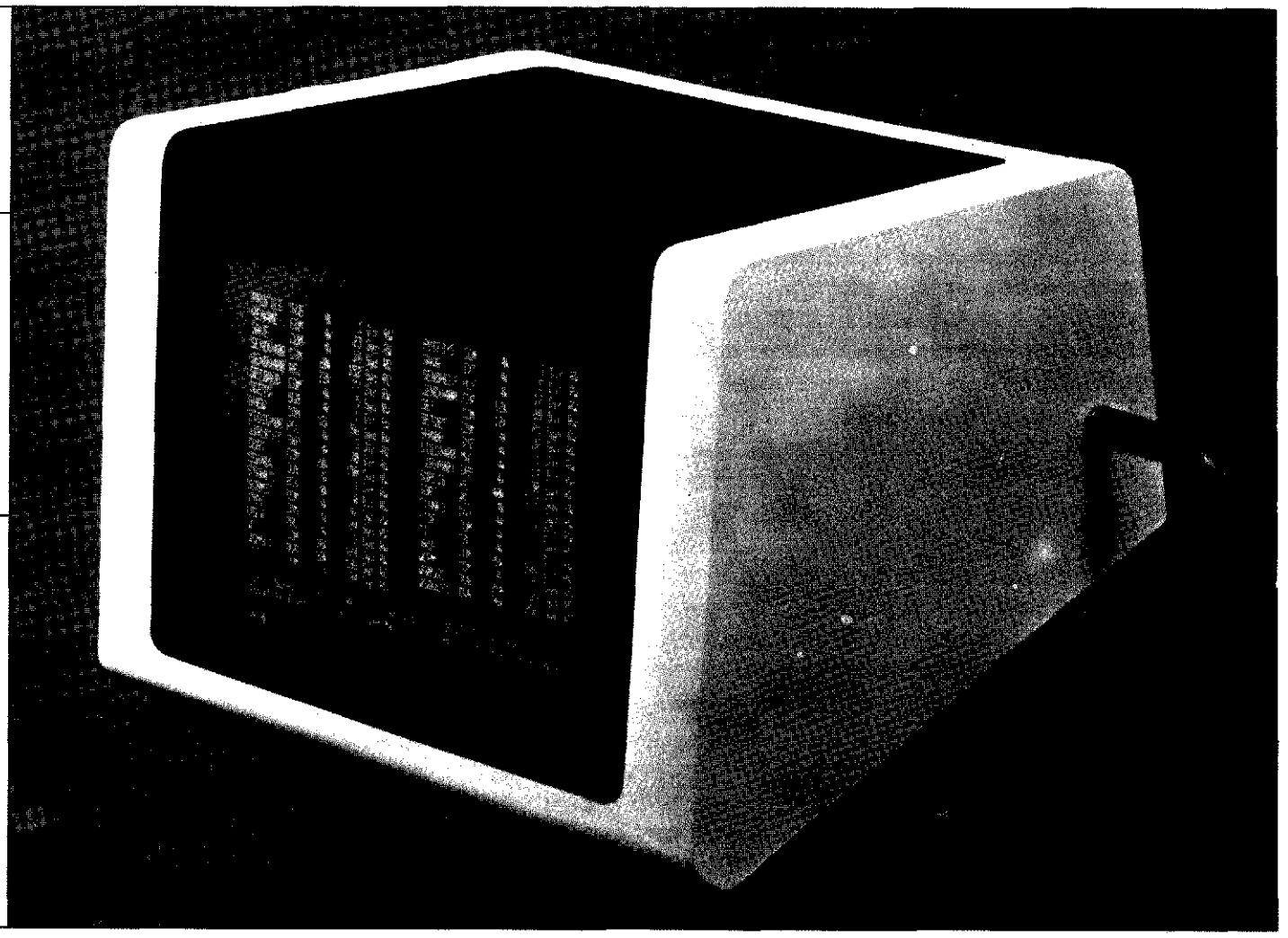
66 informatica 4

YU ISSN 0350-5596

SISTEM ZA ŠALTERSKO POSLOVANJE V BANKAH IN NA POŠTAH



računalniški sistemi delta

Sistem za šaltersko poslovanje je sodobna računalniška oprema za delo v bankah in na poštah, opremljen z ustrezno programsko opremo.
Sistem omogoča samostojno ažurno poslovanje – od posameznih operativnih del na šalterjih do zajema podatkov za nadaljnjo obdelavo. Deluje lahko povsem samostojno ali v povezavi z glavnim računalnikom (prenos informacij je mogoč prek stalno najetih ali navadnih telefonskih linij). Delovanje sistema tudi ni odvisno od razpoložljivosti računalniških kapacitet glavnega računalnika.
Sistem nadomešča raznovrstno opremo, ki se uporablja pri šalterskem poslovanju – od klasičnih mehanografskih strojev, pisalnih strojev do kalkulatorjev in deloma mikročitalnikov.

Sistem za šaltersko poslovanje je savremena računarska oprema za rad u bankama i poštama, opremljen sa odgovarajućom programskom opremom.
Sistem omogućava samostalno ažurno poslovanje – od pojedinih operativnih poslova na šalterima do zahvata podataka za dalju obradu. Može da radi sasvim samostalno, ili da komunicira sa glavnim računarom (prenos informacija je moguć preko stalno iznajmljenih ili običnih telefonskih linija). Rad sistema je takođe nezavisan od raspoložljivosti računarskih kapaciteta glavnog računara.
Sistem zamenjuje raznovrstnu opremu, koja se upotrebljava u šalterskom poslovanju – od klasičnih mehanografskih mašina, pisaćih mašina do kalkulatora i delimično čitača mikrofiševa.

informatica

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

YU ISSN 0350-5596

LETNIK 10, 1986 - ŠT. 4

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

Uredništvo in uprava:

Informatika, Parmova 41, 61000 Ljubljana, telefon (061) 312 988; teleks 31366 YU Delta.

Letna naročnina za delovne organizacije znaša 5900 din, za zasebne naročnike 1590 din, za študente 490 din; posamezna številka 2000 din.

Številka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije

Na podlagi mnenja Republiškega komiteja za informiranje št. 23-85, z dne 29. 1. 1986, je časopis oproščen temeljnega davka od prometa proizvodov

Tisk: Tiskarna Kresija, Ljubljana

Grafična oprema: Rasto Kirn

V S E B I N A

informatica

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

Headquarters:

Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia. Phone: 61 31 29 88. Telex: 31366 yu delta

Annual Subscription Rate: US$ 22 for companies, and US$ 10 for individuals

Printed by: Tiskarna Kresija, Ljubljana

Design: Rasto Kirn

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 10, 1986 – No. 4

CONTENTS

LOGIKA I PROCESIRANJE ZNANJA

INFORMATICA 4/86

Mario Radovan
Sveučilište Rijeka, SET Pula
Univarza Ljubljana, IJS Ljubljana

UDK oo1:681.3.04

U članku je data analiza mogućnosti koje logika pruža u okviru problematike predstavljanja i procesiranja znanja. Rezultati su sabrani u cjelovit prijedlog modela logičke baze znanja, koji je definiran u terminima logike i realiziran sredstvima logičkog programiranja. U kontekstu predloženog modela dati su i prijedlozi rješenja dvaju problema nepotpunosti sistema klauzalne logike sa SLDNF-rezolucijom kao metodom dedukcije.

Znanja su podjeljena na pozitivna, negativna te definiciju hijerarhije entiteta u bazi. Dati su načini predstavljanja tih znanja i njihova uloga u procesu dedukcije, nasljeđivanja svojstava i održavanja integriteta baze znanja.

Model je implementiran u Prologu, a rad sistema ilustriran je primjerima. Pokazan je način obrazlaganja uspjelih kao i neuspjelih pokušaja dedukcije (izračunavanja) odgovora na upite postavljane sistemu.

LOGIC AND KNOWLEDGE PROCESSING: The article presents an analysis of the possibilities which logic offers to the problems of representing and processing knowledge. The results are collected in a proposal of an integral model of a knowledge base, defined in terms of logic and realised through logic programming means. Within the model, solutions are proposed for two incompleteness problems of systems of clausal logic with SLDNF-resolution as a deduction method.

Knowledge is classified into positive and negative knowledge and definitions of hierarchies of entities in the knowledge base. Ways of representing these types of knowledge and their role in the process of deduction, property inheritance and maintanance of the integrity of the knowledge base are given.

The model was implemented in PROLOG, and a few examples of its behavior are included here. They illustrate the manner in which the attempts at deducing (computing) the answers to queries are explained.

1. UVOD

Često se ističe da je problem predstavljanja znanja centralni problem tehnologije znanja, a time i razvoja sistema zasnovanih na znanju, odnosno ekspertnih sistema. U ovom članku dat je prijedlog sistema za predstavljanje i procesiranje znanja, koji smo nazvali modelom logičke baze znanja. Model je definiran u terminima matematičke logike odnosno logičkog programiranja, kao osnovnog sredstva ali i jedinstvenog metodološkog pristupa problematici predstavljanja i procesiranja znanja.

Odlikama logike (prvoga reda), a time i razlozima za njenu primjenu, u datom kontekstu smatramo:

- Deklarativnost semantike jezika logike. Znanja izražena u jeziku logike (prvoga reda) direktno su "čitljiva", za razliku od znanja predstavljenih u proceduralnim jezicima, gdje se "što" (tj. logika) zagubi u "kako" (tj. proceduri).
- Precizna sintaksa i semantika jezika, čime se izbjegava problem višeznačnosti, svojstven prirodnom jeziku.
- Postojanje metoda za automatsku dedukciju (izračunavanje) ispravnih odgovora iz datog skupa premisa (tj. baze znanja), datih u (pod)jeziku logike prvoga reda.

Iako je logika prvoga reda potpuna (tj. svaka logička konsekvenca skupa premisa je i deducibilna), ne postoji (općenit) algoritam kako tu logičku konsekvencu deducirati. Kao rezultat traženja jezika, za koji postoji efikasan algoritam za (automatsko) deduciranje logičkih konsekvenci iz datog skupa premisa definiran je jezik prvoga reda koji smo nazvali jezikom definitnih klauzula.

DEF 1.1.

Definitnom klauzulom nazivamo formulu oblika

A <-- B1 & ... & Bn

gdje je A atom (tj. atomarna formula), a B1, ..., Bn su literali (pozitivni ili negativni). Pritom atom A nazivamo glavom (head) klauzule a konjunkciju literala B1, ..., Bn tijelom (body) klauzule. Definitnu klauzulu oblika

A <--

nazivamo činjenicom.

Pojam definitne klauzule uveo je van Emdem u <Emd 78>, ali u užem značenju pojma (bez negativnih literala u tijelu), nego što to ovdje činimo.

Razlog za posebno bavljenje jezikom definitnih klauzula jeste u tome što te klauzule, pored uobičajene deklarativne semantike (značenja, "čitanja"), imaju i prikladnu proceduralnu semantiku, što ih čini pogodnim da budu uzete kao jezik za predstavljanje znanja, ali ujedno i kao kompjutacijski jezik. Naime, definitnu klauzulu

A <-- B1 & ... & Bn (1)

možemo deklarativno shvatiti (čitati) kao pravilo koje kaže da istinitost literala B1, ..., Bn implicira istinitost atomarne formule A. No, istu klauzulu možemo interpretirati i proceduralno, tj. kao instrukciju, koja kaže: zadatak A izvršavamo tako da izvršimo sve zadatke B1, ..., Bn.

Klauzulu oblika

A1 v ...v Am <-- B1 & ... & Bn (2)

nazivamo nedefinitnom klauzulom. Deklarativno, klauzula (2) kazuje da istinitost (svih) literala B1, ..., Bn implicira istinitost bar jednog (ma da ne znamo kojeg i kolikih), literala od A1, ..., Am. Međutim, proceduralna interpretacija, a posebno implementacija automatske dedukcije kod nedefinitnih klauzula je znatno kritičnija. Tako npr. u <Yah 85> nalazimo zaključak da je nedefinitnost klauzula "vrlo nepoželjno svojstvo", zato što predstavlja znatno kompleksniji problem za automatsku dedukciju.

DEF 1.2.

Zapis (formulu) oblika

<-- B1 & ... & Bn

gdje su B1, ..., Bn literali, nazivamo ciljem.

Formule navedene u definicijama (1.1) i (1.2) jesu zapravo samo matrice formula logike prvoga reda, koje bi zajedno sa implicitno mišljenim kvantifikacijskim prefiksima glasile:

a) definitna klauzula iz definicije (1.1):

Vx1 ... Vxj(A <-- B1 & ... & Bn)

odnosno, činjenica:

Vx1 ... Vxj(A)

gdje su x1, ..., xj sve varijable koje se javljaju u atomu A i/ili literalima B1, ..., Bn. Dakle, svaka varijabla, koja nastupa u definitnoj klauzuli, implicitno je vezana univerzalnim kvantifikatorom.

b) cilj iz definicije (1.2), tj.:

<-- B1 & ... & Bn

čitamo: "nije istina da B1 i ... i Bn". Dakle,

Vx1 ... Vxj(n(B1 & ... & Bn)) (3)

ili - a što je u kontekstu daljnjeg rada pogodnije - na logički ekvivalentan način:

n(Ex1 ... Exj(B1 & ... & Bn)) (4)

Formule (3) i (4) su logički ekvivalentne.

Razlog za predstavljanje cilja kao negirane ('n') egzistencijalno kvantificirane tvrdnje jeste u tome, što metodom SLDNF-resolucije pokušavamo iz nekog skupa premisa S deducirati dati cilj tako, da pokažemo nekonsistentnost skupa S U {G}, gdje je G zapravo negirana tvrdnja iz cilja. U praktičkim terminima rečeno, SLDNF-resolucija, kao metoda automatske dedukcije, provjerava deducibilnost nekog cilja G iz datog skupa premisa (tj. definitnih klauzula) S. Pritom da bi se deducirao odgovor na upit oblika:

"Koji su sve x takovi da ... ?" (5)

upit biva transformiran u tvrdnju:

"Ne postoji x takav da ... " (6)

dakle, u cilj iz definicije (1.2), kako je eksplicitan (shvaćen) u formuli (4). Sada pak SLDNF-resolucija pokušava pokazati nekonsistentnost skupa premisa kojem je dodana negirana egzistencijalno kvantificirana tvrdnja (6). Ukoliko takav dokaz (zovemo ga SLDNF-opovrgnuće) uspije, onda su one vrijednosti varijable x iz (6), za koje je dokaz uspio, ujedno i odgovori na upit (5). Dakle, iako je uobičajeno govoriti o "dedukciji odgovora", ovdje se zapravo radi o "izračunavanju odgovor supstitucije", za dati upit, a ne o dedukciji (općenitih) formula, u standardnom značenju toga pojma. Više o tome rečeno je u odjeljku (3.4).

Opis same procedure SLDNF-opovrgnuća (resolucije) dat je u <Llo 84>, gdje su ujedno date definicije temeljnih pojmova iz logičkog programiranja, koje ovdje koristimo.

2. O POTPUNOSTI SLDNF-RESOLUCIJE

Sistem klauzalne logike smatramo potpunim ukoliko je za dati skup premisa S i cilj G, svaka ispravna odgovor supstitucija ujedno i izračunata (odnosno izračunljiva) odgovor supstitucija.

U <Llo 84>, str.84 - 85, nalazimo primjer skupa definitnih klauzula (premisa):

(a) p(x) <--
(b) q(a) <--
(c) r(b) <--

i cilja:

<-- p(x) & n(q(x)). (1)

Pomoću tog primjera Lloyd pokazuje nepotpunost SLDNF-resolucije za jezik definitnih klauzula, napominjući pritom da je nalaženje "nekog oblika potpunosti" od "urgentne prioritete". Naime, obzirom da u jeziku klauzalne logike izražavamo znanja u bazi, razumljivo je da je potpunost sistema od esencijalnog značaja za njegovu upotrebljivost.

Nepotpunost dedukcije za dati primjer slijedi iz toga što je supstitucija {x/b} ispravna odgovor supstitucija, ali ne i izračunata odgovor supstitucija. Dakle vrijedi:

comp(S) |= p(b) & n(q(b)) (2)

ali na upit (1) ne uspjevamo dobiti izračunatu odgovor supstituciju {x/b}, iz čega bi proizašlo da u klauzalnom sistemu sa SLDNF-resolucijom kao metodom deduciranja, ne vrijedi:

S |- p(b) & n(q(b)). (3)

Obrazložimo ukratko tvrdnje (2) i (3). Iz definicije jezika definitnih klauzula očito je da iz same teorije S ne može biti deducibilan nijedan negativan literal. Stoga je pri SLDNF-resoluciji je za dedukciju negativnih literala usvojeno nemonotono pravilo izvođenja nazvano "negacija kao konačan neuspjeh dedukcije" (negation as finite failure - NF). Prema tom pravilu, ako je za dati skup klauzula S i cilj '<-- p' pripadno SLDNF-drvo konačno i bez ijedne grane uspjeha, onda zaključujemo S |- n(p). Međutim, kako sama teorija S, date u klauzalnom jeziku, ne može imati negativne literale za logičke posljedice, pouzdanost pravila 'NF' "spašava" se uvođenjem nadopune (completition) comp(S) za teoriju S. Grubo rečeno, comp (S) nastaje iz teorije S tako da sve što iz S nije deducibilno dodamo teoriji kao negirano (tj. neistinito); za formalnu definiciju comp(S) vidi npr. <Llo 84>.

U promatranom primjeru, cilj '<-- q(b)' nije deducibilan iz teorije S. Utoliko i vrijedi S |- n(q(b)), a isto tako i

comp(S) |= n(q(b)) (4)

S druge strane, vrijedi i S |- p(x) jer je p(x) element skupa S. Odatle (na nivou logike prvoga reda) slijedi i S |- p(b), a time i

comp(S) |= p(b). (5)

Rezultati (4) i (5) zajedo, potvrđuju tvrdnju (2) o ispravnoj odgovor supstituciji {x/b} za cilj (1) i dati skup premisa S.

Pogledajmo sada zašto SLDNF-resolucija tu ispravnu odgovor supstituciju ne uspjeva izračunati. Prema definiciji SLDNF-derivacije, ako literal 'A' iz negativnog literala 'n(A)' ima opovrgnuće (a u našem primjeru je to literal 'q(x)' i ima opovrgnuće!), onda se iz cilja 'n(A)' ne derivira novi cilj. Utoliko i ne možemo stići do prazne klauzule za polazni cilj '<-- p(x) & n(q(x))' (tj. do opovrgnuća i izračunate odgovor supstitucije za taj cilj), već pokušaj SLDNF-opovrgnuća (tj. izračunavanja odgovor supstitucije), završava neuspjehom. A odatle i slijedi iznad iznešena tvrdnja (3), o nededucibilnosti promatranog cilja.

U nastavku dajemo prijedlog rješenja tog problema. Smatramo da razlog nemogućnosti izračunavanja ispravne odgovor supstitucije u promatranom primjeru leži u klauzuli 'p(x) <--' iz skupa premisa S. Naime, činjenica je ali nije temeljna, jer sadrži varijablu x, te se utoliko njenim uspješnim resolviranjem (ista) varijabla x iz negativnog literala 'n(q(x))' nije instancirala. Nadalje, obzirom da SLDNF-derivacija pri resolviranju sa negativnim literalima kao najopćenitiji unifikator uzima supstituciju identitete e, nije ni moguće očekivati da za postavljeni upit (1) izračunata odgovor supstitucija bude {x/b}.

Obzirom da problem leži u klauzuli 'p(x) <--', za samu problematiku predstavljanja znanja važno je pogledati što ta klauzula zapravo "znači" (predstavlja, kazuje).

Striktno govoreći, ta klauzula ne "kazuje" ništa, već je to samo dobro oblikovana klauzula (formula), prema definiciji (1.1). Do uobičajenog značenja te klauzule, tj. "svaki x posjeduje svojstvo p" ili pak "za svaki x vrijedi p(x)" dolazimo tek njenom interpretacijom. No, za interpretaciju su nam potrebne struktura i asignacija. Nadalje, definicija strukture temelji na odabiru nepraznog skupa kao domene (tj. prostora, svijeta) D. Tada pak i "svaki x ... " biva interpretiran (shvaćen) - u skladu sa semantičkom definicijom simbola '∀' - kao "svaki x iz domene D ... "!

Prema 'teorija - model' paradigmi u kontekstu problematike predstavljanja znanja, skup klauzula smatramo teorijom kojom težimo opisati neku strukturu kao dio realnog ili hipotetičkog svijeta. Obzirom da u tom slučaju pri formiranju teorije znamo o čemu govorimo - tj. struktura je unaprijed data - izražavanje znanja pomoću netemeljnih činjenica ne samo da dovodi do nepotpunosti deduktivnog sistema, već izgleda i neprimjernim. Naime, mi promatranom klauzulom 'p(x) <--' zacijelo nismo željeli izraziti znanje da "svi" posjeduju svojstvo p, već da to svojstvo posjeduju "svi iz strukture" koju teorijom opisujemo. Utoliko i znanje izraženo klauzulom 'p(x) <--' možemo - bar sa aspekta predstavljanja znanja - adekvatnije predstaviti u slijedećoj formi:

p(x) <-- element_strukture(x)

tj. "Svaki element, ako je iz domene strukture onda posjeduje svojstvo p." (U predloženom modelu logičke baze znanja učinjeno je to na prikladniji ali analogan način.)

Pokazani način transformacije netemeljnih činjenica u pravila navodi nas na definiciju podjezika jezika definitnih klauzula, u kojem će spomenuti princip već vrijediti. Takav jezik nazvali smo ovdje jezikom regularnih klauzula.

DEF 2.1.

Definitnu klauzulu

A <-- B1 & ... & Bn

nazivamo regularnom ukoliko zadovoljava slijedeće uvjete:

a) Ako je n = 0, tj. klauzula je činjenica, onda je to temeljni atom.
b) Ako je n > 0 onda:
   1) svaka varijabla koja se javlja u glavi klauzule javlja se i u bar jednom literalu tijela klauzule;
   2) svaka varijabla koja se javlja u negativnom literalu Bi tijela klauzule, javlja se i u bar jednom pozitivnom literalu Bj tijela klauzule, tako da literal Bj prethodi literalu Bi.

DEF 2.2.

Cilj

<-- B1 & ... & Bn

je regularan ako svaka varijabla koja se javlja u negativnom literalu Bi iz cilja, javlja se i u bar jednom pozitivnom literalu Bj iz cilja, tako da literal Bj predhodi literalu Bi.

Da bi pokazali da zahtjev po regularnosti jezika ne ograničava njegove izražajne mogućnosti već da samo zahtjeva pravilniju formulaciju (izražavanje) znanja, osvrnimo se ponovo na Lloydov primjer. Za dati primjer, svaka moguća supstitucija oblika {x/?} - osim, naravno, supstitucije {x/a} - je i ispravna odgovor supstitucija. Naime, obzirom da činjenice

q(c), q(d), ...

nisu deducibilne iz skupa premisa S, deducibilni su negativni literali

n(q(c)), n(q(d)), ... (6)

S druge strane, iz (neregularne) činjenice 'p(x)' slijedi i

p(c), p(d), ... (7)

a time - iz (7) i (6) - slijedi da su supstitucije

{x/c}, {x/d}, ...

ispravne (ali opet ne i izračunate!) odgovor supstitucije.

U jeziku regularnih klauzula, promatrani primjer izrazili bismo na slijedeći način:

```
p(x) <-- element_strukture(x)
q(a) <--
q(b) <--
element_strukture(a) <--
element_strukture(b) <--
```

Ispravna odgovor supstitucija {x/b}, koja za isto znanje - tj. premise, ali izražene u jeziku definitnih klauzula - nije bila izračunljiva, sada jeste izračunata odgovor supstitucija. Štoviše, prelaskom na regularnu formu klauzula i ciljeva, supstitucije {x/c}, {x/d}, ... , više nisu niti ispravne odgovor supstitucije, niti izračunate odgovor supstitucije. To pak držimo isto tako povoljnim (i poželjnim) efektom uvođenja jezika regularnih klauzula, jer se konstante c, d, ... u teoriji S ne javljaju, pa prema tome ne označavaju niti elemente iz strukture koju teorijom S težimo opisati. Ukoliko pak želimo da to postanu, tj. ukoliko želimo u teoriji S izreći neka nova saznanja o promatranoj strukturi, onda to možemo učiniti dodavanjem teoriji klauzula

```
element_strukture(c) <--
element_strukture(d) <--
        .
        .
```

Adekvatnost primjene regularnih klauzula ilustrirajmo slijedećim primjerom.

Neka struktura M, koju teorijom želimo opisati bude 'svijet živih bića'. Znanje da "sva živa bića dišu", možemo izraziti na dva načina:

(a) definitnom (ali ne i regularnom klauzulom)

```
diše(x) <--
```

(b) regularnom klauzulom

```
diše(x) <-- je_živo_biće(x)
```

Smatramo da regularnaklauzula (b) naprosto adekvatnije (prirodnije) izražava naše znanje o strukturi M, izraženo u prirodnom jeziku rečenicom "Sva živa bića dišu".

U <Kow 79>, str. 220, dat je primjer (drugačije prirode), kojim se pokazuje drugi slučaj (vid) nepotpunosti SLDNF-resolucije, kao metode izvođenja u sistemu klauzalne logike. Poglsdajmo taj slučaj.

Neka skup formula logike prvoga reda S bude:

{ n(p(a)) --> p(a) } (8)

Logičkom transformacijom (jedine) formule iz skupa S, dobivamo

{ n(n(p(a))) v p(a) }

a odakle i

{ p(a) }

Prema (8) vrijedi dakle, i

S |- p(a) (9)

No, predstavimo li formulu iz skupa S u klauzalnoj formi, tj. kao kalauzulu

p(a) <-- n(p(a)) (10)

onda iz te klauzule (uzete kao premise), prema SLDNF-resoluciji, 'p(a)' nije deducibilno!

No, ovaj slučaj (oblik) nepotpunosti bitno se razlikuje od ranije razmotrenog (Lloydovog) slučaja. Naime, dok je u prijašnjem primjeru pokušaj dedukcije bio konačan, dajući pritom pogrešan odgovor "ne", ovdje pokušaj deduciranja cilja 'p(a) <--' (tj. pokušaj SLDNF-opovrgnuća za taj cilj i klauzulu (10)), uopće ne završava, jer pripadno SLDNF-drvo nije konačno. Naime, pokušaj dedukcije cilja '<-- p(a)' dovodi do generiranja izvedenog cilja '<-- n(p(a))', koji se opet svodi na pokušaj dedukcije cilja '<-- p(a)', itd. Pored same neobičnosti znanja, koje izražava (interpretirana) formula iz skupa (8), tj. da:

Ako entitet 'a' ne posjeduje svojstvo 'p'
onda entitet 'a' posjeduje svojstvo 'p'.

očito je da se ovdje radi i o cirkularnom načinu definiranja i izražavanja znanja. Jer svojstvo 'p' za entitet 'a' definirano je u terminima tog istog svojstva za taj isti entitet.

Prijedlog rješenja problema konačnosti SLDNF-drva (i kompjutacije), dat je u <Llo 85>. Prema tom prijedlogu, sva svojstva (predikatni simboli), koji se javljaju u teoriji (tj. skupu klauzula S), razvrstavaju se u hijerarhijske nivoe. Pritom, u tijelu klauzule iz skupa S smiju nastupati samo svojstva (predikatni simboli) koji su nižeg nivoa od svojstva koje se javlja u glavi klauzule. U tom slučaju cirkularnost definicije u skupu klauzula S je sasvim onemogućena, tako da je svaki pokušaj SLDNF-opovrgnuća (tj. izračunavanja odgovor supstitucije) konačan, i završava uspjehom ili neuspjehom.

Međutim, takvo ograničenje postavljeno na jezik skupa premisa (klauzula) - iako garantira konačnost kompjutacije - izgleda isuviše restriktivnim. Naime, njime se isključuje mogućnost rekurzivnog definiranja (opisivanja), koje pak smatramo izrazito značajnim za predstavljanje znanja. Ilustrirajmo to primjerom.

Neka znanje, koje želimo predstaviti u jeziku regularnih klauzula bude:

Krvnu grupu nasljeđuje se od oca (11)

U jeziku regularnih klauzula to možemo učiniti sa:

```
krv_grupa_od(x,y) <-- otac_od(x,z) &
                      krv_grupa_od(z,y)   (12)
```

U jeziku logičke baze znanja, sintaktički smo poljepšali jezik regularnih klauzula, tako da bi znanje (11) bilo predstavljeno kao pravilo:

```
krv_grupa_od(X;Y) ako otac_od(X;Z) i
                      krv_grupa_od(Z;Y)   (13)
```

U jeziku sa hijerarhijskim uređenjem predikatnih simbola takvu tvrdnju ne bismo mogli izreći, jer se predikatni simbol 'krv_grupa_od' iz glave klauzule (pravila) javlja i u tijelu klauzule, što hijerarhijsko uređenje svojstava (predikata) ne dopušta. No, mogućnost rekurzivne definicije izgleda isuviše značajna da bismo ju jednostavno

zabranili. Jer "teoretska čistoća" modela (tj. garantirana konačnost svakog pokušaja SLDNF-opovrgnuća), koju zabranom rekurzivne definicije postižemo, izgleda ipak preslabom nadoknadom za izgubljene operativne mogućnosti baze znanja.

Zaključimo razmatranje iz ovog odjeljka opisom rješenja usvojenog (i implementiranog) u modelu logičke baze znanja, koji ovdje predlažemo.

Prilikom svake izmjene sadržaja logičke baze znanja - tj. upisa/brisanja pravila/činjenica - (automatski) se provjerava da li je time stvorena mogućnost postavljanja upita (tj. cilja) za koji bi pripadno SLDNF-drvo imalo beskonačnu granu. Ukoliko je takova mogućnost zaista stvorena, onda već u toku same provjere postojanja te mogućnosti, dolazi do prekoračenja raspoložive memorije na sistemu, što nam ujedno i služi kao znak postojanja beskonačne (ili barem operativno prevelike) grane. U implementaciji modela logičke baze, ilustriranoj primjerima u odjeljku (4), razvijena je naredba 'loop', pomoću koje prilikom nastupa prekoračenja, od sistema dobivamo odgovor koji je to cilj za koji bi, u ažuriranoj bazi znanja, pripadno SLDNF-drvo imalo beskonačnu granu. Na slijedeći upit - 'sh_loop' - sistem eksplicira (pokazuje) tu granu (odnosno, pokušaj SLDNF-opovrgnuća, koji ju slijedi), i to do dubine koju sami zahtjevamo. Tada je na kreatoru baze znanja (koji upis/brisanje vrši), da odluči je li zaista riječ o "očito beskonačnoj" grani SLDNF-drveta, (tj. cirkularnoj definiciji), ili pak bi bilo vrijedno ponoviti pokušaj SLDNF-opovrgnuća sa većom raspoloživom memorijom. Ukoliko potonje nije slučaj (a u pravilu nije!), onda se zahtjeva preformulacija znanja u bazi, tako da se cirkularnost izbjegne, a time i postigne konačnost svake moguće grane SLDNF-drveta odnosno svakog pokušaja SLDNF-opovrgnuća za neki dati cilj (upit) postavljen skupu premisa (tj. logičkoj bazi znanja).

Problem cirkularnosti definiranja je jedan od važnih (i otvorenih) problema logičkog programiranja, posebno u kontekstu značaja koji ima rekurzija (ali i potpunost!). Držimo, da ovdje dato operativno rješenje jeste zadovoljavajuće, posebno u kontekstu problematike predstavljanja i dedukcije znanja. Naime, njime se zadržava mogućnost rekurzivnog definiranja a ujedno i smjesta otkriva i eksplicira eventualno postojanje beskonačne (točnije: prevelike) grane. Više od toga (osim isuviše rigoroznim restrikcijama jezika!), u kontekstu neodlučivosti logike te primjene pravila "negacija kao konačan neuspjeh dedukcije", ne izgleda dosežnim.

3. MODEL LOGIČKE BAZE ZNANJA

Modelom logičke baze znanja dat je cjelovit prijedlog načina predstavljanja znanja u jeziku logičke baze znanja, sa SLDNF-rezolucijom, kao metodom deduciranja (odnosno izračunavanja) odgovora iz baze. Da bi (formalna) dedukcija u bazi (implicitno) sadržanih znanja bila moguća, moraju znanja u bazi biti data u jeziku precizne sintakse i semantike. S druge strane, obzirom da se u tom jeziku izražavaju znanja koja potječu od čovjeka i čovjeku služe, poželjno je da taj jezik bude ujedno i blizak prirodnom jeziku. Stoga smo jezikom logičke baze nazvali jezik regularnih klauzula u kojem su izvršene izmjene na nivou sintakse (točnije: abecede), i to sa ciljem da se jezik učini bliži prirodnom jeziku. U tu svrhu smo logičke simbole iz jezika regularnih klauzula zamijenili njihovim uobičajenim ekvivalentima (interpretacijama) u prirodnom jeziku. U nastavku demo, kada bude riječ o 'znanju', govoriti u terminima i notaciji jezika logičke baze znanja. Kada pak budemo nad tim 'znanjem' izvodili neke logičke transformacije (dedukcije, dokaze), činiti demo to u notaciji i terminologiji jezika regularnih klauzula.

U jezik logičke baze znanja uvodimo i disjunkciju, i to na slijedeći način: Neka su

'glava' ako 'tijelo_1' (1)

i

'glava' ako 'tijelo_2' (2)

pravila iz jezika logičke baze znanja. Tada je i

'glava' ako 'tijelo_1' ili 'tijelo_2' (3)

pravilo jezika logičke baze znanja. Drugim riječima, disjunkciju u jezik logičke baze uvodimo kao skraćenu notaciju za dva (ili više) pravila sa identičnim glavama. Da je zaista riječ samo o notacijskoj varijanti slijdi iz toga što je konjunkcija pravila (1) i (2) logički ekvivalentna pravilu (3).

Za predstavljanje atomarnih znanja predlažemo slijedeću shemu:

Svojstvo(Entitet:Vrijednost) (4)

U shemi (4) 'Svojstvo' ima istu ulogu kao i dvomjesni predikatni simbol u logici prvega reda, odnosno 'ime relacije' u relacijskoj shemi. U <Rad 86b>, umjesto izraza 'svojstvo' korišten je izraz 'opis' (description), uz napomenu da taj izraz "zvuči suviše sintaktički". U stvari, držimo da bi na nivou samoga jezika (sintakse), termin 'opis' izgledao prikladnijim, dok na nivou modela (strukture), koju tim jezikom opisujemo, adekvatnijim izgleda termin 'svojstvo'. Obzirom da je cilj logičke baze znanja (kao teorije), da opiše unapred danu strukturu, možemo smatrati da su predikatni simboli iz jezika već a priori interpretirani, te nam govoriti u terminima 'svojstava' izgleda prikladnijim.

U ulozi 'Entiteta' može se pojaviti bilo koji pojam koji označava neki entitet iz strukture, kojeg želimo opisati u terminima pridruženih mu svojstava i pripadnih vrijednosti.

'Vrijednost' iz sheme izražava vrijednost promatranog svojstva za dati entitet. Vrijednost može biti izražena numerički ili nekim atributom (pojmom, jezičkim izrazom). Na primjer u

broj_kotača_od(auto:4)

vrijednost svojstva 'broj_kotača_od' za entitet 'auto' data je numerički. S druge strane, u primjerima

pušač(petar:strastven)
pušač(ivan:umjeren)

vrijednosti su date jezičkim izrazima.

Analizom razloga upotrebe upravo (i samo) binarnih predikata (svojstava) ovdje se ne bavimo; prikaz osnove te problematike dat je u <Kow 79>.

3.1. Sadržaj baze znanja

Sadržaj logičke baze znanja sačinjavaju tri komponente:
- pozitivno znanje
- hijerarhija entiteta
- negativno znanje

Sve tri komponente čine informacijski sadržaj logičke baze, tj. sadržavaju iskaze baze znanja, shvaćene kao logičke teorije. Podjela sadržaja baze u tri komponente uvjetovana je kako samom različitošću iskaza (znanja) tako i specifičnostima SLDNF-rezolucije, kao metode deduciranja pomoću koje se generiraju (deduciraju) odgovori iz baze znanja.

a) Pozitivno znanje

Pozitivno znanje sačinjavaju ona znanja koja su izrazljiva pravilima i činjenicama iz jezika logičke baze znanja. Tako bismo na primjer (hipotetičku) zakonitost:

Svatko nasljeđuje boju očiju od majke

u jeziku logičke baze mogli predstaviti (izreći) pravilom:

boja_oči_od(X:Y) ako majka_od(X:Z) i
boja_oči_od(Z:Y). (5)

Nadalje, činjenice poput:

Ana ima smeđe oči

možemo u jeziku logičke baze izraziti sa:

boja_oči_od(ana:smeđa).

Znanja, koja na analogan način izražavamo pomoću pravila i činjenica u jeziku logičke baze, nazvali smo pozitivnim znanjima. Ta znanja iskazuju neka svojstva entiteta, tj. iskazivanjem takovih znanja entitetima pridružujemo svojstva sa datim vrijednostima, čime se entiteti pozitivno određuju.

Napomenimo da smo u pravilu (5), u skladu sa zahtjevima standardnog Prolog interpretora u kojem je model implementiran, varijable predstavili velikim slovima (X,Y,Z).

b) Hijerarhija entiteta

Hijerarhijska struktura definirana je tzv. tipizacijom entiteta, čime je stvorena mogućnost nasljeđivanja svojstava među entitetima u strukturi. Entitete tipiziramo upotrebom svojstva 'vrsta_od'. Na primjer, činjenicu (pozitivno znanje) da:

Ivan je čovjek (6)

predstaviti ćemo u jeziku baze činjenicom:

vrsta_od(ivan:čovjek).

Svojstvo 'vrsta_od' mora za svaki entitet iz baze znanja biti jedinstveno, stime da ugradnja entiteta u hijerarhijsku strukturu nije obavezna, tj. svojstvo 'vrsta_od' ne mora biti uopće dato za entitet. Naravno, u tom slučaju netipiziran entitet neće nasljeđivati nikakva svojstva, jer za njega u bazi znanja formalno i ne postoji viših entiteta. Svojstvom 'vrsta_od' uređuju se entiteti "po vrstama", što ulogu toga svojstva čini specifičnom. Stoga i znanje:

Ivan je vozač

koje - na nivou prirodnog jezika - može izgledati analogno znanju (6), nećemo izražavati u terminima svojstva 'vrsta_od', već npr. sa:

zanimanje_od(ivan:vozač)

Pored svojstva 'vrsta_od', na hijerarhijsko uređenje entiteta odnosi se i svojstvo 'je', kojem je ovdje dato nezavisno i različito značenje od svojstva 'je', korištenog u formalizmu semantičkih mreža. Ovdje je to svojstvo definirano (i implentirano na sistemu), na slijedeći način:

a) je(e:e)

za svaki entitet e iz baze znanja;

b) je(e1:e2)

ako postoji takav entitet e3, da
'vrsta_od(e1:e3)' i 'je(e3:e2)'.

Drugim riječima, svaki entitet 'je' on sam; isto tako, svaki entitet e1 ima svojstvo da 'je' i e2, ako se entitet e2 nalazi u hijerarhiji iznad entiteta e1.

Svojstvo 'je' koristimo kod izražavanja pravila poput:

diše(X:da) ako je(X:živo_biće).

čime u logičkoj bazi iskazujemo (predstavljamo) znanje:

Svako živo biće (a i
"živa bića općenito") diše

U implementaciji modela svojstvo 'je' nazivamo rezerviranim jer se ono ne definira eksplicitno za pojedini entitet već je to svojstvo implicitno dato posredstvom definicija svojstava 'vrsta_od'.

c) Negativno znanje

Znanja, kojima se odriču neka svojstava entitetima, nazvali smo negativnim znanjima. Takova znanja nisu izrazljiva pravilima i/ili činjenicama iz jezika logičke baze znanja. Na primjer, znanje:

Nitko nema crvene oči (7)

ne možemo direktno izraziti definitnom (ni regularnom) klauzulom. Naime, znanje (1) možemo u jeziku logike prvoga reda izraziti kao:

n(Ex(boja_oči_od(x,crvena))) (8)

dakle,

Ne postoji takav individual x, za koji bi vrijedilo da je boja njegovih očiju crvena.

Međutim, formula (8) egzistencijalno je kvantificirana, te ju - na putu prema klauzalnoj formi - moramo preformulirati tako da ju prevedemo u univerzualno kvantificiranu formulu. Formula (8) logički je ekvivalentna formuli

Vx(n(boja_oči_od(x,crvena))) (9)

Izostavljanjem (eksplicitno datog) univerzalnog kvantifikatora iz (9) dobivamo

```
n(boja_oči_od(x,crvena))                    (10)
```

Međutim, prema definiciji (1.1), izraz (10) nije definitna klauzula jer je to negativan literal.

Mogućnost da se takova (tj. negativna) znanja ipak izraze u jeziku logičke baze jeste da se to učini pomoću cilja (upita). Naime, prema definiciji (1.2) i pripadnom obrazloženju, formula (8) ima točno oblik cilja. Stoga ćemo znanje (7) izraženo formulom (8) izraziti regularnim ciljem (odnosno upitom iz jezika logičke baze):

```
?- boja_oči_od(X:crvena)                    (11)
```

Upit (11) jeste regularan jer u njemu negacija uopće ne nastupa.

Istaknimo ovdje razliku između 'negativnog znanja' i 'neznanja'. Obzirom da je u SLDNF-resoluciji za negaciju već usvojeno pravilo izvođenja nazvano "konačan neuspijeh dedukcije", negativna znanja ne bi ni trebalo predstavljati u bazi znanja.

Naime, na upit

```
? - boja_oči_od(X:crvena).                  (12)
```

postavljen bazi znanja - po SLDNF-resoluciji - odgovor će glasiti "ne", obzirom da za nijedan entitet c (kao moguću instancu varijable X), iz baze znanja nije deducibilno:

```
boja_oči_od(c:crvena).
```

Drugim riječima, za (neko dato) stanje baze znanja i upit (11), ne postoji ispravna odgovor supstitucija, pa stoga ni izračunata odgovor supstitucija. Međutim, ukoliko bismo u toku razvoja (ažuriranja) baze znanja unijeli, na primjer znanje

```
boja_oči_od(marko:crvena)                   (13)
```

onda bi odgovor na upit (11) glasio "marko".

No, ako znamo da "nitko nema crvene oči" (tj. (7) odnosno (8)), onda - u kontekstu toga znanja - znanje (13) nije smijelo biti unešeno u bazu. Jer baza znanja, koja sadrži znanja (8) i (13) jeste - kao logička teorija - nekonsistentna. Naime, iz (13) bi slijedilo:

```
Ex(boja_oči_od(x,crvena))                   (14)
```

što zajedno sa (8) dokazuje nekonsistentnost logičke baze kao teorije, jer je iz nje deducibilno A i n(A), tj. formule (8) i (14).

Kako navedeni primjer pokazuje, iako je samo 'neznanje' (tj. neizražavanje znanja) dovoljno za generiranje negativnih odgovora, ono nije dovoljno i za spriječavanje da se u logičku bazu unesu znanja, koja to nisu! - tj. greške, poput "znanja" (13). S druge strane, izražavanje negativnih znanja u logičkoj bazi (u obliku ciljeva (upita)), omogućava da unos (ili uopće deducibilnost) pogrešnih "znanja" (tj. grešaka) učini logičku bazu znanja (tj. teoriju), nekonsistentnom. Negativna znanja, izražena pomoću upita, smatrati ćemo stoga uvjetima integriteta logičke baze znanja. Ta znanja ne učestvuju pri samoj dedukciji odgovora (jer je negacija kao 'konačan neuspijeh dedukcije' za to dovoljna), već štite bazu znanja pred takovim promjenama (ažuriranjima), koje bi istu učinila logički nekonsistentnom.

Općenito, uvjeti integriteta, izraženi kao upiti, iskazauju negativna znanja (formule) oblika:

```
n(Ex1Ex2...Exj ('elementi cilja'))          (15)
```

Formula oblika (15) istinita je u strukturi koju znanjem iz logičke baze opisujemo, ako i samo ako ne postoje instance varijabli x1,x2, ..., xj, za koje bi formula 'elementi cilja' bila istinita u promatranoj strukturi.

Formulu oblika (15), tj. uvjet integriteta, općenito izražavamo u bazi znanja (upitom):

```
?- 'elementi cilja'                         (16)
```

Tada zadovoljivost upita (16) - dakle, bilo koja izračunata odgovor supstitucija različita od "ne" - ujedno implicira deducibilnost formule 'elementi cilja', i to za one instance varijabli koje su dobivene kao izračunata odgovor supstitucija za upit (16). To pak, onda općenito znači i deducibilnost formule:

```
Ex1Ex2 ... Exj ('elementi cilja')           (17)
```

što zajedno sa formulom (15) pokazuje nekonsistentnost baze znanja, promatrane kao logičke teorije. Na taj način je i pojam integriteta baze znanja konzekventno preveden u logičku terminologiju, i sveden na pojam konsistentnosti teorije.

Primjerom logičke baze znanja, datom u odjeljku (4), ilustrirani su neki od mogućih načina i slučajeva izražavanja negativnih znanja kao uvjeta integriteta.

Radi ilustracije, samoga principa predstavljanja znanja pomoću uvjeta integriteta, pokažimo ovdje kako možemo njihovom upotrebom adekvatno i jednostavno riješiti (famozni!) problem predstavljanja znanja:

```
Sve ptice lete osim pingvina                (18)
```

Pozitivno znanje sadržano u rečenici (18) - tj. da sve ptice koje nisu pingvini, lete - izraziti ćemo u jeziku logičke baze pravilom:

```
leti(X:da) ako je(X:ptica) i
                  ni_je(X:pingvin).         (19)
```

Pri tome je svojstvo 'ni_je' (po definiciji) ekvivalentno negaciji svojstva 'je' (tj. 'nije je') - a dato je kao način sintaktičkog uljepšavanja jezika.

Negativno znanje, tj. da "svaka ptica, koja je pingvin, ne leti", rečenicom (18) nije zapravo niti izrečeno, ali - kao što je to često slučaj u prirodnom jeziku - podrazumijeva se. Naravno, primjenom pravila (19), znanje

```
leti(pingvin:da).                           (20)
```

neće nikad biti deducibilno. Međutim, samo pravilo (19) ne može spriječiti da (20) postane deducibilno na neki drugi način. No, možemo to učiniti tako da negativno znanje, tj. "pingvin ne leti", izrazimo uvjetom integriteta (upitom)

```
?- leti(pingvin:da).                        (21)
```

Ažuriranje baze znanja, koje bi dovelo do deducibilnosti "znanja" (20), učinilo bi ujedno - zahvaljujući 'negativnom znanju' (21) - da baza znanja postane nekonsistentnom, kako je to iznad opisano. Stoga sistem logičke baze takovo ažuriranje i ne dopušta. To pak znači da ćemo - prema pravilu (19) - za svaki entitet koji "je ptica", osim za one koji su pingvini, moći deducirati "da leti". S druge strane, uvjet integriteta (21) onemogućavati će svako ažuriranje baze zna-

nja, koje bi imalo za posljedicu da "letí pingvin" postane deducibilno.

3.2. Nasljeđivanje svojstava

Ovdje ćemo ilustrirati mogućnosti izražavanja znanja, tj. pripisivanja svojstava entitetima, iz hijerarhijske strukture, tako da ta svojstva nasljeđuju ili ne nasljeđuju podređeni entiteti u strukturi, u zavisnosti od načina na koji su ta znanja izražena. U tu svrhu analizirati ćemo kako, uz adekvatno izražavanje znanja, logički konsistentna baza može sadržavati skup znanja poput:

ljudi vole životinje (22)
ljudi ne vole zmije (23)
zmije su životinje (24)

Znanje (22) možemo - kao i obično, kada je o prirodnom jeziku riječ! - shvatiti a onda i formalno (precizno) izraziti u jeziku logičke baze, na više različitih načina. Učinimo to najprije činjenicom:

```
voli(čovjek:životinja).                    (25)
```

Činjenica (25) interpretira znanje (22) kao:

Entitet 'čovjek' (kao klasa) voli entitet 'životinja' (kao klasu). (26)

Činjenicom (26) nismo ništa ustvrdili o pojedinim ljudima (tj. entitetima koji su 'vrste' čovjek), niti pojedinim životinjama.

Nadalje, znanje (22) možemo shvatiti i na način kako je to precizno izrečeno slijedećim pravilom:

```
voli(X:životinja) ako je(X:čovjek).        (27)
```

Time je znanje (22) interpretirano kao:

Entitet 'čovjek' (kao klasa), a i svaki pojedini njegov podentitet (koji 'je' čovjek), voli entitet 'životinja' (kao klasu). (28)

Interpretacija (28) znanja (22), izražena pravilom (27) isto tako ne kazuje ništa o pojedinim entitetima koji su podentiteti entiteta 'životinja'.

Konačno, mogući način shvaćanja znanja (22) bio bi i slijedeći:

Entitet 'čovjek' (kao klasa),a i svaki pojedini njegov podentitet (koji 'je' čovjek), voli entitet 'životinja' (kao klasu), a i svaki pojedini njegov podentitet (koji 'je' životinja'). (29)

Interpretaciju (29) pozitivnog znanja (22) možemo u jeziku logičke baze izraziti pravilom:

```
voli(X:Y) ako je(X:čovjek) i
             je(Y:životinja).              (30)
```

Negativno znanje (23) mogli bismo isto tako interpretirati na više načina. No, odaberimo ovdje samo slijedeći način:

Nijedan entitet koji 'je' čovjek ne voli nijednog entiteta koji 'je' zmija. (31)

Znanje (23), interpretirano kako je dato u (31), izraziti ćemo u logičkoj bazi upitom:

```
?- voli(X:Y) i je(X:čovjek) i je(Y:zmija).
                                           (32)
```

I na kraju, znanje (24) izraziti ćemo činjenicom:

```
vrsta_od(zmija:životinja).                 (33)
```

Interpretacije (26) i (28) znanja (22), izražene činjenicom (25) odnosno pravilom (27), u logičkoj bazi znanja, koja usto sadrži negativno znanje (32) i znanje (33), ne dovode do nekonsistentnosti baze znanja. Naime, negativnim znanjem (32), svakom entitetu koji 'je' čovjek, odriče se svojstvo 'voli' sa vrijednošću koja 'je' zmija. No, niti činjenica (25) niti pravilo (27) to svojstvo sa takovom vrijednošću ne pripisuju entitetu koji 'je' čovjek.

S druge strane, interpretacija (29) znanja (22), izražena pravilom (30), dovodi do nekonsistentnosti logičke baze znanja. To je (i neformalno) sasvim razumljivo, obzirom da se negativnim znanjem (32) iskazuje da nitko tko 'je' čovjek ne voli nikoga tko 'je' zmija, dok se pravilom (30) iskazuje da svatko tko 'je' čovjek voli svakga tko 'je' životinja, pa prema tome i zmiju!

Sa navedena tri načina interpretiranja (shvaćanja) znanja (22) izraženog u prirodnom jeziku, željeli smo istaći nužnost precizne (i adekvatne) interpretacije i formalnog predstavljanja znanja iz prirodnog jezika u jeziku logičke baze znanja. Činjenica da znanje (22) možemo interpretiati na više načina, od kojih neki dovode i do nekonsistentnosti, ne znači da čovjek normalno komunicira i zaključuje na osnovu nekonsistentnog skupa znanja (premisa), kako se to ponekad tvrdi. Mnijenja smo, da čovjek vrlo uspješno i adekvatno interpretira znanja izražena prirodnim jezikom (i to najčešće u zavisnosti od nekog datog konteksta), a ne da zaključuje iz nekonsistentnog skupa premisa.

Predloženim načinima pripisivanja svojstava i vrijednosti entitetima, kako je to opisano u ovom odjeljku, željeli smo omogućiti izražavanje znanja, na način koji bi adekvatno odražavao mehanizam pridruživanja i (ne)nasljeđivanja svojstava kakav (ma da neformalno) postoji u prirodnom jeziku.

3.3. Ekvivalencija

U prirodnom jeziku ekvivalenciju obično iskazujemo izrazom "ako i samo ako", iako valja istaći da u prirodnom jeziku ekvivalenciju mnogo češće mislimo (podrazumijevamo) nego što ju eksplicitno iskazujemo. U ovom odjeljku data je analiza dviju interpretacija (shvaćanja) ekvivalencije kao i prijedlog načina da se ekvivalencija (odnosno, jedna od njenih interpretacija), izrazi u modelu logičke baze znanja.

Rečenicu

p(x) ako i samo ako q(x) (34)

izraziti ćemo u logici prvoga reda formulom

p(x) <--> q(x) (35)

gdje je simbol '<-->' najčešće definiran na metanivou sistema (dakle, znak '=' pripada metajeziku), kao:

```
p(x) <--> q(x) =
     (p(x) --> q(x) & q(x) --> p(x))    (36)
```

Formulu (35) ne možemo direktno zapisati u jeziku logičke baze znanja, jer ne raspolažemo izrazom (simbolom) 'ako i samo ako', već samo sa implikativnim veznikom 'ako'. S druge strane, pokušaj da ekvivalenciju (35) izrazimo tako da u logičkoj bazi izrazimo desnu stranu definicije (36), tj. regularne klauzule:

```
p(x) <-- q(x)                          (37)
q(x) <-- p(x)                          (38)
```

isto tako ne uspijeva. Naime, u tom slučaju imali bismo očigledan primjer cirkularne definicije tj. p(x) pomoću q(x), a pritom i q(x) pomoću p(x). Naravno, tada bi npr. cilj

```
<-- p(x)                               (39)
```

zajedno sa premisama (37) i (38) imao SLDNF-drvo sa beskonačnom granom, što smo već okarakterizirali kao neprihvatljivo. No, time mogućnost izražavanja ekvivalencije u logičkoj bazi znanja još nije sasvim izgubljena. Naime, značenje (ulogu) ekvivalencije - posebno u kontekstu predstavljanja znanja - možemo shvatiti na dva različita načina:

a) Uspostavljanje ekvivalencije

U tom slučaju, koji smo nazvali 'uspostavljanje ekvivalencije', ukoliko u skup premisa koji sadrži formulu (35) dodamo činjenicu

```
p(a) <--                               (40)
```

deducibilnom će postati i formula (činjenica):

```
q(a) <--                               (41)
```

Deducibilnost činjenice (41) slijedi iz klauzule (38), koja je - prema definiciji (36) - "dio" (ili točnije: slijedi iz) formule (35), te činjenice (40). Na analogan način bi dodavanje činjenice (41), posredstvom klauzule (37), deducibilnom učinilo činjenicu (40). No, kako je već istaknuto, takovo predstavljanje ekvivalencije ne možemo primjeniti u logičkoj bazi, jer ono zahtjeva prisustvo formula (37) i (38) u bazi, što pak znači postojanje SLDNF-drva sa beskonačnom granom.

b) Održavanje ekvivalencije

Drugi način shvaćanja (i predstavljanja) ekvivalencije jeste takav da u logičkoj bazi "jedan smjer" iz ekvivalencije (35) - u skladu sa definicijom (36) - izrazimo klauzulom (tj. implikativnom formulom), dok se "drugi smjer" izvorne ekvivalencije iz (35) održava, i to zabranom unosa/brisanja onih znanja, koja bi dovela do narušavanja ekvivalencije koju tvrdi (zahtjeva) formula (35). U svrhu iskazivanja zabrane upotrijebiti ćemo cilj (upit), tj. uvjet integriteta.

Shvaćanja ekvivalencije iznesena u (a) i (b) pokušajmo dalje eksplicirati slijedećim opisima:

(a') A je istinito ako i samo ako B je istinito. Nije poznato da bi B bilo istinito. Međutim, poznato je da je A istinito. Slijedi: i B mora biti istinito.

(b') A je istinito ako i samo ako B je istinito. Nije poznato da bi B bilo istinito. Međutim, postoji tvrdnja (dedukcija) da A jeste istinito. Zaključujemo: tvrdnja A je neprihvatljiva.

Shvaćanje ekvivalencije prema (a') izgleda češćim (uobičajenim), barem pri hipotetičkom zaključivanju, u kojem ne sumljamo u nijednu od premisa, pa prema tome nemamo ni razloga za njeno odbacivanje. No, kako je iznad obrazloženo, takvo shvaćanje ekvivalencije ne možemo izraziti (predstaviti) u modelu logičke baze.

S druge strane, u kontekstu problematike predstavljanja i procesiranja znanja, zaključivanje na način kako je to pokazano u (b'), može biti vrlo primjenljivo i korisno. Naime, u realnim situacijama, deducibilnost neke informacije (tj. znanja) iz baze znanja može biti znak da neka od premisa nije istinita. Utoliko nam i mogućnost predstavljanja znanja i zaključivanja u skladu sa interpretacijom ekvivalencije (b') izgleda značajnim korakom u smjeru "više od same inplikacije".

Pokažimo sada kako ekvivalenciju izrečenu sa (35) i shvaćenu u skladu sa (b'), predstavljamo u logičkoj bazi znanja. Kao znanje koje ćemo predstaviti pravilom, odaberimo regularnu klauzulu (37), tj:

```
p(x) <-- q(x)
```

Klauzulu (38) - koju pored klauzule (37)! - ne smijemo unijeti u pozitivan dio znanja logičke baze, prevesti ćemo u uvjet integriteta kako slijedi.

Klauzula (38), zajedno sa (implicitno mišljenom) univerzalnim kvantifikatorom, glasi:

```
Vx(p(x) --> q(x))                      (42)
```

Transformacijom formule (42) dobivamo

```
Vx(n(p(x)) v q(x))
```

i dalje,

```
n(Ex(n(n(p(x)) v q(x))))
```

što konačno (po de Morganu), daje:

```
n(Ex(p(x) & n(q(x))))                  (43)
```

Formula (43) ima točno oblik slijedećeg cilja:

```
<-- p(x) & n(q(x))                     (44)
```

ili - u jeziku logičke baze - uvjeta integriteta:

```
?- p(x) i nije q(x).                   (45)
```

Uvjet integriteta (45) zabranjuje da se baza znanja mijenja tako, da iz nje bude deducibilno p(c) za neku instancu c varijable x, za koju nije deducibilno q(c).

Obzirom da je uvjet integriteta (45) dobiven logičkim transformacijama pravila (38), mogli bismo reći da smo u logičkoj bazi (ipak!) uspjeli izraziti ekvivalenciju (35). Međutim, uvjet integriteta (45) može stanje baze samo kontrolirati (a posredstvom sistema za upravljanje bazom i automatski održavati). Stoga se iz njega (tj. njegovim posredstvom), ne mogu deducirati nova znanja u bazi, pa je utoliko i moguće ekvivalenciju predstavljati samo u skladu sa njenim shvaćanjem opisaim u (b').

Uz prisustvo uvjeta integriteta (45), ažuriranje baze, koje bi dovelo do deducibilnosti činjenice

```
p(a)
```

a da pritom nije deducibilna i činjenica

```
q(a)                                   (46)
```

nedopušteno je, jer dovodi do nekonsistentnosti baze kao teorije. Naime, kada (46) nije deducibilno, deducibilnim postaje - prema negaciji kao 'konačnom neuspijehu dedukcije':

```
n(q(a))                                          (47)
```

Tada pak iz (46) i (47) slijedi:

```
p(a) & n(q(a))
```

a time i

```
Ex(p(x) & n(q(x)))
```

što, zajedno sa uvjetom integriteta (45) - odnosno formulom (43), kao njegovim ekvivalentom u logici prvoga reda, pokazuje nekonsistentnost logičke baze znanja.

U primjeru implementacije modela logičke baze znanja, izražavanje ekvivalencije ilustrirano je primjerom predstavljanja slijedećeg znanja:

```
Netko je dobra zdravlja ako i samo ako je
fizički aktivan i nije strastven pušač.
                                                 (48)
```

Rečenicu odnosno znanje (48) mogli bismu u bazi predstaviti pravilom:

```
zdravlje_od(X:dobro) ako fiz_aktivan(X:da)
                 i nije pušač(X:strastven).
                                                 (49)
```

i uvjetom integriteta (upitom):

```
?- ( zdravlje_od(X:dobro) i
                      nije fiz_aktivan(X:da) )
                           ili
   ( zdravlje_od(X:dobro) i
                        pušač(X:strastven) ).
                                                 (50)
```

Prema upitu (50), integritet logičke baze bio bi prekršen - a time i baza učinjena logički nekonsistentnom - ukoliko bi iz nje, za neku instancu c varijable X, bilo deducibilno

```
zdravlje_od(c:dobro)                             (51)
```

a pritom ne bi bilo deducibilno

```
fiz_aktivan(c:da)
```

ili pak ukoliko bi bilo deducibilno (51), ali bi pritom bilo deducibilno i

```
pušač(c:strastven)
```

Upit (50) nije jedini način da se dati uvjet integriteta izrazi. Jednostavniji način da se to učini bio bi sa slijedeća dva upita:

```
?- zdravlje_od(X:dobro) i nije
                    fiz_aktivan(X:da).           (52)

?- zdravlje_od(X:dobro) i
              pušač(X:strastven).                (53)
```

Konjunkcija upita (tj. negiranih egzistencijalno kvantificiranih formula) (52) i (53), logički je ekvivalentna upitu (50).

Općenito, upite (tj. uvjete integriteta), kao i pravila izražavati ćemo na način (tj. u obliku) za koji smatramo da najprirodnije (najčitkije) izražava neko dato znanje. Pritom nije mišljeno da se do uvjeta itegriteta dolazi formalnim logičkim transformacijama, kako je to ovdje učinjeno za klauzulu (38). Naime, direktno izražavanje "jednog smjera" ekvivalencije u obliku uvjeta integriteta, izgleda nam dosta jednostavnim, što potvrđuje navedeni primjeri (52) i (53).

3.4. Redundantnost

Ekstenzijom baze znanja nazivamo skup svih činjenica, deducibilnih iz baze. Za pravilo (ili činjenicu) kažemo da je redundantno, ukoliko se njegovim upisom (ili brisanjem), ekstenzija baze znanja ne mijenja. Drugim riječima, sve što je iz baze deducibilno, deducibilno je i bez tog pravila (ili činjenice). Pravilo (ili činjenica) može biti redundantno u trenutku upisa; isto tako, redundantnim može postati zbog upisa ili brisanja nekog drugog pravila (ili činjenice).

Način provjeravanja redundance u logičkoj bazi znanja ilustrirajmo slijedećim primjerom. Neka skup premisa (tj. baza znanja) S bude:

```
p(a) <--
q(a) <--
```

Pretpostavimo da u bazu znanja želimo upisati pravilo

```
q(x) <-- p(x)                                    (54)
```

U notaciji logike prvoga reda, pravilo glasi:

```
Vx(p(x) --> q(x))                                (55)
```

Formula (55) logički je ekvivalentna formuli

```
nEx(p(x) & n(q(x)))                              (56)
```

koja ima točno oblik upita (cilja):

```
?- p(x) i nije q(x).                             (57)
```

Za dato stanje baze znanja, odgovor na upit (57) glasi "ne". Pri tom odgovor "ne" znači da iz skupa premisa nije deducibilno 'p(c)' za nijednu instancu c varijable x, za koju ne bi bilo deducibilno i 'q(c)'. Utoliko i dodavanje pravila (54) bazi znanja ne bi promjenilo ekstenzju baze, te stoga kažemo da je redundantno. No, to ne znači da je sama formula (54) (ili pak (55)) deducibilna iz datog skupa premisa, već samo da njenim upisom - iz date baze znanja - ništa nova ne bi postalo deducibilnim. Naravno, naknadna mijenjanja baze znanja mogu učiniti da redundantna formula (pravilo) to više nije, kako je to ilustrirano primjerima (4) i (5). Zato i govorimo o izračunavanju odgovor supstitucija odnosno o dedukciji na nivou ekstenzije, a ne o (logičkoj) dedukciji u značenju koje taj pojam ima u kontekstu sistema standardne logike prvoga reda.

Ukoliko pravilo (54) usprkos redundantnosti upišemo u bazu znanja, onda će činjenica 'q(a) <--' postati redundantnom. Naime, biti će deducibilna pomoću upisanog pravila i činjenice 'p(a) <--', te utoliko njenim brisanjem iz baze znanja ukupna ekstenzija baze ne će biti smanjena (promjenjena).

U testnoj implementaciji modela dopušteno je upisivanje redundantnih znanja, kao i zadržavanje u bazi onih znanja, koja su redundantnima postala. Sistem pritom upozorava na eventualnu redundancu (i stvorenu redundancu), a na korisniku baze znanja je da odluči što sa tim (redundantnim) znanjima učiniti.

Specifičan problem javlja se pri pokušaju provjere redundance negativnih znanja. Naime, kako je iznad opisano, redundanca u bazi znanja definirana je na osnovu ekstenzija. S druge strane, odgovor na upite koji su uvjeti integriteta mora - za konsistentnu bazu - uvjek glasiti "ne". Mogli bismo stoga reći da je ekstenzija - ili skup izračunatih odgovor supstitucija - za te upite uvjek prazan. To pak ujedno znači da otkrivanje redundance na temelju ekstenzija, kako je to definirano (i implementirano) za pozitivna znanja, kod uvjeta integriteta nije moguće. Štoviše, obzirom da se dedukcija metodom SLDNF-resolucije svodi na izračunavanje odgovor supstitucija, usporedba upita koji imaju praznu ekstenziju ne dade se teoretski dovoljno konzekventno ugraditi u sam logički sistem, zasnovan na klauzalnoj logici i SLDNF-resoluciji kao pravilu izvođenja. No, problem otkrivanja redundance među uvjetima integriteta manje je značajan od problema omogućavanja adekvatnog izražavanja samih uvjeta integriteta i nalaženja efikasnog načina provjere njihovog (ne)važenja u datoj bazi znanja. Ti su pak problemi (tj. izražavanje i provjera) uspješno rješeni (teorijski i implementacijski), u kontekstu predloženog modela logičke baze.

Iz navedenih razloga, u datoj implementaciji modela, ilustriranoj primjerima u odjeljku (4), kontrola redundance odnosi samo na pozitivna znanja (tj. pravila i činjenice).

