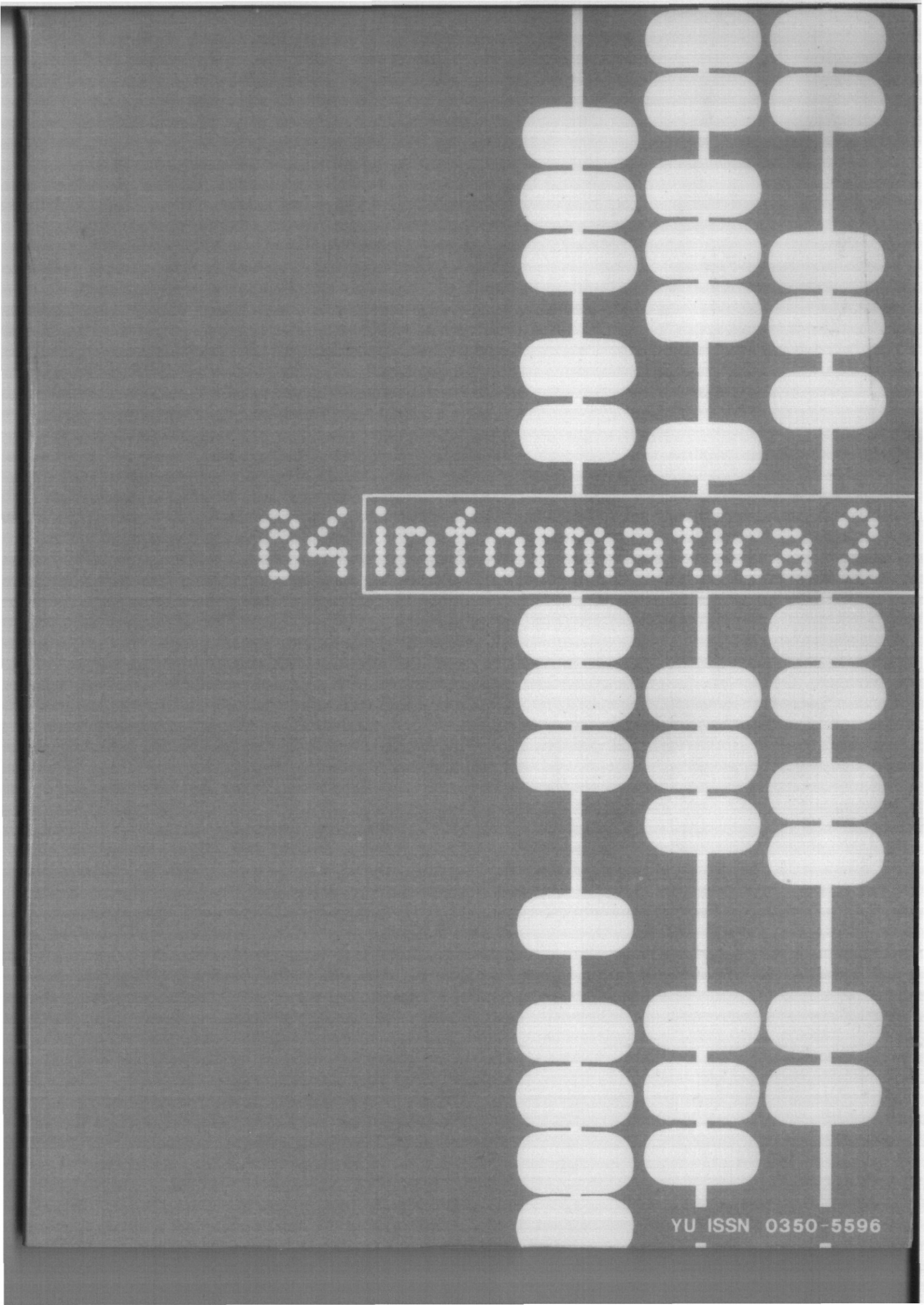


YU ISSN 0350-5596

INTERAKTIVNI GENERATOR PROGRAMOV

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

Iskra Delta

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

informatica

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

YU ISSN 0350–5596

LETNIK 8, 1984 – Št. 2

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

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

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

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije.

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

TISK: Tiskarna Kresija, Ljubljana

GRAFIČNA OPREMA: Rasto Kirn

V S E B I N A

V S E B I N A

informatica

YOURNAL OF COMPUTING AND INFORMATICS

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

YU ISSN 0350–5596

VOLUME 8, 1984 – No 2

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

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

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

C O N T E N T S

RAČUNARSKI SISTEM DELTA 800

MILOVAN V. JEFIĆ

DO ISKRA DELTA

UDK: 681.3

U članku je opisana konfiguracija, arhitektura i programska oprema sistema Delta 800.

Delta 800 Computer System. This article deals with configuration, architecture and Software system for Delta 800 Computer System.

Organizacija sistema DELTA 800

Centralni procesor

Centralni procesor je povezan na Delta sabirnicu kao subsistem i kontrolira njeno vremensko upravljanje, izvršava logične i aritmetičke operacije te dekodira instrukcije.

D800 je paralelni binarni 16 bitni sistem opće namene sa mogućnošću proširenja memorije do 4 MB. Ima zaštićenu memoriju sa osobinom relociranja, dovoljnu brzinu te hardversko rešeno množenja i deljenja. Upotrebljava instrukcijski set koji je subset instrukcijskog seta D4850 Upotrebljava prema tome i iste načine adresiranja.

Sadrži 8 šestnaest bitnih registara opšte namene, koji mogu biti upotrebljeni kao akumulatori, pokazivači, inkrementalni ili dekrementalni registri.

Poznaje četiri nivoa prekida, čiji je prioritet određen jedinicama na sabirnici. Svaka jedinica ima prekidni vektor, koji pokazuje direktno na rutinu za posluživanje prekida. Osim tih postoje i interni prekidi koji pokazuju na nestanak napajanja, vremenski problem na sabirnici, presipanje skladišta i sl.

Srednje vreme jednoga instrukcijskog ciklusa je 225 nsek.

Upravljanje memorijom određuje područja glavne memorije i aktivnosti: piše/čita, samo čita ili nema pristupa. Ono obezbeđuje rad u multiprogramskoj okolini uz rad procesora na dva načina: osnovni i korisnički.

U osnovnom načinu program ima kompletnu kontrolu i može izvršavati sve instrukcije. Taj način koriste monitori i super nadzorni programi.

U korisničkom načinu program ne može izvršavati sledeće instrukcije:

- menjati programe iz osnovnog načina,
- zaustaviti procesor,
- uzimati memorijski prostor predviđen za druge.

U slučaju multiprogramskog rada svaki korisnički program može biti memorijski rezidentan u svako vreme. U tom slučaju super nadzorni programi vrše funkciju kontrole izvršavanja korisničkih programa, dodeljuje memoriju i periferne jedinice.

Omogućeno je dinamičko dodeljivanje memorije, koje dolazi posebno do izražaja pri izvršavanju programa u višim jezicima, gde područje nije unapred određeno sa prevodiocem.

D800 je opremljen logikom, koja konstantno kontroliše napajanje te u slučaju prestanka napajanja inicializira automatski prekid.

Automatsko startanje sistema

Predviđeno je automatsko startanje sistema pri dovođenju napajanja uz obavezno predtestiranje.

Procesor aritmetike sa pokretnim zarezom

Izvršava svu aritmetiku sa pokretnim zarezom i pretvara podatke u format sa pokretnim zarezom.

Poseduje:

- jednostruku/dvostruku preciznost (32/64 bita),
- fleksibilne adresne načine,
- šest 64 bitna akumulatora,
- otkrivanje greške.

Brza memorija

To je mala brza memorija, koja kopira izabrane lokacije glavne memorije zbog bržeg ponovnog pristupa instrukcijama i podacima. Kapaciteta je 2 KB brze bipolarne memorije.

Glavna memorija

Kapacitet je 4 MB, a sastavljena je od 4 modula po 1 MB

ili 16 modula po 256 KB (zavisno od upotrebljenih elemenata).

Sabirnice

Delta sabirnica

To je dvostrana asinhrona sabirnica koja omogućava komunikaciju između bilo kojih jedinica, koje povezuje. Sastoji se od adresnih, podatkovnih i kontrolnih linija. Maksimalna brzina prenosa je 1.9 M bit/sek.

Memorijska sabirnica

Adresnim, podatkovnim i kontrolnim linijama povezuje memoriju i procesor (cache).

Instrukcijski set

Instrukcijski set sistema D800 je subset instrukcijskog seta sistema D4850. On je tako izabran, da koristu svu fleksibilnost registara opšte namene i za razliku od standardnih sistema ima umesto tri klase različnih instrukcija za sve operacije samo jednu klasu.

Upotrebljava jednostruke i dvostruke operande za reči i byt-ove.

Tipovi podataka

Instrukcijski set sistem D800 upotrebljava širok spektar tipa podataka, kao što su:

- celobrojni podaci i podaci sa pokretnim zarezom,
- alfanumerični nizovi,
- pakovni i nepakovani decimalni brojevi,
- uređene liste podataka,
- podatkovna polja promenljive dužine,
- specialne podatkovne liste.

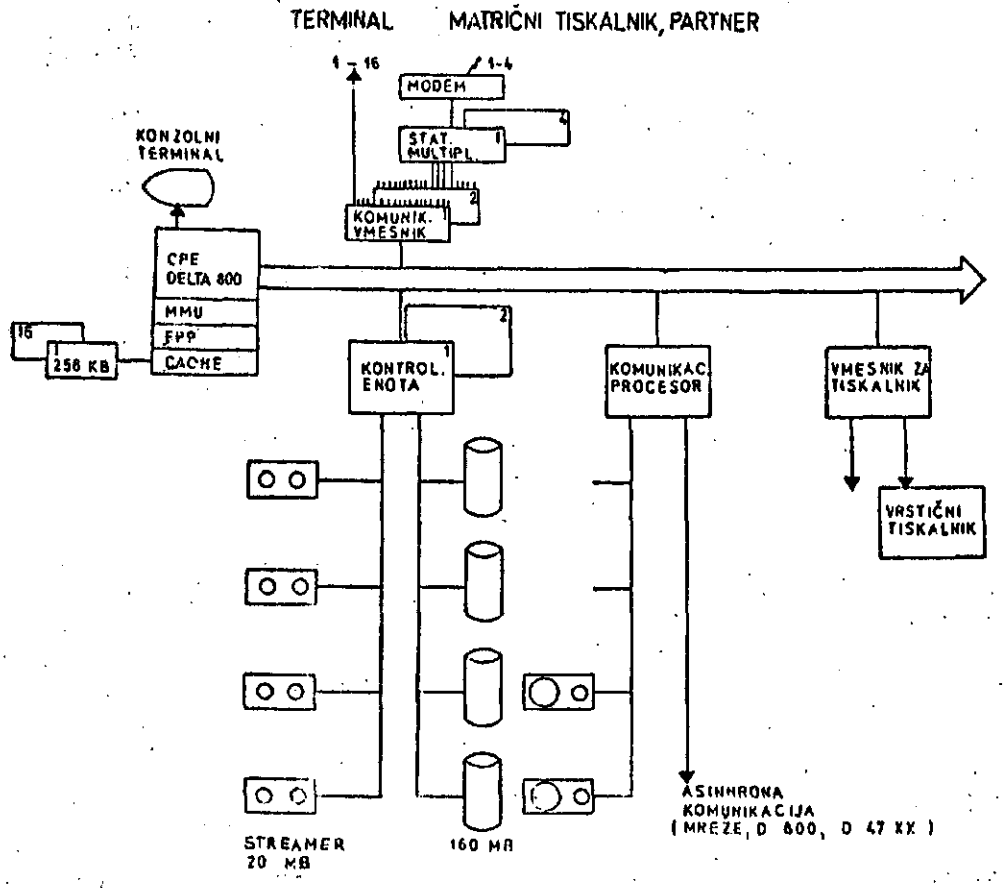
Sistem D800 upotrebljava instrukcije formata promenljive dužine i koristi osam načina adresiranja.

Upotrebljava sve standardne tipove instrukcija (logične, pomeračke, konverzione, bezuslovne, string te razne instrukcije posebnog značaja).

OPERACIJSKI SISTEM DELTA/M

Softversku podršku sistem D800 ima u OS DELTA/M. Da bi se omogućila kompletna kompatibilnost, uključujući i razvoj DELTA/M softvera na DELTA/V sistemu, implementirano je sledeće:

- hardverske podrške instrukcijskog niza, što omogućava prenos programa sa DELTA/M na DELTA/V sistem,
- sistemske usluge, sama kompatibilnost instrukcija nije dovoljna za prenos programa sa jednog sistema na drugi, jer programi upotrebljavaju drukčije sistemske usluge, što je neminovno kad je u pitanju drugi hardver. Delta/V ostvaruje kompletno rešenje tog problema, tako da za vreme izvođenja programa konvertira sistemske usluge Delta/M, u ekvivalentne Delta/V,
- komandni jezik da bi se pružila još kompletnija kompatibilnost i la-



gani prenos programa iz jedne u drugu okolinu, Delta/V podržava komandni jezik Delta/M operacijskog sistema.

Ovaj pristup omogućava i prenošenje programa, prevodioca, editora itd. sa delta/M na Delta/V sistem.

Registrovanje grešaka

Automatski se registriraju sve greške u radu hardverskih komponenti za vreme aktivnosti sistema.

Zaštita programa i podataka

Zaštita sistema, programa i podataka je ostvarena na sledeći način:

- kvote i privilegije, koje su dodeljene raznim korisnicima, osiguravaju da neovlašteni korisnici ne mogu ugroziti integritet sistema,
- kriptirane lozinke osiguravaju, da lozinku zna samo korisnik kojem ona pripada. Sistem drži samo kriptiranu lozinku i nije moguće dobiti original,
- datoteke i procesi imaju zaštitu prema korisniku, njegovoj radnoj grupi, sistemu i ostalima. Korisnik sam definira dozvolu pristupa svom procesu i datotekama svakom od navedenih kategorija.

Operacijski sistem DELTA/M

Delta/M operacijski sistem podržava veći broj programa i korisnika u isto vreme. Dodeljivanje procesorskog vremena je takoder bazirano na sistemskim dogadajima, koji mogu biti hardverski ili softverski. Sistem je načelno baziran na disku, ali postoji i verzija Delta/S, koji je rezidentan u memoriji. Delta/M hardver ne dozvoljava zadovoljavajuću implementaciju straničenja memorije i stoga ne podržano samo segmentiranje memorije. Medutim mehanizmom svapinga je postignuta mogućnost istovremenog izvodenja više programa nego što može da stane u fizičku memoriju.

Delta/M operacijski sistem čine sledeći elemenati:

- jezgro operativnog sistema,
  rutine i strukture koje iniciraju i kontrolišu izvodenje sistemskih i korisničkih procesa i pružaju interfejs na hardversku i softversku okolinu u kojoj se isti izvode.
- U/I podsistem,
  skup rutina i struktura koje u saradnji sa jezgrom omogućavaju procesima pristup do periferija.
- sistemski procesi,
  privilegirani procesi koji u suradnji sa jezgrom i U/I podsistemom ostvaraju odredenu strukturu na periferijama na koje se odnose.
- pomoćni programi,
  ti programi omogućavaju stvaranje okoline za razvoj i održavanje programa, podataka i sistema.

Navedeni elementi sistema su organizovani u celini koja izuzetno efikasno iskorištava mogućnosti hardvera. Ta celina pruža sledeću okolinu:

Korisnički proces

Svaki proces u sistemu ima svoj kontekst koji determinira:

- stanje procesa,
- hardverske i softverske potrebe procesa kao što su periferija, datoteke, komunikacija i zauzeće memorije.

Veličina adresirane memorije u procesu je hardverski ograničena na 64 KB ali veličina programa koji se izvodi u sklopu procesa ne podleže tom ograničenju jer svi delovi programa ne moraju uvek biti adresirani. Ti delovi programa se adresiraju na dva moguća načina:

- drže se u memoriji i sistem mapira proces na njih kad su procesu potrebni, na taj način može i više procesa istovremeno upotrebljavati zajedničke rutine odnosno podprograme,
- drže se na disku i unose u memoriju po potrebi.

Način adresiranja korisnik specificira sam, a sistem izvodi automatski. Proces ima dostup do periferija preko svojih logičkih jedinica, a ostvaruje ga sa posredovanjem jezgra odnosno U/I podsistema, koji vrši proveru radi očuvanja integriteta sistema i ostalih procesa.
Procesi sinhroniziraju svoj rad sa okolinom i ostalim procesima pomoću dogadajnih variabli koje mogu biti i zajedničke odnosno globalne.

Dodeljivanje procesorskog vremena

Dodeljivanje se vrši preko

- prioriteta,
- izkazane potrebe na osnovi dogadaja u sistemu (U/I operacija, vrednost dogadajne variabile, itd.), sistem sam prouzrukuje dogadaj ako je to potrebno da bi ravnopravno razdelio sistemske mogućnosti medu korisnicima sa istom prioritetom.

Dodeljivanje memorije procesima

Kad je procesu dodeljeno procesorsko vreme može doći do sledećih situacija:

- proces se nalazi u memoriji,
  taj slučaj je najmanje problematičan i proces može odmah nastaviti sa izvodenjem svog programa,
- proces se nalazi na disku,
  u tom slučaju se proces mora prvo pročitati sa diska u memoriji. U slučaju da nema dovoljno memorije na raspolaganju, sistem kompaktira slobodnu memoriju i ako je to potrebno preseli procese sa najnižom prioritetom na disk, proces kojem je dodeljen procesor se učitava u slobodnu memoriju i može odmah nastaviti sa izvodenjem svog programa. Ovde se mora istaknuti, da je preseljavanje procesa u memoriju ili disk isključivo u nadležnosti sistema i da procesi nisu svestni, da je njihovo procesiranje bilo prekinuto.

Komunikacija izmedu procesa

Sistem omogućava potpunu sinhronizaciju i komunikaciju medu procesima. To je ostvareno preko sledećih mehanizama:

- implicitne komunikacije preko zajedničkih podataka u memoriji ili u fajl sistemu,
- slanjem i primanjem poruka izmedu procesa,
- operacijom preko dogadajnih variabili, koje dobivaju vrednost: pri dogadajima u sistemu, na primeru U/I operacije eksplicitno upotrebom sistemskih usluga.

Komunikacija preko računarskih mreža

Komunikacija preko računarskih mreža je implementirana na svim nivoima u operacijskom sistemu.

Sistemske usluge

Sistemske usluge u ime procesa obavlja jezgro, integritet jezgra je hardverski zaštićen, tako da greške u procesu ne mogu ugroziti integritet jezgra odnosno sistema u celini, osim u slučaju, da je korisnik ovlašten za takav pristup sistemu (privilegiran).

Sistemske usluge se mogu razvrstati u sledeće kategorije:

- kontrola izvršenja procesa,
  te usluge omogućavaju iniciranje, zaustavljanje itd. vlastitog i ostalih procesa u sistemu,
- kontrola stanja procesa,
  omogućavaju menjanje prioriteta i drugih uslova koji utiču na dodeljivanje procesorskog vremena i memorije,
- informacije,
  ove usluge proces koristi, da bi od jezgra dobio informacije o stanju sistema i procesa, koji se u njemu izvode,
- procesiranje događajnih variabili,
  omogućava procesu, da se sinhronizira sa vanjskim događajima kao što su U/I operacije i stanje ostalih procesa u sistemu,
- kontrola prekida,
  omogućava procesu da prati, kontrolira i presretava softverske i hardverske prekide,
- U/I usluge omogućavaju procesima sledeće operacije: povezivanje sa logičnim jedinicama, odnosno periferijama, izvršenje logičnih, virtualnih i fizičkih U/I operacija, povezivanje sa hardverskim prekidima, povezivanje sa procesorom komandnog jezika. Ove usluge omogućavaju:
  - prevazilaženje hardverskih ograničenja u veličini adresnog prostora procesa,
  - upotrebu i stvaranje zajedničkih rutina, podprograma i podataka,
  - komunikacije izmedu procesa,
  - operacije na računarskoj mreži u kojoj se nalazi sistem.

Fajl sistem i dostup periferijama

Procesiranje U/I operacija se sastoji od rutina, koje omogućavaju direktan pristup periferiji kao i od programa koje toj periferiji daju generalizovanju fajl strukturu. Na fajl strukturi se datoteke mogu otvarati, čitati, pisati, zatvarati itd. Tako je omogućena i zavisnost i nezavisnost procesa od periferija, koje upotrebljava. Jednostavno su ostvarene asinhrone operacije na periferijama, tako da se U/I i procesiranje mogu preklapati odnosno vršiti istovremeno. Fajl sistem je sastavljen od direktorija i datoteka, koje se u njima nalaze.

Fajl sistem podržava sledeće organizacije podataka:
- sekvencijalne
- relativne
- indeksne

sa sekvencijalnim i direktnim pristupom do slogova.

Baze podataka

Sistem upravljanja bazom podataka podržava mrežnu organizaciju kao i direktan dostup fajl sistemu. Jezik za definiciju podataka podržava:

- šeme,
- podšeme,
- skupove.

Jezik za rukovanje podacima omogućava jednostavno čitanje, brisanje i održavanje baze podataka. Uz program za upravljanje postoje i pomoćni programi za:

- kreiranje šema, podšema itd.,
- održavanje baze podataka,
- administratora baze podataka.

Baza podataka je kompatibilna sa bazom na sistemu Delta$_4$ /V.

Komandni jezik

Komandni jezik je jednostavan, konsistentan i veoma širok. Omogućava potpunu kontrolu nad delovanjem sistema, procesa koji se u njemu izvode i podataka koje on obrađuje. Komande omogućavaju:

- potpuno interaktivni razvoj programa,
- operacije sa periferijom i fajl sistemom,
- interaktivno iniciranje i kontrolu procesa u sistemu,
- komunikaciju sa ostalim korisnicima sistema,
- pomoć pri upotrebi komandi.

Komandne procedure

Komandna procedura je datoteka, koja se sastoji od komandi u komandnom jeziku. Komande su povezane jezikom za komandne procedure te se izvode na način sličan paketnoj obradi. Komandne procedure mogu upotrebljavati sve mogućnosti komandnog jezika.

Razvoj i održavanje programa

Delta/M operacijski sistem kompletno podržava okolinu za razvoj programa, koju se može razdeliti na sledeće:

- editori,
  editor za programe, koji iskorištava sve mogućnosti video terminala, editor za formatiranje dokumenata i drugog teksta,
- programski jezici,
  podržan je veliki broj jezika: ASEMBLER (najviše za sistemske programe), COBOL (za komercialnu obradu podataka), BASIC (za jednostavnije programe, koji se brzo pišu), FORTRAN (za kalkulacije), PASCAL (za algoritme),
- linker,
  linker iz objektnih modula stvara kopiju procesa, koja se čuva u fajl sistemu i može se uvek inicirati iz komandnog jezika ili procesa. Linker obavlja sledeće operacije:
  - definira virtualne adrese u programu,
  - razrešava međumodularne reference,
  - inicijalizira fajl kopiju procesa,
  - podržava jezik za raspodelu programa u delove, koji ne moraju istovremeno biti mapirani. Delovi se drže u memoriji ili na disku (po specifikaciji) i sistem se sam brine o pravovremenom mapiranju. Na taj način se prevazilazi hardversko ograničenje o vir-

tualnom adresnom prostoru,
- stvaranje zajedničkih rutina i podprograma.

Program za interaktivno otkrivanje, lociranje i odstranjivanje grešaka u programima

Pomoću ovog programa mogu se brzo locirati i odstraniti greške u programima. Podržane su sledeće operacije prilikom izvođenja programa, koji se testira:

- zaustavljanje na bilo kojem mestu u programu,
- referenciranje adresa i registara u programu,
- izvođenje programa instrukciju po instrukciju,
- promena redosleda izvođenja instrukcija,
- promena vrednosti bilo koje adrese ili registra u programu,
- promena instrukcija u programu.

Bibliotekar

Bibliotekar podržava sledeće biblioteke:

- asemblerske module,
- objektne module,
- univerzalne module.

Sistem za upravljanje formama

Sastoji se od:

- editor za ekranske forme,
- biblioteka ekranskih formi,
- drajver za ekranske forme.

Pomoću formi se difenira izgled ekrana i polja, na kojima se vrši unos podataka.

Drugi pomoćni programi

Postoji širok spektar pomoćnih programa, koji uklanjaju potrebu za pisanjem programa za izvršenje čestih operacija:

- paket za obračun upotrebljenosti sistemskih mogućnosti (memorija, procesorsko vreme, U/I itd.),
- procesor za određivanje lozinki, privilegije i direktorija,
- program za arhiviranje i kopiranje diskova i magnetnih traka,
- program za transfer datoteka na druge medije i operativne sisteme,
- program za održavanje direktorija i datoteka,
- program za instalaciju novih sistemskih procesa,
- program za kontinuirano osmatranje procesa, memorije, periferije itd.,
- program za slanje poruka među korisnicama,
- program za editiranje u paketnoj obradi,
- program za popravljanje objektnih modula,
- program za popravljanje fajl kopije procesa,
- program za ispitivanje spiska otkrivenih grešaka,
- paket za generaciju sistema,
- program za formatiranje diskova,
- program za otkrivanje i označavanje neupotrebljivih blokova na disku,
- program za verificiranje fajl strukture,
- program za spašavanje datoteka iz nekonzistentne fajl strukture,
- program za sortiranje,
- program za analizu i konverziju datoteka,
- program za pregled datoteka,
- program za pravljenje liste referentnih simbola,
- program za uporedivanje datoteka,
- program za ispravke fajl kopije procesa.

Kontrola reda čekanja na štampač

Drži se red zahteva za štampač, a zahtevi se izvršavaju jedan po jedan, da se štampanje ne bi mešalo. Korisnik ne mora čekati, da se štampanje završi, već može neometano nastaviti rad na sistemu.

Registrovanje grešaka

Automatski se registriruju sve greške u radu hardverskih komponenti za vreme aktivnosti sistema.

Zaštita programa i podataka

Zaštita korisnika i sistema je ostvarena na sledeći način:

- korisnik ne može ugroziti sistem, ako mu za to nije dato ovlaštenje,
- kriptirane lozinke osiguravaju, da lozinku zna samo korisnik, kojem ona pripada. Sistem drži samo kriptiranu lozinku i nije moguće dobiti original,
- datoteke i procesi imaju zaštitu prema korisniku, njegovoj radnoj grupi, sistemu i ostalima. Korisnik sam definira dozvolu pristupa svojim datotekama svakom od navedenih kategorija.

Komunikaciona podrška na računarskim sistemima DELTA

Prateći svetska kretanja na području računarskih mreža, i prateći zahteve domaćeg tržišta, Iskra Delta je objavila u svojem proizvodnom programu niz produkata vezanih uz ovu problematiku.

Komunikacioni produkti Iskra Delte dele se u sledeće kategorije:

- programski paketi za komunikaciju sa računarima drugih proizvođača (IBM, CDC, UNIVAC itd.),
- programski paketi za komunikaciju inteligentnim terminalima,
- programski paketi za realizaciju međuračunarskih mreža.

Komunikacije sa računarima drugih proizvođača

U ovoj oblasti programski paketi pokrivaju uglavnom emulacije određenih uređaja drugih računarskih sistema. Pri tome je kod emulacije nastojano postići čim bolju iskoristivost mogućnosti arhitekture Delta sistema. Emulatori se uglavnom dele na dve klase:

- emulatori paketne obrade,
- emulatori za interaktivan rad.

Za svaku od ovih klasa postignute su bolje funkcionalnosti od originalnih uređaja i to sa aspekta korištenja U/I uređaja na Delta računarima i mogućnostima proširenih programskih emulacija tastatura, ekrana i sl. Bitna karakteristika je mogućnost prenosa datoteka (i automatska konverzija) za protokole i uređaje, koji to ne podržavaju u originalnim konfiguracijama.

Na ovom području veoma je intenzivan razvojni program u Iskra Delti i to na sledećim zadacima:

- implementacija prenosnog protokola X.25 standardu i prilagodavanje logike protokola budućoj javnoj mreži u SFRJ,
- implementacija programskog okruženja za bazu podataka, koja će transparentno delovati u celoj mreži,
- razvoj funkcionalnih specifikacija za inteligentne mrežne kontrolere u cilju poboljšavanja propusnosti centralnih sistema,
- implementacija veznog modula za računarske mreže drugih proizvođača.

Komunikacije sa inteligentnim terminalima

U ovu grupu dolaze programski paketi, koji omogućavaju manjim računarskim sistemima korištenje svojih lokalnih funkcija i integraciju rezultata lokalnih obrada ili transakcija u centralni računarski sistem. Funkcije,koje su ovdje realizirane su:

- emulacija originalnih "Delta" terminala,
- prenošenje i konverzija datoteka,
- komuniciranje između programa.

Primena ovih paketa svodi se uglavnom na:

- inteligentne bankarske ili slične terminale,
- povezivanje sa procesorima teksta,
- povezivanje sa "Delta" mikro računarima.

Računarska mreža

Paket za realizaciju računarske mreže omogućava komunikaciju među računarima istog hijerarhijskog nivoa, vezanih u proizvodnju topologiju. Funkcije, koje paket omogućava su:

- automatsko transportiranje podataka i funkcijskih zahteva (routing),
- komuniciranje između programa na dva računara,
- emulacija lokalnog terminala na udaljenom računaru,
- prenos datoteka između dva računara.

Pri gornjim funkcijama podrazumjevaju se bilo koja dva računara u mreži (nije važno da su susjedni).

VEČOPRAVILNA TEHNIKA V EDX OKOLJU V RAČUNALNIKU IBM SISTEM/1

KRATEK PREGLED

TOMAŽ ERJAVEC

INTERTRADE, TOZD ZASTOPSTVO IBM, CENTER ZA RAZVOJ PROGRAMSKE OPREME

UDK: 681.3-181.4

Sestavek opisuje nadzor opravil v večuporabniškem večopravilnem operacijskem sistemu EDX na računalniku IBM Sistem/1. Prikazano je upravljanje z opravili na osnovi programskih prioritet, definiranje in signaliziranje dogodkov ter ter način, s katerim si opravila prisvajajo realne ali abstraktne naprave v izključni nadzor.

MULTITASKING TECHNIQUE IN EDX ENVIRONMENT ON AN IBM SERIES/1 COMPUTER - The paper describes the task control in a multiuser multitasking operating system EDX on an IBM Series/1 computer. Task control on the basis of priority, definitions and signalling of the events and the ways of acquiring exclusive control over real or abstract devices are presented.

1. Uvod

Večopravilni sistemi postajajo danes nujnost in kar težko si že zamišljamo programiranje v jezikih ali celo sistemih, ki večopravilnega programiranja ne podpirajo. Če pomislimo, da se večopravilne tehnike pojavljajo že v najcenejših hišnih mikroračunalnikih v cenovnem razredu 100 dolarjev, potem zsornja trditev najbrž ne bo pretirana.

V tem zapisu predstavljamo opis bistvenih točk večopravilne tehnike na računalniku Sistem/1. V to tematiko bi nedvomno sodilo še mnogo več, kot je zapisano na naslednjih straneh, tako s stališča strojne opreme, operacijskega sistema EDX in še posebno jezika EDL, ki v nadaljnem besedilu sploh ni omenjen. Zato na tem mestu nekaj besed o EDX in EDL.

Event Driven Executive (Dogodkovno gnani izvajalnik) je operacijski sistem, ki tesno sodeluje z Event Driven Language (Dogodkovno gnani jezik). Že njuni imeni povedarjata, da bodo vpleteni procesni elementi, dogodki, simultanost akcij itd. V EDX se nahajajo 'emulator', katerega rutine izvajajo instrukcije jezika EDL. Z njimi lahko posegamo dokaj globoko v operacijski sistem. Po tej plati je EDL interpreter. Toda programe, ki jih napišemo v EDL jeziku, moramo prevesti. Pri tem se v prevodu nahajajo, namesto strojne kode, naslovi rutin, ki bodo izvajale operacije nad spremenljivkami v programu.

EDL je po eni strani zelo podoben zbirnemu jeziku (nekatere instrukcije opravijo identične naloge kot instrukcije v zbirniku Sistema/1), po drugi strani pa je to visok strukturiran jezik s standardnimi stavki.

2. Osnovni opis procesorja in strojnih nivojev

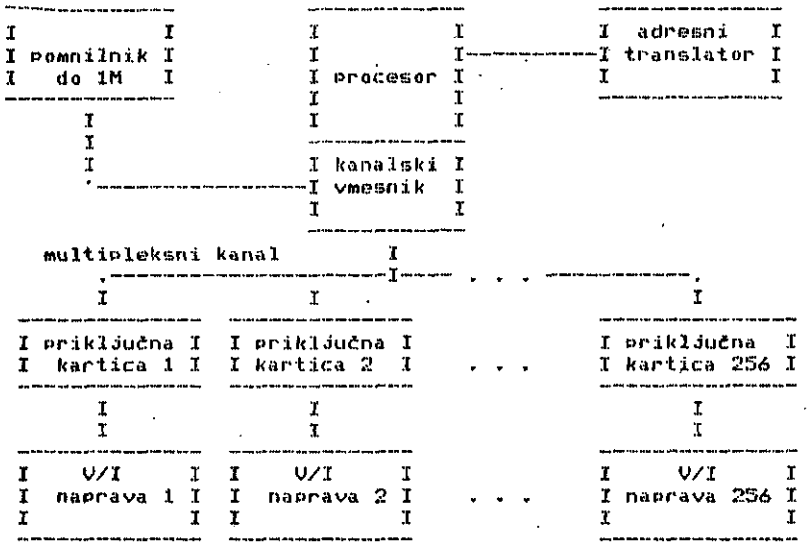
Sistem/1 je 16-bitni mini računalnik in je bil v prvi meri namenjen računalniškim komunikacijam, kasneje pa so ga začeli uporabljati tudi v robotiki in celo v poslovnih aplikacijah. Arhitektura sistema je v grobih potezah skicirana na Sliki 1.

Pomnilnik je razdeljen v particije po 64K. Naslavljanje do 1M je izvedeno preko adresnega translatorja, ki s pomočjo ključa particije dosega pomnilne lokacije v različnih particijah. Kanal je mikroprocesorsko krmiljen in lahko veže na procesor do 256 enot. Kanalni vmesnik ima neposreden dostop do pomnilnika in tako razbremenjuje procesor pri V/I operacijah.

Nivojsko delovanje procesorja

Zaradi lažjega obvladovanja prekinitev ima procesor štiri strojne nivoje. Celoten nabor procesorjevih registrov je v vsakem nivoju ponovljen. V odvisnosti od prioritete se različna opravila odvijajo na različnih nivojih. Ker je procesor le eden, prihaja do preklapljanja med nivoji in procesorjem. Če na nižjem nivoju ni nobenega opravila, dobi prvi višji nivo procesorjev nadzor. Princip je shematično prikazan na Sliki 2.

Pri operacijskem sistemu EDX sta strojna nivoja 0 in 1 rezervirana za sistemske programe, nivoja 2 in 3 pa sta namenjena uporabniku, da v njiju razporedi svoja opravila na 510 programskih prioritetah.



Slika 1.: Arhitektura Sistema/1

3. Osnovni koncept PROGRAM / OPRAVILO

Opravilo je osnovna enota dela, predanega procesorju v izvršitev. Program je zbirka enega ali več opravil na disku, ki jo lahko računalnik hkrati ali po delih včita v pomnilnik in preda procesorju v izvajanje. Prioriteto opravila določi programer (med 1 in 510). Opravila s prioriteto med 1 in 255 se izvajajo na strojnem nivoju 2, med 256 in 510 pa na nivoju 3. Prioriteta 1 je najvišja. Prioriteta 0 je rezervirana za sistem.

Načini vstopanja opravil v sistem

Programi sestojijo iz primarnih in sekundarnih opravil. Opravila, ki jih predstavlja zaključen program brez definicije nekega drugega opravila, so primarna opravila. Ko se tak program včita v pomnilnik, primarno opravilo avtomatično dobi nadzor procesorja, ko pride na vrsto.

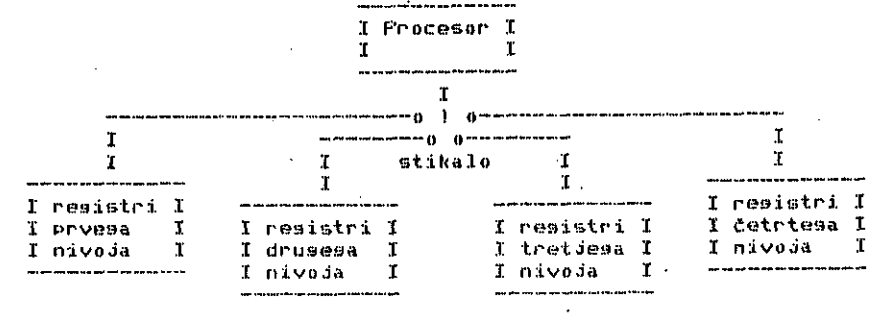
Če je v programu definirano novo opravilo, sa imenujemo sekundarno opravilo in ne more avtomatsko dobiti nadzora procesorja. Le-tega mu preskrbi programer z definiranjem točke, kjer naj neko sekundarno opravilo vstopi v dogajanje.

Opravila lahko vstopijo v sistem preko programov, ki jih poženejo v sistemu že delujoči programi. Možnost pošiljanja drugih programov imajo samo primarna opravila, sekundarna pa ne. Opravila, ki na tak način vstopijo v sistem, so popolnoma enakopravna in se z vsemi ostalimi opravili, ki so že od prej v sistemu, borijo za procesorjev nadzor.

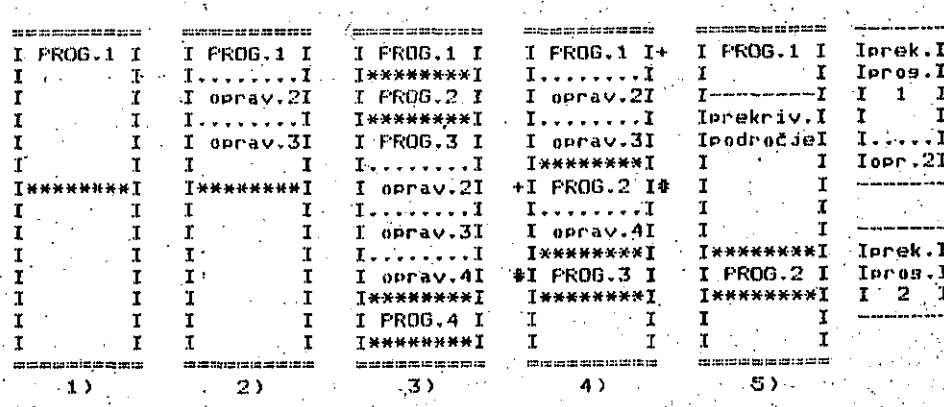
Zadnji način vstopanja v sistem je s prekrivanjem. Primarno opravilo nekega programa lahko rezervira prostor v pomnilniku za največjega od prekrivnih programov, ki bodo vstopili v sistem kasneje in tako zagotovi prostor za včitanje. Tudi taka opravila so enakopravna z vsemi ostalimi, vrstni red izvajanja je odvisen le od prioritete. Prekrivni programi ne morejo včitati novih prekrivnih programov.

Sistemski nadzorni program smatra vsako opravilo za diskretno delovno enoto in ne razlikuje med primarnimi in sekundarnimi opravili. Vsa imajo enako možnost pridobitve procesorskega nazora v okviru svoje prioritete, ne glede na to, katerega delujočega programa del so, kateri uporabnik jih je poslal ali po katerem vrstnem redu so vstopila v sistem.

Na Sliki 3.: so skicirane različne možnosti programov in opravil v pomnilniku.



Slika 2.: Procesorski nivoji



Slika 3.: Razporeditve programov/opravil v pomnilniku

Na sliki 3. so opisani naslednji primeri:

1. V pomnilniku je en sam program, ki nima definiranega nobenega sekundarnega opravila. V sistemu je eno primarno opravilo.

2. Edini program ima definirani še dve sekundarni opravili. V sistemu je eno primarno opravilo in dve sekundarni.

3. V pomnilniku so štirje programi. PROG.1 ima eno primarno opravilo. PROG.2 ima dve primarno opravilo. Prog.3 ima tretje primarno opravilo in še tri sekundarna opravila. PROG.4 ima četrto primarno opravilo. Če je PROG.3 aktiviral vsa tri sekundarna opravila, je v sistemu sedem enakopravnih opravil (odvisno od prioritete).

4. PROG.1 ima poleg primarnega opravila še dve sekundarni. Primarno opravilo je poklicalo PROG.2 v pomnilnik in sa posnalo. PROG.2 ima poleg primarnega še eno sekundarno opravilo. Njegovo primarno opravilo je poklicalo PROG.3 in sa posnalo. PROG.3 ima samo primarno opravilo. Če so vsa sekundarna opravila aktivirana, je v sistemu šest opravil.

5. V pomnilniku sta dva programa: PROG.1 s prekrivnim področjem in PROG.2. Prekrivno področje je tako veliko, da sre vanj največji prekrivni program, ki sa bo poklical PROG.1 (to je prekrivni program 1). Prekrivni program 1 ima poleg osnovnega opravila še eno sekundarno opravilo. Prekrivni program 2 ima samo osnovno opravilo. Če je v prekrivnem področju Prekrivni program 1 in je sekundarno opravilo aktivno; so v sistemu štiri opravila.

4. Nadzor nad opravili

Večopravilna tehnika je potrebna pri programih, ki zahtevajo vzporedno izvajanje različnih nalog. Vsako opravilo se izvaja neodvisno od ostalih opravil in se z njimi bori za procesorski nadzor po principu FIFO (first in first out) v okviru prioritete. Ker tečejo opravila neodvisno, obstajajo nadzorne funkcije, s katerimi je moč sinhronizirati dogajanje. Ob vsakem klicu take funkcije, se izvajanje opravila prekine; izvede se funkcija, nakar nadzorni program odloči, kako bo v nadaljnje obravnaval prekinjeno opravilo. Kot rezultat omenjene procedure se opravilo lahko nahaja v enem od naslednjih petih stanj:

1. ODKLOPLJENO: opravilo je v pomnilniku kot del včitanega programa, a ni posnano. Ob včitanju programa so vsa opravila v tem stanju. Funkcija ATTACH seznani sistem o obstoju opravila in sa požene. Od tu dalje se opravilo bori z ostalimi opravili za nadzor procesorja. Po izvedbi funkcije ATTACH se opravilo nahaja v enem od naslednjih štirih stanj.

2. PRIPRAVLJENO: Opravilo je pripravljeno za uporabo procesorja, ki pa trenutno dela z drugim opravilom. V tem stanju je lahko poljubno število opravil. Ko je procesor prost, nadzorni program na osnovi prioritete opravil določi, katero od opravil po prišlo v aktivno stanje.

3. AKTIVNO: Opravilo je na najvišji prioriteti na svojem strojnem nivoju. Nima še nadzora procesorja, ker obstaja na nižjem strojnem nivoju opravilo, ki ima prednost (nižji strojni nivo ima prioriteto pred višjim).

4. V IZVAJANJU: Opravilo ima nadzor procesorja. Ima najvišjo prioriteto na svojem strojnem nivoju in na nobenem nižjem strojnem nivoju ni nobenega opravila v stanju AKTIVNO.

5. ČAKAJAOČE: Opravilo ni pripravljeno za nadzor procesorja, ker čaka na nek dogodek ali pa na dostop do neke naprave.

Za preklapljanje stanj nekega opravila so na voljo nadzorne funkcije, ki delujejo po naslednjih pravilih:

1. ATTACH preklopi opravilo iz stanja ODKLOPLJENO v stanje PRIPRAVLJENO. Ker opravilo ne more dvakrat zaporedoma preiti v pripravljeno stanje, funkcija ATTACH vsakič testira TCB (Task Control Block), ali ni opravilo morda že v pripravljenem stanju. V taken slučaju se ne zgodi nič, nadzor pa je vrnjen opravilu, ki je izdalo funkcijo ATTACH. Če ima klicano opravilo prednostno prioriteto pred kličočim opravilom, potem je kličoče opravilo prekinjeno in postavljeno na sklad pripravljenih opravil v svoji prioritetni stopnji, postavijo se ustrezni zaznamki v TCB-jih obeh opravil, klicano opravilo je zaznamovano kot pripravljeno in nadzor predan nadzornemu programu, da postavi klicano opravilo v aktivno stanje (ker ima višjo prioriteto).

2. DEATCH preklopi opravilo iz stanja V IZVAJANJU v stanje ODKLOPLJENO. Opravilo lahko odklopi samo sebe, ne pa nekega drugega opravila. Funkcija DETACH izvede funkcijo POST za "end ECB" (Event Control Block) odklapljanega opravila. Ta zaznamek pove, da je opravilo odklopljeno. Vsa opravila, ki čakajo na ta dogodek v stanju ČAKAJOČE, so preklopljene iz čakajočega stanja v stanje PRIPRAVLJENO.

   Po zaključku funkcije DETACH se nadzor ne vrne v opravilo, ki jo je poklicalo. Če je kasneje ponovno izvedena funkcija ATTACH za to opravilo, se izvajanje nadaljuje na instrukciji, ki sledi instrukciji DETACH.

3. WAIT in ENQ funkciji imata podobni storitvi. Prva povzroči čakanje nekega dogodka, druga želi ekskluzivni nadzor nad neko napravo. Če ECB (Event Control Block) na katerega se sklicuje WAIT funkcija ali QCB (Resource Control Block) na katerega se sklicuje ENQ funkcija je zaznamovan, potem se je pričakovani dogodek že zgodil ali pa je naprava, ki jo želi ENQ funkcija, dostopna.

   Če ECB ali QCB ni zaznamovan, se dogodek še ni zgodil oziroma je naprava zasedena. Zato mora opravilo, ki je izdalo WAIT ali ENQ, preiti v stanje ČAKAJOČE. Status opravila je shranjen, opravilo pa postavljeno v sklad opravil, ki čakajo na dogodek ali pa na dostopnost neke naprave. Če je pri ENQ funkciji kličoče opravilo obenem opravilo, ki vsebuje QCB blok zahtevane naprave, potem se nadzor takoj vrne v kličoče opravilo.

4. POST in DEQ funkciji opravljata podobni storitvi. Koda, ki je podana s funkcijo, najprej shranita v ECB dogodka ali QCB naprave. Koda služi uporabniku kot zaznamek. Če nobeno opravilo na čaka na dogodek ali napravo, potem se nadzor vrne v opravilo, ki je poklicalo funkcija.

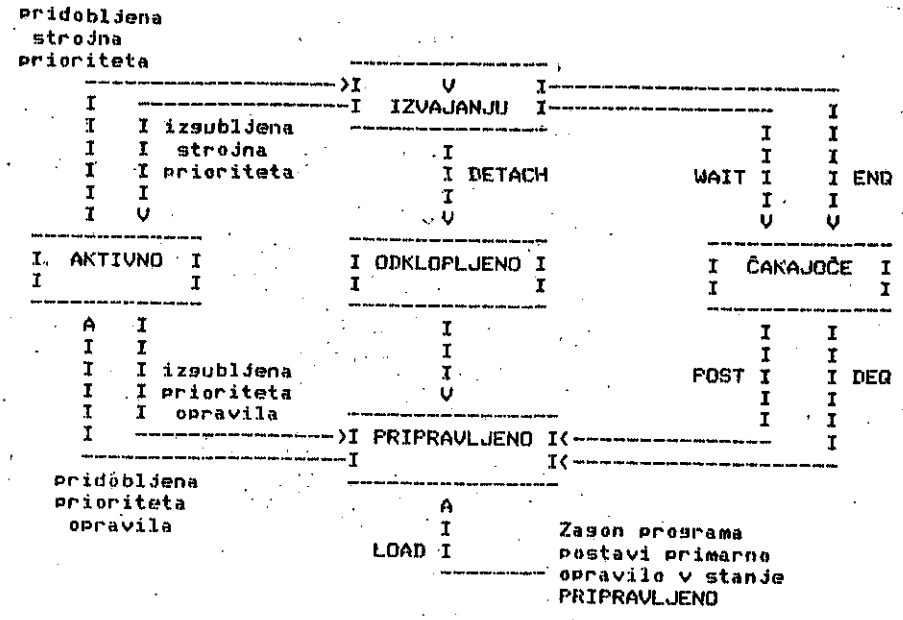
   Če kakšno opravilo čaka na dogodek ali dostopnost naprave, potem se status kličočega opravila shrani in opravilo se postavi na sklad pripravljenih opravil. Procesor je namenjen nadzornemu programu, da preklopi vsa čakajoča opravila iz stanja ČAKAJOČE v PRIPRAVLJENO. Če je bila klicana funkcija DEQ, potem se samo prvo čakajoče opravilo na skladu čakajočih postavi v stanje PRIPRAVLJENO. Nadzorni program nato na podlagi prioritete preklopi neko opravilo v stanje AKTIVNO.

Opisane funkcije in njihov vpliv na prehajanje med stanji so shematično predstavljeni na Sliki 4.

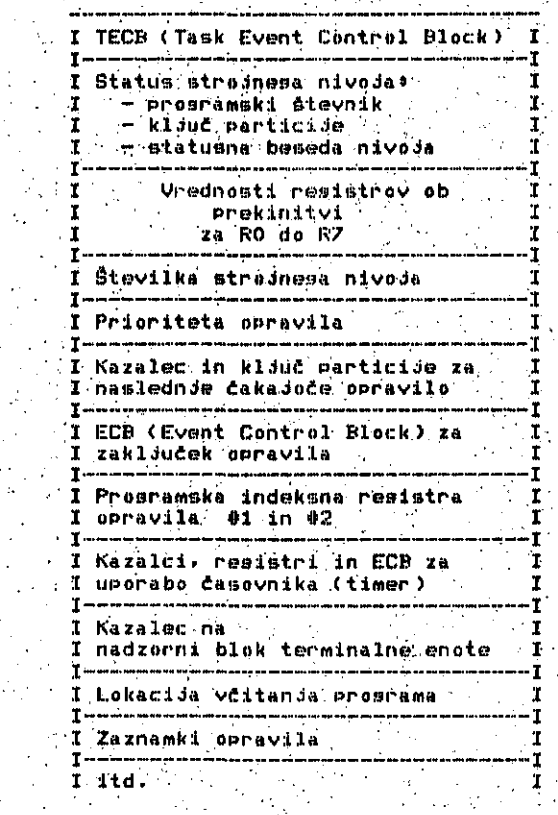
5. Sinhronizacija opravil

Nadzor nad stanjem opravil se vrši preko nadzornih blokov. Nekatere od blokov generira sistem sam, druge definira uporabnik po potrebi.

Vsako opravilo ima avtomatično priključen nadzorni blok opravila TCB (Task Control Block). Nadzorni program se uporablja za izvajanje svojih funkcij. V njem so zapisani vsi trenutni parametri in status nekega opravila. Poleg parametrov vsebuje tudi prostor za shranjevanje vrednosti registrov ob prekinitvi in delovne lokacije.



Slika 4.: Prehajanja med stanji opravil



Slika 5.: Najvažnejši podatki iz Nadzornega bloka opravil

Nadzorni program podpira sinhronizacijo in nadzor nad opravili z naznanjanjem dogodkov, čakanjem na dogodke, brisanjem informacije o dogodku, jemanjem in sproščanjem ekskluzivnega nadzora nad neko realno ali abstraktno napravo.

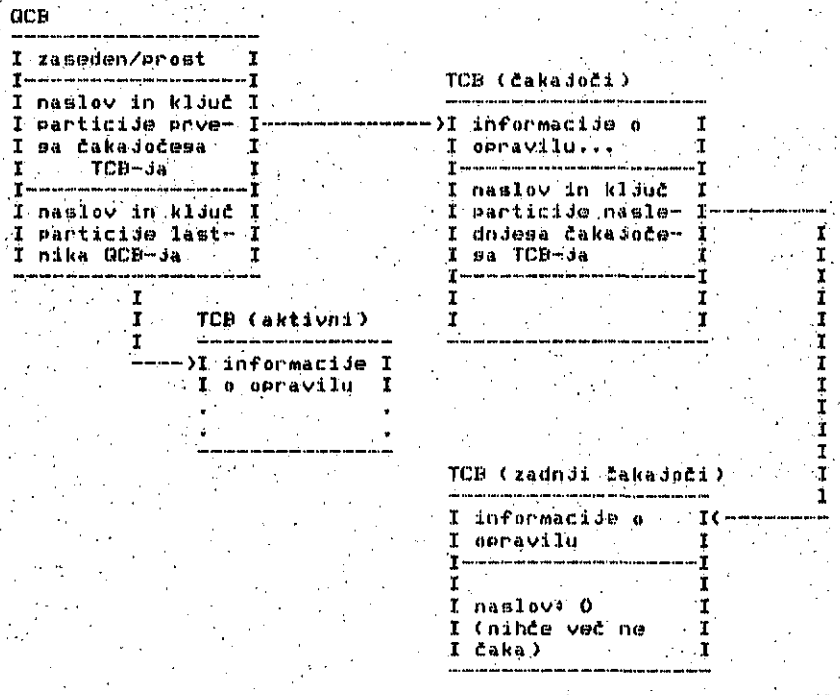
QCB in ECB

V operacijskem sistemu EDX so realne ali abstraktne naprave in dogodki predstavljeni z nadzornimi bloki po imenu QCB oziroma ECB (Resource Control Block oziroma Event Control Block), katerih naslovi morajo biti poznani sodelujočim opravilom. Za QCB je mogoče definirati dve stanji: "zaseden" ali "prost", kar po vrsti pomeni, da si je neko opravilo že prilastilo napravo, ali pa da si naprave še ni prilastilo nobeno opravilo. Podobno definiramo za ECB dve stanji nekega dogodka, ki "se je zgodil" ali pa "se ni zgodil", kar po vrsti pomeni, da je neko opravilo signaliziralo dogodek, ali pa sa ni signaliziralo nobeno opravilo.

Obravnava QCB

Prvo opravilo, ki si prisvoji prosti QCB, zaznamuje QCB z "zaseden" s funkcijo ENQ (enqueue) in postane njegov ekskluzivni lastnik. V tem stanju ostane vse dokler ne sprosti QCB-ja naprave s funkcijo DEQ (dequeue). Opravila, ki si želijo prisvojiti zasedene QCB, morajo preiti v stanje ČAKAJOČE. Postavijo se v FIFO sklad čakajočih za QCB. Ko prejšnje opravilo sprosti QCB, sa zasede opravilo, ki najdalj že čaka nanj, če pa v skladu ni nobenega čakajočega, postane QCB spet prost.

