPROM	
I 84	EPROM	
I 85	EPROM	

PETRA
ibmovski μR
3.8.1984

LIST ŠT. 3
Ž.Ž.

časovnologičnem razmerju, tako da se dosegajo potrebni učinki DMA prenosa, kot je bilo opisano v prejšnjem poglavju.

Predvideni obseg ROM pomnilnika (integrirana vezja I82, I83, I84 in I85) je 64k zlogov (list 3), vendar je mogoče ta obseg povečati z dodajanjem ROM pomnilnika na vtičnih ploščah. Za ROM pomnilnik so predvidena 28-nožična podnožja, tako da se lahko uporabijo različna vezja tipa EPROM, kot so npr. 2716 (2k zlogov), 2732 (4k zlogov), 2764 (8k zlogov) in 27128 (16k zlogov). EPROMi s 24 nožicami, kot sta 2716 in 2732, se vstavijo v spodnjih 24 luknjic pri ustrezni konfiguraciji skakačev (SK).

EPROM pomnilniki I82, ... , I85 pokrijejo najvišji del naslovnega prostora (I85) in skladno z uporabljenimi pomnilniki (2716, ... , 27128) so nameščeni skakači SK1 do SK6 v povezavi z naslovnokodirnim PROMom I45 (HM7603, list 3). V prvi fazi preizkušanja lahko uporabimo EPROMe tipa 2732 (4k zlogov). Pri uporabi drugih tipov je potrebno ustrezno spremeniti dekodirni PROM I45, tako da je vselej zaseden strnjen zgornji del naslovnega prostora. Za PROM I45 si moramo predhodno izdelati njegovo kodirno tabelo, nato pa to tabelo vprogramiramo.

V naslovni prostor za ROM pomnilnike se dostopa vselej, ko so biti sysa17, sysa18 in sysa19 visoki (I30, list3). S petimi naslovnimi biti

sysa11, sysa12, sysa13, sysa14, sysa15
ali
sysa12, sysa13, sysa14, sysa15, sysa16

za PROM kodirnik (I45) se aktivirajo linije promsel0-, ... , promsel3- za izbiro ROM pomnilnikov I82, ... , I85. Signal romsel-, ki prihaja z vezja I30 na skakač SK7 (I28, nožica 7), lahko sproži pri sklenjenem SK7 čakalno stanje (pri počasnih ROM pomnilnikih). Ko se je pojavil eden od signalov promseln-, se začne s signalom sysmemrd- pomnilniški bralni cikel in generira se eno čakalno stanje, če je SK7 sklenjeno. Veljavni podatki iz ROM pomnilnika se pojavijo na podatkovnem vodilu, ko je signal sysmemrd- postal nizek.

Normalno uporabljamo ROM ali EPROM pomnilnike s časom dostopanja do 350 ns. Uporaba čakalnega stanja s skakačem SK7 omogoča vstavitev počasnejših ROM pomnilnikov s časom dostopa do 450 ns.

EPROM pomnilniki sistema Petra lahko vsebujejo več standardnih (ibmovskih) rutin, kot so avtomatični preizkus sistema ob vključitvi, BIOS za uporabljani operacijski sistem (PC-DOS, CP/M-86) in navezovalni nalagalnik za vinčestrski ali uporabniški disk (v odvisnosti od diskovne konfiguracije sistema Petra).

Preizkus parnosti za RAM pomnilnik (vezja I59, I58, I57, I66, list 3) je bil uporabljen tudi v osebnem računalniku IBM PC. Tak preizkus je bil do nedavnega v navadi le v velikih računalnikih. S preizkusom parnosti se povečujeta sistemska zanesljivost in uporabniška zaupljivost. Za generiranje bita parnosti se uporablja parnostni generator 74LS280 (I59, list 3) in bistabilno vezje tipa D (I58, 74LS74, list 3).

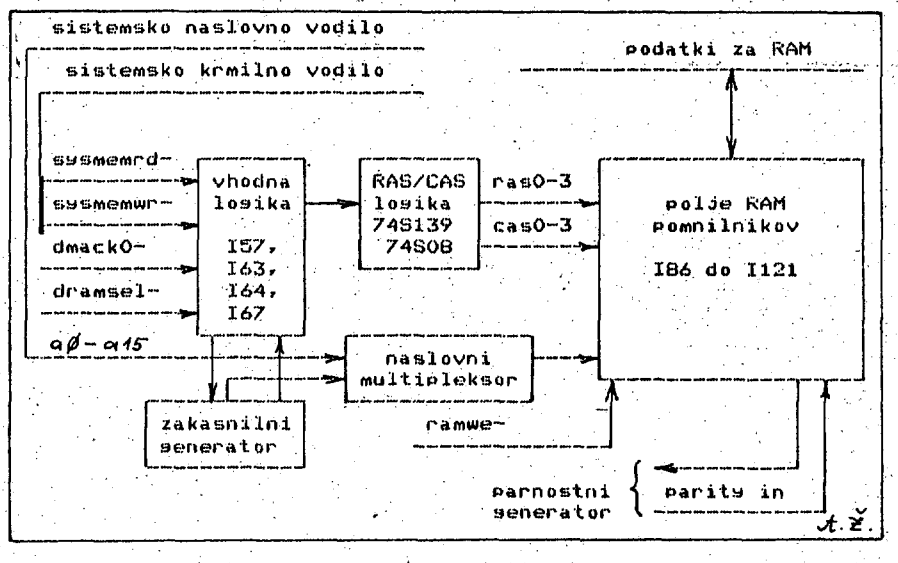
V pomnilniškem pisalnem ciklu je signal "parity out" vezja 74LS280 (I59) nizek, ker je nizek izhod vezja IC57 (in-vrata 74LS08) zaradi visokega signala sysmemrd-. Parnostni bit, ki se izračuna v vezju I59 iz osmih vhodnih podatkovnih bitov, se vpiše v posebno pomnilniško dinamično integrirano vezje (deveto vezje v vsaki pomnilniški banki) prek izhoda "parity in" vezja I59.

Ko se je pojavil cikel pomnilniškega branja, se aktivira izhod vezja IC57 (in-vrata 74LS08) in parnostni bit se iz RAMa prenese na vhod A vezja I59 in se primerja z izračunanim bitom iz RAM podatkov v vezju I59. Signal napake se potem oblikuje v odvisnosti od ujemanja ali neujemanja obeh parnostnih bitov in se prenese v bistabilno vezje I58.

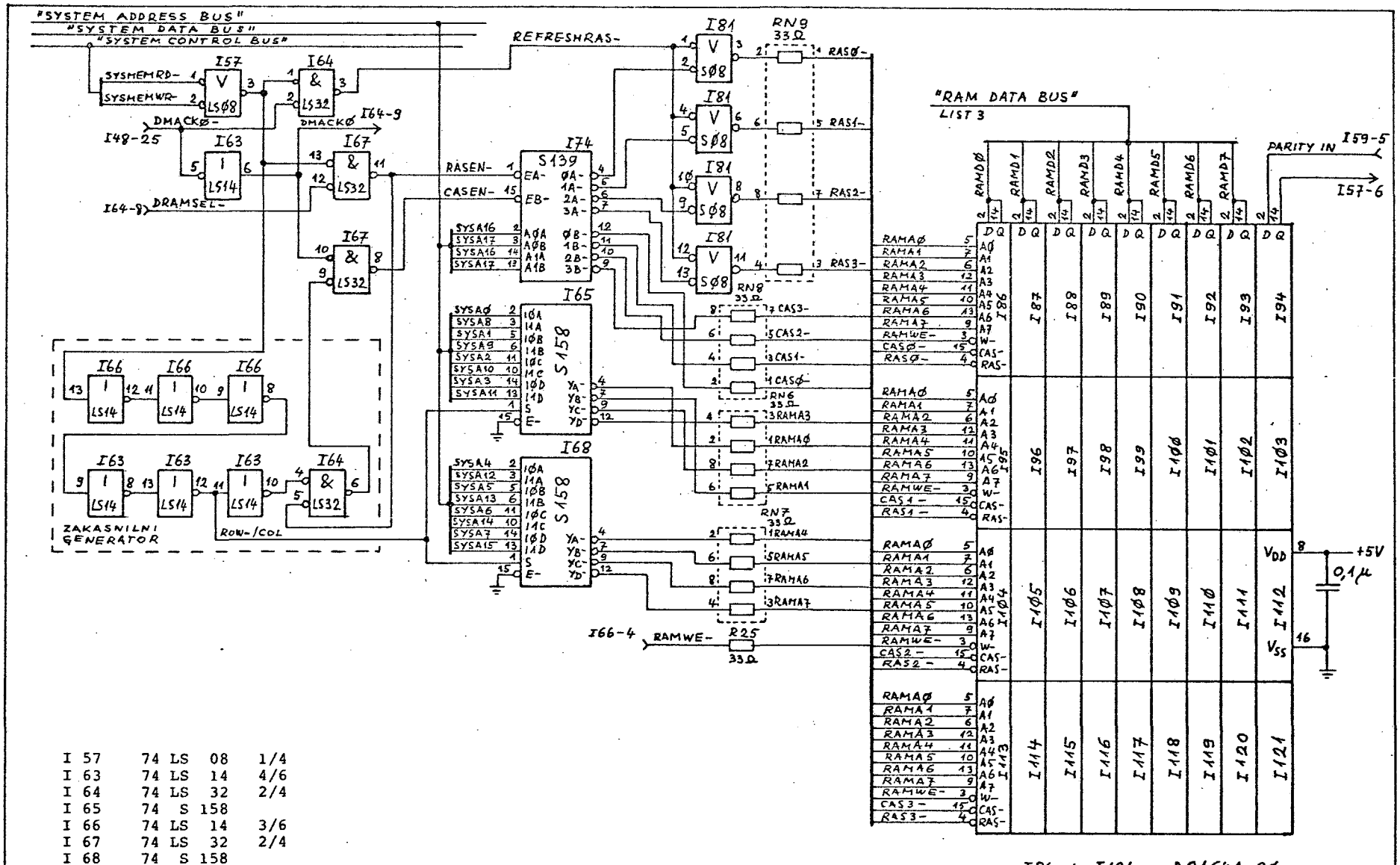
Kadar ni parnostne napake, je izhod lihe parnosti (Σ0, I59) vezja 74LS280 v visokem stanju. Pri napaki parnosti je ta izhod nizek. Signal parerr- iz vezja I58 se pošlje na vhod tkim. NMI vezja (list 1, I55, nožica 9). Bistabilno vezje I58 ostane v tem stanju do naslednjega pomnilniškega cikla ali dokler se I58 ne resetira s signalom clrpar-. Pri tem je bila ob napaki prek prekinitvenega vektorja poklicana popravna rutina.

6. Logična shema lista 4

Slika 5 prikazuje blokovno shemo logičnega vezja na listu 4. To vezje je sestavljeno iz več enot, kot so vhodna RAM logika, RAS/CAS vezje, zakasnilni generator, naslovni multipleksor in polje RAM pomnilnikov. Bistveni vhodni signali so sysmemrd-, sysmemwr-, dmack0-, dramsel- in ramwe-.



Slika 5. Ta slika prikazuje blčno shemo logičnega vezja na listu 4. List 4 vsebuje del logike RAM pomnilnika in sam RAM pomnilnik, ki je sestavljen iz štirih pomnilniških bank s po 64k zlogi (skupaj 256k zlogov). Vhodna logika je sestavljena iz vrste logičnih vrat (I57, I63, I64, I67) in krmiljena s 4 signali. Zakasnilni generator daje multipleksorski signal row-/col- (multipleksiranje naslovov za RAM). Sam RAM pomnilnik sprejema še ramwe- signal in sprejme in odda signala za parnostni generator. Podatki za RAM prihajajo in odhajajo prek dvosmernega vmesnika na sistemsko podatkovno vodilo (glej sliko 4).



I 57	74 LS	08	1/4
I 63	74 LS	14	4/6
I 64	74 LS	32	2/4
I 65	74 S	158	
I 66	74 LS	14	3/6
I 67	74 LS	32	2/4
I 68	74 S	158	
I 74	74 S	139	
I 81	74 S	08	4/4
I 86	D 2164-20		
. ..	. ....	..	
I121	D 2164-20		

PETRA
ibmovski µR
8. 8. 1984

LIST ŠT. 4
t.ž.

Poudarimo, da so bistvena integrirane vezja na listu 4 tipa S, in sicer za RAS/CAS logiko (74S139, 74S08, tj. I74, I81) in za naslovni multipleksor (74S158, tj. I65, I68). Dušilni zaporedni upori so dosledno vstavljeni v linije RAS, CAS in naslovnih signalov.

Tkim. banka dinamičnega RAM pomnilnika je sestavljena iz 9 pomnilnih vezij po 64k bitov (tipa 2164 ali 4164), kjer je deveti bit predviden za kontrolo parnosti. Osnovna plošča ima 4 take banke, torej skupaj 256k zlogov pomnilnega prostora. Procesor 8088 lahko naslovi 1M zlogov pomnilnika in ta sistem mora imeti vsaj eno spodnjo banko (naslovi od 00000H do 0FFFFH), v kateri se shranjujejo kazalci prekinitvenih storilnih rutin (na lokacijah 00000H do 003FFH). Dinamična pomnilna integrirana vezja (RAM) morajo imeti čas dostopa vsaj 200 ns in čas cikla vsaj 335 ns. Generiranje parnostnega bita in zaznavanje napake za 256k-zložni pomnilnik je bilo opisano že v prejšnjem poslavju.

Naslovni prostor RAM pomnilnika na osnovni plošči je izbran z dvema izpolnjenima pogojema: bita sysa18 in sysa19 morata biti nizka in osveževalni cikel ne sme biti v postopku (visoka vrednost signala dmackO-). Dinamični pomnilniki se morajo periodično osveževati, da se ohranjajo shranjeni pomnilniški podatki. Dinamični RAMi s 64k biti zahtevajo, da se vseh 256 vrstic naslovi vsakih 4 ms. Vrstica pomnilnika mora biti naslovljena za osveževanje približno vsakih 15 mikrosekund. Osveževanje pomnilnika se lahko doseže z DMA bralnim ciklom, in sicer z tkim. načinom "RAS-only", ko se uporabijo samo RAS vzbujevalni impulzi. Ker se osveževanje krmili z DMA vezjem (8237A-5), se ne more pojaviti nasprotje med operacijama osveževanja in pomnilniškega branja/pisanja.

Ko preide signal dmackO- v nizko stanje, se začne osveževalni cikel. Ta signal izključi RAM dekodirno vezje, prepreči generiranje CAS signala (I67) in aktivira s signalom refreshras- (I64) vse štiri RAS linije (I81). DMA krmilnik se s sistemskim inicializacijskim programom programira za avtomatično inkrementiranje naslovnega števnika po vsakem pomnilniškem osveževalnem ciklu. Delovanje dinamičnega pomnilnika je opisano v dokumentaciji proizvajalca pomnilnikov in ga tu ne bomo ponavljali. Poudarimo le, da je določeno časovnoodvisno zaporedje signalov nujno, in sicer skladno s shemo na listu 4 imamo tole zaporedje:

```
sysmemrd-  ali  sysmemwr-
rasen-
ras0- ali ras1- ali ras2- ali ras3-
vrstični naslovi
row-/col
stolpni naslovi
casen-
cas0- ali cas1- ali cas2- ali cas3-
ramwe-  (istočasno z ras- signali)
podatki
```

Dinamični pomnilniki morajo biti seveda ustrezno blokirani (zaščiteni proti medsebojnim motnjam). Proizvajalec natanko predpisuje potrebne vrednosti blokirnih kondenzatorjev (glej list 4, I112).

7. Sklep k prvemu delu

V prvem delu članka nismo opisali logičnih shem krmilnika za upogljive diske, serijsko in paralelno V/I povezavo, števnike in časovnike. Ta opis bo prikazan v drugem delu članka. Kasneje bomo opisali tudi oživljanje sistema s posebnimi začetnimi (preizkusnimi) programi in nekaterimi značilnimi signalnimi poteki in naposled oživitev računalnika z operacijskim sistemom.

Slovstvo

((1)) A.P.Železnikar, D.Novak: Mikroračunalnik Ike-1 s procesorjem 8086. Informatica 3 (1979), št. 2, str. 3-13.

((2)) B.Blatnik, A.Hadži, M.Kovačević, A.Leskovar, D.Novak, D.Šalehar, A.P.Železnikar: Mikroračunalniški sistem Delta 323/M. Informatica 6 (1982), št. 1, str. 4-22.

((3)) D.Novak, M.Kovačević, A.P.Železnikar: Mikroračunalnik Vita s procesorjem 6800. Informatica 2 (1978), št. 1, str. 14-20.

((4)) A.Uratnik, M.Rogač, A.P.Železnikar: Mikroračunalnik Iskradata 1680: moduli in sistemi. Informatica 2 (1978), št. 4, str. 7-8 in str. 77-100.

((5)) A.P.Železnikar, D.Novak: Možnosti razvoja mikroračunalniške tehnologije v SFRJ. Informatica 5 (1981), št. 1, str. 4-11.

((6)) A.P.Železnikar: Uvod v CP/M I, II, III. Informatica 5 (1981), št. 3, str. 63-67; št. 4, str. 9-23; Informatica 6 (1982), št. 1, str. 33-42.

((7)) A.P.Železnikar: Operacijski sistem CP/M Plus. Informatica 7 (1983), št. 1, str. 3-10.

((8)) S.Ciarcia: Build the Circuit Cellar MPX-16 Computer System I. Byte (1982), št.11, str. 78-114.

((9)) S.Ciarcia: Build the Circuit Cellar MPX-16 Computer System II. Byte (1982), št. 12, str. 42-78.

((10)) S.Ciarcia: Build the Circuit Cellar MPX-16 Computer System III. Byte (1983), št. 1, str. 54-80.

INFORMATICA 4/84

ODKRIVANJE NAPAK Z BERGEROVIM IN PODOBNIMI KODI I.

RUDI MURN
SAŠA PREŠEREN
DUŠAN PEČEK
BORUT KASTELIC

UDK: 681.3.325.6.08

INSTITUT JOŽEF STEFAN, LJUBLJANA
ODSEK ZA RAČUNALNIŠTVO IN INFORMATICA

Članek opisuje problematiko detektiranja napak v sodobnih digitalnih vezjih s pomočjo kodnih sistemov. Analizirani so Bergerovi kodi, ki se izkažejo zlasti primerni za detekcijo istoznačnih napak. Definirane so vrste napak v digitalnih sistemih, prikazana je učinkovitost Bergerovih kodov in realizacija ustreznih testnih vezij.

ERROR DETECTION WITH BERGER CODE AND MODIFIED BERGER CODE: This paper describes a problem of error detection in modern digital circuits by different code systems. Analysed is Berger code, which is particulary suitable for detection of unidirectional errors. Deffined are types of errors that occur in digital systems, demonstrated is the efficiency of Berger code and implementation of test circuits.

1. UVOD

Lastnosti različnih kodov za odkrivanje in korekcijo napak v digitalnih sistemih so obdelali številni avtorji tako teoretično v sklopu teorije kodiranja, kot tudi praktično v konkretnih aplikacijah. Nova tehnologija proizvodnje integriranih vezij (LSI, VLSI), kjer obstaja velika verjetnost za pojav istoznačnih (unidirectional) napak in potreba po večji zanesljivosti računalniških sistemov je povzročila pospešen razvoj učinkovitih kodnih postopkov za odkrivanje istoznačnih napak. V tej zvezi je posebnega pomena Bergerov kod.

Poleg tega, da bomo v tej študiji predstavili Bergerov kod in modificiran Bergerov kod, bomo obdelali tudi bistvene lastnosti Bergerovega koda. Podali bomo algoritme kodiranja in konkretno implementacijo testnega vezja za Bergerov kod.

Pokazali bomo prednosti in slabosti Bergerovega koda in modificiranega Bergerovega koda v primeri z ostalimi kodi. Pokazali bomo kje in kako naj posamezni kod uporabljamo in utemeljili primernost uporabe modificiranega Bergerovega koda pri testiranju PLA (programmable logic arrays) vezij.

1.1. Vrste napak v digitalnih sistemih

V računalniških sistemih se pojavljajo predvsem napake naslednjega tipa [BOS82]:

a) trajne napake:
 simetrične napake,
 asimetrične napake,
 istoznačne napake in

b) trenutne napake:
 nakljućne napake.

Tip napake predstavlja osnovo za formulacijo različnih pristopov za odkrivanje in odpravljanje napak.

Definicija 1: Napaka je simetrična, če je verjetnost za napako zaradi prehoda od 1 na 0 enaka verjetnosti prehoda od 0 na 1.

Definicija 2: Napaka je asimetrična, če je verjetnost za napako zaradi prehoda od 1 na 0 različna od verjetnosti prehoda od 0 na 1.

Definicija 3: Napake je istoznačna, če sestoji samo iz enega tipa napak, to je iz napake zaradi prehoda od 1 na 0 ali od 0 na 1, ne pa oboje hkrati.

Definicija 4: Napake so nakljućne, če ni med njimi nobene logične povezave.

Definicija 5: Napako (trajno ali nakljućno) lahko detektiramo s kodo C, če ta napaka ni spremenila dane kodne besede xcC v drugo kodno besedo ycC, pri čemer y ≠x.

Nekatere fizične napake v LSI ali VLSI so vzrok za pojav istoznačnih napak. To so napake, ki vplivajo na dekodiranje adres, na posamezne besede, napajanje, vezja za pisanje in čitanje itd [PRA80]. Znano je na primer, da posamezne stuck-at napake, napake zaradi napačnih povezav (cross-point napake) ali napake zaradi kratkih stikov v PLA vezjih povzročijo na izhodih samo istoznačne napake. Poleg tega so tudi skupinske (burst) napake, ki so posledica napake v določenem elementu pomnilnika, največkrat istoznačne.

Nakljućne napake, po drugi strani, so popolnoma slućajne in posamezne. Nakljućnih napak je po številu mnogo manj kot pa istoznačnih. Zato bomo posebno pozornost posvetili učinkovitemu odkrivanju istoznačnih napak.

Zaščito pred napakami dosežemo z odkrivanjem in popravljanjem napak. Odkrivanje napak je edini način kontrole napak pri istoznačnih napakah. To pa zato, ker napake niso nujno omejene in jih zato ni lahko popraviti.

Po drugi strani pa je korekcija primernej-

ša za odpravljanje naključnih napak, saj so te napake redke in posledica vplivov okolice ter zato prehodnega značaja. Največkrat zadoščuje, da za odpravo naključne napake ponovimo postopek vpisa pravilne informacije.

Za odkrivanje istoznačnih napak poznamo različne vrste kodov kot na primer:

m-od-n kod,
dvotirni kod (two-rail code), kjer duplicirane bite invertiramo in jih primerjamo v TSC dvotirnem vezju.

Dokazano je bilo [WAK78], da tudi

"ceneni" AN kodi (neseparacijski kodi, kjer nekodiran podatek x pomnožimo z osnovo A in tako tvorimo kodno besedo [WAK78]) in
inverzni residualni kodi z dolžino grupe m+1 lahko detektirajo vse istoznačne napake dolžine, ki je manjša ali enaka m.

