YU ISSN 0350-5596

informatica

Časopis izdaja Slovensko društvo INFORMATIKA, 61000 Ljubljana, Jamova 39, Jugoslavija

Uredništvo in uprava: 61000 Ljubljana, Institut "Jožef Stefan", Jamova 39, telef. (061)363-261, telegram JOSTIN, telex: 31 296 YU JOSTIN.

Letna naročnina za delovne organizacije je 350,00 din, za posameznika 120,00 din, prodaja posamezne številke 60,00 din.
Žiro račun št.: 50101-678-51841
Stališče uredništva se lahko razlikuje od mnenja avtorjev.

Pri financiranju revije sodeluje tudi 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

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

LETNIK 4, 1980 — št. 4

VSEBINA

informatica

Published by INFORMATIKA, Slovene Society for Informatics, 61000 Ljubljana, Jamova 39, Yugoslavia

Headquarters: 61000 Ljubljana, Institut "Jožef Stefan", Jamova 39, Phone: (061)263 261, Cable: JOSTIN Ljubljana, Telex: 31 296 YU JOSTIN.

Annual subscription rate for abroad is US $ 22 for companies, and US $ 7,5 for individuals.

Printed by: Tiskarna KRESIJA, Ljubljana

DESIGN: Rasto Kirn

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350 – 5596

VOLUME 4, 1980 – No. 4

CONTENTS

JEZIK PL/I IN MIKRORAČUNALNIKI I

ANTON P. ŽELEZNIKAR

SOZD ELEKTROTEHNA, DO DELTA

UDK: 681.3:06

Članek opisuje v prvem delu lastnosti programirnega jezika PL/I, in sicer njegov podjezik PL/I-80, ki je namenjen uporabi na mikroračunalnikih. V drugem delu članka bodo prikazani primeri nekaterih programov v jeziku PL/I-80 z ustreznimi ocenami. Najprej je opisana literatura za jezik PL/I (17 navedb) s kratkimi povzetki, tako da bralec lahko izbira med posameznimi učbeniki glede na vrsto uporabe jezika in svoje interese. Prikazane so glavne omejitve jezika PL/I-80 glede na jezik PL/I in uporabnost mikroračunalnikov v povezavi z velikimi računalniki pri izmenjavi pri prevajanju programov, napisanih v jeziku PL/I. Zadnje poglavje je pregled bistvenih konstruktov jezika PL/I-80, ko so opisani podatki, imena, deklaracije, polja, strukture, operacije, izrazi, prireditve, upravljanje pomnilnika, vhod in izhod, stavki za krmiljenje programov ter zgradba PL/I programa. V drugem delu članka bo prikazano povezovanje ločeno prevedenih PL/I programov ter nekateri zanimivi (ocenjevalni) programi.

PL/I Language and Microcomputers I. This article (first part) describes properties of the programming language PL/I, i.e. its sublanguage PL/I-80 dedicated for use with microcomputers. In the second part of the article program examples with some evaluation of PL/I-80 compiler will be presented. First, the literature for PL/I language (17 references) with summaries is listed so the reader can choose among several textbooks according to his interest and application. Main differences between PL/I and PL/I-80 are pointed out and applicability of microcomputers being connected with large computer systems is shown when exchanging and compiling PL/I programs. The last chapter of the article deals with main PL/I-80 constructs as data, identifiers, declarations, arrays, structures, operations, expressions, assignments, memory management, I/O, control statements and structure of PL/I program. In the second part of the article linking of separately compiled PL/I programs and several interesting (compiler evaluation) examples will be shown.

1. Uvod

V poslednjem času narašča uporaba nekdaj neprekosljivega programirnega jezika PL/I (Programming Language / One) tudi na področju mikroračunalniške uporabe. Vzrok za to je med drugim pojavitev prevajalnikov jezika PL/I za mikroprocesorske družine (8080A, 8085, Z 80, 8080, Z8000), ki omogočajo, da je tudi na relativno skromnih mikrosistemih (s pomnilnikom tipa RAM do 48k ter z dvema pogonskima enotama za enostranski gibki disk) moč prevajati dovolj velike računalniške programe, napisane v jeziku PL/I. V tem članku si bomo med drugim ogledali nekatere značilne primere programov v PL/I-80 in njihove prevode.

V preteklosti so bili prevajalniki za jezik PL/I uresničeni predvsem na IBMovih sistemih in prevladovalo je mnenje, da so taki prevajalniki zelo obsežni, da potrebujejo veliko pomnilnega medija (tako centralnega kot zunanjega) ter zelo zmogljive računalniške sisteme (z veliko operacijsko hitrostjo in dovolj zapleteno arhitekturo). Od teh prvih poskusov v šestdesetih letih pa do danes so se močno izpopolnile prevajalniške metode, hkrati pa so bili razviti dovolj zmogljivi mikroprocesorji in ceneni zunanji pomnilniki. Vse to je omogočilo, da je danes moč uspešno uporabiti mikroračunalnike tudi za prevajanje programov, ki so napisani v jeziku PL/I, še posebej tedaj, če se v povezavi centralnih in zunanjih računalniških enot uporabi tehnika plastenja, t.j. tehnika delitve prevajalnega programa na plasti.

Prodor jezika PL/I ne napovedujejo samo proizvajalci mikroračunalnikov v ZDA, ko se hkrati pojavljajo tudi novi paketi prevajalnikov za PL/I na novih mini in makrosistemih, marveč lahko pričakujemo podobne težnje in interes tudi v domači računalniški proizvodnji in uporabi. Tako bi določen podjezik jezika PL/I, npr. PL/I podjezik G, ki je v ZDA standariziran, lahko povzročil neposredno povezavo v računalniških mrežah, kjer bi bili lahko mikro, mini in megasistemi povezani s PL/I-G protokoli v celovit uporabniški sistem.

Opišimo kratko nekatere lastnosti jezika PL/I. Ta jezik je bil predviden tako za uporabo v znanstveno tehniških kot gospodarskih nalogah. Med temi nalogami je

razlika v tem, da potrebujejo znanstvenotehnični problemi predvsem veliko število operacij, gospodarski problemi pa velike podatkovne množice. Včasih je težaven tudi prenos podatkov v in iz računalnika. Jezik PL/I ima bogato izbiro možnosti za krmiljenje komunikacij in je zgrajen tako, da lahko vsak uporabnik brez posebnega napora izbere tiste sestavine jezika, ki jih potrebuje, dočim mu ostalih sestavin jezika ni potrebno poznati.

Kot že napisano, je bil jezik PL/I razvit v šestdesetih letih v podjetju IBM. Jezik PL/I kaže sorodnost z jezikom FORTRAN IV (formati, posredovanje parametrov proceduram), s COBOL 61 (PICTURE podatki, podatkovne strukture) in z ALGOL 60 (bločna zgradba). Z leti je bil ta jezik večkrat spremenjen oziroma dopolnjen, nastali pa so tudi njegovi standarizirani podjeziki (Subset G, PL/I-80 itn.).

2. Pregled literature za jezik PL/I

Zaradi povečanega interesa učenja jezika PL/I, njegove uporabe in uporabe nekaterih metod (npr. strukturiranega programiranja) je tudi v zadnjem času izšlo več obsežnih del, ki obravnavajo problematiko programiranja v jeziku PL/I. V pripravi je vrsta del, ki naj bi sistematično pokrila PL/I podjezik G. V našem seznamu bodo tudi knjige, ki pokrivajo jezika PL/C in SP/k, ki sta vobče vključena v okvir podjezika G ter v t.i. polno IBMovo realizacijo jezika PL/I. Polna realizacija jezika PL/I vsebuje nekatere jezikovne pripomočke, ki so izključeni iz podjezika G.

Oglejmo si naslove nekaterih učbenikov, ki jim sledi kratek povzetek; ta povzetek lahko rabi kot vodilo pri izbiri učbenika za posamezne uporabe. V naslovu je vselej navedena tudi založba, tako da knjigo lahko naročimo tudi v knjigarni.

Imamo tak seznam knjig, ki obravnavajo problematiko programirnega jezika PL/I:

(1) M. Augenstein, A. Tenenbaum: Data Structures and PL/I Programming, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979 (643 strani, trdovezana, visok stavek).

Ta knjiga vsebuje sodoben prikaz polnega jezika PL/I ter je visokošolski učbenik. Jezik PL/I pojasnjuje in prinaša z uporabo napredujočih primerov, ki pokrivajo rekurzijo, obdelavo list, dreves in grafov, sortiranje, iskanje, kodiranje s sekljanjem in upravljanje pomnilnika. Delo ima izčrpen primerjalni seznam sloystva. Poudarek tega dela je na uresničevanju podatkovnih struktur, ko se uporablja določen podjezik polnega jezika PL/I in ta podjezik se kar dobro ujema s podjezikom G. Strukturirano programiranje v tem delu ni poudarjeno.

(2) F. Bates, M. Douglas: Programming Language/One, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970 (419 strani, mehkovezana, strojepisni stavek).

To delo je enostaven uvod v jezik PL/I. Knjiga prinaša osnovne elemente polnega jezika PL/I s poudarkom na komercialnih obdelavah, ko vključuje strukture, zapise, formatiranje in obdelavo napak. Bolj je poudarjeno pojasnjevanje kot sami primeri. Strukturirano programiranje se ne obravnava.

(3) D. Cassel: PL 1 : A Structured Approach, Reston Publishing, Inc., Reston, Virginia, 1978 (219 strani, mehkovezana, visok stavek).

Delo je uvod v PL/I na srednji ravnini. Prikazan je del polnega jezika PL/I s poudarkom na paketni obdelavi in za poslovno uporabo. Jezikovni elementi so opisani jasno, vendar ni poudarjeno oblikovanje programov ali ustrezno strukturiranje, kot napoveduje naslov knjige.

(4) F. J. Clark: Introduction to PL/I Programming, Allyn and Bacon, Inc., Boston 1971 (243 strani, mehkovezana, visok stavek).

Knjiga predstavlja priročnik za samoučenje jezika PL/I s pomočjo vaj. Snov obravnava del polnega jezika PL/I s stališča običajnega kartičnega pristopa, ko začne z razpravo o binarnih številih ter nadaljuje z osnovnimi stavčnimi tipi do enostavnega V/I podatkovnega toka in zapisa. Strukturirano programiranje ni poudarjeno, dani pa so primeri komercialne obdelave.

(5) R. Conway: A Primer on Disciplined Programming, Winthrop Publ., Cambridge, Mass., 1978 (419 strani, mehkovezana, računalniški stavek).

Knjiga je učbenik za PL/C na Cornell University. To je eden od treh Conwayevih učbenikov, ki pokriva uvod v programiranje, s poudarkom na postopkih oblikovanja, razvoja in preizkušanja programov. Učbenik vsebuje tudi kratko razpravo o iskanju in urejevanju podatkovnih seznamov, o obračunavanju, operacijah nad nizi in o interaktivnih sistemih. Poudarjena je praksa strukturiranega programiranja in programirni pripomočki, manj pa izčrpni primeri delovnih programov.

(6) R. Conway, D. Gries: Primer on Structured Programming, Winthrop Publ., Cambridge, Mass., 1976 (397 strani, mehkovezana, računalniški stavek).

To je knjiga o strukturiranem programiranju s poudarkom na jeziku PL/C. Vsebina je podobna prejšnjemu Conwayevemu učbeniku (5), vendar z večjim poudarkom na uporabi programirnega sistema PL/C na univerzi Cornell.

(7) R. Conway, D. Gries, D. Wortman: Introduction to Structured Programming, Winthrop Publ., Cambridge, Mass., 1977 (420 strani, mehkovezana, računalniški stavek).

To je knjiga o strukturiranem programiranju za uporabo sistemov PL/C (Cornell) in SP/k (Toronto). Podobna je prvi Conwayevi knjigi (5) z dodatkom poglavij o obdelavi zbirk in o jezikovnem prevajanju s prevajalniki in interpreti.

(8) G. Groner: PL/I Programming in Technological Applications, John Wiley and Sons, New York, 1971 (230 strani, mehkovezana, visok stavek).

Knjiga je uvod v programiranje tehniških aplikacij v jeziku PL/I in obravnava polni jezik PL/I s primeri, ki temeljijo na paketni obdelavi in IBMovih sistemih. Oblikovanje programov je ponazorjeno z diagrami poteka in z vrsto popolnih primerov uporabe v znanosti. Prikazanih je nekaj primerov generiranja načrtov in grafov. Poudarjeno je pojasnjevanje izračunov s plavajočo binarno obliko s primeri. Programi niso posebno dobro strukturirani.

(9) J. K. Hughes: PL/I Structured Programming, Second Edition, John Wiley and Sons, New York, 1979 (825 strani, trdovezana, visok stavek).

Knjiga je izčrpen priročnik za programiranje v jeziku PL/I ter je popolna predstavitev polnega jezika PL/I. Snov vsebuje strukturirano programiranje, obdelavo enostavnih podatkovnih struktur, obdelavo zapisov, zbirk in seznamov. Poudarek je na programiranju komercialnih nalog z uporabo jezika PL/I na IBMovih sistemih.

(10) J.N.P.Hume, R.C.Holt: Structured Programming Using PL/I and SP/k, Reston Publ, Reston, Virginia, 1975 (340 strani, mehkovezana, računalniški stavek).

Knjiga je uvod v strukturirano programiranje z jezikom PL/I z uporabo modulov od SP/1 do SP/8. Vsak nadaljni modul vsebuje boljpolno podmnožico jezika PL/I. Snov se začne z osnovnimi programirnimi koncepti ter se nadaljuje z različnimi konstrukti jezika PL/I. Programi vsebujejo obdelavo nizov, polj, seznamov in zbirk. Prikazana sta tudi zbirni jezik in prevajanje. Poudarek je na strukturiranem programiranju.

(11) M. Kenedy, M.B. Solomon: Structured PL/Zero Plus PL/One, Prentice-Hall, Englewood Cliffs, New Jersey, 1977 (695 strani, mehkovezana, računalniški stavek).

Knjiga je dokaj izčrpen uvod v jezik PL/I. Knjiga vsebuje osnovne sestavine jezika PL/I dovolj podrobno z uporabo primerov z jezikom PL/C. Kratko je obravnavan IBMov jezik PL/I Level F. Primeri niso posebno razburljivi, vendar je večina jezikovnih lastnosti dobro prikazana.

(12) R.E. Lynch, J.R. Rice: Computers, Their Impact and Use; Holt, Rhinehart and Winston, New York, 1978 (440 strani, mehkovezana, visok stavek).

Knjiga je uvajalni učbenik za računalnike in PL/1, ki se uporablja v srednjih šolah in za učence neteĥničnih usmeritev. Polovica knjige je pregled računalnikov, njihove zgodovine, vpliva na družbo in kako se jih uporablja. Nadalje vsebuje knjiga operacijske sisteme, jezike in jezikovne tipe. Preostali del obravnava IBMov jezik PL/I na vrsti primerov do enostavne obdelave zbirk. Strukturirano programiranje ni poudarjeno.

(13) H. Ruston: Programming with PL/I, McGraw Hill, New York, 1978 (541 strani, mehkovezana, visok stavek).

Knjiga je izčrpen uvod v PL/I in predstavlja PL/I v obliki paketne obdelave z uporabo polnega jezika PL/I v posameznih primerih. Poudarjeno je oblikovanje programov z diagrami poteka. Prikazane so sestavine jezika PL/I vključno z enostavnimi stavki, krmilnimi strukturami, polji, nizi, procedurami in z obdelavo zbirk. Primeri so usmerjeni v znanost in obravnavajo se osnove obdelave napak. Strukturirano programiranje ni poudarjeno.

(14) J.J. Xenakis: Structured PL/I Programming, Duxbury Press, North Scituate, Mass., 1979 (413 strani, mehkovezana, visok stavek).

Knjiga je izčrpen uvod v jezik PL/I, ki je blizu podjeziku G. Prikazani so osnovni programirni koncepti s kratko zgodovino programirnih jezikov. Prikazane so sestavine polnega jezika PL/I vključno s konverzijo med podatkovnimi tipi, polja, nizi in procedure. Posebno poglavje je namenjeno enostavnemu programiranju, temu sledi poglavje o igrah, ki vsebuje tudi program za igranje igre tic-tac-toe. Knjiga ima dober pregled in je enostavna za čitanje.

(15) F. Grund, W. Issel: PL/I - Programmierung, VEB Deutscher Verlag der Wissenschaften, 1975, Berlin (511 strani, trdovezana, visok stavek).

Knjiga vsebuje opis polnega jezika PL/I z vrsto primerov in vaj, tako da rabi tudi za preverjanje razumevanja snovi. Posamezna poglavja opisujejo elemente jezika, aritmetične izraze in prireditvene stavke, zgradbo in izvajanje programov, bločno zgradbo, polja in strukture, stavke poteka (DO, TO, BY, WHILE), izraze in prireditvene stavke za nize znakov in bitov, prirejanje pomnilnika, prekinitve programov, zaporedni V/I, zapisni (stavčni) V/I, posredovanje parametrov proceduram, slikovne (mešane) podatke, obdelavo, spreminjanje programa med prevajanjem in dodatke. Strukturirano programiranje ni poudarjeno. Knjiga je dober učbenik jezika PL/I v nemškem jeziku.

(16) PL/I-80 Reference Manual, Digital Research, P.O.Box 579, 801 Lighthouse Ave., Pacific Grove, CA 93950, 1980 (98 strani, mehkovezana, računalniški stavek, licenčna dokumentacija).

Priročnik je natančen opis jezika PL/I-80, ki je realiziran za mikroračunalniške sisteme s procesorji tipa 8080A in Z80 ter ima za osnovo operacijski sistem CP/M istega proizvajalca (DR). Opisani so vsi dopustni konstrukti jezika PL/I-80, ki je podjezik jezika PL/I podjezik G, tako da je mogoče programe, ki so napisani v PL/I-80 prevajati tudi na sistemih za PL/I-G (npr. VAX11). Priročnik opisuje osnovne, programske in podatkovne strukture in agregate (polja, strukture, polja struktur), upravljanje pomnilnika, prireditve in izraze, zaporednokrmilne stavke, obdelavo V/I, tokovno usmerjeni V/I, zapisno usmerjeni V/I in vgrajene funkcije. Definicije so ponazorjene s primeri. Bistvene omejitve jezika PL/I-80 bodo obravnavane kasneje.

(17) PL/I-80 Applications Guide, Digital Research, naslov kot pod (16), 1980 (134 strani, mehkovezana, računalniški stavek, licenčna dokumentacija).

Poudarek tega priročnika je na primerih. Najprej je opisano delovanje sistema PL/I-80 in programirni stil tega jezika, nato pa še V/I stavki (OPEN, PUT, GET, FORMAT, WRITE, READ), programirni primeri (polinom, kopiranje in obdelava zbirke, informacijski upravljalski sistem), označitvene konstante, spremenljivke in parametri, izjemne obdelave (ON, REVERT, SIGNAL, ERROR itn.), obdelava znakovnih nizov in seznamov, uporaba rekurzije v PL/I-80 (faktorial, Ackermannova funkcija, izračun aritmetičnih izrazov) in ločeno prevajanje modulov ter njihovo povezovanje. Priročnik je dober uvod v programiranje v jeziku PL/I-80, hkrati pa pokaže tudi vso raznovrstnost prevajalniških zmogljivosti, s pripomočki za sledenje in preizkušanje programov, napisanih v tem jeziku.

3. Omejitve in uporabnost jezika PL/I

Visok programirni jezik PL/I-80 za mikroračunalnike je zgrajen na podlagi standarda "ANSI General Purpose Subset (Subset G) of PL/I", ki ga je določil Standarizacijski odbor X3J1 za ANS PL/I. Razlika med jezikom PL/I-80 Vers 1.3 in podjezikom G je tale:

Tile pridevki niso vključeni v PL/I-80:

- DEFINED
- FLOAT DECIMAL
- LIKE
- FILE (dovoljeno le v OPEN stavku)
- zvezdični obsegi in dinamična polja

Tele vgrajene funkcije manjkajo:

- ATANH

- DATE
- STRING
- TIME
- VALID

K jeziku PL/I-80 pa so dodani novi konstrukti:

- % REPLACE stavek
- V/I pripomočki za obdelavo ASCII zbirk:
  - READ in WRITE oblikujete ASCII zapise s spremenljivo dolžino
  - GET EDIT je razširjeno na vhod polnega zapisa v A formatu
  - krmilni znaki so dovoljeni v niznih konstantah

Dodane so vgrajene funkcije.

- ASCII
- RANK in
- FTC (Float-to-Character)

Ameriško podjetje Digital Research nudi PL/I-80 sistem, ki je popoln programski paket za uporabniško programiranje skupaj z operacijskim sistemom CP/M ali MP/M (okrajšavi pomenita Control Program/Microcomputer ali Multiuser cP/M). Na tržišču se je pojavila že verzija 1.3 prevajalnika, ki ima manj napak kot prejšnje verzije in nekatere nove lastnosti. Dodan je bil PICTURE format v PUT EDIT stavkih, ki olajšuje izhodno formatiranje pri komercialnih uporabah.

V letu 1980 je DEC (Digital Equipment Corporation) najavil prevajalnik PL/I Subset G za svoj računalniški sistem VAX 11/780. Razen tega bo mogoče programe prenašati skozi 8-, 16- in 32- bitne mikrosisteme, ki uporabljajo operacijska sistema CP/M in MP/M, v okviru mreže mikro in minisistemov z uporabo mrežnega sistema CP/NET. Tako bo VAX 11/780 vključen kot mrežni vozel v režimu CP/NET, kar bo omogočalo, da se uporabniki preko mikrosistemov vključujejo v močnejše (zmogljivejše) računalniško okolje. Z uporabo jezika PL/I-80 postanejo namreč programi neodvisni od računalniškega in operacijskega sistema.

Kasneje bomo pokazali, kako je s pomočjo prevajalnika jezika PL/I mogoče generirati programske pakete v objektnem (računalniškem) kodu. Ta način izdelave programov je zlasti primeren za proizvajalce programske opreme, ki prodajajo svoje izdelke v objektnem kodu za sorazmerno nizko licenčnino širokemu krogu uporabnikov.

4. Kratek pregled bistvenih konstruktov jezika PL/I-80

4.1. Podatki, imena, deklaracije

Podatek je konstanta ali spremenljivka. Spremenljivkam so prirejeni pridevki z DECLARE (deklaracijskimi) stavki. Podatkovne spremenljivke so enostavne, strukturirane ali poljske. Enostavni podatek je spremenljivka ali konstanta, imenujemo ga skalar. Ločimo 6 vrst podatkov: aritmetični, nizni, kazalčni, označitveni, vstopni in zbirčni.

Aritmetični podatki so numerični s pridevki FIXED BINARY, FLOAT BINARY in FIXED DECIMAL (manjka FLOAT DECIMAL, kar pa je brez bistvenih posledic). Numerični podatek ima določeno natančnost p (število dvojiških ali desetiških mest) in obseg q (število mest desno od dvojiške ali desetiške vejice). Vrednosti p in q sta lahko določeni implicitno, t.j. brez navedbe.

Imamo npr. X FIXED BINARY [(p)], $1 <= p <= 15$, $-32768 <= X <= 32767$. Če p ne navedemo, velja p = 15. FIXED BINARY konstanta se piše kot desetiško celo število. Npr.

DECLARE A FIXED BINARY; A = 1.87;

ko ima A 15 dvojiških mest.

Imamo npr. Y FIXED DECIMAL [(p [, q])], $1 <= p <= 15$, $0 <= q <= p$, $-10**(p-q) < Y < 10**(p-q)$.

FLOAT BINARY ima dva dela: mantiso in eksponentni del. Npr. za Z FLOAT BINARY (p) imamo $1 <= p <= 24$, $5.88*10**-39 <= Z <= 3.40*10**38$. Za FLOAT BINARY konstanto je npr. B = 8.5E3;

Nizni podatki so znakovni in bitni. Imejmo primera:

DECLARE A CHARACTER (12)
DECLARE B CHARACTER (20) VARYING;

DECLARE je rezervirano ime za deklaracijo (določilo podatka). Tu je A sestavljeno iz 12 znakov, če je pa krajše, se ostanek zapolni s presledki, B pa ima enega do dvajset znakov. Nizna konstanta je niz znakov med enojnima narekovajema, npr. 'Ime konstante'. V nizu znakov so lahko tudi t.i. krmilni ASCII znaki in npr. '^I' predstavlja vodoravni tabulirni znak. Notranje se vse male črke prevedejo v velike in '^m' je enako '^M'.

Za primer
DECLARE C BIT (n);
imamo $1 <= n <= 16$, če je C krajše, kot je določeno z n, se preostali biti napolnijo z ničlami. Pridevek VARYING za BIT ni dopusten. Za bitno nizno konstanto velja
'bitni niz' B n, $1 <= n <= 4$,

kjer so znaki bitnega niza lahko heksadecimalne številke. n je tu dolžina posameznega znaka v binarni obliki, ko imamo z n = 1, 2, 3, 4 baze 2, 4, 8, 16. Tako je npr.

'1101'B1 ekvivalentno '1101'B
'1101'B2 ekvivalentno '01010001'B
'320'B2 ekvivalentno '111000'B
'725'B3 ekvivalentno '111010101'B
'F8'B4 ekvivalentno '11111000'B

Dva niza lahko staknemo v en niz z uporabo znaka ||.

Kazalčni podatki se uporabljajo za naslavljanje lokacij v pomnilniku. Vrednost je naslov programske spremenljivke, npr. DECLARE X POINTER;

Označitveni podatki so konstante in spremenljivke, ko imamo npr.
DECLARE Y LABEL;

Vstopni podatki so vstopne konstante ali spremenljivke. Označitev PROCEDURE stavka imenujemo vstopna konstanta. Vstopne konstante so zunanje (vstopna točka zunanje procedure) ali notranje (vstopna točka vgnezdene procedure). Imamo npr.
DECLARE Z ENTRY VARIABLE;

Označitveni in vstopni podatki so krmilni, ker vplivajo na potek (vejitve) programa.

Zbirčni podatki predstavljajo informacijo v kaki zunanji napravi (disku) ter so konstantni in spremenljivi. Imamo konstanto
DECLARE ime_zbirke FILE;
kjer je ime_zbirke ime (identifikator). Za spremenljivko velja
DECLARE ime_zbirke FILE VARIABLE;

4.2. Polja in strukture

Polje je urejena podatkovna množica, katere elementi imajo enake pridevke. Polje se navaja z imenom, element polja pa s celoštevilskim indeksom. Polje določimo npr. z deklaracijo

DECLARE A (2,3) CHARACTER (2);

ko imamo 2 vrstici in 3 stolpce znakovnih nizov s po dvema znakoma, npr.

$$A = \begin{bmatrix} \text{'a1'}, \text{'bf'}, \text{'EE'} \\ \text{'cC'}, \text{'k2'}, \text{'AZ'} \end{bmatrix}$$

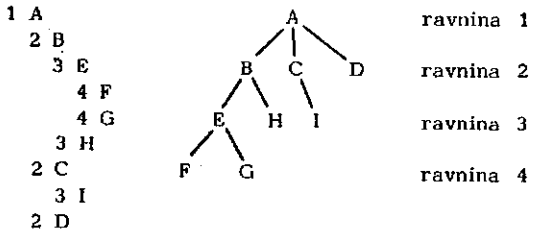
V deklaraciji

DECLARE B (-1:4, -3:3, 2:6) FIXED DECIMAL;

je določeno polje B kot trirazsežnostno z območji indeksov -1 do 4, -3 do 3 in 2 do 6, tako da imamo razpone 6, 7 in 5.

Elementi polja se notranje shranjujejo po vrsticah. Element navedemo npr. z B (0, -2, 5) za zadnji primer. Indeksi so lahko tudi izrazi tipa FIXED BINARY.

Strukture so hierarhično (drevesno) urejene podatkovne množice. Elementi strukture so lahko različnih tipov, so lahko polja in druge strukture (podstrukture). Na sliki 1 imamo strukturo in njej pripadajoče drevo. Tu je A ime strukture,



Slika 1. Podatkovna struktura in njej pripadajoče drevo

elemente te strukture pa navajamo z A.B.E.F. ali kar z F (ker v tem primeru ni dvoumnosti z drugimi elementi te ali druge strukture), A.B.H ali H itd. Strukturo na sliki 1 določimo z DECLARE stavkom, ko imamo npr.

```
DECLARE 1 A,
          2 B,
            3 E,
              4 F CHAR (12),
              4 G CHAR (16),
            3 H FIXED DEC (7),
          2 C,
            3 I CHAR (20),
          2 D CHAR (22) VARYING;
```

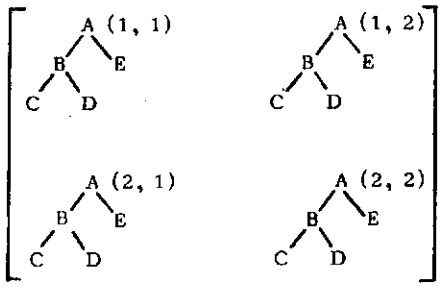
Iz tega določila je razvidno, da so bistveni končni elementi drevesa na sliki 1, ki imajo pridevke.

Strukturna imena so lahko brez pridevkov, imajo pa lahko dimenzijske pridevke ter EXTERNAL, STATIC in INITIAL pridevke. Ravninske številke (glej sliko 1) pred imeni so ločene z vsaj enim presledkom, definicije elementov so ločene z vejico. Ime strukture ima vselej ravninsko številko 1.

Konstruirati je mogoče tudi polja struktur tako, kot smo imeli strukture polj. Tako imamo za določilo

DECLARE 1 A (2, 2), 2 B, 3 C, 3 D, 2 E; (1)

polje na sliki 2. Elemente elementov tega



Slika 2. Polje, ki pripada deklaraciji (1)

polja navajamo z A (2, 1).B.C, A (2, 2).B.D itn. Pri določilu

```
DECLARE 1 G(100), 2 D CHAR(20), 2 E(10)
        CHAR(2);
```

smemo navajati element v oblikah

G(47).E(9) ali G(47,9).E ali G.E(47,9)

4.3. Operacije, aritmetični izrazi, prireditve

Prednostna urejenost aritmetičnih operacij je tale:

** , * in / , + in -

Izraz se obdela od leve proti desni in

2 + 3 * 1 pomeni 2 + (3 * 1)

Operacije so odvisne od operandnih tipov oziroma pridevkov (npr. pri seštevanju FIXED BIN, FLOAT BIN, FIXED DEC).

Tudi za znakovne nize imamo operacije, kot so stik (||) in logične operacije: ^(negacija), & (in), | (ali).

Za števila in znake imamo primerjalne operacije, kot so

> , ~> , >= , = , ~= , <= , < , ~<

s pomenom večje kot, ne večje kot, večje ali enako kot, enako, neenako, manjše ali enako kot, manjše kot, ne manjše kot.

Operandi aritmetičnih izrazov so konstante in spremenljivke; povezani so z operatorji. V izrazih se pojavljajo kot operandi tudi imena funkcij, npr. A + SIN (X) * COS (3).

Prireditveni stavki se uporabljajo za nastavitev spremenljivk na vrednosti izrazov in konstant. Splošno imamo

spremenljivka = izraz;

kjer je spremenljivka ime skalarnega elementa, polja, strukture ali psevdospremenljivke.

4.4. Upravljanje pomnilnika

Pomnilnik je mogoče dodeljevati podatkovnim območjem, in sicer statično v prevajalnem času in dinamično v času izvajanja programa. Imamo tri upravljalske pridevke:

STATIC, AUTOMATIC in BASED

Pridevek STATIC se dodeli pred izvršitvijo glavne procedure in vrednosti STATIC spremenljivk se lahko tudi inicializirajo (INITIAL ali INIT pridevek). Inicializacija nastopi po dodelitvi pomnilnika. Vzemimo primere:

```
DECLARE A FIXED BIN STATIC INIT (0);
DECLARE B (8) CHAR (2) INIT ((8)' BZ') STATIC;
DECLARE
  1 C STATIC,
    2 D FIXED (7) INIT (0),
    2 E CHAR (8) INIT ('IKE'),
    2 F CHAR (3) INIT ('DAT'),
    2 G BIT (12) INIT ('F 5 A' B 4),
    2 H (18) BIT (8);
```

Tu se A inicializira na vrednost 0, polje B dobi 8 elementov, ki so inicializirani na nize 'BZ', element C.D dobi vrednost 0, C.E vrednost 'IKE' in še 5 presledkov itn.

AUTOMATIC pridevek povzroči dodelitev pomnilnika po vstopu v PROCEDURE ali BEGIN blok, v katerem se pojavi zadevna spremenljivka. AUTOMATIC pomnilnik se dodeli statično, izjema je le rekurzija, kjer je za AUTOMATIC spremenljivke uporabljen dinamični pomnilniški mehanizem.

Spremenljivki z BASED pridevkom se dodeli pomnilnik z ALLOCATE stavkom. Pri dodelitvi pomnilnika BASED spremenljivki postane vrednost POINTER spremenljivke naslov BASED spremenljivke. ALLOCATE stavek se uporablja za dodeljevanje pomnilnika BASED spremenljivkam. Vzemimo primer:

```
DECLARE
  (P, Q) POINTER,
  X CHAR (2) BASED,
  Y FIXED BIN BASED (P);
  . . .
  ALLOCATE X SET (Q);
  ALLOCATE Y SET (P);
  . . .
  Q ->X = 'CC';
  Y = Y + 1;
```

Kazalčna spremenljivka Q kaže na naslov spremenljivke X itd.

BASED spremenljivka ostane dodeljena, dokler je ne sprostimo s FREE stavkom, ki ima npr. obliko

```
FREE P ->A;
FREE B;
```

4.5. Vhod in izhod

Vhod in izhod sta postopka, ki omogočata prenos podatkov med računalniškim pomnilnikom in zunanjimi napravami. Zunanja naprava je lahko konzola, tiskalnik ali diskovna zbirka. Imamo dve, vnaprej določeni, standardni zbirki za vhod in izhod, ki se imenujeta SYSIN in SYSPRINT, vse ostale zbirke so določene z DECLARE ime_zbirke FILE VARIABLE ;

V primeru spremenljive zbirke se spremenljivki določi zbirčna konstanta s prireditvenim stavkom.

Imamo 3 tipe zbirk: tokovno, zapisno-zaporedno in zapisno-neposredno (načini shranjevanja podatkov, njihovega dostopa in prenosa).

V/I podatkovni tok je zaporedje ASCII znakov, ki so organizirani v vrstice in strani (omejevalnik vrstice in strani). Tokovne zbirke imajo lahko formatirani in prosti V/I. Zapisno-zaporedne zbirke so dostopne le zaporedno, ko se zapisi čitajo ali pišejo linearno. Zapisno-neposredne zbirke so dostopne z uporabo ključa v READ ali WRITE stavku, torej ni potrebe za njihovo zaporednost. Vsak zapis v zapisno-neposredni zbirki ima svoj ključ, ki omogoča enolično razpoznavanje zapisa pri dostopu.

Pred V/I prenosom moramo zbirko odpreti z uporabo OPEN stavka ali implicitno pri dostopu z GET, PUT, READ ali WRITE stavkom. Eksplicitno odprtje zbirke dosežemo s stavkom

OPEN FILE (ime_zbirke) [zbirčni pridevki];

kjer so pridevki

DIRECT, INPUT, KEYED , LINESIZE, OUTPUT, PAGESIZE, PRINT, RECORD, SEQUENTIAL, STREAM, TITLE, UPDATE

Vrste pogojev se lahko sproži med V/I obdelavo. Npr. ENDFILE pogoj se pojavi pri čitanju vhoda za koncem zbirke (za znakom control-z) v STREAM zbirkah ali za fizičnim koncem RECORD zbirke itn. Podobni pogoji so ENDPAGE, KEY in UNDEFINEDFILE.

CLOSE stavek loči zbirko od zunanje podatkovne množice, ko imamo

CLOSE FILE (ime_zbirke);

S tem stavkom se izhodna zbirka stalno zapiše na disk. Seveda pa lahko tako zbirko ponovno odpremo z OPEN stavkom.

STREAM zbirke so zaporedja ASCII znakov, ki so ločeni z vrstičnimi in straničnimi znaki. Imamo 3 oblike STREAM V/I, in sicer seznamskega, urejevalnega in vrstičnega. Seznamski V/I prenaša podatke brez določitve formata. Urejevalni V/I dovoljuje formatirani dostop do znakov in vrstični V/I do spremenljivih dolžin podatkov v neurejeni obliki. Vrstični V/I je predviden za obdelavo spremenljivih zapisov ASCII znakov z READ in WRITE stavki.

Podatki vhodnega toka morajo biti aritmetične konstante, znakovni ali bitni nizi. Vgnezdeni so lahko tabulirni znaki (ctl-I), ki se obravnavajo kot presledki. Podatke čitamo z GET seznamskim stavkom, ko imamo

GET [FILE (ime_zbirke)] [SKIP [(n)]]
[LIST (vhodni_seznam)] ;

Izrazi v oglatih oklepajih so opcijski (enkrat ali nobenkrat). Če npr. izpustimo FILE opcijo, velja FILE (SYSIN).

PUT LIST stavek se uporablja za pisanje podatkov pri seznamskem V/I. Imamo

PUT [FILE (ime_zbirke)] [SKIP [(n)]]
[PAGE [(p)]] [LIST (izhodni_seznam)];

Ena od opcij se mora pojaviti. Če manjka FILE opcija, velja implicitno FILE (SYSPRINT). Manjkajoči n je kot n = 1 itn.

Pri urejevalnem V/I se podatki čitajo in pišejo skladno s formatnim seznamom, ko uporabljamo GET EDIT in PUT EDIT stavka. Formatni seznam je zaporedje

[n] f - del ... [, [n] f - del]

kjer je 1 <= n <= 254 in je n ponovitveni faktor za pripadajoči f - del. Tu je f - del lahko tudi

(formatni_seznam)

Podatkovni formati so npr. F (w, [d]), E (w, [d]), A [(w)] in B [b] [(w)] (po vrsti za števila s fiksno vejico, za izhod z znanstveno notacijo, za znake in znakovne nize).

Krmilni formatni deli pa so COLUMN (nc), X (sp), SKIP [(n)], LINE (ln) in PAGE.

FORMAT stavek določa oddaljen formatni del, to je R (formatna_označitev), in sicer z

formatna_označitev: FORMAT (formatni_seznam);

Npr.

Z28: FORMAT (A (5), F (6,2), SKIP (3), A (2));

se navaja kot oddaljeni format v stavku

GET EDIT (A, B, C) (R (Z28));

GET EDIT stavek včita podatke z uporabo formatni_ _seznam. Splošna oblika je

GET [FILE (ime_zbirke)] [SKIP [(n)]]
[EDIT (vhodni_seznam) (formatni_seznam)];

PUT EDIT stavek piše izhodne podatke skladno s formatnim seznamom. Oblika je

PUT [FILE (ime_zbirke)] [SKIP [(n)]]
[PAGE [(p)]]
[EDIT (izhodni_seznam) (formatni_seznam)];

READ in WRITE stavek sta posebnosti jezika PL/I--80 in se ju zato izogibljemo, če želimo imeti navzgornjo združljivost z drugimi PL/I sistemi. READ stavek se uporablja za čitanje nizov s spremenljivo dolžino v STREAM INPUT zbirkah, ko imamo

READ [FILE (ime_zbirke)] INTO (v);

kjer je spremenljivka v tipa CHAR VARYING. Če FILE opcija ni prisotna, imamo FILE (SYSIN).

WRITE stavek se uporablja za pisanje ASCII STREAM podatkov v nizih spremenljive dolžine. Tako imamo

WRITE [FILE (ime_zbirke)] FROM (v);

kjer je spremenljivka v tipa CHAR VARYING.

Zapisne zbirke vsebujejo binarne podatke, ki se prenašajo brez konverzije iz ali v zunanji pomnilnik. Dve obliki RECORD obdelave sta dovoljeni: zaporedna (SEQUENTIAL), kjer so zapisi dostopni v zaporedju, v katerem se pojavljajo in neposredna (DIRECT), kjer so zapisi dosegljivi z uporabo ključev.

READ stavek se uporablja za čitanje RECORD SEQUENTIAL zbirk, ko imamo

READ FILE (ime_zbirke) INTO (X);

Če zbirka ni odprta, jo READ stavek avtomatično odpre s pridevki RECORD SEQUENTIAL INPUT.

Z WRITE stavkom se prenašajo podatki iz pomnilnika v podatkovno množico brez konverzije. Za RECORD SEQUENTIAL zbirke imamo

WRITE FILE (ime_zbirke) FROM (x);

Dolžina izhodnega zapisa je tudi tu enaka x .

READ stavek s pridevkom KEY se uporablja za neposreden dostop do zapisov v zbirki. Imamo

READ FILE (ime_zbirke) INTO (x) KEY (k);

kjer je k izraz tipa FIXED BINARY, ki določa relativni dostop k zapisu. Ključne vrednosti začenjajo z nič pa ko produkt ključa in dolžine fiksnega zapisa doseže vrednost obsega diska.

READ stavek s KEYTO pridevkom omogoča, da izvlečemo ključ iz vhodne zbirke, kot da je zbirka zaporedno dostopna. Vrednosti teh ključev se shranijo v pomnilniku ali v drugi zbirki, tako da lahko imamo kasneje dostop do zapisov vhodne zbirke neposredno. Imamo

READ FILE (ime_zbirke) INTO (x)
KEYTO (k);

kjer je k spremenljivka tipa FIXED BINARY.

WRITE stavek s KEYFROM pridevkom se uporablja za neposreden dostop v izhodno zbirko. Tu je

WRITE FILE (ime_zbirke) FROM (x)
KEYFROM (k);

kjer označuje k izraz tipa FIXED BINARY, katerega vrednost je ključ.

4.6. Stavki za krmiljenje programov

V programu se izvajajo stavki zaporedno, tako kot so napisani. To zaporedno izvajanje lahko prekinemo s krmilnimi stavki, ki omogočajo pogojno in brezpogojno vejitev, pa tudi izvajanje v krmiljeni zanki. Procedurni pozivi tudi prekinjajo zaporedno izvajanje programa, zato si jih bomo v tem podpoglavju posebej ogledali.

T.i. GO TO ali GOTO stavek povzroči brezpogojni prehod (skok) na označeni stavek in ima obliko

GOTO označitvena_konstanta; ali
GOTO označitvena_spremenljivka;

IF stavek omogoča pogojno izvajanje stavka ali stavčne skupine s pomočjo boolovskega preizkusa. Splošna oblika je

IF pogoj THEN skupina_1;
[ELSE skupina_2;]

skupina_1 in skupina_2 sta enostavna stavka ali sestavljena, vsebovana v DO ali BEGIN skupini.
Če sta skupina_1 in skupina_2 enostavna stavka, ne smeta biti DECLARE, END, ENTRY, FORMAT ali PROCEDURE stavek. IF stavki so lahko vgnezdeni.

DO skupina je vobče množica stavkov, ki bo izvršena enkrat. Oglejmo si ponavljajočo DO skupino, ki ima splošno obliko

DO WHILE (pogoj);

DO krmilna_spremenljivka = do_specifikacija;

krmilna_spremenljivka je neindeksirana, pogoj je Boolov izraz, do_specifikacija pa je ena izmed teh:

[začetni_izraz [TO končni_izraz]
[BY inkrementni_izraz]] [WHILE (pogoj)]

[začetni_izraz [BY inkrementni_izraz]
[TO končni_izraz]] [WHILE (pogoj)]

[začetni_izraz [REPEAT (ponavljalni_izraz)]
[WHILE (pogoj)]

Tu začetni_izraz določa začetno vrednost krmilne spremenljivke, končni_izraz predstavlja končno vrednost krmilne spremenljivke, inkrementni_izraz se prišteva h krmilni spremenljivki po vsaki izvršitvi zanke, ponavljalni_izraz nadomesti krmilno spremenljivko po vsaki iteraciji, pogoj pa je izraz, ki ima vrednost bitnega niza in ta je pravilna, če so vsi biti niza enaki 1.

ON, REVERT in SIGNAL stavki omogočajo uporabo nekaterih lastnosti v času izvajanja za prekinitev izvajanja pri pogojih z napakami, ki bi sicer povzročile končanje izvajanja programa. Pri tem se razpoznavajo tile pogoji: ERROR (splošna napaka), FIXEDOVERFLOW, OVERFLOW, UNDERFLOW, ZERODIVIDE (računske napake) in ENDFILE, ENDPAGE, KEY, UNDEFINEDFILE (V/I napake). Tako imamo za ON stavek:

ON pogoj on_enota;

kjer je pogoj eden od zgoraj naštetih, on_enota je lahko stavek ali blok.

```
SIGNAL stavek
SIGNAL pogoj;
```

aktivira ustrezno on_enoto (npr. za pogoj ZERODIVIDE).
REVERT stavek oblike

```
REVERT pogoj;
```

deaktivira trenutno on_enoto in obnovi stanje pred njo.