QCB sestoji iz treh polj. V prvem je informacija o zasedenosti. V drugem je kazalec na sklad čakajočih. Kaže na tisti TCB, ki bo prvi na vrsti. V tretjem je kazalec na TCB trenutno aktivnega opravila, ki ima v lasti QCB. Povezava med QCB in TCB-ji je prikazana na sliki 6.

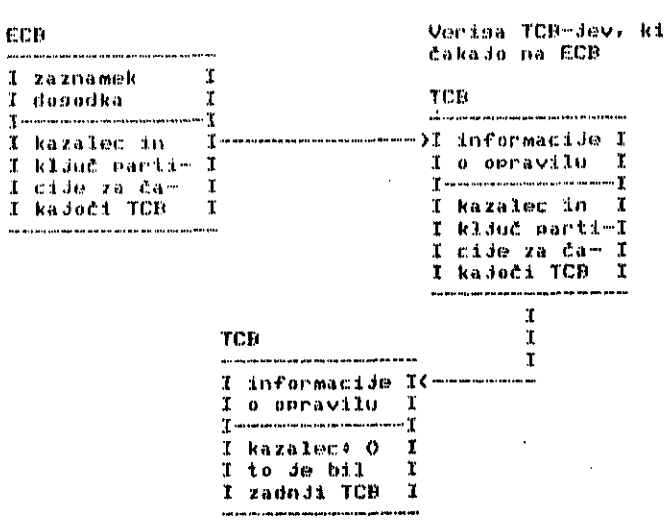


Slika 6.: Povezava med QCB in TCB

Obravnava ECB

Struktura ECB je preprostejša od OCB. Če opravilo čaka na dogodek, katerega ECB še ni zaznamovan, se ustavi, dokler neko opravilo ne zaznamuje ECB-ja. Če hoče opravilo čakati dogodek, ki se je že zgodil, teče opravilo naprej. Število opravil, ki čaka nek dogodek, ni omejeno.

ECB vsebuje dve polji. Prvo polje vsebuje informacijo o dogodku: "se je zgodil" ali "se ni zgodil". V drugem polju se nahaja kazalec in ključ particije za TCB opravila, ki je prvo v seznamu opravil, ki čakajo na dogodek. Preko tega naslova so verižno dostopni naslovi vseh TCB-jev, ki čakajo na dogodek. Vsak TCB vsebuje naprej polje s kazalcem in ključem particije za naslednji TCB. V zadnjem TCB-ju v verigi je namesto kazalca ničla. Shema je na Sliki 7.



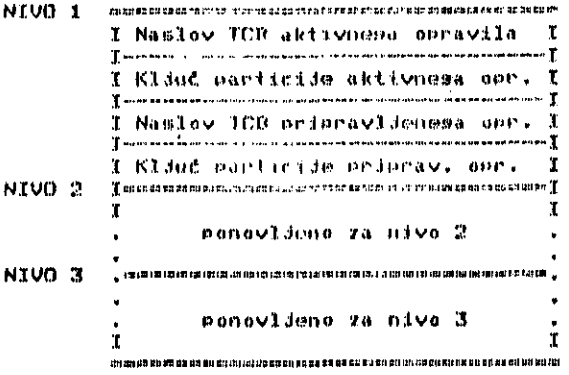
Slika 7.: Povezava med ECB in TCB

6. Nadzorni program

Nadzornik opravil deluje na strojnem nivoju 0, torej ima najvišjo možno prioriteto. Vsebuje tabelo, ki ima vnose opravil, ki so v stanjih PRIPRAVLJENO ali AKTIVNO. Tabela ima za vsak strojni nivo vnos aktivnega opravila in vnos prvega opravila v skladu pripravljenih opravil, razen za nivo 0, na katerem nikoli ne teče kak drug program kot Nadzornik opravil. Vsak vnos vsebuje naslov TCB in ključ particije. Tabela je skicirana na Sliki 8.

Če je v nekem trenutku nivo aktiven, a v njemu ni nobenega aktivnega opravila, potem na nivoju deluje Upravitelj prekinitev. Ko svojo nalogo opravi, vrne nadzor Nadzorniku opravil.

Če je v nivoju aktivno opravilo, a ni nobenega pripravljenega opravila, potem upravitelj prekinitev vrne nadzor neposredno aktivnemu opravilu. Če so v nivoju pripravljena opravila, na nobenega aktivnega, potem je prvo pripravljeno opravilo odpravljeno v stanje AKTIVNO in zapisano v tabelo kot aktivno, drugo pripravljeno



Slika 8: Shema tabele AKTIVNI/PRIPRAVLJENI

opravilo v skladu pa postane prvo. Če sta na nivoju aktivno in vsaj eno pripravljeno opravilo, in če ima aktivno višjo prioriteto od pripravljenega, potem ni potrebna nobena storitev. Če ima pripravljeno opravilo enako ali višjo prioriteto, potem je povzročena prekinitev za preklapljanje opravil.

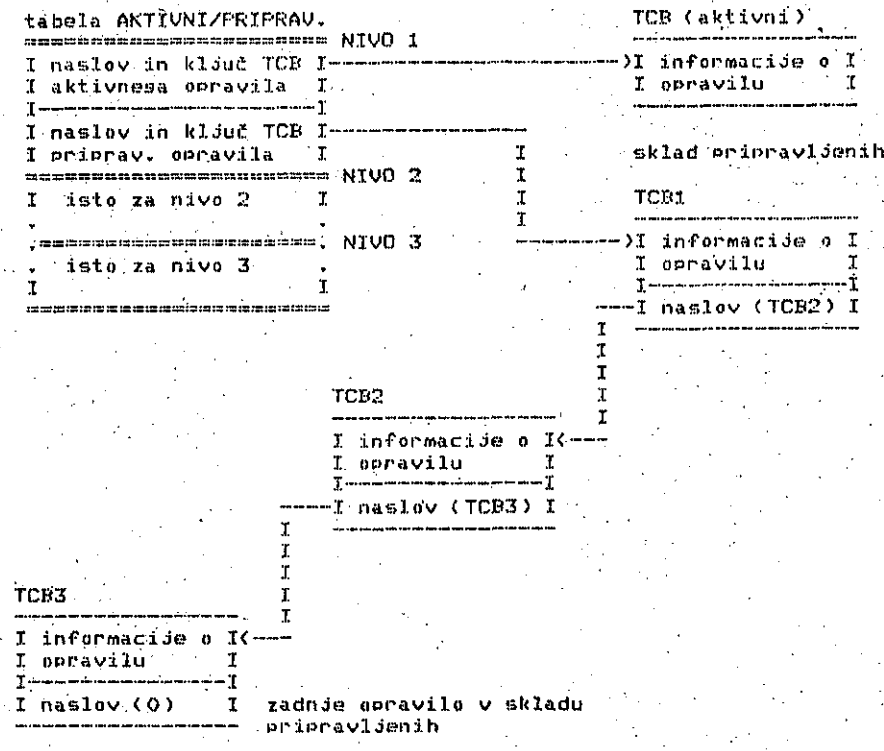
Preklapljanje opravil

Status aktivnega opravila se shrani. Opravilo se prestavi v sklad opravil z isto prioriteto. Nato je odpravljeno opravilo, ki ima najvišjo prioriteto v skladu pripravljenih opravil. Strojna instrukcija SELB (Set Level Block) kopira nivojski statusni blok opravila v strojne registre nivoja. Strojni nivo s pravimi vrednostmi v registrih sedaj čaka pripravljen na kontrolo procesorja, opravilo pa je v stanju AKTIVNO.

Opravila, ki ne morejo biti odpravljena

Preden je opravilo odpravljeno ali nadzor vrnjen prekinjenemu opravilu, Odpravnik opravil preveri, če ni opravilo zaznamovano tako, da ne sme biti odpravljeno. V takem primeru je opravilo izbrisano iz stanja AKTIVNO ali PRIPRAVLJENO. Vsa opravila nekega programa so označena proti odpravljanju, kadar njihovo primarno opravilo izda funkcijo PROGSTOP (konec programa). Enako se zgodi, kadar se je v programu zgodila huda programska napaka ali je bil program nasilno prekinjen s funkcijo za prekinitev izvajanja programa.

Povezava med tabelo AKTIVNI/PRIPRAVLJENI in TCB je prikazana na Sliki 9.



Slika 9.: Povezava med tabelo AKT./PRIP. in TCB skladom

7. Zaključek

V zgoščenih besedah je bila predstavljena večopravilna tehnika v operacijskem sistemu EDX na računalniku Sistem/1. Nadzorni programi operacijskega sistema upravljajo opravila preko tabel stanj in nadzornih blokov, ki vsebujejo informacijo o statusu abstraktno definiranih dogodkov oziroma naprav. Sinhronizacija opravil je izvedena s čakanjem na dogodke ali na dostopnost abstraktnih (ali dejanskih) naprav. Opravila prihajajo na vrsto na podlagi prioritete ter po principu FIFO, kakor vstopajo v verige čakajočih opravil za dostop do procesorja. Nemara velja potegniti še analogijo med večprocesorskimi sistemi in zametkom tega na Sistemu/1 v obliki štirih strojnih nivojev z vsemi registri, ki po prioriteti pridobivajo nadzor procesorja.

UPORABLJENI SIMBOLI

Sistem/1 = računalnik IBM Series/1
V/I = vhodno / izhodni
FIFO = first in first out
TCB = Task Control Block
ECB = Event Control Block
QCB = Resource Control Block
ENQ = Enqueue
DEQ = Dequeue

UPORABLJENI VIRI

1. IBM Series/1 Event Driven Executive - Internal Design; IBM Corp., Information Development, Department 27T, Boca Raton, Florida 33432; 1981

2. IBM Series/1 Event Driven Executive - Language Reference; isti naslov kot pod 1.; 1981

3. IBM Series/1 Event Driven Executive - Study Guide; IBM Corp., General Systems Division,Technical Publications, Department 796, Atlanta, Georgia 30055; 1980

4. IBM Series/1 Hardware Study Guide; isti naslov kot pod 3.; 1979

INFORMATICA 2/1984

CENTRALNA PROCESNA ENOTA DELTA 16/BIT-SLICE

FRANC KOGOVŠEK
JOŽE MATJAŽ,
ANDREJ TREBAR,
JANEZ ANDERLE,
ANDREJ DOBNIKAR,
VESELKO GUŠTIN,
FRANC KODER,
MIRA MAČEK,
ALOJZ VOGEL

UDK: 681.519.7

ISKRA DELTA LJUBLJANA;
FAKULTETA ZA ELEKTROTEHNIKO, UNIVERZA EDVARDA KARDELJA, LJUBLJANA

Prispevek govori o centralno procesni enoti DELTA 16/BIT-SLICE, ki je nastala kot rezultat sodelovanja med Iskro-Delto in Fakulteto za elektrotehniko v Ljubljani. To je mikroprogramiran rezinski procesor, ki emulira procesor sistema PDP 11/34. Poudarjene so le specifične lastnosti emulatorja: arhitektura, možnosti razširitev nabora instrukcij in način, kako izkoristiti fleksibilnost arhitekture za dosego bistveno večje zmogljivosti centralne procesne enote.

This article describes a central processor unit DELTA 16/BIT-SLICE, which has been developed by Iskra-Delta in cooperation with Fakulteta za elektrotehniko in Ljubljana. This microprogramed processor was designed by using AMD 2900 series components to emulate the PDP 11/34 CPU. Only the specific features are emphasized: architecture, the possibilities to expand the instruction set and the description is given, on how to use all the advantages of the flexible architecture to achive better performances of the central processor unit.

UVOD

DELTA 16/BIT-SLICE je centralna procesna enota za sistem DELTA 340. To je emulacija CPE KD 11-EA, kar pomeni, da sta enoti neposredno zamenljivi na standardnih konektorjih v sistemskem vodilu.
Arhitektura enote DELTA 16/BIT-SLICE je zasnovana tako, da omogoča pregledno in hitro tvorbo mikroprogramov za izvajanje poljubnih makro instrukcij. S tem namenom je v enoti DELTA 16/BIT SLICE zajeto vezje za izvedbo suspenzije, kar pomeni, da se mikroprogram lahko prekine in kasneje regularno nadaljuje (obravnava zahteve za prekinitev med izvajanjem instrukcije, izvedba programa ob izklopu oziroma vklopu električne napetosti in podobno). Zajeto je tudi vezje za adresiranje dodatnih 48 splošno uporabnih registrov, v instrukcijskem dekoderju in v vezju za obravnavo pasti pa je rezervirano področje za operacijske kode teh nestandardnih instrukcij. Primeri takih instrukcij so iz niza 'Commercial Instruction Set'. To je niz instrukcij, ki tečejo na sistemih PDP 11/44 ob uporabi dodatnega CIS procesorja.
Programsko in aparaturno podporo za tvorbo mikroprogramov sestavljajo:

- Macro Meta Assembler (AMDASM) z datoteko DEFENITION FILE enote DELTA 16/BIT-SLICE
- Program za vpis v mikroprogramski pomnilnik
- Vpisljiv mikroprogramski pomnilnik (WCS) s širino do 128 bitov in globino do 2K besed

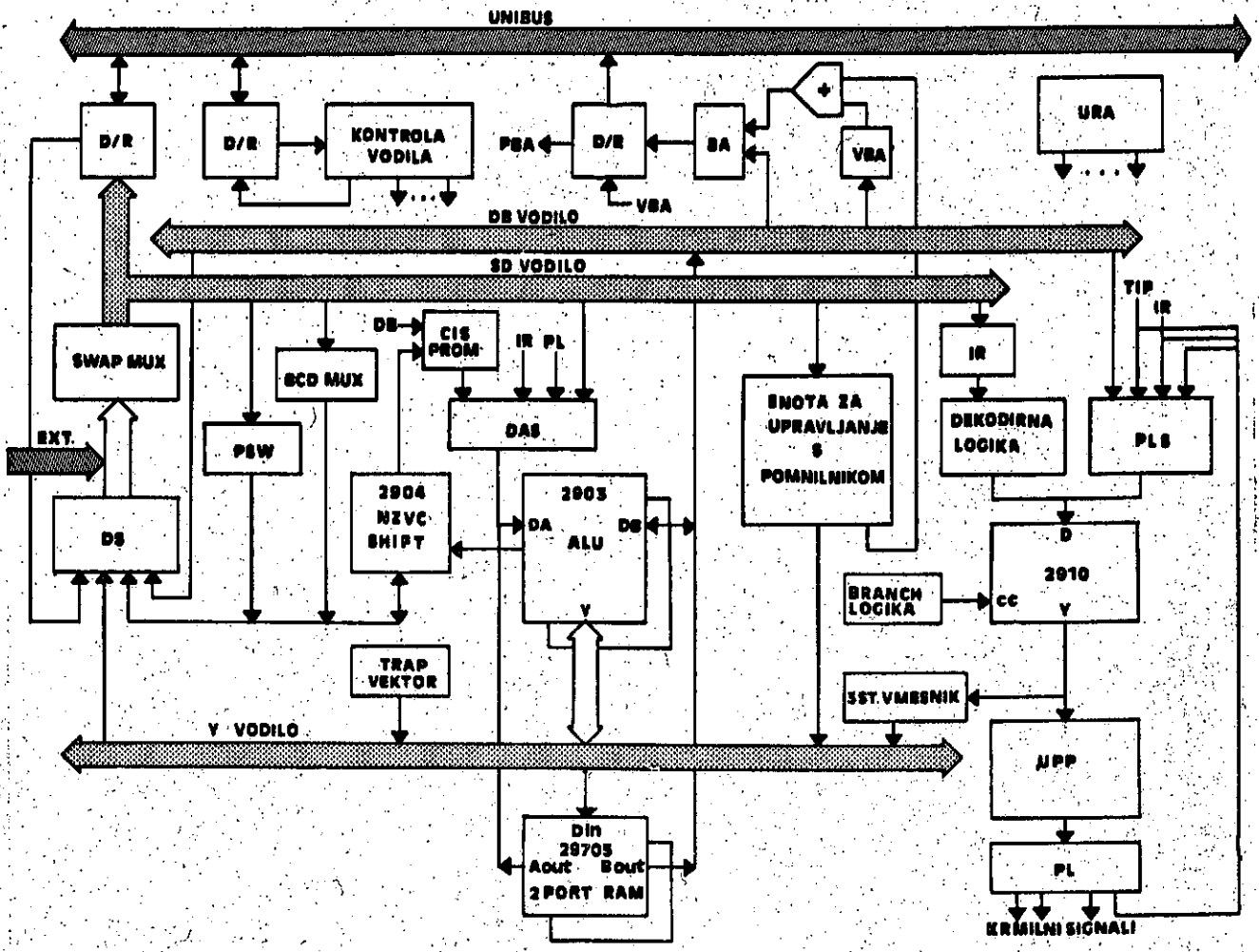
in je na voljo potencialnim uporabnikom v prostorih ISKRA DELTA, Razvoj AO, Grubarjeva nabrežje 6.

OPIS ARHITEKTURE PROCESORJA DELTA 16/BIT-SLICE

Slika 1 prikazuje blok shemo DELTA 16/BIT-SLICE mikroprogramiranega procesorja, ki ga po funkcijskih značilnostih razdelimo na štiri področja:

- podatkovna vodila (DATA PATH)
- mikroprogramski krmilnik (MICROPROGRAM CONTROL)
- enota za upravljanje s pomnilniškim prostorom (MEMORY MANAGEMENT)
- vezje za kontrolo podatkovnih prenosov (BUS CONTROL)

Podatkovni del procesorja je zasnovan na uporabi rezinskih elementov družine AM2900. Aritmetična logična enota (ALU) je sestavljena iz štirih 4-bitnih rezin AM2903, ki jim je dodan carry generator AM2902. Rezina AM2903 omogoča aritmetične in logične operacije ter specialne funkcije za množenje in deljenje. Skupaj z AM2904 omogočajo rezine pomikanje vsebine levo in desno nad zlogom, besedo in dvojno dolžino besede. Rezina AM2903 vsebuje poleg Q registra še 16 registrov, od katerih jih je osem uporabljenih kot splošno namenski registri (dostopni programerju), drugih osem pa kot delovni registri za izvorne in ponorne operande, kot tudi za vmesne rezultate.
ALU-ju je lahko dodan še dodatni niz 48 delovnih registrov, ki so uporabljeni na enak način kot notranji ALU registri.
V rezinah ni bilo mogoče realizirati procesorjeve statusne besede (PSW) in instrukcijskega registra (IR). Sestavni del PSW-ja je tudi AM2904, kjer so shranjene pogojne kode za vsako operacijo v ALU-ju.



Slika 1: Blok shema DELTA 16/ BIT-SLICE mikroprogramiranega procesorja

Rezina AM2903 ima več vhodnih in izhodnih linij, ki so lahko različno uporabljene. Linije DA<15:0> so uporabljene kot podatkovni vhod, na katerega so pripeljani podatki iz SD<15:0>, takojšnji podatek iz pipeline registra (PL), odmik iz instrukcijskega registra (IR), podatek iz CIS PROM-a in podatek iz 2-vhodnega dodatnega RAM-a. Linije DB<15:0> so uporabljene kot podatkovni izhod, ki posreduje virtualno adreso enoti za upravljanje s pomnilnikom, podatek za shranjevanje v pomnilnik preko Unibus vodila in števec iteracij mikroprogramski kontroli. Kot podatkovni vhod so DB linije uporabljene v primeru, ko želimo v ALU dostaviti podatke iz dodatnega RAM-a. Linije Y<15:0> so uporabljene kot podatkovni vhod ali izhod. V primeru, da so uporabljene kot podatkovni vhod, so rezini posredovani podatki iz SD<15:0> vodila, trap vektor adresa, mikroinstrukcijska adresa iz AM2910 in podatek iz enote za upravljanje s pomnilnikom. Če so uporabljene kot podatkovni izhod, je podatek lahko posredovan v dodatni 2-vhodni RAM, ali pa na Unibus vodilo v primeru shranjevanja v pomnilnik.

DS MUX skrbi za izbiro podatkov, ki jih pošilja CPE na Unibus vodilo oziroma podatke z Unibus vodila pošlje na SD<15:0>. Na Unibus vodilo so lahko poslani naslednji podatki: PSW in podatki iz ALU-ja preko DB<15:0> ali Y<15:0>.

V primeru zrcalnih operacij se v odvisnosti od adrese lahko zamenjata spodnji in zgornji zlog, za kar poskrbi SWAP MUX. Podatki na izhodu SWAP MUX-a so potem prisotni na SD<15:0> in na Unibus vodilu, če je ustrezno zakrmiljen D/R element.

DEKODIRANJE INSTRUKCIJ

Naloga DEKODIRNE LOGIKE je, da na osnovi vsebine v instrukcijskem registru po dostavi vsake instrukcije določi mikroprogramsko adreso, na kateri se prične izvajati instrukciji ustrezen mikroprogram.

Glede na to, da ima precej instrukcij skupne značilnosti, predvsem kar se tiče dostavljanja operandov oz. njihovih adres, smo celoten nabor instrukcij razdelili v 9 grup (8 iz standardnega nabora, eno grupo pa predstavljajo CIS instrukcije).

Zaradi grupiranja instrukcij je naloga DEKODIRNE LOGIKE dvojna: prvič, da za vsako instrukcijo določi pripadnost grupi, ter, da hkrati pripravi adreso za eksekucijski del mikroprograma, ki je seveda različen za vsako instrukcijo. Vsaki grupi ustreza mikroprogram, ki za vse instrukcije v grupi izvaja skupno dostavo operandov oz. adres operandov. V času izvajanja skupnega mikroprograma, drugi del DEKODIRNE LOGIKE pripravi ekse-

kucijsko adreso, ki ustreza instrukciji v instrukcijskem registru.
Krmiljenje vstopne adrese v sekvenčnik 2910 je pod kontrolo mikroprograma. Ob dostavi nove instrukcije je odprta pot, ki nosi adreso grupe, ob koncu skupnega dela mikroprograma, ki ustreza grupi, pa se odpre pot, ki pripelje eksekucijsko adreso instrukcije v sekvenčnik.
DEKODIRNA LOGIKA je zasnovana tako, da omogoča spremembe enostavno z reprogramiranjem njenih pomnilnih elementov PAL oz. PROM. To je pomembno z vidika instaliranja novih specijalnih instrukcij.

MIKROPROGRAMSKI KRMILNIK

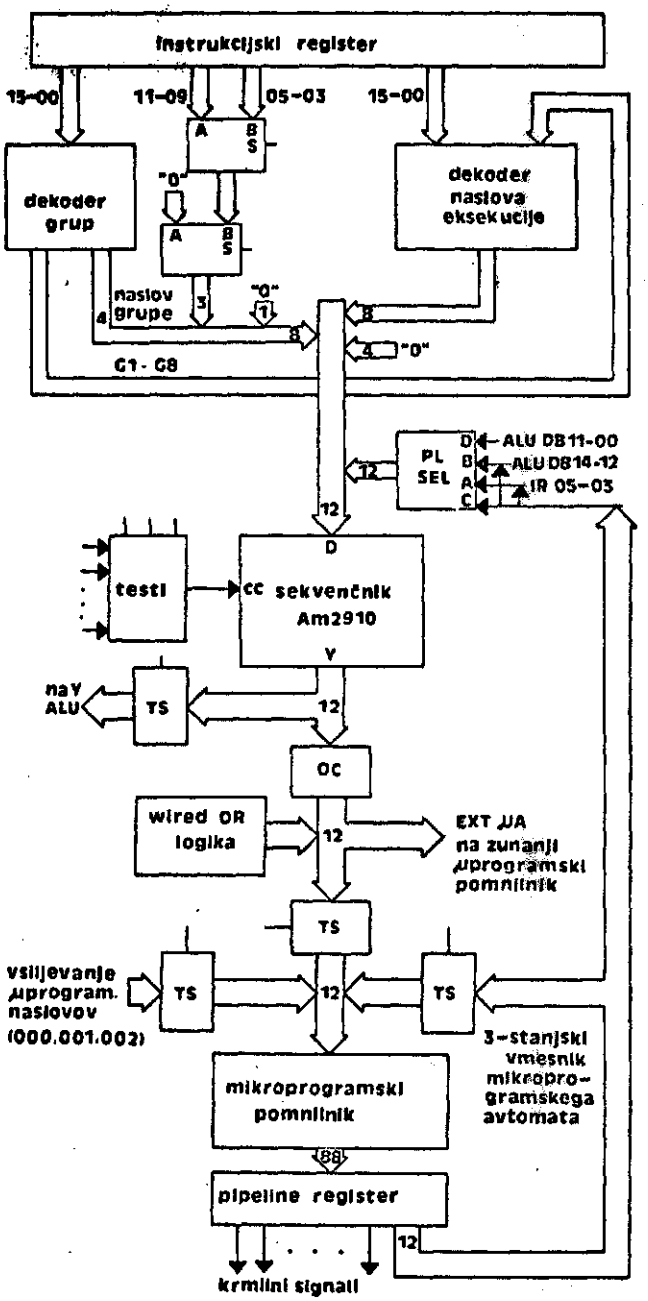
Naloga krmilnika (AM2910) je generiranje naslova za naslednjo mikroinstrukcijo. Mikroprogramski naslov se gleda na krmilno informacijo na I vhodih 2910 izbira med mikroprogramskim števnikom, vsebino mikrosklada, vsebino ponavljalnega registra ali vsebino na D vhodu. Na D vhod so priključeni trije viri. Hkrati je aktiven le eden, ostala dva pa sta v visokoimpedančnem stanju. Mikroprogramski naslov, ki ga je generirala enota 2910, je možno spremeniti v skupni točki vezij z odprtim kolektorjem, od katerih se preko 3-stanjskega vmesnika posreduje na naslovne linije mikroprogramskega pomnilnika (MPP). Obstaja tudi mehanizem naslavljanja mikroprogramskega pomnilnika na ta način, da se naslov vsili neposredno na vhod MPP, brez sodelovanja sekvenčnika. Mehanizem je aktiviran v naslednjih primerih:

- pri vklopu napetosti se aktivira signal MPC 00 L, kar ima za posledico vsilitev naslova 001 na MPP (rutina za POWER UP)
- ob pojavu CIS suspenzije se vsili naslov 002 (rutina za začetek suspendirane CIS instrukcije)
- kadar je pri prenosih CPE-glavni pomnilnik nastopila situacija, ki zahteva past, signal ABORT H prekine izvajanje mikroprograma in vsili naslov 000 in s tem mikroinstrukcijo SERVICE, ki reagira na past.

Funkcija PL-selektorja je izbira enega izmed štirih virov. Izhod je priključen neposredno na D vhod 2910 in je onemogočen le v tistih mikroinstrukcijah, pri katerih se izvaja JMAP instrukcija. Selekt vhodi PL - selektorja so krmiljeni neposredno iz pipeline registra s signaloma PLS0 in PLS1, pri čemer velja naslednja tabela:

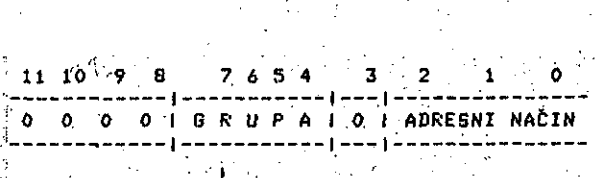
PLS1	PLS0	
0	0	PL + MODE
0	1	PL + TIP
1	0	PL
1	1	ALU NN

PL + MODE vhod uporabljamo v primerih, ko želimo večvejni skok glede na način naslavljanja, ki je z instrukcijo vred dan v IR registru.
Za CIS nabor je rezerviran večvejni skok glede na tip stringa, ki sa definirajo biti 12,13 in 14 v deskriptorju. Le-ta se nahaja v registrih ALU-ja in kadar želimo večvejni skok glede na tip stringa, posredujemo pripadajoč deskriptor na DB izhod ALU enote. Signale ALUDB12H, ALUDB13H in ALUDB14H, ki definirajo vrsto stringa, uporabimo na PL selektorju, tako da s svojo vrednostjo določajo enega izmed osmih naslovov v večvejnem skoku.



Slika 2: Mikroprogramski krmilnik

Za brezrodajne in posojne vejitve se celoten 12 bitni naslov iz pipeline registra posreduje preko PL selektorja (uporabljen je vhod PL) na D vhod 2910.
ALU NN vhod se v standardnem naboru uporablja za inicializacijo vsebine števca iteracij v 2910. Nova vsebina, ki se nahaja v ALU registru, se preko DB izhoda posreduje na PL selektor in preko tega na D vhod sekvenčnika ter se z instrukcijo LDCT vpiše v števec iteracij. V okviru CIS suspenzije je možno preko tega vhoda inicializirati tudi mikrosklad in mikroprogramski števnik.
Aktiviranje dekodirne logike na vhodu sekvenčnika omogoča JMAP instrukcija takoj za tem, ko se je nova makroinstrukcija že vpisala v IR register. Naslov, ki ga definira dekodirna logika je sestavljen na sledeč način:



Mikroprogramski sekvenčnik opravlja pogojno vejitev v primeru, da se na I vhodih nahaja CJP instrukcija ter, da je signal CCEN (1)L/PL/ na logični "0". Vejitveni naslov v tem primeru posreduje PL selektor, vejitev pa dobimo v primeru, ko se na CC vhodu 2910 pojavi logična "0".
Skupna točka vezij z odprtim kolektorjem omogoča wired-or funkcijo in na ta način daje možnost za spreminjanje mikroprogramskega naslova, kar predstavlja dodatni vejitveni mehanizem za realizacijo aparaturnih pogojnih vejitev. Ta mehanizem se aktivira s signalom BRANCH (1)H/PL/ v primeru, da se ne izvaja JMAP instrukcija na 2910, ampak sekvenčnik generira sodi vejitveni naslov z brezpogojno vejitveno funkcijo JMPBR. Kadar je pogoj izpolnjen se na ta način sodi naslov spremeni v lihega in s tem se izvede aparaturna pogojna vejitev.
Mehanizem aparaturne pogojne vejitve je uporabljen tudi pri aparaturno-mikroprogramski realizaciji vejitvenih makroinstrukcij.
Ob detekciji suspenzije se vsili koda CJS na AM2910 ter "002" na mikroprogramski pomnilnik. Generirani signal D SUS (1) L istočasno zapre pot od kontrolerja na mikroprogramski pomnilnik, ter omogoči izhodom iz PL registra "Next Address", da jih usmerimo na adresne vhode mikroprogramskega pomnilnika. Taka povezava ni nič drugega kot mikroprogramski avtomat in ta prevzame kontrolo nad shranjevanjem mikroprogramskega števca na mikrosklad v AM2910.
Enostavnejša varianta suspenzije nastopi v primeru I/O interrupt-a, zahtevnejša pa ob PFAIL interrupt-u. V prvem slučaju je potrebno shraniti le vsebino COUNT REG iz 2910 v ALU in uPC na mikroprogramski sklad (uSTACK) v 2910. V drugem slučaju je poleg tega potrebno shraniti še celoten uSTACK iz 2910 v ALU ter nato še vse registre iz ALU-ja (64) na sklad (SP). Slednje se izvede ob zadnji instrukciji rutine za izpad napetosti (PFAIL), to je v HALT instrukciji, ki je v ta namen prirejena.

RAZŠIRITEV NABORA INSTRUKCIJ

Že tako bežen pregled arhitekture kaže na izredno fleksibilnost CPE DELTA 16/ BIT-SLICE. Kot ilustracija naj služi primer nabora CIS (Commercial Instruction set). To je povsem nestandarden nabor, ki se normalno izvaja na posebnem koprocesorju. Operandi v tem naboru so stringi, bodisi znakovni bodisi numerični. Operacije pa so od premikanj stringov, iskanj vzorcev do aritmetičnih operacij nad numeričnimi stringi dolžine do 31 številk. Vsi deli aparaturne opreme, ki so bili vnešeni za podporo CIS nabora, se lahko s podobno učinkovitostjo koristijo pri realizaciji poljubnih novih instrukcij. Pri tem ni potrebno spreminjati niti instrukcijskega dekoderja pod pogojem, da si za svojo makroinstrukcijo "izposodimo" operacijsko kodo ene od CIS instrukcij. Spremeni se le vsebina v mikroprogramskem pomnilniku.
Mikroprogram se razvija s pomočjo MACRO META ASSEMBLER-ja. Za osnovo se vzame datoteko DEFINITION FILE, ki vsebuje določene mnemonike, ki so bili prirejeni posameznim funkcijskim sklopom aparaturne opreme procesorja DELTA 16/ BIT-SLICE. Asembliran mikroprogram se nato avtomatsko naloži v razvojni WCS (Writable Control Store), ki v času testiranja simulira mikroprogramski pomnilnik. Po končanem testiranju se mikrokoda vpiše v hitre bipolarne PROM elemente. V konfiguraciji torej, kakršna je trenutno, je možno uporabiti prostor v mikroprogramskem pomnilniku in sicer 0,25K oziroma 0,75K, odvisno pač od tega ali se izbere PROMe 512X8 ali pa 1024X8. Mikroprogram za osnovni instrukcijski nabor zaseda približno 0.25K pomnilniških lokacij.

RAZŠIRITEV V APARATURNI OPREMI

Povsem nove kvalitete bi pridobili s priključitvijo tretje plošče, ki bo vsebovala: 4 K besed mikroprogramskega pomnilnika na osnovi hitrih vezij RAM (čas dostopa 55ns), hitri paralelni množilnik 16X16 (izračun produkta 50ns) in niz dodatnih registrov. S takšno konfiguracijo se približamo zmogljivostim ARRAY procesorjev na področju hitrega procesiranja digitalnih signalov (digitalni filtri, spektralna analiza, matematične transformacije, manipulacije z matrikami, integrali, procesiranje slik, razpoznavanje vzorcev in tako dalje). To so področja, kjer je računalnik izkoriščen v največji možni meri. Zmogljivosti sistema na osnovi procesne enote DELTA 16/BIT-SLICE v taki konfiguraciji in na teh področjih so lahko nekajkrat večje od zmogljivosti večjih sistemov, ki opravljajo ista dela na osnovi standardnih instrucij.

ZAKLJUČEK

Z vidika arhitekture ima CPE DELTA 16/ BIT-SLICE nekaj značilnih prednosti napram enoti KD11EA sistema PDP 11/34:
- uporabniku je omogočena realizacija novih instukcij oziroma algoritmov na mikroprogramski način
- rezinska zasnova ALU enote omogoča modularno razširitev registerskega niza
- mikroprogramski krmilnik omogoča vpeljavo mikroprogramskih subrutin
- na mikronivoju je vgrajen prekinitveni mehanizem (suspenzija), ki dovoljuje regularno prekinitev daljših mikroprogramskih sekvenc
- dosežena je optimalna dolžina vsake mikroinstrukcije
- aparaturna zasnova upošteva vse posebnosti CIS nabora

Zaradi vseh navedenih prednosti se je bistveno povečala fleksibilnost in uporabnost procesne enote. Glavna odlika rezinskega procesorja je njegova hitrost pri izvajanju obsežnejših algoritmov, na kar že kažejo nekatere primerjave.
Natančne meritve hitrosti še niso bile izvedene. Rezultati teh meritev in primerjava z enoto KD11EA bodo objavljeni v eni prihodnjih številk.

INFORMATICA 2/1984

UPORABA PROGRAMSKIH GRAFOV PRI UGOTAVLJANJU VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH I.

B. DŽONOVA–JERMAN,
J. ŽEROVNIK

UDK: 519.698

INSTITUT JOŽEF STEFAN, LJUBLJANA

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

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

1. UVOD

Sodobne smeri razvoja računalniške arhitekture imajo za cilj: povečati računalniško moč sodobnih računalnikov, kar pomeni povečati njihovo zmogljivost in hitrost računanja ter predlagati rešitve, ki odpravljajo ozka grla v procesu računanja in so rezultat klasične (Von Neumannove) organiziranosti oziroma klasične zgradbe obstoječih računalniških sistemov.

Večjo računalniško zmogljivost potrebujemo zaradi naraslih potreb po reševanju problemov s področja umetne inteligence, obdelave slik, razpoznavanja govora, napovedovanja vremenskih situacij, avtomatskega prevajanja jezikov, analize seizmografskih podatkov, obdelavah multidimenzionalnih sistemov in podobno. Ocenjeno je (1), da napoved 24 urne vremenske situacije zahteva 100 milijard operacij na sek. Obdelava slike, ki ima 512 x 512 slikovnih elementov in potrebuje do 100 ukazov za slikovni element, zahteva 656 milijonov operacij na sec. (1).

Obdelava in reševanje takšnih in podobnih problemov z zadovoljivo hitrostjo je mogoče le ob uporabi računalnikov, ki omogočajo vzporedno izvajanje računalniških ukazov. Dosedanje izkušnje so pokazale, da so možnosti vzporednega izvajanja računalniških ukazov na računalnikih z Von Neumannovo arhitekturo dokaj izčrpane. Von Neumannov model računalnika ima dve bistveni značilnosti: spomin za hranjenje programov in podatkov in števec ukazov, čigar vsebina določa ukaz, ki se bo izvajal. Von Neumannov model računalnika je imel velik vpliv na razvoj programskih jezikov. Jeziki pri katerih je vpliv te arhitekture najbolj opazen tvorijo razred Von Neumannovih jezikov. V ta razred programskih jezikov so uvrščeni FORTRAN, ALGOL 68, BASIC ipd. Med sodobnimi programskimi jeziki obstajajo jeziki, ki so bolj funkcionalno zastavljeni in ki so visoko uvrščeni v hierarhijski lestvici programskih jezikov. V programih, pisanih v teh jezikih, je zaradi dovoljenih sintaksnih konstruktov jezika značilna prisotnost notranje oziroma implicitne vzporednosti (inherent parallelism).

Namen našega prispevka je predstaviti dve metodi za ugotavljanje vzporednosti v obstoječih računalniških programih. Obe metodi uporabljata tehniko predstavitve programov s programskimi grafi. Programski grafi so model s katerim predstavljamo potek obdelav v računalnikih krmiljenih s pretokom podatkov (data flow computer). Pri analizi s pomočjo programskih grafov uporabljamo dva semantična modela arhitekture računalnika krmiljenega s pretokom podatkov. Prvi model je model arhitekture s povratno zanko krmilnih signalov, ki opravlja operacije le nad elementarnimi tipi podatkov in individualnih strukturnih komponent. Drugi model se razlikuje od prvega v tem, da so dodane operacije z vektorji.

2. OSNOVNE LASTNOSTI RAČUNALNIKOV KRMILJENIH S TOKOM PODATKOV

Računalniki krmiljeni s tokom podatkov predstavljajo temeljito spremembo v arhitekturi glede na Von Neumannovo arhitekturo računalnika. Pri Von Neumannovem računalniku potek obdelave poteka po natančno določenem zaporedju ukazov, ki tvorijo program. Spomin je enodimenzionalen in naslavljanje je zaporedno. Med ukazi in podatki ni bistvenih razlik. Semantika podatkov tudi nima nobenega pomena. Krmilni tok je opredeljen s števcem ukazov, ki skrbi za tekočo oskrbo procesne enote z ukazi. Povezava med spominom in procesno enoto po kateri potujejo ukazi je v obliki zaporedja je največja omejitev Von Neumannovega modela. Uvajanje novih organizacijskih rešitev v računalniški arhitekturi so pripeljale do izredno velike zmogljivosti sodobnih računalniških sistemov. Med te rešitve, ki bistveno niso spremenile osnove Von Neumannovega modela računanja so se zlasti pokazale za uspešne sledeče:

- cevasta organizacija aritmetično - logičnih enot
- sistemi z več aritmetično - logičnimi enotami
- matrični procesorji,
- sistemi z elementi za povezovanje večkratnih funkcionalnih enot, kot so vodila za porazdeljeno obdelavo, mreže s strukturo obroča ipd.

- priročni spomini (caché memory)
- navidezni spomin (virtual memory).

Izjemna zmogljivost nekaterih sodobnih računalnikov kot je na primer IBM 360/91, CDC 6600, CRAY-1 sloni na vzporednem delovanju različnih funkcionalnih enot (procesor-spomin, spomin-spomin, procesor-vhodno/izhodne enote, procesor-procesor z vodenjem tokov podatkov). Procesne enote teh računalnikov imajo več izvajalnih enot in mehanizmov za dekodiranje in sprejemanje ukazov, ki omogočajo neprekinjen tok ukazov oziroma zmeraj poln vod skozi katerega ukazi pojejo. Pri IBM 360/91 se ukaz s plavajočo vejico najprej delno dekodira in šele zatem pošlje enoti s plavajočo vejico. Enota dokonča dekodiranje in pošlje ukaz izvajalni enoti. Izvajanje aritmetičnih operacij se prične šele takrat ko pridejo vsi potrebni operandi (iz spomina ali iz drugih enot). Dekodiranje ukazov poteka zaporedno ne glede na to, da je izvajanje nekaterih možno brez upoštevanja tega zaporedja. Zaporedno dekodiranje ukazov zagotavlja logično pravilnost rezultatov. Računalnik CRAY-1 (6) vzporedno izvaja ukaze ob uporabi matrično organiziranih izvajalnih enot. Zmogljivost računalnika pri izvajanju vektorskih ukazov je omejena le s kapaciteto spomina. Ocenjeno je (2), da se odstotek programske opreme, ki se lahko napiše v skladu z možnostmi, ki jih nudijo matrično organizirane procesne enote giblje od 1 do 90 % na področju znanstvenih aplikacij. Ostali del programske opreme pa lahko postane ozko grlo v kolikor učinkovitost vektorskih ukazov izjemno zraste. Programi pisani v Fortranu težko izrabljajo zmogljivost računalnikov kot je CRAY-1. Pisanje programov zahteva zelo dognano analizo podatkovnih tokov. Analiza podatkovnih tokov v programih pisanih v Fortranu je izjemno težka zaradi stranskih učinkov, ki jih povzročajo GO TO stavki in povezovanje spremenljivk. Razširitve v Fortranu lahko zmanjšajo te probleme, pri tem pa programiranje postane zahtevno, programi pa neprenosljivi in preveč odvisni od stroja. Uporaba več Von Neumannovih procesorjev, ki si delijo skupni spomin je podoben pristop. Poglavitni problem pri takšni računalniški arhitekturi je zagotovitev podpore v materialni opremi, ki bo skrbela za pravilnost rezultatov. Namreč, kosi programa, ki se izvajajo vzporedno in si delijo podatke, se sinhronizirajo s testnimi procedurami, (test and end), s semaforji, sporočili ipd. Nadzor sinhronizacijskih operacij je zahteven in zadostno zmogljivost pri obdelavi dosežemo le, če je izvajanje razdeljeno na večje kose programa, ki so sinhronizirani z zelo majhnim številom operacij. Razdelitev programa v takšnih kosih, ki bodo zagotavljali tudi pravilnost rezultatov obdelave, je zelo težka naloga.

Vsa dosedanja prizadevanja za izboljšanje zmogljivosti sodobnih računalnikov zgrajenih na osnovi Von Neumannovega modela računanja so pokazala, da so možnosti precej izrabljene. Klasična organizacija računalnikov ne omogoča zadostno izkoriščanje možnosti, ki jih ponuja nova tehnologija integriranih vezij (LSI, VLSI). Izhod iz nastale situacije išče strokovna javnost v različnih smereh razvoja računalniške tehnologije med katerimi imajo najbolj pomembno mesto zaradi svojega revolucionarnega koncepta računalniki krmiljeni s pretokom podatkov.

Arhitektura računalnika krmiljenega s pretokom podatkov ponuja rešitve za večino pomanjkljivosti klasično organiziranih računalnikov. Struktura teh računalnikov je zasnovana na podlagi naslednjih konceptov:

- ni spremenljivk in obdelava poteka le s konkretnimi vrednostmi,
- potek obdelave je krmiljen s pretokom podatkov in števca ukazov,
- ni shranjevanja podatkov v klasičnem smislu oziroma ni spomina z naslovi.

Programe krmiljene s pretokom podatkov opisujemo z usmerjenimi grafi. Elementi grafa ponazarjajo pretok podatkov med posameznimi ukazi. Vozlišča grafa ponazarjajo ukaze, usmerjene povezave med vozlišči pa predstavljajo vhod in izhod podatkov pred in po izvajanju ukaza. Usmerjene povezave so nosilke dejanskih vrednosti podatkov na vhodu in na izhodu iz vozlišča. Gibanje podatkov skozi usmerjeni graf predstavlja izvajanje programa s katerim smo ponazorili določeni algoritem.

Ukazi nimajo stranskih učinkov na podatke in jezik zgrajen na podlagi koncepta krmiljenja s pretokom podatkov vsebuje omejitve, glede zaporedja izvajanja ukazov, ki izhajajo naravnost iz odvisnosti podatkov v zapisanem algoritmu. Vsak program napisan v visokonivojskem jeziku s konceptom krmiljenja s pretokom podatkov lahko zapišemo v obliko grafa. Graf je podoben grafom, ki jih dobimo iz prevajalnika za Fortran po analizi podatkovnega toka. Prednost jezika s konceptom krmiljenja s pretokom podatkov je v tem, da graf o pretoku podatkov dobimo zelo enostavno in hitro. Procesne enote krmiljene s pretokom podatkov so dejansko računalniki v katerih je shranjen graf, ki ponazarja pretok podatkov v programu. Procesna enota razpozna kateri ukazi so pripravljeni za izvajanje in te ukaze izvajalna enota obdela takoj ko dobi sporočilo o tem, da so določeni viri razpoložljivi. V primerih, ko je razpoložljivih računalniških virov zadosti, računalnik izrabi vse možnosti vzporedne obdelave v programu.

Za implementacijo računalnikov krmiljenih s pretokom podatkov je bilo predlaganih več različnih inačic materialne opreme. Vsi predlogi uporabljajo kot osnovo model usmerjenega grafa za predstavljanje računalniških programov. Zato je to vrsta računalnika, ki ima jezikovno zasnovan koncept arhitekture in kot programski jezik programske grafe. Razlike med posameznimi arhitekturami računalnikov obstajajo v implementaciji konceptov programskih grafov na strojnem nivoju.

3. MODEL RAČUNALNIKA KRMILJENEGA S PRETOKOM PODATKOV

V metodah za merjenje vzporednosti v računalniških programih, ki jih bomo predstavili v nadaljevanju, je uporabljen model, ki sta ga predložila Martin in Estrin (4,5). V tem modelu ima vsako vozlišče usmerjenega grafa sledečo strukturo:

- polje ukazne kode, ki opredeljuje ukaz,
- dvoje ali več polj za sprejem operandov,
- eno ali več polj, ki opredeljujejo kam rezultati potujejo po izvršitvi ukaza.

Ukaz v vozlišču se izvrši šele takrat, ko so vsi potrebni podatki prisotni na vhodu v vozlišče. V času izvajanja vzame operator podatke iz vhoda, jih obdela ter jih postavi na izhod iz vozlišča. Za ta model obdelave je značilno, da se vozliščem direktno posredujejo parcialni rezultati obdelave. Izvajanje ukaza lahko vpliva le na ukaze, ki mu sledijo. Vsi ukazi imajo pomen funkcij. Po uporabi podatka se njegova vrednost izgubi ker ta podatek ni več na razpolago. Koncepta delitve podatkov v spominu v tem modelu ni. Tudi krmilnega toka, ki šteje in nadzoruje izvajanje ukazov ni; ukazi se izvajajo takrat ko so na vhodih v vozlišča prisotni vsi zahtevani podatki. To pomeni, da se ukazi izvajajo neodvisno in da obdelava poteka vzporedno. Seveda je pri tem nujno, da je na razpolago zadostno število virov, drugače vzporedna obdelava ne bo mogoča. Krmilni tok poteka obdelave je opredeljen s tokom podatkov, oziroma z odvisnostmi, ki obstajajo med podatki. Izvajanje programa se lahko konča asinhrono in porazdeljeno. Prednosti tega pristopa lahko strnemo v naslednjem:

- mogoča je vzporedna obdelava
- verifikacija programov je enostavnejša in lažja,
- mogoča je večja modularnost programov,
- širjenje strojne opreme je relativno enostavno
- problemi v zvezi z zaščito podatkov so manjši in
- lažji je nadzor nad napakami v programu.

Poglavitna razlika med klasičnim načinom izvajanja obdelav in med tem novim pristopom je v organizaciji poteka obdelav: Pri sistemih, ki delujejo po principu pretoka podatkov, poteka obdelava v treh fazah: izbor ukaza, pregled pogojev potrebnih za izvajanje ukaza (prisotnost operandov) in samo izvajanje ukaza. V prvi fazi poteka izbor ukaza po nekem v naprej opredeljenem pravilu. Sam izbor ukaza še ne pomeni, da bo prišlo takoj tudi do izvajanja ukaza, saj se šele v drugi fazi odloča o tem ali bo ukaz sprožen ali ne. V kolikor na vhodu v določeno vozlišče ni potrebnih operandov, se ukaz ne izvede. Pri klasični obdelavi je izbor ukaza izvršen na podlagi števca programskih ukazov. Izbran ukaz se avtomatsko tudi izvede, zato faze v kateri se operandi pregledujejo tukaj ni. Prednost obdelave, ki jo nudi koncept pretoka podatkov, je v možnosti visoke stopnje vzporednosti obdelave. Pomanjkljivost je v tem, da lahko prihaja do pogostega čakanja na vhodu v vozlišče zaradi neprisotnosti potrebnih argumentov, če ni dobro izbran algoritem glede na arhitekturo računalnika.

4. MODEL RAČUNALNIKA KRMILJENEGA S PRETOKOM PODATKOV IN USMERJENI PROGRAMSKI GRAFI S POVRATNO ZANKO

Usmerjeni programski grafi so modeli s katerimi predstavljamo izvajanje programov v računalnikih krmiljenih s pretokom podatkov. Graf je sestavljen iz vozlišč in usmerjenih povezav. Vsako vozlišče predstavlja določen korak v izvajanju programa. Najbolj razširjen model programskega grafa je t.i. model s povratno zanko. Ta model je zasnovan na vozliščih v katera prihaja najmanj ena usmerjena povezava in izhaja najmanj ena usmerjena povezava. Vhodne povezave hranijo operande za ukaz, ki ga vozlišče predstavlja, izhodne povezave pa hranijo parcialne rezultate. Semantika podatka je dana s povezavo, kar pomeni, da je v vsaki povezavi prisoten le en podatek oziroma operand. Model usmerjenega programskega grafa je asinhron in deterministične. Več vozlišč se lahko izvaja simultano in časovno neodvisno, torej asinhrono, prisotnost le enega podatka v usmerjeni povezavi pri vhodu in izhodu iz vozlišča pa govori o njegovi deterministični naravi. Poleg povezav s podatki so na vhodu v vozlišče lahko prisotni tudi drugi tipi povezav. Te povezave vsebujejo krmilne signale povratne zanke (feedback signals). Krmilne povezave imajo lahko le boolove vrednosti, ki kažejo na to ali je vozlišče prosto ali ne. Vrednost podatkov v vhodni povezavi se hrani toliko časa, da vozlišče postane prosto, kar pomeni da je krmilna povezava dobila status "prosta". Model usmerjenih programskih grafov omogoča vizualno ugotavljanje vzporednosti v računalniških algoritmih. Za ugotavljanje t.i. začasne vzporednosti, ki kaže potovanje podatkov skozi več vozlišč, si pomagamo s časovnimi enačbami. Kako to poteka bomo pokazali v podpoglavjih, ki sledijo.

5. UPORABA USMERJENIH PROGRAMSKIH GRAFOV ZA UGOTAVLJANJE VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH

Za znano arhitekturo procesne enote in podani algoritem konstruiramo programski graf na podlagi modela, katerega sta prva razvila Martin in Estrin. Model opredeljujejo množica vozlišč N (množica osnovnih operacij) in množica usmerjenih povezav A (ponazarja podatkovne odvisnosti). Preslikava T iz N v množico realnih števil podaja čas izvajanja vsake operacije v vozlišču n, ki pripada množici N. Funkcija B podaja oceno verjetnosti o prisotnosti rezultata na enem od izhodov iz vozlišča n. Če je n vozlišče z več izhodi, to pomeni, da tekom obdelave v tem vozlišču prihaja do razvejitev poteka obdelave. Če sta znani preslikava T in funkcija B, potem je graf določenega algoritma časovno ovrednoten. Časovno ovrednoteni graf pišemo kot urejeno četvorko: $G = (N, A, T, B)$. Graf G uporabimo za konstrukcijo karakterističnih časovnih enačb za vzporedno in zaporedno izvajanje algoritma. Vzporedno izvajanje je mogoče, če ni podatkovne odvisnosti med posameznimi vozlišči, zaporedno izvajanje pa se nanaša na izvajanje z več medsebojno povezanih generacij vozlišč. Enačbe konstruiramo kot funkcije števila iteracij v zankah in ocene verjetnosti realizacije posameznih vej v algoritmu. Pri konstrukciji enačb si pomagamo z algoritmom, ki prehodi vse poti v programskem grafu (7). V kolikor so nam razpoložljivi računalniški viri znani, potem nam ulomek med enačbo za zaporedno in vzporedno izvajanje da oceno o notranji vzporednosti v analiziranem programu. Natančnost ocene je odvisna od tega, kako natančno lahko v modelu predstavimo obnašanje osnovne računalniške arhitekture procesne enote in do katere mere je bila pri konstrukciji programa upoštevana možnost vzporedne obdelave.

5. ANALIZA VZPOREDNOSTI V RAČUNALNIŠKIH ALGORITMIH

5.1. Model 1

Vozlišča usmerjenega programskega grafa v modelu 1 so vozlišča, ki imajo povratno zanko (8). Do izvajanja ukaza v vozlišču pride takrat, ko je vsebina operandov znana in ko so vozlišča, ki sprejemajo izhodne veličine, ta sprejem tudi potrdili. Osnovne operacije na strojnem nivoju v Modelu 1 so razvrščene v 5 kategorij:

- operacije nad elementarnimi podatki (unarne in binarne operacije, vključno s trigonometrijskimi funkcijami in predikati),
- zaporedne operacije nad datotekami (read, readedit, write, writeedit),
- operacije za uporabo procedur in sinhronizacijo (identity, merge),
- operacije nad strukturami (append, select),
- operacija "apply" za predstavljanje funkcij v obliki podatkov (na primer ko vozlišče n posreduje konstanto, ki je rezultat poteka obdelave skozi graf, ki je v programskem grafu ponazorjen z vozliščem n).

