YU ISSN 0350-5596

informatica

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 je 500,00 din, za redne člane 200,00 din, za študente 100,00/50,00 din, posamezne številke 100,00 din
Žiro račun št.: 50101-678-51841
Stališče uredništva se lahko razlikuje od mnenja avtorjev.

Pri financiranju revije sodeluje tudi Raziskovalna skupnost Slovenije.

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

Tisk: Tiskarna KRESIJA, Ljubljana

Grafična oprema: Rasto Kirn

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

YU ISSN 0350-5596

LETNIK 6, 1982 – št. 1

V S E B I N A

informatica

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

Headquarters: Informatica, Parmova 41, 61000 Ljubljana, Phone: (061) 312-988, Telex: 31366 Delta

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

Printed by: Tiskarna KRESIJA, Ljubljana

DESIGN: Rasto Kirn

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 6, 1982 – No. 1

C O N T E N T S

INFORMATIZACIJA IN TRETJI VAL

ANTON P. ŽELEZNIKAR

DO DELTA, SOZD ELEKTROTEHNA, LJUBLJANA

Tretji val, kot ga razpoznava ameriški futurolog A. Toffler, je za področje računalništva in informatike zanimiv predvsem z vidika t.i. informatizacije družbe (javni informacijski sistemi, telekomunikacije, robotizacija proizvodnje in druge inovacije, ki so odvisne od računalniške tehnologije). Podobna izhodišča so značilna tudi za francoskega publicista J.-J. Servan-Schreiberja, katerega knjiga (2) je bila prevedena. Informatiki smo večkrat pomanjkljivo seznanjeni s futurologijo, za katero je v prihajajočem razdobju osrednji pol t.i. informatizacija življenja na planetu.

Informatization and the Third Wave. The third wave as being recognized by the American futurist Alvin Toffler (1) is essential from the aspect of the so called society informatization (public information systems, telecommunications, production robotization, and other innovations concerned with computer technology). Similar aspects are characteristic in the case of the French journalist J.-J. Servan-Schreiber whose book (2) was translated in Yugoslav languages. Workers in the area of informatics are sometimes badly informed on futurist studies in which informatization is treated as a central point of future development on the planet.

Futurolog Alvin Toffler (1) deli dogajanja v preteklosti, sedanjosti in prihodnosti na tehnološke (civilizacijske, družbene) vale, pri tem pa še posebej poudarja t.i. udarne vale in prekinitve valov, tj. tiste točke, ki so izvori novih valov. Poudarek ni toliko na zgodovinskosti in njeni zveznosti kot na zgodovinskih nezveznostih, to je na inovacijah (tehnoloških) in prekinitvah.

Publicist J.-J. Servan-Schreiber (2) deli civilizacijo na kmetijsko, industrijsko in informatizacijsko. Meje med temi civilizacijami se nekoliko razlikujejo od onih, ki jih navaja A. Toffler za valove (1), kar pa ni bistveno.

Prva obratna točka človekovega razvoja je bil vzpon poljedelstva, druga bistvena prekinitev pa industrijska revolucija. Ta dogodka seveda nista bila diskretna, trenutna in vali sprememb so se širili z določeno hitrostjo. Oba vala trajata še danes, v nekaterih zaostalih področjih planeta pa se šele začenjata (npr. poljedelstvo v nekaterih predelih Južne Amerike in Papue, industrializacija v Indiji in Kitajski).

Drugi val, ki je v nekaj stoletjih bistveno spremenil življenje v Evropi, Severni Ameriki in drugje na planetu, se še vedno širi in spreminja tipično poljedelske dežele v industrijske. Moč tega vala še ni potrošena, ko se pojavlja novi, t.i. tretji val: spreminja (transformira) vse, česar se dotakne.

Prvi val se začenja nekako 8000 let pred našim štetjem in gospodari na planetu do razdobja 1650 - 1750. Po tem razdobju se začne pojavljati industrijska civilizacija kot proizvod drugega vala. V Združenih državah Amerike (ZDA) se okoli leta 1955 pojavi desetletje, ko t.i. beli ovratniki in storitveni delavci prvič prerastejo število modrih ovratnikov. To pa je prav desetletje, ko se širi uporaba računalnikov in se uvaja komercialno potovanje z letali, pojavijo se kontracepcijski vložki in še vrsta drugih inovacij. Tretji val se kmalu razširi tudi nad Evropo, Sovjetsko zvezo, Japonsko in še kam. Tako se tehnološko visoko razvite države v poslednjem času že soočajo z nasprotji med tretjim valom in izrabljenimi, zastarelimi, nerazvitimi in okorelimi gospodarstvi in institucijami drugega vala. Vrsta političnih in drugih sporov izvira iz teh navzkrižij.

Vsebolj značilni postajajo izjalovljeni napori nekaterih nerazvitih držav, da se industrializirajo, da vstopijo intenzivneje v obdobje drugega vala, čeprav razvoj civilizacije na planetu kaže, da je ta val preživet, presežen, že poln nasprotij in bi ga bilo najsmotrneje enostavno preskočiti (2). Tu so značilni napori velikih nerazvitih držav (Indija, Kitajska), podobno usodo pa doživljamo tudi sami v naporih za dohitevanje industrijskega razvoja.

Pogled v prihodnost (futurološke napovedi) je seveda bisven, saj vpliva na politiko, industrijo in psihologijo človeka. Že ob napovedih se pojavljajo nasprotja, kako oblikovati prihodnost: tu se oblikujejo družbeni pojmi pristašev in nasprotnikov, naprednjakov in nazadnjakov, prijateljev in sovražnikov, strokovnjakov in vsevednežev, sposobnih in neusposobljenih, razvitih in zaostalih. Pogled v prihodnost daje več možnosti, omogoča izbiro, ki je za posameznika bistvena, saj mu kaže pot, kako se bo iztrgal, odrešil iz sedanjosti in postal to, kar želi biti v prihodnosti.

Obdobja, v katerih se odvija spopad med dvema valoma, starim in novim, so spreminjajoča, nestalna, nobeden od valov ni izrazito prevladujoč in slika prihodnosti je zategadelj zlomljena, razcepljena, prekrivajoča in dostikrat nejasna. Tretji val, ki ga lahko imenujemo tudi informatizacijski, informacijski, računalniški, komunikacijski, robotični ali splošno informacijsko inovativni in informacijsko transformativni, je šele na pohodu. Pojem računalnika je njegov bistveni atribut, saj kot orodje, inovacija in nakopičena inteligenca posega praktično v vsak inovativni in transformativni dosežek novega vala. Računalništvo je tedaj važna, najpomembnejša, bistvena podlaga in dejavnost na pohodu v jutrišnjo iz današnje, neobetavne civilizacije.

Slovstvo

(1) A.Toffler: The Third Wave. Bantam Book (Edition April 1981), New York.

(2) J.-J. Servan-Schreiber: Svetovni izziv. Globus, Zagreb (1981).

MIKRORAČUNALNIŠKI SISTEM DELTA 323/M

BOŽIDAR BLATNIK, ALEKSANDER HADŽI, MARKO KOVAČEVIĆ, ANDREJ LESKOVAR, DRAGO NOVAK, DUŠAN ŠALEHAR, ANTON P. ŽELEZNIKAR

UDK: 681.3.06. Delta 323/M: 181.4

DO DELTA, SOZD ELEKTROTEHNA

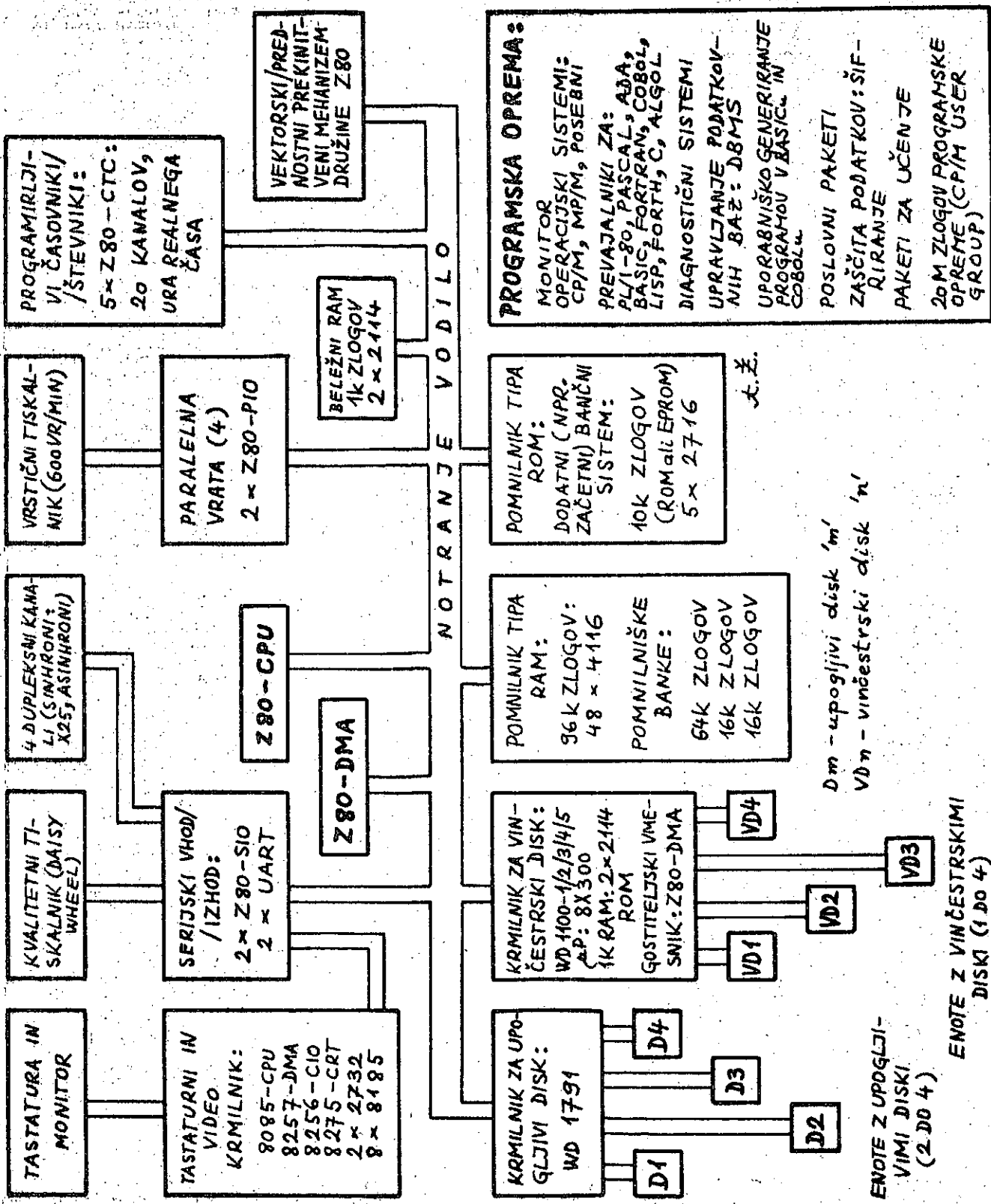
Članek opisuje domači mikroračunalniški sistem Delta 323/M, ki je bil v letu 1981 razvit doma ter se proizvaja v okviru rednega proizvodnega programa DO Delta. Mikroračunalnik 323/M uporablja 8-bitni procesor Z80 ter družino pomožnih procesorjev (Z80-PIO, Z80-CTC, Z80-DMA, Z80-SIO, WD1791, WD1100), ki omogočajo sodobno konfiguriranje mikroračunalniškega sistema. Mikrosistem 323/M ima svoj videotastaturni krmilnik četrte generacije, za periferne naprave pa lahko uporablja več terminalov, do štiri upogljive in štiri vinčestrske diske, lepopisni ter vrstični tiskalnik ter po potrebi še druge periferne enote. Hitri pomnilnik ima obseg 96K zlogov z možnostjo bančnega preklapljanja pomnilniških segmentov ter operacijska sistema CP/M* in MP/M*; slednji omogoča uporabo sistema za več uporabnikov. Takšen izbor operacijskih sistemov zagotavlja uporabo najbolj bogate zaloge programske opreme na planetu - od prevajalnikov in sistemskih programov do aplikativnih paketov za najrazličnejše poslovne, pisarniške, laboratorijske, konstrukcijske in druge uporabniške naloge. Sistem je realiziran na petih standardnih dvostransko tiskanih ploščah evropskega formata.

THE MICROCOMPUTER SYSTEM DELTA 323/M. This article describes the Yugoslav microcomputer system - Delta 323/M - developed last year and now produced within the regular Delta program. This microcomputer is based on 8-bit Z80 processor and a family of auxiliary processors (Z80-PIO, Z80-CTC, Z80-DMA, Z80-SIO, WD1791, WD1100) that enable the modern microcomputer system design. This system uses its own video-keyboard fourth-generation processor. Peripheral equipments, like a number of terminals, up to four floppy disk drives, and four winchester disk units, a quality and line printer, and other peripheral units can be included into the configuration. The fast memory of 96K bytes is devided into three memory banks within the CP/M* and MP/M* operating systems. The latter gives the system a multiuser performance. The choice of these operating systems ensures the use of the greatest ammount of world-wide software - from compilers and system programs to application packages for several business, office, laboratory, computer aided design and other user tasks. The system has been realized on five standarized, double-printed boards (European format).

1. Uvod

Mikroračunalniški sistemi (MKS) se uveljavljajo na različnih inovativnih področjih, od upravljanja do vodenja proizvodnih procesov. Ti sistemi predstavljajo ključ k t.i. informatizaciji, ki ne pomeni le uvajanja široko razvejanih mrež javnih informacijskih sistemov, marveč tudi robotizacijo proizvodnje, avtomatizacijo telekomunikacij, skratka informacijsko povezanost vseh bistvenih sektorjev človekovega dela, izobraževanja, obveščanja.

Napori DO Delta so bili v zadnjem letu usmerjeni v razvoj in proizvodnjo MKS, ki bi lahko pokrival potrebe dovolj širokega spektra informatizacije, z nezmanjšanimi možnostmi uporabe kar največje količine razpoložljive programske opreme na planetu. Takšen pristop naj bi omogočil dovolj strokovno osveščenemu uporabniku svobodno izbiro ustrezne metodologije, programov in tehnologije iz bogate zakladnice nakopičenega znanja v razvitih deželah. K temu naj bi delavci DO Delta dodali še specifično programsko opremo za zvišanje motivacije in storilnosti skozi uporabo ustreznih paketov za naše upravljalske



Slika 2.1 Bločna shema računalnika Delta 323/II. Shema daje vpogled v zgradbo ploščˇ, iz nje je razvidno, katera integrirana vezja so bila uporabljena. Prikazuje tudi možnosti priključevanja periferije in podaja kratek in jedrnat pregled razpoložljive programske opreme.

in proizvodne pristope. V prvi fazi razvoja in priprave proizvodnje MKS Delte so bili tako upoštevani bistveni razvojni vidiki mikroračunalniške tehnologije in njenega tržišča v razvitih državah z možnostmi transfera - tehnološkega, uporabniškega in psihološkega - v naš delovni in upravljalski prostor.

V prvi razvojno-proizvodni fazi je Delta oblikovala dva MKS, ki imata komercialno ime Delta 323/M1 in Delta 323/M3. Ta članek opisuje shematično in večkrat tudi podrobno zgradbo teh dveh sistemov. Sistem M1 je praviloma enouporabniški (izjemoma večuporabniški), dočim je sistem M3 večuporabniški in ima večje pomnilniške zmogljivosti (uporaba vinčestrskih diskov). Oba modela (M1 in M3) sta začetnika mikroračunalniške družine, ki se iz uporabe 8-bitnih mikroprocesorjev nadaljuje v uporabo 16- in 32-bitnih mikroprocesorjev.

Modela Delta 323/M1 in M3 sta bila razvita z lastnim znanjem in iskanjem optimalnega sistema, torej z lastno ustvarjalnostjo peščice Deltinih delavcev, brez sodelovanja visoke domače "neznanosti", ki se je izgubila v preddverje samozadovoljitve. Produkta M1 in M3 sta za razliko nastala tudi v čisti atmosferi lastnega znanja in izkušenj in razvoj obeh modelov ni zahteval niti visokih deviznih investicij (kakor drugje) niti zelo dolge razvojne dobe (M1 npr. šest mesecev).

Modela M1 in M2 sta bila skrbno koncipirana na nizko proizvodno ceno, široko uporabnost in dobro tržno referenčnost. Prav zaradi tega je bil izbran operacijski sistem CP/M (in kasneje še MP/M), ki je za 8-bitne MKS najbolj razširjen OS na planetu, ima pa tudi daleč največjo zalogo dosegljive aplikativne programske opreme.

Zanimanje potencialnih uporabnikov in tudi potencialnih proizvajalcev in soproizvajalcev za sistema Delta 323/M1 in M3 je izredno. Oba produkta sodita v kategorijo malih sistemov, osebnih računalnikov ter sta uporabljiva pri svojih zmogljivostih za različne naloge in okuse tudi kot zamenjava ali dopolnilo velikim, slabo izkoriščanim in obvladanim sistemom zvenečih imen in zmogljivosti.

2. Bločna shema računalnika Delta 323/M in njegova programska oprema

Mikrosistem 323/M je grajen z najsodobnejšo tehnologijo mikrokomponent družine Z80. Direktni pomnilniški dostop se uporablja za prenos podatkov med periferijo in glavnim hitrim pomnilnikom. Shema na sliki 2.1 prikazuje raznovrstnost, popolnost in zmogljivost sistemov Delta 323/M1 in 323/M3. Ta dva sistema sta celoti, ki jima lahko dodajamo periferijo v skladu s potrebami.

Fizične oziroma materialne enote, ki imajo obliko dvojnega evropskega formata (na dvostranskem tiskanem vezju), so tele:

- plošča s centralnim procesorjem (CPE),
- plošča z glavnim pomnilnikom (RAM),
- plošča za V/I (VIK),
- plošča s krmilnikom za vinčestrske diske (WDE),
- plošča s tastaturnim in videokrmilnikom (TVK),
- plošča s preklopnim usmernikom.

Teh šest modulov se nahaja v glavnem ohišju skupaj z diskovnima enotama.

Shema na sliki 2.1 daje vpogled v zgradbo plošč, iz nje je razvidno, katera integrirana vezja so bila uporabljena.

CPE plošča vsebuje razen procesorja Z80-CPU še beležni RAM z 1K zlogov (2x2114), štiri paralelna vrata za pogon in/ali zaznavanje periferije (dve vezji tipa Z80-PIO), pomnilnik tipa ROM/EPROM za 10K zlogov (pet vezij tipa 2716 ali podobno), dvoje dupleksnih serijskih vrat tipa UART (vezje AY-5-1013 ali podobno), časovniško/števniško vezje Z80-CTC in seveda vrsto ločilnih ojačevalnikov za notranje in zunanje vodilo (notranje vodilo ter paralelni in serijski V/I kanali).

RAM plošča je proti vodilom in motnjam izolirana s histereznimi ojačevalniki in vsebuje skupno 96K zlogov dinamičnega pomnilnika (48 vezij tipa 4116). Pomnilnik na tej plošči je razdeljen v tri banke, in sicer 64K, 16K in 16K zlogov.

VIK plošča je tiskano vezje, ki vsebuje krmilnik za upogljivi disk (1791), krmilnik za neposreden pomnilniški dostop (Z80-DMA) in dva krmilnika za serijski V/I (vezji Z80-SIO). Vezje Z80-SIO vsebuje dva dupleksna kanala s programsko nastavljivo hitrostjo (vezje Z80-CTC). Ta plošča vsebuje tudi nadaljna vezja Z80-CTC, ki opravljajo funkcijo ure realnega časa in dodeljevanja procesorskega časa uporabnikom v večprocesorskem sistemu. Tudi ta plošča je proti vodilu izolirana s histereznimi ojačevalniki (s tremi stanji).

WDE plošča je krmilnik za vinčestrske diske (univerzalnega tipa) z gostiteljskim vmesnikom (vezje Z80-DMA) za sistem Delta 323/M. Ta modul omogoča prenos podatkov med glavnim pomnilnikom in vinčestrsko ploščo z maksimalno hitrostjo. Zgradba tega modula je podrobneje opisana v tekstu.

TVK plošča je modul, ki krmili prenose, povezane s tastaturo in video zaslonom (CRT krmilnik). To je krmilnik četrte generacije, ki ima realizirane tudi vse funkcije za sodobno procesiranje tekstov.

Plošča s preklopnim usmernikom (za 200 W) daje vse potrebne napetosti (5, 12, -12, 24 V) za ostale module sistema Delta 323/M.

Iz napisanega vidimo, da je sistem Delta 323/M konstrukcijsko in uporabniško zaokrožen glede na razpoložljive module, s katerimi lahko sestavimo zmogljivo mikroračunalniško konfiguracijo.

Slika 2.1 prikazuje tudi možnosti priključevanja periferije, in sicer takole:

- do štiri enote z upogljivimi diski,
- do štiri enote z vinčestrskimi diski,
- tastatura in video monitor,
- vrstični tiskalnik (600 vr/min),
- kvalitetni tiskalnik (procesiranje teksta),
- dodatni serijski in paralelni V/I kanali za periferijo.

Takšne periferne možnosti omogočajo bogato opremljenost sistema in na štiri dupleksne serijske kanale lahko npr. priključimo še štiri uporabniške terminale v večuporabniškem sistemu, se povezujemo z različnimi sistemi z uporabo sinhronih (SDLC, X.25) in asinhronih protokolov, krmilimo različne naprave (preko paralelnih kanalov) itn.

Programska oprema sistema 323/M temelji na operacijskem sistemu CP/M (ali MP/M), je torej uporabniško izredno prilagodljiva ter ima praktično neomejeno tržno bazo. Glede na milijonsko instalirano bazo CP/M sistemov v razvitih državah so dobavljivi prevajalniki praktično za vse visoke programirne jezike (vključno za ADO), močni sistemski storitveni paketi in seveda raznovrstna uporabniška programska oprema. Po tej plati lahko sistem Delta 323/M zadovolji tudi najbolj izbirčnega uporabnika, od direktorja, konstruktorja, razvijalca, do različnih služb, kot so storitvene dejavnosti, računovodstva, planerski oddelki, poslovni sistemi itd. CP/M nudi tudi bogato izbiro iger (od vesoljskih do šaha) ter pakete za učenje (jeziki, naravoslovni paketi, računalništvo).

3. Centralni procesorski modul

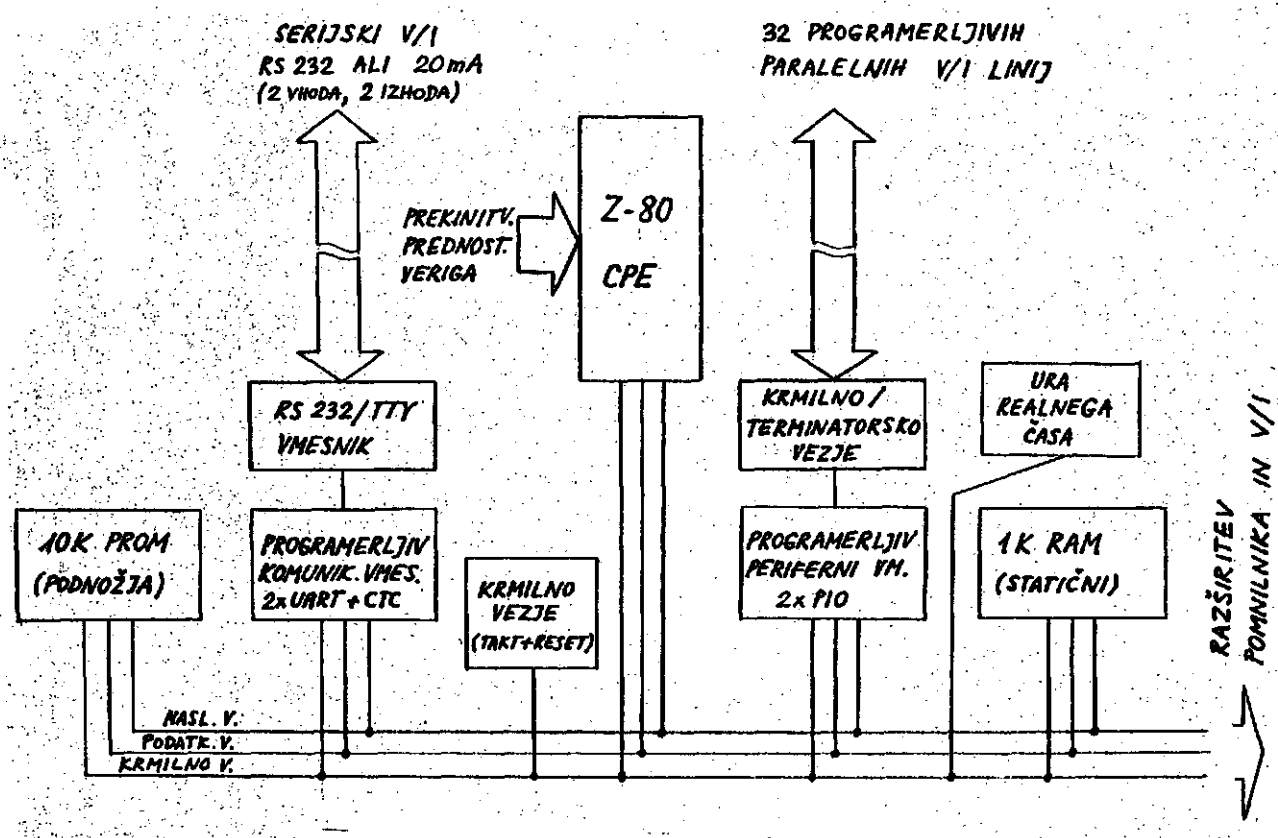
3.1. Okvirni podatki

Procesorski modul temelji na mikroprocesorju Z80 in je tako opremljen, da se lahko uporablja že sam zase. Seveda pa je zgrajen tako, da podpira še dodatne pomnilniške in vhodne/izhodne module.

Družina komponent Z80 je narejena v MOS LSI tehnologiji. Zaradi visoke stopnje integracije je možno sestaviti izredno zmogljiv sistem iz majhnega števila komponent. Po drugi strani pa se te komponente lahko kombinirajo s standardnimi TTL MSI vezji. Naštete lastnosti družine Z80 znatno znižajo materialne in programske razvojne stroške, hkrati pa omogočajo enostavno dodajanje zmogljivosti sistemu. Glavni razlog nadvlade MOS LSI komponent na mikroračunalniškem tržišču pa je nizka cena že tako majhnega števila komponent.

Glavne karakteristike modula so tele:

- avtomatsko resetiranje ob vključitvi,
- ponoven zagon je možen na lokacijah 0000H ali E000H, po izbiri,
- avtomatsko nastavljanje hitrosti serijskih prenosnih poti, odvisno od nastavitve terminala,
- ločena naslovna in podatkovna vodila,
- ura realnega časa,
- prednostna veriga prekinitev (daisy chain),
- 5 x 2K zlogov pomnilnika tipa EPROM,
- 1K zlogov statičnega RAMa (za beležko),



Slika 3.1. Bloćna shema CPE modula. Na plošči so visoko integrirana vezja iz družine Z80, dve periferni integrirani vezji UART, pomnilniška vezja tipa EPROM in RAM, taktni generator in ostala TTL krmilna vezja.

- tokovnozančni in RS-232 vmesnik,

- LED indikator za stanje HALT,

- štiri osembitna paralelna vrata (handshake),

- smeri vrat deljene po 4-bitnih blokih,

- vrata ojačena s TTL vezji, ki so na podnožjih in jih lahko konfiguriramo po izbiri.

Bločno shemo modula imamo na sliki 3.1.

3.2. Konfiguracija modula

Modul je realiziran na podaljšani plošči dvojnega evropskega formata. Vhodni in izhodni signali so speljani prek dveh standardnih konektorjev (2 x 32 priključkov). Glavni elementi CPE modula so naslednji:

- visoko integrirana vezja iz družine Z80 (CPE,PIO,CTC),

- periferni integrirani vezji UART,

- pomnilniška vezja tipa EPROM in RAM,

- krmilna TTL vezja,

- taktni generator.

Vse komponente družine Z80 lahko nadzirajo vodila in se vključujejo v prednostno verigo prekinitev brez dodatne zunanje logike. Napajane so samo z enim napetostnim nivojem (+5 V) in so med seboj popolnoma primerljive.

3.2.1. Mikroprocesor Z80

Z80-CPE (centralna procesorska enota) je 8-bitni mikroprocesor tretje generacije z izjemnimi zmogljivostmi in močjo.

V množici 8-bitnih mikroprocesorjev ima Z80 eno najmočnejših ukaznih množic (158 ukazov). Ta je primerljiva z vsemi 78 ukazi procesorja Intel 8080A (enak strojni kod), vključuje pa še operacije z nizi, biti, zlogi, besedami in bločne prenose, skupaj z učinkovitimi načini naslavljanja, kot sta indeksno in relativno.

Procesor je zelo hiter - sistemska ura 4 MHz in minimalni čas izvajanja instrukcije 1.0 us za Z80A ali 2.5 MHz in 1.5 us za Z80.

Dvojen nabor uporabniških registrov (skupaj 16) olajša načrtovanje sistemskih programov, "background-foreground" programiranje in procesiranje prekinitev na enem nivoju. 16-bitni registri omogočajo učinkovito procesiranje tabel in polj (večja propustnost in boljša izkoriščenost pomnilnika).

Procesor razpolaga s tremi načini procesiranja prekinitev: načinom 8080, prednostno verigo družine Z80 (daisy chain) in ne-Z80 perifernim načinom .(1)

3.2.2. Programirljivo paralelno periferno krmilno vezje Z80-PIO

Z80-PIO je TTL kompatibilen vmesnik med mikroprocesorjem Z80 in perifernimi enotami. CPE konfigurira PIO kot vmesno enoto do perifernih enot brez dodatne zunanje logike. Tipične takšne periferne enote so: večina tastatur, tiskalniki, PROM programatorji, čitalniki/tiskalniki papirnega traku,itd.

Vsi prenosi podatkov med CPE in periferijo se opravijo pod nadzorom prekinitvenih rutin. Vsa potrebna logika za implementacijo vgnezdenih prekinitev je že vgrajena v PIO. Druga zanimiva lastnost vezja PIO pa je ta, da lahko PIO prekine CPE ob določenih pogojih na periferiji. Zaradi tega procesor ne potrebuje zamudnega preverjanja stanj periferije.

PIO posreduje podatke med CPE in periferijo prek dveh neodvisnih večnamenskih vhodnih in izhodnih vrat (označenih z A in B). Vrata imajo po osem podatkovnih bitov in dva rokovalna signala, ki nadzirata prenos podatkov (data ready, strobe). Vrata programsko nastavimo na enega izmed štirih načinov delovanja: izhod zlogov, vhod zlogov, dvosmerni zložni (vhod in izhod zlogov) in dvosmerni po bitih. Izhodi vrat B lahko krmilijo darlingtonske tranzistorje (1.5 mA pri 1.5 V).

3.2.3. Števniško in časovniško vezje Z80-CTC

Štirikanalno števniško in časovniško vezje CTC lahko programiramo za širok spekter števniških in časovniških funkcij (na primer dajanje takta perifernim vezjem tipa Z80-SIO ali UART, realizacija ure realnega časa). Kanal ima števniško funkcijo, če mu dajemo števne impulze od zunaj. Če pa ga krmili sistemska ura, deluje kot časovnik. Programiranje je neposredno: vsak kanal se programira z dvema zlogoma oziroma s tremi, če uporabljamo prekinitve. Štirje kanali so med seboj popolnoma neodvisni, vsak ima vpisljiv in čitljiv dekrementirni števnik in izberemo lahko preddelilnik s faktorjema 16 ali 256. Ko pridejo števniki/časovniki na vredost nič, se avtomatsko nastavijo na začetno vrednost, trije kanali pa dajejo v teh primerih tudi izhodne signale (za izhodni signal četrtega kanala manjka priključek na integriranem vezju).

3.2.4. Periferno integrirano vezje UART

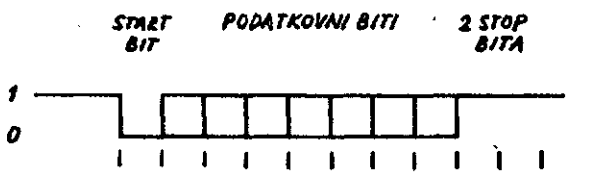
UART je univerzalno vezje glede na mikroračunalniško naslovno, podatkovno in krmilno vodilo. Vsi njegovi izhodi (podatkovni in statusni) premorejo tri stanja (stanji 0,1 ter stanje visoke impedance), njegovo krmiljenje je enolično čitalno/pisalno in selektivno. Več o vezju UART in o vezju z dvema UARTejema in CTCjem, ki je uporabljeno kot programirljiv komunikacijski vmesnik, najdemo v članku (2).

3.2.5. Pomnilniška vezja

Na CPE modulu so uporabljena vezja tipa EPROM 2716 (2K zlogov, napajanje +5V) in vezja tipa RAM 2114 (statični, 1K x 4 biti).

3.3. Avtomatsko nastavljanje hitrosti prenosa

Oglejmo si podrobnost, ki jo ima procesorski modul. Vezje UART je konfigurirano tako, da sprejema in oddaja ASCII znake v formatu na sliki 3.2.



Slika 3.2. Format ASCII znaka.

Uporabljeni algoritem za avtomatsko nastavljanje hitrosti prenosa (izbiro pravilnega takta za vezje UART na priključka RCP in TCP je tale:

- programsko detektiramo začetni impulz (začetni bit) prvega znaka, poslanega s konzole,

- izmerimo dolžino začetnega impulza,

- glede na dolžino začetnega impulza vstavimo v register časovne konstante kanala CTC vrednost, ki jo dobimo iz tabele,

- vstavljena časovna konstanta definira frekvenco odtipavanja naslednjih impulzov.

Z opisanim postopkom dosežemo želen učinek le, če je prvi poslan ASCII znak takšen, da ima podatkovni bit 0 vrednost 1 (npr. znak S). Prvi znak se seveda tudi narobe interpretira. Ta dejstva pa pri praktični uporabi ne predstavljajo nobene ovire. Startanje sistema poteka na tale način: po priklopu sistema na napetost (avtomatski reset) pritisnemo še na tipko S (lahko tudi na kakšno drugo iz zgoraj omenjene množice), ki predstavlja prvi znak, poslan s konzole, in že se monitor javi z znakom pripravljenosti (prompt).

Poglejmo še, kako je algoritem implementiran (sliki 3.3 in 3.4).

Sistem starta na lokaciji E000H. Za časovnik je uporabljen CTC kanal 0, z naslovom D8H. Inicializiramo programski števnik - registrski par DE - za merjenje dolžine začetnega impulza.

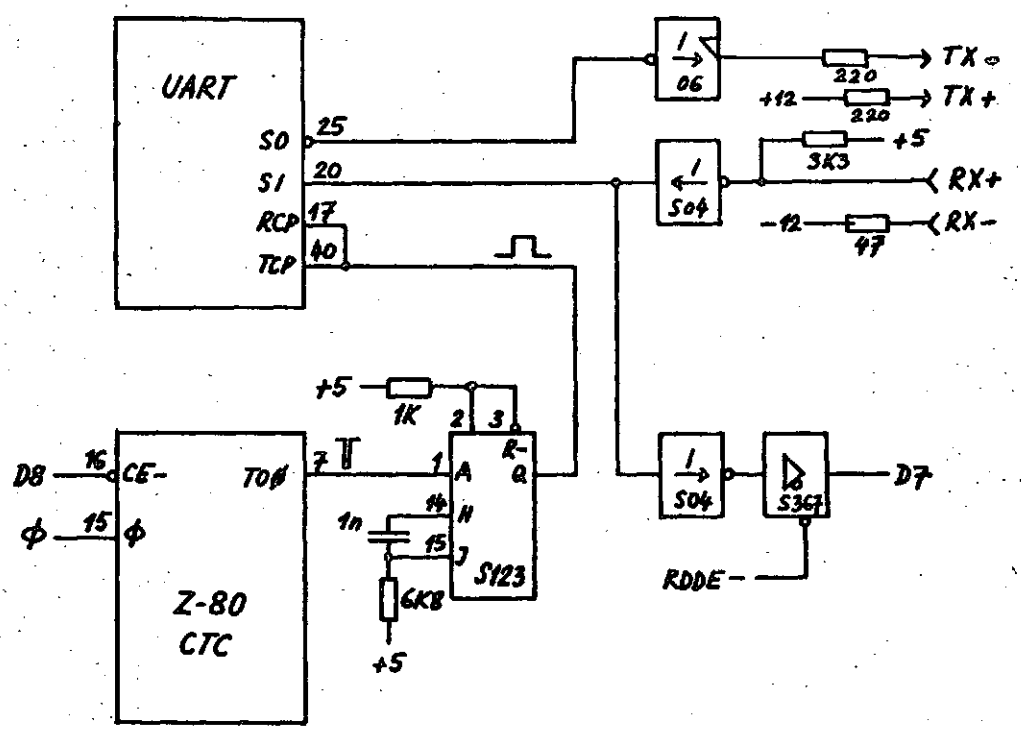
Odtipavamo podatkovno vodilo (z instrukcijo IN A,(DE)) vse dokler ni bit7 (D7) tega vodila postavljen na ena (to pomeni, da smo detektirali začetni impulz prvega znaka, poslanega s konzole - signal je namreč invertiran glede na vhod v UART). Ko detektiramo enico (v akumulatorju dobimo negativno vrednost - bit7 je znakovni bit), nadaljujemo z odtipavanjem in hkrati povečujemo vrednost registrskega para DE. Ko pade bit7 na podatkovnem vodilu zopet na nič, nam predstavlja vsebina DE dolžino začetnega impulza. Ostane nam še samo skok v tabelo, iz katere dobimo - glede na izmerjeno dolžino impulza - časovno konstanto za CTCjev register časovne konstante.

S tem je CTCjev kanal 0 programiran. Prvi zlog, ki smo ga poslali z ukazom OUT (D8),A , ko je bila v akumulatorju vrednost 5, je konfiguriral CTCjev kanal kot časovnik, v prednastavljivi register pa je dal vrednost 16. Drugi zlog, poslan z enakim ukazom, ko je v akumulatorju vrednost, dobljena iz tabele, pa je časovna konstanta za časovnik.

Hitrost prenosa (Baud rate) je definirana s formulo BR=A/i, kjer je A=0*1/16*1/16, 0=4.9152*10**6*1/2Hz, i=1,2,4,8,16,32 (časovna konstanta časovnika).

Monostabilni multivibrator 74LS123 obrne in podaljša impulze, ki jih daje CTC na sponki TO/0 (ko pade vrednost časovnika na nič, takt za UART).

Pojasnimo še oznake nekaterih signalov:
0 - sistemska ura
D7 - podatkovno vodilo bit7



Slika 3.3. Shema vezja za avtomatsko nastavljanje hitrosti serijskega prenosa. Za časovnik je uporabljen CTCjev kanal 0. Ko pade vrednost časovnika na nič, dobimo na sponki TO/0 taktni impulz.

D8 - signal, prožen z instrukcijo
OUT (D8),A, (CTC chip enable)
RDDE - signal, prožen z instrukcijo
IN A,(DE), (S367 chip enable)

```
        LD   A,05
        OUT  (D8),A      ;configure CTC
                         ;channel 0
        LD   DE,0001     ;initialize DE counter
LOOP1:  IN   A,(DE)      ;input from DE port
        JP   P,LOOP1     ;wait while input
                         ;value positive
LOOP2:  INC  DE          ;increment DE counter
        IN   A,(DE)      ;count until input
        JP   M,LOOP2     ;value negative
        LD   SP,TAB1     ;comparison table
        INC  SP
        POP  HL          ;pop value to HL
        SCF              ;CY:=1
        SBC  HL,DE       ;HL:=HL-DE-CY
        JR   C,F9        ;if HL>0 then
                         ;  pop new value to HL
        DEC  SP          ;        else
        POP  AF          ;  pop value to A
        OUT  (D8),A      ;set time constant
                         ;register
```

Slika 3.4. Del programa, ki avtomatsko nastavi hitrost prenosa.

4. Pomnilniški modul

4.1. Osnovni podatki o modulu

Pomnilniški modul obsega 96K zlogov dinamičnega pomnilnika tipa MK 4116-2. Na samem modulu se nahaja potrebna elektronika, ki omogoča izbiranje do osem pomnilniških bank s po 64K zlogov. V osnovni konfiguraciji pa je na enem modulu ena pomnilniška banka s 64K zlogi ter dve s 16K zlogi.

4.2. Opis modula

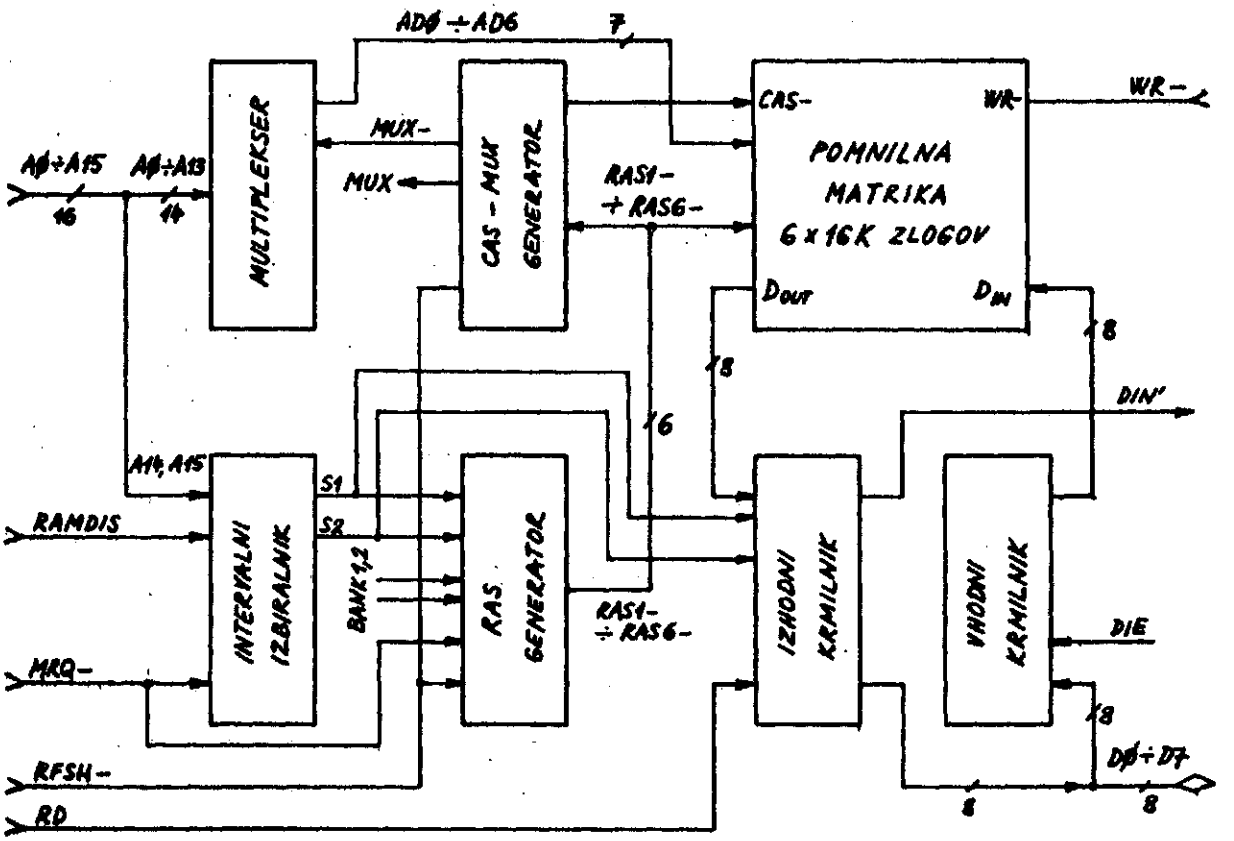
4.2.1. Signali modula

Signali, katere moramo upoštevati v krmilniku za dinamični pomnilnik, so tile:

- MRQ (memory request): signal se pojavi ob vsaki zahtevi po pomnilniku bodisi v čitalnem ali pisalnem ciklu;

- RFSH (refresh): je signal, ki se pojavi skupaj s signalom MRQ na začetku vsakega pomnilniškega osveževalnega cikla;

- WR (write): je signal, ki določa vpis podatkov iz podatkovnega vodila v pomnilnik, odsotnost le-tega pa lahko pomeni izpis podatkov iz pomnilnika;



Slika 4.1. Bločna shema pomnilniškega modula. V osnovni konfiguraciji je na plošči ena pomnilniška banka s 64K zlogi in dve s po 16K zlogi.

- RAMDIS (RAM disable) je vhodni signal, s katerim je mogoče izključiti pomnilnik v intervalu C000-FFFF, zaradi potreb po EPROM pomnilniku;

- D0,...,D7 - podatkovno vodilo;

- A0,...,A15 - naslovno vodilo;

- RD (read) - signal ko CPE čita;

- DIN (data input) je signal, s katerim se odpirajo vrata za signale s podatkovnega vodila v CPE modul;

- IORQ (I/O request) - signal, ki krmili izbiralnik bank;

- RESET - signal, ki vzpostavi začetno stanje elektronike.

4.2.2. Bločna shema

Iz sheme 4.1 je razvidno, da je pomnilni modul sestavljen iz pomnilne matrike, izhodno/vhodnega krmilnika, multiplekserja, CAS-MUX generatorja, RAS generatorja, intervalnega izbiralnika in bančnega izbiralnika.

- Pomnilna matrika je sestavljena iz 6 vrstic s po 16 K x 8 biti. Posamezne vrstice izbirajo signali RAS 1 - do RAS 6 - katere generira RAS generator. Vhodni signali za to matriko pa so še podatkovni signali DIN (0-7) in DOUT (0-7), adresni signali AD 0 do AD 7 ter CAS- in RAS-signala.

- Izhodni krmilnik omogoča ob čitanju pomnilnika prenos podatka iz pomnilne matrike na podatkovno vodilo. Vhodni krmilnik pa omogoča nasprotno funkcijo.

- Multiplekser je potreben zaradi uporabe 16 K bitnih pomnilnikov s 16 nogicami, saj se mora 14 bitni naslov naložiti v pomnilno vezje v dveh korakih, katere krmilita RAS in CAS signala.

- CAS-MUX generator zagotavlja pravilne časovne razmike med CAS in MUX signaloma, le-te pa pogojujejo RAS in RFSH signali.

- RAS generator omogoča izbiranje prave pomnilne vrstice v pomnilni matriki, hkrati pa je izvor signalov za CAS-MUX generator.

- Intervalni izbiralnik generira na podlagi vhodnih signalov A14, A15, MRQ in RAMDIS potrebne signale za krmiljenje RAS generatorja in izhodnega krmilnika.

- Bančni izbiralnik pa omogoča preklop pomnilnih bank.

4.2.3. Uporaba dinamičnih pomnilnikov s 16 nožicami

Večina sodobnih dinamičnih pomnilnih vezij je zaprtih v integrirana vezja s 16 nožicami. Zato taka vezja zahtevajo multipleksiranje naslova, da se lahko vseh 14 naslovnih bitov vnese v pomnilnik. V našem primeru imamo vezje s pomnilno matriko s 128 vrsticami in 128 kolonami. Dinamični pomnilniki shranjujejo informacijo v obliki naboja v majhnem kondenzatorju. Ta naboj pa, če hočemo obdržati informacijo, moramo občasno osveževati. To nalogo nam v veliki meri olajšuje Z80-CPU, saj izvaja osveževalni cikel vsakič, ko CPE zahteva dostavo ukaza. Zaradi tega Z80 procesorju ni potrebno rezervirati čas za osveževanje v obliki "čakalnih stanj" ali pa podaljševanje urinega cikla.

Pri tem pa se imamo nekaj omejitev pri osveževanju z Z80 procesorjem:

- reset signal mora biti < 1 ms

- podaljšani "wait state" mora biti < 1 ms

- podaljšani DMA cikel mora biti < 1 ms

- osnovni taktni cikel mora biti < 1.22 MHz.

4.2.4. Opis razvojnih izhodišč

Poskušali smo razviti dinamični pomnilnik, ki bi čimbolje ustrezal sistemu z Z80 procesorjem. Na sliki 4.1 je podana bločna shema pomnilnika.

Delovanje vezja lahko opišemo takole:

RAS- signal se generira iz MREQ, RFSH, BANK signalov ter dela naslova. Samo ena vrstica pomnilnikov dobi RAS- signal v času dosega ukaza, čitanja/pisanja v pomnilnik. RAS- pa se generira za vse vrstice v času osveževanja. MREQ- se invertira ter se ga zakasni tako, da dobimo MUX in CAS- signal. MUX se generira zakasnjen za 65 ns in se uporablja za multipleksiranje naslova. Ta zakasnitev je izbrana tako, da zagotovi pravilni čas dostopa vrstičnega naslova. Po 110 ns se generira CAS-signal, vendar samo takrat, ko ni osveževalnega cikla. Ko CAS- nastopi, je izbrana prava pomnilna lokacija.

Izračunan "worst case" čas dostopa znaša 347 ns, kar omogoča (ob zahtevanem času dostopa od CPE 450 ns) še 103 ns za dodatna vezja na kontrolnih ter naslovnih linijah.

Razviti pomnilnik daje dobre rezultate tam, kjer je zahteva po večjem obsegu dinamičnega pomnilnika, ter pri aplikacijah, ki zahtevajo direktni dostop do pomnilnika (DMA).