Procedurni bloki so omejeni z besedama PROCEDURE in END ter se pokličejo s subrutinskim CALL stavkom ali s pozivom funkcije. Podatki, ki se posredujejo proceduram, so dejanski parametri, seznam spremenljivk, ki jih procedura pričakuje in je določen s PROCEDURE stavkom, so formalni parametri. Subrutinsko proceduro aktiviramo s CALL stavkom, ko imamo

```
CALL ime_procedure [(sub_1 ... , sub_n)]
                   [(arg_1, ... arg_m)];
```

Tu predstavljajo sub_1 do sub_n seznam indeksov, če je ime_procedure indeksirana vstopna spremenljivka. Nadalje so arg_1 do arg_n dejanski parametri. Proceduro deklariramo oz. definiramo takole:

```
ime_procedure:
    procedurni_stavek
    procedurno_telo
    END [ime_procedure];
```

procedurni_stavek ima obliko

```
PROCEDURE [(parm_1, ... , parm_n)]
   [OPTIONS (MAIN)]
   [RETURNS seznam_pridevkov]
   [RECURSIVE];
```

Formalni parametri, definirani v telesu, ne smejo imeti pridevkov STATIC, AUTOMATIC, BASED ali EXTERNAL. Nadalje pomeni

```
OPTIONS (MAIN),
```

da je to prva procedura, ki se začne izvajati, ko začnemo izvajanje programa. RETURNS seznam_pridevkov je potreben pri t.i. funkcijski proceduri in z njim je določena oblika vrednosti, ki jih funkcija vrne ob svojem izstopu. RECURSIVE pridevek pove, da ta procedura kliče sama sebe posredno ali neposredno med izvajanjem procedure. Razume se, da pokličemo funkcijsko proceduro kar z njenim imenom, t.j. brez rezervirane besede CALL. Npr.

```
I = F (X, Y),
```

če je F ime procedure.

RETURN stavek vrne krmiljenje v klicajoči blok (npr. na konec procedure, kar pomeni izstop iz procedure) in njegova oblika je

```
RETURN [(return_izraz)]
```

kjer je return_izraz vrednost, ki se vrne klicajoči točki. Če je RETURN stavek v glavni (MAIN) proceduri, prevzame krmiljenje operacijski sistem.

Posebna pozornost naj velja nelokalnemu GOTO stavku, kjer imamo označitveno konstanto, ki se nahaja izven najbolj notranjega bloka, v katerem je GOTO stavek. Ta tip GOTO stavka se naj ne bi uporabljal, ker povzroča slabo strukturirane programe.

STOP stavek konča izvajanje programa, zapre odprte zbirke in vrne krmiljenje operacijskemu sistemu. Njegova oblika je

```
STOP;
```

Proceduram se dejanski parametri posredujejo s t.i. navedbo (naslov vrednosti parametra) ali z vrednostjo. Posredovanje z vrednostjo nastopi pri konstantah, vstopnih imenih, izrazih s spremenljivimi navedbami in operatorji, spremenljivkah v oklepajih, funkcijah in v izrazih, ki se ne ujemajo s specifikacijami formalnih parametrov. Npr. imejmo

```
proc: PROCEDURE (A, B, C);
    DCL A CHAR (12),
        B FIXED,
        C FLOAT,
        . . .
```

in poziv

```
CALL proc (X, (Y), Z);
```

ki pošlje proceduri tri dejanske parametre. Za dejanske parametre naj velja:

```
DCL X CHAR (10),
    Y FIXED,
    Z FIXED;
```

Tedaj se X posreduje z navedbo (ujemanje s parametrom A), Y z vrednostjo (ker se pojavlja kot izraz) in Z z vrednostjo (pretvori se v FLOAT).

Večkrat je potrebna tudi uporaba ENTRY pridevka. ENTRY podatki se uporabljajo za razpoznavanje procedurnih imen in so sestavljeni iz vstopnih konstant in vstopnih spremenljivk. Vstopne konstante so notranje procedure ali ločeno prevedene zunanje procedure. Vstopne spremenljivke dobijo med izvajanjem programa vrednosti vstopnih konstant. Oblika ENTRY določila je tale:

```
DECLARE ime_procedure [(ind_1, ... , ind_n)]
[VARIABLE][ENTRY] [(pri_1, ..., pri_m)]
    [RETURNS (ret_pridevek)];
```

ENTRY ali RETURNS se v deklaraciji morata pojaviti, ind_i pomeni indeks_i in pri_j pomeni pridevek_j.

4.7. Zgradba PL/I programa

Procedura je množica stavkov, ki je omejena s PROCEDURE in END stavkom. Procedura, ki ni vgnezdena v blok, se imenuje zunanja. Procedura, ki je v celoti vsebovana v obdajajočem bloku, je notranja. Izvirni tekst programa v PL/I-80 je lahko sestavljen iz ene zunanje procedure, ki ima vgnezdene notranje procedure in/ali bloke. Vsaka zunanja procedura se lahko prevaja ločeno ter se poveže na druge zunanje procedure v t.i. objektni program (prevod in povezava več zunanjih procedur). Ena izmed zunanjih procedur, ki oblikujejo program, mora biti glavna. Oblika glavne procedure je

označitev:

```
  PROCEDURE OPTIONS (MAIN)
  . . .
  stavki in/ali bloki
  . . .
  END označitev ;
```

4.8. Pojasnilo

V prvem delu članka smo opisali literaturo za jezik PL/I (poglavje 2), našteli omejitve (3) in prikazali v zgoščeni obliki glavne sestavine jezika PL/I-80 (4). Zaradi dolžine teksta bodo primeri nekaterih zanimivih PL/I programov prikazani v drugem delu članka, kjer bomo pogledali tudi, kakšna je kakovost prevedenih programov.

Literatura

Literatura za jezik PL/I in PL/I-80 je opisana v poglavju 2 (s kratkimi povzetki).

PROTOKOLLE FÜR VIRTUELLE TERMINALS

J. KNOP, L. WOSNITZA

UDK:681.327.8

COMPUTING CENTRE UNIVERSITY OF DUESSELDORF,
FEDERAL REPUBLIC GERMANY

In this article an overview of the general situation for definition and implementation of virtual terminal protocols is considered.

PROTOKOLI ZA VIRTUALNE TERMINALE.V članku je opisana splošna problematika definicije in implementacije virtualnih protokolov.

1. Einleitung

Terminals an zentralen DV-Systemen werden entsprechend vorgegebener Transport- und Geräteschnittstellen der jeweiligen zentralen Anlage betrieben. Beim Zusammenschluß solcher zentralen Systeme mit ihren Terminalnetzen zu heterogenen Verbundsystemen ist eine erforderliche Leistung des Verbundes, interaktive Verarbeitung von Terminals an allen Hosts des Verbundes zu ermöglichen. Dies gilt natürlich auch für einzelne Terminals oder Cluster von Terminals, die direkt am Netz angeschlossen sind. Diese Aufgabenstellung führt auf die Problematik des "virtuellen Terminals"(VT), zu der in vorliegendem Aufsatz ein Überblick gegeben wird. Grundlegende Untersuchungen zur Definition und Implementierung von Virtual Terminal Protokollen werden im Rechenzentrum der Universität Düsseldorf mit Förderung durch den Bundesminister für Forschung und Technologie durchgeführt.

2. Die VT-Problematik

Interaktive Terminals bieten die Möglichkeit, mit den Services (Dienstleistungsfunktionen), die eine Rechenanlage zur Verfügung stellt, zu kommunizieren.
Die Einführung eines Datennetzes zwischen Terminal und Rechenanlage eröffnet dem Terminal-Benutzer ein weitaus größeres Angebot an Services, wenn dadurch die Dienstprogramme aller angeschlossenen Rechner vom selben Terminal aus angesprochen werden können.
Völlig neue Aspekte bringt die Benutzung eines öffentlichen Datennetzes:

- Für Terminal-Benutzer entsteht ein nahezu unbegrenztes Service-Angebot bei geringem Kostenaufwand.
- Der Standort eines Terminals wird völlig unabhängig von der Lage der Rechensysteme.

Zusätzlich ergibt sich für Betreiber von Rechenzentren die Möglichkeit einer besseren Nutzung der Rechner.

Allerdings entstehen schwerwiegende Probleme bei solchen offenen Verbund-Systemen, da gefordert werden muß, daß jedes angeschlossene Terminal in der Lage ist, mit jedem angebotenen Service (potentiell) zusammenzuarbeiten.

2.1 Problemstellung

Ursache für diese Probleme ist die Vielzahl unterschiedlichster Terminals, die heute auf dem Markt sind, da immer nur wenige Terminal-Typen auf die speziellen Bedürfnisse eines gegebenen Rechners abgestimmt sind. Im ungünstigsten Fall müßte für jedes Dialog-Programm und jedes Terminal, das mit diesem Programm über das Netz kommunizieren soll, jeweils eine Anpassung der Ein-/Ausgabe-Bedürfnisse des Programms an den Funktionsvorrat des Terminals geschaffen werden.

Das sind bei n Applikationen und m Terminal-Typen n*m verschiedene Abbildungen!

Als einzig sinnvolle Lösung dieses Anpassungsproblems erscheint die Definition eines (oder mehrerer weniger) Standard-Terminals, so daß

a) für die Applikationen eine einheitliche logische Repräsentation eines generellen Terminals verfügbar ist.

b) Abbildungen der Funktionen des Standard-Terminals auf den Funktionsvorrat der angeschlossenen Terminal-Typen geschaffen werden können.

Auf diese Weise benötigt man für jedes Terminal nur noch eine einzige Abbildung, also nur noch m bzw. n+m Abbildungen, wenn man die Anpassung der Applikationen an die logische Repräsentation mitzählt.

Die Aufgabe besteht also in der Auswahl eines geeigneten Standard-Terminals.

Eine Möglichkeit wäre, ein möglichst weit verbreitetes und von seinem Funktionsvorrat her möglichst universelles reales Terminal für diesen Zweck heranzuziehen. Dadurch würde die Notwendigkeit der Anpassung für diesen Terminal-Typ entfallen.

Abgesehen von den wohl unlösbaren marktpolitischen Problemen, die solch eine Lösung mit sich bringen würde, eröffnet die alternative Lösung, nämlich die Definition eines abstrakten (virtuellen) Terminals als netzweites Standard-Terminal, wesentlich interessantere Perspektiven, die z.T. weit über die Lösung des Inkompatibilitätsproblems hinausgehen:

a) Durch geeignete Definition kann die Abbildbarkeit zwischen virtuellem und realem Terminal für eine möglichst große Menge realer Terminal-Typen gewährleistet werden.

b) Die Bildung von Terminal-Klassen durch Klassifizierung der Terminal-Funktionen ermöglicht

- dynamische Auswahl des Funktionsumfangs,
- leichte Erweiterbarkeit um neue Funktionsklassen,
- Optimierung der Informationsübertragung durch Voraussetzung von Intelligenz auf der Terminal-Seite bei höheren Klassen.

c) Die Bedürfnisse von Applikationsprogrammen hinsichtlich ihres Ein-/Ausgabeverhaltens können neu überdacht und bei der Definition berücksichtigt werden.

d) Die Abstraktion realer Terminal-Funktionen zu logischen Kommunikationsfunktionen ermöglicht eine Annäherung der System-I/O an Datenstrukturierungskonzepte höherer Programmiersprachen.

Die Definition eines solchen virtuellen Terminals umfaßt die Spezifikation der Service-Schnittstelle sowie ein Kommunikationsprotokoll zur Realisierung dieser Services. Die Beschreibung der Services erfolgt im allgemeinen durch Auflistung der Funktionen, die durch das Terminal-Modell unterstützt werden, und einer Erläuterung anhand der zugrundeliegenden Datenstruktur.

Das Protokoll legt fest, in welcher Form die Nachrichten des virtuellen Terminals aufgebaut sind, die die angebotenen Funktionen realisieren, und nach welchen Regeln sie ausgetauscht werden.

2.2 Begriffsbildung

Nach dem ISO-Reference-Modell für offene Systeme (ISO 79) sind oberhalb der Session-, Transport- und Netzwerkebenen folgende Kommunikationsebenen zu unterscheiden:

a) "application layer"
Auf dieser Ebene findet die Kommunikation zwischen Terminal-Benutzer und Applikations-Programm auf der Basis eines applikationsspezifischen Dialog-Protokolls unter Ausnutzung der VT-Services statt.

b) "presentation layer"
Auf dieser Ebene werden Datenstruktur und Funktionsumfang des virtuellen Terminals durch Kommunikation gemäß den Regeln des VT-Protokolls realisiert.

Das VT-Protokoll wiederum stützt sich auf die Services der tiefer liegenden Schichten der ab, so daß sich folgendes Bild ergibt:

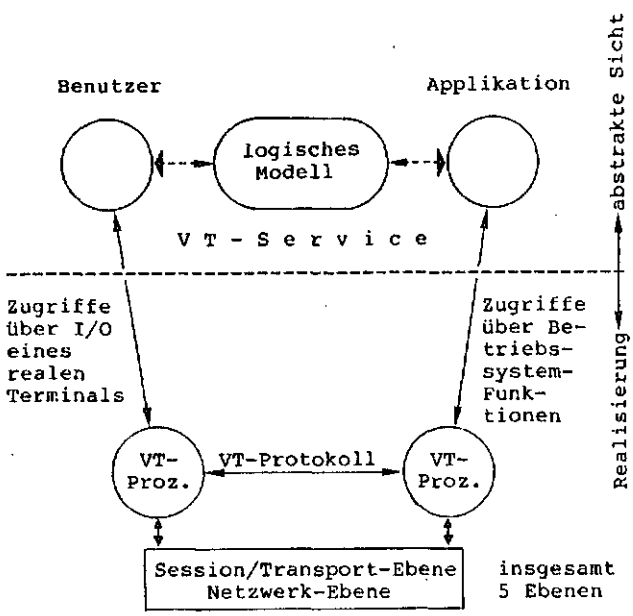


Abb. 1: ISO Modell

Die in diesem Zusammenhang auftretenden Begriffe werden bei ISO wie folgt definiert:

a) Ein Terminal besteht aus einem oder mehreren Geräten, die es einem Benutzer ermöglichen, mit den Applikationen der verschiedenen Hosts eines Rechnernetzes zusammenzuarbeiten.

b) Eine Terminal-Klasse ist eine Menge von Terminals mit ähnlichen funktionellen Eigenschaften.

c) Ein virtuelles Terminal (VT) ist die standardisierte Beschreibung einer Menge von logischen Funktionen, die die Eigenschaften einer Terminal-Klasse repräsentieren.

d) Ein VT-Protokoll (VTP) ist eine Menge von Regeln und Konventionen, die während der Kooperation zwischen den "Entities" (VT-Prozeduren), die die VT-Funktionen realisieren, gelten.

Gerade der in diesem Zusammenhang wichtige Begriff "virtuelles Terminal" wird in der Literatur oft mit mehr oder weniger stark modifizierter Bedeutung benutzt.

Eine Bedeutungsvariante sei hier explizit erwähnt.

Bei (Bauwens/Magnee 78) sowie bei (PIX 77) wird ein VT als ein logisches Gerät betrachtet, mit dem unter Kontrolle des VTP kommuniziert wird und dessen Datenstruktur und Funktionen auf ein reales Terminal abgebildet werden. Ein VT ist hier also gleichsetzbar mit einem realen Terminal, das an den Netz-Standard angepaßt ist.

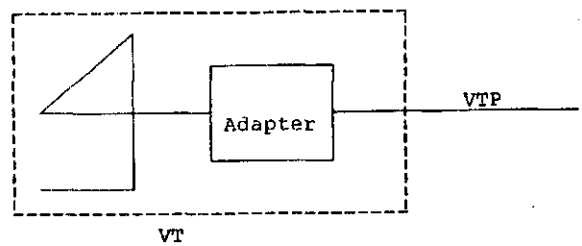


Abb. 2

3. Klassifikation von Terminal-Funktionen

Um die Vielzahl existierender Terminal-Typen, die schon auf Grund der verschiedenen Anwendungsgebiete stark unterschiedliche Funktionsvorräte aufweisen, per VT zu erfassen, ist es erforderlich, Klassen von Terminals mit vergleichbarem Funktionsumfang zu bilden und für jede Klasse ein korrespondierendes VT zu definieren.

Kriterien bei der Klassifizierung

Als Kriterien für solch eine Klassifizierung dienen in der Regel die Datenformate, die den Terminals zu Grunde liegen. Auf diese Weise entstehen Klassen wie

- "stream class" (line mode) für fernschreiberähnliche Terminals incl. Nebengeräte wie Kartenleser oder Zeilendrucker.
- "page class" (page mode) für Bildschirmgeräte mit freier Cursor-Adressierbarkeit.
- "data entry class" für Bildschirmgeräte mit Format-Unterstützung (format mode) und Darstellungs- bzw. Zugriffsattributen.

Einen weiteren Gesichtspunkt für die Klassifizierung liefern spezielle Anwendungsgebiete, wobei insbesondere die

- "graphics class" für graphische Terminals und Plotter

zu identifizieren ist.

Die Bildung weiterer Klassen wird erforderlich werden, wenn man Entwicklungen auf Gebieten wie Bildschirmtext oder Faksimile-Übertragung berücksichtigen will.

Aus diesem Grunde achten fast alle Protokoll-Designer auf eine leichte Erweiterbarkeit hinsichtlich der unterstützten Terminal-Klassen.

Auswahl einer Klasse für den Dialog

Die Einigung der beiden VT-Seiten auf eine bestimmte Klasse und zusätzliche Parameter muß selbstverständlich vor der eigentlichen Daten-Kommunikation erfolgen. Es gibt allerdings verschiedene Möglichkeiten, solch eine Klassenauswahl bzw. einen Klassenwechsel zu vollziehen:

a) Auswahl durch Parameter beim Verbindungsaufbau. Diese Möglichkeit erlaubt keinen Klassenwechsel während einer Verbindung (Schicker/Zimmermann 77, Naffah 77).

b) Auswahl durch Kommandos innerhalb einer besonderen Negotiation-Phase des Protokolls (IFIP 78, ECMA 79 ..).

c) Dynamische Anpassung an den jeweils erforderlichen Funktionsvorrat durch Einrichtung entsprechender Datenstrukturen innerhalb des normalen Protokoll-Ablaufs (PIX 77).

Die Alternativen (b) und (c) sind sicher der Möglichkeit (a) vorzuziehen, da heutige Terminals leicht in ihrem Arbeitsmodus umschaltbar sind und von daher auch physikalisch in der Lage sind, einen Klassenwechsel nachzuvollziehen. Die Unterschiede zwischen (b) und (c) sind eher logischer Natur; Ansatz (c) basiert auf einer abstrakteren Sicht.

4. Datenstrukturierung

Die zur Beschreibung eines Terminals erforderliche Datenstruktur läßt sich gut in einer höheren Programmiersprache formulieren, z.B. in PASCAL-ähnlicher Notation:

```
line = array (1..80) of char;
page = array (1..24, 1..80) of char;
```

Höhere Strukturen ließen sich in ähnlicher Weise beschreiben:

```
field  = record
            attribute: char;
            image: array (1..fieldlength)
            of char
         end;

format = record
            name1: field (fieldlength1);
            name2: field (fieldlength2);
               ...
         end;
```

oder

```
archiv   = file of page;
```

Zusätzlich gehört zur Datenstruktur eines Terminals ein Zeiger zur Referierung der aktuellen Cursor-Position.

Initialisierung

Initialisierungsfunktionen eines realen Terminals, die sich auch in einem VT-Protokoll wiederfinden lassen, sind z.B.

"carriage return" bzw. "line feed"
(im Line-Mode)
"new page" bzw. "erase screen"
(im Page-Mode)
"erase all unprotected fields" u.ä.
(im Format-Mode).

Ihre Wirkung auf die Datenstruktur:

- Cursor an den Beginn der Datenstruktur
- Belegung der Character-Positionen durch einen Initialwert (z.B. NULL, Blank, Punkt oder "undefiniert").

Adressierung

Bei realen Terminals kommen die verschiedensten Adressierungsformen für den Cursor vor:

absolut - durch Koordinaten bezogen auf den Beginn der Datenstruktur.
relativ - durch Koordinaten bezogen auf die aktuelle Position des Cursors.
sequentiell - durch Funktionen wie "new line" oder "next field".

Je nach Protokoll werden die verschiedenen (oder auch alle) Funktionen direkt unterstützt (z.B. im VDP, ECMA 79). Andere Protokolle nutzen die Tatsache aus, daß sich alle Adressierungsformen auf eine (z.B. die absolute) Form zurückführen lassen (wie bei PIX 77).

Darstellung

Data-Entry-Terminals erlauben über Attribute, die z.B. zu Beginn eines Feldes als

Character auf dem Bildschirm eingetragen werden, eine Steuerung der Darstellung der nachfolgenden Daten wie normal, hell, dunkel, invertiert, farbig, ... Desweiteren lassen sich über solche Attribute die Zugriffsrechte des Terminal-Benutzers auf einzelne Felder einschränken (protected/unprotected, numeric/alphanumeric, ...).

In VT-Protokollen werden solche Funktionen durch Operationen wie "set attribute n" unterstützt.

Die Gültigkeit einer solchen Operation variiert von Protokoll zu Protokoll (Gültigkeit des Attributes nur für das nächste Zeichen, bis zur nächsten Änderung oder Festlegung durch eine Längenangabe).

Auch die Wirkung auf die Cursor-Position ist nicht einheitlich. Entweder bleibt die Cursor-Position erhalten, oder der Cursor rückt wie beim Eintrag eines normalen Zeichens um eine Position vor.

5. Nachrichtenstrukturierung

Die Realisierung der durch ein VT-Protokoll unterstützten Funktionen erfolgt durch Austausch von Nachrichten, die zur Gewährleistung einer eindeutigen Verständigung zwischen den beiden Partnern einer exakten Syntax und Semantik unterliegen.

Während die Semantik dieser Nachrichten durchweg verbal beschrieben wird, läßt sich die Syntax sehr gut mit den bekannten Methoden der formalen Sprachen (z.B. erweiterte Backus-Naur-Form) erfassen.
Starke Unterschiede bestehen hinsichtlich der Komplexität der Nachrichten-Syntax.

Die einfachste Form einer Nachricht, im folgenden als Kommando bezeichnet, besteht aus einem Kopf, der die Codierung und die Länge des Kommandos enthält, und einem zugehörigen Datenteil:

<command> ::= <code> <length> <data>

Mit dieser extrem einfachen Syntax lassen sich bereits, wie das VDP (ECMA 79) zeigt, alle erforderlichen VT-Funktionen realisieren.

Beispiele

ADDR 2 X Y	(absolute Adresse mit Koordinaten X und Y).
CONT 4 T E X T	(Übertragung des Wortes "TEXT").

Allerdings läßt sich die Kommunikation weitaus übersichtlicher gestalten, wenn man den Nachrichtenaustausch in Phasen gliedert und die einer Phase entsprechend kennzeichnet:

<command-block> ::= <type> $\{$<command>$\}^*$

Das VT-Protokoll der (IFIP 78) z.B. unterscheidet die Phasen

- "Negotiation"	(zur Verständigung über Protokoll-Parameter)
- "Control"	(zur Resynchronisierung nach Fehlerfällen)
- "Text"	(zur normalen Daten-Kommunikation)

Eine weitergehende Strukturierung der Nachrichten durch die Syntax ermöglicht eine Steigerung der Dynamik in der Beschreibung der zugrundeliegenden Datenstruktur während der laufenden Kommunikation.

In (PIX 77) unterscheidet man - ausgehend vom Konzept der "communication variables" (Raubold 78) - die drei Kommando-Typen:

- "Definition"	(Beschreibung von Datenstrukturen für die weitere Kommunikation, z.B. Zeilen- und Seiten-Dimensionen, Formate und deren Felder o.ä.)
- "Declaration"	(Einrichtung von zuvor definierten Datenstrukturen unter Einführung eines Namens)
- "Data"	(Kommunikation gemäß der zuvor eingerichteten Datenstrukturen)

Mit der Erweiterung der Nachrichtenstruktur durch den Namen einer deklarierten Datenstruktur

<Update> ::= <DS-Name> $\{$<command-block>$\}^*$

lassen sich die elementaren Kommandos weiter qualifizieren.

Die Menge der elementaren Kommandos kann dadurch ohne Verlust an Ausdrucksfähigkeit sehr klein gehalten werden, und man erhält ein recht universelles Konzept zur Modellierung der Kommunikation.

6. Übersicht über vorhandene VT-Definitionen

Frühe Arbeiten auf dem Gebiet der Rechnernetze führten zur Entwicklung sogenannter Scroll-mode VT's (SMVT), die lediglich die Funktionen und Strukturen fernschreiberähnlicher Terminals abdecken. Solche VT's wurden für Netze wie CYCLADES, ARPANET, EPSS und EIN (Schicker/Zimmermann 77) definiert.

Als Ergänzung dazu ist das Data-Entry VT (DEVT) von Naffah (Naffah 77) zu verstehen, das Bildschirm-Terminals mit Format-Unterstützung und zusätzlichen Geräten wie Printer, Cassetten oder Floppy-Disks beschreibt.

In der darauffolgenden Zeit entstanden als direkte Weiterentwicklung mehrere VT-Definitionen, die sich in wesentlichen Punkten sehr stark ähneln. Die Neuerung gegenüber den früheren VT-Definitionen ist lediglich die Unterbringung der verschiedenen Terminal-Klassen in ein einheitliches Konzept. In diese Gruppe gehören das VT für das belgische Universitäts-Netz (Bauwens/Magnee 78), das VT der IFIP - International Network Working Group (IFIP 78), das Terminal Access Protocol aus Kanada (Canada 78) und das Data Entry VT für Euronet (EURONET 78).

Protokolle mit einem sehr starken Bezug zu realen Terminals (in Funktionsvorrat und Nachrichtenformaten) werden von AFNOR (AFNOR 78) und im Virtual Device Protocol (VDP) der ECMA (ECMA 79) vertreten. Das Network Virtual Display Terminal der Firma IBM (IBM 78) ist ebenfalls ein Beispiel dieser Gruppe.

Der genau entgegengesetzte Weg zu immer stärkerer Abstraktion und der damit verbundenen Annäherung an Konzepte höherer Programmiersprachen wurde bei PIX (PIX 77) eingeschlagen.

Übersicht: Klassifizierung in existierenden VT-Protokollen

(Eine Strecke beschreibt jeweils den Funktionsumfang einer Klasse.)

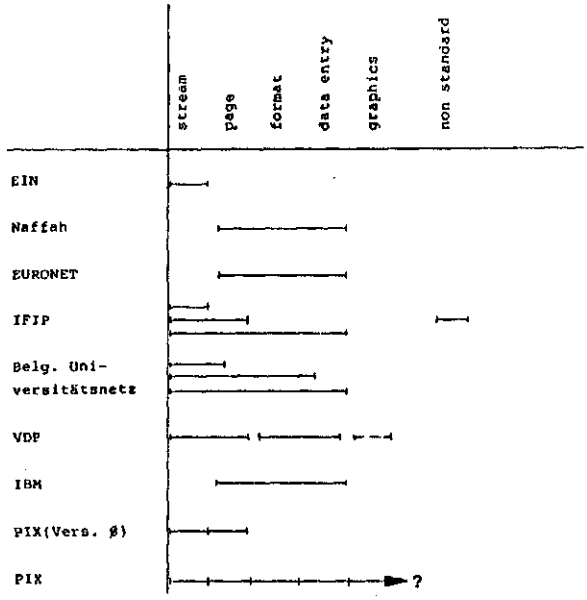


Abb. 3

7. Aufgaben für die weitere Entwicklung

Das eigentliche VT-Problem, nämlich die Lösung des Inkompatibilitätsproblems zwischen Terminals und Rechnern, läßt sich prinzipiell als gelöst bezeichnen, allerdings auf einem Niveau, das sich mit dem der Assembler-Sprachen auf dem Gebiet der Programmiersprachen vergleichen läßt: Für jede erkannte Funktion wird ein Kommando eingeführt. Bestrebungen, die Vergleiche zu höheren Programmiersprachen zuließen, sind im Moment lediglich bei PIX zu erkennen: Beschreibung einer Vielzahl einzelner Funktionen durch Strukturierung einiger weniger Kommandos.

Bei den bisher im VT-Bereich einbezogenen Terminal-Klassen fallen die Unterschiede der beiden Ansätze noch nicht sonderlich ins Gewicht, obwohl bereits bei der Beschreibung von formatierten Bildschirmen erkennbar wird, daß das PIX-Konzept elegantere Lösungen liefern kann.

Die Vorteile eines solchen Strukturierungskonzeptes werden sich allerdings dann zeigen,

wenn man beginnt, graphische Terminals mit in das VT-Konzept einzubeziehen. Die hier erforderliche Beschreibung von graphischen Symbolen, Kurven und Flächen unter Ausnutzung vorhandener Intelligenz des Terminals, die Berücksichtigung zeitlicher oder ereignisorientierter Komponenten als lokale Funktionen wie bewegte Bilder oder auf bestimmte Eingaben hin wechselnde Bilder, all dies sind Aufgaben, die mit den bisher vorhandenen Ansätzen noch nicht zufriedenstellend gelöst werden können.

8. Literatur

ISO 79: Reference Model of Open System Interconnection; ISO/TC97/SC16/N 227

Schicker/Zimmermann 77: Proposal for a Scroll Mode VT EIN/CCG/77 02

Naffah 77: High Level Protocol for Alphanumeric Data-Entry Terminals; TER 538.1 Reseau Cyclades

Bauwens/Magnee 78: Definition of the VTP for the Belgian University Network; Computer Network Protocols, Liège

IFIP 78: Proposal for a standard VTP ISO/TC97/SC16/WG2/N117

CANADA 78: Terminal Access Protocols; ISO/TC97/SC16/N58

EURONET 78: Data Entry VT for Euronet; Commission of the European Communities

AFNOR 78: VT Protocol ISO/TC97/SC16/N52

ECMA 79: Virtual Device Protocol; European Computer Manufacturers Association ECMA/TC23/79/75

IBM 78: Network Virtual Display Terminal - Systems Network Architecture; Information Paper for SC16, Part III; ISO/TC97/SC16/WG2/N26

PIX 77: The PIX Virtual Terminal Protocol; GMD Darmstadt

Raubold 78: A Model for Application Level Networking Protocols; GMD Darmstadt

---

Erstveröffentlichung in "DAS RECHENZENTRUM", Carl Hansa Verlag

AN EXPERIMENT IN AUTOMATIC LEARNING OF DIAGNOSTIC RULES

I. BRATKO, P. MULEC

UDK:681.3:616–071

FACULTY OF ELECTRICAL ENG. AND J. STEFAN INSTITUTE E. KARDELJ UNIVERSITY, LJUBLJANA' YUGOSLAVIA

The paper reports on an experiment in automatic learning of classification rules for medical diagnosis. The input to the learning process is a set of examples, i.e. already diagnosed patients. The output is a diagnostic rule, in the form of a decision tree, for diagnosing unknown examples. As a learning method we employed a slightly modified Quinlan's algorithm ID3. The lymphographic investigation served as a problem-domain for the experiment. We used the data about 150 patients, each of them described by a set of 18 discrete attributes and classified into one of 9 alternative diagnoses. The average precision of automatically derived rules obtained in a series of experiments was about 80% when diagnosing unknown patients, which compares favourably to the estimated precision of human diagnosticians. This is between 60 and 85% depending on experience.

POSKUS Z AVTOMATSKIM UČENJEM DIAGNOSTIČNIH PRAVIL. Članek opisuje poskus z avtomatskim učenjem diagnostičnih pravil za diagnosticiranje v medicini. Vhod v proces učenja je množica primerov, to je pacientov z znanimi diagnozami. Izhod je diagnostično pravilo v obliki odločitvenega drevesa za diagnosticiranje neznanih primerov. Kot metodo učenja smo uporabili nekoliko modificiran Quinlanov algoritem ID3, kot problemsko področje za naš poskus pa je služila limfografska preiskava. Uporabili smo podatke o 150 pacientih, opisanih z 18 diskretnimi atributi in klasificiranih v 9 možnih alternativnih diagnoz. Povprečna natančnost diagnostičnih pravil, avtomatsko generiranih v zaporednih poskusih, je bila okrog 80% pri diagnosticiranju neznanih primerov. Ocenjena natančnost diagnostika - zdravnika leži med 60 in 85%.

Introduction

One problem arising in the development of computer applications such as expert information systems is: How to get the problem-domain knowledge into the system? The usual way is that the human domain-expert himself describes his or her own knowledge in some suitable formal language. It often turns that this is a difficult task since the knowledge used by the expert is often intuitive, not systematic, and/or poorly formalised. Examples of problem-domains in which human experts typically use nonformalised knowledge are: medical diagnosis, economic forecasts, playing chess etc.

Another, attractive way of getting the knowledge into the system is based on the use of automatic learning from examples and counter-examples. The domain-expert's task here becomes simpler as he is no more requested to systematically formalise his entire knowledge, but only to provide the system with an adequate set of examples. This set should, hopefully, be sufficient for the system to autonomously recognise the regularities underlying the examples.

In this paper we report on an experiment in automatic learning of medical diagnosis. The diagnostic domain chosen for the experiment was lymphographic investigation. As examples for learning we used old medical data with known correct diagnoses. The result of the learning process was a diagnostic rule in the form of a decision tree. This decision tree defines a mapping between lymphographic data and the corresponding diagnosis, and can thus be used for automatic diagnosis.

Our learning algorithm was based on the Quinlan's automatic learning program ID3 (e.g. Quinlan 1979, Quinlan 1980), which had to be generalised to classification into any number of classes (ID3 could originally deal with two classes only). The results of the experiment indicated that the precision of the

automatically learned diagnostic rule superseded that of an average physician - practitioner in this field, and that it is only slightly worse than the precision of best specialists for lymphographic investigation

The learning algorithm

The algorithm used in our experiment is a version of Quinlan's ID3 system, which is based on Hunt's CLS (Concept Learning System, Hunt et. al. 1966).

The input to the algorithm are examples together with their class membership. Each example is described by a set of discrete attributes. Each attribute has typically a few values. All examples are specified by the values of all the attributes (i.e. each example is completely specified), and by the class to which the example belongs. Quinlan's original algorithm works with two classes only. As our problem of lymphographic diagnosis required 9 classes, ID3 had to be modified accordingly. The appropriate generalisation from 2 to N classes of ID3's information-theoretic evaluation function was straightforward.

The output of the algorithm is a decision tree. The nodes of this tree correspond to tests of attributes. The arcs stemming from nodes in the tree correspond to the values of the attribute corresponding to the node. Each leaf of the tree is assigned a class in such a way that this class conta ins all the examples which, according to their attribute values, fall into this leaf.

The algorithm for constructing a decision tree from examples is very simple and efficient. First, a subset, called a "window", of the example set is chosen. A decision tree which "explains" this window is constructed. Then this tree is tested against the whole example set. If the tree explains the whole set (i.e. correctly classifies all the examples in the set) then this tree is the final result of the learning process. If not, then the window is modified by the inclusion of some examples which contradict the current decision tree, whereby possibly deleting some of the members of the old window. A new decision tree is constructed for the new window, then tested against the complete example set, etc.

A decision tree for a given window is constructed in a top-down fashion. First, one of the attributes is selected to become the root of the tree. This attribute partitions the window into "subwindows", so that each subwindow contains examples with the same value of this attribute. Then, subtrees are constructed for all the subwindows. The subtrees are connected to corresponding arcs stemming from the root.

Attributes to become roots of the (sub)trees are chosen by a heuristic criterion: that attribute is chosen which most reduces the information content of the (sub)window.

An implementation of this algorithm is in more detail documented in Mulec 1980.

The problem of Lymphographic diagnosis

In the lymphographic investigation, 18 symptoms are considered. Symptoms correspond to attributes, as referred to in the previous section. There are 9 possible alternative diagnoses; that is: each example is classified into one of 9 classes. Table 1 shows a form which is to be filled in by a physician when diagnosing a lymphograph. The data in this form defines one example for our learning algorithm.

Experiment and results

In the experiment, we used the archive data about 150 patients who were lymphographically investigated at the Institute of Oncology, Ljubljana, over a 3 year period. Fig. 1 shows the diagnostic rule produced by the learning algorithm if all 150 samples were used as training examples.

By the definition of the Quinlan's algorithm, the diagnostic rule has to correctly diagnose all the examples used for training. It is interesting, however, how successfully this diagnostic rule classifies unknown samples. To investigate this question empirically, we randomly permuted all 150 examples, then used the first 100 examples as a training set for the derivation of a diagnostic rule, and then tested the rule on the remaining 50 samples as unknown cases. To eliminate the risk of pathological permutations, this experiment was repeated 10 times, each time with another

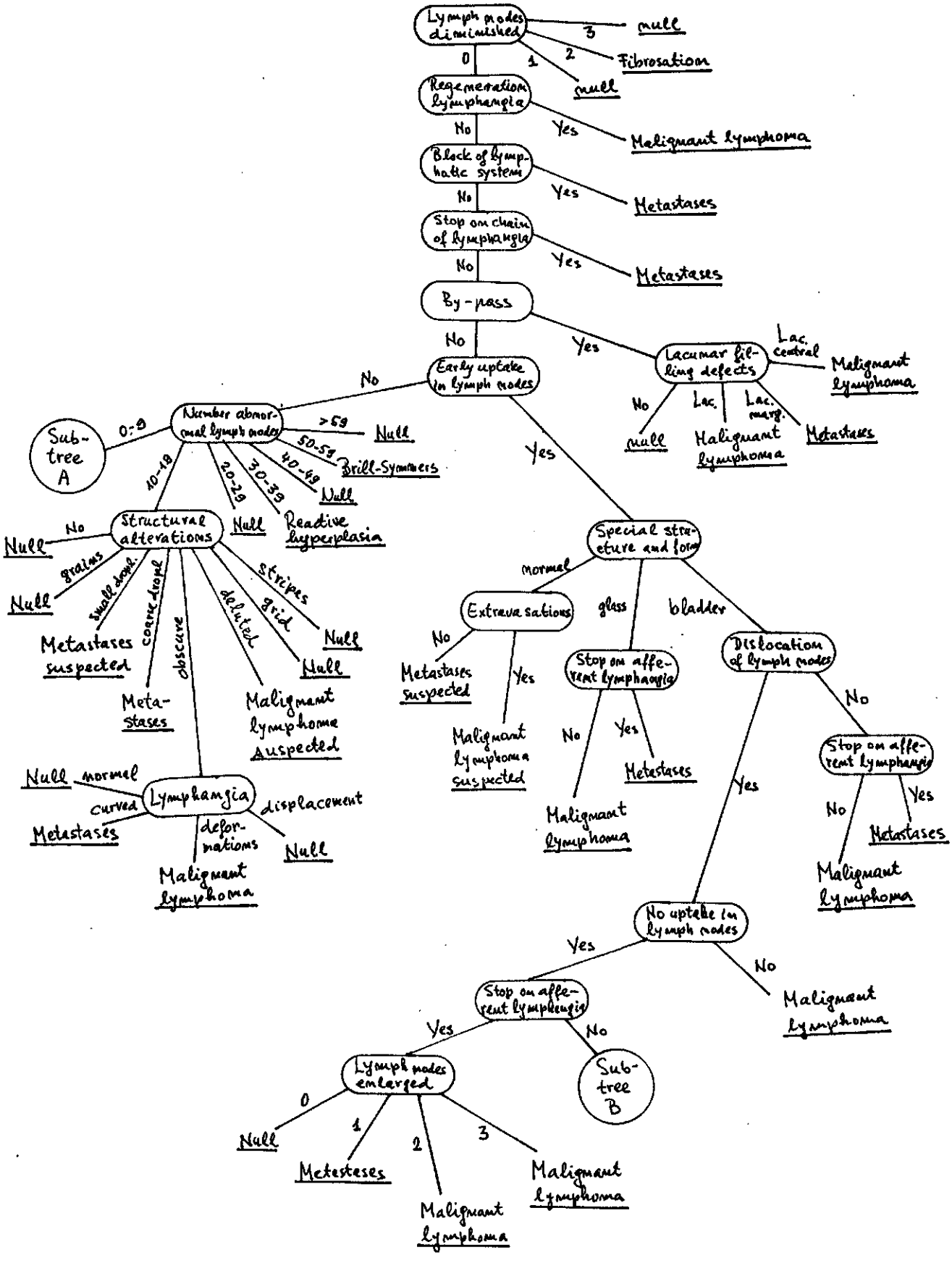


Figure 1: A diagnostic rule for lymphographic investigation.

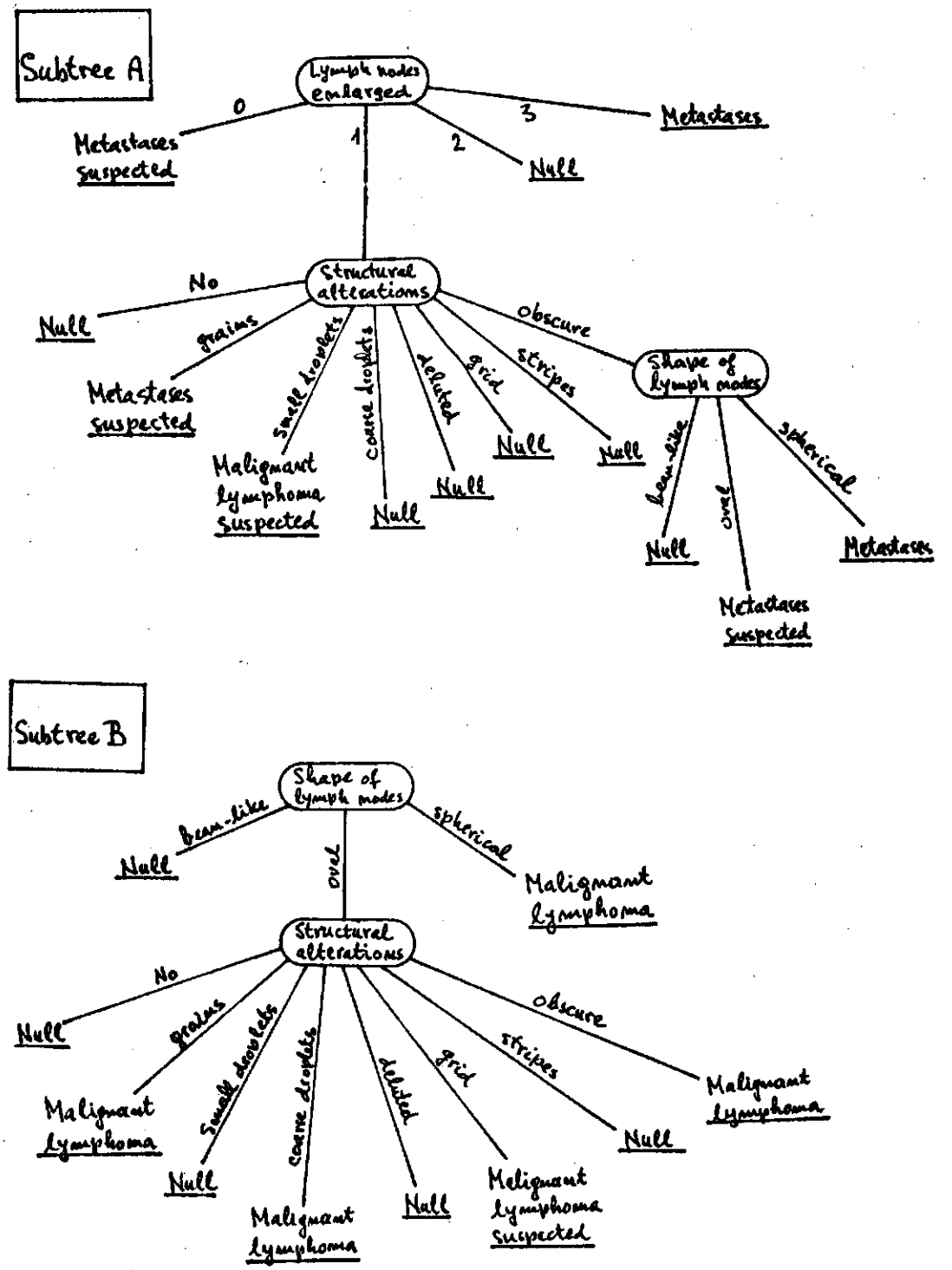


Figure 1: Continued.