4. ILUSTRACIJA RADA SISTEMA

U svrhu ilustracije modela logičke baze znanja, navedimo nekoliko primjera rada sistema za upravljanje logičkom bazom. Implementacija sistema izvedena je u C-Prologu. Neka stanje baze znanja bude slijedeće:

a) Pozitivno znanje

```
krv_gr_od(petar:ab).
krv_gr_od(X:Y) ako otac_od(X:Z)
                   i krv_gr_od(Z:Y).

otac_od(ivo:petar).

majka_od(mara:ana).

boja_oči_od(ana:smeđa).
boja_oči_od(X:Y) ako majka_od(X:Z)
                     i boja_oči_od(Z:Y).

zdravlje_od(X:dobro) ako fiz_aktivan(X:da) i
                         nije pušač(X:strastven).

fiz_aktivan(jure:da).

pušač(petar:strastven).
pušač(ana:strastven).
```

b) Hijerarhija entiteta iz logičke baze.

```
vrsta_od(životinja:živo_biće).
vrsta_od(čovjek:živo_biće).
vrsta_od(ana:čovjek).
vrsta_od(petar:čovjek).
vrsta_od(mara:čovjek).
vrsta_od(jure:čovjek).
vrsta_od(zmija:životinja).
```

Znanje dato u okviru definicije hijerarhije entiteta u bazi normalno se ažurira kao i ostale činjenice. Pritom, svojstvo 'vrsta_od' (ako je za entitet dato), mora biti jedinstveno za svaki entitet, o čemu sistem za upravljanje logičkom bazom vodi računa.

c) Negativo znanje

U implementaciji logičke baze znanja, upiti su imenovani. Imenovanje uvjeta integriteta izvedeno je sa ciljem optimizacije procesiranja znanja u bazi. Naime, integritet baze provjeravamo tako da pozivamo upite, kojima su uvjeti integriteta izraženi. No, obzirom da ažuriranje samo nekih svojstava može izazvati kršenje nekih uvjeta integriteta, ti se (pri provjeravanju integriteta baze), pozivaju selektivno i inkrementalno, na temelju aktualnih veza između pojedinih svojstava i pojedinih uvjeta integriteta, koje vrijede u datom stanju baze znanja. Da bi takovo pozivanje bilo moguće, svakom uvjetu integriteta pridruženo je jedinstveno ime. Inače, potrebne veze date su (i održavaju se automatski), posredstvom dvaju rječnika. Nadalje, znak "?", kojim upite općenito označavamo u jeziku logičke baze, nadomjestili smo izrazom "nije_istina_da". To je (u datom primjeru implementacije) učinjeno sa ciljem daljnjeg približavanja jezika baze znanja (i komunikacijskog jezika sistema), prirodnom jeziku. Naime, uvjet integriteta izražava negiranu egzistencijalno kvantificiranu formulu. Takovu formulu čitamo:

> Ne postoje vrijednosti za x1, ..., xj,
> takove da je izraz 'izraz' istinit.

To pak možemo (još) jednostavnije izreći kao:

> Nije istina da 'izraz'.

Imenovani i u opisanoj notaciji izraženi, uvjeti integriteta (tj. negativna znanja), kojima smo se bavili u odjeljku (3), glase:

```
ui_boja
   nije_istina_da boja_oči_od(X:crvena).

ui_zdravi
   nije_istina_da zdravlje_od(X:dobro) i
                              pušač(X:strastven).

ui_zdrav2
   nije_istina_da zdravlje_od(X:dobro) i
                          nije fiz_aktivan(X:da).

ui_voli
   nije_istina_da voli(X:Y) i je(X:čovjek) i
                                  je(Y:zmija).
```

Rad sistema za upravljanje logičkom bazom znanja ilustriran je primjerima koji slijede.

PRIMJER 1.

Upisivanje činjenice koja je redundantna (tj. već deducibilna iz sistema.

```
| ?- ifact.
|: zdravlje_od(jure:dobro).

Činjenica: zdravlje_od(jure:dobro)
                              je redundantna!

Dedukcija ? (d./_.)
|: d.
Činjenica : zdravlje_od(jure:dobro)
Dedukcija:

   zdravlje_od(jure:dobro) slijedi iz
          fiz_aktivan(jure:da) i
                    nije pušač(jure:strastven)
   fiz_aktivan(jure:da) - činjenica u LB
   NIJE deducibilno: pušač(jure:strastven)

Upisati činjenicu ? (d./_.)
|: n.
LB nepromjenjena
yes
```

Upis nije izvršen (tj. odgovorili smo 'n.'), tako da je baza znanja ostala nepromjenjena. Inače, obzirom da sistem dopušta upis redundantnih znanja, odgovor 'd.' bio bi učinio da se (redundantna) činjenica upiše u bazu.

Prilikom obrazlaganja deducibilnosti činjenica (tj. eksplikacije procedure SLDNF-opovrgnuća), sistem navodi instancirana pravila, koja u procesu opovrgnuća (tj. izračunavanja odgovor supstitucije), učestvuju. Tako npr. obrazlaganje počinje navođenjem instanciranog pravila:

```
   zdravlje_od(jure:dobro) slijedi iz ...
```

a ne izvornog pravila:

```
   zdravlje_od(X:dobro) ako ... .
```

Implementacija eksplikacije procedure SLDNF-opovrgnuća u terminima neinstanciranih (izvornih) pravila bila bi složenija, ma da je u C-Prologu moguća. No, razlog za eksplicitiranje instanciranih pravila jeste što je takvo obrazloženje direktnije i razumljivije. Ukoliko pak pojedino od pravila želimo vidjeti u izvornoj formi, možemo to učiniti pomoću naredbe 'lrule(Svojstvo)', koja nam (za dato svojstvo), daje neinstancirana (izvorna) pravila.

```
| ?- lrule(zdravlje_od).

Svojstvo: zdravlje_od
Pravila:

   zdravlje_od(X:dobro) ako fiz_aktivan(X:da) i
                        nije pušač(X:strastven).

yes
```

PRIMJER 2.

Upis dviju činjenica, koje ne izazivaju nikakvih drugih promjena u bazi znanja

```
| ?- ifact.
|: voli(petar:vino), brat_od(jakov:stevan).

Svojstva - brat_od - nema u rječniku. Je li njegova vrijednost (j)edinstvena ili (v)išestruka ?
|: v.
Kontrola UI OK.
Druge promjene u LB:

Novo stanje LB prihvatljivo ?  (d./_.)
|: d.
Promjena izvršena
yes
```

Valja napomenuti da za entitete 'jakov' i 'stevan', u bazi nisu dati tipovi (tj.svojstva 'vrsta_od'). Naime, u bazi smijemo definirati i svojstva onih entiteta, koji nisu uvršteni u hijerarhiju entiteta. No, u tom slučaju sistem ne može znati da su ti entiteti vrste 'čovjek', pa ih tako niti ne tretira.

U ovom slučaju smo upisivane činjenice prihvatili, jer nam prva služi u primjeru (3) a druga u primjerima (4), (5) i (6).

PRIMJER 3.

Pokušaj upisa pravila, koje bi dovelo do kršenja uvjeta integriteta.

```
| ?- irule.
|: zdravlje_od(X:dobro) ako voli(X:vino).

Uvjet integriteta:  ui_zdravi

   nije_istina_da zdravlje_od(X:dobro) i
                                pušač(X:strastven)

bio bi prekršen!
Dedukcija bi bila:

   zdravlje_od(petar:dobro) slijedi iz
                                 voli(petar:vino)
   voli(petar:vino) - činjenica u LB

   pušač(petar:strastven) - činjenica u LB

Upis nije izvršen
yes
```

PRIMJER 4.

U ovom primjeru u bazu unosimo (rekurzivno a i redundantno) pravilo, kojim iskazujemo znanje:

```
   Svatko ima boju očiju jednaku
   boji očiju svoga brata.
```

Pored unosa pravila, u primjeru je ilustrirano i obrazlaganje neuspjelih pokušaja dedukcije; komentar slijedi primjeru.

```
| ?- irule.
|: boja_oči_od(X:Y) ako brat_od(X:Z) i
                              boja_oči_od(Z:Y).

Pravilo je redundantno - ima praznu ekstenziju!

Dedukcija ? (elem./n.)
|: boja_oči_od(jakov:X).                          (1)

*** Nije deducibilno jer:

Cilj: boja_oči_od(jakov:X)
se unificira sa:
   boja_oči_od(jakov:X) ako brat_od(jakov:Y)
                        i boja_oči_od(Y:X)   (2)
Nije deducibilno:
     brat_od(jakov:X) i boja_oči_od(X:Y)     (3)
```

```
Deducibilno je: brat_od(jakov:stevan)          (4)


Cilj: boja_oči_od(stevan:X)
se unificira sa:
   boja_oči_od(stevan:X) ako brat_od(stevan:Y)
                         i boja_oči_od(Y:X)    (5)
Nije deducibilno:
     brat_od(stevan:X) i boja_oči_od(X:Y)      (6)

Za -- brat_od(stevan:X) -- nema nijedne
                           unifikacije.        (7)


Cilj: boja_oči_od(stevan:X)
se unificira sa:
   boja_oči_od(stevan:X) ako majka_od(stevan:Y)
                         i boja_oči_od(Y:X)    (8)
Nije deducibilno:
     majka_od(stevan:X) i boja_oči_od(X:Y)     (9)

Za -- majka_od(stevan:X) -- nema nijedne
                            unifikacije.      (10)

Za -- boja_oči_od(stevan:X) -- nema drugih
                              unifikacija.    (11)


Cilj: boja_oči_od(jakov:X)
se unificira sa:
   boja_oči_od(jakov:X) ako majka_od(jakov:Y)
                        i boja_oči_od(Y:X)    (12)
Nije deducibilno:
     majka_od(jakov:X) i boja_oči_od(X:Y)      (13)

Za -- majka_od(jakov:X) -- nema nijedne
                           unifikacije.        (14)

Za -- boja_oči_od(jakov:X) -- nema drugih
                              unifikacija.     (15)

Dedukcija ? (elem./n.)
|: n.
Upisati pravilo ? (d./_.)
|: d.                                          (16)
Kontrola UI OK.
Druga promjena u LB:
Novo stanje LB prihvatljivo ?  (d/_)
|: d.                                          (17)
Promjena izvršena
yes
```

(1) Obzirom da je ekstenzija pravila prazna, jasno je da nijedna činjenica nije deducibilna iz toga pravila za sadašnje stanje baze. Pogledajmo zašto nije deducibilna činjenica (1)

(2) Sa tom činjenicom unifabilno je upravo upisano pravilo, ...

(3) ... no, tada treba deducirati (djelomično instancirano) tijelo toga pravila, što iz date baze nije moguće, jer ...

(4) ... prvi od ciljeva iz tijela deducibilan je, uz istodobno daljnje instanciranje (vezivanje) varijabli, ali ...

(5) ... drugi nije. Naime, drugi cilj je unifabilan sa upravo upisanim pravilom, no ...

(6) ... pokušaj dedukcije cilja (1) uz upotrebu toga pravila ...

(7) ... ne uspijeva.

(8) Cilj (5), kao izvedeni cilj u pokušaju dedukcije cilja (1), pokušava se zatim deducirati posredstvom drugog pravila iz baze, sa kojim je unifabilan, no ...

(9) ... ni taj pokušaj ...

(10) ... ne uspijeva, ...

(11) ... čime je pokušaj dedukcije cilja (1) pomoću upravo upisanog pravila (2), završio neuspjehom.

(12) Cilj (1) pokušava se zatim deducirati pomoću drugog pravila iz baze sa kojim je unifabilan, ...

(13) ... no, ni taj pokušaj ...

(14) ... ne uspijeva, ...

(15) ... čime su bezuspješno iscrpljene sve mogućnosti dedukcije, tj. sva unifabilna pravila, te cilj (1) nije deducibilan iz datoga stanja baze znanja.

(16) Redundantno pravilo smo u bazu upisali, ...

(17) ... i novo stanje baze prihvatili.

Očito je da obrazlaganje neuspjelih pokušaja dedukcije nalazi i eksplicira redom sve grane neuspjeha na pripadnom SLDNF-drvu, za dati upit i stanje baze znanja.

PRIMJER 5.

Unosom činjenice 'boja_oči_od(stevan:zelena)' čini da pravilo unešeno u primjeru (4) ne bude više redundantno. U ovom primjeru data je i uspješna dedukcija činjenice 'boja_oči_od(jakov:zelena)', koja prije ovoga upisa nije bila deducibilna, kako je to pokazano u primjeru (4).

```
) ?- iface.
|: boja_oči_od(stevan:zelena).

Kontrola UI OK.
Druge promjene u LB:

Promjena ekstenzije pravila:

   boja_oči_od(X:Y) ako brat_od(X:Z) i
                        boja_oči_od(Z:Y)

Pokazati promjenu ? (d./_.)
|: d.
Prijašnja ekstenzija pravila:

   prazna

Sadašnja ekstenzija pravila:

   boja_oči_od(jakov:zelena)

Dedukcija ? (elem./n.)
|: boja_oči_od(jakov:zelena).

Činjenica : boja_oči_od(jakov:zelena)
Dedukcija:

   boja_oči_od(jakov:zelena) slijedi iz
                  brat_od(jakov:stevan) i
                   boja_oči_od(stevan:zelena)
   brat_od(jakov:stevan) - činjenica u LB
   boja_oči_od(stevan:zelena) - činjenica u LB

Dedukcija ? (elem./n.)
|: n.
Novo stanje LB prihvatljivo ?  (d./_.)
|: d.
Promjena izvršena
yes
```

PRIMJER 6.

```
   Upis činjenice

   brat_od(stevan:jakov)                       (a)
```

dovodi do toga da postoji upit, koji za novo stanje baze znanja, ima SLDNF-drvo sa beskonačnom granom. Prilikom kontrole integriteta baze (kao dijela postupka upisivanja), sistem za

upravljanje bazom - inkrementalnim pozivanjem ciljeva, na koje upisano znanje može utjecati - nalazi takav upit. Pokušaj njegove dedukcije (tj. SLDNF-opovrgnuća) - zbog postojanja beskonačne grane - dovodi do prekoračenja raspoloživog prostora na sistemu i do prekida pokušaja dedukcije. *Obrazloženje pojedinih koraka* slijedi primjeru.

```
| ?- ifact.
|: brat_od(stevan:jakov).
! Out of local stack during execution
{ Execution aborted }                                   (1)

| ?- loop.                                              (2)

Pokušaj dedukcije cilja
        --- boja_oči_od(X:Y) ---
doveo je do prekoračenja prostora!                      (3)
yes

| ?- sh_loop,                                           (4)
|: boja_oči_od(X:Y), 10.

   boja_oči_od(jakov:X) slijedi iz
                  brat_od(jakov:stevan) i
                        boja_oči_od(stevan:X)
   brat_od(jakov:stevan) - činjenica u LB
   boja_oči_od(stevan:X) slijedi iz
                  brat_od(stevan:jakov) i
                        boja_oči_od(jakov:X)
   brat_od(stevan:jakov) - činjenica u LB
   boja_oči_od(jakov:X) slijedi iz
                  brat_od(jakov:stevan) i
                        boja_oči_od(stevan:X)
   brat_od(jakov:stevan) - činjenica u LB
   boja_oči_od(stevan:X) slijedi iz
                  brat_od(stevan:jakov) i
                        boja_oči_od(jakov:X)

   I T D.
yes

| ?- dorule.                                            (5)
|: boja_oči_od(X:Y) ako brat_od(X:Z) i
                        boja_oči_od(Z:Y).
yes
```

(1) U (1) Prolog interpretor javlja da je došlo do prekida pokušaja SLDNF-opovrgnuća za neki cilj, i to zbog prekoračenja raspoloživog prostora.

(2) Naredbom 'loop' od sistema logičke baze tražimo koji je to cilj bio.

(3) Odgovor je dat sa (3). Da bi pri prekidu rada Prolog interpretora bilo moguće dobivanje takvog odgovora, sistem baze znanja pamti zadnji cilj koji se pokušalo deducirati.

(4) Naredbom 'sh_loop', uz navođenje cilja dobivenog naredbom 'loop' te dubine praćenja pokušaja SLDNF-opovrgnuća (u našem primjeru 10), eksplicirati ćemo pokušaj dedukcije, koji slijedi beskonačnu (ili bar preveliku) granu SLDNF-drveta. Iz pokazanih koraka u pokušaju SLDNF-opovrgnuća, trebalo bi biti vidljivo da je grana beskonačna, tj. da se radi o cirkularnoj definiciji, u kojoj se cilj

   boja_oči_od(jakov:X)   (b)

   pokušava deducirati pomoću pravila koje se poziva na izveden cilj:

   boja_oči_od(stevan:X)   (a)

   S druge strane, pokušaj dedukcije cilja (a), poziva se na cilj (b)!

(5) Pravilo, koje je dovelo do cirkularnosti definicije, brišemo naredbom 'dorule'. Tom naredbom brisanje se vrši bez provođenja ikakvih kontrola, jer bi izvođenje kontrola (integriteta i stvorene redundance), opet otkrilo cirkularnost definicije (tj. dovelo do prekida procesa).

Problem cirkularnosti definicije jedan je od otvorenih problema logičkog programiranja. Praktičko rješenje, kakvo je ovdje dato izgleda zadovoljavajućim, a i ne vidi se što bi se više od toga moglo učiniti (osim, naravno, zabraniti rekurzivne definicije i uvesti hijerarhiju svojstava, kako smo već objasnili u odjeljku (3), no takvo ograničenje smatramo isuviše rigoroznim.

Pri postojanju rekurzivnih definicija, pojava cirkularnosti najčešće ne zavisi samo o jednom (rekurzivnom) pravilu, već i o ostalom sadržaju baze znanja. Jer pravilo, koje je u ovom primjeru dovelo do cirkularnosti, bilo je u primjeru (4) redundantno; u primjeru (5) prestalo je biti redundantno, dok je cirkularnim postalo tek ovdje i to upisom činjenice (a) u bazu znanja. S druge strane, analogno pravilo, kojim se svojstvo 'boja_oči_od' definira u ovisnosti od svojstva 'majka_od' (dato u početnom stanju baze znanja), neće nikada dovesti do cirkularnosti. Ili točnije: neće dovesti do cirkularnosti sve dok u bazu znanja budemo upisivali iskaze (činjenice, pravila), koji su u 'realnom svijetu' - kao modelu baze-teorije - istiniti. Razlika između ta dva svojstva (u kontekstu pisutnosti rekurzivnih definicija), jesta, na primjer, što je svojstvo 'brat_od' simetrično, dok svojstvo 'majka_od' nije. No, to ili slična saznanje nisu dovoljna da bi *a priori* mogli znati hoće li neko pravilo ili činjenica dovesti do cirkularnosti definicije nekog svojstva u logičkoj bazi. U tom kontekstu, i dato rješenje, koje omogućava eksplikaciju cirkularnosti a time i promjenu (ispravak, redefiniciju) znanja iz baze znanja, izgleda operativno zadovoljavajućim rješenjem.

PRIMJER 7.

Primjer upisa uvjeta integriteta popratiti ćemo prikazom cjelokupnog postupka upisa znanja "Sve ptice lete osim pingvina", kojeg smo detaljno analizirali u odjeljku (3.1). Pozitivni dio znanja pritom iskazujemo i upisujemo kao pravilo, sa naredbom 'irule' u koraku (2). Predhodno smo sa (1) unijeli dva entiteta na koje se pravilo odnosi, tako da primjer bude ilustrativniji (tj. da ekstenzija pravila ne bude prazna). Dedukcije iz pozitivnog djela baze znanja (pomoću činjenica (1) odnosno pravila (2)), date su na upit 'ans' u (3). Negativno znanje, (tj. znanje o pingvinu, kao izuzetku od pravila), upisujemo sa naredbom 'insio' u (4). Time je unešeno znanje da "pingvini ne lete". Pokušaj upisa pravila u (5) ne dopušta se, jer bi to prekršilo uvjet integriteta upisan u (4), kako to sistem (u (5)) i obrazlaže.

```
| ?- ifact.
|: vrsta_od(slavuj:ptica),
|: vrsta_od(pingvin:ptica).                             (1)
```

```
Kontrola UI OK
Druge promjene u LB
Novo stanje LB prihvatljivo ? (d./_.)
d: d.
Promjena izvršena
yes



| ?- irule.
|: leti(X:da) ako je(X:ptica) i
                    ni_je(X:pingvin).   (2)

Svojstva -- leti -- nema u rječniku. Je li njegova vrijednost (j)edinstvena ili (v)išestruka ?
j: j.
Kontrola UI OK
Druge promjene u LB:
Novo stanje LB prihvatljivo ? (d./_.)
|: d.
Promjena izvršena
yes



| ?- ans.
|: X tako_da leti(X:da).                  (3)

Odgovori:

   ptica
   slavuj

Objašnjenja ? (n-torka./n.)
|: slavuj.

Činjenica: slavuj
Dedukcija:

   leti(slavuj:da) slijedi iz
                       je(slavuj:ptica) i
                       ni_je(slavuj:pingvin)
   je(slavuj:ptica) slijedi iz
                    vrsta_od(slavuj:ptica) i
                    je(ptica:ptica)
   vrsta_od(slavuj:ptica) - činjenica u LB
   NIJE deducibilno: je(slavuj:pingvin)

Objašnjenja ? (n-torka./n.)
|: n.
yes



| ?- insic.
|: ui_leti
|: nije_istna_da leti(pingvin:da).        (4)

Upis izvršen
yes



| ?- irule.
|: leti(X:da) ako je(X:ptica).            (5)

Uvjet integriteta:  ui_leti

   nije_istina_da leti(pingvin:da)

bio bi prekršen!
Dedukcija bi bila:

   leti(pingvin:da) slijedi iz je(pingvin:ptica)
   je(pingvin:ptica) slijedi iz
                    vrsta_od(pingvin:ptica) i
                    je(ptica:ptica)
   vrsta_od(pingvin:ptica) _ činjenicau LB

Upis nije izvršen
yes
```

---

5. ZAKLJUČAK

U članku je dat prijedlog modela logičke baze znanja, koji je definiran u terminima logike i realiziran sredstvima i metodama logičkog programiranja.

Model je ilustriran primjerima rada sistema za upravljanje logičkom bazom znanja, koji je realiziran u programskom jeziku Prolog.

Predloženim modelom (i implementacijom) dat je prikaz načina predstavljanja i procesiranja znanja zasnovan na jeziku (i sistemu) logike, za koju smatramo da - u kontekstu te problematike - otvara najviše mogućnosti.

7. REFERENCE

<Boj 83> Bojadžijev, D.:
Herbrandov izrek: mehanično dokazovanje izrekov in relacijske baze podatkov, DP-2936, Ins. "J. Stefan", Ljubljana, 1983.

<Bra 82> Bratko, I.:
Expert Systems and PROLOG, u Sumner, F.: Supercomputer System Technology, Pergamon Infotech Ltd. State of the Art Reports, Series 10, No. 6, 1982.

<Cla 79> Clark, K.L.:
Predicate Logic as a Computational Formalism,Res. Mon. 79/59, Imperial College, London, Dept. of Computing, 1979.

<Emd 76> van Emdem, M.H., Kowalski, R.:
The Semantics of Predicate Logic as a Programming Language, J. of the ACM, Vol.23, No.4, 1976.

<Emd 78> van Emdem, H.M.:
Computation and Deductive Information Retrieval, u Neuhold, E. (ed): Formal Description of Programming Concepts, North-Holland, 1978.

<Gal 84> Gallaire, H., Minker, J., Nicolas, J.:
Logic and Databases: A Deductive Approach, Computing Surveys, 1984.

<Kit 84> Kitakami,H. et al:
A Metodology for Implementation of a Knowledge Acquisition System, Int. Symp. on Logic Programming, Atlantic City, 1984.

<Kow 79> Kowalski, R.:
Logic for Problem Solving, North Holland, 1979.

<Kow 81> Kowalski, R.:
Logic as a Database Language, Tech. Rep., Imperial Coll. London, Dept. of Compt. 1981.

<Llo 84> Lloyd, J.W.:
Foundations of Logic Programming, Springer-Verlag, 1984.

<Llo 85> Lloyd, J.W., Topor, R.W.:
A Basis for Deductive Database Sysem, Tec. Rep. 85/1, Univ. of Melbourne, Dept. of Comp. Science, 1985.

<Nil 82> Nilson, J.N.:
Principles of Artificial Intelligence, Springer-Verlag, 1982.

<Per 85> Pereira, L.M., Porto, A., Dias, V.:
Knowledge Systems, Tec. Rep. DI/UNL - 2/85, Univ. Nova de Lisboa, 1985.

<Rad 86a> Radovan, M.:
Model deduktivne baze podataka implementiran u Prologu, Informatica, Ljubjana, br.1, 1986.

<Rad 86b> Radovan, M.:
A Knowledge Acquisition System for Logic Database, Tec. Rep. UNL - 2/86, Univ. Nova de Lisboa, Dept. de Informatica, 1986.

<Rob 79> Robinson, A.J.:
Logic: Form and Function, Edimburg University Press, 1979.

<Yah 85> Yahya, A., Henschen, L.J.:
Deduction in Non-Horn Databases, J. of Automated Reasoning, Vol.1, No.2, 1985.

RAZVRSTITEV NOVOGENERACIJSKIH RAČUNALNIŠKIH ARHITEKTUR

UDK: 681.3.02

Borut Robič in Jurij Šilc
Institut Jožef Stefan, Ljubljana

V članku predstavljava razvrstitev računalniških arhitektur, porojeno iz treh načinov vodenja računanja: krmilnega, podatkovnega in vodenja z zahtevo. Opisani so ustrezni računski modeli ter njihove realizacije na različnih organizacijah stroja.

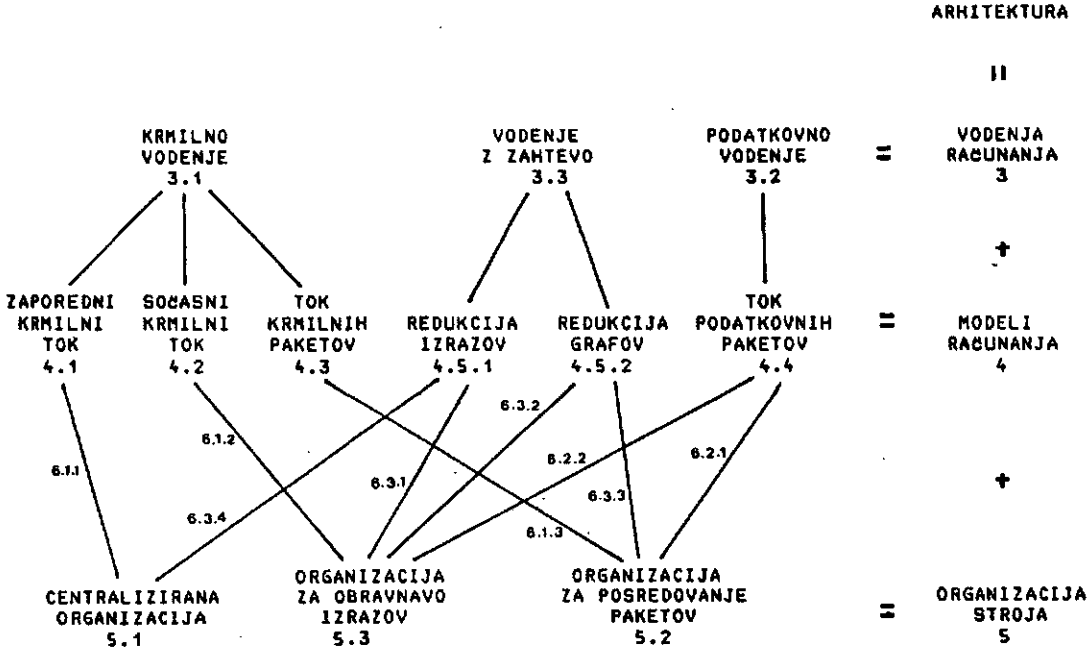
Classification of New Generation Computer Architectures - Classification of new generation computer architectures based on control driven, data driven, and demand driven computation is described. Also, we present several simple operational models and their implementations on different machine organizations.

1. Prolog

Ideja o računalnikih pete generacije je bila prvič predstavljena na mednarodni konferenci v Tokyu oktobra 1981. Peta generacija bo združevala rezultate raziskav na področjih umetne inteligence, programirnega inženiringa, VLSI (ULSI) tehnologije ter predvsem računalniških arhitektur. Slednje bodo zanesljivo prešle od zaporednega na sočasno izvrševanje. Na resnost teh raziskav kaže tudi dejstvo, da so od tokijske konference pa do leta 1984 ustanovili pet velikih nacionalnih oziroma regionalnih raziskovalnih programov:

- japonski, katerega usmerja ICOT (Institute for New Generation Computer Technology),
- ameriška DARPA (Defense Advanced Research Project Agency) in MCC (Microelectronic and Computer Technology Corporation),
- britanski Alveyjev program in
- ESPRIT (European Strategic Research Programme in Information Technology) v EGS.

Raziskave novogeneracijskih računalniških arhitektur so intenzivne in so porodile vrsto novih, ne-von Neumannovih principov računanja ter pridruženih organizacij stroja. V članku prikazujeva možno razvrstitev arhitektur, kot je bila predlagana v delu [Trel82a].



Slika 1. Elementi računalniške arhitekture.

2. Uvod

Rezultat raziskav na področju novejših računalniških arhitektur so tri osnovne družine arhitektur, ki se razlikujejo po načinu vodenja računanja [Trel81], [Trel82a]. Ustrezne arhitekture se imenujejo krmilno vodene, podatkovno vodene oziroma arhitekture, vodene z zahtevo. Znotraj vsake družine pa se arhitekture ločijo še po izbranem modelu računanja ter seveda po organizaciji stroja. Izraz računalniška arhitektura je torej določen s tremi elementi

⟨arhitektura⟩ ::= (⟨vodenje računanja⟩,
⟨model računanja⟩,
⟨organizacija stroja⟩).

Družina krmilno vodenih arhitektur združuje tradicionalne von Neumannove računalnike skupaj z njihovimi paralelnimi različicami, kot sta na primer SIMD in MIMD arhitektura. Ostali dve družini pa združujeta novejše, visoko paralelne ne-von Neumannove arhitekture.

V nadaljevanju bo prikazana podrobnejša razdelitev računalniških arhitektur ter razlike med njimi. Bralcu bo v pomoč slika 1, kjer se oznake ob posameznih elementih arhitekture nanašajo na ustrezna poglavja v članku.

3. Vodenja računanja

Računanje lahko smatramo kot zaporedno ponavljanje treh faz:
a) izbiranja /selection/,
b) preverjanja /examination/ ter
c) izvrševanja /execution/.

a) Izbiranje: v tej fazi se določi skupina ukazov programa, ki so možni kandidati za izvršitev. Izvršijo se lahko le izbrani ukazi, vendar izbor ukaza še ne zagotavlja njegove takojšnje izvršitve. Pravilo, po katerem se ukazi izbirajo, imenujemo pravilo izbiranja. Ločima tri pravila izbiranja:
- brezpogojno /imperative selection/,
- notranje /innermost selection/ ter
- zunanje /outermost selection/.

Pri brezpogojnem izbiranju se izbere ukaz ne glede na njegovo lego v programu. Izbor se izvrši na podlagi posebnega registra (programskega števca) ali pa na podlagi prisotnosti vseh krmilnih paketov. Notranje izbiranje izbere najgloblje gnezdene ukaze izraza, zunanje izbiranje pa ukaze, ki ne gnezdijo v nobenem izrazu.

b) Preverjanje: v tej fazi se ugotovi, če so izbrani ukazi izvršljivi, kar pomeni, da se preveri, če so prisotne vse potrebne vrednosti operandov v izbranih ukazih. Pravilo, po katerem poteka preverjanje, imenujemo pravilo izvršitve /firing rule/. Izvršljive ukaze prevzame tretja faza računanja, izvrševanje. Če ukaz ni izvršljiv, pa ga faza preverjanja bodisi odloži, dokler ga neka kasnejša faza izbiranja zopet ne izbere, ali pa ga poskuša napraviti izvršljivega s tem, da zahteva določitev vrednosti njegovih argumentov.

c) Izvrševanje: v tej fazi se izvršljivi ukazi dejansko izvrše.

Ločimo krmilno vodeno /control driven/ računanje, podatkovno vodeno /data driven/ računanje ter računanje vodeno z zahtevo /demand driven/.

Čeprav bodo vsa tri vodenja računanja podrobno opisana v nadaljevanju, že tu omenimo, da pri podatkovnem vodenju prisotnost vseh potrebnih operandov sproži izvrševanje ukaza, medtem ko pri vodenju na zahtevo zahteva po rezultatu sproži izvrševanje tistega ukaza, ki ta rezultat lahko priskrbi.

3.1. Krmilno vodenje

Krmilno vodeno računanje ima sledeče značilnosti:
- izbiranje ukazov je brezpogojno,
- preverjanje ukazov se ne izvaja in
- izvrševanje ukazov sledi neposredno po izbiranju.

Torej se ukazi izvrše takoj, ko so izbrani.

3.2. Podatkovno vodenje

Značilnosti podatkovnega vodenja lahko strnemo v naslednje:
- izbiranje poteka tako, da se vsakemu ukazu dodeli procesor,
- preverjanje sestoji iz sprotnega ugotavljanja prisotnosti vseh vhodnih operandov ukaza,
- izvrševanje pa sestoji iz izvršitve vseh ukazov, katerih vsi potrebni operandi so prisotni.

Torej pri podatkovno vodenem računanju ukazi pasivno čakajo na svoje vhodne operande.

3.3. Vodenje z zahtevo

Vodenje z zahtevo pa ima sledeče značilnosti:
- ukaz se izbere (navadno po pravilu zunanjega izbiranja) tedaj, ko nek prej izbrani ukaz zahteva njegov rezultat,
- v fazi preverjanja se ugotovi, če so izbrani ukazi izvršljivi, torej če so znane vrednosti njihovih operandov. Če vrednost operanda še ni znana, se posreduje zahteva tistemu ukazu, ki ga lahko izračuna,
- izvrševanje je sestavljeno iz izvršitve vseh tistih ukazov, katerih potrebni operandi so znani.

Torej se pri vodenju na zahtevo ukaz izvrši tedaj, ko je bila podana zahteva po njegovem rezultatu.

4. Modeli računanja

Vodenja računanja realiziramo z različnimi modeli računanja.

Krmilno vodenemu računanju pripadajo sledeči modeli:
- z zaporednim krmilnim tokom /sequential control flow/,
- s sočasnim krmilnim tokom /parallel control flow/,
- s tokom krmilnih paketov /control flow with control tokens/.

Podatkovno vodenemu računanju pripada le model:
- s tokom podatkovnih paketov /data flow/.

Računanje, ki je vodeno z zahtevo pa je moč realizirati z dvema modeloma:
- z redukcijo izrazov /string reduction/,
- z redukcijo grafov /graph reduction/.

V vsakem modelu računanja se zrcali izvirni način vodenja računanja v obliki sprožitvenega ter podatkovnega mehanizma.

Sprožitveni mehanizem /control mechanism/ določa način, kako izvršitev nekega ukaza vpliva na izvršitev ostalih ukazov in s tem vodenje izvrševanja programa. Sprožitveni mehanizem je lahko:

- zaporeden /sequential/: ukazi se izbirajo drug za drugim in se po vrsti izvršujejo,
- sočasen /parallel/: kjer se na nek način ugotavlja prisotnost vhodnih operandov ter povzroči začetek izvrševanja tistih ukazov, ki imajo na voljo vse potrebne operande,
- rekurziven /recursive/: kjer se posreduje zahtevo po operandih ter sproži izvrševanje tistega ukaza, katerega rezultat je bil zahtevan.

S podatkovnim mehanizmom /data mechanism/ pa je določen način, po katerem si ukazi delijo skupne operande. Pri tem si lahko ukazi delijo operand s pomočjo:

- vstavljene vrednosti /literal/: če je vrednost operanda znana že pred pričetkom računanja, se takoj vpiše v vse ukaze, ki jo uporabljajo,
- sprotne vrednosti /value/: kopije operanda, ki se je izračunal, se posredujejo vsem ukazom, ki jim je operand skupen,
- reference /reference/: tu vsebujejo vsi ukazi referenco (naslov) na skupen operand.

Zvezo med obema mehanizmoma ter različnimi modeli računanja prikazuje tabela 1.

SPREMENJIVI MEHANIZEM		PODATKOVNI MEHANIZEM: vstavljene in sprotne vred.	PODATKOVNI MEHANIZEM: vstavljene vred. in reference
	zaporeden		zaporedni krmilni tok
	sočasen	tok podatkovnih paketov	sočasni krmilni tok
			tok krmilnih paketov
	rekurziven	redukcija izrazov	redukcija grafov

Tabela 1. Modeli računanja glede na sprožitveni in podatkovni mehanizem.

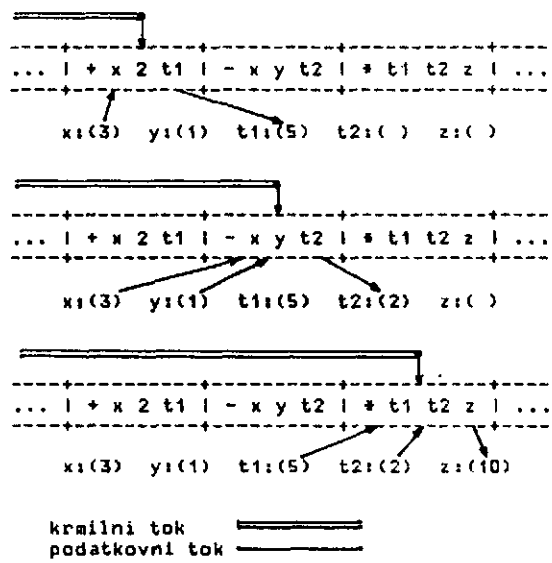
4.1. Zaporedni krmilni tok

Model z zaporednim krmilnim tokom ima naslednje značilnosti:

- krmilno vodenje računanja,
- zaporedni sprožitveni mehanizem,
- podatkovni mehanizem s pomočjo vstavljenih vrednosti in referenc.

Model z zaporednim krmilnim tokom uporablja en sam procesor za računanje in programski števec, s katerim se izbere naslednji ukaz. Torej obstaja en sam krmilni tok, ki prehaja z ukaza na ukaz in temelji na zaporednem sprožitvenem mehanizmu. Ko krmilni tok doseže ukaz, se razrešijo vse reference, kar pomeni, da se dostavijo vrednosti operandov iz naslovljenih pomnilniških lokacij. V modelu z zaporednim krmilnim tokom se torej uporablja podatkovni mehanizem z vstavljenimi vrednostmi in referencami. Ukaz se nato takoj izvrši, saj fazi (brezpogojnega) izbiranja neposredno sledi faza izvrševanja - računanje je torej krmilno vodeno. Rezultat izvršitve se vpiše v skupno lokacijo. Po izvršitvi ukaza se programski števec bodisi neposredno ali posredno ažurira, s čimer krmilni tok preide na naslednji ukaz.

Izvrševanje v modelu z zaporednim krmilnim tokom ponazorimo na stavku z := (x+2)*(x-y), ki je predstavljen z zaporedjem ukazov, pri čemer vsak vsebuje operacijo, kateri sledijo operandi (Slika 2). Ti so bodisi vstavljene vrednosti ali pa naslovi lokacij, kjer so shranjene vrednosti operandov. Prenašanje vrednosti med ukazi se vrši s pomočjo skupnih lokacij v pomnilniku.



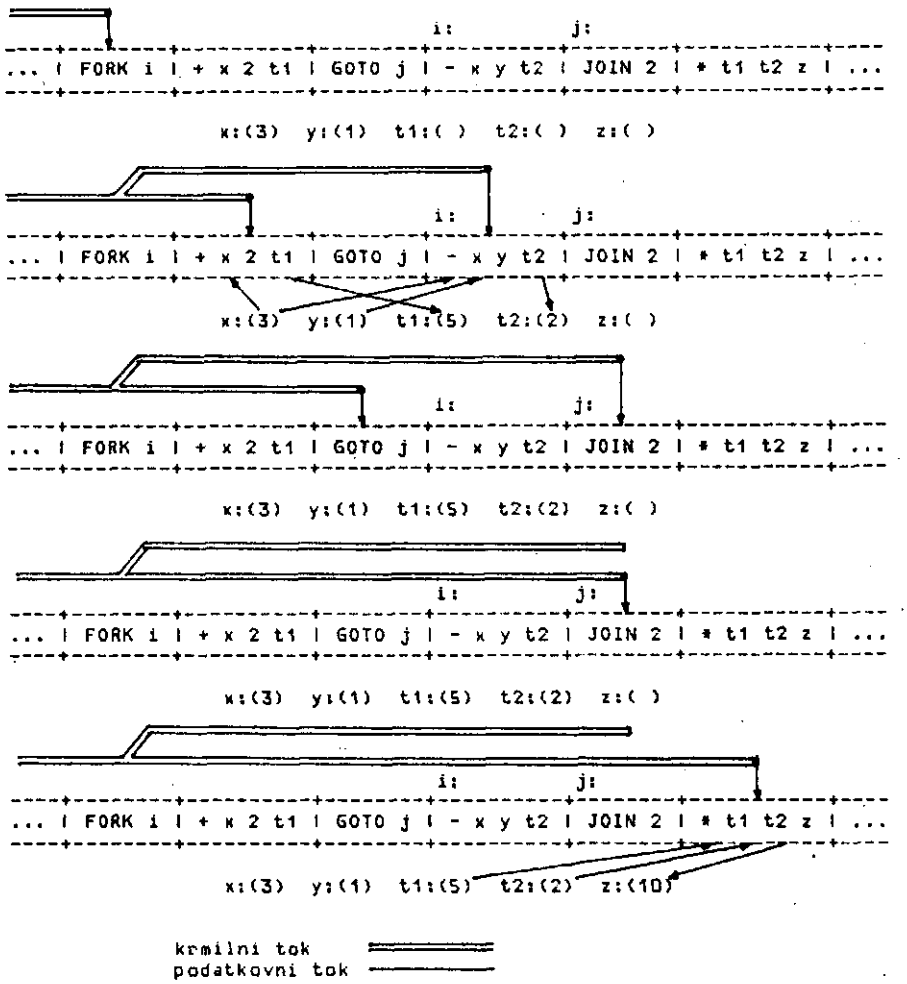
Slika 2. Računanje v modelu z zaporednim krmilnim tokom.

4.2. Sočasni krmilni tok

Model sočasnega krmilnega toka ima sledeče značilnosti:

- krmilno vodenje računanja,
- sočasni sprožitveni mehanizem,
- podatkovni mehanizem s pomočjo vstavljenih vrednosti in referenc.

V modelu s sočasnim krmilnim tokom uporabljamo ukaza FORK za začetek sočasnega računanja ter JOIN za sinhronizacijo. Ukaz FORK i povzroči nastanek novega zaporednega krmilnega toka s pričetkom pri ukazu z naslovom i. Ukaz na naslovu i ima tedaj na voljo že vse potrebne vhodne operande. Sočasni sprožitveni mehanizem v tem modelu ne zahteva preverjanja prisotnosti potrebnih operandov, saj je pričetek novega krmilnega toka eksplicitno določen z ukazom FORK. Krmilni tok, ki je izvršil ukaz FORK i, nadaljuje z izvrševanjem naslednjega ukaza (implicitni krmilni tok). Ukaz JOIN n zaustavi prvih n-1 krmilnih tokov, ki so prispeli do njega; ko prispe do tega ukaza zadnji, n-ti krmilni tok, nadaljuje z izvrševanjem naslednjega ukaza. Ker se v tem modelu sočasno vrši nekaj zaporednih krmilnih tokov, je njegov podatkovni mehanizem enak mehanizmu v modelu z zaporednim krmilnim tokom. Seveda zahteva tak model uporabo večjega števila procesorjev. Primer izračuna stavka z := (x+2)*(x-y) opisuje slika 3.



Slika 3. Računanje v modelu s sočasnim krmilnim tokom.

4.3. Tok krmilnih paketov

Model toka s krmilnimi paketi ima sledeče značilnosti:

- krmilno vodenje računanja,
- sočasni sprožitveni mehanizem,
- podatkovni mehanizem s pomočjo vstavljenih vrednosti in referenc.

Tudi pri modelu s tokom krmilnih paketov poteka računanje sočasno, zato ta model zahteva večje število procesorjev. Vsak ukaz hrani poleg operacije ter operandov še naslov svojega naslednika, ki bo uporabil njegov rezultat, ter prostore za krmilne pakete, ki jih prejme od svojih predhodnikov po njihovi izvršitvi. Ko ukaz zbere vse krmilne pakete, se sproži njegovo izvrševanje, pri čemer se najprej razrešijo vse reference ter nato izvrši operacija. Na koncu se rezultat shrani v skupno lokacijo, krmilni paket pa se posreduje vsem naslednikom ukaza. Vidimo, da je izvrševanje krmilno vodeno s sočasnim sprožitvenim mehanizmom, ki ugotavlja prisotnost potrebnih krmilnih paketov. Podatkovni mehanizem pa temelji na vstavljenih vrednostih in referencah. Potek računanja stavka z := (x+2)*(x-y) je opisan na sliki 4.

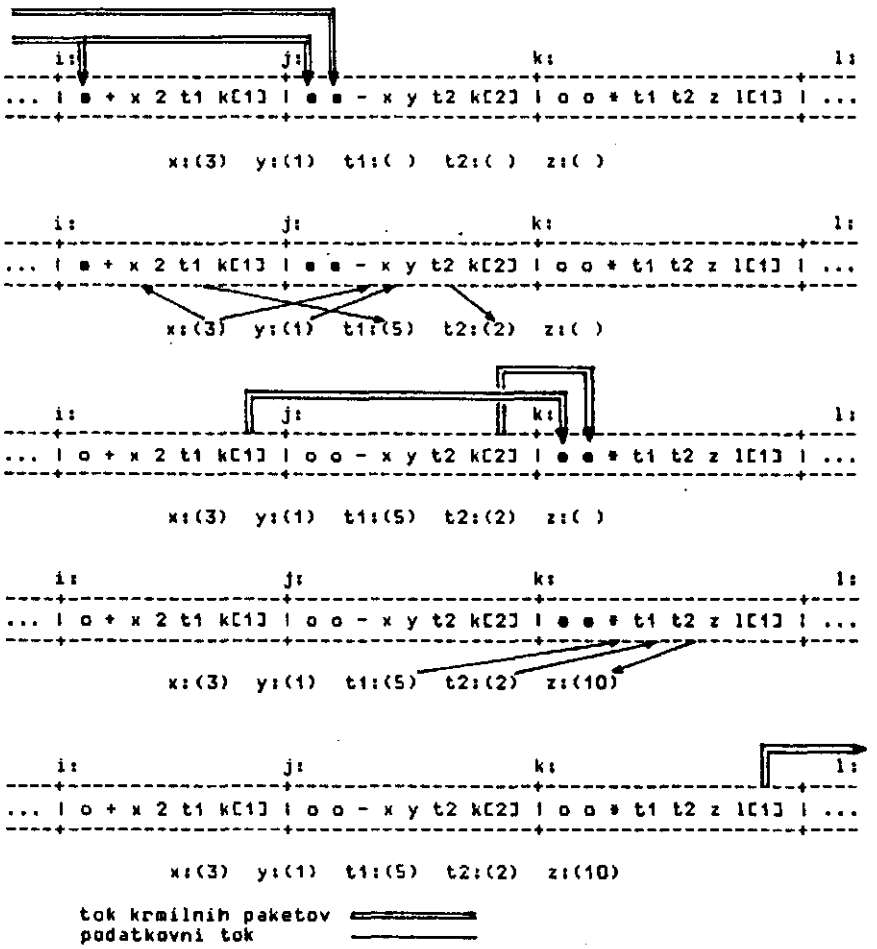
4.4. Tok podatkovnih paketov

Model s tokom podatkovnih paketov ima naslednje značilnosti:

- podatkovno vodenje računanja,
- sočasni sprožitveni mehanizem,
- podatkovni mehanizem s pomočjo vstavljenih in sprotnih vrednosti.

Ta model je podoben modelu s tokom krmilnih paketov, le da se tu izvrševanje odvija s pomočjo podatkovnih paketov. Ukazi sestojijo iz operacij, mest za operande ter oznak vseh ukazov, kamor naj se posreduje rezultat. Operandi so bodisi vstavljene ali pa sprotne vrednosti, s čimer je določen podatkovni mehanizem. Ukaz se lahko prične izvrševati takoj, ko mu je zadnji izmed njegovih neposrednih predhodnikov posredoval podatkovni paket s sprotno vrednostjo, na katero čaka. Izvrševanje je podatkovno vodeno, saj sočasni sprožitveni mehanizem v tem modelu ugotavlja prisotnost vseh potrebnih vrednosti vhodnih operandov.

V primeru toka podatkovnih paketov so programi navadno predstavljeni s pomočjo usmerjenih grafov, s katerimi je opisan tok podatkov med ukazi. Točke programskega grafa so ukazi,



Slika 4. Računanje v modelu s tokom krmilnih paketov.

povezave pa skrbijo za prenos vrednosti (podatkovnih paketov) med njimi. Točka se lahko prične izvrševati šele tedaj, ko so prisotni podatkovni paketi v vseh njenih vhodnih povezavah. Ob pričetku izvrševanja točka absorbira vse svoje vhodne podatkovne pakete (s čimer ti izginejo iz vhodnih povezav), po izvršitvi pa postavi rezultat (izhodni paket) istočasno v vse svoje izhodne povezave. Izhodni paketi točke so hkrati vhodni paketi neposrednih naslednikov.

Pri modelu računanju s tokom podatkovnih paketov je k vsakemu ukazu pridružen procesor, ki pasivno čaka na potrebne vhodne vrednosti. Faza izbiranja je torej kar pridružitev procesorjev vsem ukazom. Faza preverjanja sestoji iz ugotavljanja, če so prisotne vrednosti vseh vhodnih operandov. Če te vrednosti niso prisotne, ostane procesor neaktiven, v nasprotnem pa jih sprejme, preide v fazo izvrševanja ter zatem posreduje rezultate vsem svojim naslednikom. Izvršitev stavka z := (x+2)*(x-y) v tem modelu kaže slika 5.

4.5. Redukcija

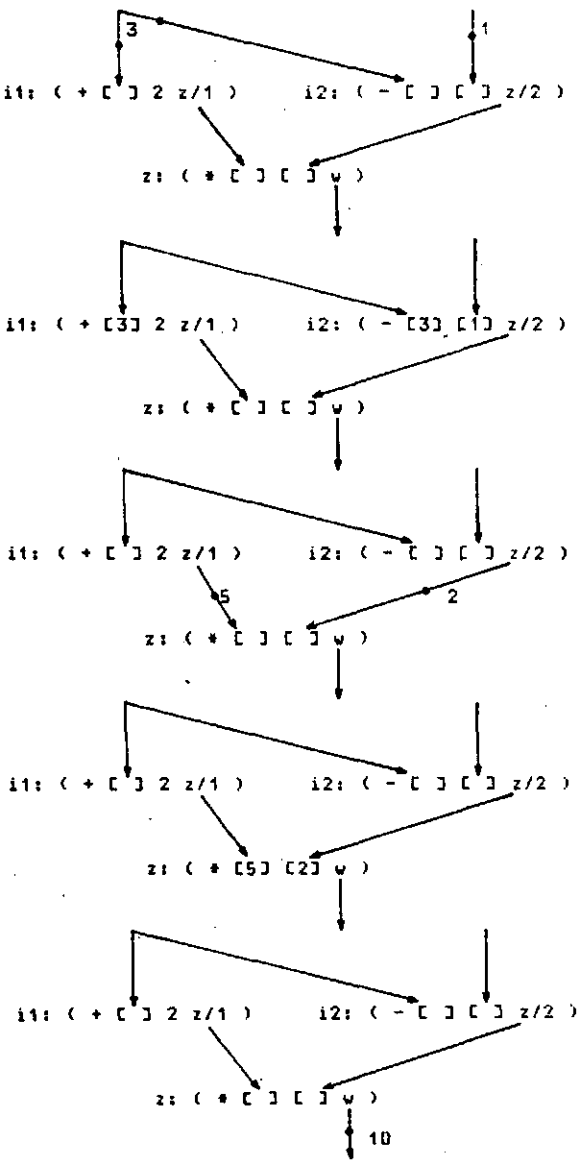
Krmilno in podatkovno vodeni modeli računanja uporabljajo ukaze, katerih velikost se ne spreminja. Modela računanja z redukcijo pa uporabljata gnezdene izraze, katerih velikost se med računanjem spreminja. Izvršitvi ukaza v krmilno ali podatkovno vodenem modelu ustreza pri redukciji uporaba /aplication/ funkcije nad njenimi argumenti. Programi v redukcijskem modelu so izrazi oblike <f><a1,...,aN>, kje so tako funkcija kot njeni argumenti bodisi atomi (npr. + ali 2) ali pa izrazi, sestavljeni iz atomov in izrazov. Uporaba funkcije je sestavljena iz dveh korakov:

- zahteve za dodelitev vrednosti njenim neznanim argumentom ter
- izračuna funkcije nad njimi.

Izračun funkcije se torej odloži, dokler niso dodeljene vrednosti vsem argumentom. Na primer, zahteva za dodelitev vrednosti argumentu a1, kjer je a1 definiran z a1 = <g><b1,...,bM>, sproži nadaljno uporabo funkcije g nad argumenti b1,...,bM. Takšne zahteve po uporabi funkcij lahko porodijo nove zahteve, s čimer je določen rekurziven sprožitveni mehanizem.

Očitno je, da mora biti vsak argument definiran z eno samo definicijo, kar imenujemo pravilo o enkratni prireditvi /single assignement rule/. Pravimo, da se dani argument sklicuje /reference/ na svojo definicijo. Poleg tega definicija vrne pri vsaki uporabi vedno enako vrednost kar imenujemo referenčna transparentnost redukcije.

Pri redukciji poteka faza izbiranja navadno



Slika 5. Računanje v modelu s tokom podatkovnih paketov.

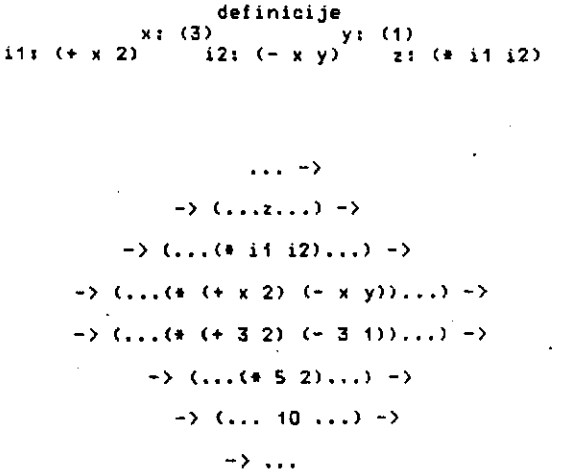
po pravilu zunanjega izbiranja. Procesorji se pridružijo ukazom v času faze izbiranja. Faza preverjanja pregleda argumente ukaza ter ugotovi, če je izračun možen. Če je, preide računanje v fazo izvrševanja, v kateri se ukaz izvrši ter nadomesti s svojim rezultatom. V nasprotnem primeru pa faza preverjanja izda zahtevo po nadomestitvi neznanih argumentov z znanimi vrednostmi. Takšna zahteva povzroči nastanek novih faz izbiranja, preverjanja in izvrševanja tistih ukazov, ki lahko vrnejo zahtevane vrednosti. Vsaka taka nova faza preverjanja seveda lahko porodi zopet nove faze računanja. Ko se vsa ta povzročena računanja končajo in vrnejo prvotno zahtevane vrednosti, začetni ukaz končno preide v fazo izvrševanja.

Ločimo dve vrsti redukcije, ki se razlikujeta po načinu obravnavanja argumentov. Pripadajoča modela računanja se imenujeta redukcija izrazov oziroma redukcija grafov.

4.5.1. Redukcija izrazov

Model z redukcijo izrazov ima sledeče značilnosti:
- vodenje računanja z zahtevo,
- rekurzivni sprožitveni mehanizem,
- podatkovni mehanizem s pomočjo vstavljenih in sprotnih vrednosti.

Zahteva za dodelitev vrednosti argumentom poteka tako, da se neatomarni argument nadomesti s kopijo definicije, na katero se sklicuje. Podatkovni mehanizem zato temelji na vstavljenih in sprotnih vrednostih. Če funkcija vsebuje več neatomarnih argumentov, se vsi nadomeste sočasno. Če so argumenti funkcije atomarni, se funkcija izračuna in nadomesti z rezultatom. Izračuni funkcij potekajo sočasno. V tem modelu je zelo pomembno dejstvo, da se postopek nadomeščanja izvrši vsakokrat, ko se zahteva izračun argumenta. Primer redukcije izrazov pri izvrševanju stavka z := (x+2)*(x-y) je opisan na sliki 6.

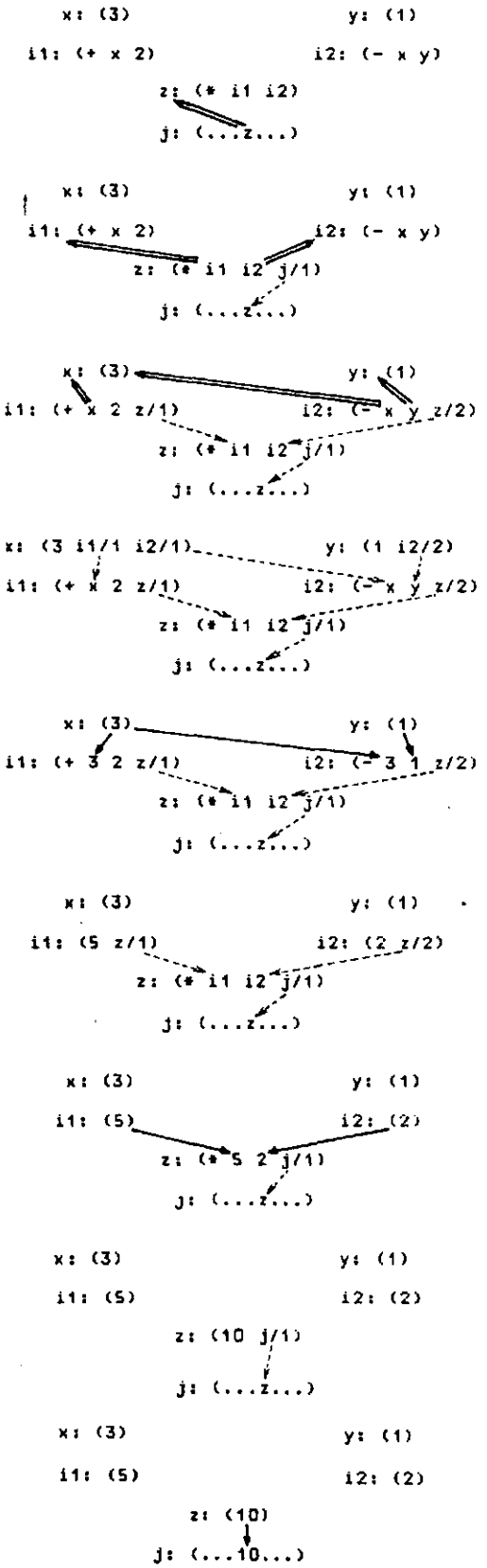


Slika 6. Računanje v modelu z redukcijo izrazov.

4.5.2. Redukcija grafov

Model z redukcijo grafov pa ima sledeče značilnosti:
- vodenje računanja z zahtevo,
- rekurzivni sprožitveni mehanizem,
- podatkovni mehanizem s pomočjo vstavljenih vrednosti in referenc.

Pri redukciji grafa se definiciji, na katero se sklicuje argument, doda oznaka tega argumenta. To pomeni, da podatkovni mehanizem temelji na vstavljenih vrednostih in referencah. Po uporabi funkcije v tej definiciji se njena vrednost posreduje čakajočemu argumentu, hkrati pa se tudi definicija nadomesti s to vrednostjo. Ob kasnejšem sklicevanju na tako definicijo se takoj vrne njena vrednost. To je temeljna prednost redukcije grafa napram redukciji izrazov. Primer izračuna stavka z := (x+2)*(x-y) je opisan na sliki 7.



Slika 7. Računanje v modelu z redukcijo grafov.

4.6. Povzetek lastnosti modelov računanja

Zaporedni in sočasni krmilni tok imata sledeče skupne značilnosti:
- prenos podatkov med ukazi poteka preko referenc na skupne pomnilniške lokacije,
- vrednosti operandov so lahko shranjene v samem ukazu (vstavljene vrednosti); s tem se pohitri njegova izvršitev,
- krmilni tok je po svoji naravi zaporeden, vendar lahko kreiramo sočasne krmilne tokove z uporabo posebnih ukazov FORK in JOIN,
- krmilni tok in tok podatkov sta ločena,
- potek računanja je popolnoma določen s krmilnim tokom.

Lastnosti računanja s tokom podatkovnih paketov so:
- sprotne vrednosti se posredujejo med ukazi neposredno, v obliki podatkovnih paketov,
- možna je uporaba vstavljenih vrednosti,
- ob pričetku izvrševanja prevzame ukaz svoje vhodne pakete, s čimer ti izginejo iz vhodnih povezav, zato jih kasneje ne more več uporabiti,
- pri izvrševanju se ne uporabljajo nikakršne skupne pomnilniške lokacije za prenos podatkov, kot je to primer pri krmilnem vodenju,
- izvrševanja programa je popolnoma določeno s tokom podatkov.

Redukcija pa ima sledeče značilnosti:
- vse programske strukture, funkcije in argumenti so gnezdeni izrazi,
- pri prenašanju vrednosti se ne uporabljajo nikakršne skupne pomnilniške lokacije,
- izvršitev ukaza vrne bodisi atomaren ali pa sestavljen izraz,
- po izvršitvi definicije se le-ta lahko nadomesti z izračunano vrednostjo,
- potek računanja je popolnoma definiran s tokom zahtev po izvršitvah.

Vse opisane modele računanja in njihov odnos glede na način vodenja opisuje tabela 2.

NAČINI VODENJA

MODELI RAČUNANJA	krmilno	podatkovno	z zahtevo
MAORDUENLAINJA	zaporedni krmilni tok	tok podatkovnih paketov	redukcija izrazov
	sočasni krmilni tok		redukcija grafov
	tok krmilnih paketov		

Tabela 2. Načini vodenja in ustrezni modeli računanja.

5. Organizacije stroja

Kot je bilo uvodoma omenjeno, pripadajo načinom vodenja računanja ustrezne družine računalniških arhitektur, ki se imenujejo krmilno vodene arhitekture, podatkovno vodene arhitekture ter arhitekture, vodene z zahtevo. Omenjene družine izvirajo iz načinov vodenja računanja, vendar pa lahko posamezne družine razdelimo na poddružine glede na izbrani model računanja ter organizacijo stroja /machine organization/. Pod izrazom organizacija stroja razumemo način sestavljanja in medsebojnega povezovanja osnovnih računalniških materialnih

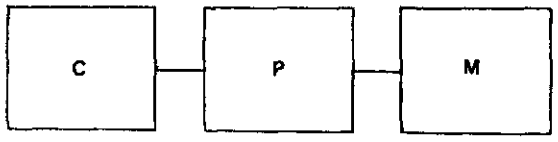
virov /resources/, kot so procesorji, pomnilniki in komunikacijske enote.

Razlikujemo tri osnovne organizacije stroja in sicer:
- centralizirano organizacijo /centralized organization/,
- organizacijo za posredovanje paketov /package communication organization/,
- organizacijo za obravnavo izrazov /expression manipulation organization/.

V nadaljevanju bomo najprej opisali vse tri omenjene organizacije stroja.

5.1. Centralizirana organizacija

Centralizirano organizacijo sestavljajo en procesor, pomnilnik ter vodilo med njima (Slika 8). Takšna organizacija ima v danem trenutku v programu en sam izvršujoč se ukaz. Po izvršitvi takšnega ukaza se prične izvrševanje njegovega natančno določenega naslednika, izbranega s programskim števcem. To je v resnici tradicionalna von Neumannova arhitektura.



P ... procesor
C ... komunikacijska enota
M ... pomnilnik

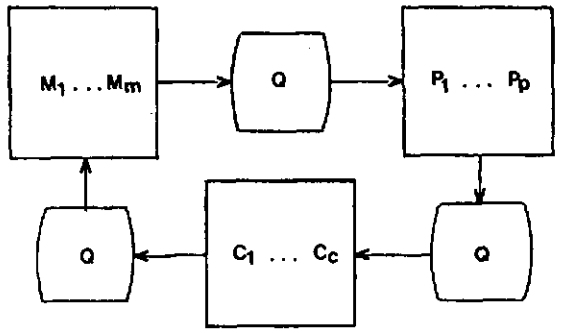
Slika 8: Centralizirana organizacija stroja.

5.2. Organizacija za posredovanje paketov

Organizacijo sestavljajo tri glavne enote: procesna enota, ki združuje večje število procesorjev, pomnilniška enota ter komunikacijska enota. Enote so krožno povezane, kot to prikazuje slika 9. Med enotami se nahajajo še vrste /pools of work/, ki opravljajo nalogo začasnega skladiščenja paketov, kadar je to potrebno. Izvrševanje programa v takšni organizaciji poteka v obliki enosmernega krožnega toka paketov skozi njene enote. Enote delujejo sočasno, torej je izvrševanje programa cevljeno /pipelined/. Paketi, ki so pripravljeni za obdelavo v eni izmed enot, čakajo v ustrezni vrsti, vse dokler jih ustrezna enota ni pripravljena sprejeti. Ko jih ta enota obdela, shrani tako spremenjene pakete v vrsto pred naslednjo enoto.