5. V/I IN DISKOVNI MODUL

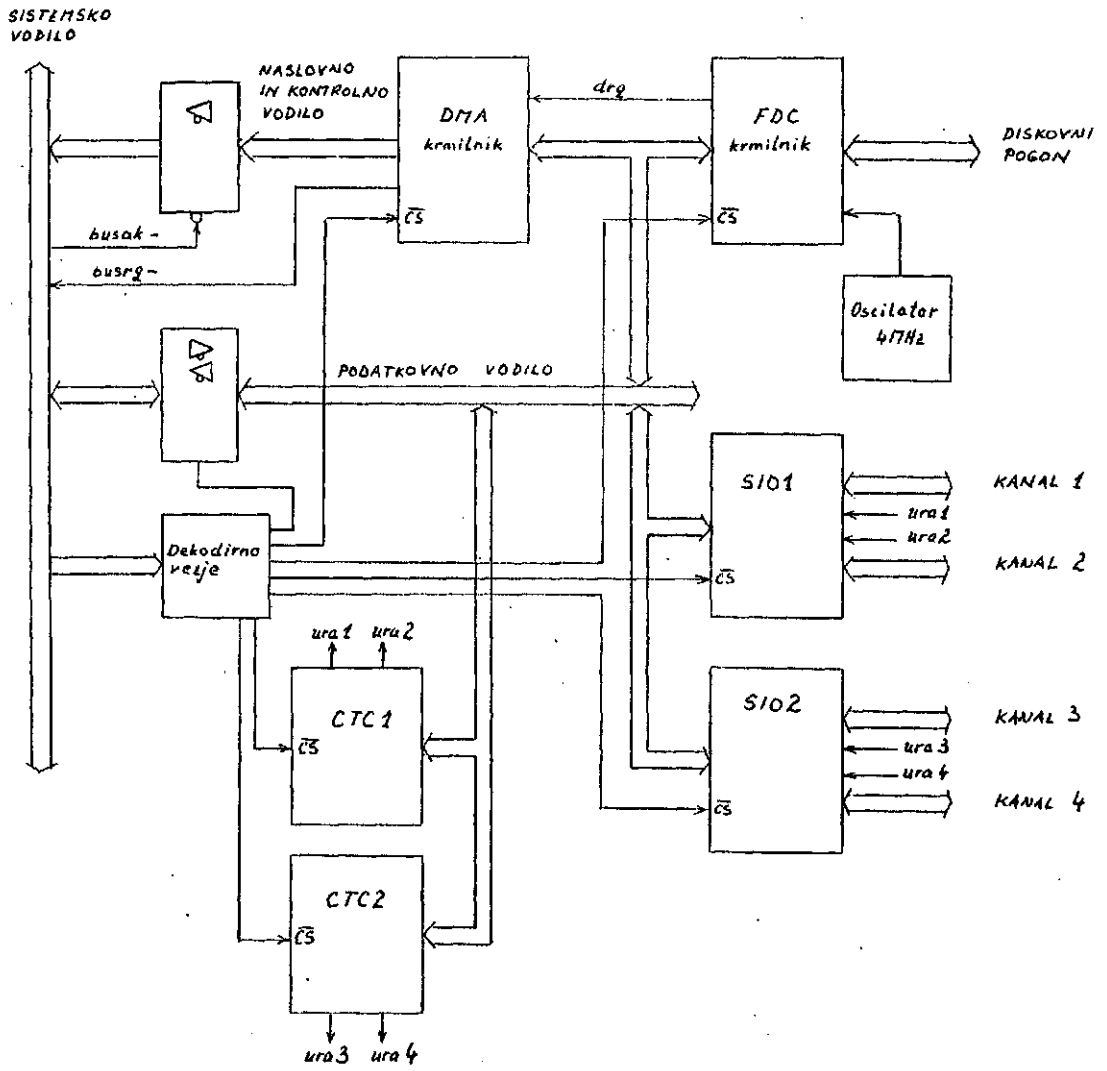
5.1. Osnovni podatki o modulu

Diskovni modul omogoča priključitev pogonov za gibke diske na sistem. Povečuje tudi komunikacijske zmožnosti sistema. Modul lahko krmili do 8 diskovnih pogonov z enojno ali dvojno gostoto zapisa. Prenos podatkov poteka po DMA principu. Procesor samo začne čitanje oz.zapisovanje, DMA krmilnik pa prenaša podatke naravnost v pomnilnik oz. iz pomnilnika. Na istem modulu so še štirje serijski kanali za prenos podatkov. Vsakemu posebej je mogoče programsko nastaviti način delovanja : sinhrono ali asinhrono in hitrost prenosa. Takt, ki določa hitrost prenosa, generirajo časovniki. Hitrost prenosa je nastavljiva v mejah od 50-38400 Baudov. Bločna shema modula je prikazana na sliki 5.1. V nadaljevanju si bomo podrobneje ogledali DMA princip, krmiljenje diskovnega pogona in možnosti, ki nam jih nudi posamezni serijski kanal.

5.2. DMA prenos

Za DMA prenos skrbi krmilnik Z80-DMA. To je 40 pinsko integrirano vezje. Od podobnih krmilnikov drugih družin se razlikuje po tem, da se v njem podatek med prenosom za kratek čas shrani. Krmilnik namreč generira regularne

Slika 5.1. Bločna shema diskovnega modula. Modul vsebuje poleg diskovnega in DMA krmilnika še štiri vhodno-izhodne kanale, ki jim je mogoče programsko nastaviti hitrost prenosa in vrsto prenosa (sinhroni, asinhroni).



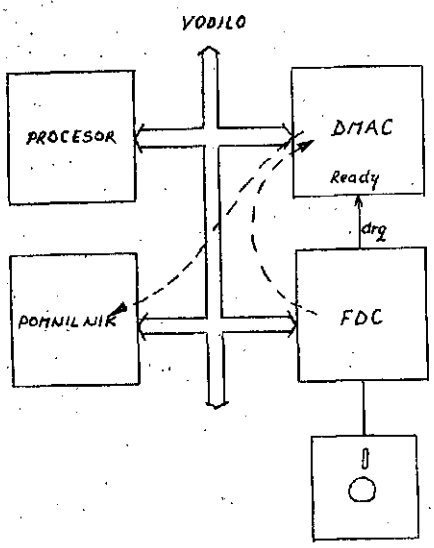
kontrolne signale in naslov za izvor in za cilj podatka. Če torej čitamo blok podatkov iz V/I kanala v pomnilnik, potem bo krmilnik generiral v enem DMA ciklu najprej naslov V/I vezja, IORQ in RD signal (V/I čitalni cikel), nato pa bo poslal podatek naprej. To stori tako, da starta cikel zapisa v pomnilnik (naslov, MREQ in WR signal). DMA krmilnik deluje torej na popolnoma enak način kot procesor. Edina razlika je v tem, da DMA krmilnik za svoje delo ne potrebuje programa v pomnilniku. To pomeni, da ne zapravlja časa s čitanjem in dekodiranjem ukazov, kar bistveno vpliva na hitrost prenosa.

Krmilnik omogoča tri vrste prenosa: zlog za zlogom, zvezni prenos in bruhajoč prenos (burst). Pri prvem načinu sprosti DMA krmilnik vodilo po prenosu vsakega zloga. Pri drugem načinu pa krmilnik ne sprosti vodila do zaključka prenosa celega bloka. Če preide med prenosom READY linija v neaktivno stanje, zadrži krmilnik vodilo in čaka na ponovno aktiviranje. Pri bruhajočem načinu pa krmilnik prenaša podatke nepretrgoma, dokler je aktivna READY linija. Ko preide ta v neaktivno stanje, sprosti krmilnik sistemsko vodilo.

V našem sistemu uporabljamo prvi način prenosa. Ko prejme krmilnik za diskovni pogon FDC (Floppy Disk Controller) en zlog iz diska, aktivira READY linijo DMA krmilnika. Ta pošlje zahtevo po vodilu procesorju. Ko prejme potrditev, prečita podatek iz podatkovnega registra v FDCju in ga zapiše v pomnilnik (slika 5.2).

Seveda moramo najprej programsko definirati režim delovanja in naslove izvora in ponora. Slika 5.3 prikazuje tak program. Ko procesor izvede podprogram "Dmainit", je DMA krmilnik pripravljen, da prenese 128 zlogov iz naslova 7 (V/I) na naslov 1000-107FH (pomnilnik). Za prenos posameznega zloga je potreben visok nivo na liniji READY. Prenos poteka brez kakršnekoli intervencije procesorja.

DMA krmilnik generira pri čitanju in zapisovanju kontrolne signale, ki imajo enak časovni potek kot procesorjevi. Časovni potek kontrolnih signalov je mogoče programsko spreminjati tako, da čimbolj ustrezajo hitrosti pomnilnika in perifernih vezij.



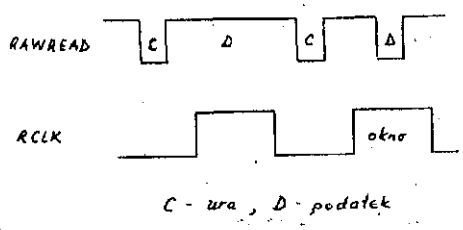
Slika 5.2. DMA prenos podatkov iz diskovnega krmilnika v pomnilnik. DMA prenos vodi DMA krmilnik, ki v času prenosa prevzame vlogo procesorja.

```
dmaini: ld      hl,data ;init kazalec
        ld      c,dmaprt
        ld      b,12
        otir
        ret
;
data:   db      79h     ;B --> A
        dw      buff    ;naslov A
        dw      length  ;dolzina bloka
        db      14h     ;A je pomnilnik, naslov
;                       ;se povecuje za 1
        db      28h     ;B=V/I port, naslov se
;                       ne spreminja
        db      85h     ;nacin zlog za zlogom
        db      fdcprt  ;naslov B
        db      8ah     ;READY: visok nivo aktiven
        db      0cfh    ;shrani naslova obeh
;                       ;portov in postavi
;                       ; stevec zlogov na 0
        db      87h     ;omogoci DMA delovanje
;
;
dmaprt  equ     0       ;naslov DMA krmilnika
fdcprt  equ     7       ;naslov FDC podat. reg.
length  equ     7f00h   ;dolzina bloka je 128 zl.
buff    equ     0010h   ;naslov buferja
;
        end
```

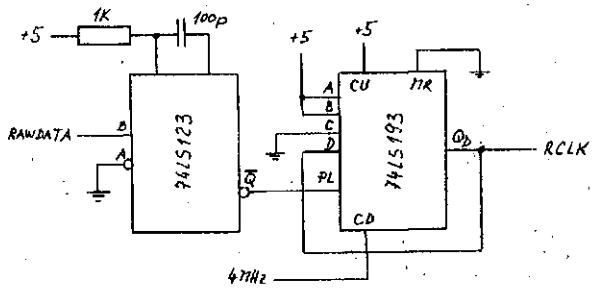
Slika 5.3. Program za inicializacijo DMA krmilnika. Vsi parametri, ki so zbrani v vektorju "data" se z OTIR ukazom vpišejo v DMA krmilnik.

5.3. Krmiljenje diskovnega pogona

Pogoni za gibke diske so postali že kar standardne periferne naprave v mikroračunalniških sistemih. Zato so se po pričakovanjih pojavila na trgu integrirana vezja za krmiljenje takih pogonov. Prva je bila na tem področju firma Western Digital s svojim klasičnim 1771 čipom. V našem modulu



Slika 5.4. Zahtevani časovni potek urnega in podatkovnega signala za kontroler 1791. Dejansko ločevanje oz. izločanje podatkovnih bitov izvrši sam kontroler.



Slika 5.5. Vezje za ločevanje urnih in podatkovnih signalov. Vezje formira samo "okno", ki ga potrebuje kontroler 1791 za separacijo.

uporabljamo za krmiljenje diskovnega pogona krmilnik 1791, ki je kompatibilen z omenjenim 1771, le da omogoča še krmiljenje pogonov z dvojno gostoto zapisa. Krmilnik ima invertirano podatkovno vodilo. Za svoje delovanje potrebuje takt s frekvenco 4MHz. Krmilnik 1791 omogoča programsko pozicioniranje pisalno/čitalne glave ter čitanje oz. zapisovanje enega ali več sektorjev oz. kompletne sledi. Poslednja funkcija nam omogoča formatiranje gibkih diskov. Edina funkcija, ki je krmilnik ne izvaja, je ločevanje podatkovnih signalov od urnih. V našem sistemu smo uporabili za ločevanje metodo s števcem. Krmilnik 1791 potrebuje na svojem RCLK vhodu signal, ki omogoča izločitev podatka iz signala, ki pride iz diska (podatki taktni signali) z enostavno IN operacijo. Vezje za ločevanje mora torej generirati "okno", ki definira veljavnost podatka. Dejansko ločevanje pa se izvrši interno v krmilniku (slika 5.4). Vezje za ločevanje je prikazano na sliki 5.5.

5.4. Vhodno/izhodni kanali

Štirje vhodno/izhodni kanali so realizirani z dvema Z80-SIO vezjema. Vsi kanali omogočajo popolen dupleksni prenos in sinhroni ali asinhroni način prenosa. Vsi krmilniki (DMA,FDC,SIO in CTC) so vezani v serijsko prekinitveno verigo. Hitrost prenosa je programsko nastavljiva v mejah od 50 - 38400 Baudov. Definirajo jo časovniška vezja Z80-CTC. Štirje časovniki (timer) so prosti in

```
        title sioini
sioini: ld      hl,data1
        ld      c,sioctr        ;init SIO
        ld      b,7
        otir
;
        ld      a,3
        out     (timer),a       ;reset casovnika
        ld      a,45h
        out     (timer),a       ;stevcni nacin
;                               ;sledi
;                               ;casovna konst.
        ld      a,const
        out     (timer),a       ;casovna konst.
        ret
;
data1:  db      18h             ;reset kanala
        db      03              ;kazalec na WR3
        db      0c1h            ;Rx8 bitov
;                               ;omogoci sprejem
        db      04              ;kazalec na WR4
        db      4ch             ;x16, 2 stop bita
        db      05              ;kazalec na WR5
        db      68h             ;Tx8 bitov
;                               ;omogoci oddajo
;
;
sioctr  equ     0ah             ;SIO ctrl/st reg.
const   equ     82h             ;konst. za 300 Bd
timer   equ     10h             ;CTC1,casovnik 0
        end
```

Slika 5.6. Program za inicializacijo asinhronega kanala. Najprej se nastavi perioda časovnika na zahtevano hitrost prenosa. Nato pa se določi vrsta in način prenosa.

jih lahko uporabnik svobodno uporablja. Pri asinhronem načinu je mogoče definirati število bitov v znaku (5 do 8), število ustavitvenih bitov in konstanto za deljenje takta (1,16,32,64).

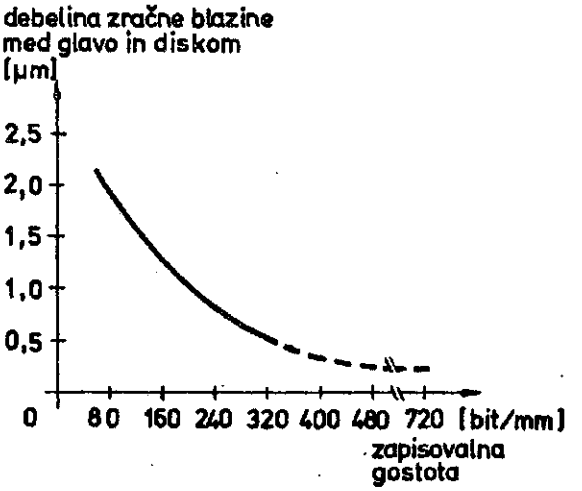
Sinhroni način pa omogoča pošiljanje in sprejemanje bitnega ali zlogovnega protokola (IBM Bisync, SDLC in HDLC). Vsi parametri so programsko nastavljivi. Vsak kanal ima poleg podatkovnih še množico kontrolnih in statusnih registrov. Vsi kontrolni in statusni registri zasedajo samo en naslov v V/I prostoru. V kontrolne registre je mogoče samo zapisovati, statusne pa je mogoče samo čitati. Na sliki 5.6 je prikazan program, ki inicializira en kanal za asinhroni prenos in nastavi hitrost na 300 Baudov.

6. Modul za uporabo vinčestrskih diskov

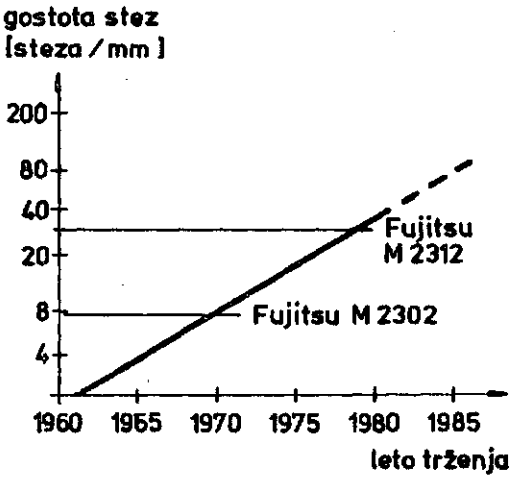
6.1 Dosežki vinčestrske tehnologije

Kot posledica zavestnega iskanja novih magnetnih materialov ter vsestranske miniaturizacije in iz potrebe po čim cenejših množičnih pomnilnikih so se v zadnjem času na tržišču pojavile takoimenovane vinčestrske diskovne enote. Zanje je značilno, da so diski kot nosilci aktivne magnetne plasti trdni, nezamenljivi in s posebnim ohišjem hermetično zaprti. Ta ločenost od okolja pa odpira nove možnosti za mnoge izboljšave, ker je nevarnost mehanskih poškodb občutljive magnetne površine in zapisne/bralne glave zaradi nečistoč praktično odpravljena.(3),(4)

Tipična vinčestrska glava tehta z uvedbo posebne tankoplastne tehnike le še 1N napram 30N pri navadnih diskovnih enotah z zamenljivimi diski. Tudi plast zraka, na kateri pri vrtenju diska glava takorekoč plava, se je pri vinčestrskih enotah zmanjšala na cca. 1 um, kar omogoča zapisovanje podatkov na disk z izredno gostoto (do 8000 bit/cola oz. 320 bit/mm). Z izboljšano zanesljivostjo in hitrostjo nastavljanja glave v radialni smeri je bilo moč povečati tudi število stez na dolžinsko enoto (do 1000 steza/cola oz. 40 steza/mm). Z vsem tem je vinčestrska tehnika dosegla znatne izboljšave napram doslej popularni tehnologiji upogljivih diskov za vsaj en velikostni razred. Tako znaša tudi hitrost prenosa podatkov na disk že nad 10 MBaud, povprečni čas dosega podatkov pa je le še okoli 10 ms (sliki 6.1 in 6.2).

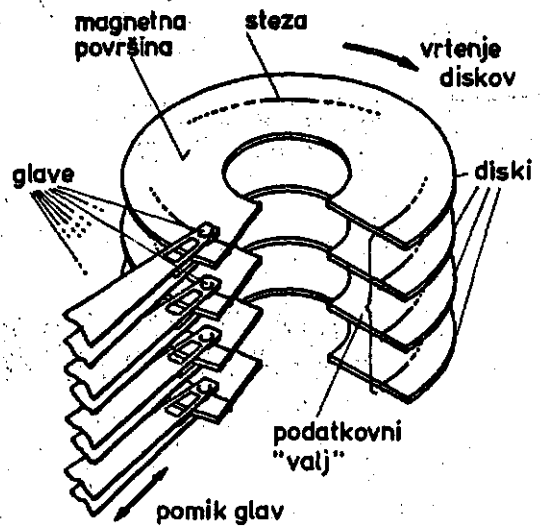


Slika 6.1. Digram prikazuje porast podatkovne gostote, če zmanjšamo oddaljenost zapisovalno/bralne glave od magnetne površine diska.



Slika 6.2. Diagram prikazuje dosegljivo gostoto stez kot posledico izboljšav pri postavljanju zapisovalno/bralne glave in v tehnologiji glave same.

Slika 6.3. Shematski prikaz razporeditve diskov, zapisovalno/bralnih glav ter podatkovnih stez na magnetni površini diskov.



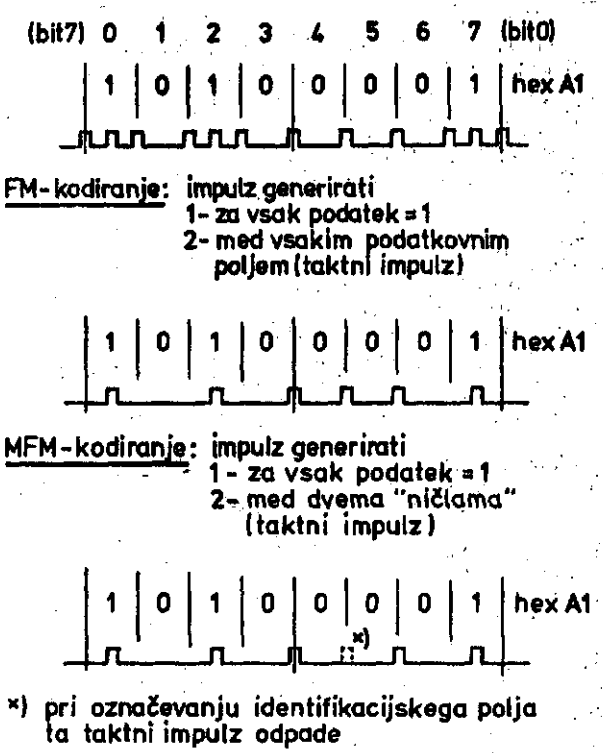
Danes se proizvajajo vinčestrske diskovne enote z enim do štirimi diski in premerom diskov 5, 8 in 14 col. Ker je magnetna plast nanešena na diske na obeh straneh, imajo take enote po 2 do 8 bralno-zapisovalnih glav, dodatno pa še posebno bralno glavo, ki v vlogi tahometra omogoča držati vrtenje diskov na konstantni vrednosti 3000 obrat/minuta (slika 6.3). Pomnilniška zmogljivost takih diskovnih enot znaša od nekaj milijonov zlogov pri majhnih in enodiskovnih enotah do več 100 milijonov zlogov pri največjih vinčestrskih enotah.

Ker so v vinčestrskih napravah diski trajno nameščeni, imajo take enote često predvideno možnost prenosa podatkov na druge medije, večinoma kasetnike ali upogljive diske, ki omogočajo fizično zamenjavo podatkov in arhiviranje.

6.2. Tehnika zapisa podatkov

Aktivna površina diskov je do 0,3 um tanka magnetna plast, ki jo zapisovalni tok skozi glavo magnetno polarizira. Pri branju pa magnetno modulirana sled na vsakem prehodu iz ene polaritete v drugo inducira v bralni glavi napetost, proporcionalno zapisnemu toku skozi isto, takrat zapisovalno glavo.

Za zapisovanje podatkov na magnetne diske, gibke ali trdne, se uporablja posebno moduliran signal. Zanj je značilno, da mora poleg serijsko nanizanih podatkov vsebovati tudi časovno merilo za sinhronizacijo, saj je le tako možno najrazličnejše impulzne nize pravilno brati, tolmačiti. Medtem ko je za zapis na gibke diske z enojno gostoto v uporabi tkzv. FM modulacija, se pri dvojni gostoti tako kot pri vinčestrskih diskih zapisuje večinoma z MFM signalom (modified-frequency-modulation). Pri FM se v enakomernih časovnih presledkih pojavljajo posebni sinhronizacijski impulzi, prostor med njimi pa je namenjen podatkovnim impulzom. Če je podatek 1, se med dvema taktnima impulzoma pojavi še podatkovni impulz; če pa je vrednost podatka 0, ta impulz izostane. Tako je treba le sinhrono s tem časovnim merilom odtipati



Slika 6.4. Primerjava FM- in MFM- kodiranja podatkov.

vmesne prostore, posamezne "bite" združiti v paralelne "zloge" in že je serijsko zapisan podatek z diska uporaben za nadaljnjo obdelavo.

Pri diskih s tkzv. dvojno gostoto zapisa ni povečana fizična gostota zapisa na disku, temveč le vsled racionalnejše modulacije je možno obseg podatkov na disku skoraj podvojiti. Tudi pri tej modificirani FM (MFM) se namreč za vrednost podatka 1 impulz pojavi, za vrednost 0 pa izostane. Novo je le, da sinhronizacijski impulzi nastopajo le, če je podatkovna vrednost dva ali večkrat zapovrsti 0. Pri MFM nastopajo torej taki "nepodatkovni" impulzi napram FM mnogo redkeje. Ker so torej skoraj vsi impulzi podatkovni, se kapaciteta istih plošč s pomočjo MFM praktično podvoji (slika 6.4).

6.3. Naloge vmesnika

Glavna naloga vmesnika je časovno usklajevanje možnosti in zahtev računalnika na eni strani s togo se odvijajočim "programom" mehanske enote, t.j. vinčestrskega diska na drugi strani. Vrtenje diska je konstantno, hitrost prenosa podatkov je konstantna in tako je informacija, kateri podatkovni sektor se prav v tem trenutku nahaja pod čitalno glavo, časovno nepremakljiva vrednost.

Komuniciranje s procesorjem računalnika se opravlja preko sistemskega vodila. Ker pa gre tu za včitavanje ali izpisovanje velikih količin podatkov, opravi to delo najgospodarneje in najhitreje specializiran procesor na vmesniku, tkzv. DMA krmilnik. Procesor računalnika mu po nekaj uvodnih operacijah omogoči direktni dostop do sistemskega pomnilnika, medtem pa je sam prost za izvajanje drugih ukazov.

Na drugi strani pa mora vmesnik pripraviti vse potrebno, da bo zapisovalna glava mogla ob pravem času na pravo mesto diska zapisati podatke in to na tak način, da bo možno te podatke pozneje zopet brez napak brati. Precejšen del te naloge opravi že elektronika vinčestrske enote sama. Ta vsebuje končne stopnje za generiranje primernega zapisovalnega toka, na njej so ojačevalniki šibkih signalov iz bralnih glav. Tudi krmiljenje koračnih motorčkov za pomik glav v radialni smeri (pozicioniranje na posamezne sledi) in zagotavljanje konstantnega vrtenja diskov preko posebnih faznih regulatorjev je stvar elektronike, ki je del diskovne enote. Zato obstaja tudi med vmesnikom in diskovno enoto stičišče, kjer signali niso več elementarni, temveč imajo tako tukaj kot na procesorski strani običajne TTL nivoje.

Prenos krmilnih signalov je asimetričen in jih je za pogon večjih bremen (dolge linije) treba še ojačiti. Taki signali so:

- izbira bralne/zapisovalne glave. S tremi linijami določamo eno od osmih glav in s tem izmenjavo podatkov z eno od osmih strani štirih diskov

- 4 signali za izbiro ene od štirih možnih vinčestrskih diskovnih enot

- 2 signala za omogočanje pisanja oz. branja

- 2 signala za določanje smeri in števila korakov pri pomikanju glave in še

- nekaj linij za javljanje pripravljenosti diskovne enote (ready), začetne marke (index), pozicioniranje glave končano (seek complete) in podobno.

Prenos "visokofrekvenčnih" signalov, kot so:

- bralni podatki (read data),

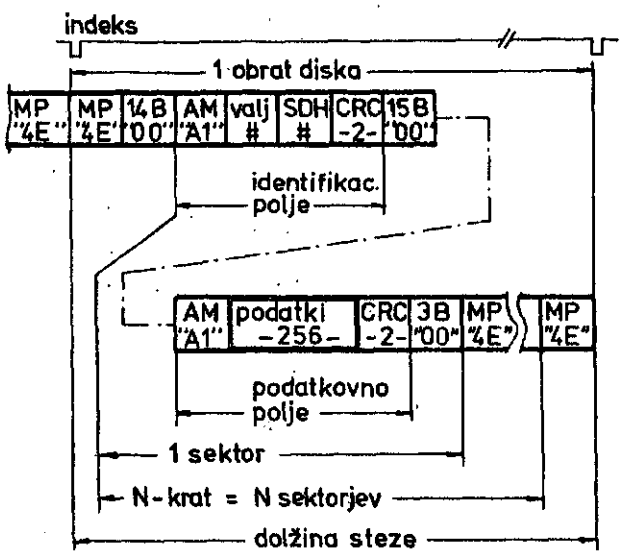
- zapisni podatki (write data) in

- izhod oscilatorja za sinhrono zapisovanje (write clock) je možen le preko impedančno pravilno zaključenih simetričnih linij. Na oddajni strani je treba predvideti posebne napajalne stopnje (line driver), na sprejemni pa diferencialne sprejemnike, ki so za motnje in presluh na istočasno obeh linijah takega simetričnega sistema neobčutljivi.

6.4. Formatiranje diska

Podatki so na diskih zapisani serijsko, bit za bitom, v koncentrično razporejenih sledeh. Ker so sledi dolge, so te razdeljene na posamezne sektorje. Čeprav je tudi sektor razdeljen na več delov in ima torej sam svoj format, je to najmanjša enota, ki jo je možno direktno nasloviti. Prvi del sektorja je tkzv. identifikacijsko polje (ID field), ki vsebuje zaporedne številke sledi, glave in sektorja, pa tudi začetno sinhronizacijsko marko in na koncu kontrolo pravilnosti podatkov (2 CRC zloga). Šele drugi del sektorja je pravo podatkovno polje (data field), ki ima tudi svojo predhodnico v sinhronizacijskem zlogu in na koncu za ponovno kontrolo pravilnosti prenosa podatkov zopet dva CRC zloga. Med tema dvema poljema, kot tudi med sektorji, so prazni prostori, ki rabijo za sinhronizacijo raznih oscilatorjev in za priprave novih operacij. Vrednost podatkov je v teh medprostorih običajno 0 in MFM signal je le nosilec sinhronizacijskih impulzov (slika 6.5).

Slika 6.5. Vsaka podatkovna steza na disku je razdeljena na več sektorjev. Ti so oštevilčeni v posebnem predhodnem identifikacijskem polju.



Dolžina podatkovnega polja ni poljubna, temveč togo vezana na dogovor ob formatiranju diska. Običajne velikosti so 128, 256, 512 in več zlogov, možne pa so tudi vmesne vrednosti. Vsota vseh zlogov podatkovnega polja, vmesnih prostorov in identifikacijskega polja daje dolžine in format sektorja. Število sektorjev v sledi je seveda omejeno, saj se vsaka sled prične z indeksnim impulzom in ima v odvisnosti od gostote zapisa le določeno kapaciteto, npr. 12000 zlogov/sled. Če ta kapaciteta ni mnogokratnik dolžine formatiranega sektorja, ostane del sledi na koncu neizrabljen. Neupoštevanje veljavnega formata je nedopustno, saj bi povzročilo zmedo med pomenom posameznih zlogov in tako onemogočilo ločitev okvirja od podatkov.

Podatkovni obseg sledi je neodvisen od njene lege na disku. Ker je hitrost medija pod zapisovalno glavo v notranjosti diska manjša, je potrebno zapisovalni tok skozi glavo tam zmanjšati. Brez te prilagoditve toka hitrosti bi premočno magnetno polje rušilo tudi sosedna področja in uničevalo podatke. S to korekturo pa mora sovpadati časovna premaknitev nekaterih MFM impulzov nasproti nominalni vrednosti. Izbiro opravi določen algoritem v odvisnosti od tega, ali podatkovni vzorec vsebuje več 0 ali več 1.

6.5. Realizacija nalog vmesnika

Na diskovni strani so naloge vmesnika mnogoštevilne, saj mora ta pri pisanju na disk

- pozicionirati zapisovalno glavo na pravo sled izbrane vinčestrske enote,

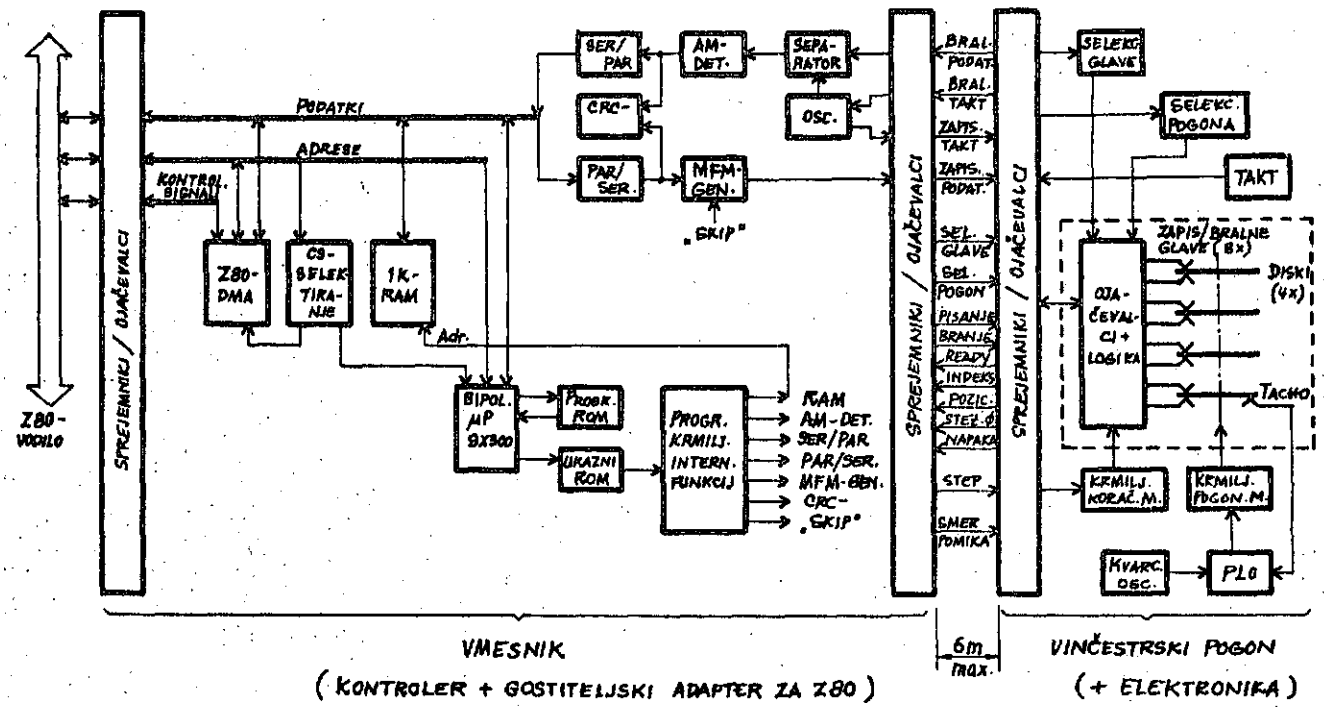
- poiskati zahtevani sektor,

- se prepričati, če pri tem niso nastopile kake napake prenosa,

- poiskati v sektorju začetek podatkovnega polja in

- nato v to polje vpisati nove podatke,

Slika 6.6. Blok sheme vinčestrskega pogona s svojo elektroniko ter vmesnika z gostiteljskim adapterjem za sistem Z80.



- nazadnje pa še prekontrolirati, če pri tem zapisu ni prišlo do napak.

Podobno je pri branju z diska, le da je smer pretoka podatkov, ko gre za podatkovno polje, nasprotna. V obeh primerih se podatke identifikacijskega polja le bere. Spreminjamo oz. na novo definiramo jih le pri operaciji formatiranja diska.

6.5.1. Pomožna integrirana vezja

Pri realizaciji teh zahtevnih nalog je v veliko pomoč niz petih integriranih vezij firme Western-Digital, tkzv. WD1100 Winchester Controller Chip Set. Izvedeni so v "N-Channel Silicon Gate" tehnologiji, ki dovoljuje prenosno hitrost 5 MBaudov in so tako uporabni tudi za komunikacijo z vinčestrskimi diskovnimi enotami Fujitsu M 2302B, ki jih uporabljamo v sistemu Delta 323/M (slika 6.6).(5)

Z nizom WD1100 je možno opraviti tele naloge:

- WD1100-03 AM-Detector: To integrirano vezje toliko časa primerja podatkovni vzorec z diska z vzorcem A1, Hex, ki ga pozna kot dogovorjenega za označevanje identifikacijskega oz. podatkovnega polja, dokler se oba zloga ne ujemata, kar ta detektor (AM, address mark) na posebnem izhodu javi. Procesor vmesnika s tem ve, da pomenijo naslednji trije zlogi (kot je bilo pri formatiranju dogovorjeno za identifikacijsko polje) številko sledi, številko glave in številko sektorja. Da se pri ugotavljanju teh podatkov ni vrinila napaka prenosa oz. da je možnost napake izločena, poskrbita naslednja dva zloga. Posebni

- WD1100-04 CRC Generator/Checker v svoji vlogi kot preverjevalec generira preko posebnega polinoma iz gornjih podatkov za številke sledi, glave in sektorja dva posebna zloga, imenovana CRC zloga. "CRC-Checker" nato ta dva zloga primerja z onima iz identifikacijskega polja. Pri enakosti lahko AM detektor poišče začetek podatkovnega polja. Po prenosu teh podatkov se CRC primerjava ponovi. CRC napako vmesnik javi kot fatalno. Taki podatki so brez vrednosti. Nadaljnji potek je stvar programske obdelave napak. Druga polovica tega integriranega vezja je CRC-generator, katerega naloga je, da na koncu vsakega podatkovnega prenosa samostojno doda po dva CRC zloga, generirana iz vsebine prenešenih podatkov. Polinom za generiranje CRC zlogov je x16+x12+x5+1.

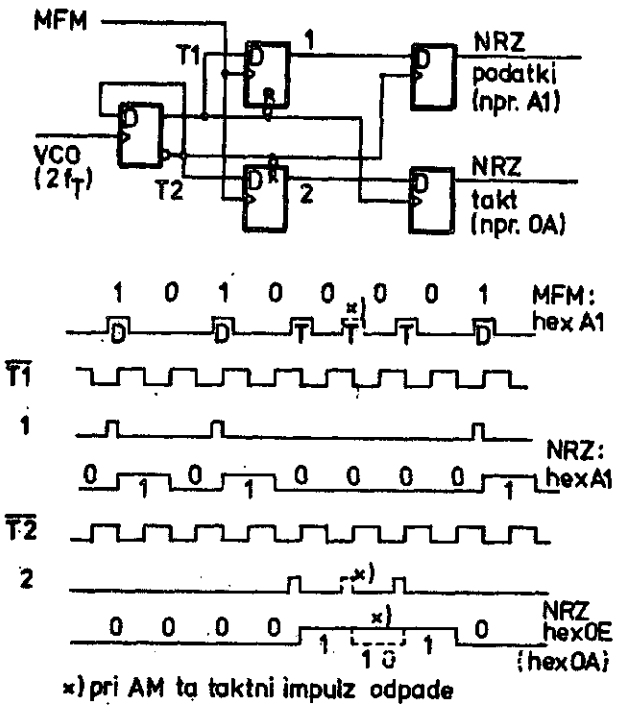
Pretvorbo v paralelne zloge serijsko nanizanih podatkov, kot prihajajo bit za bitom z diska, opravi posebni

- WD1100-01 serijski paralelni pretvornik. V nasprotni smeri, pri zapisovanju na disk, pa

- WD1100-05, paralelni/serijski pretvornik, pretvarja zloge iz računalnika v časovno zaporedno nanizane bite. Najbolj specifična naloga pri pisanju na disk pa pripada

- WD1100-02 MFM generatorju. Ta v posebnem registru pregleda serijsko vpisane podatke in ugotavlja, kdaj mora glede na zahteve MFM modulacije vriniti v podatkovno verigo še dodatni sinhronizacijski taktni impulz. Njegova naloga pa je tudi, da pri generiranju začetne označbe identifikacijskega oz. podatkovnega polja (AM) izpusti tak dodatni sinhronizacijski impulz. Ta "napaka" v MFM modulaciji je dogovorjeni znak, da AM detektor loči označevalni Hex A1 od podatkovnega Hex A1. Generiranje AM gre preko posebne "skip" logike tega vezja. Tu se generirata tudi signala

Slika 6.7. Separacije podatkov in taktnih impulzov iz vhodnega MFM-signala.



"early" in "late" za časovni pomik posameznih impulzov MFM verige. Pri pisanju na notranje sledi diskov so ob zmanjšanju zapisovalnega toka potrebne zakasnitve za 12 ns nasproti nominalni vrednosti.

6.5.2. Separacija podatkov

Problem posebne vrste je ločitev podatkovnih impulzov pri branju z diska sprejetih MFM signalov in vrinjenih sinhronizacijskih taktnih impulzov. Rešimo ga lahko, če nam je na razpolago poseben ločitveni signal. Tega dobimo iz oscilatorja, ki ga je treba preko fazne regulacijske zanke (PLL) sinhronizirati s sprejetim MFM signalom. Ker pa je ta signal zelo "neenakomeren" (presledki med impulzi so različno dolgi), mora oscilator nihati na dvojni frekvenci prenosa podatkov. Posebno sinhronizacijsko vezje mora časovno primerjavo dovoliti le v trenutku, ko v MFM signalu nastopa kak impulz. Tedaj se proporcionalno časovnemu premiku MFM impulzov nasproti oscilatorjevemu izhodu generira napetost, ki potisne oscilatorjevo frekvenco navzgor ali navzdol, dokler impulzni verigi ne sovpadata.

Pri pisanju na disk tega oscilatorja ne sinhroniziramo z MFM signalom, saj tega ni, temveč s posebnim "write clock" signalom iz vinčestrske diskovne enote. Tako je zagotovljeno, da ostajajo majhne deviacije vsled temperature in drugih zunanjih vplivov na diskovne enote brez posledic.

6.5.3. Programski del vmesnika

Vsi ukazi za izvajanje programa pri branju z diska in pisanju nanj, pri nastavljanju glave in pri formatiranju stez so združeni v hitrih PROMih. Tam se v odvisnosti od ukaza iz računalniškega procesorja in od tega, ali gre za identifikacijsko ali za podatkovno polje, za branje ali za pisanje, za formatiranje, pozicioniranje in podobno, generirajo krmilni signali za zgoraj opisana pomožna vezja in za ostalo vezje. Ta drugi del so razni registri, kot so statusni register pa registri za številko steze, glave diskovne enote ali pa ukazni register, podatkovni register itd. V te registre je možno podatke zapisovati, jih spreminjati in brati.

6.5.4. Podatkovni del vmesnika

Zaradi hitrega pretoka podatkov na disk ali z njega in za nemoten, hiter prenos teh iz pomnilnika ali vanj, je na vmesniku predviden poseben podatkovni register z obsegom enega sektorja. Obseg je nastavljiv in lahko znaša do 1K zlogov. Prenos teh podatkov iz pomnilnika oz. vanj pa opravi posebni DMA krmilnik.

6.5.5. Uporabnost vmesnika

Ker je mnogo funkcij skupnih, ima vmesnik tudi elektroniko za krmiljenje ene do štirih enot z gibkimi (floppy) diski. Tako omogoča isti vmesnik delo z eno do štirimi hitrimi in zmogljivimi, stalno instaliranimi vinčestrskimi diskovnimi enotami, istočasno pa tudi arhiviranje in izmenjavo podatkov preko prenosnih gibkih diskov.

7. Ceneni CRT krmilnik

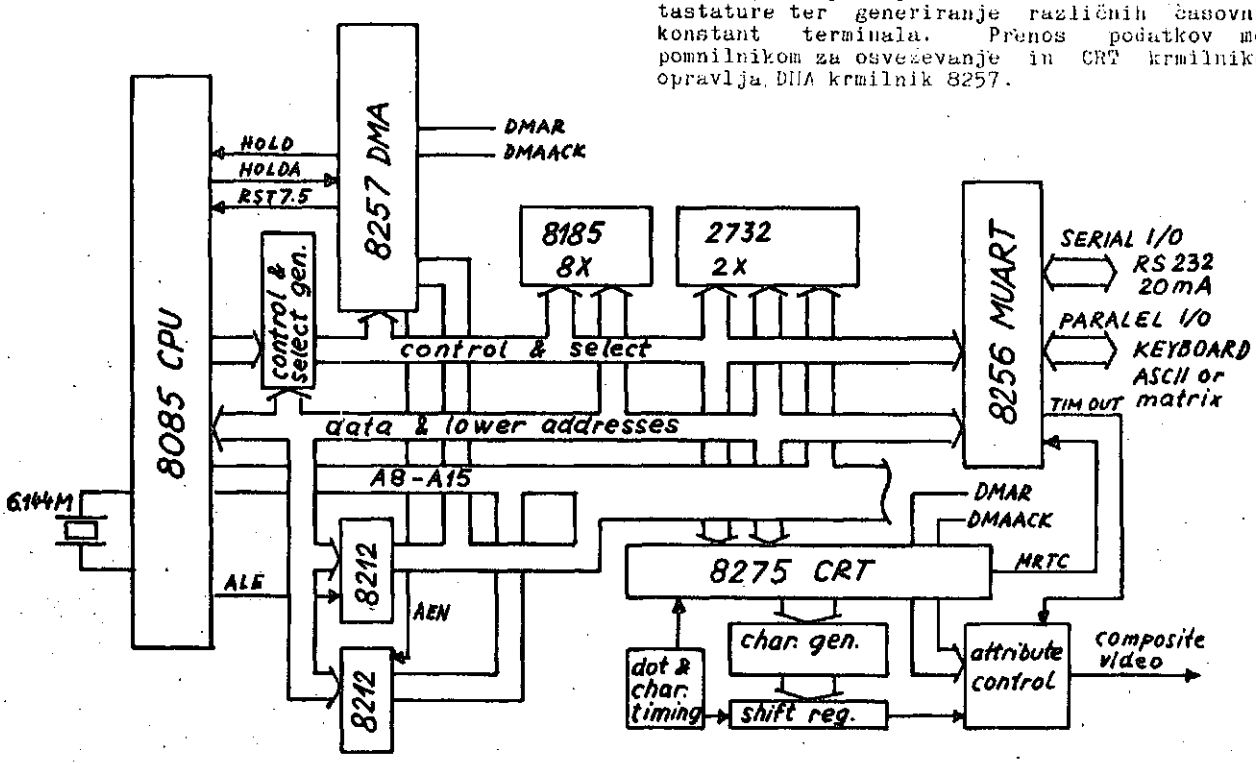
Kot je pokazano na bločni shemi 7.1, je srce CRT krmilnika procesor 8085A, t.j. 8-bitni mikroprocesor s sistemskim taktom 3 MHz. Procesor razpolaga z 8 x 8185 (1K x 8) statičnimi pomnilnimi vezji ter z dvema 2732 (2716) EPROMoma za krmilno programsko opremo terminala. Večfunkcijski univerzalni asinhroni sprejemnik/oddajnik 8256 omogoča asinhrono serijsko komuniciranje s programsko nastavljivo hitrostjo, ima možnost priključitve matrične ali ASCII tastature ter pet 8-bitnih časovnikov/števnikov, ki jih uporabljamo za generiranje različnih časovnih konstant terminala. Prenos podatkov med pomnilnikom za osveževanje in CRT vmesnikom opravlja DMA krmilnik 8257.

CRT vmesnik vsebuje programirljivi CRT krmilnik 8275. Zunanji generator skrbi za časovne poteke točk in znakov. Eden izmed časovnikov v 8256 skrbi za ustrezno časovno obliko horizontalnega pomika nazaj. EPROM 2732 (2716) je uporabljen kot generator znakov ter pomični register za pretvorbo paralelnih podatkov iz generatorja znakov v vlak bitov za iluminacijo točk na zaslonu. EPROM 2732 rabi kot generator znakov in omogoča generiranje posebnih simbolov na zaslonu, npr. za procesiranje tekstov ali v aplikacijah kontrole industrijskih procesov.

7.1. Arhitektura krmilnika

Materialna oprema krmilnika je sestavljena iz treh segmentov: procesor s svojo okolico, serijski in paralelni V/I ter CRT vmesnik.

Procesorski segment je sestavljen iz 8085 procesorja, 8 statičnih RAMov 8185, dveh 2732 ali 2716 EPROMov za kontrolni program ter 8257 DMA krmilnika. Statični pomnilniki so vezani direktno na multipleksirano vodilo procesorja, medtem ko so EPROMi vezani na demultipleksirano vodilo z uporabo vezja 8212.



Slika 7.1. Srce CRT krmilnika je procesor 8085A s sistemskim taktom 3 MHz. Procesor razpolaga z 8 x 8185 statičnimi pomnilnimi vezji ter z dvema 2732 EPROMoma za krmilno programsko opremo. Večfunkcijski asinhroni sprejemnik/oddajnik 8256 omogoča asinhrono serijsko komuniciranje s programsko nastavljivo hitrostjo, priključitev ASCII ali matrične tastature ter generiranje različnih časovnih konstant terminala. Prenos podatkov med pomnilnikom za osveževanje in CRT krmilnikom opravlja DMA krmilnik 8257.

Eden izmed časovnikov vezja 8256 je uporabljen za generiranje horizontalnega pomičnega signala, tako da je možna prilagoditev na zahteve različnih CRT monitorjev.

CRT pomnilnik komunicira z računalniškim sistemom in drugimi CRT enotami preko serijskega vmesnika. Signal RS-232C in 20 mA tokovna zanka sta izpeljana na konektorju. Vse standardne hitrosti so programsko nastavljive do maksimalno 9600 Baudov.

ASCII tastatura se priključi na eno izmed paralelnih vrat vezja 8256. Z matrično tastaturo pa zasedemo dvoje paralelnih vrat na istem vezju.

CRT monitor je krmiljen z 8275 CRT krmilnikom. Ta omogoča programsko nastavitev skoraj vseh parametrov formatu CRT monitorja, kot je pozicija kurzorja, število znakov v vrstici, število vrstic na zaslonu itd.

Vsak znak je oblikovan z matriko točk 5 x 7 znotraj večje matrike 7 x 10 točk. CRT vmesnik omogoča naslednje atribute: utripanje, podčrtavanje, večja osvetlitev ter inverzni izpis (črno na belem). Vsi atributi se nastavljajo programsko z vpisom ukaznih kodov v vrstični vmesnik. Krmilnik ima možnost dela v načinu "konec vrstice - stop DMA", kar omogoča enostavno brisanje posameznih vrstic na zaslonu ali brisanje do konca zaslona.

7.2. Krmilna programska oprema

Delovanje programskega sistema video krmilnika je zasnovano na sistemu prekinitev. Vsakemu izvoru ali ponoru podatkov je dodeljen eden izmed prekinitvenih vhodov procesorja 8085. V sistemu imamo naslednje izvore in ponore podatkov: osveževanje zaslona, tastatura, oddaja na linijo in sprejem iz linije. Ti so razporejeni po prioriteti osveževanje, sprejem, tastatura, oddaja.

Prekinitev za osveževanje se generira enkrat na vsakih 10 linij rasterja zaslona, kar je približno 650 u sekund. Prekinitveni program mora opraviti naslednje akcije: določitev začetnega naslova podatkov za naslednjo vrstico, inicializacija DMA krmilnika in start polnjenja vrstičnega vmesnika CRT krmilnika za naslednjo vrstico.

Prekinitev za sprejem znakov iz linije se aktivira kadarkoli sprejme sprejemnik v vezju 8256 znak iz linije. Prekinitveni program mora opraviti naslednje akcije: vpis sprejetega znaka v spomin za osveževanje na pozicijo, ki jo določa kazalec kurzorja, premik kazalca kurzorja na naslednjo pozicijo in obdelava posebnih znakov.

Prekinitev za sprejem znakov iz tastature se aktivira kadarkoli pritisnemo neko tipko na tastaturi (v primeru ASCII tastature). Prekinitveni program mora postaviti sprejeti znak v spomin za osveževanja na pozicijo kurzorja, premakniti kurzor naprej, postaviti sprejeti znak v vrsto za oddajo na linijo, če terminal deluje na liniji, ter obdelati posebne znake. Prekinitev za oddajo znakov na linijo je omogočena kadarkoli se v vrsto za oddajo na linijo vpiše kak znak, se pa prepreči kadarkoli je ta vrsta prazna. Prekinitveni program odda en znak iz vrste za oddajo, če ostane vrsta prazna.