Lymphographic attributes

1. lymphangia:
   0 normal
   1 curved
   2 deformations
   3 displacement

2. Stop on afferent lymphangia:
   1 no
   2 yes

3. Stop on chain of lymphangia:
   1 no
   2 yes

4. Block of lymphatic system:
   1 no
   2 yes

5. By-pass:
   1 no
   2 yes

6. Extravasations:
   1 no
   2 yes

7. Regeneration lymphangia:
   1 no
   2 yes

8. Early uptake in lymph-nodes:
   1 no
   2 yes

9. Lymph nodes diminished:
   0
   1
   2
   3

10. Lymph nodes enlarged:
   0
   1
   2
   3

11. Shape of lymph nodes:
   1 bean-like
   2 oval
   3 spherical

12. Various filling defects:
   1 no
   2 folicular
   3 big central
   4 small defects

13. Lacunar filling defects:
   1 no
   2 lacunar
   3 lacunar marginal
   4 central

14. Structural alterations:
   1 no
   2 grains
   3 small droplets
   4 course droplets
   5 deluted
   6 grid
   7 stripes
   8 obscure

15. Special structure and form:
   1 glass
   2 bladder

16. Dislocation of lymph nodes:
   1 no
   2 yes

17. No uptake in lymph nodes:
   1 no
   2 yes

18. Number of abnormal lymph nodes:
   0 0-9
   1 10-19
   2 20-29
   3 30-39
   4 40-49
   5 50-59
   6 more than 59

Diagnoses

1 normal
2 reactive hyperplasia
3 metastases suspected
4 malignant lymphoma suspected
5 metastases
6 malignant lymphoma
7 Brill-Symmers
8 fibrosation
9 other diseases

Table 1: Symptoms and diagnoses in lymphographic investigation.

random permutation of the data.

Diagnostic rules were evaluated in two ways: by "absolute precision" and by "relative precision". The relative precision was based on the physicians judgement on the seriousness of particular errors in diagnosis. Thus each possible case of misclassification was assigned a penalty value according to the physician's feeling of how serious was the difference between the wrong and the correct diagnosis.

Absolute precision is the percentage of unsuccessfully diagnosed samples. The following cases were counted as unsuccessful diagnosis:

- the patient falls into a leaf of the decision-tree labelled by another diagnosis;
- the patient falls into a leaf of the decision tree labelled by "null" (that is a leaf which did not match any example in the training set, and therefore the class of this leaf was not known);
- the patient falls into a leaf labelled "search" (that means that in this case the attributes are insufficient for unambigous diagnosis; this situation arises if patients with the same symptoms in the training set were diagnosed differently).

The last case above indicates a sort of insufficiency or incosistency of the training set. It never occured in our set of 150 patients.

The relative precision is computed so that each incorrect diagnosis (the first one of the above three cases) is penalised by a penalty value between 0 and 1. For example, to diagnose a "normal" patient "metastases" is considered to be a most serious error and is therefore penalised by 1. On the other hand, the interchange of the diagnoses "metastases" and "metastases suspected" is a small mistake (penalty 0.1). Table 2 is a penalty matrix for our experiment as proposed by a physician specialised in lymphographic diagnosis.

Table 3 contains some characteristics of the learnt diagnostic rules for all 10 experiments. Columns in the table correspond to the experiments. Each experiment is described by the following parameters:

- the size of the diagnostic rule, i.e. the number of nodes in the decision tree;
- the necessary size of the data-base, i.e. the number of examples in the window which was sufficient for the construction of a decision tree to explain all 100 examples in the training set;
- the number of unknown testing samples which matched a leaf labelled "null";
- the number of unknown testing samples which match a leaf labelled "search" (this was always 0 as our example set was "consistent");
- the number of incorrectly diagnosed samples (case 1 above);

- absolute precision (percentage);
- relative precision (percentage).

Comparatively poor precision in the first experiment can be explained by the fact that the examples in this experiment were not randomly permuted. They were chronologically ordered, covering a few years period. During this period, the human diagnostician's criteria for recognising some of the symptoms were probably changing, which made symptom-patterns of patients, distant in time, incompatible to some extent. The average absolute precision was about 80%, the average relative precision was 88%.

Discussion

To evaluate the above results let us compare the precision of our automatically learned diagnostic rules to that attained by the physicians in practice, and to that of another learning method.

The absolute precision of the lymphographic diagnosis attained by physicians - practitioners in the field, is between 60% and 85%, depending on how experienced is the diagnostician. The 80% average precision of our system compares quite favourably with this 60 - 85% interval.

M.Soklič carried out, at the Institute of Oncology, another experiment in automatic learning using the same medical data and employing his own learning method based on quasi-spherical partitioning of the pattern-space (Raziskovalna skupnost Slovenije, 1978). The precision obtained by that method was: absolute 62%, relative 70%.

These comparisons indicate that our automatically derived diagnostic rule could be successfully applied in the practice of lymphographic diagnosis. Unfortunately a straight-

forward use of our decision tree by the physician would still require considerable physician's knowledge about lymphographic investigation. This knowledge is necessary for the recognition of symptoms (i.e. attribute values) in lymphographs. It seems that for a really helpful application in this diagnostic problem, a much more sophisticated system would be needed. Such a system should guide the user also in recognising particular symptoms, or should itself be capable of recognising visual patterns.

Acknowledgement

The authors would like to thank dr. G. Klanjščček and dr. M.Soklič for advice, advice, and medical data used in our experiment, and dr. M.Zwitter for his suggestions in the preparation of this paper.

References

Hunt, E.B., Martin, J., Stone, P. (1966) Experiments in Induction, Academic Press.

Mulec, P. (1980) Algorithms for automatic learning (Undergraduate thesis). Ljubljana: Faculty of Electrical Eng. (in Slovenian).

Quinlan, J.R. (1979) Discovering rules by induction from large collections of examples, in Expert Systems in the Microelectronic Age (ed. D.Michie). Edinburgh: University Press.

Quinlan, J.R. (1980) Semiautonomous knowledge acquisition, in Expert Systems. London: Infotech.

Diagnosis	1	2	3	4	5	6	7	8	9
1 normal	-	0.33	0.66	0.66	1.00	0.85	0.66	0.50	0.66
2 reactive hyperplasia	0.33	-	0.10	0.33	0.66	0.50	0.10	1.00	0.33
3 metastases suspected	0.66	0.10	-	0.50	0.10	0.50	0.50	0.85	0.33
4 malignant lymphoma suspected	0.66	0.33	0.50	-	0.75	0.10	0.15	0.15	0.50
5 metastases	1.00	0.66	0.10	0.75	-	0.75	0.66	0.50	0.50
6 malignant lymphoma	0.85	0.50	0.50	0.10	0.75	-	0.33	0.15	0.33
7 Brill-Symmers	0.66	0.10	0.50	0.15	0.66	0.33	-	0.85	0.50
8 fibrosation	0.50	1.00	0.85	0.15	0.50	0.15	0.85	-	0.66
9 other diseases	0.66	0.33	0.33	0.50	0.50	0.33	0.50	0.66	-

Table 2: Seriousness of errors in diagnosis.

Index of experiment	1	2	3	4	5	6	7	8	9	10
Rule size	88	80	74	68	53	78	53	58	64	68
Data-base size	82	75	63	62	68	68	65	62	56	62
Null	0	5	3	1	0	6	1	4	4	3
Search	0	0	0	0	0	0	0	0	0	0
Wrong diagnosis	22	6	9	6	10	6	5	8	4	4
Absolute precision (%)	56	78	76	86	80	76	88	76	84	84
Relative precision (%)	80	85	88	91	90	82	93	85	90	91

Table 3: Results in repeated experiments.

APPENDIX: Symptoms and diagnoses in lymphographic investigation
Original form as used at the Institute of Oncology, Ljubljana (in Slovenian)

Limfografski simptomi

1. Mezgovnice:
   0 normalno
   1 loki
   2 deformacije
   3 odriv

2. Blok dovodnih mezgovnic:
   1 ga ni
   2 je

3. Blok mezgovnic verige:
   1 ga ni
   2 je

4. Blok limfatičnega sistema:
   1 ga ni
   2 je

5. Obvoz - by pass:
   1 ni
   2 je

6. Ekstravazati - jezerca:
   1 jih ni
   2 so

7. Regeneracijske mezgovnice:
   1 jih ni
   2 so

8. Zgodnje kopičenje v bezgavkah:
   1 ga ni
   2 je

9. Velikost bezgavk - zmanjšanje:
   0
   1
   2
   3

10. Velikost bezgavk - povečanje:
   0
   1
   2
   3

11. Sprememba oblike bezgavk:
   1 fižol
   2 ovalna
   3 okrogla

12. Polnitveni defekti ramifi:
   1 jih ni
   2 folikularni
   3 veliki centralni
   4 drobci

13. Polnitveni defekti lakularni:
   1 jih ni
   2 lakunarni
   3 lakunarni marginalni
   4 lakunarni centralni

14. Sprememba strukture kopičenja:
   1 je ni
   2 zrnata
   3 drobno kapljasta
   4 grobo kapljasta
   5 razredčena
   6 mrežasta
   7 progasta
   8 zabrisana

15. Posebna struktura in oblika:
   1 kelih
   2 mehur

16. Dislokacija - odriv bezgavk:
   1 ga ni
   2 je

17. Izpad kopičenja bezgavk:
   1 ga ni
   2 je

18. Število prizadetih bezgavk:
   0 0-9
   1 10-19
   2 20-29
   3 30-39
   4 40-49
   5 50-59
   6 več kot 59

Diagnoze

1 normalni izvid
2 reaktivna hiperplazija
3 sumljiv na metastaze
4 sumljiv na maligni limfom
5 metastaze
6 maligni limfom
7 Brill-Symmers
8 fibrozacija
9 ostale bolezni

AN EXTENSION OF CHEBYSHEV'S ITERATION

D.B.POPOVSKI

COLLEGE OF ENGINEERING, UNIVERSITY OF BITOLA, BITOLA

UDK:681.3.51

In this paper an extension of Chebyshev's iteration for solving equations is made. Thus, a new iteration is obtained which is more effective than Chebyshev's. An always-convergent hybrid algorithm for finding roots is described. Two Fortran IV subroutines are given in which Chebyshev's and the new iterations are used. A comparison is made. In 15 equations with 42 roots it is shown that the proposed iteration consumes less computing time than Chebyshev's.

JEDNO PROŠIRENJE ČEBIŠEVLJEVE ITERACIJE. *U radu je izvršeno proširenje Čebiševljeve iteracije za rešavanje jednačina. Tako, dobivena je nova iteracija koja je efektivnija od Čebiševljeve. Opisan je jedan hibridan, uvek konvergentan algoritam za nalaženje korena. Dati su dva Fortran IV potprograma u kojima se koriste Čebiševljeva i nova iteracija. Izvršeno je poredjenje. Na primeru 15 jednačina sa ukupno 42 korena pokazano je da predložena iteracija zahteva manje računskog vremena od Čebiševljeve.*

1. INTRODUCTION

Many one-point iterations for solving equations are proposed in the literature (see [1]-[8]). One of the most popular is Chebyshev's (see [1], [9] or [10]):

$$x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)}\left[1+0.5\,\frac{f(x_i)f''(x_i)}{f'^2(x_i)}\right]. \qquad (1)$$

This iteration is often called the method of tangent parabolas (see e.g. [11]) because it may be derived on the basis of one-point approximation by the parabola whose axis is parallel to the x-axis.

In this paper an extension of Chebyshev's iteration (1) is made. Thus, a new iteration is obtained which is more effective than Chebyshev's.

2. DERIVATION

Consider a nonlinear algebraic or transcendental equation

$$f(x)=0 \qquad (2)$$

and Schröder series (see [2,p.26])

$$r=x+k-0.5\,\frac{f''(x)}{f'(x)}\,k^2+\frac{3f''^2(x)-f'(x)f'''(x)}{6f'^2(x)}\,k^3+\ldots, \qquad (3)$$

where r is a simple root of (2) and

$$k=-f(x)/f'(x)\ . \qquad (4)$$

If we replace r in (3) by x_{i+1} and x by x_i , and if we disregarde the terms of higher order than two, we obtain Chebyshev's iteration (1), and if we disregard the terms of higher order than three and if we set $f'''(x_i)=0$, we obtain a new iteration

$$x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)}\left[1+0.5\,\frac{f(x_i)f''(x_i)}{f'^2(x_i)}\left(1+\frac{f(x_i)f''(x_i)}{f'^2(x_i)}\right)\right]. \qquad (5)$$

Both iterations (1) and (5) require three evaluations per iteration step: $f(x_i)$, $f'(x_i)$ and $f''(x_i)$. Iteration (5) is more complicated than iteration (1), but,as we shall see in section 6., it is more effective.

3. RATE OF CONVERGENCE

It is clear from (3) that iterations (1) and (5) are cubically convergent to simple roots. Chebyshev's iteration (1) has an asymptotic error constant

$$C_1=\frac{3f''^2(r)-f'(r)f'''(r)}{6f'^2(r)}\ , \qquad (6)$$

while iteration (5) an asymptotic error constant

$$C_2=-f'''(r)/[6f'(r)]\ . \qquad (7)$$

4. ALGORITHM

Neither iteration (1) nor iteration (5) have a guaranteed convergence. To make them into algorithms with guaranteed convergence they must be combined with some always-convergent method (see [12]-[13]). The most simple such method is the bisection method. The following is a description of a hybrid algorithm in which we use this (as an auxiliarly method) in place of iterations (1) and (5) when they break down. Similar such hybrids were proposed in [14]-[17].

Suppose that a real root of (2) *has been bracketed by some initial approximation* x_i *and a bracketing point* x_b. *Calculate* $f(x_i)$ *and* $f(x_b)$, *and test if*

$$signf(x_i) \neq signf(x_b) \quad . \tag{8}$$

(a) *Calculate* $f'(x_i)$ *and* $f''(x_i)$ *. Find* x_{i+1} *by iterations* (1) *or* (5) *provided that this point is between* x_i *and* x_b*. Otherwise get* x_{i+1} *by the bisection step*

$$x_{i+1}=0.5(x_b+x_i) \quad . \tag{9}$$

Calculate $f(x_{i+1})$*. If*

$$signf(x_{i+1}) \neq signf(x_i) \quad , \tag{10}$$

set $x_b=x_i$ *. In any case, replace* $(x_i,f(x_i))$ *by* $(x_{i+1},f(x_{i+1}))$ *and return to* (a).

5. SUBROUTINES

In this section we give two Fortran IV realizations of the algorithm described below. In the first, subroutine HCBM (see figure 1), Chebyshev's iteration (1) is used, while in the second, subroutine HPBM (see figure 2) , the new iteration (5) is used.

Description of parameters:

S - *Name of the external subprogram used. Its parameter list must be* F,D1,D2,X,J. *If* J=0, *it computes , for given argument* X , *only the function value* F . *Otherwise, it computes* F *and the values of the first two derivatives* D1 *and* D2.

A - *Input value which specifies the initial approximation* x_i .

B - *Input value which specifies the bracketing point* x_b.

T - *Input value which specifies the tolerance of the relative error for successive iterates.*

R - *Resultant root of the equation* $f(x)=0$.

I - *Resultant error parameter coded as follows:*
- I=1 - *Basic assumption* (8) *is not satisfied.*
- I=2,3,...,M - *No error.* I *is the number of iteration steps i.e. calls of subroutine* S.
- I=M+1 - *No convergence after* M *iteration steps.*

M - *Maximum number of iteration steps permitted.*

6. COMPARISON

To compare subroutines HCBM and HPBM i.e. iterations (1) and (5), a considerable number of examples were run on an IBM 1130 computer using floating-point arithmetic with 24-bit mantissa. The results are given in table 1. A special subprogram for time measuring on IBM 1130 is used [18]. It is clear from table 1 that proposed iteration (5) is more effective than Chebyshev's iteration (1): it is faster and consumes less computing time.

7. REFERENCES

1. J.F.Traub: *Iterative Methods for the Solution of Equations*. Prentice-Hall , Englewood Cliffs , 1964, Ch.5.
2. A.M.Ostrowski: *Solution of Equations in Euclidean and Banach Spaces* (Third ed. of Solution of Equations and Systems of Equations). Academic Press , New York and London, 1973.
3. A.W.M.Nourein: *Root Determination by Use of Padé Approximation*. BIT 16(1976), 291-297.
4. E.Hansen and M.Patrick: *A Family of Root Finding Methods*. Numer. Math. 27(1977), 257-269.
5. D.B.Popovski: *Solutions of Equations on the Basis of One-Point Logarithmic Approximation*. Proc. 14th Yug. Symp. on Information Processing - Informatica '79, Bled 1.-6. October 1979, 3/202.


```
      SUBROUTINE HCBM(S,A,B,T,R,I,M)
      I=2
      C=B
      D=A
      CALL S(E,F,G,C,0)
      CALL S(F,G,H,D,1)
      IF(E)1,15,2
    1 IF(F)16,17,5
    2 IF(F)5,17,16
    3 C=D
    4 D=R
      F=E
    5 IF(G)6,10,6
    6 E=F/G
      G=(0.5*E*H/G+1.)*E
      R=D-G
      IF(R-C)7,10,8
    7 IF(G)9,10,10
    8 IF(G)10,10,9
    9 IF(ABS(R)*T-ABS(G))11,18,18
   10 R=(C+D)*0.5
   11 I=I+1
      IF(I-M)12,12,18
   12 CALL S(E,G,H,R,1)
      IF(E)13,18,14
   13 IF(F)4,18,3
   14 IF(F)3,18,4
   15 R=C
      RETURN
   16 I=1
   17 R=D
   18 RETURN
      END
```

Fig.1 Subroutine HCBM: hybrid Chebyshev-bisection method

```
      SUBROUTINE HPBM(S,A,B,T,R,I,M)
      I=2
      C=B
      D=A
      CALL S(E,F,G,C,0)
      CALL S(F,G,H,D,1)
      IF(E)1,15,2
    1 IF(F)16,17,5
    2 IF(F)5,17,16
    3 C=D
    4 D=R
      F=E
    5 IF(G)6,10,6
    6 E=F/G
      G=E*H/G
      G=((G+1.)*G*0.5+1.)*E
      R=D-G
      IF(R-C)7,10,8
    7 IF(G)9,10,10
    8 IF(G)10,10,9
    9 IF(ABS(R)*T-ABS(G))11,18,18
   10 R=(C+D)*0.5
   11 I=I+1
      IF(I-M)12,12,18
   12 CALL S(E,G,H,R,1)
      IF(E)13,18,14
   13 IF(F)4,18,3
   14 IF(F)3,18,4
   15 R=C
      RETURN
   16 I=1
   17 R=D
   18 RETURN
      END
```

Fig.2 Subroutine HPBM : hybrid Popovski-bisection method

6. D.B.Popovski: *Numerical Solution of Equations on the Basis of Approximation by the Curve* $(x-p_1)[y(x)-p_2]^2-p_3=0$. Int.J.num.Meth.Engng 14(1979,1574.

7. D.B.Popovski: *A Family of One-Point Iteration Formulae for Finding Roots.* Inter.J.Computer Maths 8(1980), 85-88.

8. G.V.Milovanović and Dj.R.Djordjević: *The Solution of Nonlinear Equations using Iterative Processes Derived from Exponential Approximation* (Serbo-Croatian) . Proc. 4th Bos. - Herc. Symp. on Informatics - Jahorina'80, Jahorina 24. - 28. Mach 1980, Vol.2, 465/1-465/5.

9. I.S.Berezin and N.P.Zidkov: *Computational Methods* (Russian). Fizmatgiz, Moscow, Vol.2,140.

10. B.Sendov and V.Popov : *Computational Methods* (Bulgarian). Izdat. Nauka i Izkustvo, Sofia, 1976, Vol.1, 213.

11. V.A.Variuhin and S.A.Kaslaniuk : *On the Iterative Methods for Finding Equation Roots* (Russian). Zhur. Vichisl.Mat. i Mat.Fiz. 9(1969),684-687.

12. S.M.Pizer : *Numerical Computing and Mathematical Analysis.* SRA, Chicago-Paris, 1975, 216.

13. R.F.King: *Methods without Secant Steps for Finding a Bracketed Root.* Computing 17(1976), 49-57.

14. D.B.Popovski : *A Root Finding Algorithm.* Prilozi (Association for Science and Art-Bitola) 30--31(1979), 289-293.

15. D.B.Popovski: *A Hybrid Algorithm for Finding Roots.* Informatica 3(1979), 3, 16-17.

16. D.B.Popovski: *On an Algorithm for Finding Function Zeros* (Serbo-Croatian). Informatica 4(1980), 1 ,47--48.

17. D.B.Popovski and P.B.Popovski : *On Jarratt's Two-Point Method for Finding Roots.* Proc. 4th Bos. - Herc. Symp. on Informatics - Jahorina'80, Jahorina 24.-28. Mach 1980, Vol.2, 413/1-413/5.

18. Computing Laboratory , Department of Electrical Engineering , University of Belgrade : personal communication.

Table 1 Numerical examples (T=1.E-5)

	$f(x)$	A	B	R	I		Execution time [s]	
					HCBM	HPBM	HCBM	HPBM
1	$x^3-14x^2+57x-65$	0.	3.	1.92...	5	5	0.080	0.085
2		3.	5.	4.39...	5	5	0.075	0.079
3		5.	8.	7.68...	7	6	0.118	0.106
4	$x^4-20x^3+133x^2-330x+236$	0.	2.	1.18...	5	5	0.097	0.103
5		2.	4.	3.42...	5	5	0.096	0.101
6		4.	8.	6.57...	5	5	0.099	0.105
7		8.	9.	8.81...	6	5	0.124	0.106
8	$x^5-4x^4-66x^3+196x^2+713x-1000$	-9.	-5.	-6.97...	5	5	0.114	0.119
9		-5.	-3.	-3.11...	5	5	0.113	0.117
10		-3.	3.	1.17...	6	6	0.138	0.145
11		3.	5.	4.85...	6	6	0.141	0.146
12		5.	9.	8.04...	5	5	0.112	0.117
13	x^6+5x^5-1000	-6.	0.	-5.25...	5	4	0.090	0.074
14		0.	6.	2.65...	5	5	0.079	0.094
15	x^7-x^6-1	-5.	5.	1.25...	17	13	0.335	0.269
16	$x^8-6x^6-8x^5-3x^4-1$	-5.	0.	-1.39...	12	10	0.308	0.264
17		0.	5.	3.00...	8	7	0.187	0.166
18	$x^3-17x^2+76x-60-\sqrt{x}$	1.	3.	1.02...	3	3	0.070	0.073
19		3.	8.	5.87...	5	5	0.127	0.133
20		8.	15.	10.08...	6	5	0.157	0.133
21	$10(1-x)e^{1-x}+1$	0.	2.	1.11...	6	5	0.108	0.093
22		2.	5.	4.57...	6	5	0.101	0.085
23	$ln(0.25x^2-2x+4.5)-1$	0.	2.	1.02...	4	4	0.089	0.082
24		2.	9.	6.97...	5	5	0.113	0.119
25	$x^3-20x^2+121x-210-\sin x$	0.	5.	3.00...	5	5	0.119	0.124
26		5.	8.	6.94...	5	5	0.123	0.128
27		8.	12.	9.97...	7	4	0.182	0.100
28	$0.125x-1-\cos x$	0.	3.	2.35...	6	6	0.112	0.118
29		3.	5.	4.22...	5	4	0.094	0.077
30		5.	10.	7.87...	5	5	0.093	0.099
31		10.	12.	11.44...	5	5	0.094	0.099
32		12.	14.	13.39...	6	5	0.116	0.100
33	$x^3-x^2-12x+5-arctgx$	-10.	-2.	-3.26...	6	6	0.165	0.172
34		-2.	2.	0.37...	5	5	0.128	0.133
35		2.	5.	3.86...	5	5	0.133	0.138
36	$0.05x^{2.2}-x+2$	1.	5.	2.31...	4	4	0.089	0.093
37		5.	15.	10.10...	5	4	0.117	0.094
38	$x^3-35x^2+350x-1000+lnx-e^{-x}$	1.	8.	4.97...	5	5	0.146	0.152
39		8.	12.	10.04...	4	4	0.114	0.118
40		12.	25.	19.97...	6	6	0.181	0.188
41	$x^2-10x+24-arctgx-\sqrt{x}$	1.	5.	3.00...	5	4	0.163	0.130
42		5.	10.	7.26...	5	5	0.163	0.168
				Total	241	221	5.403	5.135
				Index	109	100	105	100

PROJEKTIRANJE Z INTEGRIRANIMI MIKRORAČUNALNIKI

MILJAN DAVOR

INSTITUT „JOŽEF STEFAN", LJUBLJANA

UDK:681.3–181.4

Podan je pregled nekaterih tipov integriranih mikroračunalnikov in nekaj njihovih osnovnih tehničnih lastnosti, na katere mora biti pozoren uporabnik, ki prvič uporablja integrirane mikroračunalnike. Prikazan je primer ekonomskega izračuna upravičenosti uporabe integriranih mikroračunalnikov. V prispevku je dan poseben poudarek na domačem integriranem mikroračunalniku Iskra EMZ 1001.

DESIGNING WITH SINGLE CHIP MICROCOMPUTERS. The small size of microcomputers make them ideal for replacing discrete logic in many consumer applications. This article sets out for first time users some design considerations, illustrating these with a case study. Reliability and hidden costs are considered, and some criteria for selecting an appropriate microcomputer are presented.

1. UVOD

Velike spremembe na področju mikroprocesorjev so nastale s pojavom integriranih mikroračunalnikov. Ti so odprli nove možnosti tudi proizvajalcem mikroračunalniških aplikacij. V prispevku bodo predstavljeni nekateri značilni tipi integriranih mikroračunalnikov. Podana bo ocena ekonomske in tehnične opravičenosti uporabe integriranih mikroračunalnikov. Namreč, kljub več kot petdesetim različnim tipom mikroprocesorjev in mikroračunalnikov, ki se ponujajo na svetovnem trgu, je samo manjše število v široki uporabi. Seveda se v zadnjem času vedno več proizvajalcev mikroračunalniških aplikacij odloča tudi za uporabo integriranih mikroračunalnikov.

2. INTEGRIRANI MIKRORAČUNALNIKI

Najpomembnejša pridobitev pri procesorjih t.i. tretje generacije, je združitev (integracija) ROM in RAM pomnilnikov, časovnega vezja in vhodno/izhodnega vezja v eno samo integrirano vezje skupaj s centralno enoto (CPU). Eden pomembnejših faktorjev, ki je vplival na razvoj integriranega mikroračunalnika je bil razvoj 5 V EPROM pomnilnika. EPROM pomnilnik je veliko pripomogel k razvoju prototipov. Prototipni program, ki je bil razvit na EPROM verziji integriranega mikroračunalnika, se lahko po testiranju prenese na isti tip mikroračunalnika v verziji z maskiranim programom.

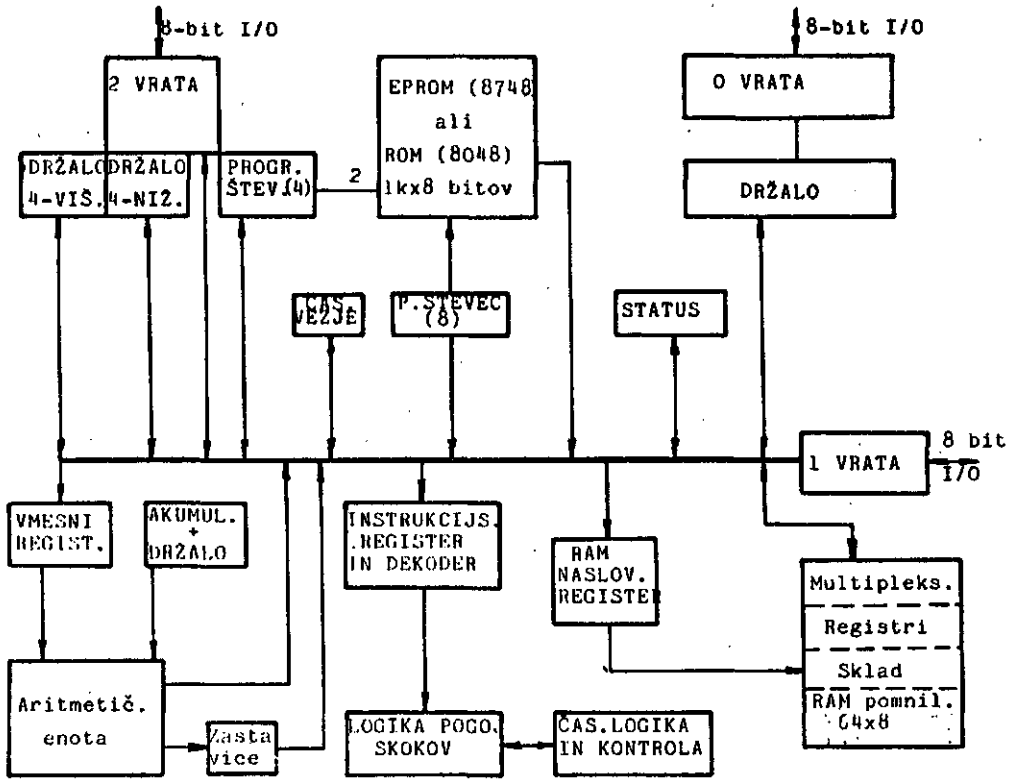
Veliko integriranih mikroračunalnikov je prilagojeno svojim procesorskim predhodnikom, glede programiranja ali periferije ali obeh. Čeprav so integrirani mikroračunalniki predvideni za samostojno delovanje, več proizvajalcev ponuja dodatna (ekspanzijska) integrirana vezja. Ta omogočajo razširitev ROM ali RAM pomnilnika ali vhodno/izhodnih zmožnosti sistema. Seveda, so v tem primeru integrirani mikroračunalniki zgrajeni tako, da omogočajo omenjene razširitve. Notranja organizacija mikroračunalnika je ostala enaka, vključene so še funkcije, ki omogočajo izvajanje dodatnih možnosti.

Značilen predstavnik 8 bitnega integriranega mikroračunalnika je Intel 8048. Slika 1. kaže poenostavljen blokovni diagram tega mikroračunalnika.

Mikroračunalnik 8048 je zgrajen tako, da omogoča uporabo 8080 LSI perifernih vezij, čeprav mikroračunalnik 8048 in mikroprocesor 8080 nimata isti pomen nožic (pin compatibiliti) in nista programsko kompatibilna.

ROM pomnilnik v mikroračunalniku 8048 ima kapaciteto 1Kx8 bitov. Adresno vodilo ni direktno dostopno od zunaj. Vodilo nadomešča troje 8 bitnih izhodno/vhodnih vrat, ki jih lahko uporabimo na dva načina:

i) Ena vrata lahko služijo kot dodatek k podatkovnem vodilu (vrata lahko služijo ne samo za prenos podatkov, temveč tudi za prenos nižjega dela adrese za zunanji pomnilnik - če le tega vključimo v sistem), medtem, ko se ena od preostalih vrat uporabi za prenos 4 bi-



Slika 1.

tov za selektiranje strani zunanjega pomnilnika.

ii) v primeru, ko ni vključen zunanji pomnilnik, se vrata uporabljajo, kot 8-bitna vhodno-izhodna vrata z zadrževanjem podatkov (Latch).

Za razliko od opisanih vhodno/izhodnih vrat, ki so "navidezno dvosmerna", je podatkovno vodilo mikroračunalnika 8048 v resnici dvosmerno [1].

Običajno ponujajo proizvajalci integriranih mikroračunalnikov "družine" med seboj kompatibilnih vezij v verzijah: z A/D ali D/A pretvorniki, z več RAM ali ROM pomnilnika, z EPROM pomnil., hitrejše verzije, z serijskimi V/I vrati, z UART-om ali USART-om, z večbitnimi števci itd.
Primer takšne družine mikroračunalnika i8048:

8035	verzja brez lastnih pomnilikov ROM,RAM,
8035L	-"- z zadrževanjem vsebine RAM pomnilnika pri nižji napajalni napetosti,
8048-6	verzija z 6MHz osnovno frekvenco
8748	verzija z EPROM pomnilnikom

Veliko proizvajalcev mikroračunalnikov je že dalo na trg integrirana vezja, ki lahko samostojno upravljajo zaprte sisteme. Veliko le-teh je programsko kompatibilno s splošno uporabnimi mikroprocesorji istega proizvajalca. Takšen primer je z mikroračunalniki:
- Z8 izhaja iz Z80
- 6801 izhaja iz 6800
- 3870 izhaja iz F8

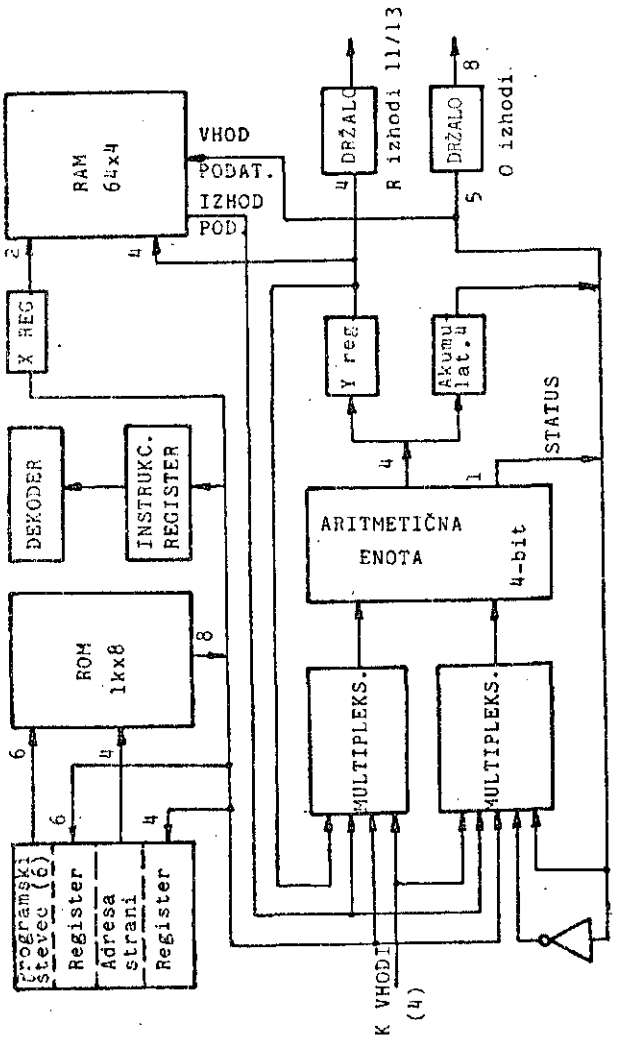
Nekaj osnovnih tipov integriranih mikroračunalnikov kaže Tabela 1.

2.1. Štiri bitni mikroračunalniki

Poleg osem bitnih mikroprocesorjev (mikroračunalnikov) predstavljajo posebno podmnožico štiri bitni mikroračunalniki, ki jih velikokrat imenujejo tudi "mikrokontrolerji".

Tudi tovrstni mikroračunalniki samostojno delujejo v zaprtih sistemih. Omenili bomo značilnega predstavnika mikrokontrolerjev, TMS 1000, ki se je že široko uveljavil v svetu. Osnovni blokovni diagram kaže slika 2. Notranja organizacija mikroračunalnika TMS 1000 je osnovana na treh registrih, na štiribitnem akumulatorju in dveh naslovnih registrih. Pri določenih instrukcijah sta naslovna registra lahko uporabljena tudi kot podatkovna registra.

Osnovne operacije tega (in podobnih) mikroračunalnikov se bistveno ne razlikujejo od instrukcij pri 8-bitnem procesorju. Posebnost omenjenega 4-bitnega mikroračunalnika je vhodno/izhodna organizacija. Po diagramu na sliki 2 lahko sklepamo, da obstajata dva tipa izhodov. Izhodi R so dekodirani iz 4 bitov v 11 ali 13, izhodi O so 8-bitni izhodi iz PLA (programmed logic array) enote. Izhod iz PLA enote je funkcijsko sestavljen iz petih vhodnih linij (4 linije iz akumulatorja in akumulatorska zastavica - flag).



Slika 2.

Takšna organizacija izhodov postane zelo uporabna, ko je potrebno formatirati vsebino akumulatorja po bitih.

Omenili bomo še domači 4-bitni integrirani mikroračunalnik Iskra EMZ 1001, ki je zanimiv v domačem okolju. Ta se nekoliko razlikuje od opisanega mikroračunalnika TMS 1000 (kot je razvidno iz slike 3) posebno v organizaciji vhodno/izhodnih vrat. Pri EMZ 1001 sta na zunaj dostopna 13 bitno adresno in 8 bitno (Tri-state) podatkovno vodilo, s tem, da se vrednosti na obeh vodilih lahko tudi zadržujejo (latched). Obdelava podatkov znotraj mikroračunalnika poteka preko 4-bitnih besed, vhodno/izhodna ter kontrolna organizacija pa sta 8 bitni [2]. Vodili I in K, ki sta 4-bitni, služita samo kot vhoda. Integrirani mikroračunalnik EMZ 1001 proizvaja tudi ameriška firma AMI (American microsystems, inc) pod oznako S 2000 [3].

3. UPORABA INTEGRIRANIH MIKRORAČUNALNIKOV

V primeru, ko želi proizvajalec nadomestiti "diskretno" logiko neke aplikacije z vezjem manjših dimenzij (višja stopnja integracije) se kot rešitev ponuja uporaba integriranega mikroračunalnika.

Majhna poraba energije, enostavnost upravljanja in kontrole izdelka in dimenzije integriranih mikroračunalnikov bo povečala zanimanje za njihovo uporabo. Nadaljnja pridobitev je povečanje zanesljivosti, ki jo prinaša visoka integracija (LSI).

Pri vseh teh pridobitvah in enostavnosti integriranih mikroračunalnikov nas ne smejo presenetiti možne visoke cene in dolgi časi razvoja aplikacij. Maskiranje programa v integrirani mikroračunalnik lahko traja dalj kot 2 meseca. Majhna napaka v programu bo zahtevala novo masko, nova 2 meseca in novih nekaj deset milionov stroškov, zato je potrebna analiza ekonomske in tehnične upravičenosti njihove uporabe.

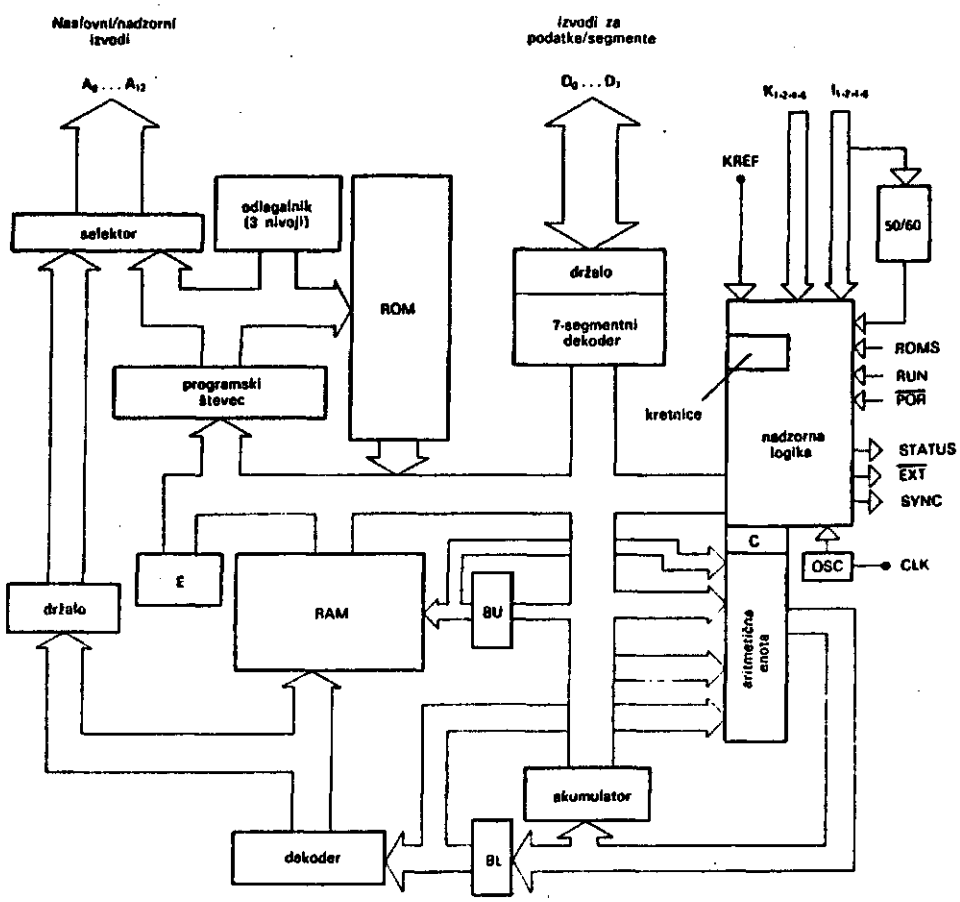
4. IZBOR INTEGRIRANEGA MIKRORAČUNALNIKA

Omenili bomo nekaj osnovnih tehničnih podatkov na katere mora biti pozoren bodoči uporabnik integriranih mikroračunalnikov. Eden od podatkov je napajalna napetost integriranega vezja. Ta je npr. pri NMOS tehnologiji tipično 5 V, kar ne prinese nevšečnosti, če je ostalo vezje, ki obdaja mikroračunalnik, zgrajeno v TTL tehnologiji. Dodatni stroški nastanejo v primerih, ko so v vezju avdio in mehanični elementi (releji, motorji,...) in je potrebno za ta vezja zgraditi dodatni napajalnik.

Naslednji pomemben podatek o integriranih mikroračunalnikih je število vhodov in izhodov, ki so integrirani v vezju. Samo omejeno število integriranih mikroračunalnikov vsebuje na zunaj dostopna vodila, ki so lahko vhodna ali izhodna. V nekaterih primerih so vodila organizirana kot skupina V/I vodil, ali pa so posamezna vodila vnaprej definirana kot vhodna ali izhodna, Če so vhodno/izhodna vodila neustrezna, ali jih je manj, kot zahteva aplikacija, bo potrebno vgraditi dodatna V/I vrata. Dodatna vezja zvišajo ceno sistema, fizične dimenzije tiskane plošče ter povečajo čas izdelave in testiranja takšnega sistema, kar postavi pod vprašanje tudi upravičenost uporabe integriranega mikroračunalnika v takšnih primerih.

Pomen podrobne predhodne študije vseh tehničnih lastosti mikroračunalnikov lahko predstavimo na primeru:

- Predpostavimo, da piše v tehničnem opisu nekega mikroračunalnika, da mora reset signal za popolno inicializacijo registrov, programskega števca, sklada itd., trajati najmanj 'n' računalniških ciklov. Če pri študiji mikroračunalnika ta podatek nismo upoštevali, lahko



Slika 3.

zapademo v nepričakovane težave pri programiranju.

- Podobne težave opazimo v primeru, ko zaradi neustreznega napajalnika, recimo, motnje v električni mreži pridejo do mikroračunalnika in sprožijo delni reset.

Omenimo še zunanjo testno točko, ki je v veliko pomoč pri testiranju (servisiranju) končnega proizvoda in jo lahko uporablja strokovnjak ali nestrokovni uporabnik. Istočasno naj proizvajalec s sistemskim programom vpiše v ROM pomnilnik integriranega mikroračunalnika še majhno testno rutino. Testna rutina naj bi opravila nekaj značilnih testov zunanjega vezja, sproži se lahko z vgrajenim mikro pretikalom (na vezju, na prednji plošči naprave ali na skritem mestu, ki ga pozna samo serviser).

Opisani pripomoček, seveda ne pride v poštev v primerih, ko je interni ROM pomnilnik mikroračunalnika že zaseden s sistemskim programom. Pri tem lahko omenimo, da dodajanje zunanjega ROM pomnilnika občutno zviša celotno ceno proizvoda z integriranim mikroračunalnikom. Uporaba mikroračunalnika v sistemih, ki rabijo več kot 4K - besed ROM pomnilnika je zelo vprašljiva. Pri tem ne smemo pozabiti, da z dodajanjem zunanjega ROM pomnilnika, (1-2 K besed) drastično zmanjšamo število vhodno/izhodnih linij, ker rabimo za adresiranje zunanjega pomnilnika 10-12 adresnih linij. Nekateri proizvajalci integriranih mikroračunalnikov omogočijo reševanje takšnih problemov z vključevanjem "multipleksnega načina" delovanja pri katerem vhodno/izhodne linije imajo različne pomene v posameznih delih računalniškega cikla (a - branje iz ROM pomnilnika, b - splošen pomen) ali ponujajo specialne ekspanzijske ROM pomnilnike za posamezne tipe integriranih mikroračunalnikov.