Zgledi algoritmov, ki jih bomo analizirali z modelom 1, vsebujejo enostavne podatkovne strukture kot npr. polja. Ne glede na dimenzijo preslikamo polja najprej v enonivojsko strukturo. Meje polja opredeljene v deklaraciji uporabljamo za izračun indeksov in za opredelitev elementov v spominu. Lokacija v spominu vsebuje bodisi odgovarjajočo vrednost elementa ali kazalec, ki kaže na shranjeno vrednost podatka. Kadar kličemo kakšen element iz polja, izračunamo najprej vrednost selektorja z uporabo standardnih aritmetičnih operacij in z uporabo operacije "append" ali "select" (8).

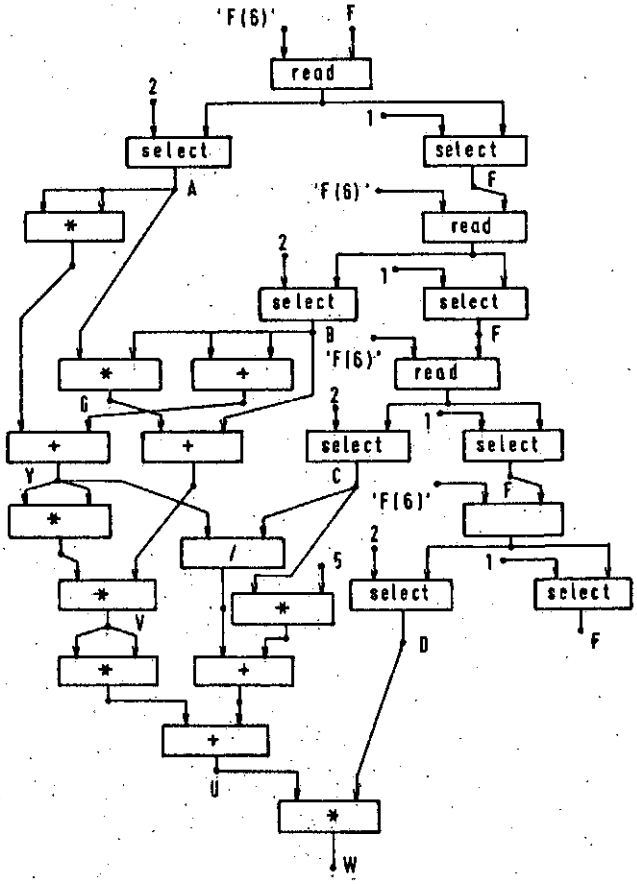
Zaporedne datotečne operacije podpirajo formatirane vhodno/izhodne ukaze. Pri ukazu "read" se najprej prebere zaporedje znakov z vrha datoteke, zatem se to zaporedje preoblikuje v notranji tip podatkov oziroma v strukturo, ki ima poleg vrednosti podatka še datotečni kazalec. Ukaz "write" poteka enako, le v obratni smeri. Ukaza "readedit" in "writeedit" podpirajo druge operacije. Ukaz "apply" zahteva za operande ime procedure in strukturo argumentov. Struktura argumenta je že vnaprej prirejena z ukazom "append". Proceduro najprej porušiti to strukturo z uporabo ukaza "select". Konec procedure pa vsebuje ukaze, ki konstruirajo strukturo rezultata. Ukaz "merge" uporablja vrednosti iz dveh virov in omogoča izvajanje konstruktov višjega nivoja le z enim vhodom in enim izhodom, kot so na primer pogojni stavki in zanke. Operacija "identity" skrbi za zamenjavo vrednosti.

Slika 1 nam ponazarja potek analize vzporednosti v algoritmih na konkretnem zgledu ob uporabi programskih grafov in ob upoštevanju računalniške arhitekture modela 1. Na sliki 1a je ponazorjen del programa v visokem programskem jeziku, na sliki 1b pa časovno ovrednoteni programski graf. Zaradi enostavnosti smo vsakem vozlišču oziroma vsaki operaciji do-

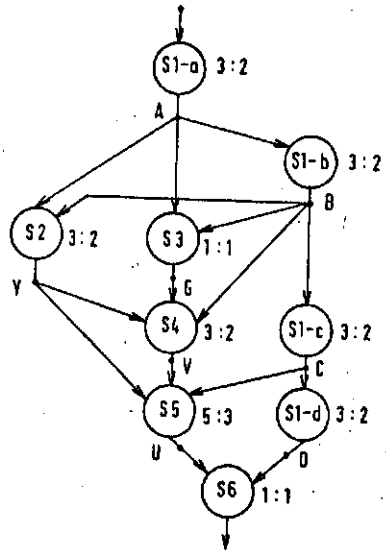
delili le eno časovno enoto. Čas vzporednega izvajanja je enak maksimalni dolžini poti v grafu pod pogojem, da imamo odgovarjajoče procesne enote. Graf iz slike 1b ima 32 podatkovno odvisnih povezav, ki so povezane s 25 vozlišči. Čas zaporednega izvajanja programa je tsec.(G) = 25, ker imamo 25 operacij. Čas vzporednega izvajanja je tpar(G) = 11, ker je globina grafa enaka 11. Oceno vzporednosti dobimo z ulomkom tsec/tpar in znaša za naš primer 2,3. Takšna analiza je precej naporna, zlasti če želimo analizirati velike programe ali kompleksne algoritme.

```
S1  input A, B, C, D, file = F
                format = 4F (6);
S2  Y:=A*A+B+B;
S3  G:=A*B;
S4  V:=Y*Y*(G+B);
S5  U:=V*V+Y/C+C*5;
S6  W:=U*D;
```

Sl. 1a. Del programa v visokonivojskem jeziku



Sl. 1b. Programski graf na strojnem nivoju
$\tau_s(G) = 25$, $\tau_p(G) = 11$, $\tau_s(G)/\tau_p(G) = 2,3$



Sl. 1c. Poenostavljena predstavitev programskega grafa
$\tau_s(G') = 25$, $\tau_p(G') = 12$, $\tau_s(G')/\tau_p(G') = 2,1$

5.2. Modifikacija modela 1

Analiza algoritmov z modelom 1 je zelo zamudna, zato model modificiramo tako, da zmanjšamo število potrebnih vozlišč. Najprej definiramo novo množico vozlišč N'. Množico N' uporabimo za predstavitev izrazov, prireditvenih stavkov, klica in izvajanje procedur (vključno s konstrukcijo argumentov), vhodno/izhodnih ukazov ipd., tako kot se ti pišejo v visokih programskih jezikih. Množica povezav A' pa nam ponazarja vrednosti podatkov med vozlišči iz množice N'. Na sliki 1c je prikazan usmerjeni programski graf G' = (N', A', T, B) z modificiranim modelom 1 za del programa iz slike 1a. Izračun časov vzporednega in zaporednega izvajanja zahteva analizo 15 povezav in 9 vozlišč. Časovno funkcijo določimo na enak način, torej kot ulomek tsec/tpar. Čas zaporednega izvajanja je enak vsoti časov vseh vozlišč. Čas potreben za izvajanje operacije v vozlišču dobimo kot vsoto vseh potrebnih operacij na nivoju stroja, ki se izvajajo v vozlišču. Tako izračunani čas ne prinaša nobene napake. Čas vzporednega izvajanja je enak globini drevesa, ki ga zgradi prevajalnik za operacije, ki se izvajajo v posameznem vozlišču. S to metodo naredimo napako pri oceni vzporednega izvajanja glede na celotni programski graf, ker poenostavljeni graf ne prikaže delnih prekrivanj, do katerih prihaja v podatkovno odvisnih vozliščih, kot so S3, S4 in S5.

REFERENCE

S. Ribarič, "Računari upravljani tokom podataka", Informatica, 6, No 4, 3 (1982).

T. Agervala, M. Arvind, Data flow systems computer, 15, 2 1982, 15.

A. Davis, R. Keller, Data flow program graphs, computer, 15, 2, 1982, 26.

D. Martin, G. Estrin, Models of computational systems, IEEE Trans. on Computers, 16, 1, 1967, 70.

D. Martin, G. Estrin, Path length computations on graph models of computation, IEEE Trans. on Comp., 18, 6, 1969, 530.

R. Russel, The Cray-1 Computer system, CACM, 21, 1, 1978, 63.

(II. del dela bo objavljen v eni od naslednjih številk časopisa)

INFORMATICA 2/1984

PRISTUP ORGANIZIRANJU BAZE PODATAKA U RASPODIJELJENIM SISTEMIMA

KUKRIKA MILAN

ELEKTROTEHNIČKI FAKULTET BANJALUKA

UDK: 681.3.06

SAŽETAK - U radu je predstavljen algoritam za rasporedjivanje zadataka u prstenastoj konfiguraciji raspodijeljenog sistema. Algoritam je postavljen tako da dodjeljivanje novog zadatka odredjenom računalu uzrokuje minimalno povećanje komunikacijskog opterećenja i da izvršno opterećenje računala u sistemu bude jednoliko.

ABSTRACT - THE APROACH TO THE DINAMIC TASKS SCHEDULLING IN RING MULTICOMPUTER SYSTEM. This paper presents a task allocation algorithm that allocates application tasks amond computers in ring distributed computing systems satisfuing minimum intercomputer communication cost and balanced utilization of each computer.

1. UVOD

Iako se elektronička računala koriste već u ranim šezdesetim godinama za upravljanja tehničkim procesima, njihova primjena bila je ograničena visokom cijenom takvih računala, kao i dugotrajnom realizacijom i tehničkim manjkavostima. Razvoj moderne tehnologije je temeljito izmijenio situaciju. Sklopovska podrška procesnih računala postala je ekonomična i pristupačna i za najmanje primjene. Tehnika mikroprocesora omogućila je podjelu zadataka na više računala (decentralizirana procesna obrada podataka) sa neočekivano širokom primjenom.

Raspodijeljeni sistemi svojom specifičnošću uvode i nove probleme, te povlače za sobom i novi pristup projektiranju složenih sistema upravljanja.

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

U slabo povezanim raspodijeljenim sistemima moguća je degradacija performansi uzrokovana učestalim zahtjevima za komuniciranjem izmedju zavisnih zadataka koji se izvode na različitim, udaljenim računalima. Problem kašnjenja uzrokovanog vremenom potrebnim da se poruka dostavi od predajnika prijemniku da se prevazići dodjeljivanjem zavisnih zdataka jednom računalu, ili pak susjednim računalima (1). Medjutim, to dovodi do nejednolikog opterećenja dijelova sistema, što za posljedicu ima smanjenje ukupnog iskorištenja sistema i brzine odgovora.

U literaturi se može naći više pristupa raspodjeli zadataka u višeprocesorskim ili višeračunarskim sistemima, koji se prema primijenjenim metodama daju općenito podijeliti u tri grupe:

- redoslijed izvršavanja zadataka može se predstaviti putem orjentiranog grafa (1,2,3,4), a ukupno komunikacijsko opterećenje minimizirati primjenom algoritama minimalnog reza. Medjutim, takav pristup postaje odviše složen ako se predpostavlja rješenje sa više od dva procesora, a teško je u model ukomponovati različita zahtijevana ograničenja,

- metode cjelobrojnog programiranja zasnivaju se na numeriranju zadataka (5,6). U takav model se lako mogu ukomponirati ograničenja. Nedostaci ovakvog pristupa su u tome što zahtijevi za vremenom i memorijom potrebnom da bi se došlo do optimalnog rješenja rastu eksponencijalno u odnosu na složenost problema,

- heuristički modeli (7,8,9) omogućuju primjenu brzih i efektivnih algoritama za pronalaženje prihvatljivih rješenja. Ovakav pristup zahtijeva manje vremena od prethodnih te se primjenjuju u kritičnim vremenskim situacijama, ili za složenije probleme.

2. O RASPOREDJIVANJU ZADATAKA PO RAČUNALIMA

Sistem upravljanja predstavljamo uredjenim parom (Z,S) odje je Z skup zadataka upravljanja, a S skup sredstava potrebnih za njihovo izvodjenje. Zadaci se izvode na osnovu stohastičkih pobuda okoline, a do konflikta dolazi kada je više istovremenih zahtjeva za isključivim korištenjem usmjereno na ista sredstva. Da bi se konflikti riješili, potrebno je primijeniti algoritam raspodjele, koji pridjeljuje sredstva zadacima odredjeni redom.

Način rasporedjivanju zadataka po računalima, kao i odredjivanje redoslijeda njihovoo izvršavanja u svakom

od računala može bitno utjecati na performanse raspodijeljenog sistema.

Pri izvodjenju sistema zadataka (Z, <), trebalo bi donijeti odluke:

- koje od računala će preuzeti na sebe kreiranje, iniciranje i izvršavanje najnovijeg zadatka kojeg bi trebalo izvesti. Drugim riječima trebalo bi odlučiti koje od računala je sa sistemskog stanovišta najpodobnije da preuzme izvršavanje najnovijeg zadatka,

- koji zadatak iz repa spremnih zadataka u pojedinom računalu nastaviti.

Ove dvije odluke odgovaraju globalnom i lokalnom rasporedjivanju, a pretpostavlja se da svako računalo u sistemu posjeduje vlastitu kopiju sistema zadataka (Z,<) te globalni i lokalni rasporedjivač.

Strategije globalnog rasporedjivanja podijeljene su u dvije skupine - nedeterminističke i determinističke.

Nedeterminističke strategije podrazumijevaju da svako od računala sadrži rasporedjivač koji će dodjeljivati zadatke računalu na osnovu probabilističkih faktora koji mogu biti:

- fiksni, tj. neovisni o parametrima sistema
- adaptivni - zavisni o jednom ili više parametara sistema kao što su stanje ostalih računala (opterećenje, brzina itd.), komunikacijskom opterećenju koje uvodi sporazumijevanje zadatka o kojem se odlučuje sa svojim partnerima koji se izvode na ostalim mjestima u sistemu, statističkim podacima dobijenim praćenjem rada sistema u eksperimentalnoj fazi itd.

Prednosti nedeterminističkih strategija, koje su neovisne o parametrima sistema, su u znatnom smanjenju komunikacijskog opterećenja u raspodijeljenom sistemu, jer rasporedjivač u odredjenom računalu ne zahtijeva pri donošenju odluke informacije o stanju svih ostalih računala u sistemu. Medjutim, u (10) je pokazano da primjena nedeterminističkih strategija koje ignoriraju stanje pojedinih računala u sistemu, a naročito medjuovisnosti zadataka koji se izvode na različitim mjestima u sistemu dovodi do vrlo loših osobina sistema.

Determinističke strategije zahtijevaju da se odluka o dodjeljivanju zadatka nekom od računala u sistemu donosi na osnovu unaprijed definiranih (determinističkih) kriterija, koji su manje-više zasnovani na informacijama o stanju sistema.

U nastavku je definiran deterministički, dinamički algoritam za rasporedjivanje zadataka koji eksplicitno uzima u obzir medjuovisnosti zadataka koji se izvode na različitim mjestima u sistemu.

3. PRIJEDLOG ALGORITMA

Raspodijeljeni sistem je sastavljen od identičnih mikroračunarskih modula. Svaki modul čine radno računalo koje izvodi zadatke, te komunikacijsko računalo zaduženo za komunikaciju u sistemu i upravljačke aktivnosti. Komunikacijska računala povezana su serijskom dupleksnom vezom, a veza komunikacijskog i radnog računala je paralelna. Detaljan opis sklopovske strukture nije od bitnog značaja za ova razmatranja, a naveden je u (11). Algoritam za rasporedjivanje razvijen je za konkretnu prstenastu strukturu prikazanu na slici 1.

Globalni rasporedjivač donosi odluku kojem računalu dodijeliti inicijalni zadatak ili zadatak čiji su svi prethodnici dovršeni prema sljedećem rasporedu:

1) Izabire se inicijalni zadatak ili prvi zadatak čiji su svi prethodnici završeni.

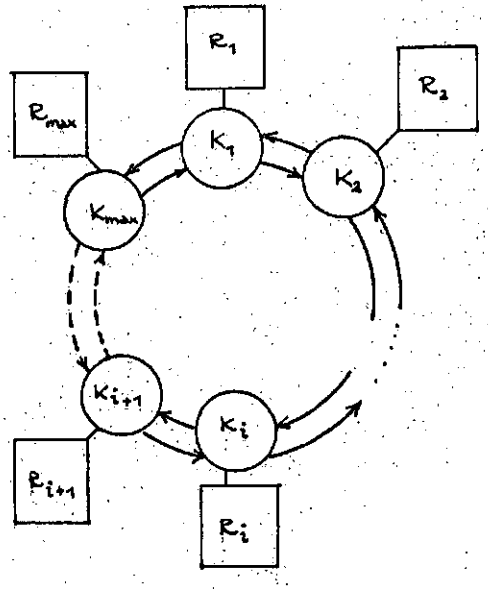
2) Ako se ne može izabrati nijedan zadatak prelazi se u stanje čekanja pa se ponavlja korak 1

3) dodjeljuje se izabrani zadatak najpogodnijem računalu, tako da se time omogući maksimalna uporednost u izvodjenju zadataka, te da izvodjenje zadatka u tom računalu izazove minimalno dodatno opterećenje komunikacijskog sistema.

4) Na osnovu informacija dobijenih od lokalnog rasporedjivača za svakog neposrednog nasljednika označava se da je neposredni prethodnik dovršen.

5) Ponavlja se korak 1

Funkcije globalnog rasporedjivača prikazane su na slici 2.



Sl.1.

Algoritam za globalno rasporedjivanje zadataka podržan je algoritmom za lokalno rasporedjivanje, koji se izvodi na svakom od računala i koji donosi odluku kojem iz repa spremnih zadataka dodijeliti procesor. Lokalni rasporedjivač manipulira sa aktivnim zadacima, ažurirajući

tri liste:

- listu spremnih zadataka
- listu zadataka koji čekaju na prijem poruka
- listu zadataka koji čekaju na prijem potvrde o predanoj poruci

Funkcije lokalnog rasporedjivača prikazane su na slici 3.

Aktivni zadatak (proces) može biti u stanjima:

R - izvodi se, ili je spreman za izvodjenje
W - čeka na prijem ili predaju poruke

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

Izmjena poruka zasnovana je na principu potvrde. Prijemnik šalje predajniku odzivnu poruku potvrdjujući korektnost prijema.

Dakle, poruka može imati dva osnovna oblika koja bi se mogla predstaviti na slijedeći način:

```
TYPE poruke = RECORD
                  vrsta: (osnovna-poruka, potvrda-
                  prijema)
                  sadržaj: tekst-sadržaja
              END
```

Da bi se korigiralo nejednoliko opterećenje u algoritmu ćemo se koristiti opteretnim faktorom C, kojeg ćemo definirati na slijedeći način:

Neka je sre-ppt srednje opterećenje sistema tj.

$$\text{sre-opt} = 1/n \sum_{i=1}^{n} \text{uk-opt}(i)$$

a V varijanca opterećenja data sa

$$V = 1/n \sum_{i=1}^{n} \text{ul-opt}(i) - \text{sre-opt}$$

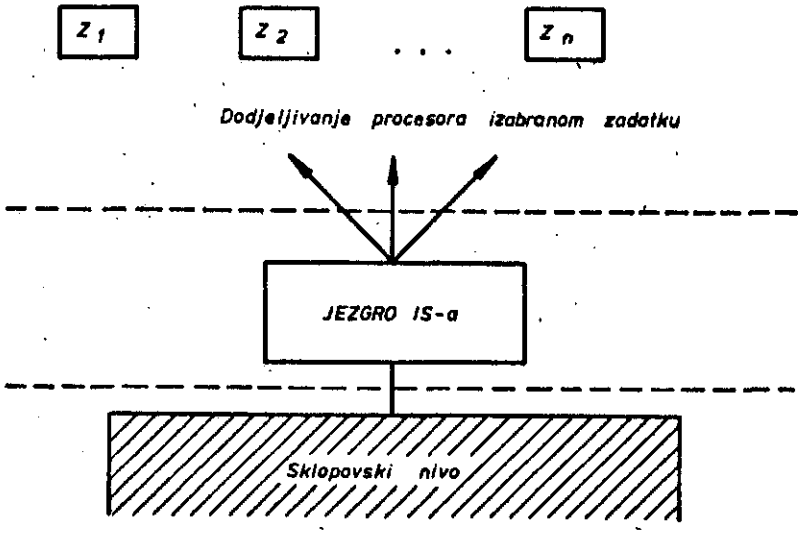
C = 1 + k * V ; gdje je k konstanta po izboru.

U slučaju da je raspodjela zadataka u sistemu jednolika V = 0, i C = 1, a ako nije V > 0 i C > 0.

Algoritam za odredjivanje koje računalo će izvesti novi zadatak odvija se uporedo u svim računalima. Informacije potrebne za odluke su slijedeće:

a) informacije sadržane u identifikatoru svakog zadatka:

e = F (ime) : prosječno vrijeme izvršavanja zadatka nov-zad (ime). Ovo vrijeme se računa samo za izvršne segmente zadatka a ne i za izmjenu poruka. Pretpostavlja se da je brzina rada svih računa identična.



Sl.2.

broj-zadataka = F (ime) : broj različitih zadataka sa kojima će zadatak komunicirati.

ime-zadatka = F (ime,i) : imena zadataka sa kojim će nov-zad (ime) komunicirati

br-por = F (ime,i) : broj poruka koje nov-zad (ime) izmjenjuje sa partnerima PR (ime-zadatka)

vk = F(ime) : kašnjenje pridruženo pojedinom zadatku

D = F (ime,računalo) : označava da li je odredjeno računalo sposobno (zbog eventualnih sklopovskih ili programskih ograničenja) izvesti zadatak

D = 1 zadatak se može dodijeliti
D = 0 zadatak se ne može dodijeliti (računalo je posebne namjene)

b) Informacije koje se dobijaju pri inicijalizaciji sistema:

n = F (računala) : broj računala u sistemu

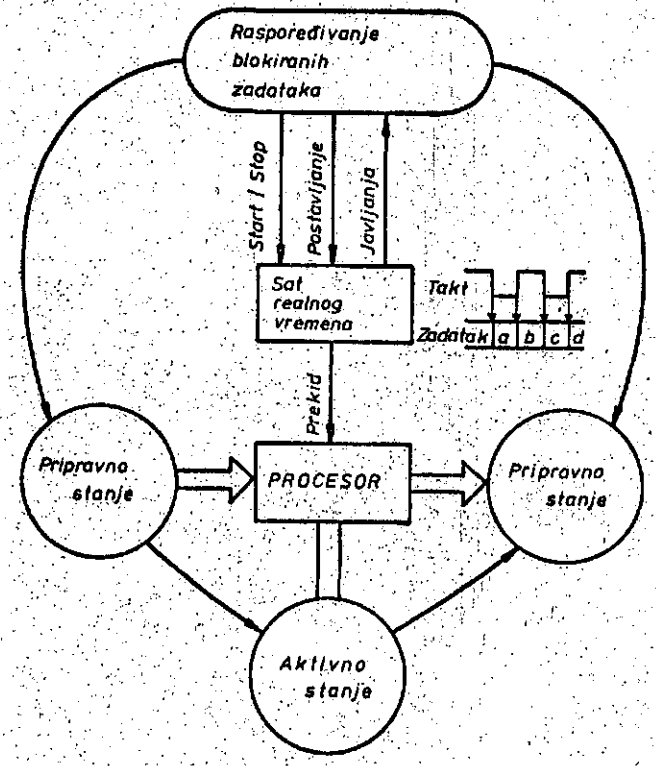
bd= F (računalo(i)): brzina kojom se poruke prenose izmedju računala "i" i susjednih računala.

c) Informacije koje će se dobiti od ostalih računala

op-rač(i)=F (računalo (i)) : opterećenje računala "i"

vre-čekⁱ(i) = F (računalo (i)) : vrijeme koje će poruka provesti u predajnom repu računala "i"

Ove vrijednosti će se izračunati i naknadno ažurirati u svakom računalu i dostaviti ostalim računalima putem procedure uzmi-nove-podatke.



Sl.3.

3.1. Izračunavanje opterećenja pojedinih računala

Opterećenje pojedinih računala dobija se kao:

op-rač(i) = kom-opt(i) + izv-opt(i)

gdje je kom-opt(i) komunikacijsko a izv-opt(i) izvršno opterećenje računala "i".

Ove vrijednosti se zavisno o promjenama ažuriraju prema sljedećem postupku:

1. startanje novog zadatka nov-zad(ime) na računalu "i",

izv-opt(i) novo = izv-opt(i)staro + e(ime)

gdje je e(ime) prosječno vrijeme izvršavanja zadatka nov-zad(ime)

kom-opt(i) novo = kom-opt(i) staro + nko (ime)

gdje je nko(ime) dodatno komunikacijsko opterećenje koje uvodi zadatak nov-zad(ime) a izračunava se pomoću procedure izračunaj-komunikacijsko-opterećenje.

2. ukoliko proces predaje ili prima poruku izvršnom opterećenju će se oduzeti vrijednost upravo izvedenog segmenta:

izv-opt(i)novo=izvr-opt(i)staro-e(ime)/(bp(ime)+1)

gdje je bp(ime) broj poruka koje će nov-zad(ime) izmijeniti sa drugim zadacima. bp(ime) se izračunava putem procedure izračunaj-komunikacijsko-opterećenje

kom-opt(i) novo = kom-opt(i) staro-pro-opt

gdje je pro-opt prosječno komunikacijsko opterećenje po poruci koju nov-zad(ime) izmjenjuje sa partnerima tj.:

pro-opt = nko(ime)/bp(ime)

što odgovara smanjenju komunikacijskog opterećenja za vrijeme potrebno da se izmijeni poruka.

3. završavanje izvodjenja zadatka:

izv-opt(i) novo = izv-opt(i) staro-e(ime)/(bp (ime)+1)

što odgovara smanjenju izvršnog opterećenja za posljednji izvršni segment zadatka.

3.2. Izračunavanje komunikacijskog opterećenja

Komunikacijsko opterećenje pojedinog računala je ovisno o broju poruka koje bi računalo trebalo izmijeniti sa partnerima, kao i o udaljenosti partnera, tj. o lokaciji na kojoj se partneri nalaze.

Procedura izračunaj-komunikacijsko-opterećenje opisana u nastavku izračunava dodatno komunikacijsko opterećenje računala ako mu se dodijeli zadatak nov-zad(ime) kao i broj poruka (br-por(ime,i)) koje će nov-zad(ime) izmijeniti sa svakim od njih. Lokacija na kojoj su smješteni zadaci ime-zadatka(ime,i) dobija se primjenom procedure nadji-računalo koja pretražuje tabelu zadataka u bazi podataka.

Procedura vre-opt dostavlja proceduri izračunaj-komunikacijsko-opterećenje vrijeme potrebno da se poruka prenese od računala "i" do računala "j" u slučaju brzine pre-

nosa "bd". To vrijeme je ovisno o

- brzini prenosa poruke preko svake dionice
- vremenu koje poruka provede u predajnom repu svakog računala što se označava sa faktorom vre-ček.

Faktor vre-ček je ovisan o dužini predajnih repova i brzini opsluživanja repa izraženoj u broju bajta u sekundi.

Komunikacijsko opterećenje računala može se izraziti sa

$$\text{kom-opt(i)} = \frac{\Sigma(\text{predajni prihvatnici})}{\text{vre-ček(i)}}$$

U nastavku je opisani postupak algoritamski predstavljen:

```
Procedure Rasporedjivač (ime) ;
(* Algoritam za odredjivanje koje računalo će preuzeti
   izvršenje novog zadatka *)
(* Ako je izvršenje zadatka PR (ime) hitno ne čekaj na
   nove podatke, iskoristi stare vrijednosti *)

IF NOT (urgent (ime)) THEN
   BEGIN
      uzmi-nove-podatke (  ) :
      (* sakupi op-rač(i) i vre-ček(i) od svih računa-
      la *)
      izračunaj-opteretni-faktor ( ): (* izračunaj
      opteretni faktor C*)
   END;
(* Traženje računala sa najmanjim kašnjenjem*)
izabrano-računalo : = 0;  (* inicijalizacija početnih
vrijednosti *)
selekcioni-faktor : = max ;

FOR i = TO (broj računala) DO
   BEGIN
    IF (D(ime,i) = 1) then
    (* ispitivanje da li je računalo sposobno da
    preuzme zadatak *)
      BEGIN
      (* izračunavanje komunikacionog opterećenja
         ako se zadatak PR (ime) dodijeli računalu "i"*)
            zadatak PR (ime) dodijeli računalu "i" *)

    kom = izračunaj-komunikacijsko-opterećenje (ime,i,
          bd)
    IF (kom + C*op-rač(i)) - < = selekcioni-faktor THEN
       BEGIN

          izabrano-računalo : = i ;
          selekcioni-faktor : = kom + C*op-rač(i);
          bp : = m(i) ;
          (* sačuvaj broj poruka izračunatih u proceduri
             izračunaj-komunikacijsko-opterećenje *)
          cm : = kom;

       END;

     END;
END;
```

```
m(ime) = bp; (* broj poruka koje će izmijeniti zadatak
nov-zad (ime)*)
ko-op = cm;
IF NOT (izabrano-računalo = 0)
  THEN
     BEGIN
         START (ime, izabrano-računalo, m(ime), ko-op(ime));
         RETURN (ime, izabrano-računalo) ;
         (* rasporedjivanje je uspješno završeno *)
     END;
  ELSE RETURN (ime,0) (* rasporedjivanje zadatka nije
               obavljeno *)
END, (* kraj procedure rasporedjivača *)

Procedure izračunaj-komunikacijsko-opterećenje (ime,
testrač,bd)
(* ova procedura izračunava komunikacijsko opterećenje i
   broj poruka u slučaju da se
   zadatak nov-zad(ime) dodijeli testiranom računalu
   testrač *)
m(ime) = 0; (* inicijalizacija broja poruka *)
test-koop = 0; (* inicijalizacija komunikacijskog opte-
                  rećenja *)
(* pronalaženje komunikacijskog opterećenja za sve za-
   datke sa kojima zadatak nov_zad (ime) komunicira*)

FOR i = 1 TO (broj-zadataka (ime)) DO
   BEGIN

     n = pronadji-računalo (ime-zadatka (ime,i)
        (* pronadji gdje se odvijaju zadaci
           sa kojima će zadatak nov-zad(ime)
           komunicirati *)

   IF n > o
        THEN
           opt = vre-pre (testrač,n,bd)

        (* izračunavanje vremena potrebnog da se poruka
              prenese od testrač do računala n
              uz pretpostavku jednolike brzine prenosa
              izmedju svih računala *)

        ELSE
           opt = pro-opt (* u slučaju da ime-zadatka ni-
                            je pronadjeno
                            uzmi prosječno opterećenje *)

        (* ažuriraj brojač poruka br(me) i komunikaci-
           jsko opterećenje *)

        (* ako je opt = 0 komunikacijski partner za-
           datka nov-zad (ime)
           je lociran u istom računalu i izračunava-
           nja nisu potrebna *)

        IF NOT (opt = 0) THEN
           BEGIN
              br(ime) = br(ime) + br-por (ime,i) ;
```

```
      (* bp(ime) je broj poruka koje
         nov-zad(ime) izmjenjuje sa drugim računalima*)

                  (* br-por (ime) je broj poruka koje
                     nov-zad(ime) izmjenjuje sa ime-zada-
                                  tka (i) *)
          test-koop = test-koop + opt*br-por(ime,i) ;
        (* ukupno opterećenje u slučaju da se
           zadatak nov-zad(ime) dodijeli računalu i *)
          END;
END;
RETURN (test-koop) ;
END;

Procedure Kaš_kan (i,j)
(* ova procedura izračunava kašnjenje koje je uzroko-
   vano čekanjem u repu poruka dok se poruka prenosila
   izmedju računala "i" i računala "j". Ovo kašnjenje
   se računa zavisno o broju dionica koje je poruka
   trebala da predje. Vrijeme potrebno da se poruka
   proslijedi izmedju susjednih računala ovisi o komu-
   nikacionom opterećenju pojedinog računala vre_ček
   (i) koje se dobija putem procedure uzmi_nove_podatke*)

IF (i > j) THEN
              BEGIN
                   s : = i;
                   l : = j

              (* s je oznaka za računalo sa manjim
                 brojem a l je oznaka za računalo sa
                 većim brojem*)

              END
            ELSE
              BEGIN

                   s : = j;
                   l : = i

              END;

brd=br_di (i,j) (* izračunavanje broja dionica izmedju
računala "i" i "j" *)
(* Izračunavanje kašnjenja poruke u svim računalima
   na putu od "i" ka "j": *)
kašnjenje : = 0 ;
IF (brd = ABS (i-j))
            THEN
              BEGIN  (* izračunavanje kašnjenja u je-
                        dnom smijeru *)
                FOR  I = s TO (l-1) DO
                   kašnjenje : = kašnjenje + vre_ček(i)

              END
            ELSE (* izračunavanje kašnjenja u suprot-
                    nom smijeru *)
              BEGIN
                FOR I = 1 TO n DO kašnjenje : = kašnje-
                          nje + vre_ček (i) ;
              END;
FOR I=1 TO (s-1)DO kašnjenje :=kašnjenje + vre_ček(i);
RETURN (kašnjenje) ;
END;
```

```
Procedure vre-opt (i,j,bd);
(* Ova procedura izračunava vrijeme potrebno da se poru-
   ka prenese od računala "i"
   do računala "j" u slučaju brzine prenosa "bd" *)

   IF NOT (i = j)
       THEN
         BEGIN
           brd = br-di (i,j);
               (* izračunavanje broja dionica izmedju
                  računala "i" i "j" *)
           pre-por = dio (bd) ;
               (* izračunavanje vremena za prenos
                  poruke izmedju dva susjedna računala*)


           kaškan = kaš-kan (i,j);

                (* izračunavanje kašnjenja zbog čekanja
                   u repu u smijeru od "i" ka "j" *)

           vrijeme = brd * (1,5 * pre-por) + 2 * kaškan;
           (* svaka poruka koju preda računalo "i" biće
              potvrdjena od strane računala "j" - zbog
              toga se uvodi 2 * kaškan . Medjutim, pošto
              je dužina potvrde kraća od dužine same po-
              ruke uzima se aproksimacija da je
              pre-por (potvrda) = 0,5 * pre-por (poruka)*)

     END;
    ELSE vrijeme : = 0 ;
   RETURN (vrijeme ;
END;

Procedure br-di (i,j)
(* Ova procedura izračunava broj dionica puta izmedju
   računala "i" i "j" *)

   X : = i - j;
   Y : = abs (X) ;
   IF (Y > n/2) THEN (* n je broj računala u mreži *)

    BEGIN
      IF (X > 0) THEN i : = n - i
                 ELSE j : = n - j ;

      Y : = i + j ;

    END;
   RETURN (Y) ;
END ;

Procedure dio (bd) ;
(* Ova procedura izračunava koje vrijeme je potrebno da
   da se poruka prenese izmedju dva susjedna računala u
   mreži. Pretpostavlja se identična dužina poruka i br-
   zina prenosa definirana sa "bd" *)
   vrijeme : = Pro_bit/bd;
(* pro_bit je prosječan broj bita koji sadrži odredjena
   poruka *)
return (vrijeme)
END ;
```

4. ZAKLJUČAK

Jedan od osnovnih problema koji obuhvata projektiranje raspodijeljenih sistema za vodjenje složenih procesa je način raspodjele zadataka koje je potrebno izvršiti u cilju upravljanja. U radu je predstavljen dinamički algoritam za rasporedjivanje zadataka u višeračunarskom sistemu, koji uzima u obzir medjuovisnost zadataka koji se izvode na različitim mjestima u sistemu, te strukturu komunikacijske mreže. Algoritam je razvijen za konkretnu prstenastu strukturu kod koje svako radno računalo sadrži vlastitu kopiju sistema zadataka, a zadaci komuniciraju medjusobno izmjenom poruka.

Globalni rasporedjivač će na osnovu proračuna minimiziranja dodatnog komunikacijskog opterećenja koje uvodi novi zadatak iz skupa alternativa izabrati ono rješenje koje će u okviru zacrtanom algoritmom maksimalno zadovoljiti postavljene kriterije.

Nedostatak predloženog algoritma je u tome što će kod složenije konfiguracije dodatno neproduktivno vrijeme potrebno za sakupljanje svih potrebnih informacija biti preveliko. Ukoliko vremenski uvjeti to ne dozvoljavaju opisani model mora se zamijeniti modelom za upotrebljivo djelovanje (9). U takvim slučajevima algoritam se može koristiti pri razvoju sistema, da se iz velikog skupa mogućih rješenja unaprijed izaberu najprihvatljivija.

LITERATURA

1. Bokhari, S.: "On the mapping problem", IEEE trans. on computers vol c30, no.3 (march 1981).
2. Bokhari, S.: "Dual processor scheduling with dinamic reassignment", IEEE trans. on soft. eng. vol SE-5 no. 4 (july 1979).
3. Chou, T.: "Load balancing in distributed systems", IEEE trans. on soft. eng. vol SE-8 no.4 (july 1982).
4. Lamport, C.: "Time, clocks and ordering of events in a distributed system" Communications of the ACM, July (1978).
5. Ma, P. et al.: "A task allocation model for distributed computing systems" IEEE trans. on computers vol c31, january 1982.
6. Desouki, O et al.: "Distributed enumeration on network computers, IEEE trans. on computers vol c-29, sept. 1980.
7. Štrkić, G., Kukrika, M.: "Neke mogućnosti rasporedjivanja zadataka u višeračunarskom sistemu sa radom u realnom vremenu" II Jugoslovensko savjetovanje u mikroračunalima u procesnom upravljanju - MIPRO, Opatija (1983).
8. Kukrika, M., Štrkić, G.: "Primjer višeprocesorskog sistema u vodjenju procesa u realnom vremenu", XXVII Jugoslovenska konferencija ETAN, Struga (1983).
9. Kukrika, M.: "Neke mogućnosti ravnomjernog korištenja računala u višeračunarskim sistemima sa radom u realnom vremenu", u pripremi za objavljivanje.
10. Chow,Y et al.: "Models for dynamic load balancing in a heterogenous multiple processor system", IEEE trans. on computers, vol. c-28, no.5 (may 1979).
11. Kukrika, M.: "Problemi komuniciranja u sistemima sa više mikroračunala", II Jugoslovensko savjetovanje o mikroračunalima u procesnom upravljanju-MIPRO, Opatija (1983).

ALGOL 60 ZA SISTEM CP/M III

A. P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.06 Algol 60:519.682

Članek opisuje prevajanje in izvajanje algolskih programov s sistemom CP/M, tj. ukazne prevajalne in izvajalne vrstice, njihove različne oblike. Opisana je organizacija simbolne tabele in notranja predstavitev simbolnih tipov, organizacija sklada v izvajalnem času z njegovimi glavnimi kazalci. Prikazana je možnost dodajanja lastnih kodnih segmentov k obstoječim algolskim programom, kot so vhodni in izhodni programski vmesniki in druge subrvutine. Opisana sta tudi sistema za sporočanje napak v prevajalnem in izvajalnem času. Na koncu članka so dodani nekateri primeri algolskih programov s področij aproksimacije, določanja korenov in drugi primeri.

A CP/M System Algol 60 Language III

This article deals with compiling and running Algol programs under CP/M, i.e. with compiling and running commands, their different shapes. Organisation of symbol tables and the internal presentation of symbol types are described and the runtime stack organisation with corresponding pointers is presented. The possibility od adding code sections to existing programs like input and output handlers and other subroutines is shown. Further, systems for compiler and runtime error messages are described. At the end some examples of Algol programs in the area of aproximation, root searching, and other cases are discussed.

4. Prevajanje in izvajanje programov s sitemom CP/M

Uporaba jezika RML Algol je dvostopenjski proces:

Prevajanje: Prevajalnik prebere izvirni algolski program in proizvede iz njega izhodno zbirko v takšni obliki, da jo bo lahko prebral izvajalni sistem.

Izvajanje: S prevajalnikom proizvedena izhodna zbirka se naloži oziroma prebere in se nato izvede (izvrši).

Najenostavnejše zaporedje ukazov pri danem izvirnem programu (zbirki) PROG.ALG bi bilo

```
za prevajanje:   ALGOL PROG
za izvajanje:    ARUN PROG
```

Oglejmo si ta postopka podrobneje. Osnovni zbirki sta

```
prevajalnik:   ALGOL.COM
izvajalnik:    ARUN.COM
```

Če diskovne enote ne navedemo, se uporabi trenutno aktivna enota. Kadar manjkajo zbirčne razširitve, velja tole:

```
izvirna zbirka:       .ALG
prevajalniški izhod:  .ASC
monitorska zbirka:    .MON
```

4.1. Prevajanje

Prevajalnik najprej prebere navedeno izvirno zbirko (tipa .ALG) in jo prevede v zbirko z enakim imenom in z razširitvijo .ASC . Če obstaja zbirka s tem imenom in s to razširitvijo, se ta zbriše in se oblikuje nova zbirka. Če prevajalnik zazna napako v izvirnem programu, zbriše nastalo izhodno zbirko, vendar s prevajanjem nadaljuje do konca programa s sporočanjem nadaljnjih napak. Sporočila o napakah se pošiljajo na konzolo. Na koncu prevajanja se sporoči obseg rezultatnega programa.

Splošnejša oblika prevajalniškega klica je

```
ALGOL outlist = inlist
```

Tako imamo npr.

```
ALGOL OUT=IOLIB, B:MATHS, PROG
```

V tem primeru se izvirni vhod oblikuje iz zaporedja navedenih zbirk; to zaporedje pa mora izpolnjevati zahteve algolskega programa od začetnega BEGIN do končnega END in FINISH. Zbirke se lahko vzamejo iz različnih diskovnih enot. V zadnjem primeru se zbirka IOLIB.ALG vzame iz trenutno aktivne enote, zbirki MATHS.ALG in PROG.ALG pa iz diskovne enote B: . Prevajalniška izhodna zbirka OUT.ASC se oblikuje na trenutno aktivni diskovni enoti. Druga metoda kombiniranja izvirnih zbirk uporablja LIBRARY stavek, ki smo ga že obravnavali. Tu poudarimo, da so knjižnični klici omejeni na zadnjo navedeno zbirko vhodnega seznama (knjižnična procedura 'findinput').

Kadar navedemo še drugi izhodni tok (OUT2), se generira seznam prevajalniških identifikatorskih tabel. Tudi prevajalniška sporočila o napakah se pošiljajo v ta tok skupaj s podatki o maksimalmih obsegih tabel, ki jih sistem podpira. Ukaz

ALGOL OUT1, OUT2 = PROG

pošlje prevajalniški izhod v OUT1.ASC in prevajalniška sporočila o napakah in identifikatorske tabele v zbirko OUT2.MON. Ukaz

ALGOL OUT1, CON:= PROG

pošilja sporočila o napakah in identifikatorske tabele na konzolo.

Kadar vhoda/izhoda (V/I) ne navedemo, ali če obstaja napaka (npr. pri slabi sintaksi ali neobstoječi zbirki), bo izdal prevajalnik zahtevo za navedbo V/I. Pri klicu oblike

ALGOL

se bo pojavila zahteva oblike

OUT = IN?

Uporabnik določi v tem primeru seznam vhodnih in izhodnih zbirk kot v prejšnjem (zgornjem) primeru.

Kadar se V/I zbirke navedejo v začetnem klicu, npr.

ALGOL PROG

se prevajalnik vrne po prevajanju v sistem CP/M. Kadar se V/I zbirke navedejo na zahtevo prevajalnika, se prevajalnik po prevajanju vrne na svoj začetek z novo zahtevo. Z znakom 'CTRL-C' se prevajalnik vrne v sistem CP/M.

4.2. Izvajalni sistem

Prevedeni program lahko pokličemo v izvajanje s klicem

ARUN ime_zbirke

Razširitev te zbirke je '.ASC'. Ta zbirka se naloži in izvrši (izvede). Kadar vhoda (imena zbirke) ne navedemo, ali imamo napako (slaba sintaksa ali neobstoječa zbirka), bo izvajalni sistem izdal zahtevo za vhodno zbirko. Npr. klic

ARUN

bo povzročil zahtevo za vhod

INPUT=

na katero bo uporabnik odgovoril z navedbo imena zbirke. Po izvršitvi programa natisne sistem znak '^' in čaka na uporabnikov odgovor. Z znakom 'CTRL-p' se sproži ponovno izvajanje programa, z znakom 'CTRL-c' pa skok v CP/M.

Kadar se v času izvajanja programa pojavi napaka, se pošlje sporočilo na konzolo. Če je bila uporabljena možnost obravnave napake (knjižnična procedura 'error'), bo sistem počakal na uporabnikovo navodilo za raziskavo napake. Od tu se lahko program ponovno požene z znakom 'CTRL-p' ali pa se vrne krmiljenje sistemu CP/M z znakom 'CTRL-c'.

Vrnitev v CP/M po končanem izvajanju programa ali po ugotovitvi napake se doseže avtomatično z uporabo klica

ioc(22);

v izvirnem programu. Klic oblike

ioc(60);

pa sproži takojšnjo ponovitev izvajanja programa od njegovega začetka. Če so bile v fazi izvajanja odprte zbirke, se te ob prekinitvi zaradi napake ne zaprejo, vendar se sprostijo zbirčni krmilni bloki.

4.3. Pomnilniško preslikane zbirke

Kadar namerava uporabnik izvajati program večkrat, ga lahko shrani na disku v kompaktnem formatu pomnilniško preslikane zbirke, ko uporabi stikalo [S] v času izvajanja, in sicer

ARUN ime_zbirke [S]

Po naložitvi zbirke zbriše sistem zbirko z razširitvijo '.ASC', ki jo je proizvedel prevajalnik in oblikuje novo zbirko z istim imenom in razširitvijo (.ASC) v pomnilniško preslikanem formatu. Nadaljnji pozivi oblike

ARUN ime_zbirke

sprožajo avtomatično naložitev in izvajanje preslikane zbirke. Prvotna zbirka tipa .ASC je bila sistemsko neodvisna, novi format pa ni premestljiv in sprememba v sistemu lahko povzroči zahtevo za ponovno prevajanje zadevne zbirke.

5. Simbolna tabela in simbolni tipi

Prevajalnik lahko izlista na konzolo ali v monitorsko zbirko seznam vseh deklariranih identifikatorjev z informacijo o njihovih tipih in naslovih v pomnilniku. Spremenljivke se shranijo v skladu in številka spremenljivke je položaj v skladu, ki je relativen glede na kazalec. Kazalec je shranjen na lokaciji PBASE v izvajalnem programu. Naslov spremenljivke se najde z množenjem številke spremenljivke s 4 in s prištetjem tega k vsebini PBASE.

Štiri števila se tiskajo z vsakim identifikatorjem v prevajalniški simbolni tabeli. Prvo število je položaj v skladu z izjemo pri oznakah in procedurah. Pri njih je natisnjena simbolna označitvena številka, ki je številčni del simbola, kot je npr. L123, ki ga izda prevajalnik.

Drugo število je številka procedure, ki obdaja proceduro, za katero je identifikator deklariran. Glavni program ima številko 0 in procedure se oštevilčujejo zaporedno, kot se pojavljajo, neglede na deklaracijsko globino. Izjemoma se natisne dejanska številka procedure namesto številke obdajajoče procedure. Tretje število je številka vrstice v izvirnem programu. Četrto število je trenutni obseg prevedenega koda. Ta podatek je lahko povezan s položajem napake v izvajalnem času.

Tipski podatek identifikatorja se izlista. Številke pomenijo notranjo predstavitev podatkovnih tipov, in sicer:

0	procedurni formalni parameter (tip še ni znan)
1	realno
2	celoštevilsko
3	boolovsko
5	realno polje
6	celoštevilsko polje
7	boolovsko polje

8 pretikalo
10 procedura
11 realna procedura
12 celoštevilska procedura
13 boolovska procedura
14 oznaka

Prevedeni kod procedure vsebuje seznam tipov parametrov. Tu se lahko pojavijo še dodatni tipi:

4 niz
21 realno z imenom (poziv)
22 celoštevilsko z imenom
23 boolovsko z imenom

6. Organizacija sklada v izvajalnem času

Podatki o organizaciji so potrebni v primeru, ko moramo dostopati k algolskim spremenljivkam iz danega računalniškega koda. Sklad se razprostira od konca izvajalnega programa do konca razpoložljivega pomnilnika. Sklad spremenljivk raste navzgor od konca programa do delovnega sklada za izračunavanje izrazov, za posredovanje procedurnih parametrov, procedurnih pozivnih ukazov, ki rastè navzdol od konca pomnilnika. Sklad spremenljivk je sestavljen iz okvirov, in sicer iz enega za glavni program in po enega za vsak procedurni poziv. V vsakem skladnem okviru se nahaja poljski sklad, ki vsebuje poljski okvir za poljubno globino poljske deklaracije. Pojem "beseda" se uporablja za 16-bitne (dvozložne) vrednosti. Imamo:

PBASE kaže na trenutni okvir sklada spremenljivk
MBASE kaže na okvir glavnega programa; ko se glavni program izvaja, imata PBASE in MBASE enako vsebino
WSBAS kaže na začetek delovnega sklada trenutnega nivoja in se uporablja za brisanje označitvenih plavačev iz delovnega sklada
ABAS kaže na trenutni poljski okvir
FSPT kaže na naslednjo prosto lokacijo v skladu spremenljivk

Pomembni so tudi tile registri:

SP kaže na vrhnji element delovnega skalda; ta podatek se mora rešiti in obnoviti pri dodajanju računalniškega koda; uporablja se tudi pri pozivnih (CALL) ukazih
IX naj se shrani, če bo uporabljen; to je kazalec interpretivnega kodnega programa
IY se mora shraniti, če bo uporabljen, saj kaže na zaporedje zastavic in na delovni prostor

Vsak skladni okvir je razdeljen v dva dela: spremenljivški in poljski. Spremenljivški del ima enote z dvema besedama (4 zlogi). Dejanski naslov enote dobimo z množenjem številke enote s 4 in s prištetjem zmnožka k osnovnemu naslovu, ki je shranjen v PBASE ali v MBASE. V okviru glavnega programa je prva deklarirana spremenljivka v enoti 2 in beseda, na katero kaže MBASE vsebuje 0, tj. nivojsko številko glavnega programa. V okvirih procedurnega tipa se enota 3 uporablja za rezultat funkcije in je neuporabljena v procedurah, ki ne dajejo rezultata. Procedurni parametri zasedajo enote od 4 navzgor, naprej pa sledijo v proceduri deklarirane spremenljivke. Prva beseda vsake procedure prevedenega programa vsebuje število spremenljivških enot, potrebnih za proceduro. Prva beseda glavnega programa kaže na zadnjo besedo prevedenega programa, ki vsebuje število spremenljivških enot, potrebnih za glavni program.

V procedurnih okvirih se prve tri enote uporabljajo za povezovalno informacijo. Začenši z besedo, na katero kaže PBASE (enota 0), vsebujejo besede tele podatke:

beseda 1: številka procedure
beseda 2: vrnitveni naslov
beseda 3: PBASE klicajoče ravnine
beseda 4: WSBAS klicajoče ravnine
beseda 5: ABAS klicajoče ravnine

Enota spremenljivčnega sklada lahko vsebuje tele tipe elementov:

1. realno število v standardnem formatu 4 zlogov
2. celo število ali boolovsko vrednost v najviše naslovljeni besedi enote; boolovska vrednost zasede le najnižji zlog te besede
3. oznako ali procedurni naslov kot procedurni parameter; naslo se nahaja v najviše naslovljeni besedi in v besedi pod njo je vrednost PBASE, ki ustreza vrednosti v trenutku, ko je bil naslov izračunan
4. naslov polja ali preklopnika kot deklarirane spremenljivke ali procedurnega parametra; naslov je v najviše naslovljeni besedi enote, preostala beseda pa je neuporabljena
5. nizni naslov ali naslov neindeksirane spremenljivke za procedurni parameter niznega ali spremenljivčnega tipa, klicanega z imenom

Naslov preklopniške spremenljivke kaže na preklopniški vektor. Kazana beseda vsebuje število elementov preklopnika, naslednje besede pa hranijo naslove oznak v preklopniškem seznamu.

Poljski del skladnega okvira vsebuje število poljskih ravnin, oštevilčenih z deklaracijsko globino v proceduri ali v glavnem programu. Ravnina 0 vselej obstaja in se nahaja neposredno za koncem spremenljivk. ABAS kaže na osnovo trenutne ravnine, katera vsebuje podatek o globini te ravnine. Naslednja beseda (z izjemo ravnine 0) vsebuje kazalec na nižjo ravnino. Nad ravninsko informacijo so obveščevalni vektorji in poljski elementi.

Poljska spremenljivka kaže na začetek zadevnega obveščevalnega vektorja. Ta vsebuje 2*(N+1) besed, kjer je N indeksno število. Prva beseda obveščevalnega vektorja vsebuje število zlogov, ki jih vsak element zaseda (1, 2 ali 4), druga beseda število indeksov in tretja spodnjo mejo prvega indeksa ((23)).

Obstajata še dve dodatni besedi za vsak dodatni indeks. Prva beseda vsebuje multiplikator za predhodno akumulirano število elementov, druga beseda pa spodnjo mejo naslednjega indeksa (dimenzije). Končna beseda obveščevalnega vektorja vsebuje naslov besede za koncem poljskih elementov. Poljski elementi se shranjujejo neposredno za obveščevalnim vektorjem.