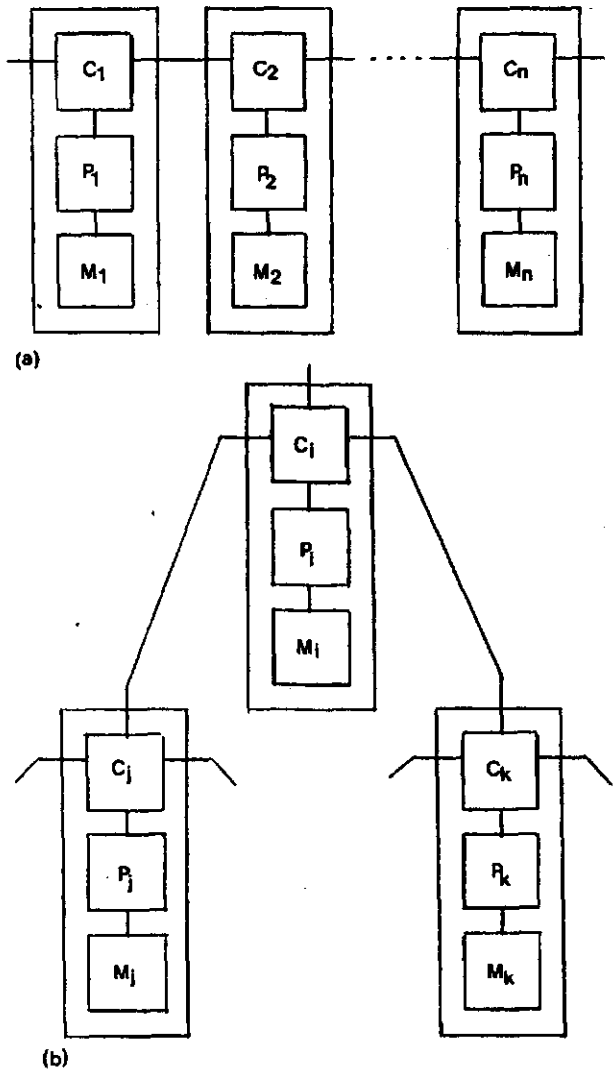
5.3. Organizacija za obravnavo izrazov

Takšno organizacijo sestavljajo enaki osnovni gradniki, med seboj povezani v pravilno strukturo /regular structure/. Vsak gradnik sestavljajo tri enote: procesor, pomnilnik in komunikacijska enota. Slednja skrbi za povezavo med procesorjem, pomnilnikom ter sosednjimi gradniki. Dve značilni strukturi takšne organizacije, vektorsko in drevesno, prikazuje slika 10. V takšni organizaciji se obravnava program kot gnezden izraz. Podizrazi so pridruženi gradnikom, pri čemer se podizraz pridruži tistemu gradniku, ki zrcali lego podizraza v izrazu. Med izvrševanjem vsak gradnik pregleda svoj pridruženi podizraz ter ugotovi, kaj mu je storiti.



P ... procesor
C ... komunikacijska enota
M ... pomnilnik
Q ... vrsta

Slika 9. Organizacija stroja za posredovanje paketov.



P ... procesor
C ... komunikacijska enota
M ... pomnilnik

Slika 10. Organizacija stroja za obravnavo izrazov; a) vektorska, b) drevesna.

6. Arhitekture

V tem poglavju bomo pokazali, kako lahko različne organizacije stroja podpirajo posamezne modele računanja. Povedano drugače, tri osnovne družine arhitektur bomo razdelili na podružine, kot je prikazano v tabeli 3.

družine arhitektur	model računanja	organizacija stroja
krmilno vodene	zaporedni krmilni tok	centralizirana
	sočasni krmilni tok	obravnava izrazov
	tok krmilnih paketov	posredovanje paketov
podatkovno vodene	tok podatkovnih paketov	posredovanje paketov
	tok podatkovnih paketov	obravnava izrazov
vodene z zahtevo	redukcija izrazov	obravnava izrazov
	redukcija grafov	obravnava izrazov
	redukcija izrazov	centralizirana
	redukcija grafov	posredovanje paketov

Tabela 3. Razdelitev računalniških arhitektur.

6.1. Krmilno vodene arhitekture

6.1.1. Zaporedni krmilni tok na centralizirani organizaciji

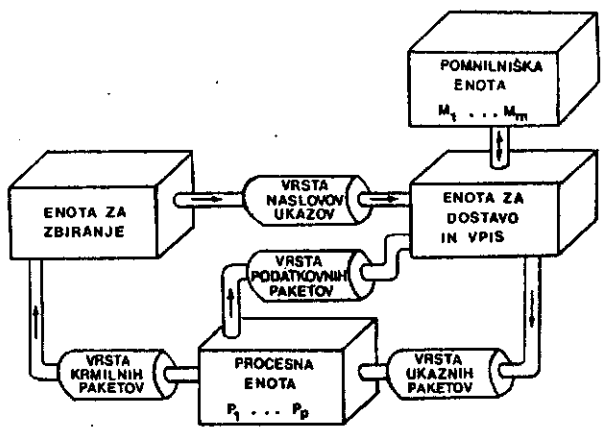
Najnaravnejši kandidat za podporo zaporednega krmilnega toka je centralizirana organizacija stroja. Krmilni tok določa programski števec v procesorju, ki zaporedno naslavlja ukaze, ki so na vrsti za izvršitev. Procesor takšne ukaze drugega za drugim, tj. zaporedno, izvršuje. Ker ta zveza določa znano von Neumannovo arhitekturo, je na tem mestu ne bomo podrobneje opisovali.

6.1.2. Sočasni krmilni tok na organizaciji za obravnavo izrazov

Sočasni krmilni tok, pri katerem uporabljamo ukaza FORK in JOIN, podpira organizacija za obravnavo izrazov. Zaporedni krmilni tokovi, ki se vršijo sočasno, potekajo v različnih gradnikih te organizacije. Krmilni tok, ki poteka v enem gradniku, lahko porodi nov krmilni tok v nekem drugem gradniku. Ko gradnik A prične izvrševati ukaz FORK i, sporoči (preko komunikacijske enote) tistemu gradniku B, v katerega pomnilniku se nahaja ukaz z naslovom i, naj prične z izvrševanjem svojega programa pri tem ukazu. V primeru, ko v B že poteka nek krmilni tok, A odloži izvršitev operacije FORK i, dokler se krmilni tok v B ne izteče. Tedaj se lahko izvrši ukaz FORK i, torej se prične krmilni tok v B pri ukazu i, gradnik A pa lahko nadaljuje izvrševanje, narekovano z lastnim krmilnim tokom. Gradnik, v katerem je krmilni tok prispel do ukaza JOIN n, se zaustavi vse dotlej, dokler se v n-1 gradnikih ne izteče zadnji med porojenimi krmilnimi tokovi.

6.1.3. Tok krmilnih paketov na organizaciji za posredovanje paketov

Krmilno voden model računanja s tokom krmilnih paketov pa je moč realizirati na organizaciji za posredovanje paketov. Arhitektura ima zgradbo, kot jo prikazuje slika 11.



Slika 11. Tok krmilnih paketov na organizaciji za posredovanje paketov

Program se nahaja v pomnilniški enoti /memory unit/. Enota za zbiranje /matching unit/ hrani za vsak ukaz programa:
- množico doslej zbranih krmilnih paketov ukaza
- naslov ukaza v pomnilniški enoti.

Ko enota za zbiranje zbere vse potrebne krmilne pakete danega ukaza, pošlje njegov naslov preko vrste naslovov ukazov /instruction address pool/ v enoto za dostavo in vpis /fetch & update unit/. Ta enota prenese iz pomnilniške enote naslovljeni ukaz, razreši reference ter sestavi ukazni paket. Le-ta vsebuje operacijo, vrednosti operandov ter naslove ukazov, ki pričakujejo rezultat. Ukazni paket se preko vrste ukaznih paketov /executable instructions pool/ prenese v procesno enoto. Ko se ukaz izvrši, se sestavijo podatkovni ter krmilni paketi. Podatkovni paketi nosijo rezultat ter naslov ukaza, ki čaka nanj in se preko vrste podatkovnih paketov /data pool/ pošljejo v enoto za dostavo in vpis, ki rezultate vpiše v naslovljene lokacije pomnilniške enote. Krmilni paketi pa se preko vrste krmilnih paketov /control token pool/ posredujejo enoti za zbiranje.

6.2. Podatkovno vodene arhitekture

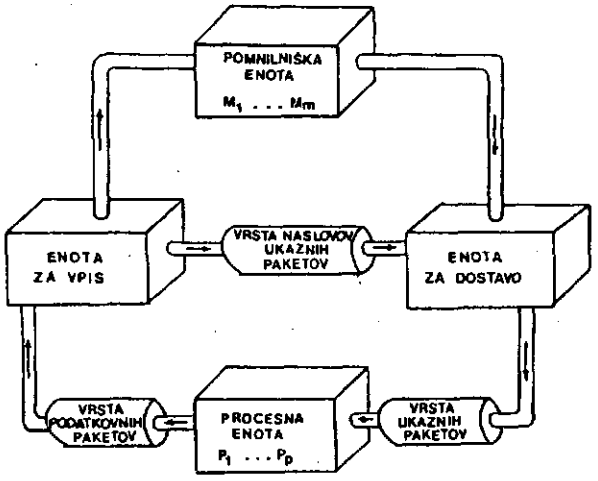
6.2.1. Tok podatkovnih paketov na organizaciji za posredovanje paketov

Model računanja s tokom podatkovnih paketov običajno realiziramo na organizaciji za posredovanje paketov. Takšno arhitekturo imenujemo podatkovno pretokovna arhitektura /data flow architecture/. Ločimo dve vrsti podatkovno pretokovnih arhitektur:

- arhitektura s shranjevanjem paketov /token store/,
- arhitektura z zbiranjem paketov /token matching/.

Prikazani sta na sliki 12 in sliki 13.

6.2.1.1. Podatkovno pretokovna arhitektura s shranjevanjem paketov



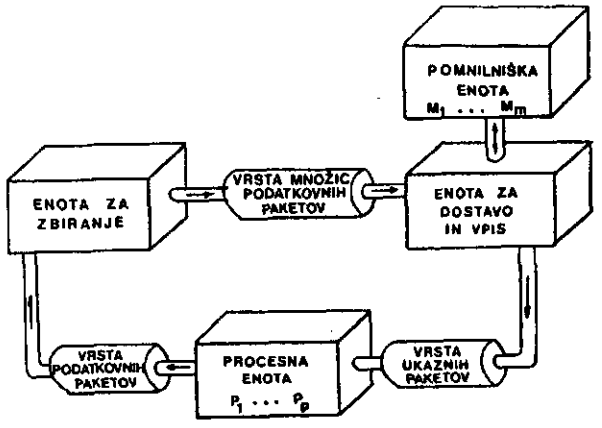
Slika 12. Podatkovno pretokovna arhitektura s shranjevanjem paketov.

Pomnilniška enota /memory unit/ vsebuje ukazne pakete, ki vsebujejo operacijo, prostore za operande ter naslove na njegov rezultat čakajočih ukaznih paketov. Enota za vpis /update unit/ hrani naslov vsakega ukaznega paketa ter število operandov, ki jih mora še prejeti, da bo postal izvršljiv. Pri tej arhitekturi se podatkovni paketi, ki jih sestavi procesna enota /processing unit/, posredujejo preko vrste s podatkovnimi paketi /data token pool/ v enoto za vpis. Vsak podatkovni paket vsebuje poleg rezultata tudi naslov čakajočega ukaznega paketa v pomnilniški enoti. Enota za vpis shrani rezultat prispelega podatkovnega paketa v čakajoči ukazni paket v pomnilniški enoti ter zmanjša število operandov, ki jih mora ta paket še prejeti. Če enota za vpis ugotovi, da je ukazni paket prejel zadnjega izmed potrebnih operandov, pošlje naslov paketa preko vrste naslovov ukaznih paketov /instruction address pool/ enoti za dostavo /fetch unit/. Ta enota nato takoj prenese ukazni paket preko vrste ukaznih paketov /executable instruction pool/ v procesno enoto, kjer se ukaz izvrši. Podatkovni paketi, ki so rezultat izršitve ukaza, se nato zopet posredujejo enoti za vpis.

Opomba: Izvrševanje, pri katerem enota za dostavo prenese izvršljiv ukazni paket v procesno enoto takoj, ko iz enote za vpis prejme njegov naslov, imenujemo takojšnje izvrševanje. S predhodno analizo programskega grafa pa lahko dobimo določeno informacijo, ki služi enoti za dostavo pri izbiranju in prenašanju izvršljivih paketov v procesno enoto. Enota za dostavo lahko tedaj počaka s prenosom na trenutek, ki je najbolj ugoden. S tem je lahko omogočeno izvrševanje dosti 'večjih' programov, ne da bi se minimalni možni čas, potreben za njihovo izvrševanje, podaljšal. Omenjena analiza je podrobno opisana v [Robi86a, Robi86b].

Projekti: Podatkovno pretokovno arhitekturo s shranjevanjem paketov so realizirali na MIT-ju pod vodstvom J.B. Dennisa [Denn83]. Druga realizacija te arhitekture je DDP (Distributed Data Processor), izdelan v Texas Instruments [Corn79]. Zanimivo je, da podpira DDP programirni jezik Fortran 66. Omenimo še francoski projekt LAU (Language à Assignation Unique), ki poteka v CERT laboratoriju, Toulouse [Comt79, Comt80].

6.2.1.2. Podatkovno pretokovna arhitektura z zbiranjem paketov



Slika 13. Podatkovno pretokovna arhitektura z zbiranjem paketov

Pri tej arhitekturi se podatkovni paketi, ki so prispeli iz procesne enote, ne shranijo neposredno v ukazni paket, kateremu so namenjeni. Namesto tega se podatkovni paketi nalagajo v množice, ki se nahajajo v enoti za zbiranje /matching unit/ paketov. Vsaka množica je preko pridruženega naslova prirejena ukaznemu paketu, ki se nahaja v pomnilniški enoti /memory unit/. Ko se množica napolni (prejme zadnji potrebni podatkovni paket), se preko vrste množic podatkovnih paketov /token sets pool/ prenese v enoto za dostavo in vpis /fetch & update unit/. Ta enota prenese iz pomnilniške enote kopijo naslovljenega ukaznega paketa, ga dopolni s prispelo množico podatkovnih paketov, ter ga preko vrste ukaznih paketov /executable instruction pool/ pošlje v procesno enoto /processing unit/. Tu se ukaz izvrši, podatkovni paketi, ki so rezultati izvršitve, pa se zopet prenesejo v enoto za zbiranje paketov.

Opomba: Podobno kot v prejšnji arhitekturi, lahko tudi tu izvršljive ukazne pakete zadržujemo, le da v tem primeru vrši zadrževanje enota za zbiranje.

Projekti: Realiziranih je več projektov, med katerimi omenimo Manchester Data Flow Computer, ki so ga izdelali pod vodstvom I. Watsona in J.R. Gurda na univerzi v Manchestru [Wats79, Gurd 83, Gurd85]. Razširjena različica EXMAN (EXtended MANchester Data Flow Computer) je predlagana v [Patn86]. Tretji projekt poteka na univerzi v Newcastlu, kjer gradijo računalnik JUMBO [Trel82a]. Omenimo še Id (Irvin Dataflow Machine) projekt, začet na kalifornijski univerzi v Irvinu, ki se trenutno nadaljuje na MIT-ju [Arvi80] in ga vodi Arvind.

6.2.2. Tok podatkovnih paketov na organizaciji za obravnavo izrazov

Tok podatkovnih paketov je moč realizirati tudi na organizaciji za obravnavo izrazov. Ko procesor v nekem gradniku prejme podatkovni paket preko komunikacijske enote iz drugega gradnika, vstavi vrednost, ki jo nosi paket v ukaz, kateremu je namenjena. V naslednjem koraku procesor ugotovi, če je ukaz izvršljiv, ter ga v tem primeru izvrši ter pošlje podatkovni paket, v katerem se nahaja rezultat, preko komunikacijske enote tistemu gradniku, v katerem se nahaja ukaz, ki ta rezultat pričakuje. Če pa ukaz še ni izvršljiv, se procesor povrne v stanje čakanja.

Projekti: Data-Driven Machine #1 (DDM1) je primer arhitekture, ki temelji na toku podatkovnih paketov, realiziranem na drevesni organizaciji za obravnavo izrazov [Davi78]. DDM1 so izdelali pod vodstvom A. Davisa na Burroughs Interactive Research Center, La Jolla, California.

6.3. Z zahtevo vodene arhitekture

6.3.1. Redukcija izrazov na organizaciji za obravnavo izrazov

Model z redukcijo izrazov je naravno podprt z organizacijo za obravnavo izrazov [Trel82b]. Kot smo že omenili, so programi v takšnem modelu računanja gnezdeni izrazi. Glavni gnezdeni izraz je porazdeljen po pomnilniških enotah gradnikov organizacije tako, da sta sosedna podizraza vpisana v pomnilniških enotah dveh sosednih gradnikov. Med izvrševanjem se izraz bodisi širi (pri vstavljanju definicij, na katere se operandi sklicujejo) ali pa krči (pri nadomestitvah izračunljivih podizrazov z njihovimi rezultati). Ko procesor pregleda podizraz v lastni pomnilniški enoti ter najde v njem nek še neizračunan operand, zahteva preko komunikacijske enote prenos definicije, na katero se operand sklicuje. Po prihodu zahtevane definicije (preko komunikacijske enote) se povzroči premik ostalega izraza vstran, s čimer se napravi prostor za vstavitev prispele definicije. Pri takšnih premikih lahko seveda preidejo podizrazi iz ene pomnilniške enote v drugo (v drug gradnik). Če pa procesor ugotovi, da je podizraz v njegovi pomnilniški enoti reducibilen (da so vsi operandi operacije znani), ga izračuna ter nadomesti z rezultatom. Posledica takšne redukcije je lahko krčenje celotnega izraza ter s tem premik delov izraza iz ene pomnilniške enote v drugo. Funkcija, ki jo pri računanju opravlja zaporedje pomnilniških enot je torej podobna funkciji dvosmernega premikalnega registra. Ker ima lahko nek podizraz več operandov, ki se sklicujejo na svoje definicije, in ker je ugodno, če se takšni operandi nadomeste z njimi sočasno (kar pa se lahko izvrši le v različnih gradnikih), morajo biti kapacitete pomnilniških enot primerno majhne. Visoka stopnja sočasnosti pri računanju je torej zagotovljena ob zadostnem številu gradnikov ter primerno majhni kapaciteti pomnilniških enot.

Projekti: Omenimo dve realizirani arhitekturi in sicer Newcastle Reduction Machine [Trel80] ter North Carolina Cellular Tree Machine [Mago80], ki ima drevesno organizacijo stroja.

6.3.2. Redukcija grafov na organizaciji za obravnavo izrazov

Tudi model z redukcijo grafov je moč realizirati na organizaciji za obravnavo izrazov [Trau85]. Graf se pred izvrševanjem smiselno porazdeli na gradnike (pomnilnike). Izračuni definicij potekajo v gradnikih, kjer so definicije argumentov, rezultati pa se posredujejo gradnikom, ki so posredovali zahtevo. Zato gradnik skupaj z zahtevo pošlje tudi svoj naslov, ki se vpiše v definicijo argumenta.

Za realizacijo te arhitekture je primernejša organizacija stroja, kjer vsebujejo gradniki le procesor in komunikacijsko enoto, pomnilnik pa je združen. Deli pomnilnika so bodisi vnaprej dodeljeni gradnikom, ali pa je celoten pomnilnik skupen za vse gradnike.

6.3.3. Redukcija grafov na organizaciji za posredovanje paketov

Pri redukciji je možno uporabiti organizacijo za posredovanje paketov le v modelu z redukcijo grafov. Izvrševanje programa na takšni organizaciji poteka z dvema vrstama paketov:

- paketi z zahtevo /demand token/,
- paketi z rezultatom /result token/.

Paket z zahtevo vsebuje:

- naslov ukaza, ki lahko izračuna zahtevano vrednost,
- naslov ukaza, ki je zahteval to vrednost.

Paket z rezultatom pa vsebuje:

- rezultat izvršitve ukaza,
- naslov ukaza, ki je ta rezultat zahteval.

Če operandi ukaza, katerega izvršitev je bila predhodno zahtevana, še niso znani, pošlje tak ukaz pakete z zahtevo vsem tistim ukazom, ki lahko te vrednosti izračunajo. Ko ukaz sprejme paket z zahtevo, se izvrši (pred tem se lahko postopek pošiljanja paketov z zahtevami ponovi), ter pošlje paket z rezultatom ukazu, ki ga je zahteval. Tedaj se lahko tudi ta ukaz izvrši. Sočasni sprožitveni mehanizem je določen z dvema pravilomi:

- za sprožitev ukaza je potreben natanko en paket z zahtevo,
- za izračun ukaza morajo biti na voljo vsi potrebni paketi z rezultati.

Projekti: Značilen predstavnik je AMPS (Utah Applicative Multiprocessing System), zgrajen na univerzi v Utahu [Kell79]. Omenimo tudi projekta na Imperial College v Londonu [Darl81] ter na univerzi East Anglia [Slee81].

6.3.4. Redukcija izrazov na centralizirani organizaciji

Pri realizaciji modela z redukcijo izrazov imajo enote v centralizirani organizaciji sledeče značilnosti:

- pomnilniška enota združuje določeno število skladov /stack/, med katerimi se ponavljajoče prenaša izraz,
- procesna enota skrbi za prenos trenutnega izraza iz začetnega /source/ sklada v končni /sink/ sklad. Pri prenašanju izraza procesna enota razpoznava njegove reducibilne podizraze, jih sproti izračunava (reducira) ter namesto njih shranjuje v končni sklad njihove rezultate. Po prenosu celega izraza prevzame končni sklad naloge začetnega in obratno. Prenašanje se ponavlja vse dokler ni izraz atomaren,

- komunikacijska enota je vodilo, po katerem se prenašajo izrazi iz začetnega sklada v procesno enoto in iz te v končni sklad.

Vsak izraz (in s tem tudi glavni izraz - program) je zapisan v prefiksni obliki, pri čemer so v taki obliki zapisani tudi vsi njegovi podizrazi. Pri prenosu izraza iz začetnega sklada v končni se mora takšna oblika ohraniti, zato navadno prenos poteka preko dodatnega, vmesnega /intermediate/ sklada.

Projekti: Primer takčne arhitekture je GMD (Gesellschaft für Mathematik und Datenverarbeitung), ki so jo realizirali v Bonnu, ZRN [Klug79].

7. Zaključek

Predstavljena razdelitev novogeneracijskih arhitektur je le ena iz vrste možnih razdelitev. Kot je že bilo omenjeno, je pri tej razdelitvi osnovno vodilo vodenje računanja. Marsikatera realizirana arhitektura težko najde mesto v eni sami predlagani poddružini (poglavje 6), saj je v novejših arhiteturah pričakovati različne kombinacije podatkovnega in sprožitvenega mehanizma, s tem pa tudi mešanih modelov računanja ter vodenja. Omenimo vzorčno vodenje /pattern driven/ računanja, kjer se ukaz prične izvrševati, ko pride do ujemanja vzorca, ki je pogoj za pričetek izvrševanja [Trel84]. Pojavljajo se tudi kompleksnejše organizacije stroja. Kot ilustracijo navedimo le dva najnovejša podatkovno pretokovna računalnika: SIGMA-1 [Shim86] in večprocesorski podatkovno pretokovni sistem na osnovi procesorjev μPD7281 [Jeff85, Silc86]. V obeh primerih gre za organizacijo stroja, ki je podobna organizaciji za obravnavo izrazov, le da so gradniki podatkovno pretokovni procesorji, katerih organizacija temelji na posredovanju paketov.

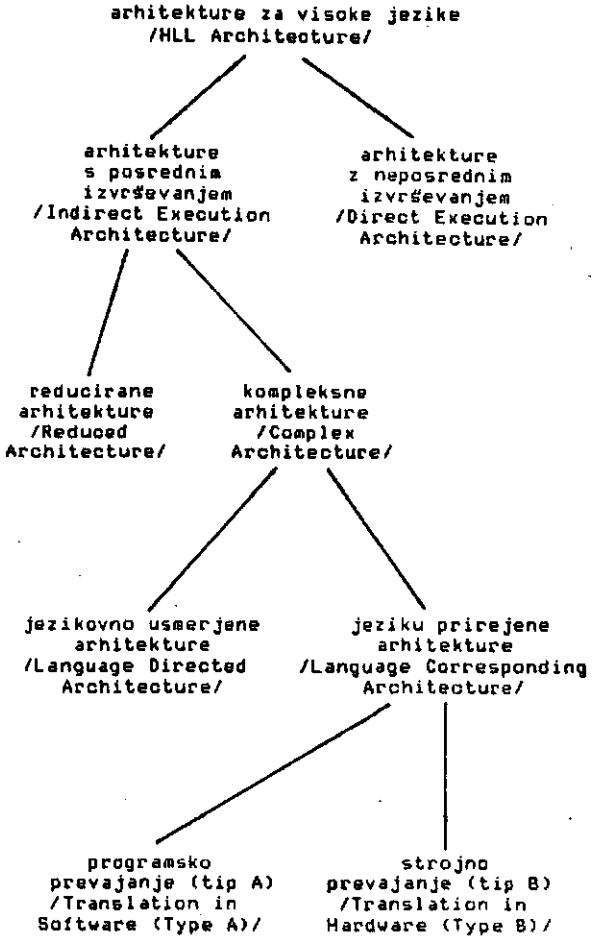
Računalniške arhitekture je moč razdeliti tudi z dveh drugih, diametralnih zornih kotov: z vidika jezikov oziroma strojne opreme.

Tako obstaja razdelitev novejših računalniških arhitektur, temelječih na visokih programirnih jezikih /high level language computer architectures/. Takšna razdelitev je podana v [Milu86] in jo prikazuje slika 14.

Čeprav ni namen članka predstavitev takšne razdelitve arhitektur, pa vendarle omenimo skupino reduciranih arhitektur, med katere sodijo IBM 801, RISC (UC Berkely), MIPS (Stanford), RIMMS (univerza Reading), Transputer (Inmos) ter VM (Purdue). Osnova teh arhitektur je strojno realiziran nabor najpogosteje prisotnih ukazov v prevedeni kodi programov (RISC nabor). Redkeje uporabljani ukazi se nadomestijo z ustreznimi zaporedji ukazov iz RISC nabora [Patt85].

Za konec omenimo še dve možni delitvi arhitektur z drugega zornega kota, to je na osnovi strojne opreme. Morda najpogostejša delitev računalniških arhitektur je po Flynnu [Flyn72], ki jih deli v štiri skupine:

- SISD (en ukazni tok in en podatkovni tok) /single instruction stream & single data stream/,
- SIMD (en ukazni tok in več podatkovnih tokov) /single instruction stream & multiple data stream/,
- MISD (več ukaznih tokov in en podatkovni tok) /multiple instruction stream & single data stream/ ter
- MIMD (več ukaznih tokov in več podatkovnih tokov) /multiple instruction stream & multiple data stream/.



Slika 14. Razdelitev računalniških arhitektur, temelječih na visokih prog. jezikih.

Shoreova delitev [Shor73] pa obsega šest razredov računalniških arhitektur glede na združevanje materialnih virov, kot so procesne enote PE /processing units/, krmilne enote KE /control units/, ukazni pomnilniki UP /instruction memories/ in podatkovni pomnilniki PP /data memories/. Teh šest razredov je:

- stroj I: so klasične von Neumannove arhitekture, z možnostjo cevljenja v PE,
- stroj II: je podoben stroju I, le da se sočasno dostavijo bitne rezine /bit slice/ vseh besed iz PP; torej v nasprotju s strojem I, stroj II obdeluje besede 'prečno' in ne 'vzdolžno',
- stroj III: je kombinacija prvih dveh strojev, zato lahko obdeluje besede bodisi 'prečno' ali 'vzdolžno',
- stroj IV: je arhitektura, ki preko ene KE združuje množico parov PE in PP, kjer PE-ji nimajo možnosti medsebojne komunikacije,
- stroj V: je podoben stroju IV, pri čemer lahko vsak par PE in PP komunicira s svojima sosednima paroma, ter
- stroj VI: za razliko od strojev I do V, kjer je PE ločen od PP, so tu funkcije PE in PP združene v eni sami enoti /logic-in-memory array/.

8. Ključne besede

- podatkovnih paketov /data token */, 6.1.3, 6.2.1.1, 6.2.1.2
- ukaznih paketov /executable instruction */, 6.1.3, 6.2.1.1, 6.2.1.2

Pripis: Pri prevajanju angleških izrazov sva nekajkrat ubrala nekoliko svobodnejšo pot, predvsem takrat, ko bi dobesedni prevod slabo opisoval dotični predmet. Upava, da sva imela pri izbiri slovenskih izrazov srečno roko.

9. Literatura

[Arvi80] Arvind, Kathail V., Pingali K. "A Processing Element for a Large Multiprocessor Dataflow Machine", Proc. Int'l. Conf. Circuits and Computers, New York, Oct. 1980.

[Back78] Backus J. "Can programming be liberated from the von Neumann style? A functional style and its algebra of programms", Comm. ACM Vol.21, No.8, Aug. 1978, pp.613-641.

[Berk71] Berkling K.J. "A computing machine based on tree structure", IEEE Trans. Computer, Vol.C-20, No.4, Jan. 1971, pp.404-418.

[Bish86] Bishop P. Fifth Generation Computers, Concepts, implementations and uses, Ellis Horwood, 1986.

[Braj86] Brajak P. "Paralelno procesiranje: arhitektura 90-tih godina, Zbornik jugoslavenskog savjetovanja o novim generacijama računala, MIPRO 86, Rijeka, Maj 1986, str.33-46.

[Corn79] Cornish M. "The TI Data Flow Architectures: The Power of Concurrency for Avionics", Proc. 3rd Conf. Digital Avionics Systems, Fort Worth, Texas, Nov. 79, pp.19-25.

[Comt79] Comte D., Hifdi N. "LAU Multiprocessor: Microfunctional Description and Technological Choices", Proc. 1st European Conf. Parallel and Distributed Processing, Toulouse, France, Feb. 1979, pp.8-15.

[Comt80] Comte D., Hifdi N., Syre J.C. "The Data Driven LAU Multiprocessor System: Results and Perspectives", Proc. IFIP 80 Congress, Oct. 1980.

[Davi78] Davis A.L. "The Architecture and System Method of DDM1: A Recursively Structured Data Driven Machine", Proc. 5th Ann. Symp. Comp. Arch., Palo Alto, Calif., April 3-5, 1978, pp.210-215.

[Davi82] Davis A.L., Keller R.M. "Data Flow Program Graphs", Computer, Vol.15, No.2, Feb. 1982, pp.26-41.

[Darl81] Darling J., Reeve M. "ALICE: A Multiprocessor Reduction Machine for the Parallel Evaluation of Applicative Languages", Proc. Int'l Symp. Functional Programming Languages and Computer Architecture, June 1981, Göteborg, Sweden, pp.32-62.

[Denn74] Dennis,J.B., Misunas,D.P. "A Computer Architecture for highly parallel signal processing", Proc.1974 Nat. Computer Conf., AFIPS Press, Arlington, Va.,1974,pp.402-409.

[Denn83] Dennis J.B., Lim W.Y-P., Ackerman W.B. "The MIT Data Flow Engineering Model", Information Processing 83, R.E.A. Mason (ed.), Elsevier Scince Publishers B.V. (North-Holland), 1983.

[Flyn72] Flynn M.J. "Some Computer Organizations and their Effectiveness", IEEE Trans. Comp. Vol.C-21, No.9, Sept. 1972, pp.948-960.

[Gilo83] Giloi W.K. "Towards a Taxonomy of Computer Architecture Based on the Machine Data Type View", The 10th Ann. Int'l Symp. Comp. Arch., June 13-17, 1983, Stockholm, Sweden, pp.6-13.

[Gurd83] Gurd J.R., Watson I. "Preliminary Evaluation of a Prototype Dataflow Computer", Information Processing 83, R.E.A. Mason (ed.), Elsevier Scince Publishers B.V. (North-Holland), 1983.

[Gurd85] Gurd J.R., Kirkham C.C., Watson I. "The Manchester Prototype Dataflow Computer", Comm. ACM, Vol.28, No.1, Jan. 1985, pp.34-52.

[Jeff85] Jeffery T. "The µPD7281 Processor", Byte, November 1985, pp.237-246.

[Kapa84] Kapauan A., Field J.T., Gannon D.B., Snyder L. "The Pringle Parallel Computer", The 11th Ann. Int'l Symp. Arch., June 5-7, 1984, Ann Arbor, Michigan, pp.12-20.

[Kell79] Keller R.M. et.al. "A Loosely Coupled Aplicative Multiprocessing System", Proc. Nat. Comp. Conf., 1979, pp.861-870.

[Mago80] Magó G.A. "A Cellular Computer Architecture for Functional Programming", Proc. IEEE COMPCON 80, Feb. 1980, pp.179-187.

[Milu86] Milutinović V. Tutorial on Advanced Microprocessors and High-Level Language Computer Architecture, IEEE Comp. Soc. Press, 1986.

[Patn86] Patnaik L.M., Govindarajan R., Ramadoss N.S. "Design and Performance Evaluation of EXMAN: An EXtended MANchester Data Flow Computer", IEEE Trans. Comp. Vol. C-35, No.3, March 1986, pp.229-244.

[Patt85] Patterson D.A. "Reduced Instruction Set Computers", Comm. ACM, Vol.28, No.1, Jan. 1985, pp.8-21.

[Prei85] Preiss B.R., Hamacher V.C. "Data Flow on a Queue Machine", The 12th Ann. Int'l Symp. Comp. Arch., June 17-19, 1985, Boston, Massachusetts, pp.342-351.

[Robi85a] Robič B., Šilc J., Mihovilovič B. "Funkcionalno programirni sistemi", Informatica 4/85, str.366-370, Nova Gorica, 1985.

[Robi85b] Robič B, Šilc J.: "Jeziki in arhitekture sodobnih računalniških sistemov", IJS DP-4058, Ljubljana, November 1985.

[Robi86a] Robič B., Šilc J., "On Choosing a Plan for the Execution of Data Flow Program Graph", Informatica 3/86, pp.11-17.

[Robi86b] Robič B., Šilc J., "Analiza statičnih podatkovno pretokovnih programskih grafov", Elektrotehniški vestnik 86/2, str.53-56.

[Shim86] Shimada T., Hiraki K., Nishida K., Sekiguchi S. "Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computations", Proc. 13th Int'l Symp. Comp. Arch., June 1986, pp.226-234.

[Shor73] Shore J.E. "Second Thoughts on Parallel Processing", Comput. Elect. Eng., Vol.1, 1973, pp.95-109.

[Slee81] Sleep M.R., Burton F.W. "Towards a Zero Assignment Parallel Procesor", Proc. 2nd Int'l Conf. Distributing Computing, April 1981.

[Snyd84] Snyder,L. "Supercomputers and VLSI: The Effect of Large-Scale Integration on Computer Architecture", Advances in Computers, Vol.23, Marshall C. Yovits, Ed.,Academic Press, 1984.

[Ston75] Stone,H.S. Introduction to Computer Architecture, SRA,Chicago 1975.

[Silc84] Šilc J., Robič B. "Računalniki krmiljeni s tokom podatkov", IJS DP-3579,Ljubljana, Oktober 1984.

[Silc85a] Šilc J., Robič B.: "Osnovna načela OF sistemov", Informatica 2/85, str.10-15.

[Silc85b] Šilc J., Robič B., Mihovilovič B. "Podatkovno vodene računalniške arhitekure", Informatica 4/85, str.371-376.

[Silc86] Šilc J., Robič B. "Procesor s podatkovno pretokovno arhitekturo",Informatica 4/86.

[Traub85] Traub K.R. "An Abstract Parallel Graph Reduction Machine", The 12th Ann. Int'l Symp. Comp. Arch., June 17-19, 1985, Boston, Massachusetts, pp.333-341.

[Trel80] Treleaven P.C., Mole G.F. "A Multi-processor Reduction Machine for User-defined Reduction Languages", Proc. 7th Int'l Symp. Comp. Arch., May 6-8, 1980, pp.121-130.

[Trel81] Treleaven P.C., Hopkins R.P. "Decentralized Computation", The 8th Ann. Symp. Comp. Arch., May 12-14, 1981, Minneapolis, Minnesota, pp.279-290.

[Trel82a] Treleaven P.C., Brownbridge D.R., Hopkins R.P. "Data-Driven and Demand-Driven Computer Architecture", Comp. Surv., Vol.14, No.1, March 1982.

[Trel82b] Treleaven P.C., Hopkins R.P. "A recursive Computer Architecture for VLSI", The 9th Ann. Symp. Comp. Arch., Apr. 26-29, 1982, Austin, Texas, pp.229-238.

[Trel82c] Treleaven P.C., Hopkins R.P., Rautenbach P.W. "Combining Data Flow and Control Flow Computing", Comput.J., Vol.25, No.1, Feb. 1982.

[Trel84] Treleavan P.C., Lima I.G. "Future Computers: Logic, Data Flow,...,Control Flow?", Computer, Vol.17, No.4, March 1984, pp.47-57.

[Wats79] Watson I., Gurd J. "A Prototype Data Flow Computer with Token Labeling", Proc. Nat. Computer Conf., New York, N.Y., June 4-7, 1979, pp.623-628.

[Zele85] Železnikar A.P. "Mednarodna konferenca o računalniških sistemih pete generacije v Tokyu", Informatica 85/3, str.68-70.

MOŽNOSTI, KI JIH PONUJA MIKROELEKTRONIKA SISTEMSKIM NAČRTOVALCEM

INFORMATICA 4/86

UDK: 681.327.6:621.38.049.7-181.4

M. Jenko in A. Vodopivec
Iskra Mikroelektronika

Podan je pregled mikroelektronskih tehnologij, njihovih razvojnih možnosti, vrst današnjih mikroelektronskih vezij, smernic razvoja načrtovalske opreme. Dani so kriteriji racionalnega pristopa sistemskega načrtovalca k mikroelektroniki. Opisan je načrtovalski postopek od zasnove logične sheme do narejenih mask v DO Mikroelektronika.

Possibilities Offered to Microelectronic System Designer

A view of microelectronics technologies, their growth alternatives, sorts of present integrated circuits, directives for evolution of design support are given. Criterions on design solutions for custom integrated circuits from the system designer's view are described. Design procedure beginning with completed logical scheme to process masks in Iskra Microelectronics div. is depicted.

Gradivo je razdeljeno na :

1. Primerjava obstoječih mikroelektronskih tehnologij in načrtovanja po možnostih, ki jih nudijo načrtovalcem
2. Smernice razvoja tehnologije, razvoja načrtovanja mikroelektronskih vezij
3. Oris načrtovalskega postopka od zasnove logične sheme do narejenih mask v DO Mikroelektronika

1 Primerjava obstoječih mikroelektronskih tehnologij in načrtovanja po možnostih, ki jih nudijo načrtovalcem

Danes obstoje tri osnovne veje mikroelektronskih tehnologij:
- planarna tehnologija na siliciju in galijevem arzenidu
- debeloslojna tehnologija
- tankoslojna tehnologija

1.1 Planarna tehnologija - osnovni tehnološki postopki:

- kontrolirano vnašanje primesi v silicij z difuzijo, ionsko implantacijo
- termično ali plazemsko stimulirana rast plasti na siliciju
- naparevanje, naprševanje metalnih povezav med elementi
- litografski postopki za definiranje topologije vezja
- debeloslojna oksidacija površine silicija za zaščito vezij, tankoslojna oksidacija za definiranje aktivnih območij tranzistorjev

1.2 Debeloslojna tehnologija

Osnovni tehnološki postopek je sitotisk. Nanašamo prevodne, uporovne, dielektrične, izolacijske paste preko sita, na katerem je definirana geometrija vezja, na keramično podlogo. Sledi zapečenje posameznih nanesenih plasti. Oblikujemo lahko upore, kondenzatorje, prevodne steze.

1.3 Tankoslojna tehnologija

Osnovni tehnološki postopek je vakuumsko naparevanje in ionsko naprševanje. Oblikujemo pasivne komponente in tankoslojne tranzistorje. Njihove karakteristike ne dosegajo monolitnih tranzistorjev.

Če se le da, se vezja realizira v planarni tehnologiji. Glavni problem je, da imamo za vse elemente, ki jih hočemo narediti, na razpolago le ene in iste tehnološke postopke. Zato je na ta način težko izdelati kombinacijo npr. hitre logike in nekaj amperov močnega izhodnega tranzistorja. Zato imata svojo vrednost tudi tankoslojna in debeloslojna tehnologija, kjer aktivne dele vezja prilepimo na podlago, povežemo in lahko opremimo s še nekaj pasivnimi komponentami, zapremo v ohišje ter prodajamo kot zaključeno vezje.

Razvoj gre v smer vse večje integracije različnih gradnikov vezja samo na rezini - planarna tehnologija.

Planarnih integriranih vezij je danes na tržišču ogromno. Lahko jih klasificiramo po tipu osnovnega aktivnega elementa (1.3.1), po stopnji integracije (1.3.2), po uporabi in pristopu k izdelavi vezja (1.3.3).

1.3.1 Bipolarna in unipolarna vezja.

Osnovni gradnik bipolarnih vezij je bipolarni tranzistor. Ta vezja so načeloma hitrejša od unipolarnih, so manj obcutljiva na elektrostatiko, njihova izdelava zahteva več tehnoloških postopkov, statična poraba je večja, porabimo več prostora.

Osnovni gradnik unipolarnih vezij je MOS tranzistor. Locimo:
- PMOS vezja
- NMOS vezja
- CMOS vezja

Začelo se je s PMOS vezji - aktivni gradnik vezja je p kanalni tranzistor. Tehnologija je manj zahtevna kot za NMOS in CMOS. Pragovna napetost je razmeroma visoka, gibljivost nosilcev toka - informacije je razmeroma nizka, zato so ta vezja sorazmerno počasna. Tudi brez preklopov porabljajo za svoje delovanje energijo, zato za aplikacije z nizko porabo niso primerna.

NMOS vezja so tehnološko zahtevnejša, pragovna napetost tranzistorjev je nižja, gibljivost nosilcev toka je približno še enkrat večja kot pri tranzistorjih s p kanalom, zato so ta vezja hitrejša. Slabost pa je še vedno statična poraba energije.

CMOS - /Complementary Metal Oxide Semiconductor/ vezja združujejo na isti rezini p in n kanalne tranzistorje. Vedno prevodni pull-up tranzistor PMOS in NMOS vezij je nadomeščen z aktivnemu komplementarnim tranzistorjem. Prevodna proga je naenkrat sklenjena le na nic ali na napajalno napetost in tako v mirovanju ni prevajanja med niclo in napajalno napetostjo. V mirovanju vezje zato praktično nima porabe. Omejitev so le plazeci tokovi nekaj mikroamperov. Ta vezja so tehnološko najbolj zahtevna, so malo večja kot logični n ali p kanalni ekvivalent, zelo so primerna za aplikacije z željeno majhno porabo.

Načrtovalsko ima vsaka družina vezij svoje male skrivnosti, osnovni principi so enaki.

1.3.2 Delitev po stopnji integracije

- SSI - Small Scale Integration - v vezju je do sto elementov
- MSI - Medium Scale Integration - v vezju je med sto in tisoč elementov
- LSI - Large Scale Integration - v vezju je med tisoč in deset tisoc elementov
- VLSI - Very Large Scale Integration - v vezju je med deset tisoc in dvesto tisoc elementov
- ULSI - Ultra Large Scale Integration - v vezju je nad dvesto tisoč elementov.

1.3.3 Delitev po uporabi in pristopu k načrtovanju vezja

- Standardna vezja
- Standardna LSI vezja
- LSI vezja po naročilu= ASIC vezja /Application Specific IC's/

Začelo se je z uporabo standardnih vezij, ker drugih v začetku ni bilo. Sem sodijo družine kot serija logičnih vezij 74, 74LS, 54, 4000 ipd, enostavna analogna vezja (operacijski ojačevalniki, komparatorji, stabilizatorji napetosti in tokov ipd) in diskretni elementi (tranzistorji, diode, senzorji ipd). Prednosti standardnih vezij so, da se jih v dolgem časovnem obdobju da kupiti praktično povsod, so relativno poceni, spremembe sistema so relativno neboleče tudi potem, ko je sistem že načrtan. Slabost je obseznost sistema, s tem združena poraba, zanesljivost je zaradi velikosti in stevila tiskanih povezav manjša, končna cena izdelka je v primeru velikoserijske proizvodnje velika, saj je evropa kartica z dvajsetimi vezji gotovo dražja od enega vezja, ki bi opravljalo isto funkcijo.

Standardna LSI vezja so:
- elementi procesorskih družin (mikroprocesorji,PIA, ACIA, ROM, RAM, timerji ipd),
- modemi, filtri, linijski oddajniki in sprejemniki, naročniška vezja za telefonske centrale.

Z standardnim LSI vezjem običajno rešimo ključne probleme sistema, sistem je običajno zanesljiv, majhen, cenen, poraba energije je majhna. Slabost načrtovanja s takimi vezji je, da ta vezja načrtovalcu sistema vsiljujejo svojo filozofijo delovanja (b skupina), zato pogosto ne more vgraditi v sistem vseh željenih funkcij, kakšne funkcije vezij pa ostajajo neizrabljene.

ASIC vezja so vezja narejena po naročilu in zasnovi načrtovalca. Sistemski načrtovalec si zamisli sistemsko zasnovo, ključne dele sistema realizira z vezji po lastni zamisli, lahko uporabi delno tudi standardna vezja. Tak pristop k stvari je gledano s stališča sistema vsekakor najbolj racionalen in sistem ima tako največ že ob nastanku vgrajenih lastnosti za uspešno trzenje.

Načrtovalec in njegov organizacijski vodja se morata zavedati določenih zakonitosti, povezanih z ASIC vezji. Praviloma je razvoj sistema s po lastni zamisli načrtanimi vezji dražji od razvoja sistema s pomočjo standardnih komponent. Tudi traja malo dalj. V primeru pametnega razdeljevanja sistema na vezja in vestnega načrtovanja vezij pa ima sistem nesluteno boljše možnosti za prodajo in doseganje večje cene na trgu kot sistem grajen izključno iz standardnih komponent. Ekonomika tako zastavljenega projekta je na koncu odločno pozitivna in zahtevnejše izdelke lahko delamo konkurenčne samo še na ta način.

ASIC vezja delimo po načinu njihovega načrtovanja na:
- ULA in UAA vezja - Uncommited Logic Array, Uncommited Analog Array, logične mreze, analogne mreze
- PLA vezja - Programmable Logic Array - programirane mreze
- CLSI vezja - Custom LSI - vezja po naročilu:
  - Full Custom - čisto načrtovanje po naročilu
  - Standard Cell Design - načrtovanje s standardnimi celicami

Logične mreže so vezja, ki so specena na zalogo. Kaj bo tako vezje delalo, določimo s predzadnjo masko - definicijo metalnih povezav med že prej narejenimi tranzistorji. V takem vezju je odvisno od tipa med sto in deset tisoc tranzistorjev. Načrtovalcu je prepuščeno, koliko jih bo uporabil. Dimenzije tranzistorjev so predefinirane, velikost vezja tudi. Predefinirane so tudi osnovne metalne povezave za izdelavo logičnih vrat, flip flopov, stevcev, pomikalnih registrov. Tudi analognim mrežam določimo delovanje samo z metalno masko - definicijo povezav med

osnovnimi gradniki. Tako največkrat oblikujemo filtre s specifičnimi lastnostmi.

Programirane mreže uporabnik sprogramira sam, da se mu na dan vhodni vzorec pojavi na izhodu želeni izhodni vzorec. Ta vezja praviloma z ozirom na zmogljivosti niso majhna, programiramo vsako posebej.

Vezja čisto po naročilu so razvojno izredno draga in ni garancije za zelo kratek razvojni čas. Načrtovalec si za zastavljen problem izbere standardne celice, ki jih razmesti in poveže tako, da porabi čim manjšo površino silicija. Vkolikor rešitev v obliki standardnih celic za vse dele vezja nima, mora narediti nove standardne celice oziroma dele vezja na novo.

Vezja iz standardnih celic so tista vezja po naročilu, kjer za vse dele vezja že obstoje preizkušene standardne celice, ki jih načrtovalec poveže v delujočo celoto. Standardne celice in povezave med njimi razmeščamo v vzdolžnih vrstah tako, da dobimo ob siceršnji omejitvi načrtovalske svobode z vzdolžnimi vrstami čim manjše dimenzije in optimalno delovanje vezja. Izdelamo vse maske za zahtevano vezje - ne le maske povezav, saj geometrija vezja ni vnaprej definirana. Prihranimo na površini silicija, izboljšamo lastnosti vezja, razvojni stroški so večji kot pri mrežah.

Je več važnih kriterijev, kako se odločiti za pravo ASIC vezje za kar najboljši ekonomski učinek. Napačna odločitev na tem nivoju je lahko katastrofalna, saj potegne za sabo od predragega izdelka do popolne zamude na trgu, kar pomeni veliko izgubo, nadaljnji odpor do take izdelave sistema in gotovo zato nadaljnjo nekonkurenčnost.

Kriteriji:
- velikost serije
- cena sistema
- zahtevana zanesljivost
- število in kompleksnost funkcij
- površina vezja
- realizacija analognih in digitalnih vezij
- poraba energije
- posebne zahteve
- doravnavanje lastnosti vezij
- možnost nadaljnjega razvoja sistema
- možnost načrtovanja in proizvodnje doma
- roki za izdelavo
- konkurenčnost na trgu

Velikost serije - Stroški izdelanega vezja se delijo na začetne in proizvodne stroške. Jasno je, da majhne serije zelo slabo prenesejo visoke začetne stroške in da gre pri velikih serijah predvšem za manjšanje proizvodnih stroškov, pa četudi je zato načrtovanje nekoliko dražje. Tako ima vsaka vrsta ASIC vezij svojo količino, v kateri ima pogoje za optimalno ekonomsko učinkovitost. Optimalna količina za PLA vezja je do nekaj deset tisoč, za ULA in UAA med nekaj tisoč in dvajset tisoč, za vezja po naročilu od nekaj deset tisoč naprej.

Cena sistema - Z ozirom na tržne zakonitosti, katerim je naš izdelek podrejen, se že v začetku odločimo, ali je končni cilj nizka cena sistema ali velike zmožnosti, zaradi katerih bo sicer dražji, pa zaradi zmožnosti ne bo imel dovolj velike konkurence, ki bi zavrla njegovo prodajo. Če je kriterij cena, bomo uporabili čim več standadnih komponent in ASIC vezja pridejo v postev le pri velikih serijah in se to v poceni obliki (plastična DIL ohišja). Če je kriterij velika zmožnost izdelka, pogosto uporabimo vezja po naročilu v kakovostnejših ohišjih. Zaključene enote običajno realiziramo z vezji po naročilu.

Zahtevana zanesljivost je dejavnik v prid vezjem ASIC, saj je vzrok odpovedi sistema navadno stik preveč ali premalo, za kar je v integriranem sistemu bistveno manj možnosti.

Število in kompleksnost funkcij nam pomagata pri odločitvi med vezjem po naročilu in med predefiniranimi mrežami. Vkolikor so vezja mešana digitalno analogna, seveda izberemo vezje po naročilu.

Površina vezja je zelo ozko povezana s ceno vezja. Cena narašča z velikostjo vezja več kot linearno, saj je zaradi napak v kristalni strukturi silicijeve rezine dober le določen odstotek vezij. Izplen pada z velikostjo vezij, saj je pri večjem vezju večja verjetnost, da se bo na njem pojavila napaka. Sledi nelinearno višja cena vezja. Priznana rešitev je načrtovanje vezja v tehnologiji s čim manjšo osnovno geometrijo - pet, trije mikroni, večkrat tudi manj in načrtovanje po naročilu, kjer načrtovalec praviloma dobro izrabi površino silicija.

Poraba energije se z uporabo ASIC vezij navadno drastično zmanjša. Nastopijo pa lahko problemi s temperaturno disipacijo, vkolikor zahtevamo tokovno zmogljivejše izhode. Rešitev je prigradnja diskretnih komponent, kar pa je kompromis z zanesljivostjo. Z energetskega stališča so najugodnejša vezja CMOS, ki praktično nimajo statične porabe.

Posebne zahteve navadno lahko uresničimo le z vezjem po naročilu. Take zahteve so povezane tudi s posebno ceno, saj samo z racionalnim načrtovanjem vedno vseh zahtev ne moremo uresničiti. Večje sistemske hiše razvijajo za potencialno ekonomsko zanimive posebne zahteve celo posebne tehnologije kot na primer za vezja, kjer je na isti rezini logika in močnostni elementi. Tudi velika hitrost je posebna zahteva. Klasične simulacije na hitrostni meji dane tehnologije odpovedo. Logična vrata simuliramo kot analogen element. Pred standardnimi komponentami ima ASIC vezje glede hitrosti vgrajeno prednost, saj je vsota parazitskih kapacitivnosti priključkov in povezav gotovo manjša. Tako lahko s pet mikronsko CMOS tehnologijo dosežemo do 25 MHz, s tri mikronsko CMOS tehnologijo do 100 MHz hitrosti.

Doravnavanje lastnosti vezij je gotovo želena možnost, saj se tako izognemo raznim potenciometrom, trimerjem, doravnavanju lastnosti vezja na tiskani plošči, ki je praviloma dražje od doravnavanja na silicijevi rezini pred inkapsulacijo. Doravnavamo tako, da izdelamo na vezju vrsto prežigalnih elementov, ki jih prežigamo s tokom ali z laserskim žarkom. S tem povežemo ali razklenemo doravnalne elemente; navadno precizno umerimo uporovno verigo, določimo kapaciteto kondenzatorja. Seveda je prežiganje ireverzibilen postopek in moramo pred prežiganjem izmeriti karakteristike vezja ter izračunati, katere elemente moramo prežgati, da bomo dosegli željene lastnosti. Zato je doravnavanje že del testiranja silicijeve tabletke na rezini in ga krmilimio z računalnikom testne naprave.

Možnost nadaljnjega razvoja sistema je vedno dobrodošla in nujna, vkolikor naj bo sistem vsaj par let tržno zanimiv. Zato moramo pri razdelitvi sistema paziti, da ga razdelimo na smiselne vase zaključene celote s čim manj zunanjimi povezavami. Modularnost omogoča, da posamezne dele vezja spreminjamo ali naredimo v novi tehnologiji, ne da bi zato morali

spreminjati cel sistem.

Možnost načrtovanja in proizvodnje doma zniža stroške vezja tudi za velikostni razred. Stroški so dinarski. Po tem kriteriju realiziramo vezja v MOS tehnologiji.

Roki za izdelavo nas včasih prisilijo v izbiro enostavnejših ASIC vezij - PLA, ULA in rešitve zato tehnično lahko v prvem koraku niso najboljše. Vendar je bolje prodati slabši sistem, ta sistem nato naprej razvijati kot priti na trg za vso konkurenco. V splošnem se čas načrtovanja z večanjem načrtovalskih izkušenj in uvajanjem simulatorjev, računalniške grafike, krajša.

Konkurenčnost na trgu je posledica majhnih, zmogljivih, cenenih sistemov, kar ASICs vsekakor omogočajo veliko bolje kot samo standardne komponente.

2 Smernice razvoja

Pri smernicah razvoja se kažeta dokaj ostro ločeni področji razvoja tehnologije in razvoja načrtovanja.

2.1 Smernice razvoja tehnologije

Razvoj tehnologije gre v več smeri, saj ima industrija več zahtev, ki niso vse združljive in naenkrat zahtevane. Glavne razvojne smernice so sledeče:

- skaliranje dimenzij
- izdelava Smart Power vezij
- izdelava BiMOS vezij
- izdelava vezij na galijevem arzenidu

2.1.1 Skaliranje

Danes so industrijski standard vezja z minimalno dimenzijo dveh mikronov. Če je minimalna dimenzija (običajno je to dolžina tranzistorja) manjša, je celo vezje manjše, načelno hitrejše, manj porabi. Same dobre stvari. Je pa vezje gotovo dražje, saj ob manjšanju dimenzij nastopi vrsta stranskih pojavov, ki jih je treba upoštevati ali eliminirati.

Po Dennardovem pravilu pri skaliranju zmanjšamo vse geometrije in napajalno napetost s faktorjem skaliranja S, koncentracijo nečistoč v substratu povečamo z 1/S. Namen tega je izdelati manjše elemente s karakteristikami večjih elementov. Teoretično s faktorjem skaliranja S zmanjšamo tok skozi elemente, zmanjšamo S-krat tudi zakasnitev skozi elemente, disipacija moči je zmanjšana z faktorjem S2. Vkolikor skaliramo tudi povezave, se jim upornost poveča - ob zmanjšanju parazitskih kapacitivnosti se RC konstanta ohranja.

V praksi povprečna dolžina povezave po skaliranju ni krajša, saj je namen skaliranja gradnja večjih (in hitrejših) vezij. Torej bo v prihodnje poleg zelo dobrega modeliranja elementov v submikronskem območju velik poudarek na tehnikah za zmanjšanje RC konstant.

So povsem fizikalne omejitve, zaradi katerih je spodnja meja uporabne minimalne dimenzije tranzistorjev, kot jih danes poznamo, pri pol mikrona.

- Napajanja v nedogled ne moremo nižati, saj je pragovna napetost tranzistorja podana z razliko energijskih pasov p in n dopiranega silicija.
- Prebojne napetosti so določene s kritično poljsko jakostjo in debelino izolacijskih plasti.
- Spodnja meja tankosti povezav je določena z začetkom migracije prevodne kovine kot posledico maksimalne tokovne gostote.

Skaliranje je omejeno tudi s stališča uporabnika. Danes je industrijski standard pet voltno napajanje in TTL logični nivoji. Prehod navzdol ekonomsko se ni upravičen, saj potegne za sabo poleg kopice formalnih problemov tudi novo poglavje o šumu in motnjah. Kljub visokim vlaganjem - za skaliranje pod dva mikrona je v precej tehnoloških korakih potrebna drugačna oprema zaradi tanjših in bolje definiranih plasti - tudi razvita Evropa sledi Japoncem in Američanom, saj je bila prva lekcija o tehnološkem zaostajanju na področju elektronike dovolj poučna.

2.1.2 Izdelava Smart Power vezij

Ogromno je aplikacij zlasti v industrijski elektroniki, avtomatizaciji, avtomobilski elektroniki, široki potrošnji, kjer na osnovi logičnih odločitev izvedemo določene močnejše preklope. Ustaljena realizacija je tiskano vezje z elementi ene logičnih družin s podatkovnim drzalom, ki krmili diskretne tranzistorje, sledijo pa lahko še releji. Velikost, temperatura, poraba, zanesljivost so dejanski problemi. Smart Power vezja, kjer je na istem čipu logični in močnostni del, nam na stopnji svoje tehnološke zrelosti te probleme odpravijo. Pobudo za taka vezja je dal uporabniški del mikroelektronike sistemskih hiš, odgovor je moral priti s strani tehnologije. Zato so začetki vidnih poskusov v letu 1983, večji razmah takih vezij pa je šele letos. Glavni tehnološki problem je ustrezna izolacija med močnostnimi stikali in logičnim delom, majhna upornost, tokovna zmogljivost, visoka prebojna napetost stikal.

Izolacijo dosegajo z zaporno polariziranimi p-n spoji, s posebnimi izolacijskimi difuzijami. Trenutno najboljša in najdražja je izolacija z dielektriki, vnesenimi v silicijevo rezino, na katerih oblikujemo močnostne elemente. Tovrstna izolacija prenese napetosti čez tisoč voltov.

Majhno upornost stikal dosegajo z več manjših stikal povezanih vzporedno. Druga bolj ravna pot je striktno manjšanje upornosti kanala, vira, ponora, dovodov toka. Ustrezni tehnološki koraki pa istočasno nižajo prebojno napetost med virom in ponorom, tako da v tej smeri ni rešitve. Trenutno znajo v svetu narediti močnostne tranzistorje z do tri miljone vzporedno vezanih stikalnih celic na kvadratno inčo. Obenem z manjšanjem upornosti pridobivamo na tokovni zmogljivosti stikal.

Visoko prebojno napetost dosežemo z razmaknitvijo vira in ponora tranzistorja. Izdelamo strukturo npn-n. Vrata so p področje, n- v primeru pozitivno polariziranega ponora služi kot izolator. Tako razlika napetosti vira in ponora ni omejena le na ozko območje p, ki predstavlja dejanski tokovni ventil.

Današnja tipska področja uporabe Smart Power vezij so krmiljenje motorjev (GESmart), avtomobilska industrija (SGS, Motorola, Mostek, International Rectyfier, Rockwell), krmilniki prikazalnikov (SGS, Supertex, Siliconix, Telmos, Sprague, TI, Mostek), napajalniki (Integrated Power Semiconductors, International Rectifier), krmiljenje koračnih motorjev (Integrated Power Semiconductors, Unitrode).

2.1.3 BiMOS vezja

so v končni obliki hitrejša in manjša od bipolarnih in MOS vezij posebej. Združujejo prednosti MOS - nizka disipacija moči, manjša občutljivost na šum, in prednosti bipolarnih tranzistorjev - preklopna hitrost, bolje definirano obnašanje v linearnem območju, boljši tokovni viri. Da se ta vezja niso

pojavila že prej, so krivi postopki izdelave. Treba je namreč postaviti proces izdelave vezja, ki istočasno zagotavlja dobre bipolarne in MOS tranzistorje. Letos tak proces zahteva med sestnajst in dvajset mask. Potencialna uporaba teh vezij je DRAM (hitrost kombinirana s čim nižjo porabo), vezja za prenos in obdelavo podatkov.

2.1.4 Vezja na galijevem arzenidu

V razvoj tovrstnih vezij se investira iz več razlogov:

- Ta vezja so do pet krat hitrejša od ECL vezij na siliciju. Gibljivost elektronov v GaAs je okrog devet krat večja od gibljivosti elektronov v siliciju.
- Mnogo laže je doseči visoko stopnjo integracije, saj je nedopiran GaAs izolator in s tem prihranimo precej procesnih korakov, ki so pri siliciju potrebni zaradi izolacije med aktivnimi deli vezja.
- Vezja na GaAs manj sumijo od vezij na siliciju.
- Toplotna prevodnost GaAs je večja od le te silicija (hlajenje).
- V temperaturnem območju -200, 200 stopinj celzija se električne lastnosti gradnikov vezij na GaAs spreminjajo znosno in lahko z dobrim načrtovanjem definiramo tako področje uporabe.
- Glavni problem danes pa so dragi substrati. So tudi krhki in imajo mnogo napak. Danes je cena GaAs rezine tridesetkrat večja od cene silicijeve rezine. Če k temu prištejemo še manjši izplen zaradi večje gostote napak, pa del razvojnih stroškov, je cena vezja na GaAs hitro dva velikostna razreda višja od cene vezja na siliciju z enako logično shemo. Največ v pridobivanje substratov investirajo Japonci, tako da lahko upravičeno pričakujemo, da bodo japonski proizvajalci vezij na GaAs osvojili večji delež trzisca. Danes je največje torišče delovanja na GaAs področje nizkošumnih ojačevalnikov, ki jih zahtevajo satelitske in vojaške komunikacije. Tako visokih frekvenc in tako nizkega nivoja suma na siliciju ne moremo doseči. Tudi logične mreže na GaAs so že razvite. Letos so poprečne zakasnitve na vratih 30 ps, poraba je 200 mW. Proizvajalci so Harris, Honeywell, TI, Ford Microelectronics, NT&T, Toshiba. Stroški Honeywellove logične mreže z dva tisoč vrati in šestinpetdeset I/O celicami so sto desettisoč dolarjev.

2.2 Smernice razvoja načrtovanja

V zadnjih nekaj letih se je programska oprema za načrtovanje integriranih vezij neverjetno razvila. Starejši programi na miniračunalnikih so postali zastareli, saj potrebujejo preveč računalniškega časa na teh relativno dragih procesorjih.

V industriji integriranih vezij je prevladalo mnenje, da je pisanje lastne programske opreme za načrtovanje integriranih vezij zelo zahtevno in neekonomično. To si lahko privoščijo le največje družbe. Večina manjših in srednjih proizvajalcev programsko opremo kupuje pri za to področje specializiranih firmah. Njihovi lastni strokovnjaki se ukvarjajo samo še z združevanjem različnih paketov v enoten sistem.

Večina te programske opreme sedaj deluje na delovnih postajah, ki jih je moč povezati z mrežo Ethernet. Te delovne postaje so opremljene s posebnimi pospeševalniki - specializirano opremo namenjeno enemu samemu postopku. Ti pospeševalniki so prišli najbolj do veljave pri logični simulaciji in simulaciji napak, kjer le to sedaj opravlja v veliki meri strojna oprema sama, kar je seveda hitrejše kot programi z najboljšimi algoritmi. Razlika v hitrosti delovanja je med 10 in 100 krat.

Za izdelavo zelo obsežnih integriranih vezij ne pride več v postev klasično načrtovanje vezij po naročilu, saj bi tako načrtovanje vezij z 100.000 do 1.000.000 vrati trajalo nekaj let, kar seveda ni sprejemljivo. Trenutno so na vrhuncu programi za avtomatično nameščanje in povezovanje standardnih celic in logičnih mrež, vedno bolj pa se uveljavljajo prevajalniki na silicij.

Končni cilj načrtovanja je čim hitrejša izdelava vezja, ki bo delovalo že v prvem poskusu. Torej je treba povsem izključiti predvidljive napake. Le te lahko nastopajo v interakciji med posameznimi programi, lahko so napake v programih samih, lahko so posledica človeškega faktorja. Prvo in drugo možnost s skrbnim preverjanjem programske opreme lahko izključimo, tretjo pa zmanjšamo tako, da čimvec načrtovalskih postopkov avtomatiziramo.

Pri načrtovanju integriranih vezij z prevajalniki na silicij načrtovalec vnese logično shemo, željene signalne zahteve, oznako procesa, v katerem bo vezje izdelano in s tem posredno izbere za ta proces specifična načrtovalska pravila. Izhod programskega paketa je končna geometrija vezja, po kateri izdelajo maske. Človeški faktor je omejen le še na napačno zastavljen problem.

Gotovo bo šel razvoj na področju programske opreme v prihodnjih letih v smer izdelave čim popolnejših prevajalnikov na siliciju. Danes znajo najboljši prevajalniki načrtati bit slice procesor. Razvojno zahtevna vezja pa bodo gotovo tudi v prihodnje delali precej ročno, saj je znanje treba najprej osvojiti, da ga lahko vgradis v samostojen načrtovalski sistem. Gotovo bodo avtomatizmi na področju načrtovanja vezja lahko pocenili, razvojni čas bo krajši in načrtovalskih napak bo manj. Tudi razvoj danes zahtevnih vezij bo lahko hitrejši in cenejši.