Posebnost prekinitvenih programov za sprejem z linije in iz tastature je obdelava posebnih znakov. Posebni znaki so na primer carriage return, escape, znaki za krmiljenje kurzorja itd. V primeru sprejema takih znakov morata ti dve rutini opraviti posebne akcije, ki so v skladu z zastavljenimi cilji.

Programski sitem video krmilnika je sestavljen še iz modulov za inicializacijo (power-up, reset) ter iz modula za nastavitev osnovnih značilnosti terminala. Inicializacijski modul ima nalogo nastaviti vse materialne krmilnike v ustrezni režim delovanja, inicializirati spomin za osvezevanje ter startati celotni sistem. Modul za nastavitev ima nalogo, da v interaktivnem pogovoru z operaterjem izbere ustrezen režim delovanja terminala. Izbirajo se naslednji parametri: lokalno - na liniji, hitrost prenosa, format znaka na liniji, format na zaslonu, izbor abecede (English, YU, Arabic, Russian ...) itn. V vsakem trenutku je možen prehod iz nastavitvenega v normalen režim in obratno z ustrezno sekvenco na tastaturi.

8. Programska oprema

V tem poglavju se navaja pregled programske opreme za sistem Delta 323/II. Večina programov se izvaja v okviru CP/II operacijskega sistema, nekaj pa je samostojnih. Nekatere programe je treba še dopolniti za konkretnega naročnika.

Bančništvo
- podatki o strankah
- posojila
- varčevanje
- planiranje investicij
- finančni nadzor
- tekoči računi in čeki
- prenosi
- interni informacijski sistemi

Inženirski in konstruktorski paketi
- grafični programi za arhitekturo
- statični izračuni za gradbeništvo
- napeljave in instalacije
- hidrodinamika
- klimatizacija
- elektro inženirstvo
- hidravlika
- kemijski in farmacevtski procesi
- zemljemerstvo
- risanje načrtov in shem

Matematika in statistika
- aritmetika
- numerična analiza
- linearna analiza
- Fourierjeva analiza
- osnovna statistika
- regresija
- interaktivna statistična obdelava
- statistična knjižnica

Operacijski sistemi
- sistemi za enega uporabnika
- sistemi za več uporabnikov
- sistemi z dodeljevanjem časa
- sistemi realnega časa

Plače
- plačilna lista in izračun plače
- personalna kartoteka
- ovrednotenje osebnih faktorjev

Pomožni sistemski programi
- preformatiranje in konverzija datotek
- diagnostični paketi
- emulatorski programi
- regeneracija pokvarjenih datotek

Prenos podatkov
- komunikacijski monitorji
- daljinsko vodenje
- vodenje terminalov
- distribuirano procesiranje
- prenos datotek
- načrtovanje podatkovnih mrež
- vzdrževanje podatkovnih mrež

Prevajalniki za programske jezike
- ADA prevajalniki
- ADU (Application Development Utilities Templates)
- BASIC prevajalnik in interpreter
- COBOL prevajalnik
- FORTH prevajalnik
- FORTRAN prevajalnik
- LISP prevajalnik
- PASCAL prevajalnik
- PL/1 prevajalnik
- zbirniki in prečni zbirniki

Prodaja in oskrba
- prodajna mreža
- nabava
- obdelava naročil
- obračunavanje
- skladiščenje
- analiza trišča

Programirni pripomočki
- programi za razvoj programov
- generatorji diagramov poteka
- predprocesorji
- generatorji programov
- izdelovanje programske dokumentacije
- simulatorji
- formatiranje zaslona
- testiranje in odpravljanje napak

Programi za razvedrilo
- računalniške igre
- muzika
- šah

Proizvodnja
- potrebe in nabava repromateriala
- podatki o proizvodnem procesu
- obračuni
- organizacija in vrednotenje procesa
- časovni plani in kontrola
- optimiziranje uporabe proizvodnih kapacitet

Računovodstvo in knjigovodstvo
- izplačila
- vplačila
- urejanje računov
- obračunavanje
- glavna knjiga

Sistemi z dialogi in meniji
- generiranje poročil
- pisanje pisem
- programi za postvljanje vprašanj

Šolstvo in izobrazba
- urniki
- podatki o slušateljih
- razrednica
- inštruktorski programi

Upravljanje podatkov in podatkovnih zbirk
- upravljanje podatkov
- upravljanje podatkovnih zbirk
- indeksiranje, registracija in dostop
- podatkovni slovarji
- podatkovni informacijski sistemi
- zaščita
- zajemanje podatkov

Urejanje in obdelava tekstov
- urejevalniki (Editor)
- obdelava besedila
- program za urejanje poštnih naslovov

Zavarovanje
- vrednotenje rizika
- pisanje polic in izdaja računov
- upravni sistem za zavarovalne agente

Zdravstvo
- vodenje ordinacije
- urejanje in pisanje receptov
- kartoteke pacientov
- obračunavanje zdravstvenih uslug
- informacijski sistem za zdravnika
- evidenca o zdravstvenem materialu
- vodenje lekarne
- sestavljanje dietalnih menijev
- program za vodenje operacijskih sob

9. Sklep

V osmih poglavjih tega članka smo prikazali tehnologijo in potencialno nakopičeno inteligenco mikroračunalniških sistemov Delta 323/M1 in 323/M3. Znanje za razvoj teh sistemov se je nakopičilo v okviru raziskovalnih nalog iz prejšnjih let, katerih rezultati so bili objavljeni, kot je razvidno iz seznama slovstva na koncu članka. Izjemo predstavlja le razvoj modula za vinčestrske diske, saj je takšna diskovna enota tehnološko nova in je bila avtorjem razvojno dosegljiva šele v letu 1981. Vinčestrski modul pomeni ob vseh drugih modulih in ob modificirani in razviti programski opremi bistven dosežek na področju najsodobnejše mikroračunalniške tehnologije. S tem modulom se Delta uvršča enakopravno in z izkazano sposobnostjo med proizvajalce mikroračunalniških sistemov v razvitih deželah.

Čeprav se je razvoj sistema začel na HKS Delta 323/M1 s tremi sodelavci pa je proti koncu leta 1981 Deltin mikroračunalniški razvoj narastel že na sedem sodelavcev, ki so kot avtorji navedeni na začetku članka. Izjemna strokovno raziskovalna raven in visoka delovna storilnost sta omogočili dokaj hiter, tehnološko zahteven in zanesljiv razvoj obeh sistemov brez tuje pomoči in velikega razvojnega investiranja v tuja podjetja. Razvijalci Delte smo s tem pokazali tehnološko in organizacijsko zrelost, razvijalno racionalnost in stabilizacijsko smotrnost.

Slovstvo

1. Mostek Z80 Technical Manual, MK 3880 CPU.

2. A. Železnikar: V/I kanali mikroračunalnika, Informatica 2, 13 (1978).

3. Disk Drives Pack Answers for Direct-Access Storage, Electronic Design, Vol. 28, No. 22, October 25, 1980.

4. Controllers Get the Most out of Winchester-to-Hostinterface, Electronic Design, Vol. 29, No. 17, August 20, 1981.

5. Controller for Hard Disks Handles Four Drives at once, Electronic Design, Vol. 28, No. 2, October 25, 1980.

6. A. Železnikar, D. Novak: Perspektive primene mikroračunara, Tehnika 25, 1905 (1976).

7. D. Novak, A. Železnikar: Programiranje mikroračunalnikov v realnem času, Elektrotehniški vestnik 43 162 (1976).

8. A. Železnikar, D. Novak: Programiranje mikroračunalnikov, Avtomatika 17, 75 (1976).

9. A. Železnikar: Psevdokodiranje: sintaksa in uporaba, Elektrotehniški vestnik 43, 213 (1976).

10. A. Železnikar, I. Ozimek, M. Kovačević, D. Novak: Programiranje mikroračunalnikov s procesorjem Z80, Informatica 1, 5 (1977).

11. M. Kovačević, D. Novak, A. Železnikar: Monitori za mikrosisteme sa procesorom 6800, Informatica 1, 20 (1977).

12. D. Novak, A. Železnikar: Morse Code to ASCII Translator Using a Microcomputer, QST 61, 39 (1977).

13. D. Novak, A. Železnikar, M. Kovačević: Struktura multimikroprocesorskih sistemov in način medprocesorskih komunikacij, Automatika 18, 15 (1977).

14. A. Železnikar, M. Kovačević, D. Novak: Razvoj dinamičnih pomnilnikovza mikroračunalnike, Informatica 1, 9 (1977).

15. D. Čavarkapa, A. Železnikar: Hali asembleri za mikroračunare,Zbornik del JUREMA 1977.

16. M. Kovačević, D. Novak, A. Železnikar: Križni asembler sa makroima, Zbornik del JUREMA 1977.

17. B. Rehberger, D. Novak, A. Železnikar: Večprocesorski sistem sprocesorji F8, Zbornik del Informatica 77, Bled 1977.

18. A. Železnikar: Uporaba časovnikov in števnikov v mikroprocesorskih sistemih s procesorjem Z80, Informatica 2, 25 (1978).

19. D. Novak, M. Kovačević, A. Železnikar: Mikroračunalnik Vita s procesorjem 6800, Informatica 2, 14 (1978).

20. A. Hadži, M. Kovačević, A. Železnikar: Audio kasetofon kot cenena vhodna/izhodna enota za mikroračunalnike, Informatica 2, 51 (1978).

21. A. Železnikar: Vrsta s sporočili za komuniciranje z uporabo mikroračunalnika, Informatica 2, 28 (1978).

22. A. Železnikar: Razširitev mikroračunalniškega sistema, Informatica 2, 15 (1978).

23. A. Železnikar: Mikroračunalniška kriptografija I, Informatica 3, 24 (1979).

24. D. Novak, A. Železnikar: Pretvarač signala kodiranih Morzeovim kodom u signale kodirane ASCII-kodom, Vojnotehnički glasnik 27, 623 (1979).

25. B. Uratnik, M. Rogač, A. Železnikar: Mikroračunalnik Iskradata 1680: moduli in sistemi, Informatica 2, 7 (1979).

26. A. Železnikar: Procesiranje teksto z mikroračunalniki I, Informatica 3, št. 3 (1979), 48-57.

27. A. Železnikar: Dopolnitev procesorja teksta, Informatica 3, št. 4 (1979), 58-60.

28. D. Novak, A. Železnikar: Mikroračunalniška kriptografija II, Informatica 3, št. 3 (1979), 22-27.

29. A. Železnikar: Razvoj računalniških sistemov, Informatica 4, št. 1 (1980), 4-12.

30. A. Železnikar: Jezik PL/I in mikroračunalniki I, Informatica 4 (1980), št. 4, 3-10.

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

32. A. Železnikar: Jezik PL/I in mikroračunalniki II, Informatica 5 (1981), št. 1, 16-27.

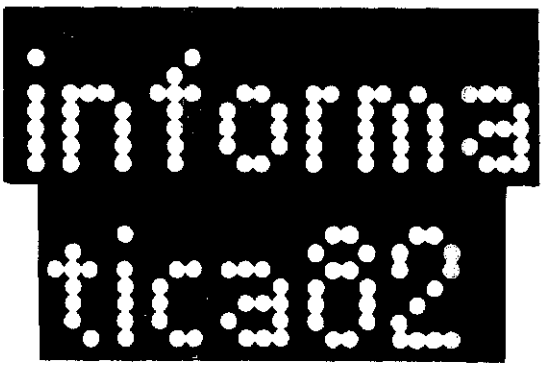
33. A. Železnikar: Jezik PL/I in mikroračunalniki III, Informatica 5 (1981), št. 2, 32-43.

34. A. Železnikar: Tiha revolucija, Informatica 5 (1981), št. 3, 3.

35. A. Železnikar: Jezik PL/I in mikroračunalniki IV, Informatica 5 (1981), št. 3, 15-24.

36. A. Železnikar: Uvod v CP/M I, Informatica 5 (1981), št. 3, 63-76.

37. A. Železnikar: Uvod v CP/M II, Informatica 5 (1981), št. 4, 9-23.



Ljubljana, 10.—14. maja 1982

Preliminarni program Preliminary programme

Simpozij
Symposium

16. jugoslovanski mednarodni simpozij za računalniško tehnologijo in probleme informatike

16th Yugoslav International Symposium on Computer Technology and Problems of Informatics

Seminarji
Seminars

Izbrane teme iz področja računalništva in informatike
Selected Topics on Computers and Informatics
Ljubljana, 10.—14. maja 1982

Organizatorji
Organizers

Slovensko društvo Informatika
Elektrotehniška zveza Slovenije
Gospodarsko razstavišče Ljubljana

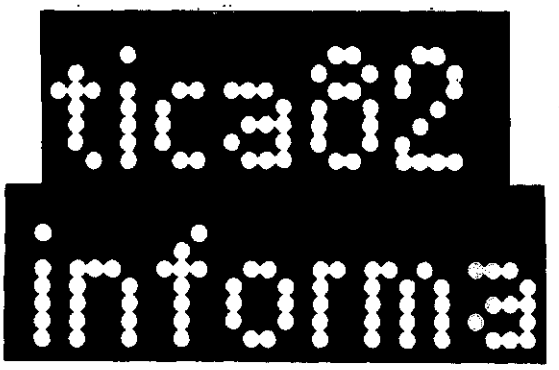
Informatika Slovene Computer Society
Slovene Association of Electrical Engineering
Ljubljana Fair

Programski odbor
Programme Committee

Anton P. Železnikar
Drago Novak
Matija Exel
Ivan Bratko
Cene Bavec
Milan Mekinda
Ciril Bezlaj
Suad Alagić

Organizacijski odbor
Organizing Committee

Marjan Krisper
Dušan Šalehar
Vladislav Rajković
Franc Žerdin
Vasja Herlec
Anton Gorup
Andrej Praprotnik



* CP/M in MP/M sta avtorsko zaščitena izdelka podjetja Digital Research, Pacific Grove, California.

PROJEKTOVANJE IZVRŠNIH SISTEMA MIKRORAČUNARA ZA RAD U REALNOM VREMENU KORIŠTENJEM JEZIKA VISOKOG NIVOA ZA SEKVENCIJALNO PROGRAMIRANJE

ZORAN ŠALČIĆ, GOJO ŠTRKIĆ

ELEKTROTEHNIČKI FAKULTET SARAJEVO,
ELEKTROTEHNIČKI FAKULTET BANJALUKA

UDK: 681.3.014:519.682

U ovome radu opisano je projektovanje izvršnih sistema mikroračunara za rad u realnom vremenu korištenjem jezika visokog nivoa za sekvencijalno programiranje. Pored kratkog osvrta na programiranje za rad u realnom vremenu, data je funkcionalna organizacija izvršnih sistema mikroračunara za rad u realnom vremenu, kao i primjer jezgra jednog takvog sistema projektovanog korištenjem jezika visokog nivoa za sekvencijalno programiranje.

In this paper the design of microcomputer real-time executive by means of high level sequential programming language is described. Besides the short insight to real-time programming, a functional organization of microcomputer real-time executive is given. Kernel of such an executive designed by means of high level sequential programming language is also presented.

1. UVOD

Konvencionalni programi opisuju aktivnosti koje mijenjaju vrijednosti varijabli u diskretnim koracima. Ove aktivnosti se izvode pomoću procesora i one uzimaju konačan iznos vremena. Spomenuto vrijeme ne zavisi od prirode programa. Ono više zavisi od samog primijenjenog procesora. Svi programi su obično napisani tako da izračunati rezultati ne zavise od brzine primijenjenih procesora. Jedan jednostavan način da se dobije vremenska nezavisnost je da se programi pišu tako da opisuju čiste sekvencijalne lance aktivnosti. Svaka aktivnost se inicijalizira tek kad se njen jedinstveni predhodnik završi. Takav program zove se sekvencijalni program.

Uvodjenje pojma vremena u programska razmatranja je čisto tehnička potreba. Glavni razlog za njegovo uvodjenje je taj da izračunavanje može biti ubrzano ako se koristi nekoliko procesora koji rade konkurentno. Programi koji koriste više procesora sastoje se od nekoliko rutina (nazvanih procesima) koje su same po sebi čisto sekvencijalne. Ove rutine se izvode konkurentno a komuniciraju pomoću zajedničkih varijabli i sinhronizacionih signala. Program koji specificira (moguću) konkurentnost zove se multiprogram ili konkurentni program [1].

Uprkos činjenici da je vrijeme bilo glavni razlog za uvodjenje konkurentnog programiranja, zgodno bi bilo što manje proširivati konceptualni kostur sekvencijalnog programiranja, kako bi se izbjegla potreba za pojmom vremena izvodjenja. Konkurentni programi moraju, kadgod je to moguće biti tako napisani da specificiraju izračunate rezultate nezavisno od apsolutnih i relativnih brzina primijenjenih procesora. Pri tome je potrebno uzeti u obzir neophodnu predpostavku da su brzine veće od nule. Pored toga, nezavisnost od brzine primijenjenih procesora dozvoljava još jednu veliku prenost pošto se valjanost programa može utvrditi dedukcijom iz samog teksta programa. Ako se odvojimo od ovog pravila i dozvolimo da valjanost našeg programa zavisi od brzine korištenih procesora, mi ulazimo u polje često zvano programiranje za rad u realnom vremenu. (Iz gore izloženog čini se da bi opisniji termin bio "programiranje zavisno od vremena izvodjenja").

Koji su razlozi za pisanje programa zavisnih od vremena izvodjenja? Ozbiljan razlog je taj što se može desiti da izvjesni procesi koji nisu programabilni, pošto su npr. dio okoline, kao takvi ne mogu da čekaju sinhronizacione signale od ostalih procesa sa kojima kooperiraju. To ima za rezultat da kooperacija sa ovakvim procesima mora da zavisi od brzine procesora.

Da se uočiti da zavisnost programa za rad u realnom vremenu od vremenskih ograničenja čini da je verifikacioni zadatak takvog programa skoro nemoguć ako se vremenska zavisnost ne ograniči na izvjesne izolirane dijelove u nekom standardnom dijelu programa.

Iz prethodnog nameće se zaključak da bi se trebala da koriste slijedeća pravila za programiranje za rad u realnom vremenu:

1. Prvo formulisati čitav program bez ikakvog oslanjanja na vremena izvodjenja i eksplicitno obezbijediti sve potrebne sinhronizacione signale.

2. Za svaki signal koji se ne može obezbijediti pomoću mašine (procesora) koja treba da bude korištena analitički napraviti vremenska ograničenja koja dozvoljavaju odsustvo signala.

3. Provjeriti da li su ova ograničenja zadovoljena u konkrentnoj realizaciji.

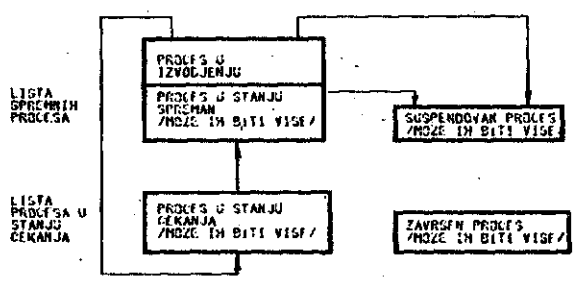
2. OSNOVNE FUNKCIJE IZVRŠNOG SISTEMA MIKRORAČUNARA U PRIMJENAMA REALNOG VREMENA

Sve funkcije izvršnog sistema koje se koriste za rad u realnom vremenu mogu se podijeliti u dvije osnovne grupe. To su primarne i sekundarne funkcije [2] .

2.1. Primarne funkcije izvršnog sistema

Svaka okolina koja je u stanju da podrži više procesa koji rade u realnom vremenu mora imati slijedeće jezgro funkcija:

- inicijalizacija i rasporedjivanje procesa,
- komunikacija medju procesima i sinhronizacija istih,
- opsluživanje vremenskih zahtjeva,
- opsluživanje prekida,
- dinamičko upravljanje procesima.



Sl.1. Stanje procesa

2.1.1. Inicijalizacija i rasporedjivanje procesa

Uz svaki proces u realnom vremenu pridružen je prioritet i stanje koje je poznato izvršnom sistemu u svakom trenutku. To su slijedeća stanja (vidjeti sl.1.):

- spreman,
- čeka,
- susprendovan.

Spreman proces je onaj proces koji nema preostalih zahtjeva prema izvršnom sistemu. Njemu može biti dodijeljen procesor onda kad ne bude više spremnih procesa višeg prioriteta nego što je on.

Proces u stanju čekanja čeka da se desi odredjeni dogadjaj prije nego što bi on nastavio. Dogadjaj može biti vanjski (npr. prekid) ili unutrašnji (npr. poruka od drugog procesa).

Suspendovani proces je onaj proces koji nije ni u stanju čekanja, niti je spreman a niti pak želi neki sistemski resurs.

Izvršni sistem je taj koji vodi računa o stanju procesa i ponovo rasporedjuje aktivnosti procesa kadgod se nešto značajno desi.

Ponovno rasporedjivanje se izvodi u slijedećim slučajevima:

- Proces u izvodjenju čeka poruku ili pak mora da odgodi svoje izvodjenje odredjeno vrijeme.
- Poslije opsluživanja prekida ili poslije vremenskog izbacivanja (timeout) došao na red proces višeg prioriteta.
- Proces u izvodjenju šalje poruku procesu višeg prioriteta i na taj ga način aktivira.

Za sam proces, tok ponovnog rasporedjivanja je nevidljiv jer se stanje njegove "virtualne mašine" restaurira svaki put kad on nastavi izvodjenje.

Većina postojećih izvršnih sistema (RMX/80 [3], REX/80 [4], RTM8 [5]) održavaju listu spremnih procesa poredanih po prioritetu. Proces u izvodjenju je onaj proces koji je uzet sa vrha liste spremnih procesa. Uobičajeno je da u izvršnom sistemu egzistira jedan lažni proces koji se stavlja na dno liste spremnih procesa. Ovaj proces se izvodi kadgod svi korisnički procesi nešto čekaju (što je vrlo čest slučaj u dobro izvedenom sistemu).

2.1.2. Komunikacija i sinhronizacija

Procesi komuniciraju pomoću poruka. Mehanizme za upravljanje i prenos poruka obezbjedjuje izvršni sistem. Ovaj pristup je dosta dobar ali zahtijeva izvjesnu disciplinu jer kad se poruka pošalje njen sadržaj ne smije biti modificiran sve dok se ne dobije potvrda o prijemu iste. Većina izvršnih sistema (RMX, RTM, REX-80) održavaju izvjesne sistemske kanale (channels, exchanges, nodes) gdje poruke i procesi čekaju po principu FIFO. Dakle, proces koji šalje poruku drugom procesu ne upućuje istu direktno nego preko sistemskog kanala.

Proces koji čeka poruku nalazi se na listi čekanja procesa u datom kanalu. Ovaj komunikacioni metod sa ugradjenim baferisanjem vrlo je moćno orudje koje se koristi u sistemima za rad u realnom vremenu.

Poruke koje se izmjenjuju medju procesima mogu se grupisati u dvije klase:

- poruke kod kojih je bitan sadržaj,
- poruke kod kojih je bitno njihovo dešavanje.

Primjer prve klase poruka može biti slučaj poruke koju šalje opslužívač terminala procesu interpreteru komandi.

Druga klasa poruka nalazi primjenu u slučajevima gdje se zahtijeva sinhronizacija izmedju dva procesa, i, što je još važnije, mehanizam medjusobnog isključivanja. Medjusobno isključivanje je potrebno kadgod dva ili više procesa zahtijeva istovremeni pristup na neki nedjeljiv resurs.

2.1.3. Vremenski zahtjevi

Da bi se obezbijedile funkcije kašnjenja u sistemu za rad u realnom vremenu potreban je neki vremenski generator (sat realnog vremena) izveden u hardveru.

Izvršni sistemi zahtijevaju mogućnost vremenskog terminiranja prekida koji se mogu ponavljati sa odredjenim periodom. Izbor perioda izmedju dva prekida mora biti pažljivo odabran. Korisnički procesi zahtijevaju da taj period bude što kraći kako bi mogli što preciznije specificirati proteklo vrijeme. S druge strane, izvršni sistem mora da uzme nešto procesorskog vremena za vrijeme svakog perioda. Ako se sa T_e označi vrijeme potrebno izvršnom sistemu a sa T period ponavljanja prekida tada se iskorištenje procesora može izraziti kao:

$$1 - T_e/T$$

Kako se frekvencija vremenskog generatora (sata realnog vremena) uvećava to se raspoloživa moć procesiranja smanjuje, tj. iskorištenje procesora se smanjuje. Veličine za T_e rijetko se mogu naći u specifikacijama izvršnih sistema. Period izmedju dva prekida je obično izmedju 10 i 100 msec.

Izvršni sistem održava listu čekanja koja je slična listi spremnih procesa. Na toj listi procesi su poredani po relativnom kašnjenju.

Korisni mehanizmi koji su na ovaj način obezbijedjeni su čekanje da prodje odredjeni vremenski period ili pak čekanje na poruku s tim da je u zadnjem slučaju obezbijedjena mogućnost za vremensko izbacivanje (time-out).

2.1.4. Opsluživanje prekida

Prekidi se opslužuju pomoću izvršnog sistema. Izvršni sistem transformiše prekid u U/I poruku koja se prosledjuje do procesa koji istu očekuje. Da bi se povećala propusnost sistema čest je slučaj da poruku prihvata proces višeg prioriteta i prosledjuje je do procesa nižeg prioriteta a on se vraća u stanje čekanja.

Nedostatak pristupa da izvršni sistem opslužuje prekide je postignuta dužina obrade prekida. Izvršni sistem mora da pohrani stanje prekinutog procesa te da rasporedi proces za obradu prekida. Vrijeme odziva može biti više stotina mikrosekundi za razliku od nekoliko desetina mikrosekundi koliko otprilike traje sam potprogram za opsluživanje prekida. Da bi se prevazišao ovaj nedostatak moguće je da se izbjegne mehanizam opsluživanja prekida pomoću izvršnog sistema na taj način što bi se obezbijedile rutine za opsluživanje prekida van kostura izvršnog sistema koje zaobilaze potrebu ulaska u izvršni sistem. Od ovih rutina bi se zahtijevalo da prije nego što se završe, potvrdu o završetku proslijede ka izvršnom sistemu.

Mehanizmi za opsluživanje prekida pomoću izvršnog sistema obično su izvedeni tako da hardver za upravljanje prekidom datog sistema dobro odgovara već utvrdjenoj šemi prioriteta. Na taj način je asinhroni vezni modul integriran sa realnim dogadjajima i formira se dosta koherentna struktura.

2.1.5. Dinamičko upravljanje procesima

U kompleksnom sistemu vrlo je važno da procesi mogu jedni drugima upravljati. Vrlo su česte konfiguracije sistema u kojima procesi mogu imati slijedeće interakcije:

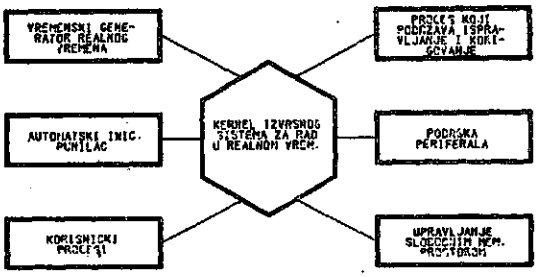
- kreiranje (inicijalizacija i stavljanje na listu spremnih procesa,
- suspendovanje (privremeno deaktiviranje),
- nastavljanje (ponovno pokretanje suspendovanog procesa),
- ubijanje (potpuno deaktiviranje).

Suspendovanje i ponovno pokretanje suspendovanog procesa su mehanizmi koje koristi proces za ispravljanje i korigovanje (debugging) za vrijeme interaktivnog rada.

2.2. Sekundarne funkcije izvršnog sistema

Izvršni sistemi koji se mogu naći na tržištu obično pružaju više mogućnosti nego što je dato u gore opisanom skupu primarnih funkcija. Dodatne funkcije služe da bi podržale svoju razvojnu okolinu ili konačnu primjenu ili pak i jednu i drugu. Ove funkcije, koje ćemo nazvati sekundarnim, implementiraju se kao procesi koje aktiviraju korisnički procesi. Važne sekundarne funkcije su (vidjeti sl.2):

- ispravljanje i korigovanje (debugging)
- upravljanje slobodnim memorijskim prostorom,
- podrška periferala,
- sat realnog vremena,
- automatski inicijalni punilac.



Sl.2. Sekundarne funkcije izvršnog sistema

2.2.1. Proces koji podržava ispravljanje i korigovanje

Zbog kompleksne i paralelne strukture programa koji rade u realnom vremenu, uobičajeni statički metodi za korigovanje i ispravljanje mikroračunarskih programa (izvodjenje u prekidnim tačkama, trasiranje, ispitivanje i modifikacija registara i memorije) pružaju dosta skromne usluge. Umjesto njih zahtijevaju se dinamički metodi koji obezbjedjuju "prozor" u sistem za vrijeme dok je sistem aktivan. Ovi metodi omogućavaju interaktivnu manipulaciju okolinom sistema koji radi u realnom vremenu. Gotovo svi sistemi koji rade u realnom vremenu imaju neki oblik procesa koji podržava ispravljanje i korigovanje, s tim da isti može biti ugradjen u prototipski sistem a izostavljen iz finalnog proizvoda sistema.

Dvije najvažnije funkcije procesa koji podržava ispravljanje i korigovanje su:

- ponovno pokretanje suspendovanog procesa,
- prikazivanje sistemskih struktura podataka.

Prva od ove dvije funkcije može, uz pažljivo planiranje da omogući da sistem za rad u realnom vremenu bude gradjen postepeno. Ako je za sistem predvidjeno da se samo odredjeni procesi aktiviraju za vrijeme inicijalizacione faze tada proces koji podržava ispravljanje i korigovanje može biti korišten da se preostali procesi uključuju u sistem jedan po jedan dok se za to vrijeme posmatra funkcija sistema. Druga funkcija procesa koji podržava ispravljanje i korigovanje pruža mogućnost da stanje sistema bude dostupno korisniku i to u čitljivoj formi. Na taj način, na zahtjev, može se na zgodan način formatizirati i prikazati trenutno stanje na listi čekanja u kanalu ili pak stanje odredjenog procesa kao i sadržaj željene poruke.
Pored spomenute dvije osnovne funkcije procesa koji podržava ispravljanje i korigovanje postoje još neke. To su: dinamičko izvodjenje, pristup na prekidne tačke i posmatranje steka.

Na kraju, mora se spomenuti i jedan nedostatak ovom pristupu ispravljanja i korigovanja. Ispravljanje i korigovanje se implementira kao proces i na taj način remeti stvarnu okolinu za rad u realnom vremenu.

2.2.2. Upravljanje slobodnim memorijskim prostorom

Upravljač slobodnim memorijskim prostorom, koji se implementira kao proces, obezbjedjuje mehanizam pomoću kog ostali procesi mogu alocirati resurs memorijskog prostora. Dakle, umjesto da proces koji šalje poruku posjeduje baferski prostor, on u slučaju potrebe traži memorijski prostor od upravljača slobodnim memorijskim prostorom. Kad sadržaj bafera bude obradjen pomoću odgovarajućeg procesa, baferski prostor se vraća upravljaču slobodnim memorijskim prostorom koji lančano spaja sve vraćene blokove u kontinualni memorijski prostor, kako bi se reducirala fragmentacija.

Potrebno je zapaziti da korištenje upravljača slobodnim memorijskim prostorom ne mijenja statičke osobine korisničke okoline.

2.2.3. Podrška periferala

Da bi se ostvarivale komunikacije sa operaterom većina mikroračunarskih sistema podržava neki oblik U/I-a preko konzole. Obično se javlja potreba da izvršni sistem obezbijedi procese koji bi obavljali terminalske funkcije kao što su editiranje i sl. Iako se može desiti da nisu neophodni za vrijeme eksploatacije datog sistema ovi pomoćni programi (procesi) su valjani za vrijeme razvojne faze (npr. da moguće da se koristi proces koji podržava ispravljanje i korigovanje). Neki raspoloživi mikroprocesorski izvršni sistemi obezbjedjuju funkcije upravljanja odredjenim periferalima. Većina mikroprocesorskih izvršnih sistema podržava komunikaciju sa flopi diskovima. Pored toga često je data podrška za analogni U/I kao i za upravljanje hardverski izvedenim operacijama u pokretnom zarezu.

2.2.4. Sat realnog vremena

Većina izvršnih sistema obezbjedjuje vrlo male mogućnosti sata realnog vremena. Vrijeme dana se obično ažurira pomoću hardvera i rutine za opsluživanje prekida. Vrijeme dana može prema potrebi biti postavljano i očitavano. Medjutim, kod nekih raspoloživih mikroprocesorskih izvršnih sistema procesi mogu biti tako rasporedjivani kako bi se aktivirali ili suspendovali u tačno odredjeno vrijeme.

2.2.5. Automatski inicijalni punilac

U nekim primjenama može biti pogodnije da se program pohrani na disk nego u ROM. U tom slučaju se, kod modifikacije korisničkih programa, radi sa datotekama na disku a ne vrši se reprogramiranje PROM-a. Pored toga moguće je koristiti isti hardver za različite primjene

a sve programe pohranjivati na disk. Automatski inicijalni punilac se koristi da dati program sa diska napuni u ROM. Medjutim, bitno je napomenuti da kernel izvršnog sistema realnog vremena zajedno sa automatskim inicijalnim puniocem i dijelom upravljača diskom mora da bude (permanentno) smješten u ROM-u.

3. PRIMJER IZVEDBE IZVRŠNOG SISTEMA (IS-a) POMOĆU JEZIKA ZA SEKVENCIJALNO PROGRAMIRANJE

Svaki upravljački sistem može biti razbijen u izvjesan broj procesa koji se mogu izvoditi konkurentno. Pri tome treba imati u vidu da procesor može izvoditi samo jedan proces u jednom trenutku a da je brzina procesora ta koja procesima pruža osjećaj paralelnog izvodjenja.

Kad programer/projektant razloži softverske funkcije u posebne sekvencijalne procese koji će medjusobno da kooperiraju,na red dolazi integracija tih procesa. Ta integracija se obavlja pomoću kernela IS-a.

Poznato je da minimalni izvršni sistem za rad u realnom vremenu može imati samo jednu listu čekanja u kojoj spremni procesi čekaju na red da bi bili izvedeni. Svaki proces koji postane spreman biva stavljen na dno te liste. Čim se završi proces koji se trenutno izvodi na red dolazi proces sa vrha liste. Pojava bilo kakvog prekida ima za posljedicu da stanje sistema bude pohranjeno, rutina za obradu prekida izvedena, stanje sistema restaurirano i izvodjenje prekinutog programa nastavljeno. Rutina za obradu prekida može (a ne mora) staviti novi proces na listu. Sistemi projektovani na ovaj način dobro zadovoljavaju većinu jednostavnih upravljačkih sistema. Kod složenijih sistema često se javlja potreba za prioritetima. Kao što je poznato, svi spremni procesi koji čekaju, moraju ranije ili kasnije biti izvedeni. Medjutim, neki procesi moraju biti izvedeni u tačno definisanim vremenskim granicama, dok drugi mogu biti izvodjeni u vremenu kad sistem nema ništa drugo da radi.

Postoji više načina da se procesima dodijele prioriteti. Kako su procesi obično numerisani to im se prioriteti mogu dodijeliti prema uzlaznoj (ili silaznoj) veličini tih njima dodijeljenih brojeva.

Drugi način bi bio da se procesi podijele u izvjestan broj grupa s tim da se svakoj grupi procesa dodijeli po jedan nivo prioriteta. Mi ćemo se odlučiti za ovaj drugi način

Pored prioriteta, postoji još jedna osobina koja može biti potrebna kod sistema za rad u realnom vremenu. To je mogućnost da se procesi "uspavaju" i da ostanu u tom stanju za jedan potreban period realnog vremena. Pretpostavićemo da zbog neke konkretne potrebe rele mora ostati zatvoren za vrijeme jedne sekunde. Većina sistema za rad u realnom vremenu ne bi mogla tolerisati dodjeljivanje procesora tako trivijalnom procesu za tako dug vremenski period. Zbog toga se u sistem mora ugraditi mogućnost za programabilna dinamička kašnjenja.

Ovdje ćemo prezentirati ideje za logičku strukturu procedura koje čine kernel izvršnog sistema.

3.1. Arhitektura kernela IS-a

Kad se odabere struktura podataka za kernel jednog izvršnog sistema time je završen jedan veliki dio posla. Preostali dio posla oko pisanja procedura koje manipulišu tom strukturom nije tako kompleksan.

Sada ćemo predstaviti strukturu našeg kernela:

1. Odabrali smo da postoje dva nivoa prioriteta: visoki i niski. Svi procesi visokog prioriteta, a koji su spremni za izvodjenje, biće rasporedjeni i izvedeni (i to po principu FIFO), prije nego bilo koji proces niskog prioriteta bude rasporedjen.

2. Bilo koji proces može biti prekinut. Medjutim, ni jedna procedura kernela ne može biti prekidana.

3. Ako se prekine proces visokog prioriteta on će biti kompletiran prije nego što bude rasporedjen bilo koji drugi proces. Ako dodje do prekida za vrijeme izvodjenja procesa niskog prioriteta, svi procesi visokog prioriteta će biti rasporedjeni i izvedeni prije nego što se kontrola vrati na proces niskog prioriteta.

4. Odabrali smo da postoje dvije liste čekanja spremnih procesa: jedna za procese visokog prioriteta a jedna za procese niskog prioriteta. Svaka lista ima glavu i rep a procesi su poredani od glave ka repu liste, i to na svakoj od lista. Procesi se rasporedjuju (uzimaju sa liste spremnih procesa) od glave liste, a "aktiviraju" (stavljaju na listu spremnih procesa) od repa liste. U oba slučaja manipulacija ide po principu FIFO.

5. Ovako povezane liste čekanja su odabrane zbog jednostavnosti koju one pružaju prilikom manipulacije. Sve informacije koje se tiču rasporedjivanja i aktiviranja procesa sadržane su u pokazivaču repa liste i pokazivaču glave liste. Procesi koji se nalaze u sredini ovih povezanih lista nemaju nikakvog dodira sa aktiviranjem i rasporedjivanjem. To znači da se procesi izvode prema redosljedu u kom se nalaze u listi čekanja, tj. po principu FIFO.

6. Odabrali smo da postoji pokazivač koji se pridružuje uz svaki proces. Ako je proces na listi spremnih procesa niskog prioriteta njemu pridruženi pokazivač će pokazivati na slijedeći proces na toj listi. Ovi pokazivači omogućuju da liste spremnih procesa budu povezane. Treba dodati da je sadržaj pokazivača zadnjeg procesa na listi jednak nuli.

7. Odabrali smo da postoji po jedan slog (record) statusa pridružen uz svaki proces. Varijabla "rdy" unu-

tar tog sloga indicira da li je dati proces na jednoj od lista spremnih procesa , dok varijabla "dly" indicira da li je proces na listi kašnjenja.

3.2. Uvodno definisanje potrebnih pojmova

Sada ćemo uvesti neke pojmove koji će biti korišteni u narednom dijelu teksta. Pri tome ćemo, zbog olakšice u praćenju programske liste date na kraju, zadržati iste nazive koji su korišteni u programskoj listi.

Varijabla NewProcess predstavlja broj procesa koji treba da bude instaliran na listu spremnih procesa ili na listu kašnjenja. Instaliranje na listu spremnih procesa vrši se pomoću procedure ActivateProcess a instaliranje na listu kašnjenja pomoću procedure ActivateDelay.

Varijabla NewDelay predstavlja vrijednost kašnjenja koje biva instalirano na listu kašnjenja kad se pozove procedura ActivateDelay.

Za proces se kaže da je u stanju izvodjenja (RUNNING) ako se isti izvodi ili se pak izvodi rutina za obradu prekida koja ga je prekinula.

Za proces se kaže da je u stanju PREEMPTED ako je taj proces prekinut da bi bio izvodjen proces višeg prioriteta.

Proces je u stanju spreman (READY) ako se on nalazi na jednoj od lista čekanja spremnih procesa.

Proces je u neizlistanom stanju (IDLE) ako nije izlistan ni u jednoj od lista, što mu indicira njegova varijabla statusa. Treba primjetiti da je manipulacijom statusom moguće potpuno onemogućiti izlistavanje neizlistanog procesa. Ovo je naročito zgodno za vrijeme integracije sistema.

3.3. Dijagram stanja

Dijagram stanja prikazuje relacije izmedju mogućih stanja procesa s tim da se vidi i koje su procedure uključene u promjenu tih stanja.

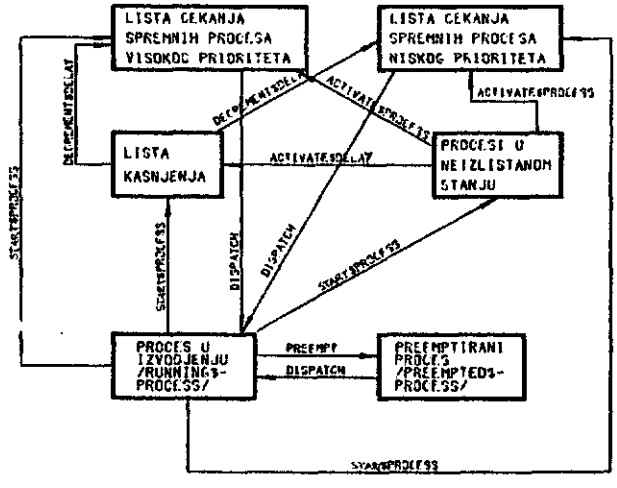
Procedura ActivateProcess postavlja proces NewProcess iz neizlistanog stanja na jednu od lista spremnih procesa i to na rep liste. Sam broj procesa indicira da li će to biti lista spremnih procesa visokog prioriteta ili pak lista spremnih procesa niskog prioriteta. Svi procesi sa brojevima ispod broja kojeg determinira varijabla FirstLowPriorityProcess su predpostavljeni da budu procesi visokog prioriteta. Na isti način, procedura ActivateDelay postavlja proces NewProcess kao i kašnjenje NewDelay na odgovarajuće mjesto u listi kašnjenja.

Tek kad se proces nalazi na listi spremnih procesa (bilo da je to lista visokog ili niskog prioriteta) on je u poziciji da prijedje u stanje RunningProcess. Ovaj prelazak se vrši pomoću procedure DispatchProcess.

Sa liste kašnjenja proces može jedino otići na jednu od lista spremnih procesa. Kad kašnjenje, pridruženo uz dati proces, padne na nulu procedura DecrementDelay poziva proceduru ActivateProcess koja instalira proces NewProcess na odgovarajuću listu spremnih procesa.

Koristeći proceduru StartProcess proces koji je RunningProccess može sam sebe staviti na rep liste čekanja spremnih procesa. On to čini uspostavljajući relaciju NewProcess = RunningProcess. Na sličan način proces u stanju izvodjenja (runningProcess) može postaviti samog sebe na listu kašnjenja pomoću slijedećih relacija: NewProcess=RunningProcess i NewDelay ≠ o. Ako me bi bio zadovoljen uslov NewDelay≠0 proces bi po završetku otišao u neizlistano stanje. Pored gore spomenutih funkcija procedure StartProcess, ona ima i tu funkciju da svaki izvedeni proces postavlja u neizlistano stanje.

Ako je proces niskog prioriteta u stanju runningProcess on odlazi u stanje prekida (PREEMPTED) ako se pojavi proces na listi spremnih procesa visokog prioriteta. To se desi pod uslovom da se tad izvodila rutina za obradu prekida procesa niskog prioriteta i da je iz nje pozvana procedura PreemptProcess. Iz stanja preemptedProcess proces se vraća u stanje runningProcess tek kad se izvedu svi procesi visokog prioriteta. To je moguće jer se procedura DispatchProcess vraća na proceduru PreemptProcess tek kad se izvedu svi procesi visokog prioriteta.



Sl.3. Dijagram stanja procesa

Da bi kernel nesmetano obavljao svoju funkciju potrebno je uvesti i neka ograničenja. Ona su slijedeća:

- Ako se pozove procedura ActivateProcess ili procedura ActivateDelay a dati proces već bude zauzet

(izlistan) onda se to notira u slogu statusa (varijabla status, error) i procedura se završi.

- Da bi proces mogao biti izlistan na jednu od lista, on mora biti u neizlistanom stanju.

- Ako je proces u stanju izvodjenja (runningProcess) on može samog sebe postaviti na listu čekanja spremnih procesa tek onda kad isti bude izveden.

- Nije dozvoljeno da se ponovo izlistava bilo koji izlistani proces (tj. nikad ni jedan proces ne može istovremeno da bude dva puta izlistan). Proces koji pokušava da aktivira (izlista) neki drugi proces koji je već zauzet, mora čekati da se zahtijevani proces izvede, postane neizlistan i tek tada postane raspoloživ za aktiviranje.

3.4. Povezane liste

U našem kernelu koriste se tri povezane liste. Dvije su liste čekanja spremnih procesa a treća je lista kašnjenja.

Svaka povezana lista čekanja sadrži pokazivač glave liste. Pokazivač glave liste sadrži broj prvog procesa u listi. Pokazivač prvog procesa pokazuje na drugi proces u listi, i tako redom. Pokazivač zadnjeg procesa na listi postavljen je na nulu da bi indicirao da je to zadnji proces u listi.

Procesi se stavljaju na dno tj. rep liste čekanja spremnih procesa a uzimaju se sa vrha tj. glave iste liste.

Lista kašnjenja ima glavu liste koju smo nazvali DelayHead. Lista kašnjenja je povezana lista. Medjutim, ona nije povezana lista čekanja pošto se kašnjenja na istu instaliraju po veličini kašnjenja a ne po redoslijedu aktiviranja. Razlog, za ovo će biti objašnjen kasnije.

Kao što je već rečeno, postoje dvije liste spremnih procesa. Jedna od njih služi za procese visokog prioriteta a jedna za procese niskog prioriteta. Pokazivači glave pridruženi uz ove dvije liste su HighPriorityHead za listu visokog prioriteta, a LowPriorityHead za listu niskog prioriteta.

Naravno, struktura kernela može biti proširena do bilo kog nivoa prioriteta, s tim da se za svaki nivo uvede po jedna nova povezana lista čekanja i da se vodi računa o procesima prekinutim procesima višeg prioriteta.

3.5. Kašnjenja

Kernel može u jednom trenutku imati više aktivnih kašnjenja. Medjutim, postoji problem ako on mora da dekrementira svako kašnjenje na svaki takt sata realnog vremena, pošto to uzima dosta vremena. Jedan od načina da se to izbjegne je da se problem premjesti iz rutine DecrementDelay u rutinu ActivateDelay. U tom slučaju, kašnjenja se smjeste u povezanu listu i to po rastućim veličinama tako da u vrijednost svakog kašnjenja upada suma svih prethodnih. Prema tome, dovoljno je dekrementirati samo jedno kašnjenje na svaki takt sata realnog vremena. Ovaj pristup ćemo ilustrovati pomoću slijedećeg primjera. Predpostavićemo da postoje slijedeći uslovi:

Proces 7 ima kašnjenje od 5 msec
Proces 3 ima kašnjenja od 8 msec
Proces 9 ima kašnjenje od 14 msec

Prema tome struktura kašnjenja izgleda ovako:

DelayHead: = 07
Proces (7). pntr: = 03
Proces (3). pntr: = 09
Proces (9). pntr: = 00
Proces (7). delay:= 05 (prvo kašnjenje =5)
Proces (3). delay:= 03 (5+3=8)
Proces (9). delay:= 06 (5+3+6=14)

Dakle, povezana lista je aranžirana tako da su kašnjenja u rastućem nizu a svako kašnjenje je jednako sumi svih prethodnih kašnjenja. Prema tome, dekrementiranjem prvog kašnjenja bivaju dekrementirana i sva ostala kašnjenja.

Potrebno je napomenuti da je nedostatak gornjeg pristupa u tome što se dobija složenija rutina ActivateDelay. Medjutim, pošto se rutina ActivateDelay izvodi rjedje nego rutina DecrementDelay to je ušteda u realnom vremenu značajnija od složenosti rutine.

3.6. Izbjegavanje mrtve petlje

Želeći da naš kernel i dalje ostane jednostavan, a da se izbjegne problem mrtve petlje na strukturu procesa se dodaje jedna posebna matrica. U toj matrici svaki element predstavlja po jedan resurs potreban prilikom izvodjenja datog procesa. Kad procedura StartProcess pozove odredjeni proces na samom početku procesa se utvrdi da li su na raspolaganju potrebni resursi. Ako su resursi na raspolaganju proces se izvodi s tim da se stanje resursa ažurira. Ako pak procesi nisu na raspolaganju proces se vraća na proceduru StartProcess koja stavlja istog na listu kašnjenja ili listu spremnih procesa.

4. ZAKLJUČAK

Jeziku za sekvencijalno programiranje se može zamjeriti to što ne posjeduje odredjene softverske primitive koje bi omogućavale sinhronizaciju i komunikaciju izmedju sekvencijalnih procesa što je posebno bitno kod sistema koji rade u realnom vremenu. Naime, primitive za sinhronizaciju i komunikaciju moraju se projektovati svaki put kad se projektuje sistem i ostavljeno je u

udio programeru da ih projektuje po svojoj zamisli što u većini slučajeva nije možda optimalno i sigurno rješenje. Pored toga tu ne postoji na raspolaganju neko podesno oruđje koje bi olakšavalo testiranje i verifikaciju spomenutih primitiva. Medjutim, prednost višeg programskog jezika za sekvencijalno programiranje u odnosu na nestruktuirane (asemblerske) jezike su brzina razvijanja i preglednost (čitljivost).