Velikokrat je pri izbiri mikroračunalnikov pomembno tudi število programskih prekinitvenih nivojev (interrupts), ki omogočajo kontrolo vhodnega vezja, vpliv časovnega vezja in podobno. V nekaterih primerih ne rabimo programske prekinitve, ker spremembe na vhodnih vezjih lahko opazujemo programsko (scanning). Seveda takšen način, plačamo z uporabo dela ROM pomnilnika.

Proizvajalec	Tip	Tehnologija	Napaj. napet. (V)	Dolž. besede (bit)	Čas cikla (µs)	RAM pomn.	ROM pomn.	Št. podpr. nivoj.	Št. prekin. nivoj.	Posebnosti
AMI	S2000	NMOS	9	4	4,5	64x4	1kx8	3		
	S2150	NMOS	9	4	4,5	80x4	1,5kx8	3		
	S2200	NMOS	9	4	4,5	128x4	2kx8	5	3	8 bit A/D, D/A convert.
	S2400	NMOS	9	4	4,5	128x4	4kx8	5	3	-"- -"-
Intel	i8021	NMOS	5	8	8,38	64x8	1kx8	8	1	Zero cros detekcija
	i8022	NMOS	5	8	8,38	64x8	2kx8	8	2	Zero cros det.,A/D conv.
	i8048	NMOS	5	8	2,5-5	64x8	1kx8	8	1	
	i8049	NMOS	5	8	1,36	128x8	2kx8	8	1	
	i8041A	NMOS	5	8		64x8	1kx8			UPI
Mostek Fairchild	3870	NMOS	5	8	1-4*	64x8	2kx8		4	
NEC	uPD7801	NMOS	5	8	2	128x8	4kx8	8	5	Serijske V/I linije vodil. komp. z 8080
	uCOM-43	PMOS/ CMOS	-10/-8 5	4	10	96x4	2kx8	3	1	dva števca (6 bit)
	uCOM-42	PMOS	-10	4	10	96x4	2kx10	4	2	
	uCOM-44	PMOS/ CMOS	-10/-8 5	4	10	64x4	1kx8	1	1	
	uCOM-45	PMOS/ CMOS	-10/-8 5	4	10	32x4	1kx8	1	1	
Signetics	8X300	Schott.	5	8	0,185					
Motorola	6801	NMOS	5	8	4*	128x8	2kx8	8	1	UART
	6805	NMOS	5	8	1*	128x8	1kx8			
RCA	CDP1804	CMOS/ SOS	5/ -10	8		64x8	2kx8			
Zilog	Z8	NMOS	5	8	8*	128x8	2kx8	8	6	UART
Rockwel	MM76	PMOS	-15	4		48x4	640x8	1	1	
	MM77	PMOS	-15	4		96x4	1344x8	2	1	
	MM78	PMOS	-15	4		128x4	2kx8	2	1	8 bit A/D convertor
Western Digital	1872	PMOS	12	4		32x4	1,5kx10	1	1	
General Instruments	PIC1645	NMOS	5	8		24x8	1,5kx12	2	1	
	PIC1650	NMOS	5	8		32x8	1,5kx12	2	1	
	PIC1670	NMOS	5	8		48x8	1kx12	2	1	
National Semicond.	MM5799	PMOS	9	4		96x4	1,5kx8	2		
	MM5781/ 82	PMOS	9	4		160x4	2kx8	2		
Texas Instrum.	TMS1000/ 1200	PMOS	15	4		64x4	1kx8	1	1	
	TMS1100/ 1300	PMOS	15	4		128x4	2kx8	1	1	
	TMS9940	NMOS	5	16		128x8	2kx8	64	4	CRU

* MHz

Tabela 1.

Naslednji tehnični podatek je število programskih nivojev (subrutine level) posameznih tipov integriranih mikroračunalnikov in je odvisen od kapacitete notranjega RAM pomnilnika. Manj kot dva podprogramska nivoja sta običajno nespremenljiva tudi za zelo preproste programe.

Pri nekaterih aplikacijah obstaja potreba po pretvorbi analognih signalov v digitalne. Nekateri integrirani mikroračunalniki (i8022, Z8) imajo tudi to možnost. Intel 8022 hardwarsko pretvarja vhodni analogni signal v 8-bitno binarno kodo (256 razdelkov), po programskem aktiviranju preproste komande. V primerih, kot so procesiranje govora in podobni, ko potrebujemo A/D pretvornike z večjo natančnostjo (12 - 16 bitna binarna koda) lahko mikroračunalniku dodamo zunanje integrirano vezje, ki nam bo to omogočilo (Codec).

Novejši interirani mikroračunalniki vsebujejo eno ali dva interna časovna vezja, ki so zelo uporabljiva v različnih aplikacijah. Običajno je interno časovno vezje mogoče programsko upravljati in kontrolirati, kar lahko uporabimo za periodično kontrolo V/I vezja, za kontrolo izvajanja programa, pri periodičnih meritvah in podobno.

Osnovno frekvenco internih oscilatorjev integriranih mikroračunalnikov lahko določamo z vgrajevanjem R-C konstante ali kvarčnega kristala. V primerih, ko ni potrebna večja natančnost frekvence internega oscilatorja kot 20 %, zadošča cenejša R-C konstanta. Pri časovno kritičnih aplikacijah, kot so komunikacije s točno določeno hitrostjo

prenosa podatkov ali ko rabi sistem referenčno frekvenco moramo, kot referenco za interni oscilator uporabiti kvarčni kristal. Osnovna frekvenca interne ure je pri posameznih tipih mikroračunalnikov zelo različna; giblje se od 1 MHz do več kot 11 MHz (tabela 1.). Podobno je pri času izvajanja instrukcij in velikosti nabora instrukcij. Ne sme nas presenetiti dejstvo, da je lahko mikroračunalnik z manjšim naborom instrukcij, v nekaterih primerih enako učinkovit kot mikroračunalnik z veliko obsežnejšim naborom instrukcij. Da bi zadostili čim večjemu številu uporabnikov zahtev so proizvajalci mikroračunalnikov poskrbeli za široko paleto različnih instrukcij in različnih načinov adresiranja.

Naloga programerja je izbrati ustrezne prijeme, ki jih mikroračunalnik omogoča in s tem kar se da racionalno rešiti dano nalogo. Pri tem ponovno poudarjamo pomembnost podrobne strokovne analize tehničnih lastnosti posameznih tipov integriranih mikroračunalnikov. Saj že krajši pregled različnih možnosti kaže na vso pestrost ponudbe na tem področju.

Poglejmo, kaj je z zanesljivostjo izdelka z integriranim mikroračunalnikom glede na zanesljivost istega izdelka zgrajenega z "diskretnimi" integriranimi vezji. Predpostavimo, da bo integrirani mikroračunalnik nadomestil 45 "diskretnih" integriranih vezij. Takšno vezje (45 integriranih vezij) zaseda dokaj veliko tiskano ploščo, ali nekaj plošč, ki morajo biti na nek način povezane (konektorji). Integrirani mikroračunalnik ne bo samo zmanjšal potrebno površino tiskane plošče, ampak bo tudi zmanjšal število konektorjev, število zalotanih točk in število metaliziranih skoznikov, kar zviša stopnjo zanesljivosti izdelka. Če predpostavimo, da je število metaliziranih skoznikov enako številu nožic integriranih vezji na dvostranski tiskani plošči lahko izračunamo:

Stari sistem (45 "diskretnih" integriranih vezij)

- število skoznikov za nožice integriranih vezij (45x16)	720
- število metaliziranih skoznikov	720
- število skoznikov za konektor	64
Skupno število skoznikov	1504

Novi sistem (integrirani mikroračunalnik)

- število skoznikov za nižice integriranega mikroračunalnika	40
- število skoznikov za dodatno vezje	40
- število metaliziranih skoznikov	80
Skupno število skoznikov	160

Razmerje števil skoznikov: 1504/160 = 9,4

Pri aplikacijah, ki velikokrat poleg integriranega mikroračunalnika rabijo še več dodatnega vezja je to razmerje nekoliko manjše. Primer nazorno kaže na točnost predpostavke, da je sistem z integriranim mikroračunalnikom zanesljivejši.

5. EKONOMSKI POGLED NA PROIZVODNJO APLIKACIJ Z INTEGRIRANIMI MIKRORAČUNALNIKI

Po kratki predstavitvi integriranih mikroračunalnikov in nekaterih lastnosti le-teh bomo pokazali primer gospodarskega pogleda na problem proizvodnje aplikacij z integriranimi mikroračunalniki.

Prvo vprašanje, ki se postavlja pred bodočega proizvajalca je vprašanje ali razvoj prototipa prepustiti zunanjim strokovnim razvijalcem mikroračunalniške opreme in programerjem ali organizirati lastno razvojno skupino. Obe možnosti imata dobre in slabe strani. Seveda je bolj smiselno razvoj aplikacij prepustiti lastni razvojni skupini, če le-ta že deluje in je materialno in programsko opremljena za tovrstne naloge. V primeru, ko se proizvajalec prvič srečuje s področjem mikroračunalnikov, bo organizacija razvojne skupine zahtevala nekaj (veliko) časa in finančnih sredstev, kar nas usmerja v drugo možnost. Ko razvoj prototipa prepustimo zunanjim strokovnjakom, lahko nastanejo problemi, če se npr. v času uporabe izdelka pojavijo kakšni tehnični problemi ali nove zahteve po spremembah izdelka in razvojna skupina, ki je izdelek razvila že sodeluje na nekem drugem projektu tako, da nam ne more pomagati. V primeru, ko imamo lastno razvojno skupino teh problemov ni, imamo pa tudi zagotovljeno servisiranje izdelka. Proizvajalca, ki se prvič sreča z mikroračunalniki, bo samo tehnična, materialna in programska oprema za razvoj novega proizvoda stala najmanj 450 tisoč dinarjev. Pri nakupu opreme mora biti proizvajalec pozoren na to, da bo oprema dovolj fleksibilna (da ima možnosti razširitve in obnavljanja) in da se bo izplačala po približno treh letih proizvodnje.

V nadaljevanju bomo prikazali na primeru, koliko lahko privarčujemo pri povprečnih stroških razvoja.

Osnovne predpostavke:

i) Mnenje proizvajalca je, da bo nek proizvod cenejši, če se obstoječe "diskretno" vezje (analogno in digitalno; skupno 45 integriranih vezij) zamenja z integriranim mikroračunalnikom;

ii) Zagotovljena je petletna prodaja najmanj 5000 komadov letno;

iii) Garancijska doba za proizvod naj bo 1 leto. Predpostavimo, da mikroračunalnik prinaša večjo zanesljivost pred "diskretnimi" integriranimi vezji in zaradi tega pričakujemo eno intervencijo na leto za vsakih 10 prodanih izdelkov;

iiii) Cena proizvodnje izdelka, ki ga kontrolira integrirani mikroračunalnik je nižja kljub dodatnim stroškom razvoja. Ti naj bi se pokrili že z enoletno proizvodnjo.

Kot kaže primer izračuna v tabeli 2, cena razvoja našega izdelka, s strokovnimi razvijalci in s polno opremo, doseže 1 236 500 din. Če se proizvajalec prvič srečuje z mikroračunalniki, nova oprema in usposabljanje strokovnjakov poveča ceno razvoja najmanj na vrednost 1 600 000 din.

V prvem primeru, se bo cena razvoja pokrila že s proizvodnjo prvih 1517 kosov novih izdelkov in je s proizvodnjo 5000 kosov letno možen dobiček 39 500 000 din.

Razvojna stopnja		Cena (din)
Načrtovanje in specifikacija materiala	480hx300 din	144.000
Administrativni posli, risanje	100hx200 din	20.000
Razvoj program. prototipa		210.000
Testiranje in popravljanje napak	150hx250 din	37.500
Maskiranje in odkup 50 protitipnih mikroračunalnikov		135.000
Izdelava 50 poskusnih izdel. z integriranimi mikrorač.	50x13.000 din	650.000
Ureditev dokumentacije (za servisiranje in uporabo)		40.000
Skupni stroški razvoja:		1,236.500

Cena izdelave izdelka brez mikroračunalnika	1.200
Cena izdelka z mikroračunalnikom:	
Integrirani mikroračunalnik	160
Podnožje	55
Konektroji, dodatno vezje	100
Testiranje, sestavljanje	70
Skupna cena:	385
PRIVARČEVANO:	815

Minimalno število proizvodov/leto,
ki bodo pokrili stroške razvoja (1,236.500/815)=1517 kos.

Mogoče je privarčevati letno (5000 kosov)	40,750.000
- stroški za servisiranje 500x2500	1,250.000
Privarčevano/leto:	39,500.000

Tabela 2.

Cena maskiranja programa, cena orodja in testnih naprav je vključena v ceno razvoja, ker so pri razvoju vsake aplikacije tudi dejansko prisotne. Res je, da v primeru, ko že obstoječe vezje zamenjujemo z novim, bolj sodobnim mikroračunalnikom, lahko za testiranje vezja uporabimo tudi nekatere stare testne naprave, ampak običajno se testni postopki nekoliko razlikujejo.

6. ZAKLJUČEK

Namesto zaključka bi lahko pregledali kakšne so možnosti uvajanja integriranih mikroračunalnikov v našem okolju. Primer proračuna proizvodnje, ki smo ga prikazali, se bi težko uporabil pri planiranju proizvodnje za naše (majhno) tržišče, za katero je 5000 tovrstnih izdelkov - letno težko dosegljiva številka. Namen podanega primera je bil prikazati določene relacije med cenami tehnološko starejših in sodobnejših načinov načrtovanja izdelkov.

V tekstu je bil že omenjen domači integrirani mikroračunalnik Iskra EMZ 1001 (AMI S2000). Pri proizvajalcu smo izvedeli nekaj podatkov, ki bodo zanimivi za bodočega uporabnika - proizvajalca aplikacij z integriranimi mikroračunalniki pri nas:

Posamezni kosi mikroračunalnikov EMZ 1001 (10 - 20 kosov) so dobavljivi takoj pri "Iskra - Mikroelektronika" po približni ceni 550 din.

To bo začetna investicija pri razvoju aplikacij s tem mikroračunalnikom. Ko smo začrtali prototip, lahko proizvodnjo izdelkov nadaljujemo s standardnimi mikroračunalniki (ne rabimo banko Ø-ROM pomnilnika) ali s specialno za nas programiranimi mikroračunalniki. V prvem primeru se cena posameznih mikroračunalnikov znižuje odvisno od števila naročenih kosov (1000, 1000, 5000,...) in ni v naprej določeno.

V primeru, ko želimo naročiti specialno verzijo mikroračunalnika, ki bo prilagojena naši nalogi je potrebno vedeti nekaj podatkov:

Najmanjša količina, ki jo lahko naročimo je 5000 kosov. (Masko za program še vedno izdeluje ameriški partner AMI). Cena 5000 kosov mikroračunalnikov z izdelavo maske je med 5600 in 6000 dolarjev, kar pomeni manj kot 1,5 dolarjev (50 din) za kos.

Čas izdelave maske je približno 2 - 3 meseca. Vse cene, ki smo jih navedli, so približne in veljajo v letu 1980 ter so objavljene samo, da bi bodoči uporabnik integriranih mikroračunalnikov dobil občutek cen tovrstnih izdelkov in storitev.

7. LITERATURA

[1] Intel, Component Data Catalog 1980

[2] Mikroračunalnik EMZ 1001, katalog, Iskra - industrija elementov za elektroniko - Mikroelektronika, Izdala: Iskra Commerce 1978.

[3] American Microsystems, Inc. (AMI), S2000 single - chip microcomputer family, Advanced product description, October 1978

[4] Bursky, D and Bodley, N, Microcomputer selection guide, Electronic Design, Vol. 26, No. 11, Maj 1978, pp 65-78

[5] Microprocessor data manual, Electronic Design, Vol. 26 No. 21, October 1978, pp 53-216

[6] Blume & Hetal, Single chip 8 bit microcomputer fills gap between calculator types and power multichip processors, Electronics, 25. November 1976, pp 99-105

[7] I. Whitworth, Review of microprocessor architecture, Microprocessors and microsystems, Vol 3, No 1, jan/feb 1979

[8] M. Moser, Designing with single chip microcomputer, Microprocessors and microsystems, Vol 3, No 3, April 79

ON DATA GRAMMARS AND PARSERS

VITOMIR SMOLEJ, TONI MILOŠ

UDK: 681.3:519.682

VISOKA ŠOLA ZA ORGANIZACIJO DELA, KRANJ
UNIVERZA V MARIBORU
INSTITUTE „JOŽE STEFAN", LJUBLJANA

The article shows, that the approach, proposed by Wirth (1) for the construction of a parser for language grammars, can be usefully aplied in the process of designing programs, which use or produce serial files. It is, however, neccessary, that the programmer understands the grammar of data he is working with. The program thus prodeced is - in the first phase of the program development - a parser for the data grammar designed. It is shown that the approach, proposed here, is identical to the methods, which form the basis of the Michael Jackson program design technology (2).

O PODATKOVNIH GRAMATIKAH IN SINTAKTIČNIH ANALIZATORJIH. Članek prikazuje, kako nam pristop, ki ga je privzel Wirth (2) pri konstrukciji parserja za gramatiko jezika, uspešno pomaga reševati probleme pri snovanju poljubnih programov, ki tvorijo ali pa uporabljajo podatke v obliki zaporednih datotek. Od programerja zahteva metoda razumevanje gramatike podatkov, s katerimi operira. Program, ki ga potem napiše, predstavlja v prvi fazi parser za privzeto gramatiko podatkov. Pokazano je, da je ta pristop identičen metodam, ki so osnova Jacksonovi tehnologiji za snovanje programov (1).

INTRODUCTION

In this article I intend to unify two approaches to program construction, which are very wide appart at first sight: the so-callled Michael Jackson technology (1) and the approach taken by Wirth (2) in parser construction. It will be shown that the two approaches are basically identical, both heavily leaning on the correspondence between the structure of data and the program. They both lead to the following two-step method of the program construction:

parser construction: in this step one tries to project as close as possible the grammar of data to be processed into the structure of the program.

semantic action: here one expands the program primitives ("process-item" for instance) into explicite commands (like "print the value of the item and add it to the grand-total").

It is also shown that it is easy to automate the first part of the program construction process, given of course that the grammar of data is known- the method proposed is in a way a "reverse-engineering" process, where one works back from data towards - a real or imaginary - program which produced data to be processed.

DATA DEFINITON AND STRUCTURE

Borrowing from the terminology used in the language analysis one can define a data grammar as a set of production rules which let one derive a given data set by retracing the steps leading from the starting symbol (a serial file in our case) to the terminal symbols (data items).

A data grammar can thus be viewed as a formalized description of a program which generated these data (if they are to be used as input) or will generate them (if we deal with output data).

We will, for the sake of simplicity, use capital-lettered notation for non-terminal symbols and lower-case lettered names for terminal symbols. The following set of production rules would produce a simple record of personal data:

<PERS-RECORD> --> <id-=> <name> <BIRTH-DATE>

<BIRTH-DATE> --> <day> <month> <year>

where PERS-RECORD and BIRTH-DATE are non-terminal symbols - they can be analyzed into terminal symbols by successive application of the given two production rules-. Symbols, written in lower-case letters (id-=, name, day, month and year) are terminal symbols - we could of course define the production rules which would obtain terminal symbols like digit, character etc -.

The meta-symbol will stand for alternatives in production, for instance:

<RECORD> --> <GOOD>|<BAD>

will produce either the non-terminal <GOOD> or <BAD>.

A symbol bracketed in will stand for repetition of a symbol. For instance:

<FILE> --> {<RECORD>}

stands for short-hand of:

<FILE> --> <RECORD><RECORD><RECORD>....

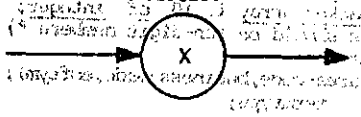
Note that the above production rule can also generate an empty sequence of RECORDs - a case of an empty file in this particular context -.

An alternative to the so-called Bakhus-Naur form of presenting the grammar are the RECOGNITION or SYNTAX graphs, which reflect the flow of control during the process of parsing the sentence of the language.

The method that helped define PASCAL for instance, can also be usefully applied to data grammars. The following rules will help us construct a recognizer graph from the grammar presented in a Bakhus-Naur form used above (What follows is a nearly wor-by-word citation of rules for graph construction in (2), ch.5.2.):

A1 Each nonterminal symbol A is mapped into a recognition graph A, whose structure is determined by the right-hand production according to rules A2 through A6, that follow.

A2 Every terminal symbol x that occured in the production is mapped into an edge labelled x enclosed in a circle.

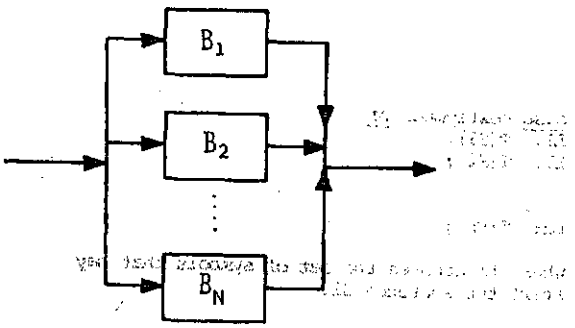


A3 Every occurence of a non-terminal symbol B is represented by an edge labelled B, which stands for the subgraph leading to the recognition of B.

A4 A production having a form:

<A> --> <B1> | <B2>|...| <Bn>

is mapped into:

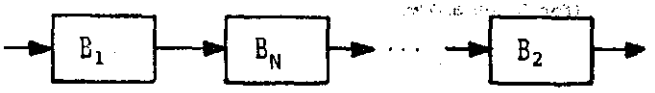


where B1, B2....Bn are subgraphs to be expanded yet.

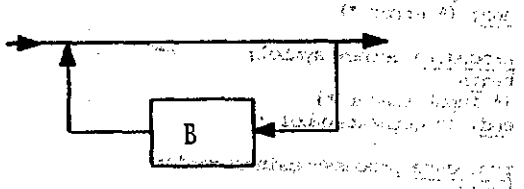
A5 A sequence of symbols:

<A> --> <B1><B2>...<Bn>

is mapped into:



A6 A repetition of the symbol (B) , is mapped into:



Occurence of terminal symbols corresponds to a recognizing statement for symbols read and the advancing of the reader to the next symbol in the input sequence. Nonterminal symbols, when occuring, lead to activation of their recognizer. In graphs the terminal symbols will be denoted by their name in a circle, while non-terminal symbols will be depicted by their name in a square.

Nothing has yet been said about the particular form of the input reader that feeds the next symbol in the input string into the data recognizer. The input data can be represented in several ways. Here is the usual physical representation of data for a particular example of data on sales of salesmen in different areas:

Physical record:
salesman-code
area-code
amount-of-business-made

Let us assume that data are sorted on salesman-code. One salesman then has an iteration of data on his performance in one or several areas. Thus the logical structure of the input data can be represented by the following grammar (in BNF)

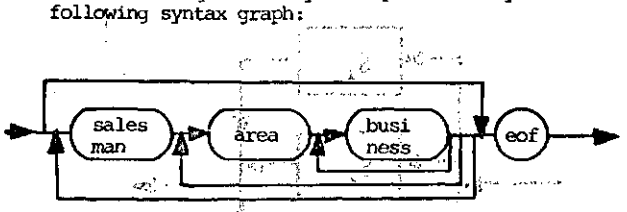
DATA GRAMMAR:

<SALESMAN> --> {<SALESMAN>}<eof>

<SALESMAN> --> <salesman-code>
<AREA DATA> {<AREA DATA>}

<AREA DATA> --> <area-code><business-made>
{<business-made>}

which can equivalently be represented by the following syntax graph:



The physical aspect of data is grammatically different from the way we look at data. Very probably the data are physically stored as a serial file of the following structure:

<SALESMAN-FILE> --> <Sales record> <eof>

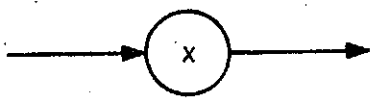
This should serve as a warning that the input reader will not be as simple as one would expect. In our case namely the reader should recognize the records for the same salesman and then "denoise" them by eliminating the salesman's code from the input sequence.

CONSTRUCTING A PARSER FOR A GIVEN SYNTAX

Given the grammar of the input data a program which accepts and parses the data is readily derived. Note that we still deal with the first part of the program construction process, the part that should help us understand the data structure. Nothing has been said about particular processes to be performed on data.

A set of rules for converting the syntax graph into a program can be stated which is equivalent to rules B1 to B6 in (2,ch.5.2.):

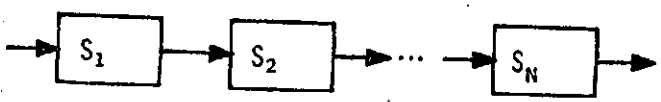
T(S) will stand for the procedure, which recognizes the sentence S.

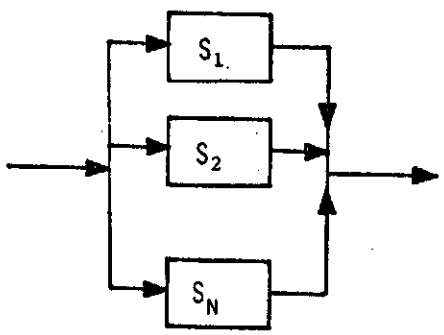


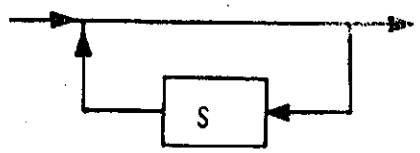

```
if nextsymbol = x
    then read (nextsymbol)
else
    error;
```

where error is a routine which deals with unexpected symbols in the input sequence

GRAMMAR







The rules cited here from (2) are applicable for the case of a context - free grammar where the recognition of the symbol is possible on the basis of knowledge of the present state of computation and on a single next symbol being read. Also, no step in the process of parsing should be revoked later on, which leads us to the common requirement of a one-symbol-lookahead without backtracking.

To be exact, one would have to write production rules right down to the level of elementary items like digits or characters. The question namely is: all the terminal symbols (except eof) in the case of the salesman are mere numbers. How is one able to recognize the semantics of the number the input reader has produced in a given instance?

There should be some way for the input reader which would enable the calling program understand the meaning of the number produced. One of the possible solutions is to introduce a variable of a type termtype:

```
type termtype = (salesman-code, area-code,
business-made,eofsym);
    .
    .
nextsymtype:  termtype;
```

The salesman example should thus be transscripted into the following PASCAL program:

```
program parse (input, output);
type symbol = packed array 1..10  of  integer;
(* all terminals should be ten-digit numbers *)
      termtype=
(salesman-code,area-code,business-made,eofsym);
var nextsymtype : termtype;
```

PROGRAM

```
begin T(S1);(S2)...T(Sn) end
```

```
case nextsymbol of
L1:  T(S1);
L2:  T(S2);
   .
   .
Ln:  T(Sn);
```

where Li denotes the set of symbols that may start the sentence Si.

A loop of repeating symbols:

```
while nextsymbol  in  L  do T(S);
```

(for L see above)

```
procedure error;
begin
(*to process errors, void now *)
end; (* error *)

procedure getnextsymbol;
begin
(* input reader *)
end; (* getnextsymbol *)

procedure process-business-made;
begin
(* processes the terminal business-made *)
end; (* business-made *)
```

```
procedure process-area-code;
(* processes the terminal area-code *);
end; (* process-area-code *)

procedure process-area-data;
(* processes the iteration on one area *)
begin
  if nextsymtype  < > area-code  then
       error
  else
       process-area-code;
  getnextsymbol;
  while nextsymtype = business-made
    do  begin
        process-business-made
    end (* same area *);
end (* area code *);

procedure process-eof;
begin
(* processes the terminal eof *)
end (* eof *);

procedure process-salesman-code;
(* processes the salesman-code terminal *)
end (* salesman-code *);

begin (* main program *)
  getnextsymbol;
  while nextsymtype  < > eofsym do
  begin
       process-salesman-code;
       getnextsymbol;
       while nextsymtype < > salesman - code
          do process-area-data;
  end (* records in the file *);
  process-eof;
end (* program *).
```

MICHAEL-JACKSON TECHNOLOGY

The way the program is built using the technology by Michael Jackson (2) is basically the same as proposed by Wirth (1). To ease the burden of a simple COBOL programmer the following simplifications have been introduced there:

(1) the production rules cannot mix sequence, iteration and selection of data. Thus productions like

    <FILE> --> {<RECORD>} <eof>

    are forbidden. This of course is no problem as any production can be rewritten to abide to the rule. For instance:

    <FILE> --> <FILEBODY><eof>
    <FILEBODY> --> {<RECORD>}

(2) productions of recursive nature are forbidden - which is fair enough, as noone expects a COBOL compiler to alow recursive PERFORMing of paragraphs.

Briefly, Jackson's rules for designing programs are:

1 determine the data structure of each file (both input and output)

2 establish the corresponding components in each data structure

3 Build the program structure

4 List all executable operations in the program

5 Allocate each executable operation to its appropriate place in the program structure

6 Write the program in schematic logic (pseudo code) paying particular attention to the control breaks.

Jaskcon's pseudo code for describing data consists - no surprise - of the basic blocks of structured programming applied to data structure:

A sequence <C> --> <A><B> is described as:

```
C  seq
   DO A
   DO B
C  end
```

A selection <C> --> <A> | <B> is described as:

```
C  sel if condition for A
   do A
C  or if condition for B
   do B
C  end
```

An iteration <C> --> <B> is described as:

```
C  iter while codition for B
   do B
C  end
```

The salesman example would thus be written as:

```
PROGRAM seq
  SALESMEN Body iter while not end of file
      process-salesman-code
      NEW-AREA body iter while same salesman
        process-area-code
        SAME-AREA body iter while same area
          process-business-made
        SAME-AREA end
      NEW-AREA end
  SALESMEN end
  process-end-of-file
PROGRAM end
```

It is evident that the pseudo code proposed by Jackson is still another way of presenting the grammar of data. It could be used as an input to the general parser - as proposed by Wirth - which would authomatically generate the program in COBOL.

Of course the main difficulty lies in defining the grammar of data in question. The process of transforming it into a working program is usually the least difficult part of the job. Still it could be automated letting the programmer concentrate on the difficult parts like the the points 1-4 of the Jackson's rules.

CONCLUSIONS

We have shown that the Jackson's programming technology is actually a simplified version of the methods used for parser construction. It was shown also that extending the analogy further it would be possible to automate the transcription of Jackson's pseudo code into working programs. We plan to develop a parser along these lines which would produce COBOL versions of the input grammars in a form similar to the Jackson's pseudo code.

REFERENCES

1. Jackson M.A.: Principles of program design
Academic Press London 1975

2. Wirth N.: Algorithms+Data Structures = Programs
Prentice-Hall Inc N.J.1976

PROLOG: OSNOVE IN PRINCIPI STRUKTURIRANJA PODATKOV

IVAN BRATKO, MATJAŽ GAMS

FAKULTETA ZA ELEKTRONIKO IN INSTITUT JOŽE STEFAN, LJUBLJANA

UDK:519.682

Prolog je programski jezik, osnovan na formalni logiki. Program v Prologu definiramo z množico trditev in ne s postopkom kot v običajnih, postopkovno orientiranih jezikih. Zato ga štejemo med t.i. nepostopkovne ali deklarativne jezike. Izvajanje programa ustreza formalnemu dokazovanju izreka, da iz danih trditev (podatkov) sledijo ustrezni rezultati. Nekatere značilnosti Prologa močno olajšujejo programiranje operacij na logično kompleksnih podatkovnih strukturah in pa sistemov umetne inteligence. V uvodnem delu tega članka so opisane nekatere osnovne komponente Prologa. Sledijo primeri implementacije podatkovnih struktur, ki kažejo, kako Prologova sintaksa omogoča strukturiranje podatkov brez uporabe kazalcev, kot je to običajno v drugih jezikih, npr. v Pascalu.

PROLOG: FUNDAMENTALS AND PRINCIPLES FOR DATA STRUCTURING. Prolog is a programming language based on formal logic. In Prolog, programs are defined by a set of assertions and not by a procedure as in usual, procedurally oriented languages. Therefore it is called a nonprocedural or declarative language. The execution of a Prolog program corresponds to a formal proof of a theorem that the results of the program logically follow from the given assertions (data). Some of Prolog features greately facilitate the programming of operations on logically complex data structures and of artificial intelligence systems. This paper presents the basics of Prolog followed by a number of examples of implementation of data structures in Prolog. The examples illustrate how the Prolog syntax can be used for data structuring without the use of pointers as e.g. in Pascal or other "usual" languages.

1. ZNAČILNOSTI PROLOGA IN PRIMER PROGRAMA

V zgodnjih sedemdesetih letih je veljala naslednja groba klasifikacija programskih jezikov (Sammet 1969):
"Vse programske jezike lahko razdelimo v dve skupini: v eni je Lisp, v drugi pa vsi ostali jeziki."

S tem je bila podčrtana razlika med programskim jezikom umetne inteligence Lispom (npr. Siklossy 1976) in ostalimi "normalnimi" programskimi jeziki, kot so Algol, Fortran, Cobol, PL1 idr. Odkar eksistira Prolog pa bi bilo umestno to klasifikacijo popraviti takole: Vse jezike lahko razdelimo v dve skupini: v prvi je Prolog, v drugi pa vsi ostali jeziki (vključno z Lispom).

Prolog je enostaven, vendar močan programski jezik. Njegova matematična osnova je formalna logika, v podobnem smislu kot tvorijo matematično osnovo Fortrana in njemu sorodnih jezikov aritmetični izrazi. Osnova Prologa je bila razvita na univerzi v Marseilleu (Roussel 1975) kot praktična realizacija ideje, da je mogoče matematično logiko uporabiti tudi kot programski jezik (npr. Kowalski 1974).

Izrazi predikatnega računa prvega reda, ki tvorijo sintakso Prologa, omogočajo enostavno definiranje relacij med podatki in pa strukturiranje podatkov. Zato je Prolog posebno močan pri simboličnem, nenumeričnem procesiranju in pri delu z bogatimi podatkovnimi strukturami. Mnoge formalizme iz teorije podatkovnih baz, npr. relacijski model, lahko uporabimo praktično neposredno kot program v Prologu. Močna komponenta Prologa je tudi nedeterminizem in z njim povezano avtomatsko vračanje(automatic backtracking), ki močno olajšuje programiranje algoritmov kombinatorične narave.

Tisto, kar tako močno razlikuje Prolog od ostalih pomembnejših jezikov je, da je Prolog nepostopkovni ali deklarativni jezik, za razliko od drugih, ki so vsi v bistvu postopkovni (ali proceduralni). V "konvencionalnih" jezikih zato opisujemo postopke, torej kako pridemo od danih podatkov do rezultatov. Ideja Prologa pa je, da opišemo samo, kakšna je relacija med podatki in rezultati. Prologov prevajalnik pa mora sam poiskati postopek operacij, ki prevede podatke v rezultate tako, da le-ti ustrezajo zahtevani relaciji.

To razliko med postopkovnimi in nepostopkovnimi jeziki opisujeta naslova dveh znamenitih publikacij. Prva (Wirth 1975) se nanaša na postopkovne jezike (Pascal), druga pa na nepostopkovne (Kowalski 1979):

Wirth: "Programs = Algorithms + Data Structures"

Kowalski: "Algorithm = Logic + Control".

Logika pove, kaj naj program naredi, medtem ko (v Prologu skromna) kontrolna komponenta določa, kako bo interpreter nalogo dejansko izvedel.

Za Prolog je značilno, da programa ne definiramo z zaporedjem operacij (tako kot v običajnih, postopkovnih jezikih), temveč z množico relacij oz. predikatov. Tako je npr. v Pascalu značilna operacija prireditveni stavek, npr.:

```
Y:= f(X)
```

Ta stavek se izvrši tako, da vzamemo vrednost X, izračunamo vrednost funkcije f(X) in to vrednost priredimo spremenljivki Y. Torej lahko gornji stavek ponazorimo z diagramom

kjer je X vhod in Y izhod.

Ekvivalentno definicijo bi v Prologu napisali z izrazom:

f(X,Y)

ki bi ga lahko prebrali kot : X in Y sta v relaciji f. Taka izjava deluje med izvajanjem programa takole: če je znan X, potem se izračuna Y tako, da je ta Y v relaciji f z X. Če pa je znan Y, potem se izračuna X tako, da je spet Y v relaciji f z X. Taka izjava f(X,Y) deluje v obeh smereh: enkrat je vhod X in izhod Y, drugič pa obratno:

Obstajajo pa še druge možnosti, npr.:

(1) Znana sta oba, X in Y; potem se izjava f(X,Y) obnaša kot neke vrste kretnica, ki testira, ali sta dana X in Y v relaciji s f.

(2) Noben od X in Y ni znan; v tem primeru se program obnaša tako, kot da bi Prolog priredil spremenljivkama X in Y vse možne pare vrednosti, ki so v relaciji f.

Drug primer značilnega stavka v konvencionalnem jeziku je npr.:

```
if N>0 then N := N-1;
```

kar lahko beremo kot: "Če je vrednost spremenljivke na lokaciji N večja od 0, potem zamenjaj vrednost na lokaciji N s staro vrednostjo minus 1".

Značilen primer stavka v Prologu pa je npr.:

```
potomec(X,Y):-otrok(X,Y).
```

kar beremo takole:

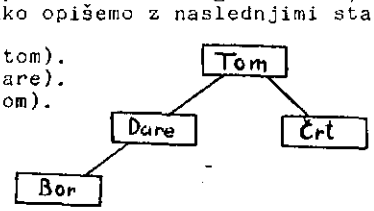
"X je eden izmed potomcev od Y, če je X eden izmed otrok od Y" ali pa

"Če je X otrok od Y, iz tega sledi, da je X potomec od Y".

Ta trditev velja za vsak X in Y. Operator :- deluje kot logična implikacija z desne v levo (v matematiki navadno znak ⟵).

Kot primer programa v Prologu vzemimo implementacijo nekaterih sorodstvenih relacij. Slika 1 kaže primer družinskega drevesa, ki ga v Prologu lahko opišemo z naslednjimi stavki:

```
otrok(dare,tom).
otrok(bor,dare).
otrok(črt,tom).
```



Slika 1. Primer družinskega drevesa.

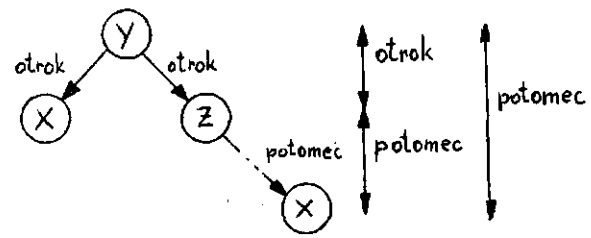
Te stavke lahko beremo npr. "Dare je otrok od Tom" ali pa "Dare in Tom sta v relaciji otrok".

Napišimo še stavke v Prologu, ki definirajo relacijo "potomec", na osnovi naslednjega razmišljanja (glej sliko 2).

(a) X je potomec od Y, če je X otrok od Y, ali
(b) X je potomec od Y, če eksistira Z, tak, da
    (1) Z je otrok od Y in
    (2) X je potomec od Z.

Ti dve izjavi opišemo v Prologu z:

```
potomec(X,Y) :- otrok(X,Y).
potomec(X,Y) :- otrok(Z,Y), potomec(X,Z).
```



Slika 2. Definicija relacije "potomec".

Zdaj lahko postavimo Prologovemu sistemu že nekaj vprašanj, npr: "Kdo je otrok od Toma in kdo je Borov oče?" To storimo z naslednjim stavkom v Prologu:

```
?-otrok(X,tom),otrok(bor,Y).
```

Odgovor, ki ga dobimo, je:

```
X=dare, Y=dare;
X=črt, Y=dare
```

Kot vidimo, sta bila na vprašanje možna dva odgovora (ker ima Tom dva otroka). Prologov interpreter je poiskal oba.

Vprašajmo še, kdo so Tomovi potomci:

```
?- potomec(X,tom).

X=dare;
X=črt;
X=bor;
```

Ta primer ilustrira način programiranja v Prologu, ki je močno različen od programiranja v konvencionalnih jezikih. V primerjavi s temi jeziki v Prologu ne eksistirajo konstrukti, kot so:

- krmilni konstrukti, npr. goto, while, repeat;
- prirejanje vrednosti spremenljivkam in spreminjanje vrednosti spremenljivk;
- dznake stavkov, globalna imena spremenljivk.

Nekatere implementacije Prologa so:

- interpreter v fortranu (napisan na univerzi v Marseillu);
- interpreter v assemblerju za DEC-10 (univerza v Edinburghu);
- kompilator, napisan v Prologu in deloma v assemblerju za DEC-10 (Edinburgh);
- interpreter v assemblerju za PDP 11/34 (Edin.)
- interpreter v Lispu za CDC CYBER (Linköping).

2. NEKATERI ELEMENTI JEZIKA IN ENOSTAVNI STAVKI

Programer v Prologu opiše svoj problem na deklarativen način, medtem ko mora Prologov sistem pri izvajanju programa vendarle izvršiti določen postopek, torej ga mora interpretatirati "postopkovno"; zato ločimo dva pomena (oz. semantiki) programov v Prologu: deklarativni in postopkovni pomen programa.

Tako npr. stavek v Prologu

```
P :- Q, R.
```

(ki je ekvivalenten zapisu v matematični logiki $P \Leftarrow Q \wedge R$) lahko beremo na naslednje načine:

(1) Deklarativno:
"Q in R implicira P" ali
"Iz Q in R sledi P"

(2) Postopkovno:
"Za to, da rešimo P moramo rešiti Q in R"
ali "En način za izvršitev procedure P je:
pokliči proceduro Q in zatem proceduro R."

Program v Prologu lahko zato razumemo kot množico logičnih trditev, ki jih mora Prologov sistem ovreči ali pa dokazati njihovo resničnost. Zato je izvajanje programa v Prologu dejansko dokazovanje trditev, ki so postavljene v programu. S tega stališča je Prologov interpreter avtomatski dokazovalnik izrekov. Stavki v Prologu imajo obliko

```
P :- Q, R, S.
```

glava stavka telo stavka

Ta stavek pomeni: "P je res, če so Q in R in S res". Q, R in S se imenujejo cilji. Število ciljev v stavku je poljubno, lahko tudi enako nič. Stavek brez ciljev je enotin stavek, npr.

```
P.
```

kar pomeni: "P je vedno res".

Z enostavnimi stavki opisujemo enostavna dejstva, npr.

```
otrok(dare,tom).
```

Vprašalni stavki se začenjajo z vprašajem,

```
?-P, Q.
```

To pomeni "ali sta P in Q resnična" oz. "reši P in Q".

Elementarni podatkovni objekti so:

(1)"atomi", npr.:
david, a, nil, tom, dare

(2)cela števila, npr.:
5, 0, -973

(3)spremenljivke (se ločijo od atomov po veliki začetnici) npr.:
X,Y, B23, List, Pilot

Spremenljivke lahko dobijo vrednost med izvajanjem programa. Sestavljeni podatkovni objekti so izrazi. Primer izraza, ki opisuje točko v tridimenzionalnem prostoru, je

```
točka(X,Y,Z)
```

funktor argumenti

Z uporabo funktorjev lahko gradimo izraze, v katerih so vgnezdeni podizrazi. Na ta način lahko strukturiramo naše podatkovne objekte. Podatkovno strukturo, ki jo imenujemo seznam, (znano predvsem iz programskega jezika Lisp) lahko npr. zgradimo z uporabo funktorja "." (pike). Seznam je zaporedje podatkov, ki ga lahko definiramo takole: seznam je bodisi (1)prazno zaporedje, ki ga po dogovoru navadno označimo z nil (no-item-list), bodisi (2) ima prvi element (imenovan glava) in pa preostanek (imenovan rep); pri tem je glava lahko karkoli, rep pa mora biti seznam (lahko tudi prazen). Primer seznama imen je npr.:

```
[ana,teja,maja]
```

Glava tega seznama je "ana", rep pa je "teja, maja".