1.2. TSC testna vezja

Pri večini digitalnih sistemov je neekonomično doseči večjo zanesljivost s povečanjem povprečnega časa med napakami. Bolj ekonomičen pristop je tisti, kjer z nekaj dodatnega vezja povečamo "resničnost" izhodov računalniškega sistema. Take računalnike imenujemo samo-testni računalniki (self-checking computer) saj računalnik vedno da pravilni rezultat ali pa javi uporabniku, da je prišlo do napake in, da rezultat ni zanesljiv. Vezje z danim vhodnim kodom in izhodom, ki je sposobno popolnoma avtomatsko odkrivati napake pravimo, da je TSC, če je hkrati sposobno

- avtomatskega testiranja samega testnega vezja in
- odkrivanja napak v kodni besedi, ki vstopa v testno vezje.

Obdelali bomo samotestna ali tako imenovana TSC vezja (totaly self checking) in podali realizacijo takega vezja za Bergerov kod.

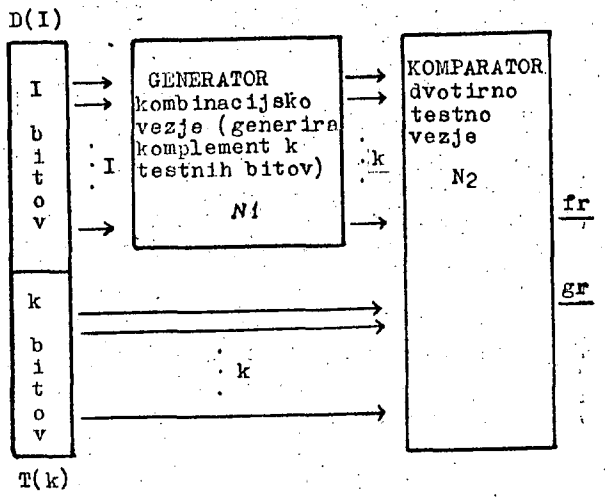
Prednost TSC testnih naprav je v tem, da:

- odkrijejo trenutne ali naključne napake,
- odkrijejo trajne napake,
- diagnostični programi niso več potrebni ali pa so zelo enostavni,
- omogoča hitro testiranje.

TSC testne naprave je enostavno testirati saj potrebujejo majhno podmnožico kodnih besed za vhodni test. Ko samotestno vezje zazna napako, je treba izvesti diagnostični test. Z diagnostičnim testom preverimo, če je napaka res posledica trajne napake (bodisi v vhodnem kodu ali v samem testnem vezju) in ne trenutne napake. Poleg tega je treba preveriti, da se ne pojavi napaka na testnem vezju, ko ga vključimo ali izključimo iz sistema. Zato je testna naprava, ki omogoča lahko in hitro testiranje še posebej zaželjena saj je sistem, ki vsebuje tako testno vezje hitrejši.

Zaželjene so testne naprave, ki imajo čim manj hardvera. Glavna prednost testne naprave, ki ima *malo hardwera je majhna verjetnost, da* pride do napake v sami testni napravi. Prav Bergerov kod je tozadevno primernejši kot pa drugi kodi, pri katerih testne naprave delujejo na principu dupliciranja in primerjave.

Splošna struktura TSC testne naprave za Bergerov kod je prikazana na Sliki 1.



Slika 1.: Struktura TSC testne naprave za separacijske kode.

Vezje N1 imenujemo generator, vezje N2 pa komparator. Kombinacijsko vezje N1 ima za vhod I informacijskih bitov D(I) in generira na k izhodih binarni komplement testnih bitov T(k)*. *Pravimo, da generator N1 generira težo informacijskega dela D kodne besede.* Pri pogojih, ko ni bilo napake, je izhod generatorskega vezja T(k)* enak komplementu testnih bitov v kodni besedi T(k). Dvotirno testno vezje N2 primerja k testnih bitov T(k) z izhodom iz N1 T(k)*. Strukturo testne naprave za ločljive kode, ki jo kaže slika 1 bomo imenovali TSC testna naprava [MAR78].

1.3. Separacijski kodi

Separacijski kodi so tiste vrste kodov, kjer vsebuje kodna beseda dva dela in sicer del z informacijskimi biti I in del s kodnimi biti k. Tak kod je sistematičen kod. Aritmetične operacije pri separacijskih kodih se za informacijske bite in testne bite izvajajo ločeno.

Naj bo C kodna beseda, ki vstopa v testno vezje N. Ko imamo dan kod C, je naloga testnega vezja N preveriti binarno kombinacijo vhodnih bitov in ugotoviti ali je vhod veljavni kod C ali ne. Izhod testnega vezja je 01 ali 10, če je vhod kodna beseda in 00 ali 11, če vhod v testno vezje ni kodna beseda. Pri pravilnem delovanju so vhodi v testno vezje N kodne besede C [ASH76].

Primer realizacije takega testnega vezja v obliki dvonivojskega AND-OR vezja kaže slika 2. Pri normalnem delovanju so vhodi v dvotirni komparator

C = { 0101,0110,1001,1010 }.

Samo za kodne besede iz množice C je izhod iz komparatorja 01 ali 10, za vse druge besede, ki niso v množici C pa je izhod 00 ali 11.

Primer 1:

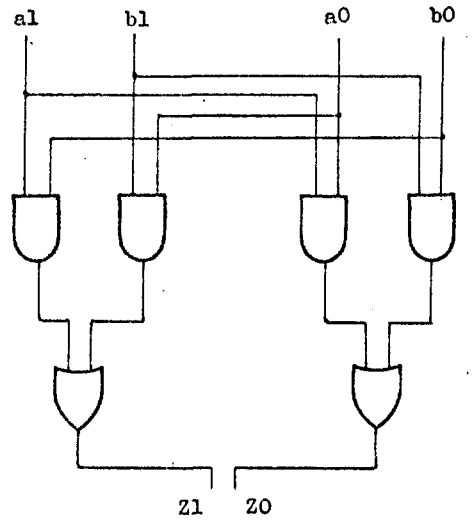
C1 = {0011, 0110, 1001, 1100}

je popolnoma separabilen kod (I=2, k=2; binarnih k-teric je 2exp(2)=4 in to so 11, 10, 01,

00) in

C2 = {0010, 0101, 1001, 1100}

je nepopolno razdružljiv kod, ker se ista binarna k-terica pojavi v testnem delu druge in tretje kodne besede in manjka k-terica 11.



Slika 2.: Primer TSC dvotirnega vezja za k=2 (k je število vhodnih parov).

2. BERGEROV KOD

Beseda zapisana v Bergerovem kodu vsebuje (Slika 3.):

- informacijski del D(I) in
- testni del besede T(k) ter ima

n = število bitov v besedi
I = število bitov informacije
k = število testnih bitov

pri čemer velja

$k = \lceil \log_2 (I+1) \rceil$ in

$n = I + k.$

Vrednost k je najmanjše celo število, ki je večje ali enako vrednosti v oglatem oklepaju.

Naj bo

I1 število enic v informacijskih bitih in
I0 število ničel v informacijskih bitih.

ter velja

I = I1 + I0.

Besedo zapisano v Bergerovem kodu tvorimo na naslednji način:

tvorimo binarno število, ki je enako številu enic v I informacijskih bitih in ga komplementiramo. Rezultirajoče binarno število predstavlja k testnih bitov (ali binarni zapis števila ničel v informacijskih bitih). To pomeni

T = I0 ali

$T = (2^k - 1) - I1$

C (kodna beseda)	
D(I)	T(k)
informacijski biti	testni biti

Slika 3.: Struktura Bergerovega koda.

Poglejmo si nekaj primerov besed zapisanih v Bergerovem kodu

I	informacijski biti	testni biti	k
2	10	10	2
3	101	01	2
4	1011	100	3
5	00011	101	3
6	111111	001	3
7	0000000	111	3
8	00001111	1011	4
12	111111111000	0110	4

Če nas zanima uporaba Bergerovega koda pri mikroračunalniku, kjer je informacija dolga 8, 16 ali 32 bitov, poglejmo kakšno je potrebno število testnih bitov:

št. informacijskih bitov	št. testnih bitov	celotno št. bitov
8	4	12
16	5	21
32	6	38

Če pa zahtevamo, da je celotno število bitov omejeno na 8, 16 ali 32, pa ustrezno spremenimo število informacijskih in testnih bitov.

št. informacijskih bitov	št. testnih bitov	celotno št. bitov
5	3	8
12	4	16
27	5	32

Prednosti Bergerovega koda so naslednje:

1. Ločljivost koda: ko moramo obdelati informacijo ne potrebujemo za razpoznavanje informacijskih bitov v kodni besedi nobenega dodatnega dekoderja.

2. Odkrije vse istoznačne napake: Najverjetneje so te napake v digitalnih sistemih, še posebej v tistih, ki vsebujejo LSI in VLSI integrirana vezja. Če pride do enkratne ali večkratne istoznačne napake v informacijskem delu kodne besede, se tam število enic in ničel spremeni. S testnimi biti takoj ugotovimo to napako.

Primer 2:

prvotna kodna beseda

I	k
1111111110001	0110

enkratna istoznačna napaka v I

I	k
0111111110001	0110

k*=0111

Vidimo, da se k in k* ne ujemata. Odkrili smo istoznačno napako v I, napake pa seveda ne moremo lokalizirati.

3. Je optimalen: Od vseh ločljivih kodov, ki odkrijejo istoznačne napake porabimo pri Bergerovem kodu za dano število I informacijskih bitov, najmanjše število testnih bitov k. (Nekateri drugi kodi n. pr. m-od-n kod potrebujejo sicer manj testnih bitov, toda ti kodi niso razdružljivi.)

Definicija 7: Naj bo C Bergerov kod. C je Bergerov kod maksimalne dolžine, če je število informacijskih bitov I enako

$$I = 2^k - 1 \qquad k>=1$$

V tem primeru dobimo isti kod, če uporabimo za tvorbo testnih bitov I0 ali I1.

Tako so Bergerovi kodi maksimalne dolžine kodi s sledečim številom informacijskih bitov:

Bergerov kod maksimalne dolžine	
št. testnih bitov	št. informacijskih bitov
2	3
3	7
4	15
5	31
6	63

Zapišimo Bergerove kode maksimalne dolžine za I=3, k=2

C = { 00011,
00110,
01010,
10010,
01101,
10101,
11001,
11100 }.

(Prvi trije biti v vsaki kodni besedi so informacijski, zadnja dva pa testna.)

Če C ni Bergerov kod maksimalne dolžine, ga imenujemo Bergerov kod nemaksimalne dolžine.

Lema 1: Naj bo C Bergerov kod. C je popolnoma separacijski kod, če in samo če je C Bergerov kod maksimalne dolžine.

Testno vezje tipa 1 je TSC testno vezje za Bergerov kod maksimalne dolžine.

3. TSC TESTNO VEZJE ZA BERGEROV KOD

Opisali bomo TSC vezje za Bergerov kod, ki je sposobno odkrivati istoznačne napake. Generatorsko vezje N1 (Slika 1.) tvori binarno število, ki ustreza številu enic v I informacijskih bitih. Generator N1 je sestavljen iz popolnih in poloviČnih seštevalnih modulov, ki paralelno seštevajo informacijske bite. Vezje N1 je sposobno samo sebe testirati na enojne in večkratne napake. Vezje je sposobno odkrivati enojne napake, ker bi pojav napake povzročil neveljavne kodne besede na vhodu v N2, kar ima za posledico javljanje napake na izhodu.

Komparatorsko vezje N2 je TSC dvotirno testno vezje. N2 je narejen tako, da odkrije vse enojne napake.

3.1. Testno vezje za Bergerov kod maksimalne dolžine.

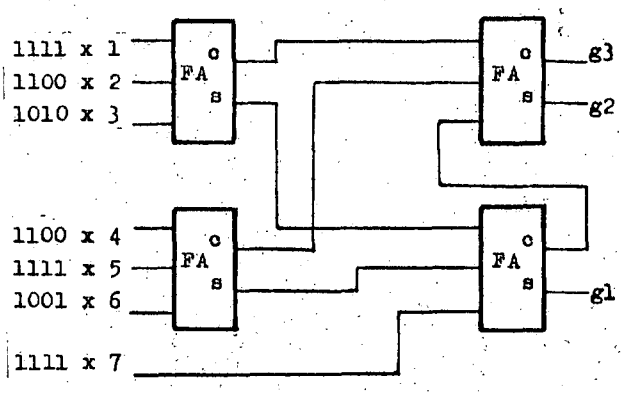
Bergerov kod z $I=2^k-1$ imenujemo Bergerov kod z maksimalno dolžino. Vezje N1 (Slika 1.) generira binarno število, ki ustreza številu enic v I informacijskih bitih. Vezje je sestavljeno iz množice seštevalnih modulov, ki paralelno seštevajo informacijske bite. Dano vezje potrebuje le osem testov za detekcijo vseh enojnih napak v vezju in vseh večkratnih napak, ki se pojavijo v posameznem seštevalnem modulu. Naj bo

$$g_k, g_{k-1}, \ldots, g_1$$

binarno število, ki ustreza številu enic v I informacijskih bitih

$$x_1, x_2, \ldots, x_{2\exp(k)-1}.$$

Za Bergerov kod z I=3, k=2 je vezje N1 že posamezen seštevalni modul, kjer je S enak sumi S=g1 in izhod carry enak C=g2. Slika 4 kaže vezje N1 za primer I=7, k=3. Vezje N1 za primer I=15, k=4, je sestavljeno iz treh skupin vezij D1, D2 in D3. Vezji D1 in D2 ustrezata vezju N1 za I=7, k=3. Vezje D3 pa je "ripple carry" seštevalnik, ki generira vektor (g4, g3, g2, g1).



Slika 4.: Generatorsko vezje N1 za Bergerov kod maksimalne dolžine za primer I=7, k=3.

Poglejmo postopek, ki podaja metodo za generiranje vezja N1.

POSTOPEK 1 [MAR78]

1. Naj bo

$$S = \{x_1, x_2, \ldots, x_{2^{exp(k)-1}}\}$$

množica vseh informacijskih bitov in je

$m=k$, $I=1$.

2. Razdelimo množico S v tri podmnožice A^I, B^I in E^I tako, da

A^I sestoji iz levih $(2^{m-1}-1)$ bitov,

B^I sestoji iz naslednjih $(2^{m-1}-1)$ bitov in

E^I vsebuje najbolj desni bit.

Naj bo binarna reprezentacija števila enic, ki se nahajajo v informacijskih bitih v podmnožicah Aexp(I), Bexp(I) in Eexp(I) označena z

$$'a^I = (a^I_{m-1}, a^I_{m-2}, \ldots, a^I_1)$$

$$'b^I = (b^I_{m-1}, b^I_{m-2}, \ldots, b^I_1)$$

$$'e^I = e^I$$

3. *Binarna reprezentacija števila enic, ki se pojavi v S*

$$'g^I = (g^I_m, g^I_{m-1}, \ldots, g^I_1)$$

je podana z naslednjo vsoto:

$$\begin{array}{l} a^I_{m-1}\ a^I_{m-2}\ \ldots\ a^I_1 \\ +\ b^I_{m-1}\ b^I_{m-2}\ \ldots\ b^I_1 \\ \quad +\ e^I \\ \hline g^I_m\ g^I_{m-1}\ g^I_{m-2}\ \ldots\ g^I_1 \end{array}$$

Za tvorbo vektorja $'g^I$ uporabimo "ripple carry" seštevalnik.

4. Če je m>2 potem

i) m = m-1
   L = I

ii) $S = A^I$
    I = I+1

in ponovi korak 2 in 3, da dobimo vektor

$$'a^L = 'g^I .$$

Vektor 'b exp(L) tvorimo z vezjem, ki je identično tistemu, ki generira 'a exp(L).

5. Konec postopka 1

To je rekurziven postopek. Vezje, ki ga generiramo s tem postopkom ima strukturo binarnega drevesa. Vsako vozlišče drevesa predstavlja ripple carry seštevalnik. Vozlišče na i-tem nivoju drevesa $1=<i=<(k-1)$ predstavlja ripple carry seštevalnik z i stopnjami. Drevo ima natanko

$$\sum_{j=1}^{k-1} 2^{j-1}(k-j)$$

popolnih seštevalskih modulov [MAR78] in ima zakasnitev največ

$$(2k - 3)$$

modulov. Struktura drevesa za N1, ki ga generira postopek 1 potrebuje le osem vektorjev za svoje testiranje.

3.2. Testno vezje za Bergerov kod nemaksimalne dolžine.

Za Bergerov kod nemaksimalne dolžine velja $I \neq 2^{exp(k)-1}$. V tem primeru je vezje sestavljeno iz popolnih seštevalskih modulov in polovičnih seštevalskih modulov. Za dano vrednost I ter k, kjer velja

$$k = \lceil \log_2 (I+1) \rceil$$

potrebujemo za detekcijo vseh enojnih napak v vezju N1 samo 8(k-1) kodnih besed. Komparatorsko vezje N2 zgradimo v obliki drevesa iz komparatorskih modulov z dvema vhodnima paroma. Za detekcijo vseh enojnih napak v vezju N2 potrebujemo le pet kodnih besed. Torej lahko vse enojne napake v testnem vezju odkrijemo z [8(k-1)+5] kodnimi besedami. Za realizacijo *testnega vezja potrebujemo torej manj kot* (12I+6logk) (log z osnovo 2) vrat [MAR78].

Generatorsko vezje N1

Zaradi večje jasnosti bomo generatorsko vezje N1 najprej predstavili s primerom in šele nato podali postopek za njegovo realizacijo.

Primer 4: Poglejmo Bergerov kod z I=12 in k=4. Naj bo X' množica informacijskih bitov

$$X' = (x1, x2, \ldots x12).$$

Generatorsko vezje N1 sestavimo na sledeč način:

1. Razdelimo množico X' v tri podmnožice A1, A2 in A3. Posamezne podmnožice naj imajo

A1: 7 = (2exp(3)-1) bitov
A2: 3 = (2exp(2)-1) bitov
A3: 2 ostanek bitov

2. Naj bo N1(A1) generatorsko vezje katerega vhodi so informacijski biti podmnožice A1 to je {x1,x2,...,x7} in katerega izhodi so binarna števila, ki ustrezajo številu enic v A1, to je g3(1), g2(1), g1(1). N1(A1) je opisan v postopku 1.

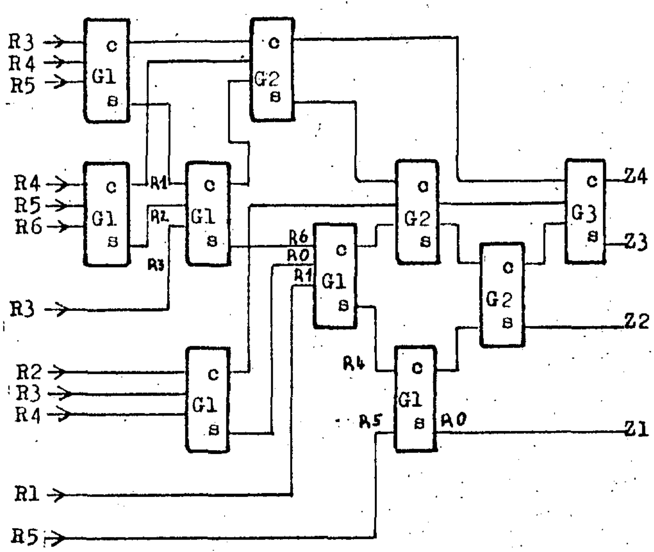
Naj bo N1(A2) vezje katerega vhodi so biti podmnožice A2 in katerega izhodi so g2(2), g1(2).

N1(A3)=N1(2) je že modul s pololnim seštevalnikom.

3. Binarno število Z4, Z3, Z2, Z1, ki ustreza številu enic, ki se pojavijo v skupini 12-ih informacijskih bitov lahko dobimo s sledečim postopkom seštevanja:

```
izhodi N1(A1)     g3(1)          g2(1)        g1(1)
izhodi N1(A2)                    g2(2)    M1  g1(2)
                           M3                 x11
              M5                 c1           x12
                                          M2  S1
                  c3             c2
                  c4       M4    S3
----------------------------------------------------
       c5=Z4      s5=Z3          s4=Z2        s2=Z1
```

4. Naj bo N1' vezje, ki je sestavljeno iz modulov s popolnimi seštevalniki in modulov s polovičnimi seštevalniki, ki opravlja opisan postopek seštevanja. Načrt vezja N1 kaže slika 5.



Slika 5.: Načrt testnega vezja N1 za Bergerov kod nemaksimalne dolžine za I=12, k=4.

POSTOPEK 2:

1. Razdelimo I informacijskih bitov v skupini A1, A2,..., An tako, da je n=<k-1. Število bitov v skupini Ai naj bo 2exp(k-i)-1; 1=<i=<k-2. Zadnja skupina An ima lahko 0, 1 ali 2 bita, odvisno od vrednosti razlike

$$I - \sum_{i=1}^{n-1} A_i$$

2. Za vse Ai, i=1,2,...,n-1 naj bo N1(Ai) generatorsko vezje, ki ga opisuje postopek 1, ki generira binarno število gm(i),...,g2(i),g1(i) in, ki ustreza številu enic, ki se pojavijo v bitih podmnožice Ai.

3. Tvorimo postopek seštevanja, kot smo ga pokazali v primeru 4 za izhode iz vseh N1(Ai) 1=<i=<n-1 in bitov iz skupine An. Rezultat seštevanja je binarno število Z(k), Z(k-1),...Z1, ki ustreza številu enic, ki se pojavijo v I informacijskih bitih.

4. Naj bo N1' vezje, ki sestoji iz modulov s popolnimi seštevalniki in modulov s polovičnimi seštevalniki, ki izvrši korake seštevanja, ki smo jih določili v postopku seštevanja v prejšnji točki.

4. ZAKLJUČEK

Obravnavali smo Bergerov kod in TSC vezje za Bergerov kod. Pokazali smo kako lahko tako vezje konstruiramo ali pa formiramo kod, ki je ekvivalenten Bergerovemu kodu in za katerega lahko tvorimo TSC vezje.

Vsi kodi za katere smo obravnavali TSC testna vezja so popolnoma razdružljivi kodi. Sistematično načrtovanje TSC testnih vezij za nepopolno razdružljive kode je težavno.

Bergerov kod je sicer bolj redundanten kot m-od-n kod. Za velike vrednosti n potrebuje Bergerov kod dvakrat toliko testnih bitov kot jih potrebuje (n/2)-od-n kod. Prednost Bergerovega koda pred m-od-n kodom je njegova razdružljivost, ki omogoča uporabo manjšega in enostavnejšega vezja za kodiranje informacijskih bitov v Bergerov kod in iz njega.

Dodatni motiv za uporabo Bergerovega koda je vzrok, da imajo številni LSI elementi istoznačne napake in Bergerov kod je optimalen ločljiv kod, ki detektira vse istoznačne napake [ASH76]. Pomembno področje uporabe Bergerovega koda je dinamično testiranje pomnilnikov [ASH76]. Na primer serija IMP-16 mikroračunalnikov, ki jih proizvaja National Semiconductor, uporablja za kontrolno enoto pomnilni(ki element, ki je 23-bitni ROM. Tako kontrolno enoto lahko dinamično testiramo z uporabo Bergerovega koda. Uporaba kakšnega drugega koda (m-od-n na primer) zahteva zaradi nerazdružljivosti uporabo decoderja, ki povzroči dodatno zakasnitev, poveča število modulov v računalniku in odpira nov problem testiranja izhoda samega dekoderja. V drugem delu si bomo ogledali lastnosti modificiranega Bergerovega koda in testiranje PLA.