DODATAK

Programska lista kernela

```
TYPE  ProcPointer = POINTER TO ProcDesc
TYPE  ProcDesc    = RECORD
                      next: ProcPointer
                      number: INTEGER
                      status: Stat
                      delay:  INTEGER
                      resources: ARRAY
                    END

TYPE  Stat        = RECORD
                      rdy: BOOLEAN
                      dly: BOOLEAN
                      error: BOOLEAN
                    END

TYPE  P           = 1.. MaxNumbOfProc

TYPE  ProcessTable: ARRAY P OF ProcDesc

VAR   HighPriorityHead, LowPriorityHead, DelayHead:
      ProcPointer

CONST MaxNumbOfProc =

(*ActivateProcess
- Ubaci neizlistani proces u odgovarajuću listu spremnih procesa.
- Ako je proces već bio postavljen na neku od lista
  (što indicira njegov status) onda u status postavi
  indikaciju da je pokušano izlistavanje procesa koji
  je već izlistan*).

PROCEDURE ActivateProcess
VAR p: ProcPointer
WITH ProcessTable (newProcess) DO
  IF status.rdy = TRUE THEN (*Process nije izlistan*)
     next: = NIL
     WITH  status DO
           rdy:=FALSE
           Error: = FALSE
     END (* with status*)
       WHILE newProcess < First lowPriorityProcess DO
   IF HighPriorityHead <> NIL THEN
       p: = High PriorityHead
       WHILE p.next <> NIL DO p: = next END
       p.next: = newProcess
   ELSE
       HighPriorityHead: = newProc
   END
   .
   .
   END (* While newProcess*)
   (*Znači da je proces niskog prioriteta*)
   IF LowPriorityHead <> NIL THEN
       p: = LowPriorityHead
          WHILE p.next <> NIL DO p: = next END
          p.next: = newProcess
   ELSE
          LowPriorityHead: = newProcess
   END (* If LowPriorityHead*)
 ELSE (* Znači da je proces izlistan pa to treba notirati u statusu*)
   status.error: = TRUE
 END ActivateProcess

(*ActivateDelay
  - postavi proces koji treba da bude zakašnjen za
    dužinu vremena "delay", na odgovarajuće mjesto u
    listi kašnjenja. (Lista kašnjenja je tako uredjena da dekrementiranjem kašnjenja prvog procesa
    bivaju dekrementirana kašnjenja svih ostalih procesa u listi*).

PROCEDURE  ActivateDelay
VAR  pointerNull, pointerOne: P
     oldDifference, difference: INTEGER
WITH processTable (newProcess), status DO
     IF rdy = TRUE THEN (*Proces nije ni na jednoj
                          listi*)
        newProcess.next: = NIL (*Priprema za ubacivane na listu kašnjenja*)
        difference: = timdelay
        pointerNull: = delayHead
        pointerOne: = NIL
          IF delayHead = NIL THEN (*Lista kašnjenja
                                    bila prazna*)
        WHILE difference > Ø  DO
            oldDifference:. = difference
            difference: = difference - processTable
                          (pointerNull) delay
            pointerOne: = pointerNull
            pointerNull: = processTable (pointerOne)
                          next
            (*NewProcess pokazuje na pointerNull*)
            processTable (newProcess) next: = pointerNull
            processTable(pointerOne) next: = newProcess
```

```
        IF pointerNull = NIL  DO
            WITH processTable (newProcess) DO
                 delay: = UNSIGN (difference)
                 status.rdy=FALSE
                 status.dly: = TRUE
        END
      END (* If pointerNull*)
    END (*While difference*)
    processTable (pointerNull) delay: = UNSIGN (di
                                        fference)
    WITH processTable (newProcess) DO
          delay: UNSIGN (oldDifference)
          status.rdy:=false
          status.dly:= TRUE
      END
    ELSE (*Proces je izlistan na listu kašnjenja*)
          status.error: = TRUE
    END (*If rdy = TRUE)
  END ActivateDelay
```

(* DecrementDelay

- Dekrementiraj kašnjenje procesa koji se nalazi na čelu liste kašnjenja. Time su dekrementirana i kašnjenja ostalih procesa u datoj listi jer je ista tako organizovana. (Proces sa čela liste ima najmanje kašnjenje).
- Kad kašnjenje procesa sa čela liste padne na nulu pozovi proceduru "ActivateProcess" kako bi isti proces bio smješten na odgovarajuću listu spremnih procesa*).

```
PROCEDURE DecrementDelay
VAR  myself: P
     WHILE delayHead <> NIL  DO
          myself: = delayHead
          WITH processTable (myself)  DO
              delay: = delay-1
          WHILE delay = 0  DO
          status.rdy: = TRUE
          status.dly: = false
          delayHead: = delayHead.next
          Activate Process(myself)
       END (*While delay)
    END (* With processTable*)
  END (*While delayHead*)
END (*DescrementDelay*)
```

(*StartProcess:

- poziva se iz procedure DispatchProcess
- starta process "runningProcess" i po njegovom završetku ažurira status istog kao i resursa sistema.
- postavlja vrijednost "runningProcess" na nulu tako da omogući da neki drugi proces može biti rasporedjen*)

```
PROCEDURE  StartProcess
  CASE     runningProcess OF
           null:
                processNull
           one:
                processOne
                .
                .
                .
  END (* Case*)
  WITH processTable (runningProcess) DO
       status.rdy: = TRUE
       status.dly: = FALSE
       resourceStatus AND NOT resources
       IF runningProcess = newProcess THEN
       IF delay <> Ø THEN
            ActivateDelay
       ELSE
            ActivateProcess
       END
       END (*If runningProcess*)
  END (*With processTable*)
       runningProcess: = 0
  END StartProcess
```

(*PreemptProcess:

- Ako se izvodi proces niskog prioriteta a u medjuvremenu se pojavi spremni proces visokog prioriteta onda se prestaje sa izvodjenjem procesa niskog prioriteta i rasporedjuje se proces visokog prioriteta*)

```
PROCEDURE PreemptProcess
WHILE PreemptedProcess= 0  DO
    IF HighPriorityHead <> NIL AND (runningProcess >=
            first LowPriorityProcess) THEN
      preemptedProcess: = runningProcess
      runningProcess: = 0
    END
    WHILE PreemptedProcess<> 0  DO
         DispatchProcess
    END
END (*While preemptedProcess*)
END PreemptProcess
```

(*DispatchProcess:

- Da bi ova procedura izvršila rasporedjivanje procesa, u stanju izvodjenja mora biti lažni proces (idle process).(U našem slučaju to je proces sa identifikatorom jednakim nuli).
- Pomoću procedure "StartProcess" startuje proces koji

je prvi na listi spremnih procesa visokog prioriteta, preemptirani proces ili pak proces koji je prvi na listi spremnih procesa niskog prioriteta*).

```
PROCEDURE  DispatchProcess
VAR  next : ProcPointer
WHILE  runningProcess: = 0  DO
  BEGIN
WHILE HighPriorityHead <> NIL DO
      runningProcess: = HighPriorityHead
      HighPriorityHead: = HighPriorityHead.next
      StartProcess(runningProcess)
  END
  IF  PreemptedProcess <> 0 THEN
      runningProcess: = peemptedProcess
      preemptedProcess: = 0
  ELSE
  IF LowPriorityHead <>NIL THEN
      runningProcess: = LowPriorityHead
      LowPriorityHead:=ProcessTable (runningProcess);
      next
      StartProcess
  END
END DispatchProcess
PROCEDURE  process  0
IF ProcessTable (runningProcess).resources PODSKUP OD
   resourceStatus THEN resourceStates: = resourceStatus
   OR processTable (runningProcess).resources
/* Kod procesa 00 */
ELSE
END (*Kraj procesa 0*)
      .
      .
PROCEDURE process 1
  IF    . . .
        . . .
PROCEDURE StartSystem
(* poziva se kad se želi izvršiti inicijalizacija i
  treba startati sistem*)
  BEGIN
(* - inicijaliziraj sve liste
   - startaj proceduru DispatchProcess*)
  END
END StartSystem
END kernel
```

LITERATURA

1. Wirth. N., "Toward a Discipline of Real-Time Programming", CACM, August 1977, Vol 20, No 8.
2. Frits van der Linden, Ian Wilson, "Real-time executives for microprocessors", Microprocessors and Microsystems, Vol 4, No 6 july/august 1980.
3. "RMX/80 user's guide" No 9800522B Intel Corp.(1979).
4. "REX-80 technical manual", Systems & Software Inc., Downers Grove, Illinois, USA.
5. "RTM 8 real-time executive user's manual" , Advanaced Micro Computers (1979).
6. "RMX/80 real time multitasking executive" Intel Application Note 33 (1979).

U V O D V CP/M* III

ANTON P. ŽELEZNIKAR

SOZD ELEKTOTEHNA, DO DELTA

UDK: 681.3.06 CP/M:181.4

Tretji del uvoda v operacijski sistem CP/M obravnava nadaljne prehodne ukaze, in sicer LOAD, DUMP, SUBMIT, XSUB, SYSGEN, MOVCPM, ASM ter okvirno zgradbo CP/M sistema z moduli CCP, BDOS in BIOS. Izčrpneje je obravnavana struktura modula BIOS ter možnosti razvoja novega in modifikacije starega BIOSa. Nazadnje je nakazana še možnost realizacije mehanizma s t.im. V/I zlogom.

An Introduction to CP/M Operating System III. The third part of the article dealing with CP/M operating system describes further types of transient commands, as LOAD, dUMP, SUBMIT, XSUB, SYSGEN, MOVCPM, ASM and shows the basic structure of CP/M system consisting of CCP, BDOS, and BIOS modules. The structure of BIOS is presented in more detail and simuntaneously, the possibilities of BIOS development and BIOS modification are pointed out. At last, a possible implementation of IOBYTE mechanism is described.

4.10. Oblikovanje izvršljivega programa z LOAD ukazom

LOAD ukaz ima eno samo funkcijo, in sicer vzame zbirko s pripono HEX in jo pretvori v izvršljivo zbirko s pripono COM.

HEX zbirka nastane npr. pri uporabi nekega CP/M zbirnika (npr. za procesor 8080A ali Z80). HEX zbirka je napisana v HEX formatu ter jo je moč preizkusiti (izvršiti) z uporabo DDT programa. HEX zbirko lahko pretvorimo v COM zbirko z uporabo ukazov DDT in SAVE, ukaz LOAD pa omogoča neposredno pretvorbo in postavi začetek izvršljive zbirke na lokacijo 100H. Splošna oblika LOAD ukaza je

LOAD d:ime_zbirke´cr´

Primer uporabe LOAD ukaza je npr.:

```
B>A:LOAD A:IKE.HEX

FIRST ADDRESS 0100
LAST  ADDRESS 0143
BYTES READ    0044
RECORDS WRITTEN 01


B>
```

Pri tem pomeni:

first address	začetni naslov programa
last address	končni naslov programa
bytes read	število zlogov v programu
records written	število zapisov v zbirki

---

* CP/M je avtorsko zaščiten izdelek podjetja Digital Research, Pacific Grove, California.

4.11. Prikaz vsebine zbirke z ukazom DUMP

DUMP ukaz prikaže vsebino zbirke v heksadecimalni obliki in imamo

DUMP d:ime_zbirke.tip´cr´

Ukaz

DUMP d:*.*´cr´

prikaže heksadecimalno prvo zbirko, ki se ujema s *.* parametri. DUMP ukaz deluje podobno kot vgrajeni TYPE ukaz, le da nimamo ASCII predstavitve marveč heksadecimalno.

DUMP ukaz lahko uporabimo za prikaz vsake neASCII zbirke, ki je zapisana binarno (COM zbirka). Imamo tale primer:

```
B>A:DUMP A:IKE.COM

0000 06 00 3A 40 00 FE 0A D2 10 01 21 17 01 4F 09 46
0010 78 32 41 00 C3 00 00 3F 06 5B 4F 66 6D 7D 07 7F
0020 6F 21 42 00 97 06 08 0E FF 87 57 7E D6 30 FE 02
0030 D2 3B 01 82 23 05 C2 29 01 0E 00 21 40 00 77 23
0040 71 C3 05 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

V levem stolpcu so naslovi začetnih zlogov v vrsticah, izpis pa je heksadecimalni.

Z znakom ´CTL s´ ustavimo prikazovanje, z vsakim drugim znakom pa se vrnemo v operacijski sistem (CP/M).

4.12. Paketna obdelava programov z uporabo ukazov SUBMIT in XSUB

CP/M sistem omogoča popolnejšo obdelavo podatkov kot največji računalniški sistemi iz šestdesetih let. V tistih časih je večina sistemov uporabljala paketni način obdelave uporabniških programov. Paket je skupina predmetov in v računalniškem sistemu razumemo pod pojmom paket zaporedje ukazov (programov) in/ali podatkov.

Večina današnjih mikroračunalniških sistemov je interaktivnih, ko imamo komunikacijo med sistemom in uporabnikom. Ta komunikacija poteka tako, da vstavimo v sistem ukaz, nakar sistem nekaj opravi in se vrne na konzolo z določenim sporočilom. V odvisnosti od tega sporočila nadaljuje uporabnik s pošiljanjem novih ukazov v sistem in tako naprej. Interaktivna obdelava postaja vsebolj pravilo kot izjema in dialogni sistemi so za uporabnika najbolj ustrezni.

Večkrat se izkaže, da je pripravnejše poslati v sistem več ukazov, ki naj se potem izvajajo zaporedno. Tak način omogoča, da med izvajanjem zaporedja nismo prisotni ob sistemu in se kompleksne obdelave opravijo avtomatično brez našega poseganja v delovanje sistema. Take naloge lahko rešujemo z uporabo ukazov SUBMIT in XSUB.

4.12.1. Avtomatično izvajanje ukazne vrstice z ukazom SUBMIT

SUBMIT ukaz ureja zaporedno pošiljanje ukazov in njihovo zaporedno izvrševanje brez dodatnega uporabniškega poseganja v postopek. Imamo dve značilni obliki SUBMIT ukaza, in sicer:

SUBMIT ime_zbirke´cr´

Oblikuje se zbirka $$$.SUB, ki vsebuje ukaze navedene (zapisane) v zbirki ime_zbirke.SUB. CP/M sistem izvrši nato ukaze iz te zbirke tako, kot bi jih izvršil z uporabo tastature.

SUBMIT ime_zbirke parametri´cr´

Oblikuje se zbirka $$$.SUB, ki vsebuje ukaze iz zbirke ime_zbirke.SUB. Nekateri deli ukaznih vrstic v zbirki ime_zbirke.SUB se nadomestijo s parametri med oblikovanjem zbirke $$$.SUB. CP/M sistem vzame ukaze iz te zbirke in jih izvrši tako, kot da bi bili poslani s tastature.

Pri uporabi SUBMIT ukaza je najprej potrebno oblikovati zbirko s pripono .SUB, ko uporabimo urejevalnik ED ali kakšen drug urejevalnik. Ta zbirka, ki jo oblikujemo z urejevalnikom, vsebuje ukaze v zaporedju, kot bodo izvršeni. V vsaki vrstici se nahaja po en ukaz.

Pri razvoju programov večkrat uporabljamo prevajalnike, ki prevajajo uporabniške programe. Temu prevajanju večkrat sledi povezovanje prevedenih modulov in potem izvršitev povezanih modulov. Za ta postopek lahko uporabimo submit ukaz, saj takšen postopek zaporednih akcij lahko traja tudi uro in več in ni potrebno, da smo med tem prisotni. Razen tega lahko verižno povežemo tudi SUB zbirke in dobimo še bolj kompleksen paket.

Oglejmo si sedaj še primer. Z urejevalnikom oblikujmo zbirko PO.SUB, in sicer takole:

```
A>TYPE PO.SUB
A:STAT B:*.PLI
ERA B:*.PLI
DIR B:*.PLI

A>
```

Tako oblikovano SUB zbirko pošljimo v izvajanje, ko imamo:

```
A>SUBMIT PO

A>A:STAT B:*.PLI

 RECS  BYTES  EXT ACC
    4     1K    1 R/W B:IKE1.PLI
   36     5K    1 R/W B:IKE2.PLI
   78    10K    1 R/W B:IKE3.PLI
   20     3K    1 R/W B:PLI.PLI
BYTES REMAINING ON B: 218K

A>ERA B:*.PLI
A>DIR B:*.PLI
NO FILE
A>
```

Kot že napisano, se ukazi zbirke tipa SUB nahajajo med izvajanjem te zbirke v posebni zbirki $$$.SUB, ki se oblikuje z ukazom SUBMIT in iz te nove zbirke se opravlja zaporedno izvrševanje posameznih ukazov. S posebnimi krmilnimi znaki lahko ustavljamo izvajanje zbirke $$$.SUB; ti znaki so ´DEL´, ´BS´ oziroma ´CTL _´, ´CTL h´.

Še zanimivejša je uporaba SUBMIT ukaza s parametri, ko imamo splošno

SUBMIT ime_zbirke a b c´cr´

Tu so a, b in c parametri. Ukazi iz zbirke ime_zbirke.SUB se nahajajo v zbirki $$$.SUB, iz katere se izvajajo. Pri tem SUBMIT ukazu lahko uporabljamo nepopolne (parametrične) CP/M ukazne vrstice. SUBMIT ukaz bo manjkajočo informacijo nadomestil s parametri a, b, c itd. Ti parametri so lahko imena zbirk ali drugi podatki. Simboli $1, $2, $3, ... se uporabljajo kot neke vrste globalne spremenljivke, ki se nadomeščajo z dejanskimi parametri.

Vzemimo primer, ko urejujemo zbirko na diskovni enoti A: in želimo imeti njeno kopijo na enoti B:, potem pa želimo preizkus prostega prostora na disketi po vsakem urejevalnem postopku. Brez uporabe SUBMIT ukaza bi morali preko tastature poslati sistemu naslednje zaporedje ukazov (seveda po vsakokratnem končanju posameznih ukaznih vrstic):

```
A> ed ike.doc´cr´
A> pip b:=a:ike.doc[v]´cr´
A> stat b:ike.*´cr´
```

Vzemimo, da je ta metoda zaželena po vsakem urejevalnem postopku, ko urejujemo različne zbirke. Najprej oblikujemo ustrezno SUB zbirko, ki jo poimenujemo z EDIT.SUB, njena vsebina pa je:

```
ed $1.$2´cr´
pip b:=a:$1.$2[v]´cr´
stat b:$1.*´cr´
```

Za uporabo te SUB zbirke imamo tale enostaven ukaz:

```
A> submit edit ike.doc´cr´
```

SUBMIT ukaz oblikuje najprej $$$.SUB iz EDIT.SUB, ko vstavi prvi parameter IKE namesto

$1 povsod tam, kjer se $1 pojavlja. Nato vzame drugi parameter DOC in ga vstavi povsod tam, kjer se pojavlja $2. Po tem postopku sproži SUBMIT ukaz topli zagon sistema in CP/M sistem poišče zbirko $$$.SUB, ki jo začne izvajati.

4.12.2. Avtomatizacija uporabniškega vhoda z ukazom XSUB

V SUB zbirko lahko vstavimo še kaj več kot samo ukaze. Ta zbirka lahko ima npr. tudi odogovre na vprašanja nekega programa ali še kakšne druge spremenljivke. Takšno delovanje dosežemo z uporabo XSUB ukaza.

V tem primeru mora biti XSUB prvi ukaz v SUB zbirki. Ko pri izvajanju SUB zbirke vstopimo v XSUB ukaz, se naloži posebna ukazna množica skupaj s CP/M v pomnilnik in ko program zahteva podatek s konzole, se ta podatek pojavi iz submit zbirke.

Podobno kot v primeru SUBMIT zbirke lahko uporabimo simbole $1, $2, ... , ki se nadomestijo s parametri. Pri tem je XSUB podmnožica glede na SUBMIT. Nikoli ne vtipkamo XSUB s konzole kot odogovr na pripravljenost CP/M sistema. XSUB se lahko pojavi samo v zbirki tipa SUB.

XSUB se najpogosteje uporablja pri programskem in sistemskem razvoju. Imejmo npr. zbirko EDIT1.SUB z naslednjimi vrsticami:

```
XSUB            izvedi XSUB
DIR $1.*        prikaži zbirke
ED $1.$2        uporabi eD za zbirko $1.$2
0A              to je ED ukaz pridružitve
B               pojdi na začetek vmesnika
i               vstavi na začetek
Delta 323/M1    tekst, ki se vstavi
^CTL z^         zaključi vstavljanje
e               končaj z urejevanjem
```

S takšnim oblikovanjem zbirke EDIT1.SUB lahko sedaj zapišemo ukaz, in sicer

```
A  submit edit1 ike.doc^cr^
```

ko imamo parametra ike in doc za $1 in $2.

4.13. Zapis CP/M sistema na disketo z uporabo ukaza SYSGEN

SYSGEN je okrajšava za generiranje sistema. Sistem, ki ga generiramo, je CP/M operacijski sistem. Generiranje pomeni zapis kopije sistema na disketo, tako da se ta disketa lahko uporablja samostojno pri mrzlem, toplem zagonu in pri Obratovanju, ko se ta sistem prepisuje z diskete v hitri pomnilnik računalnika. Pri tem velja omeniti, da sistem ni shranjen v obliki zbirke, kot bomo spoznali kasneje.

Pri sistemskem generiranju uporabimo enega od treh načinov:

1. Uporabimo samo program SYSGEN za kopiranje sistema na novo disketo.

2. Uporabimo MOVCPM in nato še SYSGEN za kopiranje novega sistema na disketo.

3. Uporabimo lahko DDT in nato še SYSGEN za kopiranje modificiranega sistema na disketo.

Modificiranje sistema bomo opisali še kasneje, sedaj pa prikažimo nekatere primere. Za uporabo samo programa SYSGEN imamo tale primer:

```
A>SYSGEN
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)A
SOURCE ON A, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

A>
```

Ta primer opisuje samega sebe (v angleščini), vendar ga še nekoliko pojasnimo. S prvo vrstico se naloži program SYSGEN v hitri pomnilnik in se začne izvajati. Najprej se izpiše verzija sistema (naslednja vrstica) nato pa zahteva za določitev izvirnega diskovnega pogona (diskete, na kateri je zapisan sistem). Če želimo ta korak preskočiti, ker že imamo sistem prekopiran z diskete, ali ko se kratkomalo že nahaja na ustreznih lokacijah hitrega pomnilnika, vtipkamo le ^cr^, nakar preide program v naslednje stanje. Ko je sistem naložen, napredujemo z znakom ^cr^. V nadaljnem koraku moramo določiti namenski diskovni pogon (ki se praviloma razlikuje od Prvotnega). Ko vtipkamo ime enote, se pojavi na zaslonu novo sporočilo o namenskem pogonu in kopiranje na novo disketo se sproži z znakom ^cr^. Po kopiranju se program javlja s sporočilom o zaključeni akciji, toda kopiranje na nove diskete lahko nadaljujemo, če to želimo (neomejeno mnogokrat), sicer pa skočimo nazaj v sistem.

Novi sistem oblikujemo najprej s programom MOVCPM, s katerim se sistem naloži na ustrezno lokacijo v hitrem pomnilniku, hkrati pa se glede na lokacijske zahteve modificirajo ukazi sistema CP/M (skočni in subrutinski ukazi), opravi se ustrezna naslovna premestitev programa v hitrem pomnilniku. Ko je to narejeno, uporabimo podobno kot v prejšnjem primeru program SYSGEN. Imamo tale primer:

```
A>MOVCPM 48 *

CONSTRUCTING 48K CP/M VERS 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM48.COM"
A>SYSGEN
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

A>
```

Program SYSGEN sprašuje po vsaki kopirni akciji za nadaljno kopirno akcijo, tako da lahko vstavljamo nove diskete ter kopiramo novi sistem poljubno mnogokrat.

Odgovorimo še na vprašanje, zakaj je SYSGEN potreben. Prvi dve srezi diskete (steza 0 in steza 1) v CP/M sistemu nista zasedeni z zbirkami. Ta prostor, ki obsega 6656 zlogov, je namenjen za shranitev navezovalnega nalagalnika pri mrzlem zagonu in za CP/M operacijski sistem. Ta dva programa pa se ne shranita v obliki zbirke in se tudi ne pojavita v imeniku, torej nista dostopna na način kot navadne zbirke. Seveda pa je večkrat potrebno, da beremo in zapisujemo tudi na ti dve stezi.

Diskovni stezi 0 in 1 sta vselej izvzeti iz zbirčnega mehanizma, neglede na to, ali sta popisani ali ne. Ni namreč potrebno, da imamo

na vsaki disketi zapisan tudi CP/M sistem. Samo disketa, ki je vstavljena v t.im. sistemski diskovni pogon, potrebuje oba zgoraj opisana programa. Preko tega pogona se namreč izvaja naložitev ob zagonu sistema, tj. mrzli zagon in tudi topli zagon, ki se proži kot izstop iz različnih prehodnih oziroma uporabniških programov, ki so bili generirani s CP/M prevajalniki.

S posebnimi diskovnimi kopirnimi programi lahko kopiramo tudi CP/M sistem oziroma stezi 0 in 1. Imamo pa tudi poseben program SYSGEN, s katerim kopiramo CP/M na nove diskete oziroma ga uporabljamo pri modifikaciji obstoječega CP/M sistema v povezavi z drugimi programi, kot sta npr. DDT in SAVE. Kot bomo spoznali, pomakne program MOVCPM le preslikavo CP/M operacijskega sistema. Pri tem se ne pomaknejo vse t.im. modifikacije tega sistema, npr. podsistem BIOS, ki ga bomo še opisali.

4.14. Prilagoditev CP/M sistema na obseg pomnilnika z ukazom MOVCPM

Proizvajalec CP/M sistema (Digital Research) dobavlja sistemsko disketo za sistem, ki ima le 20k zlogov hitrega pomnilnika. Za večino sistemskih in uporabniških programov pa tak pomnilni prostor ni zadosten in ko dodamo RAM pomnilnik, moramo ustrezno spremeniti tudi CP/M. Program MOVCPM omogoča, da prilagodimo CP/M sistem v intervalu (20, 64), in sicer z natančnostjo 1k zlogov. Tako lahko z njim generiramo sisteme za 20, 21, 22, ... ,63, 64k-zložni pomnilnik. Pri tem imamo tele možnosti:

MOVCPM ´cr´
: Pripravi novo kopijo CP/M sistema, ki uporablja celoten pomnilnik. Krmiljenje se prenese na novi CP/M, vendar se ta ne kopira na disketo.

MOVCPM nn´cr´
: Pripravi novo kopijo CP/M sistema za ´nn´ tisoč zlogov pomnilnika. Krmiljenje se prenese na novi CP/M, vendar se ta ne kopira na disketo.

MOVCPM * *´cr´
: Pripravi novo kopijo CP/M sistema, ki uporablja celoten pomnilnik. Ta kopija bo shranjena s SYSGEN ali SAVE.

MOVCPM nn *´cr´
: Pripravi novo kopijo CP/M sistema za ´nn´ tisoč zlogov pomnilnika. Ta kopija bo shranjena s SYSGEN ali SAVE.

V teh ukazih je ´nn´ dvomestno decimalno celo število, ki ima vrednosti v intervalu (20, 64).

Program MOVCPM pomakne CP/M sistem, in sicer tako, da ga pomakne in takoj nato izvrši (skoči v izvajanje novega CP/M sistema) ali pa ga pomakne in shrani na disketo. Prva možnost je tako le začasna, dokler ne izvršimo toplega ali mrzlega zagona sistema.

Kadar želimo CP/M shraniti na stezi 0 in 1, uporabimo ukaz SYSGEN, če pa ga želimo shraniti kot zbirko za kasnejšo uporabo, imamo ukaz SAVE. V tem primeru se program MOVCPM javi s sporočilom:

```
READY FOR "SYSGEN" OR
"SAVE 32 CPMnn.COM"
```

To sporočilo pove skorajda vse, tako da nismo v zadregi, kako naj nadaljujemo. Podrobna obravnava uporabe MOVCPM programa pa je vsekakor še potrebna, ko spreminjamo del sistema CP/M, ki ga imenujemo BIOS.

4.15. Zbirnik (ASM) za procesor 8080A

Program ASM prevede zbirni program procesorja 8080A v objektni kod, ki ga računalnik lahko izvrši. Zbirnik proizvede tudi listo vrstic z objektnim kodom in zbirniško mnemoniko. Ko smo npr. z urejevalnikom oblikovali program v zbirnem jeziku in ga shranili v zbirki ime_zbirke.ASM, pokličemo zbirnik za prevod te zbirke, ko imamo

A>asm ime_zbirke.abc´cr´

Pripona abc ne pomeni običajne pripone za tip zbirke (v našem primeru asm), marveč nakazuje možnost vstavitve opcij, s katerimi se prevod opravi. Tri črkovna mesta v priponi pomenijo možnost izbire treh opcij, in sicer takole:

1. Ime diskovne enote, na kateri imamo izvirno zbirko, to je zbirko ime_zbirke.asm. To je prva črka v priponi.

2. Ime diskovne enote, ki naj sprejme prevedeno zbirko tipa hex, torej zbirko ime_zbirke.hex. To je druga črka v priponi.

3. Ime diskovne enote, ki naj sprejme zbirniško listo tipa prn, torej zbirko ime_zbirke.prn. To je tretja črka v priponi.

V priponi tako lahko uporabljamo standardna imena diskovnih enot od črke A do črke P. Če torej želimo imeti vse na diskovni enoti C, bomo uporabili pripono CCC.

V priponi pa lahko uporabimo še dve posebni črki (X, Z), ki pa ne označujeta diskovnih enot. Črko Z uporabimo, če zbirke tipa hex ali prn ne želimo. S črko X pa izlistamo zbirniški program tipa prn na konzolo in ne oblikujemo zbirke ime_zbirke.prn na disketi. Če vzamemo

A>asm ike.bbz´cr´

bo zbirnik asm, ki se bo najprej naložil v hitri pomnilnik, prevedel zbirko ike.asm, ki jo bo vzel z diskovne enote b: in na isto enoto bo generiral tudi objektno zbirko ike.hex. Zbirka ike.prn se pri tem ne bo generirala (niti na disketi, niti na zaslonu). Tu smo tako spoznali, da imamo vsakokrat tri tipe zbirk, eno vhodno in dve izhodni in v prejšnjme primeru smo imeli:

ike.asm	izvirna zbirka
ike.hex	prevedena (objektna) zbirka
ike.prn	zbirniška lista (zbirka)

Zbirka tipa asm je izvirna zbirka in oblikovali smo jo z urejevalnikom ED (ali s kakšnim drugim urejevalnikom). Zbirnik je prevajalnik, ki je iz izvirne zbirke generiral objektno zbirko tipa hex. Ta zbirka je zapisana v heksadecimalnem formatu, kjer je vsak zlog izražen z dvema zlogoma v obliki dveh heksadecimalnih številk. Format tega heksadecimalnega zapisa je standarden (Intelov heksadecimalni format). Zbirka tipa prn, ki jo generira prav tako zbirnik, je nekakšna združitev zbirk asm in hex, ko imamo ob prevedeni vrstici zapisano še izvirno (mnemonično) vrstico.

Zbirka tipa hex se uporablja za oblikovanje izvršljivega programa (zbirke) tipa com, zbirka tipa prn pa rabi kot dokumentacija, ki je pomembna pri modifikaciji, vzdrževanju in za razumevanje programa.

Zbirna vrstica ima lahko zasedenih 1 do 5 polj. Pri pisanju zbirnega programa lahko uporabljamo tabulator, ko imamo za začetke posameznih polj

stolpce

1, 9, 17, 25 in 33

Splošni format zbirnega programa je tedaj tale:

št.vr. označ. oper. opand. ;komentar

Tu imamo zapovrstjo številko vrstice, označitev, operator (mnemonika), operand in komentar. Številko vrstice lahko tudi izpustimo. Kaj posamezna polja pomenijo, je zapisano v številnih priročnikih za programiranje procesorja 8080A. Opišimo na kratko zgradbo konstant in izrazov.

Številska konstanta je fiksno število v baznem sistemu (ena od štirih baz). Binarna konstanta je število sestavljeno iz številk 0 in 1 s pripono B, npr. 10100001B. Oktalna konstanta uporablja številke od 0 do 7 in ima pripono O ali Q, npr. 17600 ali 17600Q. Decimalna konstanta je decimalno število brez pripone ali s pripono D, npr. 1239 ali 1239D. Heksadecimalna konstanta je sestavljena iz številk 0 do 9 in črk A do F (velikih ali malih), npr. 0FF4H ali 0ff4h, s pripono H ali h. Torej so pripone lahko tudi male črke.

V konstantah se lahko pojavijo tudi dolarski znaki zaradi lažjega branja števil, ko imamo npr. 0110$1001 ali 01$10$10$01. Zbirnik dolarskih znakov v konstantah ne upošteva.

Naslednja konstanta je nizna in niz znakov je zaprt v narekovaja ´. Nizna konstanta je omejena na 64 znakov in dopustni so samo znaki, ki se lahko natisnejo. Male črke se vobče ne pretvorijo v velike. Tudi sam enojni narekovaj lahko vključimo v niz, tako da vtipkamo ´´. Vrednost niza se izračuna kot zaporedje osembitnih kodov, kjer je bila za osmi bit znaka vselej vstavljena ničla.

Na mestih operandov se lahko pojavljajo izrazi. Izraz je kombinacija konstant, označitev, aritmetičnih in logičnih operatorjev in oklepajev. Med prevajanjem se izraz izračuna in na ustrezno mesto se vstavi njegova vrednost.

Zbirnik lahko opravlja enostavne aritmetične operacije pri izračunu izrazov, in sicer:

A+B je vsota A in B

A-B je razlika med A in B

+B pomeni isto kot B

-B pomeni razliko 0-B

A*B je množenje A z B

A/B je kvocient deljenja A z B

A MOD B je ostanek deljenja A z B

Zbirnik opravlja operacije nad 16-bitnimi vrednostmi brez predznaka in producira 16-bitne rezultate brez predznaka po modulu 2 na 16.

Zbirnik zmere tudi logične operacije in imamo:

NOT B je bitni komplement (bit za bitom) za B

A AND B je bitni logični IN operandov A in B

A OR B je bitni logični ALI operandov A in B

A XOR B je bitni logični ekskluzivni ALI operandov A in B

A SHL B je A pomaknjen v levo za B bitnih pozicij; pri tem se zgubijo višji biti, nižji pa se zamenjajo z 0

A SHR B je A pomaknjen v desno za B bitnih pozicij; nižji biti se zgubijo, višji pa se zamenjajo z 0

Logične operacije se opravijo na 16-bitnih vrednostih brez predznaka in rezultati so 16-bitne vrednosti brez predznaka.

Operatorska prednost pri izračunu izrazov pa je tale:

* / MOD SHL SHR (najvišja)

\- +

NOT

AND

OR XOR (najnižja)

Pri tem se lahko uporabljajo pari oklepajev, ki preprečujejo pravkar opisano operatorko prednost.

Zbirnik ima vrsto posebnih ukazov, ki niso elemeti zbirnega jezika za procesor 8080A. Ti ukazi ali zbirniške direktive vplivajo na prevajalni oziroma zbirniški postopek. Oblika uporabe direktiv je podobna obliki zbirne ukazne vrstice, kot bomo opisali v naslednjih primerih.

Podatkovne oziroma pomnilniške direktive so tele:

DB	Določi vsebino zloga na tekočem naslovu
DW	Določi besedo (sestavljeno iz dveh zlogov) na tekočih dveh naslovih
DS	Določi pomnilničko območje od tekočega naslova naprej (z operandom)

Izrazi v operandnem polju teh direktiv pa so tile: DB stavek (direktiva) ima lahko poljubno število operandov oziroma izrazov, ki so ločeni z vejicami. Izrazi se tu izračunajo kot 8-bitne vrednosti ter se shranijo na zaporednih, tekočih naslovih. Izrazi v DW stavku se izračunajo kot 16-bitne vrednosti in se shranijo na tekočih naslovih. Tudi DW stavek lahko ima več izrazov. Vrednosti pa se shranjujejo v obratnem vrstnem redu: nižji naslov pripada vrednosti (zlogu) z nižjimi številskimi mesti, višji zlig (z višjimi številskimi mesti) pa se shrani na naslednji višji naslov. Vrednost izraza v DS stavku je šestnajstbitna in ta stavek lahko ima en sam operand oziroma izraz. Imamo tele splošne primere:

```
ozn.  DB  izl,iz2,iz3   ;komentar
ozn.  DW  izl,iz2,iz3   ;komentar
ozn.  DS  izl           ;komentar
```

V teh vrsticah je ozn. označitev in jo lahko tudi izpustimo. ´iz´ je okrajšava za izraz, ki je lahko označitev, število, nizna konstanta itd. Komentar seveda ni obvezen in ga dodamo po potrebi.

END stavek pove zbirniku, da je dosegel konec izvirnega programa. Oblika tega stavka je

```
ozn.  END  izraz        ;komentar
```

Ta stavek ni obvezen in če ga imamo, naj bo zadnji v izvirni listi. Izraz v END stavku določa začetek izvajanja programa in če izraz izpustimo, je začetek izvajanja pri naslovu 0000H. Komentarsko polje ni obvezna.

EQU stavek prireedi označitvi vrednost oziroma vrdnost izraza ter ima obliko

```
ozn.  EQU  izraz          ;komentar
```

Izraz je veljavno število, naslov, konstanta ali izraz. Označitev in izraz morata biti prisotna! V izrazu se lahko uporabljajo le spremenljivke, ki so bile že definirane (vnaprejšnja definicija).

IF in ENDIF stavek se uporabljata v zbirnem (prevajalnem) postopku za krmiljenje prevajanja pri določenih pogojih. Splošna oblika teh stavkov je:

```
IF izraz
   ..
   ..
stavki zbirnega jezika
   ..
   ..
ENDIF
```

Zbirnik izračuna izraz v IF stavku in če je vrednost izraza enaka ničli, zbirnik preskoči stavke med rezerviranima besedama IF in ENDIF (ne prevede tega segmenta). Ta segment pa se prevede, če je vrednost izraza od nič različna.

Pri uporabi IF stavka govorimo o pogojnem prevajanju, saj prevajanje zbirnega programa nastopi le pri določenih pogojih. IF stavek je mogoče uporabiti pri programiranju v zbirnem jeziku na več načinov. Imamo npr. tole:

```
IMAMO$TERMINAL    EQU   0FFFH
NIMAMO$TERMINALA EQU   NOT IMAMO$TERMINAL
IF IMAMO$TERMINAL
   ..
   ..
stavki zbirnega jezika, ki se prevedejo,
  če imamo terminal
   ..
   ..
ENDIF
IF NIMAMO$TERMINALA
   ..
   ..
stavki zbirnega jezika, ki se prevedejo,
  če nimamo terminala
   ..
   ..
ENDIF
```

Ta primer kaže, kako lahko spreminjamo program v odvisnosti od tega, ko terminal imamo ali ga nimamo. Pri tem moramo ustrezno spremeniti le vrednost v vrstici IMAMO$TERMINAL (EQU ukaz). NIMAMO$TERMINALA je vselej komplementarno k IMAMO$TERMINAL, za kar poskrbi operator NOT v drugem EQU stavku.

ORG ukaz posreduje zbirniku naslov, pri katerem se bo začelo nalaganje prevedenih stavkov, ki temu ukazu sledijo. Oblika ORG stavka je

```
ozn.  ORG  izraz          ;komentar
```

Izraz je pomnilniški naslov naslednjega programskega ukaza. Označitev in komentar nista obvezna.

SET ukaz priredi vrednost označitvi in dovoljuje, da kasneje v istem programu spremenimo vrednost te označitve. EQU ukaz ne dovoljuje naknadne spremembe vrednosti označitve. Oblika SET ukaza je

```
ozn.  SET  izraz          ;komentar
```

Označitev in izraz sta tu obvezna, komentar pa ni.

Pred in po prevajanju posreduje zbirnik kratka sporočila. Začetno sporočilo je

```
CP/M ASSEMBLER - VER 2.x
```

Temu sporočilu sledi sporočilo o napaki, če se je ta pojavila. Na koncu prevajanja se pojavi trivrstično sporočilo, in sicer:

```
xxxx
yyyH USE FACTOR
END OF ASSEMBLY
```

Tu je xxxx heksadecimalni naslov prve proste lokacije, ki sledi zbirnemu (prevedenemu) programu. Nadakje je yyyH označuje obseg simbolne tabele, ki je bil uporabljen. Zbirnik ima namreč omejen obseg za simbolno tabelo in yyy je heksadecimalno število med 000 in 0FF. Če delimo yyy z 0FFH, dobimo odstotek uporabljenega prostora za simbolno tabelo. Če je yyy enako 80, imamo razmerje 80H/FFH oziroma 128/255, kar pomeni, da je bilo uporabljenega 0,502 prostora za simbolno tabelo. Sporočilo END OF ASSEMBLY pomeni le, da je bil postopek prevajanja končan, kar pa ne pomeni, da je bil končan uspešno.

Na koncu razprave o ASM zbirniku omenimo še sporočila o napakah. Zbirnik daje dve vrsti takih sporočil. Imamo terminalska sporočila o napakah, kjer se navajajo vzroki, zaradi katerih zbirnik ne more končati prevajanja. Druge vrste sporočila se nanašajo na primere, ko zbirnik sicer ne more ustrezno prevesti stavka, vendar lahko kljub napaki prevajanje nadaljuje.

Terminalska sporočila o napakah so tale:

NO SOURCE FILE PRESENT
: Zbirnik ni mogel najti zbirke, je bila navedena. Napaka je lahko nastala pri napačnem vnosu imena zbirke ali imena diskovne enote, lahko pa tudi nimamo zbirke s pripono ASM

NO DIRECTORY SPACE
: Imenik na disku je poln (zaseden) in zbrisati moramo nekaj zbirk (npr. HEX BAK ali PRN. Zasedenost imenika ne pomeni, da je polna disketa. Imenik lahko shrani največ 64 imen.

SOURCE FILE NAME ERROR
: V zbirki, ki bo prevajana, ni dovoljena uporaba znakov '*' in '?'. Z enim ukazom se lahko prevaja le ena zbirka.

SOURCE FILE READ ERROR
: Prevajana zbirka vsebuje informacijo, ki jo zbirnik ne razume. Uporabi TYPE ukaz in ugotovi napako.

OUTPUT FILE WRITE ERROR
: Navedena je bila pisalno zaščitena disketa za PRN in HEX zbirko, ali pa je disketa polna.

CANNOT CLOSE FILE
: CP/M sistem je zbirko našel, toda nanjo ne more zapisovati. To se dogodi, če disketa pisalno zaščitena.

Druga skupina sporočil o napakah se pojavlja med prevajanjem izvirne zbirke. Zbirnik deluje normalno, dokler se ni pojavila napaka. Pri pojavitvi napake, pa se bo pojavila vrstica tele oblike:

```
a  bbbb  cccc  ozn.  op  opand  ;komentar
```

V tej vrstici pomeni 'a' eno od teh napak:

D
: Podatkovna napaka. Vrednost izraza se ne ujema z navedenim podatkovnim območjem; ta vrednost je npr. predolga.

E
: Izrazna napaka. Oblikovan je bil nepravilen izraz, vrdnost je predolga.

L
: Označitvena napaka. Označitev je bila uporabljena nepravilno, npr. pojavila se je na začetku več kot ene vrstice.

N Ta lastnost ni implementirana. Digital Research ima zbirnik MAC, ki sprejema še druge direktive, tiste, ki jih ASM ne more.

O Napaka prestopa. Izraz je preveč zapleten in zbirnik ga ne more prevesti. Izraz naj se razdeli na manjše dele ali naj se reducira število operatorjev.

P Fazna napaka. Označitev je spremenila vrednost med prevajanjem. Če je to res potrebno, naj se uporabi SET ukaz. Napaka nastane tudi zaradi dvojne označitve.

R Registrska napaka. Navedev je bil register izven veljavne mnemonike. Npr. POP B je veljavno, ni pa POP A.

U Nedefiniran simbol. V izrazu je bila uporabljena označitev, ki ji ni bila prirejena vrednost. Imamo npr.
U 09FC 0600 MVI B,ENA
če ENA ni bilo definirano.

V Vrednostna napaka. Pripadajoči operand (v izrazu) ni pravilen. Ta napaka se pojavi zaradi napake pri tipkanju, pri črkovanju oz. če manjka vejica.

Nadalje je ´bbbb´ heksadecimalni naslov stavka, v katerem se je napaka pojavila in ´cccc´ je strojni kod (prevod ukaza). Pri napaki je ta kod lahko v celoti ali delno sestavljen iz ničel.

5. Okvirna zgradba CP/M sistema

To poglavje je namenjeno tistemu, ki se bo ukvarjal z modifikacijo CP/M sistema. Informacija o CP/M bo dana le okvirno, na višjem pomenskem nivoju. Seveda pa bo ta informacija koristila tudi sistemskemu inženirju, da bo lahko pisal bolj učinkovite programe.

5.1. Osnovna zgradba CP/M sistema

Pri mrzlem (začetnem) zagonu računalniškega sistema se CP/M naloži vselej v najbolj zgornji del razpoložljivega pomnilnika in zasede tam 7168 lokacij (zlogov) pomnilnika. Dodatna dva skočna ukaza se naložita na lokaciji 0000H in 0005H. Prvi skok je t.im. vektor za topli (ponoven) zagon, drugi pa je t.im. FDOS vstopni vektor. Uporabnik lahko zasede pomnilnik med lokacijo 100H in dnom CP/M sistema v pomnilniku, ki je določeno prav z FDOS vstopnim vektorjem. Po naložitvi imamo v računalniškem pomnilniku tole:

lokacija (heks)	vsebina (heks)	funkcija
0000	C3	Vektor toplega zagona
0001	03	
0002	xx	
0003	00	V/I zlog (če je implement.)
0004	00	Trenutni diskovni pogon
0005	C3	FDOS vstopni vektor
0006	06	
0007	yy	
0008:005B		Prekinitveno, beležno in rezervirano območje
005C:007C		Zbirčni krmilni blok
007D:007F		Prostor naključnega zapisa
0080:00FF		Običajni diskovni vmesnik
0100:(yy0D-1)		Območje prehodnih programov, kamor se programi nalagajo in izvršujejo
yy00:zzFF		CP/M operacijski sistem: CCP, BDOS, BIOS

´xx´, ´yy´ in ´zz´ so deli heksadecimalnih pomnilniških naslovov, ki so odvisni od obsega specifičnega CP/M sistema.

Drugi način predstavitve dodelitve pomnilnika za sistem CP/M pa je tale:

```
0000:00FF          sistemski parametri
0100:(yy00-1)      območje prehodnih programov
yy0:(yy00+07FF)
                   območje CCP
(yy00+0800):(yy00+15FF)
                   območje za BDOS
(yy00+1600):(yy00+18FF)
                   območje za BIOS
zzFF=yy00+18FF     to je vrh pomnilnika tipa RAM
yy00:yy00+18FF     območje CP/M sistema
(yy00+0800):(yy00+18FF)
                   območje za FDOS
```

Kot smo iz prejšnjih tabel že spoznali, je CP/M sistem sestavljen iz treh modulov, in sicer: CCP (konzolni ukazni procesor), CDOS (osnovni diskovni operacijski sistem) in BIOS (osnovni V/I sistem). Te module bomo na kratko opisali.

5.1.1. Konzolni ukazni procesor CCP

CCP modul interpretira CP/M ukaze, ki jih pošiljamo s konzole (terminala). Ta del CP/M sistema je aktiven le tedaj, ko imamo na zaslonu znak pripravljenosti CP/M sistema za sprejem novega ukaza (A). CCP razpoznava vgrajene in prehodne ukaze ter posebne znake. Ti ukazi in znaki so bili že opisani v prejšnjih poglavjih.

Kadar CCP ne razpozna ukaza, pogleda v diskovni imenik, ali se tam nahaja zbirka tipa COM za ime, ki je bilo vtipkano kot ukaz (ujemanje do 8 znakov). Če to zbirko najde, jo naloži v pomnilnik ter jo začne izvajati kot program. Takšne zbirke imenujemo prehodni programi ali kar prehodni ukazi, saj se z njimi realizira določena izvršitev.

5.1.2. Osnovni diskovni operacijski sistem BDOS

Ta del CP/M sistema je odogovren za celotno aktivnost diskovnih pogonov. Seveda pa BDOS ni dostopen z uporabo neposrednih ukazov s konzole. Vstop v BDOS se opravi preko modula CCP ali preko prehodnega programa, ko se v register C vstavi ustrezna funkcija (vsebina zloga) in se izvrši subrutinski poziv lokacije 0005H.

Med drugimi funkcijami vsebuje BDOS tele funkcije:

- resetiranje diskovnega sistema
- izbira diskovnega pogona
- oblikovanje zbirke
- odprtje zbirke
- zaprtje zbirke
- iskanje po imeniku
- brisanje zbirke
- preimenovanje zbirke
- naključno ali zaporedno branje iz zbirke
- naključno ali zaporedno pisanje v zbirko
- ugotavljanje razpoložljivih diskov
- ugotavljanje izbranega diska
- nastavitev DMA naslova
- postavitev/brisanje zbirčnih indikatorjev

Nekatere BDOS rutine pa izvajajo tudi funkcije konzole, bralnika, luknjalnika in listalnega vhoda in izhoda.

Kadar funkcija potrebuje parametre, se ji ti posredujejo z uporabo registrske dvojice DE. Nekatere funkcije vrnejo informacijo klicajočemu programu in vrnjena informacija se pojavi v registrih mikroprocesorja. Te funkcije s pripadajočimi parametri in vrnjenimi vrednostmi so opisane v CP/M priročniku za programiranje na staneh 12 do 15.

5.1.3. Osnovni vhodni/izhodni sistem BIOS

Večina prihodnjih uporabnikov CP/M sistema ne ve, da je BIOS v izvirni dokumentaciji podjetja Digital Research napisan za določen tip mikroračunalnika podjetja Intel in da se ne prilega zahtevam nekega drugega računalniškega sistema. Zato je to podpoglavje namenjeno tistim, ki obvladajo metodologijo in znanje operacijskih sistemov. Tudi BIOS je sistem različnih subrutin, ki jih kličejo drugi deli operacijskega sistema. Navadno dobavi BIOS proizvajalec mikroračunalniškega sistema, ko ga prilagodi svoji konfiguraciji. Okvirno vsebuje BIOS tele funkcije:

- beri znak s konzole
- izpiši znak na konzolo
- beri znak iz bralnika
- izpiši znak na luknjalnik
- izpiši znak na tiskalnik
- vzemi ali postavi V/I status
- izpiši niz na konzolo
- beri niz s konzole
- ugotovi status konzole

Seveda bi lahko o BIOS še marsikaj zapisali, vendar bomo njegovo zgradbo in pomen podrobneje opisali kdaj drugič v posebnem sestavku.

5.2. Nalagalni postopek pri vključitvi sistema (mrzli zagon)

Pri začetnem zagonu sistema se začne izvajati poseben nalagalnik za naložitev prvih dveh stez diskete v hitri pomnilnik. Ta nalagalnik je shranjen v pomnilniku tipa ROM. Po naložitvi se lahko CP/M sistem še ustrezno pomakne pod sam vrh pomnilnika, iz pomnilnega prostora pa se izključi tudi pomnilnik tipa ROM z začetnim nalagalnikom in drugimi npr. diagnostičnimi programi. Nato se izvajanje programa prenese na operacijski sistem (CP/M).

Izvajanje CP/M sistema se začne v modulu BIOS z inicializacijo, ko se ustrezna informacija naloži v hitri pomnilnik in se na zaslonu generira znak pripravljenosti CP/M sistema za sprejem ukaza s konzole. Sistem je s tem že vstopil v izvajanje modula CCP (konzolni ukazni procesor) in v tem modulu se pričakuje ukaz uporabnika s konzole. CCP zbere znake s konzole v posebnem, ukaznem vmesniku in ko se pojavi znak 'cr' se zbrani ukaz obdela.

Programi, ki so bili prevedeni v COM zbirke, delujejo podobno kot CCP: zbirajo znake s konzole, uporabljajo določene dele BDOS sistema in opravljajo naloge v skladu s svojo semantiko. Programer naj bi pri sestavljanju novih programov upošteval dane možnosti modulov CCP in BDOS. Te možnosti se navadno upoštevajo že z uporabo prevajalnikov za visoke programirne jezike, imamo pa tudi izjeme (npr. pri jeziku BASIC podjetja Microsoft), ko se pri napaki vrne krmiljenje sistemu CP/M in se pri tem ne generira sporočilo o napaki (npr. BDOS ERR ON D:). Nekateri programi ne uporabljajo sistema BDOS, uporabijo pa rutine iz modula BIOS. Takšne izjeme onemogočajo uporabo ukazov XSUB in SUBMIT.

5.3. Uporaba BDOS funkcij v zbirnih programih

V okviru modula DBOS lahko uporabimo 38 različnih funkcij, kot je razvidno iz CP/M priročnika na straneh 13 do 15.

Pri uporabi BDOS funkcije naloži program številko funkcije v register C, naloži ustrezno še druge registre in pokliče lokacijo BDOS (ukaz CALL BDOS). BDOS to funkcijo izvrši in vrne krmiljenje klicajočemu programu. Opišimo primer izpisa niza na konzoli.

Naj bo 09H funkcija za izpis niza, kot je razvidno iz CP/M priročnika. V registru DE se mora nahajati naslov niza in niz mora biti zaključen z znakom $. Pri izpisu niza imamo potem tole:

1. Niz, ki je zaključen z znakom '$', se nahaja nekje v pomnilniku.
2. Program vstavi 09H v register C.
3. Program vstavi naslov niza v register DE.
4. Program pokliče BDOS.
5. BDOS izpiše niz na konzolo in vrne izvajanje klicajočemu programu.

Primer teh korakov v zbirnem programu je tale:

