YU ISSN 0350-5596

gorenje *informatika '83*

Industrija
prosto programirani sistemi
procesni mikroračunalniški sistemi

Komunikacije s prenosom podatkov
teletekst
videotekst

Informacijski sistemi
Obrambni in varnostni sistemi
Medicinska elektronika in informatika

Mali poslovni sistemi
sodelovanje z Birostrojem

Potrošna elektronika
Terminali
Paka 2000
Mikroračunalniške enote in moduli

informatica

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

YU ISSN 0350–5596

LETNIK 7, 1983 – Št. 2/3

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

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

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

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije.

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

TISK: Tiskarna Kresija, Ljubljana

GRAFIČNA OPREMA: Rasto Kirn

V S E B I N A

informatica

JOURNAL OF COMPUTING AND INFORMATICS

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

YU ISSN 0350–5596

VOLUME 7, 1983 – No 2/3

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

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

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

C O N T E N T S

OPERACIJSKI SISTEM CP/M PLUS

ANTON P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.3.06 CP/M PLUS:181.4

Ta članek pregledno opisuje lastnosti novega operacijskega sistema CP/M Plus (CP/M verzija 3.0) za uporabnike sistema CP/M 2.2, tako da prikazuje nove lastnosti sistema CP/M Plus, in sicer vgrajene ukaze, nove prehodne programe tega sistema, nove BDOS funkcije, nove BIOS funkcije, sistemski krmilni blok, diskovne podatkovne strukture, vmesnike in sekljalne tabele, zahteve za nebančni in bančni operacijski sistem in postopek modifikacije s pomočjo sistema CP/M 2.2. Splošno je opisan mehanizem pomnilniškega upravljanja, organizacija bančnega pomnilnika, bančne posebnosti BIOSa in prednosti bančnega sistema s tabelami in vmesniki.

C P / M P l u s O p e r a t i n g S y s t e m . This article is a survey of the new CP/M Plus Operating System features for the CP/M 2.2 users showing new abilities of the CP/M Plus System: built-in commands, new transient programs of the system, new BDOS functions, new BIOS functions, system control block, disk data structures, buffering and hash tables, nonbanked and banked systems requirements and updating from CP/M 2.2. In a general way memory management, banked memory, banking of BIOS, benefits of a banked system with tables and buffers are described.

1.Uvod

Sistem CP/M je bil v časopisu Informatica že dokaj izčrpno opisan (glej navedbe v slovstvu pod številkami ((1)), ((2)) in ((3)), kjer je navedena še druga literatura). CP/M Plus je nova izvedenka sistema CP/M 2.2, ki se je letos pojavila na tržišču in pomeni nove, vpodbudne možnosti za uporabo 8-bitnih mikroprocesorjev.

Novi osebni mikroračunalnik Partner, ki ga proizvaja DO Iskra Delta, uporablja nov operacijski sistem CP/M Plus (ali CP/M 3.0) za 8-bitne mikroprocesorje. Nova različica je še vedno dovolj podobna staremu CP/M 2.2, je enouporabniška in izvaja zbirke tipa .COM, ima tudi nekatere prejšnje ukaze, kot so DIR, REN, TYPE itd. Čas, ki je potreben za priučitev na novo različico operacijskega sistema, je neznaten (spoznavanje operacij na ukazni ravni). Vendar je s temi pičlimi ugotovitvami tudi konec podobnosti med novo in staro različico sistema CP/M.

CP/M Plus je bistveno različen od svojega predhodnika, saj predstavlja novo (tehnološko) generacijo. Imeniki tega sistema so sekljani (hashed). Tkim. LRU (Least Recently Used) vmesniški prenos (uporaba pomnilnih vmesnikov za sektorje s strategijo, da se uporabi vselej, ko zmanjka pomnilniških vmesnikov, tisti vmesnik, ki je bil v poslednjem času najmanj uporabljan) se opravlja v BDOSu (v osnovnem diskovnem operacijskem sistemu). Obseg diskovne enote je povečan na 512M zlogov (npr. uporaba 600M zložnega vinčestrskega diska) in največji obseg zbirke je 32M zlogov. CP/M Plus lahko uporablja pomnilnik z bankami (npr. bančni pomnilnik z 256k zlogi dinamičnega RAMa), ki prinaša vrsto prednosti, kot so razširjeno vrstično urejanje, dostop z geslom (podoben dostopu v zbirke sistema MP/M) in razširitev vmesniških (perifernih) funkcij.

Vrsta lastnosti je resda ostala nespremenjenih, vrsta pa se jih je bistveno spremenila. CCP (ukazni konzolni procesor) je lahko prehoden program. Operacijski sistem ima svoje bivališče v regularni diskovni zbirki (ne na posebnih sistemskih stezah diska) in se naloži z uporabo enostavnega nalagalnika; ta se nahaja na sistemskih stezah zadevnega diska oziroma diskete. Tudi SAVE ukaz (glej slovstvo o CP/M sistemu na koncu članka) je prehodni ukaz (v stari različici vgrajen ukaz), ki sam sebe premesti na vrh TPA (Transient Program Area oziroma območje prehodnih programov). CP/M Plus sistem se generira iz množice premestljivih zbirk z uporabo generirnega programa GENCPM na način, ki je podoben načinu generiranja v MP/M sistemu. BIOS (Basic I/O System oziroma osnovni V/I sistem, ki je poseben modul operacijskega sistema) je razdeljen na enostavnejše module, tako da je modifikacija (in graditev dodatnega dela) lažja in hitrejša in da se lahko uporabi (in postopno razvija) tudi modificirani BIOS sistema CP/M 2.2. V/I se lahko preusmerja (krmili) z enostavnim storitvenim programom. Nov operacijski sistem ima dodanih tudi več sistemskih programov, kot je npr. dokaj obsežen HELP ukaz. Celota je skupaj z BDOSom tako izpopolnjena, da zagotavlja visoko stopnjo prijaznosti, priročnosti in seveda zanesljivosti.

2. Nove lastnosti sistema CP/M Plus

2.1. Dokumentacija

Dokumentacija za CP/M Plus je v primerjavi z dokumentacijo za stari sistem (CP/M 2.2) izdelana vzorno in profesionalno. Sestavljena je iz štirih priročnikov: uporabniškega, sistemskega, programirnega in storitvenega. Vsi priročniki so vsebinsko tako zaokroženi, da vsak zase omogočajo razumevanje sistema CP/M Plus in njegovo modifikacijo iz sistema CP/M 2.2.

2.2. Vgrajeni ukazi

Dodanih je več novih ukazov, in sicer vgrajenih in prehodnih. Iz prejšnje različice so ostali ukazi DIRectory, ERAse, REName, TYPE in USER, dodan pa je bil ukaz DIRSYS, ki prikaže sistemske zbirke (prej prikazljive s prehodnim ukazom STAT). Vendar so vsi ti ukazi razširjeni (izpopolnjeni), tako da avtomatično nalagajo zadevne prehodne programe. Vgrajeni ukazi so tile:

DIR — To je storitev z razširjenim prikazom imenika. DIR vsebuje več kot 16 različnih možnosti ukaznih vrstic, prikaže lahko obseg zbirk (prej STAT ukaz), preostali prostor na disku in vstope, pridevke, datum, diskovne enote, uporabniška območja itd.

DIRSYS — Prikaže vse sistemske zbirke v trenutnem uporabniškem območju.

ERASE — Ta ukaz je podoben ukazu 2.2 ERA, vendar omogoča potrditev (soglasje uporabnika) pred izbrisom.

RENAME — Omogoča preimenovanje zbirk, lahko zbriše dvojna imena (s potrditvijo), omogoča vključitev delov v specificirana zbirčna imena.

TYPE — Ima enako funkcijo kot prej, vendar omogoča stranenje (listanje po straneh) in zahteva ime zbirke, če to ni bilo posredovano v ukazni vrstici.

USER — Ima enako funkcijo kot prej, vendar zahteva številko uporabnika, če mu ta ni bila posredovana v ukazni vrstici.

2.3. Prehodni ukazi (programi)

Novi prehodni programi sistema CP/M Plus pa so tile:

COPYSYS — To je storitev za kopiranje nalagalnika (podobno kot SYSGEN). CP/M Plus se nahaja v diskovni zbirki in COPYSYS vpiše nalagalnik na sistemske steze.

DATE — Nastavi in prikaže datum in čas.

DEVICE — Se uporablja za prireditev logičnih CP/M naprav eni ali več fizičnim CP/M napravam. Omogoča uporabniku spremembo vmesniških (perifernih) protokolov, hitrosti prenosa (baud rate) in modifikacijo višine in širine zaslona.

DUMP — Je podoben DUMP programu za prikaz zbirke v HEX in ASCII formatu.

ED — ED je tu vrstično usmerjen, ima pa vse prejšnje ukaze.

GENCOM — To je storitev, ki omogoča RSX zbirkam (Resident System eXtension) navezavo na .COM zbirke.

GET — Omogoča CP/M sistemu, da dobi konzolni vhod iz diskovne zbirke namesto s tastature.

HELP — To je storitveni program, ki s pomočjo 76K-složne podatkovne zbirke pojasnjuje uporabo ukazov.

HEXCOM — Podobno kot LOAD v CP/M 2.2 se ta program uporablja za pretvorbo .HEX zbirk v .COM zbirke.

INITDIR — Inicializira diskovni imenik in omogoči odtis časa in datuma v bančnem CP/M Plus sistemu.

LIB — Knjižnični storitveni program.

LINK — Standardni povezovalnik za proizvodnjo programskih zbirk iz .REL zbirk, nastalih z uporabo RMACa.

MAC — Ta makrozbirnik generira .HEX zbirke.

PATCH — Sistemska storitev za zbirčno popravljanje sistema.

PIP — Je enak staremu PIPu, vendar omogoča prestavitev zbirk v druga uporabniška območja in lahko zahteva potrditev za vsako proceduro; ima tudi ARCHIVE lastnost.

PUT — Omogoča, da se zapis za tiskalnik ali konzolo prenese v diskovno zbirko.

RMAC — To je premeščevalni makrozbirnik, ki generira .REL zbirke.

SAVE — Je storitev, ki sama sebe instalira na vrh TPA, se vrne, prestreže izstop NEXT programa, da tako omogoči rešitev TPA vsebine. (SAVE ukaz ni vgrajen v CP/M Plus).

SET — Omogoča nastavitev zbirčnih pridevkov. Pridevki so diskovne označitve, tip časovnega in datumskega odtisa, zaščita z geslom in drugi sistemski pridevki, ki so se uporabljali v STAT.

SETDEF — Omogoča nastavitev različnih sistemskih možnosti, kot so veriga iskanja enot (diskovnih), začasna enota in vrstni red iskanja zbirčnih tipov. Krmili tudi sistemska načina DISPLAY in PAGE: DISPLAY način povzroči prikaz imen in programskih lokacij ali naloženih zbirk tipa .SUB; PAGE funkcija omogoča stranenje konzolnega prikazovanja.

SHOW — Prikaže karakteristike diskov in enot. Ta storitev nadomešča skupaj s SET storitvijo ukaz STAT.

SID — To je nova različica za DDT.

SUBMIT — To je izboljšan izvajalnik ukaznih zbirk z možnostjo vgnezdevanja in lahko vsebuje programske vhodne vrstice. Poseben primer .SUB zbirke je PROFIL .SUB, ki se avtomatično izvaja pri mrzlem zagonu.

XREF — To je storitev za križne navedbe .ASM zbirke.

Eno je očitno: ukazi sistema CP/M Plus so zelo podobni ukazom sistema MP/M; nekateri so dobesedno prepisani. CP/M Plus ima obilo programske opreme, npr. LIB, LINK, MAC, RMAC, SID, XREF, ki je ni potrebno kupovati posebej.

2.4. Nove BDOS funkcije

BDOS funkciji 3 in 4 sta bili preimenovani iz READER INPUT in PUNCH OUTPUT v AUXILIARY INPUT in AUXILIARY OUTPUT. Funkciji 7 in 8, ki sta bili GET in SET I/O BYTE, sta opuščeni, njuni številki pa prirejeni funkcijama AUX. INPUT STATUS in AUX. OUTPUT STATUS. Več starih BDOS funkcij je bilo modificiranih, vendar so ostale približno združljive s starim sistemom.

Nove BDOS funkcije s pripadajočimi številkami za CP/M Plus so tele:

37 RESET DRIVE: se uporablja za resetiranje posameznih enot; je združljiva z MP/M.

44 SET MULTISECTOR COUNT: zagotavlja blokiranje logičnih zapisov; je združljiva z MP/M.

45 SET BDOS ERROR MODE: določa obdelavo fizičnih in razširjenih napak; je združljiva z MP/M.

46 GET DISK FREE SPACE: določa število prostih sektorjev na specifični enoti; je združljiva z MP/M.

47 CHAIN TO PROGRAM: omogoča veriženje iz enega programa na drugega; je združljiva z MP/M.

48 FLUSH BUFFERS: izsili zapis poljubnega pisalno nerešenega zapisa; je združljiva z MP/M.

49 GET/SET SYSTEM CONTROL BLOCK: omogoča dostop v krmilni blok sistema CP/M Plus.

50 DIRECT BIOS CALLS: omogoča neposredne BIOS pozive skozi BDOS v BIOS.

59 LOAD OVERLAY OR RESIDENT SYSTEM EXTENSION: naloži modulsko plast ali RSX modul v pomnilnik.

60 CALL RESIDENT SYSTEM EXTENSION: se uporablja za klicanje RSXov.

98 FREE BLOCKS: vrne v prosti prostor vse začasno dodeljene podatkovne bloke vseh trenutno aktivnih enot; uporablja se v CCP po toplem zagonu.

99 TRUNCATE FILE: nastavi zadnji zapis zbirke na določeno naključno zapisno številko

100 SET DIRECTORY LABEL: oblikuje ali popravi imeniško označitev; je združljiva z MP/M.

101 RETURN DIRECTORY LABEL DATA: vrne označitveni vstop imeniške označitve za določeno enoto; je združljiva z MP/M.

102 READ FILE DATE STAMPS AND PASSWORD MODE: vrne informacijo datumskega in časovnega odtisa in geselski način za določeno zbirko; je združljiva z MP/M.

103 WRITE FILE XFCB: oblikuje oz. popravi XFCB za specificirano zbirko; je združljiva z MP/M.

104 SET DATE AND TIME: nastavi sistemski notranji datum in čas; je združljiva z MP/M

105 GET DATE AND TIME: dobi sistemski notranji datum in čas; je združljiva z MP/M.

106 SET DEFAULT PASSWORD: omogoča programu specifikacijo zbirčnega gesla pred dostopom vanj, sicer se pojavi geselska napaka; je združljiva z MP/M.

107 RETURN SERIAL NUMBER: vrne 6-zložno serijsko številko sistema CP/M Plus.

108 GET/SET PROGRAM RETURN CODE: omogoča programu, da nastavi ali dobi vrnitveni kod pred vrnitvijo.

109 GET/SET CONSOLE MODE: to je 16-bitni sistemski parameter, ki določa akcijo več BDOS konzolnih V/I funkcij, kot je npr. Control-C funkcija, zvijanje/ ne zvijanje itd.

110 GET/SET OUTPUT DELIMITER: dobi ali nastavi trenutni izhodni omejevalnik, normalno "$".

111 PRINT BLOCK: pošlje znakovni niz, ki je lokaliziran s specificiranim CCB (Character Control Block) na logično konzolo.

112 LIST BLOCK: pošlje znakovni niz, ki je lokaliziran s specificiranim CCB na logično napravo za listanje.

152 PARSE FILENAME: analizira ASCII zbirčno ime in pripravi FCB (File Control Block).

Tudi vrsta MP/M funkcij (38, 39, 41, 42, 43), ki niso podprte, vrne ustrezen ("uspešni") kod.

2.5. Nove BIOS funkcije

BIOS skočni vektor sistema CP/M Plus je bil razširjen iz 17 na 33 skokov. Za delovanje sistema CP/M Plus ni potrebna implementacija vseh skokov, vendar mora biti 33 skokov vključenih v skočno tabelo. Navadno neuporabljeni skok kaže na RET ukaz. Eden od skokov je rezerviran za OEM implementacijo (številka 30) in je dostopen skozi BDOS funkcijo 50. Zadnja dva skoka sta rezervirana za prihodnjo uporabo. Za operativnost osnovnega CP/M Plus sistema zadostuje implementacija skokov 0-4, 8-14, 16 in 26.

BIOS funkcije 0-16 so načeloma enake kot v sistemu 2.2, z izjemo funkcij 6 in 7 (PUNCH, READER), ki sta zamenjani z AUXOUT in AUXIN, tj. z rutinama za pomožni V/I. Nove BIOS funkcije sistema CP/M Plus so tele:

17 CONOST: vrne izhodni status konzole.

18 AUXIST: vrne vhodni status pomožnih vrat.

19 AUXOST: vrne izhodni status pomožnih vrat.

20 DEVTBL: vrne naslov znakovne V/I tabele.

21 DEVINI: inicializira znakovno V/I napravo.

22 DRVTBL: vrne naslov tabele diskovne enote.

23 MULTIO: nastavi številko naslednjega sektorja za branje ali pisanje.

24 FLUSH: vsili polnjenje fizičnega vmesnika pri uporabniško podprtem deblokiranju.

25 MOVE: opravi pomnilniško pomnilniški bločni pomik. Opravi lahko tudi bančno bančni bločni pomik, če se pokliče funkcija XMOVE.

26 TIME: dobi ali nastavi čas.

27 SELMEM: izbere specificirano pomnilniško banko.

28 SETBNK: specificira banko za DMA operacijo

29 XMOVE: nastavi banki za naslednji pomik.

30 USERF: rezervirano za sistemskega implementatorja.

31 RESERV1: rezervirano za prihodnjo uporabo.

32 RESERV2: rezervirano za prihodnjo uporabo.

Nove BIOS funkcije so v celoti pojasnjene v CP/M Plus priročnikih, vendar so nekatere podrobnosti zanemarjene in jih je težko najti.

2.6. Preusmeritev V/I in sistemski krmilni blok

Sistemski krmilni blok (SCB za System Control Block) je 100-zložni blok podatkov v BDOSu. SCB hrani različne sistemske parametre, zastavice in spremenljivke, ki se navajajo, modificirajo z BDOSom, CCPjem in BIOSom. SCB definicija se nahaja v posebni zbirki sistema CP/M Plus, imenovani SCB.ASM . Vse spremenljivke v SCB.ASM so definirane kot PUBLIC spremenljivke, tako da jih BIOS lahko navaja. To je vsekakor prednost pri uporabi programa za sistemsko generiranje in zbirk tipa .REL . SCB spremenljivke so markirane kot R/O ali R/W v odvisnosti, ali so lahko ali ne modificirane z BIOSom.

2.7. Diskovne podatkovne strukture

CP/M Plus je podoben sistemu 2.2 in uporablja tudi vodnik diskovnih parametrov (DPH za Disk Parameter Header), bloke diskovnih parametrov (DPB za Disk Parameter Block), testno vsoto in dodeljevalne vektorje. Dodanih pa je tudi nekaj novosti. Vmesniški krmilni bloki (BCB za Buffer Control Block) so dodani za lociranje vmesnikov fizičnih zapisov za BDOS, dodani pa so tudi imeniški vmesniki. Kot kaže slika 1, so bili DPB (bloki diskovnih parametrov) nekoliko razširjeni z novimi strukturami in kazalci. Novi DPH (vodnik diskovnih parametrov) je malce nejasen in bolj zapleten kot njegov 2.2 predhodnik; vendar vse to povečuje hitrost sistema CP/M Plus. Ista BCB struktura se uporablja za lociranje obeh vmesnikov: imeniškega in podatkovnega. Vse to pa zahteva precej več pomnilnika kot v starem sistemu.