V Prologu smemo uporabljati za sezname prav tako notacijo, kot smo jo uporabili v gornjem primeru. Vendar je to le sintaksna nadgradnja Prologove predstavitve seznamov s funktorjem ".". Ta funktor ima dva argumenta, od katerih je prvi glava seznama, drugi pa rep seznama. Tako je

```
[ana,teja,maja]
```

ekvivalentno zapisu

```
.(ana, [teja,maja] )
```

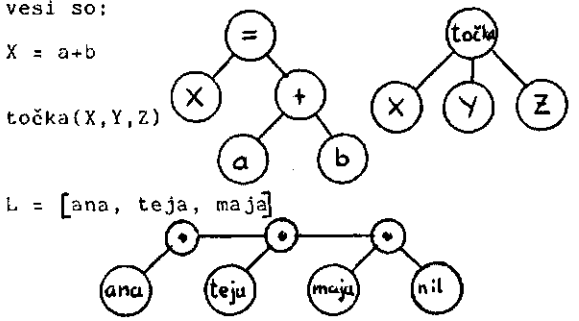
kar je ekvivalentno

```
.(ana,.(teja, [maja] ))
```

oz. naprej

```
.(ana,.(teja,.(maja,nil)))
```

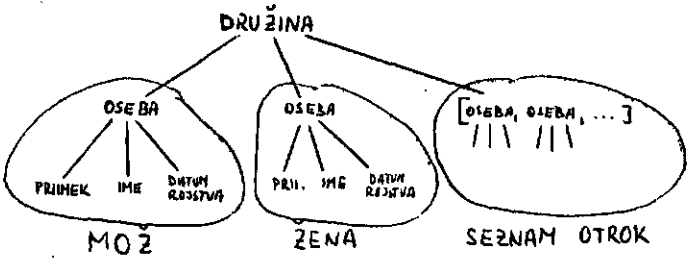
Nazorno lahko ponazarjamo izraze grafično z uporabo dreves. V takih drevesih so v notranjih vozliščih funktorji, v zunanjih pa elementarni podatki. Primeri ponazarjanja z drevesi so:



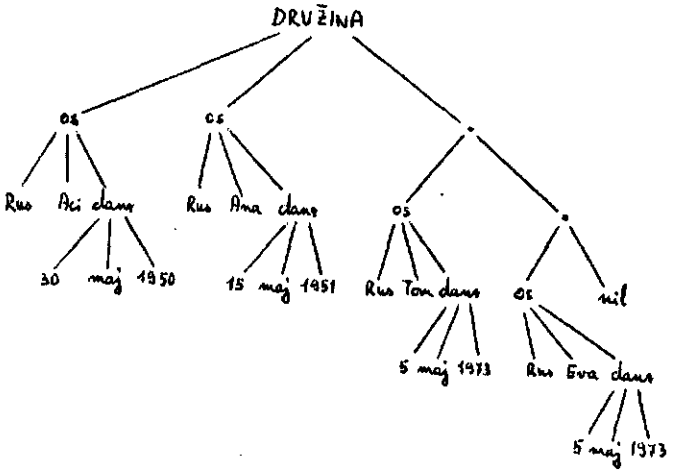
L = [ana, teja, maja]

Če pišemo Rep = [teja,maja] , potem je koristna uporaba operatorja "|", ki povezuje glavo in rep:

L = [ana | Rep]



Primer:



Slika 3. Strukturiranje podatkov o družini.

Dejstvo, da eksistira družina na sliki 3, lahko opišemo z naslednjim enotinim stavkom v Prologu:

```
druž
   (os (rus, aci, danr(30, maj, 1950)),
    os (rus, ana, danr(15, maj, 1951)),
    os (rus, tom, danr( 5, maj, 1973)),
    os (rus, eva, danr( 5, maj, 1973))).
```

Pri tem smo uvedli funktorje: druž(družina), os(oseba), danr(dan rojstva).

Leksikalni doseg spremenljivk v stavkih Prologa je omejen na stavek sam. Tako lahko v dveh stavkih nastopa ime spremenljivke X, vendar to ni ena sama spremenljivka, temveč dve povsem različni spremenljivki. Poglejmo nekaj primerov, ki ilustrirajo to pravilo:

1. Trditev "Vsakdo zaupa samemu sebi" napišemo lahko z naslednjim stavkom:

```
zaupa(X,X).
```

2. Trditev "Vsak moški X je poročen, če eksistira družina, v kateri je X mož":

```
poročen(X) :- druž(X, _, _).
```

Črtice označujejo "slepe" spremenljivke.

3. Trditev:"Katerikoli osebi A in B sta poročeni med seboj, če eksistira družina, v kateri je A mož in B žena", ali pa "če eksistira družina, kjer je B mož in A žena. " To trditev opisujeta naslednja stavka:

```
zakonca(A,B) :- druž(A,B,_).
zakonca(A,B) :- druž(B,A,_).
```

Ta stavek lahko zapišemo krajše:

```
zakonca(A,B):- druž(A,B,_); druž(B,A,_).
```

Podpičje označuje disjunkcijo med dvema ciljema.

3. DEKLARATIVNA IN POSTOPKOVNA SEMANTIKA TER IZVAJANJE PROGRAMOV

Kot smo že omenili, lahko programe v Prologu gledamo na dva načina: prvič deklarativno in drugič postopkovno. Ker imajo programi v Prologu deklarativni in postopkovni pomen, eksistirata tudi dve semantiki Prologa. Deklarativni pomen Prologovega programa govori o relacijah med podatki oz. strukturami. Torej v nekem smislu opisuje zakonitosti, ki se jim podrejajo objekti, ki nastopajo v programu, neodvisno od samega postopka. Ta nepostopkovni, deklarativni vidik Prologovih programov dviga Prolog izredno visoko v hierarhiji programskih jezikov.

Vendar pa mora Prologov interpreter priti do iskanih rezultatov vseeno po nekem postopku, zato je potrebna (kot neke vrste nujno zlo) tudi postopkovna semantika, ki natančno predpisuje, po kakšnem postopku se program izvaja. S stališča programiranja pa je nadvse pomembno to, da lahko programer v načelu razmišlja na en ali drug način, saj se programerju ni treba spuščati v detalje samega postopka. Izkaže se, da je pogosto mnogo lažje rešiti problem na deklarativen način. Deklarativna semantika Prologa definira, kdaj je kaka izjava v Prologovem programu resnična, tj. da je izjava dejstvo: "Dani cilj je dejstvo, če je ta cilj "primer" glave kakega stavka in so pri tem vsi cilji (če eksistirajo) v telesu tega stavka tudi dejstva. Primer stavka (ali izraza) dobimo s substitucijo vsake od njegovih spremenljivk z nekim drugim izrazom na vseh mestih, kjer se spremenljivka pojavlja."

Postopkovna semantika Prologa pa je dana z naslednjimi pravili:

1. Za izpolnitev cilja je treba med seboj "prilagoditi" ta cilj in glavo kakega stavka in zatem izpeljati vse cilje v telesu tega stavka ("prilagajanje" izrazov je pojasnjeno kasneje).
2. Cilje v telesu izvršuj od leve proti desni.
3. Cilj poskušaj prilagoditi glavam stavkov po vrsti od zgoraj navzdol.
4. Kadar ni mogoče (več) prilagoditi cilja nobeni glavi, se "vrni nazaj" in poskušaj izpolniti prejšnji cilj na kakšen drug način (angl. backtrack).
5. Če niti prilagajanje niti vračanje ni več mogoče, potem cilja ni mogoče izpolniti.

Prilagajanje dveh izrazov je proces, ki spremenljivkam v obeh izrazih priredi take vrednosti, da postaneta po tej substituciji spremenljivk oba izraza identična. Prilagoditev je torej množica prireditev vrednosti spremenljivkam tako, da se oba izraza izenačita. Npr. izraza

```
točka(X,Y,3) in
točka(2,Y1,Z)
```

se prilagodita takole: X=2, Y=Y1, Z=3.

Pri izvajanju Prologovega programa iščemo vedno najbolj splošno prilagoditev. Najsplošnejša prilagoditev je tista prilagoditev, ki izenači oba izraza in pri tem čimmanj natančno specificira vrednosti spremenljivk. V gornjem primeru je možna tudi naslednja, manj splošna prilagoditev :

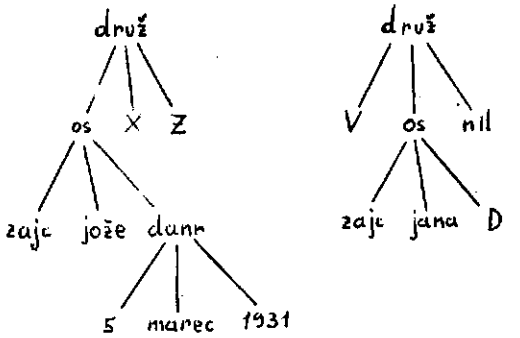
```
X=2, Y=5, Y1=5, Z=3.
```

Ta izenači oba izraza in rezultat prilagajanja je točka(2,5,3). Vendar pa ni najsplošnejša, ker sta vrednosti spremenljivk Y in Y1 po nepotrebnem povsem specificirani. Za prilagoditev zadošča namreč že omejitev Y=Y1.

Slika 4 kaže še en primer prilagajanja.

1. izraz
   druž(os(zajc,jože,danr(5,marec,1931)),X,Z)

2. izraz
   druž(V,os(zajc,jana,D),nil)



Najsplošnejša prilagoditev:
V= os(zajc,jože,danr(5,marec,1931))
X= os(zajc,jana,D)
Z= nil

Slika 4. Primer prilagajanja izrazov.

Izkaže se, da je prilagajanje izrazov izredno močno programirno orodje in da lahko že s samim prilagajanjem rešujemo probleme. Naslednji program npr. izračuna s samim prilagajanjem, katere daljice so hkrati horizontalne in vertikalne. Slika 5 ilustrira način, ki smo ga izbrali za popis nekaterih pojmov iz ravninske geometrije.

Točka je dana z dvema koordinatama, npr. t(X,Y), daljica pa z dvema točkama: daljica(T1, T2). Naslednji program definira, kdaj je kaka daljica vertikalna oz. horizontalna:

```
vertikalna(daljica(t(X,Y1),t(X,Y2))).
horizontalna(daljica(t(X1,Y),t(X2,Y))).
```

Ta program že zadošča, da lahko zastavimo nekaj vprašanj v obliki ciljev za Prologov

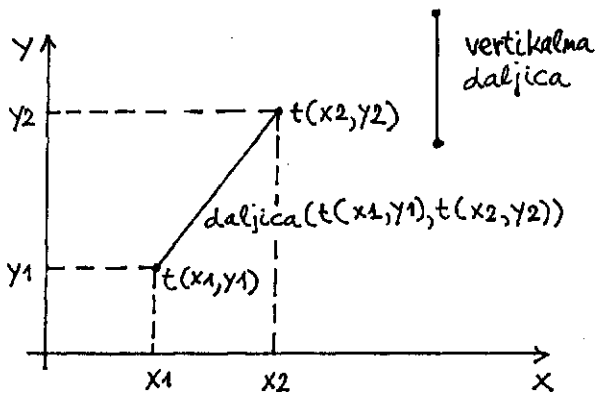
interpreter (odgovori računalnika so podčrtani, komentarji pa v oklepajih):

```
?-vertikalna(daljica(t(0,1),t(0,2))).
Yes; (odgovor je da)
?-vertikalna(daljica(t(1,2),t(2,3)).
No; (odgovor je: ta daljica ni vertikalna).
```

Vprašajmo še, ali eksistira daljica, ki je hkrati vertikalna in horizontalna:

```
?-vertikalna(D),horizontalna(D).
D=daljica(t(X,Y),t(X,Y))
```

Odgovor pomeni: da, to je vsaka daljica, ki je degenerirana v eno samo točko.



Slika 5: Opis nekaterih geometrijskih pojmov v Prologu.

Naslednji primeri bodo ilustrirali postopkovni pomen Prologovih programov.

Primer 1: relacija"ded-vnuk"

To relacijo lahko opišemo deklarativno takole: X je ded od Y, če je X roditelj od Z in obenem Z roditelj od Y, ter X je moški. To, skupaj s še nekaterimi enostavnimi trditvami, zapišemo v Prologu takole:

```
ded(X,Y) :- roditelj(X,Z),
            roditelj(Z,Y),
            moški(X).
moški(tone).
moški(peter).
ženska(ana).
ženska(eva).
moški(rok).
roditelj(ana,eva).
roditelj(tone,eva).
roditelj(eva,rok).
roditelj(peter,rok).
```

Zastavimo vprašanje : Kdo je Rokov ded?

```
?-ded(X,rok).
```

Zasledujmo izvajanje programa, to je postopkovni pomen programa:
cilj je ded(X,rok)
Ta cilj se prilagodi z glavo stavka

```
ded(X,Y):-....
```

Pri tem postane Y=rok, generirajo pa se podcilji (v telesu stavka, s katerim se je prilagodil tekoči cilj):

```
podcilj 1: roditelj(X,Z)
podcilj 2: roditelj(Z,rok)
podcilj 3: moški(X)
```

Podcilj 1 lahko izpolnimo tako, da ga prilagodimo prvemu izmed stavkov "roditelj(...)". Tako dobimo prilagoditev
X=ana, Z=eva
Po tej prilagoditvi je podcilj 1 izpolnjen (ker v telesu stavka "roditelj..." ni bilo več nobenega cilja). Zato je na vrsti podcilj 2, ki je po prilagoditvi postal
roditelj(eva,rok)
Ta podcilj je trivialen, saj je že shranjen kot dejstvo v našem programu. Naslednji, tj. tretji podcilj, je
moški(ana)
Ta podcilj se ne prilega glavi nobenega stavka, zato ga ni mogoče izpolniti. Zato se izvajanje programa vrne nazaj in skuša prejšnje podcilje izpolniti na drug način. Podcilja 2 ni mogoče izpolniti na noben drug način, zato pa je to mogoče s podciljem 1:
roditelj(X,Z)
in sicer takole:
X=tone, Z=eva
Podcilj postane zdaj roditelj(eva,rok) .
To je že enotin stavek v programu. Preostane še cilj moški(tone), ki je prav tako trivialno izpolnjen. S tem so vsi podcilji izpolnjeni. Izpiše se rešitev, ki je definirana s prilagoditvami:

X=tone

Primer 2: konkatenacija seznamov

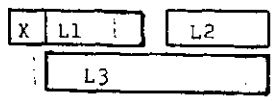
Napišimo proceduro za konkatenacijo (spajanje) seznamov, tako da bo predikat

conc(L1,L2,L3) izpolnjen,

če je seznam L3 konkatenacija seznamov L1 in L2, npr.:

conc([a,b] , [c,d,e] , [a,b,c,d,e])

Idejo za to proceduro lahko izrazimo deklarativno:
(1)Konkatenacija kateregakoli seznama L s praznim seznamom je seznam L.
(2)Če konkateniramo seznam oblike [X|L1] (glava je X, rep pa L1) s seznamom L2, dobimo seznam, katerega glava je X, rep pa konkatenacija seznamov L1 in L2 :



Ti dve trditvi izrazimo v Prologu takole:

```
conc([],L,L)
conc( [X|L1] ,L2, [X|L3] ) :- conc(L1,L2,L3).
```

Proceduro conc lahko uporabljamo na razne načine. Npr. tako, da podamo vrednosti prvih dveh argumentov in kot rezultat dobimo njeno konkatenacijo. Bolj zanimiv primer uporabe je, če je podan tretji argument, kot rezultat pa dobimo prva dva seznama, tako da je njuna konkatenacija enaka tretjemu. Če je možno tretji (dani) seznam sestaviti iz dveh seznamov na več načinov, potem dobimo kot rezultat vse možne rešitve:

```
?-conc([a,b], [1,2,3],X).
X=[a,b,1,2,3]
```

```
?-conc(L1,L2,[a,b,c]).
L1=[],L2=[a,b,c];
L1=[a],L2=[b,c];
L1=[a,b],L2=[c];
L1=[a,b,c],L2=[]
```

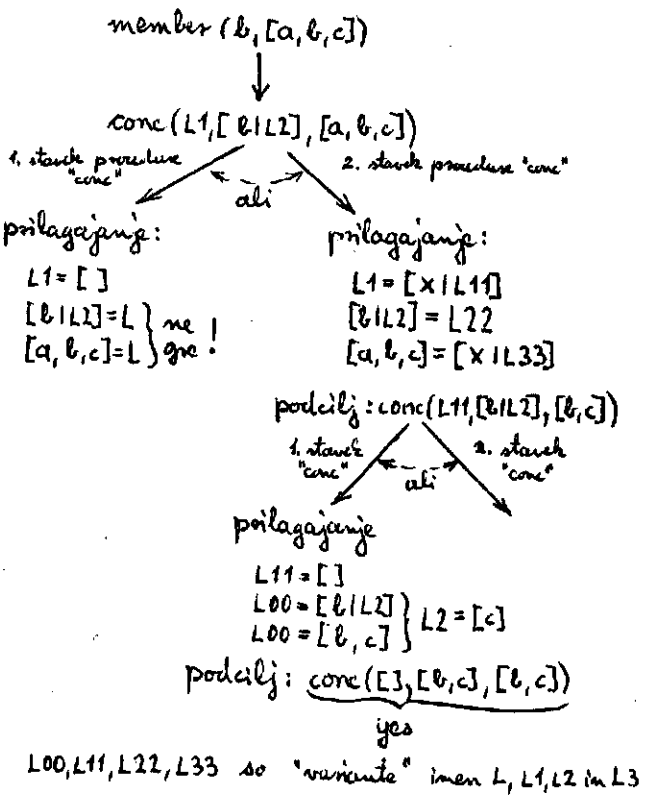
Ta zgled ilustrira nedeterminizem v Prologu: če je možnih več rešitev, Prologov interpreter generira vse, razen če tega na poseben način ne prepovemo. Definirajmo še relacijo

```
member(X,L)
```

ki je izpolnjena, če je X element seznama L. To relacijo lahko na nepostopkoven način definiramo takole: X je element seznama L, če eksistirata dva podseznama seznama L tako, da je glava drugega podseznama enaka X:

```
member(X,L) :- conc(L1,[X|L2],L).
```

Kako deluje ta presenetljiva rešitev, kaže slika 6.



Slika 6. Izvajanje procedure member ob vprašanju:
?- member(b,[a,b,c]).

Včasih že vnaprej vemo, da je generiranje alternativnih rešitev nepotrebno oz. nesmiselno. V takih primerih bi Prologov program po nepotrebnem trošil čas in prostor, zato imamo na voljo dodatni krmilni element, ki prepreči iskanje alternativnih rešitev. Ta krmilni element je t.i. "cut operator", ki ga označujemo s klicajem. Za primer definirajmo relacijo

```
max(A,B,M)
```

tako, da je M vrednost večjega od A in B. To lahko storimo z:

```
max(X,X,X).
max(X,Y,Y) :- X<Y.
max(X,Y,X) :- Y<X.
```

Kako se obnaša ta procedura, če zastavimo cilj:

```
?-max(2,3,M),M<2.
```

Prvi podcilj uspe tako, da M postane 3. Drugi podcilj postane 3<2 in ne uspe. Zato se izvajanje vrne na prvi podcilj in ga poskusi zadovoljiti še na kak drug način, torej poskusi z:

```
max(2,3,M) :- 3<2.
```

Seveda tudi to ne uspe in s tem ne uspe tudi celotni cilj. Iz naravne relacije max vemo, da lahko uspe kvečjemu eden izmed stavkov procedure max. Ko je uspel v gornjem poizkusu drugi stavek te procedure, je jasno, da tretji stavek ne more več uspeti. Zato je vračanje na ta stavek brezplodno. To vračanje lahko preprečimo z naslednjo dopolnitvijo procedure max:

```
max(X,X,X) :-!.
max(X,Y,Y) :- X<Y, !.
max(X,Y,X) :- Y<X.
```

Natančen pomen operatorja "!", imenovanega "cut", je: "!" se obnaša kot cilj, ki vedno uspe, pri tem pa povzroči kot stranski učinek, da se prepreči vračanje na prejšnje podcilje pred klicajem.

Naslednji primer nadalje pojasnjuje obnašanje operatorja "cut".

```
P :- Q1,Q2.
P :- R.
```

Ta procedura definira naslednjo logično zvezo med izjavami P,Q1,Q2 in R:

$$P \Leftrightarrow Q1 \wedge Q2 \vee R$$

Zdaj postavimo "cut" takole:

```
P :- Q1,!,Q2.
P :- R.
```

Logična zveza med izjavami se spremeni v:

$$P \Leftrightarrow Q1 \wedge Q2 \vee \neg Q1 \wedge R$$

4. ZAKLJUČEK

V sestavku smo pokazali nekatere značilnosti Prologa, predvsem tiste, zaradi katerih je način razmišljanja pri programiranju v Prologu drugačen kot pri programiranju v postopkovnih jezikih. Sintaksa predikatnega računa, na kateri temelji Prolog, je zelo posrečena za opis relacij med podatki. Po eni strani ta sintaksa omogoča delo s podatkovnimi strukturami, ne da bi pri tem eksplicitno uporabljali kazalce. Vsak sestavljeni podatkovni objekt v Prologu lahko ponazorimo z drevesom. Po drugi strani pa lahko to sintakso uporabljamo direktno kot jezik za postavljanje vprašanj (query language) za relacijski model podatkovnih baz. Že sam mehanizem prilagajanja zadošča za dokaj zahtevno iskanje relacijskih odnosov v bazi podatkov (to je v množici trditev, zapisanih v programu v Prologu).

Opisani primeri so povečini nakazali možnosti strukturiranja in iskanja podatkov. Nekatere druge operacije, ki zahtevajo več procesiranja na seznamskih, drevesnih in grafskih strukturah, so opisane v (Bratko, Gams 1981). Tam je poudarek na primerjavi Prologa z običajnimi jeziki, zato so te operacije implementirane še v Pascalu kot predstavniku konvencionalnih jezikov.

LITERATURA

1. Bratko I., Gams M. (1981, v pripravi) Prolog: procesiranje podatkovnih struktur in primerjava s Pascalom. Ljubljana: Informatika.
2. Kowalski R. (1974) Predicate logic as a programming language. Proc. IFIP Congress 74. North-Holland.
3. Kowalski R. (1979) Algorithm = Logic + Control. CACM, Vol. 22, No. 7, 572-595.
4. Pereira L., Pereira F., Warren D.H.D. (1978) User's Guide to DEC System 10 Prolog. University of Edinburgh, Department of Artificial Intelligence.
5. Roussel P. (1975) Prolog: manual d'utilisation. Raport interne GIAVER de Luminy, Universite d'Aix, Marseille.
6. Sammet J. (1969) Programming Languages: History and Fundamentals. Prentice-Hall.
7. Wirth N. (1976) Programs = Algorithms + Data Structures. Prentice-Hall.
8. Siklossy L. (1976) Let's Talk Lisp. Prentice-Hall.

MEHURČNI POMNILNIKI – II. DEL

J. ŠILC, B. MIHOVILOVIĆ, P. KOLBEZEN

UDK:681.327.664.4

INSTITUT „JOŽEF STEFAN", LJUBLJANA

Članek nas seznanja z nekaterimi osnovnimi fizikalnimi značilnostmi magnetnih mehurčkov. Najprej spregovorimo o pojavu in pogojih stabilnosti magnetnega mehurčka. V nadaljevanju pa govorimo o dinamiki magnetnega mehurčka (možnostih generiranja, širjenja, zaznavanja in brisanja), ki zagotavlja, da postane magnetni mehurček nosilec osnovne enobitne informacije. Nenazadnje je podanih nekaj osnovnih lastnosti in parametrov snovi (dolžina snovi Λ, kvaliteta Q) v katerih se pojavljajo magnetni mehurčki.

MAGNETIC BUBBLE MEMORIES - PART 2. In this paper some basic physical properties of magnetic bubbles are represented. First, we talk about an appearance and stability conditions of bubble. Next, the operation of bubble devices such as propagation, generation, detection and annihilation is described. This properties enable bits of information are represented by magnetic bubble. Finally, some elementary quality and material parameters (such as material lenght Λ and material quality Q) are given.

1. UVOD

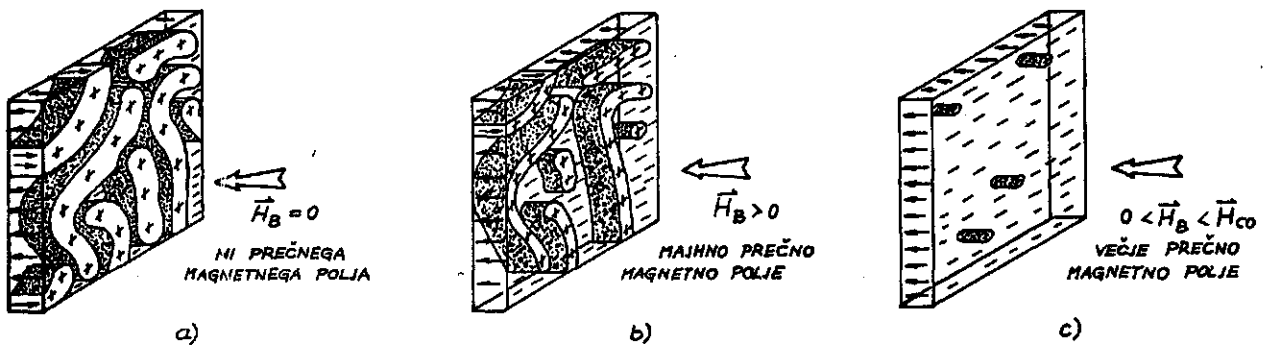
Ko je A. H. Bobeck leta 1967 [1] uspel v nekaterih ortoferitih (npr. $TmFeO_3$) tvoriti stabilne izolirane cilindrične domene relativno majhnega premera (6 μm) in jih kontrolirano premikati, so se odprle možnosti razvoja novih nezbrisljivih masovnih pomnilnikov visokih gostot ($>10^6$ bitov/cm^2), ki pomnijo informacijo s pomočjo teh cilindričnih domen (magnetnih mehurčkov). V težnji za povečanjem gostote so nadaljne raziskave potekale v iskanju novih snovi, v katerih bi bilo mogoče ustvariti stabilne cilindrične domene čim manjših premerov, tako, da se danes uporabljajo v proizvodnji mehurčnih pomnilnikov *enoosni garneti redkih zemelj, v katerih je mogoče ustvariti cilindrične domene premera 0,5 do 2 μm.* Te snovi karakterizirata dva zelo pomembna parametra. Prvi je t.i. karakteristična dolžina snovi Λ en. (12) in drugi je faktor kvalitete Q en. (15). Premer cilindrične domene je premosorazmeren karakteristični dolžini snovi Λ in obratno sorazmeren kvaliteti Q, zato naj bo Λ čim manjši in Q večji od ena.

2. KAKO NASTANE MAGNETNI MEHURČEK?

V nekaterih snoveh so mikrokristalna območja (takoimenovane magnetne domene), v katerih so medatomske sile dovolj močne, da so atomski magnetki[1] kljub termičnemu gibanju atomov urejeni v isto smer, četudi ni zunanjega magnetnega polja. Vsaka domena deluje kot nekakšen magnet, ki je sestavljen iz velikega števila (10^{15} do 10^{20}) majhnih, enako usmerjenih magnetov. Magnetno polje posameznih domen je izredno močno; gostota polja je okrog 1 T. Če se snov se ne nahaja v zunanjem magnetnem polju, so magnetni momenti posameznih domen povsem neurejeno usmerjeni, tako, da je snov navzven nemagnetna.

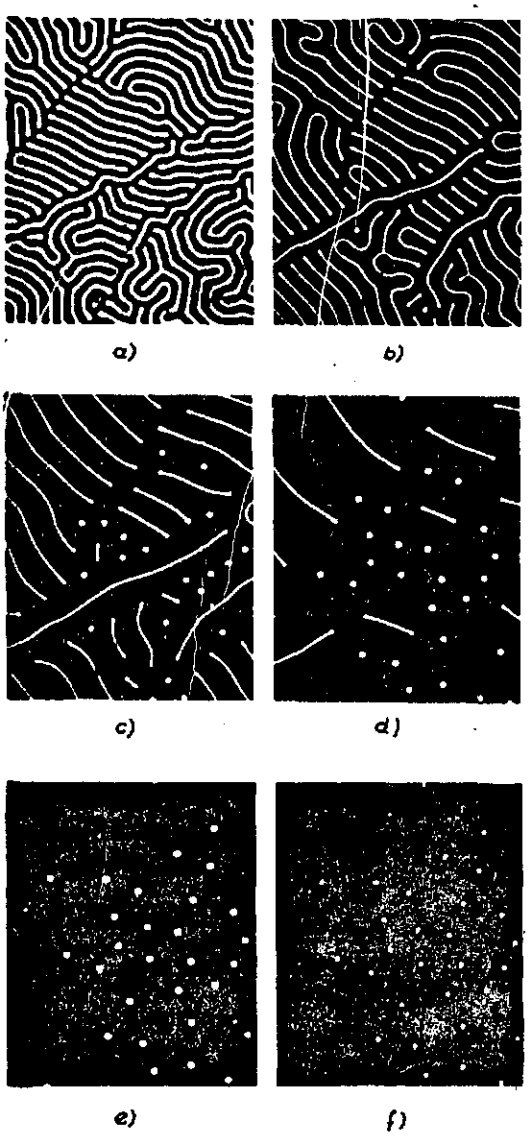
Če iz kristala določene enoosno anizotropne antiferomagnetne (ortoferiti redkih zemelj) ali feromagnetne (enoosni garneti redkih zemelj) snovi izrezemo del kristala v obliki tankega filma tako, da je lažja os magnetenja pravokotna na lice filma, opazimo v filmu valovite proge, ki so alternirajoče magnetno polarizirane, bodisi v smeri lažje osi magnetenja ali v obratni smeri (slika 1a). Te vzorce je mogoče s pomočjo Faradayevega efekta (v polarizirani svetlobi) opazovati, kot temne oziroma svetle proge (slika 2a). V primeru, ko ni prisotno prečno magnetno polje (slika 1a, slika 2a), je snov navzven nemag-

1) Vsak elektron v atomu ima lastno vrtilno količino, ki ni odvisna od gibanja elektrona in se imenuje spin. Ker se vrti tudi negativni naboj elektrona, sklepamo, da so v notranjosti elektrona tokovne zanke, torej ima elektron takoimenovani spinski magnetni moment. Pri nekaterih snoveh se spini elektronov v atomu med seboj ne kompenzirajo; atomi teh snovi imajo določen lastni magnetni moment in se obnašajo kot majhni magnetki.



slika 1.

netna, saj se magnetni momenti valovitih prog med seboj kompenzirajo. Pod vplivom prečnega magnetnega polja jakosti $\vec{H}_B$, ki deluje pravokotno na lice filma, se začno področja, ki so obratno magnetno polarizirana, zoževati, področja, ki pa imajo enako smer magnetenja kot prečno magnetno polje, pa se širijo (slika 1b, slika 2b). Ta proces se kontinuirano nadaljuje, ko večamo jakost prečnega magnetnega polja (slika 2c, d). Pri dovolj veliki jakosti prečnega polja, ostanejo v filmu samo še osamljene cilindrične domene, katerih vektor saturacijske magnetizacije $\vec{M}_s$ je obratno usmerjen od prečnega polja in se imenujejo magnetni mehurčki (slika 1c, slika 2e, f). Če prečno polje še povečujemo mehurčki izginejo (to se zgodi pri prečnem magnetnem polju jakosti $\vec{H}_{co}$), tako da dobimo enotno magnetno polariziran film (nasičenje), ki ga moramo razmagnetiti[2], če želimo ponovno tvoriti magnetne mehurčke.



Slika 2.

Na sliki 2 so s pomočjo Faradayevega efekta prikazane magnetne domene v 6 μm debelem epitaksialnem garnetnem filmu $(YGdTm)_3Fe_5O_{12}$. Na sliki 2a ni prečnega magnetnega polja, ki ga nato povečujemo od 6,4 kA/m (slika 2b), 8 kA/m (slika 2c), 8,8 kA/m (slika 2d), 9,2 kA/m (slika 2e) do 10,3 kA/m (slika 2f). Če zunanje magnetno polje povečujemo preko 10,3 kA/m ($\vec{H}_{co}$) mehurčki izginejo [3].

3. MODEL CILINDRIČNE DOMENE

Statika. Celotna energija cilindrične domene je vsota treh sumandov in sicer: energije stene cilindrične domene W_w, energije vsled prečnega magnetnega polja W_H in notranje magnetostatične energije W_M.

$$W = W_w + W_H + W_M \tag{1}$$

Energija stene cilindrične domene je enaka [3]

$$W_w = 2\pi r h \sigma_w \tag{2}$$

kjer je σ_w energijska gostota stene cilindrične domene v J/m^2, h je debelina filma in 2r premer cilindrične domene.

2) Snov je možno razmagnetiti na dva načina in sicer tako, da jo segrejemo preko Néelove temperature [3] in ohladimo (zunanje magnetno polje je nič) ali pa jo damo v nasprotno polarizirano zunanje magnetno polje ($> H_{co}$).

Energija vsled prečnega magnetnega polja jakosti $\vec{H}_B$ je enaka [3]

$$W_H = 2\pi\mu_0 r^2 h H_B M_S \tag{3}$$

kjer je M_S saturacijska magnetizacija v A/m in μ_0 permeabilnost praznega prostora ($4\pi \cdot 10^{-7}$ Vs/Am). Podrobnejši izračun notranje magnetostatične energije je podan v [3] in se glasi

$$W_M = -\pi\mu_0 h^3 M_S I(a) \tag{4}$$

pri tem je $a = 2r/h$ in $I(a) = \int_0^a F(a)\,da$ ter

$F(a) = \frac{2}{\pi} a^2 \left[\frac{\sqrt{1+a^2}}{a} E\left(\frac{a}{1+a^2}\right) - 1\right]$. Tu je $E(k)$ eliptični integral prve vrste [3]. Torej lahko pišemo celotno energijo cilindrične domene v obliki

$$W = 2\pi r h \sigma_W + 2\pi\mu_0 r^2 h H_B M_S - W_M \tag{5}$$

Cilindrična domena je stabilna pri minimalni energiji sistema, torej velja

$$\frac{\partial W}{\partial r} = 0 \tag{6}$$

in

$$\frac{\partial^2 W}{\partial r^2} > 0 \tag{7}$$

Z odvajanjem enačbe (5) na r in ob upoštevanju (6) dobimo

$$\frac{1}{4\pi\mu_0 r h M_S} \frac{\partial W_M}{\partial r} = \frac{\sigma_W}{2\mu_0 r M_S} + H_B \tag{8}$$

oziroma

$$H_M = H_{\sigma_W} + H_B \tag{9}$$

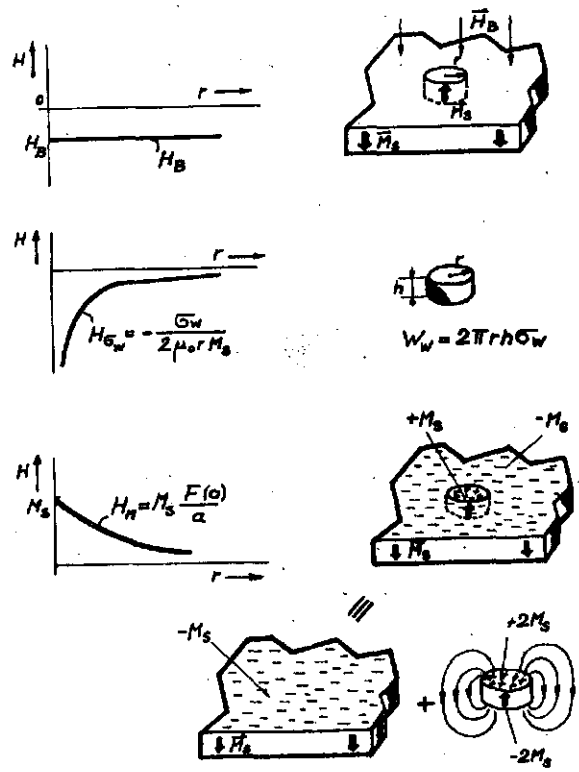
kjer je H_M povprečna jakost magnetostatičnega polja in H_{σ_W} efektivna magnetna poljska jakost, ki jo prispeve energija stene cilindrične domene (slika 3). Minimum bo dosežen le tedaj, ko bo poleg enačbe (9) izpolnjena tudi neenačba (7), ki jo lahko pišemo kot

$$\left|\frac{\partial H_M}{\partial r}\right| > \left|\frac{\partial H_{\sigma_W}}{\partial r}\right| \tag{10}$$

Grafična rešitev enačbe (9) je prikazana na sliki 4. Enačbi (9) zadoščata rešitvi r_a in r_b, vendar je cilindrična domena stabilna le pri premeru $2r_b$, saj rešitev r_a ne zadošča neenačbi (10). Enačbo (8) je mogoče ob upoštevanju (4) pisati tudi kot

$$\frac{\Lambda}{h} + a\frac{H_B}{M_S} - F(a) = 0 \tag{11}$$

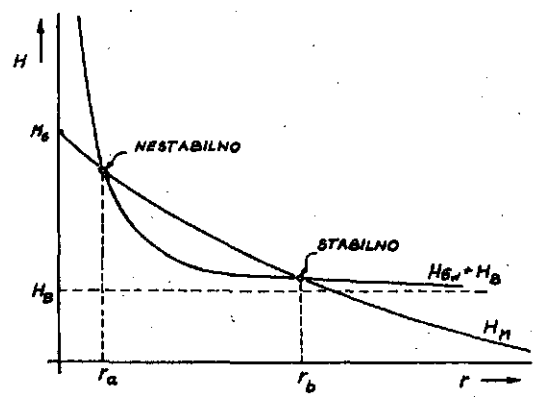
3) $E(k) = \int_0^{\pi/2} \sqrt{1 - k\sin^2\alpha}\; d\alpha$



Slika 3.

kjer je Λ značilen parameter snovi, tako imenovana karakteristična dolžina snovi (material length) in je definirana kot

$$\Lambda = \frac{\sigma_W}{\mu_0 M_S^2} \tag{12}$$

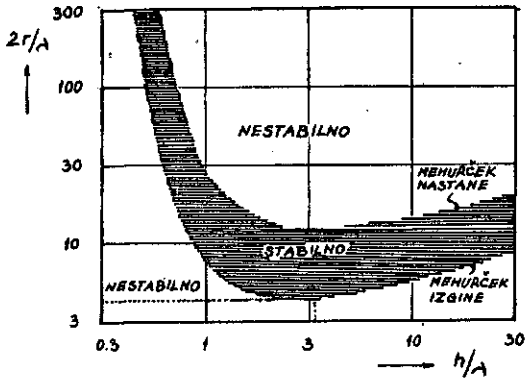


Slika 4.

Izraz (11) je potreben pogoj za nastop minimuma. Da bo minimum dosežen, mora veljati tudi (7), kar lahko pišemo kot

$$\frac{H_B}{M_S} - \frac{dF(a)}{da} > 0 \tag{13}$$

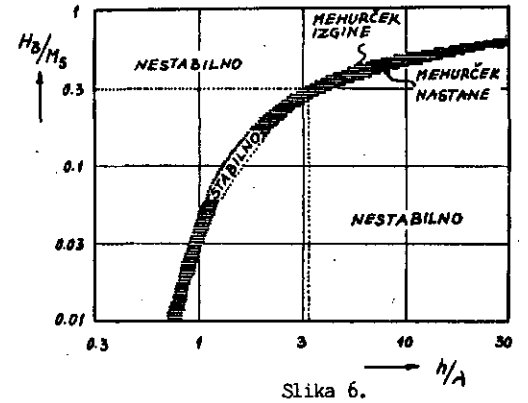
Enačbi (11) in (13) sta potreben in zadosten pogoj za nastop minimuma energije W, torej stabilne cilindrične domene. Na sliki 5 je prikazana odvisnost premera stabilne



Slika 5.

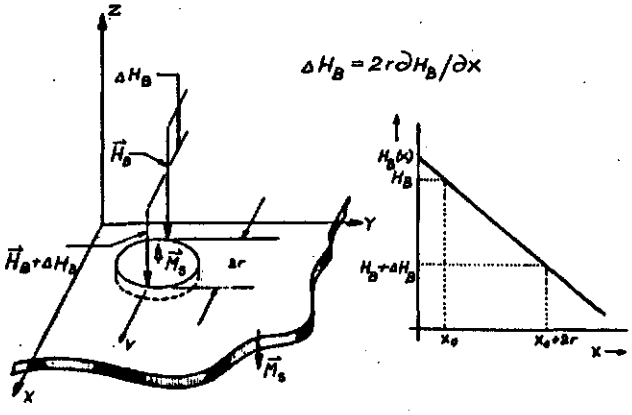
cilindrične domene od debeline filma h.

Vidimo, da je minimalni stabilni premer magnetnega mehurčka $2r_{co} \approx 3{,}9\,\lambda$ pri debelini filma $h_{opt} \approx 3{,}3\,\lambda$. Da bo pri h_{opt} dosežen minimalni premer magnetnega mehurčka $2r_{co}$ je potrebno prečno magnetno polje jakosti $H_B \approx 0{,}3\ M_s$ (slika 6).



Slika 6.

Dinamika. Do sedaj smo obravnavali obnašanje cilindrične domene v krajevno nespremenljivem prečnem magnetnem polju $\vec{H}_B$. Poglejmo sedaj kaj se zgodi, če je prečno magnetno polje krajevno spremenljivo $\vec{H}_B = \vec{H}_B(x)$, torej če obstaja gradient prečnega magnetnega polja, ki je različen od nič. Primer prikazuje slika 7. Magnetna cilindrična



Slika 7.

domena premera 2r se nahaja v tankem filmu, ki ga postavimo v ravnino x-y. Njena magnetizacija $\vec{M}_s$ je v smeri + z. Gradient prečnega magnetnega polja ΔH_B je konstanten. Vrednost magnetne poljske jakosti v točkah x_o in $x_o + 2r$ je takšna, da je cilindrična domena še stabilna. Zaradi gradienta magnetne poljske jakosti $\Delta H_B = 2r\,\partial H_B/\partial x$, se pojavi sila, ki deluje na cilindrično domeno in zaradi katere se prične cilindrična domena premikati v smeri + x osi s hitrostjo

$$|v| = \begin{cases} \frac{1}{2}\mu_w\left(|\Delta H_B| - \frac{8H_c}{\pi}\right), & |\Delta H_B| > \frac{8H_c}{\pi} \\ 0, & |\Delta H_B| \leq \frac{8H_c}{\pi} \end{cases} \qquad (14)$$

kjer je μ_w gibljivost cilindrične domene v m^2/As in H_c koercitivnost snovi v kateri se cilindrična domena pojavlja.

4. SNOVI

Snovi v katerih se pojavijo magnetni mehurčki morajo biti anizotropne v eni smeri, tako, da je lažja os magnetizacije pravokotna na lice filma (anizotropija v eni osi se lahko doseže pod posebnimi pogoji in ni nujno, da je to lastnost same snovi). Te snovi so ortoferiti redkih zemelj, heksagonalni feriti, MnBi, enoosni garneti redkih zemelj, Gd-Co, Gd-Co-Mo. Takšne snovi karakterizira poleg karakteristične dolžine snovi λ, ki je definirana z (12) in podaja razmerje med energijsko gostoto na enoto površine stene cilindrične domene σ_w in magnetostatično energijsko gostoto na enoto volumna, tudi takoimenovana kvaliteta snovi Q, ki je definirana z izrazom (15) in podaja razmerje med anizotropnim poljem in saturacijsko magnetizacijo.

$$Q = \frac{H_k}{M_s} \qquad (15)$$

H_k je anizotropno polje v A/m, ki je enako $H_k = 2K_u/\mu_o M_s$, pri tem je K_u enoosno anizotropna konstanta v J/m^3. Kvaliteto snovi je mogoče pisati tudi kot

$$Q = \frac{2K_u}{\mu_o M_s^2} \qquad (16)$$

Snovi, ki so primerne za mehurčne pomnilnike morajo imeti čim manjšo karakteristično dolžino snovi λ in faktor kvalitete Q večji od ena. Gibljivost cilindrične domene μ_w naj bo čim večja, njena koercitivnost pa čim manjša, kar omogoča velike hitrosti gibanja mehurčkov.