```
BDOS EQU 0005H          ;lokacija BDOSa
NIZ  EQU 09H            ;številka tisk. funkcije
OZN  DB  'To je naloga:$'  ;niz za tiskanje
     MVI C,NIZ          ;številka v registru C
     LXI D,OZN          ;naslov niza v regis. DE
     CALL BDOS          ;izvršitev funkcije
```

5.4. BIOS (osnovni V/I sistem)

BIOS (okrajšava za Basic Input Output System) je osnovni vhodni/izhodni sistem, ki ga uporabnik lahko spreminja. Podjetje Digital Research dobavlja le primer takega sistema in na tem primeru se uporabnik oziroma proizvajalec sistemov lahko nauči, kako napisati lasten (za poseben računalniški sistem) vhodni/izhodni sistem. Za rešitev te naloge pa je potreben razvojni sistem z zbirnikom.

BIOS opravlja nalogo dostopa do različnih perifernih naprav, ko določa pomikanje diskovnih glav, branje in diskovno zapisovanje itd. Slabo napisani BIOS je lahko povzročitelj nezanesljivega delovanja CP/M sistema, njegovih pogostnih napak.

BIOS modul začenja z zaporedjem (tabelo) skočnih ukazov. Imamo tole:

```
ORG  BIOSzacetek        ;začetek BIOSa
JMP  mrzlizagon         ;rutina za mrzli zagon
JMP  toplizagon         ;rutina za topli zagon
JMP  konzolnistatus     ;rutina za kon. status
JMP  konzolnivhod       ;rutina za kon. vhod
JMP  konzolniizhod      ;rutina za kon. izhod
JMP  listalniizhod      ;tiskalni izhod
JMP  luknjanje          ;luknjalni izhod
JMP  branje             ;bralniški vhod
JMP  pomikdomov         ;pomik disk.glave domov
JMP  izbiradiska        ;izbira diskovne enote
JMP  nastavitevsteze    ;nastavitev steze disk.
JMP  nastavitevsektorja
                        ;nastavitev sektorja na
                        ; disketi
JMP  nastavitevnaslova  ;nastavitev pomnilniške-
                        ; ga naslova vmesnika
```

```
JMP  branjesektorja   ;branje disk. sektorja
JMP  zapissektorja    ;zapis sekt. na disk.
JMP  listalnistatus   ;tiskalniški status
JMP  prevodsektorja   ;prevod sektorja
```

Pri vsakem od teh ukazov skočimo v rutino, ki se nahaja v samem BIOSu. Rutine so seveda lahko različnih dolžin in so odvisne od periferije, ki jo uporabljamo in od vmesniških vezij (perifernih krmilnikov). Pri tem pa moramo upoštevati, da je prostor za BIOS v standardnem CP/M sistemu tudi omejen in njegov obseg je le 1900 - 1600 = 300H zlogov ali decimalno le 768 zlogov. Pri taki omejitvi pa smo seveda večkrat prisiljeni, da posežemo v okviru BIOSa še po drugih pripomočkih, npr. tako, da uvedemo BIOS banko hitrega pomnilnika.

5.4.1. Možnosti zgraditve oziroma modifikacije BIOSa

Na CP/M disketi dobimo vzorec določenega BIOSa, ki pa je za naše konkretne zahteve lahko v celoti neuporaben. Iz tega BIOSa sicer razvidimo njegovo zgradbo in dobimo napotke, kako graditi samostojno, vendar moramo celoten modul napisati nanovo, s svojimi specifičnimi subrutinami in njihovimi povezavami. Pri oblikovanju novega BIOSa ali njegovi modifikaciji imamo več korakov in so tile:

1 Na sveže formatirano disketo kopiramo priloženi BIOS, urejevalnik, ASM.COM, DDT.COM, SYSGEN in MOVCPM. SYSGEN uporabimo za kopiranje nemodificiranega CP/M na disketo.

2 Kopijo BIOSa iztiskamo in to kopijo skrbno preučimo. Tako se podrobno seznanimo s strukturo BIOSa, s posameznimi rutinami in tudi s komentarji.

3 Z urejevalnikom začnemo modificirati prvotni BIOS. Pri tem vstavljamo tudi komentarje, ki pojasnjujejo vzroke sprememb v BIOSu. Ti komentarji nam bodo rabili kasneje pri dopolnjevanju in popravkih BIOSa.

4 Dodatki in spremembe ne bodo težavni, če razumemo rešitev naloge, ki je pred nami ter obvladamo delo z zbirnikom oziroma z zbirnim jezikom. V BIOSu imamo tri vrste sprememb:

- vstavitev novega teksta
- brisanje starega teksta
- sprememba obstoječega teksta

Ko vstavimo nov tekst v BIOS, ostane stari tekst nespremenjen. Včasih je priporočljivo, da novovstavljeni tekst posebej označimo, tako da ga ločimo od ostalega teksta. Priporočljivo je tudi, da ne odstranjujemo starega teksta predčasno in da iz njega naredimo komentar, ki ga po potrebi zopet lahko ukinemo. Pri vsaki modifikaciji shranimo staro kopijo in kopije opremimo z datumi, tako da je razviden vrstni red nastajanja novega modula.

5 Preden izstopimo iz urejevalnika, pogledamo ORG ukaz na začetku modula in še posebej označitve, ki so poimenovane z BIOS, BIAS itn. Z modifikacijo BIOSa se večkrat spremeni tudi njegov začetek. rogram napišemo tako, da s spremembo ene same označitve dosežemo vse druge potrebne spremembe pri premaknitvi modula BIOS. Potreba za premaknitev se pokaže pri povečanju ali pomanjšanju pomnilnega prostora. BIOS v delujočem sistemu lahko lokaliziramo z uporabo DDT ukaza, ko imamo:

```
A>DDT'cr'
DDT Vers 2.2
-L0,2'cr'
  0000  JMP  FA03
-'CTL c'
A>
```

V tem primeru smo naročili DDTju, da prikaže ukaz, shranjen na naslovih 0000H do 0002H. Na teh naslovih se nahaja ukaz za skok na topli zagon CP/M sistema. Vsak standardni BIOS začenja s tabelo 17 skokov in topli zagon je v tej tabeli na drugem mestu. Začetek sistema BIOS je tako na lokaciji FA00H, kjer imamo skok v rutino za hladni zagon.

6 Prevedi novi BIOS z zbirnikom, pri čemer lahko uporabimo tako zbirnik za procesor 8080A ali za Z80. V drugem primeru je naš mikroprocesor v računalniku Z80 in ta procesor bo sprejemal ukaze za Z80, ki so dodani k ukazom za procesor 8080A. Seveda pa je ukazna mnemonika za procesor Z80 različna od one za procesor 8080A.

Zbirnik bo sporočil tudi vse napake, ki smo jih pri pisanju novega BIOSa naredili. Ko bomo te napake popravili, bo naš prevedeni BIOS pripravljen za nadaljni postopek, ko ga moramo združiti z BDOS.

7 Naložimo in izvršimo DDT (vtipkamo DDT'cr'). DDT uporabimo za naložitev zbirke BIOS.HEX, ki smo jo dobili z zbirnikom iz izbirke BIOS.ASM. To operacijo izvedemo enostavno v okviru DDT, in sicer takole:

```
-Ibios.hex'cr'
-Rxxxx'cr'
-
```

kjer je 'xxxx' ustrezni naslov pomaknitve. Pri tem velja tole: module celotnega sistema CP/M moramo pravilno stakniti in oblikovati zbirko CPMnn.COM, nad katero bomo po njeni naložitvi uporabili SYSGEN. Ta stik se nanaša na tri module, ki jih ločeno dobimo. Z MOVCPM pridobimo v enem modulu že pravilno premeščena CCP in BDOS. Pred CCP moramo pritakniti inicialni del INITnn, ki vsebuje navadno začetno sporočilo, kopiranje s prvih dveh stez dobljenega koda (v primeru banke) in skok v izvajanje prekopiranega dela. Za BDOS pa moramo pritakniti BIOSnn. Ko je stikanje gotovo, rešimo tako dobljeno novo zbirko DOSnn.COM na disketo z ukazom

```
SAVE 34 DOSnn.COM'cr'
```

Tu je 'nn' pri vseh modulih v celoštevilskem intervalu (20,64) in označuje obseg razpoložljivega pomnilnika. Imamo tole tabelo:

nn	pomn	20k	48k	64k	obseg mod
INIT	900	3800	A380	E380	80
CCP	980	3400	A400	E400	800
BDOS	1180	3C00	AC00	EC00	E00
BIOS	1F80	4A00	BA00	FA00	380
END	22FF	4D7F	BD7F	FD7F	-

V tej tabeli je pomn absolutna pomnilniška lokacija, na katero mora biti modul neglede na 'nn' premeščen. Da dobimo ustrezen odmik za ukaz Rxxxx, moramo od pomn odštevati ustrezen naslov pri 20k, 48k ali 64k, v odvisnosti od primera, ki ga imamo. Tako dobimo za INIT64 odmik xxxx = 6580, za BIOS64 pa odmik xxxx = 2580, kar pomeni, da izvršimo

```
-Iinit64.hex'cr'
-R6580'cr'
```

in

```
-Ibios64.hex'cr'
-R2580'cr'
```

V zadnjem stolpcu imamo obseg modula in vse vrednosti so heksadecimalne.

8 Z ukazom

```
A>MOVCPM nn *'cr'
```

oblikujemo modul CPMnn (natančneje CCP In BDOS), ki je s tem ukazom že pravilno naložen (kot kaže gornja tabela za stolpec pomn). S pritaknitvijo je DOSnn.COM v celoti določen. Z ukazom SAVE shranimo novi DOSnn na disketo, kot smo opisali.

9 Zbirko DOSnn.COM uporabimo za generiranje novega CP/M sistema. Najprej jo naložimo z uporabo DDT ukaza, potem pa uporabimo še SYSGEN.

5.4.2. Realizacija mehanizma z V/I zlogom

CP/M sistem lahko znatno obagatimo z realizacijo mehanizma, ki upošteva t.im. vhodni/izhodni zlog (IOBYTE). V/I zlog je v pomnilniku rezerviran zlog, ki kaže trenutno prireditev perifernih naprav logičnim kanalom. CP/M ima štiri logične kanale:

```
CON:  konzolni kanal
LST:  listalni kanal
RDR:  bralni kanal
PUN:  luknjalni kanal
```

Če imamo npr. dva tiskalnika in dve konzoli, bo V/I zlog določal, kateri od obeh je trenutno prirejen ustreznemu kanalu.

Nalov V/I zloga ima navadno naslov 0003H. Ta zlog ima štiri ločene dvobitne indikatorje:

```
listanje      bita      7  6
luknjanje     bita      5  4
branje        bita      3  2
konzola       bita      1  0
```

Glede na vrednosti bitnih dvojic, so posamezne periferne naprave prirejene kanalom takole:

```
kanal listanje = LST:
   periferne naprave:
     00 = TTY:  01 = CRT:  10 = LPT:  11 = UL1:

kanal luknjanje = PTP:
   periferne naprave:
     00 = TTY:  01 = PUN:  10 = UP1:  11 = UP2:

kanal branje = PTR:
   periferne naprave:
     00 = TTY:  01 = RDR:  10 = UR1:  11 = UR2:

kanal konzola = CON:
   periferne naprave:
     00 = TTY:  01 = CRT:  10 = BAt:  11 = UC1:
```

Imena naprav so mnemonična (v angleščini). Periferne naprave naslavljata prehodna ukaza

PIP in STAT

Vrednost V/I zloga, ki je enaka 00100100 = 24H, pomeni, da se trenutno prirejene kanalom tele naprave:

```
konzolnemu kanalu CON: naprava TTY:
bralnemu kanalu PTR: naprava RDR:
luknjalnemu kanalu PTP: naprava UP1:
listalnemu kanalu LST: naprava TTY:
```

BIOS rutina vzame vrednost V/I zloga in ugotovi relacijo med logičnim kanalom in periferno napravo. Vsak kanal izlušči prieditev periferne naprave iz V/I zloga in skoči v izvajanje ustreznega perifernega vmesnika. Če ima npr. kanal lahko prirejene štiri različne periferne naprave, potem mora imeti tudi štiri različne periferne programske vmesnike, ki se aktivirajo skladno z vsakokratno prireditvijo periferije temu kanalu. Rutina posameznega kanala je tedaj splošno tale:

1. Vzemi V/I zlog
2. Določi periferni vmesnik
3. Skoči v izvajanje vmesnika

Seveda s temi globalnimi napotki nismo izčrpali možnosti modifikacije in dograditve CP/M sistema. Obstajajo številne izboljšave licenčnega CP/M sistema, ki delujejo do desetkrat in več hitreje, ki imajo dodatne pripomočke in so prijaznejše za uporabnika.

Literatura k delom I, II in III

(1) R.Zaks: The CP/M Handbook with MP/M, Zal Sybex, Berkeley 1980.

(2) J.Fernandez, R.Ashley: Using CP/M, Zal. John Wiley, New York 1980.

(3) T.Hogan: Osborne CP/M User Guide, Zal. Osborne/McGraw-Hill, Berkeley 1981.

(4) B.Brigham: CP/M Summary Guide, Zal. Rainbow Ass., Glastonbury 1980.

ALGORITHMS FOR THE SOLUTION OF THE GENERALIZED EIGENVALUE PROBLEM

Z. BOHTE, J. GRAD

UDK: 521.643.5

INSTITUTE OF MATHEMATICS, PHYSICS AND MECHANICS
JADRANSKA 19, LJUBLJANA, YUGOSLAVIA

In this paper we deal with generalized eigenvalue problem of matrix A(z) the elements of which are polynomials in z. The well known iterative methods of Muller, Newton and Laguerre for finding the zeros of function f(z) = det A(z) are analysed. Decompositions of matrix A(z) and its derivatives are introduced in order to simplify the computations of the values of f(z) and its first and second derivatives. Comparative analysis gives some indicators about the rate of convergence, computer time and accuracy of the computed eigenvalues for each of the method used. The analyzed methods proved generally to be stable, economical and easily applicable. FORTRAN subroutines and an example of main calling programme are added for Muller's and Laguerre's methods which proven to be more efficient than Newton's.

ALGORITMI ZA REŠEVANJE POSPLOŠENEGA PROBLEMA LASTNIH VREDNOSTI. V članku je obravnavano numerično reševanje posplošenega problema lastnih vrednosti za matriko A(z) z elementi, ki so polinomi spremenljivke z. Primerjane so dobro znane iterativne metode Mullerja, Newtona in Laguerra za računanje ničel polinoma f(z) = det A(z). Vrednosti polinoma f(z) in njegovih odvodov so izračunane na osnovi razcepa matrike A(z). Primerjava metod daje vpogled v hitrost konvergence, porabljen računski čas in natančnost izračunanih lastnih vrednosti za vsako metodo posebej. Analizirane metode so vse numerično stabilne, ekonomične in enostavno uporabne. Podprogrami v FORTRAN-u in primer glavnega programa so dodani za Mullerjevo in Laguerrovo metodo, ki sta se izkazali za bolj učinkoviti od Newtonove.

Introduction

In this paper three numerical methods for the solution of the generalized algebraic eigenvalue problem

$$A(z)x = 0 \qquad (1)$$

are described and the programmes of two of them are presented. Matrix A(z) is of the form

$$A(z) = A_0 + A_1 z + \ldots + A_m z^m \qquad (2)$$

where all A_i are real square matrices of order n.

The standard source of this problem is the solution of a system of n linear homogeneous ordinary differential equations of order m.

If the matrix A_m is non-singular then the problem (1) is equivalent to the classical eigenvalue problem

$$\begin{bmatrix} 0 & I & 0 & \ldots & 0 \\ 0 & 0 & I & \ldots & 0 \\ - & - & - & - & - \\ 0 & 0 & 0 & \ldots & I \\ -B_0 & B_1 & B_2 & \ldots & B_{m-1} \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ - \\ y_{m-2} \\ y_{m-1} \end{bmatrix} = z \begin{bmatrix} y_0 \\ y_1 \\ - \\ y_{m-2} \\ y_{m-1} \end{bmatrix} \qquad (3)$$

of order m.n where $B_i = -A_m^{-1} A_i$. A similar reduction is possible if A_0 is non-singular.

Although we have several efficient and stable numerical methods for the solution of the classical eigenvalue problem, for instance, QR algorithm [4], it is not economical to reduce the problem (1) to the standard form (3). Moreover, many practical problems like "flutter" problem in aerodynamics, are such that both A_0 and A_m are singular and the reduction to the standard form causes further difficulties [3].

It is therefore considered as more advantageous to work directly with the matrix (2) in all cases. The most natural approach is to

determine the zeroes of the polynomial

$$f(z) = \det A(z) \qquad (4)$$

using some of the many well known methods.

There are some efficient and stable algorithms for the evaluation of the determinant of the matrix, for instance, Gaussian elimination with pivoting [4]. It has been believed so far that the evaluation of the derivative of the determinant involves much more work than the evaluation of the determinant does, and therefore the interpolation methods which require only function values had been suggested for the solution of the algebraic equation

$$f(z) = 0 \qquad (5)$$

see [3], [4]. The most popular of these methods is Muller's method [4] of successive quadratic interpolation. The prejudice against the methods which require the derivatives of the function is probably based on the fact that the derivative of the determinant is a sum of n determinants. However, there can be found in [1] a formula for the logarithmic derivative of the determinant which together with its derivative enable us to solve (5) by using Newton's or Laguerre's method efficiently.

In this paper algorithms for the solution of the generalized eigenvalue problem (1), based on Muller's, Newton's and Laguerre's methods are described and compared with regard to: (i) the number of operations (multiplications and divisions) involved in iteration step, (ii) the number of iterations, and (iii) the computer time used. The computations were performed on a number of typical examples.

Muller's method

Muller's method for the solution of the equation (5) as described in [4] is the following iterative process. From the three previous consecutive approximations to a root of (5): z_{r-2}, z_{r-1}, z_r, the parabola through the points

$$(z_{r-2}, f(z_{r-2})),\ (z_{r-1}, f(z_{r-1})),\ (z_r, f(z_r))$$

is formed and its zero which is the closest to z_r is accepted as the next approximation, say z_{r+1}, to a root. The asymptotic rate of convergence to a simple root is of order 1.84 and at each iteration step only one new function value is required. It is necessary to use complex arithmetic throughout.

The algorithm is as follows.

For r=2, 3, ... we calculate

$$h_r = z_r - z_{r-1},\quad k_r = h_r/h_{r-1},\quad d_r = 1 + k_r$$

$$g_r = f_{r-2}k_r^2 - f_{r-1}d_r^2 + f_r(k_r + d_r)$$

$$z_{r+1} = z_r - 2f_r h_r d_r/(g_r \pm (g_r^2 - 4f_r d_r k_r(f_{r-2}k_r - f_{r-1}d_r + f_r))^{1/2}) \qquad (6)$$

$$f_{r+1} = f(z_{r+1})$$

The sign in the denominator of (6) is chosen so as to give the correction to z_r the smaller absolute magnitude.

The choice of the first three approximations is left to the user. There are no proofs of the global convergence of the iterative process.

In this case we have to calculate the value of

$$f = f(z) = \det A(z)$$

at each iteration step what can be done most efficiently by triangular decomposition of the matrix A(z) using partial pivoting as follows. For the given value of z we first calculate the elements of the matrix

$$A = A(z) \qquad (7)$$

what requires $m.n^2$ operations. Then we decompose A into the form

$$PA = LU \qquad (8)$$

where L is unit lower triangular matrix, U is upper triangular matrix, and P is the corresponding permutation matrix [4]. This step requires $n^3/3 - n/3$ operations. Finally we compute

$$f = \pm \det U = \pm u_{11}u_{22} \cdots u_{nn}$$

what requires n-1 operations. Together we must perform

$$op_M = m.n^2 + n^3/3 + 2n/3 - 1$$

operations for the calculation of f.

Newton's method

The well known Newton's method for the solution of (5) is the iterative process

$$z_{r+1} = z_r - f(z_r)/f'(z_r), \text{ for } r=1,2,\ldots$$

At each iteration step we evaluate the correction of the current approximation to a root as the reciprocal of the logarithmic derivative of the function. From [1], [2] we have the formula

$$f'(z)/f(z) = \mathrm{tr}(A^{-1}(z)A'(z)) = \mathrm{tr}\ X \qquad (9)$$

where $A'(z)$ denotes the derivative of the matrix $A(z)$ with respect to z.

An efficent algorithm for calculating the expression (9) is as follows.

First we calculate the matrices

$$A = A(z), \quad B = A'(z) \tag{10}$$

which requires $(2m-1)n^2$ operations. Then we decompose A as in (8) obtaining

$$PA = LU \tag{11}$$

which requires $n^3/3 - n/3$ operations, compute the matrices Y and X which satisfy

$$LY = PB \tag{12}$$

$$UX = Y \tag{13}$$

As we need only the trace of X, we can use formulae

$$x_{ik} = (y_{ik} - \sum_{j=i+1}^{n} u_{ij}x_{jk})/u_{ii} \tag{14}$$

for i=n, n-1, ..., 1 and k=1,2, ..., i. These two steps require together $(n^3/2 - n^2/2) + (n^3/6 + n^2/2 + n/3)$ operations. So we can calculate

$$f'(z)/f(z) = \sum_{i=1}^{n} x_{ii} \tag{15}$$

with

$$op_N = (2m - 1)n^2 + n^3 \tag{16}$$

operations.

Although the asymptotic rate of convergence of the Newton's method to a simple root is 2 there seem to be little advantage in using Newton's method against Muller's. There appear to be almost three times as many operations per iteration step in Newton's method than in Muller's.

Laguerre's method

The quite famous Laguerre's method requires apart from the function value also the values of its first two derivatives. The asymptotic rate of convergence to a simple root is 3. Perhaps the most attractive feature from the users point of view is its stability in global convergence for any value of the first approximation.

The iterative process is defined by [4]

$$z_{r+1} = z_r - n/(S_1 \pm ((n-1)(nS_2 - S_1^2))^{1/2})$$

for r=1,2, ..., where

$$S_1 = f'(z_r)/f(z_r)$$

$$S_2 = (f'(z_r)^2 - f(z_r)f''(z_r))/f(z_r)^2$$

The sign in the denominator is chosen so as to give the correction to z_r the smaller absolute magnitude. The form of S_1 which is suitable for numerical evaluation is defined by (9) and (10) - (15), while the formula for S_2 is derived from (9) by differentiation, see [2],

$$S_2 = \mathrm{tr}((A^{-1}(z_r)A'(z_r))^2) - \mathrm{tr}(A^{-1}(z_r)A''(z_r)) \tag{17}$$

The algorithm for evaluation of S_2 demands in addition to the starting operations for S_1, i.e. (10) and (11), the following computations:

We must first calculate the matrix

$$C = A''(z) \tag{18}$$

with additional $(m-2)n^2$ operations. Then we calculate the trace of the matrix $W = A^{-1}C$ by steps

$$LZ = PC \tag{19}$$

$$UW = Z \tag{20}$$

$$\mathrm{tr}\, W = \sum_{i=1}^{n} w_{ii} \tag{21}$$

which require $(n^3/2 - n^2/2) + (n^3/6 + n^2/2+n/3) + (n - 1)$ operations. The first term in (17) is found by calculating first the full matrix X by (12) and (13) with n^3 operations and then the trace of X^2 as

$$\mathrm{tr}(X^2) = \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ik}x_{ki} \tag{22}$$

which requires additional n^2 operations.

At each iteration step we must perform

$$op_L = (3m - 3)n^2 + 2n^3 + n^2$$

operations for the calculation of S_1 and S_2. This number is almost twice as large as (16) for the Newton's method.

Numerical tests

The programmes in FORTRAN programming language were written for all three algorithms and tested on CDC CYBER 72 computer. Single precision complex arithmetic was used with t=48, where t is the number of significant digits in the mantissa of a binary floating-point number. The iterative process was stopped after either the computed correction of the computed approximation to the eigenvalue was less than or equal to ε or if $|u_{ii}| \leq \varepsilon$, for $1 \leq i \leq n$, occured in (8) or (11). The value for ε was chosen as $10.\|A(z)\|_E \cdot 2^{-t}$.

The implicit deflation of f(z) was applied in the programmes in order to prevent the iterative process to converge to one of the previously computed eigenvalues, say $x_1, x_2, \ldots, x_m$. This introduces the following modifications of algorithms:

$$f(z) \quad \text{into} \quad f(z)/\prod_{i=1}^{m}(z-x_i) \tag{23}$$

$$f'(z)/f(z) \quad \text{into} \quad f'(z)/f(z) - \sum_{i=1}^{m}(z-x_i)^{-1}$$

$$S_1 \quad \text{into} \quad S_1 - \sum_{i=1}^{m}(z-x_i)^{-1} \quad \text{and} \tag{24}$$

$$S_2 \quad \text{into} \quad S_2 - \sum_{i=1}^{m}(z-x_i)^{-2} \tag{25}$$

Some typical examples of (1) were computed with multiple eigenvalues either of finite value or in the infinity. The values of the parameters m and n of A(z) were on intervals $1 \le m \le 2$ and $3 \le n \le 12$ respectively. No breakdown of the iterative process occured regardless the algorithm used. All three methods proved to be stable and accurate. It was found that the previous computed eigenvalue and the values round it were the most suitable starting values for computing the next eigenvalue.

In the table

	Muller		Newton		Laguerre	
i	No	T	No	T	No	T
1	55	.57	41	.58	17	.45
2	0		0		0	
3	11	.13	19	.28	11	.30
4	0		0		0	
5	3	.04	4	.06	2	.06
6	0		0		0	
7	16	.17	41	.61	3	.08
8	1	.02	2	.04	2	.06
ALL	86	.93	107	1.57	35	.95

the number of iteration steps (No) and the computer time used (T in sec) are shown for each of the computed eigenvalue z_i, i=1,...,8, of the system

$$(A_o + zA_1 + z^2A_2)x = 0$$

where $A_2 = I$, and

$$A_1 = \begin{bmatrix} 0 & -3 & 0 & -1 \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}, \quad A_o = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

The system has a triple eigenvalue at ± i and a double eigenvalue at 0 [5]. Somehow similar results were obtained with other examples.

Three more examples are

1) $A(z) = A_o + A_1 z + A_2 z^2$

$$A_o = \begin{bmatrix} 1 & -1 & 1 \\ -15 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad A_1 = \begin{bmatrix} -2 & 1 & -1 \\ 3 & 0 & 1 \\ 1 & 0.5 & 0 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 0.25 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

z_i:

1.00000000000
2.08866333896
-0.256555796702 ± 0.896010203022 i
-2.91609433069
11.3405425851

2) $A(z) = A_o + A_1 z$

$$A_o = \begin{bmatrix} -1 & -3 & -3 & -3 & -3 & -3 \\ -3 & -4 & -3.1 & -3.1 & -3.1 & -3.1 \\ -3 & -3.1 & 2.8 & 3.8 & 3.8 & 3.8 \\ -3 & -3.1 & 3.8 & 9.8 & 10.7 & 10.7 \\ -3 & -3.1 & 3.8 & 10.7 & 12.6 & 14.6 \\ -3 & -3.1 & 3.8 & 10.7 & 14.6 & 15.6 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & -1 \\ 1 & 0 & -1 & -2 & -2 & -2 \\ 1 & 0 & -1 & -2 & -3 & -3 \\ 1 & 0 & -1 & -2 & -3 & -2 \end{bmatrix}$$

z_i:

0.908770404173 ± 1.93967680102 i
0.931536974557 ± 1.97197662562 i
4.18245919165
6.13692605089

3) $A(z) = A_o + A_1 z + A_2 z^2$

$$A_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.5 & 1 & 0 & 0 \\ 0 & 0 & -8 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.4 & 1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

z_i:

-1.09579878411
-1.08865049880 ± 1.04888469390 i
-0.541227886923 ± 1.51715942168 i

$z_6 - z_{14} \approx K \cdot 10^7, \quad |K| \approx 1$

The last nine eigenvalues lie in the infinity.

Muller's and Laguerre's methods were also tested on DEC 1091 computer with t=27. In one case when some of the elements of A(z) differed up to 10^{13} in their absolute values Muller's method failed to compute correctly some of the eigenvalues while all the eigenvalues obtained by Laguerre's method were correct.

References

1. BODEWIG, E. Matrix Calculus. North-Holland Publishing Company, Amsterdam, 1959.

2. BOHTE, Z. On Algorithms for the Calculation of Derivatives of the Determinant. Proc. ALGORITHMS 79 Symposium on Algorithms, Strbske Pleso, Czechoslovakia, 1979, pp. 109-110.

3. PETERS, G., AND WILKINSON, J.H. Ax = λBx and the Generalized Eigenproblem. SIAM J. Numer. Anal. 7, 4(1970), 479-492.

4. WILKINSON, J.H. The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.

5. LANCASTER, P. Lambda-matrices and Vibrating Systems. Pergamon Press, Oxford, London, Edinburgh, New York, Toronto, Paris, Braunschweig, 1966.

Programmes

1. Comments

In this chapter two subroutines EIGENM and EIGENL for computing the eigenvalues of a general matrix (2) are presented. An example of a possible main (calling) programme for both subroutines is also given at the end of the chapter. In EIGENM the eigenvalues are computed by Muller's method while in EIGENL by Laguerre's method. If A(z) in (2) satisfies $m \leq 3$ then the subroutines can be used such as they are presented. For $m > 3$ some changes in the subroutines are necessary as explained at the end of the programmes.

The subroutines EIGENM and EIGENL are completely selfcontained (EIGENM is composed of four subroutines EIGENM, F, ALAMDA, LU and EIGENL is composed of ten subroutines EIGENL, ALAMDA, LU, DERIV1, DERIV2, LØWER, UPPER, UPPER2, TRACE, TRXSQ) and communication to them is solely through their argument lists and CØMMØN statements. The entrace to the subroutine EIGENM is achieved by

```
CØMMØN /MATRIX/ A0(ND,ND), A1(ND,ND), A2(ND,
1ND)
CALL EIGENM (M, N, ND, T, INDIC, LAM, PERMUT,
1A, L, U)
```

The entrance to the subroutine EIGENL is achieved by

```
CØMMØN /MATRIX/ A0(ND, ND), A1(ND,ND), A2(ND,
1ND), A3(ND,ND)
CALL EIGENL (M, N, ND, T, INDIC, LAM, PERMUT,
1A, D, L, U)
```

The meaning of the parameters is described in the following lines.

The elements of matrices $A_0, A_1, A_2, \ldots, A_m$ in (2) are to be stored in the first N rows and columns of the two dimensional arrays A0, A1, A2, A3,... .

M, where $M \geq 1$, is the degree of z-matrix A(z) in (2).

N, where $N \geq 1$, is the order of matrices $A_0, A_1, A_2, \ldots, A_M$.

ND, where $ND \geq N$, defines the first dimension of the two dimensional arrays A0, A1, A2,.. .., A,D,L,U and the dimension of the one dimensional array PERMUT. Dimension of the one dimensional arrays INDIC and LAM must be equal to or greater than ND*M.

T is the number of binary digits in the

mantissa of a single precision floating-point number. T is of integer type.

The array INDIC indicates the success of the subroutine as follows

value of INDIC(I)	eigenvalue (I)
0	not found after 100 iteration steps
1	found

LAM is one dimensional complex array. The computed eigenvalues will be found in the first N*M places.

The arrays PERMUT, A, D, L, U is the working storage.

The main (calling) programme should contain the following declaration statements

```
      INTEGER, M,N,T,INDIC,PERMUT
      REAL A0,A1,A2,A3
      CØMPLEX LAM,A,L,U
      CØMMØN /MATRIX/ A0(ND,ND), A1(ND,ND),
     1A2(ND,ND),A3(ND,ND)
      DIMENSIØN INDIC (ND*M),LAM(ND*M),PERMUT
     1(ND),A(ND,ND+1),L(ND,ND+1),U(ND,ND+1)
      EQUIVALENCE (A(1),L(1),U(1))
```

and the following two additional declaration statements for the subroutine EIGENL

```
      CØMPLEX D
      DIMENSIØN D(ND,ND)
```

Note that the dimensions of the arrays in DIMENSIØN and CØMMØN statements of the main programme and in CØMMØN statements within subroutines must be constants of integer type. See the example in 5 where ND was replaced by 12, ND+1 by 13 and ND*M by 36.

2. Subroutine EIGENM - Muller's method

The subroutine is composed of four subroutines EIGENM, F, ALAMDA and LU.

```
      SUBRØUTINE EIGENM(M,N,ND,T,INDIC,LAM,
     1PERMUT,A,L,U)
      INTEGER M,N,ND,T,INDIC,PERMUT,I,IND,
     1ITERST, NM, R
      REAL EPS,EPS100, IM, RE
      REAL A0,A1,A2,A3
      CØMPLEX LAM,A,L,U,DENØM1,DENØM2,DR,FR,
     1FRM1,FRM2,GR,HR,HRM1,KR,ZR,ZRM1,ZRM2
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND),
     1A2(ND,ND),A3(ND,ND)
      DIMENSIØN INDIC(1),LAM(1),PERMUT(1),
     1A(ND,1),L(ND,1),U(ND,1)
      NM = N*M
      ITERST = 102
      DØ 1 I=1,NM
         INDIC(I) = 0
    1 CØNTINUE
      I = 1
C     THE MULLER ITERATIVE PRØCESS.
C     DETERMINATIØN ØF THREE STARTING
C     CØNSECUTIVE APPRØXIMATIØNS TØ  THE
C     CØMPUTED EIGENVALUE LAM(I),FØR
C     I=1,2,...NM.
    2 ZR = (1.0,0.0)*FLØAT(I)
    3 ZRM1 = 0.5*ZR
      ZRM2 = 0.1*ZR
C     THE INITIAL CØMPUTATIØNS ØF THE NEXT
C     APPRØXIMATIØN Z(R+1) TØ LAM(I),FØR R=2.
      R = 2
      CALL F(ZRM2,FRM2,EPS,I,IND,M,N,ND,T,LAM,
     1PERMUT,A,L,U)
         IF(IND.EQ.1)GØ TØ 5
      CALL F(ZRM1,FRM1,EPS,I,IND,M,N,ND,T,LAM,
     1PERMUT,A,L,U)
         IF(IND.EQ.1)GØ TØ 6
      CALL F(ZR,FR,EPS,I,IND,M,N,ND,T,LAM,
     1PERMUT,A,L,U)
         IF(IND.EQ.1)GØ TØ 7
C
      HRM1 = ZRM1-ZRM2
      HR   = ZR  -ZRM1
      KR   = HR/HRM1
C     THE MAIN LØØP FØR  THE CØMPUTATIØN ØF
C     THE NEXT APPRØXIMATIØN Z(R+1) TØ LAM(I),
C     FØR R=2,3,...,ITERST.
    4    DR=1.0+KR
         GR=FRM2*KR**2 - FRM1*DR**2 + FR*(KR+DR)
         DENØM2 = CSQRT(GR**2-4.*FR*DR*KR*
     1   (FRM2*KR-FRM1*DR+FR))
         DENØM1 = GR+DENØM2
         DENØM2 = GR-DENØM2
         IF(CABS(DENØM2).GT.CABS(DENØM1))DENØM1
     1   = DENØM2
         KR = -2.0*FR*DR/DENØM1
         HR = HR*KR
         GR = ZR
         ZR = ZR+HR
         FRM2 = FRM1
         FRM1 = FR
         CALL F(ZR,FR,EPS,I,IND,M,N,ND,T,LAM,
     1   PERMUT,A,L,U)
C
         IF(IND.EQ.1)GØ TØ 7
         IF(CABS(ZR-GR).LE.10.*EPS)GØ TØ 7
```

```
      R = R+1
      IF(R.LT.ITERST)GØ TØ 4
      LAM(I) = ZR
      GØ TØ 9
    5 ZR = ZRM2
      GØ TØ 7
    6 ZR = ZRM1
    7 LAM(I) = ZR
      INDIC(I) = 1
      I = I+1
C     CØMPUTE THE CØMPLEX CØNJUGATE VALUE ØF
C     LAM(I).
      RE = REAL(ZR)
      IM = AIMAG(ZR)
      EPS100 = 100.0*EPS
      IF(ABS(IM).LE.EPS100)GØ TØ 8
      LAM(I) = CØNJG(ZR)
      INDIC(I) = 1
      I =I+1
C
    8 IF(NM.LT.I)GØ TØ 9
C     DETERMINATIØN ØF THE STARTING APPRØXIM
C     ATIØN TØ THE CØMPUTED EIGENVALUE LAM
C     (I+1).
      ZR = 0.9*ZR
      RE = ABS(RE)+ABS(IM)
      IF(RE.LE.EPS100.ØR.1./RE.LE.EPS100)GØ
     1TØ 2
      GØ TØ 3
C
    9 CØNTINUE
      RETURN
      END

      SUBRØUTINE F(Z,FUNC,EPS,I,IND,M,N,ND,T,
     1LAM,PERMUT,A,L,U)
      INTEGER I,IND,M,N,ND,T,PERMUT,IM1,J,K
      REAL EPS,FK
      CØMPLEX Z,FUNC,LAM,A,L,U,C
      DIMENSIØN LAM(1),PERMUT(1),A(ND,1),
     1L(ND,1),U(ND,1)
C
C     CØMPUTATIØN ØF FUNC = F(Z) = DET A(Z) =
C     DET A. MATRIX A IS DECØMPØSED INTØ THE
C     FØRM
C                        PA = LU
C     WHERE L IS UNIT LØWER TRIANGULAR MATRIX,
C     U IS UPPER TRIANGULAR MATRIX AND P IS
C     THE CØRRESPØNDING PERMUTATIØN MATRIX
C     (SEE WILKINSØN).
C
C     CALCULATIØN ØF A(Z).
      CALL ALAMDA (Z,M,N,ND,A)
C     CALCULATIØN ØF THE SMALL PØSITIVE
C     NUMBER EPS.
      C = (0.,0.)
      DØ 1 J=1,N
         DØ 1 K=1,N
            C = C + A(J,K)**2
    1 CØNTINUE
      EPS = CABS(CSQRT(C))*2.0**(-T)
C     CØMPUTATIØN ØF THE MATRICES L AND U,
C     WHERE PA = L*U.
      CALL LU(EPS, IND,N,ND,PERMUT,A,L,U)
      IF(IND.EQ.1)GØ TØ 4
C
      J = -IND
      K = (-1)**J
      FK = FLØAT(K)
      FUNC = CMPLX(FK,0.0)
      DØ 2 K=1,N
         FUNC = FUNC*U(K,K)
    2 CØNTINUE
C     MØDIFICATIØN ØF FUNC = F(Z) INTØ F(Z)/
C     ((Z-X(1))(Z-X(2))...(Z-X(I-1)) ACCØRDING
C     TØ EQUATIØN (23) ØF THIS PAPER.
      IM1 = I-1
      IF(IM1.EQ.0)GØ TØ 4
      DØ 3 J=1,IM1
         FUNC = FUNC/(Z-LAM(J))
    3 CØNTINUE
    4 CØNTINUE
      RETURN
      END

      SUBRØUTINE ALAMDA(Z,M,N,ND,A)
      INTEGER M,N,ND,I,J,K
      REAL A0,A1,A2,A3
      CØMPLEX Z,A,AIJ
      CØMPLEX Z2,Z3
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND),
     1A2(ND,ND),A3(ND,ND)
      DIMENSIØN A(ND,1)
C
C     THIS SUBRØUTINE CALCULATES THE ELEMENTS
C     ØF MATRIX A(Z), WHICH ARE FUNCTIØNS ØF Z,
C     FØR A GIVEN VALUE ØF Z.
      Z2 = Z**2
      Z3 = Z**3
      CØNTINUE
      DØ 44 I=1,N
         DØ 44 J=1,N
            AIJ = A0(I,J)
            DØ 33 K=1,M
               GØ TØ (1,2,3),K
    1          AIJ = AIJ+A1(I,J)*Z
               GØ TØ 33
    2          AIJ = AIJ+A2(I,J)*Z2
               GØ TØ 33
    3          AIJ = AIJ+A3(I,J)*Z3
   33       CØNTINUE
            A(I,J) = AIJ
```

```
   44 CØNTINUE
      RETURN
      END

      SUBRØUTINE LU (EPS,IND,N,ND,PERMUT,A,L,
     1U)
      INTEGER IND,N,ND,PERMUT,I,K,NP1,R,RC,
     1RM1,RP1,T
      REAL EPS,SRC,STABS
      CØMPLEX A,L,U,ST,URR
      DIMENSIØN PERMUT(1),A(ND,1),L(ND,1),
     1U(ND,1)
C
C     THIS SUBRØUTINE PERFØRMS THE
C     DECØMPØSITION ØF MATRIX (PA) INTØ L*U,
C     WHERE L IS UNIT LØWER TRIANGULAR MATRIX,
C     U IS UPPER TRIANGULAR MATRIX AND P IS
C     THE CØRRESPØNDING PERMUTATIØN MATRIX.
C     PARTIAL PIVØTING IS INTRØDUCED (SEE J.H.
C     WILKINSON, PAGE 225).
      IND = 0
      DØ 1 I=1,N
         PERMUT(I) = I
    1 CØNTINUE
C
      NP1 = N+1
      DØ 11 R=1,N
         RM1 = R-1
         RP1 = R+1
         RC = R
         SRC = 0
         DØ 4 T=R,N
            ST = A(T,R)
            IF(RM1.EQ.0)GØ TØ 3
C
            DØ 2 K=1,RM1
               ST = ST-L(T,K)*U(K,R)
    2       CØNTINUE
    3       A(T,NP1) = ST
            STABS = CABS(ST)
            IF(SRC.GE.STABS)GØ TØ 4
            SRC = STABS
            RC  = T
    4    CØNTINUE
         IF(SRC.GT.EPS)GØ TØ 5
         IND = 1
         RETURN
C
    5    IF(R.EQ.RC)GØ TØ 7
         T = PERMUT(R)
         PERMUT(R) = PERMUT(RC)
         PERMUT(RC) = T
C     THE FØLLØWING STATEMENT(IND=IND-1) MAY
C     BE TAKEN ØUT IN EIGENL.
         IND = IND-1
         DØ 6 K=1,NP1
            ST = A(R,K)
            A(R,K) = A(RC,K)
            A(RC,K) = ST
    6    CØNTINUE
    7    URR = A(R,NP1)
         U(R,R) = URR
         IF(R.EQ.N)GØ TØ 11
         DØ 10 T=RP1,N
            L(T,R) = A(T,NP1)/URR
            ST = A(R,T)
            IF(RM1.EQ.0)GØ TØ 9
C
            DØ 8 K=1,RM1
               ST = ST - L(R,K)*U(K,T)
    8       CØNTINUE
    9       U(R,T) = ST
   10    CØNTINUE
   11 CØNTINUE
      RETURN
      END
```

3. Subroutine EIGENL - Laguerre's method

The subroutine is composed of ten subroutines EIGENL,ALAMDA (see 2.), LU(see 2.), DERIV1, DERIV2, LØWER, UPPER, UPPER2, TRACE and TRXSQ.