7. Dodajanje kodnih segmentov

Uporabnik želi večkrat dodati svoje rutine sistemu izvajalnega časa. Te rutine so npr.:

1. V/I programski vmesniki
2. specializirane kodne subrutine
3. povezovanje teh subrutin s sitemom za sporočanje napak v izvajalnem času

Pri teh dodatkih ni potrebno posebno znanje o notranji zgradbi sistema. Uporabnik mora razumeti seznam, ki ga izda izvajalni sistem. Dodatna aplikacija je povezana s seznamom vstopov. Kazalci k tem listam se nahajajo blizu začetka sistema izvajalnega časa. Splošna oblika seznama je:

```
IME: n          : dolžina
     vstop 0
     vstop 1
     . . . .
     vstop n
```

Prvi zlog določa dolžino seznama, začenši pri 0. Za ime seznama je ustrezni kazalec blizu začetka izvajalnega programa. Natančne lokacije in vsebine teh seznamov se lahko razlikujejo od dejanskih vrednosti v porazdeljenem sistemu, toda kazalčni naslov bo pravilen. Vstopi v seznamu vsebujejo naslove ustreznih vstopnih točk. Druge bistvene spremenljivke so še:

PROGST — vsebuje začetek prostega prostora; tu se bo začel programski kod
ENDLIST — vsebuje naslov, nad katerim se seznami ne smejo pojaviti

Različni seznami se nahajajo v pomnilniku zaporedno. Če želimo dodati V/I programske vmesnike ali kodne rutine, moramo modificirati tole:

1. napišemo kod, ki začenja pari naslovu, vsebovanemu v PROGST
2. popravimo PROGST na kazalčno vrednost prostega prostora
3. oblikujemo vstop v ustreznem seznamu in popravimo seznamsko dolžino, če je to potrebno

Vsak seznam vsebuje prazne enote ali prostor na koncu, predviden za razširitev. Če to ne zadostuje, moramo premestiti celotne sezname in popraviti kazalce. Pri tem upoštevamo:

IX in IY — morata ob izstopu ohraniti vrednosti vstopa
SP — se uporablja za subrutinske pozive in za delovni sklad; izstop iz kodnih rutin poteka prek RET ukaza, zato mora SP ostati uravnotežen

Ostali registri se smejo poljubno uporabljati.

7.1. Vhodni programski vmesniki

Tu imamo:

INLP — kazalec za vhodni seznam
INLIST — ime vhodnega seznama

Vhodni programski vmesniki preberejo zlog iz vhodnega vira v register A in izstopijo prek ukaza RET. S stavkom

```
c := chin(n);
```

se prebere zlog iz naprave, ki ji ustreza vstop 'n' v seznamu.

7.2. Izhodni programski vmesniki

Tu imamo:

OUTLP — kazalec izhodnega seznama
OUTLST — ime izhodnega seznama

Izhodni programski vmesniki izdajo zlog v register A in izstopijo prek ukaza RET. S stavkom

```
chout(n,c);
```

se pošlje znak 'c' v napravo 'n' seznama.

7.3. IOC seznam in kodne subrutine

Tu je:

IOCLST — kazalec za IOC seznam
DKLST — ime IOC seznama

Procedura ioc(n) prenese krmiljenje na naslov, povezan z vstopom 'n' v IOC seznamu. Vrnitev se izvede prek ukaza RET. Kadar se ioc poziv pojavi v procedurnem telesu (npr. v procedurah knjižnice ALIB), je dostop do procedurnih parametrov neposredno urejen. Procedurni parametri začenjajo od enote 4 naprej, rezultat se pri funkcijski proceduri shrani v enoto 3 (sklad). Naslov enote je dan z izrazom

(vsebina PBASE) + 4*(številka enote)

Tako je prvi procedurni parameter določen z

(vsebina PBASE) + 16

Pri tem zasede celo število gornjo polovico 4-složne enote in h gornji vsoti je treba prišteti 2.

8. Sporočila o napakah v prevajalnem času

V primeru napake se v prevajalnem času pojavi sporočilo

```
FAIL X ON LINE Y IDENT Z SYMBOL S
kopija vrstice do (vključno) napake
```

V sporočilu je X številka napake, razvidna iz seznama napak; Y je številka programske vrstice, v kateri se napaka pojavlja; S je decimalna vrednost zadnjega simbola (glej odstavek 8.2). Pod prvo vrstico sporočila o napaki se pojavi kopija vhodne programske vrstice vključno s simbolom, pri katerem je bila napaka ugotovljena. Pri pojavitvi prve napake zbriše prevajalnik izhodno zbirko, vendar nadaljuje s preverjanjem sintakse preostalega programa. V enoprehodnem prevajalniku lahko "trdovratne" napake sprožajo vrsto sporočil in pojavi se lahko popolna (sintaksna) asinhronizacija.

8.1. Pomen številk X v sporočilih o napakah

Za številke X imamo tale pomenski seznam:

1 identifikator je v bloku dvakrat ali večkrat deklariran
2 identifikator je nedeklariran
3 manjka oglati oklepaj '[' za imenom polja; oklepaja ni, če se ime polja pojavlja kot procedurni parameter ali kot navadna procedura, uporabljena kot funkcija
4 manjka oglati zaklepaj ']' na koncu indeksnega seznama
5 v glavnem programu ali v proceduri je več kot 255 spremenljivk
6 ni besede FINISH na koncu programa (ali pa je preveč besed END)
7 ni ELSE dela v pogojnem aritmetičnem izrazu
8 ni ELSE dela v pogojnem boolovskem ali v pogojnem označitvenem izrazu
9 relacijski operator ni bil najden na pričakovanem mestu; napaka se pojavi, če je prvi aritmetičen izraz boolovskega relacijskega izraza v celoti zaprt v okrogle oklepaje
10 aritmetičen predmet ne začenja s +, -, ., (, številko ali identifikatorjem
11 operator %, MOD, !, MASK ali DIFFER nima dveh celoštevilskih operandov

12 manjka ')' v aritmetičnem izrazu
13 krmiljena spremenljivka v FOR stavku ni deklarirana ali pa je indeksirana
14 manjka ')' v boolovskem ali označitvenem izrazu
15 v območju vidljivosti je več identifikatorjev, kot jih tabele lahko sprejmejo; prevajalnik oblikuje tabele avtomatično do največjih obsegov, ki jih dani sistem dopušča
16 stavek se začenja nepravilno; če se ta napaka pojavi pri FINISH, je premalo ENDov
17 na levi strani znaka ':=' je nedeklariran ali neustrezen identifikator
18 poljska deklaracija je napačna
19 tipska specifikacija dejanskega parametra ni LABEL, PROCEDURE, REAL PROCEDURE, BOOLEAN PROCEDURE ali INTEGER PROCEDURE
20 število indeksov je napačno; v primeru formalnih polj se ta napaka zazna šele v času izvajanja
21 manjka ')' za seznamom dejanskih parametrov
22 element FOR stavka ni zaključen z ',' ali z DO členom
23 procedurno telo ni omejeno s ';'
24 znak ':=' se ne nahaja na pričakovanem mestu
25 manjka THEN za IF
26 specifikacija VALUE ni prva specifikacija procedurnih formalnih parametrov
27 FINISH je v sredini programa; bržkone obstaja nezaključen simbol BEGIN, '"' ali '
28 seznam procedurnuh formalnih parametrov se ne kočuje z ')'
29 parameter je dvakrat specificiran ali ni v formalnem seznamu ali pa specifikacija ni zaključena s ';'
30 označitveni/procedurni seznam je poln
31 UNTIL ni na pričakovanem mestu
32 manjka '(' za imenom standardne procedure (z izjemo vhodne ali izhodne)
33 THEN se pojavlja takoj za IF
34 procedurni dejanski parameter začenja z nedeklariranim identifikatorjem
35 funkcija ali spremenljivka se uporablja kot procedura
36 procedurnemu vhodu ali izhodu sledi '('
38 aritmetični izraz vsebuje boolovsko spremenljivko v neveljavnem kontekstu
39 parameter, ki je specificiran z VALUE, se ne pojavlja v formalnem seznamu
40 parametrska specifikacija je nepopolna
41 polje je klicano z vrednostjo
42 klicna napaka vhodne/izhodne procedure
44 celoštevilski literal ni v območju
46 preklopniški seznam ne končuje s ';'
47 preklopnik ima več kot en indeks
48 besedi BYTE ne sledi beseda ARRAY
49 vhodne zbirke so izčrpane brez razpoznave konca programa
50 procedura je uporabljena pred svojo deklaracijo in ima bržkone različen tip od dejanskega; preureditev procedur je nujna, da ne bo vnaprejšnjega navajanja
51 vhodna zbirka knjižnice (LIBRARY) ni bila najdena

8.2. Prevajalniška predstavitev osnovnih simbolov

Decimalne številke simbolov tiska prevajalnik v sporočilih o napakah (SYMBOL S, kjer je S številka). Jezikovne rezervirane besede so v izvirnem algolskem programu zaprte v enojne narekovaje ali izpisane z velikimi črkami, v prevajalniku pa so predstavljene s številko

*40*prva_črka + druga_črka*

Kadar sporočilo o napaki vsebuje številko, ki ni v seznamu, je bil uporabljen neveljaven simbol. Povzročitelj take napake je lahko neuravnovešen enojni narekovaj. Imamo:

črke A do Z		1-26
[	(oglati oklepaj)	27
]	(oglati zaklepaj)	29
^	(potenciranje)	30
:=		7000
!		33
"	(nizni narekovaj)	34
#	(neenakost)	35
¢		36
%	(celoštevilsko deljenje)	37
>=	(večje kot ali enako)	38
(		40
)		41
*	(množenje)	42
+		43
,	(vejica)	44
-	(minus)	45
.	(pika)	46
/	(realno deljenje)	47
številke 0 do 9		48-57
:	(dvopičje)	58
;	(podpičje)	59
<	(manjše kot)	60
=	(enakost)	61
>	(večje kot)	62
<=	(manjše kot ali enako)	63
BEGIN		85
BOOLEAN		95
BYTE		105
AND		118
ARRAY		122
COMMENT		135
DIFFER		169
DO		175
ELSE		212
END		214
EQUIVALENT		217
FALSE		241
FINISH		249
FOR		255
GOTO		295
IF		366
IMPLIES		373
INTEGER		374
LABEL		481
LIBRARY		489
MASK		521
MOD		535
NOT		575
OR		618
PROCEDURE		658
REAL		725
STRING		780
SWITCH		783
THEN		808
TRUE		818
UNTIL		854
VALUE		881
WHILE		928

9. Sporočila o napakah v izvajalnem času

Kadar se med izvajanjem programa pojavi napaka, pošlje izvajalni sistem sporočilo na izhodno napravo 1 (na konzolo ali zaslon). To sporočilo ima obliko

```
ERROR n
ADD   PBASE  PROC  LOC
aaaa  bbbb    p1    d1
aaaa  bbbb    p2    d2
. . . . . .
aaaa  bbbb    0     d0
```

To sporočilo ima tale pomen:

n	je številka napake (glej podpoglavja)
p1	je številka procedure z napako
aaaa	je naslov programskega števnika
bbbb	je vrednost kazalca PBASE pri napaki
d1	je lokacija programskega števnika relativno k začetku programa

Naslova aaaa in bbbb sta tiskana heksadecimalno.

Številko procedure pi najdemo s preštevanjem procedur od programskega začetka, začenši s številko 1. Glavni program ima številko 0. To informacijo dobimo tudi v prevajalniški identifikatorski tabeli. Če je vrednost p1 različna od 0, se v naslednjih vrsticah sporočila natisnejo podatki za pi do ravnine 0 (pi = 0). Ti podatki se lahko uporabijo za raziskovanje narave napake. Informacija d1, d2, ... je lahko povezana s podatki v prevajalniških identifikatorskih tabelah, ko se ugotavljajo položaji napak. Program se lahko ponovno izvaja od začetka z vtipkanjem znaka 'CTL-p', z znakom 'CTL-c' pa izstopimo v CP/M.

9.1. Zasledovanje napake v izvajalnem času

Pri normalnih pogojih se izvajanje programa konča z zaznavo napake. Vendar je mogoče nadaljevati z zasledovanjem napake, tako da omogočimo izstop z zaprtjem izhodnih zbirk, z navedbo diagnostične informacije, vrednosti spremenljivk itd. To zasledovanje se doseže s klicem procedure 'error' (iz knjižnice ALIB.ALG) v programu pred pojavitvijo napake, npr. z

```
error(LABEL zlom);
```

Pri zaznavi napake proizvede izvajalni sistem ustrezno informacijo in prenese izvajanje k oznaki ali označitvenemu izrazu 'zlom' v uporabniškem programu. Ta oznaka naj bi se nahajala v najbolj zunanji programski ravnini, tako da je vselej vidna iz točke napake. Številko napake je pri tem mogoče dobiti s klicem chin(13), ko imamo npr.

```
zlom: i:=chin(13);
      IF i>30 THEN GOTO cpmbug ELSE ...
```

9.2. Številke napak v izvajalnem času

Imamo tale seznam:

0 napaka je nedefinirana; ta napaka nima ustreznega vstopa v seznamu napak; ta napaka se lahko pojavi zaradi modifikacije sistema izvajalnega časa, pri čemer seznam napak ni bil dopolnjen
1 prostor za spremenljivke je prekoračen (prestop sklada); to je lahko posledica rekurzije ali preobsežnih poljskih deklaracij; zasledovanje napake se lahko pri teh pogojih izjalovi; prestop sklada se preverja z bločnim ali procedurnim vstopom in s poljskimi deklaracijami
2 procedura je bila poklicana z napačnim številom parametrov
3 procedura je bila poklicana z neujemanjem dejanskih in formalnih parametrov
4 uporabljeno je polje z napačnim številom indeksov
5 poljski indeksi so izven meja (pod spodnjo mejo)
6 poljski indeksi so izven meja (nad zgornjo mejo)
7 celo število je bilo deljeno z ničlo
8 realno število je bilo deljeno z ničlo
9 prestop realnega številskega obsega
10 prestop pri pretvorbi realnega v celo število
11 realni prestop pri normalizaciji po izvedbi realne aritmetične operacije
12 napaka je nastala pri branju; včitani znak ni zakonito število (ASCII vrednost je manjša od 48, tj. manjša kot &0)
13 podobno kot pri 12, le da je ASCII vrednost večja od 57 (tj. večja kot &9)
14 napaka je nastala pri branju; število vsebuje dve ali več decimalnih pik (vejic)
15 napaka je nastala pri branju; bil je najden znak +, -, '.' ali E brez potrebnih številk
16 to je kvadratni koren negativnega števila
17 eksponenčni argument je prevelik (večji kot 87)
18 ekponenčni argument je marginalno prevelik
19 to je logaritem negativnega števila
20 to je logaritem ničle
21 tabelni člen je izven območja (pod njim); ta napaka se pojavi pri ioc(n), chin(n), chout(n,c) itd., ko je $n < 0$
22 podobno kot pri napaki 21, ko je n večje od maksimalne vrednosti, določene v seznamu
23 konec zbirke je bil zaznan med branjem

9.3. Napake v nalagalniku

24 to je sintaksna nalagalniška napaka; izhod iz prevajalnika je pokvarjen
25 zaznan je bil konec vhoda (prebran znak 'CTL-z'), toda program ni bil naložen; tudi izbira vhodne naprave 0 povzroči to napako
26 po končanem vhodu je ostala nerazrešena vnaprejšnja navedba; bržkone je pokvarjen vhodni vir
27 ni programa pri ponovnem izvajanju
28 označitvene tabele prekrivajo program; program je za razpoložljivi pomnilnik preobsežen
29 tabele za vnaprejšnje navedbe so polne; ta napaka se le redko pojavi, izognemo pa se ji s preureditvijo procedur, tako da se generira manj vnaprejšnjih navedb, npr. ko so procedure deklarirane pred njihovimi klici
30 nepremestljiva pomnilniška vhodna zbirka ni združljiva s sistemom časa izvajanja

9.4. Napake CP/M sistema

31 številka kanala je izven območja
32 med izhodom ni bil najden imeniški prostor
33 poskus kanalskega branja ni odprt za vhod
34 imamo poskus branja iz neserijskega kanala
35 imamo poskus branja za zbirčnim koncem
36 imamo poskus pisanja v kanal brez zapisnega dostopa
37 imamo poskus pisanja v neserijsko zbirko
38 napaka je pri razširitvi zbirke
39 imamo poskus izhoda v zbirko z naključnim dostopom brez zapisnega dostopa
40 imamo poskus naključnega dostopa v serijsko dostopni kanal
41 kanal ni odprt
42 imamo poskus previtja zbirke z naključnim dostopom
43 imamo naključni dostop z negativno bločno številko
44 ni razpoložljive poti za vhod ali izhod
45 imamo poskus oblikovanja izhodne zbirke z naključnim dostopom
46 prenos z naključnim dostopom se opravlja s številom blokov, ki je manjše od nič in večje kot 255

10. Sklep

V tem članku smo pokazali vrsto klasičnih programov v jeziku Algol in z njimi tudi posebno prožnost tega jezika. Opisali smo podrobneje tudi sistem javljanja napak in sistem kazalcev in tako razjasnili do določene stopnje zgradbo algolskega prevajalnika. Opisani algolski primeri so seveda dobro ali slabo neposredno prevedljivi v druge programirne jezike; ti primeri so bili izbrani in so jezikovno optimizirani.

DODATEK 6. PRIMERI PROGRAMOV RACIONALNE INTERPOLACIJE Z UPORABO VERIŽNIH ULOMKOV, DOLOČITVE KORENA Z METODO DELJENJA INTERVALA, ERATOSTENOVEGA SITA, IZRAČUNA BINOMSKEGA KOEFICIENTA, DETERMINANTE IN OBRNITVE MATRIKE

V tem dodatku bomo obravnavali primere, omenjene v naslovu dodatka. Praktično bomo spoznali nekatere konstrukte jezika Algol.

6/1. Racionalna interpolacija z uporabo verižnih ulomkov

Procedura confr (okrajšava za continued fraction) v listi 6/1 določa k danim 'm' točkam (x_i , y_i) verižni ulomek oblike

$$a_1 + \cfrac{x - x_1}{a_2 + \cfrac{x - x_2}{a_3 + \ldots \cfrac{}{\ldots + \cfrac{x - x_{m-1}}{a_m}}}}$$

Procedura *confr določi* k temu verižnemu ulomku še racionalno funkcijo

Lista 6/1. Ta lista prikazuje program za izračun koeficientov verižnega ulomka in ekvivalentne racionalne funkcije, če so dani vhodni (npr. eksperimentalni) pari točk. Osrednji del tega programa je procedura confr, ki ima vhodne parametre m (število vhodnih točk), vhodne pare (x, y) (polji x in y) in izhodne koeficiente (polja) a (koeficienti verižnega ulomka), b in d (koeficienti racionalne funkcije). Program vključi v svoje telo zbirko LIB11.ALG (prikazano v listi 6/2), preostali del programa pa omogoči vnos in verifikacijo z odmevom vhodnih parov in izpiše rezultate v obliki razpredelnic (glej listo 6/3). Parameter dev se inicializira dvakrat (po želji) za vnos (konzola) in izpis rezultatov (tiskalnik).

```
BEGIN
COMMENT   Racionalna interpolacija z uporabo
          veriznih ulomkov

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

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

  PROCEDURE confr(m,x,y,a,b,d);

    VALUE m; INTEGER m; ARRAY x,y,a,b,d;

  BEGIN
    REAL r,s; INTEGER i,j,k,m1;
    ARRAY p,q[0:m%2];
    FOR j:=1 STEP 1 UNTIL m DO
      BEGIN
        r:=y[j]; s:=x[j];
        FOR i:=1 STEP 1 UNTIL j-1 DO
          r:=(s-x[i])/(r-a[i]);
        a[j]:=r
      END j loop;
    p[0]:=1; q[0]:=a[1]; m1:=m%2; k:=1;

  mult:
    FOR i:=1 STEP 1 UNTIL m1 DO
      BEGIN p[i]:=0; q[i]:=0 END;
    FOR i:=2 STEP 1 UNTIL m DO
      BEGIN
        FOR j:=i%2 STEP -1 UNTIL 1 DO
          BEGIN
            r:=a[i]*q[j]-x[i-1]*p[j]+p[j-1];
            p[j]:=q[j]; q[j]:=r
          END j loop;
        r:=a[i]*q[0]-x[i-1]*p[0]; p[0]:=q[0];
        q[0]:=r
      END i loop;
    IF k=2 THEN GOTO fin;
    FOR i:=0 STEP 1 UNTIL m1 DO b[i]:=q[i];
    k:=2; p[0]:=0; q[0]:=1; GOTO mult;

  fin:FOR i:=0 STEP 1 UNTIL m1 DO d[i]:=q[i]

  END confr
*********************************************;

  LIBRARY "LIB11.ALG"

  INTEGER n,dev,i,k;
  ARRAY x,y[1:20], a,b,d[0:20];

  n:=20;
  dev:=4; COMMENT Konzolni vhod in izhod;
  FOR i:=0 STEP 1 UNTIL n DO
    BEGIN
      a[i]:=0; b[i]:=0; d[i]:=0
    END i loop;
  typein(dev,n,x,y);

  dev:=4; COMMENT Izhod na tiskalnik;
  line (dev,27); skip(dev);
  text(dev,"i");
  space(dev,6); text(dev,"x[i]");
  space(dev,9); text(dev,"y[i]");
  line(dev,27);
  FOR i:=1 STEP 1 UNTIL n DO
    BEGIN
      skip(dev); write(dev,i);
      rwrite(dev,x[i]); rwrite(dev,y[i]);
    END i loop;
  line(dev,27); skip(dev); skip(dev);

  confr(n,x,y,a,b,d);
  line(dev,40); skip(dev);
  text(dev,"k");
  space(dev,6); text(dev,"a[k]");
  space(dev,9); text(dev,"b[k]");
  space(dev,9); text(dev,"d[k]");
  line(dev,40);
  FOR k:=0 STEP 1 UNTIL n DO
    BEGIN
      skip(dev); write(dev,k);
      rwrite(dev,a[k]); rwrite(dev,b[k]);
      rwrite(dev,d[k])
    END k loop;
  line(dev,40); skip(dev); skip(dev)
END confr
FINISH
```

$$\frac{b_0 + b_1x + \ldots + b_{deg}x^{deg}}{d_0 + d_1x + \ldots + d_{deg}x^{deg}}$$

kjer deg ni večje od m%2 (% označuje v RML Algolu celoštevilsko deljenje).

Procedura confr v listi 6/1 se s programom te liste preizkuša. Tej proceduri sledi knjižnični klic zbirke LIB11.ALG, ki je izpisana v listi 6/2. Na začetku izvajalnega dela programa v listi 6/1 se inicializirajo spremenljivke n, dev, a, b in d. Inicializaciji sledi klic procedure typein (iz knjižnice LIB11.ALG) za vpis in odmev vhodnih podatkov (x, y). Naslednji segment (med dev := 4; in line(dev,27); izpiše tabelo vhodnih podatkov (i, x, y). Na koncu programa liste 6/1 imamo segment za izpis tabele izhodnih podatkov (k, a[k], b[k], d[k]). S to tabelo so določeni koeficienti verižnega ulomka in racionalne funkcije.

Lista 6/2. Lista (spodaj) prikazuje deklaracije treh procedur, in sicer za izpis spremenljivo dolge črte (procedura line, dolžina črte 'n'), za izpis 'n' presledkov (space) in proceduro typein za vnos, odmev in verifikacijo (potrjevanje vnosa) podatkov. Procedura izstopi z dejanskim številom 'n' vhodnih parov (ta parameter je klican z naslovom).

```
PROCEDURE line(dev,n);
  VALUE dev,n; INTEGER dev,n;

  BEGIN
    INTEGER i;
    skip(dev);
    FOR i:=1 STEP 1 UNTIL n DO
      text(dev,"-")
  END line
-------------------------------------------------;

PROCEDURE space(dev,n);
  VALUE dev,n; INTEGER dev,n;

  BEGIN
    INTEGER i;
    FOR i:=1 STEP 1 UNTIL n DO
      text(dev," ")
  END space
-------------------------------------------------;

PROCEDURE typein(dev,n,x,y);
  VALUE dev; INTEGER dev,n;
  ARRAY x,y;

  BEGIN
    INTEGER i,a;
    FOR i:=1 STEP 1 UNTIL n DO
      BEGIN
retype: skip(dev); write(dev,i);
        x[i]:=read(dev); rwrite(dev,x[i]);
        y[i]:=read(dev); rwrite(dev,y[i]);
        text(dev,"   OK ? ");
        a:=chin(dev);
        IF a=&e OR a=&E THEN GOTO exit
        ELSE
          IF a=&y OR a=&Y THEN
          ELSE GOTO retype
      END i loop;
      n:=i-1; GOTO out;
exit: n:=i;
out:
  END typein
-------------------------------------------------;
```

Lista 6/3. V listi (spodaj) imamo vhodni protokol (odmev in potrjevanje), razpredelnico vhodnih parov (x, y) in razpredelnico koeficientov verižnega ulomka (a) in racionalne funkcije (b, d). Določeni koeficienti imajo vrednost 0.

```
 1 50   5.00000E 01 -40 -4.00000E 01     OK ?  y
 2 75   7.50000E 01 60  6.00000E 01     OK ?  y
 3 65   6.50000E 01 50  5.00000E 01     OK ?  y
 4 80   8.00000E 01 65  6.50000E 01     OK ?  y
 5 95   9.50000E 01 -35 -3.50000E 01     OK ?  e
-----------------------------
 i       x[i]          y[i]
-----------------------------
 1   5.00000E 01 -4.00000E 01
 2   7.50000E 01  6.00000E 01
 3   6.50000E 01  5.00000E 01
 4   8.00000E 01  6.50000E 01
 5   9.50000E 01 -3.50000E 01
-----------------------------


-------------------------------------------------
 k       a[k]          b[k]          d[k]
-------------------------------------------------
 0   0.00000E 00   3.24803E 05   4.34638E 03
 1  -4.00000E 01  -9.62969E 03  -1.39928E 02
 2   2.50000E-01   6.50723E 01   1.00000E 00
 3   1.20000E 02   0.00000E 00   0.00000E 00
 4   7.49995E-01   0.00000E 00   0.00000E 00
 5  -1.49276E 01   0.00000E 00   0.00000E 00
-------------------------------------------------
```

Lista 6/2 (zbirka LIB11.ALG) vsebuje tri procedure, in sicer proceduro za izpis črte (line), proceduro za izpis presledkov (space) in proceduro za vnos in odmev vhodnih podatkov (typein).

Lista 6/3 ima tri dele: interaktivni vhodni protokol za vnos, odmev in potrditev podatkov, vhodno in izhodno razpredelnico. Ko začnemo izvajati program, se v levem delu izpiše indeks 1, nato vtipkamo podatek 50 in presledek, nakar se pojavi odmev 5.00000E 01 itd. Če kakšen par (x, y) ne potrdimo, se indeks in z njim povezan vnos ponovi z novimi podatki. Tako dobimo celotno vhodno interakcijo. Ko izstopimo iz procedure typein s potrditvijo 'e' (exit), se izpišeta vhodna in izhodna razpredelnica. Program v listi 6/1 smo v našem primeru preizkusili za vhodne podatke:

i	1	2	3	4	5
x[i]	50	75	65	80	95
y[i]	-40	60	50	65	-35

Za ta primer imamo verižni ulomek

$$y = -40 + \cfrac{x - 50}{0{,}25 + \cfrac{x - 75}{120 + \cfrac{x - 65}{0{,}749995 + \cfrac{x - 80}{-14{,}9276}}}}$$

in ekvivalentno racionalno funkcijo

$$y = \frac{324803 - 9629{,}69x + 65{,}0723x^2}{4346{,}38 - 139{,}928x + x^2}$$

Posebna pozornost velja segmentu

```
FOR i:=1 STEP 1 UNTIL j-1 DO
  r:=(s-x[i])/(r-a[i]);
```

ker se tu pri danih pogojih lahko pojavi prestop območja (deljenje z ničlo). V tem primeru lahko spremenimo vrstni red dvojic (x, y) ali pa spremenimo merilo.

6/2. Določitev korena funkcije z metodo deljenja intervala na polovico

Procedura bisec v listi 6/4 izračuna najprej vrednost funkcije na mejah realnega intervala in izda rezultat (oznako) SIGNAL, če sta predznaka mejnih funkcijskih vrednosti enaka. V nasprotnem primeru (pri različnih predznakih) pa poišče procedura koren funkcije z metodo intervalnega deljenja na polovico in z izračunom vrednosti funkcije v razpolovni točki. Procedura se konča, ko je vrednost funkcije manjša od vnaprej dane vrednosti eps ali ko je razlika dveh zaporedno najdenih korenskih približkov manjša od eps1. Vrednost eps naj bi bila izbrana sorazmerno k pogrešku izračuna funkcije (ker bi sicer lahko nastopilo cikliranje), vrednost eps1 pa naj bi bila sorazmerna potrebi natančnosti korena. Vrednost eps1 naj ne bi bila manjša od dveh enot zadnjega velikostnega razreda prevajalniške aritmetike, ker se sicer zaradi zaokrožanja lahko pojavi cikliranje pri deljenju intervala.

Opisana metoda je uporabljiva za poljubno zvezno funkcijo. Procedura ne uporablja preverja-

Lista 6/4.

```
BEGIN
COMMENT   Dolocitev korena z metodo deljenja
          intervala
-----------------------------------------------

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

  PROCEDURE bisec(a,b,eps,eps1,f,signal,x);
    VALUE a,b; REAL a,b,eps,eps1,x;
    REAL PROCEDURE f; LABEL signal;

    BEGIN
      REAL y,z;
      PROCEDURE fun(y);
        REAL y;
        BEGIN
          y:=f(x);
          IF abs(y)<=eps THEN GOTO fin
        END;
      x:=a; fun(y);
      z:=b; fun(z);
      IF sign(y)=sign(z) THEN GOTO signal;
  iter:
      x:=a/2+b/2; fun(y);
      IF sign(y)=sign(z) THEN b:=x ELSE a:=x;
      IF abs(a-b)>=eps1 THEN GOTO iter;
  fin:
    END bisec
**********************************************;

  LIBRARY "LIB12.ALG"

  INTEGER dev,i,n;
  ARRAY x1,x2,x3,x4,x5[1:20];

  dev:=4; n:=20;
  typin(dev,n,x1,x2,x3,x4,x5)

END program
FINISH
```

```
PROCEDURE line(dev,n);
  VALUE dev,n; INTEGER dev,n;

  BEGIN
    INTEGER i;
    skip(dev);
    FOR i:=1 STEP 1 UNTIL n DO
      text(dev,"-")
  END line
-----------------------------------------------

PROCEDURE space(dev,n);
  VALUE dev,n; INTEGER dev,n;

  BEGIN
    INTEGER i;
    FOR i:=1 STEP 1 UNTIL n DO
      text(dev," ")
  END space
-----------------------------------------------

PROCEDURE typin(dev,n,x1,x2,x3,x4,x5);
  VALUE dev; INTEGER dev,n;
  ARRAY x1,x2,x3,x4,x5;

  BEGIN
    INTEGER i,a,g; REAL z1,z2,z3,z4,z5;
    g:=0;
    line(dev,47);
  text(dev,"*NVtipkaj podatke v vrstnem redu");
  text(dev,"*N*N"); space(dev,15);
  text(dev,"a        b        eps        eps1");
    line(dev,47);
    FOR i:=1 STEP 1 UNTIL n DO
      BEGIN
retype:
        skip(dev); write(dev,i);
        x1[i]:=read(dev); rwrite(dev,x1[i]);
        x2[i]:=read(dev); rwrite(dev,x2[i]);
        skip(dev); space(dev,3);
        x3[i]:=read(dev); rwrite(dev,x3[i]);
        x4[i]:=read(dev); rwrite(dev,x4[i]);
        text(dev," OK? ");
        a:=chin(dev);
        IF a=&e OR a=&E THEN
          BEGIN g:=1; GOTO exit1 END;
        IF a#&y AND a#&Y THEN GOTO retype;
exit1:
        text(dev,"*N   Rezultat (koren):  ");
        z1:=x1[i]; z2:=x2[i]; z3:=x3[i];
            z4:=x4[i];
        bisec(z1,z2,z3,z4,
            REAL PROCEDURE func1,
                LABEL signal,z5);
        x5[i]:=z5;
        rwrite(dev,x5[i]);
        IF g=1 THEN GOTO exit;
        GOTO endlp;
signal: text(dev,"SIGNAL");
endlp:END i loop;
exit:
    END typin
-----------------------------------------------

  REAL PROCEDURE func1(x);
    REAL x;
    func1:=x*x-2;
```

Lista 6/5. Ta lista vsebuje deklaracije štirih procedur, in sicer line, space, typin in func1. Procedura func1(x) predstavlja funkcijo za naš primer (x*x-2). Ta funkcija ima dva korena. V proceduri typin pokličemo proceduro bisec z dejanskimi parametri, pri čemer se morata funkcijski in oznakovni dejanski parameter (zaradi enoprehodnega prevajalnika) poklicati z REAL PROCEDURE func1 in z LABEL signal. Oznaka signal se nahaja na koncu procedure typin, ko se izpiše sporočilo SIGNAL. Procedura typin shranjuje vhodne (a, b, eps, eps1) vrednosti in rezultat v poljih x1, x2, x3, x4, x5.

nja odvodljivosti funkcije in izračuna realni koren neposredno. Poljubni realni spremenljivki a in b sta intervalni meji in pri funkcijskih vrednostih f(a) in f(b) z nasprotnima predznakoma se v intervalu (a,b) nahaja liho število korenov.

V programu liste 6/4 sledi proceduri bisec knjižnični (zbirčni) klic LIB12.ALG; program v tej zbirki je prikazan v listi 6/5. Procedura typin, ki je deklarirana v okviru zbirke LIB11.ALG, sprejema vhodne podatke in izračunava rezultate.

Lista 6/5 prikazuje deklaracije treh procedur, pri čemer sta prvi dve proceduri že znani iz prejšnjega primera (poglavje 6/1). Procedura typin je zasnovana širše, kot je uporabljena v programu liste 6/4. Njena polja x1, ... , x5 bi bilo moč uporabiti kasneje za izpis urejene tabele vhodnih podatkov in rezultatov. Tako pa rutina typin generira vhodni in rezultatni protokol, ki je prikazan v listi 6/6.

Lista 6/6 kaže vhodni/rezultatni protokol, ko se najprej izpiše navodilo za vnos podatkov, nakar se pod črto pojavi indeks 1; tu lahko začnemo z vnosom vhodnih podatkov (a, b, eps, eps1). Med vnosom posameznih podatkov uporabljamo presledek kot končno znamenje, tako da se protokol izpisuje nazorno. Po potrditvi (y ali e) se izračuna rezultat ali pa izpiše oznaka SIGNAL v primeru sodega števila korenov v danem intervalu (0 je sodo število).

```
RML ALGOL V4.8C
Copyright (C) 1980 by Research Machines

^

-----------------------------------------------
Vtipkaj podatke v vrstnem redu

                a         b         eps       eps1
-----------------------------------------------
1 -3 -3.00000E 00 -1 -1.00000E 00
   .001  9.99999E-04 .0001  9.99998E-06  OK? y
   Rezultat (koren):  -1.41406E 00
2 0  0.00000E 00 4  4.00000E 00
   .000001  9.99998E-07 .000001  9.99998E-07  OK? y
   Rezultat (koren):   1.41421E 00
3 -2 -2.00000E 00 2  2.00000E 00
   .001  9.99999E-04 .001  9.99999E-04  OK? y
   Rezultat (koren):  SIGNAL
4 1.3  1.30000E 00 1.5  1.50000E 00
   .000001  9.99998E-07 .000001  9.99998E-07  OK? e
   Rezultat (koren):   1.41421E 00
^
^C
```

Lista 6/6. Ta lista prikazuje vhodni protokol z rezultati, ko se išče koren funkcije x*x-2. Na prvem mestu dvojne vrstice se najprej izpiše zaporedni indeks, temu sledijo a, b, eps in eps1. V tretji vrstici se izpiše rezultat ali pa oznaka SIGNAL. Iz protokola je razvidno vsakokratno potrjevanje z izstopom iz programa.

6/3. Dodatni primeri

V nadaljevanju bomo opisali še štiri algolske procedure, in sicer

- proceduro za izračun praštevil z uporabo Eratostenovega sita
- proceduro za izračun binomskih koeficientov z rekurzivno formulo
- proceduro za izračun determinante in
- proceduro za obrnitev matrike

Lista 6/7. Eratostenovo sito se uporablja za preizkušanje zmogljivosti različnih prevajalnikov; lahko pa ga uporabimo tudi za izračun zaporedja praštevil, ki se shranijo v polju 'p', odkoder jih lahko uporabimo (izlistamo)

```
COMMENT  Procedura za izracun praštevil
         z uporabo Eratostenovega sita

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

  PROCEDURE sieve(n,p);
    VALUE n; INTEGER n; INTEGER ARRAY p;

    BEGIN
      INTEGER i,j,k;
      p[1]:=1; p[2]:=2; p[3]:=3; j:=3;
      FOR k:=3 STEP 2 UNTIL n DO
        BEGIN
          i:=2;
start:    i:=i+1;
          IF p[i] ) sqrt(k) THEN
            BEGIN p[j]:=k; j:=j+1 END
          ELSE
            IF (k%p[i])*p[i]#k THEN
              GOTO start
        END k
    END sieve
*******************************************;
```

Preizkus teh procedur v okviru določenih programov je prepuščen bralcu.

Procedura sieve(n,p) v listi 6/7 predstavlja algoritem, znan kot Eratostenovo sito in s katerim določamo praštevila (vključno z enico), in sicer do dane meje 'n', dobljena praštevila pa shranjujemo v polju 'p', katerega obseg (število elementov) ni večji kot

$$\text{entier}((n/\ln n)+1)$$

Ta procedura se uporablja tudi za ocenjevanje zmogljivosti različnih prevajalnikov.

Procedura c(n,m) v listi 6/8 izračunava binomske koeficiente

$$C(m,n) = n!/(m!*(n-m)!)$$

z uporabo rekurzivne formule

$$C(i+1,n) = ((n-i)/(i-1))*C(i,n)$$
$$\text{pri } i = 0,1,\ldots,n-1$$

```
COMMENT  Procedura za izracun binomskih koefi-
         cientov z rekurzivno formulo

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

  INTEGER PROCEDURE c(n,m);
    VALUE n,m; INTEGER n,m;

    BEGIN
      INTEGER i,a;
      a:=1;
      IF m)n THEN m:=n-m;
      FOR i:=0 STEP 1 UNTIL m-1 DO
        a:=a*(n-i)/(i+1);
      c:=a
    END c
*******************************************;
```

Lista 6/8. Ta procedura izračuna binomski koeficient pri danih dveh parametrih

Lista 6/9. Procedura v spodnji listi se lahko uporabi za izračun determinante kvadratne matrike ´a´ stopnje ´n´. Proceduro preizkusimo tako, da jo ustrezno vključimo v dani program pri različnih primerih matrik oziroma vrednosti njihovih elementov.

```
COMMENT    Procedura za izracun determinante
*******************************************;
  REAL PROCEDURE det(a,n);
    VALUE a,n; INTEGER n; ARRAY a;

    BEGIN
      REAL t,d,max; INTEGER i,j,k;
      d:=1;
      FOR k:=1 STEP 1 UNTIL n DO
        BEGIN
          max:=0;
          FOR i:=k STEP 1 UNTIL n DO
            BEGIN
              t:=a[i,k];
              IF abs(t) > abs(max) THEN
                BEGIN max:=t; j:=i END
            END loop i;
          IF max=0 THEN
            BEGIN d:=0; GOTO fin END;
          IF j#k THEN
            BEGIN
              d:=-d;
              FOR i:=k STEP 1 UNTIL n DO
                BEGIN
                  t:=a[j,i]; a[j,i]:=a[k,i];
                  a[k,i]:=t
                END
            END;
          FOR i:=k+1 STEP 1 UNTIL n DO
            BEGIN
              t:=a[i,k]/max;
              FOR j:=k+1 STEP 1 UNTIL n DO
                a[i,j]:=a[i,j]-t*a[k,j]
            END loop i;
          d:=d*a[k,k]
        END loop k;
fin:det:=d
  END det
*******************************************;
```

kjer je

$$C(0,1) = 1$$

Za preizkus imamo npr. rezultat

$$C(12,18) = 18564$$

Procedura det(a,n) v listi 6/9 izračuna determinanto matrike ´a´ velikosti n*n s triangularizacijo. Za preizkus lahko uporabimo primer

$$a = \begin{bmatrix} 10{,}96597 & 35{,}10765 & 96{,}72356 \\ 2{,}35765 & -84{,}11256 & 0{,}89732 \\ 18{,}24689 & 22{,}13579 & 1{,}11123 \end{bmatrix}$$

ko imamo

$$det = 152731{,}3$$

Procedura invert(matr,n,matr1,s) v listi 6/10 obrne kvadratno matriko matr stopnje ´n´ z uporabo več elementarnih operacij nad vrsticami matrike matr. Primer izrojene matrike je označen z vrednostjo s = 1.

Lista 6/10. Ta procedura obrne dano kvadratno matriko ´matr´ stopnje ´n´ in jo izda v obliki matrike ´matr1´. Posebnost te procedure je polje ´a´ s spremenljivo spodnjo in zgornjo mejo.

```
COMMENT      Procedura za obrnitev matrike
*******************************************;

  PROCEDURE invert(matr,n,matr1,s);
    VALUE n; INTEGER n,s; ARRAY matr,matr1;

    BEGIN
      REAL t; INTEGER i,j,k,m;
      ARRAY a[1:n,1:2*n];
      m:=2*n; s:=0;
      FOR i:=1 STEP 1 UNTIL n DO
        FOR j:=1 STEP 1 UNTIL m DO
          a[i,j]:=IF j<=n THEN matr[i,j] ELSE
            IF j=n+i THEN 1.0 ELSE 0.0;
      COMMENT  Zacetek obrnitve;
      FOR i:=1 STEP 1 UNTIL n DO
        BEGIN
          k:=i;
test0:    IF a[k,i]=0 THEN
            BEGIN
              s:=1;
              IF k<n THEN k:=k+1 ELSE GOTO fin;
              GOTO test0
            END;
          IF s=1 THEN
            FOR j:=1 STEP 1 UNTIL m DO
              BEGIN
                t:=a[k,j]; a[k,j]:=a[i,j];
                a[i,j]:=t
              END loop j;
          FOR j:=m STEP -1 UNTIL i DO
            a[i,j]:=a[i,j]/a[i,i];
          FOR k:=1 STEP 1 UNTIL n DO
            IF k#i THEN
              FOR j:=m STEP -1 UNTIL i DO
                a[k,j]:=a[k,j]-a[i,j]*a[k,i]
        END loop i;
        FOR i:=1 STEP 1 UNTIL n DO
          FOR j:=1 STEP 1 UNTIL n DO
            matr1[i,j]:=a[i,j+n];
        s:=0;
fin:END invert
*******************************************;
```

Slovstvo k tretjemu delu

((19)) G.Botenbruh: Struktura Algol-60 i ego ispol´zovanie. Izd.inostr.lit., Moskva 1963.

((20)) V.V.Stager: Čebyševskie približenija primenjaemye v rasčetah električeskih shem. Svjazizdat, Moskva 1960.

((21)) V.I.Krylov: Približennoe vyčislenie integralov. Fizmatgiz, Moskva 1959.

((22)) Algol Bulletin. Časopis IFIP WG 2.1. Naslov uredništva: Algol Bulletin, Dept. of Computer Science, University of Manchester, Manchester M13 9PL, UK.

((23)) A.P.Železnikar: Prevajalniki. Skripta (ponatis), Fakulteta za elektrotehniko, Ljubljana 1980.

TESTIRANJE CMOS KOMBINACIJSKIH VEZIJ

R. MURN,
D. PEČEK,
B. KASTELIC

UDK: 681.3.325.6.08

INSTITUT JOŽEF STEFAN, UNIVERZA EDVARDA KARDELJA, LJUBLJANA

Izkaže se,da je klasičen model stalnih napak nezadosten za opis nekaterih fizičnih napak pri CMOS digitalnih integriranih vezjih.Pri CMOS elementih se pojavi nova ne klasična napaka,imenovana "stalno-odprt"(SOD).Ta napaka povzroči,da se kombinacijski element obnaša kot sekvenčni element. V članku je opisan model napak ter postopek za detekcijo teh napak.Prav tako je podan postopek za detekcijo napak v kombinacijskem vezju,le-ta je osnovan na že znanih načelih občutljivih poti in grafa povezav razredov napak.Obravnavana je tudi problematika večkratnih napak in testiranje redundantnih rekonvergenčnih vezij.

FAULT TESTING IN CMOS COMBINATIONAL NETWORKS.The classical stuck-at fault model does not adequately define certain physical failures in CMOS digital integrated circuits.A new nonclassical logical fault has been found in CMOS circuits: the "stuck-open".Such a fault alter the combinational behavior of CMOS logic gates into a sequential one.The paper describes the modeling of this fault and develops a procedure to detect them.The test procedure for combinational networcs is generated based on the path sensitizing concept and connectivity graph of fault classes.Multiple faults and redundant reconvergent networcs are also discussed in the paper.

I. UVOD

Vzporedno z naraščujočim razvojem integriranih vezij v MOS in CMOS tehnologiji se pojavlja vprašanje preiskušanja takih izvedb. CMOS integrirana vezja visoke in zelo visoke integracije se vedno bolj uveljavljajo na tržišču, predvsem zaradi nekaterih dobrih lastnosti, kot so visoka gostota pakiranja in majhna poraba energije /1/. Preiskušanje vezij zahteva dobro poznavanje obnašanja funkcij le-teh ob pojavitvah napak. Izkaže se da je potrebno pri CMOS vezjih standardni klasični model stalnih napak (stuck - at) dopolniti.Nekateri avtorji /2/ imenujejo te dodatne napake "stalno - odprt" (stuck - open), podobno kot pri vezjih s tremi stanji. V prisotnosti imenovanih napak se lahko obnaša logični element kot pomnilniški element, tako postane kombinacijski logični element sekvenčni element. Omenjeno je vzrok, da vezja, ki so sicer kombinacijska, ne moremo preoiskušati s klasičnimi postopki za preiskušanje kombinacijskih vezij.

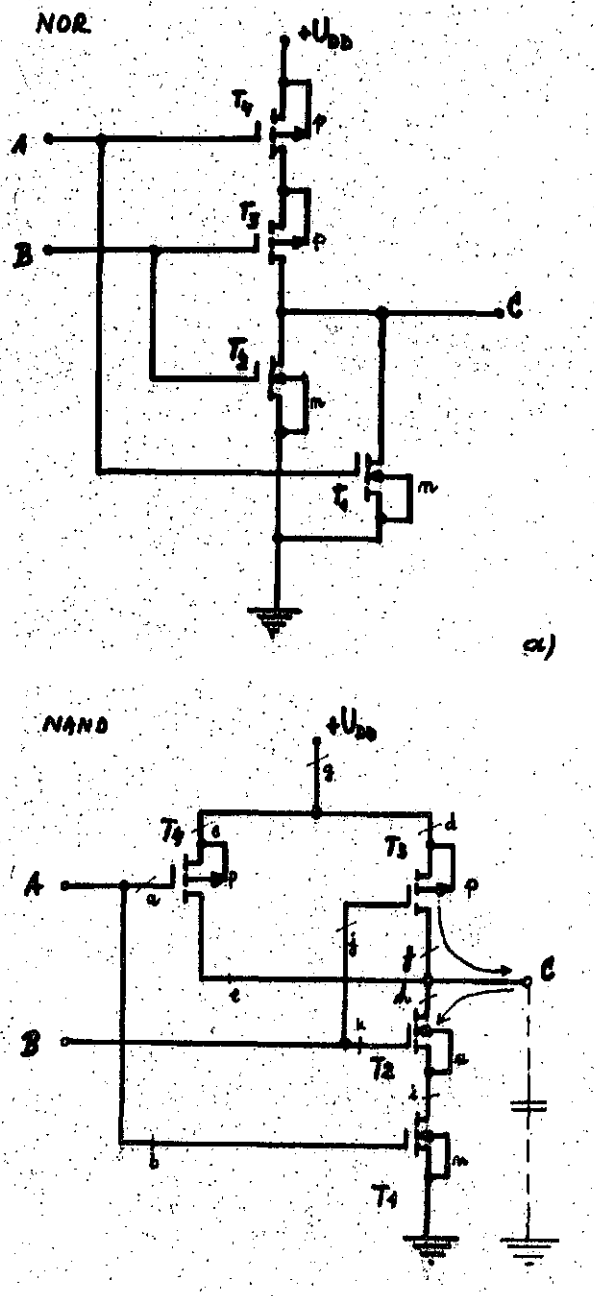
V naslednjih poglavjih bomo analizirali model napak in prikazali učinke napak na NAND in NOR vezju. Podali bomo ustrezno zaporedje preiskuševalnih naborov za detekcijo dodatnih napak, ter tvorili preiskuševalne vhodne nabore v luči že znanih postopkov za kombinacijska vezja in klasičnih napak. Diagnostične teste za kombinacijska vezja bomo analizirali s pomočjo grafov povezanosti napak ter s pomočjo algoritmov za širjenje napak v občutljivih poteh. Na koncu dela bomo komentirali problematiko testiranja rekonvergenčnih kombinacijskih vezij ter večkratnih napak.

II. MODEL NAPAK

Za predstavitev napak si poglejmo dvo vhodna CMOS NOR ($\overline{V}$) in NAND ($\overline{\wedge}$) vrata, slika 1. V klasičnem smislu razlikujemo dve vrsti napak: kaka vhodna ali izhodna linija je stalno v 0 ali stalno v 1, kar označimo s S0 ali S1, linija i je stalno v 0: (i-0) ali stalno v 1: (i-1). Pri tem ločimo n+2 različnih napak, če imajo vrata n vhodov. Značilna napaka za posamezne vrste logičnih vrat je t.i.odločilna napaka (prevladujoča) in je za NOR vrata S1 in za NAND vrata S0.

Kot se vidi na sliki 1 sestavljajo CMOS logična vrata dva p in dva n - tranzistorja. V primeru, da je kateri od teh (ali oba) v paralelni sestavi odprt, pridobi izhodni napetostni nivo logičnih vrat potencial mase ali +Udd, tistega pač, ki pripada odprti smeri. Podobno je s tranzistorjema v serijski vezavi, ki pa morata biti oba odprta. Pri normalnem delovanju dobimo na izhodni liniji logično vrednost 0 ali 1. Primer, ko sta oba sestava (p in n) odprta, ne upoštevamo in to napako ne smatramo za logično napako. Posebno zanimiv je primer, ko so tranzistorji v obeh sestavih zaprti. Razmere si bomo v tem primeru ogledali bolj natančno in zaradi enostavnosti samo za NAND vrata.

Ko je na enem od obeh vhodov logična 0, je paralelni sestav (p-tranzistorji) prevoden, serijska pot (n - tranzistorji) pa zaprta. Na izhodu imamo logično vrednost 1 in polnjenje izhodne kapacitivnosti. Ko pritisnemo na vhoda A in B logično vrednost 1 se razmere obrnejo. Serijski sestav tranzistorjev se odpre, paralelni pa zapre. Na izhodu imamo logično vrednost 0, kapacitivnost se prazni skozi serijski sestav.



Slika 1: CMOS NOR in NAND vrata

Vzemimo, da je na vhodu (AB) nabor 10. Serijski sestav je saprt, p-tranzistor, ki je priključen na vhod A je saprt, tranzistor, ki je priključen na vhod B pa odprt. V normalnem stanju (brez napak) imamo na izhodu logično vrednost 1. V primeru, da se pojavi kaka enojna napaka npr. linija f je prekinjena, se zapre prevajanje paralelnega sestava. Izhod je v "odprtem" ali visoko impedančnem stanju. Kapacitivno breme na izhodu je prevladujoče in se ne polni ter obdrži logično vrednost predhodnega stanja. Tovrstno napako bomo označili s SOD. S pomočjo slike 1b, kjer so vnesene napake odprtih lininj in s kratkim premislekom lahko analiziramo vplive ostalih napak. Nekatere od njih imajo ekvivalenten učinek, lahko so pridanem vhodnem naboru takoj razvidne, ali pa so zamaskirane. Napake SOD, ki vplivajo na p-tranzistorja so (a, j, e, f, c, d, g) in na n-tranzistorja so (k, b, h, i). Zaradi sekvenčnega značaja obnašanja CMOS logičnega elementa ob prisotnosti SOD moramo ustrezne testne nabore primerno sekvenčno urediti.

Podobno kot pri preiskušanju sekvenčnih vezij najprej ugotovimo znano stanje. Če pogledamo enojne SOD napake, ki vplivajo na p-tranzistorja lahko ugotovimo, da dosežemo znano stanje tako, da izhodno kapacitivnost najprej izpraznimo (logična 0). To dosežemo s tem, da damo na vhod nabor 11, kar da na izhodu logično 0, ne glede na obstoj omenjene napake. Naslednji korak je polnjenje izhoda s tem, da damo na ustrezen vhod logično vrednost 0, na drugi vhod pa vrednost 1. Če se izhod ne nabije na vrednost 1, pomeni, da imamo opravka s SOD iz skupine omenjenih napak. Podobno lahko preiskusimo vhod B. Enojne napake SOD, ki vplivajo na n-tranzistorja ugotavljamo tako, da najprej nabijemo izhod (vrednost je logična 1, ne glede na napako) na logično vrednost 1 in nato v naslednjem koraku le - tega izpraznimo. V primeru, da izhod ni na logični vrednosti 0, zaznamo SOD iz omenjene skupine napak.

Iz zgornjega razmišljanja sledi naslednje testno zaporedje, ki odkrije vse enojne SOD napake v dvo vhodnem NAND vezju:

$$T_{sod} = (11, 01, 11, 10).$$

Na podoben način lahko razširimo zaporedje vhodnih naborov za NAND vrata s večjim številom vhodnih linij. Prav tako lahko določimo preiskuševalno zaporedje za CMOS NOR vrata. Za dvo vhodna NOR vrata dobimo:

$$T_{sod} = (00, 01, 00, 10).$$

III. DETEKCIJA NAPAK V KOMBINACIJSKIH VEZJIH

a.) Drevesna vezja

Najprej si bomo ogledali problematiko detekcije napak v drevesnih kombinacijskih vezjih. Drevesna vezja ne vsebujejo rekonvergenc, imajo več vhodnih linij - primarnih vhodov in eno izhodno linijo - primarni izhod.

Obravnavo detekcije napak želimo navezati na že znane postopke za klasične stalne napake, ter na pripadajoče pojme, kot so občutljive poti in grafi povezav razredov napak (GPRN) /3/.