2.8. LRU vmesniki

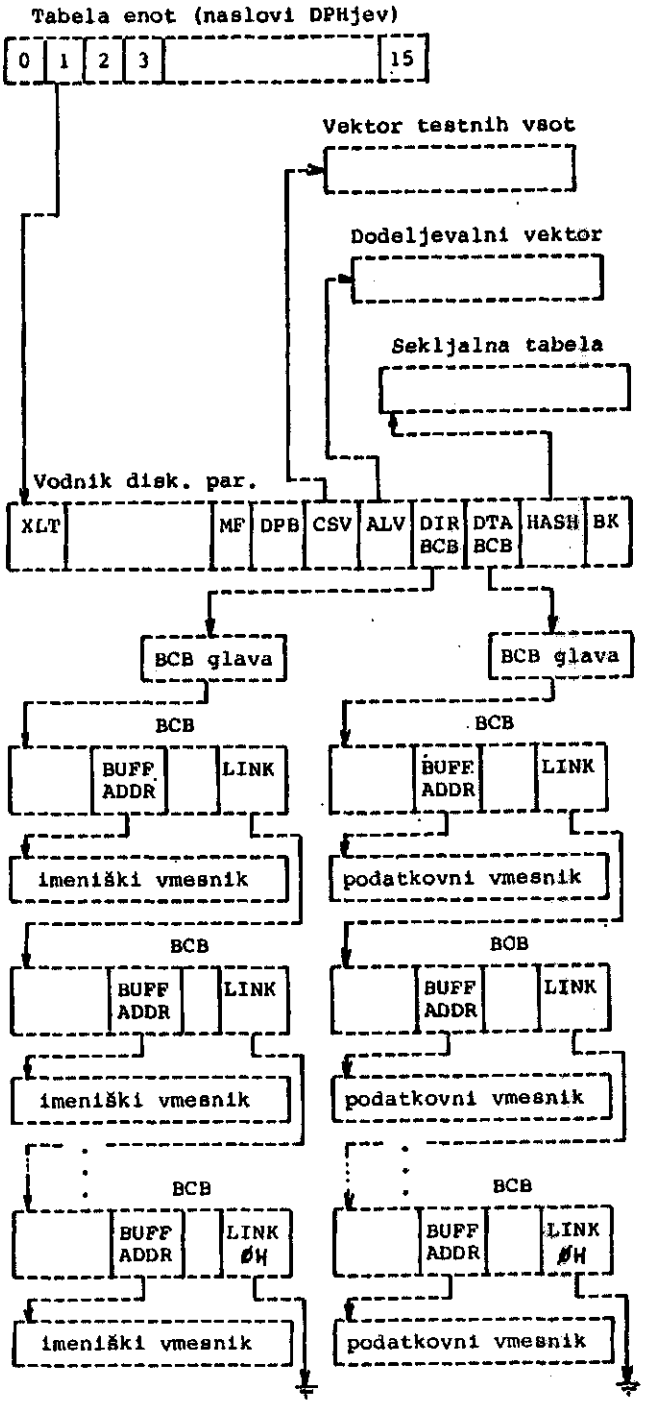
V bančnih sistemih, kot je CP/M Plus, se uporablja vmesniška shema LRU (Least Recently Used) za upravljanje skrivališča deblokirnih vmesnikov in imeniških zapisov.Če BDOS potrebuje vmesnik, izbere vselej tistega, ki je bil v poslednjem razdobju najmanj uporabljan (LRU). Shema z LRU vmesniki ne povečuje neposredno izvajalne hitrosti ali diskovnega V/I, toda lahko pridobi dragoceni čas pri ponovljenih nalaganjih pogostno uporabljane informacije.

2.9. Sekljalne tabele

CP/M Plus uporablja sekljalne tabele za povečanje hitrosti iskanja v imeniku. Pri sekljanju je položaj sektorjev v imeniku določen brez preiskovanja dejanskega imenika, kar zmanjša število diskovnih dostopov, ki bi bili potrebni za dostop v imeniški vstop. Tako je moč rešiti dragoceni čas, ki bi bil sicer porabljen za zaporedno branje imenika pri iskanju specifičnega vstopa.

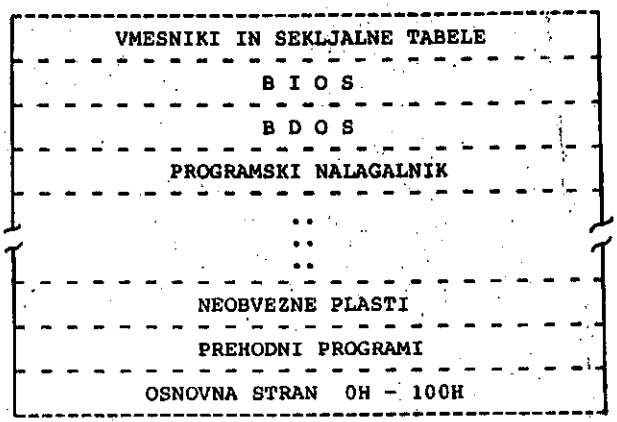
2.10. Nebančne in bančne sistemske zahteve

Čeprav ne mislimo opisovati načina instalacije



Slika 1. Bančni sistem BIOS podatkovnih struktur v sistemu CP/M Plus

CP/M Plus sistema pa vendar omenimo minimalne sistemske zahteve za implementacijo sistema CP/M Plus. Načeloma potrebuje nebančni sistem le 8,5K zlogov in še prostor za BIOS (ki je zelo odvisen od specifičnega sistema oziroma aparaturne konfiguracije) in najmanj 32K zlogov RAMa. Bančni sistem potrebuje vsaj dve banki, z najmanj 11K zlogi in prostorom za BIOS v banki št. 0 in najmanj 1,5K zlogov v skupni banki (v banki št. 1). Minimalni bančni sistem potrebuje 96K zlogov v dveh bankah.



Slika 2. Značilen nebančni sistem CP/M Plus

Na sliki 2 in sliki 3 imamo prikazana tipična primera nebančnega in bančnega sistema CP/M Plus.

2.11. Modifikacija iz sistema CP/M 2.2

Modifikacija sistema CP/M 2.2 v sistem CP/M Plus je lahko enostavna in hitra, če imamo določeno znanje in izvirni (zbirni) kod BIOSa sistema CP/M 2.2. V sistemskem priročniku sistema CP/M Plus je dana procedura, ki pojasnjuje posamezna dejanja. Seveda bo modifikacija bistvena, potrebno bo pa tudi zbirniško prevajanje z uporabo zbirnika tipa RMAC; vobče bo potrebno dodati 16 novih BIOS funkcij, razširiti vodnik in blok diskovnih parametrov. Večina od obstoječih BIOS funkcij (0 do 16) bo spremenjenih, nekatere od njih celo bistveno. Razen tega bo potrebno zgraditi manjšo različico obstoječega BIOSa za povezavo v nalagalni program sistema CP/M Plus; vendar tu ni pričakovati večjih težav. Seveda pa naloga ni tako lahka, kot se zdi in potrebnih je več ur trdega dela za odpravljanje napak v novem BIOSu, preden bo ta deloval zanesljivo. Seveda pa je CP/M Plus lažje implementirati kot sistem MP/M.

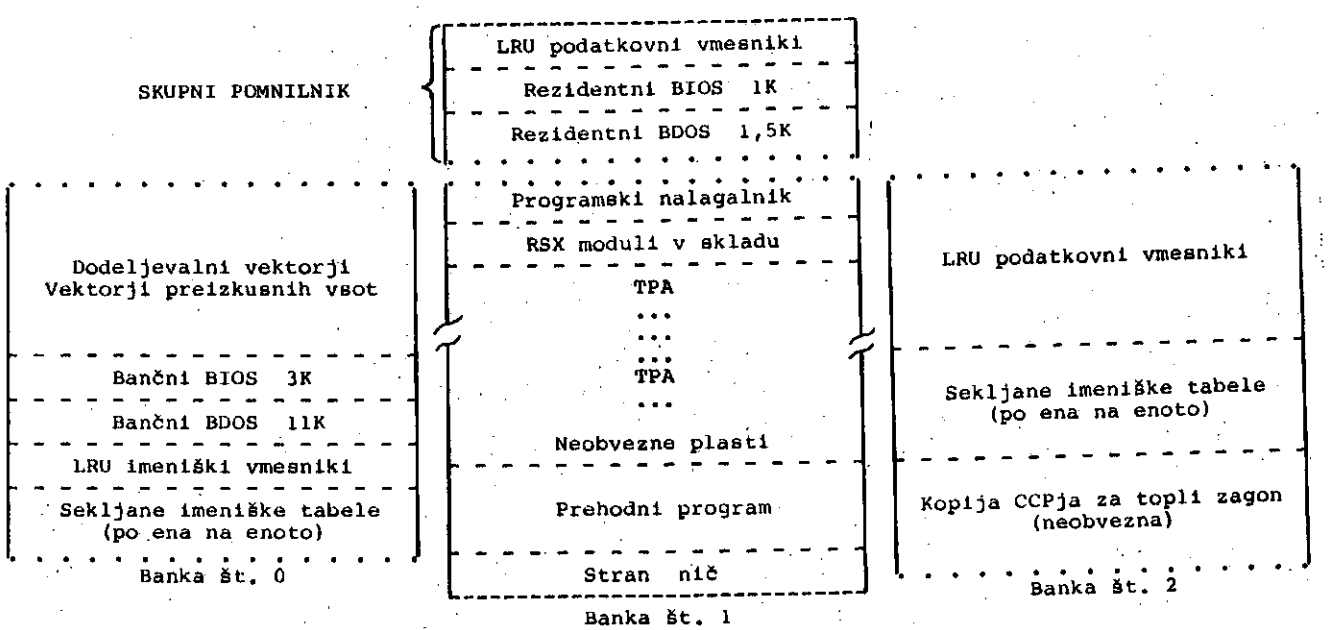
3. Nove lastnosti sistema CP/M Plus

Sistem CP/M Plus ima več novih lastnosti, kot so odtis datuma in časa v zbirke, zbirčna gesla in zmogljiv urejevalnik ukaznih vrstic. Pri tem pa so pomembnejše notranje izboljšave sistema,kot je povečana hitrost dostopa v zbirke in povečane pomnilne zmogljivosti za progam v izvajanju. Vse to pa zahteva bolj zapleten BIOS, ki ga je potrebno dodatno pojasniti. To, kar je v primerjavi sistema CP/M Plus s sistemom CP/M 2.2 bistveno, je večji pomnilni prostor za izvajajoči program pri novem sistemu, čeprav je ta sistem obsežnejši od starega. Ta lastnost je posledica posebnega mehanizma, ki ga imenujemo upravljanje pomnilnika.

3.1. Upravljanje pomnilnika

Pomnilniško upravljanje je metoda, ki je že dolgo znana pri velikih računalnikih. Pomnilniško upravljanje preseže omejenost fizičnega pomnilnega prostora, ki je na razpolago za posamezne mikroprocesorje. Za večino osembitnih procesorjev je ta prostor omejen s 16 naslovnimi biti, tako da imamo le 65536 enoličnih pomnilniških naslovov.

Kadar programer spozna, da določeni programski deli ne navajajo eden drugega, lahko te dele loči v plasti, za katere pa ni več potrebno, da se istočasno nahajajo v pomnilniku (natačneje v stanju izvajanja). To pa omogoči, da se lahko plasti nalagajo v absolutni pomnilni prostor na prekrivajoče naslove (ne na višje ali nižje), vendar v različnih časovnih intervalih. Tako lahko postane tudi absolutni pomnilni prostor procesorja, ki je samo 64k-zložen, dovolj velik za izvajanje neomejeno razsežnih programov, če so ti seveda bili ustrezno razdeljeni v plasti. Po drugi strani pa lahko plasti naložimo v paralelne pomnilnike (imenujemo jih banke), ki jih po potrebi vključujemo v izvajalni proces. Ta metoda zahteva minimalno dodatno materialno opremo oziroma preklopni mehanizem, ki ga uresničimo z uporabo V/I vrat (procesorji Z80, 8080 in 8085). Tako dobimo sistem s tkim. "bančnim preklapljanjem" ali z "razširjenim naslavljanjem".



Slika 3. Značilen bančni CP/M sistem (ki pa ni edini)

Opisano pomnilno upravljanje pa ima tudi določene omejitve. Pri 64K-zložnem prostoru ni moč prekljapljati celotnega obsega, ker bi s tem izgubili izvajalno zaporedje oziroma bi procesor moral izvajati program pri naslednjem naslovu novega prostora, kjer je v starem prostoru nehal. Vobče ni smotrno izključiti pomnilnika, v katerem se nahaja del programa za preklop. Posledica tega je, da mora ostati določen del pomnilnega prostora skupen vsem bankam. V tem delu se mora nahajati tkim. preklopni segment (del programa).

V sistemu CP/M Plus je moč definirati tkim. "skupno bazo", in pod to bazo se lahko nahaja ves kod, ki ne vpliva na preklapljanje bank. Vrednost skupne baze je odvisna materialno realiziranga mehanizma pomnilniškega upravljanja v konkretnem računalniku. Navadno se pomnilnik prekljaplja v 16K-zložnih segmentih in v tem primeru se skupna baza lahko začne pri naslovu C000H. Banke se tako preklapljajo v intervalih po 48K zlogov in segajo od 0000H do BFFFH.

3.2. Bančni pomnilnik pri CP/M Plus

Polna implementacija sistema CP/M Plus zahteva vsaj tri pomnilniške banke. Prva banka (banka št. 0) se uporablja za shranjevanje bančnega dela BIOSa in BDOSa, diskovnih dodeljevalnih vektorjev, vektorjev preizkusnih vsot, imeniških vmesnikov in sekljalnih tabel. Banka št. 1 se uporablja kot področje prehodnih programov (TPA) in v njej se izvajajo aplikativni programi in konzolni ukazni procesor (CCP).

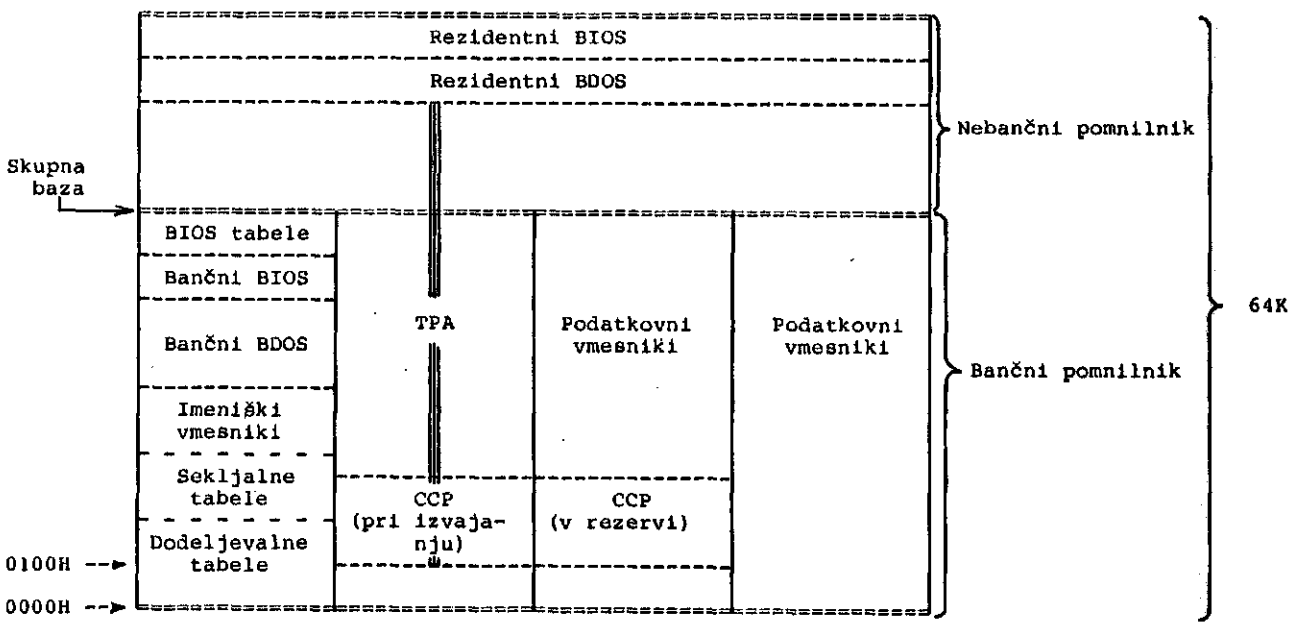
Na sliki 4 vidimo bogatejši sistem CP/M Plus s štirimi bankami. Rezidentni BDOS ima obseg le 1,5K zlogov in le mali del BIOSa mora biti rezidenten, tako da je TPA kar se da obsežno. Tako lahko ostane za TPA tudi 61K zlogov RAMa. Preostale banke se lahko uporabljajo prvenstveno kot diskovni podatkovni vmesniki. V 3K-zložnem segmentu se lahko nekje nahaja rezervna kopija CCPja, ki se kopira v banko 1 z rutino ponovnega (toplega) zagona z rutino v BIOSu. Tako ponovni zagon ne potrebuje diskovnega dostopa. Zaradi tega je dovoljena menjava diskete v enoti A v vsakem času. Sistem CP/M Plus podpira v celoti 16 bank in omogoča tako vrsto podatkovnih vmesnikov; ta metoda se bo izkazala kot zelo uporabna in smiselna.

Kot smo že povedali, se v prvi banki nahajata dela programskih modulov BDOSa in BIOSa. BDOS dobavlja Digital Research v obliki premestljivega sistemskega programa (SPR pomeni System Program Reloctable). Obstajajo tri zbirke: BDOS3.SPR je popoln BDOS za uporabo v nebančnem CP/M Plus sistemu; zbirki BNKBDOS3.SPR in RESBDOS3.SPR sta bančni in nebančni del BDOSa. Ti moduli se berejo s programom GENCPM, ki je storitev za sistemsko generiranje in postanejo del generiranega sistema.

3.3. Bančna delitev BIOSa

Delitev BIOSa na banke opravimo tako, da ugotovimo segmente, ki bodo dostopni v posameznih bankah. Tisti del BIOSa, ki bo dostopan v več kot eni banki, naj bi ostal rezidenten (nebančni, glej sliko 4). V rezidentnem delu se nahaja tudi rutina za izbiro bank in deli diskovnih čitalnih in pisalnih rutin, ki opravljajo prenos podatkov. Rezidentne so lahko tudi rutine za konzolne in tiskalne operacije, saj bodo te klicane iz TPA (banka št. 1) in iz BDOSa (banka št. 0). Ti segmenti so rezidentni in njim predhodi ukaz CSEG v zbirnem izvirnem kodu. Druge diskovne uslužnostne rutine, kot so izbira diska (Select Disk), nastavitev steze (Set Track), nastavitev sektorja (Set Sector) itd. in segmenti za odkrivanje in odpravljanje napak pa se lahko nahajajo v bančnih delih pomnilnika. Tem segmentom predhodi ukaz DSEG v zbirnem izvirnem kodu. Ko se tako modificirani BIOS prevede z zbirnikom RMAC (tj. MAC zbirnik za premeščanje) in poveže s pomočjo storilnostne rutine LINK, vsebuje rezultirajoča zbirka BNKBIOS3.SPR bančne in nebančne dele koda, ki so urejeni v dva ločena segmenta. Program GENCPM bo razpoznal ta segmenta in bo ustrezno razdelil BIOS v bančni in rezidentni del.

V zvezi z bančnim BIOSom so potrebne še nekatere minimalne spremembe v obstoječem kodu. BDOS bo vselej poklical diskovni V/I BIOS z vklop-



Slika 4. Tipični štiribančni model sistema CP/M Plus (ki pa ni maksimalen)

ljeno banko št. 0. To je dogovor. V rezidentnem delu BIOSa se nahajata rutini za diskovno branje in pisanje, potreben pa bo še poziv rutine pomnilniškega upravljanja, da se bo vključila ustrezna banka za podatkovni prenos. Po branju ali pisanju v določen sektor se mora ponovno vključiti banka št. 0 pred vrnitvijo v BDOS. Bančna številka, ki mora biti uporabljena za diskovno branje in pisanje se posreduje z novo BIOS rutino z imenom Set Bank (nastavitev banke), ki posreduje vselej bančno številko med 0 in 15. Ta številka se shrani skupaj s številkami steze in sektorja in z DMA naslovom, da bi bila lahko uporabljena pri diskovnem branju in pisanju. Zato bo potrebna izdelava rutine za izbiro pomnilnika (Select Memory), ki se pokliče z bančno številko in mora opraviti materialno odvisno opravilo preklopitve na ustrezno pomnilniško banko. Nazadnje bo potrebno izdelati še rutini MOVE in XMOVE, ki kopirata pomnilniški blok v okviru dane banke (MOVE) ali med dvema bankama (XMOVE). Procesor Z80 ima na voljo ustrezne ukaze za bločni prenos.