```
      SUBRØUTINE EIGENL(M,N,ND,T,INDIC,LAM,
     1PERMUT,A,D,L,U)
      INTEGER M,N,ND,T,INDIC,PERMUT,I,IM1,IND,
     1ITERST,J,K,NM,NM1,R
      REAL EPS,EPS100,IM,RE
      REAL A0,A1,A2,A3
      CØMPLEX LAM,A,D,L,U,CEPS,DENØM1,DENOM2,
     1S1,S2,TRACEX,TRACEW,TRACX2,ZR,ZRP1
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND)
     1A2(ND,ND),A3(ND,ND)
      DIMENSIØN INDIC(1),LAM(1),PERMUT(1),
     1A(ND,1),D(ND,1),L(ND,1),U(ND,1)
C
      NM = N*M
      ITERST = 101
      DØ 1 I=1,NM
         INDIC(I)=0
    1 CØNTINUE
      I=1
C     THE LAGUERRE ITERATIVE PRØCESS.
C     DETERMINATIØN ØF THE STARTING
C     APPRØXIMATIØN TØ THE FIRST CØMPUTED
C     EIGENVALUE LAM(1).
      ZR = (1.,1.)
    2 NM1 = NM-I
C     CØMPUTATIØN ØF THE NEXT EIGENVALUE LAM(I).
      R = 1
C
C     THE MAIN LØØP FØR THE CØMPUTATIØN ØF THE
```

```
C     NEXT APPRØXIMATIØN Z(R+1) TØ LAM(I).
C     CALCULATIØN ØF THE EQUATIØNS (10) - (22)
C     ØF THIS PAPER.
C     CALCULATIØN ØF A(Z).
    3 CALL ALAMDA(ZR,M,N,ND,A)
C     CALCULATIØN ØF THE SMALL PØSITIVE NUMBER
C     EPS.
      CEPS = 0
      DØ 4 J=1,N
         DO 4 K=1,N
            CEPS = CEPS+A(J,K)**2
    4 CØNTINUE
      EPS = CABS(CSQRT(CEPS))*2.0**(-T)
      EPS100 = 100.*EPS
C     CØMPUTATIØN ØF THE MATRICES L AND U,
C     WHERE PA = L*U.
      CALL LU(EPS,IND,N,ND,PERMUT,A,L,U)
      IF(IND.EQ.1)GØ TØ 8
C     CØMPUTATIØN ØF THE TRACE ØF MATRIX X
C     AND THE TRACE ØF X**2.
      CALL DERIV1(ZR,M,N,ND,PERMUT,D)
      CALL LØWER(N,ND,D,L)
      CALL UPPER(N,ND,D,U)
      CALL TRACE(N,ND,D,TRACEX)
      CALL TRXSQ(N,ND,D,TRACX2)
C     CØMPUTATIØN ØF THE TRACE ØF MATRIX W.
      TRACEW = 0.0
      IF(M.LT.2)GØ TØ 5
      CALL DERIV2(ZR,M,N,ND,PERMUT,D)
      CALL LØWER(N,ND,D,L)
      CALL UPPER2 (N,ND,D,U)
      CALL TRACE(N,ND,D,TRACEW)
C
    5 S1 = TRACEX
      S2 = TRACX2-TRACEW
C     MØDIFICATIØNS ØF S1 AND S2 ACCØRDING
C     TØ THE EQUATIØNS (24) AND (25) ØF THIS
C     PAPER.
      IM1 = I-1
      IF(IM1.EQ.0)GØ TØ 7
      DØ 6 J=1,IM1
         CEPS = 1.0/(ZR-LAM(J))
         S1 = S1-CEPS
         S2 = S2-CEPS**2
    6 CØNTINUE
C
    7 CEPS = CSQRT(NM1*((NM1+1)*S2-S1**2))
      DENØM1 = S1+CEPS
      DENØM2 = S1-CEPS
      IF(CABS(DENØM1).GT.CABS(DENØM2))DENØM2=
     1DENØM1
      ZRP1 = ZR-(NM1+1)/DENØM2
C
      IF(CABS(ZRP1-ZR).LE.10.*LPS)GØ TØ 9
      ZR = ZRP1
      R = R+1
      IF(R.LT.ITERST)GØ TØ 3
      LAM(I) = ZR
      GØ TØ 11
C
    8 ZRP1 = ZR
    9 LAM(I) = ZRP1
      INDIC(I) = 1
C     DETERMINATIØN ØF THE STARTING
C     APPRØXIMATIØN TØ THE NEXT CØMPUTED
C     EIGENVALUE LAM(I+1) AND CØMPUTATIØN ØF
C     THE CØMPLEX CØNJUGATE VALUE ØF LAM(I).
      ZR = 0.9*ZRP1
      I = I+1
      RE = REAL(ZRP1)
      IM =AIMAG(ZRP1)
      IF(ABS(IM).LE.1.0E-3*ABS(RE))ZR=0.5*
     1CMPLX(RE,RE)
      RE = ABS(RE)+ABS(IM)
      IF(RE.LE.EPS100.ØR.1./RE.LE.EPS100)
     1ZR=(1.,1.)*FLØAT(I)
      IF(ABS(IM).LE.EPS100)GØ TØ 10
      LAM(I) = CØNJG(ZRP1)
      INDIC(I) = 1
      I = I+1
C
   10 IF(I.LE.NM)GØ TØ 2
   11 CØNTINUE
      RETURN
      END

      SUBRØUTINE DERIV1(Z,M,N,ND,PERMUT,A)
      INTEGER M,N,ND,PERMUT,I,II,J,K,MM1
      REAL A0,A1,A2,A3
      CØMPLEX Z,A,AIJ
      CØMPLEX Z2,Z3
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND)
     1A2(ND,ND),A3(ND,ND)
      DIMENSIØN A(ND,1),PERMUT(1)
C
C     THIS SUBRØUTINE CALCULATES THE ELEMENTS
C     ØF THE FIRST DERIVATIVE ØF MATRIX A(Z),
C     FØR A GIVEN VALUE ØF Z.
      Z2 = 2.*Z
      Z3 = 3.*Z**2
      CØNTINUE
      MM1 = M-1
      DØ 55 II=1,N
         I = PERMUT(II)
         DØ 55 J=1,N
            AIJ = A1(I,J)
            IF(M.LT.2)GØ TØ 44
            DØ 33 K=1,MM1
               GØ TØ (2,3),K
    2          AIJ = AIJ+A2(I,J)*Z2
               GØ TØ 33
    3          AIJ = AIJ+A3(I,J)*Z3
```

```
   33       CØNTINUE
   44       A(II,J)=AIJ
   55 CØNTINUE
      RETURN
      END

      SUBRØUTINE DERIV2(Z,M,N,ND,PERMUT,A)
      INTEGER M,N,ND,PERMUT,I,II,J,K,MM2
      REAL A0,A1,A2,A3
      CØMPLEX Z,A,AIJ
      CØMPLEX Z2,Z3
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND),
     1A2(ND,ND),A3(ND,ND)
      DIMENSIØN A(ND,1),PERMUT(1)
C
C     THIS SUBRØUTINE CALCULATES THE ELEMENTS
C     ØF THE SECØND DERIVATIVE ØF MATRIX A(Z),
C     FØR A GIVEN VALUE ØF Z.
      Z3 = 6.*Z
      CØNTINUE
      MM2 = M-2
      DØ 55 II=1,N
         I = PERMUT(II)
         DØ 55 J=1,N
            AIJ = 2.*A2(I,J)
            IF(M.LT.3)GØ TØ 44
            DØ 33 K=1,MM2
               GØ TØ (3,3),K
    3          AIJ = AIJ+A3(I,J)*Z3
   33       CØNTINUE
   44       A(II,J) = AIJ
   55 CØNTINUE
      RETURN
      END

      SUBRØUTINE LØWER(N,ND,A,L)
      INTEGER N,ND,I,IM1,J,K
      CØMPLEX A,L,AIJ
      DIMENSIØN A(ND,1),L(ND,1)
C
C     THIS SUBRØUTINE CALCULATES THE ELEMENTS
C     ØF MATRIX M WHICH SATISFIES THE RELATIØN
C     L*M=A, WHERE L IS UNIT LØWER TRIANGULAR
C     MATRIX. M IS STØRED INTØ ARRAY A.
      DØ 2 I=2,N
         IM1 = I-1
         DØ 2 J=1,N
            AIJ = (0.,0.)
            DØ 1 K=1,IM1
               AIJ = AIJ+L(I,K)*A(K,J)
    1       CØNTINUE
            A(I,J)=A(I,J)-AIJ
    2 CØNTINUE
      RETURN
      END

      SUBRØUTINE UPPER(N,ND,A,U)
      INTEGER N,ND,I,II,IP1,J,K
      CØMPLEX A,U,AIJ
      DIMENSIØN A(ND,1),U(ND,1)
C
C     THIS SUBRØUTINE CALCULATES THE ELEMENTS
C     ØF MATRIX M WHICH SATISFIES THE RELATIØN
C     U*M=A, WHERE U IS UPPER TRIANGULAR
C     MATRIX. M IS STØRED INTØ ARRAY A.
      DØ 3 II=1,N
         I = N+1-II
         IP1 = I+1
         DØ 3 J=1,N
            AIJ = (0.,0.)
            IF(I.EQ.N)GØ TØ 2
            DØ 1 K=IP1,N
               AIJ = AIJ+U(I,K)*A(K,J)
    1       CØNTINUE
    2       A(I,J) = (A(I,J)-AIJ)/U(I,I)
    3 CØNTINUE
      RETURN
      END

      SUBRØUTINE TRACE(N,ND,A,TR)
      INTEGER N,ND,I
      CØMPLEX A,TR
      DIMENSIØN A(ND,1)
C
C     THIS SUBRØUTINE CALCULATES THE TRACE
C     ØF MATRIX A.
      TR = (0.,0.)
      DØ 1 I=1,N
         TR = TR+A(I,I)
    1 CØNTINUE
      RETURN
      END

      SUBRØUTINE TRXSQ(N,ND,A,TR)
      INTEGER N,ND,I,K
      CØMPLEX A,TR
      DIMENSIØN A(ND,1)
C
C     THIS SUBRØUTINE CALCULATES THE TRACE ØF
C     A**2.
      TR = (0.,0.)
      DØ 1 I=1,N
         DØ 1 K=1,N
            TR = TR+A(I,K)*A(K,I)
    1 CØNTINUE
      RETURN
      END

      SUBRØUTINE UPPER2 (N,ND,A,U)
      INTEGER N,ND,I,II,IP1,J,K
      CØMPLEX A,U,AIJ
      DIMENSIØN A(ND,1),U(ND,1)
```

```
C
C     THIS SUBRØUTINE CALCULATES THE
C     ELEMENTS OF THE LØWER TRIANGULAR ØF
C     MATRIX M. M SATISFIES THE RELATIØN U*M=A,
C     WHERE U IS UPPER TRIANGULAR MATRIX. M
C     IS STØRED INTØ ARRAY A.
      DØ 3 II=1,N
         I = N+1-II
         IP1 = I+1
         DØ 3 J=1,I
            AIJ = (0.,0.)
            IF(I.EQ.N)GØ TØ 2
            DØ 1 K=IP1,N
               AIJ = AIJ+U(I,K)*A(K,J)
    1       CØNTINUE
    2       A(I,J)=(A(I,J)-AIJ)/U(I,I)
    3 CØNTINUE
      RETURN
      END
```

4. Modifications

For matrix A(z) satisfying (2), where m>3, some of the statements must be replaced with other statements as shown below for the case m≤5:

1) Statements

```
      REAL A0,A1,A2,A3
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND),
     1A2(ND,ND),A3(ND,ND)
```

in the main programme, EIGENM, ALAMDA, EIGENL, DERIV1 and DERIV2 by

```
      REAL A0,A1,A2,A3,A4,A5
      CØMMØN /MATRIX/ A0(ND,ND),A1(ND,ND),
     1A2(ND,ND),A3(ND,ND),A4(ND,ND),A5(ND,ND)
```

2) Statement CØMPLEX Z2,Z3 in ALAMDA,DERIV1 and DERIV2 by

```
      CØMPLEX Z2,Z3,Z4,Z5
```

3) Statement CØNTINUE

(i) in ALAMDA by

```
      Z4 = Z**4
      Z5 = Z**5
```

(ii) in DERIV1 by

```
      Z4 = 4.*Z**3
      Z5 = 5.*Z**4
```

(iii) in DERIV2 by

```
      Z4 = 4.*3.*Z**2
      Z5 = 5.*4.*Z**3
```

4) Statement GØ TØ(1,2,3),K in ALAMDA by

```
      GØ TØ(1,2,3,4,5),K
```

5) Statement GØ TØ(2,3),K in DERIV1 by

```
      GØ TØ(2,3,4,5),K
```

6) Statement GØ TØ(3,3),K in DERIV2 by

```
      GØ TØ(3,4,5),K
```

7) Statement 33 CØNTINUE in ALAMDA,DERIV1 and DERIV2 by

```
      GØ TØ 33
    4 AIJ = AIJ+A4(I,J)*Z4
      GØ TØ 33
    5 AIJ = AIJ+A5(I,J)*Z5
   33 CØNTINUE
```

5. Main - calling programme

In the following lines an example of a possible calling programme for subroutines EIGENM and EIGENL is presented. It finds the eigenvalues of A(z) in (2) with m≤3 and n≤12. The programme stops after reading value zero for n.

```
      INTEGER INDIC,PERMUT,M,N,T,I,J,K,MN,MP1
      REAL A0,A1,A2,A3
      CØMPLEX LAM,A,L,U
      CØMMØN /MATRIX/ A0(12,12),A1(12,12),
     1A2(12,12),A3(12,12)
      EQUIVALENCE (A(1),L(1),U(1))
      DIMENSIØN INDIC(36),LAM(36),PERMUT(12),
     1A(12,13),L(12,13),U(12,13)
      CØMPLEX D
      DIMENSIØN D(12,12)
C
    1 READ 2,N,M,T
    2 FØRMAT(3I5)
    3 FØRMAT(8F10.0)
      IF(N.EQ.0)GØ TØ 12
C     THE PRØGRAMME STØPS AFTER READING ZERØ
C     FØR N.
      MN = M*N
      MP1 = M+1
      DØ 8 I=1,MP1
         GØ TØ (4,5,6,7),I
    4    READ 3,((A0(K,J),J=1,N),K=1,N)
         GØ TØ 8
    5    READ 3,((A1(K,J),J=1,N),K=1,N)
         GØ TØ 8
    6    READ 3,((A2(K,J),J=1,N),K=1,N)
         GØ TØ 8
    7    READ 3,((A3(K,J),J=1,N),K=1,N)
    8 CØNTINUE
```

```
C
      PRINT 9,((A0(K,J),J=1,N),K=1,N)
    9 FØRMAT (1H ,6E16.7)
      PRINT 9,((A1(K,J),J=1,N),K=1,N)
      IF(M.GT.1)PRINT 9,((A2(K,J),J=1,N),
     1K=1,N)
      IF(M.GT.2)PRINT 9,((A3(K,J),J=1,N),
     1K=1,N)
C
      CALL EIGENM(M,N,12,T,INDIC,LAM,PERMUT,
     1A,L,U)
C
      PRINT 10,(INDIC(I),LAM(I),I=1,MN)
   10 FØRMAT(1HØ,10X,6H INDIC,10X,9HREAL PART,
     112X,10H IMAG.PART,///(1H ,12X,I2,5X,
     1E19.12,4X,E19.12))
      DØ 11 I=1,MN
         LAM(I) = 0.0
   11 CØNTINUE
C
      CALL EIGENL(M,N,12,T,INDIC,LAM,PERMUT,
     1A,D,L,U)
      PRINT10,(INDIC(I),LAM(I),I=1,MN)
      GØ TØ 1
C
   12 CØNTINUE
      END
```

GENERISANJE PRIDEVSKIH OBLIKA U SRPSKOHRVATSKOM

DUŠKO VITAS

MATEMATIČKI INSTITUT, KNEZ MIHAILOVA 35
11000 BEOGRAD

UDK: 681.3.06:808.61

U radu je opisan postupak automatske morfonološke sinteze pridevskih oblika u savremenom srpskohrvatskom jeziku. Ovaj postupak je proširenje ranije opisanog postupka morfonološke sinteze imenica, zasnovanog na formalizaciji pravila alternacija i identifikaciji dela reči, na koji deluju alternacije.

GENERATION OF ADJECTIVAL FORMS IN SERBOCROATIAN. In this paper, the procedure of automatic morphonological synthesis of adjectival forms in contemporary serbocroatian in described. It is an extension of the synthesis of noun forms described earlier, and it is based on formalisation of alternation rules and identification of parts of the words, that are liable to alternation.

1. UVOD

Postupak generisanja paradigmi promenljivih reči u srpskohrvatskom jeziku je izložen u (Vitas/80/) na primeru generisanja imeničkih oblika. Polazeći od nominativa jednine date imenice, snabdevenog morfonološkom definicijom, moguće je generisati sve padežne oblike te imenice. Morfonološkom definicijom su, pri tome, opisana neka (ne sva) od morfoloških i fonoloških svojstava imenice koja se generiše. Ukoliko u određenom padežu, imenica dobija više različitih oblika, onda je moguće generisati ih varijacijom parametara u morfonološkoj definiciji.

U ovom radu se opisuje proširenje ovog generatornog postupka na pridevsku paradigmu. Polazeći od nominativa jednine neodređenog vida muškog roda (ili određenog vida, ukoliko neodređeni ne postoji) datog prideva generišu se oblici sva tri roda u određenom i neodređenom vidu, u pozitivu, komparativu i superlativu. Proširivanje sistema na pridevsku paradigmu je predstavljalo proveru generatornog mehanizma primenjenog na slučaj imenica. Ovaj mehanizam se sastoji u dekompoziciji oblika reči na nepromenljivi deo (nazvan osnovni segment, sadržan u osnovi reči) i promenljivi deo (nazvan sufiks-segment), čime je omogućeno da se dejstvo alternacija (glasovnih zakona) ograniči samo na sufiks-segment. Na ovaj način se problem generisanja pridevskih oblika, kao i u slučaju generisanja imeničkih paradigmi, svodi na problem odgovarajuće transformacije sufiks-segmenta.

U radu se dalje izlažu osobenosti pridevske promene u srpskohrvatskom jeziku, kao i svojstva programske realizacije generatornog postupka.

2. OPIS PRIDEVSKE PROMENE

Definicija i opis prideva sa lingvističkog stanovišta iscrpno je dat u (Žiletić/80/). Ovde ćemo stoga istaći samo one karakteristike pridevske promene koje su od značaja u postupku generisanja oblika.

Pridevi mogu imati tri stupnja poređenja (pozitiv, komparativ, superlativ). Pridev se u pozitivu može javiti u dva vida (određenom i neodređenom) i u sva tri roda (muški, ženski, srednji). U komparativu i superlativu, ukoliko

ovi stupnjevi postoje, pridevi primaju samo odredjeni vid. Ako se opredelimo da ulazni oblik prideva u sistem za generisanje bude nominativ jednine neodredjenog vida muškog roda u pozitivu (ukoliko postoji), onda se bilo koji drugi oblik paradigme u odnosu na ovaj oblik odredjuje sledećim parametrima:

mi - broj (jednina, množina)
mj - padež
mr - rod (1.muški, 2. srednji, 3. ženski)
mk - stupanj (1.pozitiv, 2. komparativ, 3.superlativ),
mv - vid (1. neodredjeni, 2. odredjeni).

Postupak generisanja se tada može odrediti kao preslikavanje koje ulazni oblik prideva pod dejstvom ovih parametara prevodi u oblik zadat tim parametrima. Ovo preslikavanje je odredjeno fonološkim i morfološkim osobinama samog prideva, koje delom, takodje, predstavljaju ulazni podatak (nizovi MORF i MFMARK).

Ove morfološke osobine su prikazane u dodatku 1 dok su fonološke definisane na isti način kao i u (Vitas/80/). Izvestan broj morfonoloških osobina prideva je moguće, kao i kod imenica, odrediti tokom generisanja oblika ispitujući svojstva sufiks-segmenta (npr. struktura konsonantske grupe u sufiks-segmentu odredjuje primenu nekih alternacija).

Treba napomenuti da pridev ne mora imati neodredjeni ili odredjeni vid, pozitiv (kada je komparativ u funkciji pozitiva, npr. gornji) ili, pak, komparativ i superlativ. Takodje, pridev može biti nepromenljiv (npr. braon).

Poseban slučaj predstavljaju tzv. poimenični pridevi, koji su po svojim morfonološkim svojstvima pridevi a po funkciji u rečenici imenice (npr. Hrvatska, Bačka, prava, itd.). Generisanje oblika poimeničenih prideva se svodi na fiksiranje nekih od pomenutih ulaznih parametara u morfonološkoj definiciji. Na primer, za poimenični pridev "prava" mogu se menjati samo parametri mi i mj, a za "Bačka" samo mj, dok je mi=1 (upotrebljava se samo jednina).

Generatorni postupak se sada sastoji u sledećim koracima:

-ispitivanje egzistencije i odredjivanje svojstava traženog oblika;
-identifikacija sufiks-segmenta i ispitivanje konsonanata i konsonantskih grupa u sufiks-segmentu;
-pozicioniranje parametara alternacija;
-odredjivanje padežnih nastavaka, infiksa i osnove nepravilnog komparativa (npr. dobar-bolji) i dopisivanje prefiksa naj- superlativu.

Realizacijom ovih koraka formira se niz MARK, kojim su opisane sve dozvoljene transformacije sufiks-segmenta. Dalji postupak je isti kao i za imenice (rutina ALTER). U programskoj realizaciji gornjih koraka učestvuje izvestan broj rutina opisanih u (Vitas/80/). Liste padežnih nastavaka su date u prilogu 2 a lista infiksa u prilogu 3. Primeri generisanih oblika su prikazani na kraju ovog rada.

3. ZAKLJUČAK

U okviru projekta "Matematička i računarska lingvistika" Matematičkog instituta u Beogradu u toku su istraživanja na generisanju glagolskih oblika, kao i na razvijanju algoritama za morfonološku analizu, koja se zasniva na upotrebi jednog "morfonološkog" rečnika, čija je struktura za sada delimično osvetljena radovima o generisanju imeničkih i pridevskih oblika. Ovakvim pristupom bi se, obično, veoma komplikovan postupak opisivanja odnosa alomorfa osnove i nastavaka, postupak koji je, inače, karakterističan za sisteme morfološke analize, značajno pojednostavio jer je dovoljno navesti u rečniku samo jedan, unapred utvrdjen oblik reči, dok su alomorfi osnove implicitno opisani pridruženim morfonološkim definicijama.

LITERATURA

1. Vitas D. /80/: Generisanje imeničkih oblika u srpskohrvatskom jeziku, Informatica 1980 (3), Ljubljana, pp. 34-39
2. Žiletić Z. /80/: Adjektiv und Adjektivphrase im Deutscen und Serbokroatischen, (biće objavljeno 1983 u Deutsch-serbokroatische kontrastive Gramatik, Bibliographisches Institut, Mannheim)

Prilog 1. Niz MORF

1. Vrsta reči (pridev,...)
2. Promenljivost

Markeri saglasnosti sa imenicom

3. Rod
4. Marker "živo-neživo"

Markeri egzistencije

5. Postoji li neodredjeni vid?
6. Postoji li odredjeni vid?
7. Postoji li pozitiv?
8. Postoji li komparativ?

Markeri komparativa:

9. Pravilno formiranje komparativa
10. Postoji li infiks pozitiva?
11. Tip infiksa za komparativ

Markeri poimeničenja

12. Rod poimeničenog prideva
13. Ima li množinu?

Produženi nastavci:

14. Nastavak u GJ
15. Nastavak u DJ
16. Nastavak u DM

Prilog 2. Lista pridevskih nastavaka

1. -ø	11. -em
2. -a	12. -og
3. -u	13. -eg
4. -i	14. -ima
5. -o	15. -oga
6. -e	16. -ega
7. -im	17. -omu
8. -ih	18. -emu
9. -oj	19. -ome
10. -om	

Prilog 3. Lista infiksa

Infiksi pozitiva	Infiksi komparativa
1. -ak-	1. -j-
2. -ek-	2. -ij-
3. -ok-	3. -š-
4. -e(o/l)-	

Primeri generisanja pridevskih oblika

N- nominativ — S - jednina
G- genitiv — P - množina
D- dativ

Pozitiv

```
TEZAK      1 6
MCRF  = 2 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0
MFMARK = T F F F F F F F F F F F
BROJ= 2 PADEZ= 2 KOMP= 1 ROD= 2 VID= 2

NASTAVAK    2   2   1   2   2   8
SUFIX  112
IZMENJENO K=TEZAK
IZMENJENO K=TEZAK
MARK=P 2100111101100000000000000000000000000001
       0000000000000000000000000000000110000000000
NAST=26   1
NAST=10   2
IZMENJENO K=TESXKI
IZMENJENO K=TESXKI
IZMENJENO K=TESXKIH
```

GP (težak) = teških

```
ZAO        1 3
MCRF  = 2 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0
MFMARK = T F T F F F F F F F F F F
BROJ= 1 PADEZ= 1 KOMP= 1 ROD= 2 VID= 1

NASTAVAK    1   1   1   1   2   2
SUFIX  118
IZMENJENO K=ZO
IZMENJENO K=ZL
IZMENJENO K=ZL
MARK=P 2100111110000000000000000000000000000001
       0000000000000000000000000000000110000000000
NAST= 1   1
IZMENJENO K=ZLA
```

NS (zao) = zla (ženski rod neodredjenog vida)

```
BRACN      1 5
MCRF  = 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MFMARK = F F F F F F F F F F F F F F F
BROJ= 2 PADEZ= 1 KOMP= 2 ROD= 1 VID= 2

PRIDEV JE NEPROMENLJIV
NASTAVAK    2   1   2   2   1   1
SUFIX 1816
MARK=P 2000000000000000000000000000000000000000001
       0000000000000000000000000000000110000000000
NAST=31   1
IZMENJENO K=BRACN
```

NP (braon) = braon (nepromenljiv pridev)

Komparativ

```
KRATAK     1 6
MCRF  = 2 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0
MFMARK = T F F F F F F F F F F F F
BROJ= 2 PADEZ= 3 KOMP= 2 ROD= 1 VID= 2

NASTAVAK    2   3   2   2   1   7
SUFIX  122
MARK=P 2100111101100000000000000000000000000001
       0000000000000000000000000000000110000000000
NAST=15   1
NAST=10   2
IZMENJENO K=KRACX
IZMENJENO K=KRACX
IZMENJENO K=KRACXI
IZMENJENO K=KRACXIM
```

DP (kratak) = kraćim

```
ZXESTOK    1 7
MCRF  = 2 1 0 0 1 1 1 1 0 1 1 0 0 0 2 1
MFMARK = F F F F F F F F F F F F F
BROJ= 1 PADEZ= 3 KOMP= 2 ROD= 1 VID= 2

NASTAVAK    1   3   2   2   1   17
SUFIX  72122
MARK=P 2100111101100022200000000000000000000000001
       0000000000000000000000000000000110000000000
NAST=24   1
NAST=15   2
NAST= 7   3
IZMENJENO K=ZXESCX
IZMENJENO K=ZXESXCX
IZMENJENO K=ZXESXCX
IZMENJENO K=ZXESXCXE
IZMENJENO K=ZXESXCXEM
IZMENJENO K=ZXESXCXEMU
```

DS (žestok) = žešćemu (produženi nastavak za DS)

```
OOBAR      1 5
HCRF  = 2 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0
HFMARK = T F F F F F F F F F F F
8ROJ= 1 PADEZ= 1 KOMP= 2 ROD= 2 VID= 1

NASTAVAK    1   1   2   1   2   2
SUFIX 1814
MARK=P 2100111100000000000000000000000000000000000000
       1000000000000000000000000000000011000000000000
NAST= 1   1
IZMENJENO K=BOLXA
```

NS (dobar) = bolja (nepravilna osnova komparativa

Superlativ

```
GLADAK     1 6
4ORF  = 2 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0
4FMARK = T F F F F F F F F F F F
4ROJ= 1 PADEZ= 2 KOMP= 3 ROD= 3 VID= 1

NASTAVAK    1   2   3   1   3   2
SUFIX  1 5
MARK=P 2100111100100000000000000000000000000000000001
       0000000000000000000000000000011000000000000000
NAST= 1   1
IZMENJENO K=NAJGLADX
IZMENJENO K=NAJGLADX
IZMENJENO K=NAJGLADXA
```

GS (gladak) = najgladja

COMPUTER APPLICATIONS OF LINGUISTICS IN PRAGUE

E. HAJIČOVÁ, Z. KIRSCHNER,
J. PANEVOVÁ, P. SGALL

UDK: 681.3.06:800

FACULTY OF MATHEMATICS AND PHYSICS,
CHARLES UNIVERSITY, PRAGUE

Automatic analysis and/or automatic synthesis of natural language is used at various levels, in various ways and to various degrees in projects worked out in the group of algebraic linguistics at the Charles University in Prague. Automatic morphological analysis is applied in a lemmatization processor in experiments with automatic compilation of book indices and in a derivational (reverse lemmatization) processor in a full text Horty type legal information retrieval system SIUT; MOSAIC - a method of fully automatic extraction of terminological expressions or aggregates of terms from texts - is based on a partial schematic morphemic, lexico-semantic and syntactic analysis; KODAS is a simple system of natural language access to a relational data base. The full-fledged apparatus of the automatic analysis and/or synthesis of natural language is applied in experiments with English-Czech machine translation of abstracts in microelectronics, and in an experiment with automatic natural language understanding - a question answering system called TIBAQ, which extracts information from texts, enriches it with the aid of a set of inference rules, stores it and retrieves it by request, the queries being formulated in natural language.

The applied projects worked out in the group of algebraic linguistics on Charles University in Prague are based on the theoretical background /see esp. Sgall, 1967, Sgall et al., 1969, Panevová, 1980/ describing the individual levels of language system, their units and the relations between them, as well as the relation between the representations of sentences on the adjacent levels. The description has the character of an effective procedure, generating first the underlying or tectogrammatical representations of sentences (which can be understood as meanings of sentences) and then transducing these representations step by step to the levels of surface syntax, morphemics, and graphemics (for the purposes concerning practical applications we do not work with a description of phonemics and phonetics, since in the given stage of development the input, and mainly also the output of spoken discourse is possible only at a very limited scale). The linguistic background of the system of levels is described in Sgall, Hajičová and Panevová (in prep.).

1. Some of the projects are relatively simple from the point of view of the depth of linguistic investigations, but they are useful for practical applications. This concerns first of all such projects as automatic compilation of indexes involving either lemmatization or a derivational processor.

The system SIUT (Cz. selekce informací z úplného textu, i.e. selection of data from a full text) aims at an automated system of text information retrieval based on a Czech variant of Horty's method of 'full text', or, in other words, at an immediate identification of texts (and their parts) containing the words or collocations given by the user at the input of the system.

The algorithms for a system based on this method, implemented and checked on an EC 1040 computer in PL/1 for Czech, and now prepared also for Slovak, deal with such difficult points of morphemics as the productive alternations k/c, ch/š, h/z, r/ř, the moving vowels (as in Cz. domek - domku, matka - matek), etc. The system works without a lexicon, and it includes a derivational processor, which generates all the derived forms of a word, if the basic form of this word was given by the user, who also has to add a denotation of the part of speech (with nouns, the gender):

Z - noun, animate (Cz. životné)
N - " , inanimate (Cz. neživotné)
F - " , feminine
S - " , neutre (Cz. střední)

A - adjective
C - numeral(Cz. číslovka)

Thus e.g. with TŘÍPOKOJOVÝ (A) BYT (N) - for "an appartment of three rooms" - both words are assigned all their word forms by the processor. With OPRAVA (F) V BYTĚ ("a repair in the appartment") or OPRAVA (F) DVEŘÍ ("a repair of the doors") only the first word is accompanied by the auxiliary symbol (for part of speech and gender), so that only for this word its other forms will be generated; the other word forms are left as they are, since it is not necessary to look e.g. for other forms of the noun BYT ("appartment").

All forms found at the input or generated by the derivational processor are then looked for in a concordance that was compiled when the texts were entered into the system (and that is complemented with every new text added to the corpus). Not only the word form itself, but also the data concerning the places (texts, paragraphs, etc.) of their occurrences are found in the concordance. It is then checked which of these data meet the condition of common occurrence; regularly the distance ± 3 words is tolerated, e.g. with our first examples also such occurrences as třípokojových a větších bytů or bytů dvoupokojových a třípokojových will be found and indicated at the output. However, the user has also the possibility to use several operators pointing out that only the exact form given at the input (without any intervening words or permutations) should be looked for, or that the individual parts of the term may occur in any distance within a single sentence, or within a paragraph. Also alternative (e.g. synonymous) words can be looked for.

The system SIUT is appropriate e.g. for texts from the domain of law, where it is rather probable that every occurrence of the term looked for can be relevant for the user. If the aim of a text retrieval system consists in the so-called retrospective search in a polytechnical domain, i.e. all texts (and their parts) should be found in which the given term plays an important role (all texts dealing with the subject denoted by this term are relevant, rather than the texts containing only casual mentions of the term), another method is more useful. For Slavonic languages with their rich systems of inflectional form the method MOSAIC was developed by Z. Kirschner. MOSAIC is a method of fully automatic extraction of terminological structures from unabbreviated texts in the field of natural science and technology. It is based on the assumption that elements of terminology and their coherent aggregates are automatically detectable in compliance with principles and general trends that assert themselves in the construction of terminology. A partial and schematic morphemic analysis drawing upon the rules of derivational and lexico-semantically based morphology helps to select potential terminological elements, and the subsequent syntactic analysis in the form of a set of categorial grammar type rules tests the coherence of the contingent sequences of them. All that is needed is a relatively limited list of four character segments mostly representing the endings or suffixes of words, or, in part, some inner segments representing the stems of words forming the most important families of terms; e.g., the experiment in the field of microelectronics in Czech contained a list of some 750 segments, the corresponding experiment in Slovak some 850 segments etc. The method can be applied with very good results in languages with sufficiently rich inflection, viz. Czech, Slovak, Polish, Russian, Serbocroatian, but also German etc. MOSAIC has been applied as a system for automatic indexing; in this application, a system of weighting the terms is operative measuring the significance of individual indices according to their position in the text, their length, relations to other terms and frequency. The applicability to such tasks as compilation of book indices has been also tested, and the results were very good. The implementation of MOSAIC on computers EC 1040, IBM 370 etc. is the work of prof. P. Pognan of the Jean Favard Research Centre of the 6th Paris University. The programmes are written in PL/1 of the level F and PL/1 Optimizer languages. The same programme is used for Czech and Slovak (only the lists of segments being changed), and presently a Russian version is being prepared.

2. Another relatively simple kind of systems concerns automatic contact with simple (e.g. relational) data bases in natural language. If the data base is not too large (not the number of the data, but that of parameters being decisive), then the method KODAS can be used, which was developed by J. Hajič on the base of experiments with similar systems carried out earlier in the Computing Centre of the Academy of Sciences in Novosibirsk. After having implemented (also in PL/1) an experimental system concerning a data base with about 15 parameters of personal data, the linguistic group of the Faculty of Mathematics and Physics, Charles University, Prague,

now develops similar systems for larger data bases with data from economy.

We want to present here just a few examples from the first experiments, where the analysis of the questions formulated in Czech was lexically based, since most words (identified by their stems, which are included in the lexicon) unambigously point to their role in the structure of the data base (the word is either a name of one of the parametres, or of a value, of an operator, etc.). Such parametres as first name, family name, date of birth, function, pay, etc. are included, and such operators as maximum, minimum, average. Questions of the following types may serve as an illustration; the answers have the shape of lists (or of numerical data):

Kdo z pracovníků oddělení numerické matematiky je nejstarší?
(Which from the collaborators of the department of numerical mathematics is the oldest?)

Který doktor věd má nejnižší plat?
(Which doctor of sciences has the lowest pay?)

Které oddělení má největší počet pracovníků? (Which department has the highest number of collaborators?)

Jaký je průměrný plat? (Which is the average pay?)

Kolik jazyků umí Jaromír Vomáčka? (How many languages does J.V. speak?)

3. Also experiments with English-to-Czech machine translation have already reached a phase in which the system is prepared for a practical application, first of all with respect to texts from electronics. Thanks to a cooperation of Canadian colleagues who belonged to the group of T.A.U.M. (Montreal), it was possible to prepare three experiments, the first of which was accomplished in the years 1978-1979. Although with each of the experiments partially different aims have been pursued, almost the same general strategies and similar tactical devices have been adhered to.

The main difference between our approach to automatic analysis and that of the Canadian system consists in that we work with dependency grammar instead of that of immediate constituents. We introduce labelling the edges in the graphs, representing the structures analyzed, indicating thus the direction of branching on the one hand and assigning the functions to the dependent (dominated) sentence elements on the other. The automatic analysis of English elaborated in Prague uses Colmerauer's Q-systems; in several respects the analysis is brought to the tectogrammatical level at which semantic functions are represented to the measure that corresponds to further orientation of the particular experiment. Especially the prepositional phrases are analyzed not only syntactically, but also semantically, i.e. their function as an adverbial of a given kind (instrument, manner, place, direction, time, purpose etc.) is identified.

In the first of the three experiments only relatively simple English input sentences (over a limited dictionary) were analyzed, yielding a representation of these sentences suitable as an input for the synthesis of the corresponding set of Czech sentences - which will accomplish the process of their translation. The programme of the analysis is divided into twelve steps which can be grouped into three main blocks: (i) morphological analysis, (ii) syntactic analysis, and (iii) transfer (the first part of which is implemented in Q-language, the second part in PL/1).

At the output of the first block all elements (if they had been identified) appear as trees in what may be called canonical form; in case they underwent morphological treatment, they are followed by standardized forms of the endings that had been separated in the course of this treatment (e. g. S for plural of nouns or third person singular of verbs, ED for past tense, EN for past participle, etc.). In the subsequent stage, such pairs are interpreted: e. g. the rewriting rule N(A* (U*)) + + S=>N(A* (U*, *PL)) represents an instruction for the interpretation of the plural ending -s. Here, the N and S are constants representing the noun category and the ending -s respectively, A*, U* are variables (variables are marked by an asterisk following a letter; letters from the beginning of the alphabet - A to F - denote variables that stand for values (labels, etiquettes), variables represented by letters I to N can stand for trees, and letters U to Z are used in the representation of variables standing for lists of trees; a list can be empty or contain one tree only; the simplest case of a tree is a value). Here, A* stands for the lexical value of the unit in question, U* denotes the list of semantic features assigned to it. The constant *PL denotes plural (semantic features, some grammatical information, etc. are usually marked by an asterisk preceding a letter or a group of letters, mostly mnemonic symbols, abbreviations and others like that).

The block of syntactic analysis is divided into several steps: first, nominal complexes are built up (by putting together nouns with their attributes and propositions); then verbal complex forms are identified. The verbal complements are attached to the verb form one by one: this is being done by filling in the "slots" contained in the dictionary entries of verbs (in their frames). As a rule, the subtree governed by the numeral "1" indicates the properties a subject of the verb must possess, that dominated by "2" those of its direct object, etc. The last step attaches free modifications (local, temporal causal, etc.). The following two rules may serve as an example of a saturation of such verbal frames:

(1) N(A¤(U¤),V¤) + AUX(B¤(X¤,T(C¤),Y¤),Z¤) ⇒
V(B¤(X¤,T(C¤),Y¤), N(A¤)(L(SUB),U¤)V¤),Z¤).

(2) V(U¤) + PP(BY MEANS OF,N(A¤(X¤),Y¤)) ⇒
V(U¤,N(A¤(R($ADV,¤MNS),X¤),Y¤)).

They are applied to the string

(3) N(CHANGE(¤A,¤PL,¤DEF)) + AUX(MONITOR (T(PRS),MDL(CAN),¤NEG,¤PSV,1(¤H,¤C,¤A), 2(¤A),¤DUR)) + PP(BYMEANSOF, N(MEASURE(¤A,¤INDEF),AD(OR(L($ATR),¤STO, ¤SFTO),AD(SINGLE (L($Q))),AD(SIMPLE (R($Q), ¤STO,¤SFTO))))).

In this example the rules and the string were simplified. A stands for adjective, AUX denotes a verbal complex with "slots" not yet filled, V a verbal complex in which the "slots" have already been saturated, PP is a prepositional phrase; T stand for tense, PRS for present tense, MDL for modality, ¤NEG for negation, ¤PSV for passive; semantic features ¤A, ¤C, ¤H stand for the features "abstract","concrete", "human", respectively. The subtrees governed by L and R are labels on the edges: they specify the position (L - left, R - right) and the function of the given part of the sentences: the functions are marked by the sign "$". In the above rules, L(§SUB) and R($ADV,¤MNS) denote the subject standing to the left and the adverbial standing to the right of the verb respectively; ¤MNS stands for the feature "means", the function §Q denotes members of a coordination series. Informally, the above rules state that (in English) if a noun is followed by a verbal complex with unsaturated participants, this noun is inserted into the "slot" of subject of the verb in question; if a verbal complex with participants saturated is followed vy a prepositional phrase with the preposition by means of, the head noun of this phrase is inserted into the complex as an adverbial of manner, specified by the feature "means".

The tree in (3) is transformed by the application of the rules (1) and (2) into the tree (4), which represents the structure resulting from the syntactic processing of the English input sentence "The changes cannot be monitored by a single or simple measure":

(4) S(V(MONITOR(T(PRS), MDL(CAN),¤NEG,¤PSV, ¤DUR),N(CHANGE(L($SUB),¤A,¤PL,¤DEF)), N(MEASURE(R($ADV,¤MNS), ¤A,¤INDEF), AD(OR(L($ATR),¤STO,¤SFTO),AD(SINGLE (L($Q))),AD(SIMPLE(R(§Q),¤STO,¤SFTO)))))).

The root S serves only for the formal purpose of indicating that the analysis is finished; the representation of the complete structure of the sentence proper starts with the node V (representing the governing verb).

The so-called transfer articulates the resulting tree into particular subtrees governed by such categories as N, V, AD, etc., changes their order (placing the dependent elements in front of the governed ones), replaces the original symbols denoting grammatemes by indices used in the synthesis of Czech and substitutes for the English lexical units the corresponding Czech ones. The last part of the transfer, which adapts the output of the Q-language programme to the notation and organization of the input for the programme of synthesis, is written in PL/1.

Several modifications of the system are incorporated in the second experiment, which is now being prepared, and which has in view practical application in an automatic translation system for the INSPEC tape service (confined in the experiment to abstracts from the field of the production and application of integrated circuits). The overall strategy of the procedure is very similar to that used in the first experiment; however, several changes and improvements have been introduced, the more important of which concern the following points:

(i) the programme contains a full system of morphological analysis of English word forms (taken over, in the main, from the first Canadian project T A U M 1973, with kind approval of the authors of the system); thus, almost all irregular, anomalous, or rare forms can be analyzed and the normal (dictionary) forms reconstructed;

(ii) to reduce the scope of the main dictionary, a new component has been added: the so-called translational dictionary, the rules of which translate the most frequent classes of terms of international usage directly into Czech, mostly by changing suffixes and executing orthographical changes - e.g. APPLICATION into APLIKACE, PHILOSOPHY into FILOZOFIE, AMPLIFIER into AMPLIFIKÁTOR, OPERATIONAL into OPERAČNÍ, etc. Such words, provided that their grammatical and semantic properties are not idiosyncraric, need not be included in the main dictionary, which always has the unfortunate tendency to grow beyond measure; at the same time, it is a way how to deal with some words that could not be identified in the previous stage;

(iii) particular attention is paid to the syntactic analysis of nominal complexes in general and compounds in particular, especially with regard to the problems of conversion; a repertoire of semantic features gradually built on what can be called a highly schematic model of the universe of discourse helps to formulate some rules that cover the regular or at least the most frequent phenomena in this domain. The corresponding section of TAUM grammar as well as that of our first experiment were very limited and simple; the texts analyzed by the second experiment grammar - abstracts from the field of electronics - are based on technical terms, abound in them, combine them in various ways, modify them, etc., and that is why they cannot be treated without an adequate apparatus of rules that solve the current problems leaving to the "lexicalist" solution as little as possible;

(iv) since the texts are rich in coordination structures, more sets of rules analyzing different types of conjunction both on the phrasal and sentential levels were included to operate at different stages of the process;

(v) wherever possible, Czech equivalents replace the English lexical values already at the initial stages of the analysis; sets of indices required for the synthesis of Czech are supplied already in the course of dictionary operations; the English semantic features as well as the "slots" in verbal frames are always deleted as soon as they have fulfilled their task;

(vi) the system gives preference to more general solutions whenever possible; e.g. rules are supplied that reconstruct elements deleted in the surface structure of sentences but necessary for the semantic interpretation of the sentence (and thus also for the proper choice of its equivalent construction in Czech);

(vii) more general or universal solutions are also preferred to meet another problem connected with the fact that English, owing to its rather poor morphology and to the almost complete lack of the means of determining referential relationships that is called *grammatical concord*, is a language more vague than Czech; extralinguistic knowledge plays a more important role in English than in Czech, where the elements bound together by referential relationship must agree in case, gender, number, and, with verbs, in person. A computer for which such an "extralinguistic knowledge" is, under present conditions, practically unattainable, will face difficulties if, e.g. it is to decide to which nominal complex the verbal attribute "using..." belongs in such a sentence as "These methods employ a Monte Carlo analysis in the parameter space using a simplicial approximation to the region of acceptability..." A layman can exclude the "space" as an agentive with the aid of the same means as the computer, which can also be endowed with the knowledge that, under normal circumstances, "space" cannot "use" anything, but as for the other two candidates - "methods" and "analysis" - the decision will be difficult without at least some idea of what "Monte Carlo analysis" and "simplicial approximation" are. An experimental system working in laboratory conditions can afford to register all ambiguities and regard a multiple solution as a success - the more so, as the computer is often able to detect ambiguities where man fails to become aware of them; however, a practical system must seek a way out and accept such a solution only in case of inevitability, and, in fact, regard it as a failure. The only chance is to make the output as linguistically ambiguous as is the original utterance in the source language and leave the decision concerning the correct interpretation to the reader. In quite a number of cases, such solution is possible: here, e.g., it can be done by translating the transgressive as "with using", in Czech "s použitím";

(viii) last but not least, a system that aspires to be applied in practice must confront the fact that time from time it can come across a phenomenon that cannot be handled by the means that stand at its disposal. As has been already pointed out (see the above paragraph (ii)), the sim-

plest and probably the most frequent case will be a word that cannot be identified by any of the dictionary operations. For this case, a "universal" noun is prepared, provided with a "universal" set of semantic features and indices and retaining its original lexical value. More difficult problems are to be expected with syntactic and other anomalies; e.g., a noun will be used figuratively so that the intersection of the set of its semantic features and the set of the features required in the "slot" of the verbal frame remains empty; in such a case, the noun fails to become integrated in the verbal complex and the construction of the sentence tree cannot be accomplished. Here, a special stage - a special Q-system - is designed to solve the most frequent failures: a sort of "emergency" grammar. It goes without saying that the particulars of such a "rescue-device" can be drawn only after sufficient experience with the whole system in experimental operation has been accumulated.

The third experiment concerns automatic analysis of English sentences serving as the source of information for the system based on the method called TIBAQ (described in Sect. 4, where also the character of the synthesis of Czech sentences is mentioned). It is closely connected with the second experiment. The front end of the system TIBAQ then will be either a Czech text or an English text; the output structures of the analysis of English sentences are complemented and augmented in order to achieve underlying representations of the English input sentences that might supply the material for compiling the knowledge representation and serve as a base for the operation of inference rules in the same way as the Czech sentences at the input.

4. As for the systems that can be characterized as modelling the understanding of a text, the Prague group decided not to work with narrative texts or common dialogues; instead, the understanding of technical texts was chosen, the purpose being to prepare the first prerequisites necessary for a system which may be called an automatic encyclopaedia (this idea and this term were taken over from David Hays). These prerequisites are prepared on the base of a method called TIBAQ (i.e. text-and-inference based answering of questions). The method TIBAQ differs from the natural language front-end data management systems above all in that it is not connected with the necessity to compile first a data base "intellectually". The set of data is compiled in a fully automatic way, only texts written in the usual form being present at the input of the system.

A system based on the method TIBAQ belongs to those systems of natural language understanding which are connected with question-answering (factual information retrievel), a relatively complete automation of which should become feasible after the prerequisites included in the prepared experiments are checked. Another characteristic feature of TIBAQ is its close connection with linguistic research, its linguistic base.

TIBAQ is based on the following major procedures (see the scheme in Fig.1): a linguistic analysis of the input text (and also of the user's questions), a set of inference rules operating on the output of the linguistic analysis, a look-up for appropriate answers, and a synthesis of the answers found; during the experiments we intend to enrich the whole procedure in several respects, one of which consists in checking every new assertion as for its consistency with the stock of information already stored and for its bringing new information, not yet included in the stock.

The central position in the whole system is that of the set of statements (meanings of input sentences), which are represented in the form of tectogrammatical representations (TR's) of sentences combined into a semantic network of a certain kind.

The syntagmatic relations are treated within the individual TR's of sentences (which are called assertions or statements, if viewed from the standpoint of question-answering) and have the shape of linearized dependency trees, *cf.* Fig.2. The paradigmatic relations are (for the first experiments only in their main parts) accounted for by the register (a list of all concepts and all their occurrences in the texts that have been processed) and by means of indices and pointers in the lexicon (rendering the relevant semantic subclassification of the concepts). It should be noted that in the first experiment we need not be concerned with problems of referential identity because the input texts chosen were found to consist only in general statements (about types of devices, their applications, etc., with no specific referents).

The input sentences are converted to the cor-

responding TR's by the first of the major procedures of the question-answering (see the scheme in Fig.1), by the linguistic analysis, which is divided (in its Czech version) in two main steps: the morphemic and the syntactico-semantic analysis. The first of these steps transduces the input sentences to their morphemic representations, identifying the morphemic values of the word forms (with many ambiguities left unresolved, this step of the analysis being restricted to individual word forms, with the exception of the complex forms of verbs). The second step combines a syntactic and a semantico-pragmatic analysis of the given sentence, taking its morphemic representation as its input and yielding in the output a disambiguated (preferred) reading, i.e. a single TR.

Our syntactico-semantic analysis (the first formulation of which has been characterized in Panevová and Sgall, 1980), thus contains not only a parser, but also an assignment of the underlying roles (Actor, Addressee, Objective, Locative, Instrument, etc.) and of the topic-focus articulation to the sentence and its parts.

It should be pointed out that our linguistic analysis does not contain a syntactic analysis "for its own sake". The distinction between well formed sentences and other strings is concerned here only to the degree in which it is helpful in solving ambiguity, and semantic properties of lexical items (denoted by indices in the lexical entries) are used whenever necessary in the syntactico-semantic analysis. We have never adhered to such slogans as "semantics without syntax", since our understanding of the structure of natural language is connected with the assumption that the meaning of a sentence, in whatever language and shape it may be adequately rendered, is in itself a complex unit with its own syntax, which is accessible to understanding only through surface syntax; in specific cases the lexical cast of a sentence allows just for a single syntactic combination, so that syntax may appear as superfluous (e.g. if the lexical items father and walk, or father, read and newspaper constitute the sentence), but in the general case syntax is indispensable (such questions as "Who saw whom?" have to be answered in the course of the interpretation of a sentence including the verb see, etc.). On the other hand, we distinguish between a theoretical description of language, which, according to our views, should include a complete description of the individual levels, and a procedure of analysis (pursuing practical aims, or modelling a part of a user's performance); within this procedure it does not appear necessary to keep the handling of syntax and of meaning (or of surface and deep syntax) separate.

Also the questions formulated by users in Czech or in English undergo the linguistic analysis, i.e. are translated to their TR's. Every TR of a question is then compared with the concordance (register, list of lexical units included in the texts that have been processed). All assertions displaying coincidences with the TR of the question in autosemantic lexical units are then chosen to form the so-called set of relevant assertions, on the base of which the answers to the question are to be found.

The set of relevant assertions is processed by the inference rules, changing TR's (or pairs of TR's) into other TR's that are entailed by the former ones. A coordinated sentence is divided here into its parts, embedded clauses with since are changed into independent sentences, in some cases nominal adjuncts are deleted, etc. The TR's yielded by this inference procedure are then added to the set of relevant assertions. Also explicit definitions (including e.g. the verb call) are identified and the inference rules allow for replacements of the definiendum by the definiens and vice versa in other assertions. In this way the set of relevant statements is broadened by means of the rules of inference; the resulting set is called the enriched set of relevant statements.

The procedure of the choice of an answer compares the TR of the question with each of the statements belonging to the enriched set. Several kinds of results are possible. The most successful among them consists in a statement having been found to correspond to the TR of the question in all respects with the exception that (a) instead of the wh-word of the question it includes a specific lexical unit (perhaps with other words dependent on it), and (b) this unit is placed at the end of the answering assertion.

An assertion meeting these conditions is denoted as the TR of a full answer to the given question. A partial answer is found if an assertion in the enriched set does not meet the above conditions, but coincides with the TR of the question in its main verb and in all lexical items that depend (immediately or not) on this verb and on which

(immediately or not) the wh-word depends, if also some other lexical unit dependent on one of these words (including the main verb) is identical in the two TR's.

A full answer must contain in its TR the given lexical items in the same underlying roles as their counterparts in the question occupy. This condition is too strong in some cases, and to some extent this limitation was removed by means of inference rules, so that e.g. a question such as "How (Manner) can the input voltage be lowered?" can be answered e.g. by the sentence "The input voltage can be lowered by (means of) the device of the type A (Instrument)". However, complex answers necessary for fully informative responses to such questions as "How can I reach the post office?" (cf. Bayer, 1980) are beyond the scope of the present experiments.

Every TR chosen as underlying an answer is then processed by the synthesis of Czech sentences, which is based on the relatively large generative grammar that was formulated several years ago and is now tested in the form of a random generation of Czech sentences (see the algorithms published in the series Explizite Beschreibung der Sprache und automatische Textbearbeitung, Prague, esp. the issues III to V, 1977 to 1979).

There are several kinds of restrictions connected with the first experiments with TIBAQ: Only a small lexicon has been included, covering two pieces of texts from a subdomain of electronics; the yes/no questions have been excluded, as well as the questions including why. Since the grammatical parts of the procedures are prepared in a rather rich form, it is possible to add thousands of new words into the lexicon without a substantial change of the rules. The linguistic aspects of an enrichment of the system thus will not be connected with problems of principle, but the broadening of the set of inference rules is a difficult task, concerning many still open questions of semantics.

The proper aim of the first experiments, carried out by a small research group at the University, is to show the task as feasible, though it can be fulfilled in a broader scope only if taken over by some larger research institute, which has the possibility to continue the research at a larger scale, to achieve a system appropriate for practical applications.

REFERENCES

Bayer J. (1980): Komplexe Antworten auf wie-Fragen. In: LAB 15: Zweite Jahrestagung der Deutschen Ges. f. Sprachw. in Berlin, 22-26.

Panevová J. (1980): Formy a funkce ve stavbě české věty, Prague.

Panevová J. and P. Sgall (1980): On Some Issues of Syntactic Analysis of Czech, PBML 34.

Sgall P. (1967): Generativní popis jazyka a česká deklinace, Prague.

Sgall P., Nebeský L., Goralčíková A. and E. Hajičová (1969): A Functional Approach to Syntax, New York.

Sgall P., Hajičová E. and J. Panevová (in prep.), The Meaning of the Sentence in Its Semantic and Pragmatic Aspects.

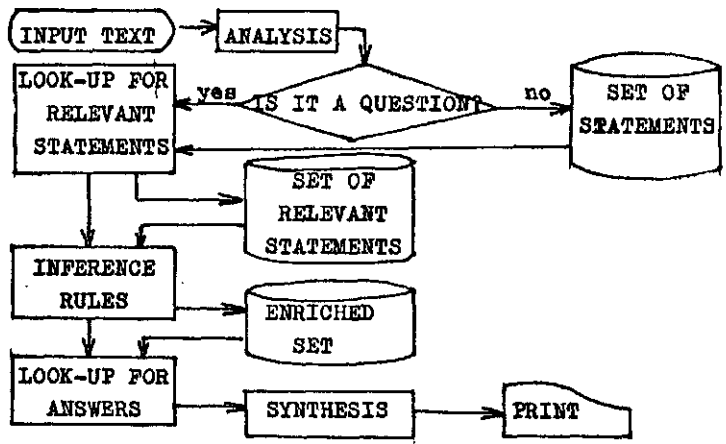
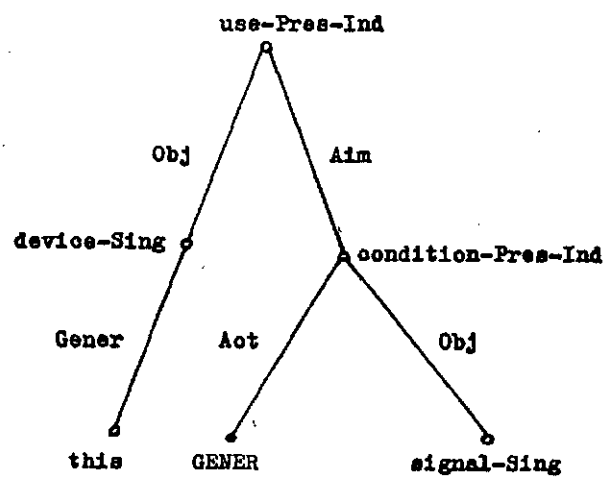


Figure 1.



Note: The labels of edges denote the participants (valency relations): Aot(or), Obj(ective), Aim, Gen(eneral relationship); the grammatical meanings are only sketched here: Pres(ent), Ind(icative), Sing(ular).

Figure 2.

RAČUNANJE BALANSA LOPATICA ZRAKOPLOVNIH TURBINA KORISTEĆI ALFABETSKU METRIKU. NEKE DRUGE METRIKE U PROSTORU PERMUTACIJA

JOVAN LONČAR

VIŠA ZRAKOPLOVNA ŠKOLA – ZAGREB

UDK: 681.3:517.518

U radu se uvodi alfabetska metrika u prostor permutacija π. Zatim se ta metrika koristi za nalaženje debalansa lopatica zrakoplovnih, hidro i parnih turbina. Navode se i druge moguće metrike.

COMPUTATION OF BALANCE OF THE AIRPLANE TURBINE SHOVELS BY USE OF THE ALPHABETIC METRIC. SOME OTHER METRICS IN THE SPACE OF PERMUTATION: In this paper alphabetic metric is introduced in the space of permutation and used for computation of balance of the airplane turbine shovels. Some other metrices in space of the permutation are defined.

UVOD

U [1] [2] [3] i [4] promatrane su lančana, invenziona, transpoziciona i leksikografska metrika. U ovom radu se uvodi alfabetska metrika a navode se i neke druge metrike koje za sada nisu u praktičnoj upotrebi. Navode se neka karakteristična svojstva pojedinih metrika koja druge metrike nemaju. Daju se i uporedni podaci o ponašanju matematičkog očekivanja i disperzije za razne metrike i razne radiuse okolina iz kojih se vrši izbor permutacija.

ALFABETSKA METRIKA

Recimo da su zadane dvije permutacije p_1, p_2 $p_1=(i_1 i_2 \dots i_n)$, $p_2=(j_1 j_2 \dots j_n)$, takve da vrijedi $i_1=j_1$, $i_2=j_2 \dots i_{s-1}=j_{s-1}$ ali je $i_s \neq j_s$, $1 \leq s \leq n-1$. Udaljenost $d(p_1,p_2)$ nazivamo broj n-s. Iz prednjeg slijedi da je udaljenost nula ako se svi elementi podudaraju, a da je maksimalno moguća udaljenost jednaka (n-1). Provjerimo da tako uvedena udaljenost zadovoljava sve aksione metrike [1].

A.1. $d(p_1,p_2)$ o, je očit

A.2. $d(p,p)=$ o, je očit

A.3. $d(p_1,p_2) =$ o $= p_1=p_2$ je očit

A.4. $d(p_1,p_2)=d(p_2,p_1)$ je ispunjen zbog simetrije relacije jednakosti.

A.5. $d(p_1,p_3) \leq d(p_1,p_2)+ d(p_2,p_3)$ slijedi iz slijedećeg izraza:

$$d(p_1,p_3)=\max\left\{d(p_1,p_2),\ d(p_2,p_3)\right\} \qquad (1)$$

Iz formule (1) slijedi da su svi trokuti u alfabetskoj metrici istokračni. Permutacija $p \in \pi$ koja leži u okolini radiusa R s centrom u $p_0=(i_1 i_2 \dots i_n)$, gdje je $1 \leq R \leq n-1$, dobije se pomoću slijedećeg algoritma.

ALGORITAM

Prvi korak: Na slučajan način izaberemo cijeli broj r iz segmenta $(1, R]$

Drugi korak: Na slučajan način sastavimo permutaciju $p=(j_1 j_2 \dots j_{r+1})$ iz r+1 simbola.

Treći korak: U permutaciji p_0 prvih n-(r+1) simbola ostavimo bez promjene a ostale simbole rasporedimo po redosljedu danom u p^*.

Tako se dobije tražena permutacija.

Na osnovu prednjeg algoritma sastavljen je program za elektronski računar.

Iz okoline R=5 s centrom u $p \in \pi$ u kojoj je dobiven min $f(p)$ pomoću metode MONTE KARLO [1], nadjeno je loo permutacija koristeći alfabetsku metriku. Za te permutacije izračunate su pripadne vrijednosti funkcionala (*) čije rezultate vidimo na sl. 1.

$$f(i_1 i_2 \dots i_n)=\left[\left(\sum_{i=1}^{n} M_i \cos\varphi_{k_i}\right)^2+\left(\sum_{i=1}^{n} M_i \sin\varphi_{k_i}\right)^2\right]^{1/2} \qquad (*)$$

gdje je $\varphi_k = \frac{2\pi k}{n}$

Dobiveno je:

Minimalni debalans =	96,49
Maksimalni debalans =	135,64
Matematičko očekivanje =	110,27

Raspored lopatica koji odgovara minimumu debalansa vidimo u tabeli 2 a za maksimum u tabeli 3.

NEKE DRUGE METRIKE

Recimo da imamo dvije permutacije $p_1=(4,1,3,2,5)$ i $p_2=(2,1,3,4,5)$ pa ako p_1 pridružimo broj 41325 a p_2 broj 2 1 3 4 5 onda se udaljenost može definirati kao $d(p_1,p_2)=41325-21345=19980$.

Budući je skup permentacija s ovako definiranom metrikom izometričan skupu nekih cijelih brojeva u euklidskoj metrici to nema potrebe provjeravati aksiome.

Udaljenost izmedju $p_1=(i_1\ i_2 \ldots i_n)$ i $p_2=(j_1 j_2 \ldots j_n)$ možemo definirati i na slijedeće načine:

a) $$d(p_1,p_2)=\left[\sum_{k=1}^{n}(i_k-j_k)^2\right]^{1/2}$$

b) $$d(p_1,p_2)=\max_{1\le k\le n}|(i_k-j_k)|$$

c) $$d(p_1,p_2)=\sum_{k=1}^{n}(i_k-j_k)$$

d) $$d(p_1,p_2)=\left[\sum|i_k-j_k|^s\right]^{1/s}\quad 1\le s<\infty$$

Niti za jedan od navedenih primjera nije potrebno provjeravati aksiome, jer se radi o podskupovima poznatih metričkih prostora.

NEKE USPOREDBE RAZLIČITIH METRIKA

U radovima [1] [2] [3] [4] uvedene su lančana, inverziona, transpoziciona i leksikografska metrika. Leksikografska metrika ima jedno svojstvo koga druge metrike nemaju. Naime, ona nije invarijantna u odnosu na početnu numeraciju objekata. Recimo da su zadani objekti A,B,C,D,E,F i da su izvršene dvije različite numeracije i to:

I numeracija: A 1, B 2, C 3, D 4, E 5, F 6

II Numeracija: A 6, B 5, C 4, D 3, E 2, F 1

Neka su objekti poredani u dva redosljeda I_1= A,B,C,D,E,F I_2= E,A,F,D,C,B .

Uzmemo li prvi način numeracije, tada redosljedu I_1 odgovara permutacija $p_1=(1,2,3,4,5,6)$ a redosljedu I_2 odgovara $p_2=(5,1,6,4,3,2)$. Medjutim ako je primjenjena druga numeracija objekata onda redosljedu I_1 odgovara $p_1=(6,5,4,3,2,1)$ a redosljedu I_2 odgovara $p_2=(2,6,1,3,4,5)$.

U leksikografskoj metrici $d(p_1,p_2)=503$, a $d(p_1,p_2)=575$. U ostalim metrikama je $d(p_1p_2)=d/p_1p_2)$.

Maksimalne udaljenosti izmedju dvije permutacije od n simbola su različite za razne metrike i to:

Lančana metrika ima $\max(p_i,p_j)=n-1$

Leksikografska metrika ima $\max(p_i,p_j)=n!-1$

Alfabetska metrika ima $\max(p_i,p_j)=n-1$

Inverziona metrika ima $\max(p_i,p_j)=n(n-1)/2$

Transpoziciona metrika ima $\max(p_i,p_j)=n-1$

Iz prednjega se vidi da najbolje odvajanje jedne permutacije od druge vrši leksikografska, zatim inverziona a onda ostale metrike. Vidimo da blizina permutacija u inverzionoj metrici zavisi ne samo od broja prekida u lancu elemenata permutacije, što je slučaj kod lančane metrike, već i od toga kako se daleko raznesu isprekidani dijelovi, jedan od drugog, kao i od toga koliko je elemenata permutacije promjenilo svoje uzajamne položaje.

U vezi s prednjim govori se da razne metrike imaju različite statističke neprekidnosti. Treba napomenuti da se kod uvodjenja ove ili one metrike u prostor permutacije polazi od čisto intuitivnih shvatanja, jer koliko je autoru poznato do danas ne postoje apriorne metode pomoću kojih bi mogli medjusobno usporedjivati dvije različite metrike u smislu njihove efektivnosti kod traženja lokalnih ekstrema. Naime, tek nakon niza eksperimenata se konstatira da za odredjenu klasu problema bolje rezultate dobivamo u jednoj metrici u odnosu na druge, jasno sve ovo u vjerojatnonostatističkom smislu. Vršeni su opsežni eksperimenti da se dobije uvid u ponašanje parametara distribucije vrijednosti funkcionala kao što su matematičko očekivanje i disperzija, i to u raznim metrikama i za razne radiuse okoline.

Za podatke iz tabele 1 dobiveni rezultati su prikazani u tabeli 4 i tabeli 5.

ZAKLJUČAK

Alfabetska metrika uvedena u ovom radu u prostor permutacija zajedno s ranije uvedenim metrikama (lančana, inverziona, transpoziciona i leksikografska) izgleda da su dovoljne da se problemi iz raznih područja mogu na prirodan način formulirati, kad se radi o računanju vrijednosti funkcionala na skupu permutacija. Kompjutorski programi za ove metrike čine samo oko 15% paketa programa koji problem rješava u potpunosti.

TABELA 1

Broj lopatice	1	2	3	4	5	6	7	8	9	1o	11	12	13	1e	15	16
Statički moment	0	109	-309	-31	-411	-179	420	-145	-501	61	181	231	-162	-341	-281	175
Broj lopatice	17	18	19	-3e	21	22	23	24	25	26	27	28	29	30	31	32
Statički moment	69	0	-69	-31	-46	176	-511	-412	-317	470	79	13	-142	35	-152	-88
Broj lopatice	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Statički moment	15	-625	- 6	21	-259	- 55	17	11	-259	15	91	-121	475	- 9	-191	81
Broj lopatice	49	5e	51	52	53	54	55	56	57	58	59	6e	61	62	63	64
Statički moment	15	42	3	-11	-249	22	5	-11	- 5	- 2	17	-113	17	-17	-39	488
Broj lopatice	65	66	67	68	69	7o	71	72	73	74	75	76	77	78	79	80
Statički moment	5	25	15	41	5	-26	22	-11	27	-23	-58	75	49	31	-457	-45
Broj lopatice	81	82	83	84	85	86	87	88	89	9o	91	92	93	9e	95	96
Statički moment	-57	- 9	9	0	-11	-25	- 7	31	65	31	44	21	61	21	51	31

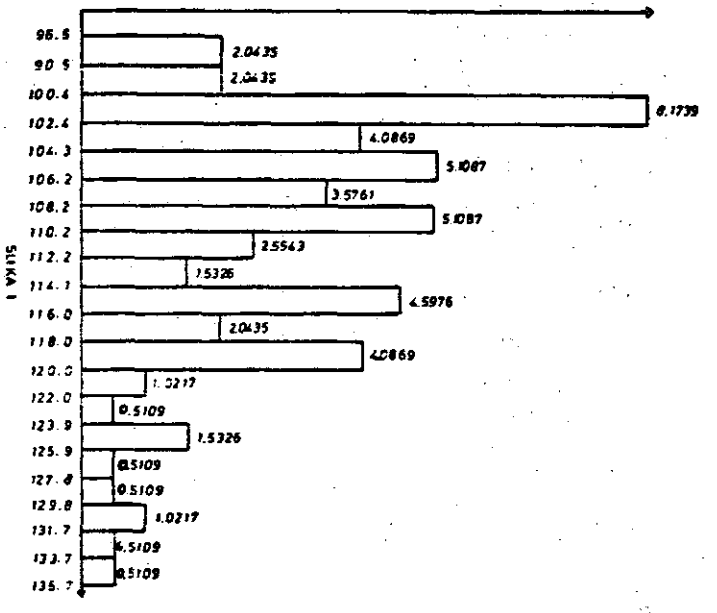


TABELA 3

Broj mjesta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Broj lopatice	52	73	16	37	13	54	79	80	81	83	86	51	4	87	88	85
Statički moment	-11	27	175	-259	-162	22	-457	-45	-57	9	-25	3	-31	- 7	31	-11
Broj mjesta	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Broj lopatice	89	90	91	92	17	19	23	28	11	53	77	32	12	72	59	75
Statički moment	65	31	44	21	49	-69	-511	13	181	-249	49	-88	231	-11	17	-58
Broj mjesta	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Broj lopatice	78	84	93	82	56	30	39	60	63	25	74	8	38	94	55	69
Statički moment	31	0	61	-9	-11	35	17	-113	-39	-317	-23	-145	-53	21	5	5
Broj mjesta	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Broj lopatice	95	96	47	67	2	35	57	1	15	26	2o	29	33	7	5	14
Statički moment	51	31	-191	15	109	- 6	5	0	-281	470	-31	-142	13	420	-411	-341
Broj mjesta	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Broj lopatice	34	36	22	41	44	46	50	49	64	58	27	48	10	21	31	42
Statički moment	-625	21	176	-259	-121	- 9	42	15	488	- 2	79	81	61	-46	-152	15
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatice	61	6	9	24	4o	45	3	18	62	65	66	70	76	71	68	43
Statički moment	17	-179	-301	-412	11	475	-309	0	-17	5	25	-26	75	22	41	91

TABELA 2

Broj mjesta	1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15	16
Broj lopatice	52	73	16	37	13	54	79	80	81	83	86	51	4	87	88	85
Statički moment	-11	27	175	-259	-162	22	-457	-45	-57	9	-25	3	-31	- 7	31	-11
Broj mjesta	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Broj lopatice	89	90	91	92	17	19	23	28	11	53	77	32	12	72	59	75
Statički moment	65	31	44	21	49	-69	-511	13	181	-249	49	-88	231	-11	17	-58
Broj mjesta	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Broj lopatice	78	84	93	82	56	30	39	60	63	25	74	8	38	94	55	69
Statički moment	31	0	61	-9	-11	35	17	-113	- 39	-317	-23	-145	- 53	21	5	5
Broj mjesta	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Broj lopatice	95	96	47	67	2	35	57	1	15	26	2o	29	33	7	5	14
Statički moment	51	31	-191	15	109	-6	5	0	-281	470	-31	-142	13	420	-411	-341
Broj mjesta	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Broj lopatice	34	36	22	41	44	46	50	49	64	58	27	48	10	21	31	42
Statički moment	-625	21	176	-259	-121	- 9	42	15	488	- 2	79	81	61	-46	-152	15
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatice	61	6	9	24	4o	45	3	18	62	65	43	76	68	71	66	70
Statički moment	17	-179	-301	-412	11	475	-309	0	-17	5	91	75	41	22	25	-26

TABELA 4

METRIKA	R = 50		R = 10		R = 5		R = 1		Matematičko očekivanje za R = 5
	Min	Max	Min	Max	Min	Max	Min	Max	
LANČANA	224,5	3.5o4,67	1o2,6	3.36o,o	39,8	3.1oo,9	134,7	134,7	1.o45,56
LEKSIKOGRAFSKA	96,3	135,79	11o,7	135,79	1o4,4	122,79	127,86	135,7	113,25
ALFABETSKA	1o1,4	1.984,84	18,9	362,21	96,49	135,64	134,75	135,7	11o,27
INVERZIONA	115,6	3.547,4	282,4	4.o79,o9	148,36	3.979,25	226,45	3487,3	1.696,97
TRANSPOZICIONA	111,9	3.273,5	198,2	2719,29	36,17	1.824,88	6,78	143o,5	668,58

Metodom Monte-Karlo dobiveno je: Minf(p)= 134,75; Maxf(p)= 3.1oo,8; Mat.oček.=1 o45,56

Tabela 5

Vrsta metrike	VRIJEDNOST DISPERZIJE ZA RAZNE RADIUSE			
	R = 50	R = 10	R = 5	R = 1
LANČANA	513 497,07	430 013,13	441 438,41	0,00
LEKSIKOGRAFSKA	58,49	38,49	32,29	3,30
ALFABETSKA	188 446,86	7 974,26	79,13	0,26
INVERZIONA	541 554,26	600 366,92	765 758,66	496 808,45
TRANSPOZICIONA	487 848,50	220 164,74	144 863,38	96 558,12

LITERATURA:

1. J.Lončar. O jednoj metodi proračuna optimalnog balansa. Informatica br.3 Vol 5. Ljubljana 1981. god.
2. J.Lončar. Primjena inverzione metrike za proračun optimalnog balansa. Informatica broj 3. Vol 5 Ljubljana 1981. god.
3. J.Lončar. Primjena transpozicione metrike za proračun optimalnog debalansa. Informatica br.4 Vol.5 Ljubljana 1982 god.
4. J.Lončar. Primjena leksikografske metrike za proračun debalansa lopatica zrakoplovnih turbina. Informatica br 4 Vol 5 Ljubljana 1982 god.
5. S.Kurepa. Funkcionalna analiza. Elementi teorija operatora. Školska knjiga.Zagreb 1981.
6. Golenko B.I. Statističesnie modeli v upravlenii proizvodstvom. Statistika 1973. god.
7. Ponomarenko L.D. Ob ispolzovanii alfavitnoj metriki v zadačah razmešćenije geometričeskih objektov Kiev 1977.
8. Kaspšickaja M.F. Ob odnom podhode k rešeniju zadać razmešenija. Kibernetika 1974. N° 5

INFORMATICA 1/1982

PREGLED JEZIKOVNIH ELEMENTOV ZA OPIS SINHRONIZACIJE PARALELNIH PROCESOV

MONIKA KAPUS

UDK: 681.3.012

INŠTITUT „JOŽEF STEFAN", LJUBLJANA

Članek podaja pregled razvoja jezikovnih elementov za opis sinhronizacije paralelni procesov. Povdarek je na predstavitvi logičnih osnov paralelnega procesiranja in na opazovanju pojava, da postajajo jeziki za multiprogramiranje vedno bolj homogeni.

The paper is an overview of the development of synchronization primitives. It presents the logical basis of parallel processing and describes the homogenization of multiprogramming languages.

0. Uvod

V začetku je bil osnovni namen paralelnega procesiranja ekonomično izkoriščanje hardvara v skladu s pravilom, da naj proces zaseda nek računalniški vir le v tistih časovnih intervalih, ko ga zares uporablja. Operacijski sistem je moral delovati tako, da so procesi čim manj čutili prisotnost drugih procesov v sistemu. Zato dela programske kode, ki sta se izvajala paralelno, nista smela biti logično povezana.

Najpreprostejša povezava med procesi se je pojavila takrat, ko so se programerji začeli zavedati, da lahko programe, ki so se prej izvajali kot en proces, na nekaterih odsekih razdelijo na paralelne procese in s tem skrajšajo čas obdelave. Taki paralelni procesi so bili logično povezani v tem smislu, da se je izvajanje programa lahko nadaljevalo šele takrat, ko so biti znani rezultati vseh paralelnih procesov, ki so se spet združili v en proces. Pravzaprav ni šlo za medsebojno povezavo paralelnih procesov, temveč za povezavo paralelnih procesov s svojim očetom, ki je med njihovim izvajanjem miroval.

Z razvojem velikih računalniških sistemov se je pogled na paralelno procesiranje spremenil. Predvsem se je večina logičnega načrtovanja prenesla s hardwarskega na programski nivo. Še več! Po splošni sistemski teoriji sta dva sistema ekvivalentna, če dajeta pri enakih vhodnih signalih enak odziv. Zato se je na programski nivo preneslo tudi funkcionalistično pojmovanje sistema. Uveljavilo se je pravilo: ena funkcija-en sistemski blok. Sistemski blok v računalništvu je navidezni (logični) stroj, ki je izveden hardwarsko, programsko ali mešano. Programsko izveden logični stroj predstavlja v sistemu poseben proces, ki je stalno prisoten, tako kot bi bil hardwarsko izveden logični stroj, ali pa ga poženemo takrat, ko ga potrebujemo. Razbitje naloge na funkcijske bloke je posebno primerno za pisanje obsežnih programov, kakršen je operacijski sistem.

1. Logična odvisnost paralelnih procesov

Funkcionalistična opredelitev procesa je najbolj naravna, saj je proces deklariran kot logični objekt-nosilec aktivnosti. Zaradi takega načina dela pa procesi niso več nujno neodvisni. Dva procesa sta medsebojno logično odvisna, če uporabljata skupne računalniške vire, ki so lahko pasivni ali aktivni predmeti, tako da se morata tega zavedati. Ker je vse dogajanje v računalniku v bistvu branje in spreminjanje podatkov, so procesi soodvisni natanko takrat, kadar izvajajo operacije na istih podatkovnih predmetih.

Podatkovne predmete, ki jih obravnava proces, delimo v tri skupine: vhodne spremenljivke, izhodne spremenljivke in spremenljivke stanja. Kot pri vsakem sistemu smemo meje procesa premikati, pri čemer zajamemo med spremenljivke stanja več ali manj njegovih operandov. Ker pa želimo obravnavati proces kot logični stroj, uvrstimo med spremenljivke stanja le tiste spremenljivke, do katerih nima neposrednega dostopa noben drug proces. Vhodne in izhodne spremenljivke procesa so v splošnem tudi vhodne ali/in izhodne spremenljivke drugih procesov.

2. Sinhronizacija procesov

Za pravilno delovanje sistema velja naslednje pravilo:

Ko eden od procesov nastavi vrednost neke spremenljivke, ki naj bi jo brala skupina procesov-čitalcev (lahko tudi proces, ki je vrednost vpisal), smejo vsi procesi to vrednost poljubno spreminjati pod pogojem, da bo imela spremenljivka pravilno vrednost v vseh tistih trenutkih, ko jo bo bral kak proces iz skupine čitalcev.

Vidimo, da imajo procesi skoraj popolno svobodo. Na spremenljivkah smejo izvajati poljubne operacije, ki niso prepovedane s statičnim zaščitnim mehanizmom podatkovnih predmetov, to je s pravili, kateri aktivni predmeti imajo pravico dostopa do nekega pasivnega predmeta in kakšen sme biti ta dostop. Od procesa zahtevamo le to, da spremenljivke po potrebi vrača na stare vrednosti. Omenimo še, da zgodovine spremenljivk ne predstavlja v vsakem primeru ena sama vrednost. Spremenljivka ima navidezno toliko vrednosti, kolikor je procesov, ki so vanjo vpisali neko vrednost z namenom, da ostane konstantna. Lahko bi

ugovarjali, da je tako razmišljanje nepotrebno, saj bi lahko dali vsakemu procesu svojo spremenljivko. Vendar vseh spremenljivk ne moremo enostavno razmnožiti. To velja predvsem za aktivne spominske celice, kakršne so registri. Organizacija sistema, v katerem obstajajo spremenljivke z več navideznimi vrednostmi, je izredno težka, posebno, kadar imamo enakopravne procese.

Problem sinhronizacije procesov zapletajo časovni pogoji. V preprostih sistemih zadošča, da uskladimo hitrosti posameznih procesov kar z vstavljanjem ukazov, ki porabijo določeno število urinih ciklov. Taka sinhronizacija procesov je statična in se ne zna prilagajati nepredvidenim motnjam v sistemu. Splošna rešitev je sinhronizacija procesov z medprocesno komunikacijo, ki je lahko neposredna preko skupnih spremenljivk ali pa posredna preko pomožnega procesa. Pri sinhronizaciji preko skupnih spremenljivk proces, ki prvi doseže sinhronizacijsko točko, v zanki testira sinhronizacijsko spremenljivko, dokler mu drug proces ne pošlje statičnega signala. Komunikacija preko pomožnega procesa je lahko taka, da partnerja pač izvajata operacije nad sinhronizacijsko spremenljivko s posredovanjem pomožnega procesa. Najpogosteje pa je posrednik sistemski proces, ki mu hitrejši proces naroči, naj ga požene, ko bo partner prišel do sinhronizacijske točke. Proces čaka pasivno, kar je zelo ugodno, ker tudi golo testiranje troši računalniški čas. Sporočilo počasnega procesa posredniku, da je dosegel sinhronizacijsko točko, predstavlja kratkotrajen signal, ki požene čakajoči proces, če seveda obstaja. Če je čakanje simetrično, torej če čaka vedno tisti proces, ki je prvi prišel na sinhronizacijsko točko, partner pa vedno pošlje signal, se bosta procesa vedno uspešno srečala (princip rendez-vous-a). Če pa vnaprej predpišemo, kdo pošilja signale in kdo jih čaka, je nevarno, da se kak signal izgubi, kar je usodno, če proces, ki je poslal signal, kar nadaljuje izvajanje. V tem primeru se procesa ne srečata. Isto velja, če proces dinamično signalizira poljuben dogodek, na katerega čaka partner. Pogoji za delo so lahko popolnoma izpolnjeni, pa proces tega ne opazi.

2.1. Problem vzajemnega izključevanja

Osnovni problem sinhronizacije je dostop več procesov do skupnega podatkovnega predmeta. Dovoljeno je paralelno branje, ni pa dovoljeno paralelno pisanje ali paralelno branje in pisanje. Dele programske kode, ki naslavljajo obravnavani predmet, imenujemo z ozirom na ta predmet kritične dele programa. Recimo, da želimo zagotoviti, da bo v vsakem trenutku izvajal svoj kritični del največ en proces. Nalogo imenujemo problem vzajemnega izključevanja. Od rešitve zahtevamo še naslednje lastnosti:

- -simetrijo (procesi ne smejo imeti statične prioritete);
- -neodvisnost od relativnih hitrosti procesov;
- -neobčutljivost na pojav, da se kak proces ustavi izven kritičnega dela;
- -kadar več procesov hkrati zahteva vstop v kritični del, se mora algoritem odločiti za enega od njih v končnem času.

Problem vzajemnega izključevanja ni direktna posplošitev vsakega sinhronizacijskega problema. V nekaterih pogledih so zahteve prestroge (npr. ne dovolimo paralelnega branja), v nekaterih preblage (npr. ne predpisujemo, da moramo vrednost podatka najprej vpisati, preden jo lahko beremo). Algoritem vzajemnega izključevanja je samo orodje, s katerim lahko rešimo katerikoli sinhronizacijski problem. Z vpeljavo dodatnih spremenljivk za medprocesno sinhronizacijo brez težav omejimo vzajemno izključevanje na tiste dele programov, kjer je res potrebno.

Problem vzajemnega izključevanja je prvi rešil danski matematik J.Dekker. Zapišimo algoritem za vzajemno izključevanja poljubnega števila procesov, ki ga je predstavil Dijkstra v [3] v Algolu 60:

```
begin integer array b,c [0:N];
      integer turn;
      for turn := 0 step 1 until N do
          begin b[turn] := 1; c[turn] := 1
end;
      turn := 0;
      parbegin
       process 1: begin.......end;
       process 2: begin.......end;
       .
       .
       process N: begin.......end
      parbegin
     end.

process i: begin integer j;
           Ai: b[i] := 0;
           Li: if turn <> i then
               begin c[i] := 1;
                     if b[turn] = 1
                     then turn := i;
                     goto Li
               end;
               c[i] := 0;
               for j := 1 step 1 until N do
                   begin if j <> i and
                                c[j] = 0
                         then goto Li end;
               kritično področje;
               turn := 0; c[i] := 1;
               b[i] := 1;
               ostanek cikla i; goto Ai
           end.
```

Namesto dokaza, da algoritem zadošča vsem zahtevam, opišimo njegovo delovanje v naravnem jeziku. Zanimivo je, da v algoritmu, ki omogoča medsebojno izključevanje procesov, vsi procesi prosto uporabljajo skupne spremenljivke. Pomisliti pa moramo, da je digitalni računalnik sekvenčni stroj, v katerem se ukazi za pisanje in čitanje po naravi izvajajo zaporedno. Če zahtevamo vpis dveh različnih vrednost a in b v isto spremenljivko, bo vrednost spremenljivke v vsakem trenutku ali a ali b, nikakor pa ne mešanica obeh vrednosti.

Konkurenčni procesi sklepajo na naslednji način:

Označim, da želim vstopiti v kritično področje. Preverim, če sem navrsti. Če nisem, pogledam, ali je tisti, ki je navrsti, končal delo. Če ne dela več, označim, da sem navrsti jaz. Preverim, ali ni morda kak drug proces tudi ugotovil, da prejšnji proces ne dela več in se je poskusil postaviti na začetek vrste. Če me je izrinil, začnem postopek za preboj na čelo vrste od začetka z vljudnim čakanjem, da proces na čelu konča delo. Ko sem na čelu vrste, označim, da vstopam v kritično področje. Spet preverim, ali morda kdo ne dela. To je čisto mogoče. Med trenutkom, ko sem ugotovil, da se imam pravico postaviti na čelo vrste, in med mojim vpisom na čelo vrste je minilo nekaj časa. Medtem je lahko kak posebno hiter proces, ki se prej še ni zanimal za kritično področje, prišel in začel z delom. Če je to res, se mu umaknem in ponovim postopek za preboj na začetek vrste, sicer pa vstopim v kritično področje. Po izstopu označim, da me ktiriično

področje ne zanima več in ustvarim možnosti za enakopraven konkurenčni boj vseh procesov.

2.2. Nedeljivost vzroka in posledice. Semaforji

Iz tega algoritma smo se naučili, da je vir težav naslednje preprosto dejstvo: Proces dobi informacijo o stanju drugih procesov in se odloči, kako bo ukrepal. Toda ko začne ukrep izvajati, nima zagotovila, da stara informacija sploh še velja. Morda je dobil signal že novo vrednost. Problem je v preveliki dinamiki procesov, ki pošiljajo signale.

Vsak signal mora izpolnjevati naslednjo zahtevo:

Signal naj traja dovolj dolgo, da ga procesi, ki jim je namenjen, opazijo in preberejo, kolikorkrat želijo, stanje, ki ga opisuje signal, pa mora trajati, dokler niso uresničeni vsi ukrepi, ki so posledice tega stanja in ne bi več ustrezali, če bi se stanje spremenilo.

To pravilo nedeljivosti vzroka in posledice upošteva Dijkstrova definicija binarnega semaforja. Ker pravilo v celoti zajema bistvo sinhronizacije, lahko rešimo z binarnimi semaforji katerikoli sinhronizacijski problem.

Binarni semafor je poseben primer splošnega semaforja, ki je nenegativna cela spremenljivka, binarni semafor pa lahko zavzame vrednosti 0 in 1. Semafor je statični signal, ki drži vrednost, dokler je eksplicitno ne spremenimo. Zato ni nevarnosti, da procesi ne bi opazili signala. Princip splošnega semaforja je redundanten, ker se da realizirati z binarnim semaforjem in navadno celo spremenljivko. Uvedli so ga zato, ker na eleganten način rešuje problem dodeljevanja enakih virov skupini procesov. Nad semaforjem sta definirani operaciji P in V. Procesi, ki čakajo, mehanizem semaforja uvršča v implicitno vrsto. Čakanje je po definiciji pasivno. Semafor mora zagotoviti enakopravnost procesov in izključiti možnost, da bi kak proces (po semaforjevi krivdi) čakal neskončno dolgo. Iz zadnje zahteve sledi, da nima prednosti proces na začetku vrste, ampak tisti prvi proces v vrsti, pri katerem so izpoljnjeni pogoji za izvedbo operacije. Pri operaciji V navidezno ni čakanja, dejansko pa se tudi na njej odraža sekvenčna narava računalnika. Zapišimo definicijo splošnega semaforja v jeziku Alphard:

```
s: semaphore (začetne vrednost);
   comment semafor je nenegativna cela
           spremenljivka za posebne
           namene, ki ji je pridružena
           implicitna procesna vrsta;
procedure V (s: semaphore) =
begin [s:= s+1;
       če je v vrsti proces,
       ki izpolnjuje potrebne pogoje,
       ga zbudi] end;
procedure P (s: semaphore) =
begin [if s>0
       then s:= s+1
       else begin
            vključi proces v vrsto];
            wait; P(s)
       end
end.
```

Operacije v oklepajih so nedeljive.

Definicija semaforja ne pove nič o njegovi izvedbi. Tudi na področju sinhronizacije procesov težimo k ločitivi deklarativnega in proceduralnega obravnavanja problema. Če zapišemo obsežno proceduro za sinhronizacijo, sicer lahko pravilno deluje, vendar njej namen ni jasno viden in preverjanje pravilnosti je težko. Zato vključujemo ukaze za sinhronizacijo v visoke programske jezike, v katerih opisujemo, kakšen naj bo odnos med procesi in podatkovnimi predmeti, gradnjo potrebnih procedur pa prepustimo prevajalniku.

2.3. Razvoj jezikovnih elementov za sinhronizacijo

Pred definicijo semaforjev je moral programer vedno znova pisati procedure, kakršen je Dekkerjev algoritem. Ukazi, ki bi združevali testiranje in spreminjanje podatkov, praktično niso obstajali. Semafor pomeni prvi podatkovni predmet, ki ni navadna spremenljivka in je namenjena izključno sinhronizaciji.

Po uvedbi semaforjev so si avtorji najprej prizadevali izboljšati sam koncept semaforjev. Operacije na semaforjih so testiranje različnosti od 0, zmanjšanje in povečanje vrednosti. Wodon je leta 1972 namesto P in V operacij uvedel operaciji up in down. Dovolil je kombiniranje pogojev in operacij na več semaforjih npr. Sa.Sb.Sc pomeni, da naj se operacije izvršijo, če so semaforji a,b,c večji od -1 (druga varianta pogoja >0); down Sa,Sq pomeni zmanjšanje, up Sa,Sq pa povečanje semaforjev a in q. Na ta način lahko realiziramo bolj kompleksne nedeljive pare pogojev in posledic (npr. Sa.Sb.: down Sa,Sb). Dijkstrov semafor je le poseben primer (s: down s = P(s); up s = V(s)). Izboljševanje P in V ukazov se je končalo s predlogi bolj sestavljenih pogojev za izvedbo operacij (AND/OR izrazi) in z razmišljanjem o organizaciji implicitne vrste.

Z razvojem teorije programskih jezikov se je pogled na ukaze za sinhronizacijo spreminjal. Prodrlo je spoznanje, da moramo ponuditi programerju taka jezikovna sredstva, ki ga bodo spodbujala k strukturiranemu programiranju. Hoare in Brinch Hansen sta leta 1972 definirala "kritično področje". Novost je v tem, da prevajalniku deklariramo, kateri podatkovni premeti so skupni. Kadar želimo delati s skupnimi podatkovnimi predmeti, to eksplicitno izrazimo in sistem sam poskrbi za vzajemno izključevanje.

```
var v : shared;
region v do operacija na v od
```

Prevajalnik pomaga programerju na ta način, da javi napako, če naslovimo skupne podatke izven kritičnega področja.

Še bolj strukturirana so pogojna kritična področja (prav tako Hoare in Brinch Hansen), ki so naravna razširitev pojma kritičnega področja in vključujejo še čakanje procesa na dogodek.

```
var v shared;
region v when B do operacija na v od
```

Ko proces vstopi v kritično področje, se testira pogoj B. Če je pogoj izpolnjen, se zahtevana operacija izvede, sicer pa proces začasno zapusti kritično področje in se postavi v procesno vrsto spremenljivke v. Vrsta je ena sama za vse procese, ne glede na to, na kakšen pogoj čakajo. Kadarkoli nek proces uspešno zaključi operacije v kritičnem področju, sistem testira pogoje, na katere čakajo procesi v vrsti, in požene enega od procesov, katerih pogoj je izpolnjen. Princip pogojev je ekvivalenten statičnem signalom.

Danes so jezikovni elementi za sinhronizacijo dosegli stopnjo, ko predstalja sinhronizacija le del opisa abstraktnega podatkovnega tipa in programerju o njej sploh ni treba razmišljati, kadar piše proceduro, v kateri naslavlja skupni podatkovni predmet. Abstraktni podatkovni tip združuje opis podatkovne strukture in operacij, ki so na njej definirane. Zunaj dela programa, kjer opišemo podatkovni tip, lahko opazujemo samo učinek definiranih operacij na predmete tega tipa, nikakor pa ne načina izvajanja teh operacij. Lastnosti abstraktnega podatkovnega tipa lahko preverjamo, ne da bi upoštevali uporabo posameznih predmetov tega tipa. Preverjati abstraktni podatkovni tip pomeni preverjati, ali izvedba operacij na podatkovni strukturi ustreza deklaraciji. Tako pojmovanje sinhronizacije omogoča hierarhično strukturiranje obsežnih programov. Iz operacij, ki smo jih definirali in preverili na nižjih nivojih, lahko gradimo operacije na višjih nivojih, ki jih testiramo le glede na deklaracije višjih nivojev.

Najenostavnejši primer podatkovnega tipa z vgrajeno sinhronizacijo so monitorji (Dijkstra, Brinch Hansen, Hoare). Monitor je konstrukt, ki vsebuje lokalne spremenljivke in procedure, ki definirajo operacije nad temi spremenljivkami. Monitorske spremenljivke lahko dosežemo le preko monitorskih procedur. Vsaka monitorska procedura sme imeti tudi svoje lokalne spremenljivke. Monitorske procedure ne smejo naslavljati spremenljivk zunaj monitorja. Vse monitorske procedure se vzajemno izključujejo, le v primeru, da kaka procedura čaka na signal, sme teči druga procedura do takrat, da prva procedura sprejme signal in spet pravzame kontrolo. Navadno monitorske procedure ne sprejemajo zunanjih signalov, ampak čakajo na signal tiste monitorske procedure, ki teče. Signali so dinamični, to pomeni, da se izgubijo, če jih nihče ne čaka. Hkrati sme čakati na signal ene vrste tudi več procedur, po signalu pa steče ena sama in sicer tista z najvišjo prioriteto ali z najdaljšim časom čakanja. Operacije wait so edine operacije, ki se smejo izvajati paralelno z drugimi operacijami. Monitorske procedure ne smejo klicati nestandardnih zunanjih procedur, še manj pa lokalne procedure drugih monitorjev.

Monitorji sicer združijo sinhronizacijo s podatki, na katere se nanaša, vendar mora programer še vedno pisati monitorske procedure s klasičnimi jezikovnimi sredstvi, iz katerih ni razvidno, kaj pravzaprav želi. Kadar uporabljajo procesi skupne spremenljivke, je vzajemno izključevanje pogosto samo tehnično vprašanje, ki le v primeru, da gre za golo dodeljevanje virov, predstavlja bistvo dogajanja. V vseh zahtevnejših aplikacijah je namen skupnih spremenljivk komunikacija med procesi, ki morajo brati in pisati vrednosti spremenljivk v določenem zaporedju. Programer potrebuje jezikovno sredstvo, s katerim bo ta zaporedja opisal. Rešitev sta podala Campbell in Habermann leta 1974 z uvedbo "opisov poti".

Opis poti je izraz, ki podaja vsa dovoljena zaporedja operacij na nekem podatkovnem predmetu. Preproste poti so ciklične, osnovni operatorji v izrazih pa so sekvenciranje, izbira in ponavljanje. Specifikacija podatkovnega tipa ima naslednjo obliko:

```
type ime =
begin
 opis podatkovne strukture
 path opis poti end
 operation opis možnih operacij
end
```

Izraz path f1 (g* + h)i i end na primer opisuje cikel, ki se obvezno začne z eno operacijo f, sledi poljubno število ponovitev operacije g ali ena operacija h, zaključi se z eno operacijo i. Vse operacije se vzajemno izključujejo. To lahko preprečimo z operatorjem paralelnosti ali z definicijo več poti na istem podatkovnem predmetu. Zanimiv element poti je tudi numerični element, ki ima obliko

path (f1-f2-...-fm)n end

No(fi) naj pomeni število ponovitev operacije fi. Z numeričnim elementom dosežemo, da velja:

No(f1)>=No(f2)>=....>=No(fm)>=No(f1)-n ali
D=<No(f1)-No(f2)=<...=<No(f1)-No(fm)=< n

Primer m=2 ilustrira pošiljanje sporočil preko vmesnika z dolžino n.

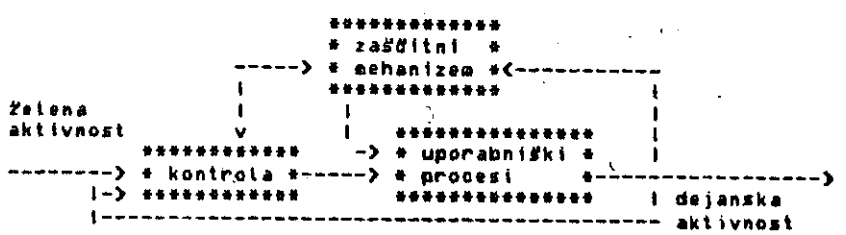
Razvoj jezikovnih elementov za opis sinhronizacije oz. komunikacije med procesi pa s tem še ni dosegel končne stopnje. Elementi naj bi imeli veliko izrazno moč, biti morajo razumljivi in enostavni za uporabo. Predvsem pa morajo biti taki, da jih bo programer uporabljal pravilno, da bo pravilnost brez težav dokazal in da ga bo sistem sam opozarjal na morebitne napake.

Opis sinhronizacijskih zahtev, ki je del opisa podatkovnega predmeta, predstavlja dinamično komponento zaščitnega mehanizma tega predmeta; zato smemo združiti opis sinhronizacijskih zahtev z opisom statičnega zaščitnega mehanizma, ki odgovarja na vprašanje, kateri procesi imajo pravico do neke stalne množice operacij nad danim podatkovnim predmetom.

Poskusimo definirati splošni zaščitni mehanizem podatkovnega predmeta!

Zaščitni mehanizem podatkovnega predmeta je funkcijski blok, ki deluje ravno obratno kot funkcijski bloki uporabniških procesov, ki jih zaščitni mehanizem nadzoruje.

1. Zaščitni mehanizem je proces, ki je aktiven, dokler obstaja podatkovni predmet, katerega ščiti. Uporabniški procesi so navadno vezani samo na obstoj predmeta, ki vsebuje njihovo programsko kodo, in na kontrolne informacije.



2.Uporabniški procesi so bloki, pri katerih kontrolne informacije sprožijo aktivnost, ki jo predpiše uporabnik. Zaščitni mehanizmi so bloki, pri katerih določena aktivnost uporabniških procesov sproži generiranje kontrolnih informacij. Torej ima računalniški sistem povratno zanko.

3.Uporabniški proces posredno ali neposredno izvaja tiste aktivnosti, v katerih (s stališča uporabnikov) nastopa kot subjekt. Zaščitni mehanizem nadzoruje oz. izvaja tiste aktivnosti, v katerih njegov podatkovni predmet nastopa kot objekt.

Zaščitni mehanizem mora predvidevati na svojem vhodu poljubno kombinacijo zahtev za izvedbo operacij nad podatki. V vsakem trenutku se lahko odloči in sproži izvajanje skupine zahtev, odloži skupino zahtev za določen čas ali skupino zahtev zavrne, s tem da generira kontrolne informacije. To pomeni, da smemo v zaščitnem mehanizmu definirati poljubno stopno paralelnosti izvajanja operacij nad podatkovnim predmetom. Zaščitni mehanizem ima tudi svoje spremenljivke stanja, v katerih hrani zgodovino operacij. Ker se vsaka aktivnost računalniškega sistema odraža na vrednosti spremenljivk, za opis odločitvene verige splošnega mehanizma za zaščito podatkov zadošča, da rečemo, da se mehanizem odloča na podlagi vrednosti vseh spremenljivk v sistemu. Pri tem pa moramo vedeti, da upošteva razen vrednosti spremenljivk, ki jih obravnavajo uporabniški procesi, tudi spremenljivke, ki povedo, kateri procesi obstajajo v sistemu, v kakšnem stanju so, kako teče realni čas in predvsem, kdo je sprožil zahtevo za izvajanje neke operacije. Identiteto procesa predstavlja njegovo lastno ime skupaj z imeni vseh prednikov.

Ves čas zanemarjamo važno dejstvo, da pomeni klicanje procedure za operacijo nad podatki komunikacijo med uporabniškim procesom in zaščitnim mehanizmom. Komunikacija poteka seveda preko skupne spremenljivke, ki jo tudi nadzoruje zaščitni mehanizem itd.. Vedno pridemo do komunikacije na nekem nižjem nivoju, za katero smemo reči, da je že zadovoljivo realizirana.

Morda se komu zdi trditev, da je zaščitni mehanizem proces, preozka. Vendar pri tem ne mislimo, da ima zaščitni mehanizem v sistemu nujno enak položaj kot uporabniški procesi. Proces nam pomeni le ime za spreminjanje skupine spremenljivk v odvisnosti od stopenjske spremenljivke - realnega oz. računalniškega časa. Od teh spremenljivk so le nekatere iz uporabniškega področja, torej deklarirane s programom. Zato se, kot smo že rekli, zaščitni mehanizem (vsaj v nekaterih jezikih) ne pojavlja v programu kot deklaracija procesa, temveč - kot del deklaracije pripadajoče podatkovne strukture. Ko pa prevajalnik jezikovni konstrukt realizira, dejstva, da je zaščitni mehanizem proces ali (v primeru, da definiramo paralelne operacije) celo družina procesov, ne moremo zanikati.

Delovanje zaščitnega mehanizma se kaže v dveh fazah:

1.Med prevajanjem programa kot zaščita, da ne zahtevamo operacij, ki nad zaščiteno podatkovno strukturo niso definirane; ali da eksplicitno ne zahtevamo nedefiniranih paralelnosti.

2.Med izvajanjem programa kot zakasnitev ali zavrnitev operacij, ki niso takoj izvedljive zaradi dinamike celotnega sistema, ki iz samega programa ni razvidna, ker prevajalnik normalno ne izvaja kompletnega vrednotenja časovnih razmer (kar bi bilo skoraj nemogoče ali neekonomično).

Če sme zaščitni mehanizem uporabljati le svoje lokalne in globalne spremenljivke, smo pri definiciji bolj kompleksnih operacij prisiljeni združevati več podatkovnih predmetov v enega. Opis dovoljenih zaporedij operacij postane zelo nepregleden, ker zaščitni mehanizem nadzoruje več kot eno funkcijo sistema ali pa se na vmesniku zaščitni mehanizem - uporabniški procesi pojavljajo operacije z različno stopnjo kompleksnosti. To ne pomeni, da prostor podatkovnih predmetov hierarhično širimo, ker predmeti, ki jih združimo, izginejo.

Boljše rezultate dobimo, če zaščitnemu mehanizmu dovolimo uporabo tudi tistih spremenljivk, ki jih ščitijo drugi zaščitni mehanizmi. Pridemo do tega, da se zaščitni mehanizem pojavlja kot enakovreden partner uporabniških procesov na vhodu podatkovnih predmetov. Edina razlika med zaščitnimi mehanizmi in uporabniškimi procesi je ta, da so zaščitni mehanizmi deklarirani kot pasivni procesi, ki tečejo samo takrat, kadar skušajo uporabniški procesi izvesti neko operacijo nad njihovim podatkovnim predmetom. V sistemu je dovoljeno dinamično generiranje procesov. Dovolimo še dinamično generiranje novih podatkovnih predmetov, pa ni več nobene ovire, da ne bi pasivnih in uporabniških procesov obravnavali kot enotno vrsto procesov v sistemu, le da imajo pasivni procesi posebno nalogo. Dovoljeno naj bo tudi spreminjanje in ukinjanje podatkovnih predmetov in njihovih zaščitnih mehanizmov.

Spreminjanje lastnosti prostora podatkovnih predmetov sme zahtevati vsak proces v sistemu, ne glede na to, ali je pasiven ali aktiven (uporabniški). Sistemski program sme izvesti zahtevo po spremembi le v primeru, da so izpolnjeni pogoju zaščite prostora podatkovnih predmetov. Proces, ki zahteva generiranje novega predmeta, mora navesti naslednje podatke:

-katere spremenljivke bo vključil novi podatkovni predmet. Vse spremenljivke morajo biti globalne, torej izven vseh obstoječih od uporabnika definiranih podatkovnih predmetov. Smisel ima tudi to, da predmet ne vsebuje nobenih lokalnih spremenljivk. V tem primeru ga generiramo samo zaradi implementacije bolj kompleksnih operacij nad spremenljivkami v drugih podatkovnih predmetih.

-katere globalne spremenljivke bo nastavljal zaščitni mehanizem novega predmeta;

-katere tuje podatkovne predmete bo nastavljajl zaščitni mehanizem in kakšne operacije bo izvajal na njih;

-katere operacije nad novim podatkovnim predmetom naj sestavljajo vmesnik podatkovni predmet - okolje;

-kakšna so legalna zaporedja operacij nad podatkovnim predmetom in pod kakšnimi pogoji.

Zaščitni mehanizem je proces, ki sme posegati v tuje podatkovne predmete med odločanjem in med izvajanjem definiranih operacij. Tako se zgodi, da procedure enega zaščitnega mehanizma kličejo procedure drugih zaščitnih mehanizmov in operacije na vmesniku z okoljem posredno posegajo tudi v druge podatkovne predmete. Zato niti ni več nujno, da je legalno zaporedje operacij nad enim podatkovnim predmetom popolno v tem smislu, da bi zaščitni sistem reševal vse probleme sinhronizacije procesov - uporabnikov na predmetu, ki ga nadzoruje. Če na tistih odsekih zaporedij, ki še niso popolni, ne dovolimo direktnega dostopa uporabnikov in vključimo pred vhod predmeta drug podatkovni predmet oz. njegov

zaščitni mehanizem, se uporabnikove zahteve večkrat filtrirajo. Tako dosežemo, da zahteve, ki so na nekem vhodu prepovedane in jih vhod ne zna izločiti, filtrira filter pred tem vhodom in prepovedane zahteve (npr. interferirajoče paralelne operacije) se nikoli ne pojavijo. Programerju ni več treba direktno graditi kompleksnih podatkovnih struktur in zaščitnih mehanizmov, gradi jih hierarhično od spodaj navzgor ali celo rekurzivno iz bolj enostavnih gradnikov.

Zaščitni mehanizem izvaja definirane operacije kot podprograme ali kot posebne procese. Zato za vsako spremembo obstoječih podatkovnih predmetov veljajo pravila za spreminjanje aktivnih podprogramov oz. procesov, ki so morda očetje ali sinovi drugih procesov in podprogramov.

Poseben jezik za opis legalnih zaporedij operacij, kakršnega predstavljajo znani izrazi za opis poti, je odveč, ker se pri velikih zaščitnih mehanizmih razraste v prav tako kompleksnost, ki je značilna za programske jezike, s katerimi opisujemo uporabniške procese. Zato naj vse dogajanje v sistemu opisuje enoten jezik po zgledu jezika ADA. Če obravnavamo zaščitne mehanizme kot posebnosti, ki jih šele pri prevajanju realiziramo z normalno programsko kodo, potrebujemo tudi posebne tehnike za preverjanje pravilnosti delovanja paralelnih procesov.

Naravo objekta za kontrolo sinhronizacije karakterizira invariantna relacija med spremenljivkami ki je izpolnjena vedno, kadar element ni aktiven. To relacijo uporabljamo skupaj z drugimi relacijami v sistemu za formalno dokazovanje pravilnosti algoritmov. Aktivnost objekta za sinhronizacijo precizno opišemo še s tem, da navedemo pogoje, ki morajo biti izpolnjeni, preden se lahko izvede dana operacija. Za objekt posebne vrste, ki ga nanovo vpeljemo v jezik, moramo odkriti njegovo invariantno relacijo, kar ni zmeraj enostavno. Res pa je, da pozneje invariantne relacije ni treba ponovno iskati, medtem ko moramo objekte za kontrolo sinhronizacije, ki niso tipizirani, opisovati sproti. Vendar to ni problem, če je objekt za kontrolo sinhronizacije zgrajen po enakih zakonih, kot ostali objekti v sistemu. S tem se ne ravna po standardu za posebno področje sinhronizacije, ampak po standardu za celoten sistem. Podrobna zgradba objekta za sinhronizacijo ni vnaprej znana, znane pa so metode za konstruiranje objekta, ki so povsem splošne.

Zaščitni mehanizem v modernem programskem jeziku je proces kot vsak drug, ima skupino lokalnih podatkov, ki jih ščiti, in skupino procedur za dostop do podatkov, ki jih drugi procesi kličejo preko procesnih vhodov. Razen podatkov z dinamično zaščito pa še vedno obstajajo podatki s statično zaščito, pri katerih predpisuje in nadzira deljivost ali nedeljivost operacij implicitni sistemski proces. Tako sploh ne moremo več govoriti o direktnem dostopu do spremenljivk in smemo znotraj računalniškega sistema vse spremenljivke, ki jih uporabljamo, vključiti v različne funkcijske bloke. Vsa sinhronizacija se pravzaprav odvija implicitno s sistemsko kontrolo procesnih vhodov po principu rendez-vous-a, klicani ali kličoči proces avtomatično čakata drug na drugega v določenih točkah algoritma.

Zaščitni mehanizem, ki je implementiran kot normalen proces, lahko predstavlja aktivni filter vhodnih klicev. Vse do sedaj omenjene vrste zaščitnih mehanizmov so imele edino funkcijo, da zahteve za izvedbo operacije ali sprejmejo ali zavrnejo ali zakasnijo. Ko je mehanizem neko zahtevo sprejel, je operacijo vedno izvedel na enak način, neodvisno od stanja sistema. Seveda bi v procedure, ki opisujejo izvajanje operacij, lahko vključili upoštevanje stanja sistema, vendar je bil osnovni namen ta, da uporabnikova zahteva kadarkoli sproži isto proceduro. Zaščitni mehanizem je bil izključno pasiven filter. Zaščitni mehanizem, ki je normalen proces, sprejema enake zahteve na več različnih procesnih vhodih, v odvisnosti od tega, do katerega vhoda je pritekel algoritem zaščitnega mehanizma. Vsak procesni vhod predstavlja stavek

when klic sprejet do akcija

in vsak procesni vhod ima svojo lastno posledico klica. Zaščitni mehanizem ni več pasivni filter, ki spušča uporabnikove klice lokalnih procedur v notranjost, temveč te klice aktivno preoblikuje. Prav tako je lahko zaščita lokalnih spremenljivk samo stranska naloga procesa. Aktivni filter ni samo nadzornik uporabnikove aktivnosti, ampak servisni proces.

Razvoj jezikovnih elementov za sinhronizacijo oz. komunikacijo med paralelnimi procesi je s tem dosegel tisto točko, ko se obravnavano posebno področje vključi v jezik na tako popoln način, da neha biti posebnost in ga ni več treba principielno raziskovati z jezikovnega stališča. Oblikovalci jezikov za paralelno programiranje se ukvarjajo v glavnem s podrobnostimi izvedbe in manjšimi spremembami jezikovnih konstruktov, ki ne vplivajo na bistvo rešitve.

Oglejmo si še problem sinhronizacije procesov, ki jih implicitno sprožajo programi v jezikih za nepostopkovno programiranje. Tudi nepostopkovne programe sestavljajo zaporedja ukazov in sicer za vnašanje novih dejstev v bazo znanja, za spreminjanje starega znanja ter za iskanje podatkov in izpeljevanje logičnih sklepov na osnovi znanja v bazi. Sistem pogosto realiziramo kot pomnilnik brez vnaprej deklarirane logične strukture, kjer so podatki spravljeni v obliki sintaktičnih konstruktov. Osnovni proces v sistemu je administrator baze, ki sprejema ukaze uporabnikov in za procesiranje vsakega ukaza sproži poseben proces. Procesi posegajo v bazo znanja kot pisci ali čitalci. Zaščitni mehanizem baze dodeli vsakemu piscu del praznega pomnilniškega prostora ali obstoječi sintaktični konstrukt, ki ga proces želi spremeniti. Proces - pisec je izključni lastnik dodeljenega objekta, dokler ne konča dela. Procesi smejo prosto brati vse konstrukte v bazi znanja, ki niso last procesov - piscev, ker samo ti konstrukti predstavljajo veljavno znanje. Čitalci se obračajo na zaščitni mehanizem baze, kadar iščejo v bazi konstrukte z dano strukturo. Zaščitni mehanizem mora sinhronizirati procese tako, da ne dovoli spreminjanja tistih konstruktov, ki jih že bere kak proces, in branja tistih konstruktov, ki jih kak proces že spreminja. Imamo klasičen primer piscev in čitalcev. Ukrepi procesov, ki uporabljajo znanje iz baze, so posledice prebranega znanja. Zato zahtevamo tudi tu določeno stopnjo nedeljivosti vzroka in posledice. Baza znanja naj bi bila slika nekega sveta. Ne glede na to, ali je ta svet realen ali abstrakten, smemo reči, da se celotno znanje spreminja v odvisnosti od stopenjske spremenljivke in logika sveta predpisuje, da se morajo nekateri elementi znanja spremeniti sočasno, to pomeni, da mora staro znanje v

trenutku preiti v novo. Zaradi vsaj delno sekvenčnega delovanja računalnika pa to ni res in obstaja nevarnost, da bo uporabnik brat kombinacije elementov znanja, ki se v svetu, ki ga opisuje baza znanja, ne bi pojavile. Zato je važna sinhronizacijska naloga zaščitnega mehanizma baze, da simulira realno spreminjanje sveta in prepreči dostop uporabnikov baze do podatkov, ki se spreminjajo, med trajanjem prehodnega pojava. Pri tem se naloga ne ujema popolnoma z blokiranjem čitalcev, enega sintaktičnega konstrukta med njegovim spreminjanjem. Obstajati mora ukaz za blokado poljubne skupine sintaktičnih konstruktov, dokler vsi ne dobijo nove vrednosti.Procesi - uporabniki baze znanja morajo za smiselno upoštevanje blokade ločiti med primeroma, da je neka izjava v bazi zanikana ali da v bazi ne obstaja niti trditna niti nikalna oblika izjave. Če izjave ni v bazi, mora proces avtomatično ponovno preiskovati bazo do preteka dovoljenega časa iskanja in šele nato podati uporabniku končno poročilo. Označevanje sintaktičnih konstruktov, ki naj se navidezno spremenijo istočasno, in njihovo vključevanje v bazo po spremembi je nedeljiva operacija, med katero procesi lahko spreminjajo konstrukte, ki so jih že prej rezervirali kot pisci, ni pa dovoljeno branje že označenih konstruktov. Sinhronizacijske potrebe v sistemu še narastejo, če en ukaz uporabnika sproži izvajanje večih procesov.

Kot vidimo, se kaže postopkovnost nepostopkovnih programskih jezikov kot komunikacija sistema in uporabnika, implicitno pa tudi med izvajanjem programov. Postopkovni in nepostopkovni jeziki se po sintaksi bistveno razlikujejo. S postopkovnimi eksplicitno opisujemo procese v računalniku, z nepostopkovnimi samo deklariramo relacije med podatki in prepustimo gradnjo procedur in organizacijo procesov za iskanje odgovorov na naša vprašanja sistemu. Med samim izvajanjem so problemi komunikacije in sinhronizacije v obeh primerih enako pomembni in se rešujejo z enakimi pristopi, le da se pri nepostopkovnem programiranju ne odražajo na jeziku in ker procese organizira sistem, so procesni sklopi preprostejši in manj raznoliki.

3. Sklep

Paralelno procesiranje se podreja majhni skupini temeljnih zakonitosti, ki jih brez težav odkrijemo in opišemo. Kljub temu, da je bil razvoj jezikovnih elementov za opis sinhronizacije paralelnih procesov precej počasen, je danes dosegel zadovoljivo stopnjo. Zato se je težišče dela preneslo na optimiranje programov v multiprocesnem okolju ob podpori visokih programskih jezikov.

4. Viri

N.Wirth
Toward a Discipline if Real-Time Programming,
Comm. ACM 20 (1977), No.8, 577-583

N.Wirth
MODULA: A Language for Modular Multiprogramming,
Software Practice and Experience 7 (1977), 3-35

E.W.Dijkstra
Co-operating Sequential Processes,
Programming Languages,
ed. F.Genuys, Academic Press, 1968

C.A.R. Hoare
Monitors: An Operating System Structuring Concept,
Comm. ACM 17 (1974), 10, 549 - 557

S. Andler
Synchronization Primitives and the Verification
of Concurrent Programs,
Carnegie-Mellon University

M.Exel, F.Prijatelj
Programiranje sprotnih in vgnezdenih sistemov:
Procesi v ADI.
Informatica 1981/2

NAVODILO ZA PRIPRAVO ČLANKA

Avtorje prosimo, da pošljejo uredništvu naslov in kratek povzetek članka ter navedejo približen obseg članka (število strani A 4 formata). Uredništvo bo nato poslalo avtorjem ustrezno število formularjev z navodilom.

Članek tipkajte na priložene dvokolonske formularje. Če potrebujete dodatne formularje, lahko uporabite bel papir istih dimenzij. Pri tem pa se morate držati predpisanega formata, vendar pa ga ne vrišite na papir.

Bodite natančni pri tipkanju in temeljiti pri korigiranju. Vaš članek bo s foto postopkom pomanjšan in pripravljen za tisk brez kakršnihkoli dodatnik korektur.

Uporabljajte kvaliteten pisalni stroj. Če le tekst dopušča uporabljajte enojni presledek. Črni trak je obvezen.

Članek tipkajte v prostor obrobljen z modrimi črtami. Tipkajte do črt - ne preko njih. Odstavek ločite z dvojnim presledkom in brez zamikanja prve vrstice novega odstavka.

Prva stran članka:

a) v sredino zgornjega okvira na prvi strani napišite naslov članka z velikimi črkami;
b) v sredino pod naslov članka napišite imena avtorjev, ime podjetja, mesto, državo;
c) na označenem mestu čez oba stolpca napišite povzetek članka v jeziku, v katerem je napisan članek. Povzetek naj ne bo daljši od 10 vrst.
d) če članek ni v angleščini, ampak v katerem od jugoslovanskih jezikov izpustite 2 cm in napišite povzetek tudi v angleščini. Pred povzetkom napišite angleški naslov članka z velikimi črkami. Povzetek naj ne bo daljši od 10 vrst. Če je članek v tujem jeziku napišite povzetek tudi v enem od jugoslovanskih jezikov;
e) izpustite 2 cm in pričnite v levo kolono pisati članek.

Druga in naslednje strani članka:
Kot je označeno na formularju začnite tipkati tekst druge in naslednjih strani v zgornjem levem kotu,

Naslovi poglavij:
naslove ločuje od ostalega teksta dvojni presledek.

Če nekaterih znakov ne morete vpisati s strojem jih čitljivo vpišite s črnim črnilom ali svinčnikom. Ne uporabljajte modrega črnila, ker se z njim napisani znaki ne bodo preslikali.

Ilustracije morajo biti ostre, jasne in črno bele. Če jih vključite v tekst, se morajo skladati s predpisanim formatom. Lahko pa jih vstavite tudi na konec članka, vendar morajo v tem primeru ostati v mejah skupnega dvokolonskega formata. Vse ilustracije morate (nalepiti) vstavitl sami na ustrezno mesto.

Napake pri tipkanju se lahko popravljajo s korekcijsko folijo ali belim tušem. Napačne besede, stavke ali odstavke pa lahko ponovno natipkate na neprozoren papir in ga pazljivo nalepite na mesto napake.

V zgornjem desnem kotu izven modro označenega roka oštevilčite strani članka s svinčnikom, tako da jih je mogoče zbrisati.

Časopis INFORMATICA
Uredništvo, Parmova 41, 61000 Ljubljana

Naročam se na časopis INFORMATICA. Predplačilo bom izvršil po prejemu vaše pozanice.

Cenik: letna naročnina za delovne organizacije 500,00 din, za posameznika 200,00/100,00/50,00 din

Časopis mi pošiljajte na naslov ☐ stanovanja ☐ delovne organizacije.

Priimek.....

Ime.....

Naslov stanovanja

Ulica.....

Poštna številka_____ Kraj.....

Naslov delovne organizacije

Delovna organizacija.....

.....

Ulica.....

Poštna številka_____ Kraj.....

Datum..... Podpis:

.....

INSTRUCTIONS FOR PREPARATION OF A MANUSCRIPT

Authors are invited to send in the address and short summary of their articles and indicate the opproximate size of their contributions (in terms of A 4 paper). Subsequently they will receive the autor's kits.

Type your manuscript on the enclosed two-column-format manuscript paper. If you require additional manuscript paper you can use similar-size white paper and keep the proposed format but in that case please do not draw the format limits on the paper.

Be accurate in your typing and through in your proof reading. This manuscript will be photographically reduced for reproduction without any proof reading or corrections before printing.

Use a good typewriter. If the text allows it, use single spacing. Use a black ribbon only.

Keep your copy within the blue margin lines on the paper, typing to the lines, but not beyond them. Double space between paragraphs.

First page manuscript:
a) Give title of the paper in the upper box on the first page. Use block letters.
b) Under the title give author's names, company name, city and state - all centered.
c) As it is marked, begin the abstract of the paper. Type over both the columns. The abstract should be written in the language of the paper and should not excesed 10 lines.
d) If the paper is not in English, drop 2 cm after having written the abstract in the language of the paper and write the abstract in English as well. In front of the abstract put the English title of the paper. Use block letters for the title. The lenght of the abstract should not be greater than 10 lines.
e) Drop 2 cm and begin the text of the paper in the left column.

Second and succeeding pages of the manuscript:
As it is marked on the paper, begin the text of the second and succeeding pages in the left upper corner.

Format of the subject headings:
Headings are separated from text by double spacing.

If some characters are not available on your typewriter write them legibly in black ink or with a pencil. Do not use blue ink, because it shows poorly.

Illustrations must be black and white, sharp and clear. If you incorporate your illustrations into the text keep the proposed format. Illustration can also be placed at the end of all text material provided, however, that they are kept within the margin lines of the full size two-column format. All illustrations must be placed into appropriate positons in the text by the author.

Typing errors may be corrected by using white correction paint or by retyping the word, sentence or paragraph on a piece of apaque, white paper and pasting it nearly over errors

Use pencil to number each page on the upper-right-hand corner of the manuscript, outside the blue margin lines so that the numbers may be erased.

INFORMATICA, Journal Headquarters
Parmova 41, 61000 Ljubljana, Yugoslavia

Please enter my subscription to INFORMATICA and send me the bill.

Annual subscription price: companies US $ 22, individuals US $ 7,5.

Send journal to my ☐ home address ☐ company's address.

Surname.....

Name.....

Home address

Street.....

Postal code _____ City.....

Company address

Company.....

.....

Street.....

Postal code _____ City.....

Date..... Signature

.....

ENKRIPCIJA S POMOČJO FUNKCIJE XOR

SMOLEJ VITOMIR

UDK: 681.3.06:003.6

VISOKA ŠOLA ZA ORGANIZACIJO DELA, KRANJ
UNIVERZA V MARIBORU

POVZETEK: logična funkcija xor (ekskluzivni or) se vse češče uporablja kot jedro (enostavnejših) algoritmov za enkripcijo. V članku se pojasni lastnosti te funkcije, zaradi katerih je primerna za te namene, in pokaže nekaj načinov vgradnje v enostavne programe za enkripcijo in dekripcijo informacije.

ENCRYPTION WITH THE HELP OF XOR FUNCTION: logical function xor (exclusive or) is being more and more frequently used for encryption. Its basic properties are discussed, particularly in connection with its use for encryption purposes, and several ways of incorporating it in simple data encryption and decryption programs are shown.

UVOD

Zasebnost podatkov v računalniku si lahko (med drugim) zagotovimo tudi z enkripcijo, to je, da predstavimo podatke na način, ki ga ni mogoče razumeti brez inverznega procesa, to je dekripcije. Res da nam večina računalniških sistemov na en ali drug način omogoča, da se zavarujemo pred možnostjo nepooblaščenega dostopa do podatkov - recimo s sistemom gesel za dostop, čitanje, pisanje in popravljanje vsebine -i Vsak sistem pa ima v svojih okopih luknje , za katere prvič niti ne vemo, drugič pa jih tudi ne bi mogli zapolniti (tipičen primer bi bil sistemski programer, ki smo se mu zamerili). S pomočjo enkripcije se lahko še dodatno zavarujemo pred nepoklicanimi očmi, ki jih zanimajo bodisi naši programi, bodisi naši podatki.

PROGRAM ZA ENKRIPCIJO

si lahko zamišljamo kot filter, ki sesa vase niz vhodnih znakov, jih pretvori in daje od sebe spremenjen, zakodiran niz znakov. Kako se bo znak na vhodu, spremenil v izhodni znak, nam določa niz znakov, ki ga bomo (seveda) imenovali ključ. Ob pomoči funkcije getc , ki v argumentu vrača prečitani znak , in subrutine putc, ki znak v argumentu izpiše na izhodno datoteko, pridemo do naslednjega programa v fortranu:

```
      CHARACTER xor, c, kljuc(10)
      integer mod, i, getc
c
c...  prečitaj ključ, dolg deset znakov
c
      accept 1, (kljuc(i),i=1,10)
      i = 1
100   if (getc(c).eq.-1) stop
c...  znak c zakodiraj z znakom kljuc(i)
c...  in izpisi - s pomočjo putc -
c
        call putc ( xor(c,kljuc(i)) )
        i = mod(i,10) + 1
        go to 100

      end
```

Funkcija getc nam vrne v primeru konca podatkov vrednost -1, drugače pa (recimo) vrednost 0. Beseda CHARACTER nam deklarira spremenljivke in funkcije znakovnega tipa: v primeru fortrana na sistemu DELTA pišemo namesto CHARACTER LOGICAL*1. Mod je standardna fortranska funkcija celega tipa: vrne nam ostanek pri celem deljenju števila i z j.V kolikor ni vgrajena, si lahko pomagamo z enovrstično funkcijo mod(i,j) = (i/j)*j - i. Za dolžino ključa smo zaradi enostavnosti privzeli vrednost 10. Gibčejšo rešitev dobimo z uporabo subrutine GETKEY(a,b), ki nam vrne v enem argumentu niz znakov ključa, v drugem pa dolžino ključa. Seveda moramo potem v programu na ustreznih mestih konstanto 10 zamenjati s spremenljivko, ki opisuje dolžino ključa.

EKSKLUZIVNI OR

Kako uporabljamo znake v ključu? V našem primeru v množici funkcij, ki iz dveh znakov naredita tretjega, uporabljamo funkcijo ekskluzivni or. Tabelarično definiramo ekskluzivni or (označimo ga z ⩛) na naslednji način :

a	b	a ⩛ b
1	1	0
1	0	1
0	1	1
0	0	0

Trditev a V b bo res samo v primeru, če imata a in b različni logični vrednosti. Ko govorimo o ekskluzivni or operaciji nad znaki, mislimo pri tem na operacijo, ki jo izvedemo med ustreznimi biti interne predstavitve teh dveh znakov. Vzemimo kar znaka A (veliki) in B (veliki) v 7-bitni ASCII kodi:

binarno	oktalno	znak
1000001	101	A
1000010	102	B
0000011	003	ETX

V kolikor ni direktno na razpolago funkcija xor, si lahko pomagamo z definicijo xor s pomočjo bolj pogostih operacij, kot so negacija, logično seštevanje in množenje:

$$a \vee b = (a \wedge \neg b) \vee (\neg a \wedge b) \qquad /1/$$

Kot smo dejali, predstavlja xor samo eno od možnosti za določitev izhodnega znaka na osnovi znaka na vhodu in znaka v ključu. Kaj je tako simpatičnega na tej operaciji?

Oglejmo si najprej nekaj preprostih resnic v zvezi z xor. Iz tabele xor in iz /1/ lahko preverimo, da veljajo naslednje enačbe:

$$a \veebar b = b \veebar a \qquad /2a/$$

$$a \veebar a = \emptyset \qquad /2b/$$

$$a \veebar \emptyset = a \qquad /2c/$$

Zamislimo si sedaj, da smo znak b zakodirali z znakom a in izpisali kot znak c na vhod:

$$a \veebar b \longrightarrow c$$

Kako bomo iz znaka c dobili nazaj prvotni znak b? Odgovor je zelo preprost in eleganten: s ponovno enkripcijo z znakom a. Dokaz sledi:

$$a \veebar c = a \veebar a \veebar b \longrightarrow$$

$$\emptyset \veebar b \text{ (zaradi 2b)} \longrightarrow b \text{ (zaradi 2c in 2a)}$$

Za dekripcijo teksta, ki smo zakodirali s pomočjo ekskluzivne or funkcije, uporabimo isti program in isti ključ. Filter za enkripcijo in dekripcijo sta torej identična.

To tudi pomeni, da lahko zlahka dvakrat zakodiramo eno in isto informacijo. Naj bo prvi ključ HEJ, drugi pa BRIGADE. Informacijo si lahko povrnemo z uporabo obeh ključev (v poljubnem vrstnem redu zaradi simetrije funkcije xor). Končni rezultat enkripcije je, kot da bi zakodirali informacijo s ključem, katerega dolžina je najmanjši skupni delitelj dolžin obeh ključev, v našem primeru 3 x 7 = 21 znakov. Teh 21 znakov pa predstavlja ključ, ki je daleč od besed Hej in brigade in ga praktično (če nismo profesionalen kriptolog) ne moremo razbiti.

IMPLEMENTACIJE FUNKCIJE XOR

Tu predlagam dve možnosti implementacije xor funkcije, in sicer najprej za DELTA sistem (to smo uporabljali na VSOD) , potem pa še predlog za CDC cyber Republiškega računskega centra.

DELTA: namesto funkcije XOR smo definirali subrutino XOR, ki je imela tri parametre (tretji parameter predstavlja ekskluzivni or prvih dveh, vsi trije so znaki). Subrutina XOR je napisana s pomočjo zbirnika MACRO:

```
        .TITLE  XOR
XOR::   INC     R5
        INC     R5
        CLR     R1
        BISB    @(R5)+,R1
        CLR     R2
        BISB    @(R5)+,R2
        XOR     R1,R2
        MOVB    R2,@(R5)
        RETURN
        .END
```

Ker pozna Fortran na CDC Cyber ekskluzivni or kot lastno (intrinsic) funkcijo, lahko program za enkripcijo prepišemo takega kot je brez sprememb. Da pa ne bi zapravljali prostora in časa, lahko pospešimo proces tako, da hkrati obdela xor funkcija deset znakov, kolikor jih gre v eno besedo računalnika. Če si program še poenostavimo in namesto primitivov getc in putc uporabimo enostavno čitanje in pisanje datotek dolžine 80 znakov, dobimo naslednji program za enkripcijo:

```
      PROGRAM CRYPT
      DIMENSION VHOD(8),IZHOD(8)
      DATA KLJUC/10HSEZAMZAPRI/
      I = 1
100   READ 1,VHOD
1     FORMAT (8A10)
      IF (EOF(3)) GO TO 999
        DO 2 I=1,8
        IZHOD(I) = XOR(KLJUC,VHOD(I))
      2 CONTINUE
        PRINT 1,IZHOD
        GO TO 100
999   CONTINUE
      STOP
      END
```

ZAKLJUČEK

Poudariti moramo, da obstajajo daleko močnejše in efektnejše metode enkripcije, kot je predstavljena, ki je (poleg cezarjevega algoritma) ena najstarejših in najbolj znanih. Ne glede na to, da se v vsakem centru najde kaka bistra glava, ki se bo z veseljem lotila razbijanja ključa, bo v večini primerov zakodirana informacija na varnem, v kolikor pač slučajno ne nakracamo ključa s svinčnikom na terminal, da ga ne bi pozabili.

Še nasvet osovraženemu sistemskemu programerju: v kolikor vam vaši sovražniki v centru skrivajo informacijo na predlagani način, uporabite naslednji pristop:

1. poiščite program za enkripcijo
2. poiščite zakodirano informacijo
3. zakodirajte podatke s pomočjo programa in ključa, ki ga seveda poznate le vi

NOVICE IN ZANIMIVOSTI

****
Računalniški model simulira
obnašanje možganskih celic
pri epileptičnih napadih
****

Simulacijska tehnika je učinkovit pripomoček pri analizi izmenjevalnih odnosov (relacij, presnove) med stotinami živčnih celic v možganih. Takšne raziskave ni mogoče opraviti neposredno v možganskem tkivu, ker zahtevajo prave (realne) meritve vsakič po en par (dvojico) celic. Računalniška simulacija pa omogoča, da se raziskujejo dogodki, ki zajamejo stotine ali tisoče določenih živčnih celic.

Epileptični napadi se sprožajo zaradi nenadnih, nenormalnih procesov električnega praznenja v živčnih celicah. Pri tem razlikujemo dve vrsti napadov: fokalne, ki so lokalizirani v enem delu (središču) možganov in generalizirane, ki zajamejo celotne možgane in sprožijo t.im. "grand mal" napad. S simulacijo so bile raziskane spremembe živčnih celic, ki se pojavljajo pred fokalnim (žariščnim) tipom napada.

V normalnem stanju izkazujejo vse živčne celice določeno električno dejavnost, ki je posledica različnih ionskih porazdelitev (npr. kalijevih in natrijevih ionov) v notranjem in zunanjem okolju celic. Nastajajoče potencialne razlike se lahko merijo z namestitvijo tankih elektrod v tkivo. Ioni so izvir zapletenih elektrokemičnih izmenjevalnih odnosov v mreži (povezani množici) živčnih celic.

Znanstveniki so preučevali en del možganov, kjer se v živčnih celicah navadno v vsaki sekundi sproži praznitev, toda ne po nekem vnaprej določenem vzorcu. Tipičen dogodek pred mogočim začetkom epileptičnega napada fokalnega tipa je, da se stotine celic začnejo prazniti skorajda sinhrono in da postane ta dogodek v zapisu možganskih valov (EEG) viden kot nenadna "interiktalna" konica. V možganih se tako pojavi neke vrste fiziološka nevihta. Iz takih sinhronih praznenj je moč sklepati, da je določen del možganov nenormalen in da se lahko pri ustreznih pogojih sproži epileptični napad. Doslej še ni znano, kakšni so pri tem robni pogoji oziroma kaj pogojuje praznitev v živčni celici, da ta steče nenadoma po sinhronem vzorcu. Za odgovor na to vprašanje so potrebne nove laboratorijske raziskave in simulacija.

Za simulacijo sinhrone praznitve je bil razvit računalniški model, ki ponazarja samostojno praznenje v živčni celici in model za praznenje v mreži celic. Pri tem se je pokazalo, da je bilo težavneje oblikovati osnovni model (za eno celico) z mehanizmom praznenja in lažje celični mrežni model. Vzrok za to težavnost je v tem, ker vrsta procesov v možganski celici, ki so povezani z praznenjem, ni dovolj znanih in raziskanih.

Računalniška simulacija električne poti skozi celico zahteva takšno ekstrapolacijo prvotno razvite teorije, da je moč pojasniti električno aktivnost aksonov, tj. povezovalnih vlaken med živčnimi celicami. Dodatno k aksonski teoriji so bile izvedene nadaljne predpostavke o notranjem obnašanju celice. Te predpostavke so privzete iz znanja o živčnih celicah v možganih nevretenčarjev, kot so npr. polži.

Razvite enačbe, ki so bile vstavljene v veliki računalnik raziskovalnega centra podjetja IBM, imajo 44 primarnih spremenljivk, ki so v medsebojnih odnosih (odvisnostih).

Rezultati računalniške simulacije kažejo, da praznenje ene celice lahko povzroči praznenje druge celice in pri tem nastane sinhroni vzorec verižne reakcije. Nadalje je z uporabo modela mogoče oblikovati sklepe o drugačnih potekih verižnih reakcij in o tem, kaj pripelje k reakciji, podobni epileptičnemu napadu. Navadno se vsaka celica sama izklopi in tako prekine verižno reakcijo. Vendar obstajajo neodvisni eksperimentalni dokazi, na osnovi katerih sklepamo, da je mogoče zavirati izklopni mehanizem živčnih celic, ki ostajajo tako v stanju vzburjenja in s tem povzročijo napad. Računalniška simulacija pokaže, da se s povečanjem števila kalijevih ionov v možganski tekočini lahko doseže stanje stalne vzburjenosti celic.

Znanstvenika Roger Taub in K.S.Wong sta poročala o svojih raziskavah v okviru "11-th Annual Meeting of the Society for Neuroscience" v Los Angelosu oktobra 1981.

A.P.Železnikar

TEORIJA KORPUSA

Od 16. do 19. decembra 1981. godine održan je u Beogradu seminar "Teorija korpusa", u organizaciji Matematičkog instituta i Filološkog fakulteta. Na seminaru su govorili prof. H.Cimerman (Univ.Sarland), dr. V.Tojbert (pom.dir.IDS u Manhajmu) i G.Frakenpol (IDS,Manhajm). Seminar je organizovan uz pomoć KIC SRN.

Dr Tojbert je izložio probleme u stvaranju korpusa (reprezentativnog uzorka) jednog prirodnog jezika, kao i različite pristupe u njihovom rešavanju. Tako su razmotreni kriterijumi pri izboru tekstova za korpus, pitanje reprezentativnosti korpusa, kao i načini obeležavanja i kodiranja tekstova u korpusu.
S druge strane, izloženi su i različiti aspekti upotrebe korpusa, kako u lingvističkim istraživanjima (npr. leksikografsko-gramatičke banke podataka, konstruisanje rečnika), tako i u računarskim (npr. prirodno-jezički interfejsi za komunikaciju sa informacionim sistemom, itd.) Posebno je prikazan sistem MOLEX/PLIDIS, koji prevodi ulazni tekst na prirodnom jeziku u internu reprezentaciju zasnovanu na predikatskom računu na osnovu koje se vrše dalja pretraživanja u informacionom sistemu. Komponenta MOLEX je generator morfološkog leksikona a deo sistema PLIDIS (PASS1) se može koristiti za morfo-sintaksičku analizu specijalizovanih korpusa.

Prof. Cimerman je obradio, pre svega, problem konstruisanja i upotrebe korpusa strukovnih jezika, ilustrujući ga primerom sistema JUDO za automatsko indeksiranje i pronalaženje pravnih dokumenata prema sadržaju. Obradjeni su posebno problemi morfološke analize, automatske lematizacije i sintaksičke i semantičke višeznačnosti.

D.Vitas

SREČANJA

UNIVERSITY COMPUTING CENTRE, ZAGREB
SVEUČILIŠNI RAČUNSKI CENTAR, ZAGREB

Prva obavijest

First notice

IV MEĐUNARODNI SIMPOZIJ »KOMPJUTER NA SVEUČILIŠTU«

IV INTERNATIONAL SYMPOSIUM »COMPUTER AT THE UNIVERSITY«

Cavtat, 24—28. svibnja 1982.

Cavtat, 24th — 28th May, 1982.

Gdje:
Where: **Cavtat (Dubrovnik), hotel CROATIA**

Kada:
When: **24—28. 5. 1982.**

Rokovi
Deadlines

Preliminarne prijave:
Preliminary registration: **15. 11. 1981.**

Kompletni radovi:
Manuscripts: **1. 02. 1982.**

Konačna prijava (poštom):
Final registration: **30. 04. 1982.**

Za sve dodatne informacije obratite se na:

SVEUČILIŠNI RAČUNSKI CENTAR
(za simpozij)
Engelsova b.b.
41000 Zagreb

Telex: 21871 yu srce
Telefoni: 041/518-203
041/518-449

For all additional information please contact:

UNVERSITY COMPUTING CENTRE
(for the Symposium)
Engelsova bb,
41000 Zagreb
Yugoslavia

Telex: 21871 yu srce
Telephones: 41/518-203
41/518-449

TEME

1. Problematika razvoja i rada sveučilišnih računskih centara
2. Izobrazba kadrova iz područja računarskih znanosti i tehnologije
3. Izobrazba nastavnika i studenata sveučilišta u primjeni informatičkih sredstava i metoda u znanstvenom, nastavnom i stručnom radu
4. Planiranje, projektiranje i izgradnja informacijskih sistema
5. Informacijski sistemi u izobrazbi
6. Informacijski sistemi za praćenje znanstveno-istraživačkog i nastavnog rada, informacijski sistemi za bibliotečno-dokumentacijsku djelatnost i poslovni informacijski sistemi
7. Baze podataka
8. Arhitektura i sklopovski elementi računarskih sistema (velika-mini i mikro računala, distribuirani sistemi, pouzdanost sistema itd.)
9. Sistemska programska podrška i proizvodi (operacioni sistemi, komunikacijska podrška i protokoli itd.)
10. Programski jezici i metode programiranja
11. Teoretski aspekti računarskih znanosti i tehnologije
12. Evaluacija i testiranje programskih proizvoda i tehnologije
13. Primjena matematičkih i statističkih algoritama i metoda
14. Primjena u području prirodnih, društvenih, medicinskih i ostalih nauka
15. Tehničke aplikacije (uključujuši upravljanje procesima)
16. Aplikacijska programska podrška i proizvodi

informatica82

predavalnici bodo predavatelju na razpolago šolska tabla, grafoskop in diaprojektor.

The opening and plenary lectures will take palace in Big Hall at Ljubljana Fair in Ljubljana. All other lectures will be held in lecture rooms of Ljubljana Fair (Gospodarsko razstavišče). Each lecture room will be equipped with a blackboard, overhead projector, and slide projector.

Prenočišča
Accommodation

Hotelske sobe lahko rezervirate s tem, da ob prijavljanju udeležbe izpolnite tudi formular »Rezervacija prenočišča«.

Reservation for Hotel accommodation can be made by completing the enclosed »Housing reservation«.

Družabni program
Social events

Organizator simpozija bo pripravil za vse udeležence spoznavni večer in nekaj drugih prireditev.

The organizer of the Symposium will organize a social evening and some other social events.

Sprejemna pisarna
Reception Desk

Sprejemna pisarna simpozija na Gospodarskem razstavišču bo odprta vse dni simpozija od 8.—19. ure.

Reception desk of the Symposium at the Gospodarsko razstavišče will be open from 8 a.m. to 7. p.m. every day during the Symposium.

Seminarji
Seminars

1. C. van de Weteringh
   Office Automation
2. D. Vojnović
   Standard for a Real-Time Executive Kernel
3. J. J. Dujmović
   Evaluation and Comparison of 8-bit and 16-bit microprocessors

Vsak seminar bo trajal 4 do 6 predavateljskih ur.
Each Seminar will cover from 4 to 6 lecture hours.

0 Povabljena predavanja
Invited lectures

1. Cor van de Weteringh
   From Card-Tray to Resource Management The Influence of Computing of Imformation Processing in a Large International Company
2. Jozo Dujmovic
   Compiler Performance Measurement and Analysis
3. J. Weiss
   Review of CAD and Computer Graphics Activities in Austria
4. C. B. Besand
   Mechanical Engineering Drafting Using Computer Graphics

I Programska oprema
Computer Software

1. S. Bingulac, B. Petrović
   Interaktivna verzija programskog paketa CSMP
2. F. De Santis, N. Di Bianco
   On Implementing Microprocessor Systems Based Lisp Interpreters by Machine Oriented Language
3. S. Djordjevic, S. Nikolić
   Struktura makro biblioteka u praksi
4. S. Djordjevic, S. Gospodinov
   Makroi za strukturno programiranje na makroasemblerskom jeziku mikroprocesora Cosmac CDP 1802
5. H. Goid, M. Kunštic
   Sinteza funkcionalnih modula upravljačeq sustava telekomunikacija
6. L. Jančić
   Ocreduvanje na optimalna veličina na realna memorija pri rabota so virtualna memorija
7. A. Kulenovic, S. Alagic
   Metod realizacije naredbe foreach u jednom relacionom Pascal sistemu
8. I. Maric, L. Cucanić
   Programska realizacija aritmetike pomičnog zareza
9. A. Mišković
   Obrada Ludolfova broja na sistemu IBM 370/138
10. J. Ožegović
    Realizacija programske podrške komunikacijskih sistema
11. Z. Salčic, Z. Cvetanovic
    Jedan izvršni sistem mikroračunara za rad u realnom vremenu
12. A. Smailagic

11. N. Milenković, S. Nikolić
Mikroračunarska realizacija G-funkcija numerički upravljanih alatnih mašina
12. B. Nemec, M. Mitanović, R. Karba, F. Bremšak
Hibridna realizacija algoritma za sprotno vodenje multivariabilnega sistema po metodi inverznih Nyquistovih diagramov
13. N. Panić, P. Peterlin
Mikroračunalniški krmilnik sinoptičnih plošč
14. A. Rupnik
Prispevek k reševanju problemov okrnjenega nabora instrukcij zbirnikov procesnih računalnikov ob primeru pretvorbe BCD kode
15. V. Rupnik
O nekem razredu zvezno parametriziranih zveznih linearnih problemov neterminalnega upravljanja
16. F. Žlahtič
Zahtevani parametri programske in strojne opreme pri obdelavi informacij za vodenje elektroenergetskih sistemov
17. S. Srečković, B. Ilić
Osobenosti projektovanja dijaloga radnik-računar (CNC)
18. A. Grebenc
Sprotno vodenje
19. S. Divjak, P. Oblak, A. Ružić
Principi učenja in krmiljenja nekaterih jugoslovanskih robotov

VI Razne aplikacije v znanosti in tehniki
Miscellaneous Scientific and Engineering Applications

1. T. Damij, F. Grad
Uporaba linearnega programiranja pri planiranju dietne prehrane
2. Š. Dembitz
Automatsko otkrivanje grešaka u tekstu
3. V. Doleček, N. Kovačina
Mehanički odziv pumpno-turbinskog agregata na periodične i aperiodične pobude
4. M. Gmitrović
Aproksimativno određivanje odziva u linearnim mrežama sa jednim nestacionarnim elementom
5. M. Gmitrović
Rešavanje nelinearnih in nestacionarnih mreža metodom integralnih jednačina
6. D. Goljanin
X — Ray Spectra Fitting Prog.
7. G. Guida, C. Tasso
Dialog with Data Bases: An Effective Natural Language Interface
8. D. Hrisoho, K. Zafirovska, N. Atanasov
Informacioni sistem za kontrolu i programiranu individualizaciju lečenja putem hemodialize
9. B. Jenko, A. Paulin, N. Bezić
Parametrična obdelava trajektorije naelektrenega delca v osnosimetričnem elektrostatskem prostoru
10. G. Jovanović-Doleček
Računarski postupci za nalaženje zadanih podskupova stabala
11. A. Jurman
Primjena AOP-a u ispitivanju smjera i veličina oscilacija sezonskih pojava
12. A. Jurman
Primjena AOP-a u statističkim ispitivanjima hipoteze ovisnosti nekoliko varijabli
13. J. Lončar
Izračunavanje debalansa lopatica zrakoplovnih i drugih turbina u različitim metrikama
14. J. Lončar
Proračun optimalnog balansa kod zrakoplovnih i drugih turbina
15. B. Marangelli
Predicative Encoding in Progressive Transmission of Screened Photos
16. M. Marinković, I. Hajzler
Praktično korišćenje računara u pripremi procesa proizvodnje primenom simulacije na odabiranju modela linearnog programiranja
17. I. Meško, S. Meglič
Računalniški program za model penalov pri normalno porazdeljenih slučajnih spremenljivkah
18. M. Oblak, R. Pušenjak
Računanje lupin po teoriji loma
19. D. B. Popovski
Two Methods for Solving x = f (x)
20. R. Pušenjak
Snovanje elektronskih optičnih sistemov s pomočjo digitalnih računalnikov (II. del)
21. S. Sekulić
Opšta metodologija za određivanje režima obrade metodom modeliranja
22. M. S. Stanković, R. S. Stanković
Analiza procesa odabiranja u Walsh Fourierovom domenu
23. M. M. Stanković, R. S. Stanković
Ispitivanje uslova simetričnosti prekidačkih funkcija u spektralnom domenu
24. A. Takač, A. Marković
Mikroprocesorska podrška atestiranja SDR
25. I. Tvrdy, G. Kandus
Implementacija naročniških funkcij v mikroračunalniško krmiljenih telefonskih centralah
26. M. Lesjak
Mikroračunalniško vodeno adaptivno merjenje koncentracije HF v ozračju
27. M. Mele, B. Grm
Študij procesa fermentacije s pomočjo on-line simuliranja procesa in dograjevanja matematičnega modela
28. T. Štebe
Metodologija strukturirane izdelave funkcijske arhitekture sistema
29. A. Jezernik, S. Bader, J. Čop, M. Peternel, Z. Živec
Zamisel menujskega sistema SICAD za računalniško konstruiranje na domači računalniški opremi
30. M. Kac, J. Čop, B. Golob, M. Tetičković
Realizacija računalniške grafike z domačo računalniško opremo

VII Vzgoja in aplikacije v humanistiki
Education and Applications in Humanities

1. F. Ružić
Microcomputers Impact on Educational Process

Uvodni seminar o uporabi on-line informacijskega sistema DIALOG
Introductory seminar of DIALOG on-line information system usage

Ljubljana, 17. in 18. 5. 1982

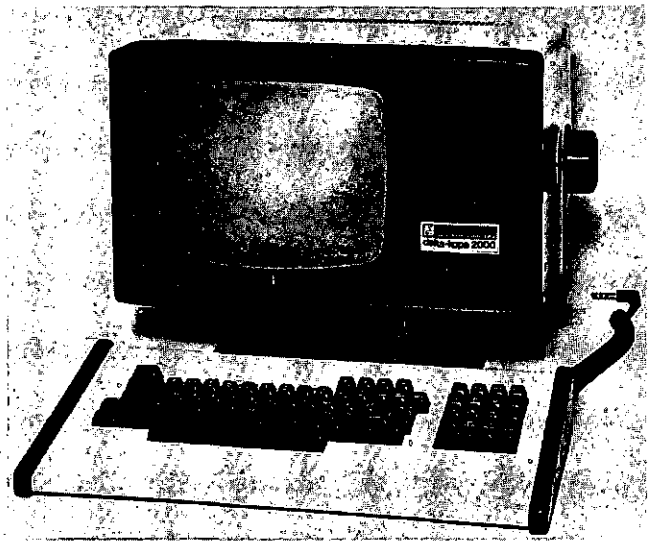
Organizator: Informacijski center v sodelovanju z DIALOG Information Systems Inc.

DELTA-KOPA 2000
video terminal

delta computer systems

The DELTA-KOPA 2000 is a video display terminal, based on the microprocessor technology and can be easily improved and qualified for more complete action execution. The terminal is simple to use, but with many features that facilitate work and improve communications.
The DELTA-KOPA 2000 is a result of our own research and development.

FEATURES
- **up to 132 characters per line**
- **double-size characters; double-width and double-height**
- **split screen capability**
- **easily upgraded to an idependent system**
- **selectable clear or dark screen**
- **7×9 dot matrix characters**
- **blinking, underline, and dual intensity characters, combination of all attributes on one character without the screen position loss**
- **settable tabs**
- ***video display stops at full screen***
- **separated keyboard**
- **standard numeric/function keypad**
- **special graphic characters**
- **extremely easy maintenance**
- **built-in self-test diagnostics**
- **special video output port**
- **special line printer output port**
- **monitor inclination adaptation**
- **table, hanging or wall mounting**
- **KOPA 700, KOPA 1000, VT 100, VT 52 compatibility mode**
- **LED indicators for programme control**
- **uncontacted feature settings**
- **chosen feature setting are held also after the power off**
- **Duplex, asynchronous communication lines**



SPECIFICATIONS

Dimensions:	
Monitor/	length 46 cm
without support	width 43 cm
	height 28 cm
with support	length 52 cm
	width 43 cm
	height 36 cm
Keyboard	length 46 cm
	width 24 cm
	height 6 cm
Weight:	15.6 kg
Working conditions:	temperature from 10 to 40°C relative humidity from 10 to 90 %
Power supply:	180–256 V/47–63 Hz/100 VA
Wideo screen:	
cathod tube	diagonally 31 cm, phosporus GR
Format	24 lines × 80 characters or 34 lines × 132 characters, selectable
Characters	7 × 9 matrix
Active screen surface	205 mm × 115 mm
Character set	96 – Character ASCII
Keyboard:	
keys	65 keys typewriter – like displayed
Additional keyboard	18 numerical keys with fullstop, comma, minus, ENTER key and 4 programme – functional keys
Klick	acustic confirmation of pressed key and margine signal for fault
Conditions:	
Type	EIA (RS-232-C)
Baud Rates(bps)	Full-duplex 50, 75, 110 (two stop bits), 134, 150, 200, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600, 19200
Character format	asynchronous
Character length	7 or 8 bits, keyboard – selectable (if 8 bits are chosen for a character, the 8th bit doesn't carry the information)
Codes	USASCII, JUS A.F0.101
Parity	Even, odd or none, keyboard – selectable
Synchronisation	keyboard – selectable, generating the XON/XOFF control code printer – selectable CTS or XON/XOFF

Iskra **Delta, Ljubljana, Yugoslavia**

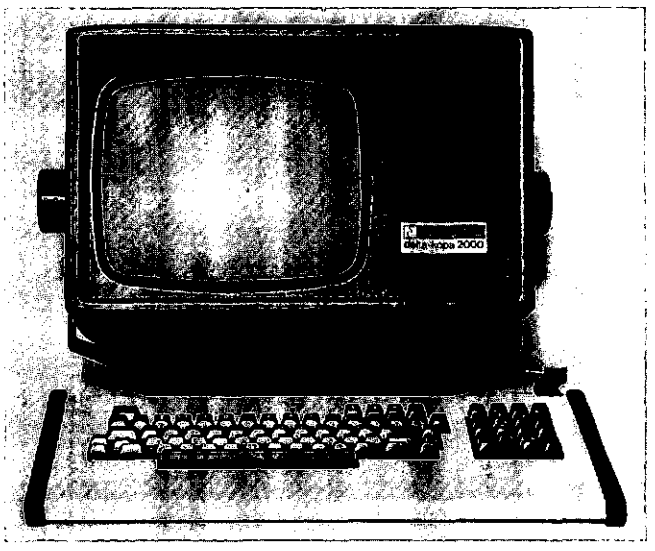
delta computer systems

THE KEYBOARD

The typewriter-like keyboard of the DELTA-KOPA 2000 is attached to the video display unit by 1,50 m coil cord. This feature allows the user to place the keyboard in a variety of positions. This not only provides a more comfortable workstation but also saves desk space. To facilitate operation, the DELTA-KOPA 2000 keyboard provides special function keys such as scroll key, cursor control keys, and keys that transmit control codes.
To the right of the keyboard is a build-in, calculator-style numeric and function pad. This pad can be used to inita-te, with one keystroke, entire sequences of application-specific numeric data entry and programmed operations commonly used at the terminal. Seven LED indicators are also provided as operator information and diagnostic aids for fault detection.

VIDEO SCREEN DISPLAY

One of DELTA-KOPA 2000 important features is that it has 80- and 132-column lines. A 132 colum permits easy viewing of wide line-printer reports on the video screen, and allows these reports to be transferred directly from screen to printer without reformatting data. By enabling the SMOOTH SCROLL feature, a user can read new lines of text easily, even if they are being received at high transmission speeds. The NO SCROLL key can stop the video display anywhere and reset it again. Split screen capability allows part of the full 24-line screen to be scrolled separately. Thus data can be displayed in a fixed area of the screen, independent of data that is entered or accessed by the user.



The SET-UP key replaces the display screen by a status screen (SET-UP A). The status screen allows the user to select character displays, to set screen lightness and tabulators. By pressing the key 5, the operator switches the status SET-UP A to SET-UP B.
This status allows the user to select transmission speeds and other features (light background on the screen, the form of screen cursor, marginal signal).

CHARACTERS

To make the screen easier to read, the DELTA-KOPA 2000 offers a 7 × 9 dot matrix character font on the 10 × 10 dot space that provides crisp characters and two-dot descenders on lower-case characters. Also available are double-width and double-size characters. Large characters not only emphasize formatted text, but also make complicated forms easier to read. Using double-size characters, messages can even be read from across a room.
The DELTA-KOPA 2000 provides white characters on dark background and allows the user to choose dark characters on a light background. This feature gives the screen the appearance of a page of printed text, making area margins more visible and providing a different screen contrast environment for to user.
The basic character set features, next to the characters, numbers and punctuations, also 39 of line drawing graphic characters that can be used to present pictorial information on the screen.

GENERAL FEATURES

The DELTA-KOPA 2000 has two mechanical switches: one for turning the terminal on and off and the other to control the power supply. All other functions, such as baud rates, tabs, parity, etc., are set via keyboard and stored in the special memory. The elimination of mechanical switches facilitates the use of diagnostics to test the terminal function and to adapt to varying environments under host control. Built-in seft-test diagnostics greatly reduce the time taken to isolate and repair faults. The use of snap fasteners provides fast access to and easy servicing.
The DELTA-KOPA 2000 uses duplex asynchronous communication lines and EIA 232 and 20 mA interfaces. The new important feature of the DELTA-KOPA 2000 video terminal is, that besides main port it has also special serial line printer port.