Ortoferiti redkih zemelj. Za izdelavo prototipov mehurčnih pomnilnikov so se najprej uporabljali ortoferiti. Njihova splošna kemijska formula je $RFeO_3$, kjer je R ustrezna kombinacija itrija Y in redkih zemelj (lantana La, prazeodima Pr, neodima Nd, samarija Sm, evropija Eu, gadolinija Gd, terbija Tb, dispozija Dy, holmija Ho, erbija Er, tulija Tu, iterbija Yb in lutecija Lu). Pri ortofe-

S N O V	M_s [kA/m]	σ_w [mJ/m^2]	λ [μm]
ortoferiti redkih zemelj			
$YFeO_3$	8,4	1,8	2,5
$NdFeO_3$	4,9	1,1	4,4
$SmFeO_3$	6,7	1,3	2,9
$EuFeO_3$	6,6	1,6	3,7
$GdFeO_3$	7,5	1,7	2,9
$TbFeO_3$	10,9	1,7	1,4
$DyFeO_3$	10,2	1,8	1,7
$HoFeO_3$	7,3	1,7	3,3
$ErFeO_3$	6,5	1,6	3,9
$TmFeO_3$	11,2	2,4	1,9
$YbFeO_3$	11,4	3,9	3,0
$LuFeO_3$	9,5	3,9	4,3
$Sm_{0,6}Er_{0,4}FeO_3$	6,6	0,35	0,80
$Sm_{0,55}Tb_{0,45}FeO_3$	8,6	0,30	0,40
enoosni garneti redkih zemelj (nanašanje z naparevanjem)			
$Er_2Tb_1Al_{1,1}Fe_{3,9}O_{12}$	10,8	0,19	1,27
$Gd_{2,34}Tb_{0,66}Fe_5O_{12}$	10,9	0,23	1,53
$Gd_{0,95}Tb_{0,75}Er_{1,3}Al_{0,5}Fe_{4,5}O_{12}$	14,4	0,083	0,35
$Eu_2Er_1Ga_{0,7}Fe_{4,3}O_{12}$	19,7	0,31	0,64
$Y_2Gd_1Al_{0,8}Fe_{4,2}O_{12}$	26,1	0,18	0,21
$Y_{1,8}Eu_{0,2}Gd_{0,5}Tb_{0,5}Al_{0,6}Fe_{4,4}O_{12}$	35,8	0,36	0,22
$Eu_{1,5}Gd_{1,5}Al_{0,5}Fe_{4,5}O_{12}$	17,4	0,29	1,78
$Eu_{1,9}Gd_{1,1}Al_{0,5}Fe_{4,5}O_{12}$	12,7	0,14	0,67
$Pr_1Gd_2Ga_{0,5}Fe_{4,5}O_{12}$	12,1	0,27	1,50
enoosni garneti redkih zemelj (epitaksija iz tekoče faze) na $Ga_3Gd_5O_{12}$ substratu			
$Eu_2Er_1Ga_{0,7}Fe_{4,3}O_{12}$	13,8	0,17	0,17
$Eu_1Er_2Ga_{0,7}Fe_{4,3}O_{12}$	9,5	0,20	2,2
$Er_{1,99}Gd_{1,01}Ga_{0,22}Fe_{4,78}O_{12}$	11,8	0,173	0,9
$Y_{0,94}Gd_{1,07}Yb_{0,57}La_{0,42}Al_{0,7}Fe_{4,3}O_{12}$	19,1	0,21	0,46
$Y_{1,03}Gd_{1,29}Yb_{0,68}Al_{0,7}Fe_{4,3}O_{12}$	13,9	0,125	0,51
$Y_{1,5}Eu_{1,5}Al_{0,7}Fe_{4,3}O_{12}$	37,8	0,29	0,17
$Y_{1,3}Eu_{1,7}Al_1Fe_4O_{12}$	19,9	0,21	0,42
$Y_1Gd_1Tm_1Fe_{4,2}Ga_{0,8}O_{12}$	16,3	0,27	0,75
$Y_{1,88}Lu_{0,2}Ca_{0,92}Ge_{0,92}Fe_{4,08}O_{12}$	13,6	0,11	0,47
enoosni garneti redkih zemelj (epitaksija iz tekoče faze) na $Sm_3Gd_5O_{12}$ substratu			
$Eu_2Y_1Fe_5O_{12}$	124,0	-	0,066

Tabela 1.

ritih je Q zelo velik in je npr. za $Sm_{0,55}Tm_{0,45}Fe\,O_3$ pri sobni temperaturi (295°K) okrog 25. V tabeli 1 so podani nekateri parametri ortoferitov.

Enoosni garneti redkih zemelj. Snovi, ki se danes največ uporabljajo za izdelavo mehurčnih pomnilnikov in ki so pokazale najboljše karakteristike, so enoosni garneti, katerih splošna kemijska formula je $R_3Fe_5O_{12}$. To so sintetične spojine z isto kompleksno strukturo kot jo ima mineral garneta. Kljub uspehom pri rasti velikih monokristalov mešanih garnetov redkih zemelj in kasnejšemu rezanju na ploščice, je bila njihova uporaba delno omejena, dokler ni bila razvita tehnika epitaksialne rasti monokristaličnih filmov. Današnja tehnologija v glavnem temelji na epitaksialni rasti filmov mešanih garnetov in sicer sta razviti dve metodi: nanašanje z naparevanjem in epitaksija iz tekoče faze, od katerih je prevladala prav slednja. V tabeli 1 so nekatere lastnosti mešanih garne-

tnih filmov in kot vidimo je karakteristična dolžina snovi λ pri garnetih mnogo manjša kot pri ortoferitih.

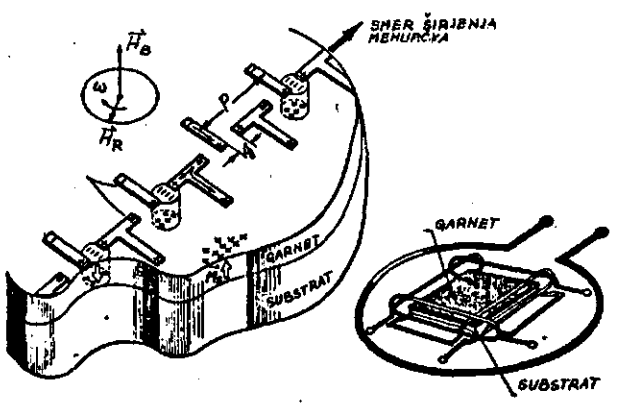
5. MAGNETNI MEHURČEK KOT NOSILEC INFORMACIJE

Do sedaj opisane lastnosti magnetnega mehurčka nakazujejo možnost uporabe le-tega kot nosilca informacije, saj nam njegova prisotnost v nekem trenutku na določenem mestu predstavlja logično "1", njegova odsotnost pa "0". Dinamične lastnosti magnetnega mehurčka, ki omogočajo njegovo časovno diskretno gibanje, nudijo možnost uporabe magnetnega mehurčka pri oblikovanju logičnih funkcij (pomik, konjunkcija,...) kakor tudi za pomnenje informacij.

Pri gradnji mehurčnega pomnilnika je potrebno omogočiti naslednje operacije:
- vpis informacije (generiranje in brisanje magnetnega mehurčka),
- dostop do informacije (širjenje magnetnega mehurčka),
- branje informacije (detektiranje magnetnega mehurčka).

Širjenje (propagation). Magnetni mehurček je mogoče premikati s spremenljivim prečnim magnetnim poljem, kot je prikazano v poglavju 3. Za tvorjenje lokalnega gradienta prečnega magnetnega polja je bilo razvitih nekaj metod [3] , od katerih se danes uporablja izključno širjenje s pomočjo rotirajočega magnetnega polja (field - access), ki naj bi ga v bližnji prihodnosti nadomestilo širjenje s tokom[4] (current - access).

Oglejmo si najprej širjenje z rotirajočim poljem. Pri tej metodi tvorimo lokalni gradient prečnega magnetnega polja ΔH_B s pomočjo tankih permalojnih vzorcev (Ni-Fe), ki jih ustrezno magnetno polariziramo z rotirajočim magnetnim poljem jakosti $\vec{H}_R$, ki ga ustvarimo z dvema med seboj pravokotnima tuljavicama (slika 8). Tipična vrednost rotirajočega magnetnega polja $\vec{H}_R$ je 0,8 ÷ 2,4 kA/m.

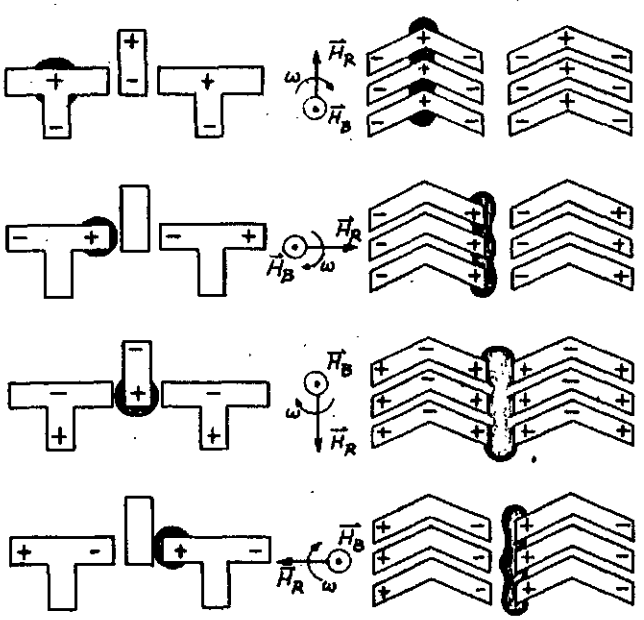


Slika 8.

4) V Bell Telephone Laboratories so razvili več variant te tehnologije, ki so opisane v [6] .

Rotirajoče magnetno polje namagneti permalojne vzorce, ki sedaj privlačijo ali odbijajo magnetni mehurček, pač odvisno od polaritete (slika 9). Z ustrezno konfiguracijo permalojnih vzorcev se doseže, da so pozicije magnetnih polov točno locirane in tako je z vsako rotacijo polja določena pozicija mehurčka.

Oblike permalojnih vzorcev so zelo različne, razen T vzorcev, ki so se najprej pojavile, se danes uporabljajo še Y, X, TX, škarnice, itd... Pri vseh naštetih konfiguracijah igra zelo pomembno vlogo medvzorčna razdalja g, ki naj bo čim krajša (za T vzorce je $g < 1/3\,d$, za škarnice pa je $g < 2/3\,d$, kjer je d širina valovite proge v filmu, ko ta še ni v zunanjem magnetnem polju) in ki še zagotavlja magnetno polarizacijo permalojnih vzorcev. V težnji za povečevanjem gostote mehurčnih pomnilnikov proizvajalci manjšajo dimenzije (perioda p = 4,5 ÷ 5,5 d) permalojnih vzorcev, vendar morajo pri tem ohraniti eksperimentalno določeno razmerje p : g, ki naj bo 8 : 1 do 16 : 1. Ker je manjšanje medvzorčne razdalje g tehnološko omejeno, so s tem omejene tudi dimenzije propagacijskih vzorcev (perioda p), kakor tudi propagacijska frekvenca f_p, ki je približno 250 kHz.



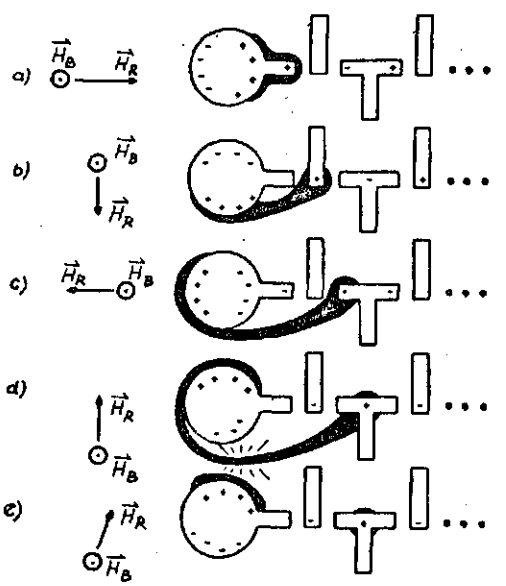
Slika 9.

Širjenje s tokom je nova tehnologija s katero dosegajo propagacijske frekvence > 1MHz in zelo visoke gostote $> 10^7$ bitov/cm^2. Velika prednost te tehnologije je, da za tvorjenje gradienta prečnega polja ne uporabljajo tuljavic, temveč posebno oblikovano prevodno (Al-Cu) plast, ki je nanešena s fotolitografskimi postopki na garnetni film.

Generacija (generation) in brisanje (annihilation). Vpis novega podatka v mehurčni pomnilnik pomeni generacijo novega mehurčka. V ta namen je bilo razvitih več postopkov generacije. V tehnologiji, ki za širjenje magnetnega mehurčka uporablja rotirajoče magnetno polje $\vec{H}_R$ in permalojne vzorce (trenutno najbolj uporabljana), sta izpopolnjeni dve metodi generacije in sicer:
- generacija s "kalitvijo" (seed domeins) in
- generacija s tvorjenjem jedra (nucleate type).

Generacijo s kalitvijo [3,7] opišimo na primeru T vzorcev (slika 10). Vzorec, ki je nosilec "kali" magnetne domene je oblikovan v obliki diska. Rotirajoče magnetno polje $\vec{H}_R$ namagneti disk in s tem določa položaj "kalne" domene na obodu diska (slika 10a). Zasuk polja, $\vec{H}_R$ za 90° v smeri urinega kazalca povzroči, da se prvi propagacijski vzorec polarizira tako, da pritegne del "kalne" domene (slika 10 b). Pri zasuku polja $\vec{H}_R$ za nadaljnih 90°, je "kalna" domena razpeta med pozitivnima poloma diska in drugega propagacijskega vzorca, da bi se ob naslednjem zasuku polja $\vec{H}_R$ za 90° pod vplivom negativnega pola diska pretrgala (slika 10 d). To spontano generacijo je mogoče s spreminjanjem bodisi prečnega polja $\vec{H}_B$ ali rotirajočega polja $\vec{H}_R$ nadzorovati. Minimalna vrednost polja $\vec{H}_R$ potrebna za generacijo novega mehurčka je večja kot za njegovo širjenje, kar pomeni, da pri ustreznem zmanjšanju polja $\vec{H}_R$ mehurčka ni mogoče generirati, možno pa ga je premikati.

Druga metoda, to je generacija s tvorjenjem jedra, je danes uporabljena skoraj pri vseh izvedbah mehurčkinih pomnilnikov. Novo magnetno domeno lahko ustvarimo z ustrezno velikim magnetnim poljem jakosti $\vec{H}_Z$, ki je obratno usmerjeno od prečnega magnetnega polja $\vec{H}_B$. Velikost polja $\vec{H}_Z$ za enoosno anizotropne snovi je podano z relacijo [5] (17).



Slika 10.

5) Stoner in Wohlfarth (1948).

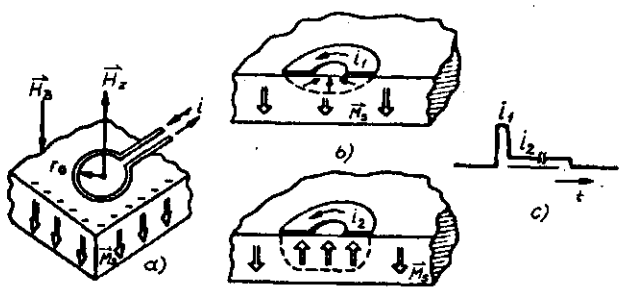
$$(H_Z - H_B)^{2/3} + H_R^{2/3} = H_K^{2/3} \qquad (17)$$

Kot vidimo je velikost polja $\vec{H}_Z$ odvisna od anizotropnega polja $\vec{H}_K$ oziroma od kvalitete snovi Q. Ker so imeli ortoferiti sorazmerno veliko kvaliteto (H_K je reda 10^4 kA/m) bi to pomenilo, da bi morali za generacijo nove domene ustvariti izredno veliko polje $\vec{H}_Z$. S pojavom enoosnih garnetov redkih zemelj, katerih anizotropno polje $\vec{H}_K$ je stokrat manjše kot pri ortoferitih, se je ta metoda zelo uveljavila. Polje $\vec{H}_Z$ je mogoče tvoriti s tokovno zanko (current nucleate generator), permalojnimi vzorci (permalloy bar generator) ali obojim.

Na sliki 11 je prikazana generacija magnetne domene s tokovno zanko polmera r_o. Tok i , ki teče po zanki ustvarja magnetno polje jakosti

$$\vec{H}_z = -\frac{i}{2 r_o}\,\vec{l}_{H_B} \qquad (18)$$

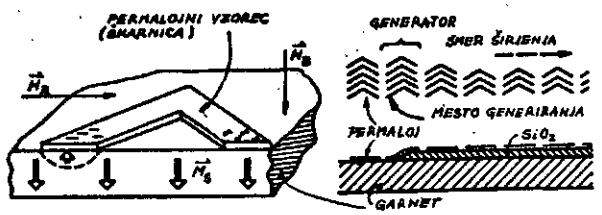
kjer je $\vec{l}_{H_B}$ enotin vektor v smeri $\vec{H}_B$ (slika 11 a). Tokovni impulz i je sestavljen iz dveh delov (slika 11 c), prvi del, katerega amplituda je i_1, je sorazmerno kratek (nekaj ns) in ustvari močno magnetno polje $\vec{H}_Z$, ki premagneti del garnetnega filma neposredno ob tokovni zanki (zasnova domene). Drugi del impulza, ki je nekaj desetkrat daljši in katerega amplituda i_2 je nekajkrat manjša od i_1, omogoča dokončno oblikovanje stabilne cilindrične domene (magnetnega mehurčka).



Slika 11.

Magnetni mehurček je mogoče ustvariti tudi s permalojnimi vzorci, ki jih magneti rotirajoče polje $\vec{H}_R$ in ki so nanešeni neposredno na garnetni film (slika 12). Jakost magnetnega polja $\vec{H}_Z$ je proporcionalna magnetni gostoti pola in lahko doseže jakost $M_s/2$, kar zadošča za nastanek nove magnetne domene. Z odmikanjem permalojnega vzorca od garnetne osnove, jakost polja $\vec{H}_Z$ hitro pada (pri 12 μm oddaljenosti znaša le še 1/30 prvotne vrednosti), tako da novih domen ni več mogoče generirati. Permalojni vzorci ločeni od garneta s plastjo SiO_2 služijo le za širjenje magnetnega mehurčka.

Generacija s tvorjenjem jedra, ki združuje oba opisana postopka (tokovno zanko in permalojne vzorce) je pokaza-



Slika 12.

la najboljše rezultate in se danes najčešče uporablja.

Brisanje informacije pomeni uničenje magnetnega mehurčka. To dosežemo pravzaprav na enak način, kot je izvedena generacija s "kalitvijo" (slika 10), le da rotirajoče polje $\vec{H}_R$ sučemo v nasprotni smeri.

Detekcija (detection). Detekcija magnetnega mehurčka, to je branje informacije, predstavlja razmeroma velik problem spričo ekstremno majhnih dimenzij in energije magnetne domene. Do sedaj so izpopolnili štiri metode: optična detekcija, detekcija na principu spremembe magnetnega pretoka, polprevodniški Hallov efekt in najbolj razširjena magnetorezistenčna detekcija.

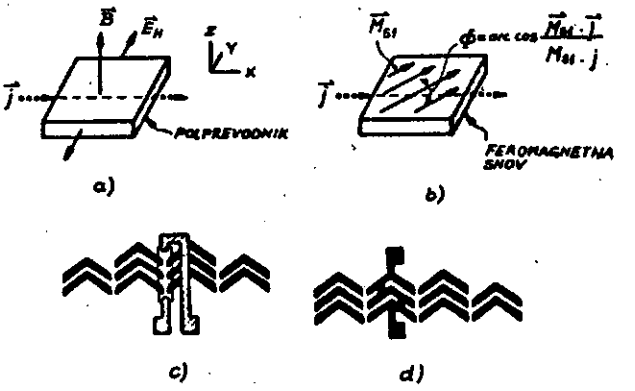
Optična detekcija izkorišča Faradayev efekt, kjer s pomočjo izvora polarizirane svetlobe naredimo magnetni mehurček "viden", ter njegovo prisotnost na dani spominski lokaciji detektiramo s fotodiodo.

Druga metoda deluje na principu merjenja spremembe magnetnega pretoka, ki se pojavi pri premiku magnetnega mehurčka preko detektorske zanke. Napetost, ki se inducira je zelo majhna ($\approx 1\ \mu V$), tako, da je koristni signal zelo težko razbrati iz močnega šumnega ozadja.

Magnetni mehurček je mogoče detektirati tudi s Hallovim elementom (polprevodniškim) preko katerega teče tok gostote $\vec{j}$ (slika 13 a). Zaradi spremembe magnetnega polja, ki nastane pri prehodu magnetnega mehurčka mimo detektorja, nastane sprememba električnega polja $\vec{E}_H$. Npr. pri prehodu magnetnega mehurčka (v $TmFeO_3$ ortoferitu) preko Hallovega elementa dimenzij 50x50x1,4μm in tokovne gostote $j = 0{,}1\ mA/m^2$ se generira Hallova napetost 7 mV.

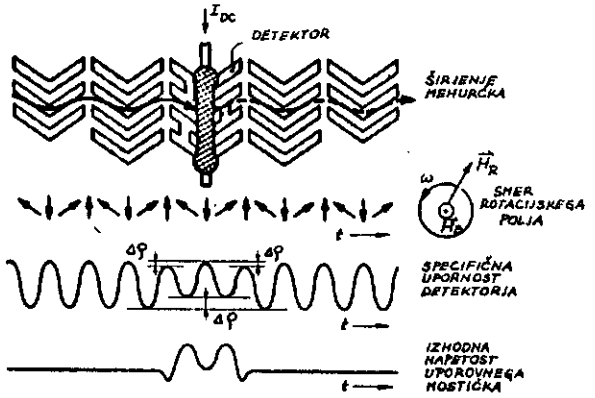
Kot rečeno je najbolj razširjena metoda z magnetnorezistenčnim detektorjem. Zakaj pravzaprav gre? Tu izkoriščamo magnetorezistenčnost Hallovega elementa (feromagnetnega), to je odvisnost njegove specifične upornosti ϱ od velikosti magnetizacije $\vec{M}_{sl}$ in kota ϕ, ki ga oklepata tokovna gostota $\vec{j}$ in vektor magnetizacije $\vec{M}_{sl}$ (slika 13b), ki jo pišemo kot

$$\varrho = \varrho_0 + M_{sl}^2 (k_1 + k_2 \cos^2 \phi) \qquad (19)$$



Slika 13.

Pri tem je ϱ_0 specifična upornosti v primeru ko ni zunanjega polja in $\vec{M}_{sl}$ vektor saturacijske magnetizacije Hallovega elementa. Praktično je takšen detektor realiziran tako, kot prikazujeta sliki 13 c, kjer je Hallov element, to je permalojni trak, nanešen pod propagacijske vzorce in slika 13 d, kjer že sam propagacijski element (ustrezno oblikovan) služi kot detektor. Vektor magnetizacije $\vec{M}_{sl}$, ki sledi rotirajočemu magnetnemu polju $\vec{H}_R$ in s tem spreminja kot ϕ, povzroči spreminjanje specifične upornosti detektorskega elementa, preko katerega teče enosmerni tok I_{DC}. Dokler v garnetnem filmu, ki se nahaja pod detektorjem, ni magnetnega mehurčka, je amplituda nihanja specifične upornosti konstantna, ko pa se magnetni mehurček pojavi, se spremeni magnetizacija detektorja , vsled tega pa se spremeni tudi njegova specifična upornost (slika 14). Relativna sprememba specifične upornosti $\Delta\varrho/\varrho_0$ je nekaj procentov (npr. za 88 % Ni - 12 % Fe permaloj



Slika 14.

je 5 %) in jo detektiramo z uporovnim mostičkom. Mostiček je zgrajen tako, da sta v dveh vejah konstantni upornosti, v eni je aktivni detektor in v drugi "slepi" (dummy) detektor, ki je enako kot aktivni podrejen vplivom rotirajočega polja $\vec{H}_R$, le da pod njim ne vodimo magnetnih mehurčkov. Izhod mostička vežemo na ustrezno elektronsko vezje, ki napetostni signal ojači in ustrezno oblikuje, tako, da so logični nivoji izenačeni z nivoji okoliške logike.

6. ZAKLJUČEK

V članku smo skušali podati osnovni vpogled v fizikalne lastnosti mehurčnega pomnilnika, pri čemer smo v poglavju 5 nekoliko podrobneje opisali eno od danes dominantnih tehnologij, to je uporabo rotirajočega polja in permalojnih propagacijskih vzorcev. V želji za izboljšanjem osnovnih lastnosti mehurčnih pomnilnikov (krajši čas dostopa in višja gostota), potekajo raziskave v vodilnih svetovnih laboratorijih, kot sta Bell Telephone Laboratories in IBM, na razvoju novih tehnologij.

IBM je v želji za povečanjem gostote razvil takoimenovano urejeno mrežo mehurčkov (Bubble Latice File), kjer je informacija shranjena v obliki različnih magnetizacij stene mehurčkov, ki jih je mogoče detektirati (branje) ali spreminjati (vpis). Takšna struktura omogoča petkratno zvečanje gostote [10] .

Da bi povečali propagacijske frekvence (>1MHz) skušajo pri Bellu nadomestiti tuljavici, ki ustvarjata rotirajoče magnetno polje, s fotolitografskim nanašanjem prevodnih plasti (ena ali dve) na garnetni film (širjenje s tokom - current access) [6] . A. H. Bobeck je dejal, da ne vidi razlogov, da ne bi s to novo tehnologijo dosegli propagacijskih frekvenc 20 MHz.

7. LITERATURA

[1] A.H.BOBECK: The Bell System Tech. Journal, Vol. 46, No. 8, pp. 1901 - 1925, Oct. 1967

[2] S. CHIKAZUMI: Physic of Magnetism, J. Wiley & Sons, 1964

[3] A.H. BOBECK, E. DELLA TORRE: Magnetic Bubbles, Amsterdam, The Netherlands: North-Holland Publishing, 1975

[4] A.H. BOBECK, R.F. FISCHER, A.J. PERNESKI, J.P. REMEIKA & L.G. VAN UITERT: IEEE Trans. on Magnetics, Vol. MAG-5, No. 3, pp. 544-553, Sept. 1969

[5] A.H. BOBECK, P.I. BONYHAR & J.E.GEUSIC: Proceedings of the IEEE, Vol. 63, No. 8, pp. 1176-1195, August 1975

[6] A.H. BOBECK, S.L. BLANK, A.D. BUTHERUS, F.J. CIAK & W. STRAUSS: The Bell System Tech. Journal. Vol. 58 No. 6, July - August 1979

[7] A.J. PERNESKI: IEEE Trans. on Magnetics, Vol. MAG-5 No. 3, pp. 554-557, Sept. 1969

[8] S.V. AHAMED: The Bell System Tech. Journal, Vol. 51, No. 2, pp. 461-485, Feb. 1972

[9] D.C. MARKHAM: Electronic Engineering, pp. 85-99, June 1979

[10] M.S. COHEN, H. CHANG: Proceedings of the IEEE, Vol. 63, No. 8, August 1975

[11] S. BESENIČAR, D. KOLAR: IJS Delovno poročilo, DP-1936 Januar 1980

MICROCOMPUTER SYSTEM BUSES

FRANC NOVAK

INSTITUT JOŽEF STEFAN, LJUBLJANA

UDK:681.3 (083.7)

The paper reviews some most popular microcomputer common buses which appeared together with the generation of 16-bit devices. Bus specifications considerably differ from each other, since most of them are primarily suited to a certain microprocessor family. The paper describes the most important functional characteristics of the following microcomputer buses: S-100, VERSAbus, Multibus, TM 990 bus and ZBI bus. A brief summary of connector types and card dimensions for each bus is given as well.

MIKRORAČUNALNIŠKA VODILA. Članek podaja pregled mikroračunalniških vodil, ki so nastala vzporedno z generacijo 16-bitnih mikroprocesorjev. Vodila se med seboj precej razlikujejo, saj so v večini primerov "pisana na kožo" ene od mikroprocesorskih familij. Članek predstavlja mikroračunalniška sistemska vodila: S-100, VERSAbus, Multibus, TM 990 bus in ZBI bus z ozirom na njihove najpomembnejše funkcionalne karakteristike. Za vsako vodilo so opisani tudi tipi konektorjev in dimenzije kartic.

I. INTRODUCTION

One of the most important features of microcomputer system design is the implementation of the system bus. A common bus is a composition of unidirectional or bidirectional lines which transfer information and electrical power between the various components of a microcomputer system. These lines are characterised from a functional point of view (e.g. address lines, data lines,...), as well as from an electrical point of view (e.g. open collector line, three state line,...).

There are two basic types of functional elements that connect to the bus: masters and slaves. A bus master is any module having the ability to control the bus. It is capable to address bus slaves by generating proper control and address signals. A bus master has the capability to transfer messages to or from the addressed slave. A bus slave decodes the address lines and acts upon the command signals from the masters.
The overall behaviour of the events on the bus is defined by the bus protocol.

The last but not least important feature of the bus are its mechanical specifications (e.g. connector types, board size etc.). The bus structure is important to the users when they intend to upgrade their systems by, say, commercially available memory boards or peripherals.

II. MICROCOMPUTER BUS PROPOSALS

In the area of 8-bit microcomputers almost as many different backplane packaging and functional specifications were proposed as there were microprocessor manufacturers. Obviously, some of them gained more popularity than others (SBC 80, EXORcisor, Z-80 bus, PRO-LOG, to cite just a few of them). Still none is the favorit. As the 16-bit devices began to appear on the

market the same story happened again though with a considerably smaller number of bus proposals. Intel Multibus, Motorola VERSAbus, TM 990 bus and ZILOG ZBI bus are widely proclaimed but they all tend to be specific to the computer they were designed for. In some cases, the design of the bus is constrained by the fact that compatibility with preceding members of the microprocessor family is provided. VERSAbus and ZBI bus are here outstanding exceptions. They still provide means for supporting 8-bit microprocessors but their upper limit is 32 rather than 16-bit data transfer.

Despite differences in internal CPU architecture, bus protocol of any microprocessor system exerts similar master - slave relations. For example, communication with memory and peripherals requires similar signals and timing relationship. Consequently, a universal bus structure which is independant of actual microprocessor type should be possible to define. The S-100 bus resulted from the effords to create such "universal" bus. With its specification openly published by the IEEE it could meet most requirements for the present microcomputer systems and it had been adopted by many manufacturers. Yet, a drawback is that it had been originally designed for 8-bit data and 16-bit address lines. Its proposed exetention to 16-bit data lines has not been widely accepted.

In October 1980, the IEEE Computer Society proposed the 796 Bus Standard which is actually only a slight modification of the Multibus and is stated to be "a cooperative industry effort toward establishing a standard for a large number of manufacturers and users of microcomputer modules".

The purpose of this paper is not to make an assessment of advantages and disadvantages of the contenders for the title of "industry standard". Instead, most common features of microprocessor bus proposals are reviewed to acquaint the reader with their basic design.

III. ADDRESS AND DATA LINES

All the above mentioned buses provide means for 8 and 16-bit data transfer, but still considerable differences exist in the way to accomplish this. Likawise, the number of address lines differs from each other resulting in different addressing capabilities.
The data bus of the S-100 bus consists of 16 lines which are grouped as two unidirectional 8-bit buses for byte transfer and as a single bidirectional bus for 16-bit word format. The address bus consists of 16 lines for standard memory addressing (64 k locations), or of 24 lines for extended memory addressing (16 M locations).

The Multibus provides 16 bit data bus and 20 bit address bus. The implementation of the 8 and 16-bit data transfer employs a special Swap Byte Buffer which must be included in any 16-bit master or slave in order tc maintain compatibility with previous 8-bit masters and slaves. The 20 address lines allow a maximum of 1 M bytes of memory to be accessed.

The VERSAbus provides 16 bit data and 24 bit address lines which can both be expanded to 32 lines using the second connector.

The TM 990 data bus consists of 16 lines. The 20 address lines are comprised of 16 basic and 4 extended address lines. Up to 1 M bytes can be addressed by means of the memory mapping technique.

The most significant difference between the ZBI bus and other buses is in the way of the transfer of data and address information. The ZBI bus uses a multiplexed bus structure with 32 data or address lines. This gives a smaller number of bus interconnections, at the expense of a slight increase in the complexity of the circuits connected by the bus. 32 lines provide means to accomodate future 32-bit microprocessors and offer wide addressing capability.

IV. BUS ARBITRATION

All 16-bit microprocessors are designed to allow multiprocessing. They have facilities to handle bus control which can accomodate several bus masters on the same system, each taking control of the bus for its own data transfer.

The S-100 bus arbitration system uses 4 bus lines for arbitrating among 16 temporary masters. In general, one permanent master may exist in the system and it has the highest arbitration priority. Each temporary master has a unique priority number which it asserts on the arbitration bus at an appropriate time. A temporary master can get the control of the bus from the permanent master only for an arbitrary number of bus cycles. Then it must return the control to the

permanent master. The parallel priority resolution is implemented by means of additional logic stored on each master board interfacing to the arbitration lines.
There are another 4 lines on the bus to disable the line drivers of the permanent master making together 8 control lines for the DMA arbitration.

In cases where the existence of the permanent master in the system may prove to be inefficient, a dummy master may replace it. The dummy master merely passes the control of the bus from one temporary master to another. It is especially suitable when a number of processors co-exist in a single system.

The Multibus offers two bus exchange priority techniques: serial and parallel. Serial priority resolution is accomplished with a daisy - chain technique. A relatively small number of masters can be accomodated in this way. Due to the present time hardware limitations only up to 3 masters can be accomodated in a system with bus clock period of 100 ns.
In the parallel technique, the priority is resolved by means of a priority resolution circuit. This circuit must be externally supplied and its position in the system is not strictly determined by the bus standard. If the parallel technique is used a practical limit of 16 masters in a system is stated.

The VERSAbus provides 5 level bus arbitration. Each level can be implemented as a daisy - chain to increase the number of bus masters. The priority is first resolved among the five levels and then within the elements of the daisy - chain of the level which gained the priority.

In the TM 990 bus specification the arbitration scheme is limited to a single daisy- chain like the Multibus serial resolution.

Similarly, the ZBI bus employs a serial daisy - chain resolution technique and four control lines to enable multiple processors to share the bus. While in the previously mentioned bus specificatoins a DMA module uses the same arbitration scheme as other processor modules in the system, the ZBI bus makes a distinction: it provides separate daisy - chain for the DMA arbitration.

V. INTERRUPT REQUEST SCHEME

The interrupt request scheme in the S-100 bus is comprised of an 8-level maskable vectored interrupt system and a non - maskable interrupt which is an optional control input to the bus masters. The eight interrupt request lines are inputs to a bus slave or an interrupt controller which masks and prioritizes the requests. As a result it outputs the interrupt request to the permanent master.

The Multibus can support bus vectored and non-bus vectored interrupts at the same time. Bus vectored interrupts are generated by the slave Priority Interrupt Controllers which transfer the vector address to the bus master. The non-bus vectored interrupts are handled on the bus master. A bus slave that requests the interrupt does not convey interrupt vector address on the bus, it is generated by the interrupt controller on the master.
Eight interrupt request lines enable 8-level priority interrupts.

To handle interrupts the VERSAbus provides a 7-level priority control. The 7 interrupt request lines are of the wire-ORed configuration so that each level can be expanded by a daisy - chain. The VERSAbus accomodates bus vectored interrupts.

15 general purpose vectored interrupts are defined in the TM 990 bus each having its own priority level. The interrupts are maskable by the primary master.

Interrupts on the ZBI signal lines can be implemented as maskable or non-maskable, vectored or non-vectored depending on how the CPU is configured. 9 interrupt lines organized into 3 independed groups exist on the bus each having its unique priority level.

VI. MISCELLANEOUS FUNCTIONS

A number of signals and their functions have not been described not to go too much into detailes. Nevertheless, let us review just a few interesting features. The decision, if they are worth mentioning is left to the reader.

The ZBI bus and the VERSAbus include parity check bits for data and address lines, while in general, each bus includes at least one generalized error line that indicates that the current operation is producing an error.

For the case of power failure, control signals are provided that indicate that the power is going to fail enabling the system to enter the power fail sequence. The system may store important data in memory which is powered by the standby power supply.

To enhance the system's integrity, clock signal lines are accompanied by their own signal-ground lines on the backplane.

	NUMBER OF CONNECTORS	CONNECTOR DESCRIPTION	CARD DIMENSIONS (in inches)
S-100	1	(50/100 pin) board edge connecter	5.125x10
VERSAbus	2	P1: (70/140 pin) board edge c. P2: (60/120 pin) board edge c.	9.25x6.5 (reduced size) or 9.25x14.5
Multibus	2	P1: (43/86 pin) board edge c. P2: (30/60 pin) board edge c.	12.00x6.75
TM 990	1	(50/100 pin) board edge c.	11x7.5
ZBI bus	2	P1: (32/32/32 pin) wirewrap c. P2: (32/32/32 pin) wirewrap c.	6.3x3.9 (Eurocard) or 6.3x9.2 (Double Euroc.)

Table 1

VII. MECHANICAL SPECIFICATIONS

Table 1. summarizes some most important mechanical specifications concerning the physical design of a bus backplane and the dimensions of the printed circuit boards that plug into the bus interface. Some remarks are necessary:

Signals of the VERSAbus P1 connector allow the implementation of a complete 16-bit device on a reduced size board. The P2 connector provides extension to service future 32-bit devices. P1 connector of the Multibus is the primary connector and P2 is an auxiliary connector. The ZBI system signals are all gathered together on the P1 connector, the P2 connector is reserved entirely for the user application.

VIII. CONCLUSION

Standardization of the microcomputer bus is an important element to be considered when microcomputer systems are designed. A number of bus standards have been proposed in the past. Each microprocessor maker was solving problems from his own point of view which resulted also in the different bus specifications.
This may be the reason why the IEEE works on a processor - independant standard P896 referred as the "future bus". The future bus, if it becomes the IEEE standard, will operate at rates over 10 MHz, the arbitration scheme will handle up to 64 masters in a sofisticated arbitration protocol, the interrupt control will handle 1024 priority levels.
Another bus design is reported from Intel. Developing the new 32-bit microprocessor iAPX 432, new bus interconnection concepts are announced. The last word has not been said yet.

IX. LITERATURE

1). F. Taylor, J. Race: Standards for microsystems, (Microproc. and Microsystems, vol4,n8)
2). L.Yencharis:...But the right bus evokes the system's best, (Electronic design, May 1980)
3). M. Marshall: IEEE readies backplane standard, (Electronics, September 25, 1980)
4). A. Clements: Computer system buses, (Microproc. and microsystems, vol 3, no 9)
5). J. Rattner, W. Lattin: Ada determines architecture of 32-bit microprocessor, (Electronics, February 24, 1981)
6). Standard specification for S-100 bus interface devices, (Computer, July 1979)
7). L. Bender: ZBI:a system bus for the Z 8000. (Mini Micro Systems, June 1980)
8). VERSAbus, Multibus, TM 990 bus, (Manuals)
9). Proposed microcomputer system 796 bus standard, (Computer, October 1980)

NOVICE IN ZANIMIVOSTI

ENOTE Z GIBKIM DISKOM: TEHNOLOŠKA NAPOVED

V letu 1980 je dvostranski gibki disk dosegel svojo tehnološko zrelost. Doslej so bili ti diski problematični ter so predstavljali vprašljiv proizvod v prepričanju kupcev. Z dobrimi tehnološkimi rešitvami se je začela velikoserijska proizvodnja dvostranskih diskovnih enot. T.i. OEM kupci so odkupili že 80 % proizvodnje. Enostranske diskovne enote so dosegle svoj prodajni višek, poraba dvostranskih enot pa je skokovito narasla, tako da lahko pričakujemo padec prodaje enostranskih enot ter prenehanje njihove proizvodnje.

Rešitev problema obrabe. Intenzivna obraba materiala na dvostranskih diskih je nastajala zaradi problema nameščanja glave, zaradi prevelike sile približevanja glave do stika z medijem, kar je povzročalo vreznine in izdolbine na zapisni površini. Ta problem je bil dokončno rešen v letu 1980. Pomikanje glave je bilo izboljšano z dopolnitvijo pomikalnega in nameščevalnega mehanizma. Proizvajalci so uporabili različne metode za nameščanje glave (električne, mehanične in elektromehanične), tako da glava pristane mehko in brez odbijanja in ne povzroča vreznin in izdolbin površine.

Spremembe pri oblikovanju mehanizma glave so bile domiselne in bistvene. Razpoznani so bili problemi, ki so bistvenega pomena za zdržljivost medija pri pogojih dvostranske uporabe diska. Ukrivljenost (zaobljenost) glave, materiali in topokotnost robov, so bili upoštevani kot nova izhodišča pri oblikovanju. Tako je nastala nova proizvodnja glav, ki imajo visoko zdržljivost. Z izboljšanjem magnetne prevleke se je obraba disketne površine občutno zmanjšala.

Seveda niso vsi mehanizmi za nameščanje glave enako oblikovani. Nameščanje je krmiljeno tako, da se ne zmanjšujejo dosedanje zmogljivosti (npr. čas nameščanja). Nekateri proizvajalci so uvajali te izboljšave postopoma. Učinkovita je npr. elektromehanična metoda, ki namesti glavo zelo hitro, vendar upočasni postopek približevanja tik pred stikom z medijem; takšen način zahteva seveda določeno krmiljenje postopka nameščanja.

Masovna proizvodnja. Dvostranske diskovne enote imajo določene prednosti v primerjavi z enostranskimi, vendar mora biti njihova proizvodnja ekonomična in masovna ob visoki kakovosti izdelka. Proizvodnja dvostranskih diskovnih enot ni več cenena, saj mora upoštevati visoke, kritične tolerance, ki se približujejo tolerancam natančnih instrumentov. Cena naprave za OEM tržišče se tako ne more znižati pod $ 500.

Dodatek druge glave povzroči dodatno tolerančno razsežnost. Razporeditev radialnih zapisnih sledi postane kritična pri povečani gostoti zapisa zaradi izotropičnih lastnosti zapisnega medija. Bitni pomik, ki se pojavi kot podatkovna napaka, je odvisen od vzporednosti obeh glav (azimat). Pomičnost glave je posledica določene mehanične uravnoteženosti: prevelika sila povzroči obrabo medija, premajhna pa čitalne/pisalne napake. Tolerančne spremembe so pri mehanizmu z dvojno glavo odvisne od temperature, vlage okolice in položaja naprave in so lahko kritične glede na zapisni medij. Industrija se končno zaveda, da mora naprave z gibkimi diski proizvajati pod enakimi pogoji kot trdne diskovne pogone.

Za trdne diske velja pravilo, da morajo biti izpolnjeni določeni pogoji za stabilno obratovanje, da npr. pogonska enota in disk dosežeta stabilno temperaturo. Ta temperatura je zlasti važna pri justiranju enote. Tudi za gibke diske velja, da jim moramo vsaj 30 minut pustiti za ogrevanje in le pri teh pogojih bo disk obratoval zanesljivo tudi pri uporabniku. Da bi uporabnik lahko upošteval te važne parametre diskovne enote, mu vrste proizvajalcev z dokumentacijo dobavlja tudi diagrame s korekcijskimi faktorji. Ti faktorji zagotavljajo, da bo enota obratovala zanesljivo v območju plus/minus 30 % nastavitvenega odstopanja (upoštevaje izotropične lastnosti medija ter tolerance, odvisne od temperature in vlage).

Seveda je nenavadno, da vsi ti problemi niso bili upoštevani v dokaj dolgi zgodovini proizvodnje gibkih diskov, oziroma ni bila upoštevana dokumentacija proizvodnje trdnih diskov. Enostranske enote, ki so dominirale na tržišču v prejšnjih letih, so imele manjše tolerance kot dvostranske enote in zaradi tega so bile (v okviru teh ozkih toleranc) tudi dovolj zanesljive. Seveda pa je bil prenos izkušenj iz industrije trdnih diskovnih enot tudi počasen.

Proizvajalci so bili tako soočeni z vrsto notranjih proizvodnih izzivov, ki so se pojavili ob nameščanju glave in ustrezni obdelavi. Toda prva zahteva je, da se kakovost ne sme znižati z velikoserijsko proizvodnjo dvostranskih enot. Sprejemljivo razmerje napak za enote, ki se dobavljajo na teren (za servisiranje) je 5 do 8 %. Da bi se izravnale izgube, ki nastopijo pri transportu in njegovi obdelavi, morajo proizvajalci doseči 95 % izplen.

Vzdrževanje visoke kvalitete izdelka in velikoserijska proizvodnja zahtevata izkušeno osebje. Za prejšnjo proizvodnjo enot gibkih diskov je veljalo načelo, da so izkušnje potrebne le za preizkusno fazo. Pri novih enotah se zahteva natančno nameščanje praktično v vsakem delovnem položaju proizvodne linije. Delavci, ki enote sestavljajo, morajo imeti relativno visoko izkustveno ravnino, vsaj tako, kot se navadno zahteva pri proizvodnji tiskanih vezij.

Proizvajalec mora imeti tudi več kontrolnih mest na proizvodni liniji, da bi obdržal potrebno kvaliteto. Nekateri proizvajalci uporabljajo t.i. notranji procesni nadzor, ki je programiran del proizvodnega postopka. Proizvajalci lahko zmanjšajo napake tudi z uporabo zahtevnejših orodij in sestavljalskih pripomočkov.