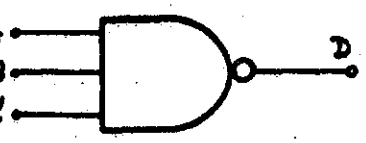
Oglejmo si najprej razmere na prvem nivoju drevesnega vezja t.j. na nivoju posameznih vrat. Zanima nas stopnja združevanja napak, le-ta povezuje napake na vhodnih in izhodnih linijah v ekvivalenčni razred napak. To nam potem pomaga pri sledenju občutljivih napak od primarnega vhoda do primarnega izhoda, ter pri določanju neločljivih napak. Logična vrata vrste AND (IN) imajo pri tem naslednje značilne vrednosti: logična vrednost 0 predstavlja odločino vrednost in napake S0 so odločilne in neločljive. Napake S1 so nevezane napake. Logična vrednost 1 predstavlja pogoj za tvorjenje kake občutljive poti. Logična vrata OR (ALI) imajo naslednje značilne vrednosti: logična vrednost 1 predstavlja odločilno vrednost in napake S1 so odločilne in neločljive. Napake S0 so nevezane napake. Logična vrednost 0 na kaki vhodni liniji predstavlja pogoj za tvorjenje občutljive poti v kaki preostali vhodni liniji.

Brez težav lahko dokažemo, da detektiramo vse enojne napake v logičnih vratih z n vhodnimi linijami s n+1 testnimi nabori. Pri tem

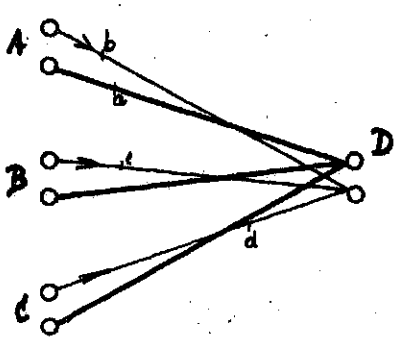
uporabimo n testnih naborov za detekcijo nevezanih napak in en nabor za detekcijo odložilnih napak. Nevezane napake so med seboj ločljive, v danem trenutku je lahko občutljiva samo ena vhodna napaka (linija).

S pomočjo GPRN lahko grafično prikažemo razrede napak in pripadajoče relacije. Vsaki liniji pripadata dva krogca, gornji predstavlja napako S1, dolnji pa napako S0. Kot primer vzemimo NAND vrata s tremi vhodi, slika 2.

a	b	c	d	
1	0	1	1	A
1	1	0	1	B
1	1	1	0	C



Slika 2a: Logični element z vhodnimi nabori

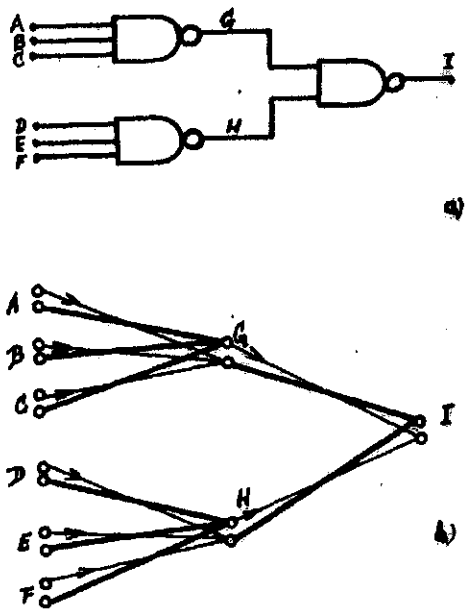


Slika 2b: Ustrezni GPRN

Iz strukture GPRN je razvidno, da testni nabori ki detektirajo vse vhodne enojne napake, detektirajo tudi vse izhodne napake. Pri drevesnem vezju se lahko brez težav prepričamo, da množica vhodnih testnih naborov, ki detektira vse enojne napake na primarnih vhodih vezja, detektira tudi vse enojne napake v celotnem drevesnem vezju. Pri drevesnem vezju so vse linije neodvisne in se lahko vedno generira vse enojne občutljive poti. Ker v teh vezjih ni rekonvergeno, ne pride do vzajemnega markiranja kake napake. Če na vsakem vhodu logičnih vrat zagotovimo testno zaporedje Tsod odkrijemo vse S0, S1 in SOD napake. Poglejmo to na sledečem vzgledu, slika 3.

Vrednosti vhodnih naborov logičnih vrat so podane v razpredelnici I.

Prvih šest testnih naborov je za SOD napake za vhode A, B, C, pri tem zagotovimo z vhodi D, E, F pogoj za občutljivo pot A, B, C, G, I, t.j. vrednost 1 na liniji H. Drugih šest testnih naborov pa za SOD napake za vhode A, B, C. Množica vhodnih testnih naborov je zadostna tudi za klasične napake in je tako ustrezni GPRN prekrit. Poglejmo, katere enojne klasične napake odkrijemo s pomočjo testnih naborov:



Slika 3: Dvo nivojsko vezje in ustrezni GPRN

VHODI	TESTNI NABORI 1	2	3	4	5	6	7	8	9	10	11	12
A	1	0	1	1	1	1	0	0	0	0	0	0
B	1	1	1	0	1	1	1	1	1	1	1	1
C	1	1	1	1	1	0	1	1	1	1	1	1
D	0	0	0	0	0	0	1	0	1	1	1	1
E	1	1	1	1	1	1	1	1	1	0	1	1
F	1	1	1	1	1	1	1	1	1	1	1	0
G	0	1	0	1	0	1	1	1	1	1	1	1
H	1	1	1	1	1	1	0	1	0	1	0	1

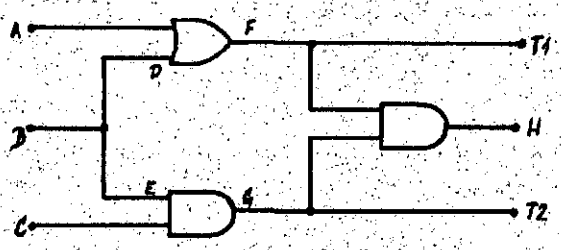
Razpredelnica I

test 1:
(1 - 0), (2 - 0), (3 - 0), (7 - 1), (9 - 0)
test 2:
(1 - 1), (4 - 1), (7 - 0), (8 - 0), (9 - 1)
test 4
(2 - 1), (4 - 1), (7 - 0), (8 - 0), (9 - 1)
test 6:
(3 - 1), (4 - 1), (7 - 0), (8- 0), (9 - 1)
test 8
(4 - 0), (5 - 0), (6 - 0), (8 - 1), (9 - 0)
test 10:
(1 - 1), (6 - 1), (7 - 0), (8 - 0), (9 - 1)
test 12:
(1 - 1), (6 - 1), (7 - 0), (8 - 0), (9 - 1)

Iz opisanega je razvidno, da potrebujemo za detekcijo klasičnih napak sedem testov, za detekcijo vseh obravnavanih napak pa dvanajst testov. Brez težav lahko postopek testiranja razširimo na drevesna vezja s večjim številom nivojev.

b) SPLOŠNA KOMBINACIJSKA VEZJA

Splošna kombinacijska vezja vsebujejo tudi razvejišča in stične točke (rekonvergenčne točke), kjer se razvejene poti zopet združijo. Vsebujejo lahko tudi več primarnih izhodov. Posledica rekonvergenc so več dimenzionalne občutljive poti, kjer se lahko zgodi, da kakih napak nemoremo detektirati. Kaka napaka je s paralelno potjo maskirana, ker le - ta predstavlja redundantno občutljivo pot. Potem lahko detektiramo napake v enojnih občutljivih poteh skozi stično točko in napake od primarnih vhodov do razvejišča in napake od stičišča do primarnega izhoda. Na sliki 4 imamo primer enostavnega rekonvergenčnega kombinacijskega vezja.



Slika 4: Enostavno rekonvergenčno vezje

Nekaj enojnih napak v tem vezju ne moremo odkriti. Oglejmo si dvodimenzionalno občutljivo pot od B do H. Ugotovimo lahko, da odkrijemo s vhodnim naborom 011 naslednje napake na tej poti:

(B - 0), (D - 0), (E - 0), (F - 0), (G - 0), (H - 0) in (C - 0).

Z vhodnim naborom 001 pa samo (B - 1) in (H - 1), kar lahko razvidimo tudi s pomočjo ustreznega GPRN.

Podobno kot pri drevesnih vezjih bomo tudi pri splošnih kombinacijskih vezjih stremeli po tem, da bo vsak logični element preiskušen z zaporedjem za SOD. Za posamezne primarne vhode in vhodne logočne elemente, bomo tvorili občutljive poti do primarnih izhodov, pri tem pa bomo pregledovali vhode posameznih logičnih elementov in zahtevano zaporedje naborov. Problematika maskiranja napak pri rekonvergenčnih poteh je pri tem enaka, kot pri klasičnih napakah.

c) VEČKRATNE NAPAKE

Pri dosedanjih izvajanjih smo dovoljevali enojne napake v kombinacijskih vezjih. V primeru večkratnih napak, so lahko postopki za detekcijo zelo kompleksni, saj je poznano, da je v kombinacijskem vezju s n povezavami lahko $3^n - 1$ večkratnih napak. Zopet si bomo pomagali s izdelki za klasične napake. Za drevesno vezje s m vhodi je dokazano /4/, da potrebujemo za detekcijo večkratnih klasičnih napak, največ m+1 vhodnih testov. Postopek za detekcijo vseh napak v drevesnih kombinacijskih vezjih je potem enak opisanemu, s tem, da predpišemo nogojnim vhodom v logična vrata maksimalno neobčutljive poti. (Maksimalno neobčutljiva tri vhodna AND vrata, imajo vhodne nabore 000). Za naš primer iz slike 3, potem dobimo sledeče vhodne testne nabore, razpredelnica II:

VHODI	TESTNI NABORI 1	2	3	4	5	6	7	8	9	10	11	12
A	1	0	1	1	1	1	0	0	0	0	0	0
B	1	1	1	0	1	1	0	0	0	0	0	0
C	1	1	1	1	1	0	0	0	0	0	0	0
D	0	0	0	0	0	0	1	0	1	1	1	1
E	0	0	0	0	0	0	1	1	1	0	1	1
F	0	0	0	0	0	0	1	1	1	1	1	0
G	0	1	0	1	0	1	1	1	1	1	1	1
H	1	1	1	1	1	1	0	1	0	1	0	1

Razpredelnica II.

Pri močno rekonvergenčnih kombinacijskih vezjih je problematika preiskušanja zelo zamotana in zaenkrat ni na voljo učinkovitega splošnega postopka za diagnostiko večkratnih napak. Tudi na tem mestu predlagamo vpeljavo dodatnih testnih točk /5/, ki jih smatramo kot dodatne primarne testne izhode.

Dodatne testne točke namestimo na rekonvergenčne poti, kot sta to npr.: T1 in T2 na sliki 4. S pomočjo dodatnih primarnih izhodov razdelimo celotno vezje na več manjših drevesnih vezij ki jih že znamo uspešno testirati.

IV. SKLEP

V delu je opisan model napak, ki je specifičen za CMOS digitalna integrirana vezja. Določitev primernega in realnega modela napak, je izredno pomembno pri diagnostiki napak in simulaciji vezij, ki uporabljajo CMOS logične elemente. Pokazano je, da so znani postopki za diagnostiko kombinacijskih vezij, ob predpostavki klasičnega modela napak, nezadostni za diagnostiko CMOS vezij. Lahko pa se enostavno adaptirajo s tem, da se upošteva pravilno zaporedje testov. Testni postopek za drevesna vezja, kjer upoštevamo SOD, klasične enojne in večkratne napake, je zgrajen na osnovi že znanih pravil za tvorjenje občutljivih poti in razredov napak. Tudi za splošna kombinacijska vezja, uporabljamo znanje o generiranju občutljivih poti. Opisan model napak pa seveda vključimo tudi v kakšen drugačen postopek za detekcijo napak.

LITERATURA

/1/ P.Biljanović: Mikroelektronika: integrirani elektronički sklopovi: Školska knjiga Zagreb, 1983.

/2/ R.L.Wadsack: Fault Modeling and Logic Simulation of CMOS MOS Integrated Circuits: The Bell System Technical Journal, May - June, 1978.

/3/ R.Murn, J.Korenini: Graf povezav razredov napak kombinacijskih logičnih vezij in detekcija enojnih in večkratnih napak: Simpozij Informatica, Bled 1975.

/4/ T.Pisanski, R.Murn: Method for Multiple Fault diagnosis in Free Networks: Intern. Symp. on Fault - Tolerant Computing, Paris, 1975.

/5/ R.Murn, T.Pisanski: An Application of FanoutFree Network Diagnosis to General Combinational Network: Proc. of the third Intern. Seminar, Applied Aspect of the Automata Theory, Varna 1976.

INFORMATICA 2/1984

NADZORNI PROGRAM ZA MULTIPROGRAMIRANJE PRI SPROTNEM VODENJU PROCESOV

B. ČRNIVEC

ISKRA AVTOMATIKA, TOZD SISTEMI

UDK: 681.519.7

Članek opisuje koncept nadzornega programa, prirejenega za sprotno vodenje procesov v industriji in energetiki, ki smo ga razvili v okviru obsežnega programskega paketa za nadzor procesov. Paket je osnovan na aparaturni opremi mikroračunalniškega teleinformacijskega sistema TI-30E in obsega množico zaključenih programskih modulov, ki jih povezuje nadzorni program.

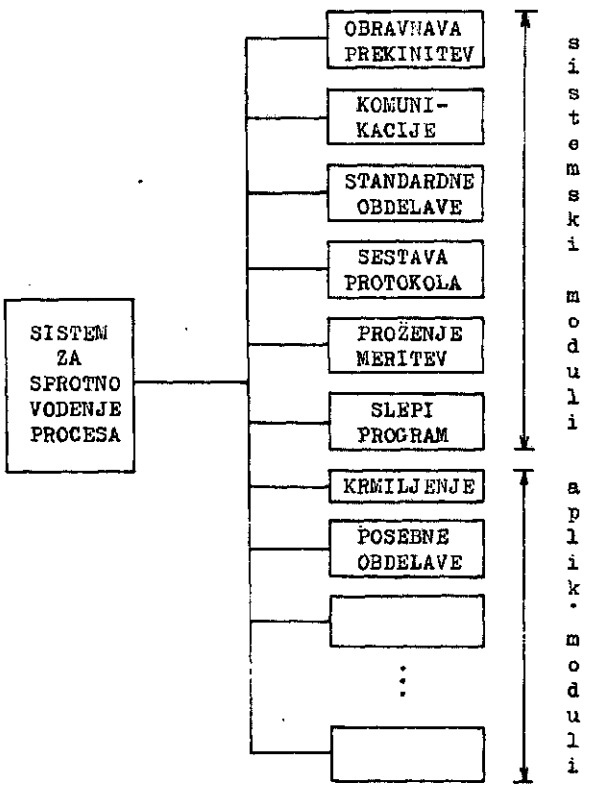
KERNEL FOR MULTIPROGRAMMING IN REAL TIME SYSTEMS. The paper deals with the basic principles of a multiprogramming kernel, adapted for real time control in industry and energetics. It has been developed as part of a complete software package for process control based on the hardware modules of the microcomputer teleinformation system TI-30E.

1. UVOD

Sprotno spremljanje in vodenje procesov je celovit problem, v okviru katerega naletimo na najrazličnejše naloge. Procesi, ki jih želimo krmiliti, se zelo razlikujejo - po obsegu in vrsti vhodnih in izhodnih informacij, po principih in zahtevah za regulacijo, po zahtevanih funkcijah (npr. arhiviranje, statistične obdelave), po načinu spremljanja procesa oziroma izdaje informacij o procesu (vodenje sprotnih in cikličnih protokolov, prikaz na sinoptični plošči, grafičnih terminalih ipd.).

Zaradi tako velikih razlik med posameznimi aplikacijami zahtevamo prilagodljivost sistema za nadzor in krmiljenje procesov v najširšem obsegu. To najlažje dosežemo z veliko modularnostjo aparaturne in programske opreme (dekompozicija nalog na funkcijsko zaključene enote - programske module). Modularnost programske opreme dosežemo z multiprogramiranjem. Fiksno programiranje bi pri apliciranju zahtevalo preglobоke posege v zgradbo programov, čemur pa se v veliki meri izognemo, če sistem razbijemo na neodvisne programske module, ki jih povezuje nadzorni program - monitor.

Kljub tako različnim nalogam pa se pojavi vrsta funkcij,ki so skupne vsem aplikacijam (npr. sprejem informacij iz vhodnih signalnih vrednikov,vpis v banko podatkov, izdaja informacij na standardne izhodne enote ipd.).Programske module - ali krajše - programe zato razdelimo na sistemske in aplikacijske (slika 1). Prvi so nujni za sinhronizirano delovanje celotnega sistema, so v vseh primerih prisotni in le različno parametrirani, drugi pa izvajajo vse specialne, nestandardne naloge, ki se pri dani realizaciji pojavijo. Prednost multiprogramskega modularnega sistema za sprotno spremljanje in vodenje procesov je tudi v tem, da si počasi izgradimo "banko" programskih modulov, iz katerih bomo vsakič le ustrezno izbrali, parametrirali in povezali določene programe.



Slika 1: primer razdelitve nalog med programe

Sistem za sprotno vodenje procesov mora biti grajen tako, da informacije prevzame takrat, ko so na voljo, in da jih uspe obdelati. Najbolj kritične situacije se pojavijo ravno v fazi zajemanja podatkov, ko je potrebno ob nastopu pomembne informacije nasilno prekiniti aktivnost sistema in selektivno ukrepati. Fiksno programiranje v splošnem ne omogoča enostavnega zajemanja in obdelave različno pomembnih signalov. V multiprogramskem sistemu dodelimo programom različne prioritete, s čimer omogočimo sprejem in ukrepanje po pomembnosti. V takem sistemu moramo zato imeti na voljo dovolj hiter in zmogljiv nadzorni program, ki bo usklajeval odvijanje posameznih funkcij in učinkovito reagiral na prekinitve.

2. NALOGE NADZORNEGA PROGRAMA

Nadzorni program predstavlja jedro multiprogramskega sistema za vodenje procesov. Delovati mora sinhrono z obravnavanim procesom, v realnem času in kot tak opravljati tri osnovne funkcije:

- prvi zagon celotnega sistema okoli jedra je potrebno izgraditi in organizirati sistem programov - to je razdeliti pomnilniški prostor, formirati in sestaviti množico različnih podatkovnih struktur, ki bodo omogočale paralelno programiranje;

- razvrščanje programov in dodeljevanje centralne procesne enote; naenkrat se lahko izvaja le en izmed množice prisotnih programov, zato mora monitor določati vso časovno in informacijsko koordinacijo, izbirati točke preklopa med programi in te preklope v ustrezno hitrem času izvršiti;

- sinhronizacija z zunanjim svetom; iz razdelitve nalog med neodvisne programe, sledi, da morajo nekateri dogodki v procesu bistveno in v najkrajšem času vplivati na strategijo preklopa in izbiro najustreznejšega programa za izvajanje; vsa sinhronizacija sistema z zunanjimi dogodki mora zato potekati preko prekinitvenih linij.

Nadzorni program mora pri izvrševanju svojih funkcij izpolnjevati vrsto splošnih zahtev. Med najpomembnejše sodijo naslednje:

- omogočati mora zaščito skupnih virov (sočasna izključitev programov pri uporabi skupnih virov), kar je predpogoj za pravilno delovanje sistema;

- algoritem izbiranja programov za centralno procesno enoto mora zagotavljati enoumno determiniran in jasen vrstni red odvijanja programskih enot in funkcij;

- delovanje v realnem času - velika hitrost preklopa med programi; tej hitrosti je direktno sorazmerna kapaciteta (število obravnavanih informacij) in zmogljivost sistema;

- omogočati mora časovno odvisno startanje programov: start ob absolutno (astronomsko) nastavljenem času, periodično startanje z nastavljivo periodo, enkratno startanje ob izteku nastavljenega časovnega intervala;

Izdelani koncept in princip delovanja nadzornega programa za multiprogramiranje v realnem času, opisan v naslednjih poglavjih, izpolnjuje vse zgoraj navedene kriterije in je popolnoma splošen. Šele konkretna realizacija sistemskih podatkovnih struktur in procedur se mora zaradi zaželene čim večje hitrosti delovanja opirati na značilnosti centralno procesne enote in periferne aparaturne opreme, izkoriščati vse njene prednosti in posebnosti.

Realizirani nadzorni program je implementiran na aparaturni opremi teleinformacijskega sistema TI-30E, ki je osnovana na mikroprocesorski družini M6800. Izdelan je bil v prvi vrsti za podporo programskega paketa za spremljanje in vodenje procesov v energetiki. Značilnost teh procesov je, da je pri "normalnem stanju" povprečno število signalov (dogodkov v procesu) na časovno enoto sorazmerno majhno, ob "konicah" (npr. okvara procesa) pa se to število lahko tudi nekaj tisočkrat poveča. Zato monitor in ves pripadajoči multiprogramski sistem izpolnjujeta še specifične zahteve:

- hitro in učinkovito razpoznavo in obravnavo velikega števila prekinitvenih signalov brez izgube relevantnih informacij (v elektroenergetiki je zahtevana ločljivost dogodkov 10 ms);

- v primeru trenutnega nastopa velikega števila vhodnih informacij monitor ne sme izvajati neekonomičnih preklopov; šele po sprejemu vseh "hkrati" prispelih signalov mora izbrati najustreznejši program za izvajanje (npr. hitra ustavitev procesa v primeru kritičnega alarma);

- želimo popolno neodvisnost programskih modulov, zato mora nadzorni program izpeljati prvi zagon (kreiranje sistema) tako, da formira sistemske informacijske strukture, vsak program pa generira in inicializira svoje lokalne strukture in vmesnike; le na ta način se izognemo poseganju v strukturo programov ob konfiguriranju različnih aplikacij.

3. REALIZACIJA NADZORNEGA PROGRAMA

V naslednjih razdelkih bodo podrobneje opisane vse operacije, vključene v nadzorni program. To so generiranje celotnega sistema, dodeljevanje centralno procesne enote, sinhronizacija med programi in z realnim časom, izključitev tekmujočih programov ter sprejemanje prekinitvenih signalov. Vse te operacije se odvijajo v priviligiranem režimu delovanja, pod prepovedjo zunanjih prekinitev.

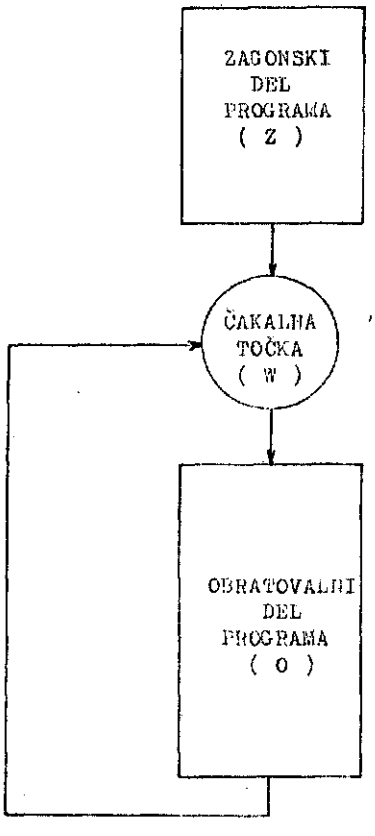
3.1. Prvi zagon in zgradba programov

Aparaturna oprema TI-30E trenutno ne obsega perifernih pomnilniških enot (gibki disk ipd.), zato so nadzorni program in vsi programski moduli fiksno spravljeni v hitrem pomnilniku (EPROMu). Nadzorni program skoči v sekvenco generiranja ob nastopu zunanjega (reset) impulza. Formirajo se sistemske informacijske strukture: vsakemu programu pripadata prekinitveni pomnilnik, ki je hkrati sklad in deskriptor. V prvem so shranjene vsebine notranjih registrov procesorja, drugi pa daje dinamično sliko o stanju programa in logičnih pogojih za njegovo izvajanje (1). Ob zagonu postavimo v prekinitvene pomnilnike startne adrese programov (2) in naravnamo kazalce po teh pomnilnikih (stack pointers), vse deskriptorje postavimo na vrednost, ki zahteva enkratno izvajanje pripadajočih programov (glej razdelek 3.2.). Zatem sprožimo operacijo dodeljevanja procesorja in s tem je s stališča monitorja generiranje sistema končano. Tak postopek zahteva posebno zgradbo

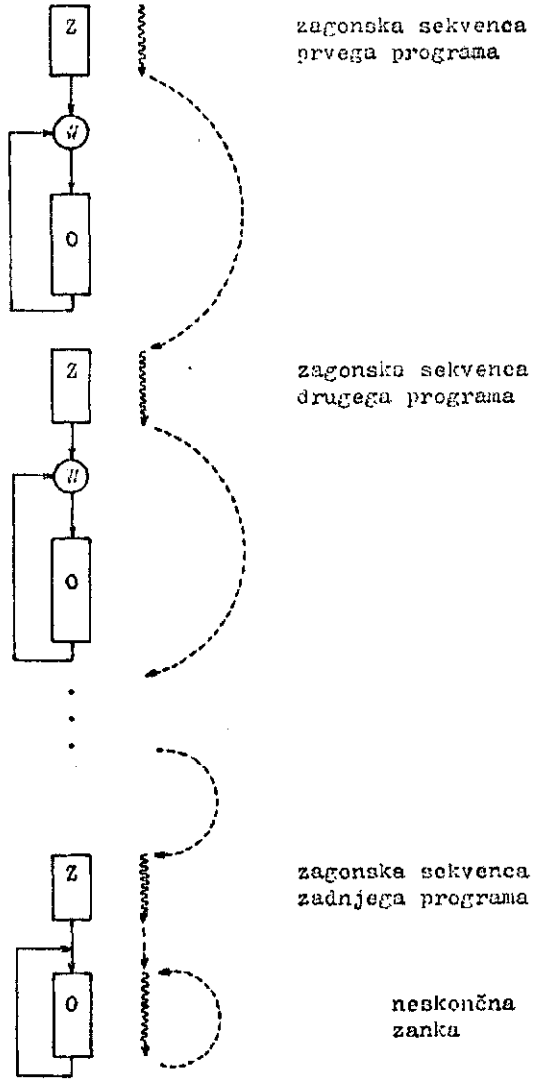
vseh programov, ki jo ilustrira slika 2. Vsak program je sestavljen iz "zagonskega" in "obratovalnega" dela. Ob taki zgradbi poteka postopek prvega zagona avtomatično naprej po pravilih za preklapljanje in razvrščanje (razdelek 3.2.), kar ima za posledico, da se zaporedno izvršijo zagonski deli vseh programov (slika 3), na koncu pa pademo v slepi program, ki je neskončna zanka (1). S tem je sistem dokončno generiran.

Vsi programi so ves čas naloženi v centralnem pomnilniku, zato nista implementirani operaciji kreiranja in brisanja programov. Ob vključitvi perifernih pomnilniških enot jo bo potrebno dodati, vplivali pa bosta le na formiranje in odstranitev deskriptorjev pripadajočih programov (1).

Zagonski del vsakega programa se izvrši natanko enkrat, to je ob zagonu sistema in služi za generiranje lokalnih informacijskih struktur in za postavitev začetnih vrednosti notranjih spremenljivk. Zato bomo odslej pod izrazom program pojmovali le obratovalni del - to je resnični "delovni" del programa, ki se lahko poljubnokrat izvede. Zaključen je sam vase (sl. 2) preko čakalne točke, ki predstavlja točko preklopa (program prostovoljno odstopi procesor). Vsakič, ko je program startan, se začne izvajanje - obratovalnega dela - neposredno za čakalno točko, torej se vsakič izvede isti program, seveda z različnimi vhodnimi podatki.



Slika 2: zgradba programov



Slika 3: generiranje sistema

3.2. Dodeljevanje procesorja

Monitorski programi se razlikujejo po "politiki" dodeljevanja procesorja, to je po kriterijih za izbiro programa, ki naj se izvaja (razvrščanje), in po točkah, v katerih lahko pride do take izbire (točke preklopa). Te operacije izvrši del nadzornega programa, ki ga imenujemo razvrščevalnik (scheduler).

Pogosta je realizacija (1;3), kjer programe razporedimo na nekaj prioritetnih nivojev (ponavadi 4, 8 ali 16). Na vsakem nivoju dinamično razvrščamo programe v čakalni seznam, glede na trenutno stanje sistema. Ko pride dana prioriteta na vrsto, se začne izvajati program na vrhu seznama. Medsebojna sinhronizacija je dosežena s pomočjo signalov (binarnih spremenljivk), ki aktivirajo programske module. Ti imajo na razpolago več stanj, npr: aktivno, pasivno, čakanje na signal, čakanje na prekinitev (na en signal lahko čaka več programov - zopet se formira čakalni seznam). Taka realizacija nudi zelo široke možnosti uporabe, vendar pri večjem številu programskih modulov in signalov zelo težko predvidimo vse možne situacije, kar pa je nujno za pravilno delovanje.

S podrobnim proučevanjem nalog in funkcij že realiziranih sistemov za spremljanje procesov ugotovimo, da se različna pomembnost vhodnih informacij direktno odraža v nalogah in operacijah, ki dobijo tipično hierarhičen značaj. Sistem za sprotno vodenje procesov predstavlja razširitev sistema za spremljanje procesov, kjer krmiljenje formalno predstavlja le množico logičnih pogojev ter strogo determiniranih vzročnih akcij, pri pogoju, da so vhodne informacije vedno na razpolago. Zato je ugodno to hierarhijo izkoristiti in jo fiksno vgraditi v nadzorni program - tako da postane število prioritetnih nivojev enako številu programov. V primeru normalno pričakovanega, hierarhičnega teka dogodkov, je izvedba razvrščevalnika na taki osnovi bistveno hitrejša, ker je seveda primitivnejša. V primeru "nestandardnega" teka dogodkov, moramo uporabiti postopke "posebnih" sinhronizacij (razdelek 3.3.), da dosežemo nestandarden vrstni red odvijanja programov s preprostim razvrščevalnikom.

S sintezo izbranega principa sinhronizacij (3.3.) in zgoraj navedenih zaključkov pridemo do razvrščevalnika, ki deluje po naslednjih zakonitostih:

1. Vsakemu programu pripada deskriptor, ki vsebuje konstanto P ter spremenljivki S in KŠ, kjer pomenijo:

   P ... prioriteta programa, za vsako aplikacijo fiksno določena; med množico programov in naborom prioritet obstaja bijektivna preslikava,

   S ... stanje programa: program se vedno nahaja v enem izmed dveh razpoložljivih stanj - "pripravljen" na izvajanje, ali "ustavljen" (oblika binarnega semaforja),

   KŠ... koordinacijski števec: stanje števca je nenegativno število, ki določa, kolikokrat se mora program še izvršiti (startati).

2. Naenkrat se lahko izvaja le en program (ker imamo monoprocesorski sistem).

3. Izvajanje programa se prekine v dveh primerih:

   - v programu naletimo na klic razvrščevalnika (procedura schedule); ta klic je avtomatično vgrajen na koncu programa - v čakalni točki,

   - nastop zunanjega prekinitvenega signala (prekinitve perifernih V/I enot, časovnih vezij ipd. obravnavamo enako kot resnične dogodke v kontroliranem procesu, vse je vezano na prekinitvene linije).

4. Do preklopa (razvrščanja) pride natanko po vsaki prekinitvi izvajanja programa. Izbira programa, ki mu bomo dodelili procesor, poteka po naslednjem preprostem algoritmu:

   - vsi programi v stanju "ustavljen", ne tekmujejo za CPE (procesor),
   - vsi programi, katerih koordinacijski števec je enak nič, ne tekmujejo za CPE,
   - izmed preostalih programov, dodelimo CPE programu z najvišjo prioriteto.

   Potrebno je poudariti, da absolutna vrednost koor. števca, ki je večja od nič, nima nobenega pomena več na izbiro - dodelitev CPE. Na najnižji prioriteti imamo v prejšnjem razdelku omenjeni "slepi" program. Ves čas je v stanju "pripravljen", ves čas ima KŠ različen od nič, zato je po zgornjem algoritmu vedno izbran, ko noben od "dejanskih" programov nima izpolnjenih pogojev za izvajanje. Ta program predstavlja neskončno zanko, iz katere nas lahko reši le nastop zunanjega prekinitvenega signala.

Preklop fizično izvršimo tako, da pospravimo stanje notranjih registrov procesorja v sklad programa, ki je bil prekinjen, napolnimo registre procesorja iz sklada programa, ki je izbran za izvajanje, in le-to sprožimo. Izbiranje programa izvršimo pred operacijo zamenjave vsebin registrov. To je pomembno zaradi povečanja hitrosti če dejanskega preklopa ni (za izvajanje je izbran isti - predhodno prekinjeni program), kar je pri izbranem konceptu pogost primer.

3.3. Sinhronizacija med programi

Pri nalogah procesnega vodenja moramo zagotoviti sinhronizacijo med (sodelujočimi) programi, sinhronizacijo programov z dogodki v obravnavanem procesu ter sinhronizacijo z realnim (astronomskim) časom. Če inkrement ure obravnavamo kot dogodek v procesu in če zajemanje dogodkov iz procesa (prekinitve) realiziramo kot zaključen program (razdelek 3.5.), reduciramo vse primere v problem sinhronizacije med programi.

Za sinhronizacijo programov smo uporabili principe, dobljene s sintezo poenostavljenih načinov sinhronizacije po metodi signalov (1;3) in po metodi koordinacijskih števcev s pripadajočim prenosom informacij med programi preko krožnih medpomnilnikov (4). Princip koordinacijskih števcev je za procesne namene zelo zanimiv, saj po potrebi enostavno omogoča večkratno (zaporedno) startanje istega programa z različnimi vhodnimi podatki, ki se kopičijo v vhodnem krožnem medpomnilniku. S tem uspešno rešujemo problem preobremenitve procesorskega sistema v kritičnih situacijah, kot je zahtevano v poglavju 2.

Kot osnovni princip sinhronizacije zato sprejmemo koordinacijski števec s principom medpomnilnikov in zahtevano vgraditev "čakalne točke" v vsak program, kot je že prikazano na sliki 2. V kompleksnih sistemih ima lahko vsak program več koor. števcev (4), v našem primeru pa smo se omejili na odnos, da vsakemu programu pripada natanko en števec, kar v splošnem popolnoma zadostuje, vpliva lahko le na način razdelitve nalog med programe. Kot osnovno operacijo zato uvedemo proceduro start (program), ki služi za startanje poljubnega programa (parameter), pri čemer prenese vhodne podatke v sprejemni medpomnilnik dotičnega programa ter mu inkrementira koor. števec. To je realizacija operacije tipa send (signal) (1), pri čemer bi vsak signal pripadal natanko enemu programu. Ko se program izvrši do konca, skoči v čakalno točko, kjer sam sebi najprej dekrementira koor. števec (glej definicijo KŠ v 3.2.) in nato kliče proceduro schedule. To predstavlja avtomatično (obvezno) izvršitev procedure tipa wait (signal) (1), pri čemer bi vsak program zopet čakal samo na svoj (en sam) signal. Ker lahko naloge pri vodenju procesov razdelimo tako, da je večina programov sodelujočih, izbrana zgradba programov zelo poveča preglednost in enostavnost sinhronizacij med programi, ki tako operacijo (wait) zahtevajo.

Zaradi preprostosti tega mehanizma, bi morali v kompliciranih situacijah naloge razdrobiti

na ogromno število programov, kar je nepregledno in neekonomično. Zato dodamo še princip "posebne" sinhronizacije,kjer vsakemu programu dodelimo binarni semafor, ki označuje stanje programa ("ustavljen"/"pripravljen"), kar realiziramo s procedurama ustavi (program) in pripravi (program). Prva ustavi (prepove) izvajanje danega programa, druga pa le-to zopet omogoči. Stanje semaforja vpliva na dodeljevanje procesorja, kot je bilo opisano v prejšnjem razdelku.

Iz opisanih zakonitosti odvijanja programov (3.2.) in sinhronizacij sledi, da naj bodo programi prioritetno razporejeni tako, da jih je čim več v stanju "pripravljen", tako da prioriteta in stanje koor. števcev določata normalni vrstni red izvajanja. Semafor ustavljen/pripravljen uporabljamo le pri kompliciranih primerih koordinacije med programi (primer: takojšnje izvajanje nekega programa dosežemo tako, da vse programe z višjo prioriteto postavimo v stanje "ustavljen", pri čemer koor. števci obdržijo informacijo, kolikokrat bodo morali programi še steči, da bodo vse naloge opravljene).

Pri sprotnem vodenju procesov je sinhronizacija programov z realnim časom bistvenega pomena, zato se uporablja več tovrstnih operacij, običajno procedure vrste run (program, čas) in delay (program, zakasnitev) (3). Prva sproži izvajanje programa ob določenem (astronomskem) času, druga zakasni izvajanje programa za določeno število časovnih enot. Poleg gornjih smo uvedli še proceduri pers (program, perioda) in briši (program). Prva razporedi program za periodično startanje (v konstantnih časovnih intervalih), druga pa dani program briše iz razporeda za periodično startanje.

3.4. Zaščita skupnih virov

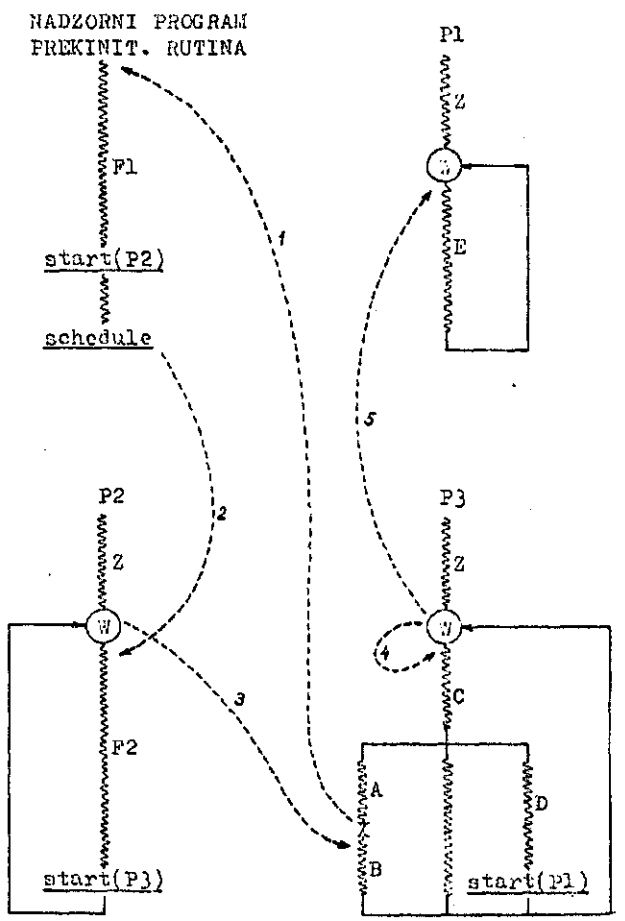
Zaščito dosežemo s sočasno izključitvijo programov, ki te vire uporabljajo. Skupne vire predstavljajo določene pomnilniške lokacije (npr. krožni medpomnilniki) pa tudi nekatere V/I enote. Zaščito prvih enostavno dosežemo s prepovedjo zunanjih prekinitev, saj to onemogoča preklop in s tem hkratno poseganje v podatkovne strukture. Ti posegi so v splošnem zelo kratki, zato prepoved prekinitev časovno ni kritična. Dodeljevanje V/I enot (ali pa obsežnih struktur, kjer bi bili časi posega predolgi) pa si organiziramo v okviru sistemskih programskih modulov, ki krmilijo vse istovrstne enote - npr. program "komunikator" dodeljuje in nadzoruje vse serijske asinhrone vhode in izhode, naj si bodo ti vezani na druge procesorje, tiskalnike, alfa-numerične terminale, ipd. Tu uporabljamo metode "posebnih" sinhronizacij, s čimer programom onemogočimo hkraten dostop do perifernih enot (če nek program zahteva že zasedeno enoto, ga enostavno "ustavimo").

3.5. Sprejemanje prekinitev

Že v 2. poglavju smo ugotovili, da moramo vse pomembne informacije v procesu vezati na prekinitvene linije procesorja. Izkoristek CPE pa bistveno povečamo, če tudi oskrbovanje vseh ostalih vhodnih in izhodnih naprav realiziramo z uporabo prekinitvenih signalov. Tako se srečamo s problemom sprejemanja velikega števila prekinitev, od česar je direktno odvisna kapaciteta sistema (število možnih vhodnih signalov).

Sprejemnje zato razbijemo na dve fazi. Prva predstavlja osnovno strežno prekinitveno rutino, ki se odvija pod okriljem nadzornega programa (v priviligiranem režimu). V tej fazi opravimo le najnujnejše - poiščemo periferno napravo, ki je prekinitev generirala, sprejem potrdimo (brisanje prekinitve),sam dogodek, opisan z enim ali dvema zlogoma, pa vnesemo v krožni medpomnilnik in sprožimo izvajanje programa za obravnavo prekinitev. V okviru tega programa, ki mu vedno dodelimo zelo visoko (pogosto najvišjo) prioriteto, realiziramo drugo fazo streženja prekinitev. Ta faza se odvija v normalnem režimu (brez prepovedi prekinitev), obsega pa čitanje omenjenega medpomnilnika (FIFO) in klic ustrezne strežne sekvence, glede na opisani dogodek (npr. čitanje stanja vhodov, prevzem meritve iz A/D pretvornika ipd.).

Ura realnega časa generira enake prekinitvene impulze (periodično), ki jih v prvi fazi preslikamo v časovne zastavice (markerje) v opisanem medpomnilniku. Tako v drugi fazi enostavno vodimo programsko uro procesnega časa, kjer natanko vemo, kateri dogodki so se zgodili v



1: (0,0,1)
2: (0,1,1)
3: (0,0,2)
4: (0,0,1)
5: (1,0,0)

Slika 4: zgled

danem časovnem intervalu. V primeru preobremenitve sistema, lahko procesna ura določen čas zaostaja za realnim časom, vendar bodo tako vsi dogodki opremljeni s pravim realnim časom nastopa.

3.6. Zgled

Vse osnovne zakonitosti, opisane v predhodnih razdelkih, najlepše ilustriramo z zgledom, ki ga prikazuje slika 4. Črtkane puščice, s pripadajočimi številkami, označujejo zaporedje preklopov, pod sliko pa je zapisano stanje koordinacijskih števcev (KŠ1,KŠ2,KŠ3) ob vsakem preklopu. Črka W predstavlja čakalno točko programa, kjer pride do dekrementiranja koor. števca programa, razvrščanja (klic procedure schedule) in pripadajočega preklopa. Črka Z označuje zagonske dele programov, ki v času obratovanja niso več bistveni.

V sistemu naj bodo na prvih treh prioritetah program za hitro ustavitev procesa (P1), program za obravnavo prekinitev (P2), program za standardno obdelavo (P3). Vzemimo, da so vsi v stanju "pripravljen", da ravno poteka obdelava nekega predhodno sprejetega signala, kar pomeni, da ima koor. števec programa P3 (KŠ3) vrednost 1. V točki X naj se na vhodu pojavi signal, ki zahteva hitro ustavitev procesa. Obdelava se torej prekine, izvede se prva faza sprejemanja prekinitve (F1), v kateri sprožimo P2 - dokončni zajem signala. Na koncu te druge faze (F2) zahtevamo obdelavo (vsebinsko razpoznavo) novo prispele informacije - torej start programa P3. V okviru P3 pa se najprej izvrši sekvenca B - do konca obdela prejšnja informacija (tudi ta bi lahko bila pomembna !). Pri prehodu skozi čakalno točko, kjer se dekrementira KŠ3, ima le-ta še vedno vrednost 1, zato ponovno steče obdelava novo prispelega signala (sekvenca C). Ko spoznamo, da je zahtevana hitra ustavitev procesa, v sekvenci D startamo P1 (inkrementiranje KŠ1 !), kar povzroči takojšnje izvajanje programa P1, kjer sekvenca E predstavlja algoritem za ustavitev.

4. ZAKLJUČEK

Zasnovani mehanizem dodeljevanja procesorja, odvijanja programov in sinhronizacij zahteva pri zapletenejših preklopih klicanje večjega števila procedur, zato pa je izredno enostaven in jasno determiniran ter izpolnjuje vse pogoje glede hitrosti in enoumnega zaporedja odvijanja posameznih funkcij.

Nadzorni program je implementiran na procesorju M6800, kjer zavzame približno 1K pomnilniških lokacij. Procedura schedule je realizirana z ukazom SWI, vse ostale procedure - start, ustavi, pripravi, run, delay, pers in briši - pa kličemo kot podprograme, ker je taka realizacija najhitrejša.

5. LITERATURA

(1) D.Novak, M.Exel, M.Kovačević, B.Kastelic : Jedro za podporo implementacije sprotnih sistemov, Informatica, vol.4, No.2, s. 3-7

(2) M.Damke: Introduction to Multiprogramming, Byte, Sept. 1979, s. 20-32

(3) Monitorski program MON6800, Priročnik za uporabo, dokum. št. 421 630 015/401, Iskra Avtomatika, Ljubljana, 1981

(4) ORG 310/Mobi, Beschreibung P7100-Bo310-X-X -35, Siemens Systeme 300

COMMUNICATIONS FACILITY

TOMAŽ ERJAVEC

UDK: 007:681.3

INTERTRADE, TOZD ZASTOPSTVO IBM, CENTER ZA RAZVOJ PROGRAMSKE OPREME

Sestavek predstavlja strukturo računalniške mreže CF, zgrajene za povezovanje računalnikov IBM Sistem/1 in uporabniških računalnikov v mreže ali lokalne mreže. Na koncu je še primerjava med nivoji CF in ISO nivoji.

COMMUNICATIONS FACILITY, DESCRIPTION AND COMPARISON TO ISO LEVELS
The article presents the structure of CF network, desisned to link IBM Series/1 computers and host computers into a network or a local network. The levels of CF are compared to ISO levels at the end.

1.0 UVOD

Communications Facility, s polnim imenom IBM Series/1 Event Driven Executive Communications Facility, od tu dalje označevan s kratico CF, je skupek programske opreme, ki ob dodatku potrebne strojne opreme omogoča zgraditev računalniške mreže, lahko tudi lokalne. Lokalno mrežo nadzira LCC (Local Communications Controller), ki podpira konfiguracijo obroča s 16 računalniki S/1.

Mreža je grajena prvenstveno za povezovanje računalnikov S/1 z večjimi IBM računalniki, IBM pa je predvideval, da se bodo ob dograditvi ustreznih modulov lahko priključevali v mrežo tudi uporabniški računalniki drugih proizvajalcev.

Pri tem gre za enostavno priključitev S/1 na večji sistem ali pa za graditev kompleksne mreže, v kateri S/1 prevzema le funkcijo vozlišča. CF podpira neposredno priključitev v SNA omrežje.

V nadaljnem besedilu je opis lastnosti CF, pregled programskih modulov v vozliščnem računalniku, prikaz pretoka in usmerjanja sporočil v vozlišču in primerjava z ISO nivoji.

2.0 KRATEK OPIS CF

2.1 Komunikacija S/1 - uporabniški računalnik

Povezava med S/1 in uporabniškim računalnikom je lahko izvedena na tri načine:

BSC multipoint linija
SNA povezava
S/1 - S/370 kanal

Prenos podatkov med S/1 in uporabniškim računalnikom je navadno 3270 podatkovni tok. S/1 je emuliran tako, da uporabniškemu računalniku izgleda kot kontrolna enota 3270.

2.2 Komunikacija S/1 - S/1

Povezava med S/1 računalniki je lahko izvedena na dva načina:

BSC linija v delovanju (point-to-point) ali (multipoint).

LCC (Local Communications Controller) v konfiguraciji obroča.

Omogočena je tudi povezava med S/1 RPS Communications Monitor in S/1 EDX Communications Facility preko LCC.

2.3 Nadzor nad mrežo

CF omogoča sproten nadzor mreže. Z vsakega terminala je mogoče ob vsakem trenutku izvesti naslednje ukaze:

Definirati novo postajo.
Vključiti postajo v mrežo.
Ustaviti tok podatkov med postajami.
Izključiti postajo iz mreže.
Povezati dve postaji v eno ali obe smeri kot privzeta naslova.
Spremeniti definicijo postaj.
Izpisati statistične podatke o postaji.

2.4 Definicija sistema

CF prepoznava enote programske in strojne opreme iz definicij po naslednjem dogovoru:

VOZLIŠČE je vsak računalnik S/1 v konfiguraciji.

POSTAJA je lahko naprava, komunikacijska povezava ali aplikacijski program. Postaje so lahko različnih tipov.

Pri definiranju vsakega vozlišča v mreži se kreira definicijska datoteka z informacijo o vseh postajah lokalnega vozlišča, o vseh vozliščih, s katerimi lahko lokalno vozlišče komunicira ter o vseh postajah v oddaljenih vozliščih, s katerimi bo lokalno vozlišče komuniciralo neposredno. Vozlišč, ki sicer obstajajo v mreži, a lokalno vozlišče z njimi ne bo komuniciralo, ni potrebno navesti v definiciji vozlišča. Izvorna postaja lahko pošlje sporočilo preko svojega vozlišča v neko drugo vozlišče, ki ni neposredno povezano z njenim vozliščem.

2.5 Centralni nadzor in vzdrževanje mreže

S centralnega vozlišča je mogoče upravljati naslednje naloge:

prenašati datoteke
alocirati in brisati datoteke v oddaljenem vozlišču ali uporabniškem računalniku
startati neko oddaljeno vozlišče
nadzirati vozlišča z uporabniškim računalnikom

2.6 Dostop do oddaljenih diskov

Aktivni programi imajo lahko s poljubne točke v mreži dostop do diskov vozliščnih računalnikov. Datoteke na disku oddaljenega vozlišča se preslikujejo postaji, kot da so na lokalnem vozlišču.

2.7 Diagnostika

Diagnostična pomagala omogočajo funkcije:

poimensko izpisati postaje
vzpostaviti postaje
zbrisati postaje
pošiljati in sprejemati sporočila
prikazati vsebino pomnilnih področij nekega sistema

2.8 Potrebe za delovanje

Vozlišči računalniki so lahko S/1 procesorji 4952, 4955 ali 4956 z ustrezno strojno opremo in pomnilnikom večjim od 96K. Vozlišče je lahko že samo procesna enota brez terminalov in ostale opreme.

Programski del opreme teče pod operacijskim sistemom EDX verzija 3 ali več z ustreznimi prevajalniki in servisnimi programi za instalacijo. Za povezavo z SNA je potreben EDX-SNA program.

3.0 OPIS VOZLIŠČA

3.1 Tipi postaj

V vsakem vozlišču je potrebno definirati vse postaje, ki so priključene na lokalno vozlišče, ter vse oddaljene postaje, s katerimi naj vozlišče komunicira. Definicije postaj so shranjene v $.SYSNET datoteki, ki se nahaja v vsakem vozlišču in mu definira njemu dostopno strukturo mreže.

UPORABNIŠKA POSTAJA predstavlja program, ki je lahko uporabniški ali pa V/I program CF-a. Uporabniški program mora imeti postajo, če sprejema sporočila iz mreže, zahteva potrditev odposlanih sporočil ali če ga hočemo ustaviti s transakcijsko komando. Ob startanju uporabniške postaje poženemo program, ki ga predstavlja.

POSTAJA ZA SPOROČILA je pomnilni blok, ki sprejema sporočila. Lahko ne predstavlja ničesar za seboj. Obstaja posebna postaja WASTE za sprejemanje sporočil, ki ne najdejo naslovnika v vozlišču in iz vozlišča ne vodi nobena pot proti naslovniku.

POSTAJA ZA NAPRAVE predstavlja napravo, ki je priključena na vozlišče (terminali,tiskalniki). Vsebuje kontrolni blok za napravo, ki s postajo komunicira preko V/I programa.

LINIJSKA POSTAJA predstavlja komunikacijsko povezavo z drugim vozliščem. Linijska povezava je lahko LCC, BSC linija ali kanalska povezava z uporabniškim računalnikom. Linijska postaja vsebuje kontrolni blok, ki je povezan z V/I programom. V/I program posanja komunikacijo preko linije.

VOZLIŠČNA POSTAJA predstavlja oddaljeno vozlišče LCC obroča. Njena naloga je hraniti sporočila, ki se odpošiljajo naslovniku v oddaljenem vozlišču. Povezana je postajo LCC linije.

TERMINALNA POSTAJA je postajni blok s pomnilnikom za sporočila, ki predstavlja pravi ali emulirani terminal 3270. Uporablja se za BSC povezavo ali kanal na uporabniški računalnik.

SNA PU IN LU POSTAJI služita za komuniciranje z uporabniškim računalnikom preko SDLC linije. Uporabniški računalnik vidi vozlišče kot 3274-SDLC kontrolno enoto. LU predstavlja emulirani terminal ali tiskalnik, PU pa predstavlja informacije, skupne vsem LU postajam v vozlišču. PU postaja je lahko v vozlišču le ena.

3.2 V/I kontrolni programi IOCP

Naloga IOCP programov je upravljati naprave in postaje, ki jih predstavljajo. IOCP programi nadzirajo dve različni vrste naprav: terminalne naprave in linije, ki se delijo na linije med vozlišči in na linije med vozlišči in uporabniškimi računalniki. En IOCP program lahko nadzira več naprav, pri čemer ima en glavni program in podprogram za vsako napravo. Glavni program sprejema navodila za vklop, izklop in ustavitev postaj, ki jih IOCP nadzira. Podprogram za neko postajo poženo v trenutku, ko dobi navodilo za njen vklop. Podprogrami v celoti upravljajo napravo. Berejo podatke od naprave in jih pošiljajo v vozlišče v obliki CF sporočil. Sprejemajo sporočila od postaj v vozlišču, ki predstavljajo naprave, in jih podajo napravi. Nekaj najvažnejših IOCP programov:

IOCP za nadzor BSC linije upravlja povezavo med BSC linijskim priključkom na uporabniški računalnik in linijsko postajo. Vozlišče izsleda uporabniškemu računalniku kot 3271 kontrolna enota.

IOCP za nadzor kanalske povezave med uporabniškim računalnikom in vozliščem. Vozlišče izsleda uporabniškemu računalniku kot 3272 kontrolna enota.