3.4. Prednosti bančnega sistema

Prva prednost bančnega sistema je v tem, da je za uporabniške programe na razpolago več pomnilnika. Večina operacijskega sistema biva v posebni pomnilniški banki in tako ne obremenjuje TPA prostora kot sistem CP/M 2.2. Tudi bančni BDOS ima nove lastnosti, ki so za uporabnika prijaznejše in bolj raznovrstne. Obstaja zaslonsko usmerjen urejevalnik, ki je vgrajen v ukaz branja konzolnega vmesnika in se uporablja pri vstopu ukazov pri CCP, PIP in drugih storitvah. Ta editor omogoča ponoven vpoklic zadnje vrstice konzolnega vhoda, omogoča pomikanje kurzorja z možnostmi vstavljanja in brisanja. Modificirana ukazna vrstica se potem ponovno obdela. To pa skrajšuje tipkanje pri podobnih ukazih, popraviti pa je mogoče tudi napako, ki se je pojavila na začetku dolge vrstice. Tudi sporočila o napakah so daljša in bolj izrazita, pokaže se številka BDOS funkcije in ime zbirke, kjer se je napaka pojavila. Naslednja bistvena lastnost bančnega BDOSa je zaščita zbirk z gesli. S tem je zaščitena zasebnost zbirk pri uporabi sistema z več uporabniki. Prav tako je moč zbirke zaščititi pred izbrisom, pisanjem ali branjem pred nepoklicanimi osebami.

3.5. Tabele in vmesniki

K opisanim vidnim zunanjim lastnostim sistema CP/M Plus je treba dodati še prednosti vrste notranjih izboljšav v BDOSu. Te izboljšave bistveno prispevajo k hitrosti večine diskovnih operacij. Te izboljšave se pokažejo tudi pri nebančnem sistemu, vendar je pri bančnem sistemu omogočena njihova polna uporaba, ki ne gre na račun dragocenega TPA prostora. Na poseben način se lahko GENCPM določi obseg in mesto vseh teh tabel.

Dve vrsti tabel, ki ju lahko pomaknemo v banke in sta znani iz sistema CP/M 2.2, sta vektorja preizkusnih vsot in dodeljevanja. Vektor preizkusnih vsot je tabela, ki vsebuje enozložne preizkusne vsote za vsak sektor v diskovnem imeniku. Med imeniškimi operacijami uporablja CP/M te preizkusne vsote za razpoznavanje različnih disket, ki so bile vstavljene v enoto. Razlika pri sistemu CP/M Plus je v tem, da je moč vektorje preizkusnih vsot pomakniti v banko št. 0 in tako razbremeniti TPA prostor.

Druga znana tabela je dodeljevalni vektor. Ta tabela se uporablja za evidenco zasedenih blokov (dodeljevalnih skupin) na disku. V sistemu CP/M 2.2 in v nebančnem sistemu CP/M Plus se uporablja en bit za evidenco vsakega diskovnega bloka. Ta bit se popravlja vsakokrat, ko je nov blok uporabljen in celotna tabela se rekonstruira pri toplem zagonu oziroma pri vstavitvi nove diskete.

V bančnem sistemu CP/M Plus se uporabljata dva bita za vsak blok. Prvi bit kaže uporabljenost bloka za zbirko, ki še ni bila zaprta; drugi bit pove, da je bila zbirka, povezana s tem blokom zaprta in tako potrjuje permanentno dodelitev. Ta dvobitna shema omogoča sproščanje blokov, ki so bili dodeljeni izvajanju trenutnega programa, ko se enota resetira ali ko je program izstopil brez zapiranja svojih zbirk. Tako ni več potrebno preiskovanje imenika (kot v sistemu CP/M 2.2) za prestrukturiranje dodeljevalnega vektorja med toplim zagonom. Ta pristop povzroči izdaten časovni prihranek pri toplem zagonu. V primeru, da se CCP prenese iz sosednje banke, postane tkim. topli zagon navidezno odvečen.

Imeniško sekljanje (hashing) je nova lastnost sistema CP/M Plus. V rutini GENCPM je s pomočjo dialoga moč izbrati sekljanje na nekaterih ali na vseh diskovnih enotah. To povzroči nastanek dodatne tabele s štirimi zlogi za vsak imeniški vstop, ki se dodelijo z rutino GENCPM in se oblikujejo, ko se posamezna disketa vpiše. To tabelo uporablja BDOS za direkten izračun lokacije zbirke v imeniku, tako da ni več potrebno zaporedno preiskovanje imenika kot v prejšnjem BDOSu. Povečanje hitrosti je tu znatno pri odpiranju, zapiranju, preimenovanju in brisanju zbirk, saj nastopi vselej operacija nad enim samim sektorjem.

Nadaljna notranja izboljšava BDOSa je uporaba domiselne sheme hitrih vmesnikov (cache) pri diskovnem dostopu. Med izvajanjem rutine GENCPM se lahko vsaki enoti v sistemu dodeli množica (bazen) vmesnikov. Ločeni bazeni se vzdržujejo za imeniške in podatkovne sektorje. Vsak bazen se lahko dodeli eni sami enoti ali pa je skupen za več diskovnih enot po prosti izbiri. Vsak bazen lahko vsebuje do 255 vmesnikov in celotno število vmesnikov bo omejeno le z razpoložljivim pomnilnim prostorom. Imeniški vmesniki se namestijo v banko št. 0 (kot kaže slika 4) tako da so lahko dostopni za imeniške BDOS funkcije. Podatkovni vmesniki se lahko predvidijo v bankah št. 2 do 15.

Hitri vmesniki (cache) se uporabljajo z namenom, da se preprečijo ponovna branja že prebranih diskovnih sektorjev. Ko je bil sektor enkrat prebran ali vpisan, ostane v pomnilniku. Ponovno branje istega sektorja bo uporabilo obstoječo pomnilniško kopijo in tako ne bo potrebno ponovno branje z diska. Ko sistem uporabi vse razpoložljive vmesnike, sprosti najmanj uporabljenega v zadnjem razdobju. Obstaja poseben BDOS poziv (rutina) za sproščanje vmesnikov. Programi, kot so PIP z verifikacijsko možnostjo, pa morajo opravljati vsakokraten diskovni dostop.

Učinek te vmesniške sheme na prevajalnike, tekstovne procesorje, pakete podatkovnih baz in druge programe, ki uporabljajo ponovne dostope k istim podatkom, je izredno zanimiv. Ko je bil enkrat opravljen prehod skozi zbirko, se na disk praktično ne dostopa, če ni vpisa v zapis. Programi s plastmi se izvajajo hitreje, ker lahko plast ostane v pomnilniku še od svoje prejšnje uporabe. Sistem hitrih vmesnikov daje videz, kot da se program izvaja v velikem pomnilnem prostoru.

4. Sklep

Sistem CP/M Plus je veliko bolj zmogljiv kot sistem CP/M 2.2 in brškone dosega mejno zmogljivost za 8-bitne mikroprocesorje. S tem pa se podaljšuje tudi življenska doba uporabe 8-bit-

nih procesorjev v prihodnosti. Te njegove izjemne zmogljivosti pa je potrebno v celoti prenesti tudi na aplikativno programsko opremo. Aplikativni programi se vobče lahko izvajajo na novem sistemu, ne morejo pa se izvajati nekateri diskovni storitveni programi. Ena glavnih prednosti sistema CP/M Plus je povečana hitrost diskovnega dostopanja, ki je pogojena z dodatnim (bančnim) pomnilnikom.

Implementacija pomnilniškega upravljanja je tako priporočljiva. Pohitritev diskovnega dostopa pa je zlasti pomembna pri velikih (vinčestrskih) diskih. Bistveno je tudi povečanje TPA prostora, kar omogoča priročnejše izvajanje daljših uporabniških programov brez plastenja.

Slovstvo

((1)) A.P.Železnikar: Uvod v CP/M I. Informatica 5 (1981), št. 3, str. 63-76.
((2)) A.P.Železnikar: Uvod v CP/M II. Informatica 5 (1981), št. 4, str. 9-23.
((3)) A.P.Železnikar: Uvod v CP/M III. Informatica 6 (1982), št. 1, str. 33-42.
((4)) CP/M Plus (CP/M Version 3) Operating System: User's Guide (1982), Digital Research.
((5)) CP/M Plus (CP/M Version 3) Operating system: System Guide (1982), Digital Research.
((6)) CP/M Plus (CP/M Version 3) Operating System: Programmer's Guide (1982), Digital Research.
((7)) CP/M Plus (CP/M Version 3) Operating System:. Programmer's Utilities Guide for the CP/M Family of Operating Systems (1982), Digital Research.
((8)) D.Hardy, K.Jackson: CP/M Plus: An Overview for CP/M 2.2 Users. Microsystems 4 (1983), No. 2, pp. 20-25.
((9)) B.R.Ratoff: Implementing the Advanced Features of CP/M Plus. Microsystems 4 (1983), No. 2, pp. 26-29.

---

* CP/M Plus in CP/M sta zaščitna znaka podjetja Digital Research, P.O.Box 579, 160 Central Avenue, Pacific Grove, CA 93950, U.S.A.

****
L I S A je nov dosežek
****

Podjetje Apple je dalo v prodajo svoj nov enouporabniški osebni računalnik z imenom Lisa. Ta grafično usmerjena in z miško vodena delovna postaja je namenjena predvsem pisarniškemu delu (po naše administrativnemu delu), njena cena pa je $ 9995. Lisa je podobna sistemu Star podjetja Xerox, je pa cenejša. Njena značilnost je tkim. lokalno integrirana programska arhitektura, ki omogoča njeno uporabnost z minimalnim vtipkavanjem prek tastature. Podobno kot sistem Star je tudi Lisa močno grafično usmerjena, z menujskim načinom komunikacije in z uporabo miške (ta miška se giblje prosto po mizi) za manipulacijo kurzorja (zaslonskega kazalca). Z miško se kurzor pomakne k določeni besedi ali sliki (Xerox imenuje tak objekt podoba) na zaslonu in s pritiskom na miškin gumb se potem nekaj zgodi, npr: prikaže se ukazni seznam, odpre ali zapre se zbirka, pomakne se beseda ali cel odstavek, generira se določen stolpec podatkov ali pa se oblikuje preglednica.

Osebni računalniki so se doslej že udomačili na določenih poslovnih področjih, pri poslovanju na veliko in na malo, v vladnih in upravljavskih pisarnah, v šolah in v raziskovalnih laboratorijih; cilj novega plasmaja osebnih računalnikov je sedaj poslovna eksekutiva in strokovnjaki (ekonomisti, inženirji, komercialisti, svetovalne strokovne službe).

Lisa uporablja mikroprocesor 68000, 12-colski črnobeli zaslon, 5MB vinčestrski disk in dve petinčetrtcolski enoti za upogljivive diske z obsegom 870KB, selektrično tastaturo z numeričnim dodatkom in miško. V to materialno opremo je integriranih šest programskih paketov:

- LisaList omogoča oblikovanje in vzdrževanje seznamov poljubnega tipa za osebno podatkovno bazo
- LisaCalc je modelirni pripomoček za razpredelnične pole in finance, kjer se pola lahko razprostira do 255 vrstic in do 255 stolpcev
- LisaProject je vidni pripomoček za projektno upravljanje, ki omogoča uporabniku videnje kritičnih poti, medsebojnih odvisnosti in postavljanje vprašanj tipa "kaj - če"
- LisaWrite je besedni (tekstni) procesor
- LisaGraph je namenjen poslovni grafiki
- LisaDraw je bistven del integrirane programske opreme in nudi menu črt, krogov, likov, za oblikovanje skic, shem, tehničnih diagramov in temu podobno

Lisa je rezultat dela 200 človek-let, od tega velik del za izdelavo programske opreme. Investicija v ta projekt je znašala 40 do 50 milijonov dolarjev, projekt pa se je začel v letu 1979. Na osnovi tega projekta je podjetje Apple ustanovilo poseben oddelek za osebne pisarniške sisteme s 100 inženirji v letu 1980. Kljub temu se pričakuje, da bo 90% programske opreme za Liso narejene izven podjetja Apple (neodvisni programski proizvajalci).

Podjetje Digital Research bo imelo CP/M za Liso, podjetje Microsoft pa operacijski sistem Zenix (verzija Unixa). Na vidiku so tudi visoki programirni jeziki Cobol, Basic, Pascal in Fortran. Iz drugih virov se bodo pojavili tudi aplikativni programi. Predviden je tudi paket AppleNet za lokalne mreže, s širino 1 Mbit in s podporo za 128 naprav. Ta paket naj bi omogočil povezavo z drugimi mrežami, kot so Ethernet in širokopasovne mreže vključno s storitvami podatkovnih baz in elektronske pošte.

Lisa je namenjena pisarniškemu tržišču, strokovnjakom, direktorjem in administrativnim pomočnikom. Namenjena je pa tudi manjšim in srednje velikim podjetjem z manj kot 200 milijoni dolarjev brutoprodikta na leto.

Podjetje Apple je v letu 1982 doseglo produkt 583 milijonov dolarjev, s cenami svojih sistemov med 500 in 6000 dolarji. Leto prej je imelo še 23% tržišča v ZDA, lani pa samo še 19% zaradi vstopa IBMa na to tržišče.

A.P.Železnikar

AN APPROACH TO THE COMPARISON OF MACHINE INSTRUCTION FORMATS

JOZO J. DUJMOVIĆ

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF BELGRADE, YUGOSLAVIA

UDK: 681.3.02

A simple quantitative model for evaluation and comparison of machine instruction formats is proposed. The suitability of instruction formats for assembly language programming is analyzed. Using various empirical studies of major high level languages an indicator of the average length of an assignment statement is derived. This indicator is then applied for organizing a formal criterion for evaluation and comparison of machine instruction formats.

JEDAN PRILAZ KOMPARACIJI FORMATA MAŠINSKIH INSTRUKCIJA. U radu se predlaže jednostavan kvantitativni model za vrednovanje i komparaciju formata mašinskih instrukcija. Model je zasnovan na analizi pogodnosti raznih formata instrukcija za programiranje na simboličkom mašinskom jeziku. Primenom više empirijskih studija glavnih viših programskih jezika izveden je pokazatelj prosečne dužine instrukcija dodele vrednosti. Ovaj pokazatelj se zatim primenjuje za realizaciju formalnog kriterijuma za vrednovanje i komparaciju formata mašinskih instrukcija.

INSTRUCTION FORMATS

Traditional organization of a machine instruction assumes an opcode field and up to three address filds referencing memory locations (denoted M) or explicitly addressable processor registers (denoted R). Let m denote the number of memory addresses per instruction, and r denote the number of general purpose register addresses per instruction. For zero-, one-, two-, and three-address instructions the ten possible instruction formats can be defined as shown in Table I.

Table I. Ten Instruction Formats

m \ r	0	1	2	3
0	∅	R	RR	RRR
1	MA	MR	MRR	-
2	MM, MMA	MMR	-	-
3	MMM	-	-	-

Instruction formats

The format ∅ denotes stack machines. In the case m=1, r=0 we assume a computer with an accumulator A that holds one operand and receives the computed result (this explains the symbol MA). In the case m=2, r=0, both the version with an accumulator (MMA) and the version without the accumulator (MM) are possible. The formats R, RR, and RRR can be considered auxiliary formats since they are used either as special cases derived from instruction formats having m>0, or the registers contain the addresses of memory locations to be referenced and consequently such R-formats are equivalent to the corresponding M-formats. Of pourse, the availability of general purpose registers enables the efficient handling of intermediate results of arithmetic operations, but the M--fields are crucial since they enable fetching of operands from memory. Some machines have only one instruction format, e.g. the only instruction format of the IBM 1130 is MA. Modern computers, however, more frequently have multiple instruction formats [6]. For example, the instruction formats of the DEC PDP-11 are MM, MR, and RR. Similarly, the instruction formats of the IBM/370 are MM, MR, RR, and MRR. Since the assembly language programming is easier in the cases with more various logical instruction formats the number of available formats can be used as one of criteria for comparison of competitive processors.

AVERAGE LENGTH OF AN ASSIGNMENT STATEMENT

Various instruction formats yield various levels of complexity of assembly language programs. Of course, from a programmer's standpoint short programs are more convenient than longer ones. Consequently, the length of the program can be used for comparison of various instruction formats. This immediatelly rises the issue of selecting a representative ("typical") program.

In the area of high level languages the studies of typical real-life programs included programs written in Algol [1] , Fortran [2,3] Cobol [4,5,6] , and PL/1 [7,8] . These analyses showed that the simplest programming patterns are the most frequent in practical programming. Consequently, the majority of executed statements are assignments. According to Knuth [2] the dynamic frequency of assignment statements is 67%. Static frequencies of assignments show a rather consistent pattern:

Knuth [2] (Fortran)	41% - 51%
Robinson & Torsun [3] (Fortran)	36.1%- 38.1%
Al-Jarrah & Torsun [5] (Cobol)	37.6%- 46.3%
Strebendt (Kuck) [6] (Cobol)	49.6%
Elshoff [7] (PL/1)	41.2%

Therefore, we believe that the risk of oversimplification is sufficiently low if the

Table II. Assembly language programs for an assignment statement

	m=0 (format ∅)	m=1 (format MA)	m=2 (format MMA)	m=3(format MMM)
	push M_1	A := M_1	A := M_1*M_2	T := M_1*M_2
	push M_2	A := A*M_2	A := A*M_3	T := T*M_3
	*	A := A*M_3	A := A*M_4	T := T*M_4
	push M_3	A := A*M_4	A := A*M_5	T := T*M_5
	*	...	...	...
	...	...	...	...
	push M_{k-1}	...	...	...
	*	A := A*M_{k-1}	A := A*M_{k-1}	T := T*M_{k-2}
	pop M_k	M_k := A	M_k := A	M_k := T*M_{k-1}
$L_k(m)$	2k-2	k	k-1	k-2

Table III. The average length of an assignment statement

k	p_k	$L_k(0)$	$p_kL_k(0)$	$L_k(1)$	$p_kL_k(1)$	$L_k(2)$	$p_kL_k(2)$	$L_k(3)$	$p_kL_k(3)$
2	0.43	2	0.86	2	0.86	1	0.43	1	0.43.
3	0.30	4	1.20	3	0.90	2	0.60	1	0.30
4	0.12	6	0.72	4	0.48	3	0.36	2	0.24
5	0.05	8	0.40	5	0.25	4	0.20	3	0.15
6	0.05	10	0.50	6	0.30	5	0.25	4	0.20
7	0.05	12	0.60	7	0.35	6	0.30	5	0.25
		L(0) = 4.28		L(1) = 3.14		L(2) = 2.14		L(3) =1.57	

comparison of instruction formats is based on the assembly language programs which realize assignments. Let M_1, M_2,..., M_k denote a sequence of memory addresses. The general assignment statement can be defined as follows:

$$M_k := M_{k-1}*M_{k-2}*\ldots*M_2*M_1 \quad , \quad k>1,$$

where * denotes an arbitrary associative binary operation. Of course, all sizes of assignment statements are not equally frequent. The analysis of Robinson and Torsun [3] showed the following relative frequencies for large application programs:

k=2 k=3 k=4 k≥5

p_2=0.43 p_3=0.3 p_4=0.12 $p_5+p_6+\ldots$=0.15 .

This distribution is consistent with Knuth's measurement [2] showing for various samples p_2=0.45 , 0.49 , and even 0.68. Since p_5, p_6,.. were not provided by the original measurement [3] we will adopt the values $p_5=p_6=p_7$=0.05 since they yield

$$\bar{k} = 2p_2+3p_3+4p_4+5p_5+6p_6+7p_7 = 3.14$$

which is consistent with the average length of assignment statement, $\bar{k}$=3.2 , measured by Wichmann [1] .

The general assignment statement is programmed in assembly language in the way shown in Table II. The bottom line in this table shows the total number of assembly language instructions, i.e., the length of the program $L_k(m)$. The average length of the assembly language program realizing an average assignment statement is

$$L(m) := \sum_{k=2}^{7} p_kL_k(m)$$

and can be computed according to Table III.