3 Oris načrtovalskega postopka od zasnovane logične sheme do narejenih mask v DO Mikroelektronika

Načrtovanje integriranih vezij je zelo zahteven postopek. Integrirano vezje lahko sestavlja nekaj sto do nekaj sto tisoč medsebojno povezanih aktivnih in pasivnih elementov. Podatkovne baze za opis takega vezja so zato velike, kljub temu pa ne smejo vsebovati napak. V naslednjih nekaj vrsticah bi rada pokazala možnosti in perspektive načrtovanja integriranih vezij v Iskri.

Zaradi obsežnosti podatkov, ki opisujejo integrirano vezje, je možnost vnosa napake pri načrtovanju zelo velika. Vsaka napaka ponavadi pomeni, da vezje ne bo delovalo tako, kot je bilo zamišljeno, zato se vedno bolj uveljavljajo postopki, ki avtomatizirajo posamezne postopke v procesu načrtovanja.

Integrirana vezja po naročilu delimo glede na metodo načrtovanja na tri skupine:

- naročniška vezja
- vezja iz standardnih celic
- logične mreže

V Iskri DO Mikroelektronika imamo precej zmogljiv sistem za računalniško podprto načrtovanje integriranih vezij, ki omogoča načrtovanje vseh treh zvrsti. Postopki za načrtovanje se od skupine do skupine razlikujejo. Razlikuje se tudi potrebni čas za razvoj takega vezja, s tem pa tudi stroški razvoja. Ti direktno vplivajo na končno ceno vezja. Drug pomemben faktor, ki vpliva na

končno ceno, pa je površina vezja na siliciju. Stroški razvoja so obratno sorazmerni površini vezja na siliciju. Najdražji in najbolj dolgotrajen je razvoj naročniških vezij, ki da najmanjšo površino na siliciju. Najcenejši je razvoj logičnih mrež, pri katerih je za končno konfiguracijo potrebno izdelati eno samo masko. Seštevek obeh stroškov pokaže, da se izplača izdelovati glede na število kosov:

- do 20.000 vezja na osnovi logičnih mrež
- od 20.000 do 100.000 vezja na osnovi standardnih celic
- nad 100.000 naročniška vezja.

Nekateri postopki so pri vseh treh metodah načrtovanja enaki, drugi pa se razlikujejo. Zajem shematskega opisa vezja in logična simulacija sta n.pr. enotna medtem ko je izdelava geometrije različna. V nadaljevanju bova najprej opisala programsko opremo, ki jo imamo v Mikroelektroniki, nakar bova podrobneje obdelala vrstni red postopkov pri vseh treh metodah načrtovanja.

Težko je oceniti čas, ki je potreben za izdelavo logičnega oz. funkcionalnega opisa vezja. Ta je odvisen predvsem od izkušenosti načrtovalca in kompleksnosti vezja. Mnogo laže je oceniti čas, ki je potreben od trenutka, ko je opis vezja gotov, do končne realizacije.

3.1 Pregled programske opreme

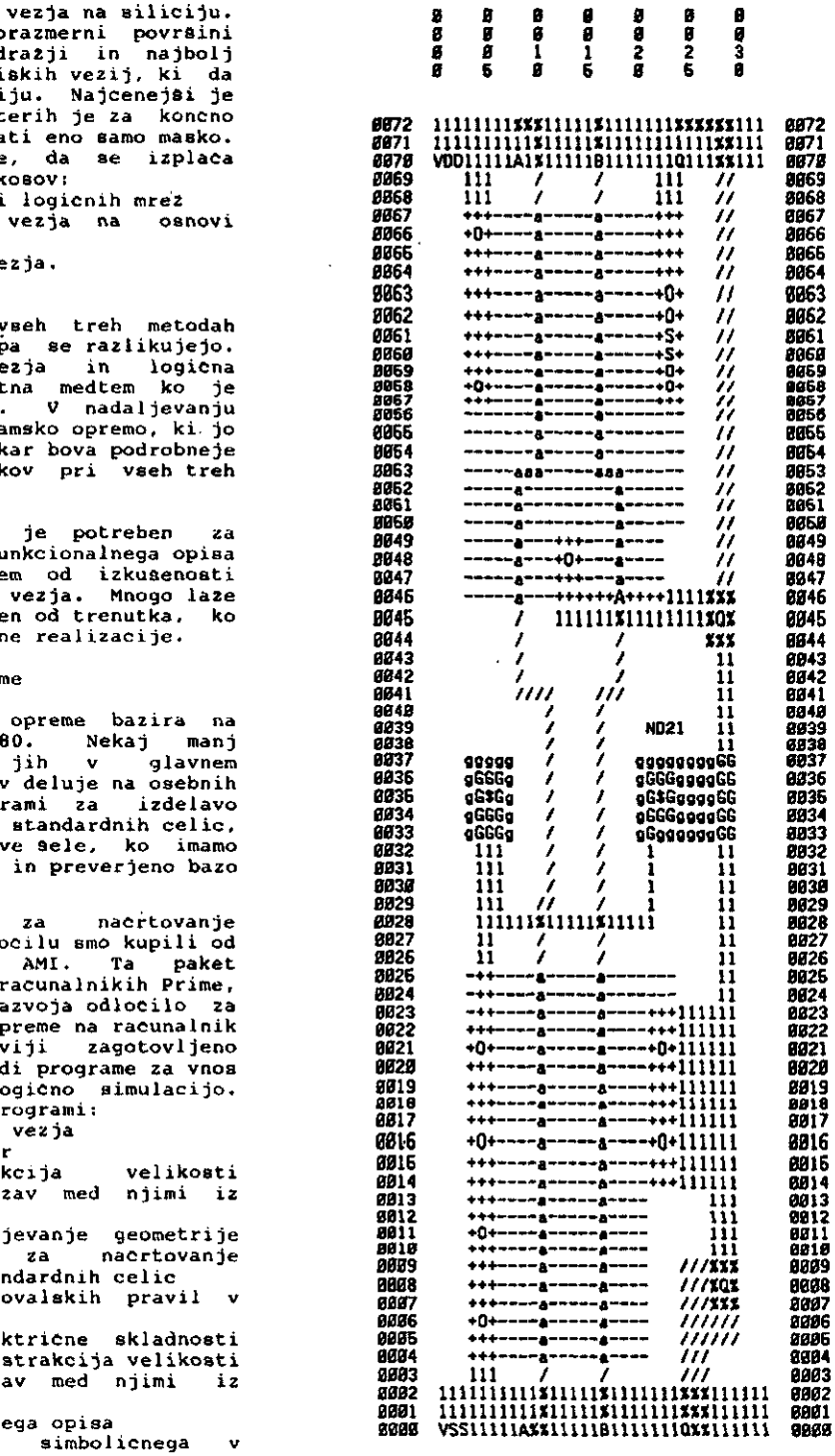
Večina obstoječe programske opreme bazira na miniračunalniku VAX 11/780. Nekaj manj obsežnih programov, ki jih v glavnem uporabljamo za vnos podatkov deluje na osebnih računalnikih. To so programi za izdelavo integriranih vezij z metodo standardnih celic, ki pa pridejo do prave veljave šele, ko imamo na miniračunalniku obsežno in preverjeno bazo podatkov.

Programski paket SIDS za načrtovanje integriranih vezij po naročilu smo kupili od licenčnega partnerja Gould AMI. Ta paket programov je deloval na računalnikih Prime, zato se je tedanje vodstvo razvoja odločilo za konverzijo te programske opreme na računalnik VAX, ki ima v Jugoslaviji zagotovljeno vzdrževanje. Kupili smo tudi programe za vnos logičnega opisa vezja in logično simulacijo. Kupljeni so bili naslednji programi:

- BOLT - prevajalnik opisa vezja
- SIMAD - logični simulator
- HOLDNET - ekstrakcija velikosti tranzistorjev in povezav med njimi iz logičnega opisa
- SIDSED - simbolično urejevanje geometrije vezja uporabljamo za načrtovanje naročniških vezij in standardnih celic
- SDRC - preverjanje načrtovalskih pravil v simbolični bazi podatkov
- STRACE - preverjanje električne skladnosti simboličnega opisa in ekstrakcija velikosti tranzistorjev in povezav med njimi iz simboličnega opisa
- SPRINT - izpis simboličnega opisa
- STP - pretvarjanje iz simboličnega v geometrični opis
- GPLOT - izris geometrične baze podatkov
- COMPARE - primerjava velikosti tranzistorjev in povezav med njimi, ki jih dobimo iz logičnega in iz simboličnega opisa.
- SPICE - analogni simulator

Kritični elementi v tem paketu so:

- SIMAD, ki je za zelo obsežna vezja zelo počasen. Simulacija poteka tudi po nekaj dni
- STRACE je za zelo obsežna vezja zelo počasen



Slika 1

Izris simbolične baze podatkov za dvovhodna vrata NAND na elektrostatičnem risalniku.

Nekaj osnovnih znakov:

"a"- vrata tranzistorja

"-"- difuzija

"1"- metal

"/"- polisilicij

- SDRC, ki je za obsežna vezja zelo počasen. Preverjanje načrtovalskih pravil poteka tudi po nekaj dni. Ni ga moč uporabiti za zelo obsežna vezja
- STP, ki je za obsežna vezja zelo počasen. Pretvarjanje simboličnega opisa v geometrični opis poteka tudi po nekaj dni. Ni ga moč uporabiti za zelo obsežna vezja
- COMPARE, ki je za zelo obsežna vezja zelo počasen. Simulacija poteka tudi po nekaj dni.

Problem smo reševali z uvedbo metode načrtovanja s standardnimi celicami, pri kateri podatke zajemamo in logično simuliramo na osebnih računalnikih s programskim paketom SCEPTRE. Pomanjkljivost tega paketa je, da je njegov izhod komandna procedura za program SIDSED, ki zloži celotno vezje v simbolični obliki. Celice so načrtane tako, da ne moremo kršiti načrtovalskih pravil pri njihovem nameščanju in povezovanju, zato preverjanje načrtovalskih pravil na simboličnem opisu odpade. Compare prav tako odpade, ker ga do neke mere nadomesti programska oprema SCEPTRE. Problem predstavlja pretvorba iz simboličnega v geometrični opis, zato smo sami napisali program, s katerim je moč komandno proceduro, ki je rezultat programskega paketa SCEPTRE, direktno pretvoriti v geometrično obliko. Ta postopek je na ta način stokrat hitrejši.

Pri AMI-ju smo kupili tudi paket CIPAR, ki omogoča avtomatično nameščanje in povezovanje standardnih celic. Program je dokaj neučinkovit. Izkoriščenost površine na siliciju se precej izboljša z uporabo optimizatorja Timberwolf 1.0 z Berkeleyske univerze, še bolj pa bi se z uporabo procesa z dvema plastema metala.

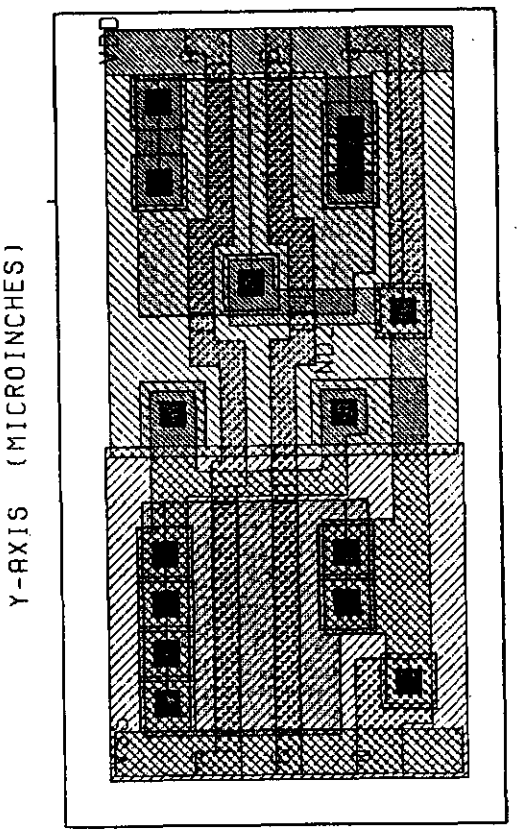
Shematski vnos in logično simulacijo iz paketa SCEPTRE uporabljamo tudi za načrtovanje vezij po metodi logičnih mrež. Sami smo morali napisati program za grafično urejanje geometrije vezja.

Končni produkt postopka načrtovanja so datoteke, ki jih zapišemo na trak, s katerim krmilimo generator vzorcev - napravo, s katero se izdelajo maske in vzorci za avtomatsko testno opremo. Program za pretvorbo geometrijske baze podatkov v datoteke za generator vzorcev (PG) smo napisali sami.

3.2 Načrtovanje z logičnimi mrežami

Kot sva že omenila, je to metoda, s katero najhitreje pridemo do integriranega vezja. Samo načrtovanje je zelo podobno načrtovanju z metodo standardnih celic, bistvena razlika pa je končni produkt procesa načrtovanja. Pri metodi logičnih mrež dobimo eno masko, medtem ko pri metodi načrtovanja s standardnimi celicami dobimo toliko mask, kot jih proces zahteva (8 - 9 mask). Ko je maska pripravljena, je pot do končnega vezja zelo kratka. Logične mreže se izdelujejo na že pripravljenih rezinah, ki so izdelane do določene stopnje, tako da jih je moč dokončati v enem tednu.

Načrtovanje se začne z vnosom logičnega opisa v računalnik. Pri tem imamo dve možnosti. Logiko lahko z urejevalnikom teksta vnesemo na alfanumeričnem terminalu računalnika VAX ali pa na osebnem računalniku Sirius ali IBM PC/XT. Vnašanje logike je lažje na osebnem računalniku, kjer je v paketu programov SCEPTRE grafični urejevalnik, ki omogoča shematski vnos. Pri vnosu logike je treba zelo paziti, da je možno vezje na zunanjih sponkah zadovoljivo testirati. Pri intergiranih vezjih namreč ni možno merjenje signalov med



Slika 2

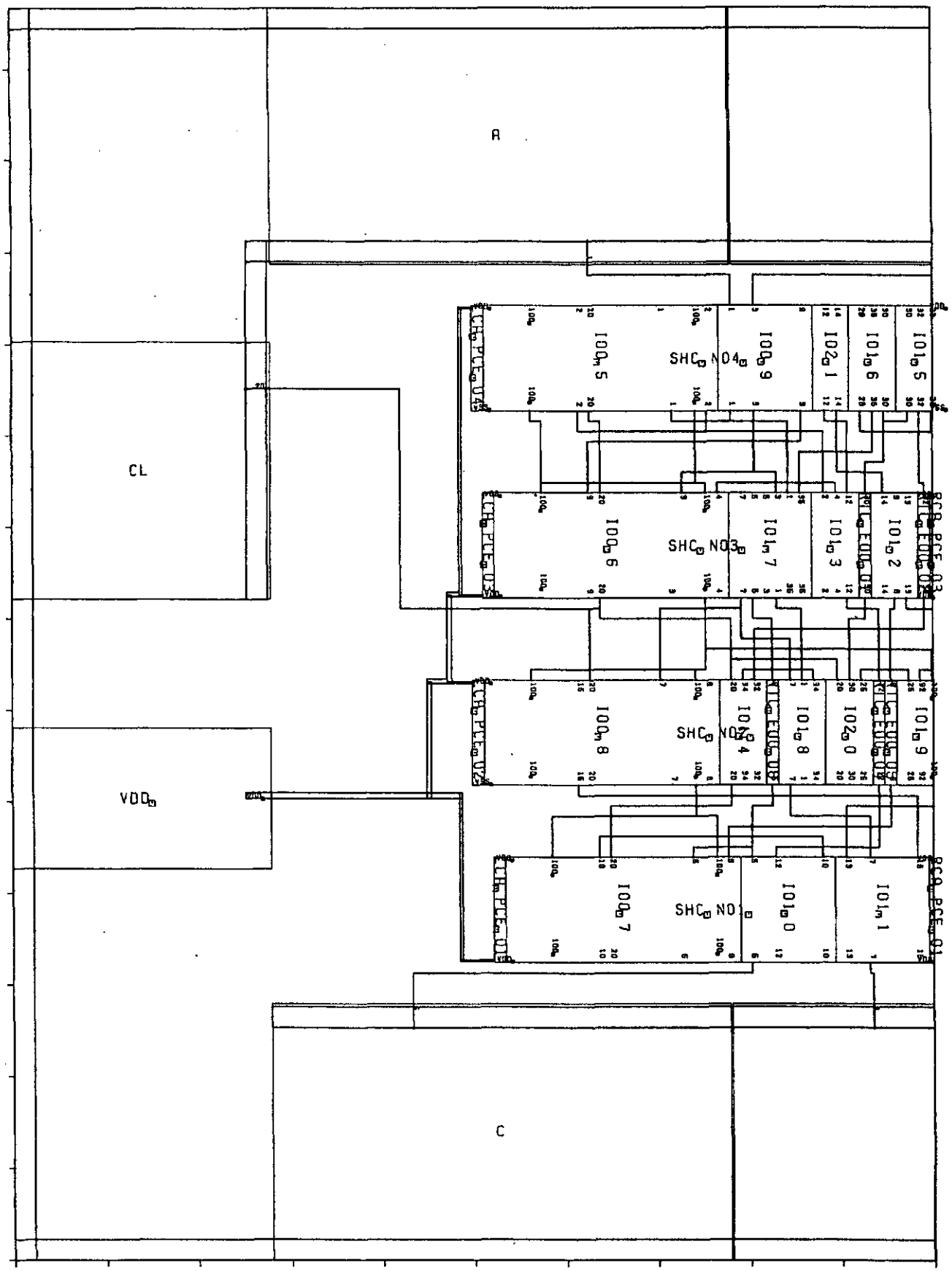
Izris geometrične baze podatkov za dvovhodna vrata NAND

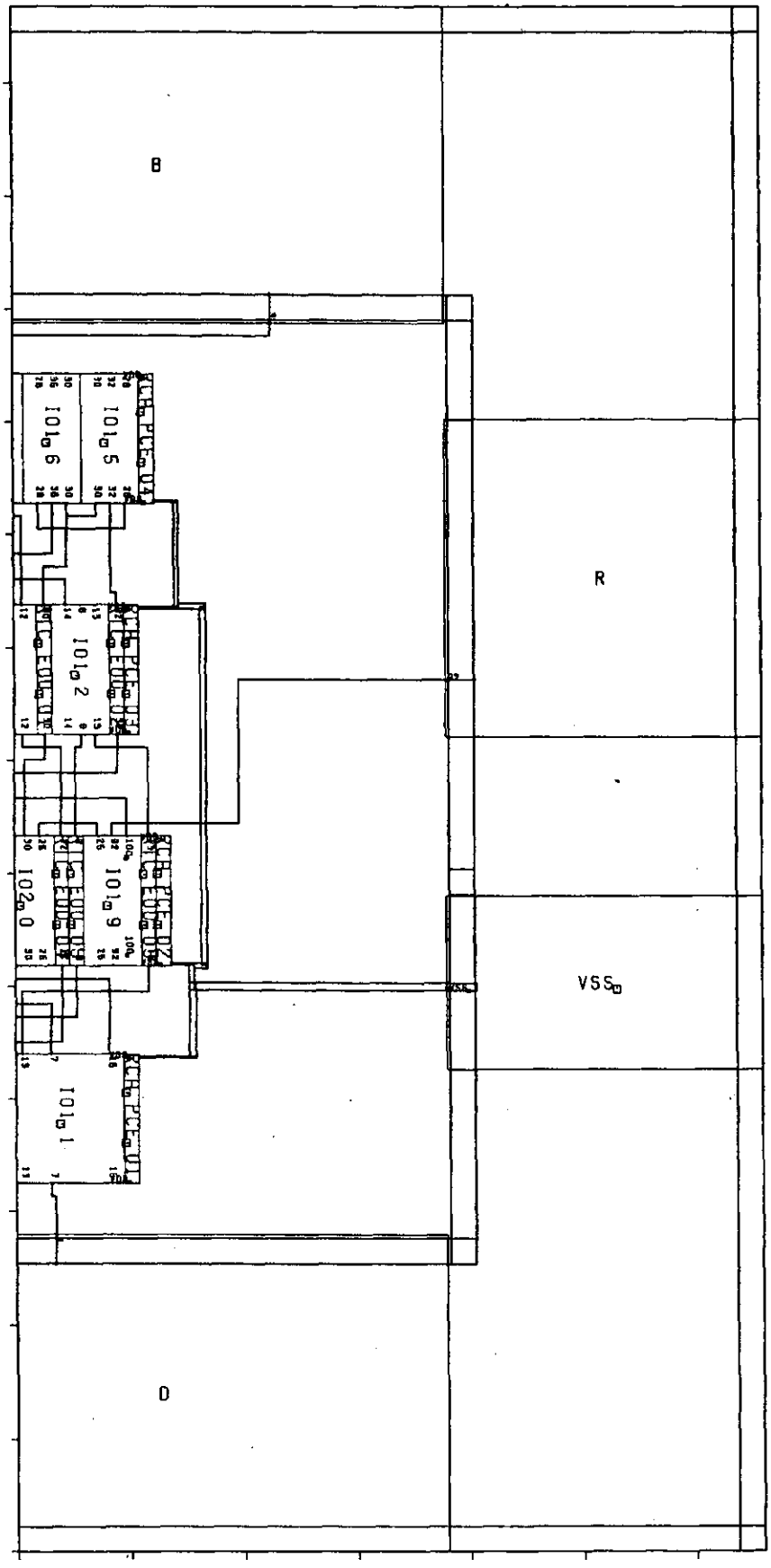
posameznimi elementi vezja.

Ko je logika vnesena, jo preizkusimo s programom za logično simulacijo. Če izhodni vzorci, ki jih dobimo s simulacijo, ne ustrezajo zamišljenim, z grafičnim urejevalnikom popravimo shemo in ponovno simuliramo. Postopek ponavljamo toliko časa, da smo zadovoljni z odzivi vezja, ki jih pokaže simulacija. Logično shemo lahko izrišemo na matričnem tiskalniku ali pa na risalniku, ki omogoča izdelavo dokumetacije vezja. Pravilnost logičnega opisa je seveda predpogoj za pravilno delovanje vezja. Logičen opis zatem prenesemo na VAX, kjer vezje ponovno simuliramo s programom SIMAD. Ta simulacija je samo dodatna kontrola. Izhodne datoteke, ki jih dobimo, pa je možno z drugim programom pretvoriti v testne vzorce za avtomatsko testno napravo.

Vnos logike in izdelava potrebne dokumentacije vzame načrtovalcu en do dva tedna. Tu prevzame delo načrtovalec sestavnice mask ali pa ga nadaljuje načrtovalec sam.

Ko je simulacija gotova, pristopimo k zadnjemu interaktivnemu postopku. S posebnim programom začnemo sestavljati geometrijo. Na predefinirani mreži postavljamo in povezujemo celice. Program je uporabniško prijazen, zato je delo hitro in učinkovito. Vse spremembe, ki jih vnesemo s pomočjo tipkovnice in miške, se sproti pojavljajo na zaslonu, na katerem lahko prikažemo manjši del vezja in s tem večje podrobnosti ali celotno vezje zaradi boljšega globalnega pregleda. Tudi geometrijo lahko izrišemo na matričnem tiskalniku. Za to delo


A
CL
VDD
C



Slika 3
Primer vezja iz standardnih celic, ki ga je zložil program za avtomatično nameščanje in povezovanje celic. Vezje je razdeljeno na periferijo, ki jo sestavljajo vhodno izhodne celice (A, B, C, D, CL, R, VSS, VDD) in jedro. Z jedrnimi celicami realiziramo željene logične ali analogne funkcije, ki jih uporabljamo preko vhodno izhodnih celic, ki hkrati varujejo vezje pred elektrostatskimi razelektritvami.

potrebujemo enega cloveka nekaj dni.

Ko je postavljanje in povezovanje celic gotovo, drug načrtovalec preveri skladnost med logično shemo in geometrijo, napake se popravijo. Podatke s posebnim programom nato prenesemo v obliki tekstovne datoteke po terminalski liniji na VAX, kjer drug program celice nadomesti z njihovo vsebino, povezave pa zamenja z ustreznimi pravokotniki. Do končnega rezultata, t.j. maske za generator vzorcev, moramo izvesti se nekaj avtomatičnih postopkov. Dobljeni geometrijski bazi podatkov pripišemo podatke, ki so na metalnem nivoju fikani (napajalne linije, ...), vezje za dokumentacijo izrišemo na elektrostatičnem risalniku in nazadnje pretvorimo geometrično detoteko v datoteke za generator vzorcev.

Izurjen načrtovalec potrebuje za celoten proces načrtovanja dva do tri tedne. Izdelava mask traja en dan, vendar je treba zaradi zasedenosti opreme včasih čakati tudi po dva do štiri tedne. Rezine z logičnimi mrežami je treba še dvakrat maskirati. Od vezja do vezja se spreminja samo metalna maska. Izdelati je treba metalne povezave in narediti odprtine v pasivaciji na mestih, kjer se pritrdijo žičke, ki silicijevo ploščico povezujejo z nožicami na ohišju. Procesiranje poteka nekaj dni. Z dobro časovno usklajenostjo posameznih postopkov je tako pri logičnih mrežah možno priti od logične sheme do vezja v enem mesecu.

3.3 Načrtovanje s standardnimi celicami

Z metodo standardnih celic dosežemo boljši izkoristek površine na siliciju. Tu je postopek načrtovanja podoben opisanemu postopku načrtovanja po metodi logičnih mrež. Rezultat načrtovanja so maske za vse nivoje. Pri naših standardnih osem in devet nivojskih procesih traja procesiranje približno en mesec.

Metoda standardnih celic omogoča izdelavo logičnih in analognih integriranih vezij. Pri analognih vezjih je pogosto potrebno načrtati novo ali prirediti obstoječo celico. Načrtovalec mora zato znati uporabljati analogni simulator Spice.

Načrtovanje se začne z vnosom logičnega oz. funkcionalnega opisa vezja v računalnik. Ta je povsem enak kot pri logičnih mrežah. Tudi tu se po vnosu podatkov menjavajo simulacije in popravki.

Preverjen logičen opis zatem prenesemo na VAX, kjer vezje ponovno simuliramo s programom SIMAD. Ta simulacija je samo dodatna kontrola, izhodne datoteke, ki jih dobimo, pa je možno z drugim programom pretvoriti v testne vzorce za avtomatsko testno napravo.

Vezja iz standardnih celic so včasih kompleksnejša kot logične mreže, ki so omejene z največjim modelom, ki ga izdelujemo (t.j. 1200 ekvivalentnih vrat). Vnos logike in izdelava potrebne dokumentacije vzame načrtovalcu en do tri tedne.

Ko smo z simulacijo logičnega oz. funkcionalnega opisa zadovoljni, je treba namestiti in povezati celice. Za to imamo dve metodi. Prva je ročno nameščanje in povezovanje na osebnem računalniku, druga pa je avtomatično namescanje in povezovanje na miniračunalniku.

Za avtomatičen postopek uporabljamo program Cipar iz paketa programov Cipar. S pomočjo optimizatorja Timberwolf se izboljša izkoriščenost površine silicija. Vhodni podatki za program so obstoječe datoteke z logičnim oz. funkcionalnim opisom vezja. Program sam postavi in poveže celice integriranega vezja. Obdeluje jih kot pravokotnike, ki jih postavi paroma v vrste in jih poveže v kanalih med vrstami. Dodatni programi iz paketa Cipar omogočijo izris geometrije na elektrostatičnem risalniku in pretvorbo geometrije v pravo geometrično bazo podatkov, med katero program nadomesti pravokotnike z njihovo vsebino, povezave pa s pravokotniki. Izdelava geometrije vzame en do dva dni.

Boljšo izkoriščenost površine silicija dosežemo z grafičnim urejevalnikom iz paketa programov SCEPTRE na osebnem računalniku. Razlika v površini integriranega vezja je med 10% in 30%. Program dovoljuje nameščanje celic in povezav na mrežo, katere vozlišča so dovolj oddaljena drugo od drugega, da ni mogoče kršiti načrtovalskih pravil. SCEPTRE omogoča tudi primerjavo med izdelano geometrijo in logičnim opisom.

Ko je doseženo ujemanje med geometrijo in logičnim opisom, podatke s posebnim programom prenesemo v obliki tekstovne datoteke po terminalski liniji na VAX, kjer drug program celice nadomesti z njihovo vsebino, povezave pa zamenja z ustreznimi pravokotniki. Nadaljni postopek za izdelavo trakov za generator vzorcev je enak kot pri logičnih mrežah.

Izurjen načrtovalec potrebuje za celoten proces vnosa logičnega oz. funkcionalnega opisa en do tri tedne. Načrtovalec geomerije mask potrebuje za ročno nameščanje in povezovanje celic en do dva tedna. Izdelava mask traja nekaj dni. Rezine z vezji načrtanimi po metodi standardnih celic morajo skozi celoten proces, kar zaradi množice različnih postopkov traja približno en mesec. Z dobro časovno usklajenostjo posameznih postopkov je tako pri standardnih celicah možno priti od logične sheme do vezja v dveh do treh mesecih.

3.4 Načrtovanje naročniških vezij

Načrtovanje naročniških (full custom) vezij je najdolgotrajnejši postopek. S to metodo dosežemo najboljši izkoristek, kar se pri zelo velikih serijah seveda izplača. Rezultat načrtovanja so maske za vse nivoje. Pri naših standardnih osem in devet nivojskih procesih traja procesiranje približno en mesec.

Ta metoda ne pride v naših razmerah velikokrat v poštev, saj so serije vezij, ki jih pri nas naročajo razna podjetja ponavadi premajhne, da bi cena posameznega vezja prenesla razvojne stroške. Naročnik mora potrebovati vsaj 100.000 vezij, da se mu to izplača. Tudi ta metoda omogoča izdelavo logičnih in analognih vezij.

Načrtovanje se začne z vnosom logičnega oz. funkcionalnega opisa vezja v računalnik. Ta postopek je povsem enak kot pri logičnih mrežah in standardnih celicah. Tudi tu se po vnosu podatkov menjavajo simulacije in popravki.

Preverjen logičen opis zatem prenesemo na VAX, kjer vezje ponovno simuliramo s programom SIMAD. Ta simulacija je samo dodatna kontrola, izhodne datoteke, ki jih dobimo, pa je možno z drugim programom pretvoriti v testne vzorce za avtomatsko testno napravo.

Kompleksnost naročniških vezij je podobna kot je kompleksnost vezij s standardnimi celicami. Vnos logike in izdelava potrebne dokumentacije vzame načrtovalcu en do tri tedne.

Pri naročniških vezjih se tu delo šele začne. Sedaj je treba posamezne logične oz. funkcionalne bloke izdelati tako, da zavzamejo

čim manj površine na siliciju. Vsako celico je potrebno električno simulirati in optimizirati za dano mesto v vezju. To delo je dolgotrajen postopek, ki zahteva izkusenega načrtovalca.

Ko je geometrija posameznih blokov narejena, jih je treba namestiti čimbolj skupaj in povezati. Med nameščanjem in povezovanjem je včasih potrebno katerega od blokov spremeniti. Geometrijo načrtujemo na semigrafičnih barvnih terminalih s pomočjo programa Sidsed. Ko je geometrija gotova s programoma Sdrc in Strace preverimo, če je vezje načrtano po načrtovalskih pravilih in električno skladnost simboličnega opisa. Strace nam izloči tudi velikosti tranzistorjev in povezav med njimi iz simboličnega opisa. S programom Compare nazadnje primerjamo velikosti tranzistorjev in povezav med njimi, ki jih dobimo iz logičnega in iz simboličnega opisa. Vse to preverjanje zagotavlja, da načrtano vezje nima napak. Zadnji program iz paketa SIDS Stp nam pretvori simbolični opis v mnogokotnike in jih zapiše v geometrično bazo podatkov. To je moč izrisati na elektrostatični risalnik.

Izurjen načrtovalec potrebuje za celoten proces vnosa logičnga oz. funkcionalnega opisa en do tri tedne. Skupaj z načrtovalcem geomerije mask potrebuje izdelavo blokov in za ročno nameščanje in povezovanje le teh nekaj mesecev. Ostali postopki trajajo toliko časa kot pri metodi standardnih celic, kar da skupaj več kot tri mesece.

Viri:

1. Power MOSFETs: Power for the 80s, Duncan Grant, Allan Tregidga, Solid State Technology, Nov. 1985

2. Mixed process chips are about to hit the big time, Bernard Conrad Cole, Electronics, March 3, 1986

3. 1. šola za načrtovanje mikroelektronskih vezij v Iskri; interno gradivo , Nov. 1985

4. Stretching the limits of software, Bernard Conrad Cole, Electronics, June 23, 1986

POVEZOVALNE MREŽE VEČPROCESORSKIH SISTEMOV

INFORMATICA 4/86

UDK: 681.324

Slavko Mavrič, Branko Mihailović, Peter Kolbezen
Institut Jožef Stefan, Ljubljana

Članek predstavlja funkcijo povezovalne mreže v večprocesorskem sistemu in spregovori o njenih bistvenih lastnostih. Opisane so pomembnejše topologije enostopenjskih in večstopenjskih povezovalnih mrež in narejena kvalitativna primerjava med njimi. Na koncu je podan primer sistema na osnovi skupnega vodila.

In the article the function and the basic properties of the interconnection network in a multiprocessor system are presented. We describe the principal topologies of singelstage and multistage interconnection networks. Also, the qualitative comparition between them is made. Finally, we present an example of global bus based system.

1. UVOD

Sodobna elektronska tehnologija omogoča gradnjo paralelnih računalniških sistemov, ki jih sestavlja na stotine ali celo na tisočine procesorjev. Eden izmed največjih problemov pri gradnji takih paralelnih sistemov je v izbiri prave povezovalne mreže za učinkovito medsebojno povezavo procesorjev, pomnilniških modulov in drugih naprav. Optimalna izbira povezovalne mreže za nek sistem zavisi predvsem od namembnosti sistema (aplikacije), velikosti sistema, zahtevane hitrosti, cenovnih omejitev... Z gotovostjo lahko rečemo, da šele prava povezovalna mreža omogoči pričakovano učinkovitost celotnega sistema.

2. VLOGA POVEZOVALNE MREŽE

V logičnem smislu predstavlja povezovalna mreža zaključeno enoto z M vhodi in N izhodi ter lastno preslikovalno funkcijo, ki opisuje povezovalne lastnosti mreže. Statično preslikovalno funkcijo najlaže podamo v obliki matrike z M vrsticami in N stolpci. Element matrike l_{ij} je enak 1, če mreža omogoča povezavo med vhodom i in izhodom j, sicer pa je enak 0. Statična preslikovalna funkcija govori o povezovalnih lastnostih "prazne" povezovalne mreže, kar pomeni, da v trenutku, ko ocenjujemo možnost povezave i,j še ne obstaja nobena povezava v mreži.

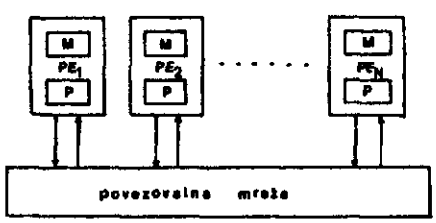
Naloga povezovalne mreže večprocesorskega sistema je ta, da omogoča povezavo med množico procesorjev in množico pomnilniških modulov sistema. Pri tem obstajata dva osnovna pristopa.

Pri prvem pristopu tvori vsak procesor skupaj s pripadajočim lokalnim pomnilnikom procesni element (PE). Ti procesni elementi so medsebojno povezani prek povezovalne mreže tako, da je vsak procesni element priključen na vhodno in izhodno linijo povezovalne mreže. Imenujmo tak način povezave PE-na-PE pristop (Slika 1).

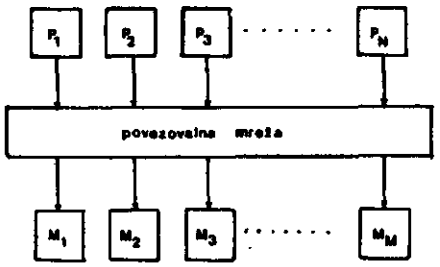
Drug pristop postavlja povezovalno mrežo med množico procesorjev in množico pomnilniških enot. Imenujmo ga P-na-M pristop (Slika 2).

Glavna prednost prvega pristopa je v hitrem dostopu do lokalnega pomnilnika, medtem ko je prednost drugega pristopa v tem, da si procesorji delijo pristop do obsežnih podatkovnih blokov in da je obseg pomnilnika, ki pripada *posameznemu procesorju* spremenljiv.

Možna je tudi kombinacija obeh pristopov, pri kateri se procesni elementi povezujejo preko mreže med seboj, kot tudi z globalnimi pomnilniškimi enotami (PE-na-M pristop).



Slika 1: PE-na-PE pristop



Slika 2: P-na-M pristop

3. OSNOVNE ZNAČILNOSTI POVEZOVALNIH MREŽ

Na izbor arhitekture povezovalnih mrež vplivajo predvsem naslednji dejavniki: delovni način, strategija krmiljenja, metoda preklapljanja in topologija mreže.

Glede na delovni način razlikujemo med sinhrono in asinhrono komunikacijo. Pri prvi se komunikacijske poti vzpostavljajo sinhrono s funkcijami manipulacije s podatki in posredovanjem podatkov in instrukcij, medtem ko se pri drugem načinu zahteve za komunikacijo posredujejo dinamično. Sistem je lahko zasnovan tako, da predvideva sinhrono in asinhrono procesiranje, tako da govorimo pravzaprav o treh delovnih načinih povezovalnih mrež: sinhroni, asinhroni in kombinirani delovni način.

Ker tipična povezovalna mreža sestoji iz množice preklopnih elementov in povezav, se komunikacija med dvema vozliščema sistema vzpostavi z odgovarjajočim krmiljenjem preklopnih elementov, pri čemer sta splošno uporabljana dva pristopa. Prvi pristop združuje krmilne funkcije vseh preklopnih elementov v centralnem krmilniku, medtem ko je pri drugem krmiljenje prepuščeno posameznim preklopnim elementom povezovalne mreže. V prvem primeru govorimo o centralnem, v drugem primeru pa o porazdeljenem krmiljenju.

Dve glavni metodologiji preklapljanja sta aparaturno in paketno preklapljanje. Pri prvem se v času komuniciranja vzpostavi fizična pot med izvorom in ponorom. Pri paketnem preklapljajanju pa podatki, organizirani v pakete, potujejo skozi povezovalno mrežo brez vzpostavljene celotne poti. Velja, da je aparaturno preklapljanje bolj primerno za veliko množino podatkov, medtem ko je paketno preklapljanje **primernejše** za krajša sporočila. Tretja metodologija, integrirano preklapljanje, združuje lastnosti obeh, tako da lahko govorimo o treh metodologijah: aparaturno, paketno in integrirano preklapljanje.

Topologija povezovalne mreže je lahko predstavljena z grafom, kjer vozlišča predstavljajo elemente mreže, povezave grafa pa ponazarjajo komunikacijske povezave med njimi. Topologije lahko klasificiramo v dve glavni skupini, to so eno in večstopenjske. Enostopenjske povezovalne mreže sestoje iz le ene stopnje preklopnih elementov, ki ji sledi nek povezovalni vzorec. Pri večstopenjskih povezovalnih mrežah pa srečamo množico stopenj preklopnih elementov in povezovalnih vzorcev, tako da pot med poljubnim vhodom in izhodom mreže vodi skozi vse stopnje.

Kartezični produkt navedenih dejavnikov oblikuje prostor povezovalnih mrež večprocesorskih sistemov. Seveda vsebuje ta prostor nekaj neza-

nimivih točk, vendar nas izbiranje primerne arhitekture omeji na podprostor primernih rešitev za postavljene zahteve.

Topolgija mreže je ključnega pomena pri določanju primerne arhitekture, zato se bomo v nadaljevanju posvetili topologijam povezovalnih mrež.

4. ENOSTOPENJSKE POVEZOVALNE MREŽE

Bistveno za to skupino mrež je, da sestoje iz le ene stopnje preklopnih elementov, kar pomeni, da je za polje N procesnih elementov potrebno polje N preklopnih elementov. Mrežo tega tipa je mogoče enostavno razdeliti tako, da se posamezni preklopni elementi pridružijo aparaturni opremi pripadajoče procesne enote. Enostopenjske mreže lahko klasificiramo v dve skupini, to so statične in dinamične mreže. Pri statičnih mrežah so povezave pasivne, kar pomeni, da je fizična povezava med dvema procesnima elementoma fiksna in je ni mod preusmeriti za komunikacijo z drugim procesnim elementom. Pri dinamičnih mrežah pa se komunikacijske poti lahko rekonfigurirajo s krmiljenjem preklopnih elementov povezovalne mreže. Ker statične enostopenjske povezovalne mreže v splošnem ne omogočajo vseh povezav direktno ampak iterativno, mora v splošnem nek podatek na poti od izvora do ponora večkrat potovati skozi mrežo.

Opisali bomo nakaj topologij enostopenjskih mrež s tem, da bomo definirali funkcije mrežnih preklopnih elementov z njihovo preslikovalno funkcijo, oziroma naborom preslikav, ki jih le ti lahko generirajo.

Imejmo N vozlišč in jih označimo v vrstnem redu od 0 do N-1. Neka preslikava P(x) = y nam pove, da obstaja povezava od vozlišča x k vozlišču y.

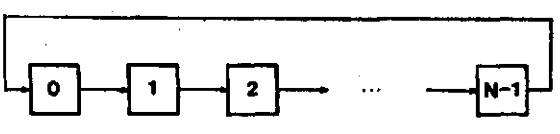
4.1. Statične enostopenjske mreže

Obroč

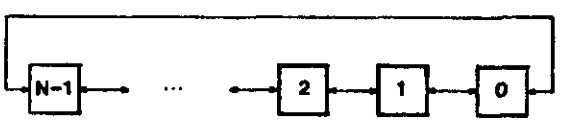
Mrežo definira premikalna preslikava.

$P_{+1}(x)=x+1 \mod N$ +1 - premikalna preslikava

To je najpreprostejša mreža, ki sestoji iz obroča procesnih elementov, tok podatkov pa je enosmeren. (slika 3.)



Slika 3: Obroč



Slika 4: Mreža sosedstva

Mreža sosedstva

Predstavlja enostavno razširitev obroča tako, da omogoča dvosmeren tok informacij. Generira dve preslikave:

$P_{+1}(x)=x+1 \mod N$ +1 - premikalna preslikava
$P_{-1}(x)=x-1 \mod N$ -1 - premikalna preslikava

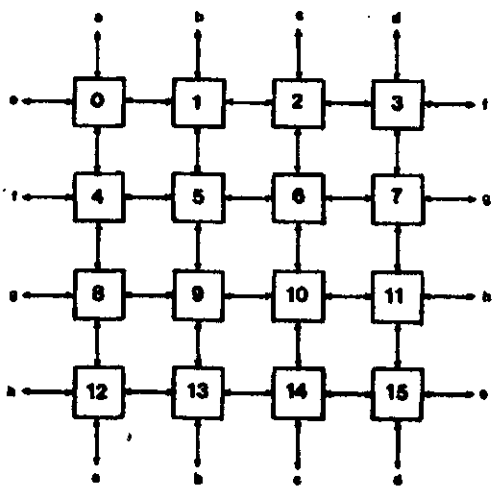
Obroč in mreža sosedstva sta primerna za relativno majhno število procesnih elementov, saj za povprečni čas potovanja podatkov (Td) pri obroču velja Tp=(N/2)Td pri mreži sosedstva pa Tp=(N/4)Td, kjer Td predstavlja čas potreben za komunikacijo med dvema sosednima procesnima elementoma.

Illiac mreža

Če mrežo sosedstva obogatima z dvema novima preslikavama, tako da imamo

$P_{+1}(x)=x+1 \mod N$ +1 - premikalna preslikava
$P_{-1}(x)=x-1 \mod N$ -1 - premikalna preslikava
$P_{+n}(x)=x+n \mod N$ +n - premikalna preslikava
$P_{-n}(x)=x-n \mod N$ -n - premikalna preslikava

in če je N popolen kvadrat ($N = n^2$), dobimo Illiac mrežo, ki je uporabljena v sistemu Illiac IV. V tej mreži je vsak procesni element povezan s svojim severnim, južnim, vzhodnim in zahodnim sosedom, kot prikazuje slika 5.



Slika 5: Illiac mreža

Mreža hiper-kocke

Pri tej in naslednjih predstavljenih topologijah opišemo preslikave laže z logičnimi in premikalnimi funkcijami nad biti v dvojiškem zapisu oznake vozlišča. Preslikava:

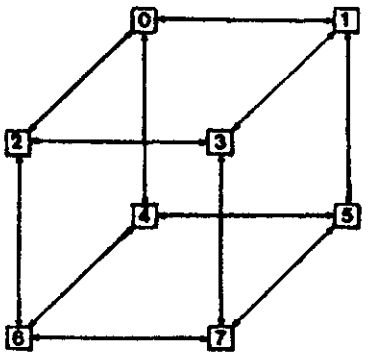
$$O(p_{q-1}p_{q-2}\ldots p_1p_0) = r_{q-1}r_{q-2}\ldots r_1r_0$$

pove, da obstaja povezava od vozlišča z oznako $p_{q-1}p_{q-2}\ldots p_1p_0$ k vozlišču $r_{q-1}r_{q-2}\ldots r_1r_0$.

Mrežo hiper-kocke dimenzije m definira m preslikav tipa:

$$C(p_{m-1}p_{m-2}\ldots p_1p_0) = p_{m-1}p_{m-2}\ldots\bar{p}_i\ldots p_1p_0$$
$$i = 1,2,\ldots,m$$

Če si procesne elemente take mreže predstavljamo kot oglišča m-dimenzionalne kocke vidimo, da je vsak procesni element povezan z m sosedi, kar prikazuje slika 6.



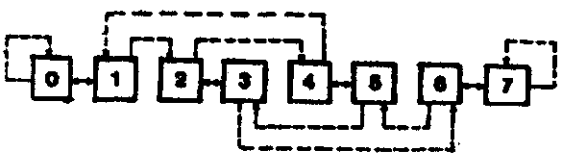
Slika 6: Mreža hiper-kocke (m = 3)

Mreža popolnega mešanja

Mrežo popolnega mešanja (perfect shuffle) definirata preslikavi mešanja in zamenjave.

$$S(p_{q-1}p_{q-2}\ldots p_1p_0) = p_{q-2}p_{q-3}\ldots p_0p_{q-1}$$
$$E(p_{q-1}p_{q-2}\ldots p_1p_0) = p_{q-1}p_{q-2}\ldots\bar{p}_0$$

Prikazuje jo slika 7. Cele črte ponazarjajo zamenjavo, prekinjene pa mešanje.



Slika 7: Mreža popolnega mešanja

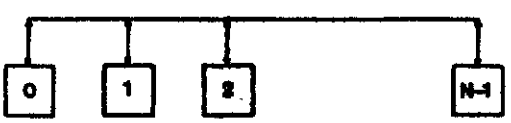
4.2. Dinamične enostopenjske mreže

Skupno vodilo

Mrežo opisujejo preslikave:

$$B(x) = y \qquad y = 0,1,\ldots,N-1 \qquad y = x$$

Ta mreža omogoča vse preslikave direktno, toda zaradi le ene fizične poti samo eno naenkrat.



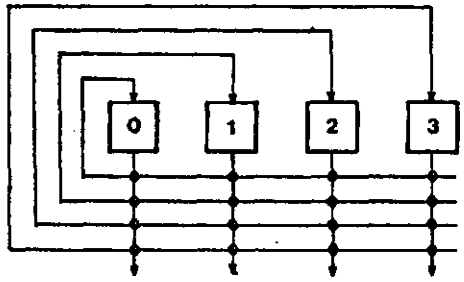
Slika 8: Skupno vodilo

Popolno stikalno polje (cross bar)

Preslikovalna funkcija te mreže je enaka kot pri skupnem vodilu, torej:

$$B(x) = y \qquad y = 0,1,\ldots,N-1 \qquad y = x$$

Bistvena razlika je v tem, da omogoča ta mreža vse povezave hkrati.



Slika 9: Popolno stikalno polje

Poudarimo še enkrat, da zahteva statična enostopenjska mreža iterativen prenos podatkov skoznjo za dosego poljubne preslikave. Z iterativnostjo mislimo na dejstvo, da mora nek podatek, poslan z izvornega procesnega elementa proti ponornemu elementu, na svoji poti obiskati neko množico vmesnih oziroma posredovalnih elementov in nekajkrat potovati skozi povezovalno mrežo. Zato je za neko mrežo bistvenega pomena podatek o številu potrebnih iteracij za dosego poljubne preslikave. Z namenom primerjave predstavljenih mrež vpeljimo pojem razdalje. Razdalja med dvema elementoma i in j je enaka številu potrebnih iteracij za dosego preslikave i --> j.

Povezovalne mreže bomo primerjali po dveh kriterijih: kriteriju maksimalne oddaljenosti in kriteriju distribucije.

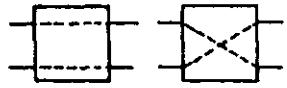
Kriterij maksimalne razdalje podaja največjo razdaljo med dvema elementoma mreže, medtem ko kriterij distribucije opisuje razdaljo potrebno za posredovanje neke informacije vsem elementom v mreži, oziroma je enak številu zaporednih grup paralelnih preslikav potrebnih za dosego tega cilja, s tem, da so znotraj neke grupe vse preslikave istega tipa. Oba kriterija predstavljata nekakšno merilo učinkovitosti posamezne mreže. Za opisane povezovalne mreže sta podana v spodnji tabeli.

Pov. mreža	Maks. razdalja	Distribucija
Obroč	N-1	N-1
M.sosedstva	N/2	N-1
Illiac	$\sqrt{N}$-1	2($\sqrt{N}$-1)
Hiper-kocka	log N	log N
Pop. mešanje	2log N-1	2log N-1
Skupno vodilo	1	N-1
Pop.st.polje	1	log N

5. VEČSTOPENJSKE POVEZOVALNE MREŽE

Ta tip povezovalne mreže omogoča vse preslikave vhodov na izhode direktno. Dobljen je z rekurzivno dekompozicijo popolnega stikalnega polja (cross bar) do preklopnih elementov velikosti 2 X 2. Večstopenjske povezovalne mreže torej sestoje iz več stopenj preklopnih elementov oz. celic. Bistveni parametri večstopenjske mreže so: uporabljena preklopna celica, topologija in način krmiljenja preklopnih elementov.

Kot rečeno je za preklopno celico navadno uporabljeno stikalo z dvema vhodoma in dvema izhodoma, ki je lahko v enem izmed dveh (slika 10a) ali štirih stanj (slika 10b). V prvem primeru omogoča mreža vse preslikave tipa eden-na-eden, v drugem primeru pa tudi eden-na-več.



Slika 10a: Preklopna celica z dvemi stanji

Slika 10b: Preklopna celica s štirimi stanji

Topologijo mreže predstavlja uporabljen povezovalni vzorec med naborom vhodnih in izhodnih linij povezovalne mreže. Večstopenjsko mrežo sestavlja M stopenj s po N/2 preklopnih celic, kjer za M velja neenačba:

$$\log N \leq M \leq 2\log N - 1$$

kjer je N enak številu vhodnih oziroma izhodnih linij povezovalne mreže. Iz tega je razvidno, da kompleksnost povezovalne mreže raste s faktorjem O(Nlog N), kar je v primerjavi z $O(N^2)$ kompleksnostjo popolnega preklopnega polja precejšen prihranek predvsem pri velikih vrednostih N. Večstopenjske mreže so torej najbolj primerne za sisteme z velikim številom procesnih elementov.

Kar zadeva strategijo krmiljenja in metodo preklapljanja srečamo pri večstopenjskih povezovalnih mrežah centralno in porazdeljeno krmiljenje, kot tudi aparaturno in paketno preklapljanje. Izbrani koncept določa izgled preklopne celice, saj mora le-ta upoštevati protokol, ki ga določata strategija krmiljenja in metoda preklapljanja. Opisali bomo topologije najbolj uporabljanih večstopenjskih povezovalnih mrež.

Mreža osnovne linije (Baseline)

Sestavlja jo log N stopenj ,katerih vsaka ima N/2 preklopnih celic 2 X 2. Pripišimo mreži sledeče oznake: stopnje naj bodo oštevilčene od leve proti desni s številkami od 0 do log N-1, povezovalne vzorce označimo v isti smeri od 0 do log N. Preklopnim celicam posamezne stopnje pripišimo vrednosti od 0 do N/2-1 v smeri od zgoraj navzdol, prav tako pa označimo posamezne povezave znotraj vsakega povezovalnega vzorca od 0 do N-1. Preslikava podana v obliki:

$$PE(P)_i = (R)_j \text{ s povezavo } (Q)$$

pomeni, da obstaja povezava med preklopnim elementom P stopnje i in preklopnim elemntom R stopnje j in da je to povezava Q. P, R in Q bodo nastopali v obliki dvojiškega zapisa.

Topologijo mreže osnovne linije podajata naslednji pravili:

$$BC(p_{i}p_{i-1}\ldots p_{1}p_{i}) = (p_{i}\ldots p_{i+1}\,0\,p_{i-1}\ldots p_{2}p_{1})_{i+1}$$

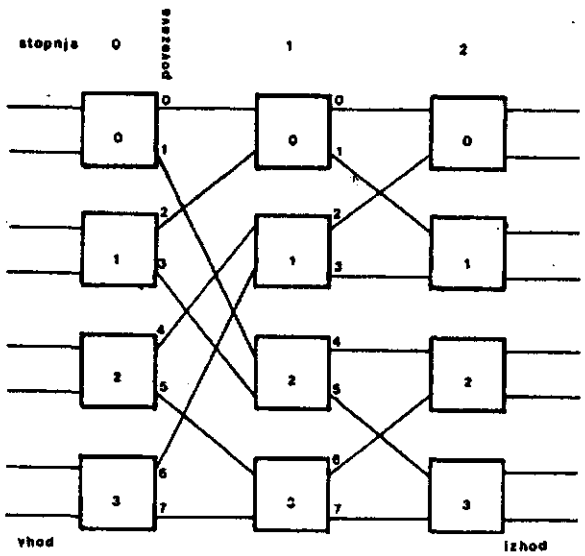
s povezavo $(p_{i}p_{i-1}\ldots p_{1}0)$ $0 \le i < l$

in

$$BC(p_{i}p_{i-1}\ldots p_{1}p_{i}) = (p_{i}\ldots p_{i+1}\,1\,p_{i-1}\ldots p_{2}p_{1})_{i+1}$$

s povezavo $(p_{i}p_{i-1}\ldots p_{1}1)$ $0 \le i < l$

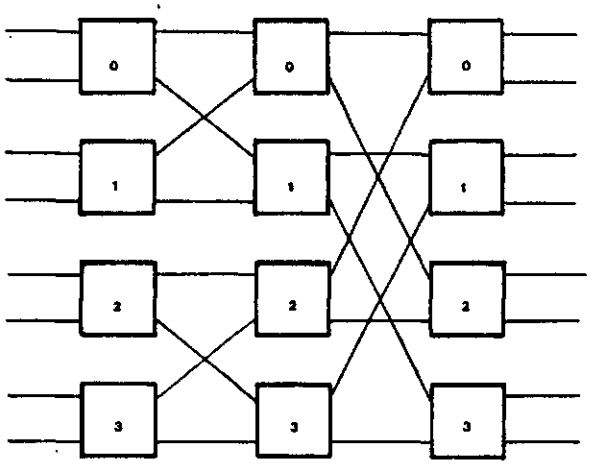
Mrežo osnovne linije z N=8 podaja slika 11.



Slika 11: Mreža osnovne linije

Posredna n-kocka (indirect n-cube)

Povezovalni vzorci te mreže temelje na principu povezav pri hiper-kocki s tem, da je v vsaki stopnji realizirana ena izmed povezav, to pomeni, da se poljubni dva preklopni celici dveh sosednih stopenj razlikujeta kvečjemu v enem bitu svojih oznak. Formalni opis topologije podajata naslednji formuli:



Slika 12: Mreža posredne n-kocke

$$KC(p_{i}p_{i-1}\ldots\ldots p_{1}p_{i}) = (p_{i}\ldots p_{i+2}\,0\,p_{i}\ldots p_{2}p_{1})_{i+1}$$

s povezavo $(p_{i}p_{i-1}\ldots p_{1}0)$ $0 \le i < l$

in

$$KC(p_{i}p_{i-1}\ldots\ldots p_{1}p_{i}) = (p_{i}\ldots p_{i+2}\,1\,p_{i}\ldots p_{2}p_{1})_{i+1}$$

s povezavo $(p_{i}p_{i-1}\ldots p_{1}1)$ $0 \le i < l$

Če tej povezovalni mreži zamenjamo vrstni red povezovalnih vzorcev med stopnjami, dobimo topologijo, ki je znana kot Banyanova mreža. Izgled posredne n-kocke za N=8 je na sliki 12.

Omega povezovalna mreža

Ta mreža, prav tako kot gornji, sestoji iz log N stopenj s po N/2 preklopnimi celicami. Povezovalni vzorci med posameznimi stopnjami mreže so vsi enaki in temelje na preslikavi popolnega mešanja po naslednjem pravilu:

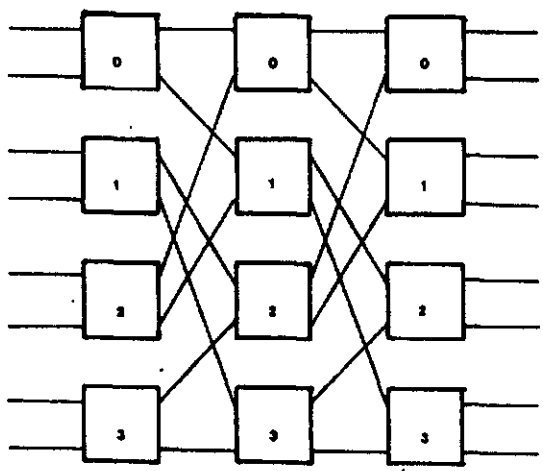
$$OC(p_{i}p_{i-1}\ldots\ldots p_{1}p_{i}) = (p_{i-1}\,p_{i-2}\ldots p_{1}0)_{i+1}$$

s povezavo $(p_{i}p_{i-1}\ldots p_{1}0)$ $0 \le i < l$

in

$$OC(p_{i}p_{i-1}\ldots\ldots p_{1}p_{i}) = (p_{i-1}\,p_{i-2}\ldots p_{1}1)_{i+1}$$

s povezavo $(p_{i}p_{i-1}\ldots p_{1}1)$ $0 \le i < l$

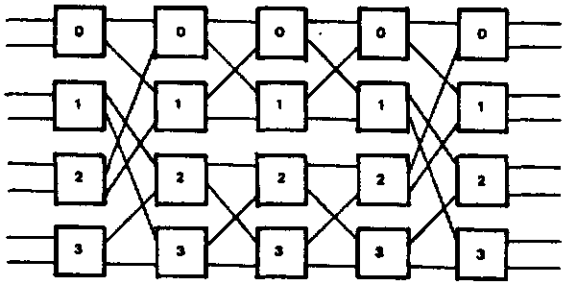


Slika 13: Omega mreža

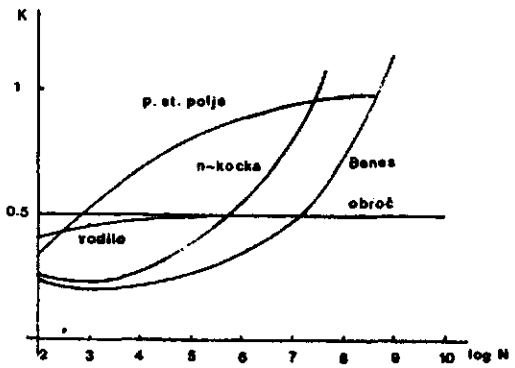
Vse tri predstavljene povezovalne mreže sestavlja log N stopenj, katerih vsaka ima N/2 preklopnih elementov 2 x 2. Tak tip mreže omogoča direktno povezavo poljubnega vhoda s poljubnim izhodom, vendar ne sočasno. Lahko bi pokazali, da so te mreže topološko ekvivalentne ena z drugo, oziroma, da imajo izomorfno topologijo. To pomeni ,da z ustrezno premestitvijo preklopnih elementov pri enem tipu mreže pridemo do drugega.

Benesova mreža

Povezovalna mreža, pri kateri se lahko v vsakem trenutku doseže povezava nekega vhoda z željenim izhodom s preureditvijo že obstoječih povezav, mora imeti večje število stopenj, navadno M = 2log N - 1. Primer take mreže je Benesova mreža. Povezovalni vzorci te mreže so isti kot pri mreži osnovne linije s tem, da vsak vzorec nastopa dvakrat. Prikazuje jo slika 14.



Slika 14: Benesova mreža



Slika 15: Diagram kvalitete mrež

Mreža	P	T	E	L	K
Obroč	N	N	N	N	1/2
Vodilo	1	2	N	1	N/(2N+2)
Pop.st.	N	4	N/4	2N	N/(N+8)
n-kocka	N/2	logN	NlogN/2	N+NlogN	N/(logN(2+3logN))
Benes	N	2logN-1	NlogN-N/2	2NlogN	2N/((2logN-1)(6logN-1))

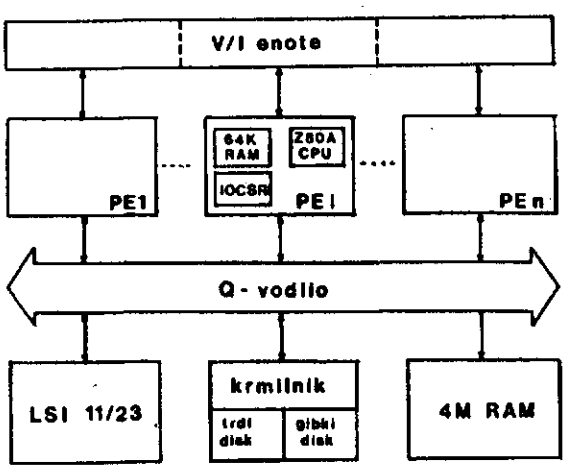
6. OCENA KVALITETE MREŽ

Predstavili smo topologije pomembnejših povezovalnih mrež, ki so se pojavile v paralelnih računalniških sistemih. Poudarili smo že, da je izbira povezovalne mreže odvisna od več parametrov, med katerimi je najpomembnejši število procesnih elementov. Število procesnih elementov določa fizično kompleksnost mreže in njene funkcionalne lastnosti. Fizična kompleksnost mreže se izraža v številu preklopnih elementov in številu fizičnih povezav, o funkcionalnih lastnostih pa govori maksimalno število sočasnih zvez v mreži in pa podatek o času, potrebnem za prenos paketa podatkov skozi mrežo. Vpeljimo pojem kvalitete mreže, ki nam bo služila kot kriterij pri določanju primerne mrežne topologije za določeno število procesnih elementov. Kvaliteto mreže definira naslednja enačba:

$$K = NP/(T(E+L))$$

N.....število procesnih elementov sistema
P.....maksimalno število sočasnih zvez
T.....čas prehoda skozi mrežo
E.....število preklopnih elementov mreže
L.....število fizičnih povezav mreže

V gornji tabeli smo podali posamezne parametre in kvaliteto nekaj mrež. Izmed statičnih enostopenjskih mrež smo izbrali obroč, izmed večstopenjskih z M=log N pa mrežo posredne n-kocke. Lahko bi se prepričali, da je kvaliteta ostalih statičnih enostopenjskih mrež istega reda kot kvaliteta obroča, kvaliteta večstopenjskih mrež z M=log N pa istega reda kot kvaliteta posredne n-kocke. Iz diagrama na sliki 15. lahko vidimo, da je za majhno število procesnih elementov najboljša rešitev popolno stikalno polje, dobra pa sta tudi obroč in skupno vodilo. Za več kot 64 procesnih elementov pa sta kvalitetnejši posredna n-kocka in Benesova mreža.



Slika 16: Arhitektura sistema na Q-vodilu

7. ZAKLJUČEK

Naj namesto zaključka predstavimo arhitekturo eksperimentalnega večprocesorskega sistema s skupnim vodilom, ki nastaja na našem institutu. Arhitekturo sistema prikazuje slika 16. Uporabljeno je asinhrono DEC-ovo Q vodilo. Funkcijo arbitraže skupnega vodila prevzema procesor LSI 11/23, ki je hkrati tudi skupni vmesnik masovnega pomnilnika; vsak procesor sistema lahko posega po masovnem pomnilniku le preko LSI 11/23. Trenutno je procesni element (PE) zasnovan na mikroprocesorju Z80A. Vsebuje 64K zlogov lokalnega pomnilnika. Z zunanjim svetom ga povezuje 8 nadzorno/statusnih besed (IOCSR). Vsak izmed procesnih elementov lahko posega neposre-

dno po svojem lokalnem pomnilniku, po skupnem pomnilniku in po nadzorno/statusnih besedah drugega procesnega elementa. Ker ima Q-vodilo možnost 22 bitnega naslavljanja, lahko v sistemu uporabimo do 4M zlogov skupnega pomnilnika.

8. LITERATURA

[1] R.W.Hockney, C.R.Jesshope: Parallel Computers, Adam Hilger Ltd, Bristol 1981

[2] Tse-Yun-Feng: A Survey of Interconnection Networks, Computer, December 1981 pp.12-27

[3] Chuan-Lin-Wu, Tse-yun-Feng: On a Class of Multistage Interconnection Networks, IEEE Transactions on Computers, August 1980 pp. 694-702

[4] H.J.Siegel: Interconnection Networks for SIMD Machines, Computer, Junij 1979 pp. 57-65

[5] H.J.Siegel: A Model of SIMD Machines and a Comparison of Various Interconnection Networks, IEEE Transactions on Computers, December 1979, pp. 907-917

[6] H.J.Siegel: The Multistage Cube:A Versatile Interconnection Network, Computer, December 1981, pp. 65-76

[7] Chin-Yuan-Chin, Kai-Hwang: Connection Principles for Multipath Packet Switching Networks, The 11th Anual International Symposium of Computer Architecture, June 1984, pp. 99-108

[8] P.Brajak: Paralelno procesiranje: arhitektura 90-tih godina, Zbornik jugoslavenskog savjetovanja o novim generacijama računala, MIPRO 86, Maj 1986, pp. 33-46

LADIJSKI INFORMACIJSKI SISTEM IN RAČUNALNIŠKO TOVORJENJE LADIJ

INFORMATICA 4/86

UDK: 681.323:656.616.3

Andrej Novak
Višja pomorska šola Piran

Članek opisuje široko uporabo elektronskih in računalniških sistemov v pomorskem prometu. Sledi opis ladijskega informacijskega sistema in njegovih podsistemov. Prikazano je sedanje stanje na ladjah in možnosti, katere imamo pri nas za vključitev v to panogo industrije. Ena izmed naših možnosti je izdelovanje računalniških sistemov za tovorjenje ladij. Članek je zaključen z opisom osnovnih načel za izdelavo takega sistema.