5. LITERATURA

[ASH76] M.J.Ashjaee in S.M.Reddy, On Totally-Self-Checking Checkers for Separable Codes, Int. Symp. on Fault-Tolerant Computing, pp.151-156, 1976.

[BOS82] B. Bose in T. Rao, Theory of Unidirectional Errors Correcting/Detecting Codes, IEEE Trans. on Comp., vol. C-31, pp.521-530, June 1982.

[CAR80] W.C.Carter in A.B. Wadia, Design and Analysis of Codes and their Self-Checking Circuit Implementation for Correction and Detection of Multiple b-Adjacent Errors, IEEE, pp. 35-40, 1980.

[DON82] H.Dong, Modified Berger Codes for Detection of Unidirectional Errors, IEEE, pp 317-320, 1982.

[DON82] H.Dong in E.J. McCluskey, Concurrent testing of programmable logic arrays, CRC Technical Rep. No. 82-11, Stanford Univ., California, junij 1982.

[MAR78] M.A. Marouf in A.D. Friedman, Design of Self-Checking Checkers for Berger Codes, The Eight Annula Int. Conf. on Fault-Tolerant Computing, pp.179-184, June 1978.

[PRA80] D.K. Pradhan, A New Class of Error-Correcting/Detecting Codes for Fault-Tolerant Computer Applications, IEEE Trans. on Comp., vol. C-29, pp. 471-481, June 1980.

[WAK78] J. Wakerly, Error Detecting Codes, Self-Checking Circuits and Applications, Elsevier North-Holland,Inc. The Computer Science Library, 1978.

INFORMATICA 4/84

ALGEBRA KLJUČEVA

SINIŠA J. DJORDJEVIĆ

UDK: 512.5:681.3

Ključevi se mogu interpretirati i preko skupova čiji su elementi ključevi, fizičke adrese organizacionih jedinica podataka ili same organizacione jedinice podataka. Na osnovu toga, u radu je prikazana algebarska interpretacija polja kao organizacionih jedinica podataka, memorisanja polja i formiranja ključeva.

KEYS ALGEBRA: Keys can be interpreted by sets of keys, sets of physical addresses of data organization units or sets of data organization units. For that reason, this paper presents interpretation of fields storing and keys forming by algebra.

1. Uvod

Pod algebrom ključeva podrazumevaće se interpretiranje ključeva preko skupova čiji su elementi ili ključevi ili fizičke adrese organizacionih jedinica podataka ili same organizacione jedinice podataka. Svi odnosi u organizaciji podataka mogu se predstaviti relacijama i preslikavanjima u algebri ako se atributi organizacije podataka pravilno organizuju u skupove.

Algebarska, matematička, interpretacija organizacije podataka prikazana je u /4/ gde se ključ tretira samo kao organizacioni element što znači da se posmatra jedino kao veza izmedju organizacionih jedinica podataka. Odnos izmedju ključa i sadržaja organizacionih jedinica podataka zanemaren je jer samo posredno utiče na organizaciju podataka, na fizički razmeštaj organizacionih jedinica podataka. Ovaj odnos definiše samo formalni redosled organizacionih jedinica podataka koji ima značaja za pristupe i algoritme obrade podataka ali je moguće da nema nikakvog uticaja na fizički razmeštaj podataka. Na primer, sa jednim istim ključem, i istim odnosom ključ -sadržaj organizacione jedinice podataka, može se pristupiti različitim organizacijama podataka pa se mogu upotrebiti i indeks sekvencijalne i direktne i indirektne datoteke.

U ovom tekstu posmatraće se upravo taj odnos, odnos ključa i sadržaja organizacione jedinice podataka za koju se definiše odredjeni ključ s ciljem da se taj odnos algebarski interpretira.

Za praćenje daljeg izlaganja potreban je najosnovniji kurs iz algebre koji se može naći u /8/ . Za upotrebljene pojmove iz algebre korišćene su definicije iz /8/ .

2. POLJA KAO ORGANIZACIONE JEDINICE PODATAKA

Definicija 1 : Polja kao organizacione jedinice podataka predstavljaju binarnu relaciju u skupu koji je Dekartov proizvod skupa sadržaja i skupa ključeva.

Skup ključeva je skup koji se na odredjeni način formira. Skup sadržaja su podaci na osnovu kojih se organizuje informacioni sistem. Polje se uzima u širem smislu te predstavlja bilo koju organizacionu jedinicu podataka koja je nezavisna u okviru odredjenog informacionog sistema. Ovako definisano polje postaje sloj u organizovanju podataka preko datoteka. Nezavisne organizacione jedinice podataka mogu biti i bez ključeva što nije izvan definicije 1, ako je skup ključeva prazan skup onda je Dekartov proizvod skup sadržaja.

Elementi Dekartovog proizvoda su uredjeni parovi pa je redosled skupova u definiciji jedan od atributa definicije. Ovi uredjeni parovi u organizaciji podataka često se inverzno predstavljaju što nema značaja za definiciju 1.

Na osnovu definicije 1 jedan sadržaj može imati više ključeva i to više bilo primarnih

bilo sekundarnih ključeva. Prema /5/, primarni ključ je ključ koji je upotrebljen za definisanje formalnog fizičkog redosleda polja u odnosu na dodeljivanje organizacionih jedinica ćelijama u memoriji. Jedna organizacija podataka može imati samo jedan formalni fizički redosled. Takodje prema /5/, sekundarni ključevi su relacije sadržaja (podatka) sa ključevima na osnovu kojih se definišu formalni (fizički ne mogu biti) redosledi na osnovu kojih se omogućuju realizacije odredjenih zahteva u traženju i pretraživanju. Definicijom 1 obuhvaćeni su i jedni i drugi ključevi. Ako jedan sadržaj ima više primarnih ključeva to znači da se jedan podatak memoriše više puta. Ovakve organizacije podataka vrlo su retke te se neće uzimati u obzir njihov uticaj. U kontekstu formiranja ključeva sekundarni ključevi se mogu tretirati kao podaci. Zajedno sa podacima mogu se uzimati kao jedan sadržaj, ovo je opravdano jer se sekundarni ključevi i formiraju tako što se sadržaj "cepa" na sadržaje koji će biti upotrebljeni za definisanje formalnih redosleda. Prema tome, u definiciji 1, pojam "ključ" podrazumevaće primarni ključ a i u daljem tekstu ako drukčije nije napisano važiće isto što i u definiciji 1.

Ako se zanemare preopširne organizacije podataka, jedan sadržaj je memorisan više puta, onda polja postaju preslikavanje (funkcija) sadržaja na ključeve.

Ako dva sadržaja mogu da dobiju iste ključeve onda je znatno otežano dodeljivanje ćelijama memorije i javljaju se mnogi problemi oko traženja i pretraživanja a da se ne dobijaju nikakva organizacijska poboljšanja.

Može se zaključiti da sa stanovišta organizacije podataka polja predstavljaju biunivoko jednoznačno preslikavanje sadržaja (organizacionih jedinica) na ključeve.

Ovaj zaključak ne može da predstavlja definiciju jer predstavlja posebne uslove u odnosu na definiciju 1.

3. MEMORISANJE POLJA

Memorisanje polja predstavlja dodeljivanje polja kao organizacionih jedinica ćelijama memorije. Svako polje dobija svoju adresu na osnovu koje se identifikuje njegov položaj u memoriji.

Definicija 2 : Memorisana polja u odnosu na svoje adrese predstavljaju binarnu relaciju u Dekartovom proizvodu skupa polja i skupa adresa.

Ako je polje memorisano ono mora da ima svoju adresu. Ako se jedno polje memoriše samo jednom relacija iz definicije 2 postaje preslikavanje. Ovo preslikavanje je biunivoko jednoznačno jer dva različita, nezavisna polja ne mogu da imaju istu fizičku adresu.

Na osnovu definicija 1 i 2 zaključuje se da je memorisanje proizvod (kompozicija) preslikavanja sadržaja na adrese. Ako je x odredjeni sadržaj, f(x) polje onda je F(f(x)) adresa sadržaja ali i polja u memoriji. Kada se sadržaju ne dodeljuje ključ onda se preslikavanje f(x) može zanemariti pa je adresa $F_1(x)$ $F_1(x)$ je biunivoko jednoznačno preslikavanje ako se jedan sadržaj memoriše samo jednom.

Može se desiti da je $F(f(x_1))=F(f(x_2))$, na primer kod indirektne datoteke, ali su to specijalni slučajevi gde se preslikavanje F definiše ne samo na osnovu sadržaja x već i na osnovu pretraživanja kao relacije, opisano u /5/, pa se tim dodefinisanjem preslikavanje F ostaje biunivoko jednoznačno. Preslikavanje ključ-adresa nije biunivoko jednoznačno ali je preslikavanje F(f(x)) biunivoko jednoznačno jer sadržaji imaju različite stvarne adrese a do njih se dolazi složenijim postupkom.

4. FORMIRANJE KLJUČEVA

Ključevi se formiraju iz dva razloga. Prvi je predstavljanje složenih sadržaja kodovima (skraćivanje) čime se omogućuje efikasnije čuvanje i jednostavnija komunikacija podataka dok je drugi definisan organizacijom računara kao medijuma obuhvata podataka. Datoteka kao oblik organizacije podataka predpostavlja ključeve zbog univerzalnosti rešavanja problema memorisanja složenih sadržaja. Ovi složeni sadržaji predstavljaju se jednobrazno (slogovima) a kodiraju se ključevima radi jednostavnije i brže manipulativnosti. Kod banki podataka (savremeni oblik organizovanja podataka) ključevi gube svojstvo koda, preslikavanja složenih sadržaja, jer se od jednostavnih sadržaja formiraju organizacione jedinice, ali zadržavaju svoju ulogu u manipulativnosti sa podacima.

Prema /2/ tri osnovna zadatka ključeva su identifikacionost, klasifikacionost i pružanje informacija. Ako se ključevi shvate kao slike kod preslikavanja sadržaja onda se svaki od ovih zadataka ključeva može jednostavnije interpretirati.

Identifikacionost podrazumeva biunivoko jednoznačno preslikavanje sadržaja na ključ čime se svaki sadržaj može na osnovu ključa da identifikuje. Kod identifikacije skup ključeva je skup elemenata koji su nezavisni, koji nisu ni u kakvoj vezi sa sadržajem.

Kod preslikavanja složenih sadržaja pribegava se preslikavanju takvom da su slike preslikavanja delovi sadržaja koji se preslikava. Od skupa složenih sadržaja formira se i skup uvek istih delova tog sadržaja na koji se preslikavaju ovi složeni sadržaji i to tako da se svaki složeni sadržaj preslikava na svoj deo. Ovi delovi biraju se tako da u okviru sprege sadržaja sa informacionim sistemom doprinose najefikasnijoj interakciji. Ovakvim preslikavanjem ključevi postaju klasifikacioni ili informacioni. Ako se jedan od delova složenog sadržaja može predstaviti kao stablo (specijalni graf) onda se to stablo, njegovi čvorovi mogu ali i ne moraju da se kodiraju, često upotrebljava kao ključ jer je efikasno za spregu sadržaja sa informacionim sistemom i takav ključ je klasifikacioni ključ. Kod informacionog ključa jeden deo (ili njegov kod) sadržaja postaje ključ.

Praktično su definisana tri načina formiranja ključeva koji se mogu proizvoljno kombinovati.

Složeni sadržaj predstavlja uredjenu n-torku $(x_{1,i},x_{2,i}, \dots ,x_{n,i})$ gde su $x_{j,i}$ nezavisni pojedinačni sadržaji. U terminologiji informacionih sistema često se memorisana n-torka označava kao slog (rečenica) a pojedinačni nezavisni sadrzaji se označavaju kao polja. U /2/ se daju tri tipa definicija za organizacione jedinice ali se sa tim tipovima definisanja pokušava da se organizacione jedinice podataka definišu i sa aspekta organizovanja podataka i sa aspekta organizovanja informacionih sistema. U ovom tekstu se te dve stvari odvajaju pa se organizovanje podataka posmatra nezavisno od informacionog sistema. Razlog za to je što je organizovanje podataka nezavisna promenljiva u odnosu na informacioni sistem odnosno u jednom istom informacionom sistemu mogu biti primenjene različite organizacije podataka.

Datoteka u ovom kontekstu postaje n-arna relacija u skupu $X_1xX_2x...xX_jx...xX_n$ (Dekartov proizvod), kao i svaki skup sadžaja, sa aspekta sadržaja slogova datoteke. Prema tome, polje postaje preslikavanje n-torke na ključ.

Uredjeni par:

$$((x_{1,i},x_{2,i}, \dots ,x_{n,i}),y) \qquad \dots (1)$$

je polje dok je y (slika preslikavanja) ključ i može predstavljati element proizvoljnog skupa pa. Ako je:

$$y = (x_{j1,i},x_{j2,i}, \dots ,x_{jk,i}) \qquad \dots (2)$$

onda je u pitanju formiranje ključa od složenog sadržaja preko odabranih sadržaja koji su neki delovi tog složenog sadržaja.

Ako je:

$$y = (g_1(x_{j1,i}),g_2(x_{j2,i}), \dots ,g_k(x_{jk,i})) \dots (3)$$

onda je u pitanju složeni ključ gde se svakom odabranom sadržaju (delu) posebno dodeljuje ključ a onda se od tih ključeva formira ključ polja. Sada se i klasifikacioni i informacioni ključevi mogu predstaviti sa (2) odnosno sa (3). Za klasifikacione ključeve potrebne je da izmedju delova sadržaja $(x_{j1,i},x_{j2,i}, \dots ,x_{jk,i})$ postoje interne relacije. I oblici složenog organizovanja ključeva (na primer paralelno šifriranje opisano u /2/) jednostavno se mogu predstaviti sa (1) preko (2) i (3).

Preko (1), (2) i (3) potpuno se formalizuje generisanje ključeva čime se pojednostavljuje formiranje svih oblika složenih ključeva.

U formiranju ključeva preko (1), (2) i (3) jednostavno se formalizuje prilagodjavanje ključeva efikasnim traženjima i pretraživanjima. Prema /5/ traženje i pretraživanje interpretiraju se kao preslikavanje i relacija, respektivno, pa se sa (1), (2) i (3) formiranje ključeva lako povezuje sa traženjem i pretraživanjem.

5. ZAKLJUČAK

Osobine ključeva, navedene u /3/ , mogu se preko (1), (2) i (3) jednostavno algebarski interpretirati kao i kompresija ključeva opisana u /1/.

Jednakost ključ-podatak, opisana u /7/, može se predstaviti uredjenim parom :

$$(x_{1,i},x_{1,i}) \qquad \dots (4)$$

što znači da su argument (sadržaj) i slika preslikavanja (ključ) jednaki.

Zaštita ključeva opisana u /6/ postaje kompozicija preslikavanja (složeno preslikavanje)sadržaja na ključ koji se potom preslikava na zaštićeni ključ. Informacionost zaštićenog ključa je binarna relacija u Dekartovom proizvodu skupa polja i skupa informacionih ključeva.

U algebarskoj interpretaciji ključeva nije od značaja da li se uredjeni parovi memorišu u celini ili je neki od elemenata uredjenog para na bilo koji način sadržan u onom drugom a ovo je moguće zbog specifičnosti preslikavanja.

Implikacije teksta su algebarske interpretacije koje su same navedene opisom a izostavljene su zato što se jednostavne mogu formulisati.

Algebarskom interpretacijom ključeva izvršena je generalizacija čime se analegijske zaključivanje u vezi sa ključevima može zameniti deduktivnim zaključivanjem.

6. LITERATURA

1/ J. Martin, COMPUTER DATA - BASE ORGANIZATION, second edition, PRENTICE - HALL, ENGLEWOOD CLIFFS, NEW JERSEY 07632.

2/ H. Wedekind, ORGANIZACIJA PODATAKA,ZAK,

3/ V. Ferišak, SISTEMI ŠIFRIRANJA, PRAKSA, decembar 1979, br. 12 .

4/ S. Djordjević, MATEMATIČKA INTERPRETACIJA BAZE PODATAKA, ZBORNIK RADOVA DEVETOG SIMPOZIJUMA JUPITER SISTEMA, Beograd 15 i 16. maja 1979.

5/ S. Djordjević, VIŠEKLJUČNA ORGANIZACIJA PODATAKA, ZBORNIK RADOVA VIII JUGOSLOVENSKOG SAVETOVANJA O INFORMACIONIM SISTEMIMA, Beograd 17. - 18. maja 1979. godine.

6/ S. Djordjević, ZAŠTITA KLJUČEVA I INFORMACIONOST ZAŠTITE KLJUČEVA, ELEKTROTEHNIKA, Novembar-Decembar 1983, br. 6 .

7/ S. Djordjević, JEDNAKOST KLJUČ-PODATAK, PRAKSA, februar 1979, br. 2 .

8/ D. S. Mitrinović, D. Mihailević, P. M. Vasić, LINEARNA ALGEBRA POLINOMI ANALITIČKA GEOMETRIJA, GRADJEVINSKA KNJIGA, Beograd, 1968.

PRINCIPI KVALITATIVNEGA MODELIRANJA

IGOR MOZETIČ

INSTITUT JOŽEF STEFAN, JAMOVA 39, LJUBLJANA

UDK: 681.3:519.682.6

Članek utemeljuje uporabnost kvalitativnega modeliranja, podaja pregled različnih pristopov, njihovo primerjavo in zmožnost vzročne razlage fizikalnih pojavov. Z uporabo teh principov smo razvili kvalitativni model srca. Model simulira električno aktivnost srca in je sposoben za vsako možno kombinacijo okvar v srcu izpeljati ustrezne opise EKG.

PRINCIPLES OF QUALITATIVE MODELING. The paper explains motivation for qualitative modeling, presents and compares various approaches to it and briefly explores their ability to causally explain physical processes. Using this principles we have developed a qualitative model of the heart. The model simulates electrical activity of the heart. Its implementation can derive corresponding ECG descriptions for any possible combination of cardiac arrhythmias.

1. UVOD

V zadnjem času se je na področju umetne inteligence močno povečalo zanimanje za razvoj mehanizmov kvalitativnega sklepanja. Kvalitativno sklepanje naj bi bilo sposobno napovedati obnašanje in podati vzročno razlago delovanja raznih naprav s kvalitativnimi pojmi. Potreba po takem načinu sklepanja izvira na eni strani iz množice potencialnih aplikacij programov, ki so tega zmožni, na drugi strani pa iz osrednje vloge, ki jo ima tak način sklepanja v človekovem razmišljanju.

Programi za vodenje, vzdrževanje in popravljanje kompleksnih sistemov (npr. jedrske elektrarne) morajo razumeti, kaj se v sistemu dogaja. Kajti le če so sposobni pojasnjevati in utemeljevati svoje odločitve, jim bo uporabnik zaupal. Poleg tega je formalen opis takšnih kompleksnih sistemov z metodami klasične fizike (npr. s sistemom diferencialnih enačb) marsikdaj preveč kompliciran, da bi bil uporaben. Pogosti so tudi primeri (npr. v medicini), ko mehanizmi delovanja sploh niso dovolj dobro poznani, da bi jih lahko razložili z znanimi fizikalnimi zakoni (npr. človeški možgani). V našem primeru (električna aktivnost srca) smo se srečali s problemom, ko je fizikalna slika delovanja sicer jasna, vendar pa so parametri fizikalnega modela praktično nemerljivi (npr. hitrosti prevajanja električnih impulzov v različnih delih srca, odvisnost parametrov od posameznega pacienta).

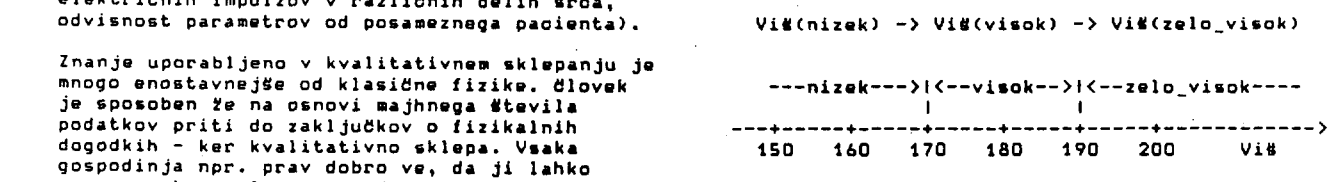
Znanje uporabljeno v kvalitativnem sklepanju je mnogo enostavnejše od klasične fizike. Človek je sposoben že na osnovi majhnega števila podatkov priti do zaključkov o fizikalnih dogodkih - ker kvalitativno sklepa. Vsaka gospodinja npr. prav dobro ve, da ji lahko raznese ekonom-lonec, če je ventil za odvajanje pare zamašen. Pa ji zato ni potrebno pravničesar vedeti o zakonu o ohranitvi toplote ali o natančni temperaturi grelca. Zadošča ji znanje, ki izvira iz vsakodnevnih izkušenj.

Kvalitativno sklepanje je pomemben del našega splošnega znanja. Njegovo razumevanje nam olajšuje razumevanje človekovega načina razmišljanja in omogoča izdelavo "pametnejših" programov.

Članek je sestavljen iz štirih delov. Najprej podamo osnovne pojme kvalitativnega modeliranja in nato na enostavnih primerih orišemo tri različne pristope. Na netrivialnem "realnem" problemu modela srca pokažemo uporabnost kvalitativnega modeliranja in zaključimo članek s primerjavo vseh pristopov.

2. OSNOVI POJMI

V klasični fiziki opisujemo pojave s sistemi enačb. Domena spremenljivk v enačbah je tipično množica realnih števil. Kvalitativna fizika opisuje pojave z relacijami med spremenljivkami. Vendar zavzamejo spremenljivke le majhno število različnih vrednosti. Njihova domena je definirana s t.im. prostorom količin (quantity space) [Forbus 82a], ki razdeli množico realnih števil na intervale. Prostor količin je urejena množica kvalitativnih (simboličnih, verbalnih) opisov vrednosti (primer na sliki 1).



Slika 1: Prostor količin za višino.

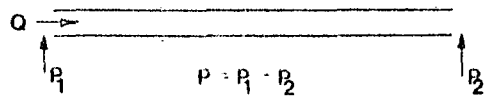
Pomembne spremenljivke v fiziki so časovni odvodi funkcij. V kvalitativni fiziki navadno zavzamejo le tri vrednosti: +1, 0, -1, glede na to, če vrednost funkcije raste, je konstantna ali pada.

Aritmetika nad takimi spremenljivkami je definirana ad-hoc, navadno v obliki tabel. Primer seštevanja odvodov je na sliki 2.

X: / Y:	-1	0	+1
-1	-1	-1	?
0	-1	0	+1
+1	?	+1	+1

Slika 2: Seštevanje odvodov X + Y.

Atomarni fizikalni pojavi so opisani z relacijami med spremenljivkami. Relacije imajo lahko obliko omejitvenih enačb, neenačb ali pa so logične izjave.