Therefore, in the best case of the three--address format an average assignment can be realized with 1.57 assembly language instructions. In the worst case corresponding to a stack machine the necessary number of assembly language instructions is 4.28.

EVALUATION AND COMPARISON OF INSTRUCTION FORMATS

Evaluation and comparison of instruction formats represents a step in the computer evaluation and selection process. Using the LSP method for system evaluation [9,10] a criterion for system evaluation is organized as a logical aggregation of a number of elementary criteria. One among the elementary criteria is the elementary criterion for the evaluation of instruction formats.

An elementary criterion is defined as a mapping of a value of a performance variable into the corresponding value of elementary preference. The performance variable represents a relevant system performance indicator influencing the system capability to satisfy some given requirements. In the case of instruction format the performance variable is m and the requirement is to achieve the shortest possible assembly language program. The length of an assembly language program is approximated by the average length of an assignment statement, L(m), and therefore L(m) can be used for evaluation and comparison of various instruction formats. The corresponding elementary preference E_m is rigorously interpreted as the degree of truth in the statement asserting that "the value of m (or L(m)) completely fulfills all given requirements" (thus $0 \le E_m \le 1$). Approximately, the elementary preference can be interpreted as a percentage of fulfilled requirements. Consequently, the elementary criterion for instruction format evaluation is a function $m \mapsto E_m$.

A rational way to assign preferences (E_0, E_1, E_2, and E_3) to various instruction formats is to assign the maximum preference (i.e. 1) to the three-address format, and to assign the minimum preference E_{min} to the zero-address format. This reasoning yields the following elementary criterion:

$$E_m := \frac{L(0) - L(m) + E_{min}[L(m) - L(3)]}{L(0) - L(3)} ,$$

$$m = 0, 1, 2, 3.$$

The minimum preference E_{min} reflects the evaluator's standpoint, i.e. the requirements of some specific environment. One way to select E_{min} is to take into account that some real stack machines are restricted only to high-level languages [11]. Of course, in that case the inconvenience of assembly language is not the only reason for not allowing general programmers to use the assembly language. That allows us, however, to conclude that in an extreme (but not unrealistic) case one can adopt $E_{min}=0$ yielding the results

$$E_0=0, \quad E_1=0.42, \quad E_2=0.79, \quad E_3=1 ,$$

Several alternative criteria for instruction format evaluation are shown in Fig. 1.

CONCLUSION

From 40% to 50% of statements in programs written in high-level languages are assignments. The assignments also dominate in assembly language programs. The average number of operands per an assignment statement is approximately $\bar{k} = 3.14$. The length of an assembly language program depends on the number of addresses per machine instruction, m, and can be approximated by the length of the assembly language program realizing an average assignment statement, L(m). For the single-address machine format the length of an average assignment statement is $L(1) = \bar{k}$, i.e. it is nesessary to write one assembly language instruction per each operand. For other instruction formats we have

$$L(0) = 1.36\ \bar{k} \doteq \frac{e}{2} L(1)$$

$$L(2) = 0.68\ \bar{k} \doteq \frac{e}{4} L(1)$$

$$L(3) = 0.50\ \bar{k} \doteq \frac{1}{2} L(1) \quad \text{(and } L(1) \doteq \pi\text{)}.$$

Therefore, the machine instruction formats can be compared approximately as follows:

(1) The two-address format is two times more efficient than the zero-address format, and the three-address format is two times more efficient than the single-address format, and (2) the single-address format yields 27% shorter programs than the zero-address format, the two-address format yields 32% shorter programs than the single-address format, and the three-address format yields 27% shorter programs than the two-address format. These relations can be useful both for assessing various assembly language programming efforts, and for comparing different processor architectures.

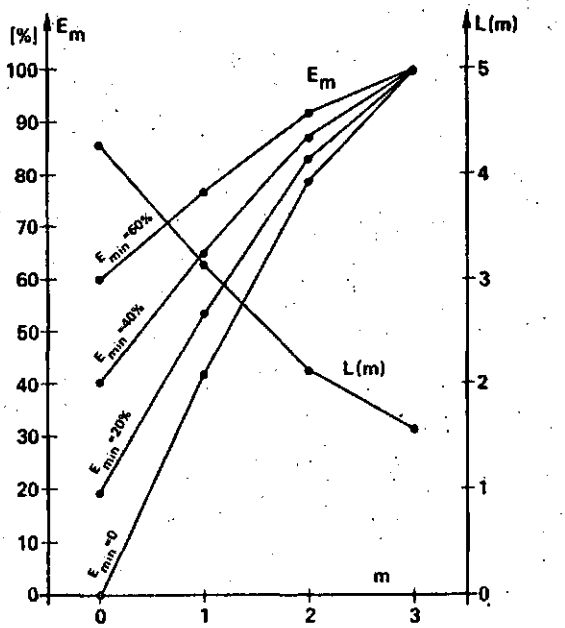


Figure 1. Criteria for instruction format evaluation

REFERENCES

[1] B.A. Wichmann, "Algol 60 Compilation and Assessment." Academic Press, 1973.

[2] D.E. Knuth, "An Empirical Study of Fortran Programs." Software - Practice and Experience, 1, pp. 105-133, 1971.

[3] S.K. Robinson and I.S. Torsun, "An Empirical Study of Fortran Programs." The Computer Journal, 19, 1, pp. 56-62, 1975.

[4] A. Salvadori et al,"Static Profile of Cobol Programs." Sigplan N. 10,pp.20-23, Aug.1975.

[5] M.M. Al-Jarrah and I.S. Torsun, "An Empirical Analysis od Cobol Programs." Software--Practice and Experience 9,pp.341-359,1979.

[6] D.J. Kuck, "The Structure of Computers and Computations." Vol. 1. John Wiley, 1978.

[7] J.L. Elshoff, "A Numerical Profile of Commercial PL/1 Programs." Software,6, 1976.

[8] J.L. Elshoff, "An Analysis of some Commercial PL/1 Programs."IEEETSE, June 1976.

[9] J.J. Dujmović, "The Preference Scoring Method for Decision Making - a Survey, Classification and an Annotated Bibliography." Informatica, No.2, pp. 26-34, 1977.

[10] J.J. Dujmović, "Computer Selection and Criteria for Computer Performance Evaluation." International J. of Computer and Information Sciences, 9, No.6,pp.459-482, 1980.

[11] W,M, McKeeman, "Stack Computers." Chapter 7 in H. Stone (Ed.) "Introduction to Computer Architecture" Second Edition, SRA, 1980.

INFORMATICA 1/1983

PRODUCTION PLANNING BY LOMP

IVAN MEŠKO,
BOJAN PEVEC

VEKŠ MARIBOR
MESNA INDUSTRIJA MURSKA SOBOTA

UDK: 681.3.06:338

PRODUCTION PLANING BY LOMP. The production process can be clearly illustrated by the graph. For this reason the process must be broken down into several production unite. By using the graph one can construct the mathematical model for optimization, for which it is assumed to be linear, because for otherwise it would not be applicable to large syatems. Optimizing the production system by its subsystems is disadvantageous because of the sinergetic effects. That is to say, in general one cannot construct the optimum of the system by putting together the optimums of its subsystems. In the processing industry, from which an example is taken, the method LOMP is used. This is the method of Linear Optimisation of the Multiphase Production.

OPTIMIRANJE PROIZVODNJE Z METODO LOMP. Proizvodni proces je mogoče nazorno prikazati z grafom, če ga razčlenimo na posamezne tehnološke postopke oziroma načine proizvodnje. S pomočjo grafa nato konstruiramo matematični model za optimiranje, ki mora biti linearen, sicer ni uporaben za velike sisteme. Optimiranja proizvodnje sistema zaradi sinergijskih učinkov ni primerno izvesti s pomočjo podsistemov. V splošnem namreč ne moremo dobiti optimuma celotnega sistema s sestavljanjem optimumov podsistemov. V predelovalni industriji, od koder je vzet primer, uporabljamo metodo linearnega optimiranja multifazne proizvodnje, to je metodo LOMP.

1. Mathematical model

We break down the production process into several elementary processes. For the requirements of the optimization, we distinguish the elementary processes only according to the quantity of the elements consumed or produced per unit of production, where the elements include production elements and products.

Denote by N_i the index set of the elementary processes at which the i-th element is consumed, by P_i the index set of the elementary processes at which the i-th element is produced and by x_j the level of production for the j-th elementary process. Decision variables x_j then satisfy the following condition:

$$\sum_{j\in P_i} a'_{ji}x_j - \sum_{j\in N_i} a_{ij}x_j + e_i - e'_i \geq 0 \qquad (1)$$

where e_i represents the quantity purchased and e'_i the quantity sold of the i-th element, a_{ij} the input coefficient and a'_{ji} the output coefficient of the j-th elementary process.

If the production of the i-th element (first sum in (1)) is greater than the quantity of the element consumed (second sum in (1)), then it follows:

$$e_i = 0 \qquad e'_i > 0 \qquad (2)$$

If

$$\sum_{j\in P_i} a'_{ji}x_j < \sum_{j\in N_i} a_{ij}x_j$$

holds, then it follows

$$e_i > 0 \qquad e'_i = 0 \qquad (3)$$

Both requirements are automatically fulfilled if the objective function is suitably formulated.

If there are more or less favourable sources and sinks of the i-th element instead of (1) we get the inequality:

$$\sum_{j\in P_i} a'_{ji}x_j - \sum_{j\in N_i} a_{ij}x_j + \sum_{k\in B_i} e_{ik} - \sum_{k\in B_i} e'_{ik} \geq 0 \qquad (4)$$

where B_i means the index set of sources, S_i the index set of sinks for the i-th element, e_{ik} means the quantity of the i-th element supplied by the k-th source and e'_{ik} the quantity of the i-th element, supplied to the k-th sink.

When the available capacity of the k-th source is bounded by b_{ik} and when the need of the k-th sink is bounded by b'_{ik}, we must consider also the following constraints:

$$e_{ik} \leq b_{ik} \qquad k \in B_i \tag{5}$$

$$e'_{ik} \leq b'_{ik} \qquad k \in S_i \tag{6}$$

Denote by c'_{ik} the price of the i-th element at the k-th sink reduced for the marginal cost of the sale. Denote by c_{ik} the price of i-th element of the k-th source increased for the marginal cost of purchase. Let us set for the objective the profit, defined as

$$z = \sum_i \Big(\sum_{k \in S_i} c'_{ik} e'_{ik} - \sum_{k \in B_i} c_{ik} e_{ik} \Big) - \sum_j m_j x_j \tag{7}$$

where the third sum contains those variable costs which are not included in the second sum.

The objective function (7) ensures that we will utilize first of all the most favourable sources and that we will satisfy the most favourable sinks. When we find this unsatisfactory, because we want to favour some sources or sinks we can achieve that by introducing additional constraints in the form of equalities or inequalities. We can also use the methods of multiobjective programming. The variables e'_{ik} and e_{ik} appear only in (7), in (4) and possibly in (5) or (6). Because of the marginal cost of sale and that of purchase, for the coefficients in (7) $c'_{ik} < c_{ik}$ holds, from where it follows (2) or (3).

If the coefficients a_{ij} or a'_{ji} depend on the production level of the j-th elementary production process and if they may not be approximated by the constant then we must take them piecewise constant and linearize the left side of equations (1) or (4). The same holds for m_j.

2. Realization of optimization

Because of the development of technology production systems become more complicated and mutually dependent. For this reason we break them down and present them on a graph. In this graph each element E_i has an allocation node which is represented by the circle. To the elementary process X_j the transformation node X_j is arranged which is represented by the square. The value of the arc (E_i, X_j) is a_{ij} and the value of the arc (X_j, E_i) is a'_{ji}.

The technological data which in the graph are presented by the values of the arcs, are used for the formulation of the constraints (1) or (4). Furthermore we also need the source and the sink data, so that variables e_{ik}, e'_{ik} and the constraints (5), (6) can be defined and that the coefficients of the objective function can be determined.

The constraint (4) is general, but for special cases it can be simplified. For instance, if the i-th element cannot be produced, the first and the fourth sum disappear. So we get the constraint:

$$-\sum_{j \in N_i} a_{ij} x_j + \sum_{k \in B_i} e_{ik} \geq 0 \tag{8}$$

Additional simplification is possible if there is only one source, so that the second sum in (8) consists of only one term. From the analogy of (1) we get

$$-\sum_{j \in N_i} a_{ij} x_j + e_i \geq 0 \tag{9}$$

The inequation (9) can be simplified, if the coefficient of the variable e_i in the objective function equals 0. This happens when the i-th element represents the appliance with given capacity b_i which could not be hired. In this case there is no need to use the variable e_i and instead of (9) we get

$$\sum_{j \in N_i} a_{ij} x_j \leq b_i \tag{10}$$

So the model is simplified and the computing is shortened and this would be of great importance for large models.

The constraint (9) can be also written in the form (10) although the coefficient in the objective function belonging to E_i is not equal to 0. In that case the consumption of i-th element in the objective function must be considered in the marginal costs m_j of all those elementary processes at which this element is consumed and the variable e_i is not defined. In this way the model is simplified and computer time is shortened, although the coefficients m_j are harder to compute. We can

avoid that problem so that for all elements which are sold or bought the variables e_i or e_{ik} are included in the model. In this case $m_j=0$ for all elementary production processes. In this way the optimal solution is more complete. However, more work is left to the computer, especially if large models are considered. If the graph contains around 1000 transformation and 1000 allocation nodes, which happens when optimizing the production of an average-sized company, the computing time will be of the order of about 10 hours.

The constraint (4) is simplified in some other cases too. If the i-th element is a semi-product which cannot be sold, bought or stored, the following constraint arises:

$$\sum_{j\in P_i} a'_{ji}x_j - \sum_{j\in N_i} a_{ij}x_j \geqq 0 \qquad (11)$$

With large models, inequations are not written down at all, for this is almost impossible to do. Those data which can be read from the graph are entered directly into the computer. The control of the data and check of the reality of the results is done by means of the graph as well.

Using the computer and the representation of the production process by the graph makes it possible for the model to be successfully used at some production enterprises. At the "Mesna industrija Murska Sobota" two years ago, a graph with 693 allocation and 516 transformation nodes was constructed. The graph was drawn on several pages of format A0. In order to get a useful result the model had been improved several times and further nodes were added. When the graph is constructed, it is possible to work out in two days the production, purchase and selling plan of the company producing 200 different final products [3]. If the planned financial result is not realised, it is now possible to find the reasons and to quantify their consequences.

3. Example

Let us consider a simplified example from the milk industry. The data can be seen from the tables. For the input elements the following data are stated for each source: the variable representing the purchase quantity searched for, the minimal and the maximal purchase quantity and finally the purchase price increased for an eventual marginal cost of purchase.

For the products the following is stated for each sink: the variable representing the quantity of sale searched for, the minimal and the maximal quantity of sale and the sale price decreased for an eventual marginal cost of sale. For the capacities of the machinery, the capacities and marginal operating costs are stated. For the technological procedures the following is stated: the normatives of the input and autput elements and those marginal costs which are not included in the input elements and machinery capacities. The data for the semi-products are seen from the graphical representation of the production.

Survey of input elements

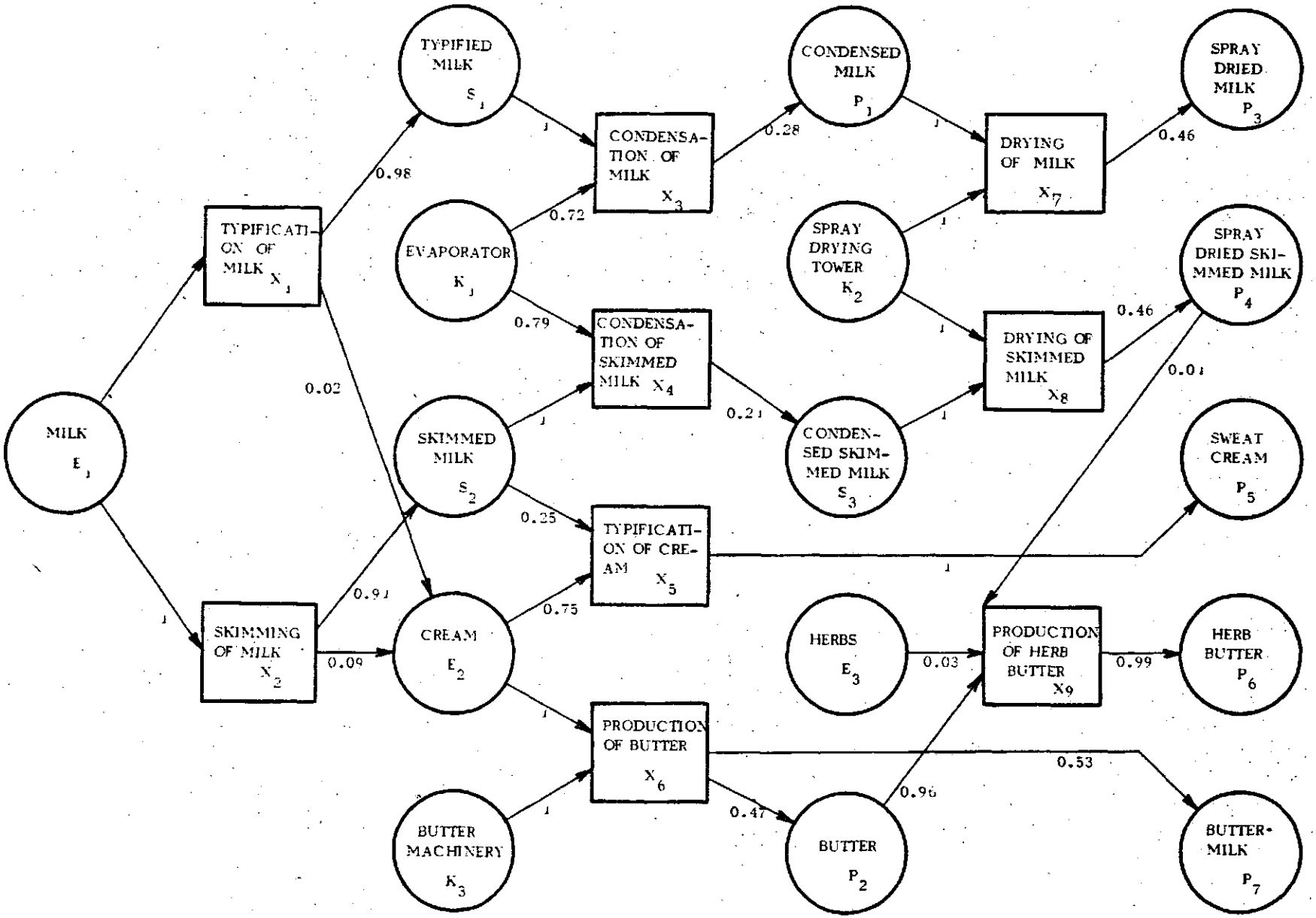
Element	Variable	Min t	Max t	Price m.u./t
E_1 Milk	e_1	15 000	20 000	20
E_2 Cream	e_{21}	1 000	1 500	140
	e_{22}	0	1 000	150
E_3 Herbs	e_3	0	∞	200

Survey of equipment capacities

Element	Capacity	Cost
K_1 Evaporator	14 000 m^3 H_2O	5m.u./m^3
K_2 Spray drying tower	5 000 t	4m.u./t
K_3 Butter machinery	2 500 t	0

Survey of final products

Element	Variable	Min t	Max t	Price m.u./t
P_1 Condensed milk	p_{11}	0	∞	100
	p_{12}	0	500	120
P_2 Fresh butter	p_2	0	∞	380
P_3 Spray dried milk	p_3	500	1500	220
P_4 Spray dried skimmed milk	p_{41}	300	∞	150
	p_{42}	0	500	160
P_5 Sweet cream	p_5	0	∞	150
P_6 Herb butter	p_6	0	500	430
P_7 Buttermilk	p_7	0	∞	6