Ship's information system
and loading of ships by computer.

This paper describes the wide use of electronic and computer systems in the sea transport. A description of the ship's information system and its subsystems follows. The present situation on board the ships and our possibilities to be included in that field of industry is pointed out. One of our possibilities is the building up of such computer systems for the loading of ships. Finally a description of the basic principles for the building up of such system is included.

1. UVOD

Hitro uvajanje elektronike, avtomatizacije in računalništva opažamo v vseh panogah gospodarstva, prav tako tudi v pomorskem gospodarstvu.

Potrebno je pripomniti, da so se zgoraj omenjene tehnologije pojavljale v pomorskem prometu vedno z manjšo časovno zakasnitvijo. Vzrok za njo so strogi predpisi pomorskih uradov. Ti namreč predpisujejo zaradi varnosti plovbe v praksi predhodno preverjene in varne sisteme.

Tako so se z zaporednim razvojem elektronike, avtomatizacije in računalništva v istem kronološkem zaporedju pojavili na ladjah v prvih desetletjih tega stoletja najprej radio-oddajniki, sprejemniki, radiogoniometer in vrtavčni kompasi. Obdobje druge svetovne vojne je dalo sisteme Consol, Loran, Decca in radar. V naslednjih dveh desetletjih pridobi pomorstvo globalne navigacijske sisteme Omega, satelitski in inercialni sistem. V zadnjih petnajstih letih pa so se na vseh področjih pomorske dejavnosti v veliki meri začeli uporabljati računalniški sistemi. Tako današnja poveljniška mesta v pomorskem prometu predstavljajo elektronsko računalniški sistemi, v katerih se klasični pomorščak srednjih let le s težavo znajde.

Danes bi informacijske sisteme v pomorskem prometu lahko razdelili v sledeče skupine:

1.1. Komercialno-operativni informacijski sistem

Nahaja se v pomorski delovni organizaciji. Njegova naloga je ekonomsko in operativno vodenje delovne organizacije in njej pripadajočih ladij.

1.2. Ladijski informacijski sistem

Nahaja se na ladjah. Služi za varnost ladij v plovbi in njihovo ekonomsko izkoriščenost.

Zgoraj omenjena sistema sta povezana s satelitskim komunikacijskim sistemom.

1.3. Vremensko vodenje ladij

Ta sistem sestavljajo sinoptični center, računalniški center in ladje. Glede na vremenske in oceanografske pogoje ter plovne lastnosti ladij računalnik za vsako ladjo posebej izračunava njeno dnevno optimalno pot. Ladjarji, ki so uporabljali ta sistem, so beležili vidne prihranke zaradi časovno krajše plovbe, manjše porabe goriva in nepoškodovanega tovora.

1.4. Informacijski sistemi za varnost plovbe

To so radarsko računalniški sistemi novejšega datuma. Osrednji računalniški sistem se nahaja največkrat v pristanišču. Radarske postaje kot senzorji se nahajajo v pristanišču, pred njim in v področjih gostega ali nevarnega prometa. Računalnik spremlja gibanje ladij. Pri tem opozarja na spremembe v njihovih gibanjih, na plovbo, katera ni v skladu s predpisi in na ladje, ki so v nevarnosti. Kontrola prometa in ladje so zaradi potrebnih informacij povezani s komunikacijskim sistemom.

1.5. Simulacijski sistemi

Tudi ti sistemi so novejšega datuma. To so navigacijski, radarski, strojni, protipožarni simulacijski sistemi, simulacijski sistemi za tovorjenje ladij in manevriranje z njimi ter drugi. Uporabljajo se predvsem za izobraževanje pomorskih kadrov, pa tudi za raziskovalno delo. Izobraževanje pomorskih kadrov na simulatorjih je ekonomično, obširno, na visokem nivoju in hitro.

Tudi pri gradnji ladij uporabljajo tuje in naše ladjedelnice CAD in druge računalniške sisteme.

V dalnjem tekstu bomo opisali le ladijski informacijski sistem, kasneje pa bomo opis še zožili na načela izdelave enega njegovih podsistemov, to je podsistem za računalniško tovorjenje ladij.

2. LADIJSKI INFORMACIJSKI SISTEM

Ladijski informacijski sistem je sestavljen iz več podsistemov. Ukrepi za varno in ekonomično plovbo, ki sledijo iz zbranih informacij, so na današnji stopnji še vedno prepuščeni človekovi presoji. Zato so sistemi le delno povezani, niso pa povezani v integrirani sistem za popolno avtomatično vodenje ladje, čeprav so se taki poskusi vršili. Večina današnjih podsistemov deluje na osnovi računalnikov. Vendar ladja zopet nima centralne procesorske enote. Njena preobremenitev ali izpad bi delno ali popolnoma ohromila ladjo. Tako ima vsak podsistem in večkrat celo vsak instrument v njem lasten računalnik. Izpad posameznega računalnika tako ne predstavlja nevarnosti za plovbo, saj se sistemi prekrivajo, velikokrat pa so podvojeni.

Ladijski informacijski sistem bi lahko razdelili na devet podsistemov.

2.1. Navigacijski podsistem

Sprejemniki tega podsistema se nahajajo na poveljniškem mostu ladje. Podsistem služi za varno vodenje ladje po poti od točke odhoda do točke prihoda. Po tej poti vodita ladjo vrtavčni kompas in avtopilot. Vendar se ladja zaradi učinkov vetra in morskih tokov večkrat oddalji od predpisane poti. Da bi ugotovili pravi položaj ladje in jo usmerili na predpisano pot, smo do nedavnega uporabljali klasične terestrične in astronomske metode. Danes v ta namen večinoma uporabljamo elektronsko računalniške sprejemnike navigacijskih sistemov. To so radiogoniometer, Decca, Loran, Omega in satelitski sprejemnik. Ker oddajniki omenjenih sistemov niso pod nadzorom ladje, so ti sistemi strateško odvisni. Zato so neprestana prizadevanja za izpopolnjevanje strateško neodvisnega inercialnega sistema. Morsko globino in hitrost ladje merimo z elektronskimi globinomerji in brzinomerji. Za določanje položaja ladje v obalni plovbi in preprečevanje trčenj na morju uporabljamo radar. Pri tem je dosežena visoka stopnja avtomatizacije s sodobnimi ARPA radarji, kateri vsebujejo v sebi iz dneva v dan močnejše računalnike, danes že do nekaj MB.

Zgoraj omenjeni sprejemniki dobivajo informacije od oddajnikov sistema s posredovanjem elektromagnetnih valov. Ker so ti pri širjenju skozi atmosfero podvrženi refrakciji, položaj ladje ni popolnoma točen. Če v geodeziji ali pri raziskavah naftnih ležišč zahtevamo večjo točnost položaja, uporabljamo integrirani navigacijski sistem. Ta združuje dva ali več zgoraj opisanih sprejemnikov. Centralni procesor nato obdeluje dobljene informacije, iz vsakega sistema izloči pomanjkljivosti in tako daje najtočnejši možni položaj ladje.

2.2. Pogonski podsistem

S tem podsistemom nadzorujemo pravilno delovanje glavnega ladijskega stroja, pomožnih strojev in generatorjev. Služi za sinhronizacijo generatorjev ter pravilno nastavitev strojev, s čimer zmanjšujemo porabo pogonskega goriva. Podsistem tudi opozarja na bodoča vzdrževalna dela. V delovnem času ta podsistem signalizira napake in okvare pogona v kontrolno kabino pogona. Izven delovnega časa pa se alarmi prenesejo v kabino dežurnega častnika stroja in na poveljniški most. Večinoma je ta podsistem vezan na navigacijskega na poveljniškem mostu, odkoder lahko tako neposredno upravljamo z glavnim ladijskim strojem.

2.3. Komunikacijski podsistem

Za interne komunikacije na ladji uporabljamo telefon in alarmne naprave. Za komunikacije med ladjami ter ladjo in kopnom uporabljamo na razdaljah do 40 navtičnih milj VHF sistem, na velikih razdaljah pa radiotelegrafijo in radiofonijo. Novost predstavlja teleprinterska zveza preko telekomunikacijskih satelitov.

2.4. Protipožarni podsistem

Služi za ugotavljanje in gašenje požarov. Senzorji podsistema se nahajajo v kabinah, strojnem prostoru in v tovornih skladiščih ladje. Detektorja se nahajata na poveljniškem mostu in v kontrolni kabini pogona. Pravočasen alarm podsistema omogoča hitro in uspešno gašenje šele nastajajočega požara. Protipožarna sredstva se lahko vključujejo iz poveljniškega mosta, kontrolne kabine pogona ali iz ostalih protipožarnih mest na ladji.

2.5. Meteorološki podsistem

Meteorološke in oceanografske informacije so izredno važne za varnost ladje in tovora ter ekonomičnost plovbe. Te informacije so na ladji potrebne, da se z njo izognemo manjšim toda nevarnim baričnim telesom kot so tropski orkani, ali da z ladjo izvedemo pravilen manever v večjih baričnih tvorbah kot so cikloni srednjih širin.

Meteorološke in oceanografske informacije dobivamo v glavnem na tri načine.

2.5.1. Klimatološke informacije dobivamo iz publikacij.

2.5.2. Sinoptične informacije so najvažnejše. Normalno jih dobiva ladja štirikrat dnevno iz vremenoslovnih centrov s pomočjo radiotelegrafije, radiofonije ali radio faksimila. Slednji vsakih 6 ur avtomatično izriše vremensko karto. Če je ladja vključena v vremensko vodenje ladij, kot je to že opisano v poglavju 1.3., dobiva telegrafsko sporočila o njeni optimalni

dnevni plovni poti.

2.5.3. Mikroklimatološke informacije o temperaturah in vlagi v ladijskih skladiščih dobivamo s pomočjo senzorjev ali z merjenji. Ti podatki so pomembni za preprečevanje poškodb tovora.

2.6. Podsistem za tovor

V analogni izvedbi ta podsistem ni nudil zadovoljive natančnosti. V digitalni izvedbi, ki je novejšega datuma, pa ta podsistem nudi potrebno točnost. V tej izvedbi je nujno potreben na kontejnerskih ladjah, služi za popolno avtomatizirano tovorjenje večjih tankerjev, vse hitreje pa se uvaja tudi na ostalih tipih ladij. S tem podsistemom vršimo pravilno razporeditev tovora v ladijskih skladiščih, izračunavamo vgrez, trim, prečno in dinamično stabilnost ladje, določamo upogibne momente in strižne sile. Temu sledi računalniški izpis podatkov in rezultatov ter risanje plana tovora. Podatke o potovanjih shranjujemo v datotekah. Najnovejše izvedbe tega podsistema izračunavajo obnašanje ladje pri prodoru vode. Ena od možnosti razvoja tega podsistema je optimizacija tovorjenja ladje. Bolj obširno bo računalniška problematika tega podsistema opisana v petem poglavju tega članka.

2.7. Podsistem za ladijske zaloge

Ta podsistem daje pregled zalog rezervnih delov za krovno službo in pogon. Bistvena je povezava takih ladijskih podsistemov z osrednjim v ladjarskem podjetju.

2.8. Administrativni podsistem

Ta računalniški podsistem vodi pregled nad dokumenti ladje in posadke, ter opozarja na njihovo iztekanje. Z njim vodimo zaloge hrane in ekonomičnost njenih nakupov, ter obračunavamo OD posadke.

Trgovske ladje, katere ne prevažajo potnikov, večinoma nimajo zdravnika. Imajo pa ambulanto in apoteko s predpisanimi zdravili, prvo pomoč pa nudijo častniki krova. V težjih bolezenskih primerih je današnja praksa povezava ladje preko komunikacijskega sistema z določenimi zdravstvenimi centri za pomorstvo, ki dajejo za bolnika diagnozo in terapijo. Tu je možnost razvoja ekspertnega sistema, ki bi na osnovi anamneze iz dialoga računalnik-človek postavil diagnozo in terapijo za bolnika na ladji.

2.9. Upravni in pravni podsistem

Ta podsistem daje poveljniku ladje podatke o upravnih normah v različnih državah sveta in njihovih pristaniščih, ter pravna navodila v izjemnih primerih in primerih nezgod.

3. SEDANJE STANJE NA LADJAH

V celoti tak ladijski informacijski sistem s svojimi devetimi podsistemi, kot je opisan v prejšnjem poglavju, na ladjah večinoma še ne obstoja. Obstojajo podsistemi 1 do 5, podsistem za tovor se hitro uvaja, podsistema za zaloge in administracijo se šele uvajata, dočim je upravni in pravni podsistem šele zamisel, katero bo koristno izvesti.

V primerjavi z vrednostjo ladje cena celotnega ladijskega informacijskega sistema ni visoka. Vendar so klasične metode v primerjavi z elektronskimi in računalniškimi še cenejše. Zakaj se torej na ladje uvajajo sodobni informacijski sistemi? Prvi vzrok je varnost plovbe. Upravni predpisi zato določajo, da morajo biti ladje opremljene z radiogoniometrom, VHF primopredajnikom in nad določeno nosilnostjo z ARPA radarjem. Povečana varnost zmanjšuje škode, temu primerno se tudi zniževajo visoke zavarovalne premije, odkoder vidimo, da se v varnosti pravzaprav skriva ekonomski vzrok. Drugi vzrok je čisto ekonomskega značaja. Avtomatizirani, elektronsko in računalniško vodeni procesi zmanjšujejo drage ladijske posadke, zmanjšujejo porabo goriva, ter skrajšujejo čase potovanja ladje in njenih postankov v pristaniščih. Privzemimo, da je življenska doba ladje približno 15 let. Po izračunih nekaterih podjetij se zaradi prej omenjenih vzrokov investicijska vrednost avtopilota, satelitskega navigacijskega sprejemnika in računalniškega sistema za tovorjenje ladje izplača prej kot v enem letu. Uporaba teh sistemov prinaša v naslednjih 14 letih čisti dobiček ladjarju. Tako je uporaba sodobnega ladijskega informacijskega sistema za razumnega ladjarja ekonomska nujnost.

Oglejmo si stanje pri nas. Kot primer vzemimo novo ladjo Portorož, katero je Splošni plovbi iz Pirana letos junija predala ladjedelnica Uljanik iz Pule. Na poveljniškem mostu ima ladja radiogoniometer, radar, ARPA radar, globinomer, merilec hitrosti, VHF primopredajnik, požarni detektor, radio faksimil in satelitski sprejemnik. Ker je satelitski sistem globalnega značaja, prekriva sicer z manjšo natančnostjo področja lokalnih sistemov Loran in Decca. Zato ladji nista nujno potrebna sprejemnika teh sistemov. Pogon je avtomatsko krmiljen iz kontrolne kabine pogona, upravljanje z glavnim strojem pa se lahko vrši iz poveljniškega mosta. V delovnem prostoru se nahaja računalnik za tovorjenje ladje. Ladja nima informacijskih podsistemov za zaloge, administracijo in upravno pravnega. Iz navedenega vidimo, da je ladja v pogledu ladijskega informacijskega sistema zadostno in sodobno opremljena. Isto opažamo tudi pri ostalih novogradnjah flote SFRJ in na novejših tujih ladjah.

4. NAŠE MOŽNOSTI RAZVOJA IN IZDELAVE LADIJSKIH INFORMACIJSKIH PODSISTEMOV

Danes domače ladjedelnice gradijo ladje za domače in tuje ladjarje. Domači ladjarji pa gradijo ladje tudi v tujih ladjedelnicah. Pri tem plačujemo visoko ceno v devizah za nakup celotne tehnologije ladijskega informacijskega sistema, katero v celoti vedno uvozimo iz tujine.

Oglejmo si ponovno podsisteme ladijskega informacijskega sistema v istem vrstnem redu, kot smo jih že opisali.

Sedanja globalna navigacijska sistema sta Omega in satelitski TRANSIT sistem. Prvi naj bi imel 8 oddajnikov razvrščenih po vsem svetu, drugi pa 6 lastnih satelitov. Razvoj in vzdrževanje takih sistemov zahteva velike investicije, katere zmorejo le tehnološko in ekonomsko najbolj razvite dežele. Sedaj je v razvoju nov satelitski sistem NAVSTAR, ki naj bi začel delovati okoli leta 1993. Pri njegovi izvedbi bo soudeležena večina vrhunskih dosežkov tehnoloških in naravoslovnih znanosti. Tudi izdelava radarjev zahteva visoko tehnologijo.

Navigacijska sistema Loran in Decca sta lokalnega pomena. Njun razvoj je zanimiv v vojaške namene. Oddajnike sistema je potrebno varovati, posebno pa še, če so izven državnih meja. Eno od Loran verig, katera pokriva Sredozemlje in Jadran, uporabljamo tudi v trgovski mornarici. Navigacijska točnost sistema je par sto

metrov. To točnost smo imeli priliko opaziti v zadnjem ameriškem napadu na Libijo. Zato je razumljiv povračilni poskus poškodovanja loran oddajnika na Lampedusi.

Pri nas izdelujemo dobre komunikacijske sisteme. Tu je prehod v pomorsko gospodarstvo lahek. Tudi avtopilote lahko izdelujemo doma.

Meteorološki sistem za pomorstvo je v Jadranu delno razvit. Zanimivo bi bilo raziskati nadalnje možnosti njegovega razvoja in uvedbe meteorološkega vodenja ladij v Jadranu.

Iz zgoraj omenjenega sledi, da zahteva razvoj prvih petih podsistemov ladijskega informacijskega sistema obširen razvoj, večje investicije in visoko tehnologijo. Razvidno je tudi, da je pri nas trenutno najlažji prestop v pomorsko gospodarstvo na področju komunikacijskega in meteorološkega podsistema.

Bistveno drugačen pa je položaj na področju zadnjih štirih podsistemov. Njihova osnova je nova, hitro se razvijajoča računalniška industrija, kjer še lahko ujamemo korak s časom.

Pri podsistemu za tovor je potrebno izdelati programsko opremo za vsak tip ladij posebej. Če so ladje znotraj tipa ladij različno pregrajevane ali dograjevane, je potrebno programsko opremo prirediti za vsako ladjo posebej. Tipov ladij pa je v svetu in pri nas veliko. Tu je možnost dela za veliko prizvajalcev programske opreme, torej tudi za nas. Način vodenja zalog in administracije je odvisen od posamezne delovne organizacije. Najverjetneje bomo morali za naše delovne organizacije tako programsko opremo izdelovati sami. Ekspertni sistem za postavljanje diagnoze in terapijo bolnika na ladji ter upravno in pravni podsistem sta globalnega značaja. V kolikor tu ne bi pohiteli z razvojem, nas bodo večji proizvajalci zlahka izrinili iz tržišča.

Omenjeni štirje podsistemi uporabljajo manjše računalniške sisteme, take kot jih izdelujemo pri nas. Vendar je potrebno pripomniti, da morajo ti sistemi zadovoljevati pogoje delovanja na ladjah. Delovati bi morali pri temperaturah od 0 do 45 stopinj C, pri visoki vlažnosti, slanosti, dimu in prahu. Sistemi bi morali delovati pri zibanju ladje v plovbi in večjih spremembah električne napetosti v pristaniščih. Končno bi morali delovati še pri vibracijah, katere ladji v plovbi povzročajo udarci valov in delovanje glavnega stroja.

Sklenimo sedaj zadnje ugotovitve. Cena tuje programske opreme, ki je izdelana za tip ladje ali poslovanja, torej za manjše količine proizvodov, je bistveno višja od domače cene. Pri nas imamo strokovnjake za izdelavo tovrstne programske opreme. Kar zadeva zahteve, katere so postavljene aparaturni opremi za delovanje na ladjah, je problem rešen, ker taki sistemi delujejo. V kolikor naša aparaturna oprema ne bi zadoščala postavljenim zahtevam, bi jo izboljšali ali le delno dokupili v tujini. S tem bi zaposlili naše strokovnjake, ladjarjem pa znižali stroške za opremo, pri čemer bi bili stroški dinarski. Ko bi se domači podsistemi pojavili na naših ladjah, bi bili pod istimi prizvodnimi pogoji hitro konkurenčni tudi na zunanjem tržišču.

5. RAČUNALNIŠKO TOVORJENJE LADIJ

Oglejmo si sedaj potrebo in namen računalniškega sistema za tovorjenje ladij in osnovna načela za njegovo izvedbo.

5.1. Potreba in namen računalniškega tovorjenja ladij

Potreba tovorjenja ladje s pomočjo računalnika je za neke tipe ladij postala ekonomska nujnost.

Integralni transport je v pomorske prevoze uvedel kontejnerje. Tako se je razvil poseben tip ladij za prevoz kontejnerjev. Nalaganje kontejnerjev na tako ladjo in razlaganje kontejnerjev iz nje poteka v pristaniščih zelo hitro. Pri tem so za varnost ladje potrebni obširni izračuni njene stabilnosti. Te izračune mora izvršiti ladijska posadka. Edino ona je po določbah zakonov odgovorna za varnost ladje. Te izračune za kontejnersko ladjo pa lahko v realnem času izvrši posadka le s pomočjo računalnika. Ekonomičnost in hitrost poslovanja kontejnerske ladje pa je odvisna tudi od pravilne izbire tovora zanjo in njegove razporeditve v pristanišču. Zato vršijo predhodne plane tovorjenja kontejnerske ladje, prav tako z računalnikom, luška ali ladjarska podjetja.

Posadke velikih tankerjev za prevoz tekočih goriv morajo izvrševati nekoliko manj obsežne izračune stabilnosti. Imajo pa težave z odpiranjem in zapiranjem ventilov ter pogonom črpalk za tovorjenje ali iztovorjenje goriv, ki so na taki ladji na večjih razdaljah. Prihranke v času in denarju so beležili tisti ladjarji, ki so na take ladje uvedli računalniško vodene sisteme, kateri poleg izračunov stabilnosti tudi računalniško daljinsko krmilijo ventile in črpalke pri tovorjenju in iztovorjenju goriv.

Znižane voznine v svetovnem pomorskem prometu narekujejo zmanjševanje dragih ladijskih posadk. Da bi zmanjšana posadka lahko opravila vsa tekoča dela, potrebuje za izvrševanje del višjo tehnologijo. Tako se računalniki za tovorjenje ladij danes pojavljajo na vseh tipih ladij.

Namen računalniškega tovorjenja ladje je torej hitrejše opravljanje standardnih izračunov. To so izračuni vgrezov ladje, trima, ter prečne in dinamične stabilnosti. Hitrejše računanje omogoča izračun večih variant in izbiro najboljše. Novejše ladje so večje in ekonomičneje grajene. Zato so pri njih potrebni dodatni izračuni dovoljenih upogibnih momentov in strižnih sil. Vsem tem izračunom sledi računalniški izpis podatkov in rezultatov, ter risanje plana tovora. Končno spravimo vse podatke in rezultate za vsako potovanje v datoteko.

5.2. Zahteve pri računalniškem tovorjenju ladij

Da bi računalniški sistem za tovorjenje ladij opravičil svoj namen, mora izvrševati izračune bistveno hitreje od dosedanjih tabelaričnih postopkov. To postavlja izvedbi sistema sledeče zahteve:

-Sistem mora biti avtonomen. To pomeni, da operater vnaša v računalnik samo najnujnejše meritve in podatke. Sistem torej izloča vsakršno uporabo grafov ali tabel.

-Začetne vrednosti spremenljivk vstavlja sistem sam.

-Menu-ji morajo omogočati tako izbiro podprogramov in njim pripadajočih strani, da je vnos najnujnejših skupin spremenljivk najhitreje izvršen.

-Na oba načina vstavljene spremenljivke shranimo. Shranjene morajo biti tako, da se po men-

javi strani zopet prikažejo na ekranu. Po spremembi poljubne spremenljivke morajo biti vse spremenljivke in njim pripadajoče vrednosti dostopne za zaključni račun.

-Obstojati mora možnost, da z enim ukazom spremenimo vrednosti določeni značilni skupini spremenljivk in njim pripadajočih vrednosti.

-Med vnosom dveh zaporednih spremenljivk se morajo delni podprogrami izvrševati v realnem času. Ta je manjši od ene sekunde.

-Zaradi izvrševanja delnih podprogramov v realnem času morajo biti njihovi algoritmi časovno učinkoviti. Program pa mora biti tudi prostorsko učinkovit. Mejo med obema učinkovitostima določa predpisani realni čas.

Nadalje mora sistem zadovoljiti še sledeče zahteve:

-Točnost rezultatov mora biti enaka ali večja kot pri sedanjih tabelaričnih postopkih.

-Programska oprema sistema mora biti zaščitena proti uporabi, katera ni v skladu s predpisi o tovorjenju ladij.

Če sistem ne bi izpolnjeval slednjih dveh zahtev, izračune sistema pomorski uradi ne bi priznali.

-Sistem mora biti prijazen do uporabnika.

-Sistem mora dajati zvočna in pisana opozorila. Z njimi opozarja operaterja na nepravilnosti pri tovorjenju kot so pretovorjenost ladje, slaba stabilnost in drugo.

-Izpis dokumentacije mora biti onemogočen v primeru, ko tovorjenje ni v skladu s predpisi.

-Terminologija pri interaktivnem delu mora biti v angleškem jeziku.
V nasprotnem bi bil nadzor nad tovorjenjem ladje in pripadajočo dokumentacijo otežan tujim strokovnjakom v tujih pristaniščih.

5.3. Osnovna načela tovorjenja ladje

Algoritmi za tovorjenje ladij so zasnovani na načelih mehanike, pri čemer privzamemo ladjo kot elastični nosilec.

Pri vzdolžni stabilnosti računamo vgreze ladje in vzdolžni nagib ladje (trim). Te vrednosti dobimo iz vzdolžnih momentov, katera povzročata sili vzgona in teže celotne ladje na njima pripadajočima ročicama, merjenima od sredine ladje. Ročica, na kateri deluje vzgon, je funkcija vgreza ladje in je izračunana iz ladijskih linij. Ročico na kateri deluje teža ladje izračunamo, če vzdolžne momente, katere povzročajo vse teže na ladji, delimo s težo celotne ladje. Težo celotne ladje predstavlja vsota teže same ladje, mrtvih tež, tež v tankih goriv, maziv, balasta, pitne vode in tež tovora. Vzdolžne momente omenjenih tež pa dobimo, če vsako težo pomnožimo s pripadajočo razdaljo od njenega težišča do sredine ladje.

Pri prečni stabilnosti ponovno računamo moment, katerega povzročata dvojici sil vzgona in teže celotne ladje. Ta moment računamo zato, ker on vrača nagibajočo se ladjo v stabilni položaj. Iste račune momentov, katere smo prej vršili v vzdolžni ravnini ladje, vršimo sedaj v njeni prečni ravnini. Pri tem teže ostajajo iste, ročice pa merimo od kobilice ladje. Te ročice bomo v dalnjem tekstu imenovali višine težišč ladijskih prostorov iznad kobilice. Če vsoto vseh nastopajočih momentov po višini delimo s težo celotne ladje, dobimo višino težišča ladje. Vektor vzgona ladje seka prečno simetralo ladje v točki, katero imenujemo metacenter. Višina metacentra nad kobilico ladje je važna funkcija vgreza, katero se izračuna iz ladijskih linij. Razdaljo med težiščem ladje in metacentrom imenujemo metacentrična višina. Metacentrično višino končno popravimo za vpliv svobodnih površin. Tako izračunana in popravljena metacentrična višina predstavlja mero za stabilnost ladje. Njena vrednost mora ustrezati predpisom za tovorjenje ladij.

5.4. Algoritmi za tovorjenje ladij

Zelo zgoščen opis osnovnih načel za tovorjenje ladij v prejšnjem poglavju je imel le ta namen, da ugotovomo, da se algoritmi za tovorjenje ladij načeloma sestojajo iz treh delov.

Kot prvo moramo poznati oddaljenost prijemališča vzgona od sredine ladje, nato višino metacentra in še druge vrednosti. Vse te vrednosti so funkcije, katere so izračunane iz ladijskih oblik, podane pa so za argument vgreza.

Kot drugo moramo poznati volumne ladijskih tankov za gorivo, maziva, balast in pitno vodo. Volumni in specifične teže določajo teže tekočin v tankih. Za računanje momentov moramo poznati položaj težišč tankov. To pomeni, da moramo poznati razdaljo težišč od sredine ladje in višino težišč iznad kobilice. Sredinski tanki v dvojnem dnu ladje imajo največkrat obliko kvadra. Bočni tanki v dvojnem dnu imajo sicer ravne stene, razen dveh, kateri sledita oblike dna in boka ladje. Tanki v bližini strojnega prostora so zaradi prostorske izkoriščenosti često dograjevani. Višinski tanki leže večinoma ob boku ladje, zato njihove oblike sledijo ukrivljenosti boka in odlivnega dela krova. Geometrijsko najbolj nepravilne oblike imajo tanki na pramcu in krmi ladje. Ti tanki so često še dograjeni. Na koncu velja pripomniti, da tanki niso vedno polni. Tako moramo vršiti izračune volumna in položaja sistemnega težišča v funkciji višine tekočine (sonde) za nepravilna geometrijska telesa.

Nekoliko lažje je računati volumne in položaje sistemnih težišč za ladijska skladišča. Ta so pravilnejših oblik, vendar so v njih pregrade, ventilatorji in rebra ladje. Tako je v istem ladijskem skladišču volumen in položaj sistemnega težišča različen za razsute tovore, generalni tovor ali kontejnerje.

Ko ladijska posadka računa stabilnost ladje, gornje izračune ni sposobna izvesti. Ti izračuni so izvršeni v ladjedelnicah. Obliko ladijskih reber je v primerjavi z vodnimi linijami matematično lažje izraziti. Zato v ladjedelnicah vršijo račune volumnov in sistemnih težišč ladijskih prostorov z numerično integracijo v glavnem v vzdolžni smeri ladje.

Tako so izračunane funkcije oddaljenosti prijemališča vzgona od sredine ladje, višina metacentra ter druge za argument vgreza ladje, volumni in položaji sistemnih težišč tankov za argumenta sonde in trima, volumni in položaji sistemnih težišč za tovorna skladišča pa glede na posamezne vrste tovorov. V tovornih skladiščih je te vrednosti potrebno popraviti, če so skladišča s tovorom le delno zapolnjena.

Vse te funkcije za navedene argumente izdelajo v ladjedelnicah v obliki grafov. Grafi niso natančni, delo z njimi pa je zamudno. Zato so omenjene funkcije v novejšem času ustrezneje prikazane s tabelami. Množica tabeliranih

funkcij za določeno ladjo je zbrana v 'Knjigi stabilnosti'. Knjiga stabilnosti je danes s strani pomorskih uradov edini priznani dokument za pravilno tovorjenje ladje.

Ko smo dobili iz Knjige stabilnosti vse potrebne podatke, nam končno ostane še zadnji, tretji del izračuna vzdolžne in prečne stabilnosti. Ostalo nam je obilo množenj za izračun też in momentov, seštevanj też za izračun celotne teże ladje, ter končno nekaj deljenj in odštevanj. Ti zadnji algoritmi za tovorjenje ladij zahtevajo torej le štiri osnovne računske operacije.

Poglejmo sedaj, kako je s tovorjenjem ladje s pomočjo računalnika. Če naj računalniški sistem za tovorjenje ladij opraviči svoj namen, mora izračune izvrševati hitro in točno. To pa izključuje zamudno uporabo grafov ali tabel. Računalniški sistem mora biti torej avtonomen, vse izračune mora izvrševati sam. Operater dela samo z računalnikom in pri delu z njim ne uporablja Knjige stabilnosti.

Zahteva po avtonomnosti računalniškega sistema za tovorjenje ladij postavlja izvedbi sistema nalogo ponavljanja ladjedelniških izračunov. Zato moramo poznati matematične izrażave oblik ladje v prečni ali vodoravni ravnini v funkciji argumenta v vzdolžni ali navpični osi. Le tako lahko izvršimo numerične integracije z ustreznim korakom argumenta. Tako dobljeni rezultati bi zadoščali zahtevi točnosti sistema. Izkaże pa se, da taka računalniška numerična integracija v zankah še zdaleč ne zadošča zahtevi dela v realnem času tudi pri delu s hitrimi in večjimi računalniki.

Na sedanji stopnji razvoja računalniških sistemov za tovorjenje ladij pravzaprav mnogo hitreje izvršujemo račune, katere danes vršimo na osnovi podatkov iz tabel Knjige stabilnosti. Na tej stopnji se nam ponuja možnost pridobitve veliko hitrejših računalniških algoritmov.

Če bi si ogledali grafe funkcij pripadajočih argumentov v Knjigi stabilnosti, bi najprej opazili, da so vse te funkcije zvezne. Končno bi opazili, da so te funkcije večinoma monotono naraščajoče funkcije, le izjemoma so to monotono naraščajoče in padajoče funkcije ali obratno.

Prva možnost, ki jo sedaj imamo, je ta, da funkcije iz Knjige stabilnosti vnesemo s programom v pomnilnik računalnika. Pri tem funkcijske vrednosti tvorijo člene seznama. Korak argumenta moramo izbrati tako, da zadostimo prostorski učinkovitosti in točnosti algoritma. Pri vnosu funkcijskih vrednosti za konstanten korak argumenta dobivamo funkcijske vrednosti za tabelirane in netabelirane vrednosti argumentov z linearno interpolacijo. Ta algoritem je torej računalniška inačica sedanjih tabelaričnih postopkov računa stabilnosti. Tudi pri njih iščemo funkcijsko vrednost za netabeliran argument z linearno interpolacijo.

Drugo možnost izrażave takih zveznih monotonih funkcij nudijo statistične metode. Učinkovita se izkaże metoda najmanjših kvadratov. Tako lahko večino funkcij iz Knjige stabilnosti izrazimo z regresijskimi polinomi, le malo pa z regresijskimi eksponentnimi funkcijami. Koeficiente regresijskega polinoma izračunamo iz vrednosti koordinat točk funkcije. Zahtevi točnosti algoritma zadostimo, če izračunamo koeficijente regresijskega polinoma iz zadostnega števila koordinat in z izbiro ustrezne stopnje polinoma.

Izkaże se, da so algoritmi na osnovi regresijskih polinomov časovno učinkovitejši od algoritmov na osnovi seznamov. Oboji pa zadoščajo zahtevi o delu v realnem času.

Potrebno je pripomniti, da so funkcijske vrednosti v Knjigi stabilnosti zaokrožene na zadnjem mestu. To povzroča zaokrožitveno napako pri izračunanih vrednostih, računanih z obemi algoritmi. Tako iznaša napaka na zadnjem mestu pri seznamih do +1, pri regresijskih polinomih pa med -1 in +1. Omenjene napake bi bilo možno odpraviti, če bi bile funkcijske vrednosti v Knjigi stabilnosti podane na dve decimalni mesti več.

Za današnjo stopnjo razvoja računalniških sistemov za tovorjenje ladje smo na ta način dosegli dvoje:
-Prvič smo dobili časovno in prostorsko učinkovite algoritme.
-Kot drugo pa so ti algoritmi zasnovani na vrednostih iz Knjige stabilnosti. Tako ni vzroka, da bi tako narejen sistem pomorski uradi ne priznali.

Pri tako izvedenem računalniškem sistemu za tovorjenje ladij bo v bližnji bodočnosti dobila Knjiga stabilnosti isti pomen, katerega imata danes magnetni kompas in astronomske metode. Knjigo stabilnosti bomo za tovorjenje ladij uporabljali samo v slučaju okvare računalnika.

Na višji stopnji razvoja računalniških sistemov za tovorjenje ladij ne potrebujemo le funkcij iz knjige stabilnosti za argument trima, temveč tudi za argument prečnega nagiba (list-a). Pred algoritma na osnovi seznama ali regresije se tu ne postavlja vprašanje časovne, temveč prostorske učinkovitosti. Postavlja se vprašanje, če ne bi bile na tej stopnji prostorsko učinkovitejše metode numerične integracije. Trdimo lahko le, da bi bile časovno neučinkovite. Časovno učinkovitost pa bi dosegli z integracijami na kratkih intervalih pri poznanih funkcijskih vrednostih na spodnjem delu intervala. To pa je zopet v škodo prostorske učinkovitosti. Rešitve tega problema ta članek ne obravnava.

5.5. Učinkovitost algoritmov

V tem poglavju bomo funkcije iz Knjige stabilnosti na kratko imenovali funkcije.

V prejšnjem poglavju smo omenili, da so regresijski polinomi časovno najučinkovitejši algoritmi.

Posamezno funkcijo je pri zahtevani točnosti večkrat možno izraziti z enim regresijskim polinomom do 10 stopnje. Če v računalniškem programu zapišemo polinom v obliki

$$f(x) = a_n * x^n + a_{n-1} * x^{n-1} + \ldots\ldots\ldots + a_0$$

se bo pri visokih stopnjah polinoma računalniški čas bližal zgornji meji realnega časa. To pa zato, ker za en argument izračunavamo več funkcijskih vrednosti, katere so izražene s takim zapisom polinomov. Če pa v programu zapišemo polinome v obliki

$$f(x) = ((..(a_n * x + a_{n-1}) * x + a_{n-2}) * x + \ldots \ldots\ldots + a_0$$

nam računalniški čas ne bo delal težav. Prvi zapis polinoma ima namreč kvadratno časovno učinkovitost v primerjavi z drugim zapisom, ki ima linearno časovno učinkovitost. Zaradi

časovne učinkovitosti zato v programih načelno zapisujemo polinome v drugi obliki. Prostorsko učinkovitost programskega zapisa polinoma izboljšamo tako, da decimalke njegovih koeficientov okrajšamo do meje, katera zadošča zahtevani točnosti izračunanih funkcijskih vrednosti.

Če ima funkcija singularne točke ali točke obračajev, je ni možno izraziti z zahtevano točnostjo z enim regresijskim polinomom, četudi bi bila njegova stopnja višja od desete. V takih primerih s pogojnimi stavki v programu funkcijo med omenjenimi točkami izrazimo odsekoma, v večini primerov s polinomi nižjih stopenj. Pogojni stavki in polinomi nižjih stopenj povečajo časovno učinkovitost. Prostorska učinkovitost se seveda zmanjša, vendar ne presega prostora v pomnilniku, katerega bi zavzela ista funkcija, če bi jo zapisali v obliki seznama.

Kadar funkcijo ne moremo izraziti z regresijskim polinomom, ali je ne moremo izraziti z zahtevano točnostjo, jo izrazimo s seznamom. S seznami pa izražamo tudi tiste funkcije, pri katerih je potrebna pri računu stabilnosti visoka točnost. V prejšnjem poglavju smo namreč videli, da napaka zaokrožitve vpliva na izračun funkcijske vrednosti manj, če je funkcija izražena s seznamom.

Pri čitanju kratkih seznamov je računalniški čas zadovoljiv. Pri čitanju dolgih seznamov pa precej preseže realni čas. V takem primeru moramo seznam razdeliti na podseznamе. S pogojnimi stavki v programu pri določeni vrednosti argumenta računalnik čita samo en podseznam. Na ta način časovno učinkovitost nastavimo na poljuben čas.

Omenjeni postopek pa seveda zmanjšuje prostorsko učinkovitost. Ker so funkcijske vrednosti večinoma racionalna števila z enakim številom decimalnih mest, jih vpišemo v sezname kot cela števila tako, da jih predhodno pomnožimo z ustrezno potenco števila 10. Tako so v seznamu prikazana z označeno točnostjo INTEGER in ne SHORT, kar izboljša prostorsko učinkovitost. Časovna učinkovitost pri tem ni zmanjšana, saj samo rezultat delimo z ustrezno potenco števila 10.

Kot bomo videli v naslednjem poglavju je uporabniški program računalniškega sistema za tovorjenje ladje sestavljen iz glavnega programa in podprogramov. Podprogrami pa so sestavljeni iz modulov. Velikost modula določa velikost strani zaslona ali zahtevana hitrost izvajanja izračunov. Izberimo sedaj stran zaslona, na katero vnašamo n argumentov. Ob vnosu vsakega argumenta delni podprogram izračuna temu argumentu pripadajoče fukcijske vrednosti. Privzemimo, da so v delnih podprogramih funkcije izražene s seznami, katerih časovna učinkovitost je na meji realnega časa. Sedaj želimo spremeniti le m-ti argument in s tem njemu pripadajoče funkcijske vrednosti. Predno ta argument dosežemo s kurzorjem, smo zapravili m-1 sekund. Uvedimo sedaj v delne podprograme majhne spremembe. Pred začetkom vsakega delnega podprograma dodajmo novo spremenljivko, katera naj dobi vrednost starega argumenta. Nato vnesimo novi argument. Delnim podprogramom dodajmo sedaj pogojni stavek. Ta naj predpiše brezpogojni preskok preko delnega podprograma v primeru, da sta stara in nova vrednost argumenta enaki. Na ta način smo prihranili m-1 dragocenih sekund računalniškega časa, prostorsko učinkovitost programa pa smo le neznatno zmanjšali.

Opisali smo nekaj načinov izboljšav časovne in prostorske učinkovitosti algoritmov. Pripomnimo, da mejo med obema učinkovitostima določa predpisani realni čas ter tip aparaturne opreme in njenega operacijskega sistema. Ponovimo še, da so algoritmi na osnovi regresijskih polinomov hitrejši, algoritmi na osnovi seznamov pa počasnejši, toda točnejši. Izkaže pa se, da je pri najbolj zahtevnih funkcijah za oba tipa algoritmov prostorska učinkovitost enaka.

5.6. Opis sheme programa

Uporabniški program računalniškega sistema za tovorjenje ladij mora izpolniti vse zahteve, katere so navedene v poglavju 5.2. Pri izdelavi programa dajemo do meje dela v realnem času popolno prednost časovni učinkovitosti programa. Šele nato posvetimo vso pozornost njegovi prostorski učinkovitosti. Izkaže se, kot bomo videli v naslednjem poglavju, da so taki programi zelo obširni.

Da bi bil program časovno učinkovit, ne zadostuje, da vsebuje samo časovno učinkovite algoritme. Program mora biti tudi ustrezno strukturiran. Zato je sestavljen iz glavnega programa, podprogramov z moduli in datotek.

5.6.1. Glavni program

Sestavljata ga glava in telo programa.

Glavo programa sestavljajo globalni ukazi, naslov programa, označitev točnosti spremenljivk s čimer povečamo prostorsko učinkovitost in glavni menu. Menu mora biti prijazen do uporabnika. Ob vnosu nepravilnega argumenta dobimo zvočno in pisano opozorilo, kursor pa se vrne na mesto nepravilno vnešenega argumenta. S tem je onemogočen premik teksta menu-ja po zaslonu. Pri tovorjenju ladje velikokrat vnašamo ali spreminjamo le neko značilno grupo podatkov in takoj preidemo k zaključnemu računu stabilnosti. Tako značilno grupo podatkov vsebuje podprogram. Namen glavnega menu-ja je, da omogoča hiter dostop do takih podprogramov. Poleg tega nam glavni menu prikaže osnovne ladijske podatke, odpira in briše datoteke, ter vnaša v datoteke začetne vrednosti spremenljivk.

V telesu programa se nahajajo rutine za izvajanje operacij katere se večkrat ponavljajo in modul z delnimi podprogrami, kateri omogočajo izvajanje menu-ja.

5.6.2. Podprogrami

Ločimo dve vrsti podprogramov. Prvo grupo podprogramov tvorijo tisti podprogrami, v katerih posamezen podprogram združuje značilno grupo podatkov. To so podprogrami za tovor, goriva, maziva, balast in pitno vodo. Drugo grupo podprogramov tvorijo podprogram za zaključni račun stabilnosti, podprogram za izpis podatkov in podprogram za risanje plana tovora. Vsaka grupa podprogramov ima svojo značilno strukturo programa.

Kot primer strukturiranosti prve grupe podprogramov si oglejmo podprogram za tovor.

Glavo podprograma tvorijo lokalni ukazi, označitev točnosti spremenljivk in menu podprograma. Menu podprograma omogoča izbiro strani zaslona. Število strani in vsebino vpisov na njih določata število vrstic na zaslonu in željena hitrost izvajanja računskih operacij. Tako ima podprogram za tovor tri strani. To so strani za tovor na krovu, tovor v medkrovju in tovor v spodnjih skladiščih ladje. Nadalje menu omogoča izvrševanje posebnih operacij. Te predstavljajo razstovorjenje vsake od teh treh skupin tovorov z enim ukazom. To povečuje časovno učinkovitost dela s programom.

Sledi ukaz za povratek v glavni menu.

Telo podprograma sestavljajo rutine za ponavljajoče se operacije in moduli. Vsaki strani zaslona in vsaki posebni operaciji pripada lasten modul. Moduli so končno sestavljeni iz delnih podprogramov. Vsakemu argumentu pripada lasten delni podprogram. Moduli, ki pripadajo raznim stranem podprograma, imajo enako strukturo. Najprej se iz datotek izvrši čitanje začetnih ali že vnešenih argumentov. Sledi vnos novih argumentov ali sprememba posameznega ter procesiranje pripadajočih funkcijskih vrednosti. Sledita izpis argumentov in rezultatov v datoteke, ter povratek v menu podprograma. Pri modulih za posebne operacije čitanje odpade, procesiranje in izpis v datoteko pa se vrši ob zadanem pogoju z enim ukazom. Tako zasnovani moduli so avtonomni.

Avtonomnost modulov in strukturiranost programa omogočata, da lahko spremenimo samo en argument na poljubni strani. Po zaključitvi te strani so vsi podatki takoj dostopni za zaključni račun stabilnosti.

Kot primer strukturiranosti druge grupe podprogramov si oglejmo podprogram za zaključni račun stabilnosti.

Glavo programa tvorijo izpis naslova programa, lokalni ukazi in označitev točnosti spremenljivk.

Telo programa tvorijo trije moduli. Program prvega modula prečita iz datotek podatke, kateri so potrebni za račun stabilnosti. Sledi modul s programom postopkovno usmerjenega tipa, s katerim računamo stabilnost. V tretjem modulu se nahajajo delni podprogrami za opozorila v primerih nepravilnega tovorjenja ladje. Takemu opozorilu sledi abort podprograma in povratek v glavni program, izpis dokumentacije pa je onemogočen.

5.6.3. Robni pogoji

Šele na tem mestu lahko dokončno odgovorimo na vprašanje o problemu robnih pogojev. Za funkcije iz Knjige stabilnosti, katere so podane za argument sonde, bi na to vprašanje lahko odgovorili že prej. Funkcijske vrednosti teh funkcij so na krajiščih intervalov točno definirane. Drugače pa je pri funkcijah, katere so podane za argument vgreza. Te se namreč nahajajo v podprogramu za zaključni račun stabilnosti. Funkcijske vrednosti le teh so na spodnjem krajišču intervala točno določene z vgrezom, ki pripada lahki ladji. Problemi pa bi lahko nastopili v zgornjem delu intervala. Te robne probleme pa rešuje zgoraj opisani modul za opozorila pri nepravilnem tovorjenju ladje, katerim sledi abort podprograma. V tako zasnovanem programu ni več opaziti problemov robnih pogojev.

5.6.4. Datoteke

Glavni program odpira datoteke za značilne grupe podatkov. To so datoteke za shranjevanje argumentov, izračunanih tež in ročic, vrednosti svobodnih površin in rezultatov zaključnega računa. Začetne vrednosti vpisuje v datoteke glavni program. Spremenjene vrednosti vpisujemo v datoteke z moduli. Podatki iz datotek so vedno dostopni za zaključni račun, izpis dokumentacije ali risanje plana tovora.

Tako strukturiran program za računalniško tovorjenje ladje omogoča hitro, operativno in varno tovorjenje ladje.

5.7. Velikost sistema

Poskusni uporabniški program za računalniško tovorjenje ladje je bil izdelan za ladjo tipa Concord C. Te ladje imajo nosilnost 18.000 ton, osem skladiščnih prostorov in 20 tankov za goriva, maziva, balast in pitno vodo. Ta števila uvrščajo ta tip ladij v spodnji srednji velikostni razred ladij. Ladij tega tipa ima podjetje Splošna plovba Piran pet.

Poskusni program izpolnjuje vse zahteve, katere so za računalniško tovorjenje ladij opisane v poglavju 5.2. Program je strukturiran in vsebuje avtonomne module.

Na osnovi tega poskusnega programa in ekstrapolacije lahko ocenimo velikost in hitrost delovanja računalniškega sistema za tovorjenje ladij.

S stališča prostorske učinkovitosti programske opreme zavzema program za ladjo na ravni kobilici, tovorjenje generalnega tovora in izračune vzdolžne in začetne prečne stabilnosti 34 kB pomnilniškega prostora. Če programiramo funkcijske vrednosti še za argument trima, se program poveča na 100 kB.

Časovna učinkovitost takega programa je dvajsetkrat večja od sedanjih tabelaričnih postopkov. Ker so funkcije podane za argument sonde in trima, je na računalniški sistem možno priklopiti senzorje. Z njimi bi bila časovna učinkovitost sistema glede na tabelarične postopke povečana do petdesetkrat. Pravilna nastavitev senzorjev pa bi povečala prostorsko učinkovitost.

Če programu dodamo program za tovorjenje razsutih tovorov, se le ta poveča za dodatnih 16 kB. Če dodamo še program za krivuljo stabilnosti in izpis podatkov, potrebujemo dodatnih 17 kB programa. S temi dodatnimi programi ostane časovna učinkovitost celotnega programa ista.

Program za tovorjenje kontejnerjev bi zahteval dodatnih 17 kB. Pri tem pa bi časovna učinkovitost celotnega programa padla izpod zgoraj omenjenih časovnih učinkovitosti. Bistveno pa je, da je računalniški čas tovorjenja kontejnerjev manjši od realnega časa za dejansko tovorjenje kontejnerjev na ladjo, česar tabelarični postopki ne omogočajo.

Vsi zgoraj navedeni programi bi tvorili program velikosti 150 kB. Program te velikosti bi avtonomno izvajal vse izračune, katere lahko za ladjo tipa Concord C danes vrišimo na osnovi Knjige stabilnosti in tabel sondaž, razen računov upogibnih momentov in striŠnih sil.

Zgoraj omenjeni programski opremi bi morala pripadati ustrezna aparaturna oprema. Aparaturna oprema bi morala zadoščati tudi zahtevam za delovanje na ladji, katere smo opisali v poglavju 4. Pripadajoči tiskalnik bi moral izpisovati dokumente na formatu A4 s hitrostjo 80 znakov v sekundi.

Opisani računalniški sistem za tovorjenje ladje predstavlja sistem srednjega razreda za ladjo srednje velikosti. Velikost sistema pa se spremeni glede na drugačno velikost ladje ali želje naročnika.

Večinoma se danes nahajajo na ladjah manjši sistemi. Vgrajeni pa so tudi veliko večji sistemi. Nahajajo se na velikih tankerjih, kjer je tovorjenje in iztovorjanje tekočih tovorov v celoti računalniško krmiljeno.

Programska oprema velikih sistemov vsebuje še druge programe. Če dodamo programe funkcij za argument lista, le ti sami lahko dosežejo velikost do 600 kB. Dodati moramo programe za račun upogibnih momentov, strižnih sil, grafične prikaze v barvah, računalniško krmiljenje tovorjenja, programe za ukrepe v primeru prodora vode in programe za optimizacijo tovorjenja. Tako lahko programsko opremo velikih sistemov ocenimo na red velikosti MB. Pripadajoča aparaturna oprema zahteva zaslone visoke grafične ločljivosti, risalnike za format A3, A/D pretvornike in dobro senzorsko tehnologijo.

6. ZAKLJUCEK

Članek je opisal široko uporabo informacijskih in računalniških sistemov v sodobnem pomorskem prometu in ladjedelništvu. Prikazano je sedanje stanje in naše možnosti za vključitev v razvoj in izdelavo tovrstne sodobne tehnologije.

Naše možnosti so na področju že obstoječe komunikacijske tehnologije in nadalnjem razvijanju meteorološkega informacijskega sistema na Jadranu. Veliko zanimanje za nadaljevanje razvijanja komercialno-operativnega informacijskega sistema v podjetju za kontejnersko službo je sedaj pri slovenskem ladjarju.

Imamo možnost razvoja ekspertnega sistema za nego bolnika na ladji.

Velike možnosti pa imamo za izdelavo ladijskih informacijskih podsistemov, kateri zahtevajo le računalniško tehnologijo. To so podsistemi za administracijo, zaloge, upravno in pravni, ter podsistem za tovor.

Kot primer ponovimo primerjalne prednosti, katere imamo pred tujimi proizvajalci pri izdelavi opisanega računalniškega sistema za tovorjenje ladij. Domači ladjar ali ladjedelnica določita, na katero ladjo bo sistem postavljen in kakšna bo njegova velikost. Za kontejnersko ladjo bi račun verjetno pokazal, da se investicija za uvožen sistem izplača v enem letu. Nadalnji račun bi pokazal, da je programska oprema takega tipa sistema doma petkrat cenejša kot v tujini. Poleg tega je cena dinarska. Tu imamo torej dve prednosti. Prva je prihranek pri nabavi opreme za ladjarje in ladjedelnice. Druga pa je prodor in prodaja našega znanja na domače in tuje tržišče.

Po uveljavitvi naših proizvodov na domačem in tujem tržišču, bi z razumno politiko cen ustvarili akumulacijo za nadalnji razvoj in razširjeno reprodukcijo.

PARALELNI SUSTAV ZA RAZPOZNAVANJE ZNAKOVA

Izv. prof. dr. Nikola Pavešić, dipl. ing.
Fakulteta za elektrotehniko v Ljubljani
Izv. prof. dr. Slobodan Ribarić, dipl. ing.
VVTŠ KoV JNA, Zagreb

UDK: 681.3.05

U članku je opisana izvedba paralelnog sustava za raspoznavanje rukom pisanih numeričkih znakova. Sustav se temelji na teoriji dvodimenzionalnih prilagodjenih filtera i primjeni procesora viših hijerarhijskih razina. U prvom dijelu članka dane su teorijske osnove dvodimenzionalnih prilagodjenih filtera i izvorne definicije procesora različitih hijerarhijskih razina. U drugom dijelu članka opisana je izvedba visokoparalenog sustava za raspoznavanje koji za osnovu ima procesore hijerarhijske razine H_2 i H_1.

THE PARALLEL CHARACTER RECOGNITION SYSTEM: In this paper we describe realization of the parallel system for the recognition of handwritten numerical characters. The system is based on the theory of two-dimensional matched filters and application of processors from higher hierarchical levels. In the first part of the paper we desribe theoretical aspects of two-dimensional matched filters and we give original definitions of processors from different hierarchical levels. In the second part of the paper the realization of highly parallel pattern recognition system is presented. The system has processors from hierarchical levels H_1 and H_2 as the basic elements. The experimental results are given.

1. UVOD

U posljednjih desetak godina interes za područje digitalne obrade i raspoznavanje slika u stalnom je porastu. Taj interes je usmjeren u dva glavna pravca [1]:

a) poboljšanje vizualnog izgleda slike, odnosno pretvaranje slike u pogodniji oblik za interpretaciju od strane čovjeka,

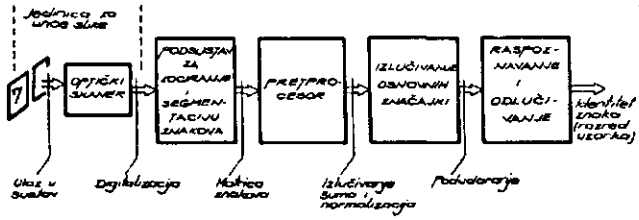
b) obrada i pretvaranje slike u oblik pogodan za automatsku percepciju i strojnu interpretaciju.

Pravac istraživanja opisan pod b) bio je uvjetovan prodorom računala u poslovne i industrijske sredine i prouzrokovao je razvoj uredjaja koji se obično nazivaju zajedničkim imenom - uredjaji za izravni unos podataka s dokumenata ili optički čitači znakova (OCR - Optical Characters Reader). Prema složenosti uredjaji OCR mogu se razvrstati u tri razreda [2]:

a) jednostavni uredjaji za čitanje kodova (npr. bar code) ili zapisa s magnetnom tintom,

b) uredjaji srednje razine složenosti koji se upotrebljavaju za izravan unos podataka s dokumenata, čekova, liječničkih recepata i sl. Skup ulaznih znakova je tipiziran [3].

c) Složeni uredjaji za raspoznavanje rukom pisanog teksta (npr. brojeva, brojčano-slovčanih znakova, Katakana znakova i sl. [3]).



Slika 1. Blok-shema tipičnog sustava za raspoznavanje znakova

Slika 1 prikazuje blok-shemu tipičnog sustava za raspoznavanje znakova. On se sastoji od:

- jedinice za unos i digitalizaciju slike,
- podsustava za lociranje i segmentaciju znakova,
- pretprocesora,
- podsustava za izlučivanje osnovnih značajki,
- podsustava za klasifikaciju.

Složenost pojedinih komponenti sustava zavisi od njegove namjene: u jednostavnim uredjajima OCR sa malim skupom tipiziranih ulaznih znakova neke komponente (npr. pretprocesor) mogu biti izostavljene, dok u složenim sustavima za raspoznavanje pojedine komponente su izvedene tako da imaju bazu znanja i mehanizam za zaključivanje, odnosno predstavljaju ekspertne sustave [4],[5].

Upotreba sklopova izvedenih u tehnologiji visokog stupnja integracije (LSI) u konstrukciji uredjaja OCR poboljšala je omjer performansa/cijena i omogućila primjenu egzaktnijih metoda u postupku raspoznavanja. U članku je opisana izvedba paralelnog sustava za raspoznavanje rukom pisanih znakova. Postupak raspoznavanja temelji se na primjeni dvodimenzionalnih prilagodjenih filtera.

2. DVODIMENZIONALNI PRILAGODJENI FILTERI

H.C. Andrews [6] je teoriju L.G. Turina [7] o jednodimenzionalnim prilagodjenim filterima proširio na dvodidemzionalne prilagodjene filtere. Osnovna karakteristika tih filtera je prijenosna funkcija koja u odredjenoj točki izlazne funkcije linearnog sustava maksimizira omjer signal-šum (S/N).

U ovom odjeljku izvest ćemo prijenosnu funkciju za dvodimenzionalni prilagodjeni filter u slučaju kada se

ulazna funkcija sastoji od signala $f_I(x,y)$ kojem je aditivno primješan šum $n_I(x,y)$:

$$g_I(x,y) = f_I(x,y) + n_I(x,y). \quad (1)$$

Izlazna funkcija $g_o(x,y)$ sastoji se od signala $f_o(x,y)$, koji je prouzrokovan signalom $f_I(x,y)$ prisutnim na ulazu u sustav, i od šuma $n_o(x,y)$ koji je posljedica šuma $n_I(x,y)$ prisutnog na ulazu u sustav.

Pretpostavimo da je šum proces koji je stacionaran i ergodički u autokorelaciji $R_{n_I}(\tau,T)$ sa spektralnom gustoćom:

$$S_{n_I}(u,v) = \mathcal{F}[R_{n_I}(\tau,T)], \quad (2)$$

gdje je $\mathcal{F}$ Fourierov operator.

Izlazna spektralna gustoća slučajnog šuma $n_o(x,y)$ dana je izrazom:

$$N = \int_{-\infty}^{+\infty}\int S_{n_I}(u,v) \cdot |H(u,v)|^2 \, dudv, \quad (3)$$

gdje je H(u,v) prijenosna funkcija linearnog sustava. Ako je $F_I(u,v) = \mathcal{F}[f_I(x,y)]$ tada je Fourierova transformacija izlazne funkcije $f_o(x,y)$:

$$\mathcal{F}[f_o(x,y)] = \mathcal{F}[f_I(x,y)] \cdot H(u,v), \quad (4)$$

što u vremenskom prostoru odgovara konvoluciji:

$$f_o(x,y) = f_I(x,y) \circledast h(x,y).$$

Energija signala S u točki (ξ,n) dana je sa:

$$S = |f_o(\xi,n)|^2 = \left|\int_{-\infty}^{+\infty}\int F_o(u,v) \exp\{j(u\xi+vn)\} \, dudv\right|^2 \quad (5)$$

$$= \left|\int_{-\infty}^{+\infty}\int F_I(u,v) \, H(u,v) \exp\{j(u\xi+vn)\} \, dudv\right|^2$$

Upotrebom jednadžbe (5) i (3) dobivamo omjer signal-šum koji se treba maksimizirati:

$$\frac{S}{N} = \frac{\left|\int_{-\infty}^{+\infty}\int F_I(u,v) \, H(u,v) \exp\{j(u\xi+vn)\} \, dudv\right|^2}{\int_{-\infty}^{+\infty}\int S_{n_I}(u,v) \, |H(u,v)|^2 \, dudv} \quad (6)$$

Pomoću Schwarzove nejednadžbe slijedi da je omjer (6) maksimalan [6] ako je

$$H(u,v) = K \frac{\overline{F}_I(u,v) \exp\{-j(u\xi+vn)\}}{S_{n_I}(u,v)}, \quad (7)$$

gdje je K proizvoljna kompleksna konstanta i $\overline{F}_I(u,v)$ konjugirano kompleksna vrijednost spektra signala ulazne funkcije. Jednadžba (7) predstavlja optimalnu prijenosnu funkciju koja maksimizira omjer S/N u točki (ξ,n).

3. HIJERAHIJA PROCESORA

Procesori su važna komponenta digitalnog sustava za raspoznavanje. Oni gotovo izravno utječu na performansu sustava. U ovom dijelu članka predlažemo izvornu klasifikaciju procesora s obzirom na broj koraka koji su potrebni za računanje aritmetičkih i logičkih izraza.

Aritmetički izraz E možemo definirati kao pravilno tvoren niz sastavljen najmanje iz jedne od četiri aritmetičke operacije {+, −, *, /}, lijevih i desnih zagrada (ukoliko su potrebne) i atoma koji su konstante ili varijable [8]. Aritmetički izraz sastavljen od n atoma označavat ćemo sa E<n>.

Logički izraz L je pravilno tvoren niz sastavljen najmanje iz jedne logičke operacije, lijevih i desnih zagrada (ukoliko su potrebne) i atoma koji su logičke varijable ili konstante. Logički izraz L sastavljen od n atoma označavat ćemo sa L<n>.

Od izvedbe procesora zavisi broj koraka T koji je potreban za računanje aritmetičkih E<n> i logičkih L<n> izraza.

3.1 Procesor hijerarhijske razine H_0

Tvrdnja 1

Primjenom potrebnog broja procesora sa ulazima samo za dva atoma (procesor izvodi binarne i unarne operacije) aritmetički izraz E<n> može se izračunati u

$$T[E<n>] \geq \lceil \log_2 n \rceil \text{ koraka}, \quad (8)$$

gdje $\lceil x \rceil$ označava gornju cjelobrojnu vrijednost broja x. Dokaz tvrdnje 1 je trivijalan [8].

Tvrdnja 2

Broj potrebnih koraka za računanje logičkog izraza L<n> pomoću procesora s ulazima za samo dva atoma je

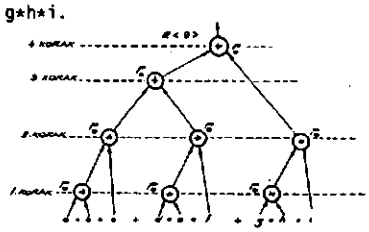
$$T[L<n>] \geq \lceil \log_2 n \rceil. \quad (9)$$

Definicija 1

Procesor koji izvršava osnovne aritmetičke i logičke operacije, te ima ulaze za dva operanda i zadovoljava tvrdnje 1 i 2 naziva se *procesor* p_0 ili procesor *hijerarhijske razine* H_0.

Procesor p_0 je komponenta većine današnjih računalskih sustava i predstavlja jednu od osnovnih značajki arhitekture računala koja se zasniva na von Neumannovom modelu računala.

Paralelno računanje nekog aritmetičkog izraza primjenom potrebnog broja procesora razine p_0 obično se prikazuje tzv. stablom redukcije po razinama. Na primjer, slika 2 prikazuje računanje izraza E<9> = a*b*c + + d*c*f + g*h*i.



Slika 2. Stablo redukcije po razinama za E <9> uz pomoć procesora p_0

3.2 Procesor hijerarhijske razine H_1

Jednovrsni aritmetički izraz $E_u<n>$ je pravilno tvoreni niz sastavljen samo od jedne vrste aritmetičkih operacija (ili +, ili −, ili *, ili /), lijevih i desnih zagrada (ukoliko su potrebne) i n atoma koji su konstante ili varijable.

Na sličan način možemo definirati jednovrsni logički izraz: $L_u<n>$ je pravilno tvoren niz sastavljen od jedne vrste logičkih operacija, lijevih i desnih zagrada (ukoliko su potrebne) i n atoma koji su logičke konstante ili varijable.

Tvrdnja 3

Procesor p_1 neka je takav da ima m ulaza; (m > 2) i izvodi m-arne operacije. Upotrebom potrebnog broja procesora p_1 jednovrsni aritmetički izraz izračunava se u:

$$T[E_u<n>] \geq \lceil \log_m n \rceil \text{ koraka.} \qquad (10)$$

Tvrdnja 4

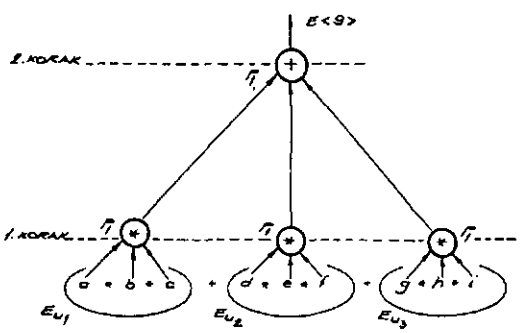
Broj koraka za računanje jednovrsnog logičkog izraza $L_u<n>$ pomoću potrebnog broja procesora p_1 je:

$$T[L_u<n>] \geq \lceil \log_m n \rceil . \qquad (11)$$

Definicija 2

Procesor koji izvodi jednovrsne aritmetičke i logičke operacije i ima m > 2 ulaza za atome (izvodi m-arne operacije), te zadovoljava tvrdnje 3 i 4, naziva se *procesor* p_1 ili *procesor hijerarhijske razine* H_1.