Našteti faktorji, ki zagotavljajo kakovost proizvoda velikoserijske proizvodnje (delo z izkušnjami, notranji procesni nadzor in dodatni sestavljalski pripomočki) ter bistvene spremembe v proizvodnem postopku zaradi izdelave dvostranskih diskovnih enot so povzročile povečanje proizvodnih stroškov že v letu 1981. Zaradi porasta stroškov je postalo dodatno financiranje proizvajalcev življenskega pomena za ostalo mikroračunalniško industrijo.

Dolgoročni pogoji. Letos (1981) bodo kupci dvostranskih enot soočeni s problemi produkcijske sposobnosti proizvajalcev. V letu 1982 se bo povečal obseg proizvodnje in razmerje cena/zmogljivost enote bo postalo bistveno. V letu 1982 se bo povečalo zanimanje za večje količine podatkov. Leto 1983 bo prineslo poln razvoj dotedanjih zamisli. V letih 1984 in 1985 se bodo uporabniki dvostranskih diskovnih enot odločali za proizvode z večjimi zmogljivostmi in novimi tehnologijami.

Prihodnja tehnologija. Problemi razvoja dvostranskih diskovnih enot so bili v glavnem rešeni v letu 1980. Uvedenih je bilo 96 sledi na colo, vendar je ta gostota zanesljiva le pri 5,25 - colskih diskih. Velikoserijska proizvodnja bo močno odvisna od ponudbe glav (z mehanizmi), ko se bo dokončno uveljavil način nameščanja glave s trakom (prej tudi z vijakom - polžem).

Tudi 8-colski diski bodo dosegli večje gostote v letu 1982. Realna bo postala gostota nad 96 sledi na colo ter shranitev 2 M zlogov in več na eno ploščo.

Za doseganje še večje gostote shranjevanja bodo morali biti razviti novi zapisni mediji, ki bodo šli v korak s tehnologijo pogonske enote. Tako bo nastala 5 M-zložna diskovna enota. Uporaba 5,25-colskih diskov se bo povečala zlasti pri procesiranju teksta, v inteligenčnih terminalih in povsod tam, kjer se zahteva majhna velikost. Te enote bodo postale bolj kompaktne, zmanjšale se bo njihova električna poraba (do 30 %), njihovo obratovanje bo manj hrupno itd.

V letu 1982 se bodo pojavile tudi inteligenčne diskovne enote, ki bodo uporabljale lastne mikroprocesorje. Tako bo mogoče dodatno razbremeniti CPE mikroračunalnike, saj bo diskovna enota lahko imela lokalno upravljanje zbirk. Verjetno se bodo po letu 1981 pojavile na disku tudi tankoplastne prevleke.

Namesto še manjših diskovnih enot z diski 2,5 ali 3 cole, se bodo uporabljali mehurčni pomnilniki, saj bo njihova cena v teh primerih bolj ugodna

A.P. Železnikar

BARVNA GRAFIKA

Z željo, da premosti razkorak med malimi poslovnimi računalniki in računalniškimi konjičkarji, je Comodore predstavil svoj 300,- dolarski računalnik VIC-20, ki omogoča barvno grafiko. Računalnik vsebuje vmesnik za običajni barvni televizor, vendar je prikaz omejen na 23 vrstic po 22 znakov. Sistem uporablja Petov Basic, minimalna konfiguracija zahteva 5-K zlogov delovnega pomnilnika, je pa razširljiva na 32-K zlogov RAM/ROM pomnilnika. Sistem lahko naslavlja vhode za magnetno kaseto, RS-232-C vmesnik, serijski vmesnik za gibki disk ter vhod za razširitev pomnilnika.

ADA - jezik postaja aktualen

Po kakšnih petih letih razvoja je jezik ADA dozorel za uporabo. Veliko hiš, ki se ukvarja s programsko opremo, uvaja ADA jezik, saj omogoča uspešno implementacijo programskih aplikacij z dodajanjem standardnih paketov:

- Telesoftware Inc. nudi ADA prevajalnik za 16- bitni Microengine procesor (proizvaja ga Westeren Digital Corp.). Ta prevajalnik zahteva 128 K - zlogov delovnega pomnilnika, je pa podmnožica originalne ADA definicije. Firma načrtuje, da bo ta prevajalnik uporabljala še na drugih 16- bitnih mikro računalnikih.
- Sperry Univac razvija interpreter imenovan Bda, ki bo tekel na Univac-u 1100. S pomočjo tega interpreterja bo možno napisati Ada-prevajalnik v ADA jeziku ter ga prevesti v strojni jezik.
- Harris Corp. uvaja ADA jezik kot programski jezik za svoje znanstvene računalnike.

Razvojni sistem za 16- in 32- bitne mikroprocesorje

EXORmacs razvojni sistem omogoča razvojno delo na 16- bitnih in bodočih 32- bitnih aplikacijah. Bazična strojna oprema vsebuje M68000 mikroprocesor, MMU (memory management unit), razvojni modul (debugging module), 2 x 128 K- zložni dinamični delovni pomnilnik in univerzalni diskovni kontroler.

Diskovni kontroler omogoča priključitev dveh 96-M zložnih diskovnih pogonov in štirih pogonov gibkih diskov.

Programska oprema vsebuje VERSAdes multitasking operacijski sistem, urejevalnik teksta (screen oriented), strukturiran zbirnik, povezovalnik in nalagalnik, Pascal prevajalnik in simbolični prevajalnik iz strojnega jezika v mnemonično kodo.

Periferna oprema vsebuje 32-M zložni disk in EXORterm 155 prikazovalnik. Dodatni pomnilnik, prikazovalniki, večkanalni RS-232 vmesniki, pa omogočijo istočasno delo osmim uporabnikom. Cena osnovnega sistema je 40.000 US $.

PASCAL - 100 dvoprocesorski sistem

Pascal - 100 mikro računalniški sistem vsebuje Z-80 mikroprocesor. Sistem omogoča do 10-krat hitrejše izvajanje programov, pisanih v pascalu za procesor Z-80. Posebnost tega dvoprocesorskega sistema je kompatibilnost z obstoječo programsko opremo pisane za 8 bitne Z-80 ali 8080 sisteme oziroma sisteme, ki uporabljajo CP/M operacijski sistem, prav tako pa z novo programsko opremo kot je UCSD Pascal, ki zahteva že 16- bitne procesorje. Pascal - 100 stane 1500 US $.

32 - bitni mikroprocesor

V oktobru je Intel predstavil svoj 32- bitni mikroprocesor iAPX-432 kupcem. Procesor je zanimiv zaradi naslednjih lastnosti:

- sistemu se procesna moč povečuje z enostavnim dodajanjem procesnih enot, pri čemer ni potrebno spreminjati programske opreme,
- minimalna konfiguracija je po svojih sposobnostih primerljiva z VAX 11/780 in IBM 370/158,
- cena mikroprocesorja (tri VLSI integrirana vezja) je 3000 US $, v letu 1984 pa napovedujejo ceno 200 US $,
- funkcije operacijskega sistema so vsebovane v strojni opremi procesorja,
- mapiranje virtualnega pomnilnega prostora 2^{40} zlogov v fizični pomnilnik velikosti 2^{24} zlogov,
- za 32 bitno seštevanje porabi procesor 0,4 mikrosek, za 32 bitno množenje 5 mikrosek, za 80 bitno množenje s pomično vejico pa 27 mikrosek,
- strojna oprema je prirejena visokemu programskemu jeziku ADA ter omogoča standardno manipulacijo s podatkovnimi konstrukti jezika ADA. To so preprosti podatkovni tipi, kompleksni podatkovni elementi, vgnezdene procedure, medprocesorska sporočila, vzporedni procesi, procedure, ki poganjajo večje število procesorjev, itd.

D. Šalehar

Operacijski sistem Unix

Interaktivni multi uporabniški OS Unix so razvili v Bellovih laboratorijih z namenom, da bi dobili primerno orodje za razvoj programske opreme. Danes je najbolj množična verzija 6, instalirana na računalnikih tipa PDP-11.

Vsak interaktiven OS mora vsebovati interpreter ukazov in sistem, ki upravlja periferijo. Multi uporabniški sistem ima še algoritme za razvrščanje poslov in zaščito datotek.

Dober OS ima kratke, jedrnate in logične ukaze, ki se jih da lahko naučiti in hitro uporabljati; sistem za upravljanje periferije mora omogočati učinkovito uporabo prostora na diskih.

Poglejmo, kako so te stvari rešene v sistemu Unix.

Posebnosti Unix-a

Unix-ov interpreter ukazov se imenuje lupina (shell). Osnovni format ukaza za lupino je beseda (navadno dve črki - ime programa, ki izvrši ukaz). Eventuelne parametre podamo v ukazni vrstici. $ je znak pripravljenosti. Z operatorjem '<' in '>' lahko povemo, od kod naj dobi podatke oz. kam naj se izpišejo rezultati programa. Primer: $ prog > dat 1

Način delovanja lupine je dokaj kompleksen. Potem, ko je interpretirala ukaz in našla odgovarjajoči program na disku, starta poseben proces, imenovan vilice (fork). Le-ta kreira dve verziji lupine: roditeljski (parent) proces in otroški (child) proces. (Proces se v Unix-u imenuje vsak program, ki je v stanju pripravljenosti ali izvrševanja.) Zdaj izvrševalni program naloži kod za otroški proces in ga začne izvajati. Vse informacije, ki jih ima roditeljski proces, otroški proces podeduje. Medtem je roditeljski proces še vedno aktiven, čeprav včasih nima kaj početi, dokler otroški proces ni končan. Primer: ukaz, ki se zaključi z znakom '&', bo med izvrševanjem vrnil kontrolo nazaj lupini, tako da lahko le-ta paralelno obravnava drugi ukaz. Ukazi, ločeni z znakom ':', se obravnavajo zaporedno.

Sistem datotek je močan in enostaven: ima drevesasto strukturo. Koren je direktorij, končne veje pa so programi in podatkovne datoteke. Na vsakem mestu v drevesu lahko dodamo poddrevesa. Vsaka datoteka - vključno direktorij - vsebuje zaporedje znakov, konča pa se z znakom 'CTRL-D'. Ob danem trenutku zasede vključeni uporabnik določeno mesto v drevesu. Poti po grafu podamo v ukazni vrstici po vozlih ali relativno. Primer: $ dat > /dev/sup, kjer dev označuje vse vhodne in izhodne enote, sup pa pomeni serijski vrstični pisalnik.

Lupina lahko interpretira celo vrsto redundančnih oznak, npr. a?e pomeni abe, aae, itd; podobno [..] pomeni "na intervalu", npr. [a..c]xyz pomeni axyz, bxyz in cxyz. Znak '*' pomeni karkoli ali katerokoli zaporedje.

Datoteke so varovane z gesli. Vsak uporabnik se vključi v del datotečnega drevesa, za katerega poda geslo. Pri vsaki zahtevi za datoteko je potrebno povedati še za kakšen namen bomo datoteke rabili: r - branje iz, w - pisanje na, x - izvrševanje.

Načrtovalci Unix-a so si prizadevali, da bi bile operacije - gledano s strani uporabnika - kratke in učinkovite. Pomankljiv pa je sistem diagnostike napak - uporabnik bi naj bil inteligenten. Vso pozornost posveča Unix uporabniku, zato pa toliko bolj obremenjuje stroj: potrebuje velike količine pomnilnika in prostora na diskih, algoritem za razvrščanje poslov pa ni tako prefinjen, kot bi lahko bil.

Prilagodljivost

Noben univerzalen sistem kot je Unix, tako rekoč "s police", ne more biti idealen v posameznih posebnih primerih. Prava moč OS se kaže v njegovi prilagodljivosti, to je sposobnosti, da se z razvojem programske opreme razširja, krpa in gradi naprej. Unix je večinoma pisan v visokem programskem jeziku C (tudi produktu Bellovih laboratorijev), ki ima primerne podatkovne strukture za sistemsko programiranje in producira zelo učinkovit kod. Vsebuje veliko število programov, ki omogočajo dostop do sistemskih tabel in drugih informacij, ki so bistvene za sistemsko programiranje. Grajen je modularno in po metodi od zgoraj navzdol, zato je posamezne aktivnosti lahko prilagajati.

Unix lahko implementiramo enostavneje kot večino drugih OS, ker potrebujemo le dober C-prevajalnik, ki generira ustrezen strojni kod. Razen programov za upravljanje periferije je cel sistem lahko zgrajen kot navaden program.

Samo lupino lahko "programiramo" na ta način, da ji postrežemo z datoteko, ki sestoji iz zaporedja ukazov. Le-te bo lupina procesirala enega za drugim. Implementirane so strukture tipa if - then - else, case, for-do, while-do. Izhod iz enega programa lahko uporabimo kot vhod v paralelno izvršujoči se program, ne da bi morali postopek posebej sinhronizirati ali generirati začasno vmesno datoteko.

Ukazi

Standardni Unix je opremljen z običajnim vrstično orientiranim editorjem ed, z nekaj utrujajočimi lastnostmi, kot na primer to, da na začetku vrstice ne da znaka pripravljenosti. Dobi pa se tudi izboljšana verzija em in ekransko orientiran ded. Od prevajalnikov so na voljo: C-prevajalnik in prenosni C-prevajalnik, več verzij fortranskih (Fortran 77) in pescalskih prevajalnikov. Nadaljnje ponudbe so še: APL, Lisp, BCPL, ALGOL 68, POP-11. Ima tudi veliko izbiro razvojnih programov, kot so prevajalnik prevajalnikov, leksikalni analizator, sistem za spreminjanje in dopolnjevanje sekvence vezanih programov, možnosti postopnega izvrševanja programov, izpisovanje vsebine pomnilnika, posebej nalagalnik in povezovalnik objektnega koda, itd. Neobičajno veliko je programov za najrazličnejše manipulacije s teksti. Obstojajo programi za prenašanje sporočil od enega uporabnika do drugega in komuniciranje med več Unix sistemi, ki so na nek način povezani.

Sklep

Unix je primerno orodje za sistemske programerje, saj je pisan v visokem programskem jeziku in razpolaga z ukazi in sistemskimi klici, ki omogočajo dostop do sistemskih informacij in operacij. Programer, ki dela na zasedenem multi uporabniškem sistemu, pogreša učinkovite urejevalne rutine in dober sistem izračunavanja stroškov. Za začetnika bi bilo treba pripraviti še nekaj enostavnejših ukazov z bolj očitnimi imeni.

B. Blatnik

ZAPISNIK OBČNEGA ZBORA SLOVENSKEGA DRUŠTVA

INFORMATIKA, dne 19. 3. 1981

Na občnem zboru Slovenskega društva Informatika, ki je bil v Klubu delegatov, 19.3.1981 ob 11 uri, je bilo prisotnih 31 članov društva.

Ad 1) Otvoritev občnega zbora SDI

Občni zbor je otvoril predsednik društva, tov. Železnikar in pozdravil vse navzoče.

Ad 2) Izvolitev delovnega predsedstva in komisij zbora (verifikacijska, volilna)

V delovno predsedstvo so bili predlagani:
tov. Krisper, tov. Žerdin, tov. Železnikar.
Za zapisnikarja je bila predlagana tov. Seršen.
Za overovatelja zapisnika pa sta bila predlagana tov. Mekinda in tov. Banovec.
Delovno predsedstvo je bilo soglasno izvoljeno.

V volilno komisijo sta bila predlagana tov.Popović in tov. Bratko.
V verifikacijsko komisijo pa sta bila predlagana tov. Štrancar in tov. Selšek.

Predlagani komisiji sta bili potrjeni.

Ad 3) Poročila organov SDI (Izvršnega odbora (IO), nadzornega odbora (NO), disciplinskega odbora (DO) itn.)

3.1 Poročilo IO

O sedanjem delu je poročal predsednik društva, tov. Železnikar.

Razprava o poročilu IO

Vprašanje tov. Bratka: Zakaj ni bil organiziran simpozij Informatica '80?

3.2 Poročilo NO

Poročilo nadzornega odbora je podal tov. Faleskini.

3.3 Poročilo DO

O poročilu disciplinskega odbora je poročal tov. Žerdin.

Ker ni bilo konkretnih nalog, so člani DO aktivno delovali v IO društva, kot je bilo razvidno iz poročila tov. predsednika.

Vsa poročila so bila dana občnega zboru društva na glasovanje. En član se je glasovanja vzdržal, ostali člani društva so poročila soglasno sprejeli.

Vsa poročila so v prilogi zapisnika.

Ad 4) Razrešitev organov SDI

Soglasno je bila sprejeta razrešitev organov SDI.

Ad 5) Obravnava in sprejem kandidatnih list

Tov. predsednik je prebral kandidatno listo, ki smo jo napisali na tablo.

Ad 6) Kandidati IO:

1) Milan Mekinda - predsednik društva
2) Rado Faleskini - podpredsednik
3) Franc Žerdin - član
4) Vladimir Rajković - podpredsednik
5) Anton Gorup - podpredsednik
6) Borut Justin - član
7) Saša Divjak - član
8) Tomaž Seljak - član
9) Katja Seršen - tajnica SDI

Kandidati za NO:

1) Tomaž Herman - predsednik
2) Franc Mandelc
3) Jernej Virant

Kandidati za DO:

1) Ljubo Pipan
2) Marko Rogač
3) Andrej Jerman-Blažič - predsednik

Poročilo verifikacijske komisije

Verifikacijska komisija je ugotovila:

1. Zaradi nezadovoljive prisotnosti članov društva je bil občni zbor odložen za 1 uro (po statutu društva).
2. Po enournem čakanju je bilo prisotnih 31 članov, kar po statutu zadostuje za sklepčnost občnega zbora.

Sklep 1: Kandidatne liste so bile sprejete.

Sklep 2: Glasovanje je javno.

Sklep 3: Soglasno so bili izvoljeni novi organi društva: IO, NO, DO.

Dosedanji predsednik je vsem čestital.

Ad 7) Program dela za naslednje obdobje

je podal tov. Žerdin.

Z vso programsko usmeritvijo se je zbor društva strinjal.

Program dela je bil dan na glasovanje.

Sklep: Program dela in statut bomo objavili v časopisu Informatica.

Zbor društva je potrdil tudi programski in organizacijski odbor za simpozija Informatica '81 in '82.

V programski odbor je še dodatno predlagan tov. Bratko.

Predlog je bil soglasno sprejet.

Ad 8) Spremembe statuta SDI

je utemeljil tov. Krisper in poudaril, da izhajajo iz poročil in programa dela. Nanašajo pa se na:

- sedež društva
- mandate
- vsebinske spremembe
- organizacijske spremembe
- institut častnega predsednika
- volilni občni zbor je vsaki 2 leti

Sklep: Spremembe statuta so bile na občnem zboru soglasno sprejete.

Tov. Železnikar je bil za dolgoletno delo v SDI predlagan in izvoljen za častnega predsednika društva.

Tov. Železnikar se je za izkazano častno funkcijo zahvalil ter povedal nekaj o prehojeni poti Zveznega društva in SDI od leta 1965 dalje.

Ad 9) Razno

Tov. Banovec je opozoril, da vsi simpoziji informatike po republikah nosijo enako ime: Informatika.

Ali se lahko programi teh simpozijev razdelijo oz. racionalizirajo?

Novo izvoljeni predsednik društva, tov. Mekinda se je zahvalil za zaupanje. SDI mora povečevati vpliv na vsa področja računalništva in informatike ter razširiti članstvo.

Akcija včlanjevanja traja še naprej.

Predlog članarine:

1. Naročnina časopisa Informatica je tudi članarina
2. Članarina za študente
3. Člani, ki ne želijo časopisa

Sprejeta je bila nova naročnina in članarina:

1. Za redne člane SDI	200,00 din
2. Za študente (s časopisom Informatica)	100,00 din
3. Za študente brez časopisa Informatica)	50,00 din

Občni zbor je bil zaključen ob 13,15 uri.

Zapisala: Katja Seršen

Overovatelja zapisnika: Tomaž Banovec, Milan Mekinda, l.r.

Predsednik del. predsed. zbora SDI: A. P. Železnikar

RAZISKAVE RAČUNALNIŠTVA IN AVTOMATIKE V SKUPNEM PROGRAMU RSS ZA LETO 1981

Dne 26. februarja 1981 se je v prostorih Instituta "Jožef Stefan" sešel na 1. sejo Programski svet za računalništvo in avtomatiko skupnega programa Raziskovalne skupnosti Slovenije zaradi razprave o predlogih raziskav za leto 1981. Seje so se poleg članov programskega sveta (dodatek A) udeležili tudi vodje usmerjenih raziskovalnih programov ter koordinatorji raziskovalnih sklopov pri raziskovalnih organizacijah. Seja pod vodstvom predsednika Programskega sveta se je pričela ob 9. uri, z delom pa smo končali ob 12.30.

Dnevni red seje je obsegal:

- *splošno informacijo o organiziranosti raziskovalne dejavnosti*
- *osnutek izhodišč za delovanje programskih svetov*
- *metodološka izhodišča za oblikovanje in ocenjevanje predlogov URP in RP*
- *razpravo o predlogih URP za obdobje 1981-85*
- *razpravo in sprejem predlogov raziskav za leto 1981*

Organizacijska in metodološka izhodišča:

Predsednik sveta je v uvodu povzel temeljne naloge Programskega sveta pri oblikovanju raziskovalnih programov na področju računalništva in avtomatike, pri njihovem usklajevanju s gospodarskimi in družbenimi potrebami, pri nadzoru nad izvajanjem raziskav ter pri seznanjanju strokovne in širše javnosti s raziskovalnimi dosežki. Pozval je člane sveta k zavzetemu sodelovanju pri pripravljanju ustreznih vsebinskih in organizacijskih osnov za izvajanje omenjenih nalog. Opozoril je tudi na to, da je temu Programskemu svetu poverjena skrb za sklop URP-ov, ki je po obsegu raziskav med najmočnejšimi, po vsebini pa temeljnega pomena za naš gospodarski in družbeni razvoj ter za skladen razvoj mnogih znanstvenih disciplin. Zato bo delo v tem svetu povezano s visoko stopnjo strokovne in družbene odgovornosti.

Tajnik tehničnih ved skupnega programa RSS tov. Peternel je spregovoril o zakonu o raziskovalni dejavnosti ter vlogi programskega sveta. Predsednik sveta je predstavil osnutek izhodišč za delovanje programskih svetov in pozval člane k pripravi predlogov za delovanje tega programskega sveta. Svet je zadolžil delovno skupino v sestavi Bratko, Faleskini, Kac, Strmčnik, Suhelj, Špegel in Viličič, da izdela predlog delovnega načrta programskega sveta za leto 1981 in ga predloži v obravnavo na naslednji seji sveta.

Programski svet bo svoje naloge na področju spremljanja in usmerjanja raziskav izvajal na osnovi periodičnih poročil o napredku na raziskavah. V ta namen bodo vodje usmerjenih raziskovalnih programov na področju računalništva in avtomatike s četrtletnimi poročili pismeno obveščali programski svet o napredku na raziskavah ter o tem osebno poročali na sejah programskega sveta. Predsednik sveta je dolžan zagotoviti, da bodo člani ta poročila prejeli pravočasno. Svet priporoča vodjem URP-ov, da si zagotovijo potrebno pomoč s strani koordinatorjev programskih sklopov ter tromesečna poročila obravnavajo na sestankih programskih odborov URP-ov (oziroma na sestankih raziskovalcev v primeru manj obsežnih URP-ov).

PREDLOGI SREDNJEROČNIH RAZISKOVALNIH PROGRAMOV

Programski svet je obravnaval predloge srednjeročnih programov raziskav za naslednje usmerjene raziskovalne programe:

- *Temeljne raziskave računalniške tehnike*
- *Računalniške komunikacije*
- *Temeljne raziskave programske opreme*
- *Metode umetne inteligence*
- *Računalniška avtomatizacija sistemov in procesov*
- *Robotika in manipulatorji*
- *Modeliranje, identifikacija in avtomatsko razpoznavanje*

V razpravi o teh predlogih, v kateri so med drugim sodelovali Stiglic, Kralj, Rutar, Jeglič, Banovec in Virant smo ugotovili:

- *da so predloženi srednjeročni programi raziskav tehtni in dobro utemeljeni, da izhajajo iz potreb združenega dela in jih programski svet podpira;*
- *da pa mora programski svet takoj sprožiti intenziven proces vsebinskega in organizacijskega usklajevanja teh programov in njihovega povezovanja s srednjeročnimi plani uporabnikov za obdobje 1982-1985;*
- *da bomo pri usklajevanju dali poudarek skladnosti razvoja posameznih področij v okviru računalništva in avtomatike ter povezovanju temeljnih raziskav s aplikativnimi na osnovi dogovorjenih prednostnih nalog in ciljev;*
- *da bomo srednjeročne plane raziskav vskladili tako v republiškem kot v širšem jugoslovanskem okviru ter zmanjšali razdrobljenost raziskav;*
- *da bomo srednjeročne programe dopolnjevali tako, da bo združeno delo pripravljeno s sovlaganji omogočati uresničitev večjih konkretnih raziskovalnih projektov;*
- *da bomo poskrbeli za sprotno informiranje strokovne in širše javnosti o raziskovalnih programih na področju računalništva in avtomatike ter o njihovem poteku;*
- *da bomo organizirali skupno računalniško dokumentacijo raziskav na področju računalništva in avtomatike.*

PROGRAM RAZISKAV V RAČUNALNIŠTVU IN AVTOMATIKI ZA LETO 1981

Vodje URP-ov ter vodje programskih sklopov po raziskovalnih organizacijah - izvajalkah oziroma nosilci raziskav so predstavili programe raziskav za leto 1981. Programski svet je ugotovil, da so predloženi programi izdelani v skladu s predpisano metodologijo in izkazujejo za letošnje pogoje dela zadovoljivo stopnjo kvalitete, medsebojne usklajenosti in usklajenosti s potrebami združenega dela. Zato jih je svet predlagal v sprejem v skupščini RSS s pridržkom, da vodje nekaterih URP-ov odpravijo ugotovljene manjše metodološke in vsebinske pomanjkljivosti.

1. Usmerjeni raziskovalni program

TEMELJNE RAZISKAVE RAČUNALNIŠKE TEHNIKE

Koordinator: Fakul.za elektrotehniko, Ljubljana
Vodja programa: prof.dr.J.Virant

1.a. Programski sklop IJS, Ljubljana
Vodja sklopa: dr. P. Kolbezen

1.a.1. RAZISKAVE IN ŠTUDIJA MASOVNIH POMNILNIKOV S POSEBNIM POUDARKOM NA UVAJANJU NOVIH TEHNOLOGIJ

Nosilec: dr. P. Kolbezen

- študijo teoretskih osnov in principov mehurčnih pomnilnikov ter fizikalno tehnoloških postopkov s posebnim poudarkom na perspektive v tehnologiji;
- študijo mehurčnih pomnilniških elementov in sistemov ter stanja na tržišču;
- raziskave kontrolne logike in ustrezne programske opreme masovnih pomnilnikov. Primerjave med posameznimi tehnologijami (med različnimi mediji) gibki disk - mehurčni pomnilniki in kontrolna logika;
- osnovne raziskave komunikacije mikroračunalnik - masovni pomnilnik s posebnim poudarkom na mikroračunalniške aplikativne sisteme, ki jih že ali jih bo v najbližji prihodnosti uporabljalo naše industrijsko okolje.

1.a.2. PROBLEMI RAZVOJA MIKRORAČUNALNIŠKIH APLIKATIVNIH SISTEMOV

Nosilec: dr. Peter Kolbezen

- metodologija razvojne materialne in programske opreme za integrirane računalnike;
- raziskave in razvoj učinkovitega emulatorja za domač integrirani mikroračunalnik ter razvoj materialne in programske opreme;
- problemi načrtovanja mikroračunalniških aplikativnih programov na večjem računalniku ter razvoj križnega zbirnika za domači integrirani mikroračunalnik, ki bo zgrajen na osnovi makro definicij.

1.a.3. VEČ-ZLOGOVNI MIKRORAČUNALNIKI

Nosilec: dr. P. Kolbezen

- komparativni študij med posameznimi tehnologijami in organizacijami s posebnim poudarkom na tehnologiji bitnih rezin (lastnosti, uporabnost, trendi) ter standardizacijo organizacij in tehnologij;
- konceptualni razvoj tehnološke banke podatkov za računalniško tehnologijo ter definicija podatkovnih struktur ter postopkov za gradnjo, uporabo in ažuriranje banke podatkov;
- eksperimentalna implementacija tehnološke banke podatkov.

1.a.4. ASINHRONI MULTIMIKROPROCESORSKI SISTEMI

Nosilec: dr. P. Kolbezen

- raziskava problemov komuniciranja med mikroprocesorskimi in mikroračunalniškimi sistemi, ki so sestavljeni iz komponent različnih lastnosti in arhitektur (8, 16-bitni, mikro, mini);
- reševanje problemov učinkovitosti pretoka podatkov na že izdelanem modelu MPS ob uporabi različnih grafičnih metod in struktur odločitvenih dreves;
- preučitev konfliktnih situacij pri naseganju skupne pomnilniške banke.

1.a.5. DIAGNOSTIKA MIKRORAČUNALNIKOV

Nosilec: dr. Janez Korenini

- odkrivanje napak na osnovi časovnih performans sistema;
- raziskave uporabe diagnostičnih števcev za primere, ko računalnik komunicira s računalniškimi perifernimi napravami, drugimi računalniki ali s objekti, ki ga krmili;
- raziskava problema obratne procedure s stališča aparaturne opreme (shranjevanje informacije o mestu programa, v katerem je bilo izvajanje prekinjeno) kakor tudi s stališča programske opreme (analiza informacije, ki jo moramo ob prekinitvi izvajanja programa ohraniti brez škodljivih posledic posega operaterja;
- analiza programskih grafov na nivoju paralelnega izvajanja procesov na nivoju operacijskega sistema ter razvoj metod za reduciranja grafov tega nivoja;
- analiza možnosti za shranjevanja tekstov informacije o stanju izvajanja programov (izbor ustreznih pomnilniških medijev);
- analiza potrebnih informacij o statusu sistema in mestu izvajanja programa (tip informacije - adresa, vrednosti različnih parametrov, obseg informacije, format, idr.)

1.a.6. *POSTOPKI ZA POVEČEVANJE ZANESLJIVOSTI DIGITALNIH SISTEMOV NA OSNOVI DOPUŠČANJA NAPAK*

Nosilec: dr. R. Murn

- *študija in formalizacija osnovnih lastnosti ter analiza učinkovitosti korekcijskih kodov;*
- *raziskava detekcijskih kodov s separacijskimi lastnostmi ter z možnostjo ugotavljanja enojnih in enoznačnih kodov;*

1.b. *Programski sklop FE, Ljubljana*

NAČRTOVANJE REZINASTIH PROCESORJEV IN KRMILNIKOV

Nosilec: prof. dr. J. Virant

- *metodologija rezinskega načrtovanja;*
- *primerjava obstoječih družin rezin s stališča načrtovanja CPU-enot in krmilnikov;*
- *izdelava neposrednih postopkov načrtovanja:*
- *primer, ki kaže uporabo metodologije in postopkov načrtovanja;*

1.c. *Programski sklop VTŠ-VTO Elektrotehnika Maribor*

MIKROPROGRAMIRANA ARHITEKTURA ZA IMPLEMENTACIJO VISOKEGA PROGRAMSKEGA JEZIKA

Nosilec: mag. V. Žumer

- *pregled jezikov in tehnik za izdelavo mikroprogramov;*
- *izdelava učinkovite metodologije za razvoj in popravljanje mikroprogramov;*
- *raziskava implementacije algoritmov v visokih programskih jezikih z mikroprogrami in preučitev vpliva teh algoritmov na mikroprogramirane arhitekture;*
- *predlog za take algoritme v visokem programskem jeziku, ki jih zahteva dobra mikroprogramirana arhitektura.*

2. *Usmerjeni raziskovalni program*

RAČUNALNIŠKE KOMUNIKACIJE
==========

Koordinator: Institut J. Stefan, Ljubljana
Vodja programa: dr. T. Kalin

2.a. *Programski sklop IJS*

POVEZOVANJE LOKALNIH MREŽ

Nosilec: dr. T. Kalin

- *raziskava problema povezovanja lokalnih mrež v sodelovanju z institucijami v okviru COST 11;*
- *razširitev mrežnega kontrolnega jezika na probleme lokalne mreže;*
- *izdelava predloga protokola za lokalne mreže, ki bo omogočala efektivno povezavo.*

2.b. *Programski sklop FE, Ljubljana*

NOVI ALGORITMI ZA SINTEZO DIGITALNIH FILTROV IN NJIHOVA UPORABA V RAČUNALNIŠKIH KOMUNIKACIJAH

Nosilec: dr. D. Kodek

2.c. *Programski sklop VTŠ-Elektrotehnika, Maribor*

LOKALNE RAČUNALNIŠKE IN TERMINALSKE MREŽE

Nosilec: mag. B. Horvat

- *teoretična obdelava najnovejših dosežkov na področju lokalnih računalniških mrež glede s stališča topologije in komuniciranja med računalnikom in terminali;*
- *raziskava osnovnih arhitekturnih pristopov k računalniški mreži (SNA, DECNET, itd.) in uporaba spoznanj za snovanje lastne lokalne mreže.*

3. *Usmerjeni raziskovalni program*

TEMELJNE RAZISKAVE ZA PROGRAMSKO OPREMO
==========

Koordinator: Institut J. Stefan, Ljubljana
Vodja programa: dr. M. Špegel

3.a. *Programski sklop IJS, Ljubljana*
Vodja sklopa: dr. M. Špegel

3.a.1. *MODELI DISTRIBUIRANEGA PROCESIRANJA*

Nosilec: dr. I. Bratko

- *študij formalnih modelov za distribuirano oziroma paralelno procesiranje;*
- *programski jeziki za distribuirano oziroma paralelno procesiranje;*
- *interpreter za jezik za paralelno programiranje na sekvenčnem računalniku;*

3.a.2. *ZASNOVA ZA STRUKTURIRANJE OPERACIJSKIH SISTEMOV*

Nosilec: mag. M. Exel

- *primerjalna študija novih sistemskih programskih jezikov (predvsem jezika ADA) z obstoječimi sistemskimi programskimi jeziki (npr. Modula, Concurrent, Pascal);*
- *organizacija jedra operacijskega sistema, raziskava primitivnih operacij za signale, semaforje in razvrščanje ter implementacija monitorjev;*

3.a.3. *SIMBOLIČNA ALGEBRA ZA MATEMATIČNE PROBLEME FIZIKE, KEMIJE IN TEHNIKE*

Nosilec: prof. dr. M. V. Mihajlović

- *pregled uporabnosti obstoječih programskih paketov za simbolično algebro na računalnikih v Sloveniji in ovrednotenje teh paketov;*
- *razvoj računalniških postopkov za simbolično računanje z matrikami;*

3.a.4. *RAČUNALNIŠKO NAČRTOVANJE MIKROPROGRAMIRANIH RAČUNALNIKOV*

Nosilec: R. Reinhardt

- *definicija jezika za opisovanje podatkovnih poti mikroprogramiranih procesorjev in avtomatično generiranje simulatorja podatkovnih poti;*
- *razvoj jedra mikrosbirnika in simuliranih modulov ter njihova implementacija na računalniku;*
- *razvoj eksperimentalnega prevajalnika za prevajanje iz omejenega visokega programskega jezika v mikroprogram predpisanega mikroprogramiranega mikroprocesorja;*

3.a.5. *PROGRAMSKA ZASNOVA CENENEGA GRAFIČNEGA TERMINALA*

Nosilec: M. Martinec, dipl.ing.

- *zasnova večplastnega programskega sistema za računalniško grafiko na cenenem grafičnem terminalu;*
- *zasnova programskega jezika za implementacijo grafičnih postopkovnih in interaktivnih sistemov;*
- *zasnova interaktivnega urejevalnika za opis in sintezo računalniških vezij in tiskanin;*

3.a.6. *PROGRAMIRANJE TISKARSKEGA STAVLJENJA*

Nosilec: dr. M. Špegel

- *raziskava konceptov, metod in postopkov pri računalniško podprtem stavljenju v slovenskih tiskarnah;*
- *zasnova abstraktne osvetljene enote za stavljanje gladkega in zahtevnega stavka;*
- *razvoj uporabniškega programskega jezika za opisovanje stavnih postopkov na abstraktni osvetljeni enoti;*

3.a.7. *RAČUNALNIŠKE METODE RAZISKAVE BIOSISTEMOV*

Nosilec: mag.B. Jerman-Blažič

- *razvoj in izdelava notacijskega sistema za računalniško zapisovanje, shranjevanje in iskanje skupnih molekulskih sistemov;*
- *razvoj in implementacija algoritmov za enumeracijo resonančnih oblik molekulskih sistemov, ki bo obsegalo generiranje nadomestnih grafov in identitete molekulskih sistemov ter oceno časovne in prostorske kompleksnosti teh algoritmov;*
- *izdelava primerjalne analize topoloških indeksov (ASC, TRE, indeks razvejanosti grafa);*

3.a.8. *METODE ZA MATEMATIČNO IN STATISTIČNO OBDELAVO PODATKOV*

Nosilec: Z. Radalj, dipl. mat.

- *razvoj metod za multiregresionalno aproksimacijo eksperimentalnih podatkov, definiranje kriterijev za vključevanje oz. izključevanje parametrov in raziskava odvisnosti končne rešitve od začetnega stanja;*
- *analiza in napovedovanje časovnih vrst ter preučitev možnosti za zmanjšanje pomnilniške zahtevnosti teh metod (predvsem metod ARIMA);*
- *razvoj metode za računalniško grupiranje časovnih vrst na osnovi avtokorelacije, korelacije z odlogi in faktorske analize;*
- *matematično opisovanje problemov optimalnega upravljanja ekonomskih procesov in razvoj računalniških postopkov za reševanje teh problemov;*

3.b. *Programski sklop FE, Ljubljana*
Vodja sklopa: doc. dr. B. Vilfan

3.b.1. *REGULARNI ALGORITMI ZA MNOŽENJE MATRIK*

Nosilec: B. Vilfan

3.b.2. *NAČRTOVANJE KONCEPTUALNE SHEME PODATKOVNE BAZE*

Nosilec: mag.T. Mohorič

3.c. *Programski sklop VTŠ- Elektrotehnika Maribor*

FUNKCIONALNOST MIKRORAČUNALNIŠKEGA MONITORJA V REALNEM ČASU

Nosilec: M.Colnarič

- *izgradnja master-slave mikroračunalniške strukture iz mikroračunalnikov ISKRA DATA 1680;*
- *vgraditev monitorja v to strukturo:*
- *izdelava simulatorjev realnega okolja (predvsem na zahteve železniškega omrežja n elektrodistribucije);*
- *določanje ekstremnih pogojev in izjemnih dogodkov ter preučevanje obnašanja monitorja ob teh dogodkih;*
- *korekture monitorja v kolikor se bi pokazalo potrebno;*
- *izdelava priročnika za aplikacijskega programerja (programer, ki bo realiziral vgraditev v namensko mikroračunalniško strukturo);*

3.d. *Programski sklop VTŠ-Strojništvo Maribor*

RAZVOJ OSNOVNE PROGRAMSKE OPREME ZA RAČUNALNIŠKO KONSTRUIRANJE SKLEPOV, KONSTRUKCIJ IN NAPRAV V STROJNIŠTVU

Nosilec raziskave: dr. A. Jezernik

- *dokončanje izdelave osnovne programske opreme za računalniško konstruiranje in dela uporabniške programske opreme, ki bo omogočala računalniško konstruiranje elementov konstrukcij na trdnost v strojništvu in graditeljstvu ter računalniško konzipiranje elementov konstrukcij in naprav z interaktivnim menujskim načinom dela;*

4. *Usmerjeni raziskovalni program*

METODE UMETNE INTELIGENCE

Koordinator: Institut J. Stefan, Ljubljana
Vodja programa: dr. I. Brako

4.a. Programski sklop IJS
Vodja sklopa: dr. I. Bratko

4.a.1. PROGRAMSKA OPREMA ZA SISTEME UMETNE INTELIGENCE

Nosilec: dr. I. Bratko

- *instalacija kompilatorja za programski jezik Prolog na računalnku DEC-1-0 v RCU, Univerza Edvarda Karedelja v Ljubljani;*
- *prilagoditev in instalacija prevajalnika za Prolog na računalniku PDP-1-1;*
- *vzdrževanje programskih sistemov za Lisp, Prolog in Utlog na računalnikih CDC CYBER os. DEC-1-0 in PDP-1-1 ter podpora za širjenje kroga uporabnikov teh jezikov;*

4.a.2. EKSPERTNI SISTEMI

Nosilec: dr. I. Bratko

- *razvoj programskega sistema in jezika AL3 za nepostopkovno programiranje (ta jezik ne bo omejen, tako kot je Prolog, samo na Hornovo obliko logičnih stavkov);*
- *mehanizmi za pojasnjevanje rešitev: naloga teh mehanizmov je generiranje takih pojasnil, ki so za uporabnika lahko razumljiva in mu omogočajo vpogled v delovanje sistema oz. modela ter olajšujejo popravljanje modelov;*
- *prototip splošnega ekspertnega sistema, temelječega na interpretaciji semantičnih mrež;*

4.a.3. EKSPERTNI SISTEM ZA VARJENJE

Nosilec: dr. M. Špegel

- *razvoj programskega sistema za geometrično modeliranje varjencev in za avtomatično programiranje varilnih poti;*
- *razvoj in računalniška implementacija baze postopkov in strategij za računalniško vodenje varilnega avtomata s tremi prostostnimi stopnjami ter mikroračunalniškim mehanskim tipalom odmikov varilne šobe od načrtovane varilne poti;*
- *analiza zmožnosti razvitega ekspertnega sistema pri reševanju konkretnih problemov pri varjenju cevastih varjencev;*

4.a.4. KOMUNICIRANJE Z RAČUNALNIKOM V SLOVENSKEM JEZIKU

Nosilec: mag. P. Tancig

- *modeliranje naravnega jezika z uporabo sistemov formalne logike;*
- *implementacija ATN analizatorja in sintetizatorja teksta v naravnem jeziku;*
- *eksperimentalni integrirani sistem za računalniško razumevanje slovenščine kot primer komunikacije v naravnem jeziku z bazo podatkov;*

4.b. Programski sklop FE, Ljubljana

AVTOMATSKO UČENJE

Nosilec: dr. I. Bratko

- *implementacija in eksperiment s sistemom za avtomatsko učenje znovi na osnovi primerov in protiprimerov v medicinski diagnostiki;*
- *razvoj metode učenja na osnovi spoznavanja odnosov med elementarnimi objekti, pri čemer bo model učenja ustrezal problemu konstruiranja in adaptacije digitalnih računalniških struktur;*

5. *Usmerjeni raziskovalni program*

ROBOTIKA IN MANIPULATORJI

Koordinator: Institut J. Stefan, Ljubljana
Vodja programa: mgr. P. Oblak

5.a. Programski sklop IJS
Vodja sklopa: mgr. P. Oblak

- *študij in razvoj dinamičnega modela industrijskega robota osnovi Hamiltonovih enačb dinamike;*
- *razvoj metodologije določanja optimalnih parametrov mehanske konfiguracije robota s pomočjo sinteze optimalnega modela;*
- *razvoj ustreznega simulacijskega programskega paketa;*
- *razvoj matematičnega konsepta za vodenje industrijskega robota od točke do točke in implementacija na mikroračunalniku;*
- *študij kinematike industrijskih robotov in delovnega okolja;*

5.b. Programski sklop FE, Ljubljana

DINAMIKA IN GENERIRANJE DELOVNIH KRIVULJ MANIPULATORJEV

Nosilec: prof. dr. A. Kralj

- *kompletiranje in zagon eksperimentalnega manipulatorja z električnim pogonom in 3-5 prostostnimi stopnjami in odprtozančno upravljanje;*
- *raziskava računalniških metod za opis delovnih prostorov;*
- *študija in sinteza metod za računalniško analizo in opis kinematike;*
- *sestava osnovne materialne opreme za generiranje delovnih krivulj;*
- *študij problemov sestave delovnih krivulj za precizno manipuliranje majhnih bremen v industriji elementov in za naloge asembliranja sestavov;*

5.c. Programski sklop VTŠ-Strojništvo Maribor

IZDELAVA IN KRMILJENJE MIKROPROCESORSKEGA DELA RISALNIKA

Nosilca: dr. P. Leš in doc. dr. A. Jesernik

- *izdelava ravninskega risalnika velikosti A0;*
- *izdelava elektronskega krmilja;*

- *vklajevanje delovanja posameznih komponent;*
- *izdelava osnovnih programov;*
- *razvoj zasnove in izdelava mikroprocesorske grafične kartice z grafičnim pomnilnikom;*

6. Usmerjeni raziskovalni program

RAČUNALNIŠKA AVTOMATIZACIJA SISTEMOV IN PROCESOV

Koordinator: Institut J. Stefan, Ljubljana
Vodja programa: dr. Stanko Strmčnik

6.a. Programski sklop IJS, Ljubljana
Nosilec: dr. Stanko Strmčnik

6.a.1. DOGRADITEV LABORATORIJSKE MODELNE NAPRAVE

- *izpopolnitev tehnološkega dela modelne naprave (model multivariabilnega sistema s petimi vhodi in štirimi izhodi);*
- *izdelava komandne plošče za ročno upravljanje;*
- *izbor, nabava in montaža senzorjev in aktuatorjev;*
- *razvoj in izdelava vmesnikov;*
- *razvoj funkcionalnega matematičnega modela naprave;*
- *priključitev mikroračunalnika na modelno napravo;*
- *izdelava programske opreme za zbiranje podatkov;*
- *izvajanje prvih testnih meritev;*

6.a.2. SINTEZA OPTIMALNEGA VODENJA Z VEČNIVOJSKIMI PRISTOPI

- *primerjava različnih odprtozančnih večnivojskih algoritmov;*
- *selekcija najboljših algoritmov;*
- *preverjanje uporabe odprtozančnih algoritmov z omejitvami za načrtovanje zaprtozančnih optimalnih regulatorjev;*
- *primerjava večnivojskega pristopa k sintezi z enonivojskim pristopom;*
- *izdelava uporabniško orientiranih računalniških programov;*

6.a.3. NEGRADIENTNE METODE OPTIMIRANJA NA MALIH RAČUNALNIKIH

- *metoda direktnega iskanja;*
- *metoda fleksibilnih poliedrov;*
- *omejitve in metoda fleksibilnih poliedrov;*
- *uporaba metode fleksibilnih poliedrov pri optimalnem upravljanju;*

6.b. Programski sklop ISKRA AVTOMATIKA, TOZD RAZVOJNI INŠTITUT Ljubljana

6.b.1. DISTRIBUIRANA PODATKOVNA BAZA V SISTEMIH DALJINSKEGA VODENJA

Nosilec: mgr. N. Panič

- *študij problemov distribuiranih podatkovnih baz in vloga le-teh v sodobnih 'mini-mikro' računalniških sistemih vodenja in nadzora;*
- *dekompozicija podatkovne baze v mikroračunalniškem sistemu vodenja in nadzora in določitev obsega teh funkcij podatkovne baze na posameznem nivoju;*
- *izbira organizacije podatkovne baze na nivoju sistema, objektov (postaj), podatkovnih blokov in podatkovnih točk z opredelitvijo potrebnih atributov (podatkovnih elementov) v sistemu TI-30 na osnovi izbranega podatkovnega modela;*
- *zasnova in realizacija programskega sistema za nadzor in dostop do podatkov v podatkovni bazi v mikroračunalniškem sistemu vodenja (aplikacija v sistemu TI-30);*
- *preizkus izbranih in realiziranih konceptov distribuirane podatkovne baze na laboratorijskem modelu, ki bo sestavljen iz ene centralne in končne postaje mikroračunalniškega sistema vodenja TI-30 v preizkusni povezavi z miniračunalnikom;*

6.b.2. RAČUNALNIŠKO KRMILJENJE IN REGULIRANJE INDUSTRIJSKIH PROCESOV

Nosilec: Branko Robave, dipl. ing.