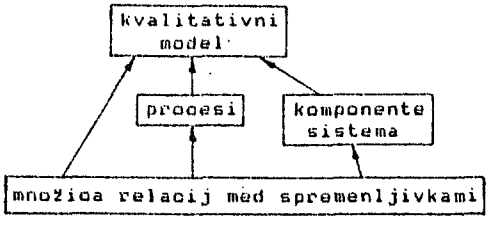


Slika 3: Pretok skozi cev.

Vzemimo primer cevi na sliki 3. Pojav pretoka skozi cev je opisan z omejitveno enačbo: dP = dQ, ki pomeni: vsaka sprememba tlaka vzdolž cevi je sorazmerna spremembi pretoka skozi cev.

Relacije opisujejo fizikalne pojave na jedrnat in razumljiv način. Poleg tega obstajajo zanje popolne metode reševanja. Za reševanje sistemov omejitvenih enačb, ki temelji na zadovoljevanju omejitev (constraints-satisfaction) je to npr. jezik CONLAN (CONstraints LANguage), za logične izjave pa avtomatski dokazovalci izrekov, npr. PROLOG (PROgramming in LOGic). Množico relacij lahko interpretiramo tako proceduralno (lahko jih izvajamo), kot tudi deklarativno.

Z množicami relacij in dodatnimi pogoji opisujemo bolj abstraktne fizikalne pojme kot so npr. procesi ali pa komponente sistema. Prav v tem je temeljna razlika različnih pristopov kvalitativnega modeliranja. Kvalitativni model sistema je lahko opisan z delovanjem komponent sistema, s procesi, ki tečejo v sistemu ali pa celo samo z množico relacij (slika 4).



Slika 4: Shema različnih pristopov opisovanja kvalitativnih modelov.

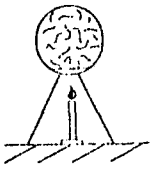
S kvalitativno simulacijo model "oživimo" oziroma dobimo njegovo obnašanje. Za dane vrednosti nekih spremenljivk zadovoljujemo omejitve (constraints propagation) ali dokazujemo izreke in tako dobimo vrednosti ostalih spremenljivk. Kvalitativna simulacija nam tako da razlago delovanja sistema na nekem nivoju, pač odvisno od tega, kaj smo modelirali kot atomaren fizikalni pojav.

3. VZROČNO STRUKTURNI OPIS

Kuipers [Kuipers 82, Kuipres 83] opisuje kvalitativne pojave s petimi tipi omejitev nad spremenljivkami:

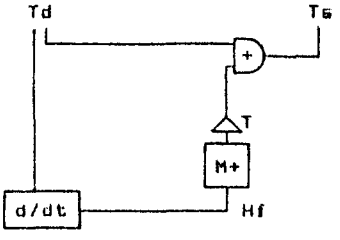
- Aritmetične omejitve določajo, da morajo biti vrednosti spremenljivk v vsakem trenutku v definirani relaciji.
- Funkcijske omejitve so tipa Y = M+(X), kar pomeni, da je Y strogo naraščajoča funkcija X (ali padajoča za M-(X)).
- Odvodi Y = dX/dt pomenijo, da je Y proporcionalen hitrosti spremembe X.
- Neenačbe in pogojne omejitve določajo pogoje, pod katerimi veljajo druge omejitve.

Za primer vzemimo situacijo na sliki 5. Sistem je sestavljen iz posode z vodo (temperatura Td) in grelca (temperatura Ts).



Slika 5: Grelec in posoda s tekočino.

Kuipers opiše tak sistem s t.im. "vzročno strukturnim opisom" (causal structural description), v katerem so vse omejitve nad spremenljivkami modela (slika 6).



Slika 6: Vzročno strukturni opis situacije iz slike 5.

Vzročno strukturni opis je dejansko sistem omejitev:

$$\Delta T = Ts - Td \quad (1)$$
$$Hf = M+(\Delta T) \quad (2)$$
$$Hf = dTd/dt \quad (3)$$

Pretoka toplote Hf v posodo z vodo je strogo naraščajoča funkcija temperaturne razlike ΔT, kjer je Hf = 0 za ΔT = 0. Obenem je Hf sorazmeren hitrosti naraščanja temperature v posodi Td. Model se reši s kvalitativno simulacijo, t.j. s širjenjem omejitev po modelu, tako da vse spremenljivke dobijo vrednosti. Vzemimo, da je temperatura grelca konstantna, in si izberimo prostor količin za spremenljivki Td in ΔT:

Td = manjša_od_Ts, Ts, večja_od_Ts
ΔT = manjša_od_0, 0, večja_od_0

če predpostavimo: Td = manjša_od_Ts
dobimo iz (1): ΔT = večja_od_0
in iz (2,3): Hf = +1 = dTd/dt
torej se temperatura vode veča.

če predpostavimo: Td = Ts
dobimo iz (1): ΔT = 0
in iz (2,3): Hf = 0 = dTd/dt
torej temperatura vode ne narašča več, je konstantna.

Kuipers utemeljuje svoj pristop z analizo sistematičnega opazovanja zdravnikov pri delu [Kuipers 83]. Trdi, da ima zdravnik "vzročni model" pacienta, ki ga uporablja za simulacijo normalnega delovanja telesa, patološkega obnašanja in individualnih posebnosti pacienta. Podrobnejši rezultati opazovanja zdravnika pri pri delu so naslednji:
- Zdravnik uporablja za razlago relativno majhno število objektov in relacij v domeni.
- Simbolični opisi količin in vrednosti so izraženi s kvalitativnimi pojmi (npr. smer toka, porast količine, nizek tlak, ...). To kaže na dejstvo, da so simbolični opisi izraženi v glavnem s pojmi urejenih relacij med vrednostmi.
- Opisi strukturnih odnosov sistema so izraženi ločeno od opisov dinamičnega delovanja (mehanizmov) sistema.

4. KVALITATIVNI PROCES

Po Forbusu [Forbus 81, Forbus 82a] je proces osnovni izvor sprememb v fizikalni situaciji (npr. gretje, vrenje, gibanje, ...). Razvil je teorijo kvalitativnih procesov (qualitative process theory) [Forbus 82a], ki opisuje proces s: - pogoji, pod katerimi teče,
- relacijami med spremenljivkami
- vplivi na spremenljivke.

Procesi vplivajo na objekte sistema na različne načine. Večino teh vplivov je možno modelirati s spremembami parametrov objektov. Vzemimo za primer situacijo z grelcem in posodo s tekočino na sliki 5. Proces "Toplotni tok" (ki teče od grelca v posodo) vpliva na tekočino v posodi tako, da spreminja njeno temperaturo, toploto in volumen. Parameter objekta je definiran s količino in odvodom, oboje pa ima velikost in predznak. Uporabljali bomo sledeče oznake (za primer temperature):

```
   T(tekočina) ... temperatura tekočine
  zT(tekočina) ... predznak temperature (+,-)
  dT(tekočina) ... predznak odvoda, kjer pomeni:
                     +1  rast temperature,
                      0  konstantno temperaturo
                     -1  padanje temperature
hitT(tekočina) ... velikost odvoda temperature,
                   t.j. kako hitro se spreminja
                   temperatura
```

Velikosti količin so definirane s prostorom količin (quantity space), ki je delno urejena množica. Za primer na sliki 5 je prostor količin definiran takole:

```
K ... količina
T ... temperatura
P ... tlak
V ... volumen
H ... toplota

Prazno -> K(tekočina) -> Polno

                         /-> T(vrelišče)
T(led) -> T(tekočina)
                         \-> T(grelec)

P(stisk) -> P(posoda) -> P(eksplozija)
```

Proces je formalno definiran z:
- objekti, ki nastopajo v procesu,
- predpogoji (zunanjimi), ki morajo biti izpolnjeni, da proces lahko teče,
- pogoji, katerim morajo zadoščati spremenljivke, da proces lahko teče,
- relacijami med spremenljivkami,
- vplivi procesa na spremenljivke.

V relacijah lahko nastopa poleg aritmetičnih izrazov tudi simbol "0<", ki označuje kvantitativno sorazmernost med dvema spremenljivkama. Vplivi so izraženi s simboloma "I+" in "I-", ki označujeta pozitiven ali negativen vpliv procesa na spremenljivko. Za popolno omejitev določene spremenljivke je potrebno zbrati vplive vseh procesov nanjo.

Za primer definirajmo procesa "Toplotni tok" in "Vrenje" v situaciji na sliki 5:

```
Toplotni_tok( izvor, ponor)
   objekti:     izvor, ponor
   predpogoji:  Toplotna_povezava( izvor, ponor)
   pogoji:      T(izvor) > T(ponor)
   relacije:    hitH(ponor) 0< T(izvor)-T(ponor)
   vplivi:      I- H(izvor)
                I+ H(ponor)

Vrenje( tekočina)
   objekti:     grelec, tekočina, plin, para
   predpogoji:  Prostor( plin) &
                Skupna_površina( tekočina, plin)
   pogoji:      Toplotni_tok( grelec, tekočina),
                T(tekočina) = T(vrelišče)
   relacije:    Snov( para) = Snov( tekočina),
                hitK(para) =  hitK(tekočina),
                hitK(para) 0< hitH(tekočina),
                T(para) = T(tekočina)
   vplivi:      I- K(tekočina),
                I+ K(para)
```

Poglejmo, kaj se lahko zgodi v dani situaciji. Če je temperatura grelca večja od temperature tekočine v posodi, je izpolnjen pogoj za proces "Toplotni tok". Porast toplotnega toka je sorazmeren razliki temperatur in ima pozitiven vpliv na toploto in (ker se količina ne spreminja) temperaturo tekočine. Ker je grelec izvor toplote, lahko negativen vpliv na njegovo toploto zanemarimo. Glede na prostor količin za temperaturo:

```
                         /-> T(vrelišče)
T(led) -> T(tekočina)
                         \-> T(grelec)
```

bo temperatura tekočine slej ko prej dosegla T(vrelišče), če je le T(grelec) > T(vrelišče). S tem pa so že izpolnjeni vsi pogoji za proces "Vrenje".

Ker ima proces "Vrenje" negativen vpliv na količino tekočine, se glede na prostor količin:

```
Prazno -> K(tekočina) -> Polno
```

proces lahko konča tako, da se količina tekočine v posodi spremeni v Prazno. Vendar pa so še druge možnosti. Pozitiven vpliv vrenja na količino pare pomeni na osnovi fizikalnega zakona:

```
P(para) * V(para) = K(para) * T(para)
```

in glede na to, da je temperatura pare konstantna dT(para) = 0, povečanje volumna dV(para) = +1 in ker je posoda zaprta tudi povečanje tlaka dP(para) = +1. Ker se para dotika posode in tekočine velja:

```
P(para) = P(posoda) = P(tekočina)
```

kar pomeni, da naraščata tako tlak v posodi, kot tlak tekočine. Glede na prostor količin za tlak:

```
P(stisk) -> P(posoda) -> P(eksplozija)
```

lahko torej porast tlaka v posodi privede do eksplozije posode. Obstaja pa še ena možnost.

Kot vemo iz fizike je temperatura vrenja sorazmerna tlaku tekočine:

T(vrelišče) 0< P(tekočina)

iz česar sledi, da se temperatura vrelišča viša dT(vrelišče) = +1. To pa pomeni, da je toplotni tok iz grelca v tekočino večji in se tako vrenje nadaljuje pri vedno višjih temperaturah.

Kaj smo torej dobili na koncu? Če ni nobenih zunanjih vplivov, se lahko proces vrenja konča takole:

- K(tekočina) = Prazno, vsa tekočina se upari, para se segreje na T(grelec).
- T(tekočina) = T(grelec), doseženo je termalno ravnovesje.
- P(posoda) = P(eksplozija), posoda eksplodira.

Za natančno določitev konca procesa je potrebno več informacij, vendar smo dobili vsaj opozorilo na možnost katastrofe.

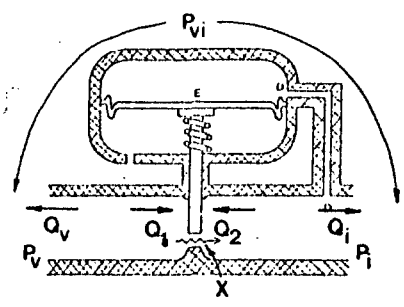
Forbus navaja primere uporabe teorije kvalitativnih procesov še na drugih področjih, kot npr.:
- Simulacija gibanja in trkov kroglic v dvodimenzionalnem prostoru [Forbus 80, Forbus 82a].
- Simulacija delovanja redukcijskega ventila [Forbus 82b].
- Interpretacija meritev v fiziki [Forbus 83a].
- Učenje zaokroženih področij fizike [Forbus 83b].

5. MODELIRANJE S KOMPONENTAMI

Po de Kleeru in Brownu [de Kleer 79, de Kleer 83] je vsaka naprava sestavljena iz fizično različnih delov. Njuna ideja je izpeljati obnašanje naprave iz obnašanja njenih sestavnih delov. Obnašanje naprave, pravzaprav mehanistično razlago obnašanja, dobimo s kvalitativno simulacijo, ki jo de Kleer včasih imenuje "envisioning".

Osnovni opisni pojem je t.im. kvalitativna deferencialna enačba ("confluence"), ki deluje kot omejitev nad spremenljivkami in odvodi. Ker ta enačba dejansko definira relacijo med spremenljivkami, predstavlja več hkratnih vplivov posameznih spremenljivk na ostale. Ista spremenljivka se lahko pojavlja v različnih enačbah in je tako vplivana na več različnih načinov. Vsaka enačba mora biti izpolnjena samostojno.

Model sestavljene naprave je definiran s topologijo naprave in slovarjem njenih sestavnih delov. Topologija opisuje strukturo naprave, t.j. povezavo različnih delov. Model posameznega dela naprave lahko uporablja spremenljivke, ki so skupne le sosednjim delom.



Slika 7: Regulator tlaka. Pretok označujemo s Q, tlak z P in površino odprtine ventila z X.

Vzemimo za primer regulator tlaka na sliki 7. Regulator je sestavljen iz ventila in senzorja tlaka. Kvalitativna diferencialna enačba za ventil je:

dPvi + dX = dQ1

kjer je Pvi padec tlaka od vhoda v ventil do izhoda, Q1 pa pretok v ventil. Pretok je premosorazmeren padcu tlaka skozi ventil in površini odprtine. Enačba za senzor tlaka je:

dX = -dPi

kjer je Pi tlak na izhodu regulatorja. Ta se spreminja obratnosorazmerno z odprtino ventila. Pretok na izhodu regulatorja je sorazmeren tlaku:

dQi = dPi

Ostale enačbe formulirajo zakone ohranitve materije in definicijo pritiska:

dQv = -dQ1
dQi = -dQ2
dQ1 = -dQ2
dPvi = dPv - dPi

Obnašanje naprave dobimo, če najdemo take vrednosti spremenljivk (+1, 0, -1), da so izpolnjene vse kvalitativne diferencialne enačbe. Če tlak na vhodu regulatorja narašča dPv = +1, obstaja le ena rešitev:

dPvi = +1
dQ1 = +1
dQ2 = -1
dQv = -1
dQi = +1
dPi = +1
dX = -1

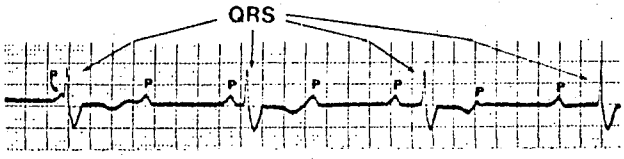
Padec tlaka skozi regulator torej narašča, zato se veča pretok na izhodni strani, tlak na izhodni strani in zato se površina odprtine manjša. To rešitev je možno dobiti na različne načine, npr. z zadovoljevanjem omejitev (constraints satisfaction).

De Kleer navaja primere uporabe vzročnega sklepanja in kvalitativne simulacije pri analizi elektičnih vezij [de Kleer 79a, de Kleer 79b]. Za reševanje problemov v mehaniki predlaga [de Kleer 77] večplastno predstavitev znanja. Kvalitativna simulacija se uporablja za reševanje enostavnih problemov in omogoča uporabo kvalitativnih argumentov, kadar je to možno. Če pa na ta način ne pridemo do rešitve problema, uporabimo klasično fiziko in matematične enačbe.

6. KVALITATIVNI MODEL SRCA

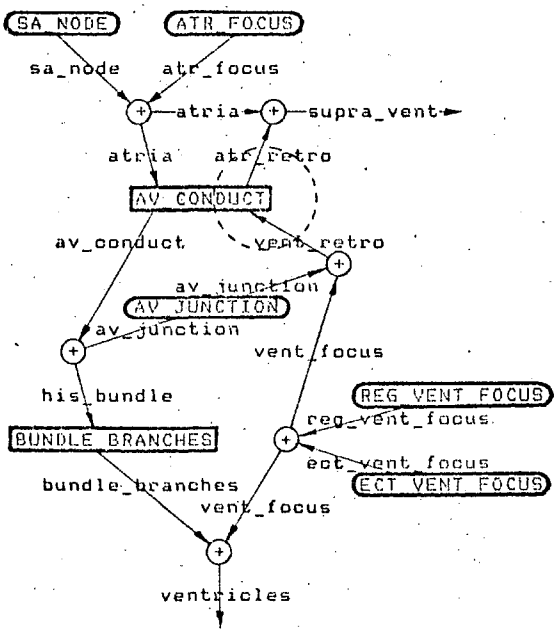
Razvili smo kvalitativni model srca, ki simulira njegovo električno aktivnost [Mozetič 83]. Zdravniki dobijo informacije o električni aktivnosti srca na elektrokardiogramu (EKG). Iz nepravilnih oblik na EKG sklepajo na okvare v srcu - med drugim tudi na srčne aritmije. Srčne aritmije so motnje v generiranju ali prevajanju električnih impulzov v srcu. Naš sistem za EKG diagnostiko pozna 30 osnovnih srčnih aritmij.

Definirali smo jezik za kvalitativni opis EKGja, s katerim je možno opisati EKG vsake aritmije (ali hkratne kombinacije več aritmij). Primer opisa na sliki 8 smo dobili s kvalitativno simulacijo modela srca.




```
arr_ecg( [ sinus_rhythm,
           av_block_3,
           ventricular_rhythm ], [
    rhythm : [regular],
    regular_P : [normal],
    rate_of_P : [between_60_100],
    relation_P_QRS : [independent_P_QRS],
    regular_PR : [meaningless],
    regular_QRS : [wide_RBBB,wide_LBBB,
                   wide_LBBB_RBBB],
    rate_of_QRS : [under_60] ]).
```

Slika 8: EKG krivulja in njen kvalitativni opis, zapisan v PROLOGovi sintaksi. Aritmija, ki povzroči tak EKG je kombinacija treh osnovnih aritmij: sinusni ritem, kompleten AV blok in ventrikularni ritem. EKG je opisan z lastnostmi:
- ritem utripov je pravilen,
- P valovi so normalni,
- frekvenca P je med 60 in 100 na sek.,
- P valovi so neodvisni od QRS kompleksov,
- o PR intervalu je nesmiselno govoriti,
- QRS kompleksi so široki,
- frekvenca QRS je pod 60 na sek.



Slika 9: Shema modela srca. Ovali označujejo generatorje impulzov:
- SA_NODE je sino-atrialen vozel,
- ATR_FOCUS je žarišče kjerkoli v atrijih,
- AV_JUNCTION je žarišča med atrio-ventrikularnim vozlom in junkcijo,
- REG_VENT_FOKUS je žarišče kjerkoli v ventriklih, ki oddaja redne impulze,
- ECT_VENT_FOKUS je žarišče kjerkoli v ventriklih, ki oddaja občasne impulze.

Pravokotniki označujejo prevodne poti:
- AV_CONDUCT je prevod iz atrijev v ventrikle,
- BUNDLE_BRANCHES pa je prevod skozi krake.

Sumatorji impulzov so označeni z ⊕, imena impulzov pa so napisana z malimi črkami.

Model srca je predstavljen kot mreža, v kateri so generatorji električnih impulzov, sumatorji in prevodne poti (slika 9). Objekti, ki nastopajo v modelu so:
- deli srca,
- redni in občasni impulzi,
- atributi EKG.

Vsaka aritmija je definirana s stanjem (ali disjunkcijo stanj) enega dela srca. Kombinacija aritmij določa nenormalna stanja nekaj delov srca, popolno stanje srca pa dobimo s privzetjem ostalih delov za normalne. Stanje srca gre skozi omejitve, ki izločijo fiziološko nemogoče in medicinsko nesmiselne kombinacije. Nato se sproži kvalitativna simulacija modela, ki generira ustrezne EKG opise. Tipično ustreza kombinaciji aritmij več opisov (disjunktivnih) EKG.

Model srca je predstavljen kot zaporedje 35 pravil. Formalno so pravila logične izjave oblike:

$$\forall x \; \exists y : \; A(x) \Rightarrow B(y) \; \& \; C(x,y)$$

kjer sta x in y vektorja spremenljivk, A(x) in B(y) konjunkciji pozitivnih literalov (objektov v srcu), C(x,y) pa je logičen izraz, ki ima vlogo pogoja. Če A(x) ni res, pravila ne uporabimo. Če je A(x) res in lahko dokažemo C(x,y) je dokazan tudi B(y) (tu gre za posamezne instance spremenljivk x in y); gremo na naslednje pravilo. Če pa dokažemo ~C(x,y), smo prišli do protislovja, kar pomeni, da A(x) ne more biti res. Zato se moramo vrniti na eno od prejšnjih pravil in poskusiti dokazati A(x) za kako drugo instanco x. Primer pravila, ki modelira retrogradni prevod impulzov iz ventriklov v atrije (obkroženo na sliki 9) je na sliki 10.

Kvalitativna simulacija poteka tako, da množici aksiomov (modelu srca) dodamo množico dejstev (stanje srca) in izpeljemo vse možne posledice (izreke). Iz njih izluščimo opise EKG, ki logično sledijo iz danega stanja srca, torej pripadajo dani kombinaciji aritmij.

O netrivialnosti problema priča podatek, da smo za cca. 2400 možnih kombinacij aritmij avtomatsko izpeljali čez 140.000 opisov EKG. Praktično zanimiva podmnožica tako generirane baze znanja je uporabna v ekspertnem sistemu za diagnostiko in poučevanje srčnih aritmij. Program, ki je dolg 25 strani PROLOGove kode, teče na računalniku DEC-10 in je za generacijo kompletne baze znanja porabil 1 uro CPU časa.