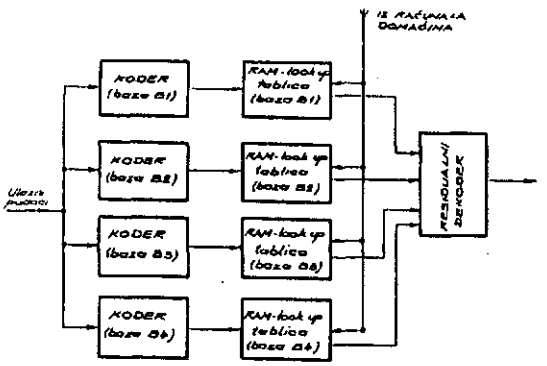
Primjeri izvedbe procesora hijerarhijske razine H_1 prikazan je u radu N. Konvarasa, gdje je opisan procesor za istovremeno zbrajanje m n-bitnih podataka [9]. Slika 3 prikazuje primjer računanja aritmetičkog izraza E<9> = a * b * c + d * e * f + g * h * i, gdje se a * b * c, d * e * f i g * h * i mogu promatrati kao tri jednovrsna aritmetička izraza E_{u_1}, E_{u_2} i E_{u_3}, a njihovi rezultati kao operandi za jednovrsni aritmetički izraz sastavljen od tri atoma.



Slika 3. Računanje E<9> uz pomoć procesora p_1

3.3 Procesor hijerarhijske razine H_2

Procesor hijerarhijske razine H_2 predstavlja neku vrstu „računskog demona". Namjerno smo upotrijebili izraz koji je koristio O.G. Selfridge za opisivanje postupka učenja u sustavima za raspoznavanje uzoraka [10], jer se procesor u hijerarhijskoj razini H_2 korjenito razlikuje od onoga što pod izrazom „procesor" podrazumijevamo u računalskim znanostima. Na primjer, slika 4 prikazuje inačicu procesora koji se djelotvorno upotrebljava u sustavu za obradu slika RADIUS (tvrtka Hyghes Research). Iako ovaj procesor nema sve značajke procesora hijerarhijske razine H_2 može nam poslužiti kao dobra ilustracija u kojoj mjeri procesori mogu odstupati od onoga što pod tim izrazom podrazumijevamo. Procesor (sl. 4) upotrebljava simbolički način predstavljanja brojeva (tzv. residualni brojevni sustav). Ulazni podatak se jednoznačno predstavlja kao ostatak dijeljenja sa bazama B1, B2, B3 i B4 gdje su baze izabrane kao prim brojevi (B1 = 31, B2 = 29, B3 = 23 i B4 = 19). Točnost predstavljanja brojeva je $B_1 \cdot B_2 \cdot B_3 \cdot B_4 = 2^{18 \cdot 6}$. Središnji dio procesora je memorija RAM koja se upotrebljava kao "look up" tablica za „računanje" funkcija f(a,b), gdje su a i b ulazni podaci. Procesor je „programabilan", odnosno računalo domaćin puni tablicu u zavisnosti od tipa funkcije f.



Slika 4. Procesor u sustavu RADIUS

Očigledno je da procesor u računalu RADIUS nema potpuno zbrajalo, sklop za posmak i druge sklopove koji su karakteristični za procesor u von Neumannovom računalu, medjutim on djelotvorno izvodi složene aritmetičke operacije kao što je npr. konvolucija s jezgrom od 5 × 5 elemenata [11].

Definicija 3

Procesor p_2 ili procesor hijerarhijske razine H_2 opisan je trojkom:

$$p_2 = (U, I, \mu), \qquad (12)$$

gdje je U skup raspoloživih atoma aritmetičkog ili logičkog izraza, I skup rezultata aritmetičkih ili logičkih operacija koje se izvode nad raspoloživim atomima i μ je funkcija:

$$\mu : U \to I.$$

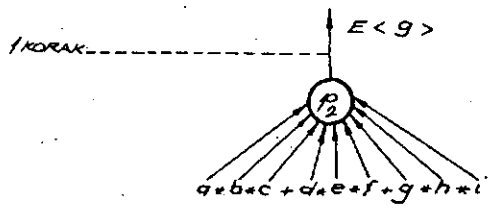
Funkcija μ predstavlja preslikavanje skupa raspolo-

živih atoma, u skladu s aritmetičkim ili logičkim izrazom, u skup rezultata.

Procesor p_2 u jednom koraku računa aritmetički ili logički izraz bez obzira na tip operacije (jednovrsna ili raznovrsna) i broj operanada n.

Primjeri izvedbi procesora p_2 su funkcionalna memorija [12] i 2-D procesorski segment u sustavu Cytocomputer [13].

Slika 5 prikazuje primjer računanja aritmetičkog izraza E<9> = a * b * c + d * e * f + g * h * i pomoću procesora p_2.



Slika 5. Računanje E<9> uz pomoć procesora p_2

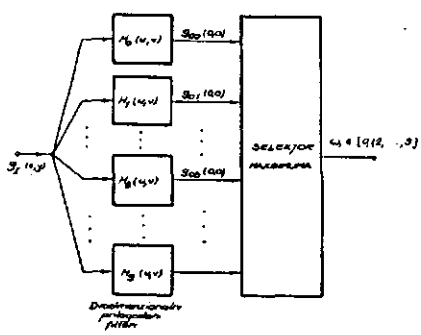
Procesori u razinama H_0, H_1 i H_2 imaju takve značajke da vrijedi slijedeće:

Ako neki procesor p ima svojstva hijerarhijske razine H_2, tada taj procesor ima i svojstva procesora razina H_1 i H_0.

Neka je procesor p iz razine H_2. Ograničimo funkciju μ samo na jednovrsne operacije i neka je m > 2. Tada procesor poprima značajke procesora razine H_1. Ako uvedemo i dodatno ograničenje m = 2 dobivamo procesor hijerarhijske razine H_0 (u skladu s definicijama 1-3).

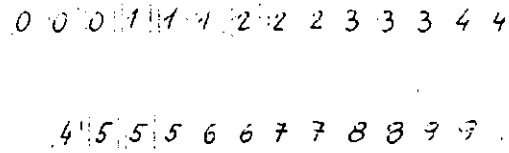
4. ORGANIZACIJA SUSTAVA ZA RASPOZNAVANJE

Sustav za raspoznavanje rukom pisanih numeričkih znakova oblikovan je pomoću deset dvodimenzionalnih prilagodjenih filtera. Svakom razredu uzoraka ω_i; i = 0,1, ...,9 priredjena je prijenosna funkcija $H_i(u,v)$. Slika 6 prikazuje organizaciju sustava. Svi prilagodjeni filteri imaju zajednički ulaz koji je ujedno i ulaz u sustav za raspoznavanje. Izlazi iz dvodimenzionalnih prilagodjenih filtera su spojeni s jedinicom za detekciju maksimalne vrijednosti.



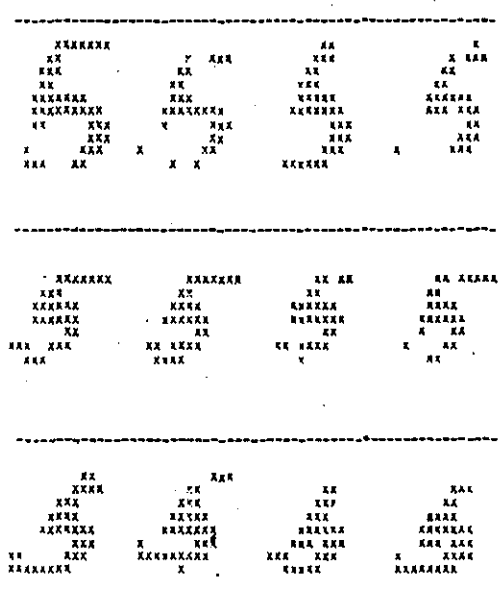
Slika 6. Organizacija sustava za raspoznavanje

Ulazni podaci - rukom pisane brojke - ispisani su na posebnim obrascima (sl. 7). Znakovi su napisani u kvadratima označenim svjetloplavim okvirima dimenzija 7 × 10 mm i u medjusobnoj udaljenosti dva milimetra.



Slika 7. Ulazni podaci napisani u obrascima

TV kamera s odgovarajućim upravljačkim sklopovima pretvara sliku u analogni signal, a digitalizator upravljan računalom pretvara sliku u matricu dimenzija 16 × 16 slikovnih elemenata. Analizom spektra i odredjivanjem graničnih frekvencija signala utvrdjeno je da dimenzije matrice 16 × 16 zadovoljavaju i da se u skladu sa Shannonovim teoremom ulazni signal može rekonstruirati [16]. Mnoge baze podataka sustava za raspoznavanje imaju slične dimenzije ulaznih uzoraka (npr. Highleymanova baza podataka ima matricu dimenzija 12 × 12 [3], a neki komercijalni sustavi imaju i manje dimenzije (npr. 9 × 7 [14])). Digitalizator kvantizira svaki slikovni element u 256 razina (8 bita). Analiza histograma pokazala je da su oni bimodalni, te su za opis uzoraka rukom pisanih znakova dovoljne samo dvije razine. Binarnu sliku dobili smo postupkom usporedjivanja s pragom p, gdje je prag p zavisan od osvjetljenosti slike, vrste i boje pisaljke te vrste i boje podloge. Slike smo binarizirali u odnosu na prag p koji se nalazi na minimumu izmedju dva vrha histograma [15]. Histogrami su dobiveni uz pomoć ulaznih uzoraka iz skupa za učenje. Slika 8 prikazuje binarizirane matrice ulaznih uzoraka.



Slika 8. Binarizirane matrice ulaznih uzoraka

Da bismo provjerili djelotvornost sustava za raspoznavanje, sustav smo simulirali na računalu CYBER 72/24. Simulacijski programi su napisani u Fortranu i sastoje se od devet nezavisnih modula tako da nam omogućava djelotvorno mijenjanje i eksperimentiranje sa sustavom za raspoznavanje [16]. Kao ulaz u simulator upotrebljeni su binarizirani uzorci.

Simulacija je podijeljena u dvije faze. Prva faza, koja odgovara fazi učenja sustava za raspoznavanje, izvršava se pomoću sedam programskih modula. Ti moduli odredjuju signalnu komponentu spektra ulazne funkcije, spektralnu gustoću šuma i prijenosne funkcije filtera. Zbog jednostavnosti simulacije prijenosne funkcije se odredjuju u frekventnom prostoru primjenom brze Fourierove transformacije (FFT).

4.1 Odredjivanje signalne komponente ulazne funkcije

Pomoću skupa uzoraka za učenje $\{g_I(x,y)\}$ odredili smo skup statističkih funkcija:

$$St_i(x,y) = P_i(g_I(x,y) = 1|\omega_i), \qquad (13)$$

za $x = 0,1,2,\ldots, 15$ i $y = 0,1,2,\ldots, 15$, gdje je $\omega_i \in \{0,1,2,\ldots, 9\}$ razred uzoraka.

Primjer funkcije $St_0(x,y)$ prikazan je na slici 9 i predstavlja vjerojatnost (u postocima) pojavljivanja jedinice u pojedinim elementima matrice uzorka 0.

Skup funkcija $St_i(x,y)$ se preslikava u skup signala $f_{Ii}(x,y)$ pomoću slijedeće relacije:

$$f_{Ii}(x,y) = 1 \text{ ako je } St_i(x,y) > q_i \quad \text{i}$$
$$f_{Ii}(x,y) = 0 \text{ ako je } St_i(x,y) \leq q_i\ , \qquad (14)$$

gdje je $q_i \in \{q_0, q_1, \ldots, q_9\}$ odredjen eksperimentalno.

Budući da je transformacijsko jezgro separatibilno i simetrično Fourierovu transformaciju, odnosno spektre signala $F_{Ii}(u,v)$; $i = 0,1,\ldots, 9$ dobivamo sljednim računanjem $F_{Ii}(u,x)$ i $F_{Ii}(u,v)$ pomoću brze Fourierove transformacije.

4.2 Odredjivanje spektralne gustoće šuma

Šum za pojedini ulazni uzorak odredjujemo kao razliku:

$$n^i_{Ik}(x,y) = g^i_{Ik}(x,y) - f^i_{Ik}(x,y), \qquad (16)$$
$$k = 1,2,\ldots, M_i,$$

M_i je broj uzoraka za učenje u razredu ω_i; $i = 0,1,2,\ldots,9$.

Spektralnu gustoću računamo pomoću izraza:

$$S^i_{n_I}(u,v) = \frac{1}{M_i}\sum_{k=1}^{M_i} |\mathcal{F}[n^i_{Ik}(x,y) \cdot w(x,y)]|^2 \qquad (17)$$

Pojedine funkcije $n^i_{Ik}(x,y)$ smo umnožili s Hammingovom funkcijom [17] zato da bismo umanjili Gibbonov utjecaj [18].

Pomoću Huangovog teorema [19] možemo Hammingovu funkciju napisati u obliku:

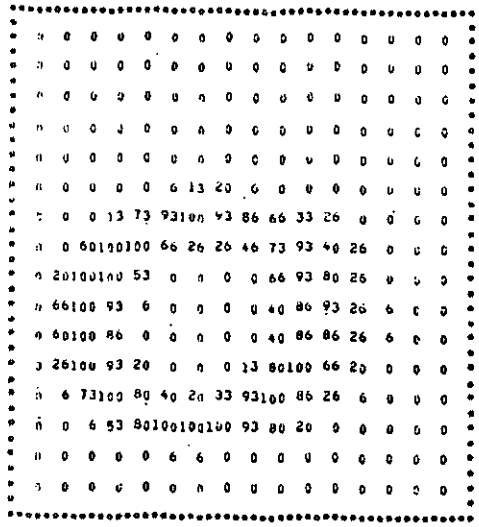
$$w(x,y) = 0{,}54 + 0{,}46 \cos \frac{2\pi}{16}\sqrt{x^2 + y^2}$$

za x i $y \in \{0,1,2,\ldots, 15\}$ i

$$w(x,y) = 0 \qquad (18)$$

za x i $y \notin \{0,1,2,\ldots, 15\}$.

STATISTICNA SLIKA VZORCA 0



Slika 9. Primjer funkcije $St_0(x,y)$

Izraz [17] računamo primjenom brzih postupaka za odredjivanje autokorelacijske funkcije [20],[21].

Za svaki razred uzoraka ω_i $i = 0,1,\ldots, 9$ možemo sada odrediti prijenosnu funkciju dvodimenzionalnog prilagodjenog filtra: ($\xi = \eta = 0$):

$$H_i(u,v) = \frac{\overline{F}_{Ii}(u,v)}{S_{n_{Ii}}(u,v)} \qquad (19)$$

Time je postupak učenja sustava za raspoznavanje završen.

4.3 Rezultati raspoznavanja

Druga faza simulacije odgovara procesu raspoznavanja. Na ulazu sustava dovodi se nepoznati uzorak. Preostala dva programska modula daju odziv filtera i razvrstavaju uzorak na temelju maksimalne vrijednosti odziva.

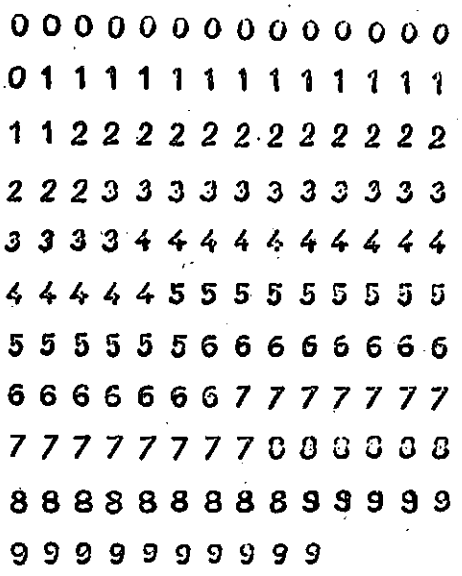
Opisani sustav za raspoznavanje smo testirali na četiri zbirke rukom pisanih numeričkih znakova.

Pokus I

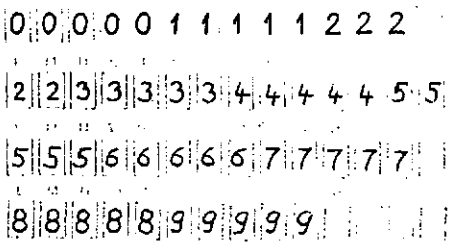
Prva zbirka sastoji se iz 150 numeričkih znakova. Brojke su pisane tušem i tehničkim pismom (sl. 10). Sustav je bez greške pravilno razvrstao uzorke.

Pokus II

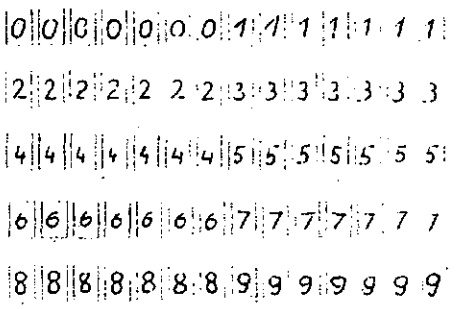
Druga zbirka sastoji se iz 50 numeričkih znakova koji su napisani kemijskom olovkom i „normalnim" pismom (sl. 11). Rezultati raspoznavanja su i u ovom slučaju 100% pravilno razvrstani.



Slika 10. Brojke pisane tušem i tehničkim pismom



Slika 11. Zbirka ulaznih podataka za drugi pokus



Slika 12. Dio zbirke ulaznih podataka za treći pokus

Pokus III

Sastavili smo zbirku iz 77 rukom pisanih numeričkih znakova. Znaci su pisani kemijskom olovkom. Neki od znakova nisu bili zaključeni (npr. 0 ili 9) (sl. 12). I u ovom slučaju rezultat raspoznavanja bio je 100%.

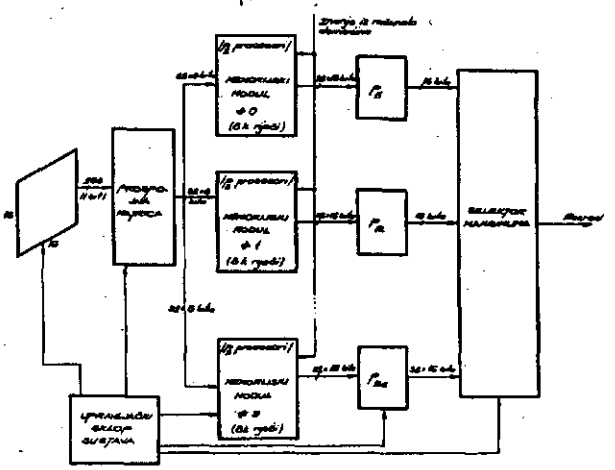
Pokus IV

Sastavili smo zbirku iz 132 znaka. Znakove smo pisali tako da smo dozvolili 20% promjene u visini i širini znakova (sl. 13). U ovom slučaju točno razvrstanih znakova bilo je 88,64%.

Slika 13. Dio zbirke ulaznih podataka za četvrti pokus

5. SINTEZA SUSTAVA ZA RASPOZNAVANJE

Slika 14 prikazuje blok dijagram sustava za raspoznavanje. Izvedba gore opisanog sustava za raspoznavanje jednostavnija je ako se zadatak raspoznavanja izvodi u vremenskom prostoru. Na temelju gore opisanog postupka simulacije - u prvoj fazi odredjene su prijenosne funkcije filtera $H_i(u,v)$ u frekventnom prostoru.



Slika 14. Blok-dijagram sustava za raspoznavanje

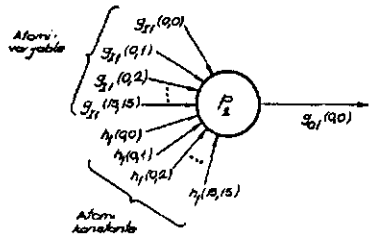
Vrijednosti izlazne funkcije dvodimenzionalnih prilagodjenih filtera g_{0i}, i = 0,1,2,..., 9 u točki (0,0) su odredjene izrazom:

$$g_{0i} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \mathcal{F}^{-1}\,[\overline{H}_i(u,v)]\; g_{Ii}(x,y), \qquad (20)$$

gdje $\mathcal{F}^{-1}$ označava inverznu Fourierovu transformaciju, a $\overline{H}_i(u,v)$ konjugirano kompleksnu vrijednost prijenosne funkcije $H_i(u,v)$ i N = 16.

Jednadžba (20) predstavlja, u stvari, konvoluciju $h(x,y) \circledast g_{Ii}(x,y)$ u točki (0,0).

Računalo domaćin (CYBER 72/24) u fazi inicijalizacije sustava za raspoznavanje prosljedjuje vrijednosti komponenata prijenosnih funkcija filtera sustava za raspoznavanje (sl. 14). Sustav može biti tako oblikovan da su komponente prijenosne funkcije h za pojedine filtere čvrsto upisane (PROM memorija). Medjutim, primjenom veze računalo domaćin - sustav omogućavamo veću prilagodljivost (programabilnost) sustava za raspoznavanje. Vrijednosti $g_{Oi}(0,0)$ i = 0,1,2,..., 9 mogu se dobiti u jednom koraku ako se upotrijebe procesori p_2 odnosno „računski demoni" (definicija 3), (sl. 15).



Slika 15. Procesor p_2

Na primjer, za računanje vrijednosti izlazne funkcije dvodimenzionalnog prilagodjenog filtera $g_{01}(0,0)$ procesor p_2 ima 256 ulaznih varijabli $g_{I1}(0,0)$, $g_{I1}(0,1),\ldots, g_{I1}(15,15)$ i isto toliko atoma - ulaznih konstanti $h_1(0,0)$, $h_1(0,1),\ldots, h_1(15,15)$. Procesor p_2 mogao bi se realizirati kao "look-up" tablica sa svim pohranjenim kombinacijama suma $h_1(x,y)$ budući da je $g_{I1}(x,y) \in \{0,1\}$ (slike ulaznih uzoraka su binarizirane).

Općenito, vrijednost izlazne funkcije $g_{Oi}(0,0)$ dobivala bi se tako da se na adresne ulaze memorije (u kojoj je pohranjena tablica) dovede binarna kombinacija koja odgovara vrijednosti $g_{Ii}(x,y)$; x = 0,1,2,..., 15 i y = 0,1,2,..., 15. Na nesreću, za takvu izvedbu potrebna je memorija vrlo velikog kapaciteta (2^{256} lokacija). Upravo zbog toga prilikom praktične izvedbe postupak računanja izraza (20) razdijelit ćemo u dva segmenta (sl. 16). U prvom se računaju djelne sume produkata $\Delta_k = h_i(x,y)\ g_{Ii}(x,y)$:

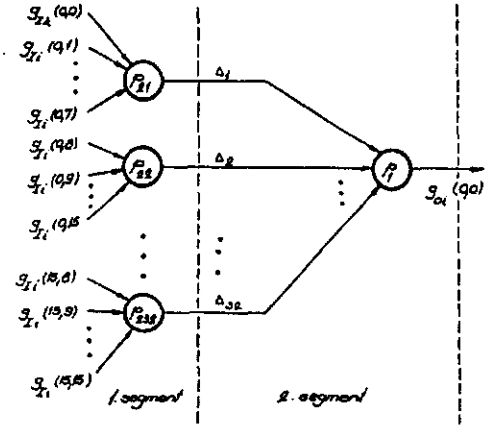
$$\Delta_k = \sum_{y=0}^{7} h_i(c, y + 8 \cdot \ell)\ g_{Ii}(c, y + 8 \cdot \ell), \qquad (21)$$

za k = 1,2,..., 32, gdje je $c = \lceil (k/2) \rceil - 1$ i $\ell = (k-1) \bmod 2$. $\lceil x \rceil$ označava gornju cjelobrojnu vrijednost x-a.

U drugom segmentu dobivamo $g_{Oi}(0,0)$, pomoću procesora p_1 (definicija 2), kao vrijednost jednovrsnog aritmetičkog izraza:

$$g_{Oi}(0,0) = \sum_{k=1}^{32} \Delta_k \qquad (23)$$

Procesor p_2 koji računa djelnu sumu produkata Δ_k



Slika 16. Računski modul s procesorima $p_2 - p_1$

izveden je kao "look-up" tablica RAM sa samo 256 lokacija. Ulazni operandi u procesor su $g_{Ii}(x,y)$, gdje je $x = c$ i $y = 1 + 8 \cdot \ell,\ldots, 7 + 8 \cdot \ell$, c i ℓ su definirani izrazom (22).

Atomi-konstante $h_i(x,y)$ i sve kombinacije njihovih suma (256 kombinacija) su pohranjene u procesoru. Na primjer, procesor p_2 koji u jednom koraku računa djelnu sumu produkta Δ_1 izveden je kao memorija sa slijedećim sadržajima:

Adresa (binarno)	Sadržaj
0 0 0 0 0 0 0 0	0
0 0 0 0 0 0 0 1	$h_1(0,0)$
0 0 0 0 0 0 1 0	$h_1(0,1)$
0 0 0 0 0 0 1 1	$h_1(0,0)+h_1(0,1)$
0 0 0 0 0 1 0 0	$h_1(0,2)$
0 0 0 0 0 1 0 1	$h_1(0,0)+h_1(0,2)$
.	.
.	.
.	.
1 1 1 1 1 1 1 1	$h_1(0,0)+h_1(0,1)+h_1(0,2)+\ldots$ $\ldots+h_1(0,7)$

Konstante $h_i(0,0)$, $h_i(0,1),\ldots, h_i(15,15)$ odredjene su u fazi učenja klasifikatora, a njihove djelne sume izračunate su i pohranjene u memoriju. Analize su pokazale [16] da je za zapis pojedine konstante dovoljna duljina riječi od dvanaest bita.

Adresa "look-up" tablice je, u stvari, osmorka koja se sastoji od vrijednosti slikovnih elemenata $g_{Ii}(x,y)$.

Vrijednost $g_{Oi}(0,0)$ dobiva se u jednom koraku upotrebom procesora p_1 hijerarhijske razine H_1. Arhitektura procesora koji istovremeno računa sumu m; (m>2), n-bitnih brojeva opisana je u radu N. Konvarasa et al. [9]. Slika 16 prikazuje organizaciju dvodimenzionalnog prilagodjenog filtera koji je realiziran u dva segmenta pomoću procesora p_2 i p_1. Ocijenimo kapacitet "look-up" tablica sustava za raspoznavanje znakova na bazi dvodi-

menzionalnih prilagodjenih filtera. Ukupni kapacitet memorije C potreban za izvedbu potrebnog broja procesora p_2 u sustavu za raspoznavanje je:

$$C = c_p \cdot n_p \cdot M \cdot d \text{ [bita]}$$

gdje je:

c_p - broj riječi memorije jednog procesora p_2 koji računa djelnu sumu produkta Δ_k,

n_p - broj procesora potrebnih za izvedbu jednog prilagodjenog filtra,

M - broj razreda uzoraka (broj dvodimenzionalnih prilagodjenih filtera),

d - duljina riječi izražena u bitovima.

Za sustav opisan u ovom radu C je:

$$C = 256 \times 32 \times 10 \times 16 = 10 \times 2^{17} \text{ bita} \qquad (24)$$

odnosno 84920 riječi duljine šesnaest bita.

Selektor maksimuma rješava se programski ili primjenom integriranih sklopova s tom funkcijom [23].

6. ZAKLJUČAK

U radu smo pokazali kako se teorija dvodimenzionalnih prilagodjenih filtera može primijeniti u izvedbi sustava za raspoznavanje znakova. Simulacija sustava za raspoznavanje pokazala je da je sustav djelotvoran posebno za tipizirane znakove ili rukom pisane znakove čije su varijacije po visini i širini manje od 20%. Upotrebom procesora iz viših hijerarhijskih razina prikazali smo model sustava koji je sačuvao visoki stupanj paralelizma svojstven ovom načinu raspoznavanja. U završnom dijelu članka prikazali smo putove praktične izvedbe paralelnog sustava upotrebom LSI komponenti.

7. LITERATURA

1. E.C. Lyons, Digital Image Processing: An Overview, Computer, Vol. 10, No. 8, August 1977, str. 12-14.
2. L.D. Harmon, Automatic Recognition of Print and Script, Proc. of the IEEE, Vol. 60, No. 10, October 1972, str. 1165-1176.
3. C.Y. Suen, et al., Automatic Recognition of Handprinted Characters - The State of the Art, Proc. of the IEEE, Vol. 68, No. 4, April 1980, str. 469-487.
4. D.A. Rosenthal, R. Baycsy, Visual and Conceptual Hierarchy: A Paradigm for Studies of Automated Generation of Recognition Strategies, IEEE Trans. on PAMI, Vol. PAMI-6, No. 3, May 1984, str. 319-325.
5. A.M. Nazif, M.D. Levine, Low Level Image Segmentation: An Expert System, IEEE Trans. on PAMI, Vol. PAMI-6, No. 5, September 1984, str. 555-577.
6. H.C. Andrews et al., Computer Techniques in Image Processing, Academic Press, 1970.
7. L.G. Turin, An Introduction to Matched Filters, IRE Trans, on Information Theory, June 1960, str. 311-329.
8. D.J. Kuck, A Survey of Parallel Machine Organization and Programming, ACM Computing Surveys, Vol. 9, March 1977, str. 29-60.
9. N. Konvaras et al., A Digital System for Simultaneous Addition of Several Binary Numbers, IEEE Trans. on Computers, Vol. C-17, October 1968, str. 992-997.
10. O.G. Selfridge, Pandemonium: A Paradigm for Learning, u Pattern Recognition (ed. L. Uhr), J. Wiley & Sons, New York, 1966.
11. G.R. Nudd, Concurrent Systems for Image Analysis, u VLSI for Pattern Recognition and Image Processing, (ed. K.S. Fu), Springer-Verlag, Berlin, 1984, str. 107-132.
12. P.L. Gardner, Functional Memory and Its Microprogramming, Implications, IEEE Trans. on Computers, Vol. C-20, No. 7.
13. R.M. Laugheed et al., Cytocomputers: Architectures for Parallel Image Processing, Proc. Workshop Picture Data Descr. and Management, Pacific Grove Calif. August 27-28, 1980, str. 281-286.
14. S.T. Tou, R.C. Gonzales, Pattern Recognition Principles, Addison-Nesley. Pub. Com., 1974.
15. A. Rosenfeld, A. Kak, Digital Picture Processing, Academic Press, New York, 1976.
16. N. Pavešić, L. Gyergyek, Character Recognition System Based on Two-Dimensional Matched Filters, Electrotechnical Review, Vol. 41, No. 3-4, 1974, p. 13-16.
17. B. Gold, C.M. Rader, Digital Processing of Signals, McGraw Hill, New York, 1969.
18. A. Papoulis, The Fourier Integral and its Applications, McGraw Hill, New York, 1962, str. 30-31.
19. T.S. Huang: Two-Dimensional Windows, IEEE Trans. on Audio and Electroacoustics, AU-20, No. 1, March 1972, str. 88-90.
20. H.C. Andrews, A High Speed Algorithm for the Computers, C-17, No. 4, 1968, str. 373.
21. T.G. Stockham, High-Speed Convolution and Correlation, 1966 Spring Joint Computer Conf. AFIPS Conf. Proc., str. 229-233.
22. B.G. Batchelor (ed.), Pattern Recognition, Ideas in Practice, Plenum Press, New York, 1978.
23. S. Ribarić, N. Pavešić, Mikroprocesorski klasifikator numeričkih znakova s mrežom memorijskih komponenti LSI, Informatica 2/3, 1983, str. 162-167.

PLOT 5 – JEDAN JEDNOSTAVNI PROGRAMSKI SISTEM ZA GRAFIČKE TERMINALE I KOORDINATNE CRTAČE

INFORMATICA 4/86

UDK: 681.3.022.06 PLOT 5

Jozo J. Dujmović i Miodrag Levnaić
Elektrotehnički fakultet Beograd

U radu se opisuje PLOT5 - jedan jednostavni sistem programske podrške za grafičke terminale i koordinatne crtače. Sistem je projektovan za potrebe inženjerskih aplikacija kod kojih su zahtevi za grafikom ograničeni na jednostavno i efikasno crtanje linijskih crteža. Osnovni zahtev koji je postavljen za PLOT5 je bio da se korisnik može u najkraćem vremenu osposobiti za njegovo korišćenje, a da istovremeno može sa njim da proizvodi najsloženije linijske crteže uključujući familije krivih, dvodimenzionalne i trodimenzionalne objekte. PLOT5 je napisan na FORTRAN-u i lako je prenosiv. Implementiran je na računarima VAX/VMS i trenutno podržava simultani rad sa grafičkim terminalima tipa Tektronix serije 4110 i koordinatnim crtačima tipa CALCOMP 81.

PLOT5 - A SIMPLE SOFTWARE FOR GRAPHIC TERMINALS AND PLOTTERS. The paper presents PLOT5 - a simple software system supporting graphic terminals and plotters. The system is designed for general engineering users interested in simple and efficient production of line drawings. The basic requirement fulfilled by PLOT5 is to enable its user to quickly learn how to use the system for producing complex line drawings (e.g. families of curves, two-dimensional, and three-dimensional objects). PLOT5 is written in FORTRAN, and it is highly portable. It is implemented on VAX/VMS systems and it currently simultaneously supports graphic terminals Textronix 4110 series, and CALCOMP 81 plotters.

1. UVOD

Prvi sistemi za grafički prikaz rezultata inženjerskih proračuna pojavili su se početkom šezdesetih godina. Na primer, popularni računari za naučno-tehničke primene kao što su IBM 1620 i kasnije IBM 1130 koristili su koordinatni crtač 1627 koji je omogućavao crtanje linijskih crteža [1]. Crtanje je redovno realizovano iz FORTAN programa pozivajući skup od 5 potprograma (SCALE, GRID, CHAR, PLOT, i POINT) čija je namena bila uvodjenje faktora razmere, crtanje koordinatnih osa, ispisivanje tekstova, crtanje linijskog segmenta i crtanje markera [2]. Najznačajnija osobina ovog grafičkog sistema bila je njegova jednostavnost i efikasnost: korisnici su mogli da nauče osnovno programiranje rada sa crtačem za približno 60 minuta.

Kasniji razvoj kretao se dominantno u pravcu usavršavanja interaktivne računarske grafike [3,4]. Pri tome je došlo do razvoja velikog broja nekompatibilnih grafičkih uredjaja i do brojnih jednako nekompatibilnih osnovnih softverskih sistema za njihovu podršku. Naravno, na taj način je softver koji koristi grafičke uredjaje postao neprenosiv, pa se pojavila hitna potreba za standardizacijom u ovoj oblasti [5]. U ulozi standarda za sada se pojavljuje GKS [6,7] koji definiše preko 100 osnovnih standardnih funkcija koje omogućavaju široki spektar najrazličitijih grafičkih primena. Nažalost, univerzalnost GKS-a je dobijena po cenu gubitka jednostavnosti. Tako korisnik za crtanje najprostije krive mora da pozove od 10 do 30 raznih GKS funkcija, a priručnik za GKS uzima znatno više od 100 strana. Prema tome, korisnici koji žele da primenjuju GKS moraju uložiti značajan napor da njime ovladaju i za veći broj korisnika koji nisu specijalisti za računarsku grafiku, već povremeno koriste grafičke uredjaje za prikaz odredjenih rezultata, ovakav prilaz može da bude neefikasan.

Imajući u vidu navedene probleme u ovom radu je predložen jedan efikasan sistem za linijsku grafiku čija se upotreba može savladati u roku od 2 sata. Sistem je razvijen kao generalizacija grafičkog sistema za IBM 1130 i u početku je imao za cilj da omogući jednostavan prelaz korisnicima sa IBM 1130 na VAX/VMS sisteme. Baziran je na 5 osnovnih potprograma i odatle naziv PLOT5. Ovi potprogrami se mogu pozivati iz FORTRAN programa, a takodje i iz drugih jezika koji omogućavaju isti način prenosa argumenata. Pri razvoju PLOT5 sistema učinjen je pokušaj da se, zadržavajući jednostavnost u rukovanju, u sistem ipak ugradi što više savremenih osobina koje omogućavaju moderni grafički raster terminali i crtači u boji. Nadalje, posvećena je posebna pažnja efikasnosti implementacije čije su bitne odlike sledeće:

1. Nazavisnost od grafičkog uredjaja koja se ogleda u tome da se uredjaj bira jednim jedinim identifikatorom u programu i prelaz sa uredjaja na uredjaj može se obaviti bez prekidanja rada.

2. Automatsko odsecanje svih delova crteža van zadate prikazne površine čime je omogućena maksimalna upotrebljivost slike u svim uslovima.

3. Realizacija sistema u vidu delimično povezanog "reentrant" modula, čime su svedeni na minimum kako utrošak memorijskog prostora za izvršne verzije programa, tako i vreme povezivanja sa korisničkim programima.

Trenutno je PLOT5 sistem implementiran na grafičkim terminalima serije TEKTRONIX 4110 i koordinatnim crtačima tipa CALCOMP 81, a u toku

su implementacije za nekoliko drugih grafičkih uredjaja. Rad sa grafičkim uredjajima se obavlja tako da se osnovne funkcije grafičkog prikazivanja podataka realizuju korišćenjem pet PLOT5 potprograma sa sledećom namenom:

1. Inicijalizacija grafičkog uredjaja i definicija parametara preslikavanja.
2. Crtanje koordinatnih osa i mreže sa kotiranjem.
3. Crtanje tačaka i drugih oznaka na željenom mestu ekrana.
4. Crtanje prave linije od trenutne tačke do odredisne tačke.
5. Ispisivanje tekstova uz crtež.

Detaljan opis ovih funkcija sledi u nastavku.

2. PRIKAZNA POVRŠINA, AKTIVNA POVRŠINA I KORISNIKOV PROSTOR

Ekran grafičkog terminala i prikazna površina koordinatnog crtača su pravougaonici različitih fizičkih dimenzija. Pri realizaciji programske podrške za grafičke uredjaje uobičajeno je da se "prikazna površina" ("display") meri u normalizovanim jedinicama, tako da je uvek 0<=x<=xdim<=1 i 0<=y<=ydim<=1. Pri tome se podrazumeva da važi jedna od dve mogućnosti:

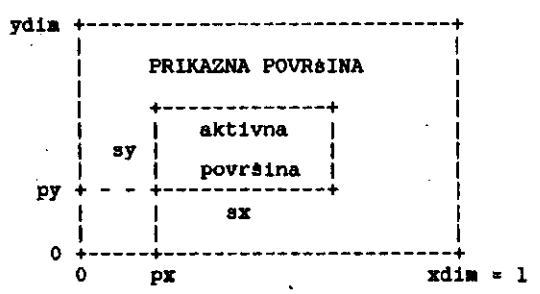
xdim < ydim = 1,
ili
ydim < xdim = 1.

U slučaju grafičkih uredjaja TEKTRONIX 4105 i CALCOMP 81 važi xdim=1 pri čemu je:

ydim = 0.765 za grafički terminal TEK 4105 i za kopir aparat TEK 4695
= 0.8284 za koordinatni crtač Calcomp 81.

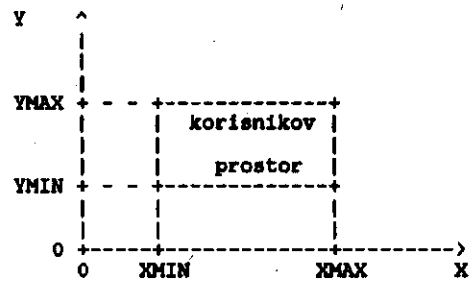
Veličine pravougaone prikazne površine su 200x153 mm za grafički terminal i kopir aparat i 338x280 mm za koordinatni crtač Calcomp 81 .

Prikazna površina grafičkog uredjaja data je na slici 1. Redovno korisnik ne koristi celu prikaznu površinu. Pod "aktivnom površinom" ("viewport") podrazumeva se onaj deo prikazne površine na koji korisnik namerava da ograniči svoj crtež.



Slika 1.

Sa druge strane, "korisnikov prostor" ("window") je u opštem slučaju pravougla oblast u korisnikovom koordinatnom sistemu koja obuhvata sve ono što korisnik namerava da posmatra (slika 2.) Korisnikov prostor se po X osi proteže od XMIN do XMAX a po Y osi od YMIN do YMAX, gde su XMIN, XMAX, YMIN, i YMAX zadati u korisnikovim jedinicama i prema tome predstavljaju proizvoljne racionalne brojeve. Kod prikazivanja na grafičkim uredjajima korisnikov prostor se preslikava na aktivnu površinu.



Slika 2.

Dimenzije aktivne površine (sx i sy) kao i koordinate njenog levog donjeg ugla (px i py) zadaju se u koordinatama jediničnog kvadrata pa prema tome moraju da zadovolje uslove

0 <= px < 1, 0 <= py < ydim,
0 < px+sx <= 1, 0 < py+sy <= ydim .

Na primer, korisnik može da nacrta kvadratnu sliku najveceg formata birajuci px=py=0, sx=sy=ydim.

Prema tome, celokupno grafičko preslikavanje može se definisati sa skupom od sledećih 10 parametara:

xdim,ydim	= parametri prikazne površine (fiksno definisan za svaki konkretni uredjaj)
px,py,sx,sy	= parametri aktivne površine
XMIN,XMAX,YMIN,YMAX	= paramteri korisničkog prostora

Delovi crteža mogu da izadju i van okvira aktivne površine a i van okvira prikazne površine. Na crtačima i na grafičkim terminalima se delovi crteža van prikazne površine uvek automatski odsecaju bez bilo kakvih posledica na deo crteža koji se nalazi na prikaznoj površini. Delovi crteža van aktivne površine mogu se takodje automatski odsecati, ali samo na eksplicitan zahtev korisnika dat u potprogramu SCALE.

3. BOJE, PISALJKE I KOPIRANJE CRTEŽA

Grafički terminal radi sa 8 boja koje su kodirane brojevima 0,1,...,7. Sistem PLOT5 daje ovim brojevima, prema standardu TEKTRONIX-a, sledeće početne vrednosti:

0 - crno 4 - tamno plavo
1 - belo 5 - svetlo plavo
2 - crveno 6 - ljubičasto
3 - zeleno 7 - zuto

Navedene vrednosti mogu se po potrebi menjati koristeći "setup" proceduru terminala ali u svakom momentu broj prisutnih boja koje terminal simultano prikazuje ne može biti veći od 8. Prema tome, u svim programima koji slede pod pojmom "boja" podrazumeva se jedna cifra cija je interpretacija definisana gornjom tabelom. Pri tome terminal podrazumeva da je boja pozadine crteža početno definisana sa kodom 0 pa je stoga pozadina uvek crna (neosvetljena). Ako se želi to izmeniti onda se mora intervenisati u "setup" proceduri sa kojom se može kodu 0 dodeliti proizvoljna boja,

ili odrediti da boji pozadine ne odgovara kod 0, već neki drugi od postojećih kodova. Crteż se sa ekrana može direktno kopirati na papir, pri čemu se koristi kopir aparat TEK 4695. Kopiranje traje oko 4 minuta, pa ga treba koristiti sa merom i to uvek samo za konačne rezultate.

Koordinatni crtač ima 8 pozicija za pisaljke koje mogu biti u raznim bojama i/ili u raznim debljinama. Promena pisaljke programski je ekvivalentna promeni boje na grafičkom terminalu. Početni razmeštaj pisaljki u 8 raspoloživih kućišta obavlja se pre početka rada ručno od strane operatora. Naravno, pisaljke se mogu menjati i u toku rada ako se za to predvidi odgovarajuća pauza.

4. INICIJALIZACIJA UREDJAJA I DEFINISANJE PARAMETARA PRESLIKAVANJA

Da bi se korisniku olakšalo programiranje svi PLOT5 potprogrami za rad sa grafičkim uredjajima su isti za sve uredjaje. Naravno, fizičke osobine uredjaja i način komuniciranja sa njima se medjusobno razlikuju. Stoga je na početku rada neophodno definisati uredjaj sa kojim se namerava raditi, izvršiti inicijalizaciju uredjaja i uvesti parametre preslikavanja. To se postiže pozivom potprograma

```
CALL SCALE ( ID,   px,py, sx,sy,
                   XMIN,XMAX, YMIN,YMAX )
```

pri čemu je ID identifikator grafičkog uredjaja i načina odsecanja:

```
| ID | = 1 , za grafički terminal TEK 4105
     = 2 , za koordinatni crtač Calcomp 81.
```

Ako je ID>0 onda se odsecanje obavlja na granici prikazne površine, a ako je ID<0 onda se odsecanje obavlja na granici aktivne površine.

Smisao veličina px,py, sx,sy, XMIN,XMAX, YMIN,YMAX je jasan sa slika 1 i 2. Pri tome ponavljamo da su px,py,sx,sy izraženi u delovima jediničnog kvadrata (tj. u normalizovanim koordinatama), dok su XMIN,XMAX, YMIN,YMAX izraženi u korisnikovim jedinicama koje će se isključivo koristiti u ostalim PLOT5 potprogramima. Korisnikov prostor obuhvata pravougaonik

```
XMIN <= X <= XMAX
YMIN <= Y <= YMAX ,
```

pri čemu XMIN, XMAX, YMIN, i YMAX mogu biti proizvoljne pozitivne, nulte ili negativne veličine, čime se odredjuje gde će biti koordinatni početak (vidljiv unutar ili na rubu slike, ili nevidljiv izvan slike). To znači da se tačka korisnikovog prostora sa koordinatama XMIN,YMIN preslikava u tačku sa koordinatama px,py prikazne površine (tj. u levi donji ugao aktivne površine).

Prilikom crtanja na koordinatnom crtaču (a i na drugim grafičkim uredjajima) najčešće se polazi od željenih dimenzija slike koje se zadaju u dužinskim jedinicama (cm ili mm). To su:

```
H = horizontalna dimenzija (širina) slike
V = vertikalna dimenzija (visina) slike.
```

Veličine H i V odredjuju fizičke dimenzije aktivne površine. Sa druge strane, fizičke dimenzije prikazne površine (horizontalna dimenzija HH i vertikalna dimenzija VV) konstantne su i zavise od uredjaja. Korisnik takodje redovno želi da njegova slika ima donju (horizontalnu) marginu i levu (vertikalnu) marginu čije su dimenzije HM i VM respektivno. Koristeći zadate veličine H, V, HH, VV, HM i VM mogu se direktno izračunati parametri aktivne površine. Ako je xdim=1 ovi parametri su:

```
px = VM/HH,   py = HM/HH
sx = H/HH,    sy = V/HH
```

i mogu se direktno uneti u poziv potprograma SCALE.

Potprogram SCALE definiše trenutno korišćeni uredjaj koji se može tokom rada menjati. Tako će, na primer, pozivom CALL SCALE(1,...) biti korisniku dodeljen grafički terminal na koga će se odnositi svi naredni PLOT5 potprogrami. Ako se u istom programu zatim izvrši poziv CALL SCALE(2,...) onda će se grafički terminal osloboditi i korisniku dodeliti crtač na koga će se zatim primenjivati preostali potprogrami. Novi poziv CALL SCALE(1,...) oslobadja crtač (i drugi korisnici ga mogu odmah koristiti) i opet zauzima grafički terminal. Treba napomenuti da se pri promeni grafičkog uredjaja po pravilu menjaju parametri prikazne i aktivne površine u potprogramu SCALE, ali pozivi svih ostalih potprograma ostaju neizmenjeni.

5. CRTANJE KOORDINATNIH OSA I MREŽE SA KOTIRANJEM

Ova funkcija se obavlja pozivom

```
CALL GRID ( DX, NX, DY, NY, ID )
```

gde su DX i DY veličine jednog podeoka (t.j. razmak izmedju dve susedne linije koordinatne mreže) po x i y koordinati izražene u korisnikovim jedinicama, NX i NY predstavljaju broj označenih podintervala na koje je izdeljen svaki podeok (ako nema podintervala onda je NX=NY=1), a ID je četvorocifreni identifikator načina crtanja definisan sa četiri cifre, okmb, sledećeg značenja:

```
o = pravougaoni okvir korisnikove slike
    (0 = bez okvira, 1 = sa okvirom slike)
k = kotirane ose (0 = bez osa, 1 = sa osama i
    kotiranjem)
m = mreža (0 = bez mreže, 1 = mreža od punih
    linija, 2 = mreža od duzih crtica, 3 =
    mreža od kraćih crtica Cna grafičkom
    terminalu to su tačkice]; što je veća
    vrednost za m to je rad crtača sporiji)
b = boja (kodirano sa identifikatorom boje
    0,...,7)
```

Ako ose ne prolaze kroz korisnikovu sliku onda se u slučaju ID=110b automatski kotiraju sve četiri strane okvira, a u slučaju ID=010b se crtaju one dve kotirane strane okvira koje su najbliže koordinatnim osama. U slučaju ID=10mb nema kotiranja, pa vrednosti NX i NY nemaju uticaja. Kad god se traži mreža automatski se crta i (kotirani ili nekotirani) okvir.

6. CRTANJE TAČAKA I DRUGIH OZNAKA NA ŽELJENOM MESTU PRIKAZNE POVRŠINE

Ispisivanje željene oznake u tački X,Y (izraženo u korisničkim koordinatama) realizuje se pozivom

```
CALL POINT ( X, Y, ID )
```

gde identifikator ID definise tip oznake sa dve ili tri cifre pri čemu zadnja cifra (b) označava boju oznake :

```
4105 i 81
==============================
1b = .
2b = + (mali plus)
3b = + (veći plus)
4b = X (veći iks)
5b = veći kvadrat
6b = *
------------------------------


 Samo 81
==============================
7b  =  <          | strelice za
8b  =  >          | koordinatne
9b  =  ^          | ose i oznake
10b =  v          | tačaka
11b =  V
12b =  mali ispunjeni kvadrat
13b =  mali neispunjeni kvad.
14b =  x (manji iks)
------------------------------


Samo 4105
==============================
15b = 0 (nula)
16b = kvadrat sa tačkom unutra
17b = ispunjeni kvadrat
18b = romb
19b = ispunjeni romb
```

Identifikator ID može biti i negativan. U tom slučaju X i Y su u korisnikovim jedinicama izražene veličine RELATIVNOG POMAKA iz date polazne tačke (t.j. X i Y se ne izražavaju u odnosu na koordinatni početak već u odnosu na trenutni položaj ukazatelja ekrana). Tip oznake odredjen je apsolutnom vrednošću identifikatora ID, na način prikazan gornjom tabelom.

U slučaju grafičkog terminala potprogram POINT se može primeniti i za brisanje ekrana, što se realizuje pomoću identifikatora ID=0. U tom slučaju X označava broj sekundi čekanja pre brisanja ekrana a Y označava broj sekundi čekanja posle brisanja ekrana. Stoga, na početku programa, u većini slučajeva treba primeniti poziv CALL POINT(0.,0.,0); crtač na ovo ne reaguje.

7. CRTANJE PRAVE LINIJE OD TRENUTNE TAČKE DO ODREDIŠNE TAČKE

U svakom momentu pisaljka crtača ili ukazatelj ekrana grafičkog terminala se nalaze u nekoj tački prikazne površine. Ovu tačku nazivamo trenutna pozicija i označavamo sa A. Neka su X,Y koordinate proizvoljne odredišne tačke B. Tačke A i B mogu se spojiti pravom linijom pomoću poziva

```
CALL LINE ( X, Y, ID )
```

pri čemu su X i Y izraženi u korisnikovim jedinicima. Ako je ID>0 onda su X i Y apsolutne koordinate u odnosu na koordinatni početak. Ako je ID<0 onda su X i Y u korisnikovim jedinicama izražene veličine relativnog pomaka iz polazne tačke A (t.j. X i Y se ne izražavaju u odnosu na koordinatni početak već u odnosu na trenutni položaj). Tip linije i boja (b) su odredjeni apsolutnom vrednošću identifikatora ID na sledeći način:

```
1b =  pomak bez crtanja linije
2b =  _____________          (puna linija)
3b =  .............          (tačkasta linija)
4b =  - . - . - . - .        (tačka - crta)
5b =  - - - - - - - -        (crtkana linija -
                               kratke crtice)
6b =  - - - - - - - -        (crtkana linija -
                               duge crtice)
7b =  - ... - ... - ..       (tri tačke - crta)
                              [samo 4105]
8b =  __ _ __ _ __ _         (duga crta -
                               kratka crta)
9b =  _ _ _ _ _ _ _          (crtkana linija -
                               dugi razmak)
```

8. ISPISIVANJE TEKSTOVA UZ CRTEŽ

Na korisnikovoj slici moguće je ispisivati proizvoljne tekstove pomoću potprograma

```
CALL TEXT ( X, Y, DATA, F, ID )
```

pri čemu argumenti imaju sledeću interpretaciju:

X,Y = koordinate levog donjeg ugla teksta izražene u korisnikovim jedinicama.

DATA = podatak koji se ispisuje: može biti znakovnog, celobrojnog ili racionalnog tipa.

F = veličina koja označava format ispisivanja na sledeći način:

- F = <ukupan broj znakova>, ako je DATA niz znakova ili ceo broj
- F = <ukupan broj znakova>.<broj znakova desno od decimalne tačke>, za slučaj da je DATA racionalan broj.

ID = [-]uvbst (identifikator, do 7 cifara) kojim se odredjuje nagib slova, ugao linije teksta, veličina slova, boja, smer, i tip teksta na sledeći način:

Negativan identifikator označava za CALCOMP 81 slova nagiba 75 stepeni, a pozitivan identifikator slova nagiba 90 stepeni (kod grafičkog terminala nagib slova je uvek 90 stepeni).

u = ugao linije teksta sa +x osom dat u stepenima (1 do 3 cifre, pri čemu je u >= 0). Koordinatni crtač ovaj ugao može da tačno prikaže, dok grafički terminal vrši zaokružavanje zadatog ugla na najbliži multipl od 90 stepeni.

v = oznaka veličine slova iz skupa (0,1,...,9) kojom se odredjuje širina slova prema sledećoj tabeli:

širina slova u mm.

v	0	1	2	3	4	5	6	7	8	9
TEK	2.1	2.1	4.2	6.3	8.4	10	12.6	14	16.5	19
CAL	1.2	1.8	2.4	3.0	3.6	4.2	7.2	12.6	20	30

Za koordinatni crtač CALCOMP 81 visina slova se odredjuje automatski

i iznosi 3/2 širine. Za TEKTRONIX 4105 visina slova je 7/5 širine.

b = boja teksta (0,1,...,7)

s = smer ispisivanja iz skupa od 4 smera u odnosu na liniju koju definiše odabrani ugao u (0=desno, 1=gore, 2=levo, 3=dole).

t = tip teksta kojim se odredjuje šta se ispisuje, na sledeći način:

t = 1, ako je DATA celobrojnog tipa
t = 2, ako je DATA racionalnog tipa
t = 3, ako je DATA znakovnog tipa

9. IMPLEMENTACIJA PLOT5 SISTEMA

Grafički sistemi koji nude mogućnosti kao PLOT5, u tipičnom slučaju se sastoje od stotinak potprograma i prevedeni mogu zauzimati od 50KB na više. To znači da bi praktično svaki grafički program morao obuhvatiti celokupan proces povezivanja i da bi izvršna verzija svakog programa bila opterećena sa celokupnim softverom grafičkog sistema. Na taj način se pored gubitka vremena dolazi do potpuno neprihvatljive situacije u kojoj se na diskovima nalazi onoliko kopija celokupnog grafičkog softvera, koliko ima grafičkih programa. U uslovima velikog broja korisnika i velikog broja malih grafičkih programa ovo može da bude izuzetno ozbiljan problem. Zbog toga je neophodno da implementacija grafičkog softvera zadovolji dva fundamentalna zahteva:

1. Parcijalno povezivanje svih grafičkih potprograma sa izuzetkom najvišeg nivoa čime se formira bazični grafički modul
2. Fiksno i jednokratno instaliranje bazičnog grafičkog modula u memoriju računara tako da je istovremeno dostupan svim korisničkim grafičkim programima.

PLOT5 sistem zadovoljava navedena dva zahteva. Parcijalnim povezivanjem su potprogrami SCALE, GRID, POINT, LINE i TEXT jednom za uvek povezani sa svim potprogramima koji se dalje lančano pozivaju i tako je formiran bazični grafički modul. Proces povezivanja korisničkog programa sa bazičnim grafičkim modulom je tada veoma brz jer se svodi na realizaciju samo onoliko veza koliko u korisničkom programu ima poziva osnovnih PLOT5 potprograma. Sa druge strane, izvršni programi su veoma mali jer obuhvataju jedino korisničke instrukcije i veze sa bazičnim grafičkim modulom. Naravno, za ovakav način implementacije neophodna je i odgovarajuća podrška operativnog sistema koja je u slučaju VMS-a raspoloziva.

10. PRIMERI PRIMENE PLOT5 SISTEMA

Verovatno najčešći primer primene grafičkih sistema je crtanje familija krivih. Stoga je u nastavku prikazan jedan opšti program za ispitivanje osobina familije krivih prema sledećim zahtevima:

(1) Korisnik prikazuje familiju krivih na grafičkom terminalu zadajući interaktivno koordinate korisničkog prostora XMIN, XMAX, YMIN, YMAX čime bira i proizvoljno uvećava deo XY ravni koji želi da posmatra.

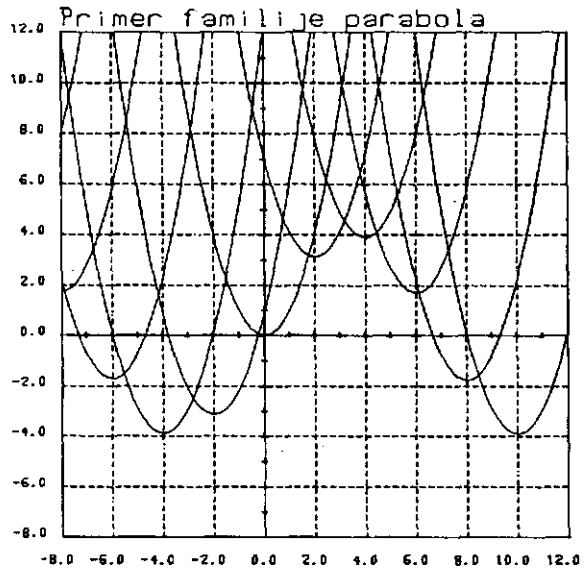
```
C-----------------------------------------+
C    CRTANJE FAMILIJE KRIVIH NA TERMINALU I  !
C            KOORDINATNOM CRTACU             !
C-----------------------------------------+
C   HH,VV = Dimenzije  prikazne povrsine u mm
C   H ,V  = Zeljene velicine  aktivne povrsine
C           (velicina crteza) u mm
C   HM,VM = Sirine  horizontalne  i vertikalne
C           margine u mm
C   SS    = Dimenzije sirine slova u mm
C   IS1   = Ident. zeljene sirine slova (kote)
C   IS2   = Id. zeljene sirine  slova (naslov)
C   NX,NY = Broj kotiranih podeoka duz osa
C   NK    = Opseg u kome se krece parametar
C           familije krivih K (-NK <= K <= NK)
C   NSEG  = Broj linijskih segmenata po krivoj
C-----------------------------------------
      DIMENSION  HH(2), VV(2), H(2), V(2),
     * HM(2), VM(2), SS(0:9,2), IS1(2), IS2(2)
      CHARACTER*30   NASLOV      ! Naslov slike

      DATA  HH, VV /200., 338., 153., 280./,
     *       H,  V /110., 102., 110., 102. /
      DATA  SS  /
     *2.1,2.1,4.2,6.3,8.4,10.,12.6,14.,16.5,19.
     *,1.2,1.8,2.4,3.,3.6,4.2,7.2,12.6,20.,30./
      DATA  IS1/1,1/, IS2/2,4/,  NX,NY/10,10/,
     * NK/5/, HM,VM/36.,80.,45.,30./, NSEG/150/

C---- Familija krivih sa parametrom K
      F(X,K) = 4*SIN(0.9*K) + (X-2*K)**2

      NASLOV = 'Primer familije parabola'
      DO WHILE (.TRUE.) ! Pocetak glavne petlje

C---- Unos parametara i izbor uredjaja
        PRINT *, 'XMIN,XMAX,YMIN,YMAX, Tip',
     *  ' uredjaja (1=terminal, 2=plotter) ? '
        READ  *, XMIN, XMAX, YMIN, YMAX, ID
        IF (ID .LT. 1 .OR. ID .GT. 2)  STOP

C---- Inicijalizacija uredjaja i definisanje
C     parametara preslikavanja
        CALL SCALE( ID,
     *       VM(ID)/HH(ID), HM(ID)/HH(ID),
     *       H(ID)/HH(ID),  V(ID)/HH(ID),
     *       XMIN, XMAX, YMIN, YMAX  )
        CALL POINT( 0.,0.,0 ) ! Brisanje ekrana

C---- Prikaz koordinatne mreze
        PODEOKX = (XMAX-XMIN) / NX
        PODEOKY = (YMAX-YMIN) / NY
        CALL GRID(PODEOKX, 2, PODEOKY, 2, 1131)

C---- Ispisivanje naslova i kotiranje osa
        S = SS(IS1(ID),ID)  ! Sirina slova u mm
        SIR = S*(XMAX-XMIN)/H(ID) ! Dim slova u
        VIS = 1.5*S*(YMAX-YMIN)/V(ID)! kor.jed.
        CALL TEXT(XMIN,YMAX+VIS/2,NASLOV,30,
     *            1000*IS2(ID) + 103)
        DO i = 0, NX           ! Kotiranje X ose
           XI = i * PODEOKX + XMIN
           CALL TEXT( XI-2.5*SIR, YMIN-2*VIS,
     *          XI, 4.1, 1000*IS1(ID) + 102 )
        END DO
        DO i = 0, NY           ! Kotiranje Y ose
           YI = i * PODEOKY + YMIN
           CALL TEXT( XMIN-5.5*SIR, YI,
     *          YI, 4.1, 1000*IS1(ID) + 102 )
        END DO

C---- Prikaz krivih F(X,K)
        CALL SCALE(-ID) ! Promena granice odsec.
        DX = (XMAX - XMIN) / NSEG
        DO  K = -NK,NK     ! K = Parametar krive
          CALL LINE( XMIN, F(XMIN,K), 11)
          DO IX = 1,NSEG       ! Petlja za crta-
            X = XMIN + DX*IX   ! nje jedne krive
            CALL LINE( X, F(X,K), 21 )
          END DO
        END DO
        CALL LINE ( XMIN, YMIN, 11 ) ! Povratak
      END DO             ! na  granicu  aktivne
      STOP               ! povrsine na kraju rada
      END
```

(2) PLOT5 programi vrše automatsko ispisivanje odgovarajućih vrednosti koordinata uz ose.

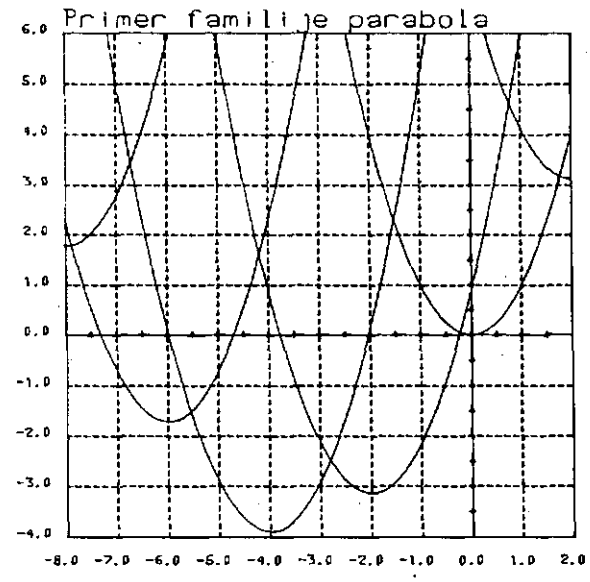
(3) Kada se interaktivno odabere pogodan korisnički prostor onda se slika prikazuje na crtaču.

Ovakav rad je omogućen činjenicom da PLOT5 automatski odseca delove familije krivih van aktivne površine i automatski postavlja koordinatni sistem na odgovarajuće mesto na aktivnoj površini. Prikazani program je pisan tako da služi kao paradigma primene PLOT5 sistema. Može se univerzalno koristiti za prikazivanje familija krivih i stoga koristi niz parametara definisanih u DATA specifikacijama. Modifikacijom ovih parametara program se može lako prilagoditi drugim primenama. I pored jednostavnosti (samo tridesetak instrukcija) ovaj program prikazuje koordinatni sistem, koordinatnu mrežu, kotiranje x i y osa, ispisivanje naslova slike, crtanje niza krivih i podešavanje proizvoljne veličine crteža uz simultani interaktivni rad na dva grafička uredjaja i efekte zumiranja. Ovakvim programom se omogućava svestrana analiza osobina familija krivih i njihov kvalitetan graficki prikaz. U poredjenju sa univerzalnim grafičkim sistemima navedeni PLOT5 program je znatno kompaktniji i lakši za korišćenje.

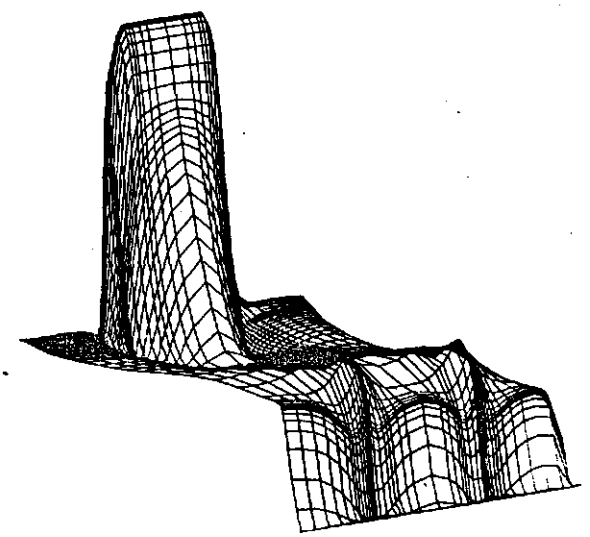
PLOT5 sistem se može uspešno koristiti i u slučaju programa za realizaciju složenijih linijskih crteža. Jedan takav primer primene PLOT5 sistema ilustruje slika 5.