IOCP za upravljanje SNA povezave preko SDLC linije. Vozlišče izsleda uporabniškemu računalniku kot 3274-SDLC kontrolna enota, torej kot PU postaja z ustreznimi LU postajami.

IOCP za komunikacijo med vozlišči preko BSC linije. Upravlja postaje 3270 in emulira vozlišče kot 3270.

IOCP za komunikacijo med vozlišči v LCC obroču. Prenaša sporočila z z slavami in repi. Omogoča povezavo CF in CM (na RPS opr.sistemu).

IOCP za komunikacijo med vozlišči preko BSC točka-točka linije.

V vozliščni definiciji morajo biti poleg vseh postaj definirani tudi vsi IOCP programi, saj vozlišču definirajo topologijo njemu vidne mreže.

3.3 Razpošiljatelj sporočil

Naloga razpošiljatelja sporočil je določiti pot, po kateri bo sporočilo prišlo do naslovnika. Aktivira sa instrukcija SEND, ki jo izda IOCP ko sprejme sporočilo od terminalne postaje ali od programa. Pri sprejemanju sporočil pa razpošiljatelj sporočil opozori IOCP, da izda RECEIVE instrukcijo, ki od njega prevzame sporočilo in sa preda terminalni postaji ali programu.

Postaja, ki pošilja sporočilo, mora poleg naslovnika določiti tudi prioriteto sporočila med 1 (najvišja) in 127 (najnižja). Od prioritete je odvisno, kam bo razpošiljatelj zapisal sporočilo v sklad sporočil, s katerega instrukcija RECEIVE jemlje sporočila po vrsti in jih dostavlja naslovnikom. Sporočila z najnižjo prioriteto se zapisujejo na vozliščni disk, ostala pa se ob prihodu v naslovno vozlišče shranijo v vozliščni dinamični pomnilnik.

CF pozna pet vrst sporočil.

Podatkovno sporočilo vsebuje podatke, ki imajo pomen le za naslovnika. Tok teh sporočil je opisan v poglavju 4.1.

Transakcijsko sporočilo vsebuje informacije, ki poženejo nek transakcijski program nekje v mreži. Te informacije se nahajajo v glavi transakcijskega sporočila. Ostali del sporočila so podatki za transakcijski program. Izvor vseh transakcijskih sporočil je razpošiljatelj transakcij. Tok teh sporočil je opisan v 4.2.

Ukazno sporočilo vsebuje ukaz, ki sa sprejme Procesor ukazov nekega vozlišča. Ukazno sporočilo išče naslovnika na enak način kot podatkovno sporočlo. Procesor ukazov je opisan v poglavju 3.4.

Statusno sporočilo začasno prekine program ali pa sa povsem ustavi. Usmerjanje je enako kot pri podatkovnem sporočilu.

Zapisniško sporočilo vsebuje informacije o dogajanju v mreži ali informacije o napakah. Ne razpošilja jih Razpošiljatelj sporočil, temveč so usmerjana naravnost v sistemski zapisnik.

Podrobnejši opis usmerjanja sporočil je v poglavju 4.1.

3.4 Razpošiljatelj transakcij

Razpošiljatelj transakcij (Program Dispatcher) usmerja transakcije in upravlja transakcijske programe. Transakcijski program je lahko vsak program, ki se odzove na transakcijo, tako sistemski transakcijski programi, kot uporabniški programi. Transakcijski program lahko sprejme sporočilo, ki je priključeno transakciji ali pa tudi ne.

Razpošiljatelj transakcij izsleda CF kontrolnemu programu kot postaja, ki predstavlja program. Iz sklada sporočil sprejme transakcijsko sporočilo kot navadno sporočilo; nato sa usmeri na pravi naslov in odpošlje proti pravemu vozlišču. Razpošiljatelj transakcij sestavlja svoja dejanja na osnovi informacij iz transakcijskih sporočil, tabele poti in tabele transakcij. Vsako vozlišče, kjer teče Razpošiljatelj transakcij, vsebuje svojo tabelo poti in tabelo transakcij. Podrobnejše informacije o usmerjanju transakcij so v poglavju 4.2.

3.5 Procesor ukazov

Procesor ukazov izvaja CP instrukcije , ki omogočajo nadzor nad CF sistemom in obveščenost o stanju sistema. CF instrukcije omogočajo definiranje nove postaje, zagon in ustavitev postaje, stalno povezanost med dvema postajama, itd.

Procesor ukazov skupaj z Razpošiljateljem transakcij izvaja PD instrukcije, ki omogočajo nadzor nad Razpošiljateljem transakcij, npr. spreminjati tabelo poti, zamenjati identifikacijo celice, poslati transakcijo, itd. (podrobnejše v poglavju 4.2).

3.6 Procesor zapiskov

Procesor zapiskov zapisuje sistemska sporočila, jih formatira in opremi z datumom, časom, imenom , sporočila, izvorom in ostalimi informacijami ter zapiše v sistemski zapisnik. Vsako vozlišče vodi svoj sistemski zapisnik. Vse zapisnike je mogoče preusmeriti v en centralni zapisnik v središču mrežne konfisuracije.

4.0 PRETOK IN USMERJANJE SPOROČIL

4.1.1 Pretok sporočil v vozlišču

V opisu bomo uporabili naslednje simbole:

A je program ali terminalna enota

A-postaja je postaja, ki predstavlja A

IOCP-A je IOCP program, ki posreduje med A in A-postaja

DMSG-POOL je sklad, v katerem se skladiščijo sporočila za MSG-DISP

MSG-DISP je Razpošiljatelj sporočil

BMSG-POOL je sklad, v katerem se skladiščijo prispela sporočila za B

IOCP-B je IOCP program, ki posreduje med B in B-postaja

B-postaja je postaja, ki predstavlja B

B je program ali terminalna enota

Pretok sporočila iz A v B:

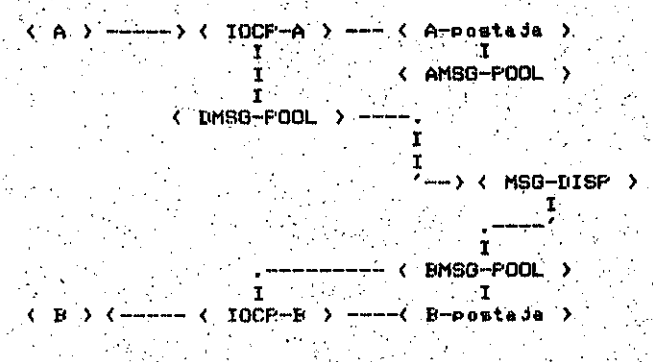
1. IOCP-A prebere podatke iz A in zgradi sporočilo v svojem pomnilnem prostoru. (Če je A program, točka 1. odpade).

2. IOCP-A izda SEND instrukcijo, definira A za izvorno postajo in posleda, če je v sporočilu naslov ponora. Če sa ni, vanj vpiše privzeti naslov (stalna linija, če obstaja, sicer WASTE postaja za nenaslovljena sporočila). SEND zgradi slavo sporočila (tip sporočila, prioriteta, izvor, ponor). IOCP-A pozna ime naslovnika (razen pri privzetem naslovu), ne ve pa, kje se naslovnik nahaja. Sporočilo z slavo spravi v DMSG-POOL. (Če je A program, potem sam izda SEND instrukcijo).

3. MSG-DISP prebere sporočilo iz DMSG-POOL in presleda slavo. Poišče pot do naslovnika (B-postaja) in pošlje sporočilo na njegov BMSG-POOL.

4. IOCP-B izda RECEIVE instrukcijo, ki odstrani slavo s sporočila. (Če je B program, potem sam izda RECEIVE instrukcijo in IOCP-B ni potreben).

5. IOCP-B preda očiščeno sporočilo B. (Če je B program, točka 5. odpade).



slika 4.1. : pretok sporočila v vozlišču

4.1.2 Pretok sporočil med vozlišči

Uporabljeni bodo isti simboli kot v 4.1.1 in še:

LINE1 je postaja, ki predstavlja linijo v vozlišču 1

LINE2 je postaja, ki predstavlja linijo v vozlišču 2

LINE1-IOCP je IOCP program, ki posreduje med LINE1 in priključno kartico na fizično linijo.

LINE2-IOCP je IOCP program, ki posreduje med LINE2 in priključno kartico na fizično linijo.

NODE2 je postaja, ki predstavlja vozlišče 2 v vozlišču 1.

NODE1 je postaja, ki predstavlja vozlišče 1 v vozlišču 2.

NODE2MSG-POOL je sklad, na katerem se skladiščijo sporočila, ki so namenjena v vozlišče 2.

NODE1MSG-POOL je sklad, na katerem se skladiščijo sporočila, ki so namenjena v vozlišče 1.

Pretok sporočila iz A v B:

1. Isti potek kot v poslavju 4.1.1.

2. Isti potek kot v poslavju 4.1.1.

3. MSG-DISP prebere sporočilo iz DMSG-POOL in presleda slavo. Išče pot do naslovnika. Ker je naslovnik (B-postaja) v vozlišču 2, preda sporočilo na NODE2MSG-POOL, to je sklad sporočil za postajo NODE2, ki v vozlišču 1 predstavlja vozlišče 2.

4. Ko LINE1-IOCP zazna prisotnost sporočila na NODE2MSG-POOL, izda instrukcijo RECEIVE, ki odstrani sporočilo iz NODE2MSG-POOL in sa prenese v delovni pomnilnik LINE1-IOCP.

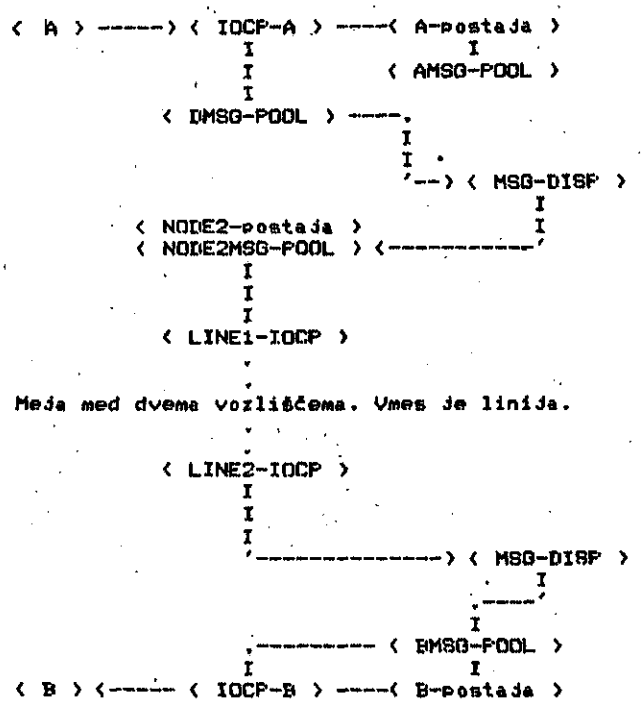
5. LINE1-IOCP pošlje sporočilo po komunikacijski liniji in na drusi strani sa LINE2-IOCP sprejme.

6. LINE2-IOCP pošlje sporočila v MSG-DISP v vozlišču 2. V slavi sporočila so podatki o izvoru (A-postaja) in ponoru (B-postaja).

7. MSG-DISP v vozlišču 2 preslesa slavo in ugotovi, da je ponor B-postaja, zato pošlje sporočilo na BMSG-POOL.

8. Isti potek kot točka 4. v poslavju 4.1.1.

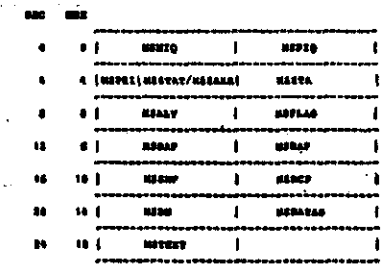
9. Isti potek kot točka 5, v poslavju 4.1.1.



Slika 4.2. : pretok sporočila med vozliščema

4.1.3 Delovanje Razpošiljatelja sporočil

Razpošiljatelj sporočil v lokalnem vozlišču čaka, da se ECB (Event Control Block), ki signalizira prisotnost sporočila v vrsti sporočil, postavi v potrditveni položaj.
Ko ECB potrdi prisotnost vsaj enega sporočila, sa Razpošiljatelj sporočil vzame iz vrste in testira njegov status v glavi. Če je sporočilo ukazno, sa prenese na vrsto ukazov in opozori procesor ukazov na dogodek. Struktura glave je prikazana na sliki 4.3.
Če sporočilo ni ukazno, presleda glavo za prisotnost ciljnega naslova. Če najde naslov, presleda, ali se naslov ciljne postaje nanaša na lokalno vozlišče ali na oddaljeno. Če se nanaša na oddaljeno, se smatra, da je sporočilo namenjeno drugemu vozlišču. V tem primeru formira vozliščni naslov iz zgornjega dela ciljnega mrežnega naslova (naslovi vozlišč in postaj). Nato prenese sporočilo v vrsto sporočil, od koder bo šlo naprej v določeno vozlišče. Ko je sporočilo shranjeno v vrsti sporočil, se postavi ECB za enoto, ki nadzira omenjeno vrsto. Enota je navadno linijska postaja ali LCC linijska postaja in sporočilo v naslednji fazi prenese v naslednje vozlišče.
Če v glavi ne najde ciljnega naslova, preveri ali je sporočilo transakcija. Če je, vstavi v glavo naslov postaje Razpošiljatelja transakcij v sosednji celici (vozlišču).
Če sporočilo ni transakcija, poišče v glavi izvorni naslov in preveri, ali je morda izvorna postaja stalno povezana z neko drugo postajo. Če je, pošlje sporočilo na postajo, ki vodi k cilju in postavi njen ECB.
Če ne najde izvornega naslova, potem preveri, če obstaja postaja WASTE, Če jo najde in je aktivna, zgradi vzročno sporočilo z datumom, uro in ostalimi podatki in prenese sporočilo v njeno vrsto.



Slika 4.3.: struktura glave sporočila

Na sliki 4.3. je prikazana struktura glave sporočila. Najpomembnejše informacije so:

M$NIQ je naslov naslednjega sporočila. 0 pomeni, da je zadnje.
M$PRI je prioriteta sporočila.
M$STAT je status sporočila.
M$OAF je izvorni naslov.
M$DAF je ciljni naslov.
M$SNF je dolžina sporočila, ki jo je podal izvor.
M$DCF je dolžina celotnega sporočila z glavo.
M$TEXT je začetek izvornega sporočila.

4.1.4 Naslovi vozlišč in postaj

VOZLIŠČNO DOLOČILO je dvoštevilčna hex vrednost, ki določa vozlišče. Vozliščno določilo mora biti predpisano vsakemu vozlišču, ki je povezano z ostalimi preko BSC linije ali LCC linije. Pri komunikaciji z Communications Monitor (ki ima vozliščno določilo opisano z dvema alfameričnima simboloma) mora biti CF vozlišče označeno z EBCDIC kodo črke, ki podvojena označuje ustrezno vozlišče v CM (npr. AA in X'C1').

IME POSTAJE je sestavljeno iz 1 do 8 alfameričnih znakov. Vsaka postaja mora imeti svoje ime, ki je drugačno od vseh ostalih. Pri komuniciranju z CM morajo imeti vse postaje ime dolgo natanko 4 znake.

MREŽNI NASLOV je sestavljen iz 4 hex številk. Zgornji dve predstavljata vozliščno določilo vozlišča, v katerem se nahaja postaja. Spodnji dve predstavljata naslov postaje, ki določa postajo v vozlišču. Vsaka postaja v vozlišču mora imeti svoj naslov, ki je drugačen od ostalih.

4.2 Pretok transakcijskih sporočil

Transakcijska sporočila imajo določen format glave, ki pove katera transakcija se bo izvedla in v kateri celici. CELICA je vozlišče, kjer deluje Razpošiljatelj transakcij ali uporabniški računalnik, ki lahko izvaja transakcije. Format transakcijskega sporočila je :

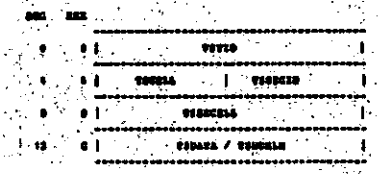
TID1 C1 TID2 C2 transakcijski podatki

kjer je TID1 4 znakovni transakcijski identifikator.
C1 je primarni identifikator celice iz 2 znakov. Opcija je '**', kar pomeni, da je transakcija emisijska in mora priti do vseh celic, do katerih ima dostop.

TID2 je sekundarni transakcijski identifikator, ki sa dekodira transakcijski program.
C2 je sekundarni identifikator celice in sa dekodira transakcijski program.

TRANSAKCIJSKI PODATKI so kakršni koli podatki, poslani transakcijskemu programu.

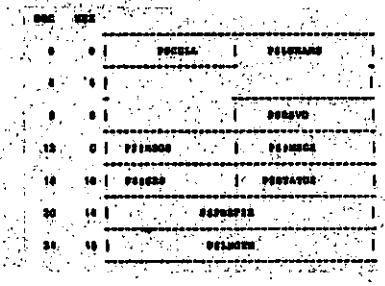
Na sliki 4.4. je struktura glave transakcijskega sporočila. Pri pošiljanju v oddaljene vozle, Razpošiljatelj sporočil naloži na celotno transakcijsko sporočilo še svojo glavo.



Slika 4.4.: Glava transakcijskega sporočila

TABELA POTI ima vnos za vsako oddaljeno celico, ki jo mora prepoznati lokalni Razpošiljatelj transakcij. Vsak vnos vsebuje identifikator celice, ime postaje, ki predstavlja linijo (pot) do te celice in nazaj in še nekaj dodatnih informacij. Razpošiljatelj transakcij uporablja to tabelo za pravilno usmerjanje sporočil.

Na sliki 4.5. je struktura Tabele poti. Z njeno pomočjo usotovi, na katero linijo lokalnega vozlišča mora poslati sporočilo.



Slika 4.5.: Struktura Tabele poti

Simboli na sliki 4.5. pomenijo:

P$CELL je ime celice v EBCDIC kodi, do katere vodi pot.
P$LUNAME je ime postaje, na katere sklad sporočil je treba poslati sporočilo,da bo prišlo do celice v P$CELL.
P$#MSGS je število sporočil, ki so bila poslana po tej poti.
P$#MSGR je število sporočil, ki so bila sprejeta po tej poti.
P$#ERS je število napak, ki so bile usotovljene na tej poti.

TRANSAKCIJSKA TABELA ima vnos za vsako transakcijo, ki naj se izvaja v lokalni celici. Vsak vnos vsebuje transakcijski identifikator, ime programa, ki izvaja transakcijo in še nekaj dodatnih informacij. Razpošiljatelj transakcij uporablja to tabelo za pravilno dodeljevanje transakcijskih sporočil programom, ki jih izvajajo.

4.2.1 Usmerjanje transakcij

Izvor vsakega transakcijskega sporočila je Razpošiljatelj transakcij, ki nato usmeri transakcijsko sporočilo proti končnemu cilju (transakcijski program) neposredno ali pa preko drugih celic. Oddaljeni celici pošlje transakcijsko sporočilo preko postaje, ki predstavlja to celico (vozlišče). Ti podatki so navedeni v tabeli poti. Razpošiljatelju sporočil je transakcijsko sporočilo enako podatkovnemu sporočilu. Enako se postavi na sklad sporočil postaje poti. Razpošiljatelj transakcij torej uporablja funkcije Razpošiljatelja sporočil.

POSTAJA POTI je postaja, ki ne predstavlja nujno fizične zveze med dvema vozliščema (celicama), predstavlja pa logično povezavo med celicama. V vsaki celici Razpošiljatelj sporočil poskrbi za pravilno nadaljne usmerjanje transakcijskega sporočila. Poti so definirane v Tabeli poti tako, da je za celice, ki niso neposredno povezane vedno navedena Preferenčna pot, po kateri se usmeri transakcija če je v njeni glavi naslov vozlišču neznane celice.

EMISIJSKA TRANSAKCIJA ima namesto identifikatorja celice '**'. Ko jo Razpošiljatelj transakcij sprejme, jo izvrši v svoji celici, nato pa pošlje po Splošni poti naprej, če je definirana. Za tem jo pošlje še vsem ostalim sosednjim celicam, ki jih ima definirane v tabeli poti, pri čemer zamenja '**' z naslovom sosednje celice.

PRIMER : Na sliki 4.6. je konfisuracija štirih celic. Poti naj bodo definirane takole:

C1 in HT
C1 in C2
C1 in C3
C1 in C4 preko C2
C2 in C4

Preferenčne poti v celicah so:

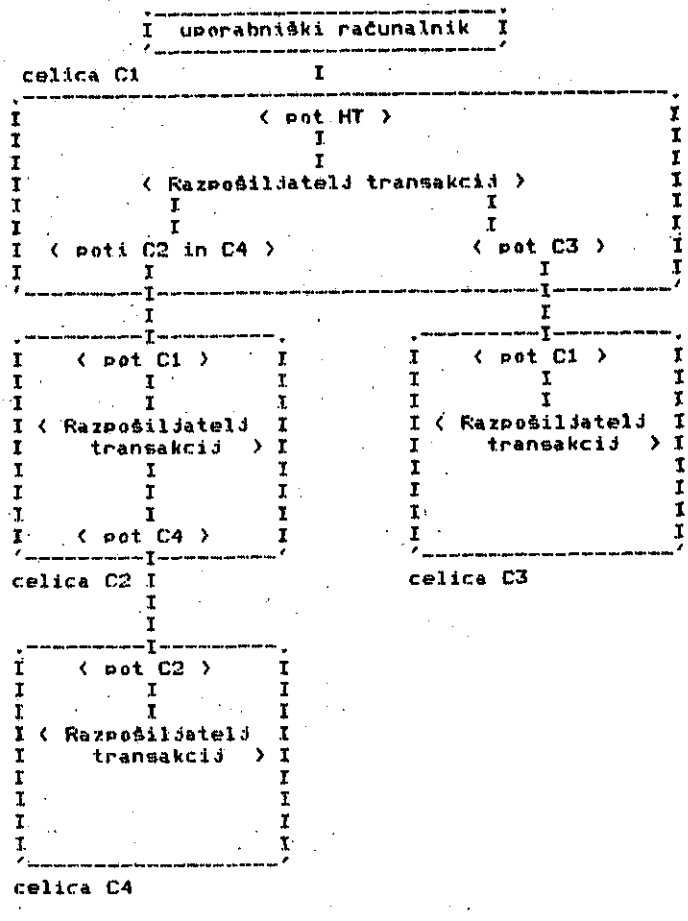
V celici 2 je C1
V celici 3 je C1
V celici 4 je C2

Pot transakcijskega sporočila iz C2 v C3 poteka takole:

1. Razpošiljatelj transakcij v celici C2 sprejme transakcijo s sklada sporočil in presleda identifikator celice. Ker se ne nanaša na njegovo celico (vozlišče), preišče tabelo poti, če se C3 nahaja v njej. Ker se C3 ne nahaja v tabeli, pošlje transakcijsko sporočilo po preferenčni poti proti Razpošiljatelju transakcij v C1.

2. Razpošiljatelj transakcij v C1 presleda identifikator celice v glavi sporočila. Ker naslov ni C1, preišče tabelo poti. Ker se C3 v njegovi tabeli ujema z C3 v glavi sporočila, pošlje transakcijsko sporočilo po poti proti C3.

3. Razpošiljatelj transakcij v C3 presleda identifikator celice v glavi sporočila. Spozna sporočilo in sa pošlje v izvajanje.



Slika 4.6. : pretok transakcijskega sporočila med celicami

5.0 PRIMERJAVA MED CF IN ISO NIVOJI

Proizvajalec CF-a ni opredelil nivojsko in najbrž tudi ni poskušal srediti cf v skladu z nivoji ISO za računalniške mreže. Vseeno pa se CF nivoji do neke mere prekrivajo s predlaganimi nivoji ISO. V CF nastopajo naslednji sradbeni bloki:

Priključne kartice, ki so vmesnik med programskim delom in linijo. Kartica in njen programski nadzor v operacijskem sistemu predstavljata virtualno zanesljivo linijo. Vsak sradbeni blok namreč deluje po principu 'sprejmi naloso in se odzovi, ko bo pravilno opravljena, ali pa javi napako nivoja nad seboj, zaradi katerega naloga ni bila pravilno opravljena'. Priključna kartica in njen programski nadzor predstavljata nasproti ostalemu delu CF izhod iz linijskega nivoja po ISO.

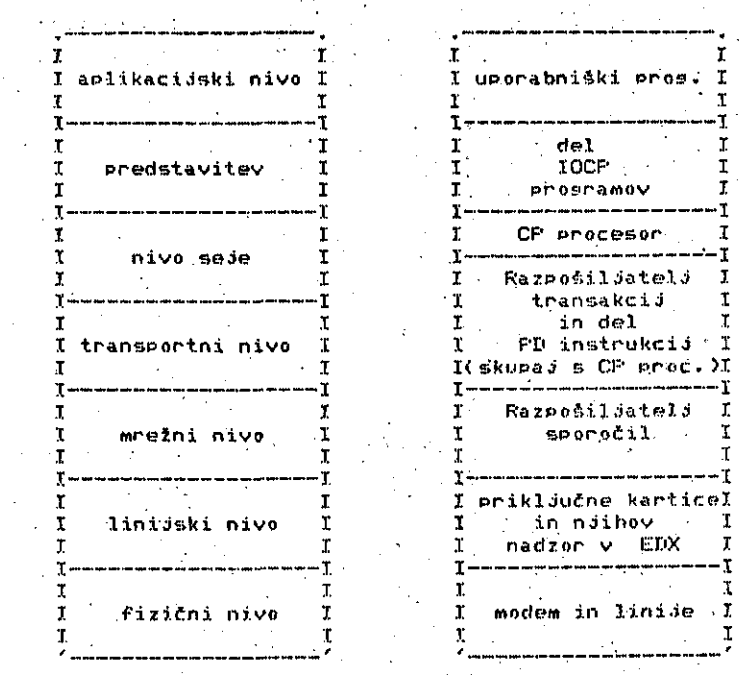
Razpošiljatelj sporočil opravlja naloso iskanja pravih poti za vse sporočila, ki so mu bila zaupana. Njegova naloga je tipična, saj mu enote predajajo sporočila in jih sploh ne zanima več, kako bodo našle naslovnika. Zato lahko Razpošiljatelja sporočil opredelimo kot mrežni nivo po ISO.

Razpošiljatelj transakcij sprejema transakcijska sporočila, ki so naslovljena na cilj poimensko. Omogoča emisijo sporočil na vse dostopne naslove. Za vsako oddajo transakcijskega sporočila zahteva usluso Razpošiljatelja sporočil. Ko sporočilo preda Razpošiljatelju sporočil, se zanj ne zanima več. Zaradi omenjenih funkcij lahko Razpošiljatelja transakcij uvrstimo v transportni nivo po ISO.

Procesor ukazov izvaja različne instrukcije okrog vzpostavljanja zvez med postajami in startanja postaj. Skupaj z Razpošiljateljem transakcij izvaja instrukcije o vzpostavljanju logičnih poti med postajami, omogoča vnos sporočil za naslovnika itd. Podatki so lahko vneseni s posebnim ukazom, lahko pa jih prebere iz datoteke, sprejme od uporabniškega programa ali od servisnega programa. Zaradi teh lastnosti bi sa lahko uvrstili v nivo seje po ISO.

IOCP programi skrbijo za prilagoditve med programsko opremo vozlišča in vsemi priključki strojne opreme na vozlišče. Sem sodijo ostala vozlišča, ki so priključena preko linijskih kartic, uporabniški računalniki, ki so priključeni preko posebnih linijskih kartic in terminalne enote, ki so tudi vezane na svoje kartice, bodisi z paralelnim ali serijskim prenosom. En del IOCP programov sprejema ukaze za vzpostavitev linije med postajami, ostali del pa skrbi za konverzijo med čistimi podatki končnih enot in CF sporočili. Zaradi te zadnje funkcije in emulacij raznih naprav (npr. 3270) bi lahko IOCP programe delno uvrstili v predstavitveni nivo po ISO.

Razdelitev je prikazana grafično na sliki 5.1.



Slika 5.1. : Poskus primerjave med nivoji ISO in sestavnimi deli CF

Opredelitev Razpošiljatelja sporočil v mrežni nivo ISO

Opredeljuje način servisa. Omogoča emisijo sporočil (Razpošiljatelju transakcij). Prenaša transakcije (Razpošiljatelju transakcij). Servisni program , Razpošiljatelj transakcij ali uporabniški program mu preda sporočilo in se zanj ne zanima več. Vrstni red oddaje sporočil na sprejemni strani ni zagotovljen, ker se jim lahko spreminja prioriteta in s tem vrstni red prihoda na cilj. Pošiljatelj ne preverja, ali je sporočilo prišlo na cilj. Vsako sporočilo nosi s seboj v glavi naslov ciljne postaje. V slučaju, da nosi sporočilo naslov, za katerega ne more najti naslovnika, odloži paket na postajo WASTE ali pa ga zavrže.

Algoritem za usmerjanje sporočil je neadaptiven. Usmerja na podlagi tabel, ki se nahajajo v vsakem vozlišču. Usmerjanje je torej statično.

6.0 ZAKLJUČEK

CF je kompleksen programski produkt s spremljevalno strojno opremo v obliki priključnih kartic. Vozlišče (S/1) ima lahko čisto funkcijo vozlišča ali pa lahko obenem posanja uporabniške programe. Povezava je lahko najbolj primitivna zveza med uporabniškim računalnikom in S/1 v točka-točka obliki, lahko se oblikuje lokalna mreža v obliki obroča ali pa kompleksna mreža s S/1 kot izključno vozlišča in nanje priklopljenimi uporabniškimi računalniki.

Paketi podatkov se usmerjajo statično in neadaptivno na podlagi tabel v vsakem vozlišču, ki pa jih je moč spreminjati s servisnimi programi. Preklapljanje paketov omogoča prenos do 32k v enem paketu.

SEZNAM UPORABLJENIH SIMBOLOV

S/1 = računalnik IBM Series/1 ali Sistem/1
CF = Communications Facility, komunikacijski paket za EDX
CM = Communications Monitor, komunikacijski paket za RPS
EDX = Event Driven Executive, operacijski sistem za S/1
RPS = Realtime Programming System, operacijski sistem za S/1
LCC = Local Communications Controller, komunikacijski obroč
IOCP= Input/Output Control Program, vmesnik za V/I operacije
SNA = Systems Network Architecture, mrežna arhitektura IBM
SDLC= Synchronous Data Link Control, linijski protokol
BSC = BiSynchronous Communication, linijski protokol
ECB = Event Control Block, Blok za signalizacijo dogodka

SEZNAM UPORABLJENIH VIROV:

1. CF Design and Installation Guide, IBM Corp., S/1 Program Product Development, 1501 California Avenue, Palo Alto, CA 94304, Oktober 1982

2. CF Debugging Guide, isti naslov kot pod 1., Februar 1982

3. A.S.Tannenbaum: Computer Networks, Pratience-Hall Inc., Englewood Cliffs, New Jersey, 1981

INFORMATICA 2/1984

PRISTUP DINAMIČKOM RASPOREDJIVANJU ZADATAKA U PRSTENASTOJ MREŽI RAČUNALA

KUKRIKA MILAN

ELEKTROTEHNIČKI FAKULTET BANJA LUKA

UDK: 681.324

SADRŽAJ - Pri ažuriranju zalihosnih kopija baze podataka trebalo bi primijeniti mehanizme koji bi garantirali konzistentnost baze podataka, a istovremeno što više smanjili neproduktivno vrijeme potrebno za sinhronizaciju pojedinih računala u sistemu. Dat je prijedlog za ažuriranje zalihosne baze podataka u raspodijeljenom sistemu koji podržava uporedno izvodjenje zadataka koji pristupaju bazi. Prednost predloženog rješenja je u minimiziranju dodatnog neproduktivnog vremena potrebnog za sinhronizaciju manipulatora bazom.

ABSTRACT - AN APROACH TO MULTIPLE COPY UPDATE IN DISTRIBUTED SYSTEMS. The redundant update problem is to develop tecniques for updating redudantly stored data that preserve database consistency and minimize intercomputer synchronization. An suggestion for updating of redundant data base in distributed system is given, which enables parallel data base task execution. The vantage of the suggested solution is minimisation of communication overhead.

1. UVOD

Zahtjevi za ostvarivanjem stvarnovremenog rada dovode nas do problema podjele vremena i mjesta korištenja sredstava za izvodjenje pojedinih zadataka, jer nam je zbog povećanja brzine odgovora, cilj da se što više zadataka izvodi uporedo. Procesor sekvencijalno sprovodi obradu definiranu programom, pa se izvodjenje skupa zadataka može povjeriti ili moćnom procesoru, čija će brzina ostaviti dojam paralelnog izvodjenja, ili pak zadatke razdijeliti na više računala. Ekonomska razamtranja karakteristika sklopovskih i programskih komponenti pokazuje da bi moglo biti ekonomičnije umjesto velikih, složenih samostalnih računala koristiti računala u paralelnom radu.

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

Raspodijeljeni sistemi, za koje vrijede razmatranja provedena u ovom radu, odlikuju se slijedećim karakteristikama:

1. uključuju proizvoljan broj sistemskih i korisničkih zadataka,
2. arhitektura je modularna sa mogućnošću proširenja,
3. komuniciranje se odvija izmjenom poruka putem zajedničke komunikacijske strukture (isključujući zajedničku memoriju),
4. pretpostavlja se varijabilno kašnjenje u prijemu poruka jer su učesnici medjusobno udaljeni,
5. na nivou sistema definirano globalno upravljanje omogućava kooperaciju i sinhronizaciju zadataka koji suradjuju na ograničenom skupu sredstava.

Zbog karakteristike 5 primjena centralizovanog upravljanja kod sistema koji posjeduju karakteristike 1,2 i 3 ne bi dovela do optimalnih rezultata. Cijeli sistem u tom slučaju bio bi podložen velikim komunikacijskim kašnjenjima i ovisan o mogućnostima i trenutnom stanju središnjeg vodstva.

Navedeni razlozi nameću da se za raspodijeljene sisteme rješenje traži u decentraliziranoj upravljačkoj i informacijskoj strukturi, bez obzira što dekompozicija unosi nove informacijske tokove u sistem vezane uz funkcionalnu interakciju blokova upravljanja.

Decentralizacija informacijske strukture povlači za sobom problem očuvanja konzistentnosti višestrukih kopija podataka. U radu je prvo predstavljen pristup konzistentnom ažuriranju klasične, centralizirane baze podataka, da bi se na osnovu tih razmatranja dao prijedlog ažuriranja raspodijeljene, potpuno zalihosne baze podataka.

2. O BAZI PODATAKA

Za izvodjene zadataka potrebna su sredstva. Svako sredstvo može se apstrahirati putem skupa podataka koji ga predstavljaju (njegovom reprezentacijom) te skupom dozvoljenih operacija ili funkcija, kao i pridruženih parametara koji se mogu primijeniti nad tim podacima.

Za dva sredstva kažemo da su istog tipa ako i samo ako imaju identične specifikacije, tj. ako im je ista

reprezentacija i skup definiranih operacija. Operacije nad sredstvima izvršavaju moduli koje ćemo nazvati manipulatorima.

Bazu podataka možemo predstaviti sa

$$D = (X, Y, Z, \ldots)$$

pri čemu veličina pojedinih slogova (X,Y,...) ne igra nikakvu ulogu. Slog X može poprimiti vrijednosti iz dom (X), slog Y iz dom (Y) itd., a trenutno stanje baze podataka je jedan od elemenata:

$$dom(D) = dom(X) \times dom(Y) \ldots \times dom(Z)$$

Neka je $Z=(Z_1, Z_2, \ldots Z_b)$ skup svih zadataka koji pristupaju bazi podataka. Zadatak i (i=1,2,... b) čine struktura podataka i niz akcija f_i čije izvodjenje ima za posledicu transformaciju u strukturi podataka.

Definirajamo dvije operacije:

- R: $(Z_i \mid i = 1,2\ldots) \rightarrow 2^D$, koja odredjuje dio (podskup) baze podataka, nazovimo ga RS_i, iz kojeg će se uzeti podaci potrebni za izračunavanje i

- W: $(Z_i \mid i = 1,2\ldots) \rightarrow 2^D$, koja odredjuje dio (podskup) baze podataka, nazovimo ga WSi, u koju će se upisati rezultati.

Dakle, za potrebe našeg razmatranja zadatak Z_i (i = 1,2,...n) možemo apstrahirati operacijom čitanja R tokom koje se uzimaju vrijednosti varijabli iz skupa $RS \subset D$, nužnih za izračunavanje akcija f_i, a zatim slijedi operacija upisivanja novih vrijednosti promjenjenih varijabli W, koja ažurira vrijednosti varijabli iz skupa $WS \subset D$. Skupovi RS i WS označavaju se kao dijelovi baze podataka koji će biti pročitani, odnosno upisani, a akcije f_i mogu se predstaviti kao preslikavanje

$$f_i : dom (RS) \rightarrow dom (WS)$$

Izvodjenje zadataka (R i W operacija) na bazi podataka bi trebalo definirati tako da se pod predpostavkom da je ulazni skup podataka konzistentan, takav bude i izlazni skup podataka. Skup operacija na bazi podataka promatraćemo na osnovu sistema TAU=(D,Z,RS,WS) gdje je D baza podataka, Z skup zadataka, RSi je definiran sa $(Z_i \mid i = 1,2\ldots) \rightarrow 2^D$ a WS_i sa $(Z_i \mid i=1,2\ldots) \rightarrow 2^D$.

Razmotrimo kada je dozvoljen samo sukcesivni pristup bazi podataka, a kada je moguće akcije pojedinih zadataka interpolirati. Raspored izvršenja skupa naredbi predstavljen je sa oznakom "log". Log predstavlja parcijalno uredjenje R i W operacija odredjujući redoslijed kojim će operacije biti izvršene na bazi podataka. Dodatno ograničenje je da sve konfliktne operacije moraju biti totalno uredjene. (Parcijalno uredjenje je binarna relacija < koja je refleksivna, (a < a) antisimetrična (a < b i b < a implicira da je a = b) i tranzitivna (a < b i b < c implicira da je a < c). Totalno uredjenje je parcijalno uredjenje u kojem je svaki par elemenata parcijalno uredjen. (tj. za V a,b je ili a < b ili b < a).

Za dati sistem TAU = (D,Z,WS) definira se log L=s1,s2, ... kao slijed simbola K iz alfabeta S:

$S = (R_i, W_i \mid T_i \in T)$ uz ograničenja:

- da se svaki simbol iz S pojavi u K samo jednom i
- da R_i prethodi W_i za $\forall i$.

Očito je da postoji veliki izbor načina rasporedjivanja zadataka.

Različiti pristupi rasporedjivanju mogu biti primijenjeni, uzrokujući i različita stanja baze podataka.

U općem slučaju dva loga su ekvivalentna ako njihovo izvršenje rezultira u istom konačnom stanju baze podataka.

Ograničenja konzistentnosti zahtijevaju da se izvodjenje R i W operacija na bazi podataka definira tako da se pod pretpostavkom da je ulazni skup podataka konzistentan, takav bude i izlazni skup podataka.

Jedini potpuno siguran način očuvanja konzistentnosti je da se jednom startani zadatak ne prekida. Serijski (sukcesivni) log predstavlja postupno uredjenje redoslijeda izvodjenja zadataka tako da svaki par zadataka sve akcije jednog zadatka (apstrahirane sa R_i i W_i) prethode prvoj akciji drugog tj.

$$K = Ri_1\ Wi_1\ Ri_2\ Wi_2 \ldots R_{in}\ W_{in}$$

Takav način rada bi se u raspodijeljenim sistemima pokazao kao vrlo neefikasan.

Pretpostavimo da bi nam odgovarao sljedeći, izmjeničan (uporedan) redoslijed izvršavanja zadataka Z_1, Z_2 i Z_3:

$$K = R_1\ R_2\ W_1\ R_3\ W_3\ W_2$$

Dakle, sa izvršavanjem zadatka Z_2, bi se počelo prije nego što je završen zadatak Z_1, a sa izvršavanjem zadatka Z_3 prije završetka zadatka Z_2.

Ovakav raspored izvodjenja zadataka dao bi korektne rezultate, tj. bio bi funkcionalno ekvivalentan postupnom izvodjenju pod uvjetom da je:

$$R_2 \cap W_1 = 0 \text{ i } R_2 \cap W_3 = 0$$

a ekvivalentni postupni pristup koji garantira konzistentnost bi bio:

$$R_1\ W_1\ R_3\ W_3\ R_2\ W_2$$

Na osnovu ovog primjera možemo zaključiti da je izmjeničan pristup bazi podataka korektan ako je funkcionalno ekvivalentan postupnom pristupu. Takav izmjeničan pristup označićemo sa SR (serializable), i možemo garantirati da

će izvršenje skupa zadataka rezultirati u konzistentnom stanju baze podataka ako je redoslijed njihovog izvodjenja SR log (9,10,11,12).

4.1. Sistemska baza podataka u raspodijeljenom sistemu

U raspodijeljenom sistemu i sistemska baza podataka mora biti raspodijeljena. Pri tome mogu postojati sljedeće mogućnosti:

- striktno podijeljena baza - ne postoje kopije pojedinih komponenti
- potpuno zalihosna baza - postoje raspodijeljene kopije cjelokupne baze podataka
- djelomično zalihosna baza - postoje raspodijeljene kopije nekih od komponenti baze

Višestruke kopije cijele baze podataka ili pak nekih njenih dijelova skraćuju vrijeme pristupa podacima (pa samim tim i odziva sistema), a doprinose i povećanoj pouzdanosti sistema u slučaju kvara pojedinih dijelova.

Raspodijeljenu bazu podataka možemo predstaviti sa

$$DB = (x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots, z_1, z_2, \ldots)$$

pri čemu su x_i kopije sloga X, y_j kopije sloga Y itd.

Ažuriranje raspodijeljene baze podataka mora biti tako provedeno da se raspodijeljene komponente, kopije pojedinih komponenti ili pak kopije cijele baze održe konzistentnim pri čemu se misli na (3):

- uzajamnu konzistentnost zalihosnih kopija
- internu konzistentnost svake kopije

Za kopije baze podataka ili kopije dijelova baze podataka kaže se da su medjusobno konzistentne ako su identične. Identičnost kopija u slučaju svakog pristupa jednoj od njih je vrlo strog zahtjev (znatno povećanje komunikacionih i sinhronizacionih mehanizama) pa se traži da kopije budu identične nakon što se zaključe sve akcije pristupa nekoj od njih.

Interna konzistentnost svake kopije podrazumijeva (kao i u slučaju konvencionalne baze podataka) da se podržavaju:

- koncept semantičkog integriteta, tj. da spremljeni podaci vjerno odražavaju sredstva koja predstavljaju. Prije svakog pristupa sredstvima trebalo bi utvrditi da li je udovoljeno zahtjevima semantičkog integriteta.

- koncept neprekidivih (nedjeljivih) operacija koji zahtijeva da ili nijedna ili sve akcije odredjene operacije budu provedene, tj. da pristup bazi podataka bude funkcionalno ekvivalentan postupnom pristupu.

Da zaključimo - održavanje raspodijeljene baze podataka sa djelimičnom ili potpunom zalihošću (baze podataka sa višestrukim kopijama) zahtijeva kreiranje mehanizama koji će garantirati medjusobnu konzistentnost da bi se održale identične, interno konzistentne kopije u slučaju konkurentnog ažuriranja njenih dijelova.

Problem održavanja konzistentnosti raspodijeljene baze podataka koja sadrži višestruke kopije svodi se na problem koordinacije akcija skupa raspodijeljenih manipulatora koji im pristupaju.

Pošto akcije inicirane od strane različitih zadataka mogu biti uzajamno povezane tako da mogu dovesti do nekorektnih rezultata (tj. uzajamne ili interne nekonzistentnosti kopija) postavlja se uvjet nedjeljivosti operacija zadataka koji se može postići bilo neposrednim dogovorom samih zadataka bilo odredjivanjem redoslijeda kojim će se operacije odvijati.

Skup manipulatora odgovara skupu memorijskih procesora čije djelovanje se sastoji u čitanju i pisanju formirajući korake izvršne faze zahtijevane operacije. Memorijski procesori su odgovorni za manipuliranje raspodijeljenim podacima.

U slučaju potpuno zalihosne baze podataka memorijski procesor je pridružen svakoj kopiji baze podataka. Ažuriranje zahtjeva podrazumijeva da zadatak inicira identične neprekidive akcije na svim memorijskim procesorima tj. da budu izvršene ili sve akcije ili nijedna. U slučaju striktno raspodijeljene baze podataka izvršenje rezultira u iniciranju različitih akcija na odgovarajućim memorijskim procesorima, a djelomična zalihost zahtijeva identične akcije na odgovarajućim memorijskim procesorima (onima koji sadrže kopije dijelova baze).

U općem slučaju ako su dva ili više manipulatora medjusobno interaktivni zahtijeva se:

- korektna sukcesivnost akcija svakog od memorijskih procesora kako bi se garantirala interna konzistentnost i

- ako postoji interaktivnost sa više memorijskih procesora mora se postići kompatibilna sukcesivnost koja će garantirati internu konzistentnost striktno raspodijeljene baze podataka, ili pak uzajamnu konzistentnost ako se radi o zalihosnoj bazi podataka.

U nastavku ćemo razmotriti postupke za konzistentno ažuriranje zalihosne baze podataka.

Zadaci se u slučaju potpuno zalihosne baze podataka izvode u dva koraka prema sljedećim pravilima:

U toku prvog koraka zadatak se izvodi na računalu na kojem je i iniciran koristeći vlastitu kopiju baze podataka. Sve R i W operacije se obavljaju lokalno, a lokalni manipulator obezbjedjuje potrebnu sinhronizaciju generirajući i održavajući listu svih promjena baze podataka. Izvodjenje zadatka t na računalu m označeno je kao primitivna operacija L_m^t.

- Kada je prvi korak dovršen u drugom koraku lokalni manipulator obavještava sve ostale manipulatore o svim promjenama (1). To može imati za posljedicu veliko dodatno komunikacijsko opterećenje. Problemi komuniciranja su detaljno opisani u (4,5,6,7). Konačno, ažuriranje

kopija baze podataka od strane manipulatora m označava se kao primitivna operacija U_n^t, gdje je n broj preostalih kopija. Dakle, izvodjenje nekog zadatka može se predstaviti kao jedna operacija L_m^t, koju slijedi onoliko operacija U_n^t, koliko ima kopija baze podataka.

Primitive L_m^t i U_n^t su neprekidive dok kombinacija L i U operacija zadatka t ne mora biti neprekidiva.

Na pr. rasporedjivanje izvodjenja tri zadatka t_1, t_2 i t_3 na sistemu sa tri kopije baze podataka može se odvijati prema sljedećem rasporedu:

$$K = L_2^1 \; U_1^1 \; U_3^1 \; L_1^2 \; U_2^2 \; L_1^3 \; U_2^3 \; U_3^3 \; U_3^2$$

Sa izvodjenjem trećeg zadatka počelo se na prvom računalu prije nego što su unesene promjene nastale izvodjenjem zadatka 2 na prvom računalu (U_3^2) što bi moglo dovesti do nekonzistentnog stanja baze podataka.

U prethodnom poglavlju smo istakli da nam je zbog povećanja brzine i iskorištenja sredstava cilj postići što veću uporedivost zadataka. Pitanje na koje bi pri tome trebalo odgovoriti je kada je dozvoljeno preklapanje L i U operacija, što opet ovisi o vrsti operacije (L ili U), o tome da li se operacije izvode na istom računalu i o preklapanju RS i WS podskupove zadataka.

L i U operacije koje se izvode na istom računalu se ne mogu preklapati (L-U i U-L) ako L čita podatke koji će se ažurirati od strane U, tj. preklapanje je dozvoljeno ako je $RSt1 \cap WSt2 = 0$. Uslov $WSt1 \cap WSt2 = 0$ se ne postavlja zbog mehanizma vremenskog kašnjenja/16/.

L i U operacije koje se izvode na različitim računalima mogu se po volji preklapati, izuzev ako pripadaju istom zadatku (L uvijek mora doći prije U). U operacije (U-U) se mogu uvijek preklapati. L akcije se mogu preklapati (L-L) ako vrijedi

$$RS(L_1) \cap WS(L_2) = 0 \text{ i}$$

$$WS(L_1) \cap WS(L_2) = 0$$

Redoslijed izvodjenja L i U operacija u nekom računalu označava se kao lokalni log. Globalni log zadužen za konzistentnost cjelokupne raspodijeljene baze podataka predstavlja skup lokalnih logova. Globalni log se kreira u dva koraka. Prvo se definiraju lokalni logovi za svako računalo kombiniranjem mogućih sekvenci izvodjenja. Zatim se lokalni logovi integriraju u globalni log tako da je efekat izvršenja globalnog loga za svako računalo ekvivalentan originalnom lokalnom logu. Globalni log je dakle, jedan od mogućih puteva izvodjenja zadataka na raspodijeljenoj bazi podataka. Pri tome se mora voditi računa da se za zadatak t sve operacije ažuriranja U_n^t mogu pojaviti u logu tek nakon operacije L_m^t.

Operacije L i U mijenjaju stanja baze podataka, pa globalni log G_i možemo definirati kao funkciju g_i koja prebacuje bazu podataka iz jednog stanja DS_1 u drugo stanje DS_2, tj.:

$$g_1(DS_1) = DS_2$$

Za dva globalna loga G_1 i G_2 kažemo da su ekvivalentni ako uključuju iste L i U operacije i ako je

$$g_1(DS_1) = g_2(DS_2) \text{ za } \forall DS_1 .$$

Globalni log GS u kojem svaku operaciju L_m^t neposredno slijede sve njegove operacije U_n^t nazivamo serijskim (postupnim) globalnim logom. Uporedni globalni log koji je funkcionalno ekvivalentan serijskom slobodnom logu označićemo kao SR-globalni log. SR-globalni log dozvoljava preklapanja L i U operacija različitih zadataka, ali dovodi bazu podataka u isto (konzistentno) konačno stanje kao i serijski globalni log.

Na osnovu formalnih analiza (1,2,3) ustanovljeno je da neki tipovi zadataka zahtijevaju različite nivoe sinhronizacije koji mogu biti izraženi na osnovu n jednostavnih algoritama - protokola.

Pristup opisan u (2) predlaže korištenje jednog od četiri različita protokola, od najjednostavnijeg P_1 do najsloženijeg P_4, u zavisnosti o situaciji i zahtijevanom nivou sinhronizacije. Najjednostavniji od ovih protokola P_1 ne uvodi skoro nikakvo neproduktivno sinhronizaciono vrijeme, dok se to vrijeme znatno povećava korištenjem protokola P_2-P_4. Odluka o tome na koje zadatke primijeniti koji protokol bazirana je na formalnoj matematskoj analizi o načinima interakcije medju zadacima.

Zadaci su prema nivou sinhronizacije (podacima kojima pristupaju) podijeljeni u klase C. Klasa C_m^i je definirana kao skup zadataka pridruženih računalu m uz kojeg je locirana kopija baze podataka, a svaki od zadataka pristupa samo podskupovima RS (C_m^i) i WS (C_m^i).

Da bi se obezbijedila konzistentnost baze podataka može se koristiti tzv. L-U graf. L-U grafovi mogu prikazati sve moguće kombinacije globalnih logova za konačan skup zadataka Z. L-U graf čine parovi čvorova označeni sa L i U za svaki zadatak iz klase C_m^i. L čvor reprezentira L operacije u C_m^i dok U čvor predstavlja U operacije u C_m^i. Čvorovi se povezuju prema sljedećim pravilima:

Čvor L_m^i povezuje se sa ostalim čvorovima na osnovu jedne od sljedećih kombinacija:

I1 - vertikalnom linijom sa U čvorom iste klase

I2 - horizontalnom linijom sa čvorom L_m^j druge klase na istom računalu m ako je

$$W(C_m^i) \cap R(C_m^j) \neq 0, \quad i \neq j \quad \text{ili}$$

$$W(C_m^j) \cap W(C_m^i) \neq 0, \quad i \neq j$$

I3 - horizontalnom linijom sa čvorom L_n^j na drugom računalu ako je

$$W(C_m^i) \cap W(C_n^j) \neq 0, \quad m \neq n$$

I4 - dijagonalnom linijom sa U-čvorom druge klase na drugom računalu ako je

$$R\,(C_m^i) \cap W(C_n^j) \neq 0, \quad m \neq n$$

U_m^i povezan je sa drugim čvorovima prema:

I5 - dijagonalnom linijom sa L_n^j ako je

$$W\,(C_m^i) \cap R(C_n^j) \neq 0, \quad m \neq n$$

Nakon što se konstruira L-U graf za dati sistem izabire se jedan od 4 protokola prema sljedećim pravilima:

S1. Ako se ne izvodi L_m^i iz klase C_m^i, ili ako se on izvodi uz direktnu vezu sa drugim klasama prema kombinacijama (I2, I2), (I3,I3), (I2,I3), (I2,I5), (I3, I5) izabraće se najmanje ograničavajući protokol P_1. Protokol P_1 osigurava neprekidnost L_m^i postavljanjem lokalne raspodijeljene zabrane pristupa na podskupu RS i lokalne isključive zabrane pristupa na skupu WR za svaki zadatak $t \in C_m^i$.

S2. Ako se izvodi L_m^i, a postoji direktna veza sa ostalim klasama prema kombinacijama (I4, I4) izabire se protokol P_2 za svaki zadatak $t \in C_m^i$.

S3. Ako se izvodi L_m^i, a postoji direktna veza sa ostalim klasama prema kombinacijama (I2,I4), (I3,I4), (I4,I5) izabire se protokol P_3.

Protokoli P_2 i P_3 su identični, osim što se razlikuju u vremenskim intervalima.

4. ZAKLJUČAK

Sa stanovišta kreiranja poogramske podrške raspodijeljene sisteme možemo tretirati kao niz uporednih zadataka koji medjusobno komuniciraju. Ekvivalentnost množine pojedinačnih djelovanja i jedinstvene sistemske svrhe se u višeprocesorskim sistemima može jednostavno osigurati preslikavanjem, kod multiprogramskih sistema dobro razradjenih, mehanizama centralnog upravljanja. U raspodijeljenim sistemima se ne može ostvariti uvjet postojanja zajedničkog memorijskog prostora, tako da je jedini način komunikacije i sinhronizacije medjuzavisnih zadataka razmjena poruka. Pošto su učesnici medjusobno udaljeni mora se pretpostaviti varijabilno kašnjenje u prijemu poruka. Ako bi se i kod raspodijeljenih sistema primijenili centralistički mehanizmi, cijeli sistem bi bio podložan velikim komunikacijskim kašnjenjima i ovisan o mogućnostima i trenutnom stanju središnjeg vodstva. Rješenje se mora potražiti u decentraliziranom upravljanju, bez obzira što i ovakav pristup unosi nove informacijske tokove u sistem vezane uz interakciju zadataka koji se izvode na različitim mjestima u sistemu.