MILK
E_1
TYPIFICATION OF MILK
X_1
SKIMMING OF MILK
X_2
0.02
0.98
0.09
0.91
TYPIFIED MILK
S_1
SKIMMED MILK
S_2
CREAM
E_2
BUTTER MACHINERY
K_3
EVAPORATOR
K_1
1
0.72
0.79
0.25
0.75
1
1
CONDENSATION OF MILK
X_3
CONDENSATION OF SKIMMED MILK
X_4
TYPIFICATION OF CREAM
X_5
PRODUCTION OF BUTTER
X_6
0.28
0.21
0.47
CONDENSED MILK
P_1
CONDENSED SKIMMED MILK
S_3
BUTTER
P_2
1
1
1
1
1
SPRAY DRYING TOWER
K_2
HERBS
E_3
0.03
0.96
DRYING OF MILK
X_7
DRYING OF SKIMMED MILK
X_8
PRODUCTION OF HERB BUTTER
X_9
0.46
0.46
0.01
0.99
0.53
SPRAY DRIED MILK
P_3
SPRAY DRIED SKIMMED MILK
P_4
SWEAT CREAM
P_5
HERB BUTTER
P_6
BUTTER-MILK
P_7

Survey of technological procedures

	Tehnological procedure	Input Element	Input a_{ij}	Output Element	Output r_{ij}	Cost m.u./t
x_1	Typification of milk	E_1	1	S_1 E_2^2	0,98 0,02	1
x_2	Skimming of milk	E_1	1	S_2 E_2^2	0,91 0,09	2
x_3	Condensation of milk	S_1 K_1	1 0,72	P_1	0,28	0
x_4	Condensation of skimmed milk	S_2 K_1	1 0,79	S_3	0,21	0
x_5	Typification of cream	E_2 S_2	0,75 0,25	P_5	1	0
x_6	Production of butter	E_2 K_3	1 1	P_2 P_7	0,47 0,53	20
x_7	Drying of milk	P_1 K_2	1 1	P_3	0,46	1
x_8	Drying of skimmed milk	S_3 K_2	1 1	P_4	0,46	1
x_9	Production of herb butter	P_2 P_4 E_3	0,96 0,01 0,03	P_6	0,99	15

On the basis of the data which can be seen from the tables, the graph is constructed and that is followed by the mathematical model. The model is expressed in the form of the maximum of the function:

$$z = -20e_1-140e_{21}-150e_{22}-200e_3+5k_1+4k_2-x_1-$$
$$-2x_2-20x_6-x_7-x_8-15x_9+100p_{11}+120p_{12}+$$
$$+380p_2+220p_3+150p_{41}+160p_{42}+150p_5+$$
$$+430p_6+6p_7$$

where the variables are nonnegative and they satisfy the constraints:

$$-x_1-x_2+e_1 \geq 0 \qquad \text{E1G (12)}$$
$$15000 \leq e_1 \leq 20000 \qquad \text{E1L, E1G2 (13)}$$
$$0,02x_1+0,09x_2-0,75x_5-x_6+e_{21}+e_{22} \geq 0 \qquad \text{E2G1 (14)}$$
$$1000 \leq e_{21} \leq 1500 \qquad \text{E2L1, E2G2 (15)}$$
$$e_{22} \leq 1000 \qquad \text{E2L2 (16)}$$
$$-0,03x_9+e_3 \geq 0 \qquad \text{E3 (17)}$$
$$0,72x_3+0,79x_4 \leq 14000 \qquad \text{K1 (18)}$$
$$x_7+x_8 \leq 5000 \qquad \text{K2 (19)}$$
$$x_6 \leq 2500 \qquad \text{K3 (20)}$$
$$0,98x_1-x_3 \geq 0 \qquad \text{S1 (21)}$$
$$0,91x_2-x_4-0,25x_5 \geq 0 \qquad \text{S2 (22)}$$
$$0,21x_4-x_8 \geq 0 \qquad \text{S3 (23)}$$
$$0,28x_3-x_7-p_{11}-p_{12} \leq 0 \qquad \text{P1G (24)}$$
$$p_{12} \leq 500 \qquad \text{P1L (25)}$$
$$0,47x_6-0,96x_9-p_2 \geq 0 \qquad \text{P2 (26)}$$
$$0,46x_7-p_3 \geq 0 \qquad \text{P3G1 (27)}$$
$$500 \leq p_3 \leq 1500 \qquad \text{P3L, P3G2 (28)}$$
$$0,46x_8-0,01x_9-p_{41}-p_{42} \geq 0 \qquad \text{P4G1 (29)}$$
$$p_{41} \geq 300 \qquad \text{P4G2 (30)}$$
$$p_{42} \leq 500 \qquad \text{P4L (31)}$$
$$x_5-p_5 \geq 0 \qquad \text{P5 (32)}$$
$$0,99x_9-p_6 \geq 0 \qquad \text{P6G (33)}$$
$$p_6 \leq 500 \qquad \text{P6L (34)}$$
$$0,53x_6-p_7 \geq 0 \qquad \text{P7 (35)}$$

Inequations (12) - (17) correspond to input elements, inequations (18) - (20) correspond to working means, inequations (21) - (23) correspond to semi-products and inequations (24) - (35) correspond to products. The variables k_1 and k_2 are slakes in (18) and (19).

The optimal solution was obtained by the computer program ALINO performed on the computer ISKRA DATA. The first table of results presents for each variable its optimal value and the interval of the coefficient of the objective function on which the solution is not sensitive for changes. The second table of results presents for every constraint the marginal profit of the constraint and also the interval of the constant term on which this profit is not sensitive for changes of the constant term. The constraints are ordered in the following way: firstly the constraints of the form less or equal, and then those of the form greater or equal.

Z(MAX) 353465

VARIABLE	SOLUTION ACTIVITY	FOR PRICE FROM	TO	REDUCED PROFIT
X1	15407	-2,641	38,284	0
X2	4593	-41,284	0,359	0
X3	15099	-1,675	40,086	0
X4	3106	-426,087	1,851	0
X5	4295	-9,187	17,582	0
X6	0	$-\infty$	-7,477	-12,523
X7	1087	$-\infty$	2,800	0
X8	652	-2029,986	7,813	0
X9	0	-59,108	10,579	0
E1	20000	-26,830	∞	0
E21	1500	-195,898	∞	0
E22	1000	-195,898	∞	0
E3	0	$-\infty$	0	-200
P11	2641	96,200	120	0
P12	500	100,000	∞	0
P2	0	$-\infty$	425,946	-45,946
P3	500	$-\infty$	228,261	0
P41	300	$-\infty$	179,159	0
P42	0	$-\infty$	179,159	-19,159
P5	4295	140,813	167,582	0
P6	0	385,446	455,837	0
P7	0	0,000	29,628	0

From the first table of results we can see with the help of the graph, that we should typify 15 407t of the fresh milk (x_1= 15 407). In this way we produce 15 099t of typified milk which we condense (x_3 = 15 099). 2 641t

of condensed milk we sell at the price of 100 monetary units per one ton (p_{11} = 2 641) and 500t with price of 120 m.u./t (p_{12} = 500).

CONSTRAINT	REDUCED PROFIT	FOR BOUND FROM	TO
E1L	6,830	15000	20964
E2L1	55,898	1000	27099
E2L2	45,898	0	26599
K1	0	13324	∞
K2	0	1739	∞
K3	0	0	∞
P1L	20	0	3141
P3L	0	500	∞
P4L	0	0	∞
P6L	0	0	∞
E1G1	-26,830	-964	9697
E1G2	0	-∞	20000
E2G1	-195,898	-25599	2646
E2G2	0	-∞	1500
E3	0	-∞	0
S1	-24,400	-938	9431
S2	-12,307	-849	8533
S3	-77,413	-652	1792
P1G	-100,000	-∞	2641
P2	-425,946	0	0
P3G1	-228,261	-500	1215
P3G2	-8,261	0	1500
P4G1	-179,159	-300	824
P4G2	-29,159	0	1124
P5	-150	-∞	4295
P6G	-430	-500	0
P7	-6	-∞	0

1 088t of condensed milk is used for drying (x_7 = 1 088) and 500t of dried milk is sold. Similarly the other results from the first table can be explained.

The second table presents the marginal profit of the constraints. If we were to increase the available quantity of fresh milk for 1t, the profit would increase for 6,830 m.u. (marginal profit of the constraint E1L is 6,830). However, this increase holds only till 20 964t. By increasing the capacity of the evaporator the profit cannot be increased (the marginal profit of the constraint K1 is 0). If the capacity were to decrease below 13 324t H_2O, the profit would start decreasing. The marginal price of the first half-product, that is typified milk, is 24,4 m.u./t (marginal profit of the constraint S_1 is -24,4). Similarly other results from the second table can be explained.

Literature

1. Meško I., Optimiranje in obračun večfazne proizvodnje, Naše gospodarstvo, 1979, št. 5-6,
2. Meško I., Optimiranje poslovanja I, VEKŠ Maribor 1982,
3. Meško I., B. Pevec, Optimiranje poslovanja kod višefazne proizvodnje, Proizvodnja, april 1982.

INFORMATICA 1/1983

PRIKAZ SINHRONIZACIJE PARALELNIH PROCESOV NA PROBLEMU PROIZVAJALCEV IN POTROŠNIKOV

RAJKO SABO

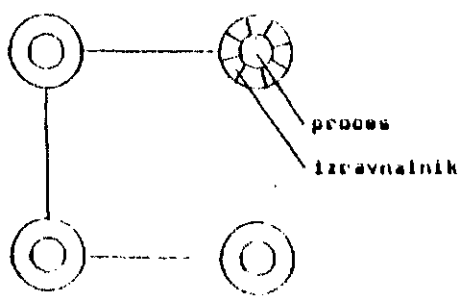
INSTITUT JOŽEF STEFAN, LJUBLJANA

UDK: 681.3.013/014

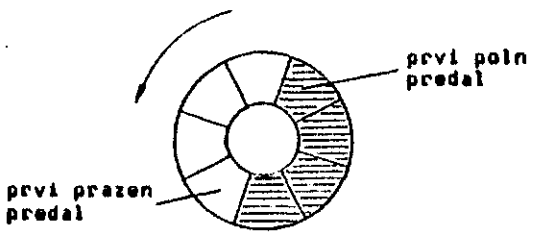
Zanimiv in poučen primer uporabe medsebojne sinhronizacije paralelnih procesov je problem "proizvajalcev in potrošnikov". V pričujočem članku je opisana inačica tega problema. Najprej je predstavljen problem, v naslednjih razdelkih sta opisani dve rešitvi - z monitorjem in s semaforji, nato pa je podana analiza druge rešitve. Dokazano je, da je operiranje z izravnalnikom sinhronizirano, da je onemogočeno neomejeno dolgo čakanje, da je onemogočen nastop mrtve točke na izključitvenem semaforju in slednjič, pod kakšnimi pogoji je onemogočen nastop mrtve točke zaradi krožnega čakanja na vire.

Interesting and educational example on synchronization of parallel processes is the producer - consumer problem. In this paper a variant of this problem is demonstrated. First, the problem is introduced and then two solutions are described : with monitor and with semaphores. For the latter the following properties are proved : the synchronization considering buffer manipulations, the absence of infinite overtaking (starvation to death), the avoidance of deadlock arising from mutual exclusion and, finally, under which conditions the deadly embrace on resources is prevented.

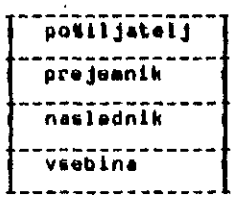
Zamislimo si naslednji model : imamo množico enakih procesov. Vsak ima virtualni lastni krožni izravnalnik (1. slika).Procesi so med seboj povezani, po povezavah si pošiljajo sporočila (2. slika) v izravnalnike (3. slika). Za graf povezav zahtevamo naslednje omejitve : biti mora brez zank (zanka je povezava vozlišča s samim seboj), neusmerjen (vse povezave so neusmerjene) in med dvema vozliščema je lahko največ ena povezava. Proces lahko vpisuje sporočilo le v izravnalnik tistega procesa, do katerega pelje povezava od njega, bere pa lahko le iz lastnega izravnalnika. Za aktiviranje in ustavljanje procesov skrbi sistemski razvrščevalnik (scheduler). Ko se proces aktivira, najprej preveri, ali je njegov izravnalnik prazen. Če ni prazen, potem kot "potrošnik" jemlje sporočila ven. Obstojajo tri možnosti :
a) da je naslovnik sporočila on sam. Tedaj sprožilo "prebere" (sprocesira) in ga nato zavrže;
b) da je naslovnik kdo drug. Proces tedaj poišče pot do njega in če jo najde, pošlje sporočilo prvemu procesu na poti do naslovnika;
c) da do naslovnika ne najde poti. V tem primeru sporočilo zavrže.



1. slika: primer sistema procesov z izravnalniki.



2. slika: zamisel izravnalnika.



3. slika: oris sporočila.

Če je izravnalnik prazen, začne kot "proizvajalec" (naključno) ustvarjati sporočila za druge procese in jim jih poskuša poslati. Če lahko določeno število sporočil odda drugim procesom, se ustavi, vendar ostane v stanju "pripravljen" in ga lahko razvrščevalnik kadarkoli spet aktivira. Če pa se zgodi, da želi proces poslati sporočilo v nek izravnalnik, ki je že poln, ga sinhronizacijske funkcije sistema blokirajo. Tak proces se neha izvajati. Sistem bo spremenil stanje tega procesa iz "blokiran" v "pripravljen", ko bo odpravljen vzrok, ki je povzročil njegovo blokado (torej, ko tisti izravnalnik ne bo več poln).

Problem, ki ga je potrebno tu rešiti, je - sinhronizacija procesov v primerih :
- poskusa vlaganja v poln izravnalnik in
- poskusa jemanja iz praznega.
V literaturi ([1], [2], [4], [5], [6], [7]) sem zasledil rešitve, pri katerih se v obeh primerih procesi zaustavijo. V vrsti problemov "proizvajalec - potrošnik", kjer so procesi različni : eni so proizvajalci, drugi potrošniki - je ta rešitev izvedljiva. Če se namreč zaustavijo vsi proizvajalci - zaradi polnih izravnalnikov, se aktivirajo potrošniki in jih izpraznijo in obratno - če so izravnalniki prazni, se zaustavijo potrošniki in čakajo, da se spet napolnijo.

V naši inačici, ko nimamo take dvopolne delitve procesov, temveč eno samo univerzalno obliko, tako da lahko vsak proces deluje enkrat kot proizvajalec, drugič kot potrošnik in tretjič kot posrednik,ta rešitev ni bila sprejemljiva, zato sem se odločil za drugačno rešitev. Zaustavitev (blokiranje) sem dovolil le, če je proces hotel vložiti sporočilo v poln izravnalnik. Če pa je hotel jemati iz praznega, mu nisem dovolil, da bi se zaustavil in čakal na sporočila, temveč sem zahteval, da nadaljuje svojo dejavnost (to je - ustvarjanje novih sporočil).

ZAPIS PROCESA "PROIZVAJALCA IN POTROŠNIKA"

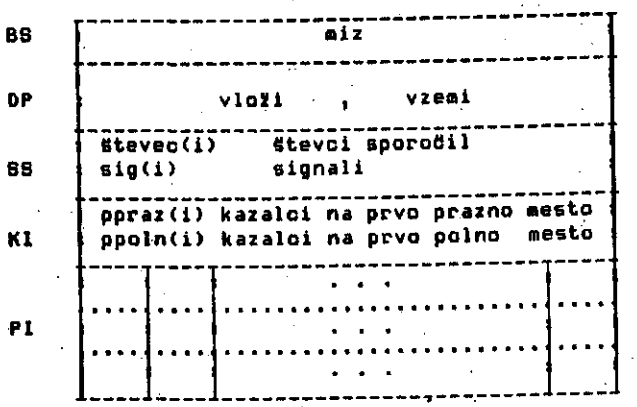
```
const nspormax ( število sporočil , ki jih naj
                 ustvari v enem krogu );
      n      ( število procesov );

var  prazen : Boolean ( indikator praznosti );
     nspor  : integer ( število      ustvarjenih
                        sporočil );
     sporočilo : record;


proces i :
begin
  nspor := 0;
  while nspor < nspormax do
    begin
      vzemi( i, sporočilo, prazen );
      while prazen = false do
        ( prazni svoj izravnalnik )
        begin
          if i = naslovnik then
            "izpiši-sporočilo";
          else
            begin
              "poišči-naslednika";
              if "naslednik je najden" then
                vloži( nasledniku, sporočilo )
              else
                "izpiši-sporočilo"
            end;
          vzemi( i, sporočilo, prazen )
        end;
      ( ustvari novo sporočilo )
      tvori( i, sporočilo );
      nspor := nspor + 1;
      "poišči-naslednika";
      if "naslednik je najden" then
        vloži( nasledniku, sporočilo )
      else
        "izpiši-sporočilo"
    end
  end;
  sleep; ( prostovoljni odstop procesorja )
  go to proces i
end;
```

4. slika: program procesa "proizvajalca in potrošnika"

Pri procedurah, ki niso pomembne za to temo, se ne bomo ustavljali ("tvori-sporočilo", "poišči-naslednika", "izpiši- sporočilo"), ogledali pa si bomo proceduri "vloži" in "vzemi" ter sinhronizacijske operacije.

REŠITEV Z MONITORJEM

Vsi izravnalniki so združeni v posebni strukturi - monitorju ([8], [11], [14], [15], 5. slika).



BS ekskluzivni binarni semafor za zaščito monitorja,
DP dosežni proceduri,
SS sinhronizacijske spremenljivke,
KI kazalci v posameznih izravnalnikih,
PI polje izravnalnikov (št. procesov x dolžina izravnalnika)

5. slika: oris monitorja.

Izključno pravico dostopa do izravnalnikov imata monitorski proceduri "vloži" in "vzemi" (6. slika).

```
const  N    ( velikost izravnalnikov );

var  miz    init( 1 )          ( binarni semafor
                 za ščitenje kritičnih delov );

     števec  array[1..n] of integer
             init( N, N,...,N )       ( števec
                   sporočil v izravnalnikih );
     sig     array[1..n] of integer
             init( 0, 0,...,0 )    ( signali );



procedure vloži(i, sporočilo);

var i    ( identiteta ),
    sporočilo;

begin
   P(miz)   ( ščitenje kritičnega dela );
   do while števec(i) >= N
      begin
         onemogoči prekinitve;
         V(miz);
         wait(sig(i));
         P(miz);
         omogoči prekinitve;
      end;
   "vloži sporočilo v izravnalnik";
   števec(i) := števec(i) + 1;
   V(miz)
end;
```

```
var prazen : Boolean      { indikator praznosti
                            izravnalnika }

procedure vzemi(i, sporočilo, prazen);

var i   { identiteta };
    sporočilo;

begin
   P(miz)   { ščitenje kritičnega dela };
   if števec(i) = 0 then
      begin
         prazen := true;
         V(miz);
         return;
      end;
   else
      begin
         prazen := false;
         "vzemi sporočilo iz izravnalnika";
         števec(i) := števec(i) - 1;
         send(sig(i));
         V(miz)
      end
end;
```

6. slika: monitorski dosežni proceduri "vloži" in "vzemi".

Slaba stran te rešitve je, da ni varna v tem smislu, da dopušča neomejeno prehitevanje in s tem neomejeno dolgo čakanje ("starvation to death"). Proces, ki želi vložiti sporočilo, najprej zahteva izključni dostop do izravnalnika. Ko ga dobi, testira, ali je izravnalnik poln. Če je tako, zapusti kritični del ter se zaustavi. S tem se spet znajde na začetku, saj mora ob ponovni aktivaciji postopek testiranja ponoviti. To lahko ponavlja neomejeno mnogokrat, medtem pa ga ostali procesi lahko prehitevajo.

REŠITEV S SEMAFORJI

V jedru operacijskega sistema [9] imamo implementiran binarni semafor z Dijkstrovima operacijama P_b in S_b [6], zato ga razširimo do splošnega na običajni način [13].

```
var pr : array[1..n] of integer
       init( N, N,...,N )  { splošni semaforji,
    ki štejejo prazna mesta v izravnalnikih };

    nepo : array[1..n] of integer
        init( 0, 0,...,0 ) { binarni semaforji
          za sinhronizacijo  poln - nepoln } ;

    miz    init( 1 )       { binarni semafor za
                             ščitenje kritičnih delov };

P(pr(i))      pr(i) := pr(i) - 1;
              if pr(i) < 0 then
                 begin
                    V_b(miz);
                    P_b(nepo(i));
                    P_b(miz);
                 end;

V(pr(i))      pr(i) := pr(i) + 1;
              if pr(i) =< 0 then
                 V_b(nepo(i));
```

7. slika: operaciji P in V nad splošnim semaforjem.