Slika 3.



Slika 4.



Slika 5.

REFERENCE

[1] IBM, "IBM 1627 Plotter", (GA26-5710).

[2] IBM, "IBM 1130/1800 Plotter Subroutines", (GC26-3755).

[3] Newman, W,M, and R.F. Sproull, "Principles of Interactive Computer Graphics". Second Edition, McGraw-Hill, 1981.

[4] Foley, J.D. and A. Van Dam, "Fundamentals of interactive Computer Graphics". Addison-Wesley, 1983.

[5] ACM SIGGRAPH Committee, "Status Report of the Graphics Standards Committee", Computer Graphics 13(3), August 1979.

[6] Hopgood, F.R.A., D.A. Duce, J.R. Gallop, and D,C Sutcliffe, "Introduction to the Graphical Kernel System (GKS)". Academic Press, 1983.

[7] Enderle, G., K. Kansy, and G. Pfaff, "Computer Graphics Programming / GKS - The Graphics Standard". Springer-Verlag 1984.

PROCESOR S PODATKOVNO PRETOKOVNO ARHITEKTURO

Institut Jožef Stefan, Ljubljana
UDK: 681.3.02

Jurij Šilc, Borut Robič

Članek predstavlja prvi procesor s podatkovno pretokovno arhitekturo. Notranja krožna pipeline organizacija, ki temelji na podatkovnem vodenju, daje procesorju veliko moč. Bogat nabor ukazov je prirejen tako, da je procesor zelo uporaben za hitro obdelavo slikovnih in govornih signalov. Potreba po takšnih obdelavah se pojavlja v računalnikih pete generacije.

Data Flow Architecture Based Processor - A new data flow architecture based processor is described. The processor employs token based data flow and pipelined architecture to achieve a very high throughput rate in the realm of digital image and speech processing.

1. UVOD

Pri načrtovanju sistemov za obdelavo slik smo običajno prisiljeni poiskati kompromis med hitrostjo in fleksibilnostjo sistema. Okornost in počasnost sistema, ki ga sestavljata miniračunalnik za obdelavo slike in masovni pomnilnik za njeno shranjevanje, sta nesprejemljivi za delo v realnem času. Z dodatkom posebne materialne opreme se hitrost obdelave poveča, toda žal na račun fleksibilnosti, saj vsaka sprememba programske opreme narekuje spremembo materialne opreme. Omenjeni razkorak med hitrostjo in fleksibilnostjo sistema je moč omiliti z uporabo programabilnega slikovnega procesorja, ki se odlikuje s podatkovno vodeno arhitekturo.

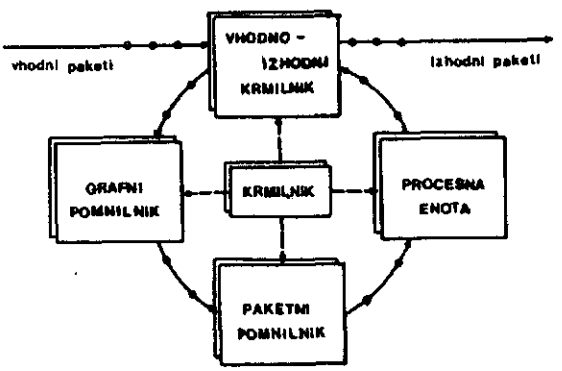
2. PODATKOVNO PRETOKOVNI PROCESOR

V nasprotju s von Neumannovimi procesorji, ki delujejo na podlagi dostave ukazov, temelji delo podatkovno pretokovnega procesorja na zbiranju in obdelavi paketov (tokens).

2.1. Osnovna arhitektura procesorja

Podatkovno pretokovni procesor ne rabi dostave ukaza. Namesto tega vsebuje grafni pomnilnik (tabeli povezav in točk), v katerega se pred pričetkom izvrševanja vpiše programski podgraf. Pretok podatkov se vrši s pomočjo paketov, ki vsebujejo naslov procesorja, identifikator, podatkovno ter krmilno polje. Med pretokom po procesorju paket še nekajkrat spremeni vsebino in dolžino, kot bo razvidno iz kasnejšega opisa.

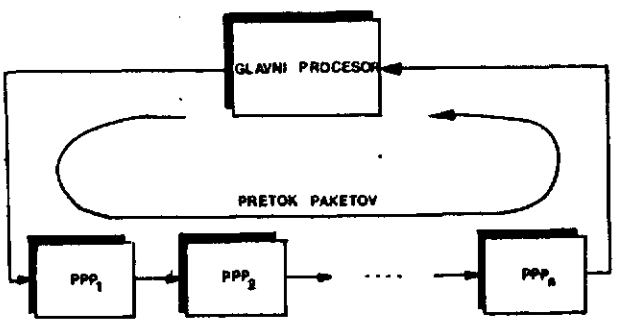
Notranja krožna pipeline arhitektura (Slika 1) omogoča procesni enoti neprekinjeno delovanje. Procesna enota vsebuje množilnik in ALU, ki omogoča standardne aritmetično logične operacije. Nabor ukazov je širši kot pri večini klasičnih procesorjev.



Slika 1: Krožna pipeline arhitektura podatkovno pretokovnega procesorja.

2.2. Večprocesorski podatkovno pretokovni sistem

Večprocesorski podatkovno pretokovni sistem sestavlja kaskada podatkovno pretokovnih procesorjev, povezanih z glavnim procesorjem, kot je prikazano na Sliki 2.

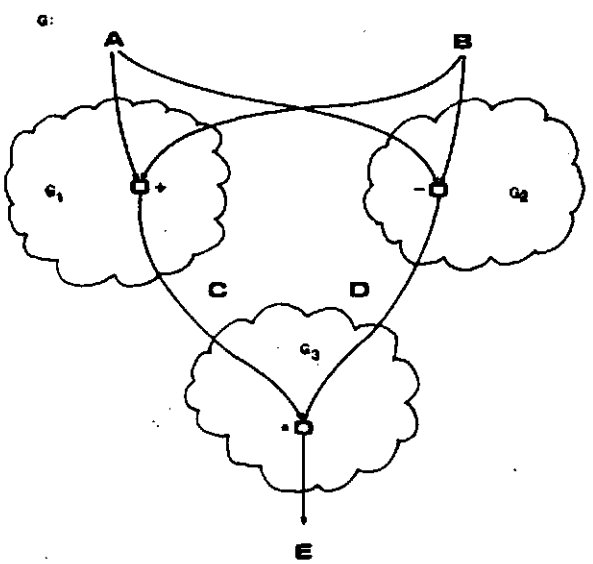


Slika 2: Podatkovno pretokovni računalnik.

Komunikacija med podatkovno pretokovnim procesorjem in okolico poteka s pomočjo vhodno-izhodnega krmilnika. Glavni procesor pošlje paket podatkovno pretokovnim procesorjem. Iz naslovnega polja paketa podatkovno pretokovni procesor ugotovi, če je paket namenjen njemu ter ga v tem primeru sprejme, izloči naslovno polje in pošlje v pretok - najprej v grafni pomnilnik (tabelo povezav). Paket, ki ni namenjen danemu procesorju, se nespremenjen pošlje v istem ciklu preko vhodno-izhodnega krmilnika naslednjemu procesorju. Procesor je tako za tuje pakete transparenten. Torej vsak podatkovno pretokovni procesor zbira svoje pakete.

2.3. Delovanje podatkovno pretokovnega sistema

Delovanje podatkovno pretovnega računalnika, kot ga prikazuje Slika 2, ilustrirajmo z naslednjim primerom. Dan naj bo program za izračun E = (A+B)*(A-B), opisan s podatkovno pretokovnim programskim grafom G (Slika 3). Graf G lahko razbijemo na tri podgrafe G_1, G_2 in G_3. Ker se lahko izvršujeta G_1 in G_2 vzporedno, ju je smiselno vpisati v dva različna podatkovno pretokovna procesorja P_1 in P_2. G_3 lahko vpišemo v katerikoli procesor, npr. v P_1. Program se prične izvrševati takoj, ko glavni procesor pošlje vhodna paketa A in B. Tako mora za izvršitev operacije C = A + B procesor P_1 prejeti paketa, ki nosita vrednosti A in B, šele nato lahko izvrši operacijo '+'. Vhodna paketa lahko prispeta v poljubnem zaporedju, saj je procesor sposoben razpoznavati pakete in jih hraniti v paketnem pomnilniku. Iz opisa podgrafa G_1, ki se nahaja v njegovem grafnem pomnilniku, P_1 ugotovi, da paketa A in B pripadata preko operacije '+' paketu C, zato ju združi in pošlje v procesno enoto. Vrednost, ki je rezultat izvršitve v procesni enoti, se vstavi v paket in opremi z oznako C. Istočasno se v procesorju P_2 izračuna paket D, ki ga P_2 pošlje glavnemu procesorju. Ker je C vhodni paket nove operacije, ki se mora izvršiti v istem procesorju P_1, se shrani v njegov paketni pomnilnik, dokler iz glavnega procesorja ne prispe paket D. Tedaj sta v procesorju P_1 oba paketa za izvršitev operacije '*' in že prej opisani postopek se ponovi. Vidimo, da zagotavlja pravilna izbira predhodno vpisanega podgrafa izkoriščanje vsebovane vzporednosti ter neprekinjeno delo procesorjev.



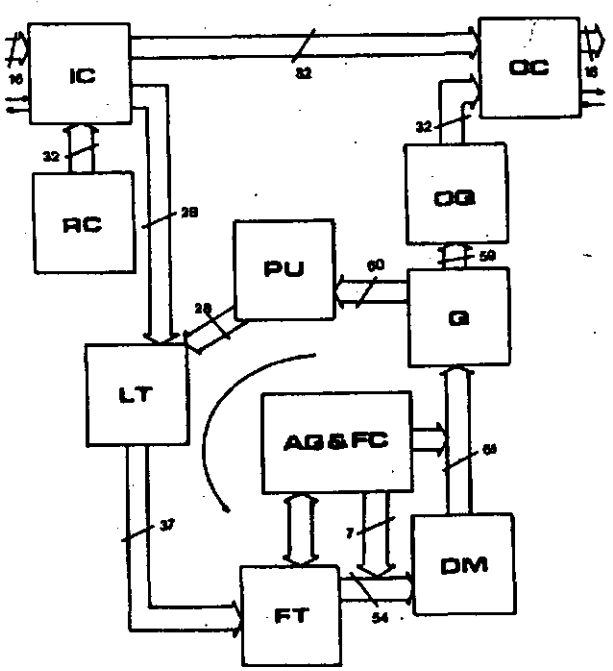
Slika 3: Podatkovno pretokovni graf za izračun E = (A+B)*(A-B).

3. PODATKOVNO PRETOKOVNI PROCESOR μPD7281

Primer podatkovno pretokovnega procesorja je NEC μPD7281, katerega moč temelji na krožno organizirani pipeline arhitekturi ter bogatem naboru ukazov. μPD7281 je prvi VLSI čip, ki deluje po načelih podatkovno pretokovne arhitekture [3]. Ta omogoča večjo učinkovitost procesorja v mnogih večprocesorskih aplikacijah, kot sta procesiranje slik ter razpoznavanje vzorcev na področju umetne inteligence, kjer se uporabljajo algoritmi za dvodomenzionalno konvolucijo, povečavo, pomanjšavo in rotacijo. Njegova učinkovitost postane očitna predvsem pri procesiranju slik v realnem času, kjer dodobra izkorišča vsebovane vzporednosti uporabljenih algoritmov. μPD7281 ni uporaben le pri procesiranju slik, temveč tudi pri zahtevnih numeričnih izračunih, kot so matrično matrično množenje, matrično vektorsko množenje, aritmetika s plavajočo vejico ter izračuni transcendentnih funkcij v realnem času.

3.1. Pipeline organizacija procesorja

Kot je prikazano na Sliki 4 sestavlja procesor deset funkcionalnih enot: vhodni krmilnik (IC), izhodni krmilnik (OC), tabela povezav (LT), tabela točk (FT), paketni pomnilnik (DM), vrsta (Q), procesna enota (PU), izhodna vrsta (OQ), generator naslovov in krmilnik pretoka (AG&FC) ter osveževalni krmilnik (RC).

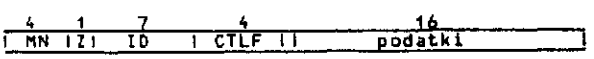


IC: vhodni krmilnik (Input Controller)
OC: izhodni krmilnik (Output Controller)
LT: tabela povezav (Link Table)
FT: tabela točk (Function Table)
DM: paketni pomnilnik (Data Memory)
Q: vrsta (Queue)
PU: procesna enota (Processing Unit)
OQ: izhodna vrsta (Output Queue)
AG&FC: generator naslovov (Address Generator) in krmilnik pretoka (Flow Controller)
RC: osveževalni krmilnik (Refresh Controller)

Slika 4: Arhitektura procesorja μPD7281.

3.2. Vhodni IC in izhodni OC krmilnik

32 bitni vhodni paket vstopi v IC v obliki, kot jo prikazuje Slika 5.



Slika 5: 32 bitni vhodno-izhodni paket.

IC primerja lastni naslov, ki mu je bil dodeljen ob resetu, z MN poljem vhodnega paketa. Če se naslova ne ujemata, se tuji paket takoj prenese v OC. Pakete takšnih oblik imenujemo PASS paketi. Če pa se naslova ujemata, se MN polje izloči in tako spremenjen paket pošlje v LT. IC hkrati ugotavlja, če je v procesorju prostor za novi paket in ga po potrebi zadrži. OC pošilja 32 bitne izhodne pakete, katerih oblika je enaka vhodnim paketom (Slika 5). Izhodni paketi so bodisi podatkovni paketi iz OQ, statusni paketi, ki jih generira OC v primeru napak, dump paketi ali tuji vhodni paketi, dobljeni iz IC, ki se tu imenujejo PASSO paketi.

Tuji paket na vhodu (vh. paket PASS):

MN'	0	ID	CTRL	podatki

Tuji paket na izhodu (izh. paket PASSO):

MN'	0	ID	CTRL	podatki

Opomba: MN' se ne ujema z MN danega procesorja.

3.2.1. Načini delovanja:

Procesor lahko deluje v treh načinih: normalnem (Normal), testnem (Test) in prekinitvenem (Break) načinu. Po hardverskem resetu je procesor v normalnem načinu delovanja. V tem načinu poteka vpis, branje in izvrševanje podgrafa. V testnem načinu delovanja je omogočeno testiranje izvrševanja. Procesor preide v testni način, ko sprejme vhodni paket SETBRK. Če med delovanjem pride do nasičenja vrste DQ ali GQ, preide procesor v prekinitveno delovanje, opisano z vhodnim paketom SETMD. V prekinitveni način lahko preide procesor tudi iz testnega načina, tako da dobi paket CBRK. V prekinitvenem načinu delovanja je omogočeno dumpanje. Iz obeh načinov se vrne v normalni način po sprejetju vhodnega paketa CRESET.

Prehod v testni način (vh. paket SETBRK):

MN	0	ID	01101	M(1) Count(15)

Opomba: M=1 prekini izvrševanje po Count ciklih
M=0 prekini po Count dostopih do LT lokacije, naslovljene z ID

Prehod v prekinitveni način (vh. paket CBRK):

0000	0	01001

Vrsta prekinitvenega načina (vh. paket SETMD):

MN	0	01011	par. za prek. način

Opomba: Parametri za prekinitveni način določajo stopnjo vhodnih omejitev ter njihovo trajanje.

Sporočilo o napaki (izh. paket ERR):

0000	0	0000000	01001	MN(4) MODE(4) OOOSY(5)

Programski reset (vh. paket CRESET):

MN	0	01000

Opomba: Programski reset povzroči le nadaljevanje izvrševanja v normalnem načinu, za razliko od hardverskega, ki resetira tabele ter dodeli naslove procesorjem.

3.2.2. Vpis podgrafa:

Predhodni vpis podgrafa poteka s pomočjo vhodnih paketov SETLT, SETFTR, SETFTL in SETFTT.

Vpis povezave v LT (vh. paket SETLT):

MN	0	adr. v LT	11000	podatki za v LT

Vpis točke v FTR (vh. paket SETFTR):

MN	0	adr. v FT	11010	podatki za v FTR

Vpis točke v FTL (vh. paket SETFTL):

MN	0	adr. v FT	11101	podatki za v FTL

Vpis točke v FTT (vh. paket SETFTT):

MN	0	adr. v FT	11111	podatki za v FTT

3.2.3. Izpis podgrafa:

Vsebino LT in FT lahko beremo s pomočjo paketov RDLT, RDFTR, RDFTL in RDFTT, ki pomenijo zahtevo po branju. Prebrana vsebina lokacij v LT in FT pa se nahaja v izhodnih paketih LTRDD, FTRRDD, FTLRDD in FTTRDD.

Zahteva za branje iz LT (vh. paket RDLT):

MN	0	adr. v LT	10001

Prebrani podatki iz LT (izh. paket LTRDD):

0000	0	adr. v LT	11000	podatki iz LT

Zahteva za branje iz FTR (vh. paket RDFTR):

MN	0	adr. v FT	11001

Prebrani podatki iz FTR (izh. paket FTRRDD):

0000	0	adr. v FT	11001	podatki iz FTR

Zahteva za branje iz FTL (vh. paket RDFTL):

MN	0	adr. v FT	11010

Prebrani podatki iz FTL (izh. paket FTLRDD):

0000	0	adr. v FT	11010	podatki iz FTL

Zahteva za branje iz FTT (vh. paket RDFTT):

MN	0	adr. v FT	11011

Prebrani podatki iz FTT (izh. paket FTTRDD):

0000	0	adr. v FT	11011	podatki iz FTT

3.2.4. Izvrševanje podgrafa:

Ko je podgraf vpisan v procesor, se lahko prične njegovo izvrševanje - pretok podatkov po podgrafu. Procesor dobi vhodne podatke (operande) s pomočjo vhodnih paketov EXEC, rezultate pa vrne s paketi OUTD.

Vhodni podatki (vh. paket EXEC):

MN	0	ID	00CS1	operand

Izhodni podatki (izh. paket OUTD):

MN	0	ID	00CS1	rezultat

3.2.5. Dumpanje:

V prekinitvenem načinu delovanja je omogočeno tudi opazovanje vsebine delov paketov. V ta namen se uporabljata vhodni paket DUMP ter pripadajoči izhodni paket DUMPD.

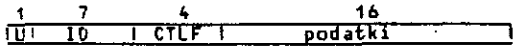
Zahteva za izpis danega polja (vh. paket DUMP):

MN	0	x(4)	DUMP(3)	10111

```
Izpisana vsebina polja (izh. paket DUMPD):
|0000|010(4)DUMP(3)|011111|   dumpani podatki   |
Opomba: glede na bite DUMP(3) dobimo:
        DUMP(3)      dumpani podatki
         000    x(5) GQ(5) DQ(6)
         001    x(4) u(1) ID(7) CTLF(4)
         010    DATA(16)
         011    x(3) u ID(7) x C S C S
         100    xx FTL(spodnjih 12) xx
         101    DATA (16)
         110    DATA (16)
         111    x(9) ID(7)
```

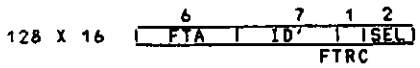
3.3. Tabela povezav LT

LT je 128 X 16 bitni dinamični RAM. V LT vstopi 28 bitni paket kot ga prikazuje Slika 6.



Slika 6: 28 bitni LT paket.

Identifikator LT paketa, ki ga pošlje IC, služi za naslovitev lokacije v LT. Vsebina naslovljene lokacije vsebuje 6 bitni naslov lokacije v tabeli točk (FTA), 7 bitni identifikator (ID'), FTRC bit in 2 bitno selekcijsko polje (SEL). Prikazana je na Sliki 7.



Slika 7: Vsebina lokacije v LT.

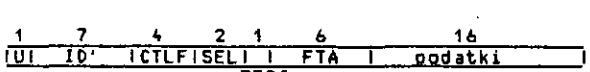
ID LT paketa se nadomesti z novim ID', vsebovanim v naslovljeni lokaciji v LT. Torej vsakokrat, ko paket preide čez LT, dobi novi identifikator. LT paketu se dodajo še FTA, FTRC in SEL polje. FTA polje služi za dostop do lokacije v FT. FTRC bit in SEL polje pa služita pri določanju tipa ukaza.

SEL	Tip ukazov
11	AG/FC
01	PU
10	GE
00	OUT

Opomba: Tipi ukazov so opisani v poglavju 3.11.

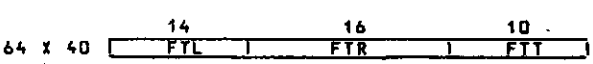
3.4. Tabela točk FT

FT je 64 X 40 bitni dinamični RAM. Vanj vstopijo paketi, ki jih pošlje LT v obliki, kot jo prikazuje Slika 8.



Slika 8: 37 bitni FT paket.

FTA polje je naslov lokacije v FT, katere vsebino sestavljajo 14 bitno polje FTL, 16 bitno polje FTR in 10 bitno polje FTT, ki vsebujejo krmilne podatke o različnih tipih ukazov. FT lokacijo prikazuje Slika 9.



Slika 9: Vsebina lokacije FT.

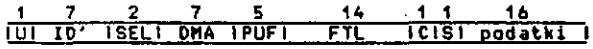
SEL polje FT paketa določa tip ukaza. Ukazi so lahko tipa OUT, GE, PU ali AG/FC. Ukazi tipa OUT narekujejo neposreden prenos paketa v OQ. Ukazi tipa GE se uporabljajo za generiranje paketov znotraj procesorja. PU ukazi omogočajo aritmetično logične operacije, AG/FC ukazi pa služijo AG&FC enoti pri delu z DM. Pri ukazih tipa PU se uporablja FTL polje, pri ostalih ukazih pa tudi FTR in FTT polje.

3.5. Generator naslovov in krmilnik pretoka AG&FC

AG&FC generira naslove lokacij v DM ter krmili njihovo branje in vpisovanje. AG&FC ugotavlja, če prispeli paket vsebuje ukaze z enim ali dvema operandoma. Če potrebuje en operand, se paket prenese direktno v Q. Če pa potrebuje ukaz dva operanda, morata biti na voljo oba, da se lahko prenese v Q. Paket, ki vsebuje prevega izmed prispelih operandov, se začasno shrani v DM, dokler ne prispe paket z drugim operandom. Ko paket z drugim operandom izstopi iz FT, AG&FC generira naslov lokacije v DM, v kateri je shranjen paket s prvim operandom in pošlje oba v Q.

3.6. Paketni pomnilnik DM

DM je 512 X 18 bitni dinamični RAM, ki služi za hranjenje prvega od prispelih operandov ukaza. Paket, ki je nosilec drugega operanda, vstopi v DM v obliki prikazani na Sliki 10.



Slika 10: 54 bitni DM paket.

Polje DMA vsebuje naslov lokacije v DM v kateri je shranjen prvi operand. Podatki o ukazu namenjenemu PU se nahajajo v FTL. DM se uporablja tudi za začasno hranjenje drugih, npr. vhodno-izhodnih podatkov.

3.7. Vrsta Q

Q je 48 X 60 bitni dinamični RAM, ki služi kot FIFO pomnilnik. Paketi, ki vstopajo v Q, nastanejo iz DM paketa tako, da se DMA polje izloči in nadomesti z vsebino lokacije DM na katero je prej kazalo polje DMA (prvi operand). Q začasno hrani pakete, namenjene v PU ali OQ. Q je razdeljen v dva FIFO pomnilnika: 32 X 60 bitno podatkovno vrsto (DQ) in 16 X 60 bitno generatorsko vrsto (GQ). DQ je namenjen ukazom tipa PU, OUT in AG/FC in hrani pakete, ki so namenjeni v PU ali OQ. Vrsta GQ pa je namenjena le ukazom tipa GE, ki generirajo nove pakete. Vrsta DQ zadržuje izhodne pakete, če je izhodna vrsta OQ polna. Podobno DQ zadržuje pakete, namenjene v PU, če je le-ta zasedena. Nekritično pošiljanje paketov v krožni obtok bi lahko privedlo do nasičenja vrste Q, zato je delovanje Q omejeno s sledečo zahtevo: če se v DQ nahaja osem ali več paketov, je branje iz GQ onemogočeno, če pa je v DQ manj kot osem paketov, ima branje iz GQ višjo prioriteto od branja iz DQ. Do nasičenja vrste Q bi lahko prišlo še v primeru, ko bi bila hitrost procesiranja manjša od hitrosti prihajanja novih vhodnih paketov v procesor. Zato v primeru, ko se v DQ nahaja več kot 23 paketov, procesor preide v prekinitveni način delovanja, s čimer se izogne nasičenju Q.

3.8. Izhodna vrsta OQ

OQ je 8 X 32 bitni statični RAM, organiziran kot FIFO pomnilnik. Služi začasnemu shranjevanju izhodnih paketov, prispelih iz DQ, ki se nato preko OC pošljejo na izhodno podatkovno vodilo. Če je OQ poln, pošlje ustrezen signal v DG, ter preneha sprejemati pakete. OQ paket se ujema s Q paketom.

3.9. Procesna enota PU

PU izvršuje ukaze tipa PU in GE. Koda PU ukaza se nahaja v polju FTL PU paketa. Ukazi tipa GE se uporabljajo za generiranje novih paketov, kopiranje posameznih ali skupin paketov in spreminjanja vsebine CTLF polja. Če potrebuje PU za izvršitev ukaza več kot en cikel, pošlje signal vrsti Q in IC, ki zadrži njuno delovanje. PU paket je po obliki enak Q paketu.

3.10. Osveževalni krmilnik RC

RC avtomatično generira pakete za osveževanje vseh dinamičnih RAMov v procesorju. Paket, ki ga generira RC, vstopi v IC, ter nato po vrsti še v LT, FT, DM in Q in vsakokrat povzroči osvežitev ustreznega RAMa. Po prihodu v Q se osvežitveni paket uniči.

3.11. Nabor ukazov

Kot smo omenili v poglavju 3.4. obstajajo štirje tipi ukazov: AG/FC, PU, GE in OUT.

3.11.1. Ukazi tipa AG/FC:

Tip AG/FC vsebuje 16 ukazov, ki jih razdelimo v tri skupine: AG, FC in AG/FC tip. Ukazi tipa AG so: RDCYCS, RDCYCL, WRCYCS, WRCYCL, RDWR in RDIDX. Ukazi tipa FC so: PICKUP, COUNT, CUT, DIVCYC, DIV, DIST, CONVO, SAVE in CNTGE. Ukaz QUEUE pa je tipa AG/FC.

Ukazi tipa AG/FC so opisani v FTR polju tabele FT, kjer zgornji štirje biti predstavljajo operacijsko kodo. Ostali del polja FTR in polje FTT pa vsebujejo ostale parametre AG/FC ukazov. Pomeni ukazov tipa AG/FC so:

- QUEUE: shrani prvi prispeli operand dvomestnega ukaza v DM.
- RDCYCS: ciklično branje podatkov iz izbranega področja v DM, kjer je največja dolžina področja 16 lokacij.
- RDCYCL: enako kot RDCYCS, le da je največja dolžina področja 256 lokacij.
- WRCYCS: ciklično vpisovanje podatkov v izbrano področje v DM, kjer je največja dolžina področja 16 lokacij.
- WRCYCL: enako kot WRCYCS, le da je največja dolžina področja 256 lokacij.
- RDWR: branje ali vpisovanje v DM. Z bitom FTRC izbiramo bodisi branje ali vpis.
- RDIDX: branje iz DM.
- PICKUP: izloči vsak n-ti paket in mu priredi ID + 1.
- COUNT: podvoji vsak n-ti paket in mu priredi ID + 1.
- DIVCYC: na vsakih n paketov izloči prvih m med njimi in jim priredi ID + 1.
- DIV: po vsakem prihodu paketa s FTRC = 1 se naslednjim n paketom ID ne spremeni, ostalim pa se priredi ID + 1 (paket s FTRC = 1 se uniči).
- DIST: zaporedje vhodnih paketov vsebuje pakete s FTRC = 0 ali 1. Po prihodu j-tega paketa s FTRC = 1 se vsem nadaljnim paketom, ki imajo FTRC = 0 in naslednjemu paketu s FTRC = 1, priredi ID + (j MOD k).
- SAVE: uporablja se pri nastavljanju ID polja.
- CUT: po vsakem prihodu paketa s FTRC = 1 se se ta paket skupaj z naslednjimi n paketi uniči, ostali pa ostanejo nespremenjeni.
- CONVO: uporablja se za komulativno računanje, npr. vsot in produktov.
- CNTGE: uporablja se pri generiranju več kot 16 kopij danega paketa (skupaj z ukazom COPYBK tipa GE, ki je opisan spodaj).

Konstante k, m in n so podane v FTT polju.

3.11.2. Ukazi tipa PU:

Ukazi tipa PU so shranjeni v FTL polju tabele FT. Uporabnih je le 12 spodnjih bitov, med katerimi so biti 0 do 4 operacijska koda ukaza, preostali biti pa se uporabljajo za dodatno informacijo o številu operandov (eden ali dva), legi operandov pri nekomutativnih operacijah, številu rezultatov (eden ali dva) ter tipu primerjave, podanim z biti PNZ (EQ, LT, LE, GT, GE in NE). Ukazi tipa PU so:

- Logični: OR, AND, EXOR, ANDNOT in NOT.
- Aritmetični: ADD, SUB, MUL, INC, DEC, NOP in ADDSC, SUBSC, MULSC ter NOPSC. Ukazi *SC najprej izračunajo operacijo * in vrnejo lego skrajnega levega setiranega bita.
- Premikalni: SHL in SHR ter SHLBRV in SHRBRV, kjer ukaza *BRV predhodno obrneta vrstni red bitov.
- Primerjalni: CMPNOM, CMP in CMPXCH. Vhodna operanda A in B se primerjata na način podan s PNZ poljem. Rezultata primerjave sta X in Y, kot sledi

			C_x	S_x	X	C_y	S_y	Y
CMPNOM	pri	PNZ=FALSE	0	0	0000H	0	0	0000H
	pri	PNZ=TRUE	1	0	0001H	1	0	0000H
CMP	pri	PNZ=FALSE	0	S_a	A	0	S_b	B
	pri	PNZ=TRUE	1	S_a	A	1	S_b	B
CMPXCH	pri	PNZ=TRUE	C_a	S_a	A	C_b	S_b	B
	pri	PNZ=FALSE	C_b	S_b	A	C_a	S_b	A

- Operacije nad biti: GET1, SET1 in CLR1.
- Testiranje bitov: ANDMSK in ORMSK.
- Pretvorba podatka: CVT2AB in CVTAB2. CVT2AB pretvori 16 bitni operand, zapisan v sistemu z dvojiškim komlementom, v 17 bitno besedo v sistemu predznak-absolutna vrednost. Predznak se nahaja v bitu S . Drugi ukaz je inverzen prvemu. Ob prekoračitvi se postavi bit C .
- Popravki pri dvojni natančnosti: ADJL. Če imata besedi, s katerima je zapisan večnatančnostni operand, različna predznaka, ju ADJL popravi.
- Kumulativno seštevanje: ACC. Ukaz povzroči seštetje podatkovnih polj v zaporednju paketov. Vsebina vsakega prispelega paketa se prišteje k delni vsoti v registru ACC. Paketi v zaporedju so bodisi tipa 1 ali 2. Paket tipa 1 se po prištetju svoje vsebine zbriše, paket tipa 2 pa po prištetju svoje vsebine prevzame vsebino registra ACC.
- Kopiranje kontrolnega bita : COPYC.

3.11.3. Ukazi tipa GE:

Ukazi tipa GE so opisani v polju FTL tabele FT. Operacijsko kodo sestavljata dva bita, preostali biti pa določajo: če je treba operanda pred vstopom v Q zamenjati, število operandov ter število kopij pri generiranju paketov. V to skupino sodijo trije ukazi:

- COPYBK: generiranje bloka kopij danega paketa. Vsi novi paketi imajo enak ID kot originalni paket, le zadnji ima ID + 1. Vrednosti v podatkovnih poljih se lahko povečujejo ali zmanjšujejo za konstantno vrednost.
- COPYM: generiranje skupine kopij danega paketa z različnimi ID. Podatkovno polje se lahko spreminja podobno kot pri ukazu COPYBK.
- SETCLT: branje ali spreminjanje vsebine tabel LT in FT. Ukaz SETCLT omogoča programe, ki se sami spreminjajo, saj se uporablja v času izvrševanja grafa.

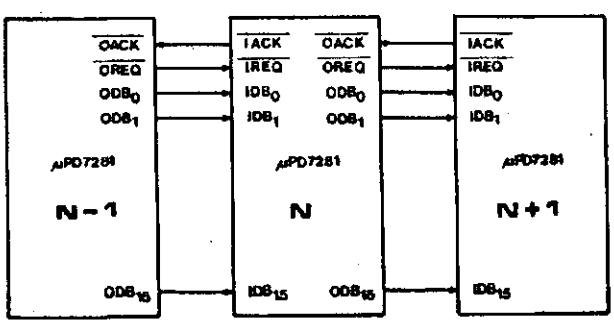
3.11.4. Ukazi tipa OUT:

Ukazi tipa OUT so opisani v FTL polju tabele FT. Polje vsebuje podatke o tem, če nastopa OUT ukaz samostojno ali skupaj z nekim AG/FC ukazom, če je treba operanda pred vstopom v OQ zamenjati, operacijsko kodo OUT ukaza ter naslov procesorja MN, kateremu je namenjen izhodni paket. Ločimo dva ukaza tipa OUT:

- OUT1: povzroči prenos 32 bitnega paketa na izhodno vodilo. Paket vsebuje podatek, njegov identifikator, ter naslov procesorja, kateremu je namenjen.
- OUT2: povzroči prenos 64 bitnega paketa na izhodno vodilo. Ukaz se uporablja pri pošiljanju števil z dvojno natančnostjo.

3.12. Načini povezovanja μPD7281

Procesorji μPD7281 se povezujejo v večprocesorski sistem na dva osnovna načina: kaskadni in krožni. Pri kaskadnem povezovanju ni potrebna dodatna materialna oprema. Največ 14 čipov povezujemo neposredno, kot prikazuje Slika 11. Izmenjava vhodno-izhodnih paketov poteka s pomočjo signalov zahteva/potrditev (REQ/ACK).



Slika 11: Povezovanje procesorjev μPD7281.

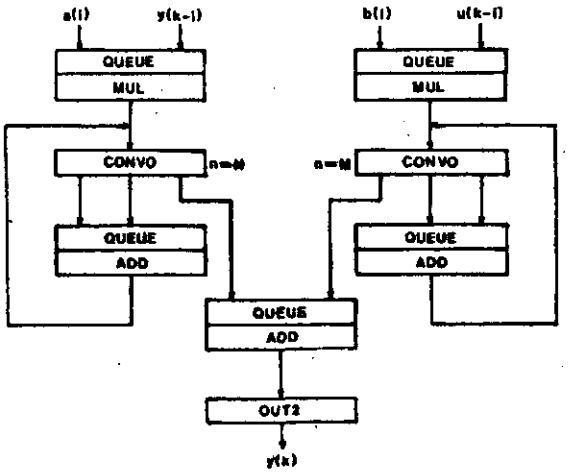
Za krožno arhitekturo je potrebna dodatna materialna oprema, zato je v razvoju podporni čip MAGIC (Memory Access and General bus Interface Chip).

3.13. Programiranje μPD7281

Bogat nabor ukazov omogoča zelo učinkovito programiranje problemov s področja obdelave signalov ter zahtevnega numeričnega računanja. Za ilustracijo si oglejmo realizacijo linearnega digitalnega filtra [5], podanega z enačbo

$$y(k) = \sum_{i=1}^{N} a(i)y(k-i) + \sum_{i=1}^{M} b(i)u(k-i),$$

kjer je u(k) vhodni in y(k) izhodni signal. Pripadajoči program (graf), zapisan v 'strojnem' jeziku, prikazuje Slika 12.



Slika 12: Rekurzivni izračun digital. filtra.

Seveda je razvit tudi zbirni jezik, ki omogoča lažji opis programskega podgrafa [23]. Tudi uporabo zbirnika ilustrirajmo na primeru linearnega digitalnega filtra, tokrat realiziranega v prostoru stanj, kot ga podajata enačbi

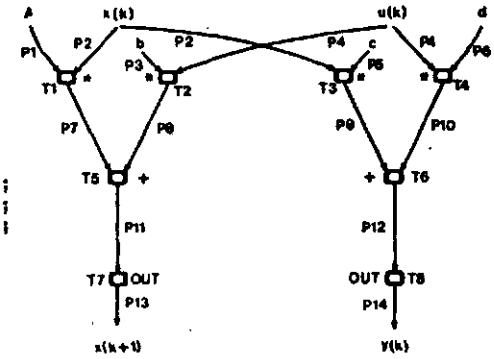
$$x(k+1) = Ax(k) + bu(k)$$
$$y(k) = cx(k) + du(k),$$

kjer so A, b, c in d usrezne matrike in vektorji. Program bi se v primeru, ko so matrike in vektorji enodimenzionalni, glasil:

```
EQUATE    HOST  = 0;
MODULE    PROC1 = 1;
INPUT     P1,P2,P3,P4,P5,P6;
OUTPUT    P13,P14;
LINK      P7  = T1 (P1, P2);
LINK      P8  = T2 (P3, P4);
LINK      P9  = T3 (P2, P5);
LINK      P10 = T4 (P4, P6);
LINK      P11 = T5 (P7, P8);
LINK      P12 = T6 (P9, P10);
LINK      P13 = T7 (P11,  );
LINK      P14 = T8 (P12,  );
FUNCTION T1 = MUL,QUEUE(Q1,1);
FUNCTION T2 = MUL,QUEUE(Q2,1);
FUNCTION T3 = MUL,QUEUE(Q3,1);
FUNCTION T4 = MUL,QUEUE(Q4,1);
FUNCTION T5 = ADD,QUEUE(Q5,1);
FUNCTION T6 = ADD,QUEUE(Q6,1);
FUNCTION T7 = OUT1(HOST,0);
FUNCTION T8 = OUT1(HOST,0);
MEMORY    Q1 = AREA(1);
MEMORY    Q2 = AREA(1);
MEMORY    Q3 = AREA(1);
MEMORY    Q4 = AREA(1);
MEMORY    Q5 = AREA(1);
MEMORY    Q6 = AREA(1);
```

Zgornji program se izvaja samo v procesorju PROC1, čeprav je v algoritmu prisotna določena stopnja vzporednosti. Težava je v tem, da je potencialna vzporednost slabo razvidna, saj je to le enodimenzionalni zapis programskega (pod)grafa. Zato običajno poteka programiranje tako, da najprej konstruiramo programski graf, določimo vzporedno izvršljive podgrafe ter jih zapišemo v zbirniku. Zgornji program izhaja iz grafa, ki je podan na Sliki 13. Opazimo, da se lahko levi (izračun x(k+1)) in desni (izračun y(k)) del grafa izvršujeta vzporedno, zato levi del priredimo procesorju PROC1, desni del pa procesorju PROC2. Ustrezna programa, ki sta v tem primeru zelo podobna, sta podana na Sliki 13. Razlika med njima je le v tem, da PROC1 pošilja rezultat procesorju PROC2, medtem ko procesor PROC2 pošilja rezultat glavnemu procesorju HOST.

```
EQUATE    PROC2 = 2;
MODULE    PROC1 = 1;
INPUT     P1,P2,P3,P4;
OUTPUT    P13;
LINK      P7  = T1 (P1, P2);
LINK      P8  = T2 (P3, P4);
LINK      P11 = T5 (P7, P8);
LINK      P13 = T7 (P11,  );
FUNCTION  T1 = MUL,QUEUE(Q1,1);
FUNCTION  T2 = MUL,QUEUE(Q2,1);
FUNCTION  T5 = ADD,QUEUE(Q5,1);
FUNCTION  T7 = OUT1(PROC2,0);
MEMORY    Q1 = AREA(1);
MEMORY    Q2 = AREA(1);
MEMORY    Q5 = AREA(1);
```




```
EQUATE    HOST  = 0;
MODULE    PROC2 = 2;
INPUT     P2,P4,P5,P6;
OUTPUT    P14;
LINK      P9  = T3 (P2, P5);
LINK      P10 = T4 (P4, P6);
LINK      P12 = T6 (P9, P10);
LINK      P14 = T8 (P12,  );
FUNCTION  T3 = MUL,QUEUE(Q3,1);
FUNCTION  T4 = MUL,QUEUE(Q4,1);
FUNCTION  T6 = ADD,QUEUE(Q6,1);
FUNCTION  T8 = OUT1(HOST,0);
MEMORY    Q3 = AREA(1);
MEMORY    Q4 = AREA(1);
MEMORY    Q6 = AREA(1);
```

Slika 13: Realizacija digitalnega filtra v prostoru stanj.

Pri večini programskih grafov je določanje vzporedno izvršljivih podgrafov zahtevno, kar narekuje razvoj metod za avtomatično iskanje vzporednosti ter optimizacijo glede na število uporabljenih procesorjev [4]. Zelo dobrodošel bi bil grafični zbirnik za generiranje kode neposredno iz programskega grafa ter pascalski ali C prevajalnik za generiranje in optimizacijo programskih grafov.

4. ZAKLJUČEK

Rezultati testov opravičujejo uporabo večprocesorske arhitekture z μPD7281 [1]. Tako npr. rotacija binarne slike velikosti 512 X 512 točk zahteva 0.6s pri krožni povezavi treh procesorjev; en procesor pa potrebuje 1.5s. Za izračun funkcije cos(x) potrebuje en procesor 40μs, kaskada treh procesorjev pa 15μs. V splošnem se čas obdelave eksponentno zmanjšuje z večanjem števila uporabljenih procesorjev, žal pa ne neomejeno, saj se pri večjem številu procesorjev pojavijo zastoji pri pretoku vhodno-izhodnih paketov med procesorji. Zato je odvisno od same aplikacije, ali jo bomo razbili glede na podatke ali na programski graf. Vzemimo npr. obdelavo slike, ki obsega naloge kot so premik, normiranje, rotacija itd. Če se odločimo za razbitje glede na podatke, razbijemo sliko na podslike, tako da se vsaka podslika istočasno obdela v svojem procesorju. To pomeni, da je v vseh procesorjih cel programski graf, vanj pa vstopajo le podatki o pripadajoči podsliki. Razbitje glede na graf (program) pa pomeni, da prvi procesor opravi premik slike, drugi normiranje, tretji rotacijo itd. To pomeni, da je v vsakem procesorju drug podgraf in vanj vstopajo podatki o celi sliki. Seveda pa je razbitje grafa smiselno le tedaj, ko se lahko podgrafi izvršujejo vzporedno.

5. LITERATURA

[1] Chong Y.M.: Data Flow Chip Optimizes Image Processing, Computer Design, Oct. 15, 1984, pp.97-103

[2] Jeffery T.: The μPD7281 Processor, Byte, November 1985, pp.237-246

[3] μPD7281 Image Pipelined Processor, NEC Electronics, February 1985

[4] Robič B., J.Šilc: On Choosing a Plan for the Execution of Data Flow Program Graph, Informatica 3/86

[5] Ohba N., T.Saito, Y.Hoshiko: Signal Processing on a Data-Flow Processor, Microprocessing and Microprogramming 14,1984, pp.17-27

TRANSPUTER – OSNOVNI GRADNIK VEČPROCESORSKIH SISTEMOV

UDK: 681.3-181.4

Branko Mihoilović, Slavko Mavrič, Peter Kolbezen
Institut »Jožef Stefan«, Ljubljana

Transputer je osnovni VLSI materialni gradnik večprocesorskih sistemov, ki za povezovanje z ostalimi transputerji v sistemu koristi "point-to-point" komunikacijske povezave. Jezik OCCAM je osnovni programski gradnik večtransputerskega sistema (VTS). S tem jezikom opišemo VTS kot zbir procesov, ki se izvajajo sočasno in med seboj komunicirajo na nivoju sporočil preko definiranih komunikacijskih kanalov.

TRANSPUTER-THE BASIC COMPONENT OF MULTIPROCESSOR SYSTEMS - The Transputer is a basic VLSI hardware component of multiprocessor systems with communication links for point-to-point connection to other transputers. Occam is a language that enable a multiprocessor system to be described as a collection of processes that operate concurrently and communicate using messages passing via named channels.

1. Uvod

Danes je popolnoma jasno, da tudi ekonomskih preprek ni več, ki bi zavirale gradnjo "močnih" računalniških sistemov; in sicer takšnih, ki v sebi združujejo naprimer 1000 sočasno delujočih procesorjev. Takšen računalnik je preprosto "prilagojen" aplikaciji v smislu popolne izkoriščenosti inherentnih sočasnosti. Takšno gledanje je vodilo tudi strokovnjake firme INMOS pri načrtovanju transputerja. Čemu so transputerji in tudi njim podobne komponente namenjene?

- Namenjene so načrtovanju standardnih računalniških produktov v smislu lažjega programiranja in uporabe le-teh,
- pridobivanju kar največje učinkovitosti s temi komponentami zgrajenih sistemov,
- koriščenju novih razvojnih smeri v VLSI tehnologiji in sicer znotraj kompatibilnih družin,
- načrtovanju takšnih programabilnih komponent s pomočjo katerih bi gradili sisteme z velikim številom sočasno delujočih računalniških sistemov.

Transputer, ki je zaščiteno ime za mikroračunalnik (mikroprocesor), je po velikosti in funkcijah primerljiv z mikroprocesorji kot sta NS 16032 ali I 80286, vendar je pri njem uporabljena povsem drugačna teoretska osnova pri načrtovanju velikih računalniških sistemov.

Transputer je zgrajen tako, da je lahko povezan z ostalimi transputerji v takoimenovani mreži paralelnih procesov.Vsak od transputerjev izvaja točno določeno opravilo (proces) z minimalno izgubo procesnega časa in minimalnim balastom, ki se lahko pojavi v fizični povezavi med transputerji. Pri računalnikih, ki potrebujejo velike podatkovne baze, ki so programirani v naravnih jezikih in omogočajo tudi druge "inteligentne" aktivnosti, predstavlja sočasno procesiranje eno od bistvenih lastnosti takšnih računalnikov. Če želimo to doseči z medsebojnim povezovanjem konvencionalnih procesorjev, naletimo na praktično nerešljive probleme varnega dodeljevanja virov, sinhronizacije pri takoimenovanih paralelnih vodilih in podobno. V transputerski arhitekturi koristimo visoko stopnjo sočasnega izvajanja procesov. Ta je zagotovljena preko takoimenovanega decentraliziranega računalniškega modela. Lokalni procesi se izvajajo nad lokalnimi podatki, med seboj pa takšni procesi komunicirajo preko hitrih komunikacijskih kanalov z izmenjavo sporočil. Tako izbran način lokalnega procesiranja in komuniciranja narekuje temeljite spremembe v konceptu programiranja in načrtovanju novih algoritmov.

Sočasno z razvojem transputerja, je potekal razvoj na majhnem (22 rezerviranih besed), izbranem jeziku, imenovanem OCCAM. Namenjen je programiranju transputerjev, odnosno je njihov zbirni jezik. Osnovna lastnost (in prednost) tega jezika je ta, da je ustvarjen za programiranje sočasnih aktivnosti v večtransputerskem sistemu; to je za učinkovito implementacijo transputerskega sistema.

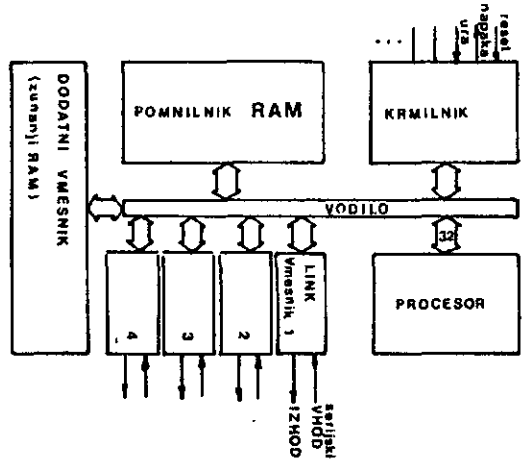
2. Zgradba sistema

Na tem mestu je potrebno povdariti naslednje: Zgradba tako samega transputerja kot

transputerskega sistema je definirana z OCCAM-om. S tem, da smo arhitekturo definirali na tem nivoju pomeni, da lahko na eni strani transputerski sistem zgradimo z različnimi, danes že standardnimi komponentami, na drugi strani pa lahko tak sistem optimiziramo za različne aplikacije.

Glavna sestavna dela transputerja sta 32 bitni RISC (reduced instruction set computer) in zelo hiter RAM pomnilnik (4K besed). Čip je narejen v 2. mikronski CMOS tehnologiji. 5MHz. ura mu omogoča,da izvrši 10 milionov instrukcij na sekundo, kar zdaleč presega zmožnosti današnjih mikroračunalnikov. Blok shemo transputerja prikazuje slika 1.

Transputerji standardne Von-Neumannove strukture se povezujejo med seboj s pomočjo štirih komunikacijskih povezav in tako tvorijo sistem, katerega lahko pogledamo z dveh strani. Prvi, logični pogled nam pove, kako je sistem med seboj povezanih transputerjev načrtovan in programiran. Drugi pogled, imenovan fizični, nam pove, kako so VLSI komponente med seboj povezane in krmiljene.



SLIKA 1

2.1. Logična zgradba transputerskega sistema

Pri podrobnejšem opisu logične zasnove sistema, je potrebno opozoriti na osnovno načelo tega vidika in to je, da lahko transputer in transputerski sistem programiramo (poleg OCCAM-a) tudi v drugih visokih programskih jezikih, transputer pa bo prevedene programe zagotovo učinkovito izvajal. Program, ki ga transputer izvaja je formalno ekvivalenten procesu v OCCAM-u. Proces je tako materialni in programski gradnik večprocesorskega sistema. Mrežo transputerjev direktno popisujemo z OCCAM-skim programom. Načrtovanje VTS pomeni v bistvu povezovanje množice procesov med seboj. Notranja zgradba procesne enote (transputerja) je na tem nivoju prikrita in povsem nepomembna. Proces, ki ga takšna enota izvaja, je popolnoma določen s pomočjo podatkov, ki jih procesna enota sprejema odnosno oddaja.

Jezik OCCAM se popolnoma podreja temu konceptu. Tisto kar je v OCCAM-u definirano kot proces, je dejansko realizirano kot transputer. Pravimo, da se v transputerju izvaja množica takoimenovanih komunikacijskih procesov, ki v svoji notranji zgradbi koristijo "point-to-point" komunikacijske povezave. Slednje so izvedene tako, da omogočajo preprosto povezovanje med transputerji in s tem enostavno gradnjo VTS. Naštejmo nekaj prednosti, ki jih imajo PTP kom. povezave v primerjavi z vodili v VPS:

- Ni nikakršnih spornosti v komunikacijah med elementi VPS, ne glede na število transputerjev v sistemu.
- Ni nikakršnega povečanja obremenitve komunikacijskih poti pri povečanem številu transputerjev v sistemu.
- Zaradi povečanja sistema ne more priti do zasičenja v takoimenovanem komunikacijskem območju. Vse povezave med transputerji so kratke in imajo lokalni značaj.

Komunikacijske poti so grajene tako, da omogočajo preprosto programiranje večtransputerskega sistema. Gre za enostavno obravnavanje sočasnosti v komunikacijah. Sinhronizacija med procesi na vseh štirih komunikacijskih poteh je avtomatska in ne zahteva nikakršnega dodatnega programiranja. Podatki se med transputerji prenašajo serijsko Na pravilno sprejete podatke transputer odgovori s potrditvijo.

2.2. OCCAM model

V OCCAM-u se procesi povezujejo tako, da tvorijo sistem sočasnih končnih procesov. Vsak proces ima začetek, akcijski del in konec. Akcijski del lahko ponovno sestavljajo bodisi sekvenčni procesi ali paralelni procesi. V vsakem OCCAM-skem programu ločimo tri osnovne procese: vhodni, prireditveni ter izhodni proces. Prireditveni proces postavi določene vrednosti spremenljivk in izračuna vrednost nekega izraza. Vhodni in izhodni proces skrbita za prenos vrednosti med occamskimi spremenljivkami po takoimenovanih occamskih kanalih. Zapis

```
c ! x
```

pomeni prenos vrednosti spremenljivke x v kanal z imenom c; odnosno

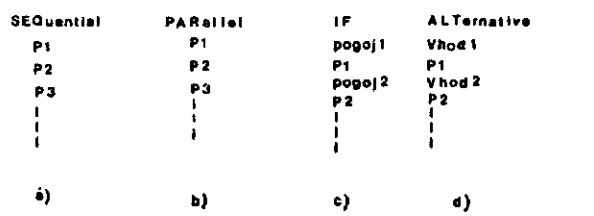
```
c ? y
```

pomeni zapis vrednosti iz kanala c v spremenljivko y. Seveda so kanali, tako kot spremenljivke deklarirani na začetku procesa, ki dotične kanale uporablja. Prireditveni proces ima naslednjo obliko:

```
v := e
```

kar pomeni, da spremenljivka v prejme vrednost izraza e.

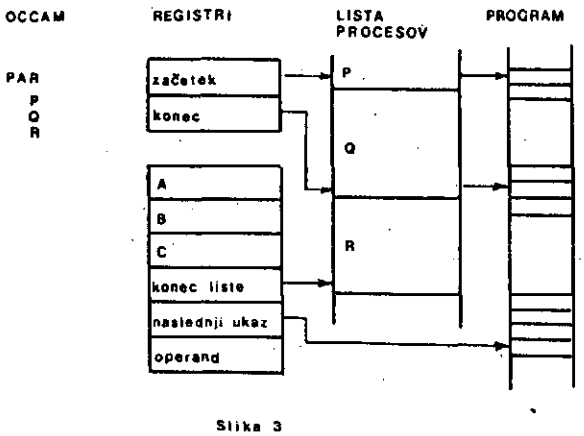
Poleg opisanih treh osnovnih procesov, predstavljajo konstrukti najpomembnejši del jezika OCCAM. Ločimo štiri osnovne konstrukte in sicer: sekvenčni, paralelni, pogojni in izbirni. Osnovna lastnost konstruktov je ta, da se začnejo s karakteristično označbo čemur sledi lista osnovnih procesov. Sekvenčni konstrukt je ponazorjen na sliki 2a.



SLIKA 2

Izvrševanje vsakega naslednjega procesa se začne takrat, ko je predhodni končan. V primeru paralelnega konstrukta (slika 2b) se vsi procesi izvajajo sočasno, konstrukt pa se zaključi takrat, ko so končani vsi procesi. Veljavnost pogoja 1 je pogoj za izvrševanje procesa P1 (slika 2c). V primeru na sliki 2d pa bo proces P1 začel z izvrševanjem takrat, ko se bo izvršil vhodni proces 1.

Tako kot v sekvenčnih jezikih, poznamo tudi tu tipe, deklariranje, polja, posebnost pa predstavlja takoimenovano oblikovanje occamskega programa. V occamu zapisan program se lahko izvaja na enem ali več transputerjih, zato mora biti ustrezno temu pravilno oblikovan. Razvojni sistem namreč omogoča, da se program, ki se bo izvajal na večih transputerjih, tudi pravilno "porazdeli" mednje. Oblikovanje programa (configuration of occam program) ne upliva na logično zasnovo programa. Z konstruktom PLACED PAR se namreč zagotovi, da se bo vsak proces, podan v paralelnem konstruktu, lahko izvajal na svojem transputerju.



Slika 3

2.3. Transputer T424

Procesor z naborom instrukcij je zasnovan tako, da omogoča:

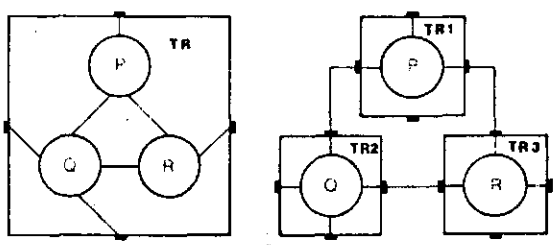
- učinkovito uporabo jezika OCCAM, posebno pri izvajanju večjega števila procesov,
- enostavno prevajanje programa, pri vsem tem pa ni potrebno definirati arhitekture na nižjem nivoju,
- izvajanje programov na procesorjih z različnimi dolžinami besed, in sicer brez predhodnega dodatnega prevajanja,
- lociranje programa in delovnega pomnilniškega prostora kjerkoli v pomnilniku transputerja,

Na učinkovitost celotnega transputerskega sistema pa vplivata vglavnem dva činitelja: število besed v celotnem programu in hitro izvajanje le-tega. Posebna skrb je posvečena ravno drugemu činitelju, saj se tako pomnilniške kot aritmetično-logične operacije izvršijo v enem ali kvečjemu dveh ciklih.

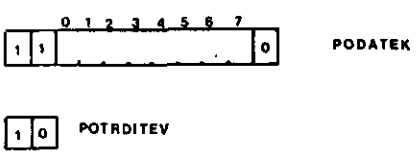
Procesor koristi za naslavljanje pomnilniških lokacij linearni adresni prostor,pri tem pa ne razlikuje med notranjim delovnim pomnilnikom in dodanim zunanjim pomnilnikom. Osnovni element linearnega adresnega prostora je kazalec, ki je sestavljen iz dveh delov: naslova pomnilniške besede ter kazalca zloga znotraj pomnilniške besede. Posebni instrukciji (load local pointer, word subscript) podrobnejše opisujeta strukturo pomnilniške besede.

Procesor je organiziran tako, da koristi pri izvajanju sekvenčnih procesov šest registrov. S pomočjo treh registrov lahko oblikuje sklad, shranjuje operande, odnosno parametre klicanih podprogramov. Eden od registrov je namenjen shranjevanju kazalca na vsebino delovnega pomnilnika, drugi pa hrani kazalec naslednje instrukcije, ki se bo izvršila.

Izhodišče za procesorsko multipleksiranje, odnosno dodeljevanje procesorskega časa je podano v multiprogramskem jedru. Aktiven proces čaka na izvršitev potem, ko je uvrščen na listo aktivnih procesov (slika 3). Dodeljevanje procesorskega časa med procese je v multiprogramskem jedru definirana z določeno prioriteto (fixed priority). Z occamskim konstruktom PRI PAR, čemur sledi lista procesov, določimo prioriteto procesom, če se le-ti izvajajo na enem transputerju. Procesor pozna dva prioritetna nivoja. Procesi z nižjo prioriteto se izvajajo samo če ni procesov z višjo prioriteto; njihovi krajši deli (opravila) pa se izvršujejo v periodično dodeljevanih časovnih rezinah. Proces z višjo prioriteto lahko prekine proces z nižjo prioriteto, vendar največ za dolžino ene časovne rezine.



SLIKA 4



SLIKA 5

Vse instrukcije imajo enak format (osem bitne dolžine). Zgornji štirje biti so funkcijski kod, spodnji štirje pa podatkovna vrednost. Takšen instrukcijski format omogoča, da ima procesor preprost in hiter dekodni mehanizem. Tudi dostavni mehanizem je preprost in hiter. Če je procesor 32 bitni pomeni, da v vsakem dostavnem ciklu sprejme 4 instrukcije.

Pri načrtovanju transputerskega sistema se postavlja verjetno najpomembnejše vprašanje, kako je z komunikacijami med procesi. Rekli smo že, da se lahko več procesov izvaja sočasno na enem transputerju ali pa na mreži transputerjev (slika 4). V obeh primerih so v OCCAM-μ za komunikacijo med procesi definirani takoimenovani komunikacijski kanali. V prvem primeru poteka komunikacija med procesoma preko ene pomnilniške besede; komunikacija med transputerji pa poteka preko takoim. point-to-point povezav.. Tako kot v occamskem modelu, se komunikacija lahko začne, ko sta

vhodni in izhodni proces za to pripravljena, kar pomeni, da transputer za sinhronizacijo ne koristi vmesnega pomnilnika. Podatki se med transputerji prenašajo serijsko. Transputer, ki je drugemu posredoval podatke počaka, da mu slednji sprejem tudi potrdi (slika 5).

4. Zaključek

Zaključimo prispevek z naslednjim: Sam transputer ima v bistvu v standardno arhitekturo okvirjen sorazmerno velik delovni pomnilnik in po številu instrukcij skromen a hiter procesor. Komunikacije znotraj transputerja (kljub vodilu) so zelo hitre. Enako hitre so tudi komunikacije med transputerji. Skratka, transputer odstopa od danes že zastarelega koncepta enoprocesorskih računalnikov, katerih osnova je izredno zahtevna programska oprema in pri katerih je že vprašljiva učinkovitost celotnega sistema. Danes poznamo že takoim. interpretacijske sheme za visoke programske jezike, ki omogočajo uporabo preprostih, a zelo učinkovitih instrukcij in prevajanje na nivoju, ki je višji od nivoja mikrokoda. To pomeni boljše razmerje materialna/programska oprema in s tem poenostavljena sistemska programska oprema velikih večprocesorskih sistemov.

5. Literatura

[1] INMOS Limited, OCCAM language overview, November 1985.

[2] INMOS Limited, Transputer architecture, November 1985.

[3] INMOS Limited, IMS T414 Transputer, November 1985.

[4] Dick Pountain, The Transputer and its special language, OCCAM, BYTE, August 1984

[5] Brian Heal, Multiprocessor solution in OCCAM to an NP-complete problem, Microprocessors and Microsystems, Vol. 9, No.4, Maj 1985.

OKOLJE NOVIH TELEMATSKIH STORITEV IN ISDN

UDK: 681.327.8.01

Iztok Tvrdy
Institut Jožef Stefan, Jamova 39, Ljubljana

V članku podajam najprej opis splošnih ciljev razvoja teleinformatike. Pojasnjujem prihajajoči ISDN ter funkcije in telematske storitve v njegovem okviru. Posebno pozornost posvečam integraciji govora in podatkov v distribuiranih komutacijskih sistemih, saj predstavlja zahteva po integraciji tisti problem, ki je najbolj vzpodbudil razrast teleinformatike. Omenjam tudi odnos med tehnologijo in zmožnostmi PBX in LAN, ki je zanimiv za nadaljnji razvoj; pri tem navajam tudi generacijsko klasifikacijo PBX. Na koncu podajam še pregled CCITT priporočil, s poudarkom na serijah V, X, I in T. Na kratko se dotikam tudi lokalnih računalniških mrež in njihove standardizacije.

NEW TELEMATIC SERVICES ENVIRONMENT AND ISDN: General goals of developing the teleinformatics are described in the paper. ISDN, its functions and telematic services are clarified. Problems of Voice/Data integration in distributed switching systems are carefully inspected. For the future developement, interesting relations among technology and abilities of PBXs and LANs are briefly discussed. Generations of PBXs are classified. Overview of CCITT recommendations (with emphasis of the series V, X, I and T), LANs and their standardization are given at the end of the paper.

1. Uvod

Cilji oziroma vizije razvoja teleinformatike kot presečnega področja telekomunikacij in računalništva pokrivajo vse od trenutnih razvojnih nalog pa vse do smernic evolucije teleinformatike in s tem tudi informacijske družbe. Poglavitni del premišljevanja o bodočih telekomunikacijskih omrežjih tvori prihajajoči ISDN (Integrated Services Digital Network, po naše Digitalno omrežje z integriranimi storitvami).

ISDN je digitalno omrežje, prirejeno za prenos informacij med dvema ali več točkami v enakem formatu, ne glede na izvorno naravo podatkov (govor, podatki, besedilo ali slike).

ISDN naj bi nastal kot nadgradnja z združitvijo zmožnosti vseh dosedanjih specializiranih omrežij (telefonskega in telegrafskega omrežja, javnega omrežja za prenos podatkov, različnih privatnih omrežij) ter s hkratno implementacijo novih, doslej šele načrtovanih zmožnosti.

Tako omrežje bo omogočalo številne funkcije. Nekaterim od njih pravimo (zaradi kompleksnosti problematike, ki jo pokrivajo) s posebno besedo telematske storitve. Uporabniško orientirane funkcije v okviru ISDN pa imenujemo tudi telestoritve.

Pri specifikaciji telematskih storitev (in telestoritev) se naslanjamo na referenčni model odprtih sistemov povezovanja (ISO OSI RM), ki nam nudi strukturiran pogled na način komunikacije med dvema ali več uporabniki. Za izvedbo storitev potrebujemo linijo brez napak med končnima uporabnikoma (oziroma procesoma), torej govorimo tedaj le o višjih nivojih referenčnega modela OSI.

Omenimo še dolgoročnejšo perspektivo: širokopasovni ISDN, preko katerega bomo lahko prenašali tudi običajne televizijske kanale in HI-FI radio signale. Tako naj bi bil širokopasovni ISDN tudi nadgradnja kabelske televizije oziroma sedanjega "broadcastinga".

2. Odnos PBX / LAN

Zaradi potrebnih velikih vlaganj v izvedbo javnega ISDN in v zvezi s tem zaradi njegove počasne in postopne izgradnje se bo področje ISDN najprej in najintenzivneje razvijalo na tako imenovanem zasebnem področju. Zato je za nadaljnji razvoj zlasti zanimiv odnos med tehnologijo in zmožnostmi PBX (naročniške telefonske centrale - NTC) in LAN (lokalne računalniške mreže). Še ne dolgo tega ni bilo jasno, katera rešitev je superiorna na področju komunikacij bodočnosti: lokalne računalniške mreže ali naročniške telefonske centrale.

Lokalne računalniške mreže lahko omogočajo zelo hiter prenos podatkov, vendar je zmogljivost mreže, gledano s strani enega uporabnika, obratno sorazmerna s številom hkratnih uporabnikov mreže. Zato je LAN primeren za prenos in komutacijo podatkov, toda zelo težko izvaja (obsežen) prenos govora. Na drugi strani pa na zmogljivost kanala, ki ga vzpostavi uporabnik skozi PBX, število drugih hkratnih uporabnikov ne vpliva. Zato je PBX zelo primeren za govor, manj pa za obsežen prenos podatkov, saj tak prenos zaradi relativne počasnosti zelo dolgo traja. Analogni PBX starejše tehnologije imajo tudi zelo slabe prenosne karakteristike, tako da tudi po tej strani ovirajo prenos podatkov, pri digitalnih PBX pa se ta situacija že precej izboljšuje.