Orjentacija ka decentraliziranom upravljanju povlači za sobom i decentralizaciju informacijske strukture. U radu su predstavljene metode za kreiranje i ažuriranje baze podataka u raspodijeljenim računarskim sistemima. Pri ažuriranju zalihosnih kopija baze podataka trebalo bi primijeniti mehanizme koji bi garantirali konzistentnost baze podataka, a istovremeno što više smanjili neproduktivno vrijeme potrebno za sinhronizaciju pojedinih računala u sistemu.

Problem održavanja konzistentnosti raspodijeljene baze podataka koja sadrži višestruke kopije svodi se na problem koordinacije akcije skupa raspodijeljenih manipulatora koji im pristupaju. Skup manipulatora odgovara skupu memorijskih procesora čije djelovanje se sastoji u čitanju i pisanju formirajući korake izvršne faze zahtijevane operacije. Memorijski procesori su odgovorni za manipuliranje raspodijeljenim podacima.

U slučaju potpuno zalihosne baze podataka memorijski procesor je pridružen svakoj kopiji baze podataka. Ažuriranje zahtjeva podrazumijeva da zadatak inicira identične neprekidive akcije na svim memorijskim procesorima tj. da budu izvršene ili sve akcije ili nijedna. U slučaju striktno raspodijeljene baze podataka izvršenje rezultira u iniciranju različitih akcija na odgovarajućim memorijskim procesorima, a djelomična zalihost zahtijeva identične akcije na odgovarajućim memorijskim procesorima (onima koji sadrže kopije dijelova baze).

Da bi se što je moguće više smanjilo opterećenje komunikacijske mreže zadaci su prema nivou sinhronizacije (podacima kojima pristupaju) podijeljeni u klase C. Klasa C_m^i je definirana kao skup zadataka pridruženih računalu m uz kojeg je locirana kopija baza podataka a svaki od zadataka pristupa samo podskupovima RS (C_m^i) i WS (C_m^i). Odluka o tome na koje zadatke primijeniti koji od četiri protokola zasnovana je na formalnoj matematskoj analizi o načinima interakcije medju zadacima.

Prednost predloženog rješenja je u minimiziranju dodatnog neproduktivnog vremena potrebnog za sinhronizaciju manipulatora bazom.

LITERATURA

1. Bernstein, A. et all: "Analyzing Concurrency control algorithms wen user and system operations differ", IEEE transactions on software engineering, may 1983.
2. Bernstein, A. et all: "The concurrency control mechanism of SDD-1: A system for distributed databases", IEEE transactions on software engineering, vol. se-4 no. 3, (may 1978).
3. Parker, D: "Detection of mutual inconsistency in distributed systems" IEEE transactions on software engineering, may 1983.
4. Keki, I: "Methodology for the design of communication networks and the distribution of data in distributed supercomputer systems", IEEE transactions on computers, vol. c-31, no. 5, (may 1982).
5. Lenahan, J.: "Performance of cooperative loosely coupled microprocessor architecture in an iteractive data base task", IEEE transactions on computers, vol. c-29, no. 2, (feb. 1980).

6. Chen, P. et al.: "Optimal design of distributed information systems" IEEE transactions on computers, vo vol. c-29, no.12, (dec. 1980).

7. Gardarin, G and Chu, W.: "A distributed control algorithm for reliably and consistently updating replicated databases", IEEE transactions on computers, vol. c-29, no. 12, (dec. 1980).

8. Hevner, A.: "Query processing in distributed database systems" IEEE transactions on software engineering, vol. se-5 no.3, (may 1979).

9. Hewitt, C.: "Specification and proof techniques for serijalizers", IEEE transactions on software engineering, vol. se-5, no.1, (jan. 1979).

10. Bernstein,P. et all: "Formal aspects of serializability in database concurrency control", IEEE transactions on software engineering, vol. se-5, no.3, (may 1979).

11. Gardarin, G. et al. "Proving consistency of database transactions written in extended Pascal", IEEE transactions on software engineering, vol. se-8, no.4, (july 1982).

12. Stonebaker, M.: "Concurrency control and consistency of multiple copies of data in distributed INGRES", IEEE transactions on software engineering, vol. se-5, no.5, (may 1979).

13. Muntz, R.: "Locking and deadlock detection in distributed databases", IEEE transactions on software engineering, vol. se-5, no.5, (may 1979).

14. Voss, K.: "Using predicate/transition nets to model and analyse distributed database systems" IEEE transactions on software engineering, vol. se-6, no.6, (nov. 1980).

15. Minoura, T.: "Resilient extended true-copy token sheme for a distributed database system", IEEE transactions on software engineering, vol. se-8, no.3, (may 1982).

16. Thomas, R.: "A solution to the concurency control problem for multiple copy databases", COMPCOM 78 (1978).

INFORMATICA 2/1984

RAČUNALNIŠKI PROGRAMI ZA POSLOVNO PLANIRANJE

IVAN MEŠKO

VISOKA EKONOMSKO-KOMERCIALNA ŠOLA MARIBOR

UDK: 681.3.06

Sestavljanje poslovnega plana zahteva med drugim mnogo tehničnega dela, ki pogosto tako okupira odločevalce, da za analizo plana ne ostane dovolj časa. Tehnični del lahko prenesemo na računalnik, ki ga opravi hitreje in natančneje, strokovne službe in odločevalci pa se zato lahko posvetijo preučevanju in pravilni realizaciji planskih odločitev. S pomočjo računalnika lahko pri planiranju upoštevamo mnogo več pogojev in povezav, zato je tako dobljen plan kvalitetnejši. Večina del se pojavi samo pri prvi uporabi računalniškega programa, zato so prihranki na času tem večji, čim večkrat sestavljamo plan oziroma delamo rebalans plana.

COMPUTER PROGRAMS FOR BUSINESS PLANNING. Preparing a business plan requires among others a lot of administrative work what usually occupy decision-makers so hard that there left no more time for analysis of the plan. Replacing administrative repetitive work by computer which do this work faster and more accurate specialist services and decision-makers can dedicate now more time to analyse consequences and to implementing of planned decision. Using a computer planning is improved as one can take into account considerable more conditions and connections. As the most work in connection with the planning by computer appears only at the first application it is obvious that time savings are bigger at repeating the planning or rebalancing the plan.

Poslovni plan je mogoče sestavljati s pomočjo računalnika le, če poslovnemu procesu priredimo model. V ta namen moramo poslovni proces razčleniti na aktivnosti in elemente. Aktivnosti predstavljajo načine nabave, proizvodnje in prodaje, pod pojmom elementi pa tukaj razumemo elemente poslovnega procesa, polizdelke in končne izdelke. Aktivnosti moramo odbrati tako, da bo njihova sinteza predstavljala celoten poslovni proces, za vsako aktivnost pa bodo definirani normativi potrošnje vhodnih elementov in normativi proizvodnje izhodnih elementov. Za vsako aktivnost moramo imeti tudi merske enote, kajti zanimali se bomo za količine aktivnosti, to je elementarnih procesov. Razčlenjenemu poslovnemu procesu je mogoče prirediti ovrednoteni graf [2] in tako dobiti nazorno sliko poslovnega procesa. Vedno seveda ne bo mogoče predočiti celotnega poslovnega procesa, saj imamo lahko nekaj tisoč elementov in aktivnosti. Tedaj lahko predočimo osnovni proces in nekaj tipičnih specialnih procesov.

Ko je poslovni proces razčlenjen in nazorno predočen, vsaki aktivnosti in vsakemu elementu priredimo šifro, ki sme imeti največ 8 znakov. Pri izboru šifer moramo paziti na to, da bo mogoče elemente in aktivnosti smiselno sortirati in na ta način prispevati k preglednosti računalniških izpisov. Poleg tega pa morajo biti šifre za elemente ločene v skupine za vhodne elemente poslovnega procesa, polizdelke in končne izdelke.

Glavnino podatkov tvorijo tehnološki podatki, to so normativi. Za vsako aktivnost moramo podati normative vseh vstopnih in izstopnih elementov. Za vsak normativ povemo šifro aktivnosti in šifro elementa, iz načina vpisa normativa pa mora biti razvidno, ali se normativ nanaša na vstopni ali na izstopni element.

Za elemente moramo podati minimalno in maksimalno možno nabavljeno oziroma prodano količino in ceno, iz šifre elementa pa mora biti razvidno ali element nabavljamo ali prodajamo. Z maksimalno količino vstopnega elementa poslovnega procesa zagotovimo, da med izvajanjem poslovnega procesa ne bomo potrebovali več tega elementa, kot ga imamo na razpolago. Z maksimalno količino končnega izdelka pa dosežemo, da ne bomo imeli odvečnih količin končnih izdelkov, ki jih ni mogoče prodati oziroma uporabiti. Pomena vseh minimalnih in maksimalnih količin v splošnem primeru ni primerno opisovati, saj je v konkretnih primerih lahko razumljiv.

Poleg navedenih podatkov nastopajo še različni drugi podatki, le teh pa ni mnogo, zato lahko z njimi model ročno dopolnjujemo.

Pri prvi uporabi modela je pomembna kontrola podatkov in preverjanje pravilnosti razčlenitve poslovnega procesa. Pri tem delu nam je v veliko pomoč graf. Ko so podatki prekontrolirani in vneseni na primeren medij, na primer na diskete, začnemo z izvajanjem programa za kreiranje matematičnega modela in urejene podatke oziroma parametre modela skupaj z ročno dodanimi parametri spravimo na trak. Podatke s tega traku nato uporabimo pri programu za linearno optimiranje, ki ga izvajamo, ko je trak s podatki pripravljen. Ko tudi ta program konča s delom, moramo najprej ugotoviti, ali so v podatkih logične ali formalne napake, zaradi katerih s pomočjo rezultata ni mogoče sestaviti plana in po potrebi popravimo podatke in ponovno izvajamo program za linearno optimiranje. Ko dobimo rešitev, se moramo zavedati, da je ta rešitev dobljena pri upoštevanih pogojih, zato se utegne zgoditi da zaradi nepopolnosti podatkov ali napačne razčlenitve poslovnega procesa rešitev ni uporabna. Seveda rešitve ne smemo po svoje spremeniti, ampak moramo ugotoviti, katere podatke je potrebno spremeniti oziroma kako jih je potrebno dopolniti, da dobimo uporabno rešitev, te podatke popraviti in nato ponovno izvajati program za optimiranje. Vsako kritiko rezultatov moramo zavrniti, če ni mogoče ugotoviti napak v podatkih. Če pa ugotovimo, da v modelu niso pravilno upoštevani vsi bistveni pogoji poslovanja, moramo model popraviti in ponovno dobiti reaultat s pomoč-

jo programa za optimiranje. Način dela je prikazan z diagramom poteka.

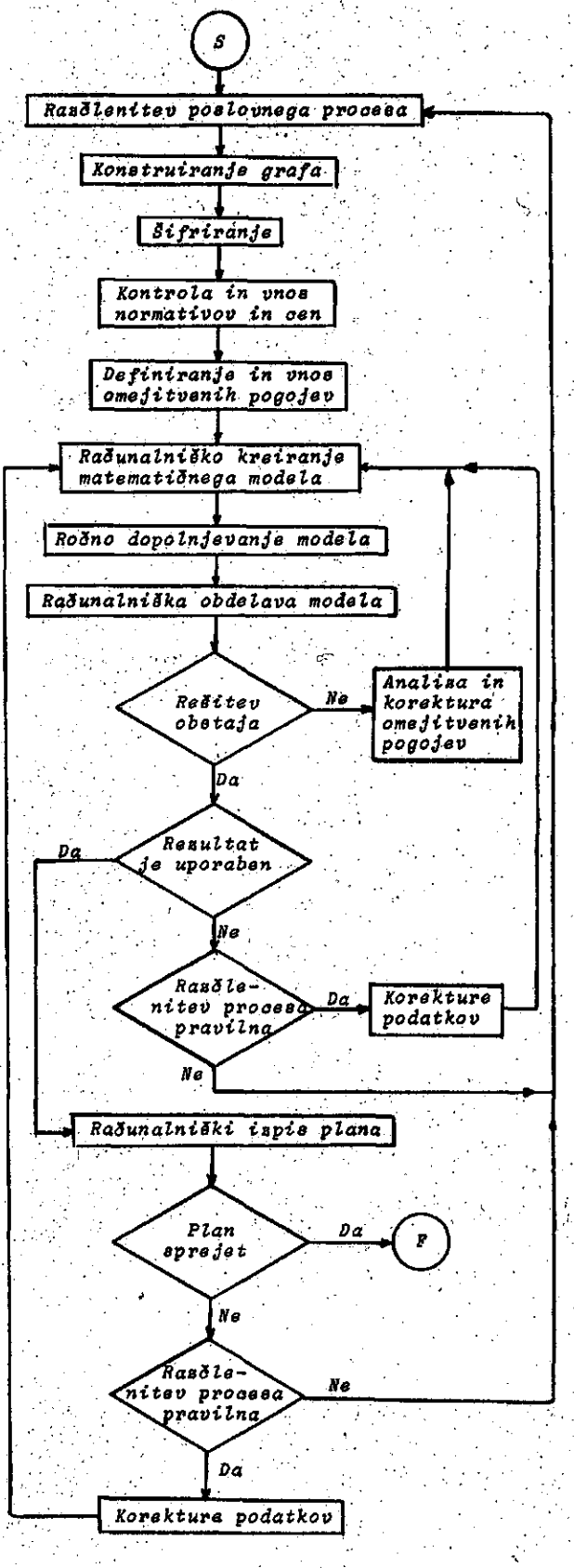
Ko dobimo uporaben resultat, začnemo izvajati program za izpis plana, ki ga nato predložimo pristojnim organom v potrditev. Če plan ni sprejet, moramo v podatkih ali v razčlenitvi poslovnega procesa najti vzroke za njegovo neprimernost, podatke popraviti in ponoviti proceduro, ki je razvidna tudi z diagrama poteka.

Na tak način je bil izdelan poslovni plan v mesni industriji PIK Belje za leto 1984. Poslovni proces je bil razčlenjen na 1162 aktivnosti, upoštevanih je bilo 1431 elementov, večina elementov je imela predpisano spodnjo ali zgornjo mejo, nekateri pa tudi obe. Program za računalniško kreiranje modela je bil sestavljen tako, da je lahko upošteval pet različnih grup vhodnih podatkov. Računalniška obdelava modela je bila narejena s pomočjo universalnega programa LOMP in je na računalniku IBM 4341 trajala 36 minut.

V mesni industriji v Murski Soboti že več let uporabljajo matematični model za izdelavo optimalnega letnega poslovnega plana in kvartalnih planov. Za leto 1983 so izdelali optimalni plan s pomočjo programa LPS, pri čemer so ročno kreirali matematični model. Obdelava modela s 677 aktivnostmi in 766 elementi pa je na računalniku IBM 370 trajala preko štiri ure.

Prednosti računalniškega načina planiranja se ne kažejo samo v tem, da strokovnjakom prihranimo delo ampak predvsem v tem, da dobimo kvalitetnejši plan. Model namreč pri najbolj zapletenem in povezanem poslovnem procesu pravilno upošteva vse oportunitetne stroške, ki jih povzroči proizvodnja nekega izdelka. Izdelka torej ne karakterizira kot ugodnega oziroma neugodnega na osnovi kalkulacije, ampak na osnovi vseh spremenljivih stroškov, pri čemer upošteva tudi oportunitetne stroške, ki jih brez uporabe modela ne moremo v celoti predvideti. Nadaljnja prednost računalniškega načina planiranja se kaže v tem, da s pomočjo modela poslovnega sistema pridemo do novih spoznanj o poslovnem sistemu. Model namreč lahko uporabimo kot scenarij, s pomočjo katerega testiramo poslovne odločitve s področja nabave, proizvodne tehnologije, prodaje in investicij.

Uvajanje računalniškega načina planiranja in sprejemanja poslovnih odločitev predstavlja investicijo, saj je glavnino dela potrebno opraviti samo ob prvi uporabi modela. Ta investicija vpliva na poslovni resultat tem bolj, čim bolje uspemo realizirati optimalni poslovni plan. Za realizacijo optimalnega poslovnega plana je potreben dodaten napor. Lažje je namreč realizirati poslovni plan, pri katerem vseh razpoložljivih resursov ne izkoristimo optimalno, se pravi, da pušćamo reserve. Zato v prvem letu uporabe optimalnega plana njegova realizacija le delno uspe. Kljub temu pa so se vlaganja v kompjuterizacijo planiranja na primer v mesni industriji v Murski Soboti doslej že več kot desetkratno povrnila, četudi ni mogoče vseh pozitivnih vplivov ovrednotiti.



Literatura

1. Meško I., Uputstva za program i metodu LOMP, VEKŠ Maribor, v tisku.

2. Meško I., B. Peveo, Production planning by LOMP, Informatica 1, Ljubljana 1983.

UPORABNI PROGRAMI

Reševanje enačb (določanje korenov) z Newtonovo metodo

```
*****************************************
* Informatica UP 12                     *
* Solution of Equation by Newton's Method *
* januar 1984                           *
* modificirala Jelena Ficzko            *
* sistem CP/M, Delta Partner            *
* prevajalnik Pascal MT/+               *
*****************************************
```

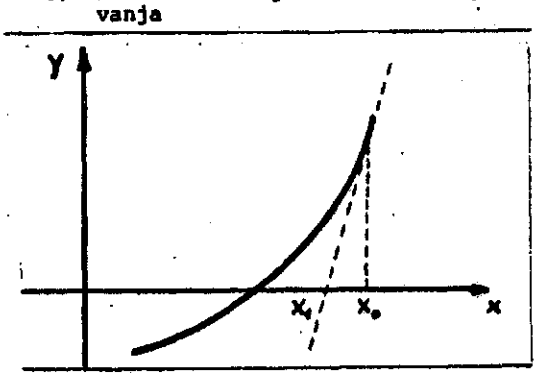
1. Področje uporabe

Reševanje nelinearnih enačb večkrat ni možno brez uporabe numeričnih metod. V takih primerih lahko uporabimo Newtonovo tangentno metodo, ki daje ob dovolj dobrem začetnem približku poljubno natančne rešitve.

```
PROGRAM newton(output);

  {Program za izracun korenov nelinearnih e-
     nacb}
  VAR x,x1:real;
{-------------------------------------------}

  PROCEDURE funcl (x:real;VAR fx,dfx:real);

    BEGIN
      fx:=x*x-2.0;
      dfx:=2.0*x
    END {funcl};
{*******************************************}

  PROCEDURE nwt (VAR x:real);
    CONST
      tol=1.0E-6;
    VAR
      fx,dfx,xi,dx:real;

    BEGIN
      REPEAT
        x1:=x;
        funcl(x,fx,dfx);  {globalna spremenlj.}
        dx:=fx/dfx;
        x:=x1-dx;
        writeln('x=',x1,',fx=',fx,',dfx=',dfx)
      UNTIL abs(dx) <= abs(tol*x)
    END {nwt};
{*******************************************}


PROCEDURE izpis(VAR x:real);
    BEGIN
      nwt(x);
      writeln;
      writeln('Resitev enacbe je ',x)
    END {izpis};
{-------------------------------------------}

      { G l a v n i   p r o g r a m }

BEGIN
  x:=2.0;
  izpis(x)
END{newton}.
```

Lista 1. Program za reševanje enačb z Newtonovo metodo je napisan v jeziku Pascal (natančneje v jeziku Pascal MT/+)

Slika 1. Newtonova tangentna metoda približevanja



Newtonova metoda temelji na ugotovitvi, da je točka tangentnega preseka z abscisno osjo približna ničla funkcije. Če je x0 začetni približek, je smerni koeficient tangente v točki (x0, f(x0)) enak

$$f(x0)/(x0-x1) = f'(x0);$$

tu je x1 točka na abscisi, in sicer boljši približek. Tako velja (glej sliko 1)

$$x1 = x0 - f(x0)/f'(x0).$$

2. Opis programa

Za primer imejmo kvadratno funkcijo

$$f(x) = x^{**}2 - 2$$

Procedura funcl v listi 1 izračunava funkcijsko vrednost v danem korenskem približku in vrednost odvoda v tej točki. Procedura nwt v listi 1 izračunava ničelni približek rekurzivno. Približevalni postopek se konča, ko je razlika med zadnjima približkoma manjša od 1.0E-6. V listi 1 imamo še proceduro za izpis in glavni (preizkusni) program.

3. Izvajanje programa

Program je bil preveden in izvajan na sistemu Delta Partner. Najprej je bil napisan z besednim urejevalnikom in shranjen na disku v obliki zbirke z imenom NEWTON.SRC .Ta zbirka se je prevajala in popravljala, dokler ni bila brez sintaksnih napak. Prevod je bil dosežen s klicem

```
x= 2.00000E+00,fx= 2.00000E+00,dfx= 4.00000E+00
x= 1.50000E+00,fx= 2.50000E-01,dfx= 3.00000E+00
x= 1.41666E+00,fx= 6.94346E-03,dfx= 2.83333E+00
x= 1.41421E+00,fx= 6.19888E-06,dfx= 2.82843E+00
x= 1.41421E+00,fx=-2.38418E-07,dfx= 2.82842E+00

Resitev enacbe je  1.41421E+00
```

Lista 2. Izvajanje programa z liste 1 s približki in končnim rezultatom

MTPLUS newton

Po odpravi sintaksnih napak in prevodu se mora prevod povezati z ustreznimi knjižničnimi procedurami, ki so shranjene v knjižničnih zbirkah. Povezava se opravi z ukazno vrstico

```
LINKMT newton,trancend,fpreals,paslib/s
```

kjer je /s pretikalo za preiskovanje knjižnic časa izvajanja (run-time library search switch). Tu je npr. FPREALS knjižnica aritmetičnih rutin za operacije s pomično vejico. Povezovanje generira novo zbirko NEWTON.COM, ki je neposredno izvršljiva na CP/M sistemu. Izvršitev te zbirke, ki obsega komunikacijo s terminalom, je prikazana v listi 2.

Izboljšava trigonometričnih funkcij za jezik CBasic-2

```
*********************************************
* Informatica UP 13                         *
* Improved Trigonometric Functions for CBasic *
* januar 1984                               *
* priredil A. P. Železnikar                 *
* sistem CP/M, Delta Partner                *
* prevajalnik CBasic-2, verzija 2.07P       *
*********************************************
```

1. Področje uporabe

Basic je jezik za komercialno programiranje, pri tehničnih in znanstvenih uporabah pa si ni pridobil velikega ugleda. Glavni vzrok njegovega zaostajanja je v BCD aritmetiki, ki je počasnejša od binarne aritmetike. Vendar CBasic podpira definicijo uporabniških, večvrstičnih funkcij. Tudi 14-mestna natančnost realnih števil omogoča uporabo jezika CBasic pri tehniških nalogah.

Implementacije vgrajenih (preddefiniranih) trigonometričnih funkcij jezikov tipa CBasic kažejo določene slabosti. Kadar se npr. argument približuje ničli, se rezultata za sinusno in tangensno funkcijo odrežeta pri sedmih mestih. Kosinusna funkcija pokaže podobno slabost blizu argumenta PI/2. Ker ima podobno slabost tudi arctangensna funkcija, se ta slabost prenese tugi na arcsinusno in arccosinusno funkcijo, ki sta izraženi z arctangensom. Nadaljna slabost vgrajenih trigonometričnih funkcij je tudi njihova počasnost pri velikih argumentih (npr. 10000 in 100000).

Naštetim slabostim se lahko ognemo tako, da definiramo svoje elementarne funkcije, ki bodo natačnejše in precej hitrejše.

Lista 1. Definicije trigonometričnih funkcij

```
A>TYPE TRIGFUN.BAS
\-----------------------------------------\
\                                         \
\       TRIGFUN.BAS     %INCLUDE ZBIRKA   \
\     TRIGONOMETRICNE FUNKCIJE ZA JEZIK   \
\                  CBASIC2                \
\                                         \
\-----------------------------------------\

REM--------------  KONSTANTE  -----------------

Z.PI = 3.14159265358979
Z.SQR2 = 1.41421356237310

Z.0 = +0.10385171455197E4  REM KOEFICIENTI ZA
Z.1 = -0.17805646714386E2  REM   FUNKCIJO
Z.2 = +0.26247864594320E-1 REM  FN.TAN.HALF(X)
Z.3 = +0.26445621951222E4
Z.4 = -0.18128328348540E3

Z.5 = +0.21606230789724E3  REM KOEFICIENTI ZA
Z.6 = +0.32266207001325E3  REM   FUNKCIJO
Z.7 = +0.13270239816397E3  REM   FN.ATN(X)
Z.8 = +0.12888363034157E2
Z.9 = +0.21606230789724E3
Z.10 = +0.39468283931228E3
Z.11 = +0.22105088302841E3
Z.12 = +0.38501486508351E2

REM--------------  FUNKCIJE  -----------------

DEF FN.TAN.HALF(X)
    Z=ABS(X)/(Z.PI+Z.PI)
    Z=8.0*(Z-INT(Z))
    Z%=0
    WHILE Z>1.0
        Z=0.5*Z
        Z%=Z%+1
    WEND
    ZZ=Z*Z
    Z=Z*((Z.2*ZZ+Z.1)*ZZ+Z.0)/((ZZ+Z.4)*ZZ+Z.3)
    WHILE Z%>0
        Z=(Z+Z)/(1.0-Z*Z)
        Z%=Z%-1
    WEND
    FN.TAN.HALF=SGN(X)*Z
    RETURN
FEND

DEF FN.SIN(X)
    X=FN.TAN.HALF(X)
    FN.SIN=(X+X)/(1.0+X*X)
    RETURN
FEND

DEF FN.COS(X)
    X=FN.TAN.HALF(X)
    X=X*X
    FN.COS=(1.0-X)/(1.0+X)
    RETURN
FEND

DEF FN.TAN(X)
    X=FN.TAN.HALF(X)
    FN.TAN=(X+X)/(1.0-X*X)
    RETURN
FEND

DEF FN.ATN(X)
    Z=ABS(X)
    Z%=0
    IF Z>Z.SQR2+1.0 THEN Z%=2: Z=-1.0/Z
   IF Z>Z.SQR2-1.0 THEN Z%=1: Z=1.0-2.0/(1.0+Z)
    ZZ=Z*Z
    Z=Z*(((Z.8*ZZ+Z.7)*ZZ+Z.6)*ZZ+Z.5)
    ZZ=(((ZZ+Z.12)*ZZ+Z.11)*ZZ+Z.10)*ZZ+Z.9
    Z=Z/ZZ
    IF Z%=1 THEN Z=Z+0.25*Z.PI
    IF Z%=2 THEN Z=Z+0.5*Z.PI
    FN.ATN=Z*SGN(X)
    RETURN
FEND

DEF FN.ASIN(X)
    Z=ABS(X)
    IF Z>1.0 THEN PRINT "PREVELIK ARGUMENT"
   IF Z=1.0 THEN FN.ASIN=SGN(X)*0.5*Z.PI:RETURN
    FN.ASIN=FN.ATN(X/SQR(1.0-X*X))
    RETURN
FEND

DEF FN.ACOS(X)=0.5*Z.PI-FN.ASIN(X)

REM-------  KONEC ZBIRKE TRIGFUN.BAS ----------
```

2. Opis programa

Program v listi 1 prikazuje definicije sedmih trigonometričnih funkcij, in sicer

FN.TAN.HALF(X)	polovični tangens
FN.SIN(X)	sinus
FN.COS(X)	cosinus
FN.TAN(X)	tangens
FN.ATAN(X)	arcus tangens
FN.ASIN(X)	arcus sinus
FN.ACOS(X)	arcus cosinus

Funkcija FN.TAN.HALF(X) je odvisna od standardnih (vgrajenih) funkcij

ABS(X)	absolutna vrednost
INT(X)	celoštevilski del
SGN(X)	algebraični predznak

in te funkcije ne vplivajo na natančnost. Natančnost rezultata je odvisna od vgrajenih aritmetičnih funkcij (+, -, *, /). Pri izračunu funkcijske vrednosti FN.ASIN(X) je uporabljena še vgrajena funkcija SQR(X), ki pa izpolnjuje pogoje za natančnost pri različnih argumentih.

Lista 2 prikazuje preizkusni program, kjer se preizkuša natančnost definiranih trigonometričnih funkcij s primerjavo z natančnostjo standardnih trigonometričnih funkcij. Zbirka iz liste 1 se pokliče v ta program z ukazom

```
%INCLUDE TRIGFUN.BAS
```

z nadaljnimi PRINT stavki pa se izračunavajo vrednosti, prikazane v listi 3. Program v listi 2 vsebuje WHILE zanko, tako da lahko izračunavanje ponavljamo z različnimi vhodnimi vrednostmi, iz zanke pa izstopimo z vrednostjo 99.

3. Izvajanje programa

Program iz liste 2, ki vsebuje zbirko iz liste 1, izvajamo za več vhodnih argumentov X in dobimo rezultatno listo 3. Iz te liste je razvidno, da je natančnost vgrajenih trigonometričnih funkcij pri malih vrednostih argumentov le sedemmestna, pri novih funkcijah (zadnji stolpec) pa vobče 12-mestna (prva skupina rezultatov liste 3). V drugi skupini rezultatov vidimo, da so nove ločne funkcije natančnejše (saj se vse natanko ujemajo pri ločni funkciji inverzne funkcije argumenta z argumentom). Za druge argumente so rezultati razvidni iz nadaljnih skupin liste 3. V zadnjem stolpcu so rezultati novih trigonometričnih funkcij.

```
CRUN VER 2.07P
      PREIZKUS TRIGONOMETRICNIH FUNKCIJ
-------------------------------------------------------
ARGUMENT X:       8.76543210123456E-6
-------------------------------------------------------
SIN(X):              8.765432E-07          8.76543210123E-07
COS(X):              1                     1
TAN(X):              8.765432E-07          8.76543210124E-07
ARCSIN(SIN(X)):      8.765432E-07          8.76543210124E-07
ARCCOS(COS(X)):      4.472135E-07          8.944272E-07
ARCTAN(TAN(X)):      8.765432E-07          8.76543210124E-07
-------------------------------------------------------
ARGUMENT X:       0.333333333333333
-------------------------------------------------------
SIN(X):              0.327194696796        0.327194696796
COS(X):              0.944956946315        0.944956946315
TAN(X):              0.346253549511        0.346253549511
ARCSIN(SIN(X)):      0.333333327859        0.333333333333
ARCCOS(COS(X)):      0.333333325357        0.333333333333
ARCTAN(TAN(X)):      0.333333327859        0.333333333333
-------------------------------------------------------
ARGUMENT X:       1.5708
-------------------------------------------------------
SIN(X):              0.999999999994        0.999999999993
COS(X):             -3.67328515E-06       -3.67328515751E-06
TAN(X):             -272241.808657        -272241.804395
ARCSIN(SIN(X)):      1.57079272123         1.57079265256
ARCCOS(COS(X)):      1.5708                1.5708
ARCTAN(TAN(X)):     -1.57079265357        -1.57079265359
-------------------------------------------------------
ARGUMENT X:       3.14159
-------------------------------------------------------
SIN(X):              2.6535898E-06         2.65358972076E-06
COS(X):             -0.999999999997       -0.999999999996
TAN(X):             -2.65358980001E-06    -2.65358972077E-06
ARCSIN(SIN(X)):      2.6535898E-06         2.65358972077E-06
ARCCOS(COS(X)):      3.14159008453         3.14158998901
ARCTAN(TAN(X)):     -2.6535898E-06        -2.65358972077E-06
-------------------------------------------------------
ARGUMENT X:       1001.1
-------------------------------------------------------
SIN(X):              0.876265724443        0.876265724426
COS(X):             -0.481828164568       -0.481828164595
TAN(X):             -1.81862703113        -1.81862703099
ARCSIN(SIN(X)):      1.06805649126         1.06805649518
ARCCOS(COS(X)):      2.07353616079         2.07353615841
ARCTAN(TAN(X)):     -1.06805649125        -1.06805649518
-------------------------------------------------------
ARGUMENT X:       99
-------------------------------------------------------
```

Lista 3. Na levi imamo rezultatno listo izvajanja programa z liste 2 (na naslednji strani) z vključitvijo programa z liste 1 (na prejšnji strani). Ta lista kaže rezultate izračuna vrednosti trigonometričnih funkcij z uporabo vgrajenih (preddefiniranih) trigonometričnih funkcij prevajalnika za jezik CBasic2 (srednji stolpec leve liste) in z uporabo v listi 1 definiranih trigonometričnih funkcij.

Natančnost vgrajenih funkcij je očitno mnajša od natančnosti definiranih funkcij pri dokaj malih argumentih (velikostni razred E-6), kar je razvidno iz prve skupine rezultatov v levi listi.

V drugi skupini rezultatov se pokaže večja natančnost definiranih trigonometričnih funkcij pri ločnih funkcijah, ko so 12-mestni rezultati natanko enaki 12-mestnemu argumentu, pri vgrajenih funkcijah pa je to ujemanje le 7-mestno.

V tretji skupini rezultatov je prikazana okoliza točke PI/2 in tu nimamo bistvenih razlik med obema skupinama trigonometričnih funkcij. Primerjavi rezultatov v okolici točke PI in pri vrednosti 1000,1 (velik argument) sta možni v četrti in peti skupini rezultatov leve liste.

Na koncu izstopimo iz programa tako, da vnesemo argument z vrednostjo 99.

```
A>TYPE TEST.BAS
\-----------------------------------------\
\     PREIZKUS NATANCNOSTI TRIGONOMETRICNIH  \
\                FUNKCIJ                    \
\-----------------------------------------\

%INCLUDE TRIGFUN.BAS

DEF FN.ASN(X)=ATN(X/(SQR(1.0-X*X)))
DEF FN.ACS(X)=0.5*2.*PI-ATN(X/SQR(1.0-X*X))

PRINT "       PREIZKUS TRIGONOMETRICNIH FUNKCIJ"
PRINT "----------------------------------------";
PRINT "--------------------"
INPUT "ARGUMENT X:      "; X
PRINT "----------------------------------------";
PRINT "--------------------"
WHILE X<>99
    PRINT "SIN(X): ", SIN(X), FN.SIN(X)
    PRINT "COS(X): ", COS(X), FN.COS(X)
    PRINT "TAN(X): ", TAN(X), FN.TAN(X)
    PRINT "ARCSIN(SIN(X)): ", FN.ASN(SIN(X)),\
          FN.ASIN(FN.SIN(X))
    PRINT "ARCCOS(COS(X)): ", FN.ACS(COS(X)),\
          FN.ACOS(FN.COS(X))
    PRINT "ARCTAN(TAN(X)): ", ATN(TAN(X)),\
          FN.ATN(FN.TAN(X))
    PRINT "----------------------------------";
    PRINT "----------------------"
    INPUT "ARGUMENT X:      "; X
    PRINT "----------------------------------";
    PRINT "----------------------"
WEND
END
```

Lista 2. Ta lista prikazuje preizkusni program za vrsto trigonometričnih funkcij, ki smo jih definirali v listi 1. Rezultati izvajanja tega programa so vidni v listi 3.

Slovstvo

((1)) J.F.Hart et al.: Computer Approximations. Krieger P.C., Huntington, NY. 1978.

((2)) Cody, White: Software Manual for the Elementary Functions. Prentice-Hall. 1980.

****
* Naključne novice *
****

Podjetje DEC je znižalo ceno svojih nižjih sistemov VAX 11/730 za 30%, tako da je to prvi VAX s ceno pod $20000. DEC namerava v kratkem predstaviti svoj namizni VAX z oznako 11/610, ki bo uporabljal nova VAX integrirana vezja in novo vodilo visoke hitrosti.

Digital Research in Microsoft vodita bitko za odločilen vpliv na tržišču mikroračunalniške sistemske programske opreme. IBM je sprejel operacijski sistem MS-DOS (PC-DOS) podjetja Microsoft in podjetje Digital Research je moralo vložiti dodatne napore, da je to prednost konkurenta nadoknadilo. Digital Research je izdal MS-DOS emulator, ki se izvaja pod operacijskim sistemom CP/M-86, tako da se lahko uporablja MS-DOS programska oprema na sistemu CP/M-86. Medtem je tudi IBM začel tržiti Concurrent CP/M skupaj s paketom Wordstar, ki ga lahko tako uporablja več uporabnikov hkrati (istočasno). Podjetje Digital Research je najavilo poln standard prevajalnika ANSI-77-Fortran z grafiko za Personal Basic. Digital Research razvija skupaj z DEC nižjo različico VMS operacijskega sistema.

Intel je do konca prejšnjega leta izvozil v Japonsko 20% svoje proizvodnje. Intel se z japonskimi podjetji dogovarja za recipročno izmenjavo proizvodnje. Ustanovljena je bila tudi posebna japonskoameriška skupina za visoko tehnologijo.

IBM je najavil uporabo (proizvodnjo) 512k-bitnih dinamičnih pomnilnikov s časom dostopa 120 ns. IBM ima tudi podobno 256k-bitno vezja s časom dostopa 80 ns. 512k-bitno vezje ima 4-bitno strukturo (4 x 128k). To vezje ima posebne signalne vhode za tkim. pomnilniško stranenje.

Podjetje Maxtor Corp, 61 E Dagget Dr, San Jose, CA 95134, USA je najavilo proizvodnjo 5 1/4" vinčestrskih enot za 380M zlogov. Družina EXT-4000 ima pomnilne obsege 75 MB (dva diska), 175 MB (štirje diski) in 380 MB (osem diskov).

DEC je začel izdajati nov časopis Digital Review (P.O.Box 2989, Boulder, CO 80322, USA), ki izhaja mesečno in je namenjen izključno DECovim mikroračunalnikom. Ta časopis obravnava naslednja področja: materialno opremo (poslednji tehnološki razvoj miniračunalnikov, mikroračunalnikov, periferne opreme, telekomunikacij), programiranje (programirne metode, programske aplikacije), računalniška grafika (barvna, finančna, laboratorijska; inovacije), obdelava besedil (nove metode, učinkovitost), programska oprema (operacijski sistemi, mreže) in V/I (uporabniške mreže). Cena letnika je $24,97, dveh letnikov $43,97, treh letnikov $57,97, k temu pa je treba za Evropo dodati še letno poštnino $13. Triletni prihranek pri naročnini znaša 36%.

Digital Research napoveduje novo veliko integrirano vezje, v katerem bosta združena procesor Z80 in operacijski sistem CP/M. To vezje se bo proizvajalo v ZDA, Japonski in v Evropi. S tem vezjem bo Digital Research povečal instalirano bazo 1,5 milijona sistemov CP/M na prek 5 milijonov v 24 mesecih. Sistem CP/M bo dobavljiv tudi posebej v samostojnem ROMu, imel pa bo tudi določene inovacije, in sicer menujski sistem s pomožnimi zaslonskimi predstavitvami in vrsticami.

IBM je naročil pri NS večjo količino (1000) vzorcev vezja 16032, kar kaže na pospešen razvoj določenih novih računalniških proizvodov pri IBM. Vendar se potrjuje tudi, da bo IBM uporabil procesor 80386 za svoje namizne 32-bitne mikroračunalnike, dočim naj bi bil procesor 16032 (ali 32032) uporabljen v novih IBMovih miniračunalnikih.

NOVICE IN ZANIMIVOSTI

IBMov osebni računalnik z lastnostmi sistema /370

Na ameriškem tržišču je IBM predstavil razširjeno različico svojega osebnega računalnika (PC). Modela XT/370 in 3270 PC (glej tabelo) naj bi omogočila uporabniku dostop k velikim IBMovim sistemom: PC postaja tako podatkovna postaja tipa 370.

Osebni računalnik XT/370 je mogoče uporabiti na gostiteljskih sistemih, ki so podprti z operacijskim sistemom VM/CMS (Virtual Machine/Conversational Monitor System). Razen tega pa je mogoče s PC v lokalnem režimu izvajati VM/370 aplikacije v virtualnem pomnilniku do 4M zlogov. IBM je objavil tudi programsko opremo za XT/370, ki vključuje uporabo programov iz sistemskih družin 370, 4300 in 30XX na PC, kjer je mogoče uporabljati znane programirne jezike in ukazne procedure.

Z drugim sistemom (3270 PC) naj bi imeluporabnik na voljo na svojem delovnem mestu zmogljivosti večkratnih gostiteljskih procesorjev brez izgube prednosti lokalnega obratovanja PC. Osebni računalnik 3270 PC daje uporabniko funkcijo istočasnega krmiljenja sedmih aplikacij, pri čemer lahko štiri aplikacije v ozadju izvirajo iz večjih računalnikov, kot sta 4300 ali 308X sistem. Računalnik 3270 PC je dobavljiv v treh standardnih izvedbah z glavnim pomnilnikom med 256k in 640k zlogi. Cene teh modelov se gibljejo med $4290 in $7180. Masovna dobava je predvidena v prvem kvartalu leta 1984.

Osebni računalnik XT/370 je XT nadgradnja s tremi dodatnimi tiskanimi ploščami. Prva plošča daje sistemu XT sposobnost prikazovalne postaje tipa 3277 Model 2 (emulacija), ko se PC poveže prek krmilnika tipa 3774. Druga plošča doda realni pomnilni prostor 12k zlogov in možnost virtualnega pomnilnika do 4M zlogov. Tretja plošča vsebuje Intelov matematični procesor 8087 in dva Motorolina mikroprocesorja 68032, ki izvajata podmnožico ukazne množice sistema 370 (emulacija z uporabo mikrokoda v teh procesorjih). Zmogljivost računalnika XT/370 je polovična glede na zmogljivost sistema 4321, ki je najmanjši procesor družine 4300; v znanstvenih aplikacijah bo doseženih 0,4 MIPS, kar je dvojna zmogljivost enega 4331. PC višjega razreda bo dobavljiv v drugem kvartalu 1984, njegova cena z 10M zložnim vinčestrskim diskom in brez operacijskega sistema bo $8995. Za $3790 lahko XT uporabniki razširijo računalnik v XT/370 konfiguracijo.

Sinclair podpisal pogodbo z LR Kitajsko

Podjetje Sinclair Research, lastnik licence za domače računalnike ZX81 in ZX Spectrum, ki prodaja te računalnike v ZDA in Japonski, je podpisalo pogodbo za sestavljanje teh računalnikov v LR Kitajski. Sinclair bo pošiljal Kitajski komponente za sestavljanje obeh domačih računalnikov. Kitajci so zgradili tovarno v mestu Kanton in Sinclair napoveduje dobavo velike količine komponent, če bo začetni dogovor uspešen.

OSEBNI RAČUNALNIKI PODJETJA IBM

Osebni računalnik	PC	PC XT	PC XT/370 Model 568	PC XT/370 Model 588	3270-PC Model 2	3270-PC Model 4	3270-PC Model 6
CPU	8088	8088 8087	2x68032 8087	2x68032 8087	8088	8088	8088
Pomnilnik (kB) (min - max)	64-544	64-640	256-640 PC Mode 256-480 370 Mode	256-640 PC Mode 256-480 370 Mode	256-640	256-640	256-640
Floppy (F) Winchester (W)	1x160kB 2x320kB F	1x320kB 1x10MB W	2x320kB F	1x320kB F 1x10MB W	1x320kB F	1x320kB F	1x320kB F
Masovni pomnilnik	20 MB	20 MB	20 MB	20 MB	20 MB	20 MB	20 MB
Zaslon	PC zasl	PC zasl	PC zasl 3278/ 3279 adp	PC zasl 3278/ 3279 adp	5151/ 5272	5151/ 5272	5151/ 5272
Operacijski sistem	PC-DOS CP/M-86	PC-DOS CP/M-86	VM/CMS PC-DOS 2.0	WM/CMS PC-DOS 2.0	PC-DOS 2.0	PC-DOS 2.0	PC-DOS 2.0
Komunikacija	3101 3270 SDLC/BSC	3101 3270 SDLC/BSC	3101- emulac, SDLC/BSC	3101- emulac, SDLC/BSC	3270- adp, 3178 3278/79	3270- adp, 3178 3278/79	3270- adp, 3178 3278/79
Cena osnovnega sistema	približ. DM 10000	približ DM 15000	$6720	$8995	$4300	-	-

Sinclair je podpisal pogodbi z dvema kitajskima podjetjema (South China Computer Company in China Electronics Import and Export Corporation) po predhodnih razgovorih z direktorjem podjetja Sinclair Research (Nigel Searle). Kitajski inženirji se bodo šolali v ZDA, kjer se Sinclairovi računalniki sestavljajo. Britansko podjetje Prism bo v Pekingu razstavljalo te računalnike in raziskalo možnosti plasmaja (marketinga) v Kitajski. Sinclairova pogodba naj bi bila le začetek pogodb med zapadnimi podjetji in Kitajsko. Kitajci že izdelujejo mikroračunalnike, ki so posnetki ameriških in japonskih, podjetje Commodore pa izdeluje na Kitajskem nekatere svoje komponente.

Cilj Kitajske je v izgradnji lastne računalniške industrije, ki bi v prvi fazi zadovoljevala potrebe domačega tržišča pozneje pa tudi tujih tržišč. Zahteve po izdelkih potrošniške elektronike so v zadnjih dveh letih skokovito narastle. Kitajska občuti tehnološko zaostajanje za razvitim svetom in sprejema licenčno proizvodnjo. Kitajsko tržišče bo najprej zrelo za japonsko tehnologijo (podobnosti v pisavi, 3000 znakov).

Tudi pri nas bi lahko znatno več naredili za izboljšanje računalniške pismenosti. Iskra in Gorenje doslej nista uspela razviti in proizvajati domači računalnik, ki bi bil podoben Sinclairu. Za tak proizvod je potrebnih več inovacij, ki jih bržkone ni moč dobiti iz mikroračunalniškega kluba ali iz rok amateriziranih razvijalcev. Tehnološka inovacija bi bil npr. mikropaket (micro package), v katerem bi bila združena inovacijska in standardna integrirana vezja. K temu bi bilo treba dodati še organizacijsko inovacijo, ko bi npr. prek malega gospodarstva organizirali dovolj množično in inovativno proizvodnjo. Marketinška študija bi morala izhajati iz plačilne sposobnosti občana in inovativno podrediti zmogljivost računalnika, njegovo proizvodnjo in množičnost tej predpostavki. Temu razvoju in proizvodnji bi morala načelovati vrhunska konzultantska skupina, sestavljena iz izkušenih razvijalcev, najboljših ekonomistov in organizatorjev. Na začetku pa bi bil vsekaker potreben tudi zagonski kapital in upravna družbena podpora (posebno zakonsko določilo).

* Tiha invazija *

Do leta 1987 bodo japonski proizvajalci zagospodarili v ZDA na bistvenih področjih informacijske industrije, med drugim tudi na področju zasebnih telefonskih central. Japonski prodor na ameriško tržišče komunikacij, obdelave podatkov in pisarniških naprav bo podoben prodoru japonskih kamer, osebnih vozil in stereo naprav. Že danes zasedajo Fujitsu, NEC in Hitachi 50% ameriškega tržišča visokointegriranih pomnilnih integriranih vezij. Ta položaj ni posledica nizkih japonskih cen marveč visokokvalitetnih izdelkov.

Na področju CPU izdelkov se bo japonski delež v ZDA povečal do leta 1987 na 30%, kar bo znatno vplivalo na razvoj mikroračunalniških izdelkov. Tržišče upogljivih diskov bo s 45% zasedeno z japonskimi izdelki, tržišče matričnih tiskalnikov pa s 35%. Glaven japonski prodor se pričakuje na tržišču pisarniških izdelkov (95%) in na področju zasebnega posredovanja informacij v ZDA (32%).

* Ostrenje mišljenja s programiranjem? *

Baje se s programiranjem ostri mišljenje, zato naj bi se učili programiranja. To je seveda zasebno prepričanje nekega fizika, ki programira, vendar ni programer. Hkrati trdi, da piše programe profesionalno, da je napisal vrsto uspešnih programov, čeprav ni pravi programer. Programiranje ni bilo nikoli njegova osnovna dejavnost in ta fizik ni nikoli obiskoval programirnih tečajev.

S programiranjem se seveda lahko ostri mišljenje samo za programiranje. Pa še to ni vedno res, ker je ostrenje odvisno od občutljivosti, motivacije, razgledanosti pa tudi od programirne inteligence programerjevega okolja. Slabo programirno okolje proizvaja slabe programerje, bolj otopljene kot priostrene.

Vobče velja, da programirnaje ni tista magična dejavnost, ki bi jo obvladovali le programirni praktiki oziroma pravi programerji. Programiranje vobče ni tako tvegano, kot je npr. nevrokirurgija, astonavtika, jedrska fizika. Vendar to ni čisto res, če je tudi programiranje oziroma kvaliteta programa povezana z življenskim tveganjem. Program lahko nosi v sebi semantiko, zamisel, domišljenost programerja, tako da je usoda nekega procesa odvisna tudi od semantične in ne samo spretnostne domišljenosti programerja. Programer mora bržkone poznati tudi problemski prostor svojega programa, zato bo fizik še najbolje reševal programe za obvladovanje fizikalnih procesov. Seveda pa je programiranje vobče skupinski podvig, kjer je skupina sestavljena iz poklicnih programerjev in iz specialistov problemskega prostora. V taki skupini pa bo fizik prej specialist kot pravi programer. Programerske skupine brez specialistov proizvajajo semantično pomanjkljive programe, ki nimajo ustrezne funkcionalne zmogljivosti in prave (tržne) konkurenčnosti.

Računalniška množičnost pa pušča tudi življenski prostor ljubiteljskim programerjem. S programom je mogoče tudi eksperimentirati in tako popravljati in ugotavljati njegovo zgradbo. Zato se programiranja lahko učimo podobno kot kakšnega drugega predmeta (matematike, kemije, fizike), vendar moramo vselej nekoliko poznati tudi problemski prostor programa. Ljubiteljsko programiranje je najbolj razširjeno v nekaterih posebnih programirnih jezikih, kot so Basic, Cobol, Fortran, Pascal. Poklicni programerji uporabijo tudi PL/1, Algol 68, Ado, C, Apl itd. Predvsem pa pišejo programirne sisteme, posegajo v operacijske sisteme računalnikov, gradijo obsežne programske komplekse, kjer morajo poznati problemski prostor organizacije programov, operacijskih sistemov, prevajalnikov, sistemov podatkovnih baz itd. Poklicni programer naj bi obvladoval programirno metodologijo.

V šolah se s programiranjem prav gotovo ostri um učencev, predvsem pa se ta um razširja na pomembno tehnološko področje. Preveliko zapadanje v ljubiteljsko programiranje ima dobre in slabe posledice, tako kot velja to za vsako drugo ljubiteljsko obsedenost. Tudi pri poklicnih programerjih lahko ostrina otopi tedaj, ko se programiranje omeji samo na določene, stereotipske sisteme (operacijske sisteme, prevajalnike, programske generatorje, računalnike). Ti programerji zgubijo stik z novejšo metodologijo, ne morejo se hitro prilagajati novim sistemom in novi tehnologiji. Zato naj bi tudi poklicni programerji ohranjali svojo sposobnost z reševanjem posebnih, nestereotipnih programskih ugank.

Ali je programiranje res samo pisanje, dopisovanje, sestavljanje dopisov? Dober, zapleten program je lahko podoben filozofskemu spisu, ki ga zlahka ne razumemo. Vendar ta program opravlja neko povsem določeno funkcijo, lahko pa ima pri svoji zapletenosti še vrsto stranskih učinkov, seveda neugotovljenih. Kompleksen program ima tako lahko svojo diskurzivnost, podobno kot filozofski spis, vendar je ta diskurzivnost pri programu nezaželena, pri filozofskem spisu pa regularna (dobra filozofija). Program zahteva semantično urejenost, pomensko regularnost, namensko funkcionalnost, dovolj bogato in vsestransko; to je merilo njegove kakovosti.

Programiranje ostri in širi um le tedaj, ko se osvaja novo znanje, ko novo znanje nastaja v osebku, ko se s programiranjem rešuje problem, nastajajo problemske asociacije, ko se pojavlja zadovoljstvo, ko je problemska potreba zadovoljena. Programiranje je tudi komunikacijski problem: komunikativnost programa, njegova izrazna kakovost je bistvena, je posledica programerjeve kulture, njegovih umskih sposobnosti, občutljivosti za lepo in popolno.

****
* Predlog standarda IEEE P695 *
****

Univerzalni format objektnih modulov za različne mikroprocesorje, ki bo omogočil neodvisno povezovanje in premeščanje od lastnosti namenskih arhitektur, se nahaja pred sprejetjem kot IEEE standard. Ta standard določa univerzalni objektni format, ki je prenosljiv (portabilen) in predstavljen kot zaporedje ASCII znakov, dovoljena pa je tudi binarna predstavitev. K temu je določena uniformirana ukazna zgradba, ki podpira povezovalno urejevanje, premeščanje, izračun izrazov in nalaganje.

Zmogljivosti mikroprocesorskih sistemov naraščajo, pri večjih pomnilnikih se povečujejo tudi programi in z njimi potrebe po delitvi programov v manjše, laže spremenljive enote (module), ki se medsebojno povezujejo v končni program. Iz teh zahtev raste potreba za premestljivimi in povezljivimi kodnimi moduli. Ista programska oprema naj bi se uporabljala na različnih operacijskih sistemih in računalniških konfiguracijah različnih proizvajalcev. Možnost premestitve in povezave kodov različnih proizvajalcev je uporabniška potreba in to možnost je mogoče uresničiti s standardnim formatom objektnih modulov.

Druga potreba za standarizacijo formata objektnih modulov izvira iz tega, da določen uporabnik gradi svoje sisteme z različnimi mikroprocesorji. Ta uporabnik bi moral tako podpirati vse svoje procesorje, uporabljati vrsto prevajalnikov itd. Pri standardnem formatu se lahko popravijo samo specifične funkcije, povezovanje in premeščanje objektnih modulov pa se opravi za vse procesorje z enim samim programom. MUFOM (Microprocessor Universal Format for Object Modules) je uporabljiv za vrsto namenskih strojev. Funkcije povezovanja in premeščanja so ločene od specifikacije namenske arhitekture, tako da zadostuje en sam program za izvajanje teh funkcij pri različnih mikroprocesorjih.