Za sinhronizacijo prazen - neprazen pa sem moral operaciji P in V spremeniti, tako da ne bo prišlo do zaustavitve ob praznem izravnalniku. V operaciji P sem zatorej opustil čakanje in zato je v operaciji V poslalo pošiljanje signalov za konec čakanja nepotrebno. Uvedel pa sem logično spremenljivko (prazen), ki ob uresničitvi spremeni dejavnost procesa iz "potrošnika" v "proizvajalca". Operaciji P' in V' sta prikazani na 8. sliki.

```
var  sp      : array [1..n] of integer
                 init( 0, 0,...,0 )  ; { splošni
                 semaforji, ki štejejo sporočila
                 v izravnalnikih }
     prazen : Boolean ;

P'(sp(i))    : prazen := false;
               sp(i)  := sp(i) - 1;
               if sp(i) < 0 then
                  begin
                     prazen := true;
                     sp(i) := sp(i) + 1;
                     V_b(miz);
                     return { iz procedure };
                  end;

V'(sp(i))    : sp(i) := sp(i) + 1;
```

8. slika: Operaciji P' in V' nad splošnim semaforjem.

Z navedenimi sinhronizacijskimi operacijami imata monitorski proceduri vloži in vzemi naslednjo obliko :

```
procedure vloži( i, sporočilo):
   var   i : integer  { identiteta };
         sporočilo : record ;
   begin
      P_b(miz);
      P(pr(i));
      "vloži sporočilo v
      izravnalnik(i)";
      V'(sp(i));
      V_b(miz)
   end;


procedure vzemi( i, sporočilo, prazen):
   var   i : integer  { identiteta };
         sporočilo : record;
   begin
      P_b(miz);
      P'(sp(i));
      "vzemi sporočilo iz
      izravnalnika(i)"
      V(pr(i));
      V_b(miz)
   end;
```

9. slika: proceduri "vloži" in "vzemi".

SINHRONIZACIJA

Habermann je v [10] dokazal naslednje:
ob predpostavki, da v kritični del vstopamo in izstopamo samo skozi operaciji P in V (10. slika), velja :
a) izvaja se lahko samo en kritični del hkrati in
b) če ni noben od procesov v svojem kritičnem delu, ne more biti zaustavljen (blokiran) noben proces. (1)

```
P(s);
kritični del(s);
V(s);
```

10. slika: izvajanje kritičnih delov.

Ali sme proces, ki izvrši P, nadaljevati ali ne, je odvisno od tega, kolikokrat sta bili P in V izvršeni prej. Stanje usklajevanja med procesi lahko opišemo z naslednjimi količinami:

- C(s) - nenegativna celoštevilska konstanta, podana ob inicializaciji;
- NW(s) - število izvršenih (klicanih) P(s);
- NS(s) - število izvršenih V(s) in
- NP(s) - število v celoti izvršenih P(s) z nadaljevanjem.

Pomen konstante C(s) je, za koliko sme število klicanih operacij P presegati število operacij V .Priščitenju kritičnih delov (10. slika) uporabimo vrednost C(s) = 1 , pri sinhronizaciji nad izravnalnikom pa C(s) = N , dolžina izravnalnika .

Izvršitev operacije P(s) ima naslednji učinek na NW(s) in NP(s) :

```
NW(s) := NW(s) + 1;                          (1.1)
if  NW(s) =< C(s) + NS(s)  then
    NP(s) := NP(s) + 1 ,
```

torej izvršitev operacije P(s) ne povzroči čakanja, dokler je število izvršitev P(s) kvečjemu za C(s) večje kot število izvršitev V(s).

Učinek operacije V(s) pa je :

```
if  NW(s) > C(s) + NS(s)  then
    NP(s) := NP(s) + 1 ;                     (1.2)
NS(s) := NS(s) + 1;
```

izvršitev V(s) odpravi morebitno čakanje, povzročeno s P(s).

Habermann trdi :

```
NP(s) = min( NW(s), C(s) + NS(s) )           (1.3)
```

je invarianta za operaciji P(s) in V(s).

Dokazal bom, da ta trditev (1.3') in trditvi (1a) in (1b) veljajo tudi za P' in V'.

Začetne vrednosti naj bodo :

```
C(s) >= 0 , NW(s) = NP(s) = NS(s) = 0 !
```

Najprej zapišimo učinka operacij P' in V' s količinami C , NW , NS in NP ! Dodatno vpeljemo še logično spremenljivko prazen, ki ima vrednost "prav", če je izravnalnik prazen, in vrednost "napak" sicer.
Učinek izvršitve operacije P'(s) je :

```
if NP(s) >= NS(s) then prazen := true
else
   begin
      prazen := false;
      NW(s)  := NW(s) + 1;                   (1.4)
      NP(s)  := NP(s) + 1
   end;
```

Učinek izvršitve operacije V'(s) pa je :

```
NS(s) := NS(s) + 1.                          (1.5)
```

(i) Pokažimo, da ima spremenljivka prazen vrednost "prav" natanko tedaj, ko je izravnalnik prazen ! Če je izravnalnik prazen, pomeni to, da je število izvršenih jemanj enako številu izvršenih vlaganj. Iz določitev operacij vloži in vzemi (9. slika) ter učinkov operacij P in V ((1.1) in (1.2)) sledi, da velja tedaj

$$NP(s) = NS(s) .$$

Če pa izravnalnik ni prazen, pa pomeni, da je v njem nekaj sporočil, torej da je število vlaganj večje od števila jemanj,

$$NP(s) < NS(s) .$$

Ker druge možnosti ni, pomeni hkrati, da NP(s) > NS(s) ne more nastopiti. Če sedaj primerjamo dobljene relacije z učinkom operacije P'(s) (1.4), vidimo, da je vrednost spremenljivke prazen pravilno določena.

(ii) Z indukcijo bomo dokazali, da je izraz (1.3) invarianten za operaciji P' in V'.
Relacija (1.3) na začetku velja, ker je NP(s) = NW(s) = 0 in C(S) + NS(s) >= 0.

(iii) Recimo, da relacija (1.3) velja in da se bo izvršila operacija P'(s) ! Če je NP(s) >= NS(s), izvršitev operacije P'(s) samo vrne indikator prazen := "prav" . Ker ostanejo vse ostale količine nespremenjene, izraz (1.3) obdrži svojo veljavnost.

Če pa je NP(s) < NS(s) , imamo

```
NW(s) := NW(s) + 1 ,
NP(s) := NP(s) + 1 ,
```

(vrednosti spremenljivke prazen smo obravnavali posebej (i)).
Iz NP(s) < NS(s) sledi tudi

$$NP(s) < C(s) + NS(s) ,$$

torej v tem primeru pred izvršitvijo operacije P' velja

$$NP(s) = NW(s) < C(s) + NS(s) ,$$

po izvršitvi pa

$$NP(s) = NW(s) =< C(s) + NS(s)$$

in je potemtakem izraz za operacijo P' res invarianten.

(iv) Predpostavimo, da je izraz (1.3) veljaven in da se bo izvršila operacija V'(s) ! Naj velja

$$NP(s) = NW(s) =< C(s) + NS(s) !$$

Po izvršitvi operacije V'(s) se poveča le :

```
NS(s) := NS(s) + 1
```

in velja sedaj

$$NP(s) =NW(s) < C(s) + NS(s) ,$$

torej izraz (1.3) še vedno velja.

Kaj pa, če je NP(s) = C(s) + NS(s) > NW(s) ? Iz učinka operacije P'(s) ((1.4)) vidimo, da P'(s) spreminja količini NP(s) in NW(s) vedno obe hkrati, in ker sta na začetku enaki, sta vedno enaki. Torej zadnji primer, ko je NP(s) > NW(s) , ne more nastopiti. Zato je izraz (1.3) invarianten tudi za izvršitev operacije V'(s) in po indukcijskem sklepu velja vedno.

(v) Predpostavimo, da je izraz (1.3) invarianten . Kakšen je potem učinek operacij P' in V' ?
Najprej si oglejmo P'(s) ! V (i) smo ugotovili, da izravnalnik ni prazen natanko tedaj, ko je NP(s) < NS(s) . Tedaj se vzetje lahko začne

$$NW(s) := NW(s) + 1 ,$$

pa tudi konča

$$NP(s) := NP(s) + 1 ,$$
$$prazen := false .$$

Če pa je izravnalnik prazen,

$$NP(s) = NS(s) ,$$

ni ne vzetja ne čakanja, torej nobene spremembe razen prazen := true. Vidimo, da je učinek tak, kot je zapisan v (1.4).

(vi) Predpostavimo, da je izraz (1.3) invarianten, kakšen mora biti potem učinek V' ? Iz učinka P' smo ugotovili, da vedno velja

$$NP(s) = NW(s) ,$$

zato je

$$C(s) + NS(s) >= NP(s) ,$$

vidimo, da se NS(s) lahko vedno poveča, pa bo (1.3) ostal invarianten. Torej je učinek V'(s) res

$$NS(s) := NS(s) + 1$$

brezpogojno, tako kot ga opisuje (1.5).

Q.E.D.

Pravkar dokazano trditev bomo uporabili za dokaz pravilnosti sinhronizacije med procesi glede na prazen oziroma poln izravnalnik. Dokazati želimo, da ne pride do vlaganja v poln izravnalnik ali do jemanja iz praznega, torej da je vedno res :

$$0 =< NP(pr(i)) - NP(sp(i)) =< N , 1 =< i =< n. \quad (1.6)$$

Iz procedure vloži (9. slika), trditve (1.3) in trditve (1.3') sledi :

$$NS(sp(i)) =< NP(pr(i)) =< C(pr(i)) + NS(pr(i))$$

oziroma

$$NS(sp(i)) =< NP(pr(i)) =< N + NS(pr(i)) . \quad (1.7)$$

Iz procedure vzemi (9. slika), trditve (1.3) in trditve (1.3') pa sledi :

$$NS(pr(i)) =< NP(sp(i)) =< C(sp(i)) + NS(sp(i))$$

oziroma

$$NS(pr(i)) =< NP(sp(i)) =< NS(sp(i)) \quad (1.8)$$

Iz relacij (1.7) in (1.8) sledi po eni strani:

$$NP(pr(i)) =< N + NS(pr(i)) =< N + NP(sp(i)) ,$$

torej

$$NP(pr(i)) - NP(sp(i)) =< N$$
(drugi del neenačbe (1.6)).

Po drugi strani pa iz (1.7) in (1.8) sledi tudi:

$$NP(sp(i)) =< NS(sp(i)) =< NP(pr(i)),$$

torej

$$0 =< NP(pr(i)) - NP(sp(i))$$
(prvi del neenačbe (1.6)).

Ker je i poljuben, velja (1.6) za vse izravnalnike. Q.E.D.

Trditvi (1a) in (1b) sta trivialni posledici invariantnosti (1.3).

a) Iz (1.3) sledi

$$NP(s) =< 1 + NS(s) ,$$

to pomeni, da je število vstopov v kritični del(s) lahko kvečjemu za 1 večje od števila izstopov, kar že pomeni, da v kritičnem delu(s) ni več kot en proces.

b) Iz (1.3) sledi v primeru, ko imamo čakajoče procese (ko je NP(s) < NW(s)) :

$$NP(s) = 1 + NS(s) .$$

To pa ne more biti res, ko je NP(s) = NS(s), kar opisuje položaj, ko ni noben proces v kritičnem delu(s).

Vidimo, da se lahko izvaja samo en kritični del hkrati in da ne more priti do zaustavitve procesa, če so vsi procesi zunaj svojih kritičnih delov.
Ker je naša rešitev izvedena s semaforji, ti pa so implementirani tako, da uvrščajo čakajoče procese po načelu - prvi noter, prvi ven -, ni mogoče, da bi kateri proces prehitel drugega, ki je v isti vrsti pred njim. S tem je onemogočeno, da bi nek proces čakal neomejeno dolgo .

MRTVA TOČKA

Za nastop mrtve točke so potrebni trije pogoji :
- medsebojno izključevanje,
- nespreščanje ("nonpreemption") in
- krožno čakanje na vire [3].

Če hočemo doseči, da do mrtve točke ne bo prišlo, moramo onemogočiti enega teh pogojev. Medsebojnega izključevanja ne moremo opustiti, saj ga uporabljamo za ščitenje kritičnih delov (operiranje nad izravnalnikom) in je nujno za pravilen potek izvajanja procesov.

Ravno tako ni ustrezno popolno spreščanje virov, kajti če npr. proces, ki ima neko sporočilo za drugega, ne bo čakal, dokler ga ne bo vložil, bo to sporočilo izgubil ("povozil"), ko bo vzel naslednje sporočilo. Vendar pa to vseeno izkoristimo, kot bomo pokazali pozneje.

Na voljo nam ostane še tretji pogoj, pogoj krožnega čakanja. V krožno čakanje sta v primeru, ko procesi ne morejo čakati na lastne vire, vedno vključena vsaj dva procesa. (Lahko jih je več, vendar število ni bistveno za sam pojav.) Krožno čakanje pomeni v tem primeru, da nek proces zahteva nesproščen vir nekega drugega procesa pri nesproščenem svojem viru, drugi proces zahteva nesproščen vir tretjega procesa itd., zadnji v tej zanki pa zahteva pri nesproščenem svojem viru vir prvega procesa. Privzeli smo še, da ima vsak proces samo en vir, ki je izključni in ga bere samo on sam.

1. trditev.

Pri predpostavkah, da:
1. ima vsak proces samo en vir in
2. proces ne more čakati na svoj vir,
je potreben pogoj za nastop krožnega čakanja na vire ta, da nek proces zahteva nesproščen vir drugega procesa pri nesproščenem svojem viru.

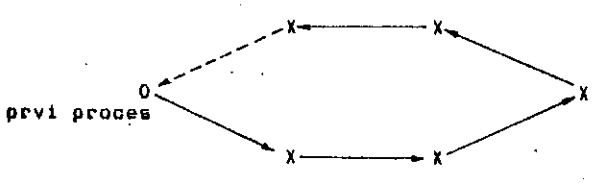
Dokaz.

Oglejmo si vse možne situacije, v katerih je lahko proces (s svojim virom) glede na drug proces (oz. vir)! Za boljšo preglednost jih zapišimo v obliki tabele (11. slika)

položaj	lastni vir	odnos procesa	tuj vir
(1)	sproščen	ne zahteva	sproščen
(2)	sproščen	ne zahteva	nesproščen
(3)	sproščen	zahteva	sproščen
(4)	sproščen	zahteva	nesproščen
(5)	nesproščen	ne zahteva	sproščen
(6)	nesproščen	ne zahteva	nesproščen
(7)	nesproščen	zahteva	sproščen
(8)	nesproščen	zahteva	nesproščen

11. slika: tabela možnih položajev procesa glede na vire.

Proces je v odnosu do nekega tujega vira gotovo v enem od položajev (1) - (8) . Privzemimo, da je v stanju krožnega čakanja na tuj vir, in poglejmo, v katerem položaju se tedaj lahko nahaja! V položajih (1), (2), (5) ali (6) se ne more nahajati, saj tu vira sploh ne zahteva. V položajih (3) in (7) bi bila njegova zahteva za tuji vir odobrena, ker je ta sproščen, in proces se ne bi zaustavil in čakal. Kot edina položaja, v katerih lahko pride v poštev krožno čakanje, ostaneta položaja (4) in (8). Pokažimo, da tudi v položaju (4) krožnega čakanja ne more biti! V položaju (4) je prvi proces svoj vir sprostil in zahteva tuj nesproščen vir. Zaradi svoje zahteve se sicer zaustavi, toda nobena čakalna zanka, kot je opisana na prejšnji strani, ni mogoča oziroma zaključena, kajti zaustavitev in čakanje na vir prvega procesa ni mogoče, ker je ta vir sproščen (12. slika).



12. slika: krožna čakalna zanka, ki ni zaključena.

Tako smo pokazali, da v položajih (1) - (7) krožno čakanje na vire ni mogoče, torej je mogoče le v položaju (8), ta pa opisuje ravno pogoj 1. trditve.

Q.E.D.

Pojdimo sedaj korak dalje in poiščimo še kakšen zadosten pogoj za krožno čakanje!

2. trditev.

Če velja pogoj 1. trditve vsaj za dva procesa v sistemu, ki sta povezana z dvosmerno povezavo, je to zadosten pogoj za nastop krožnega čakanja na vire.

Dokaz. Označimo procesa, za katera velja pogoj iz 2. trditve, z a in c, njuna vira pa z A in C! Torej lahko proces a pri nesproščenem viru A zahteva nek vir B nekega procesa b, proces c lahko pri nesproščenem viru C zahteva vir D poljubnega procesa d. Toda nobene ovire ni, da ne bi smeli za c vzeti kar b, za d pa a in imamo naslednji položaj :
- proces a zahteva vir procesa b,
- proces b pa zahteva vir procesa a
in ker sta oba vira nesproščena, se oba procesa ustavita, čakajoč drug drugega, da bo sprostil svoj vir.

Q.E.D.

Toda pogoj 2. trditve, za katerega smo pravkar pokazali, da je zadosten, je zgrajen s potrebnim pogojem (1. trditev). To nas navede na misel, da je mogoče tudi sam potreben. Recimo, da ni izpolnjen! To pomeni
a) ali, da za noben proces ne velja pogoj 1.trditve - to pa je že tudi zanikanje 2. trditve, ki je potreben pogoj, torej v tem primeru krožnega čakanja ne more biti;
b) ali, da velja 1. trditev za natanko en proces. Ker pa smo že ugotovili, da sta v krožno čakanje vključena vedno vsaj dva procesa, to pomeni, da tudi v tem primeru ne more biti krožnega čakanja.

Tako smo dokazali

3. trditev.

2. trditev predstavlja potrebni in zadostni pogoj za nastop mrtve točke v obliki krožnega čakanja na vire.

Naša prva ugotovitev v boju proti mrtvi točki je ta: sistem procesov je varen (pred pojavom krožnega čakanja), če 1. trditev ne velja za noben par procesov, ki je povezan z dvosmerno povezavo. Posledica je, da sme biti v sistemu, v katerem so vse povezave dvosmerne, pogoj 1. trditve izpolnjen samo za en proces.

Vir je sproščen, kadar je na razpolago, torej kadar ga nihče ne zahteva. Če proces zahteva vir (izravnalnik), pomeni, da bo ali iz njega jemal ali vanj vlagal. Za samo jemanje ni težav, saj ima to pravico za vsak izravnalnik samo njegov "lastnik". Jemanje je organizirano kot kritični del, ščitenje kritičnih delov pa je urejeno z binarnim semaforjem (miz) za medsebojno izključevanje. Varnost pri ščitenju kritičnih delov, da ne pride do mrtve točke, ker bi vsi procesi čakali na izključitveni semafor, smo dosegli z izključitvijo drugega potrebnega pogoja za nastop mrtve točke - to je nesproščanja, saj proces vedno, ko je vstopil v kritični del (z operacijo P(miz)) in zahteval dostop do izravnalnika, pa ga ni dobil, izstopi iz kritičnega dela (z operacijo V(miz)) ter se šele nato ustavi. S podobnim postopkom je

zagotovljena varnost s sproščanjem izključitvenega semaforja pri vlaganju. V tem primeru proces, ki je želel vložiti sporočilo v izravnalnik, ki pa je bil poln, najprej izvrši operacijo V(miz) ter se šele nato postavi v čakalno vrsto za tisti izravnalnik.

Problem je torej : zahtevati nek vir (izravnalnik) za vlaganje. Ne sprostiti svojega vira in zahtevati tujega pomeni za nek proces v naši inačici, da na vlaganje v njegov izravnalnik še čakajo procesi, ta proces pa zahteva vlaganje v nek drug izravnalnik. To pa je mogoče v dveh primerih.