7. ZAKLJUČEK

En od ciljev kvalitativnega modeliranj je izpeljava vzročne razlage obnašanja. Razlaga je zaporedje stavkov, kjer je vsak stavek odvisen le od prejšnjih. Razlaga je lahko izražena kot logični dokaz v naravnem dedukcijskem sistemu [de Kleer 83]. Izrek ima v tem primeru vlogo napovedi, utemeljitve v stavkih pa so posamezni logični sklepi dokaza. Logični dokaz je primeren za razlago, ker so tako dobljeni zaključki neovrgljivi in jedrnati. Vendar pa kaže de Kleer tudi na probleme, ki jih prinese uporaba logičnega dokaza za razlago. Včasih je namreč potrebna uporaba predpostavk in indirektnih dokazov, kar pa je za razlago intuitivno nezadovoljivo. Poleg tega so lahko logični dokazi vzročno obrnjeni. Problem je v tem, da logika razlaga zakaj se sistem obnaša tako kot se, ne pa kako se obnaša, kar je naloga vzročne razlage.

```
[permanent( vent_retro:
      form( Focus, RhythmO, RateO )),
 heart( av_conduct: Conduct )]
=> [permanent( atr_retro:
         form( Retro_cond, Rhythm, Rate ))]
   & ( ( Conduct = blocked ; RateO = zero ;
         RateO >- between_100_250 ),
         Rhythm = quiet, Rate = zero
     ; not_blocked( Conduct ), RateO >- zero,
       between_250_350 >- RateO,
       retrograd( Focus, Retro_cond ),
       Rhythm = RhythmO, Rate = RateO ).


%%  Način retrogradnega prevoda "Retro_cond"
%%  je odvisen od izvora impulza "Focus".

retrograd( av_junction,     faster     ).
retrograd( av_junction,     slower     ).
retrograd( av_junction,     equal      ).
retrograd( reg_vent_focus,  equal      ).
retrograd( reg_vent_focus,  slower     ).
retrograd( ect_vent_focus,  equal      ).
retrograd( ect_vent_focus,  slower     ).


%%  AV prevod ni popolnoma blokiran, če je v
%%  enem od ostalih možnih stanj.

not_blocked( Conduct ) :-
   Conduct in
   [normal, kent, james, delayed,
    progress_delayed, partial_blocked].


%%  Definicija relacije "večji" nad
%%  frekvencami.

A >- B :-
   cons( _, [B | Higher],
            [zero,
             under_60,
             between_60_100,
             between_100_250,
             between_250_350,
             over_350] ),
   A in Higher.


   %%  Definicija elementa seznama.

   X in [X | _].
   X in [_ | L] :- X in L.


   %%  Povezava dveh seznamov.

   cons( [], L, L ).
   cons( [X | L0], L1, [X | L] ) :-
      cons( L0, L1, L ).
```

Slika 10: Pravilo za retrogradni prevod impulzov iz atrijev v ventrikle z vsemi pomožnimi procedurami. Napisano je v PROLOGovi sintaksi, kjer pomeni ":-" implikacijo v levo, "," konjukcijo, ";" disjunkcijo, "." konec stavka, "[X|L]" pa seznam, kjer je X glava seznama, L pa rep. Vsi ostali simboli: ":", "=>", "&", ">-", "in" so definirani kot infiksni binarni operatorji in nimajo nobenega semantičnega pomena v samem PROLOGu.

V primeru modela srca tega problema nismo imeli, ker je postopek dokazovanja (modus ponens) enostaven in lahko razumljiv. Razlaga je enostavno sled uporabljenih logičnih izjav. Zaporedje logičnih izjav v modelu srca pa ustreza vzrokom in posledicam dejanskega dogajanja v srcu. Tako se je omejitev formalizma za opis modela srca (določen vrstni red pravil) izkazala kot prednost pri vzročni razlagi delovanja. Poleg tega so možni, brez spremembe modela, celo različni nivoji abstrakcije. Potrebno je le spremeniti poimenovanje aritmij (npr. atrijske in av_junkcijske motnje združiti v supraventrikularne) ali jezika za opis EKG (npr. različne tipe širokih QRS združiti pod en pojem "širok") in že dobimo bolj (ali manj) kompaktne definicije aritmij.

Pristopi Kuipersa, Forbusa in de Kleera temeljijo na opisovanju fizikalnih pojavov z omejitvami. Za širjenje omejitev po sistemu kvalitativnih enačb uporabljajo jezik omejitev CONLAN [Steele 80].

Forbus modelira fizikalno situacijo s kvalitativnimi procesi. Njegov pristop je torej procesno orientiran, kar pomeni, da je potrebno zbrati vplive vseh procesov na posamezno spremenljivko, če želimo zvedeti za omejitve, katerim mora zadoščati. Opisovalec modela je sicer obremenjen z identifikacijo abstraktnih procesov, ki tečejo v dani situaciji, vendar so ti procesi zelo blizu intuitivni ideji fizikalnega procesa. Poleg tega je njegova teorija še najbolje formalizirana.

De Kleerov pristop je komponentno orientiran, saj gradi na objektivnem, eksplicitnem opisu strukture naprave. Njegov kvalitativni model nazorno kaže, kako vsak posamezen del naprave vzročno prispeva k celovitemu delovanju naprave. Pri obeh torej formalni, kvalitativni model situacije odraža intuitivno sliko fizične situacije.

Na drugi strani je Kuipersov "vzročno strukturni opis" po našem mnenju zavajajoč pojem. Saj ni niti vzročen (v fizikalnem smislu) niti nima nobene zveze s fizikalno strukturo dejanske situacije. Kuipers sicer nudi formalizem za predstavitev kvalitativnih diferencialnih enačb in metode za njihovo reševanje. Vendar pa je to kvalitativa matematika in ne kvalitativna fizika.

Mi opisujemo model srca z logičnimi izjavami. Vsaka izjava ponazarja dogajanje v nekem delu srca, povezujejo pa jih skupni literali (ki ponazarjajo objekte v srcu, npr. impulze). Model odraža fizikalno sliko dogajanja v srcu. Kvalitativno simulacijo dosežemo z dokazovanjem izrekov, saj za dane aksiome (model srca) in dejstva (stanje delov srca) izpeljemo vse izreke (med drugim tudi opise EKG).

Dokazovalec izrekov je implementiran v PROLOGu [Clocksin 81], ki je že sam po sebi avtomatski dokazovalec izrekov v Hornovi logiki. Tako smo že imeli na voljo primerna orodja (unifikacija, delno instanciranje spremenljivk, avtomatsko vračanje, ...), ki močno olajšujejo implementacijo. Smatramo, da bi bila tudi implementacija jezika omejitev CONLAN in podobnih v PROLOGu relativno enostavna.

8. LITERATURA

[Clocksin 81]
Clocksin WF, Mellish CS : Programming in Prolog. Springer-Verlag, Berlin, Heidelberg, New York, 1981.

[de Kleer 77]
de Kleer J : Multiple Representations of Knowledge in a Mechanics Problem-Solver. Proc. of the Fifth International Joint Conference on Artificial Intelligence, str. 299-304, MIT, Cambridge, 22.-25. avg. 1977.

[de Kleer 79a]
de Kleer J : Causal and Teleological Reasoning in Circuit Recognition. MIT, Artificial Intelligence Lab. TR-529, Cambridge, 1979.

[de Kleer 79b]
de Kleer J : The Origin and Resolution of Ambiguities in Causal Argument. Proc. of the Sixth International Joint Conference on Artificial Intelligence, str. 197-203, Tokyo, 20.-23. avg. 1979.

[de Kleer 83]
de Kleer J, Brown JS : The Origin, Form and Logic of Qualitative Physical Laws. Proc. of the Eight International Joint Conference on Artificial Intelligence, str. 1158-1169, Karlsruhe, 8.-12. avg. 1983.

[Forbus 80]
Forbus KD : Spatial and Qualitative Aspects of Reasoning about Motion. Proc. of the First Annual National Conference on Artificial Intelligence, str. 170-173, Stanford University, 18.-21. avg. 1980.

[Forbus 81]
Forbus KD : Qualitative Reasoning about Physical Processes. Proc. of the Seventh International Joint Conference on Artificial Intelligence, str. 326-330, Vancouver, 24.-28. avg. 1981.

[Forbus 82a]
Forbus KD : Qualitative Process Theory. MIT, Artificial Intelligence Lab. Memo No. 664, Cambridge, 1982.

[Forbus 82b]
Forbus KD : Implementation Issues in Common Sense Reasoning Programs. Proc. of the Workshop on Prolog Programming Environments, str. 25-49, Linkoping University, 24.-26. marec 1982.

[Forbus 83a]
Forbus KD : Measurment Interpretation in Qualitative Process Theory. Proc. of the Eight International Joint Conference on Artificial Intelligence, str. 315-320, Karlsruhe, 8.-12. avg. 1983.

[Forbus 83b]
Forbus KD, Gentner D : Learning Physical Domain: Towards a Theoretical Framework. Proc. of the International Machine Learning Workshop, str. 198-202, Monticello, Illinois, 22.-24. junij 1983.

[Kuipers 82]
Kuipers B : Getting the Envisionment Right. Proc. of the National Conference on Artificial Intelligence, str. 209-212, Pittsburgh, 18.-20. avg. 1982.

[Kuipers 83]
Kuipers B, Kassier JP : How to Discover a Knowledge Representation for Causal Reasoning by Studying an Expert Physician. Proc. of the Eight International Joint Conference on Artificial Intelligence, str. 49-56, Karlsruhe, 8.-12. avg. 1983.

[Mozetič 83]
Mozetič I, Bratko I, Lavrač N : An Experiment in Automatic Synthesis of Expert Knowledge through Qualitative Modeling. Proc. of Logic Programming Workshop '83, str. 553-559, Albufeira, Portugal, 26. junij - 1. julij 1983.

[Steele 80]
Steele GL : The Definition and Implementation of a Computer Programming Language Based on Constraints. MIT, Artificial Intelligence Lab. TR-595, Cambridge, 1980.

UPORABNI PROGRAMI

```
=============================================
=                                           =
=        Kvadraturno integriranje z         =
=             Gaussovo metodo               =
=                                           =
=============================================
```

```
*********************************************
* Informatica UP 16                         *
* Quadrature Integration Using Gauss Method *
* maj 1984                                  *
* modificirala Jelena Ficzko                *
* sistem CP/M, Delta Partner                *
* prevajalnik Pascal MT/+                   *
*********************************************
```

1. Področje uporabe

S pomočjo Gaussove metode za kvadraturno integriranje bomo izračunali vrednosti integralov v danih integracijskih mejah.

2. Opis programa

Računali bomo tri (p = 3) določene integrale na intervalu (a,b) za a = 2 in b = 5. Ta interval razdelimo na m (m = 10) enakih podintervalov, kjer bomo izvajali n-točkovno integriranje za n = 3, 8 in 12. Dane so tudi konstante c in u za n-točkovno integriranje, ki jih zapišemo v polji c in u z elementi c[k] in u[k].

Funkcija func nam omogoča v enem izvajanju integracijo treh (ali več) funkcij, in sicer so to y = x*x, y = x*(1+x*x) in y = 1. S func pa izberemo tudi j-to funkcijo, ki jo bomo v proceduri kvad integrirali in izračunamo njeno funkcijsko vrednost v točki t.

Procedura kvad uporabi ta rezultat in s pomočjo Gaussove formule izračuna integral

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

Lista 1. V listi na desni je podan program za kvadraturno integriranje z Gaussovo metodo v Pascalu. Program sestavljajo procedura quad, func, csum in procedura za izpis. Najpomembnejši sta proceduri quad in func; csum rabi le za preizkus natančnosti koeficientov.

V programu se namesto oslatega predklepaja uporablja znak '$', namesto oslatega zaklepaja pa znak 'ć' (YU abeceda).

Lista 1 se nadaljuje na naslednji strani.

```
PROGRAM quad2(output);
TYPE  tip12=ARRAY$1..12ć OF real;
      tip3=ARRAY$1..3ć OF real;
VAR   p,m :integer;
      sum:tip3;
      u3,c3:tip12;
      u8,c8:tip12;
      u12,c12:tip12;
      a,b: real;
(*********************************************)
FUNCTION func(x:real;j:integer):real;
  BEGIN
    IF j=1 THEN
      func:=x*x
      ELSE
        BEGIN
          IF j=2 THEN
            func:=(1+x*x)*x
             ELSE
              func:=1
        END
  END;

(*********************************************)
PROCEDURE kvad(a,b:real;m,n,p:integer;
                                c,u:tip12);
VAR h,r,t:real;
    i,j,k:integer;
BEGIN
  h:=(b-a)/m;
  FOR j:=1 TO p DO
    BEGIN
      sum$jć:=0;
      r:=a-h;
        FOR i:=1 TO m DO
          BEGIN
            r:=r+h;
              FOR k:=1 TO n DO
                BEGIN
                  t:=r+u$kć*h;
                  sum$jć:=sum$jć+c$kć*func(t,j)
                END
          END;
         sum$jć:=h*sum$jć
    END
END;
(*********************************************)
PROCEDURE csum(n:integer; c:tip12);
VAR i:integer;
     a:real;
BEGIN
  a:=0;
  FOR i:=1 TO n DO
    a:=a+c$ić;
  writeln('n=',n,'     vsota = ',a)
END;
(*-----------------------------------------*)
PROCEDURE izpis(n:integer);
VAR j:integer;
BEGIN
  FOR j:=1 TO p DO
    writeln('n=',n,
            '     Integral ',j,' = ',sum$jć)
END;
(*-----------------------------------------*)
BEGIN (*glavni program*)
  p:=3;
  m:=10;
  u3$1ć :=0.112702;      c3$1ć :=0.277778;
  u3$2ć :=0.5;           c3$2ć :=0.444444;
  u3$3ć :=1-u3$1ć;        c3$3ć := c3$1ć;

  u8$1ć :=0.198551E-01; c8$1ć :=0.506143E-01;
  u8$2ć :=0.101667;     c8$2ć :=0.111191;
  u8$3ć :=0.237234;     c8$3ć :=0.156853;
  u8$4ć :=0.408283;     c8$4ć :=0.181342;
```

```
  u8$5Ć :=1-u8$4Ć;        c8$5Ć :=c8$4Ć;
  u8$6Ć :=1-u8$3Ć;        c8$6Ć :=c8$3Ć;
  u8$7Ć :=1-u8$2Ć;        c8$7Ć :=c8$2Ć;
  u8$8Ć :=1-u8$1Ć;        c8$8Ć :=c8$1Ć;

  u12$1Ć:=0.921968E-02; c12$1Ć :=0.230877E-01;
  u12$2Ć:=0.479414E-01; c12$2Ć :=0.534699E-01;
  u12$3Ć:=0.115049;     c12$3Ć :=0.800392E-01;
  u12$4Ć:=0.206341;     c12$4Ć :=0.101584;
  u12$5Ć:=0.316085;     c12$5Ć :=0.116746;
  u12$6Ć:=0.437383;     c12$6Ć :=0.124574;
  u12$7Ć:=1-u12$6Ć;     c12$7Ć :=c12$6Ć;
  u12$8Ć:=1-u12$5Ć;     c12$8Ć :=c12$5Ć;
  u12$9Ć:=1-u12$4Ć;     c12$9Ć :=c12$4Ć;
  u12$10Ć:=1-u12$3Ć;    c12$10Ć:=c12$3Ć;
  u12$11Ć:=1-u12$2Ć;    c12$11Ć:=c12$2Ć;
  u12$12Ć:=1-u12$1Ć;    c12$12Ć:=c12$1Ć;

a:=2; b:=5;
csum(3,c3);
kvad(a,b,m,3,p,c3,u3);
izpis(3);
csum(8,c8);
kvad(a,b,m,8,p,c8,u8);
izpis(8);
csum(12,c12);
kvad(a,b,m,12,p,c12,u12);
izpis(12)
END.
```

Lista 1 (nadaljevanje s prejšnje strani). Abscise u in uteži c so dane le z 6-mestno natančnostjo.

V tej formuli predstavlja h dolžino podintervala, torej h = (b-a)/m.

S proceduro csum seštejemo elemente polja c, idealna vsota pa je 1. Ker pa so vrednosti polja zaokrožene, kot vidimo v listi rezultatov, se vsota elementov polja ne ujema vedno s to vrednostjo, a odstopanja so majhna.

Sledi še procedura izpis, ki izpiše vrednost integrala j-te funkcije pri n-točkovnem integriranju.

3. Izvajanje programa

Program se je izvajal na sistemu Delta Partner. Shranjen je bil na disku kot zbirka QUAD2.SRC. Zbirka se je prevajala s klicem MTPLUS quad2 in popravljala, dokler niso bile odpravljene vse sintaktične napake. Po prevodu se je ta zbirka povezala s knjižničnimi procedurami, ki so potrebne za izvajanje programa, in sicer z ukazom

```
LINKMT quad2,trancend,fpreals,paslib/s.
```

Rezultat izvajanja programa pa je prikazan v listi rezultatov.

Če primerjamo rezultate iz liste z idealnimi vrednostmi

$$\int_2^5 x^{**}x\,dx = 39\,, \qquad \int_2^5 x(1+x^{**}x)\,dx = 167{,}75\,, \qquad \int_2^5 dx = 3$$

vidimo, da so vsi trije integrali najmanj natančno izračunani pri 12-točkovnem integriranju.

```
2A)quad2
 n=3      vsota =  9.99999E-01
 n=3      Integral 1 =  3.89999E+01
 n=3      Integral 2 =  1.62749E+02
 n=3      Integral 3 =  2.99999E+00
 n=8      vsota =  1.00000E+00
 n=8      Integral 1 =  3.89999E+01
 n=8      Integral 2 =  1.62749E+02
 n=8      Integral 3 =  2.99999E+00
 n=12     vsota =  9.99001E-01
 n=12     Integral 1 =  3.89608E+01
 n=12     Integral 2 =  1.62586E+02
 n=12     Integral 3 =  2.99699E+00
```

Lista 2. Ta lista prikazuje rezultate programa z liste 1, ko imamo 3-, 8- in 12-točkovno integriranje (n) pri desetih podintervalih (m).

Generiranje kombinacij (k-teric)

```
**********************************************
* Informatica UF 17                          *
* Generation of Combinations                 *
* julij 1984                                 *
* priredil Anton P. Železnikar               *
* sistem CP/M, Delta Partner                 *
* prevajalnik Janus/Ada, verzija 1.5.0       *
**********************************************
```

1. Področje uporabe

Procedura combination v listi 1 oblikuje v zaporednem vrstnem redu (z njeno zaporedno uporabo) k-terice, ki so kombinacije iz števil 0, 1, 2, ... , n-1, če sta dani vrednosti k in n in predhodna (začetna) kombinacija. V vektorju

$$j(1), \ldots, j(k)$$

ki je trenutna kombinacija, se elementi spreminjajo v območju (0, n-1) in vselej se pri vstopu in izstopu iz procedure oblikuje monotono, strogo naraščajoče zaporedje j(1), ... , j(k). Kadar je vhodni vektor sestavljen iz samih ničel, se kot prva kombinacija pojavi

$$n-k, \ldots, n-1$$

Ta kombinacija se pojavi tudi po kombinaciji

$$0, 1, \ldots, k-1$$

in je zadnja v ciklu.

2. Opis programa

Adovski paket v listi 1 je sestavljen iz procedure combination in glavnega preizkusnega programa. Pri oznaki ((final)) uporabimo stavek null; , ker pred besedo END moramo imeti znak ';'. V glavnem programu izberemo k = 6 in n = 7, začetna kombinacija pa je 0, 1, 2, 3, 4, 5. Nato generiramo 20 k-teric, ki jih izpisujemo v dveh stolpcih, ko med posamezne elemente k-terice postavljamo vejice in k-terice ogradimo z oklepaji.

```
---------------------------------------------------
--          Generiranje  kombinacij             --
---------------------------------------------------

WITH util;
PACKAGE BODY testcomb IS
   TYPE polje IS ARRAY (1 .. 20) OF integer;
   -- Ta procedura je predmet naše pozornosti!
--*************************************************

   PROCEDURE combination ( n,k: IN integer;
                           j: IN OUT polje) IS
     a,b,m: integer;
     BEGIN
       FOR b IN 1 .. k-1 LOOP
         IF j(b) >= b THEN
           a := j(b) - b - 1;
           FOR m IN 1 .. b LOOP
             j(m) := m + a;
           END LOOP;
           GOTO final;
         END IF;
       END LOOP;
       FOR m IN 1 .. k LOOP
         j(m) := n - k - 1 + m;
       END LOOP;
<<final>> null;
     END combination;
--*************************************************

   --         Glavni preizkusni program           --
   j: polje; i,k,l,n: integer;
   BEGIN
     -- Določitev začetne kombinacije
     k := 6;
     n := 7;
     FOR i IN  1 .. k LOOP
       j(i) := i-1;
     END LOOP;
     new_line; put( "Začetna kombinacija   ( ");
     FOR i IN 1 .. k LOOP
       put( j(i),2);
       IF i < k THEN put( ","); END IF;
     END LOOP;
     put( " )"); new_line;
     FOR i IN 1 .. 20 LOOP
       combination(n,k,j); put( "( ");
       FOR l IN 1 .. k LOOP
         put( j(l),2);
         IF l < k THEN put( ","); END IF;
       END LOOP;
       put( " )");
       IF i mod 2 = 0 THEN new_line;
         ELSE put( "   "); END IF;
     END LOOP;
   END testcomb;
```

Lista 1. Lista na levi prikazuje paketno telo za preizkus generiranja k-teric (kombinacij), ki so vselej strogo monotono naraščajoče (glej listo 2). Procedura combination vzame iz celoštevilskega polja j trenutno k-terico in jo transformira v novo (naslednjo) k-terico. Parametra n in k izbiramo, pri tem velja n >= k. V glavnem programu se najprej oblikuje začetna k-terica in iz nje in njenih naslednic se oblikujejo nove k-terice, dokler se cikel ne ponovi.

---

3. Izvajanje programa

Lista 2 prikazuje rezultate izvajanja programskega paketa z liste 1. Prikazana je začetna kombinacija elementov in nato še 20 naslednjih kombinacij. Iz liste je razviden kombinacijski cikel, ki se začne ponavljati v zadnji vrstici liste. Pri večji vrednosti n (izbrali smo n = 7) bi dobili daljši kombinacijski cikel.

```
13A>testcomb

Začetna kombinacija   ( 0, 1, 2, 3, 4, 5 )
( 1, 2, 3, 4, 5, 6 )  ( 0, 2, 3, 4, 5, 6 )
( 0, 1, 3, 4, 5, 6 )  ( 0, 1, 2, 4, 5, 6 )
( 0, 1, 2, 3, 5, 6 )  ( 0, 1, 2, 3, 4, 6 )
( 1, 2, 3, 4, 5, 6 )  ( 0, 2, 3, 4, 5, 6 )
( 0, 1, 3, 4, 5, 6 )  ( 0, 1, 2, 4, 5, 6 )
( 0, 1, 2, 3, 5, 6 )  ( 0, 1, 2, 3, 4, 6 )
( 1, 2, 3, 4, 5, 6 )  ( 0, 2, 3, 4, 5, 6 )
( 0, 1, 3, 4, 5, 6 )  ( 0, 1, 2, 4, 5, 6 )
( 0, 1, 2, 3, 5, 6 )  ( 0, 1, 2, 3, 4, 6 )
( 1, 2, 3, 4, 5, 6 )  ( 0, 2, 3, 4, 5, 6 )
```

Lista 2. V tej listi imamo začetno k-terico in iz nje izvedene naslednje k-terice. Cikel se začne ponavljati v zadnji vrstici liste. V glavnem programu lahko določimo vrednosti k in n (s prireditvenima stavkoma) in dobimo tako k-terice, ki so sestavljene iz števil intervala (0, 1, ... , n-1), pri čemer so elementi k-terke paroma različni in se pojavljajo v monotono naraščajočem vrstnem redu.

Operacijski sistem UNIX: njegov nastanek in perspektive I

1. Uvod

Operacijski sistem UNIX (Bell Laboratories) postaja prejkoslej standard na področju uporabe 16- in 32-bitnih mikroprocesorskih sistemov. Tudi domača računalniška industrija mora upoštevati nastanek tega de facto standarda, ki bo hkrati tudi industrijski in institucionalni standard (ANSI, ISO, IEEE).