Na osnovi dolge tradicije in čedalje hitrejšega razvoja tehnologije se je izoblikovala naslednja generacijska klasifikacija PBX od ročnih in elektromehanskih sistemov prve generacije do bodočih neblokirajočih distribuiranih sistemov četrte generacije, ki bodo integrirali tehnologiji LAN in PBX:

- 1. generacijo PBX predstavljajo ročni in elektromehanski sistemi v analogni tehnologiji, ki podpirajo le prenos govora;

- 2. generacijo PBX predstavljajo distribuirani računalniško vodeni sistemi (običajno že v povsem digitalni tehnologiji), ki imajo le omejeno zmožnost prenosa podatkov kot dodatek k prenosu govora;

- 3. generacijo PBX predstavljajo neblokirajoči sistemi, to so taki, ki imajo že v zasnovi zmožnost takega prenosa in komutacije podatkov in govora, da zmorejo prenašati vse prometne obremenitve z vseh priključkov hkrati;

- 4. generacijo PBX pa predstavljajo sistemi, ki združujejo prednosti lokalnih računalniških mrež (LAN) (prilagojenost hitremu prenosu podatkov) in PBX tretje generacije (majhna občutljivost za promet, vrojena zmožnost učinkovitega prenosa govora).

Sistemi četrte generacije integrirajo tehnologiji PBX in LAN najpogosteje tako, da je LAN povezovalni element med PBX, PBX pa služijo kot serverji.

Vse doslej so bili računalniško krmiljeni PBX sprogramirani nestrukturirano, in šele v zadnjem času se je ideologija sedemnivojskega OSI modela uveljavila tudi na tem področju. Četrta generacija PBX nudi osnovo za razvoj ISPBX ("ISDN PBX" oziroma "Integrated Services PBX"), ki bo nudil integrirane storitve na podlagi ustreznih standardov. Po svetu so se že začeli pojavljati prvi ISPBX, ki pa rahlo odstopajo od dosedaj sprejetih standardov in priporočil (Plessey). V nadaljevanju si bomo ogledali, kaj sploh pomeni pojem "integracija" v tem konceptu ter kakšni standardi so že sprejeti in kakšne še pričakujemo.

3. Klasifikacija podatkov

Podatke v širšem smislu lahko razdelimo na štiri karakteristične oblike:

- <u>podatki v ožjem smislu</u> imajo zelo strukturirano naravo in so organizirani v obliki datotek, zapisov in polj ter so s tem zelo primerni za organizacijo v obliki podatkovnih baz; procesiranje podatkov je odvisno tudi od kategorije - ločimo znanstvene in poslovne podatke;

- <u>besedila</u> imajo manj strukturirano naravo od zgoraj omenjenih podatkov; osnovne enote pri urejanju in procesiranju besedil so besede, stavki, odstavki, strani in dokumenti; vendar so zmožnosti avtomatskega procesiranja besedil (brez uporabnikove ali operaterjeve navzočnosti) bistveno manjše kot pri podatkih (v ožjem smislu);

- <u>slike</u> imajo še slabšo strukturiranost - enoto predstavljata tu točka (pixel) in cela stran oziroma slika; obsežnost informacije o sliki pa pri kolikor toliko spodobni resoluciji močno naraste;

- <u>govor</u> je najpomembnejša oblika podatkov (v širšem smislu) in bo še dolgo zavzemal več kot 90% vseh prometnih zmogljivosti PBX; načelno nima nobene fizično očitne strukture, pri njegovem shranjevanju na magnetne računalniške medije pa lahko s pridom uporabljamo le zapis v obliki datoteke.

Pri vsaki od navedenih oblik podatkov so poleg strukturiranosti važne še naslednje značilnosti:
- obsežnost (nevarnost "eksplozije podatkov" pri določenih načinih uporabe),
- burnost prenosa in
- možne osnovne operacije nad njimi.

Shematični pogled na klasifikacijo podatkov je prikazan na sliki 1, s komunikacijami kot "lepilom" na sredi. Oprema, ki je vrisana znotraj črtkanega okvirja, lahko uporablja komunikacijski podsistem, medtem ko oprema zunaj okvirja te možnosti nima.

Pri komuniciranju lahko posamezne oblike podatkov med seboj kombiniramo, nekatere kombinacije pa so za uporabnike izjemno atraktivne:
- videokonferenca na primer zahteva simultani prenos govora in slike,
- pri elektronski pošti oziroma v izmenjavi dokumentov so kombinirana besedila, podatki in slike, itd.

Na sliki 1 je vmes med področji prikazana tudi oprema za tak kombiniran simultan prenos različnih oblik podatkov.

Integriran prenos vseh vrst podatkov oziroma integracija storitev pomeni, da omrežje za prenos ne ve in ne sme vedeti, s kakšnimi oblikami podatkov oziroma s kakšnimi storitvami se ukvarja, ampak mora izvesti "prozoren" prenos dane informacije (v obliki povorke bitov).

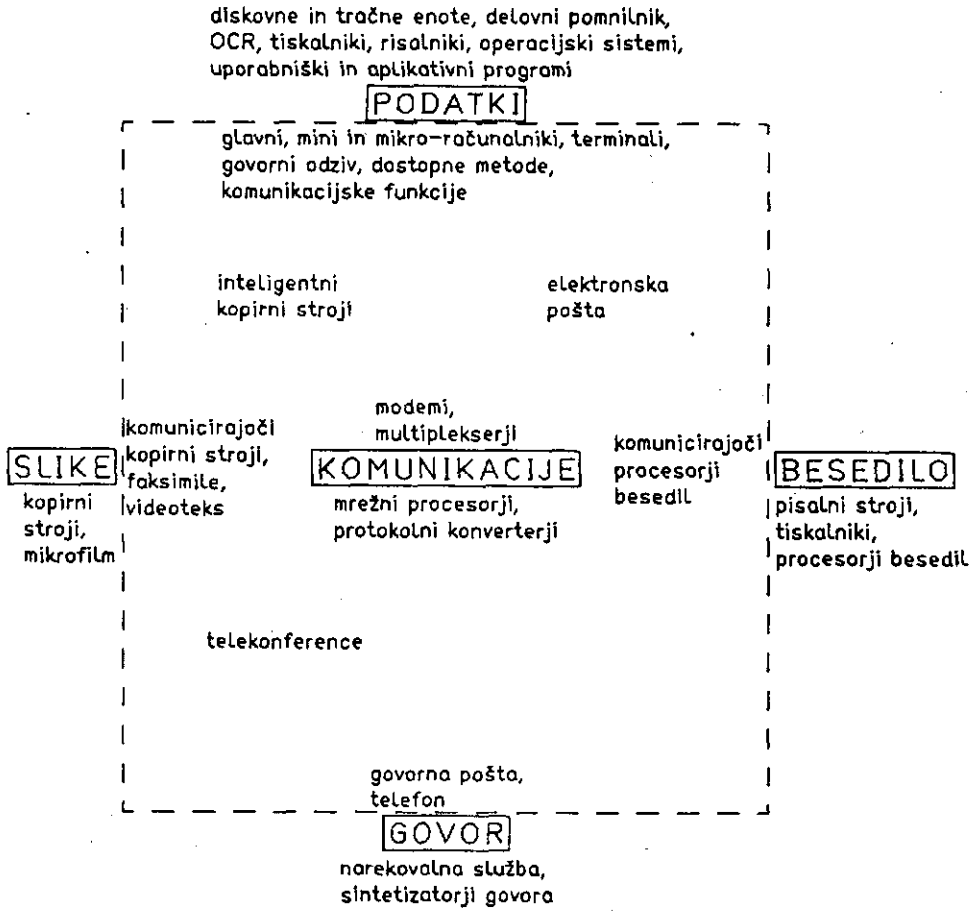
4. Struktura sistema z integracijo govora in podatkov

Glede na v prejšnji točki opredeljene značilnosti podatkov potrebujemo komutacijski sistem za učinkovit prenos vseh oblik podatkov na čimbolj enak način (integrirano). Njegova struktura naj bi bila taka, da ima:

- sposobnost komutacije, zasnovane na popolnoma digitalnem prenosu podatkov,
- neblokirajočo arhitekturo,
- združene prednosti LAN (prilagojenost hitremu prenosu podatkov) in PBX (majhna občutljivost za promet, vrojena zmožnost učinkovitega prenosa govora) ter s tem povezano
- zmožnost operacij (ki so doslej omejene le na posamezne PBX) na nivoju mreže;
- vsaj predvideno možnost kasnejše razširitve na širokopasovni prenos podatkov.

Komunikacijski sistemi, ki bodo sedaj instalirani, pa morajo imeti tudi še dve drugi važni sposobnosti:

- polno uporabljati obstoječo, instalirano opremo za govorne in podatkovne komunikacije in
- imeti tako strukturo, da se bo dalo enostavno dodajati novo standardno govorno in podatkovno opremo, ki bo predvidoma v kratkem na voljo na tržišču.



Slika 1. Klasifikacija podatkov in ustrezna oprema

5. Terminali

Za izvedbo raznih telematskih storitev in različnih funkcij (ter s tem povezanim oblikovanjem podatkov) uporabljamo zelo različno terminalno opremo, od "dobrega starega" analognega telefona do specializiranih posebnih naprav za povsem določene namene. Oglejmo si le dva splošna tipa terminalov, ki bosta po predvidevanjih v bodoče najbolj razširjena. To sta:

- digitalni telefon in
- delovna postaja.

Digitalni telefon je funkcionalno podoben analognemu telefonu, le da govor takoj pretvori v digitalno obliko, omogoča pa tudi izmenjavo raznih sporočil med centralo in telefonom, ki jih telefon izpisuje na LCD prikazovalniku.

Delovna postaja je enota s tipkovnico in prikazovalnikom (ekranom) za ustvarjanje, popravljanje in shranjevanje podatkov, besedil in slik za izpis na zaslon, izpis na tiskalnik ali za komunikacijo z drugimi sistemi; ima možnost prenosa poljubne vrste podatkov preko telefonske linije (IVD (Integrated Voice & Data) terminal); nudi možnost uporabe različnih programov za poslovno in osebno rabo (elektronski urnik, kalkulator, procesorji besedil, preglednice, OCR (Optical Character Recognition)).

6. Telematske storitve

Sedaj, ko vemo, kaj želimo prenašati (vse vrste podatkov, na integriran način), s čim prenašati podatke (omrežje, ISDN) in s čim oblikovati podatke (terminali), moramo povedati še, za kaj in kako bomo uporabili te podatke: za izvedbo raznih telematskih storitev.

V okviru CCITT priporočil so specificirane naslednje telematske storitve:

- javni faksimile,
- teleteks in
- videoteks.

Te tri storitve so sorazmerno precej dobro standardizirane: določen je način delovanja teh storitev v mednarodnem prometu, specificirane so zahteve za terminale za izvajanje teh storitev, dejanska izvedba storitev na nacionalnem nivoju pa je prepuščena posameznim PTT organizacijam.

Javni faksimile je kopiranje na daljavo: odčitavanje slike (dokumenta,...) pri pošiljatelju, prenos tako generiranih podatkov skozi omrežje in reprodukcija slike na ponorni strani.

Teleteks služi za urejanje, oblikovanje, prenos in shranjevanje besedil med različnimi uporabniki. V bistvu je teleteks izboljšani teleks glede na kvaliteto dokumentov in hitrost prenosa.

Videoteks je storitev za dostop (in ažuriranje) javnih podatkovnih baz za izvedbo različnih poizvedovanj, pa tudi za elektronsko pošto.

Poleg teh treh obstoji še cela množica drugih, bolj ali manj definiranih storitev. Navedimo jih le nekaj:

Elektronska pošta je storitev za prenos besedil ali podatkov (v ožjem smislu) med različnimi uporabniki. Sestavljajo jo naslednje funkcije: urejanje ter oblikovanje, pošiljanje, prenos, sprejem, arhiviranje in brisanje pošte.

V različnih virih pa z elektronsko pošto povezujejo še naslednja področja:
- MHS (Message Handling Systems) je sistem za izmenjavo sporočil med uporabniki, definiran v okviru X.400 CCITT priporočil in se uporablja predvsem za realizacijo elektronske pošte;
- predvsem na Japonskem vključujejo med elektronsko pošto tudi "poštni faksimile" ali "komunicirajoče inteligentne kopirne stroje", saj so ugotovili, da je za sporočila, pisana s katakano, tak način prenosa najprimernejši;
- nekateri si pod elektronsko pošto predstavljajo le del videoteksa;
- v zaprtih sistemih (v Sloveniji na primer DeltaNet) se pojavljajo programski produkti s tem imenom, ki poleg običajnih funkcij zahtevajo še osebno vročitev pošte osebam brez dostopa do terminala, pismo s povratnico, itd.

Telekonferenca je storitev, ki omogoča dvem ali več skupinam ljudi medsebojno hkratno komuniciranje na govorni in/ali vidni način. Telekonference razvrščamo v naslednje kategorije:

- zgolj avdio-konferenca se omejuje le na prenos govora in je lahko fiksna oziroma vnaprej določena (traja neprekinjeno) ali pa jo naročnik ali posredovalec vzpostavi z izbiranjem;

- audiografska konferenca je avdio-konferenca z dodano omejeno zmožnostjo prenosa grafične informacije, običajno s pomočjo elektronskih tabel in elektronskih skicirk, faksimile naprav in mikrofilmov;

- videokonferenca kombinira prenos govora, grafike in slik udeležencev; prenos slik je lahko na način počasne TV (slow-scan), lahko je običajni (analogni) prenos ali pa komprimirani video (prenesejo se le spremembe na sliki);

- računalniška konferenca je storitev, s katero se medsebojno povezujejo udeleženci konference s pomočjo svojih terminalov v nekem centralnem računalniku.

Večino sedaj obstoječih videokonferenc po svetu uporabljajo za poslovne sestanke in le nekaj procentov za izobraževanje in nadaljnje usposabljanje.

Kabelska televizija, HI-FI radio in teletekst so danes marsikje že delujoče storitve, vendar le v okviru posebnih omrežij. Zanje lahko pričakujemo, da se bodo v okviru širokopasovnega ISDN omrežja združile z ostalimi telestoritvami.

7. Ekspertni sistemi v teleinformatiki

Z leti postajajo centrale oziroma njihove zmožnosti multipleksiranja vse večje, kontrolni sistemi zanje pa vse kompleksnejši (z nazivom kontrolni sistem imenujemo vso programsko opremo od sistemske in aplikativne pa vse do administrativne ter podporne programske opreme). Z uporabo mikroprocesorjev smo dosegli zelo močne, fleksibilne kontrolne sisteme, hkrati s tem pa je močno narasla tudi težavnost odkrivanja napak v njih. Za testiranje, vzdrževanje in modificiranje sistemov zato čedalje bolj prihaja do veljave (kot skoraj edina možna rešitev) uporaba metod umetne inteligence oziroma ekspertnih sistemov.

Naredimo še kratek pregled možnih ekspertnih sistemov v okviru teleinformatike:

- testiranje nove oziroma popravljane programske opreme v PBX,

- načrtovanje konfiguracij PBX central oziroma privatnih omrežij na osnovi PBX central tretje ali četrte generacije z materialnega in programskega stališča glede na potrebe kupca ter hkratna določitev arhitekture in celotnega designa tega sistema,

- načrtovanje uporabniško prijaznih vmesnikov v teh sistemih,

- nadzor nad stanjem kablov v omrežju in odkrivanje ter sporočanje okvar,

- diagnosticiranje izpadov napajanja v elektronskih PBX,

- upravljanje z informacijami (inteligentne podatkovne baze),

- vzdrževanje opreme,

- razpoznava in interpretacija različnih signalizacij (pomembno pri motnjah, zakasnitvah, ...),

- razvrščanje in dodeljevanje resursev za telekonference,

- diagnosticiranje izpadov satelitskega prenosa in delovanja,

- zasledovanje in navigacija satelitov, itd.

Seveda zadnja dva primera za nas žal še nista primerna.

8. Standardizacija

Na kratko preglejmo CCITT priporočila, s poudarkom na serijah V, X, I in T.

Serija V se ukvarja s podatkovnimi komunikacijami preko telefonskega omrežja.

Serija X se ukvarja z omrežji za prenos podatkov:
- zmožnosti;
- vmesniki;
- prenos, signalizacije in preklapljanje;
- vidiki omrežja;
- vzdrževanje;
- OSI (odprti sistemi povezovanja);
- medsebojno sodelovanje;
- MHS (sistemi za prenos in obdelavo sporočil).

Serija I vsebuje priporočila za ISDN:

- I.100: splošni pregled (okvirna zgradba in terminologija; opis ISDN; splošne metode oblikovanja; smeri razvoja);
- I.200: zmožnosti storitev (vidiki storitev; prenosne storitve; telestoritve);
- I.300: splošni vidiki in funkcije omrežja (funkcionalni principi omrežja; referenčni modeli (protokoli in funkcionalna arhitektura modela, hipotetične referenčne povezave); principi ošteviličenja, naslavljanja in usmerjanja; tipi povezav; zahtevane lastnosti (povezovanje s pomočjo tokokrogovnega ali paketnega preklapljanja));
- I.400: vmesniki med uporabnikom in omrežjem (splošni vidiki vmesnikov med uporabnikom in omrežjem (referenčne konfiguracije, struktura kanalov in zmožnosti dostopa); aplikacije vmesnikov med uporabnikom in omrežjem; priporočila nivoja 1 (vmesniki za bazično in za primarno hitrost); priporočila nivoja 2 (LAPD); priporočila nivoja 3; multipleksiranje, prilagajanje hitrosti, podpora obstoječih priporočil (X.21, X.21 bis, X.25, serija V, 56K bps));
- I.500: vmesniki med centralami v omrežju;
- I.600: principi vzdrževanja (principi na uporabnika nanašajočega se testiranja in vzdrževanja).

Serija T se ukvarja s terminali za izvajanje telematskih storitev.

Omenimo še standarnizacijo računalniških mrež (LAN). Najbolj odmevna s tega področja je serija standardov IEEE 802, ki se ukvarjajo predvsem z načinom dostopa do prenosnega medija:

- 802.1 tehnična rdeča nit: odnos med standardi in ISO OSI referenčnim modelom
- 802.2 logični nadzor linije:
  - LLC-1 - brezpovezavni
  - LLC-2 - povezavno usmerjeni
- 802.3 vodilo z dostopom CSMA/CD (Ethernet)
- 802.4 vodilo z žetonom
- 802.5 obroč z žetonom
- 802.6 mestna omrežja

V Evropi je (vsaj v univerzitetnih krogih) precej priljubljena lokalna računalniška mreža Cambridge Ring. Standardizirana je v okviru ISO z dokumentom DP 8802.6 (slotted ring).

Pričakovana standardizacija v svetu in v Jugoslaviji: v svetu se standardi izredno hitro razvijajo (dopolnjevanje standardov za ISDN, standardi za širokopasovni ISDN, za pisarniško avtomatizacijo in še za mnogo drugih področij), pri nas pa se na področju standardizacije podatkovnih omrežij ne dogaja skoraj nič. Eden od prvih korakov pri nas v tej smeri bo verjetno standardizacija postopkov, uporabljenih v Jupaku (X.25, itd.). Vsaj za upočasnitev zaostajanja na tem področju je treba delo takoj zelo intenzivirati.

9. Literatura

International Telecommunication Union, CCITT, Red Book, VIII th Plenary Assembly, 1984 (Geneva 1985)

Serija standardov ANSI/IEEE 802.n (Standard for Local Area Networks), n = 1, 2,..., 6

ISO standardi:
DP 8802.6 - Slotted Ring
DIS 7498 - Open Systems Interconnection

Datapro Reports on Data Communications, Datapro Research Corporation, Delran, NJ (monthly updated)

IEEE Spectrum, Special issue: Telecommunications, Volume 22, november 1985

M. J. Bevan: Image Processing May Cause Future Problems with Network Loading, Data Communications, march 1986

R. C. Hawk: The Integrated Voice/Data Option, Conference Transcript of "Voice/Data Integration Conference", Oyez Scientific and Technical Services, London, december 1983

P. Kahl: A Review of CCITT Standardization to Date, IEEE Journal on Selected Area in Communications, may 1986, Vol. SAC-4, No.3

W. Karavatos: The Fourth Generation of PABXs, Conference Transcript of "3rd and 4th Generation PABXs Conference", Oyez Scientific and Technical Services, London, february 1985

W. B. Rauch-Hindin: Upper Level Network Protocols, Electronic Design, march 1983

A. S. Tanenbaum: Computer Networks, Prentice Hall, Englewood Cliffs, NJ, 1981

ADVANCED MICROPROCESSORS AND HIGH-LEVEL LANGUAGE COMPUTER ARCHITECTURE

Knjiga z naslovom ADVANCED MICROPROCESSORS AND HIGH-LEVEL LANGUAGE COMPUTER ARCHITECTURE je zbirka najpomembnejših člankov, poročil in ekspertiz, ki jih je zbral in uredil avtor Veljko Milutinović za potrebe poučevanja predmeta z enakim naslovom na Purdue University v ZDA.

Knjiga je učbenik za študente ter hkrati tudi dober pripomoček načrtovalcem novih računalniških sistemov in vodjem razvoja, ki potrebujejo znanje o sodobnih trendih v računalniških arhitekturah. Knjiga sestoji iz 39 člankov, ki so razdeljeni v 7 delov in 14 poglavij ter ima skupno skoraj 600 strani.

Knjiga govori o arhitekturah, ki temeljijo na visokoprogramskih jezikih, to je o HLL (High Level Language) računalniških arhitekturah. Avtor deli HLL računalniške arhitekture na dve osnovni skupini:

- arhitekture z indirektnim izvajanjem ter
- arhitekture z direktnim izvajanjem.

Pri arhitekturah z indirektnim izvajanjem je za izvajanje programa potrebno izvorni kod prevesti v izvajalni kod. Pri arhitekturah z direktnim izvajanjem pa računalnik lahko direktno izvaja izvorni kod.

Nadalje deli Milutinović arhitekture z indirektnim izvajanjem v dva podrazreda:

- reducirane arhitekture in
- kompleksne arhitekture.

Kompleksne arhitekture se nadalje delijo v

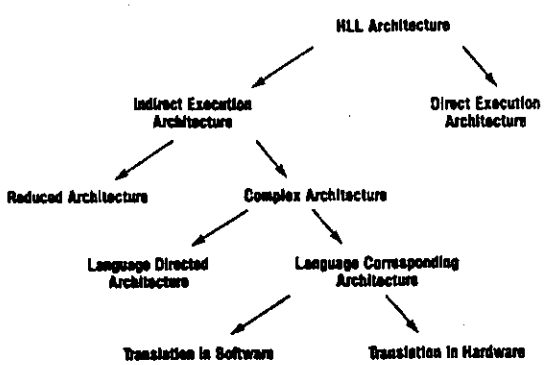
- jezikovno vodene arhitekture in
- jezikovno odvisne arhitekture.

Jezikovno vodene arhitekture lahko definiramo kot arhitekture, kjer se prožijo konstrukcije strojnega jezika na relativno visokem nivoju toda ne na nivoju HLL sintakse. Pri jezikovno odvisnih arhitekturah pa imamo direktno enolično odvisnost med HLL konstrukcijo in konstrukcijo strojnega jezika.

Na koncu so razdeljene jezikovno odvisne arhitekture še v:

- arhitekture s programskimi prevajalniki ter
- arhitekture s strojnimi prevajalniki.

Slika 1 kaže razdelitev HLL arhitektur.



Slika 1: Razdelitev HLL arhitektur.

Vtis o knjigi si najhitreje ustvarimo s pregledom kazala knjige, ki podaja poleg naslovov vseh člankov tudi strukturo knjige in razdelitev vsebine v posamezne dele in poglavja.

Table of Contents

V prvem poglavju knjige so 4 uvodni članki svetovno poznanih avtorjev (Flynn, Wulf, Myers,

Ditzel), ki obravnavajo odnos med programskim jezikom in arhitekturo računalnika. Hkrati predstavljajo izbrani članki pregled področja ter omogočajo bralcu, da na podlagi bibliografije razširi svoje znanje v posameznih specialnostih.

Drugo poglavje vsebuje 2 članka o vlogi VLSI tehnologije na računalniške arhitekture. Prvi članek (Hennessy) obravnava VLSI tehnologijo, ki temelji na klasični silicijevi tehnologiji, drugi članek (Milutinović) pa obravnava novo in hitro prodirajočo tehnologijo vezij na osnovi galijevega arzenida GaAs. Pristop pri načrtovanju računalniške arhitekture v mnogočem zavisi od izbrane tehnologije. Tako silicijeva kot tudi GaAs tehnologija imata vsaka svoje značilnosti, ki postavljajo specifične zahteve pri načrtovanju optimalne računalniške arhitekture. Poglavje obravnava reducirane arhitekture in sicer RISC računalnik iz Berkeley, MIPS iz Stanforda in ostale.

Prvi članek drugega poglavja, katerega avtor je Hennessy, primerja dva osnovna pristopa v arhitekturi, to je arhitekturo, ki zajema kompleksen nabor ukazov in arhitekturo z reduciranim naborom ukazov, kjer je obakrat procesor realiziran v VLSI tehnologiji na osnovi silicija. Članek analizira učinkovitost izvajanja prevedenega koda visokoprogramskega jezika.

Drugi članek je napisal Milutinović in predstavlja GaAs kot možno osnovo za implementacijo VLSI procesorjev. Pri tem pa nam nova tehnologija narekuje novo arhitekturo vezij in rešitve, ki so bile razvite za silicijevo okolje na splošno niso uporabne pri vezjih, ki temeljijo na GaAs. Kaže, da je pri vezjih, ki temeljijo na GaAs učinkovita samo arhitektura z reduciranim naborom ukazov.

Poglavja 3, 4 in 5 so posvečena kompleksnim arhitekturam, to je jezikovno odvisnim in jezikovno vodenim arhitekturam. V tretjem poglavju je narejen pregled različnih pristopov pri zasnovi in implementaciji novih mikroprocesorjev in HLL arhitektur ter analiza performans različnih sistemov.

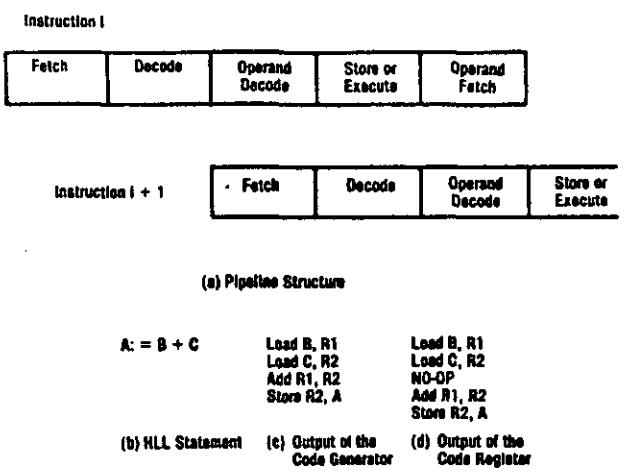
Četrto poglavje je posvečeno RISC (Reduced Instruction Set Computer) arhitekturam. Ukvarja se z izborom reduciranega nabora ukazov in nekaterimi specifičnimi implementacijami, ki temeljijo na reduciranem naboru ukazov. Posebej je v prvem članku (Petterson) obdelan projekt University of California at Berkeley UCB-RISC, ki omogoča hitrejše izvajanje programa napisanega v visokoprogramskem jeziku. Bistvo rešitve problema je v metodi kako zasnovati *arhitekturo*, da bo čim bližje tistim konstrukcijam strojnega jezika, ki se pri visokoprogramskih jezikih najpogosteje uporabljajo. Hkrati naj bo arhitektura računalnika zasnovana tako, da bo zmanjšana potreba po komunikaciji z ostalimi vezji . Tako ima RISC I arhitektura le 31 ukazov od katerih večina opravlja le preproste ALU in pomične (shift) operacije na registrih.

Drugi članek v tem poglavju (Petterson) obravnava implementacijo VLSI vezja za cache ukaze za UCB-RISC računalnik. To VLSI vezje je neobčutljivo na napake (fault tolerant), vsebuje poseben programski števec, izvaja komprimiranje kode in omogoča razširljivost. Končni cilj skupine v Berkeleyu je združiti ukazni cache pomnilnik ter podatkovni cache pomnilnik s centralno procesno enoto v eno vezje (chip).

Naslednji članek v četrtem poglavju (Tamir) se ukvarja z različnimi strategijami in metodami, ki so povezane z vodenjem večkenskih registerskih datotek v računalnikih kot je na primer UCB-RISC. Pri visokoprogramskih jezikih je klicanje procedure časovno najbolj zamudna operacija. V splošnem imajo lahko RISC programi še več klicev procedur kot običajni računalniki saj so kompleksni ukazi, ki jih najdemo v CISC arhitekturah realizirani kot podprogrami v RISC arhitekturi. Zato morajo biti klici procedur pri RISC računalnikih kar se da hitri. To je realizirano z registerskimi okni. Večokenska registerska datoteka je eden od tistih pristopov, ki bistveno pripomore k zmanjšanju potreb po komunikaciji z drugimi vezji. Ideja je v tem, da v vezje vgradimo določeno število delno prekrivajočih se registerskih oken, pri čemer je ob vsakem trenutku dostopno prevajalniku ali programerju v zbirnem jeziku samo posamezno okno. Ostala okna vsebujejo spremenljivke za ostale procese. Eden od problemov pri tem pristopu je kako ravnati, če pride pri registerski datoteki do presežka (overflow). Avtorja članka ugotavljata, da je optimalno število registrov, ki shranjujejo presežek enako ena.

Zadnji članek v četrtem poglavju (Ungar) opisuje zasnovo in implementacijo mikroprocesorja, ki je namenjen za jezik Smalltalk. Ta procesor temelji na reduciranem naboru ukazov, a se bistveno razlikuje od UCB-RISC procesorjev. To nam potrjuje domnevo, da se vsaka reducirana arhitektura navezuje na določeno aplikacijo zato različno aplikativno okolje vodi do različnih konkretnih rešitev RISC procesorja.

Peto poglavje vsebuje štiri članke in je posvečeno MIPS pristopu. MIPS je okrajšava za Microprocessor without Interlocked Pipeline Storage. Ta koncept so razvili na Stanford University. MIPS pristop je primeren za vezja, ki temeljijo na tehnologiji silicija, kot tudi za vezja ki temeljijo na tehnologiji GaAs. Če primerjamo MIPS arhitekturo z arhitekturo UCB-RISC vidimo, da je za MIPS arhitekturo značilna nizka kompleksnost VLSI vezij, zahteva pa kompleksnejši način programiranja in prevajanja. Eno osnovnih prednosti MIPS arhitekture je dejstvo, da je cevna sinhronizacija izvedena s programskim pristopom in ne več v materialni računalniški opremi. To je ugodno predvsem zaradi lažje implementacije VLSI vezij in tudi hitrosti procesorja. Seveda pa mora prevajalnik sedaj generirati kod, ki je brez cevnih konfliktov. Generator koda zadosti tem zahtevam tako, da vstavi ukaze NO-OP povsod tam, kjer je to potrebno. Optimizator koda poskuša nato zamenjati čim več ukazov NO-OP s kodo od kjerkoli drugje pod pogojem, da na ta način ne vpliva na odvisnost podatkov. Tu vidimo potrebo po kompleksnejši tehnologiji prevajalnikov.



Slika 2.: Primer seštevanja z MIPS arhitekturo, ki kaže izhod generatorja koda in reorganizatorja koda.

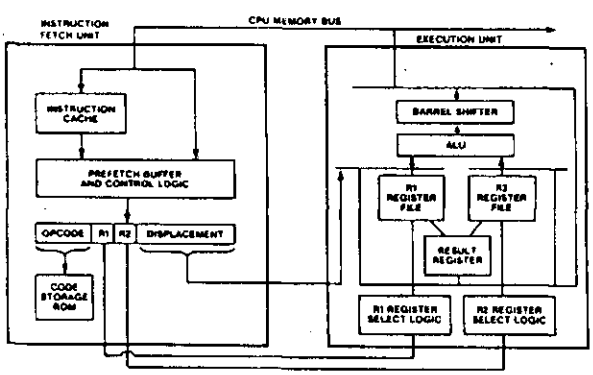
Naslednji članek (Przybylski) je tehnično poročilo Stanford University, ki opisuje VLSI implementacijo MIPS. Ta članek obravnava področja kot so nabor ukazov, cevna organizacija, ukrepanje pri izjemah, podpora za virtualni pomnilnik, podpora operacijskega sistema in visokoprogramskega jezika itd. Članek vsebuje tudi podatke o merjenju performans sistema.

Naslednji članek (Gross) se osredotoča na probleme, ki so povezani s programsko implementacijo aritmetike s plavajočo vejico za MIPS mikroprocesorje. Pri izbiri reduciranega nabora ukazov se pojavlja vprašanje, kako je tak reduciran nabor primeren za aritmetiko plavajoče vejice in za aritmetiko na sploh. Nabor ukazov za MIPS arhitekturo vsebuje ukaz, ki ustreza množenju po Booth algoritmu, kar je dalo relativno dobre performanse.

Zadnji članek v petem poglavju (Hennessey) analizira številne činitelje, ki so povezani z optimizacijo koda v MIPS okolju. Članek definira tip cevnih konfliktov do katerih lahko pride v MIPS mikroprocesorju. Govori o razlikah pri optimizaciji koda, ki se izvaja po alokaciji registrov ali sočasno z alokacijo. MIPS optimizator koda uporablja prvi pristop. Podana je teorija MIPS kodnega optimizatorja in podatki o oceni performans.

6. poglavje je posvečeno arhitekturam za direktno izvajanje. Vsebuje štiri članke, ki obravnavajo najzanimivejše procesorje z reduciranim naborom ukazov. Trije taki procesorji izhajajo iz industrije, to so: IBM (IBM 801), Ridge (Ridge-32) in Pyramid (Pyramid 90X), četrti procesor pa je bil razvit na University of Reading v Angliji. Angleški procesor je narejen v VLSI tehnologiji (RIMMS) ostali trije pa temeljijo na hitri SSI/MSI tehnologiji. Intenzivne raziskave in razvoj procesorjev z reduciranim naborom ukazov tečejo tako v industrijskem okolju (Fairchild, Hewlett-Packard, RCA, TRW, Inmos) kot tudi na univerzah (Caltech, Purdue, ULCA, Wisconsin in drugje).

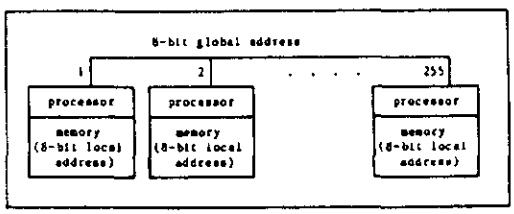
Nekateri tovrstni projekti so se pričeli že sredi sedemdesetih let. Tak je na primer projekt za miniračunalnik IBM 801. Članek, katerega avtor je Radin, govori o tem projektu in podaja nekatere značilnosti samega procesorja in tudi sistema kot celote. Opisuje na primer nabor ukazov, ki je realiziran popolnoma v materialni računalniški opremi, hierarhičnost pomnilnika, organizacijo V/I komunikacije, ki omogoča centralnemu procesorju izvajanje ukaza v skoraj vsakem ciklu in drugo. Projekt je bil poleg tega usmerjen k realizaciji sposobnejših prevajalnikov.



Slika 3.: Zgradba Ridge-32 procesorja.

Basart in Folger v svojem članku predstavljata zasnovo in implementacijo superminiračunalnika Ridge-32, ki je verjetno prvi komercialno dobljiv RISC računalnik. Članek se ukvarja z osnovnimi cilji arhitekture, z izborom nabora ukazov, z implementacijo sistema in oceno performans. Ta računalnik je narejen za hitro izvajanje zahtevnih grafičnih aplikacij. Slika 3 kaže strukturo Ridge-32 procesorja, ki je implementiran s komercialno Schottky bipolarno logiko. Procesor ima ločeno dostavno (fetch) enoto in izvajano enoto. Dostavna enota vedno vsebuje naslednji ukaz, ki se bo izvedel. Dostavna enota in izvajalna enota tvorita štiristopenjsko cevno arhitekturo, ki omogoča prekrivanje izvajanja ukazov. Ridge-32 vsebuje tudi začasen pomnilnik (cache), kjer se hranijo vsi pred kratkim izvedeni ukazi.

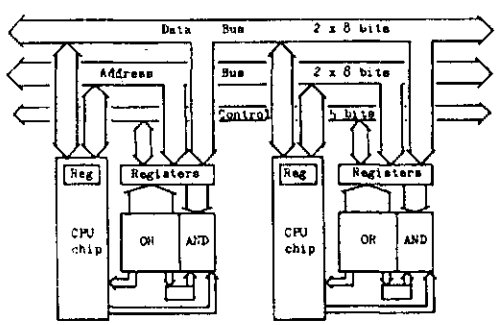
Foti s sodelavci v četrtem članku šestega poglavja opisuje projekt RIMMS mikroračunalnika (Reduced-Instruction set Multi-Microprocessor System project). RIMMS sestoji iz linearnega polja 255 mikroračunalnikov, ki komunicirajo preko skupnega vodila kot kaže slika 4.



Slika 4.: Zgradba RIMMS mikroračunalnika z 255 procesorji.

Vsak mikroračunalnik ima preprost procesor in 256 zlogov lokalnega pomnilnika.

Ta mikroprocesor ima skrajno reduciran nabor ukazov in je namenjen uporabi v multimikroprocesorskem okolju. Zato so posebno pozornost posvetili strojni računalniški podpori za medprocesorsko komunikacijo. Vodilo sestoji iz 16 bitnega naslova, 16 bitnega podatka in dodatnih bitov za delo s pomnilnikom, kar omogoča izvajanje NO-OP posega po pomnilniku. Slika 5 kaže organizacijo in kontrolo vodila ter pomnilnika za RIMMS mikroračunalnik.



Slika 5.: Nadzor vodila in pomnilnika za RIMMS mikroračunalnik.

Med najzanimivejše lastnosti tega mikroprocesorja sodijo metode, ki jih uporabljajo za podporo programskih konstrukcij kot so FORK, JOIN, REMOTE LOAD in REMOTE STORE. Razvili so jezik za paralelno procesiranje imenovan BASAL, ki podpira osnovno idejo mikroračunalnika z

reduciranim naborom ukazov za multimikroprocesorsko okolje.

Ragan-Kelley opisuje razvoj superminiračunalnika pri Pyramid Technology Corporation, ki prav tako temelji na reduciranem naboru ukazov. Cilj projekta je sposoben računalnik, ki podpira UNIX operacijski sistem in visokoprogramske jezike kot so C in Pascal. Računalnik je namenjen večuporabniškem okolju. Na razvoj tega projekta je verjetno vplival projekt RISC iz Berkeley-ja, kljub temu pa vsebuje ta projekt številne nove zamisli.

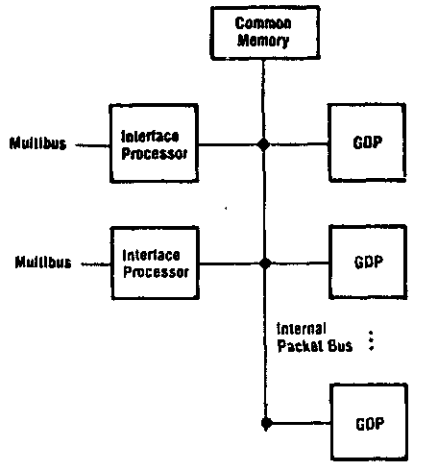
Sedmo poglavje se ukvarja s skladovnimi (stack) računalniki. Sklad omogoča učinkovito organizacijo preklaplanja konteksta kot tudi aritmetiko. S skladovno arhitekturo se posebej ukvarjajo Burroughs, Hewlett-Packard, Microdata, Intel, Xerox in Zilog. Prvi članek v tem poglavju opisuje osnovno idejo skladovne arhitekture, še posebej povezovanje in kontrolo podprogramov ter evaluacijo izrazov. Podana je primerjava skladovnih računalnikov in tradicionalnih računalnikov, ki temeljijo na splošnih registrih.

Blake je avtor članka, ki je uvod v optimalno zasnovo skladovnega računalnika in podaja pristop firme Hewlett-Packard. Članek analizira potrebe in značilnosti programov pisanih v visokoprogramskih jezikih in njihov vpliv na sklad. Posebna pozornost je posvečena strojni opremi skladovnih računalnikov, naboru ukazov, naslovnem prostoru za procese, spremljanju procesov itd.

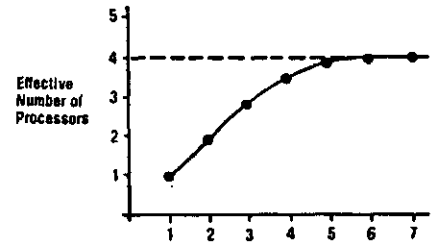
Earnest v tretjem članku sedmega poglavja obravnava več skladovnih računalnikov, ki so jih razvili pri Burroughs v zadnjih 20 letih. Eden glavnih razlogov, da se nekateri proizvajalci osredotočajo na skladovne računalnike je dejstvo, da je relativno lahko pisati prevajalnike za skladovne računalnike in, da je ta koda ponavadi precej kompaktna.

Osmo poglavje se ukvarja s sodobnimi mikroprocesorji za kompleksen nabor ukazov, ki jih pogosto imenujemo CISC (Complex-Instruction-Set Computers). Poglavje vsebuje 8 člankov, ki govorijo o 32 bitnih procesorjih, ki so namenjeni delu z visokoprogramskimi jeziki. Imajo bogat nabor ukazov, od katerih nekateri zelo spominjajo na konstrukte, ki so značilni za moderne visokoprogramske jezike. Raziskave in razvoj na tem področju hitro napredujejo in novi ter popolnejši 32 bitni mikroprocesorji se stalno pojavljajo v ZDA, na Japonskem in v Evropi. Uveljavljeno je prepričanje, da so CISC računalniki bolj primerni za podporo programske zanesljivosti kot pa RISC računalniki. Prav tako imajo močnejše sposobnosti obdelave aritmetičnih izrazov. Saveda pa so CISC računalniki v primerjavi z RISC računalniki počasnejši pri izvajanju prevedenega programa, ki je pisan v visokoprogramskem jeziku. Ker so CISC procesorji kompleksnejši, zahtevajo daljši čas za razvoj procesorja. Dejstvo pa je, da so novi CISC procesorji zelo sposobni, uporabljajo visoko stopnjo cevnega paralelizma in vsebujejo že na čipu cache pomnilnik ter mehanizme za delo s pomnilnikom. Končna odločitev in tekma med RISC in CISC mikroprocesorji je še pred nami in ni jasno kdo bo zmagovalec.

Gupta v svojem članku primerja 4 zgodnje 32 bitne mikroprocesorje. Tehnično poročilo firme Intel podaja bistvene značilnosti procesorja iAPX 432. Dobra lastnost procesorja iAPX 432 je razširjivost sistema. Večjo sposobnost sistema dosežemo brez spreminjanja programske opreme tako, da povečamo število procesorjev v sistemu na največ 5 GDP (General Data Processors). Pet takih procesorjev si deli isto vodilo in imajo zato performanse, ki so enake vsoti treh neodvisnih GDP. Razširitev iAPX v multiprocesorski sistem in performanse iAPX 432 multiprocesorskega sistema kaže slika 6.



(a) An iAPX 432 Multiprocessor System



(b) The Efficiency Function

Slika 6.: iAPX 432 multiprocesorski sistem in njegove performanse kot funkcija števila procesorjev.

Beyers v svojem članku podaja notranjo organizacijo procesirja HP-FOCUS. McGregor s sodelavci obravnava bistvene značilnosti Motorolinega procesorja MC68020. Mateosian opisuje glavne značilnosti procesorja NS32032 proizvajalca National Semiconductor. Na koncu poglavja pa je Patel podal pregled Zilogovega procesorja Z80,000. Vsi ti članki so zgolj pregledni članki.

Deveto poglavje podaja nekaj zgodnjih poskusov na področju HLL arhitektur s programskimi prevajalniki. To so arhitekture z naborom strojnih ukazov, ki je v enoličnі (eden proti enemu) zvezi z ukazi značilnega visokoprogramskega jezika. Prevajanje je izvršeno programsko. Pogosto te arhitekture imenujemo jezikovno skladne arhitekture s programskim prevajanjem. Zanimanje za te arhitekture se je začelo že leta 1960, ko se je pojavila potreba po računalniku, za katerega bi bilo lažje pisati sistemski in uporabniški kod. Glavni problem tega pristopa pa je kompleksnost materialne računalniške opreme.

To poglavje vsebuje 2 članka. Prvi, katerega avtor je Hassit et al., opisuje enega prvih tovrstnih eksperimentov pri IBM. Rezultat tega eksperimenta je bil računalnik, ki je bil namenjen za jezik APL. To je bil pravzaprav računalnik IBM S/360 M/25, ki je bil mikropro-

gramiran in je generiral APL kod. Osnovni razlog za ta razvoj je bila želja razviti učinkovito arhitekturno podporo za jezike kot so APL ali SNOBOL. Ti jeziki ponavadi potrebujejo interpreter in konvencionalen pristop ima lahko za posledico znatno zmanjšanje hitrosti izvajanja programa.

Nielsen predstavlja v drugem članku devetega poglavja glavne rezultate študije, katere cilj je bil načrt računalniške arhitekture in programskega jezika za vesoljske aplikacije.

Oba članka iz tega poglavja podajata rezultate eksperimenta. Računalnika, ki sta pri tem nastala sta danes zastarela, omenjena pa sta v tej knjigi zato, ker sta s svojimi idejami in eksperimentalnimi rezultati vplivala na sodobne tokove pri zasnovi novih računalniških arhitektur.

Deseto poglavje opisuje nekatere sodobne raziskave na MIT. Vsebuje dva članka, ki se navezujeta na jezikovno skladne arhitekture. Te so zanimive tako z gledišča VLSI načrtovanja vezij, kot tudi z gledišča mikroprogramiranja. Večina računalnikov, ki temelji na tem pristopu vsebuje mikroprogram za podporo konstrukcij visokoprogramskega jezika. Trenutno stanje razvoja VLSI lahko učinkovito podpira velike ROM pomnilnike za mikroprogram na čipu. Vemo pa, da je VLSI tehnologija premalo močna za podporo kompleksnih visokoprogramskih jezikov na popolni enolični korespondenci. Torej lahko z VLSI tehnologijo podpiramo bodisi samo izbrano podmnožico visokoprogramskega jezika ali pa uporabimo tehnologijo, ki ni VLSI in podpiramo celotni visokoprogramski jezik.

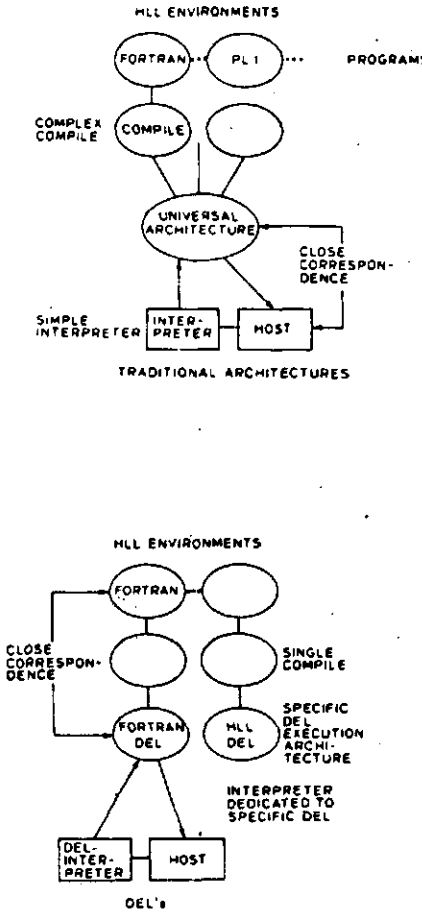
Prvi članek, katerega avtor je Sussman s sodelavci, opisuje zasnovo in implementacijo mikroračunalnika na enem vezju, ki direktno interpretira Scheme-79, to je dialekt jezika Lisp. Skupina na MIT je razvila interpreter v obliki mikrokoda, ki so ga realizirali v strojni računalniški opremi. Pri tem so uporabili dodatne in nekonvencionalne hardverske pripomoč ke ter s tem povečali učinkovitost računalnika.

Drugi članek desetega poglavja, katerega avtor je Batali et al., opisuje vezje Scheme-81, ki je naslednik vezja Scheme-79. Scehme-81 je namenjen za okolja, kjer so računalniki specializirani in kjer velike skupine računalnikov sodelujejo pri reševanju posamezne zahtevne naloge.

V 11. poglavju je govora o dveh zanimivih eksperimentih. Prvi izhaja iz šestdesetih let in opisuje Symbol računalniški sistem, to je računalnik, ki ima nekonvencionalno arhitekturo in o katerem je bilo v preteklosti veliko diskusij. Symbol je računalnik, ki ima splošen programski jezik in time-sharing operacijski sistem implementiran v materialni računalniški opremi. Vzpodbuda za tak eksperiment temelji na dejstvu, da so sčasoma postali prevajalniki zapleteni in jih je bilo težko napisati. Po drugi strani je cena materialne računalniške opreme padala, kompleksnost načrtovalnih orodij za razvoj vezij pa se je boljšala.

Drugi članek enajstega poglavja (Burkle) predstavlja laboratorijski eksperiment, ki je rezultiral v računalnik z imenom Abacus. Abacus je nastal pod vplivom računalnika Symbol, ima pa vrsto originalnih novih rešitev kot na primer kontrolo poslov (job control), učinkovito delo s podatkovnimi tipi in drugo.

Dvanajsto poglavje vsebuje dva članka, ki obravnavata DEL (Directlay Executed Language) pristop to je arhitekturo za direktno izvajanje jezika. To so, po razdelitvi iz slike 1, HLL arhitekture s strojnimi prevajalniki. Pri teh arhitekturah uporablajmo za programiranje običajen visokoprogramski jezik. S predprocesiranjem izvornega koda se prevede izvorni program v DEL obliko, kar je optimalni vmesnik med posameznim visokoprogramskim jezikom in izvajalnim računalnikom (Slika 7).



Slika 7: Primerjava med DEL arhitekturo in tradicionalno arhitekturo.

Prvi članek v tem poglavju, katerega avtor je Hoevel, podaja pregled različnih metod, ki se uporabljajo za generiranje DEL arhitekture.

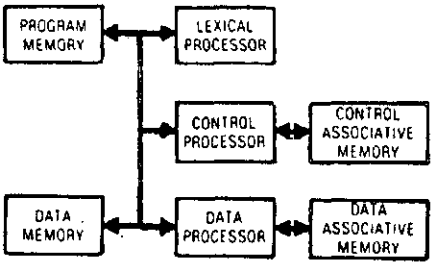
Flynn je avtor drugega članka in opisuje DELtran eksperiment, ki se navezuje na okolje programskega jezika Fortran II. Izvajalni računalnik pri tem eksperimentu je mikroprogramibilni računalnik EMMY z univerze Stanford. Članek podaja primerjavo v performansah, ko se uporablja DELtran ali pa druge konvencionalne arhitekture kot na primer IBM 370. Prihranek pri velikosti programa in naslavljanju pomnilnika je pri uporabi DELtran-a znatem.

Flynn in Hoevel sta s svojim delom močno vplivala na razvoj celotnih HLL arhitektur, vključno z RISC arhitekturami.

Trinajsto poglavje opisuje v dveh člankih arhitekturo računalnika za direktno izvajanje, ki so jo zasnovali na University of Maryland. Bistvo pristopa univerze Maryland je dejstvo, da arhitektura omogoča direktno interpretiranje izvornega koda programov. Na ta način je semantični prepad med programskim jezikom in arhitekturo popolnoma premoščen.

Prvi članek v trinajstem poglavju (Chu) se ukvarja s sistemskimi vprašanji interaktivnega visokoprogramskega jezika in z arhitekturo za njegovo direktno izvajanje. Članek obravnava razliko med sistemom z interaktivnim prevajanjem in sistemom z interaktivnim interpetiranjem.

Drugi članek trinajstega poglavja (Chu) analizira interakcijo med procesorjem in visokoprogramskim jezikom pri arhitekturh za direktno izvajanje (Slika 8). Procesor za to arhitekturo sestoji iz treh glavnih delov, to je slovarskega dela procesorja, nadzornega dela procesorja in podatkovnega dela procesorja. V članku je prikazana interakcija teh treh delov procesorja na primerih, ko uporabljamo visokoprogramski jezik.



Slika 8: Organizacija računalnika za direktno izvajanje.

Knjiga se konča s 14 poglavjem, ki obravnava pregledne članke o razvoju HLL arhitektur v različnih razvojnih centrih po svetu. Poglavje vsebuje dva članka, ki predstavljata raziskovalne in razvojno dosežke na področju HLL računalniških arhitektur v Japonski in v Evropi. Yamamoto je v članku, ki daje pregled teh raziskav na Japonskem, prikazal strukturo PL/I procesorja, konfiguracijo Cobol računalnika, več različnih Lisp računalnikov, NEC-ov single-chip Pascal procesor in druge zanimivosti.

Drugi članek zadnjega poglavja (Sylbey) daje pregled raziskav s področja računalniških arhitektur v Evropi. Omenjeni so dosežki s področja reducirnih arhitektur z University of Kent, univerze v Bonnu in Gesellschaft fur Mathematik und Datenverarbeitung. S podatkovno vodenimi arhitekturami se ukvarjajo na University of Manchester, Katholieke Universiteit Leuven in na univerzi v Dortmundu. Z reduciranimi arhitekturami se ukvarjajo na University of Reading. Jezikovno vodene arhitekture razvijajo na Tehnični univerzi Berlin in na Univerzi Dortmund. HLL arhitekture razvijajo na univerzi Paul Sabatier v Toulouse ter na univerzah v Dortmundu, Frankfurtu ter Kaiserslautern. Arhitekture za direktno izvajanje razvijajo na Švedskem in v Franciji. Seveda je centrov, kjer raziskujejo in razvijajo HLL računalniške arhitekture v Evropi še mnogo več. V tem članku je izbranih samo nekaj najbolj zanimivih. Na koncu je priložen še spisek evropskih revij in simpozijev s področja HLL računalniških arhitektur.

Pripravil: Saša Prešern

Pisma bralcev

Poštovani dr. Železnikar

U broju 3/86 Vašeg časopisa "Informatica", kojeg inače sa zadovoljstvom pratim i neobično cenim, objavili ste članak autora Vilfana, Mahniča i Mohoriča: "Ocena programskih orodij IDA". Iako znam da nije praksa Vašeg časopisa da prenosi polemike u vezi sa informatičkim temama ipak osjećam potrebu da na ovaj članak na neki način reagiram.

Mi u Jadroagentu u Rijeci koristimo IDA alate a posebno IDA-Bazu od samih njihovih početaka tako da smo dosta detaljno upoznati sa njima pa nas je posebno zanimalo jedno stručno mišljenje pogotovo kada ono dolazi sa Fakulteta za elektrotehniku iz Ljubljane i to iz Laboratorija za programsku opremu kako su se autori potpisali.

Ideja da se uzme u stručno razmatranje jedan programski proizvod je izuzetno zanimljiva pogotovo kada se radi o IDA-Bazi proizvodom Iskra-Delta iz Ljubljane za koju možemo slobodno reći da je prvi jogoslavenski sistem za upravljanje bazom podataka koji je našao svoj put do tržišta. Još će više zaintrigirati čitaoca najava da će proizvod biti uspoređen sa ULTROM poznatim programskim proizvodom svjetski renomirane firme CINCOM iz Cincinnatija u SAD.

U članku je uvodno dosta dobro dan pregled proizvoda IDA sa njegovim osnovnim karakteristikama ali se odmah očava da autori zaobilaze onaj dio koji će sve sadašnje i buduće korisnike IDA-alata zanimati a to su praktičan rad sa IDA-alatima (i usporedba sa ULTROM) kao i rezultati mjerenja performansi.

Da bi se dala ocjena jednog programskog proizvoda pogotovo tako složenog kao što je IDA nije dovoljno proučiti samo njegov priručnik već je potrebno određeno vrijeme raditi sa tim produktom a pri ruci imati razna mjerila kako bi se ocjene mogle kvantificirati.

Ako se želi dati paralelna ocjena u usporedbi sa drugim proizvodom tada treba razraditi (i objasniti) metodologiju usporednog testa i dati paralelne podatke kako bi budući i sadašnji korisnici tih produkata (jer smatramo da je za njih članak i napisan) mogli dobiti realnu sliku o kvalitetama i performansama tih produkata.

Zato se obraćam Vama druže uredniče sa molbom i željom da i ubuduće u Vaš časopis uvrstite članke koji daju ocjene softverskih proizvoda ali opremljene sa svim potrebnim podacima o performansama kako bi čitaoci mogli dobiti realnu sliku o njima.

Uz želje za uspješan napredak Vašeg časopisa drugarski pozdravljamo,

Ranko Smokvina, dipl. ek.
ruk. informatičke službe
Jadroagent Rijeka

Rijeka, 12. 07. 1986

Odgovor urednika:

Spoštovani kolega R. Smokvina,

Hvala za vaše pismo, ki me je vzpodbudilo k nekoliko bolj izčrpnem odgovoru, saj vaša vprašanja dregajo v splošnejši koncept in prakso strokovne kritike in polemike v okviru računalništva in informatike.

Praksa časopisa Informatica so lahko samo tisti predmeti, za katere se uredništvo in predvsem avtorji opredeljujejo, da so relevantni in na določeni razvojni, kulturni stopnji tudi potrebni in zaželeni. Polemični in kritični prispevki so danes še kako potrebni, čeprav je pismenost na tem področju in sploh pri strokovnem delu šibka oziroma se ne privzgojuje in tako tudi ne prakticira. Seveda pa polemičnost in kritičnost vobče nista zaželeni, ker vznemirjata naša tradicionalno mirna delovna okolja. To sem celo ob izraziti nepolemičnosti časopisa Informatica izkusil tudi na svoji koži.

Ocenjevanje sistemskih in aplikativnih programskih paketov oziroma izdelkov je v razvitem strokovnem tisku domala standarizirano. Ocena ali benchmark je dovolj raznolična kvantitativna in kvalitativna primerjava zmogljivosti programskega izdelka, ki ob cenovni primerjavi daje napotek potencialnemu kupcu, da se lažje odloča za izbiro glede na svoj specifičen primer. Ob benchmarku je seveda zaželena še polemična obravnava, ki razjasnjuje tista mesta, določene semantične atribute, ki jih z standardnim ocenjevanjem ni mogoče zajeti. Časopis Informatica je široko odprt za vsako argumetirano polemiko in kritiko programskih izdelkov in seveda tudi za druga področja strokovne problematike.

S kolegom R. Smokvino se strinjam, da zadevni članek, ki ocenjuje programska orodja IDA, ne dosega ocenjevalnih standardov in da kakovost ocene ni nujno odvisna od naslova institucije, ki ji avtorji pripadajo. Zato sprejemam tihi in skrajno obzirni poziv R. Smokvine, da je določena pozornost, kakovost, strokovna doslednost pri ocenjevanju potrebna, če naj ocena prinaša korist tudi potencialnim interesentom. Strinjam se tudi, da ocene določenega produkta ni mogoče dati s prebiranjem priročnikov in da je lahko ozadje neke ocene le strokovno utemeljeno delo s produktom.

S pozdravom

A. P. Železnikar

Ljubljana, 13. 8. 1986

SRCE

SVEUČILIŠNI RAČUNSKI CENTAR
UNIVERSITY COMPUTING CENTRE

POZIV NA SUDJELOVANJE

9. MEĐUNARODNI SIMPOZIJ **»KOMPJUTER NA SVEUČILIŠTU«**
18 — 22. SVIBNJA 1987.

Mjesto:
DUBROVNIK/CAVTAT, HOTEL CROATIA

Organizator:
SVEUČILIŠNI RAČUNSKI CENTAR, ZAGREB

Teme:
- INFORMATIKA I OBRAZOVANJE
- RAČUNARSKI SISTEMI I MREŽE, OSOBNA RAČUNALA
- SOFTWARE-SKO INŽINJERSTVO
- INFORMACIJSKI SISTEMI I BAZE PODATAKA
- ANALIZA PODATAKA, STATISTIKA I STATISTIČKI SOFTWARE
- MODELIRANJE, SIMULACIJA I OPTIMIZACIJA
- DIZAJN I PROIZVODNJA POMOĆU RAČUNALA (CAD/CAM)
- UMJETNA INTELIGENCIJA I EKSPERTNI SISTEMI
- PRIMJENA INFORMATIČKIH SREDSTAVA I METODA U PRIRODNIM I DRUŠTVENIM ZNANOSTIMA
- DRUŠTVENI I PRAVNI ASPEKTI INFORMATIKE

Rok za sažetke (1-2 stranice):
15. STUDENI 1986.

Obavijest o prihvaćanju:
15. PROSINAC 1986.

Rok za radove:
15. VELJAČE 1987.

Struktura simpozija:
PREDAVANJA, POSTER SEKCIJE, PREZENTACIJE HARDWARE-A I SOFTWARE-A, PANEL DISKUSIJE, IZLOŽBE

Informacije:

Sekretarica simpozija

SVEUČILIŠNI RAČUNSKI CENTAR, 41000 Zagreb, Engelsova b.b., Jugoslavija
Tel.: 041/510-099, Tlx.: 21871

CALL FOR PAPERS

IFIP

WG 6.5

IFIP 6.5 International Working Conference on

MESSAGE HANDLING SYSTEMS

State of the Art and Future Directions

27 to 29 April 1987

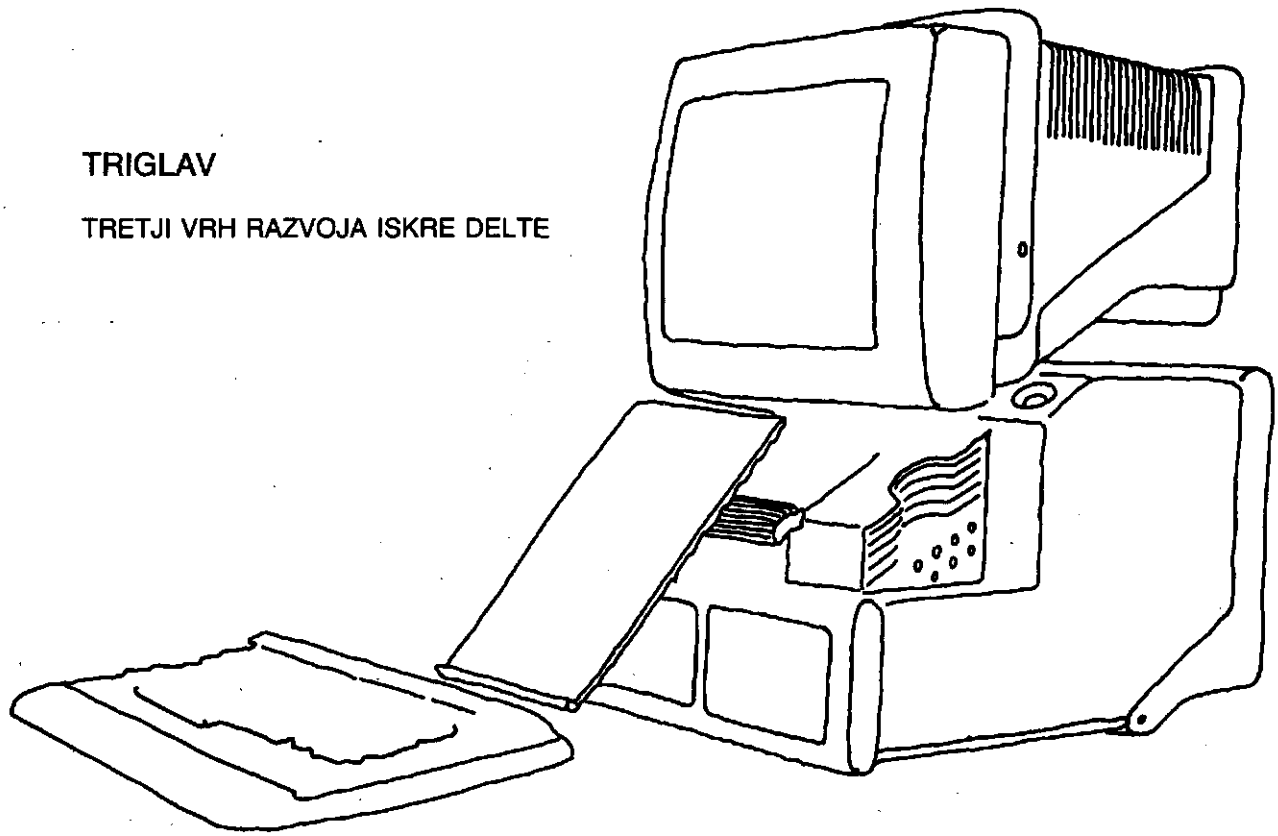
Munich
Fed. Rep. of Germany

Sponsored by IFIP TC 6
and the Gesellschaft für Informatik

To: Message Systems '87
Mrs. Stenzel
Siemens AG
D-AP.11
Otto-Hahn Ring 6
D-8000 München 83
Fed. Rep. of Germany

TRIGLAV

TRETJI VRH RAZVOJA ISKRE DELTE



Računalniški sistem TRIGLAV lahko deluje na treh procesnih enotah. Z enostavno menjavo procesorskih modulov in operacijskih sistemov je družina TRIGLAV kompatibilna z družinama mikro in miniračunalnikov vodilnih svetovnih proizvajalcev in seveda z računalniki in s programsko opremo ISKRE DELTE.

Sistem TRIGLAV je zasnovan za uporabo v:

- vodenju proizvodnje
- avtomatizaciji procesov
- robotizaciji
- kot grafično delovno mesto za projektiranje
- kot večuporabniški poslovni sistem
- kot komunikacijska enota

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

Tiskarna Kresija