1) Proces je svoj izravnalnik popolnoma izpraznil in nato začel pošiljati svoja sporočila ; pri tem pa na njegov izravnalnik še čakajo procesi. Kdaj pa je to mogoče? Ker so na njegov izravnalnik čakali procesi, pomeni, da je ta bil poln. Recimo, da je velikost izravnalnika N mest! Proces je izravnalnik izpraznil in pri vsakem vzetju je lahko z operacijo V poslal signal enemu čakajočemu procesu, torej največ N signalov. Po izpraznitvi začne ustvarjati in pošiljati svoja sporočila in ker na njegov izravnalnik po privzetku še čakajo procesi, pomeni, da jih je pred začetkom praznenja čakalo več kot N. Označimo z n(i) število povezav, ki vodijo do i-tega procesa! Ker proces ne more čakati na svoj izravnalnik, lahko na i-ti izravnalnik čaka kvečjemu n(i) procesov. Če opredelimo premajhen izravnalnik kot izravnalnik, ki ima manj mest, kot je povezav do prirejenega procesa, N < n(i) , lahko v tem prvem primeru zaključimo, da

obstoja proces s premajhnim izravnalnikom. (2)

2) Predpostavimo, da izravnalnik nekega procesa še ni prazen (in ni sproščen). Proces ga hoče sprostiti, zato ga začne prazniti. Vzame sporočilo in če je zanj, ga "prebere" in odvrže ter vzame naslednje sporočilo. To lahko ponavlja, dokler so sporočila zanj. Če tako izravnalnik popolnoma izprazni in čakajočih procesov ni več, je izravnalnik sproščen, če pa je še nekaj čakajočih procesov, pa imamo prvi primer, zato privzemimo, da še pred izpraznitvijo vzame sporočilo, ki je prehodno (od nekoga drugega in za nekoga drugega)! Tega mora nato vložiti nasledniku in pri tem se že lahko zaustavi (če je zahtevani izravnalnik nesproščen ali poln). V tem primeru se lahko zgodi, da je poslal manj signalov, kot pa je čakajočih procesov, v najslabšem primeru samo enega (če je že prvo sporočilo prehodno), ne glede na velikost izravnalnika niti na število čakajočih procesov. Skratka, v drugem primeru lahko zaključimo, da

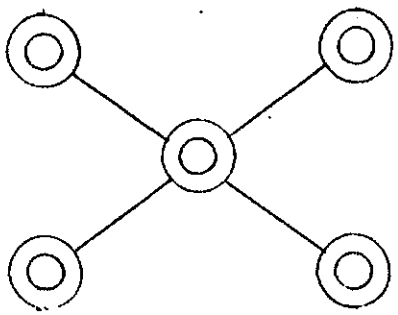
obstoja proces,ki sprejema prehodna sporočila. (3)

Potrebni in zadostni pogoj za varnost opisane inačice problema "proizvajalcev in potrošnikov" je posledica 1. in 2. trditve ter (2) in (3) in se glasi :

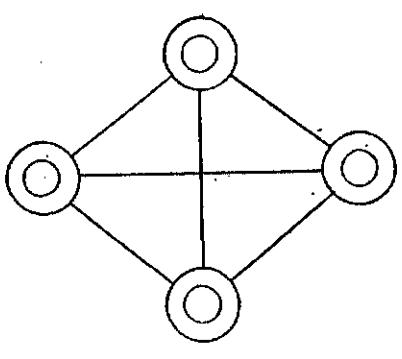
4. trditev.

Pri inačici problema "proizvajalcev in potrošnikov", kot je opisana v tem članku, so varni natanko vsi tisti sistemi procesov, pri katerih ne obstajata dva taka procesa, ki sta povezana z dvosmerno potjo in ki imata premajhen izravnalnik ali prehodna sporočila drug za drugega.

Sistem procesov, pri katerem obstoja <u>en</u> tak "kritični" proces (kot je omenjen v 4. trditvi), je zvezdast (13. slika). Sistem procesov <u>brez</u> takega procesa je tak, da je povezan vsak proces z vsakim (14. slika).



13. slika: zvezdasti model - vsebuje eno "kritično" vozlišče.



14. slika: model "vsak z vsakim" - je brez "kritičnih" vozlišč.

ZAKLJUČEK

1. V članku sta obravnavana dva načina sinhronizacije sočasnih procesov: z monitorjem in s semaforji. Na opisanem konkretnem modelu se je pokazal drug način boljši, s čimer pa ne trdim, da je to na splošno res. Na uporabnost monitorjev kaže dejstvo, da jih uporablja precej novejših operacijskih sistemov. Monitor je več kot samo konstrukt - lahko je tudi način programiranja. Programer ima večjo izbiro in večji vpliv (lahko tudi negativen) na sinhronizacijo, na primer z izbiro sinhronizacijskih spremenljivk, logičnih testov ob vstopanju v monitor. Za sinhronizacijo lahko upošteva tudi izvenmonitorske pogoje, vpliva lahko na razvrščanje procesov v čakalne vrste. Zato pa je dokazljivost pravilnosti otežena ali celo nemogoča, ali pa se je treba nasloniti na druge dodatne lastnosti, na primer hitrost - "počasnost" perifernih enot glede na osrednjo procesno enoto ipd.

2. Dobra stran semaforjev je v tem, da so preprosti in zato pregledni. Točno določajo način sinhronizacije, ki procesom samim ni dostopen. To ima za posledico določeno togost, ki v nekaterih primerih programerja ovira. Operaciji P in V nad semaforji sta komutativni, zato vrstni red njune izvršitve ni važen. To pa za operaciji send(s) in wait(s) ne velja.

3. Pri sočasnem programiranju ne drži več, da se bodo procesi sočasno izvajali pravilno, če se izvajajo pravilno vsak sam zase. Na varnost sistema morejo vplivati dejavniki, ki jih zlahka prezremo. Na primer v sistemu, ki teče pravilno in varno, dodamo enak proces, kot so ostali, pa se bo sistem zablokiral. Vložiti je treba več časa v analizo programov in njihov medsebojni vpliv pa tudi v njihovo testiranje. Vloženi trud pa se izplača z večjo učinkovitostjo.

SLOVSTVO

[1] S. Andler : Synchronization Primitives and the Verification of Concurrent Programs, Carnegie-Mellow University;

[2] E.G. Coffman, P.J. Denning : Operating Systems Theory, Prentice Hall, Englewood Cliffs, N.J.;

[3] E.G. Coffman, M.J. Elphick, A. Shosani : System Deadlocks, Computing Surveys, 3 (1971), št. 3;

[4] R. Devillers, G. Louchard : Improvements of parallelism in a finite buffer sharing policy, The Computer Journal, 19 (1976), št. 3;

[5] E.W. Dijkstra : Co-operating Sequential Processes, Programming Languages, Ed. F. Genuys, Academic Press, 1968;

[6] E. W. Dijkstra : Hierarchical ordering of sequential Processes, Operating Systems Techniques, Acad. press, 1972;

[7] M. Exel : Komunikacija med sekvenčnimi procesi - pregled, 1. in 2. del, Informatica 1 (1977), št. 1 in 2;

[8] M. Exel : Monitorji, IJS Ljubljana, november 1980;

[9] D. Novak, M. Kovačevič, B. Kastelic : AMM operacijski sistem, IJS Ljubljana, junij 1980;

[10] A. N. Habermann : Synchronization of Communicating Processes, CACM, 1.15 (1972), št. 3;

[11] C.A. Hoare : Monitors: An operating System Structuring Concept, CACM, 17 (1974), št. 10;

[12] R. Sabo : Modeliranje namišljene telefonske centrale s sočasnimi procesi, dipl. delo, FNT, Ljubljana, 1980;

[13] A. C. Shaw : The Logical Design of Operating Systems,Prentice-Hall, Englewood Cliffs, N.J. 1974;

[14] N. Wirth : The Use of Modula, Software-Practice and Experience, 7 (1977);

[15] N. Wirth : Toward a Discipline of Real-time Programming, CACM 1.20 (1977), št. 8.

PROGRAMIRANJE V ADI III

ANTON P. ŽELEZNIKAR
ISKRA DELTA, LJUBLJANA

UDK: 681.3.OGADA:519.682

Tretji del članka opisuje najprej ločeno prevajanje v jeziku Ada, in sicer prevajalne enote (paketna specifikacija in paketno telo), WITH člen, vrstni red prevajanja in delovanje povezovalnika. V Adi je V/I zgrajen na konceptu zbirke. Obstaja vrsta zbirčnih procedur, kot so create, open, close, delete, read, write, end_of_file, is_open in name. Zbirke so binarne in tekstovne. Obstaja še vrsta drugih V/I funkcij, kot so get, put, new_line, skip_line itd. Ada omogoča uporabniško definiranje knjižničnih procedur. Obstajajo procedure dispose (sproščanje pomnilnika), memavail (vrnitev obsegov), maxavail (proste lokacije) in procedure za manipulacijo bitov in zlogov, kot so hi, lo, setbit, clrbit, tstbit, nadalje so tu logične funkcije lnot, land, lor, lxor in funkcije za programsko krmiljenje halt, err_exit, command_line, chain, prog_call, prog_return. Nestandardni podprogrami pa so disk_full, eof, put_hex, read_blk, write_blk, rclose. Rutine za manipulacijo nizov pa so tele: length, remove, insert, extract, position, char_to_str, str_to_int in int_to_str. Tekst spremljajo značilni (formalni) primeri. Na koncu je opisana še uporaba Janus prevajalnika.

P r o g r a m m i n g i n A d a III . The third part of the article describes separate compilation in Ada: compilation units (package specification and package body), WITH clause, order of compilation and work od linker. In Ada, I/O is built around the concept of a file. Several file procedures exist like create, open, close, delete, read, write, end_of_file, is_open, and name. Files are of binary and text type. There are several other I/O functions like get, put, new_line, skip_line, etc. Ada enables user defined library procedures. There are routines like dispose (releases the storage), memavail (returns the size), maxavail (the size in bytes) and procedures for manipulation of bits and bytes like hi, lo, setbit, clrbit, tstbit; further, there are logical functions like lnot, land, lor, lxor and functions for program control like halt, err_exit, command_line, chain, prog_call, prog_return. Non-standard I/O subprograms are disk_full, eof, put_hex, read_blk, write_blk, rclose. Routines for string manipulation are the following: length, remove, insert, extract, position, char_to_str, str_to_int, and int_to_str. In the article several typical (formal) examples are presented. At the end the usage of Janus compiler is described.

4.8. Zgradba programa in ločeno prevajanje

L o č e n o p r e v a j a n j e v Adi naj bi imelo enak učinek kot prevod programa v enem samem kosu. Prevajalnik Janus ne porabi za ločeno prevajanje več časa, kot bi ga porabil za prevod programa v enem kosu.

Prevajalne enote v Janus Adi so paketne specifikacije in paketna telesa. Paketi, ki se uporabljajo skupaj, morajo imeti različna imena. S paketnimi določili se določajo podatki, tipi in podprogrami iz paketov, ki jih (podprograme) uporabljajo zunanji paketi. Specifikacija je lahko sestavljena samo iz podatkovnih tipov in paketno telo pri tem ni potrebno. Vsaka specifikacija, ki deklarira podprograme z uporabo v zunanjih paketih, mora imeti paketno telo z definicijo teh podprogramov.

Paketno telo in njegova specifikacija (če ta obstaja) morata imeti enako ime kot p r i p a d a j o č a z b i r k a. Sufiksa tega imena pa sta seveda različna, in sicer 'PKG' za telo in 'LIB' za specifikacijo. V paketno telo lahko združimo podprograme, ki pripadajo določeni aplikaciji (kot je npr. IO knjižnica).

W I T H č l e n se uporablja za določanje zunanjih paketov, ki so potrebni pri kompilacijski enoti. V povezavi z USE členom je moč v Adi doseči pogoje za ločeno prevajanje (kompilacijo).

Z WITH členom le imenujemo imena tistih zunanjih paketov, do katerih mora imeti zadevni paket dostop. Pri tem ni potrebno navesti tistih paketov, ki jih uporabljajo v WITH členu imenovani paketi. WITH členi določajo odvisnosti med

prevajalnimi paketi. Povezovalnik uporabi te odvisnosti za določitev pravilnega povezovalnega zaporedja in zagotovi tako pravilno obdelavo deklaracij. Če sta dva paketa vzajemno povezana z WITH členoma, bi se klicala medsebojno odvisno, kar pa ni dovoljeno (javljanje napake s prevajalnikom ali povezovalnikom). Uporabnik mora to stanje sam popraviti.

Paketna specifikacija in telo lahko imata različna imena v WITH členih. Deklaracije v telesu potrebujejo pakete, ki jih specifikacija ne potrebuje. Paketi, ki so povezani z WITH členi v specifikacijah, so avtomatično povezani tudi v pripadajočih paketnih telesih. Zaradi nazornosti je priporočljivo, da tudi v telesu prikažemo vse WITH povezave.

Podenote kompilacijskih enot v Janus Adi niso implementirane.

Vrstni red kompilacije je odvisen od pravil vidljivosti. Glavno pravilo je tole: če paket potrebuje podatke o drugem paketu, potem mora biti specifikacija paketa, ki podatke ima, kompilirana pred uporabo teh podatkov. WITH člen določa natanko tiste pakete, ki morajo biti kompilirani pred trenutnim paketom. Če paket nima WITH člena, se kompilira kot prvi. To je oblika tkim. knjižničnih paketov. Pri spremembi paketne specifikacije je potrebno ponovno kompilirati vse pakete, ki so od te specifikacije odvisni.

Kadar imamo spremembe v paketnem tekstu, ki ne spremenijo paketne specifikacije, ostalih paketov ni potrebno ponovno kompilirati. To je prijetna lastnost ločenih specifikacij in teles.

Zaradi zmogljivosti sekundarnega pomnilnika mikroračunalnika prevajalnik Janus ne uporablja programske podatkovne baze. Vsak paket ima lastno simbolno zbirko, ki vsebuje vso potrebno informacijo za preverjanje tipov ločenih modulov. Kompilacijsko zaporedje se ne preizkuša na pravilnost, ker bi to zahtevalo uporabo sistemskega takta. Korektno kompilacijo dosežemo tako, da pred ponovnim prevajanjem zbrišemo vse ´SYM´ zbirke.

Povezovalnik uporabi WITH člene za določitev korektnega povezovanja in vrstnega reda povezovanja. Prevajalnik Janus še nima optimizacije, ki je odvisna tudi od uporabljenega sistema (procesorjev).

4.9. Izjeme

Oglejmo si možnosti obravnave napak in drugih izjemnih pojavov, ki nastanejo pri izvajanju programa. Izjema je dogodek, ki ukinja normalno izvajanje programa. Izvajanje akcij, sproženih s pojavom izjeme, imenujemo obdelava izjeme. Ta obdelava v Janus Adi še ni implementirana.

Janus ne podpira generičnih programskih enot. Predstavitvene specifikacije omogočajo uporabniku obdelavo na ravni materialne opreme; ta možnost v Janus Adi še ni implementirana.

4.10. Vhod in izhod

Program je vselej na določen način povezan z vhodom in izhodom (V/I). Problem Ade je v težavnosti pisanja dolgih V/I pozivov. V Janusu trenutno še ni implementiran splošni V/I naključni dostop.

V/I je v Adi zgrajen na konceptu zbirke. Zbirka je lahko diskovna ali pa je naprava. V/I je v obeh primerih enak. V Janus Adi so zbirke določene z objekti tipa ´file´. Tu je ´file´ omejeni zasebni tip in zbirke se smejo uporabljati kot komponente struktur in se lahko posredujejo v obliki parametrov; ne smejo pa se primerjati ali prirejati (:=).

Tip file_mode je določen za uporabo zbirk. Imamo:

```
TYPE file_mode IS (no_access, read_only,
                   write_only, read_write);
```

Ta tip določa vrsto dovoljenega zbirčnega dostopa. Z zbirko v načinu no_access ni moč nič česar storiti, v načinu read_only ni moč vpisovati itd. Zbirke v načinu no_access niso uporabne in njihovega oblikovanja ne pričakujemo. Zbirke v načinu read_write se lahko uporabljajo le pri V/I z naključnim dostopom.

Procedure, ki lahko delujejo nad zbirkami, so tele:

```
PROCEDURE create (zbirka: IN OUT file;
                  ime: IN string;
                  način: IN file_mode);
```

Ta procedura oblikuje novo zbirko ´ime´ z načinom ´način´. Oblikovanje zbirke načina read_only ima za posledico prazno zbirko, ki praktično ni uporabna. Napaka se pojavi, če poskušamo oblikovati napravo. Če je ´zbirka´ že odprta, se pojavi napaka. Če zbirka ´ime´ že obstaja, bo IO rezultat vseboval 254. Pri polnem imeniku bo IO rezultat enak 255.

```
PROCEDURE open (zbirka: IN OUT file;
                ime: IN string;
                način: IN file_mode);
```

Ta procedura odpre zunanjo zbirko ´ime´ v načinu ´način´. Napaka se pojavi, če je ´zbirka´ že odprta. Procedura open vrne vrednost 255 kot IO rezultat, če zbirke ne najde. Branje in pisanje se začneta na začetku zbirke. Proceduri create in open se ne ujemata z Ada standardom.

```
PROCEDURE close (zbirka: IN OUT file);
```

Ta procedura zapre zbirko. Pri branju zbirke ni treba zapreti. Napaka se pojavi, če zbirka ni bila odprta.

```
PROCEDURE delete (ime: IN string);
```

Ta procedura zbriše zbirko. Pri neustrezno oblikovanem imenu ali imenovani napravi se pojavi napaka.

```
PROCEDURE read (zbirka: IN file;
                predmet: OUT element);
```

Ta procedura prebere ´predmet´ binarno iz zbirke ´zbirka´. Element je lahko poljubnega Janus tipa. Vsi elementi, ki se berejo iz ene zbirke, morajo biti enakega tipa. Branje nastopi pri trenutnem položaju v zbirki in položaj napreduje. Procedura read vrne vrednost 255 kot IO rezultat, ko najde eof (konec zbirke). Če ´zbirka´ ni bila odprta, se pojavi napaka, prav tako v primeru načina nečitljivosti.

```
PROCEDURE write (zbirka: IN file;
                 predmet: IN element);
```

Ta procedura vpiše ´predmet´ binarno v zbirko ´zbirka´. Ostale zahteve so podobne onim za proceduro read.

```
FUNCTION end_of_file (zbirka: IN file)
                      RETURN boolean;
```

Ta funkcija vrne vrednost TRUE, ko je najden konec zbirke; če se uporabi nad zbirko, ki je samo vpisljiva (ni čitljiva), nastopi napaka v času izvajanja.

```
FUNCTION is_open (zbirka: IN file)
                          RETURN boolean;
```

Ta funkcija vrne vrednost TRUE, če je zbirka trenutno odprta.

```
FUNCTION name (zbirka: IN file)
                          RETURN string;
```

Ta funkcija vrne ime odprte zbirke. Vrnjeno ime se lahko uporabi za odprtje drugih zbirk.

Funkciji read in write sta vgrajeni v prevajalnik, ostale funkcije so v knjižničnem paketu IO (knjižnični diski).

Oglejmo si primer branja iz ene zbirke in vpisovanja v drugo zbirko po obdelavi:

```
PACKAGE BODY zbir_test IS
  v_zbirka, iz_zbirka: file;
  spr: poljuben_tip;

  BEGIN
    open (v_zbirka, "VHOD.FIL", read_only);
      -- Odpri zbirko ´v_zbirka´ za branje. Ime
      --  zbirke je VHOD.FIL na trenutnem disku
    delete ("B:IZHOD.FIL");
      -- Zbriši zbirko z imenom IZHOD.FIL iz
      --   diska b. Priporočljivo je, da se
      --   zbirka zbriše pred vpisovanjem.
    create (iz_zbirka, "B:IZHOD.FIL");
      -- Oblikuj novo zbirko IZHOD.FIL.
      --   iz_zbirka je odprta za vpisovanje.
    WHILE NOT end_of_file (v_zbirka) LOOP
      -- Beri iz ´v_zbirka´, dokler se ne
      --  pojavi konec zbirke.
      read (v_zbirka, spr);
        -- Beri en element ´poljuben tip´ iz
        --   v_zbirke. Obdelaj ´spr´.
      write (iz_zbirka, spr);
        -- Vpiši en enlement ´poljuben tip´ v
        --   ´iz_zbirka´.
    END LOOP;
    close (iz_zbirka);
      -- ´iz_zbirka´ se zapre, da se rešijo re-
      --   zultati V/I. Vpisani podatki se lah-
      --   ko izgubijo, če se zbirka ne zapre.
    close (v_zbirka);
      -- To zaprtje ni vselej potrebno, je pa
      --   priporočljivo.
  END zbir_test;
```

Razen binarnih zbirk se v Adi uporabljajo tudi tekstovne zbirke in vse kar je veljalo za binarne, velja tudi za tekstovne zbirke (prejšnji podprogrami).