UNIX se je razvil v visoko zmogljiv programski sistem v raziskovalnem, razvojnem in tudi v komercialnem okolju, tako da je v letu 1986 predviden njegov plasma v vrednosti 5 milijard dolarjev samo na področju mikrosistemov. Rast UNIXovega tržišča je pogojena prav z možnostjo njegove standarizacije, ki naj bi odpravila divjo rast različnih operacijskih sistemov in s tem programsko nezdružljivost. Izvedenke UNIXa so bile razvite za bistveno različne računalniške sisteme, vendar so obdržale osnovno arhitekturno združljivost. Tako so različne izpeljanke UNIXa uporabljive na različnih strojih in ne zahtevajo velikih dodatnih programskih modifikacij.

UNIXova primernost in uspešnost sta pogojeni z njegovo široko uporabo na univerzah, kjer se že študentje naučijo ceniti njegovo zmogljivost pri razvoju programov. Ta proces njegove uporabe se podaljšuje v industrijo celo tedaj, ko to za razvoj aplikacij ni nujno potrebno. Nove zahteve tržišča in podpora programerjev zagotavljajo dolgotrajno uporabo sistema UNIX, to pa povzroča nastajanje nove sistemske programske opreme.

2. Razvoj operacijskega sistema UNIX

Operacijski sistem UNIX se je rodil v zgodnjem miniračunalniškem obdobju v okviru posebne raziskovalne skupine pri Bell Laboratories. Po njegovem rojstvu sa je prvo sprejelo podjetje DEC za svoje 16-bitne miniračunalnike PDP-11 in sa je začelo dobavljati za nominalno ceno univerzam in raziskovalnim laboratorijem.

UNIX se je uveljavil zaradi nenavadnih lastnosti miniračunalniškega operacijskega sistema že v začetku 70-ih let, in sicer zaradi:

- hierarhičnega zbirčnega sistema,
- združljivosti zbirk,
- združljivosti naprav,
- medprocesnega vhoda/izhoda,
- zmogljivega, programirljivega ukaznega interpretiranja,
- podpore za začenjanje asinhronih opravil in zaradi
- bogate in lahko razširljive množice programirnih orodij.

Danes so te lastnosti operacijskega sistema tako pomembne za proizvajalce računalniških sistemov, da je podobnost njihovih proizvodov z UNIXom bistvena tržna referenca.

Najbolj razširjena izvedenka UNIXa se je razvila na PDP-11 sistemih. Na izvedenki UNIX verzija 7 so osnovani sistem III in sistem V podjetja Western Electric in komercialni sistemi Xenix, Idris in Coherent. Osnovni področji uporabe teh sistemov sta obdelava besedil in razvoj programov. Bell Laboratories je prenesel UNIX tudi na stroj VAX-11/780, torej na 32-bitni računalnik z arhitekturo virtualnega pomnilnika, kasneje pa so na univerzi v Berkeleyu (Kalifornija) dodali k sistemu še upravljanje virtualnega pomnilnika. Berkeleyska izpeljanka je dobila oznako 3BSD (okrajšava za 3-rd Berkeley Software Distribution for UNIX). Ta sistem se je uporabljal na več kot 1000 sistemih VAX/UNIX izven Bell Laboratories.

Ko je DARPA (Defense Advanced Research Projects Asency) iskala nov stroj, ki bi zamenjal močno razširjeni miniračunalnik PDP-10 v raziskovalnih ustanovah, se je odločila za stroj tipa VAX. Pri tem se agencija ni zadovoljila s standardnim sistemom VAX/VMS, marveč je sprožila nov razvoj na osnovi 3BSD, ki naj bi dal prenosno izvedenko UNIXa, uporabno za različne računalniške arhitekture in za prihodnje raziskovalne projekte.

UNIX 4.2BSD (ki sa bomo še opisali) je bil rezultat tega projekta, vsebuje pa vrsto izboljšav, ki jih ne najdemo v drugih izpeljankah. Te lastnosti so:

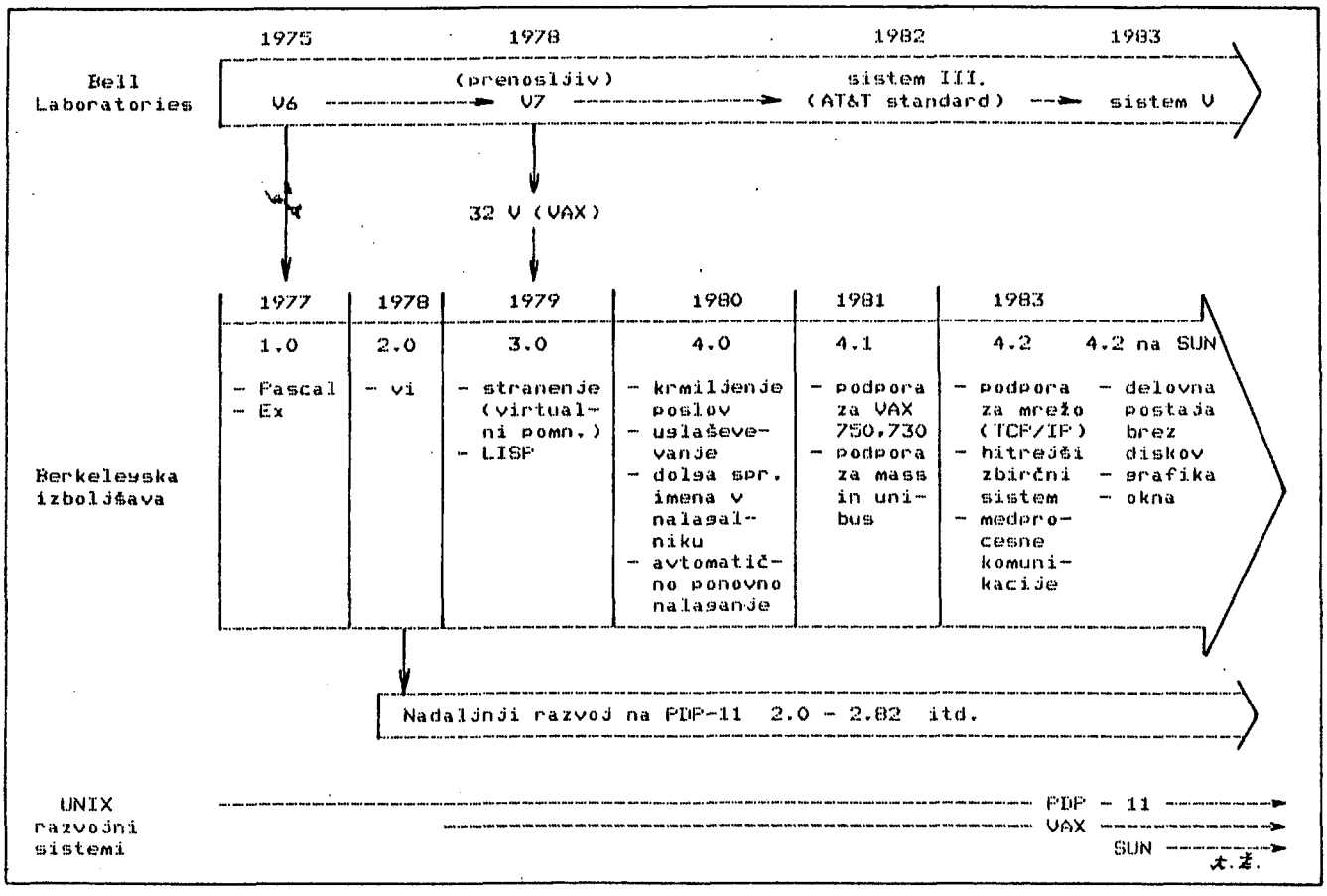
- -- ospredje in ozadnje krmiljenje poslov,
- -- avtomatično ponovno nalaganje operacijskega sistema pri sistemskem zlomu in
- -- visoka zmogljivost novega simbolnega popravljalnika napak.

Slika 1 prikazuje razvoj operacijskega sistema UNIX in njegovih izpeljank.

3. Nadomeščanje (zamena) strani

Podpora virtualnemu pomnilniku v sistemu UNIX 4.2BSD omogoča izvajanje zelo obsežnih procesov, ker sistem pomika strani v in iz sekundarnega pomnilnika po potrebi. Skrivnost tega mehanizma je globalni algoritem nadomeščanja strani, ki povzroča minimizacijo uporabljenega pomnilnika v vsakem procesu, uporabljajoč znano strategijo nadomeščanja strani, ki so bile v poslednjem razdobju najmanj uporabljane. Zaradi tega se lahko hkrati izvaja več velikih procesov, katerih obseg je večji od razpoložljivega pomnilnika. Procesi do velikosti 16M zlogov se lahko izvajajo v sistemu 4.2BSD na stroju VAX/11 ali na delovni postaji SUN (ta uporablja procesor MC68010), čeprav sta njuna pomnilnika manjšega obsega.

Primerjalne ocene so pokazale, da se je z dodajanjem mehanizma virtualnega pomnilnika k sistemu UNIX 4.2BSD lahko povečalo število uporabnikov v enostavnem sistemu z razdeljevanjem procesorskega časa. Na osebni delovni postaji tipa SUN pa se je razen možnosti obdelave večjih programov izboljšala tudi sistemska odzivnost. Pri klicu velikega programa se ta program ne naloži v hitri pomnilnik, nalagajo pa se po potrebi le določene strani programa. Pri vsaki naložitvi strani izbere algoritem za



Slika 1. Razvoj UNIX sistemov (Bell Laboratories: V6, V7, sistem III in sistem V; berkeleyska izboljšava: 1.0, 2.0, 3.0, 4.0, 4.1, 4.2; razvojni sistemi: PDP-11, VAX, SUN)

premeščanje strani stran, ki bo v hitrem pomnilniku zamenjana skladno s strategijo najmanj uporabljane strani v poslednjem razdobju.

Če se sistem 4.2BSD izvaja na stroju brez upravljanja virtualnega pomnilnika, se lahko še vedno uporabi prednost pomnilniške preslikave na ravnini strani, tako da se posamezne strani nalagajo na razpršene pomnilniške lokacije celo tedaj, ko stroj ne podpira ukazov za ponoven začetek, potrebnih pri popravljanju poskusov za dostop na nezasedene strani (napake pri stranenju). Tako je mogoče izvajati sistem tudi na procesorju 68000, ki ne podpira virtualnega pomnilnika. Če pa se stranenje ne uporablja, se poveča zakasnitev pri začenjanju procesov in manj procesov se lahko hkrati izvaja.

4. Hitri zbirčni dostop

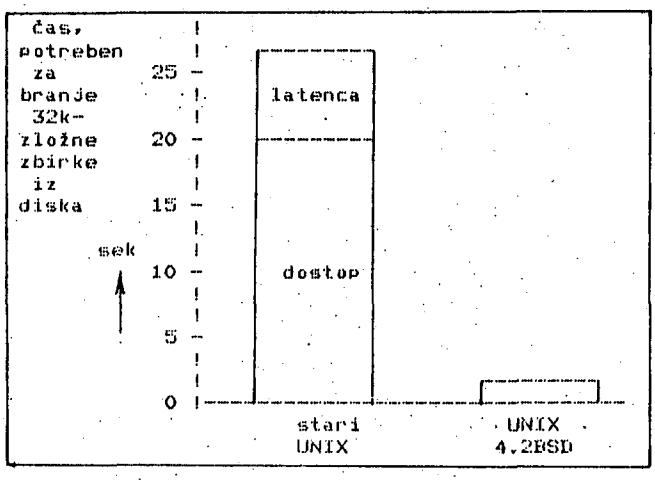
Tradicionalni 16-bitni UNIX sistemi imajo zgleden hierarhični zbirčni sistem, ki pa ni posebno zmogljiv. Na disku s hitrostjo prenosa 1M zlog/sek ni dosegljivi pretok pri prenosu velikih zbirk večji od 50k zlog/sek zaradi slabe razporeditve blokov na disku. Čeprav je ta hitrost zadostna pri aplikacijah z razdeljevanjem časa (npr. pri razvoju programov in pri obdelavi besedil) pa ne zadošča potrebam velikih aplikacij ali implementacij mrežnih zbirčnih sistemov. UNIX 4.2BSD vsebuje novo implementacijo zbirčnega sistema, ki ima izboljšane podatkovne strukture in algoritme.

V zgodnjih izvedenkah UNIXa je imel zbirčni sistem povezan seznam prostih blokov. Pri oblikovanju zbirčnega sistema se je ta seznam uredil z vstavljanjem blokov v prosti seznam z namenom hitrega dostopa. Vstavljanje blokov diskovne zbirke je razširjena metoda za prenos podatkov iz diska v pomnilnik. Brez metode vstavljanja bi sistemska zakasnitev za pripravo naslednjega logičnega bloka narastla v časovni prestop, tako da bi bila potrebna naslednja diskovna rotacija. Z metodo vstavljanja pa se doseže prekrivanje sistemske zakasnitve in časa, potrebnega za preskok vstavljenega bloka, preden se doseže namenski blok.

V zgodnjih izvedenkah UNIXa se je bralo le 512 zlogov z diska z enostavno operacijo prenosa. Večji prenosi so zahtevali dve ali več V/I operacij. Sčasoma, ko so se zbirke oblikovale in ukinjale, se je prosti seznam napolnil in premešal. Zbirki dodeljeni zaporedni bloki so bili na disku dodeljeni daleč narazen in potrebno je bilo več iskalnih operacij za branje značilne zbirke.

UNIX 4.2BSD je razrešil problem učinkovitosti (zmogljivosti) zbirčnega sistema s prepričljivo boljšim razumevanjem diskovne geometrije, tako da je namestil zaporedne podatkovne bloke in z njimi povezano indeksirno informacijo na sosedna diskovna območja. Tako so zbirke v imeniku skoraj vselej shranjene na eni sami stezi ali diskovnem obroču in iskalni časi se izredno zmanjšajo. Tudi shranjevanje zbirčnih podatkovnih blokov v večjih paketih kot 512 zlogov je izboljšalo prenosne hitrosti.

Zbirčni sistem 4.2BSD uporablja bloke s 4096 zlogi, manjši blok se uporablja le za ostanek vsake zbirke. Branje zbirke z 32k zlogi se opravi z osmimi zapisi, ki se nahajajo na isti stezi oziroma obroču.



Slika 2. Razlika v hitrosti branja z diska staresa in novesa UNIXa je velika. V UNIXu 4.2BSD se zbirke shranjujejo v blokih 4096 zlogov namesto v 512-zložnih blokih. Vstavljanje fizičnih blokov na disk povečuje hitrost diskovnesa podatkovnesa dostopa.

5. Komunikacije

Prvotni operacijski sistem UNIX je bil razvit pred nastankom lokalnih mrež. Ta sistem je imel podporo samo za izmenjavo podatkov prek telefonskih linij in za komunikacijo med miniračunalniki in kabinetnimi sistemi; ni pa razpolagal z lastnostmi medprocesne komunikacije. Tkim. "cevna" (pipe) lastnost zsodnjesa UNIXa je vsebovala le lastnost zlogovnesa toka med procesi in posameznim strojem; ta sistem še ni bil posplošen za sporočilno usmerjeno komuniciranje s procesi in drugimi sostitelji.

Namesto nove arhitekture za lokalne mreže uvaja UNIX 4.2BSD podsistem za podporo mrežnesa protokola. Podpirajo se različne mrežne arhitekture v sistemu skladno z različnimi diskovnimi enotami. Tako kot so operacije branja, zapisovanja, iskanja in formatiranja skupne vsem diskovnim krmilnikom, tako abstrahira komunikacijski podsistem lastnosti, ki so značilne vsem komunikacijskim sistemom in oskrbi z njimi uporabnike, skrivajoč podrobnosti določenih mrežnih protokolov. UNIX 4.2BSD zsradi svoje komunikacijske storitve na vrhu tkim. zunanjih mrežnih pripomočkov. Zaradi tesa je ta izpeljanka UNIXa bolj odvisna od zunanjih mrežnih pripomočkov kot tisti operacijski sistemi, ki imajo opredeljene svoje lastne mrežne protokole. Po drugi strani pa je 4.2BSD bolj prožen pri prilagajanju na nova mrežna okolja.

Seveda bi bil lahko sistem 4.2BSD zsrajen brez mrežne podpore v svojem operacijskem jedru; namesto tesa bi lahko uporabljal mrežne storitvene procese, ki bi bili diostopani z drugimi aplikacijami z uporabo UNIXove izvirne medprocesne komunikacije. Ti procesi bi celo lahko neposredno posanjali mrežno materialno opremo. Vendar je bila ta metoda zavrnjena zaradi preobremenitve, ki se je pojavila v kontekstnem preklapljanju tedanje materialne arhitekture, tako da je bil bistveno zmanjšan mrežni podatkovni pretok. Tako je ostal UNIX odprt operacijski sistem, ki bo lahko upošteval sedanje in prihodnje mreže in njihove protokole.

6. Eno- ali dvosmerna povezava

UNIX 4.2BSD razpolaga z imenskimi prostori, v katerih se lahko procesi medseboj povezujejo, ko pošiljajo enosmerne podatkosrame, tj, nepotrjevane in zaradi tesa nezanesljive sporočilne tokove. Sistem se interpretira s podatkosrami ali prek vezij poslanih sporočil, toda pušča podatkovno pretvorne, predstavitvene in transformacijske elemente za ravnino aplikativnih programov.

Pripomočki sistema 4.2BSD so oblikovani kot podpora storitvenesa modela lokalnih mrežnih uslug. Mrežni viri se odzivajo na posamezne zahteve s podatkosrami ali pa oblikujejo dolgožive storitve z uporabo virtualnih vezij. Tu opisane komunikacijske pripomočke uporabljajo storilniki za oblikovanje storitev in uporabniki za dostopanje k njim.

Komunikacijski model sistema 4.2BSD uporablja grafno usmerjen komunikacijski prostor, v katerem se vrhovi grafa imenujejo podnožja in so končne komunikacijske točke. Poziv

```
s = podnožje(domena, tip)
```

oblikuje podnožje in vrne deskriptor 's', h kateremu bo uporabljena komunikacijsko usmerjena operacija. Podobno kot deskriptorje UNIXovih zbirk je mogoče manipulirati tudi s podnožnimi deskriptorji, npr. z uporabo DUP poziva, ki podvoji deskriptor ali s CLOSE pozivom, ki sa uniči. Argument 'domena' določa aplikativno sfero, v kateri se bo pojavila komunikacija in je izbran iz množice, podprte z lokalno izvedenko operacijskesa sistema. Operacije se interpretirajo z domensko implemetacijo in naj bi bile čimbolj neodvisne od domenske karakteristike. Sistem 4.2BSD podpira tako notranjo komunikacijsko domeno UNIXa kot zunanjo komunikacijsko domeno Xeroxovesa Internet protokola. Prihodnje domene naj bi podpirale družino protokolov Xeroxovesa mrežnesa sistema (XNS) in protokolov ECMA (European Computer Manufacturers Association).

7. Mrežna podnožja

Različni tipi prenosne semantike opredeljujejo podnožne tipe. Začetna izvedenka sistema 4.2BSD opredeljuje dva primarna podnožna tipa:

SOCK_IGRAM je podatkovnosramska povezava in
SOCK_STREAM je virtualno vezje.

Ko storilnik (server) oblikuje podnožje, mu mora podeliti ime, ki sa uporabljajo drugi procesi za povezovanje (kontaktiranje) s storitvijo. Podnožno ime se uporablja pri pošiljanju sporočila podnožju in pri oblikovanju povezave med podnožjema za izmenjavo več kot enesa sporočila. Naslov se dodeli podnožju s klicem

```
bind(s, ime)
```

V UNIXovi domeni so 'ime'na imena UNIXOvih zbirčnih sistemskih poti, v Internetovi domeni pa so dvojice, ki so sestavljene iz 32-bitnesa Internetovesa naslova in iz 16-bitne Internetove številke vrat.

Enostavne mrežne storitve s podatkosrami delujejo z oblikovanjem podnožij in s pošiljanjem in sprejemanjem podatkosramov, ko se uporabljata klica

```
sendto(s, sporočilo, k)
recvfrom(s, sporočilo, od)
```

Argumenti klica sendto so podnožje 's', iz katerega bo odposlan podatkogram; 'sporočilo', ki je sestavljeno iz polja zlogov, ki bodo odposlani; naslov sprejemnika 'k'. Podatkogram se lahko sprejme na podnožju, ki mu je bil dodeljen naslov 'k' s klicem 'bind'. Naslov podnožja, iz katerega je bil odposlan podatkogram, je določen na sprejemnem podnožju kot 'od'.

V najbolj enostavnem odnosu storilnik-naročnik oblikuje storilnik podnožje in ga veže na dobro znano ime. Naročnik, ki si želi stik s storitvijo, oblikuje podnožje in izvrši sendto klic, ko pošlje sporočilo k storilniku. Storilnik sprejme sporočilo z uporabo klica recvfrom in lahko odpošlje odgovor z uporabo naslova 'od'. Komunikacijski podsistem 4.2BSD priredi 'ime' podnožju, ki se bo uporabljalo za pošiljanje sporočil, če mu ni bilo nič prirejeno s klicem bind. Tako obstaja vselej naslov, h kateremu se lahko pošiljajo odgovori.

Mrežno povezana storitev se oblikuje s podnožjem, z njegovo povezavo k naslovu in z izdajo klicev

```
listen(s, backlog)
naročnik = accept(s, od)
```

Funkcija listen naroči sistemu, da se vrstno poveže s podnožjem 's', ko je 'backlog' največje dovoljeno število hkratnih vrstnih povezav. Storilnik deluje z ukinjanjem povezav v vrsti in z njihovim popravljanjem. Povezave se ukinjajo s klicem

```
s = accept(q, od)
```

ki vrne podnožni opisovalnik.

Ta vrstni povezovalni model predpostavlja, da so bile povezave zgrajene pred njihovim sprejetjem; storilnik ne more izbirno zavrniti povezave pred klicem 'accept'. Tako se sporočilo poveži-potrdi prenosnega protokola odpošlje, ko je bila povezava uvrščena. Sporočilo poveži-zavrni pa se pošlje, ko ni več prostora v vrsti ali ko naslovljeni storilnik ni aktiven.

Domensko neodvisne primitivne funkcije tega modela ne zahtevajo znanje o mehanizmih gradnje povezav, ki nastajajo z uporabo klicev poveži-sprejmi (connect/accept). Kadar je potreben zapleten protokol za zgraditev povezave v komunikacijski mreži, se to lahko doseže s preslikavo povezovalne sprejemljivosti na začetno stopnjo protokola in z graditvijo dodatnih povezav v aplikacijskih procesih.

Klic accept se lahko izdaja ponavljajoče z ukinjanjem vstopov v povezovalni vrsti. Naročnik, ki želi stik s storilnikom, oblikuje podnožje in določi ime storilnika s klicem

```
connect(s, k)
```

kjer je 'k' naslov storilnika.

Ko je tako povezava zgrajena, se lahko uporabijo navadni UNIX klici za branje in zapisovanje na vezju. Daljinske storitve log-in in file-transfer se oblikujejo z uporabo lastnosti vezja.