P r o c e s i . MUFOM je določen tako, da objektni modul v svojem življenskem ciklu lahko preživi pet procesov in da so ti procesi posebni, ločeni programi ali pa so sestavljeni v en sam program.

Prvi življenski proces objektnega modula je njegovo oblikovanje s prevajalnikom ali zbirnikom. Ta proces je povezan z arhitekturo namenskega stroja in ni določen s standardom.

Drugi življenski proces je povezovanje ločeno prevedenih objektnih modulov v en sam modul in razrešitev zunanjih navedb. Delno povezovanje lahko vzame nekaj modulov in izdela nov modul z nerazrešenimi zunanjimi navedbami, tako da je potrebno še nadaljnje povezovanje. Povezovanje ni potrebno pri prevodih, ki ne navajajo zunanjih procedur.

Tretji proces je premeščanje. Ta proces prireja absolutne pomnilne lokacije modulskim lokacijam in prilagaja naslovna polja, ki so bila generirana z naslovnimi konstantami (ali izrazi) relativno k modulskim vstopnim točkam. Največkrat se premeščanje opravi v istem času kot povezovanje. Objektni modul, ki pride iz premeščevalnika, je navadno absoluten in ima v celoti določena vsa naslovna polja.

Četrti življenski proces izračunava nalagalne izraze. Ti izrazi se lahko generirajo v procesu povezovanja in premeščanja ali pa lahko nastanejo že v začetku pri prevodu izvirnega jezika. Po premeščanju so vsi potrebni operandi za izračun določeni. Rezultat izračuna je pomnilniška preslikava, ki je že določena za naložitev. Izračun izrazov vobče ni odvisen od namenskega stroja.

Peti proces je nalaganje absolutnega modula v pomnilnik namenskega stroja. Ta proces je lahko odvisen od dolžine besede namenskega stroja. Nalaganje je sicer neodvisno od arhitekture procesorja. Večkrat je nalagalna funkcija vsebovana v premeščanju in izračunu izrazov. Z nalagalnim procesom se konča življenski cikel objektnega modula.

MUFOM je enolično določen za vseh pet procesov in nalagalni modul ima format, ki je prava podmnožica formata za absolutni modul in prava podmnožica formata za premestljivi modul, ta pa je prava podmnožica za povezljivi modul. Ena sama sintaksa določa vse tri formate z ustreznimi ukazi za povezovalnik, premeščevalnik in nalagalnik.

P r e n o s l j i v o s t . MUFOM je osnovan na tkim. CUFOMu (CERN Universal Format for Object Modules). MUFOM je predviden za kar največjo prenosljivost objektnih modulov med konformnimi sistemi. Zaradi tega so vsi ukazi določeni kot ASCII besedilo z minimalno znakovno množico. Za notranjo uporabo je predviden ekvivalentni binarni format. Vrsto obstoječih modulskih objektnih formatov je mogoče prevesti v MUFOM. MUFOMski ukaz je vselej enostavna funkcija.

Predloženi standardni format zadeva celotno področje programskega razvoja. Izhodni objektni modul prevajalnika ali zbirnika je standariziran, povezovalni urejevalnik je standariziran, premeščevalnik z različnimi tipi premeščanja je standariziran in standariziran je tudi nalagalnik. Razvijalci novih prevajalnikov in zbirnikov bodo morali upoštevati formatni vzorec neglede na namenski stroj. Morali se bodo priučiti tudi nestandardnih formatov novih strojev. Uporabljali bodo lahko svojo programsko opremo na gostiteljskih računalnikih, ki podpirajo standard. Če ne bo nalagalnika za standardni format, se bo lahko pred nalaganjem uporabil formatni pretvornik, ki bo izdelal absolutni nalagalni modul v formatu namenskega stroja.

MUFOMsko jedro sestavljajo izrazi (+, -, NEG), spremenljivke in ukazi. Sintaksa obsega množico pravil za kontekstno svoboden jezik standarda.

Koncept posameznih procesov je prikazan v tabeli 1.

Tabela 1

vhod/izhod	konceptualni proces
različni moduli	povezovalni urejevalnik (ta proces sestavlja različne premestljive module z razreševanjem zunanjic)
različni odseki	premeščevalnik (ta proces priteja absolutni začetni naslov vsakemu modulu)
en sam "odsek"	izrazni računalnik (ta proces izračunava vse izraze v MUFOMskih ukazih)
absolutni odsek	nalagalnik (ta proces prenese binarne podatke v pomnilnik)
napolnjen pomnilnik	

****
* IBMov SNA postaja standard *
****

IBMova strategija mreženja postaja industrijski standard kljub mednarodnim naporom, da se uveljavi OSI (Open Standards Interconnection). OSI standarde podpira in izdaja ISO (International Standard Organisation), vendar prihajajo ti standardi z zakasnitvijo, ko so se drugi že uveljavili in pot nazaj ni več ekonomska. To velja za IBMov standard SNA (System Network Architecture), s katerim OSI ne bo mogel več uspešno tekmovati. To bo veljalo še zlasti na področju pisarniško avtomatizacije, kjer so se SNA mreže znatno razširile in uveljavile.

Tudi približno 50% uporabnikov velikih sistemov uporablja SNA neglede na proizvajalca velikega sistema. Tehnološko ustrezno mreženje je bilo namreč pogoj za povezavo namiznih računalnikov (mikrosistemov) s kabinentnimi sistemi. Ponudniki mrežnih storitev so pomagali podjetjem pri izgrajevanju njihovih komunikacij na dokaj velikem tržišču z vpeljavo SNA. Mikroračunalniška revolucija je tako uveljavila tudi IBMov SNA, saj je v tej revoluciji IBM daleč najuspešnejše podjetje v proizvodnji mikrosistemov in rasti te proizvodnje. Ob tem se je razvila tudi specifična industrija aparaturne in programske opreme za SNA.

IBMov spopad na področju mrežnih računalniških sistemov se odvija v ZDA s podjetjem AT&T v Evropi pa s PTT podjetji. Pri tem še ni povsem jasno, kje se mreža končuje in kje začenja uporabniška povezovalna naprava. Tu se skriva namreč veliko tržišče naprav za končnega uporabnika. IBMova strategija temelji na nudenju omejenih obdelovalnih zmogljivosti končnemu uporabniku, tako da bo ta kasneje zahteval povezavo na mrežo. Evropska PTT podjetja in AT&T bi želela zaslužiti čimveč denarja in zato uporabnikom ne pomagajo tehnološko, marveč iščejo predvsem svoj vpliv. Medtem je bil SNA razširjen na mrežno upravljanje in na nove mrežne konfiguracije. IBMov PC (Personal Computer) bo omogočal uporabniku vzdrževanje večkratnih istočasnih analogovnih programskih zasedanj. IBM je objavil tudi lokalno SNA mrežo z uporabo odvečnih superminijev 4300, ki bodo skrbeli za selektivno porazdelitev delov centralizirane baze podatkov.

****
* NS se utrjuje na področju 32-bitnih *
* mikroprocesorjev *
****

Podjetje National Semiconductor (NS) je objavilo proizvodnjo svojega integriranega vezja 32032, katerega množična proizvodnja se bo začela v aprilu letos. Podjetje je pokazalo tudi nov sistem SYS-16, ki je večuporabniški razvojni sistem za 16-bitno mikroprocesorsko družino 16000 in ISE (In System Emulator), ki lahko oskrbuje 8 uporabnikov. Mikroprocesor 32032 uporablja taktno frekvenco 6 MHz, ima 32-bitno intervalno arhitekturo in podatkovno vodilo in lahko obdela milijon ukazov v sekundi (1 Mips). Različica za takt 10 MHz bo dobavljiva letos v aprilu. V razdobju naslednjih treh let naj bi se pojavila še CMOS izvedba tega mikroprocesorja.

Novo integrirano vezje ima 80000 tranzistorjev v CPE, ne vsebuje pomnilniške krmilne enote (MMU) in drugih vezij. Do leta 1990 bo več kot 50% mikroprocesorjev imelo 32-bitno arhitekturo. Pri tem je bistveno v kakšni meri bodo mikrokrmilniki lahko podpirali prevajalnike visokih programirnih jezikov. Procesor 32032 je združljiv z 16-bitno procesorsko družino 16000, ki je že podprta z matematičnim procesorjem za pomično vejico 16081 in z MMU 16082 kot z drugimi perifernimi krmilniki. Cena procesorja 32032 bo $220 (količina 100), v letu 1985 pa samo še $20 do $60.

Podjetje NS trdi, da je 32032 prvi komercialni monolitni 32-bitni mikroprocesor na tržišču. Tudi podjetje Zilog je objavilo specifikacije za svoj 32-bitni procesor Z80000 in vzorci so že dobavljivi. Zilog je pripravil tudi matematični procesor 8070 in visokozmogljivi 8-bitni mikroprocesor Z800. Medtem je Intelov procesor iAPX 432 dobavljiv že 2 leti, vendar ni kodno združljiv z Intelovo 16-bitno družino. Dobavljiva je tudi 32-bitna družina podjetja NCR s štirimi vezji že več kot eno leto. Tudi podjetji Hewlett-Packard in Bell Laboratories imata svoje 32-bitne procesorje, ki pa naj bi se uporabljali samo v lastnih izdelkih.

Medtem je Motorola potrdila, da ne namerava izdati 32-bitni procesor 68020 v letu 1984. Motorola ima modificirani procesor 68000, ki ga imenuje 32-bitno vezje in mu poskuša povečevati zmogljivost. Ta procesor naj bi dosegel hitrost 3 do 4 Mips.

Intelov 32-bitna izvedba procesorja 8086 se bo pojavila letos na tržišču z oznako iAPX 386. Tudi NS ima določeno rezidentno sistemsko programsko podporo za svoj 32-bitni procesor 32032, in sicer operacijske sisteme Unix, Concurrent CP/M-16, Idris, BLMX-16 in še druge z uporabo prečnega podpornega paketa NSX-16, ima pa tudi vrsto prevajalnikov za visoke programirne jezike, vključno za Pascal in C. V marcu letos bo dobavljiva tudi razvojna ocenjevalna tiskana plošča za novi procesor, ki bo vsebovala tudi vse bistvene krmilnike družine 16000.

NS namerava proizvajati tudi CMOS izvedbo mikroprocesorja 16032, ki bo dobavljiva v sredini letošnjega leta. Naslednja izvedba 32-bitnega procesorja 32032 z oznako 32132 bo dobavljiva v letu 1986 v CMOS tehnologiji. Ta izvedba bi imela reducirano ukazno množico, vendar bo vsebovala MMU in matematični koprocesor. Drugi

proizvajalec mikroprocesorja 16032 bo tudi podjetje Fairchild.

****
* Spor med podjetjema IBM in Hitachi *
****

Spor med IBM in japonskim podjetjem Hitachi zaradi kraje poslovnih skrivnosti se je končala v korist podjetja IBM. Hitachi mora med drugim dovoliti podjetju IBM, da nadzoruje nove proizvode pred njihovo objavo v naslednjih petih letih. Hitachi mora tudi vrniti vse IBMove podatke, ki jih ima, obveščati mora IBM o posameznikih, ki so prišli do IBMovih zaupnih podatkov in mora pokrivati vse izdatke, ki nastanejo s preiskavami. V zameno bo IBM umaknil svoje obtožbe proti uslužbencem Hitachija. Sporazum med obema podjetjema niso dosegli pravniki, temveč je bil dosežen med podpredsednikoma podjetij. Hitachi je bil v tak sporazum prisiljen zaradi japonskih uporabnikov sistemov, ki so združljivi s sistemi IBM.

****
* Nemogoča dobava procesorja 80186 *
****

Intel je potrdil, da ne more zadostiti skokovitemu naraščanju naročil njihovega 16-bitnega mikroprocesorja 80186. Tako so nastale določene težave pri oskrbovanju proizvajalcev, ki planirajo te procesorje v svojih napravah. Intel potrjuje, da povečuje proizvodnjo teh procesorjev in da bo lahko količinsko zadostil tržne potrebe že v sredini letošnjega leta. Pri tem se je kot velik naročnik teh procesorjev pojavilo tudi podjetje IBM, ki ima procesor 80186 v svoji novi mikroračunalniški generaciji. Pri tem Intel zanika, da bi dajal kakršnokoli prednost določenemu naročniku (IBM) in da so vsi naročniki enakopravni.

****
* 1984: leto 32-bitnih mikroprocesorjev *
****

Tehnološka evolucija se nadaljuje z nezmanjšano močjo v smeri tkim. supermikroprocesorjev. Konec leta 1984 bodo na tržišču 32-bitni mikroprocesorji podjetij

Data General Corp (DG),
Digitam Equipment CORP (DEC),
Hewlett-Packard (H-P),
Inmos,
Intel,
Motorola,
National Semiconductor (NS),
NCR,
Western Electric (WE) in
Zilog

Kakšni 32-bitni mikroprocesorji bodo to in kje se bodo uporabljali?

32-bitni mikroprocesor naj bo opredeljen takole:

- 32-bitna arhitektura,
- popolna 32-bitna implementacija in
- 32-bitno podatkovno vodilo k pomnilniku

A r h i t e k t u r a je povezava med strojem in programerjem. Z arhitekturo je določena

- množica dostopnih registrov,
- ukazna množica,
- pomnilniški model in
- načini strojnega naslavljanja

Stroj z 32-bitno arhitekturo izpolnjuje te pogoje:

- podatkovni in naslovni registri imajo dolžino 32 bitov
- ukazna množica podpira v celoti 32-bitne podatkovne tipe
- indeksni in drugi naslovni modifikatorji se uporabljajo za strojne načine naslavljanja z 32-bitno predstavitvijo

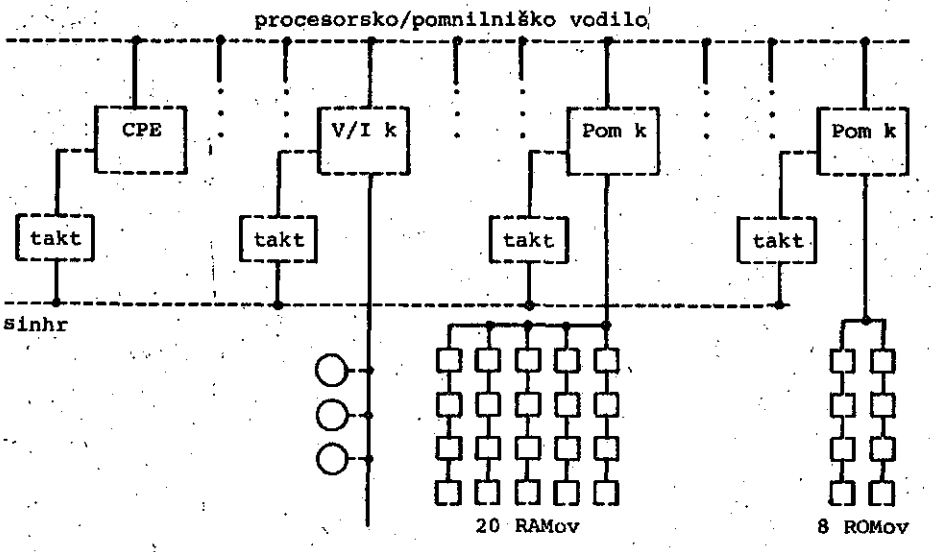
Npr. DECov VAX vobče nima 32-bitne arhitekture, ker so zgornji naslovni biti predvideni za posebne namene in so tako dejanska naslovna polja manjša od 32 bitov. Vrsta 32-bitnih arhitektur ima kompaktne oblike naslavljanja ali naslovnih modifikatorjev, kjer je uporabljenih manj kot 32 bitov, vendar dosegajo polne 32-bitne oblike.

Naslednja bistvena ideja je polna podpora tkim. 32-bitnih podatkovnih tipov. To pomeni, da je mogoče pomikati 32-bitne podatke z enostavnimi ukazi in da je mogoče navadne aritmetične in logične operacije izvajati nad 32-bitnimi operandi in dobivati 32-bitne rezultate. Problemi nastanejo pri množenju in deljenju. Pri množenju dveh 32-bitnih podatkov mora imeti produkt 64 bitov, ker sicer ne moremo zmnožiti dveh 32-bitnih podatkov. Podobno velja za deljenje, kjer pričakujemo 32-bitni kvocient in ostanek, torej mora biti deljenec 64-biten.

I m p l e m e n t a c i j a . Arhitekturna odstopanja so laže razumljiva kot implemetacijska. Prvi 16-bitni mikroprocesorji (8086, Z8000, 68000) so bili implementirani s 16-bitnimi podatkovnimi potmi in računalnimi enotami, čeprav so imeli tudi nekatere 32-bitne arhitekturne lastnosti. Tudi inovativni Intelov iAPX 432 ima 16-bitno računalno enoto in 16-bitne notranje podatkovne poti. Vzemimo mikroprocesor s 16-bitno računalno enoto (aritmetično in logično enoto - ALE), ki sprejme 16-bitna vhoda in izda 32-bitni produkt. Če želimo imeti 32-bitne podatke in dobiti 64-bitni rezultat, bomo v procesorju imeli materializirano subrutino za 16-bitno multiplikacijo. Takšne "subrutine" so problematične pri obravnavi prekinitev in pri virtualnem pomnilniku. Niso namreč bistveno hitrejše od programskih subrutin. Zato te subrutine preložimo na kak drug procesor in dobimo tako procesor s pravo 32-bitno arhitekturo.

P o d a t k o v n o v o d i l o n a p o m n i l n i k . Ta del opredelitve za 32-bitni mikroprocesor je najbolj jasen. Obstajati mora 32 ali več V/I vodov za prenos podatkov med centralno procesno enoto (CPE) in pomnilnikom. Pri tem je važna hitrost prenosa ukazov in podatkov, povečevanje hitrosti z DMA mehanizmi. 32-bitna podatkovna vodila so na področju mikroprocesorjev nov pojav. Npr. iAPX432 in NS 16032 imata sicer 32-bitno arhitekturo toda le 16-bitno podatkovno vodilo.

P o d r o č j a u p o r a b e . Zmogljivost 32-bitnih mikroprocesorjev je velika, tako da se pojavi vprašanje njihove smiselne uporabe. V letu 1984 bodo ti procesorji uporabljeni predvsem v CAD sistemih in v načrtovalnih delovnih postajah. Pojavili se bodo tudi univerzalni računalniki z bistveno nižjo ceno od enakozmogljivih kabinetnih računalnikov. Pojavili se bodo osebni računalniki najvišjega tipa za poslovno in osebno uporabo. Izredno bo olajšano in pohitreno prevajanje visokih programirnih jezikov. Bistvene prednosti 32-bitnih arhitektur so tele:



Slika 1. Bločni diagram mikroprocesorja Focus podjetja Hewlett-Packard

- veliki, enoviti naslovni prostori
- podpora za virtualni pomnilnik
- adresni načini, ki podpirajo visoke programirne jezike
- simetrija ukazne množice glede na operacije, velikost operandov in operandne naslovne načine
- uporaba standardnih operacijskih sistemov

P r i h a j a j o č a t e h n o l o g i j a . V letu 1984 se bo pojavila vrsta novih 32-bitnih mikroprocesorjev. Po abecednem vrstnem redu podjetij bomo imeli tele mikroprocesorje:

-- Microeagle (DG)
-- Micro VAX 1 (DEC)
-- Focus (H-P)
-- Transputer (Inmos)
-- iAPX386 (Intel)
-- NS32032 (NS)
-- MC68020 (Motorola)
-- NCR/32 (NCR)
-- WE32000 (WE)
-- Z80000 (Zilog)

DG je napovedal Microeagla v novembru 1983, DEC pa Micro VAX 1 v oktobru 1983. H-Pjev Focus je srce računalniškega sistema HP-9000 in se posebej ne bo prodajal. Intelov iAPX386 uradno še ni bil napovedan, vendar se je začelo njegovo oglašanje, tako da ga je mogoče pričakovati konec leta 1984. Motorola je najavila svoj MC68020, čeprav še ni izdala podrobnih specifikacij; vzorci bodo na voljo v sredini leta 1984. Komercialno pa sta že dobavljiva procesorja NCR/32 (množica vezij) in NS32032 (s pripadajočimi vezji). WE uporablja svoj WE32000 (prejšnja oznaka Bellmac-32) že od leta 1981. Ta procesor se uporablja interno v WE in v nekaterih miniračunalnikih tipa VAX in posebej ni dobavljiv. Tudi Z80000 je bil najavljen, dokumentacija je dobavljiva in vzorci tudi.

Razen naštetih komercialnih mikroprocesorjev se pojavlja tudi mikroprocesorska družina RISC (Reduced Instruction Set Computer). Ti 32-bitni mikroprocesorji so bili razviti na univerzah v Stanfordu in Berkeleyu (Kalifornija). Ti procesorji niso komercialno dobavljivi, vendar so imeli določen vpliv na razvoj 32-bitnih mikroprocesorjev (npr. Inmosov Transputer).

M i c r o e a g l e (D G) . Ta VLSI vezja so različica stroja, ki ga opisuje T. Kidder v svoji knjigi. Množica vezij opravi operacije med registri v 400 ns ciklu in uporabi dva cikla pomnilniško/registrski pomik. Centralni procesor uporablja koprocesor plavajoče vejice, ki opravi 64-bitni seštevanje v štirih ciklih.

M i c r o V A X 1 (D E C) . VAX je znanana 32-bitna arhitektura dveh tkim. supermini implementacij, in sicer VAX-11/780 in VAX-11/750, v šibkejši izvedbi pa VAX-11/730. Micro VAX 1 je implementacija enake arhitekture v obliki več VLSI vezij, ki so razvrščena na tiskani plošči, podobni LSI-11 implementaciji arhitekture PDP-11. Bistvena lastnist družine VAX je njena navzdolnja in navzgornja združljivost. Micro VAX 1 ostaja brez nekaterih "komercialnih" ukazov, vendar je obdržal pasti, ki omogočajo emulacijo s programsko opremo.

F o c u s (H - P) . H-P doslej ni objavil nobenih podrobnosti o arhitekturi procesorja Focus, čeprav je bila objavljena vrsta člankov o njem. Več integriranih vezij sestavlja popolno integriran 32-bitni večprocesorski sistem. Razvoj je trajal 7 let. Šest vezij je: osrednji procesor, V/I procesor, pomnilniški krmilnik, RAM, ROM in taktnik. Vsa ta vezja so izdelana v 1-mikronski tehnologiji in uporabljajo takt 18 MHz. Sistemski blok je prikazan na sliki 1.

Najbolj zapletena je centralna procesna enota, ki vsebuje 450000 tranzistorjev. 230 ukazov je mikrokodiranih v 9216 besedah dolžine 38 bitov krmilnega ROMa; ti ukazi se zajemajo in izvršujejo s hitrostjo 18 MHz. Posamezen 38-bitni mikroukaz se izvrši vsakih 55 ns. Ukazna množica uporablja skladni mehanizem, kar spominja na Burroughsove računalnike. Vrsta ukazov deluje nad vrhnjimi elementi sklada in pušča rezultate v skladu. Za optimizacijo teh ukazov so vrhnji elementi sklada avtomatično shranjeni tudi v registrih; ta prenos je avtomatičen (brez dodatnih ukazov).

Centralna procesna enota vsebuje izredno zmogljivo aritmetično/logično enoto (ALE), ki lahko izvaja 32-bitne celoštevilske adicije med registri ali desne in leve pomike do 31 bitov v enem mikrociklu (55 ns). Deljenje 64-bitnih operandov v IEEE formatu traja le 16 mikrosekund (to je tudi najdaljša operacija ALE).

Pomnilniški model izvršilnega procesa Focusove CPE je sestavljen iz kodnih, skladnih, globalnih podatkov in zunanjih podatkovnih segmentov. Kod se naloži na zahtevo, in sicer celoten segment hkrati. H-P ocenjuje izvajalno hitrost Focusove CPE na približno 1 MIPS (milijon ukazov na sekundo) pri tipični ukazni mešanici. Tipičen ukaz porabi približno 18 mikrociklov za svojo izvršitev. Čeprav je izvajanje in operandno zaseganje v pomnilniku zaporednoparalelno (pipelined) pa CPE nima hitrega vmesnega pomnilnika (cache) ali TLB (translation lookaside buffer). Focusova CPE izrabi le 30% pasovne širine pomnilniškega vodila (18 Mzlogov/s) in v tem je skrita velika rezerva za multiprocesorske konfiguracije. To pa je tudi osnovna težnja H-P, saj ukazna množica centralnega procesorja podpira paralelno procesiranje.

Focusova ukazna množica podpira visoke programirne jezike in vsebuje tkim. poskusni/obnovitveni pripomoček, ki uporablja skladne označevalnike. Focus naj bi podpiral programiranje v H-Pjevem modularnem Pascalu, imenovanem Modcal. Celo najnižja ravnina jedra operacijskega sistema HPUX (H-Pjeva različica sistema Unix) je programirana v Modcalu.

CPE lahko izda do tri naslove na svoje multipleksirano naslovno/podatkovno vodilo pred sprejetjem ustreznega podatka iz pomnilnika. Podatkovne besede se pojavljajo na naslovno/podatkovnih vodih v natančnih, predvidenih časih, kar zahteva popolno sinhronizacijo dejavnosti centralnega procesorja, pomnilniškega krmilnika in pomnilniških vezij. CPE je sestavljena iz 25 enot in kar 83 vodov je uporabljenih za njihovo medsebojno komuniciranje. Preostali vodi so naslovno/podatkovni (32), napajalni in taktni (26).

T r a n s p u t e r (I n m o s) . To je RISCovski računalnik, ki se bo pojavil proti koncu leta 1984. Ime tega vezja označuje Inmosovo težnjo, da bi bilo to vezje tako povsodno kot tranzistor. Transputer bo izveden v enem samem vezju z 250000 tranzistorji v 2 mikronski CMOS tehnologiji. Vseboval bo procesor, pomnilnik in komunikacijska vezja. Transputer je predviden za uporabo v multiprocesorskih sistemih in v podatkovnopretokovnih strojih, programiranih v Inmosovem jeziku Occam.

i A P X 3 8 6 (I n t e l) . Veliko podatkov o tem novem procesorju temelji na govoricah, vendar obstajajo zanesljivo veljavna dejstva o tem procesorju:

-- je nadaljevanje arhitektur 8086 in 80286
-- ima 32-bitne odmike v podatkovnih segmentih
-- je implementiran v CMOS tehnologiji in uporablja takt 16 MHz
-- predstavlja pohitritev ukazov procesorja iAPX286
-- v procesorju je vgrajen hitri ukazni vmesnik (cache)
-- vsebuje 270000 tranzistorjev
-- ima vgrajeno tkim. stranenje in segmentirani virtualni pomnilnik tipa 286
-- ima nove ukaze za vektorska in bitna polja
-- njegov numerični soprocesor je veliko hitrejši od procesorja 80287
-- njegovo izpopolnjeno vodilo podpira popravljanje napak (fault tolerance)

M C 6 8 0 2 0 (M o t o r o l a) . To je 32-bitni član družine MC68000. Le malo je znanega o lastnostih tega procesorja, ki se bo začel vzorčevati v letu 1984. Motorola je izdala več neuradnih podatkov, vrsta podrobnosti pa je bilo prikazanih tudi sedanjim uporabnikom družine 68000.

MC68020 je 32-bitna izvedba široko uporabljanega procesorja 68000. Ta izvedba je delno združljiva navzdol, tako da bo veliko ukazov procesorja 68000 izvedljivih tudi na 68020 (vendar ne vsi). Največ ukaznih sprememb je na področju naslavljanja, obstaja tudi vrsta novih ukazov za BCD nize, manipulacijo bitnih polj in za 32-bitno množenje in deljenje. Dodanih je bilo nekaj novih načinov naslavljanja, podpirajo pa se tudi 32-bitni pomiki.

MC68020 ima tudi nekaj novih lastnosti. Vsebuje ukazni vmesnik (cache), dinamično obseganje vodila, soprocesorski vmesnik, ki podpira aritmetični soprocesor MC68881, mehanizem virtualnega pomnilnika, podporo vodilu z napakami in pohitritve pomikanja, množenja, deljenja, bitnih manipulacij itd.

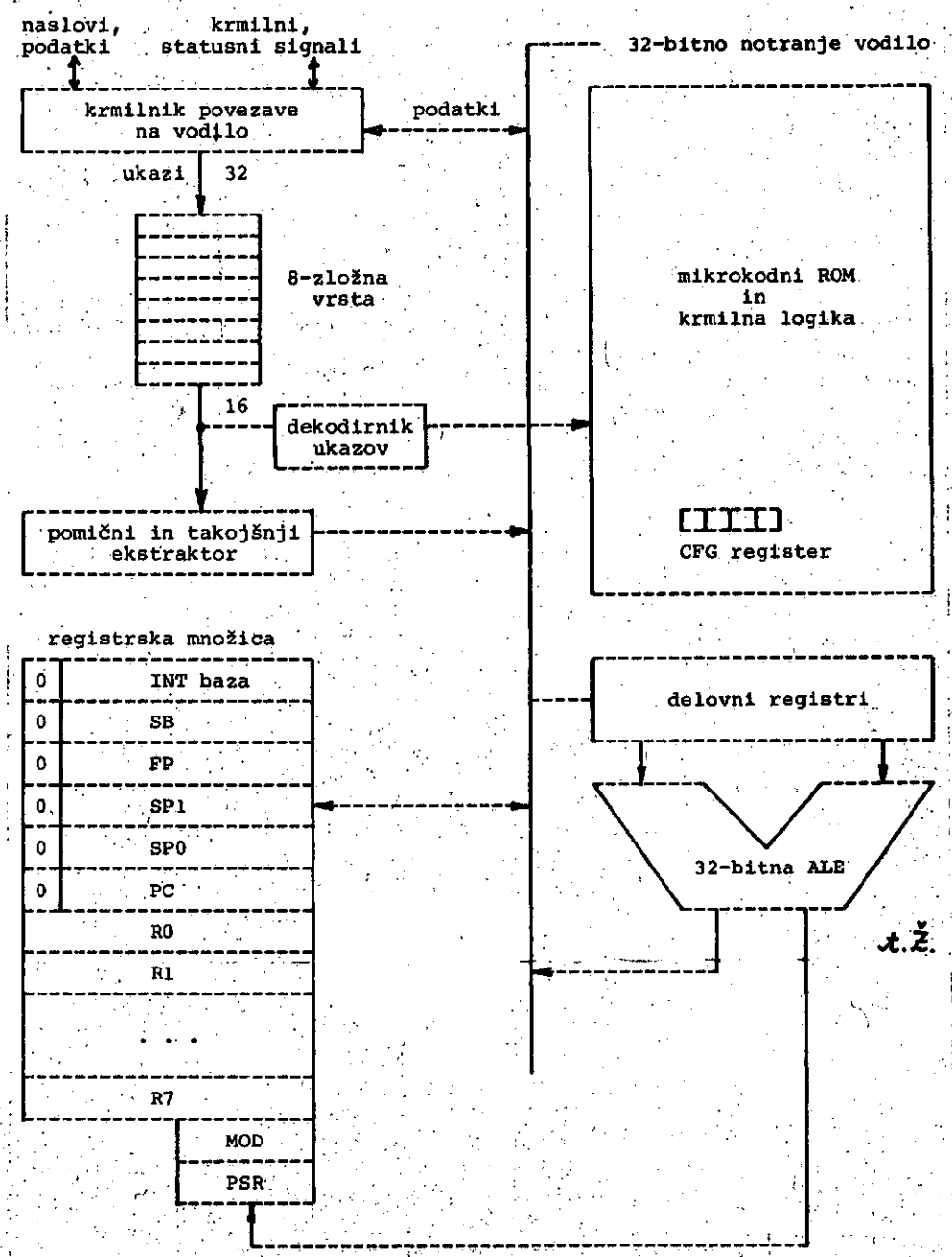
68020 bo proizvajan z novim procesom, ki je 90 % CMOS, v kritičnih vezjih pa bo uporabljena NMOS tehnologija. Procesor vsebuje približno 170000 tranzistorjev in se bo dobavljal za hitrosti 16 in 20 MHz. Procesor bo približno dvakrat hitrejši od 68000.

N S 3 2 0 3 2 (N S) . Ta procesor je 32-bitni član družine NS16000, ki jo sestavljajo NS08032, NS16032 in NS32032. Vezji NS08032 in NS16032 sta 8- in 16-bitni izvedbi za vodilo. Podobno kot velja za VAX izvedbe, imajo vse procesne enote družine 16000 enako 32-bitno arhitekturo. Ker so razlike le na vodilu, je tudi notranja implementacija pri teh procesorjih enaka.

Proizvajalec družine NS16000 trdi, da imajo njeni procesorji arhitekturo kabinetnih računalnikov v enem vezju, da nudijo prave možnosti virtualnega pomnilnika, aritmetične podpore, visoko regularno in kompaktno dekodirano ukazno množico, ki podpira visoke programirne jezike. Procesor 32032 bo povezan s posebno enoto (vezjem) za upravljanje pomnilnika (MMU), ki nosi oznako NS16082 in z aritmetično enoto NS16081. Ukazi za aritmetiko s plavajočo vejico in za upravljanje pomnilnika so vgrajeni v centralni procesor in komunicirajo s podpornimi vezji s pomočjo pomožnega procesorskega protokola. Ta zamisel omogoča transparentno integracijo treh vezij v en sistem.

Slika 2 je blokovni diagram centralne procesne enote NS32032. Bistvena lastnost tega vezja je tudi 8-bitna vrsta za zaseganje ukazov. Ta mali FIFO vmesnik omogoča centralni enoti 32-bitni prenos ukaznega toka, čeprav imajo ukazi procesorja različne dolžine. Ta mali 8-bitni vmesnik nadomešča v bistvu hitri ukazni vmesnik, kot ga poznamo v velikih procesorjih (ukazni cache). Zaseganje ukazov je tu asinhrono glede na njihovo izvrševanje in se opravlja z maksimalno hitrostjo frekvenčne širine vodila. Tudi pri zunanjem upravljanju pomnilnika se ukazi hitreje zasegajo kot izvršujejo, čeprav deluje CPE s polno hitrostjo. Pri značilnih uporabah izrabi CPE 32032 manj kot 50% frekvenčne širine vodila, tako da ostane še dovolj časa za uporabo več centralnih procesorjev, za DMA prenos in za hitro grafiko.

Procesor NS32032 je implementiran s 3,5 mikronskim NMOS procesom in vsebuje 70000 tranzistorjev. Njegova taktna frekvenca je 10 MHz. Že v letu 1984 bo obstajala njegova izvedba v CMOS tehnologiji z znatno povečano taktno frekvenco. Procesor uporablja tristopenjsko zaporednoparalelno (pipelined) ukazno izvajanje. Prva stopnja je nalaganje, ki jemlje ukaze iz vrste; druga stopnja je predprocesiranje, kjer se ukazi dekodirajo; tretja stopnja obsega mikrokodno izvajalno enoto. Mikrokod se izvršuje s hitrostjo enega mikroukaza na taktni cikel (100 ns).

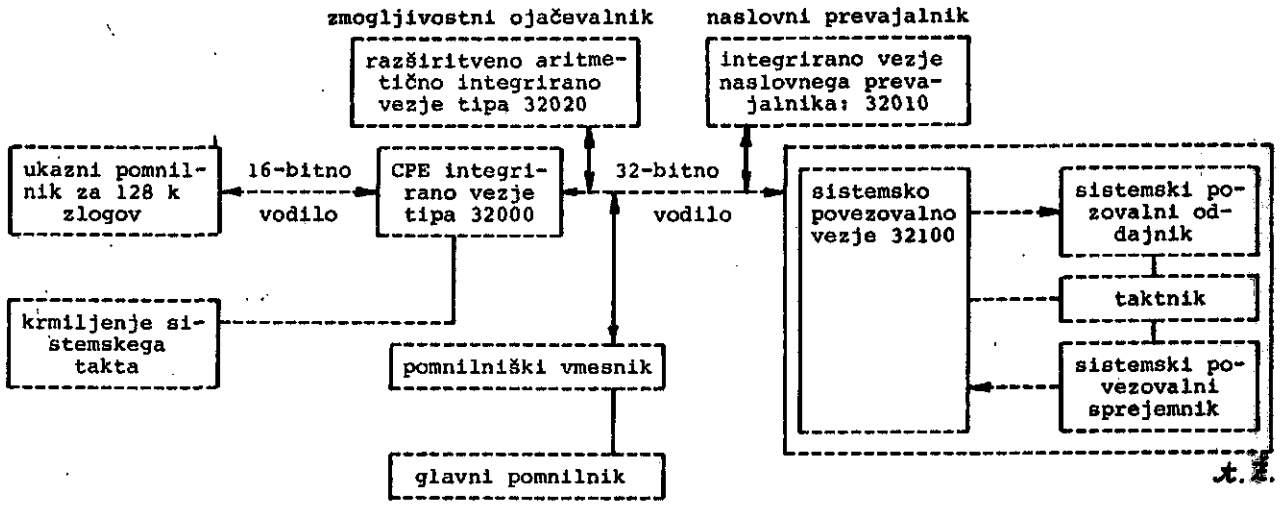


Slika 2. Bločna shema centralne procesne enote NS 32032 podjetja National Semiconductors

N C R / 3 2 (N C R) . To mikroprocesorsko vezje je precej drugačno od doslej opisanih procesorjev. To vezje je pripravljeno za zunanje mikroprogramiranje, tako da lahko emulira druge računalnike, in sicer predvsem srednjevelike IBMove kabinetne računalnike, kot je npr. sistem 370. Množico vezij sestavljajo:

-- NCR 32-000 CPC je centralna procesna enota. Vsebuje 40000 tranzistorjev in je izdelana s 3-mikronskim silicidnim NMOS procesom. Deluje s 13,3 MHz taktom, ko imajo notranji strojni cikli po dva taktna cikla (150 ns). 16-bitni mikroukazi, ki se berejo iz 128k-zložnega zunanjega pomnilnika, izbirajo 95-bitne besede iz notranjega ROMa za krmiljenje 179 operacij, ki so zvečine registrsko aritmetične in logične nad 4-, 8-, 16-, 32-bitnimi in poljskimi podatkovnimi tipi. Mikroukazi se izvršujejo v tristopenjskem zaporednoparalelnem (pipelined) procesu (zaseganje, interpretiranje, izvrševanje). Osem 16-bitnih skočnih registrov podpira bogato množico pogojnih operacij na mikrokodni ravnini in posebno množico mikroukazov za emulacijo IBMovega sistema 370.

-- NCR 32-010 ATC je enota za upravljanje pomnilnika. Razen naslovnega prevajanja in zaščite dostopa opravlja to vezje še krmiljenje pomnilniškega osveževanja, preverjanje napak in njihovo popravljanje (ECC), ima register za čas dneva, intervalno prekinjanje časovnega izteka in prekinjanje pri vpisovanju na določen virtualen naslov. Šestnajst prevajalnih registrov podpira preslikavo 32- ali 24-bitnih virtualnih naslovov v 24-bitni fizični naslovni prostor z uporabo stranenja s stranmi obsega 1k, 2k ali 4k zlogov.



Slika 3. Konfiguracija sistema NCR/32

-- NCR 32-020 EAC je ojačevalno vezje za aritmetične operacije. To vezje podpira IBMovsko združljivo enojno in dvojno natančnostno binarno aritmetiko in aritmetiko s plavajočo vejico, zgoščeni in nezgoščeni decimalni pomnilnik in formatno pretvorbo. Enojno natančnostno seštevanje v plavajoči vejici traja približno 1,6 mikrosekund.

-- NCR 32-500 SIC povezuje procesorsko pomnilniško vodilo za 24 Mzlogov/s s počasnejšimi zunanjiki in drugimi sistemi.

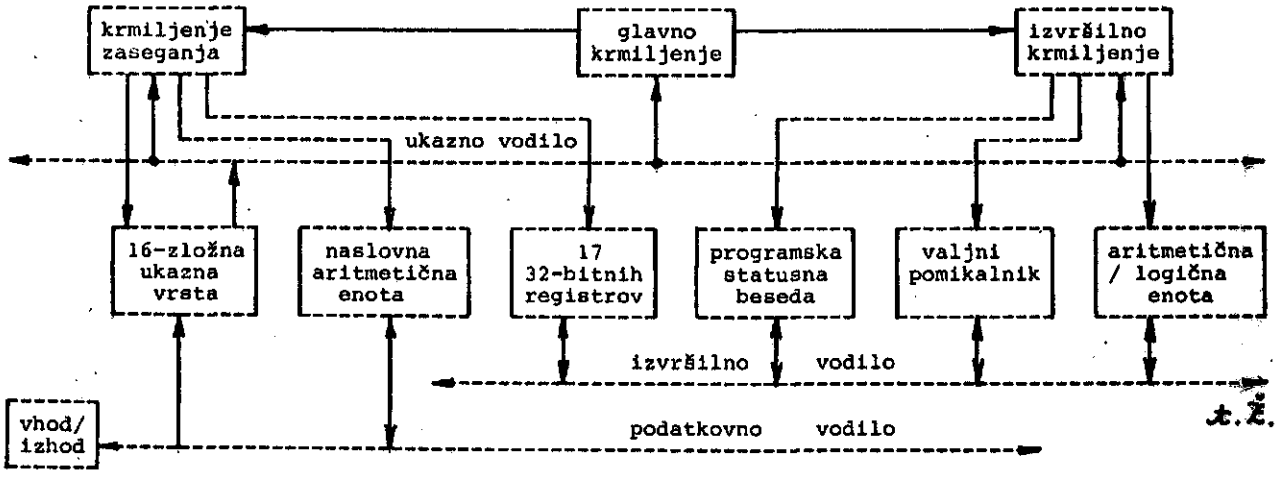
Konfiguracija sistema NCR/32 je prikazana na sliki 3. Doslej niso bili objavljeni primerjalni rezultati (benchmarks), vendar NCR trdi, da je zmogljivost tega procesorja približno štirikrat večja od 10 MHz procesorja MC68000.

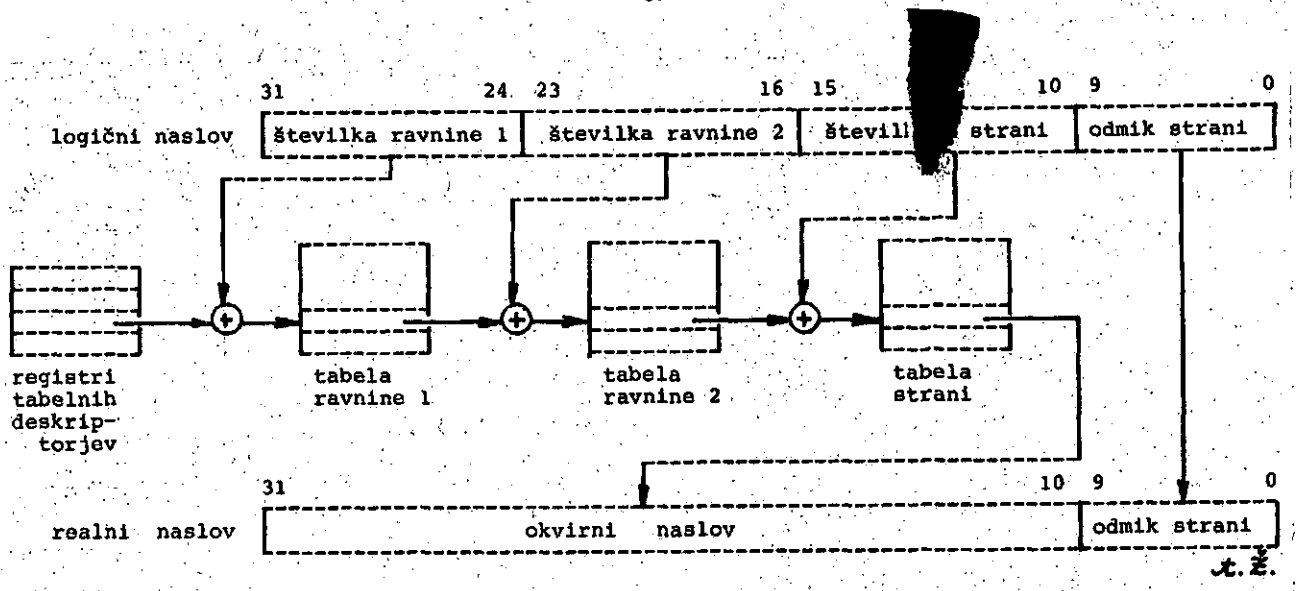
WE 32000 (Western Electric). Ta mikroprocesor je bil načrtovan za podporo operacijskega sistema Unix in jezika C. Osnovno množico procesorja sestavljata CPE in MMU, ki sta izdelana v 2,5 mikronski domino CMOS tehnologiji in uporabljata takt 8 MHz. Obstajajo izvedbe z veliko večjo hitrostjo, ki pa jih proizvajalec ni uradno objavil. Centralna procesna enota vsebuje 146000 tranzistorjev, MMU pa 92000. Slika 4 prikazuje blokovni diagram centralne procesne enote.

Ukazna množica procesorja WE32000 ima visoko stopnjo operacijske ortogonalnosti, adresnih načinov in izbire operandnih obsegov. Vse binarne aritmetične operacije so izvedljive dvo- in trinaslovno in vse unarne operacije imajo dvonaslovno obliko. Operatorski kod določa operandni obseg, tako da so vsi operandi razširljivi na 32 bitov, preden se operacija začne. Pri operacijah z operandi različnih obsegov se upošteva razširjeni tip. WE32000 razpolaga z bitnopoljskimi in niznimi ukazi. Vsi nizi se končujejo z ničtim zlogom, kot je standardno v jeziku C. V ukazni množici so tudi ukazi za aritmetiko s pomično vejico, vendar obstaja za te operacije visoko zmogljivo zunanjo vezje. Zaenkrat WE ni objavil podatkov o takem vezju.

Procedurna povezava procesorja WE32000 je podobna povezavi pri VAXu. Vseh šestnajst 32-bitnih registrov je navedljivih z strojnimi naslovnimi načini. Sedem registrov ima posebno funkcijo: programski števnik (PC), prekinitveni skladni kazalec, kazalec procesnega krmilnega bloka, procesorska statusna beseda, skladni kazalec (SP), okvirni kazalec (FP) in argumentni kazalec (AP). Posebno vlogo imajo pri procedurnem povezovanju PC, SP, FP in AP. SAVE ukaz, ki

Slika 4. Blokovni diagram procesorja WE32000 (Western Electric)





Slika 5. Naslovna prevajalna shema procesorja Z80000 (Zilog)

se izvrši po vstopu v proceduro, reši vsebine registrov R3, ... , R8 z enim samim ukazom. Registri R0, R1 in R2 so namenjeni prenosu podatkov med klicateljem in klicanim in se s procedurnim klicanjem in vračanjem ne spreminjajo.

WE32000 podpira večopravilne operacijske sisteme, kot je Unix. Procesor podpira štiri privilegirane ravnine in mehanizem krmiljenega prenosa (podobno sistemskemu klicu pri strojih, ki poznajo le uporabniški in nadzorniški način). Toda proces ima le en izvršitveni sklad, ki opravlja funkcijo prenosa argumentov med ravninami. Ker se predpostavlja, da je operacijsko jedro v naslovnem prostoru vsakega procesa, ni potrebna delitev vmesnikov med uporabnikom in jedrnimi rutinami. Procesni kontekst (centralni procesni registri in naslovne prevajalne tabele) se shranjuje v pomnilniških procesnih krmilnih blokih in posebni centralni procesni ukazi hitro preklapljajo med procesi. Mehanizem se uporablja tudi za prekinitve, ki se obravnavajo kot procesi, klicani nepričakovano.

Z 8 0 0 0 0 (Z i l o g) . Ta procesor je 32-bitni član družine Z8000. Je navzgornja razširitev arhitekture procesorja Z8000, vendar ima dinamično obseganje vodila, šestnajst 32-bitnih splošnih registrov, hitri podatkovni in ukazni pomnilni vmesnik (cache za 256 zlogov) in pomnilniško upravljanje s pomnilniškimi tabelami (avtomatično upravljani TLB s 16 vstopi). Shema naslovne preslikave omogoča linearno naslavljanje ali več navzgor združljivih razširitev segmentnega naslavljanja procesorja Z8000. Virtualni pomnilnik z zahtevo po stranenju je podpiran z 1k-zložnim obsegom strani. Slika 5 prikazuje metodo naslovnega prevajanja.

Procesor je implemetiran v 2 mikronski NMOS tehnologiji in premore hitrosti do 25 MHz. Začetna izvedba procesorja uporablja takt 10 MHz. Notranji strojni cikli porabijo po dva taktna cikla. Z visoko zmogljivo pomnilniško konfiguracijo se značilni programi izvršujejo z ukazom na vsakih 6,8 taktnih ciklov, tako da znaša izvajalna hitrost procesorja 1,47 MIPS. Tako obseg vodila kot časenje vodila (timing) sta krmiljena dinamično, uporabljajo pa se Z-BUS protokoli, značilni za periferno družino Z8000. Skupinski zlogovni prenos podpira predhodno zaseganje ukazov v hitri vmesnik (cache). Tudi Z80000 je predviden za uporabo zunanjega matematičnega procesorja, ki bo dobavljiv v letu 1985.

K a t e r i p r o c e s o r j e n a j b o l j š i ?

Prva ugotovitev, ki jo lahko podpremo, je, da je mogoče nekatere procesorje razvrstiti v dve skupini; prva je podobna VAXovskim, druga pa PDP-11-ovskim procesorjem. Skupino tipa VAX oblikujeta

-- WE 32000 in
-- NS 32032

Skupino tipa PDP-11 sestavljata

-- MC 68020 in
-- Z 80000

Focus (H-P), ki ima od Burroughsa sposojeno arhitekturo, ni podoben nobenemu od ostalih procesorjev. Ni še znano, kam bo mogoče uvrstiti procesor iAPX386, vendar bo verjetno bližji Z80000 in MC68020, kot kateremu drugemu. NCR/32 ima možnost emulacije z zunanjim mikrokodom in spada v razred RISCovskih strojev, ki mu pripada tudi Inmosov Transputer.

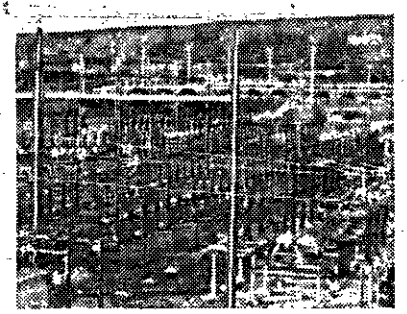
Aritmetična podpora je izredno pomembna lastnost. Tu se uveljavlja IEEE operacijski format z uporabo soprocesorskega vezja. Le H-Pjev Focus ima tak procesor v samem sebi. NCR/32 sledi IBMovem aritmetičnem formatu, ki je različen od IEEE standarda. Tudi VAX uporablja svoje formate, ki pa so podobni IEEE formatom.

Virtualni pomnilnik s stranmi na zahtevo pomeni univerzalno izbiro z uporabo posebnega vezja za pomnilniško upravljanje. Le Zilog in Intel imata to vezje že v osnovnem procesorju. Zilogova metoda se je razvila iz ločenih vezij, Intel pa je dograditl in dopolnil ta pomnilniški mehanizem z izkušnjami pri vezjih 8086 in 80286.

Izvrševalna hitrost še ni zanesljiv podatek, čeprav obstajajo številne primerjave (benchmarks); to velja zlasti za Intelove procesorje v primerjavi z VAXom in z drugimi mikroprocesorji. Med obravnavanimi 32-bitnimi procesorji so razlike v hitrosti do faktorja 4.

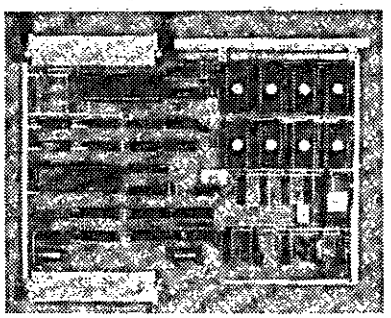
SISTEMI ZA ENERGETIKO

Ljubljana, Tržaška c. 2



DALJINSKO IN LOKALNO PROCESNO VODENJE Z RAČUNALNIKI MINIRAČUNALNIKI IN MIKRORAČUNALNIKI V NAŠIH DOMAČIH SISTEMIH DIPS-11 IN DIPS-85

RAZISKAVE, RAZVOJ, PROIZVODNJA, INSTALACIJA, VZDRŽEVANJE
SPECIALISTIČNO ŠOLANJE KUPČEVIH STROKOVNJAKOV
ELEKTROENERGETIKA, PLINOVODI, NAFTOVODI, VODOVODI, INDUSTRIJA



SODOBNA TEHNOLOGIJA - NAŠ TEMELJ PRI RAZVOJNEM DELU
RAČUNALNIKI - NAŠI SOPOTNIKI NA POTI NAPREDKA
OBIŠČITE NAS IN SE PREPRIČAJTE

Že veliko let se ukvarjamo z raziskavami, razvojem in proizvodnjo sistemov za daljinsko in lokalno procesno vodenje. Temeljno vodilo našega delovanja na tem področju je slediti napredku v svetu in ga presajati na naša domača tla. Vedno smo zavračali nosilno licenčno povezovanje s tujimi firmami povsod tam, kjer smo jasno videli, da vodi v dolgoročno odvisnost in tehnično nazadovanje. Verjeli pa smo v moč lastnega marljivega dela in v ustvarjalnost naših delavcev ter z vstrajnim delom dosegli uspehe, katere nam lahko zavidajo neprimerno večji in bogatejši tekmeci.

Prav zaradi lastne poti in lastnega znanja smo s svojim razvojnim delom ves čas uspeli slediti najnovejšim tehnološkim dosežkom v svetu. V praktično življenje (računalniški nadzor v elektroenergetiki) smo vpeljali najsodobnejše mikroračunalnike.

Tako smo od prvih računalniških korakov pred več kot petnajstimi leti dospeli do sedanjih kompleksnih sistemov za procesno vodenje.