- *mikroprocesorski krmilni sistem (analiza obstoječih rešitev v svetu, izbira koncepta sistema, detajliranje koncepta, izdelava blokovnih in detaljnih shem vhodno-izhodnih modulov, izdelava modelov in testiranje, analiza testiranja in zaključki);*
- *elektromotorni regulirani pogoni (analiza metod za realizacijo sistema, matematična opredelitev sistema, osvojitev koncepta regulacije, zgraditev fizikalnega modela, meritve in preizkusi na modelu, analiza resultatov in zaključki);*
- *digitalno tehtanje in doziranje (preučevanje znanih rešitev in idejna zasnova, realizacija modela, meritve in preizkusi na modelu, verifikacija resultatov, zaključki);*
- *digitalne regulacije v napajalnih napravah III. faza (idejna postavitev modela, matematična verifikacija, izdelava modela, meritve na modelu, verifikacija in zaključki);*

6.c. Programski sklop VTŠ-ELEKTROTEHNIKA Maribor

Vodja sklopa: doc. dr. K. Jezernik

6.c.1. RAČUNALNIŠKO VODENJE PROCESOV

Nosilec: dr. Dalij Džonlagič

- *raziskave modelov regulacijskih sistemov v industrijskih procesih;*
- *določitev kriterijev za izbiro opreme za vodenje in upravljanje teh procesov;*
- *začetek gradnje in opremljanja računalniškega sistema;*

6.c.2. UPRAVLJANJE IN SIMULACIJA VELIKIH SISTEMOV

Nosilec: dr. Karel Jezernik

- razvoj mikroračunalniškega sistema s paralelno aritmetično enoto za obdelavo podatkov in simulacijo s plavajočo vejico;
- obdelava simulacije sinhronskega stroja v tranzientnem delovanju, regulacije frekvence in vzbujanja z mikroračunalnikom;
- dopolnitev procesorja z D/A in A/D enotami;
- obdelava problematike izmenjave energije med postroji na paralelni strukturi več procesorjev kot prispevek k decentralizaciji uporabe velikih sistemov;
- nadaljevanje gradnje paralelnega procesorja na zasnovi vmesnika v povezavi z nadrejenim računalnikom;

6.d. Programski sklop FE, Ljubljana

TEORIJA IN OSNOVNE RAZISKAVE PROGRAMABILNE PROCESNE AVTOMATIKE

6.d.1. RAČUNALNIŠKO VODENJE DINAMIČNIH PROCESOV

Nosilec: prof. dr. France Bremšak

- sinteza multivariabilnih regulatorjev na osnovi premikanja polov;
- sinteza regulatorjev z razstavljanjem multivariabilnih sistemov;
- adaptivno sprotno računalniško vodenje sistemov in sinteza diskretnih modelno referenčnih adaptivnih sistemov in parametersko adaptivnih sistemov;
- razvoj ustrezne interaktivno orientirane računalniško programske opreme in simulacija načrtovanih sistemov s pomočjo povezave analogno-hibridnega računalnika EAI 680 z digitalnim računalnikom PDP 11/34;

6.d.2. TEORIJA IN OSNOVNE RAZISKAVE PROGRAMSKE PROCESNE AVTOMATIKE

Nosilec: prof. dr. Peter Šuhel

- študija z analizo in sintezo enoprocesorskega krmiljenja programabilne procesne avtomatike

7. Usmerjeni raziskovalni program

MODELIRANJE, IDENTIFIKACIJA IN RAZPOZNAVANJE PODATKOV

Koordinator: Fakul.za elektrotehniko, Ljubljana
Vodja programa: prof. dr. L Gyergyek

7.a. Programski sklop IJS, Ljubljana
Vodja sklopa: dr. Stanko Strmčnik

7.a.1. IDENTIFIKACIJA Z REFERENČNIM MODELOM

- pregled teoretičnih osnov identifikacije multivariabilnih sistemov z referenčnim modelom;
- analiza in ocena različnih optimizacijskih postopkov s stališča identifikacije parametrov;
- preverjanje uspešnosti različnih modelnoreferenčnih metod za identifikacijo parametrov z digitalno simulacijo;
- prehod na sprotno identifikacijo s pomočjo diskretnega referenčnega modela in preverjanje uspešnosti metod na hibridnem računalniškem sistemu;
- zaključne analize resulatatov in kometiranje dela;

7.a.2. EKSPERIMENTALNE METODE MODELIRANJA IN IDENTIFIKACIJE

- teoretske osnove identifikacije multivariabilnih modelov ARIMA;
- izdelava metodologije identifikacije;
- teoretske osnove ocenjevanja multivariabilnih modelov ARIMA;
- izdelava metodologije ocenjevanja;
- izdelava splošnega programskega paketa za identifikacijo multivariabilnih modelov ARIMA;
- izdelava priročnika za uporabo paketa;

7.b. Programski sklop FE, Ljubljana;

PREPOZNAVANJE GOVORNIH ODTISOV

Nosilec: prof. dr. L. Gyergyek

- digitalna predobdelava signalov;
- analiza fonemov v smislu iskanja informacije, ki najboljše opiše govornika;
- izbira takega postopka razpoznavanja, ki bo primeren za identifikacijo in za verifikacijo govornika;

7.c. Programski sklop VTŠ-Strojništvo Maribor:

ZASNOVA PODATKOVNIH STRUKTUR, POSTOPKOV IN SISTEMSKIH PROGRAMOV INTERAKTIVNE RAČUNALNIŠKE GRAFIKE

Nosilec: dr. Milan Kac

- zasnova in kodiranje enotne prikazovalne datoteke;
- zasnova in strukturiranje podatkovne baze grafičnih prvin;
- študij in izvedba postopkov in programov za ugotavljanje vidnosti elementov trorazsežnih grafičnih objektov;
- identifikacija grafičnih prvin, ki jih približno pokažemo z markerjem, da bi jih zbrisali ali spremenili;
- zasnova programov, s katerimi bi splošno interaktivno grafiko prikrojevali posebnim področjem uporabe, npr. simulacijskim problemom;

7.d. Programski sklop VTŠ-Elektrotehnika Maribor:

TEORETIČNO RAČUNALNIŠKO IN MODELNO REŠEVANJE SISTEMOV Z ZVEZNIMI IN DISKRETNIMI METODAMI

Nosilec: dr. Alojz Paulin

- razširitev programa CERN-W 126, ki ima matriko s 3250 elementi tako, da bo imel matriko s 6500 elementi pri nespremenjeni vrednosti ostalih parametrov v programu;

- *uporaba programa ETAN 2, ki daje numerične vrednosti za eksaktno rešitev enačbe gibanja naelektrenega delca, za izbrano obliko osnosimetričnega elektrostatičnega polja;*

- *določitev uporabnih intervalov za začetno kinetično energijo in pospeševalno napetost v programu POLESLED za vsak dan elektronsko optični problem ter izdelava metode za določitev teh intervalov;*

7.e. Programski sklop ISKRA AVTOMATIKA, TOZD Razvojni inštitut, Ljubljana

STRUKTURNA ANALIZA VELIKIH SISTEMOV IN MODELOV

Nosilec: mg-r. Jože Kanduč

- *sgradba algoritmov za dekompozicijo velikih sistemov s pomočjo usmerjenih grafov in binarnih matrik. Izdelava programov po sgrajenih algoritmih in njih testiranje;*

- *prikaz praktične uporabnosti izdelanih programov za dekompozicijo, predvsem na področju računalniškega programiranja;*

- *strukturna analiza aparaturne in programske opreme telsinformacijskega sistema z uporabo izdelanih programov;*

DODATEK A:

PROGRAMSKI SVET ZA RAČUNALNIŠTVO IN AVTOMATIKO SKUPNEGA PROGRAMA RSS

Izvajalci:

Marjan Špegel, IJS, predsednik

Ivan Bratko, FE in IJS, Ljubljana
France Bremšak, FE, Ljubljana
Karel Jezernik, VTŠ, Maribor
Milan Kac, VTŠ, Maribor
Janez Korenini, IJS, Ljubljana
Pavel Oblak, IJS, Ljubljana
Stanko Strmčnik, IJS, Ljubljana
Peter Suhel, FE, Ljubljana
Jernej Virant, FE, Ljubljana

Uporabniki:

Tomaž Banovec, Center SRS za Informatiko
Rado Faleskini, Elektrotehna - Delta
Andrej Jerman-Blažič, Rep. kom. za inf.
Pavel Mencej, Avtomontaža
Marko Rutar, Gorenje Muta
Bruno Štiglic, Iskra Avtomatika
Peter Tovšak, Gorenje Velenje
Demetrij Uran, IMP Ljubljana
Andrej Uratnik, Iskra Kranj
Davor Viličič, RRC, Ljubljana

M. Špegel

SREČANJA

8-10 april, Paris, Francija

2nd International Symposium on Distributed Computing Systems

Organizator : IFIP
Informacije: T.Bricheteau, IRIA, B.P. 105, 78 78150, Le Chesnay, France

6-10 april, Firence, Italija

International Congress on Logic, Informatics and Law

Organizator: Instituto per la Documentazione giuridica of Consiglio Nazionale delle Richerche
Informacije: Instituto per la documentazione giuridica , Via Panciatichi, 56, 16-50127, Florence, Italy

13-16 april, Mexico City , Mexico

15th International Symposium on Mini and Microcomputers

Organizator: International Society for Mini and Microcomputers
Informacije: ing. Jorge Gil, Academic Secretary , MIMI IIMASUNAM, Apardo Postal 20-726, Mexico 20 D.F.

19-25 april, Peniscola , Španija

Formalization of Programming Concepts

Organizator: European Association for Theoretical Computer Science
Informacije: ICFPC, Facultat d" Informatica, Universitat Politecnica de Barcelona, Jordi Girona 31, Barcelona 34, Spain

19-21 april, Firence, Italija

International Symposium on Local Computer Networks

Organizator: IFIP, TC 6
Informacije: Studio Publiche Relazioni, Via S.Reparata 40, 50129, Florence Italy

20-24 april, Zagreb, Jugoslavija

JUREMA

Organizator: JUREMA
Informacije: Tajništvo JUREMA, ul. Djure Salaja 5 /IV pp 398, 41 000 Zagreb, Jugoslavija

26-29 april, Annapolis, Md., ZDA

9th Annual Telecommunications Policy Research Conference

Organizator: National Science Fondation, Federal Communications Commission, Markle Fondation
Informacije: TPRC Organizing Committee, c/o William Taylor, Bell Laboratoires 2C-258, 600 Mountain Avenue, Murray Hill, NJ 07974 USA

27-30 april, Tunis, Tunis

Data Bases, Models, Evaluation, Documentation and Information Systems

Organizator: APCET, French Div. of ACM, CNAM-IIE, EDF, Institut de Programmation
Informacije:M-elle M. du Monceau, Chapitre Francais de l'ACM , 5, rue de L'Eglise, 92420 Vaucresson, France

27-28 april, Le Chesnay, Francija

Nouvelles Classes de Materiels Graphiques Interacifs pour La C F A O (ecrans a memoire de trame type video ecrans a plasma, imprimantes electrostatiques)

Organizator: Groupe de travail "Graphique" de l'AFCET
Informacije:INRIA Bat. 28, Domaine de Voluceau, Rocquencourt, 78350 Le Chesnay, France

29 april- 1 maj, Ann Arbor, ZDA

ACM International Conference on Management of Data

Organizator: ACM , SIGMOD
Informacije: TobyJ.Teorey Data Base Systems Research Group, Graduate School of Business Administartion, The University of Michigan, Ann Arbor, MI 48109

30 april -1maj, Pittsburg, Pa., ZDA

12th Annual Pittsburgh Conference on Modeling and Simulation

Organizator: University of Pittsburgh in cooperation with Pittsburgh section of IEEE, Systems Man and Cybernetics Society
Informacije:William Vogt or Marlin Mickle, Modeling and Simulation Conference, 348 Benedum Engineering Hall, University of Pittsburgh, Pittsburgh, Pa 15262, USA

4-6 maj, Hamilton, Canada

First Canadian Conference on Industrial Computer Systems

Organizator:National Research Council, Associate Comm. on Automatic Control
Informacije:Joe Wright, Xerox Research Center of Canada , 2480 Dunwin Drive, Mississauga, Ontario, Canada L5L 1J9

11-13 maj, Milwaukee, Wis., ZDA

13th Annual ACM Symposium on Theory of Computing

Organizator: ACM, SIGACT, University of Wisconsin, Milwaukee
Informacije:george Davida, Dept. of EECS, University of Wisconsin, Milwaukee, WI 53201, USA

11-15 maj, Budimpešta, Madžarska

International Symposium COMNET 81, Networks from the Usex' Points of View

Organizator: IFIP, UNESCO
Informacije: COMNET Secretariat, John V Neumann, Society for Computer Sciences, P.O.B, 240, H-1368, Budapest , Hungary

12-14 maj , Minneapolis, ZDA

8th International Symposium on Computer Architecture

Organizator: ACM SIGART
Informacije: V.R.Franta, Computer Science Dept., Minneapolis, University of Minesota, 143 Space Center, MN 55455, USA

17-20 maj, Ann Arbor, Mich., ZDA

5th International Conference on Computers and the Humanities

Organizator: Assoc. for Computers and the Humanities, Assoc. for Literary and Linguistic Computing, University of Michigan
Informacije: GregoryA. Marks, Institute for Social Research, University of Michigan, Ann Arbor, MI 48 104 USA

18-21 maj, Brighton, V Britanija

Automan 81, 4th annual conference of the British Robot Association and 2nd international conference on assembly automation

Organizator:British Robot Association
Informacije: Brian Rooks, (Conference), 35-39, High Street, Kempston, Bedford MK 42 7B 1, UK

19-22 maj , Paris, Francija

SICOB, Congres Bureautique - AFCET

Organizator: AFCET
Informacije:AFCET, 156, Bld. Pereire, 75017 Paris, France

18-19 maj, Dubrovnik, Jugoslavija

Increasing of Computer and Information Literacy

Organizator:Inter University Centre of postgraduate Studies
Informacije:Inter University Centre of Postgraduate Studies, Frana Bulića 4, Dubrovnik 60 000, Jugoslavija

25-28 maj, Cavtat, Jugoslavija

Computer at the University

Organizator: University Computing Center - Zagreb
Informacije: SRCE, for Symposium, Engelsova bb, 41 000 Zagreb, Jugoslavija

25-27 maj, Marseille, Francija

TELEMAT 81 - Les Grand Projets Telematiques

Organizator:Ecole national Superieure de Physique
Informacije: TELEMAT 81, 13397 Marseille, Rue Henri Poincare, cedex 13, France

27-30 maj, Rousse, Bugarija

Organization and Automation of the Experimental Research

Organizator: State Commeettee of Science and Technical Progress and Central Council of Scientific and Technical Unions in Bulgaria
Informacije: The Organizing Commettee, 1000 Sofia, Rakovski str. 108, Bulgaria

3-5 juni, Nica, Francija

Premier Congres sur La Conception des Systemes Telematiques

Organizator: CITEL, Université de Nice
Informacije:Nice Congres, Palais des Expositions, 06300 Nice, France

3-5 juni, Amsterdam, Nizozemska

Second Seminar on Distributed Data Sharing Systems

Organizator: Vrije Universiteit, Vakgroep, Informatica, INRIA, IGDD
Informacije: Prof. Dr.R.P van de Riet, Vrije Universiteit Vakgroep Informatica, Wiskundig Seminarium, De Boelelaan 1081, NV Amsterdam, the Netherlands

4-5 juni, Washington, ZDA

Surviving the Technological Explosion

Organizator: ACM SIGCPR
Informacije:Carol Vaughan, Census Bureau, FB-3, Rm 3142, Suitland, MD 20233, USA

16-18 juni, London, V.Britanija

PROMLCON 81, Process Measurement and Control Conference

Organizator:Wembley Conference Centre
Informacije: The Institute of Measurement and Control, 20 Peel Street, London, W8 7PD, UK

10-12 juni, Waterloo, Ontario, Kanada

7th Conference of The Canadian Man Computer Communication Society

Organizator: Canadian Man Computer Society
Informacije: Marceli Wein, Computer Graphic Section Division of EE, National Research Council, Ottawa Ontario, Canada, K1A OR8

10-12 juni, Nürnberg, Z R Nemčija

CONPAR 81, Conference on Analyzing Problem Classes and Programming for Parallel Computing

Informacije:W.Handler, IMMD, Universitat Erlangen Nürnberg, Martenstrasse 3, D 8520 Erlangen, F.R.Germany

18-19 juni, Montpellier, Francija

Data Bases

Organizator: AFCET
Informacije: AFCET, 156, Bld.Periere 75017 Paris, France

25-27 juni, Rijeka, Jugoslavija

MIPRO 81,Primjena mikroprocesora

Organizator:ETAN,SOUR Rade Končar,SOUR EI Niš, RO PTT Rijeka, Centar za radničko stvaralaštvo
Informacije:Društvo strojarskih i elektrotehničkih inženjera i tehničara - Rijeka, Klub privrednika Trg Palmira Togliattija br. 4, 51000 Rijeka

15-17 juni, Portorož, Jugoslavija

Second Yugoslav Symposium on Applied Robotics

Organizator:ETAN, Institut Mihailo Pupin, Institut Jožef Stefan, Institut za Automatiku i Računarske nauke Energoinves
Informacije:ETAN, Symposium on Applied Robotics, POB 356 11001 Beograd, Jugoslavija

24-26 juni, Portland, ZDA

11th International Symposium on Fault Tolerant Comuting

Organizator:IEEE-CS
Informacije: Chung-Jen Tan, IBM T.J. Watson Research

Box 218, Yorktown Heights, NY 10598 USA

29 juni -1 juli, Nashville, ZDA

18th Design Automation Conference

Organizator: ACM SIGDA, IEEE-CS
Informacije: R.J. Smith II, V-R Information Systems Inc., 5758 Balcones Drives, Suite 205, Austin, TX 78731, USA

13-17 juli, Haifa, Izrael

8th International Colloquium on Automata, Languages and Programming

Organizator: European Association for Theoretical Computer Science
Informacije: S. Even (ICALP 81), Computer Science Dept. The Technion, Haifa Israel

27-31 juli, Lausanne, Švica

3rd World Conference on Computers in Education

Organizator: IFIP TC 3 and its Working Groups 3.1 3.3, 3.4
Informacije: P. Immer, Ecole Polytechnique Federal de Lausanne, Switzerland

5-7 avgust, Snowbird, Utah, ZDA

ACM Symposium on Symbolic and Algebraic Computation

Organizator: ACM SIGSAM, European SEAS, Nordic Interest Group for Symbolic and Algebraic Manipulation
Informacije: B.F. Caviness, Dept. of Mathematical Sciences, Rensseler Polytechnic Institut, Troy NY 12181, USA

24-28 avgust, Paris, Francija

6th International Seminar on Computational Aspects of the Finite Element Method

Organizator: International Assoc. for SMIRT, ID, Universität Stuttgart
Informacije: J.F. Cloudman, Macneal Schwendler Corp. 7442 No. Figueroa St., Los Angeles, CA 90041

24-28 avgust, Vancouver, Kanada

Seventh International Joint Conference on Artificial Intelligence

Organizator: IJCAI
Informacije: Patrick Hayes, University of Rochester, Dept. of Computer Science, Mathematical Science Building, Rochester, NY 14627

24-28 avgust, Stirling, Škotska

CO 81, Conference on Combinatorial Optimisation

Organizator: Stirling University
Informacije: L. Wilson (CO 81), Dept. of Computing, Stirling University, Stirling Scotland

24-28 avgust, Kyoto, Japonska

8th IFAC World Conference

Organizator: IFAC
Informacije: IFAC Secretariat, A 2361 Laxemburg, Schlossplatz 12, Austria

25-28 avgust, Chester, Anglija

Vector and Parallel Processors in Computational Sciences

Organizator: IMA Numerical Algorithms Group
Informacije: S.A. Lowndes, Science Research Council Daresbury Laboratory, Daresbury, Warington WA4 4AD, England

25-28 avgust, Bangkok, Tailand

International Conference on Computing for Development

Organizator: Asian Institute for Technology and Carl Duisberg Gesellschaft in cooperation with ACM
Informacije: M. Nawaz Sharif, Director, Regional Computer Center, Div. of Computer Applications Asian Institute of Technology, P.O.B 2754, Bangkok Thailand

31 avgust-1 september, Kyoto, Japonska

IFAC/IFIP Workshop on Real-Time Programming

Organizator: IFAC/IFIP
Informacije: c/o Prof. T. Hasegawa, Dept. of Appl. Math. &Phys. Faculty of Engineering, Kyoto University Kyoto, 606, Japan

31 avgust-4 september, Strbske Pleso, ČSSR

10th International Symposium on Mathematical Foundations of Computer Science

Organizator: Computing Research Center Bratislava
Informacije: Jozef Gruska, Computing Research Center, Dubravska 3, 885 31 Bratislava, $\check{C}S_{SR}$

7-9 september, Kaiserslautern, ZR Nemčija

International Conference on Computer Hardware Description Languages and Their Applications

Organizator: IFIP Tech. Comm. TC 10 and its Working Group W.G. 10.2 in cooperation with SIARCH and SIGDA, IEEE-CS, GI, NTG
Informacije: Reiner Hartenstein, Universitat Kaiserlautern, Fachbereich Informatik Postfach 3049, D-6750, Kaisenslautern, F.R. Germany

9-11 september, Darmstadt, ZR Nemčija

Eurographics 81, Annual Conference of the Eurographic Society

Organizator: German Computer Society
Informacije: J. Encarnaco, Eurographics 81, Technische Hochschule Darmstadt, FG Graphisch Interactive Systeme, Steubenplatz 12, D-6100 Darmsadt F.R. Germany

8-10 september, Paris, Francija

EUROMICRO 81

Organizator: Universität Dortmund
Informacije: Lutz Richter, Universität Dortmund Informatik Postfach 500500, D-4600 Dortmund 50, F R Germany

14-16 september, Paris, Francija

International Conference on Performance of Data Communication Systems and Their Applications

Organizator: Ecole National Superieure des Telecocomunications, INRIA
Informacije: INRIA, Service des Relations Exterieures Domaine de Voluceau, B.P. 105, 78150 Le Chesnay, France

9-11 september, Cannes, Francija

7th International Conference on VeryLarge Data Bases

Organizator: ACM, INRIA, IEEE-CS
Informacije: INRIA, Service des Relations Exterieures Domaine de Voluceau, B.P. 105, 78150 Le Chesnay, France

15-18 september, Wroclaw, Poljska

Systems Science, International Conference

Organizator: Technical University of Wroclaw, Institute of Engineering Cybernetics
Informacije: Jerzy Swiatek, Technical University of Wroclaw, Institute of Engineering Cybernetics Janiszewskiego St. 11 /17, 50-370 Wroclaw, Poland

16-18 september, Nancy, Francija

Patern Recognition and Artificiel Intelligence

Organizator: AFCET
Informacije: AFCET, 156, Bld. Pereire, 75017 Paris, France

28-30 september, Brno, ČSSR

International Conference on Fault - Tolerant Systems and Diagnosis

Organizator: Czechoslovak Scientific and Technical Society Czech. Central Commettee for Electronics, Central Professional Group for Diagnostics in Electronics, House of Technology ČSVTS in Češke Budejovice
Informacije: DUM Teahniky ČSVTS, Trida 5 Kvetna 42, 37 021, Češke Budejovice ČSSR

6-7 oktober, , Ljubljana, Jugoslavija

XV Jugoslovanski Simpozij o Telekomunikacijah

Organizator: Elektrotehniška zveza Slovenije
InformacijeElektrotehniška zveza Slovenije, Titova 50, Ljubljana 61000, Jugoslavija

19-23 oktober, Munich, Z R Nemčija

Third Conference of the European Cooperation in Informatics

Organizator: European Cooperation in Informatics
Informacije: A.J.W. Duijvestijn, P.O.B 217, 7500 Enschede, The Netherlands

26-29 oktober, Amsterdam, Nizozemska

International Symposium on Algorithmic Languages

Organizator: Mathematical Center, IFIP TC2
Informacije: Algorithmic Languages 1981, Mathematical Centre Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

4-6 november, Amsterdam, Nizozemska

Performance 81, 8th IFIP WG 7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation

Organizator: IFIP WG 7.3 with ACM SIGMETRICS
Inormacije: F.J. Kylstra, Dept. of EE, Technische Hogeschool Eindhoven, P.O. Box 513, 5600 MB Eindhoven The Netherlands

17-20 november, Gif sur Yvette, Francija

Informatique 81

Organizator: AFCET
Informacije: AFCET, 156, Bld Pereire, Paris, France

13-16 december, Pacific Grove, ZDA

8th Symposium on Operating Systems Principles

Organizator: ACM SIGOPS
Informacije: David P. Reed, Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge MA 02139

RAZSTAVE

3-8 maj
Electronic Distribution Show, Atlanta, ZDA

30 maj - 4 juni
Internationales Fernsehsymposium und Technische Ausstellung, Montreux Schweiz

8-11 juni
Automatic Test Equipment, Boston, Ma, ZDA

5-8 oktober,
Electronics Test & Measurement Conference and Exhibition Hyatt Regency Chicago, ZDA

Seminarji in šole

20-22 maj, Dubrovnik
On-line next ten years
New information Technology and New Industries in the 1980s
Man-Computer Communication in the 1980
Library, Laboratory, Factory, Office and Bank
Distribution of usage as design parameter in information systems

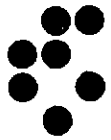
25-27 maj, Dubrovnik
On-line methods and examples
Participative Information System Design
Software Engineering and Software Products
Data Base Methods and Examples

Za oba seminarja informacije dobite na naslov:
Inter University Centre of postgraduate studies
Frana Bulića 4, 50 000 Dubrovnik, Jugoslavija

29 juni-10 juli, Bonas Francija

Automatic Speech Analysis and Recognition
Informacije: Jean-Paul Haton, Centre de Recherche en Informatique, Universite de Nancy I -C.O. 140 - 54037 Nancy, France

univerza edvarda kardelja
institut "jožef stefan" ljubljana, jugoslavija



ODSEK ZA RAČUNALNIŠTVO IN INFORMATIKO

Odsek za računalništvo in informatiko Instituta J. Stefan ima dvajset let izkušenj v temeljnih in aplikativnih raziskavah na področju računalništva in informatike. Odsek ima 40 sodelavcev, specialistov na različnih področjih računalništva. Zaradi vedno večjega obsega dela stalno vključujemo nove sodelavce na obetavne raziskovalne in razvojne projekte.

DEJAVNOSTI ODSEKA:

- temeljne raziskave na področju računalništva in informatike
- razvoj računalniških sistemov in naprav
- razvoj računalniških aplikativnih sistemov
- prototipna proizvodnja računalniške opreme
- uvajanje računalništva v srednje šole
- ekspertize in svetovanja

RAZISKOVALNI PROGRAM ODSEKA:

- digitalna tehnika, mikroračunalniki in sistemi
- sistemsko programiranje in operacijski sistemi
- informacijski sistemi
- računalniške komunikacije in mreže
- umetna inteligenca
- računalniški vid in inteligentni roboti

RAZVOJNI PROGRAM ODSEKA:

- razvoj za proizvodnjo računalnikov
- nadzorni in signalizacijski sistemi za hotele in bolnice
- računalniško vodenje vodovodnih sistemov in čistilnih postaj
- računalniški telekomunikacijski sistemi in računalniške mreže
- poslovni in tehnični informacijski sistemi
- računalniški fotostavni sistemi
- inteligentni robot za obločno varjenje

Vse nadalnje informacije dobite na Odseku za računalništvo in informatiko.

INSTITUT "JOŽEF STEFAN"

Jamova 39, 61000 Ljubljana
tel. 263-261/int. 318

NOVI PROGRAMSKI PRODUKTI ODSEKA:

1. SYNC/RT-11 - komunikacijski paket

Komunikacijski programski paket SYNC/RT-11 omogoča sinhrono računalniško povezavo računalnikov PDP-11 pod operacijskim sistemom RT-11 z računalniki Control Data (CDC) tipa 3000, 6000 in družine Cyber 170.

2. DECMAK - programski paket za odločanje

Programski paket DECMAK omogoča računalniško vodenje participativnega večparameterskega odločitvenega postopka. Prednosti takega odločanja so: neposreden in udoben stik z uporabnikom - odločevalcem, avtomatsko evidentiranje odločitvenega postopka in enostavna razlaga odločitev. Primeri uporabe: izbor ustrezne računalniške opreme, izbor primerne lokacije objektov, izbor ustrezne tehnologije itd.

3. PERSONAL - programski paket za vodenje personalnih evidenc

S tem programskim paketom zajemamo in ažuriramo podatke personalne evidence. Prednost tega paketa je v enostavnosti njegove uporabe in njegovi fleksibilnosti z ozirom na iskanje in prikazovanje podatkov.

4. Programski paket za obdelavo davčnih zavezancev

Ta programski paket je v osnovi namenjen davčnim upravam občinskih skupščin. S tem je avomatiziran ves proces obdelave dolga davčnih zavezancev od vnašanja podatkov, preko ažuriranja, izračuna davčne obveznosti do izpisa položnic.

Navedeni programski paketi pod točkami 2, 3 in 4 tečejo na računalnikih DELTA in jih lahko uporabljamo brez računalniškega predznanja. Po dogovoru jih prilagodimo tudi za druge računalnike.

NAVODILO ZA PRIPRAVO ČLANKA

Avtorje prosimo, da pošljejo uredništvu naslov in kratek povzetek članka ter navedejo približen obseg članka (število strani A 4 formata). Uredništvo bo nato poslalo avtorjem ustrezno število formularjev z navodilom.

Članek tipkajte na priložene dvokolonske formularje. Če potrebujete dodatne formularje, lahko uporabite bel papir istih dimenzij. Pri tem pa se morate držati predpisanega formata, vendar pa ga ne vrišite na papir.

Bodite natančni pri tipkanju in temeljiti pri korigiranju. Vaš članek bo s foto postopkom pomanjšan in pripravljen za tisk brez kakršnihkoli dodatnik korektur.

Uporabljajte kvaliteten pisalni stroj. Če le tekst dopušča uporabljajte enojni presledek. Črni trak je obvezen.

Članek tipkajte v prostor obrobljen z modrimi črtami. Tipkajte do črt - ne preko njih. Odstavek ločite z dvojnim presledkom in brez zamikanja prve vrstice novega odstavka.

Prva stran članka :

a) v sredino zgornjega okvira na prvi strani napišite naslov članka z velikimi črkami;
b) v sredino pod naslov članka napišite imena avtorjev, ime podjetja, mesto, državo;
c) na označenem mestu čez oba stolpca napišite povzetek članka v jeziku, v katerem je napisan članek. Povzetek naj ne bo daljši od 10 vrst.
d) če članek ni v angleščini, ampak v katerem od jugoslovanskih jezikov izpustite 2 cm in napišite povzetek tudi v angleščini. Pred povzetkom napišite angleški naslov članka z velikimi črkami. Povzetek naj ne bo daljši od 10 vrst. Če je članek v tujem jeziku napišite povzetek tudi v enem od jugoslovanskih jezikov;
e) izpustite 2 cm in pričnite v levo kolono pisati članek.

Druga in naslednje strani članka:
Kot je označeno na formularju začnite tipkati tekst druge in naslednjih strani v zgornjem levem kotu,

Naslovi poglavij:
naslove ločuje od ostalega teksta dvojni presledek.

Če nekaterih znakov ne morete vpisati s strojem jih čitljivo vpišite s črnim črnilom ali svinčnikom. Ne uporabljajte modrega črnila, ker se z njim napisani znaki ne bodo preslikali.

Ilustracije morajo biti ostre, jasne in črno bele. Če jih vključite v tekst, se morajo skladati s predpisanim formatom. Lahko pa jih vstavite tudi na konec članka, vendar morajo v tem primeru ostati v mejah skupnega dvokolonskega formata. Vse ilustracije morate (nalepiti) vstaviti sami na ustrezno mesto.

Napake pri tipkanju se lahko popravljajo s korekcijsko folijo ali belim tušem. Napačne besede, stavke ali odstavke pa lahko ponovno natipkate na neprozoren papir in ga pazljivo nalepite na mesto napake.

V zgornjem desnem kotu izven modro označenega roba oštevilčite strani članka s svinčnikom, tako da jih je mogoče zbrisati.

Časopis INFORMATICA
Uredništvo, Institut Jožef Stefan, Jamova 39, Ljubljana

Naročam se na časopis INFORMATICA. Predplačilo bom izvršil po prejemu vaše položnice.

Cenik: letna naročnina za delovne organizacije 350,00 din, za posameznika 120,00 din.

Časopis mi pošiljajte na naslov ☐ stanovanja ☐ delovne organizacije.

Priimek.....

Ime.....

Naslov stanovanja

Ulica.....

Poštna številka_____ Kraj.....

Naslov delovne organizacije

Delovna organizacija.....

.....

Ulica.....

Poštna številka_____ Kraj.....

Datum..... Podpis:

.....

INSTRUCTIONS FOR PREPARATION OF A MANUSCRIPT

Authors are invited to send in the address and short summary of their articles and indicate the opproximate size of their contributions (in terms of A 4 paper). Subsequently they will receive the outor's kits.

Type your manuscript on the enclosed two-column-format manuscript paper. If you require additional manuscript paper you can use similar-size white paper and keep the proposed format but in that case please do not draw the format limits on the paper.

Be accurate in your typing and through in your proof reading. This manuscript will be photographically reduced for reproduction without any proof reading or corrections before printing.

Use a good typewriter. If the text allows it, use single spacing. Use a black ribbon only.

Keep your copy within the blue margin lines on the paper, typing to the lines, but not beyond them. Double space between paragraphs.

First page manuscript:

a) Give title of the paper in the upper box on the first page. Use block letters.
b) Under the title give author's names, company name, city and state - all centered.
c) As it is marked, begin the abstract of the paper. Type over both the columns. The abstract should be written in the language of the paper and should not excesed 10 lines.
d) If the paper is not in English, drop 2 cm after having written the abstract in the language of the paper and write the abstract in English as well. In front of the abstract put the English title of the paper. Use block letters for the title. The lenght of the abstract should not be greater than 10 lines.
e) Drop 2 cm and begin the text of the paper in the left column.

Second and succeeding pages of the manuscript:
As it is marked on the paper, begin the text of the second and succeeding pages in the left upper corner.

Format of the subject headings:
Headings are separated from text by double spacing.

If some characters are not available on your typwriter write them legibly in black ink or with a pencil. Do not use blue ink, because it shows poorly.

Illustrations must be black and white, sharp and clear. If you incorporate your illustrations into the text keep the proposed format. Illustration can also be placed at the end of all text material provided, however, that they are kept within the margin lines of the full size two-column format. All illustrations must be placed into appropriate positons in the text by the author.

Typing errors may be corrected by using white correction paint or by retyping the word, sentence or paragraph on a piece of apaque , white paper and pasting it nearly over errors

Use pencil to number each page on the upper-right-hand corner of the manuscript, outside the blue margin lines so that the numbers may be erased.

Časopis INFORMATICA
Uredništvo, Institut Jožef Stefan, Jamova 39, Ljubljana

Please enter my subscription to INFORMATICA and send me the bill.

Annual subscription price: companies 350,00 din (for abroad US $ 22), individuals 120,00 din (for abroad US $ 7,5).

Send journal to my ☐ home address ☐ company's address.

Surname.....

Name.....

Home address

Street.....

Postal code _____ City.....

Company address

Company.....

.....

Street.....

Postal code _____ City.....

Date..... Signature

.....

delta računalniški sistemi ®

SOZD ELEKTROTEHNA, o. o., DO DELTA, proizvodnja računalniških sistemov in inženiring, p. o.
61000 Ljubljana, Parmova 41
Telefon: 061/310-393
Telex: 31 578 YU ELDEC

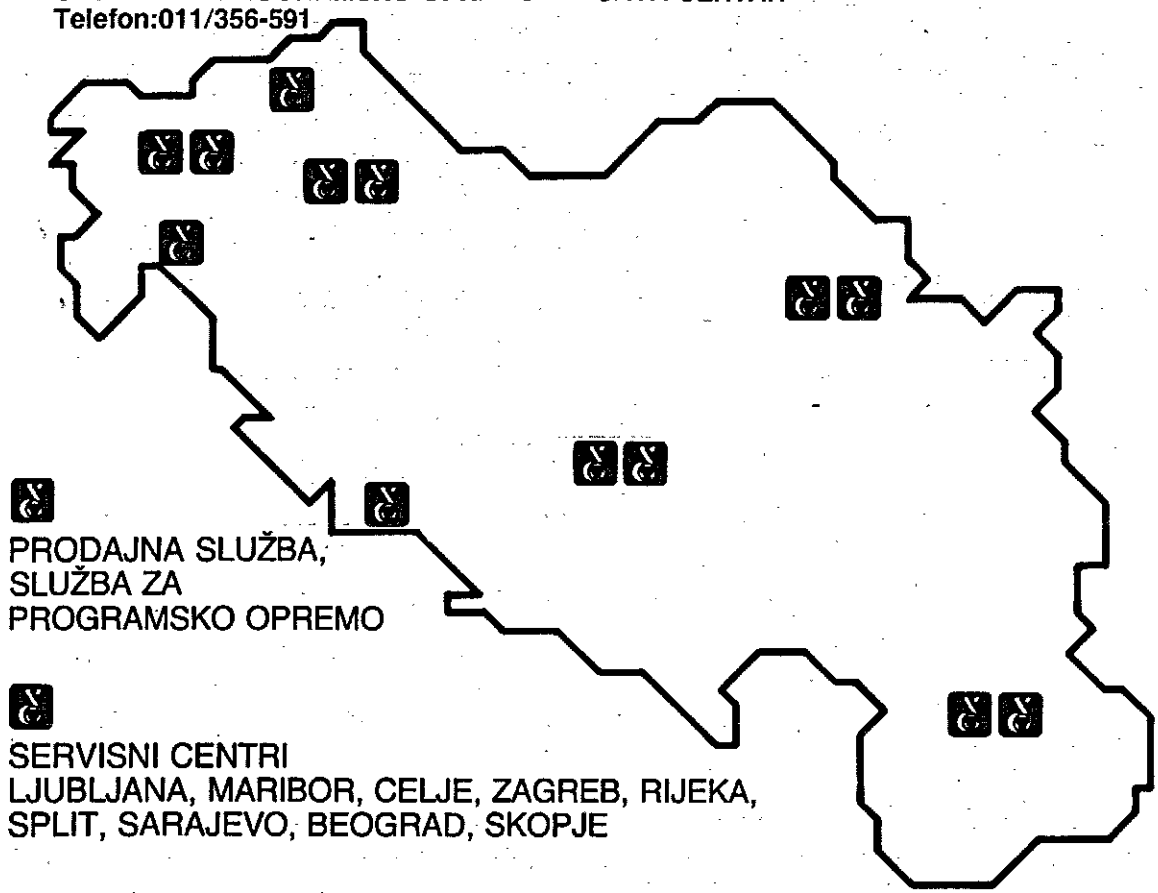
POSLOVNA ENOTA ZAGREB

ZAGREBAČKI VELESAJAM, II. UPRAVNA ZGRADA
ALEJA BORISA KIDRIČA 2, 41000 ZAGREB
Telefon: 041/520-003, 516-690

— PROIZVODNJA RAČUNALNIŠKE OPREME
61000 LJUBLJANA, LINHARTOVA 62a
Telefon: 061/323-585, 326-661
— VZDRŽEVALNA SLUŽBA
61000 LJUBLJANA, LINHARTOVA 62a
Telefon: 061/323-585, 326-661
— SLUŽBA ZA PROGRAMSKO OPREMO
TITOVA 51, 61000 LJUBLJANA, Telefon: 061/327-654
— DELTA IZOBRAŽEVALNI CENTER
Telefon: 061/345-673
— PRODAJNA SLUŽBA
TITOVA 51, 61000 LJUBLJANA
Telefon: 061/320-241, int. 397, 420
— DELTA INŽENIRING, PARMOVA 41, 61000 LJUBLJANA
Telefon: 061/314-394
— SLUŽBA ZA RAZVOJ STROJNE OPREME
Telefon: 061/23-251, 21-874
— SLUŽBA ZA RAZVOJ PROGRAMSKE OPREME
Telefon: 061/28-216

POSLOVNA ENOTA BEOGRAD

— VZDRŽEVALNA SLUŽBA — KARAĐORĐEV TRG 13, 11080 ZEMUN
Telefon: 011/694-537, 695-604
— PRODAJNA SLUŽBA, »SAVA CENTAR«
MILENTIJE POPOVIĆA 9, 11070 NOVI BEOGRAD
Telefon: 011/453-885
— SLUŽBA ZA PROGRAMSKO OPREMO — »SAVA CENTAR«
Telefon:011/356-591



PRODAJNA SLUŽBA,
SLUŽBA ZA
PROGRAMSKO OPREMO

SERVISNI CENTRI
LJUBLJANA, MARIBOR, CELJE, ZAGREB, RIJEKA,
SPLIT, SARAJEVO, BEOGRAD, SKOPJE