Komunikacijski programi morajo često multipleksirati več hkratnih V/I aktivnosti. V UNIX 4.2 BSD je to omogočeno na dva načina: uporabi se lahko mehanizem za izbiro množice V/I aktivnosti, ki se izvajajo sinhrono brez blokiranja ali s programiranimi prekinitvami, ko se pojavi programska prekinitev pri pripravljenem vhodu ali ko je omogočen izhod prek predhodno blokiranega kanala.

8. Praktični dokazi

Izkušnje so pokazale, da so zmogljivosti sistema UNIX 4.2BSD v lokalnih mrežah odlične. Na procesorjih VAX 11/750 in MC68000 (10 MHz) so bili doseženi protokolski časi pod 1 ms za paket (tokovni protokol) in hitrosti do 1000 paketov v sekundi (za podatkograme). Te hitrosti so do petkrat večje kot pri TCP/IP (Transmission Control Protocol/Internet Protocol) na podobnih procesorjih.

Različni mrežni protokoli so še v razvoju. Več skupin razvija podporo za tiskalnike, zbirčne storilnike, poštne protokole in standarde. UNIX 4.2BSD lahko sprejme katerikoli protokol in njegove aplikacije, ker je lahko prilagodljiv za nove protokole.

Podjetje Sun Microsystems gradi storitveno mrežo za svojo delovno postajo s procesorjem MC68010 in z UNIX 4.2BSD. Možna je tudi povezava z uporabo sistema Ethernet (VAX in delovne postaje s sistemom UNIX 4.2BSD).

9. Pomanjkljivosti standardnih operacijskih sistemov tipa UNIX

Zapletena organizacija operacijskega sistema UNIX (Bell Laboratories) je kot dvorezen meč: medtem ko nudi uporabniku prvovrstne storitve, znižuje zmogljivosti obstoječih programskih struktur. Prva posebnost je npr. večopravilna lastnost sistema UNIX, v okviru katere lahko obstaja več hkratnih procesov v pomnilniku in pri časovnem dodeljevanju procesorja, kot sta npr. diskovni V/I in komunikacije. Dokler posamezna opravila ne potrebujejo večjih pomnilniških količin, ima ta lastnost določene prednosti.

Standardni UNIX je izmenjevalni sistem: kadar mora pomakniti del uporabniškega procesa na disk zaradi pomanjkanja prostora v glavnem pomnilniku, mora kopirati celoten proces. Ta taktika se lahko izkaže kot pomanjkljivost, kadar želi uporabnik izvrševati več sočasnih procesov in je za vsakega od njih potrebna izmenjevalna operacija. V tem primeru lahko pride do pojava, ki ga označimo z zlomom sistema, ko so viri v celoti zasedeni in nastopi pomikanje procesov med diskom in glavnim pomnilnikom.

Alternativa k tej izmenjevalni proceduri je tkim. stranenje na zahtevo. Tu se posamezne strani procesa sibljejo na disk in iz diska v odvisnosti od tega, kaj proces v danem trenutku potrebuje. V tem primeru se lahko uporabljajo tudi programi, ki presegajo obseg fizičnega pomnilnika.

Izvedenke UNIXa v naslednjih letih bodo imele lastnost stranenja. Berkeleyska izvedenka UNIXa to lastnost že ima. Zagonski čas velikih interaktivnih programov se lahko zmanjša z branjem strani novega procesa po potrebi, namesto da se kopira celoten proces v pomnilnik pred njegovim izvrševanjem.

Svojo lastnost multiprocesiranja uporablja UNIX svobodno. Njegov interaktivni ukazni interpret (lupina OS) vzpodbuja uporabnike, da oblikujejo kratkožive procese. Nekatere aplikacije lahko z uporabo lupine oblikujejo več procesov na sekundo. Lupina oblikuje procese v dveh korakih.

Najprej naredi kopijo (z uporabo vilične operacije) same sebe, nato pa prekrije sebe (z uporabo exec operacije) z novim procesom.

Na močno obremenjenem sistemu izdeluje jedro OS večkrat odvečne kopije lupine (ali natančneje njegovih zapisljivih podatkovnih območij), ki jih pomika (izmenjuje) med diskom in pomnilnikom. Zmogljivost je mogoče povečati (izboljšati) s tako lupino (in z drugimi programi, ki oblikujejo procese), ki proizvajajo procese brez nepotrebnega diskovnega kopiranja.

Tudi upravljanje diskovnega V/I z izmenjevanjem ni UNIXova značilna lastnost, čeprav zmogljivo zbirčno upravljanje znatno pomaga uporabniku. UNIXovi imeniki so lahko hierarhično vsnezdeni do poljubne globine in lahko vsebujejo zbirke in nadaljne podimenike. Več podatkov in besedila se lahko doda k poljubni zbirki s pripisovanjem in kadarkoli se lahko oblikujejo nove zbirke in imeniki.

V tej uporabniški prijaznosti UNIXa pa se skrivajo nekateri problemi njegove zmogljivosti. Ker uporabniku nikoli ni potrebno določiti obsega zbirke, mora operacijski sistem naključno dodeljevati pomnilniške bloke zbirki, ki narašča. Naključno dodeljevanje pa povzroča razpršenost zbirčnih blokov na disku. Ker uporabnik nikoli ne navede obsega imenika, se tudi imeniški bloki razpršijo po disku z rastjo imenika. Tako je potrebnih več iskalnih operacij za posamezno komponento. Velik zbirčni sistem lahko ima imena z 10 do 15 komponentami in odprtje take zbirke lahko zahteva 30 diskovnih iskanj, 30k-zložni prenos z diska v pomnilnik in še pripadajočo obdelavo.

Zmogljivost zbirčnega sistema se poslabša tudi tedaj, ko je dovoljena neomejena rast nekaterih imenikov. Imenik ni dejansko nič drugega kot seznam zbirčnih in podimeniških imen s kazalci k ustreznmih naslovom na disku. Navedba imeniškega člena pomeni preiskovanje imenika, dokler ni najdeno ime. Če ima imenik 1024 členov, je za značilno navedbo potrebnih 10 diskovnih iskanj in blokovnih branj: osem iskanj za branje polovice imenika in eno ali dve iskanji za branje imeniškega i-vozlišča.

V povezavi z zmogljivostnimi problemi diskovnega V/I povzroča branje znaka v ali iz terminala posebno prekinitev centralne procesne enote (to velja za vrsto sistemov UNIX). Kadar je aktivnih več uporabnikov z zaslonsko usmerjenim urejevalnikom, ki pogosto preslikuje zaslon, postanejo znakovne prekinitve glavna obremenitev procesorja. Nekateri UNIXovi sistemi uporabljajo posebno IMA materialno opremo, s katero se prenaša v posamezni prekinitvi več znakov. Nekateri sistemi uporabljajo vhodne vmesnike, ki se odtipavajo z omrežnim taktom (60 Hz), da se tako izognejo vhodnim prekinitvam.

Ti poskusi za izboljšanje zmogljivosti pa imajo stranske učinke, tako da XON-XOFF protokol (usposobitev in onesposobitev oddaje) ne deluje več zanesljivo. Več zunanjih naprav, kot so npr. tiskalniki, uporablja ta protokol za krmiljenje svojih vhodov. Ko se vmesnik približuje svoji meji, pošlje zunanja naprava signal XOFF sostitelju z namero, da ustavi oddajanje. Če ima naprava več vmesniškega prostora, pošlje signal XON za sprejem iz UNIXovega sostitelja. Vmesniški terminalni V/I pa lahko zakasni signal XOFF in povzroči izgubo podatkov.

A.P.Železnikar

MOJ MIKRO

Junij 1984

MOJ MIKRO sta pripravila uredniško revije TELEKS in software redakcija RADIA ŠTUDENT ● Izdaja in tiska ČGP DELO, tozd Revije, Titova 35, Ljubljana ● Predsednik skupščine ČGP Delo JAK KOPRIVC ● Glavni urednik ČGP Delo BORIS DOLNIČAR ● Direktor tozda Revije BERNARDA RAKOVEC ● Glavni in odgovorni urednik revije Teleks VILKO NOVAK – Urednica junijske številke revije Moj mikro CIRIL KRAŠEVEC IN ŽIGA TURK ● Oblikovanje in tehnično urejanje ANDREJ MAVSAR, FRANCI MIHEVC ● Naslov uredništev Teleks, Radio Študent, Ljubljana, Študentsko naselje, blok 8, telefon 216-965 – Oglasi: Stik, oglasno trženje, Ljubljana, Titova 35, telefon h. c. 315-365 ● Cena številke revije Moj mikro 200 din ● MOJ MIKRO je oproščen plačila posebnega davka po mnenju republiškega komiteja za informiranje, dopis št. 421-1/72 z dne 25. 5. 1984

dr Suad ALAGIĆ
vanredni profesor Elektrotehničkog fakulteta u Sarajevu

RELACIONE BAZE PODATAKA

Prvo izdanje

„SVJETLOST",
OOUR Zavod za udžbenike i nastavna sredstva,
Sarajevo, 1984.

PREDGOVOR

U razvoju metoda projektovanja i efikasnog korišćenja informacionih sistema prelomnu tačku, tačku u kojoj ova oblast prerasta u formalno zasnovanu naučnu i inženjersku disciplinu, predstavljaju modeli podataka, odnosno baze podataka. Eksplicitno iskazivanje činjenice da informacioni sistem, da bi zadovoljio sve zahteve koji se pred njega postavljaju, mora da predstavlja adekvatan model realnog sistema u kome deluje, donosi revoluciju u pristupu projektovanju informacionih sistema. Relacioni model baze podataka, kao prvi model sa sveobuhvatno definisanim formalnim aparatom, s jedne strane, i izuzetno jednostavnim i prihvatljivim konceptima, sa druge, u ovim trendovima razvoja ima najznačajnije mesto i kao teorijska osnova razvoja i kao, u poslednje vreme, konkretna praktična realizacija. Ova knjiga se znači bavi veoma značajnim teorijskim i praktičnim aspektima projektovanja i korišćenja informacionih sistema.

Knjiga sadrži pet poglavlja: Modeli i jezici, Logičko projektovanje, Strukture i algoritmi, Integritet podataka i Distribuirane baze.

Prva dva poglavlja tretiraju relacioni model kao takav (strukturu, ograničenja i operatore) i logičko projektovanje relacionog modela (normalizaciju i apstrakciju). Preostala tri poglavlja tretiraju fizičku realizaciju mode.. (strukture i algoritmi), osnovne probleme rada u dinamičkom okruženju (konkurentno izvršavanje transakcija) i u poslednje vreme praktično najinteresantniji slučaj realizacije — distribuirane baze podataka.

U poglavlju Modeli i jezici uveden je prvo relacioni model podataka (predstavljanje entiteta relacijama, relaciona algebra, relacioni upitni jezici), zatim su razmatrane logičke zavisnosti između atributa relacija (funkcionalne i višeznačne) i na temelju takvog pristupa uvedeni su hijerarhijski i mrežni model.

Poglavlje Logičko projektovanje sadrži dva dela, jedan je posvećen normalnim oblicima, a drugi se bavi semantičkim postupcima u logičkom projektovanju. Od normalnih oblika razmatrane su druga, treća, četvrta i Boyce-Coddova normalna forma. Semantički aspekti modeliranja uključuju analizu prednosti i nedostataka nenormalizovenih modela, a zatim i postupaka agregacije i generalizacije. Uslovima integriteta je u oba postupka posvećena posebna pažnja, a u okviru generalizacije je razmatrano i uvođenje podmodela.

Poglavlje Strukture i algoritmi posvećeno je fizičkoj (strukturalnoj) reprezentaciji modela. Kod toga su razmatrane sekvencijalna, indeks-sekvencijalna, indeksirana, direktna i mrežna reprezentacija. Za svaku reprezentaciju razvijeni su algoritmi za realizaciju restrikcije, projekcije, i spajanja. Data je analiza strukture dinamičkih indeksa tipa B-drveta.*

Integritet podataka i njegovo respektovanje kod uporednog izvršenja skupa transakcija nad bazom podataka predmet su četvrtog poglavlja. Razmatrani su protokoli zaključavanja objekata u bazi, logički i fizički lokoti, a zatim i postupci uspostavljanja konzistentnog stanja baze nakon povrede uslova integriteta.

Poslednje poglavlje posvećeno je distribuiranim sistemima baza podataka. Izložena je, na funkcionalnom principu, arhitektura takvih sistema, razmatrani su problemi integriteta podataka kod distribuiranog izvršenja skupa transakcija, postupci distribuirane obrade upita i protokoli kojima se obezbeđuje konzistentno ažuriranje višestrukih kopija podataka u distribuiranoj bazi.

Za knjigu se u celosti može reći da sadrži obiman materijal koji se do sada mogao naći samo u visokospecijalizovanim časopisima. Ona je rezultat samostalnog pristupa dr Suada Alagića problemima sistema baza podataka, nije nastala ugledanjem na već postojeći, domaći ili strani, uzor, ali se veoma dobro poredi sa najpoznatijim (Date, Ullman).

Autor je pravilno izabrao najmoderniji, relacioni model, i formalni aparat koji za njega ide, kao osnovu pristupa problemima baza podataka. Tim jednostavnim formalnim aparatom dokazan je niz značajnih rezultata iz teorije baza podataka, a istovremeno su upotrebom tog aparata sistematski i na originalan način izloženi različiti oblici fizičke reprezentacije.

Skup algoritama koji knjiga sadrži prvi put u ovom obliku razjašnjava probleme realizacija relacionih upitnih jezika, a neki algoritmi se po prvi put objavljuju.

Nasuprot većini do sada objavljenih knjiga iz baza podataka, posebna pažnja je posvećena integritetu podataka i problemima usklađivanja uporednog izvršenja skupa transakcija nad centralizovanom i distribuiranom bazom podataka.

Poseban kvalitet knjige u pedagoškom smislu je izbor izuzetno velikog broja primera koji su korišćeni kod svakog, formalnog ili neformalnog, objašnjenja. Pristup u knjizi je formalan gde materija to zahteva, ali je i tada lak za čitanje i uvek potkrepljen primerima koji se sistematski daju pre formalnih objašnjenja. Oni aspekti knjige koji su od posebnog interesa za neformalne korisnike baza podataka, kao što su upitni jezici ili semantički modeli, naročito su detaljno analizirani, pa to knjizi daje još veću širinu. Čitav niz praktičnih postupaka, kao što su oni za uspostavljanje konzistentnog stanja baze podataka i algoritama, biće od nesumljive koristi stručnjacima koji se u praksi bave bazama podataka, bez kojih je teško zamisliti moderno poslovanje složenih organizacija udruženog rada u privredi i društvenim delatnostima. Knjiga je napisana tako da može da posluži kao udžbenik za dodiplomske i poslediplomske studije na tehničkim, matematičkim, organizacionim i ekonomskim fakultetima.

Reč je, dakle, o retkom podulvatu u informatici, disciplini koja je od veoma velikog značaja za dalji razvoj, o knjizi zasnovanoj na modernim trendovima, koja će biti aktuelna dugi niz godina. Zato mi je zadovoljstvo da predstavim ovu knjigu, prvu knjigu ovakvog sadržaja u nas, izuzetnu i u poređenju sa najboljim stranim u ovoj oblasti.

Prof. dr Branislav Lazarević

SADRŽAJ

Predgovor 5

Uvod 7

I. Modeli i jezici 11
1. Relacioni model 11
1.1. Predstavljanje entiteta relacijama 11
1.2. Relaciona algebra 16
1.3. Relacioni upitni jezici 19
2. Logičke zavisnosti 25
2.1. Funkcionalne zavisnosti 25
2.2. Višeznačne zavisnosti 29
3. Hijerarhijski i mrežni model 33
3.1. Hijerarhijski model 33
3.2. Mrežni model 37

II. Logičko projektovanje 42
1. Normalne forme 42
1.1. Druga normalna forma 42
1.2. Treća normalna forma 45
1.3. Boyce-Coddova normalna forma 48
1.4. Četvrta normalna forma 50
2. Apstrakcije 53
2.1. Nenormalizovani relacioni model 53
2.2. Agregacija 55
2.3. Generalizacija 60

III. Strukture i algoritmi 67
1. Fizička reprezentacija logičkih skupova 67
2. Sekvencijalna reprezentacija i sekvencijalni algoritmi 70
2.1. Sekvencijalna reprezentacija 70
2.2. Sekvencijalni algoritmi 74
3. Indeksirane reprezentacije 79
3.1. Indeks-sekvencijalna reprezentacija 79
3.2. Indeksirana reprezentacija 84
3.3. Struktura indeksa 89
3.4. Direktna reprezentacija 95
4. Mrežna reprezentacija 98

IV. Integritet podataka 102
1. Trarsakcije i integritet 102
2. Izvršenja skupa transakcija 104
3. Protokoli zaključavanja 107
4. Logički lokoti 115
5. Fizički lokoti 120
6. Restauracija konzistentnog stanja baze 125

V. Distribuirane baze podataka 134
1. Arhitektura sistema baze podataka 134
2. Distribuirano izvršenje i integritet 138
3. Distribuirana obrada upita 146
4. Distribuirano ažuriranje 154

Bibliografske pribilješke 159
Literatura 161
Indeks 163

GLASILO DELOVNEGA KOLEKTIVA SOZD ISKRA Številka 25 – Leto XXIII – 16. junij 1984

Peta računalniška generacija prihaja

V začetku devetdesetih let tega stoletja bosta ZDA in SZ dogradili stalni orbitalni postaji za nadaljnje odkrivanje vesolja in za osvajanje novih tehnologij. To je vznemirljiv projekt, ob katerem lahko marsikdo pripomni: „Tako kmalu, le borih sedem, osem let!" Da, današnji razvoj napreduje res zelo hitro in to ne le pri osvajanju vesolja. Še pomembnejši in bolj neverjetni projekti bodo uresničeni v prihajajočem razdobju. Eden takih, morda celo najpomembnejših, bo nekakšna orbitalna postaja *izgradnje pete računalniške generacije; ta projekt naj bi bil v prvi fazi* uresničen do leta 1992 v hudem tekmovanju med Japonsko, ZDA in EGS.

Danes ni moč niti približno predvideti, kaj vse bo tu nova tehnologija prinesla; vendar je takojšnje prilagajanje na nove tehnologije in metodologije nujno, da bi preprečili neprijetna presenečenja v tistih tehnoloških dejavnostih, ki so usmerjene v izvoz.

Nikakor ne bi smeli dopustiti, da bi se šele čez sedem, osem let začeli presenečeno ozirati naokoli in spraševati, od kod se je za nas tako nenadoma pojavila ta peta računalniška generacija, na kateri bo osnovana *robotizacija proizvodnje in informatizacija življenja nasploh.*

Zavest o prihodu pete računalniške generacije in njenih posledic je nujna in naj bi postala stabilna zavest našega delavca, da tako ne bi ostalo pri nemočnem vzkliku (vzdihu) upravljavske baze: „Bo že kako!"

V smislu takih razmišljanj brž kone ne bo odveč naš pogovor z vidnim računalniškim strokovnjakom in sooblikovalcem mikroračunalniškega sistema Partner, prof. dr. Antonom P. Železnikarjem.

Kaj je osnovna značilnost pete računalniške generacije?

Železnikar: Peta računalniška generacija naj bi bila zmogljivejša predvsem na področjih človekovih intelektualnih (umskih, razumskih; logičnih, oblikovalnih) dejavnosti. Bila naj bi to, kar bi upravičeno lahko imenovali inteligenčni ojačevalnik, oz. celo inteligenčni pospeševalnik.

Peta računalniška generacija bo po definiciji mrežni računalniški sistem, ki naj bi prek lokalnih (podjetniških, laboratorijskih, področnih) mrež prerastel v planetarno mrežo s hitrimi in učinkovitimi komunikacijskimi potmi. Moč pete generacije naj bi bila v zaseganju tiste obdelovalne moči v mrežnem sistemu, ki je za rešitev določenega problema v določeni mrežni točki potrebna. Področij človekovega intelektualnega dela pa je več vrst (specializacij), kot si to navadno predstavljamo.

Bistvo nove računalniške generacije naj bi bila med drugim lastnost logičnega oblikovanja, ekstrapolacije, sklepanja, oz. kar reševanja problemov. Oblikovanje novih pojmov, njihovo povezovanje na različnih delovnih področjih, samostojno izbiranje in iskanje podatkov v velikih informacijskih bazah naj bi pripomoglo k samostojnemu reševanju s človekom in sistemom oblikovanih raznovrstnih problemov.

V začetku naj bi bile takšne zmogljivosti nove generacije omejene na posebna področja (npr. na razvojne in proizvodne probleme, ključne tehnologije, konstrukcijske dejavnosti, standardizacijo, medicino, tehniko, znanost, umetno inteligenco), kasneje pa zaokrožene v vedno večje celote.

Tako bi imeli določene strokovne, ožje usmerjene sisteme pete generacije (npr. za telefonijo, interno medicino). Prek računalniških mrež bi bilo mogoče v vsakem primeru, tudi če je obdelovalno mesto v mreži navaden terminal (zaslon in tipkovnica), mobilizirati ustrezno obdelovalno moč v določenem, dovolj zmogljivem mrežnem vozlišču.

S takšno obdelovalno strategijo bi lahko ob sodelovanju dovolj močnih in specializiranih vozlišč centrov rešili problem poljubne težavnostne stopnje. Za reševanje problemov bi sistem razpolagal z novim tipom podatkovne, oz. že informacijske baze, ki se imenuje baza znanja, oz. relacijska baza znanj na določenem področju dela, igre, dejavnosti.

Peta generacija uvaja dejansko pojem informacijske in ne le podatkovne baze. Podatek povzroči pri določenih pogojih nastanek informacije o tem podatku, njegovih relacij z drugimi podatki in informacijami v odvisnosti od danega problema. Informacija o podatku zajame tako njegov pomen v odvisnosti z drugimi podatki in podatkovnimi povezavami. Informacija postane tako nadgradnja podatka, razumeti pa jo moramo tudi dinamično, spremenljivo v odvisnosti s problemi, ki so v ospredju trenutne problemske pozornosti.

Uporaba sistemov pete generacije bo povzročila zaradi njihovih izrednih zmogljivosti bistvene spremembe v organiziranosti, kakovosti, raznovrstnosti, procesnosti, zapletenosti in zanesljivosti industrijske velikoserijske in maloserijske proizvodnje. Računalniško avtomatizirana in robotizirana proizvodnja bo prevzela večino fizičnega dela in enoličnih ponavljajočih opravil *že* pred nastopom pete generacije, s peto generacijo pa se bo začela prava tehnološka in organizacijska revolucija proizvodnih procesov (industrijskih, načrtovanih, razvojnih, individualnih). V neposredni proizvodnji bo človekova prisotnost minimalna in se bo postopno še zmanjševala.

Kljub vizijam revolucionarnega nastopa pete računalniške generacije bo razvoj računalniške tehnologije zvezen, postopen in v mejah obstoječe tehnološke revolucije, temelječ na zmogljivostih današnje in jutrišnje elektronske in informacijske (programske) tehnologije. Vendar bo ta razvoj znatno pospešen, hitrejši in bo zahteval večjo hitrost in inteligenco prilagajanja.

Ali bomo ob našem sedanjem in *jutrišnjem pospešenem računalniškem* razvoju čez nekaj let vendarle neprijetno presenečeni?

Železnikar: Pojav pete računalniške generacije nas bo na določen način vrgel iz utečenih (tradicionalnih) delovnih kolesnic, vendar ne nujno na neprijeten način. Potrebna bo natančnejša strategija hitrega prilagajanja (kopiranja, prevzemanja, osvajanja, prisvajanja, sprejemanja novih tehnologij in metodologij), seveda z učenjem in poučevanjem bistvenih tehnoloških, programirnih in organizacijskih dejavnosti.

Lahko pa bi čez sedem, ali osem let postalo boleče tudi spoznanje, da v tem pripravljalnem obdobju nismo ustrezno ukrepali, da problema pete računalniške generacije nismo postavili v ospredje naše gospodarske, družbene, politične, obveščevalne in planske pozornosti.

Razvoj in tehnologija pete generacije dejansko zahtevata planiran gospodarski pristop zaradi ustrezne koncentracije nalog, kadrov, sredstev in organizacije, tudi pri skromnih človeških in materialnih virih.

Zaradi določenih pomislekov naše današnje tehnološke in gospodarske naravnanosti bomo prejkoslej morali korigirati obstoječe planske dokumente, še posebej tiste, ki segajo v naslednje desetletje ali pa celo do leta 2000.

Toda kaj lahko naredimo?

Železnikar: Naredimo lahko veliko več, kot smo v tem trenutku pripravljeni storiti. Najprej je potrebna tekmovalna *priprava in sicer tehnološka*, organizacijska, marketinška, izobraževalna. Če se sprijaznimo z izhodiščem, da je tehnološko tekmovanje zdravo, poživljajoče, zanimivo in se vključujemo v tekmovalni sistem trendov (tehnoloških tokov) in strategij, za katerimi stojijo naši ljudje, močno usposobljene in zdržljive strokovne skupine z različnih področij dela in družbenj, *gospodarski in izobraževalni dejavniki* kot celota, potem v bistvu že začenjamo reševati probleme usmerjanja v nove možnosti dela in bivanja.

Tako začenjamo bolje izkoriščati tudi umske sposobnosti delavcev, dvigujemo njihovo delovno pripravljenost in zadovoljstvo. Zlagoma prihajajo v ospredje dejavniki, kot so delovna spontanost, neprisiljenost in naposled *sposobnost (zrelost, obvladovanje nas samih).*

Za uspešno tehnološko tekmovanje so določene sposobnostne stopnje nujno potrebne, sposobnost se mora izražati na najzahtevnejših delovnih področjih. Tekmovalnost lahko marsikje olajša delovne napore zaradi prisotnosti motivacijskih dejavnikov, delovne smiselnosti in seveda tržne, *razvojne in proizvodne uspešnosti.* Nekdo je pripomnil: „Pustimo občasno tudi sposobnim njihovih pet minut".

Kadar govorimo o peti računalniški generaciji, menda ne moremo mimo Japoncev?

Železnikar: Japonci so de facto osredotočili svoje napore v osvojitev cilja, za katerega še sami ne vedo, kako ga bodo osvojili. Japonci so v zadnjih 25 letih dokazali, da so izredno hitro prilagodljiv narod, z veliko mero vztrajnosti, pridnosti in tehnološke nadarjenosti. V tej smeri so naredili in dosegli več kot druga, bolj razvita tehnološka okolja. Japonsko načelo kopičenja sposobnosti, sposobnostne kadrovske in materialne organizacije in usklajenosti je naposled rodilo učinkovito in uspešno delujočo industrijo, ki je v računalništvu dosegla svoj vrhunec.

Japonsko tveganje v osvajanju pete računalniške generacije se je tako spremenilo v motivacijo za njihovo celotno industrijo (socioekonomski kompleks). Japonski primer tudi kaže, kako je mogoče usklajevati interese velikih gospodarskih kompleksov, kako je mogoče dosegati delitev nalog s skupnimi cilji in kako je mogoče koncentrirati kapitalne vire bank, državne uprave in podjetij. Peta računalniška generacija je za Japonce dolgoročni gospodarski načrt, ki ga v poslednjem času kopirajo tudi drugi (Američani, Britanci, Francozi, Nemci).

Kaj po vsem tem ostaja nam?

Železnikar: Usmeritev kadrovskih virov, nastajanja znanja in razvijanja tehnoloških sposobnosti naj bi bila vzpodbujana načrtno (plansko), individualno in organizacijsko. Delitev dela med domačimi in tujimi organizacijami je nujna, nujen je princip prilagajanja v tehnološkem smislu, saj se prav z njim dosegajo največji razvojni učinki. Le tako je mogoče ob pičlih materialnih sredstvih dosegati zadovoljive rezultate. Samo kadri niso dovolj, potrebni so sposobni kadri, ki imajo zadostno percepcijo (občutljivost za spremembe v tehnološkem okolju), zadostno izkušenost (pridobljeno delovno sposobnost), stalen in samostojen dotok znanja (individualno pridobivanje znanja z delom in obveščenostjo) in delovno neuklonljivost (samozavest, napredujočo moralo). Okoli sposobnostnih kadrovskih jeder naj bi se gradile in razširjale delovne skupine in zapleteni proizvajalni procesi.

Vse to je laže povedati, kot uresničiti, tudi zaradi sedanjega kadrovskega stanja. Imeli smo in še imamo nekaj ključnih, zgrešenih investicij v razvoj, proizvodnjo, marketing in celo v organizacijsko strukturo. Kako lahko planiramo ob pomanjkanju vizije svetovnega tehnološkega razvoja, oz. kako naj bi prišli do svoje, naprednejše razvojne vizije?

Železnikar: Tisti, ki odločajo (vodilni delavci, strokovnjaki), navadno nimajo strokovne in poglobljene predstave o svetovnem razvoju na svojem področju dela, saj take predstave v našem dosedanjem prostoru niti niso bile potrebne.

Gospodarske učinke smo lahko dosegali z negospodarskim orodjem, z zadolževanjem, prelaganjem bremen, s socializacijo na različnih ravninah.

Mischie osveževitve so potrebni okosteneli upravni aparati (zamenjava kadrov) pa tudi nacionalna znanost in raziskave, ki svoje funkcije že nekaj časa ne opravljajo več. Zaradi tega se nam bo zavest o potrebi tehnološkega napredovanja dovolj okrepila šele do konca tega desetletja, ko se bodo razmere tako zaostrile, da bomo iskali izhod v sili.

Vizija našega tehnološkega, oz. tudi računalniškega napredka še bo tako izoblikovala šele proti koncu tega desetletja, ko bomo brezkompromisno ukinili življenje neuspešnih, polovičnih, nezrelih, nedomišljenih in materialno ne dovolj podprtih današnjih usmeritev. Medtem pa moramo zlasti na strokovnem področju narediti vse, da bodo dotekale kvalitetne in za nas bistvene informacije iz sveta tehnološko razvitih okolij.

Kakšna naj bi bila vloga politike, ko je peta računalniška generacija že na vidiku z vsemi posledicami njene *uporabe?*

Železnikar: Razvite kapitalistične države so začele uvajati na področju tehnološkega osvajanja pete računalniške generacije državno planiranje razvoja raziskav, proizvodnje in sredstev za plansko realizacijo.

Japonci so bili v tem prvi in so pokazali, s kakšno organizacijo je to mogoče (v okviru ministrstva za mednarodno trgovino in industrijo – MITI). Tudi Velika Britanija je postavila svoj državni plan skozi politično pobudo.

Evropska gospodarska skupnost je oblikovala skupen projekt (ESPRIT) s finančno podporo skupnosti in velikih podjetij. ZDA so koncentrirale sredstva za izvajanje projektov pete računalniške generacije pod okriljem obrambnega ministrstva. Zahodna Nemčija postavlja pravkar svoj petletni načrt razvoja in proizvodnje novih računalniških sistemov. V Franciji je intervencija za osvajanje novih tehnologij pete generacije državna in družbena, posega pa zlasti tudi na področje izobraževanja.

Pri nas doslej nismo zasledili družbene, podjetniške, izobraževalne, ali podobne pobude za spremljanje tujih in načrtovanje naših možnosti osvajanja pete računalniške generacije. Obstajajo pa individualne pobude strokovnjakov, ki opozarjajo na sprejetje določenih planskih in organizacijskih ukrepov.

Bržkone pa je politična aktivnost za določene premike v smeri hitrejšega računalniškega napredovanja pri nas šele na svojem začetku.

Boris Čerin

74192	2.200	74LS95	3.000	74LS221	2.800
74193	2.000	74LS107	1.500	74LS240	3.000
		74LS109	1.100	74LS241	3.000
		74LS112	1.100	74LS242	3.000
		74LS113	1.100	74LS243	3.000
74LS00	1.100	74LS114	1.100	74LS244	4.800
74LS01	1.000	74LS122	1.650	74LS245	6.400
74LS02	1.000	74LS123	3.000	74LS247	3.200
74LS03	900	74LS124	3.100	74LS248	3.200
74LS04	1.450	74LS125	1.300	74LS249	3.200
74LS05	900	74LS126	1.300	74LS251	2.000
74LS08	900	74LS132	1.500	74LS253	1.900
74LS09	900	74LS133	1.100	74LS256	2.700
74LS10	900	74LS136	1.100	74LS257	1.700
74LS11	900	74LS138	1.400	74LS258	1.700
74LS12	900	74LS139	2.000	74LS259	2.300
74LS13	900	74LS145	2.800	74LS260	1.000
74LS14	1.600	74LS147	5.000	74LS266	1.100
74LS15	900	74LS151	1.400	74LS273	3.200
74LS20	900	74LS152	2.000	74LS279	1.600
74LS21	900	74LS153	1.400	74LS280	5.500
74LS22	900	74LS154	2.400	74LS283	2.000
74LS26	1.000	74LS156	1.550	74LS289	4.400
74LS27	900	74LS156	2.700	74LS290	1.400
74LS28	1.000	74LS157	1.500	74LS293	2.800
74LS30	1.200	74LS158	1.500	74LS295	2.200
74LS32	1.000	74LS160	1.850	74LS298	2.200
74LS33	1.100	74LS161	3.100	74LS323	7.500
74LS37	1.100	74LS162	1.900	74LS352	2.200
74LS38	1.600	74LS163	2.000	74LS353	2.300
74LS40	1.100	74LS164	1.700	74LS362	8.800
74LS42	1.500	74LS165	3.400	74LS365	1.400
74LS47	2.500	74LS168	2.000	74LS366	1.600
74LS48	2.500	74LS169	3.800	74LS367	1.600
74LS49	2.500	74LS170	4.000	74LS368	1.600
74LS51	1.100	74LS173	1.900	74LS373	3.900
74LS54	900	74LS174	2.600	74LS374	4.650
74LS55	900	74LS175	1.400	74LS377	3.000
74LS73	1.900	74LS181	4.200	74LS378	2.000
74LS74	2.100	74LS189	4.100	74LS379	2.000
74LS75	1.650	74LS190	2.000	74LS386	1.100
74LS76	1.650	74LS191	2.000	74LS390	3.000
74LS78	1.650	74LS192	2.000	74LS393	2.400
74LS83	2.350	74LS193	2.000	74LS395	2.200
74LS85	2.000	74LS194	2.000	74LS490	3.000
74LS86	1.550	74LS195	1.900	74LS502	4.800
74LS90	1.500	74LS196	1.900	74LS670	5.000
74LS92	1.500	74LS197	1.900		
74LS93	1.500				

Nekatere zanimive knjige

Chris Crawford: The Art of Computer Game Design. Osborne/McGraw-Hill. $14,95.

J.S.Mallozzi, N.J. De Lillo: Computability with Pascal. Prentice-Hall. $26,95.

J.E.Volkstorf: Graphics Programming on the IBM Personal Computer. Prentice-Hall. $39,95.

R.Haskell, G.A.Jackson: IBM PC Basic Programming. Prentice-Hall. $13,95.

E.H.Carlson: Kids and the IBM-PC/PCjr. Datamost. $19,95.

D.S.Walonick: A Library of Subroutines. Scott, Foresman and Co., $19,95.

B.Kay: Mathematics for Computer Programmers. Prentice-Hall. $24,95.

R.E.Crandall: Pascal Applications for the Sciences. John Wiley & Sons. $14,95.

G.VanDiver: The Software Guide, IBM Personal Computer & XT. Micro Information. $49,95.

Vtični moduli za IBM PC

Podjetje AST Research Inc. je vodilno v oblikovanju, proizvodnji in distribuciji vtičnih modulov (tiskanih vezij) za mikroračunalnike IBM PC in XT. Opišimo nekatere od teh modulov.

SixPakPlus

Ta plošča ima od 64k - 384k RAM pomnilnika s kontrolo parnosti, po ena serijska in paralelna vrata, uro s koledarjem in igralna vrata.

MegaPlus II

Lastnosti: 64k - 512k RAM pomnilnik s kontrolo parnosti; igralna vrata; dvoje serijskih vrat; paralelna vrata; program SuperPak itd.

ComboPlus

Lastnosti: 64k - 256k RAM pomnilnik s kontrolo parnosti; serijska vrata; paralelna vrata; ura s koledarjem; dodatne programe.

MonoGraphPlus

Lastnosti: grafična združljivost s paketom Lotus 1-2-3; IBM monokromatični vmesnik; grafika visoke ločljivosti; paralelna in serijska vrata; ura s koledarjem; dodatni programi.

Podjetje AST proizvaja tudi komunikacijske dodatke za IBMove kabinetne sisteme, tako da je mogoče povezati osebne računalnike v mrežo s temi sistemi (AST-3780, AST-SNA, AST-5251, AST-BSC, AST-PCnet II, AST-PCOX).

Mega Board: tiskana plošča za IBM PC

Računalnik na eni plošči, ki je enak računalniku IBM PC, je mogoče zgraditi z uporabo tiskane plošče Mega Board proizvajalca Display Telecomunications Corp., 4100 Spring Valley Road, Suite 400, Dallas, Tx 75234. Cena te plošče z navodili je $99,95!!! Kaj vsebuje ta plošča ?

Osem ibmovskih priključnic za vtične module; vmesnik za ibmovsko tipkovnico; gumb za resetiranje; močnostno priključnico; procesorja 8088 in 8087; periferna vezja; konfiguracijske ibmovske preklopnike; zvočna vrata; območje za dodatno ožičevanje; posebno razširitveno priključnico (sistemsko vodilo); pet ROM pomnilnikov (28-nožičnih); 1M zlog RAM pomnilnika. Velikost te plošče je 10,5" x 13,5". Ta plošča je izredno zanimiva za amaterje in tiste, ki si želijo zgraditi cenen ibmovski osebni računalnik. Seveda je podobno kot pri IBM PC potrebno dodati še nekatere vtične module (npr. za uporabo diskov, barvne grafike, uro s koledarjem itd.).

Abecedno avtorsko kazalo člankov letnika Informatica 8 (1984)

Bekić Z.: Analiza multiprocesorskih sistema s lokalnom cache memorijom. Št. 1, str. 65.

Černivec B.: Nadzorni program za multiprogramiranje pri sprotnem vodenju procesov. Št. 2, str. 46.

Djordjević S.J.: Algebra ključeva. Št. 4, str. 75.

Džonova-Jerman B., J.Žerovnik: Uporaba programskih grafov pri usotavljanju vzporednosti v računalniških algoritmih I. Št. 2, str. 20.

Džonova-Jerman B., J.Žerovnik: Uporaba programskih grafov pri usotavljanju vzporednosti v računalniških algoritmih II. Št. 3, str. 53.

Erjavec T.: Večopravilna tehnika v EDX okolju v računalniku IBM Sistem/1. Št. 2, str. 9.

Erjavec T.: Communications Facility. Št. 2, str. 52.

Faleskini R.: Razvoj informacijske industrije Jugoslavije do leta 2000. Št. 1, str. 3.

Faleskini R.: Razvoj računalništva in usmerjeno izobraževanje. Št. 3, str. 3.

Gams M., I.Bratko, V.Batagelj, R.Reinhardt, M.Martinec, M.Špegel, P.Tancig: Programski jezik Pascal I. Št. 1, str. 22.

Gams M., I.Bratko, V.Batagelj, R.Reinhardt, M.Martinec, M.Špegel, P.Tancig: Programski jezik Pascal II. Št. 3, str. 43.

Gerkeš M.: Metodologija snovanja izvršilnega procesorja za 32-bitni računalniški sistem. Št. 1, str. 14.

Jefić M.V.: Računarski sistem Delta 800. Št. 2, str. 3.

Jenko M.,B.Delak: Mikroračunalniški sistem za nadzor in vodenje dnevnih kopov. Št. 1, str. 46.

Jocković M.B.: Estimation of the Average Transfer Time of Randomly Chosen Blocks from a Disc Cylinder. Št. 1, str. 41.

Kastelic B., R.Murn, D.Peček: Testiranje ROM pomnilnikov. Št. 3, str. 49.

Knop J.V., K.Szymanski, N.Trinajstić: Future Developments of Computer Architecture. Št. 1, str. 48.

Kosovšek F., J.Matjaž, A.Trebar: Centralna procesna enota Delta 16/Bit-Slice. Št. 2, str. 16.

Kokol P., V.Žumer: Generiranje LALR tabel. Št. 4, str. 20.

Kukrika M.: Pristup dinamičkom razpoređivanju zadataka u prstenastoj mreži računala. Št. 2, str. 24.

Kukrika M.: Pristup organiziranju baze podataka u razpodijeljenim sistemima. Št. 2, str. 60.

Kukrika M.: Pristup kreiranju raspoređivača zadataka u distribuiranom izvršnom sistemom sa radom u stvarnom vremenu. Št. 3, str. 38.

Marić I.: Algorithms for Fast B/D Conversion of Integers. Št. 3, str. 21.

Marković R.: Interaktivni generator programa - Sirup. Št. 3, str. 27.

Meško I.: Računalniški programi za poslovno planiranje. Št. 2, str. 66.

Mihovilović B., P.Kolbezen: Paralelno izvajanje opravil v večprocesorskem sistemu I. Št. 3, str. 31.

Mihovilović B., P.Kolbezen: Paralelno izvajanje opravil v večprocesorskem sistemu II. Št. 3, str. 35.

Miletić M.M.: PMP-11, 16-bitni mikroračunar kompatibilan sa PDP-11 miniračunarima. Št. 4, str. 17.

Mozetič I.: Principi kvalitetnega modeliranja. Št. 4, str. 79.

Murn R., D.Peček, B.Kastelic: Testiranje CMOS kombinacijskih vezij. Št. 2, str. 42.

Murn R., S.Prešern, D.Peček, B.Kastelic: Odkrivanje napak z Bergerovim in podobnimi kodi I. Št. 4, str. 68.

H.Nežić: Primjer izvedbe izvršnog sistema za mikroračunala u realnom vremenu. Št. 4, str. 3.

Prešern S.: Optična kontrola plošč tiskanega vezja. Št. 4, str. 51.

Stojmenović I., V.Stojković, L.Jerinić, J.Mirčevski: O implementaciji prevodioca Lispkit LISP-jezika na jezik SECD mašine izvršenoj na Fortran jeziku. Št. 1, str. 57.

Tuta M.: Nova družina mikroprocesorjev Z800. Št. 4, str. 43.

Vukadin D.: Priključitev pisalnega stroja na mikroračunalnik. Št. 4, str. 39.

Wechtersbach D.: Statistično multipleksiranje. Št. 4, str. 13.

Železnikar A.P.: Proizvodnja mikroračunalniškega sistema Partner. Št. 1, str. 10.

Železnikar A.P.: Algol 60 za sistem CP/M II. Št. 1, str. 27.

Železnikar A.P.: Algol 60 za sistem CP/M III. Št. 2, str. 31.

Železnikar A.P.: Programi za pisanje programov 1. Št. 3, str. 7.

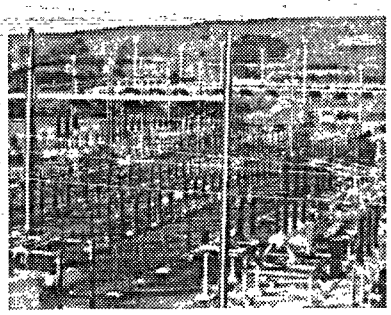
Železnikar A.P.: Programiranje z zbirkami v jeziku CBasic. Št. 3, str. 57.

Železnikar A.P.: Uporaba jezika Ada v številskih izračunih. Št. 4, str. 26.

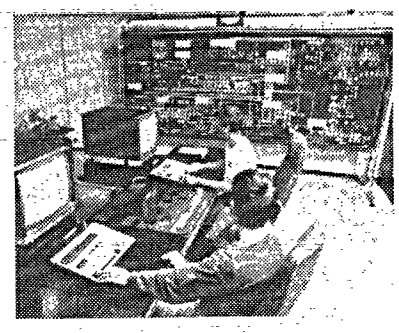
Železnikar A.P.: Ibmovski osebni računalnik Petra I. Št. 4, str. 55.

SISTEMI ZA ENERGETIKO

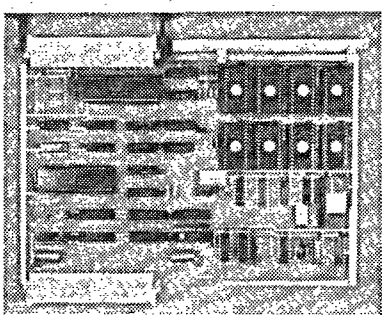
Ljubljana, Tržaška c. 2



DALJINSKO IN LOKALNO PROCESNO VODENJE Z RAČUNALNIKI MINIRAČUNALNIKI IN MIKRORAČUNALNIKI V NAŠIH DOMAČIH SISTEMIH DIPS-11 IN DIPS-85



RAZISKAVE, RAZVOJ, PROIZVODNJA, INSTALACIJA,VZDRŽEVANJE
SPECIALISTIČNO ŠOLANJE KUPČEVIH STROKOVNJAKOV
ELEKTROENERGETIKA, PLINOVODI, NAFTOVODI, VODOVODI, INDUSTRIJA



SODOBNA TEHNOLOGIJA - NAŠ TEMELJ PRI RAZVOJNEM DELU
RAČUNALNIKI - NAŠI SOPOTNIKI NA POTI NAPREDKA
OBIŠČITE NAS IN SE PREPRIČAJTE

Že veliko let se ukvarjamo z raziskavami, razvojem in proizvodnjo sistemov za daljinsko in lokalno procesno vodenje. Temeljno vodilo našega delovanja na tem področju je slediti napredku v svetu in ga presajati na naša domača tla. Vedno smo zavračali nosilno licenčno povezovanje s tujimi firmami povsod tam, kjer smo jasno videli, da vodi v dolgoročno odvisnost in tehnično nazadovanje. Verjeli pa smo v moč lastnega marljivega dela in v ustvarjalnost naših delavcev ter z vstrajnim delom dosegli uspehe, katere nam lahko zavidajo neprimerno večji in bogatejši tekmeci.

Prav zaradi lastne poti in lastnega znanja smo s svojim razvojnim delom ves čas uspeli slediti najnovejšim tehnološkim dosežkom v svetu. V praktično življenje (računalniški nadzor v elektroenergetiki) smo vpeljali najsodobnejše mikroračunalnike.

Tako smo od prvih računalniških korakov pred več kot petnajstimi leti dospeli do sedanjih kompleksnih sistemov za procesno vodenje.