Procedure get, put, new_line in skip_line imajo kratke oblike, v katerih predpostavljamo manjkajoče (odsotne) V/I zbirke. Manjkajoče zbirke se lahko spreminjajo.

```
FUNCTION standard_input RETURN file;
FUNCTION standard_output RETURN file;
```

Ti dve funkciji vrneta začetni trenutni zbirki. Ti sta vselej uporabniška konzola. Funkciji se uporabljata za shranitev manjkajoče zbirke po njeni spremembi v njo samo; uporabljata se lahko tudi za sporočanje na konzolo.

```
FUNCTION current_input RETURN file;
FUNCTION current_output RETURN file;
```

Ti dve funkciji vrneta manjkajoči (trenutni) zbirki; to sta zbirki za current_input in current_output, če ju uporabnik ni spremenil. Ti funkciji se uporabljata kot sistemska parametra in kot uporabniško določeni rutini, ki ne dovoljujeta manjkajoče zbirke.

```
PROCEDURE set_input (zbirka: IN file);
PROCEDURE set_output (zbirka: IN file);
```

Ti proceduri postavita manjkajoči (trenutni) zbirki na vrednosti njunih parametrov. Če zbirka ni bila odprta ali če nima pravilnega načina, se pojavi napaka. Vhodna zbirka mora biti čitljiva, izhodna pisljiva.

Te rutine omogočajo prenos vseh trenutnih V/I zbirk v poljubno ali iz poljubne zbirke. To velja tudi za vsa sistemska sporočila (npr. o napakah). To pa je lahko pripomoček pri popravljanju programov, saj lahko konzolni izhod preusmerimo v zbirko. Navedene rutine so vsebovane v knjižničnem paketu UTIL.

```
PROCEDURE get (zbirka: IN file;
               predmet: OUT element);
PROCEDURE get (predmet: OUT element);
```

Ti proceduri dobita predmet tipa element iz tekstovne zbirke ´zbirka´. Predmet je tekstovni niz in tipa sta character ali integer.

```
PROCEDURE put (zbirka: IN file;
               predmet: IN element);
PROCEDURE put (predmet: IN element);
```

Ti proceduri vstavita predmet tipa element v tekstovno zbirko ´zbirka´. Izhod ima obliko tekstovnega niza, vstavljajo pa se lahko predmeti skalarnega in niznega tipa.

Za cela števila se lahko uporabljata obliki

```
PROCEDURE put (zbirka: IN file;
               predmet: IN int_type;
               dolžina: IN integer := 0);
PROCEDURE put (predmet: IN int_type;
               dolžina: IN integer := 0);
```

´dolžina´ se nanaša na uporabljeno polje; to se lahko naknadno podaljšuje.

Za preštevne tipa, vključno boolovske, se lahko uporabita obliki

```
PROCEDURE put (zbirka: IN file;
               predmet: IN enum_type;
               dolžina: IN integer := 0);
PROCEDURE put (predmet: IN enum_type;
               dolžina: in integer := 0);
```

Preštevnim vrednostim sledi toliko presledkov, da se napolni polje. Vrednosti se tiskajo z velikimi črkami.

```
PROCEDURE new_line (zbirka: IN file);
PROCEDURE new_line;
```

Ti proceduri povzročita dodajanje konca vrstice v zbirki.

```
PROCEDURE skip_line (zbirka: IN file);
PROCEDURE skip_line;
```

Ti proceduri povzročita branje konca vrstice iz zbirke.

```
FUNCTION end_of_line (zbirka: IN file)
                                RETURN boolean;
```

Ta funkcija vrne TRUE, če je trenutno brana pozicija na koncu vrstice.

```
FUNCTION get_line (zbirka: IN file)
                                RETURN string;
```

Ta funkcija prebere celotno vrstico iz zbirke do vrstičnega mejnika. Ta rutina se lahko uporabi za uporabniško kontrolo V/I, ko se včitani niz iz konzole ponovno interpretira.

Procedure get, put, new_line in skip_line so vgrajene v prevajalnik, ostale rutine pa so vsebovane v IO knjižničnem paketu.

4.11. Knjižnične procedure

Zanimiva lastnost Ade je možnost uporabniškega definiranja knjižničnih rutin. Zradi te lastnostri jezik Ada nima definicij standardnih knjižničnih procedur. Obstaja pa nekaj pomembnih operacij, ki zahtevajo globlje razumevanje prevajalnika in te bomo kratko opisali.

Imamo p o d p r o g r a m e z a u p r a v l j a n j e k u p o v (heap).

```
PROCEDURE dispose (kazalec: IN OUT p_dostop);
```

Ta podprogram sprosti pomnilnik objektov, na katere kaže ´kazalec´, ki je lahko poljubnega dostopnega tipa (´p_dostop´). Če je kazalec enak NULL kazalcu, se ne zgodi ničesar. Kazalec se postavi na NULL po vrnitvi iz procedure dispose. Ta procedura je ekvivalentna generični Ada proceduri unchecked_deallocation. Procedura je avtomatično določena za poljuben dostopni tip.

```
FUNCTION memavail RETURN integer;
```

Ta funkcija vrne obseg (število zlogov) pomnilnika, ki obstaja med kupom in skladom. Ta del pomnilnika se lahko uporabi ali za kup ali za sklad. Poljuben blok se lahko dodeli iz tega dela pomnilnika. Na razpolago je več pomnilnnika za kup, če je bila uporabljena procedura dispose.

```
FUNCTION maxavail RETURN integer;
```

Ta funkcija vrne število prostih lokacij, ki so uporabne za kup, vključno s tistimi, pridobljenimi z dispose. Nekateri pomnilni bloki so lahko premali za uporabo. To ni dobra metoda za štetje prostih lokacij.

Primer:

```
dispose (kazalec); dispose (vozlišče.levo);
put (memavail()); put (" preostalih zlogov");
```

Procedura dispose je vgrajena v prevajalnik. Funkciji memavail in maxavail se nahajata v knjižničnem paketu UTIL.

Janus ima več podprogramov za m a n i p u l a c i j o b i t o v i n z l o g o v. Manipulacija je možna v okviru celih števil. Bitni parametri za rutine setbit, clrbit in tstbit morajo biti v intervalu 0..15.

```
FUNCTION hi (vred: IN integer) RETURN byte;
FUNCTION lo (vred: IN integer) RETURN byte;
```

Ti funkciji vrneta višji (hi) oziroma nižji (lo) zlog celoštevilske vrednosti ´vred´.

```
PROCEDURE setbit (vred: IN OUT integer;
                  bit: IN integer);
```

Ta procedura postavi bit (na vrednost = 1) na mestu ´bit´ v ´vred´.

```
PROCEDURE clrbit (vred: IN OUT integer;
                  bit: IN integer);
```

Ta procedura izniči bit (na vrednost = 0) na poziciji ´bit´ v številu ´vred´.

```
FUNCTION tstbit (vred: IN integer;
                 bit: IN integer)
                      RETURN boolean;
```

Ta funkcija vrne TRUE, če je vrednost na poziciji ´bit´ števila ´vred´ enaka 1 oziroma vrne FALSE pri bitni vrednosti 0.

```
FUNCTION lnot (vred: IN integer)
                      RETURN integer;
```

Ta funkcija vrne logični ´ne´ bitov v vrstnem redu števila ´vred´ (eniški komplement).

```
FUNCTION land (vred1, vred2: IN integer
                      RETURN integer;
```

Ta funkcija vrne logični ´in´ pripadajočih bitov števil ´vred1´ in ´vred2´.

```
FUNCTION lor (vred1, vred2: IN integer)
                      RETURN integer;
```

Ta funkcija vrne logični ´ali´ pripadajočih bitov števil ´vred1´ in ´vred2´.

```
FUNCTION lxor (vred1, vred2: IN integer)
                      RETURN integer;
```

Ta funkcija vrne logično ´vsoto po modulu 2´ pripadajočih bitov števil ´vred1´ in ´vred2´.

Primeri:

```
a := lo(-123); b := hi(vred);
setbit(vred,3); clrbit(presl(i),j);
IF tstbit(12,i) THEN ...
zast := lnot(spr);
začasnica := land(presl(i), zast);
zastavice := lxor(lor(zast, 8#70000#), spr);
```

Funkcije z izjemo hi in lo so v knjižničnem paketu BIT. hi in lo sta v paketu UTIL, ki je zbirniški paket z velikim številom malih rutin. Knjižnični paketi se nahajajo na knjižnični disketi Janus Ade.

Za p r o g r a m s k o k r m i l j e n j e je predvidenih več rutin.

```
PROCEDURE halt;
```

ustavi program z izstopom skozi halt vstopno točko (v knjižnici časa izvajanja). Ta procedura nima vrnitve.

```
PROCEDURE err_exit;
```

ukine izvajanje programa v času izvajanja podobno kot pri napaki.

```
FUNCTION command_line RETURN string;
```

Ta funkcija vrne ukazno vrstico, s katero je bil program poklican. Vse tri zadnje rutine so v knjižničnem paketu UTIL.

```
PROCEDURE chain (ime: IN string);
```

Ta procedura začne izvajati program z imenom ´ime´. Ta program se naloži in se izvaja kot nov program. Če imenovani program ni najden, se tiska sporočilo.

```
PROCEDURE prog_call (ime: IN string);
```

Ta procedura (ki je ni pri 8-bitnih sistemih) izvaja program ´ime´, klicajoči program ostane rezidenten. Klicani program lahko izvaja prog_return, ki povzroči vrnitev v klicajoči program.

```
PROCEDURE prog_return;
```

Ta procedura (ki je ni pri 8-bitnih sistemih) vrne krmiljenje klicajočemu programu. Procedure chain, prog_return in prog_call so v knjižničnem paketu CHAIN.

Nestandardni V/I podprogrami so tile:

```
FUNCTION disk_full (zbirka: IN file)
                      RETURN boolean;
```

Ta funkcija vrne TRUE, če je disk poln ali če

se je pojavila napaka pri vpisu ali put pozivu za zbirko 'zbirka'. Ta funkcija poenostavi preizkus diskovne napake.

```
FUNCTION eof (zbirka: IN file)
                              RETURN boolean;
```

Ta funkcija je enaka funkciji end_of_file, vendar deluje nad binarnimi zbirkami. Funkcija end_of_file lahko vrne nepravi TRUE, če jo uporabimo nad binarno zbirko.

```
PROCEDURE put_hex (zbirka: IN OUT file;
                   vred: IN integer);
```

Ta procedura zapiše 'vred' v 'zbirka' heksadecimalno. Število se zapiše z natanko 4 številkami. Vendar to število ni izraženo v Ada formatu baznega števila (ki ne ustreza za zbirnik).

Primera:

```
put_hex(tren_izhod(),254); -- tiska 00FE
put_hex(tren_izhod(),-20); -- tiska FFEB
```

Podprogrami disk_full, eof in put_hex so v knjižničnem paketu IO.

```
PROCEDURE read_blk (zbirka: IN OUT file;
                    sek: OUT sector;
                    blok: IN integer);
```

Ta procedura prebere logični sektor 'sek' z 128 zlogi iz 'zbirka', implementira pa omejeni, naključni dostop, ko uporabnik sam blokira in deblokira vrednosti. Bločno število 'blok' mora biti pozitivno. Preizkusa na zbirki ni; zato je potrebna določena previdnost pri uporabi te procedure.

```
PROCEDURE write_blk (zbirka: IN OUT file;
                     sek: IN sector;
                     blok: IN integer);
```

Procedura zapiše logični sektor 'sek' s 128 zlogi v 'zbirka'.

```
PROCEDURE rclose (zbirka: IN OUT file);
```

Ta rutina ima enak učinek kot rutina close nad zbirko z naključnim dostopom; nahaja se v paketu BLKIO.

Podprogrami za manipulacijo nizov sprejemajo nizne tipe; pri tem se njihovi argumenti ne modificirajo. Celoštevilski parametri in rezultati so v območju 0..255, če ni drugače določeno. Vse te rutine so v paketu STRLIB.

```
FUNCTION length (niz: IN string)
                              RETURN integer;
```

Ta funkcija vrne dolžino niza 'niz'.

```
FUNCTION remove (niz: IN string;
                 pozicija, obseg: IN integer)
                              RETURN string;
```

Ta funkcija odstrani 'obseg' znakov iz niza 'niz', začenši pri znaku 'pozicija'. Napaka se pojavi, če 'pozicija' ni v okviru niza 'niz' ali če ni dovolj znakov za odstranitev v nizu.

```
FUNCTION insert (izvir, namen: IN string;
                 pozicija: IN integer)
                              RETURN string;
```

Ta funkcija vstavi izvirni niz v namenskega pred znak, ki je označen s pozicijo in vrne rezultatni niz. Napaka se pojavi, če pozicija ni v okviru namenskega niza ali če je rezultatni niz predolg za predstavitev.

```
FUNCTION extract (niz: IN string;
                  pozicija, obseg: IN integer)
                              RETURN string;
```

Ta funkcija izvleče 'obseg' znakov iz niza 'niz' pri položaju 'pozicija'. Napaka se pojavi, če 'pozicija' ni v okviru niza ali če v nizu ni dovolj znakov (glede na 'obseg') za izvleček. Ta funkcija je podobna Basic funkciji MID$ in funkciji LEFT$ in RIGHT$ v Basicu je moč simulirati s funkcijo extract.

```
FUNCTION position (vzorec, niz: IN string)
                              RETURN integer;
```

Ta funkcija vrne pozicijo prve pojavitve vzorca v nizu; če vzorca ne najde, se vrne ničla.

```
FUNCTION char_to_str (znak: IN character)
                              RETURN string;
```

Ta funkcija pretvori znak v niz dolžine 1. Janus ne dovoljuje stikanja znakov v niz in ta funkcija je potrebna za vrsto elementarnih operacij.

```
FUNCTION str_to_int (niz: IN string)
                              RETURN integer;
```

Ta procedura pretvori niz v celo število. Niz mora predstavljati celo število v veljavnem območju Janusa. Vodeči presledki in tabulirni znaki se ne upoštevajo. Prvi nenumerični znak označuje konec pretvorbe.

```
FUNCTION int_to_string (c_štev: IN integer)
                              RETURN string;
```

Ta funkcija pretvori celo število v niz. Vsako celo število se sprejme, vrne se najkrajši mogoči niz.

Primer:

```
WITH strlib;
PACKAGE BODY nizteст IS
  USE strlib;
  zn: character;
  niz, niz2: string;
BEGIN
  niz := "nizni test";
  niz2 := "programska oprema";
  put("niz = "); put(niz); new_line;
  put("niz2 = "); put(niz2); new_line;
  put("dolžina(niz) = "); put(length(niz));
                                new_line;
  put("položaj("" "",niz) = ");
    put(position(" ",niz)); new_line;
  put("položaj(""programska"",niz2) = ");
    put(position("programska",niz2)); new_line;
  put("položaj(niz2,niz) = ");
    put(position(niz2,niz)); new_line;
  put("odstrani(niz,8,3) = ");
    put(remove(niz,8,3)); new_line;
  put("odstrani(niz,položaj(""zni"",niz),3) = ");
    put(remove(niz,position("zni",niz),3));
    new_line;
  put("vstavi(""rutinski "",niz,7) = ");
    put(insert("rutinski ",niz,7)); new_line;
  put("vsravi(niz2,odstrani(niz,1,5),12) = ");
    put(insert(niz2,remove(niz,1,5),12));
    new_line;
  put("izvleci(niz2,12,6) = ");
    put(extract(niz2,12,6)); new_line;
    -- To je podobno MID$(niz,poz,obseg) v
    --  Basicu. Za LEFT$(niz,obseg) bi imeli:
    --  extract(niz,1,obseg) kot primer.
  put("izvleci(niz,1,5) = ");
    put(extract(niz,1,5)); new_line;
    -- Za RIGHT$(niz,obseg) bi imeli npr.:
    --  extract(niz,length(niz)-obseg,obseg)
  put("izvleci(niz,dolžina(niz)-4,4) = ");
    put(extract(niz,length(niz)-4,4));
    new_line;
```

```
  put("niz & znak_v_niz('$') = ");
    put(niz & char_to_str('$')); new_line;
    -- niz & '$' bi bilo nepravilno!
  get(zn);
  put("vstavi(znak_v_niz(zn),niz,4) = ");
    put(insert(char_to_str(zn),niz,4));
    new_line;
  put("niz_v_celo_št("" - 123"") = ");
    put(str_to_int(" - 123")); new_line;
  put("celo_št_v_niz(432) = ");
    put(int_to_str(432)); new_line;
    -- Te procedure se uporabljajo za oblikova-
    --  nje besedil in za uporabniško prijazen
    --  vhod.
END nizni_test;
```

Ta program generira tale izpis:

```
niz = nizni test
niz2 = programska oprema
dolžina(niz) = 10
položaj(" ",niz) = 6
položaj("programska",niz2) = 1
položaj(niz2,niz) = 0
odstrani(niz,8,3) = nizni t
odstrani(niz,položaj("zni",niz),3) = ni test
vstavi("rutinski ",niz,7) = nizni rutinski test
vstavi(niz2,odstrani(niz,1,5),12) =
                              programska testoprema
izvleci(niz2,12,6) = oprema
izvleci(niz,1,5) = nizni
izvleci(niz,dolžina(niz)-4,4) = test
niz & znak_v_niz('$') = nizni test$
-- Če je vtipkani znak (zn) enak '?', imamo:
vstavi(znak-v-niz(zn),niz,4) = niz?ni test
niz_v_celi_št(" - 123") = -123
celo_št_v_niz(432) = 432
```

5. Uporaba Janus prevajalnika

5.1. Prevajalne možnosti in ukazna vrstica

Janus prevajalnik pokličemo v izvajanje s formatom:

JANUS [d:] ime_zbirke [.raz] {/opcija}

Tu je ime_zbirke CP/M zbirka na disku [d:] z razširitvijo [.raz] in prevajalniškimi opcijami {/opcija} . Če diskovna enota ni imenovana, se aktivira trenutni disk; če razširitev ni imenovana, obstaja zbirka tipa .PKG .

Prevajalniške opcije pa so tele:

D - ne generiraj popravljalnega koda;
B - kratka sporočila o napakah; vrstica z napako se ne tiska;
W - izda se pojasnilo o nezdružljivosti z Janus Ado;
C - pogojno prevajanje je vključeno;
L - oblikuj listo zbirke z imenom ime_zbirke.PRN na istem disku, kot je ime_zbirke.PKG.
Ld - oblikuj listo zbirke na disku 'd'; izbira je v intervalu 'A' .. 'W';
LX - oblikuj listo zbirke na konzolni enoti (CON:);
LZ - oblikuj listo zbirke na tiskalni enoti (LST:);
Rd - usmeri .JRL zbirko na disk 'd' (interval je 'A' .. 'W'); če 'd'-ja ni, bo ta zbirka na trenutno aktivnem disku;
Sd - preusmeri vmesne in končne zbirke na imenovani disk; ta potreba se lahko pojavi pri diskovnih sistemih, ki imajo manj kot 250k zlogov na disketi (npr. pri 5-colskih disketah); uporaba te opcije dovoljuje zamenjavo diskete za naložitev paketne specifikacije (.SYM zbirke); prevajalnik išče pri tej opciji .SYM zbirke na vstavljenih disketah in če jih ne najde, se lahko prevajalniška disketa zamenjuje z disketami za nadaljnje iskanje; tudi povezovalnik ima podobno možnost, kar omogoča razvoj velikih programov na sistemih z malimi disketnimi obsegi;
Q - tiha sporočila o napakah; ta opcija povzroči, da prevajalnik ne čaka na akcijo uporabnika po napaki; v navadnem načinu prevajalnik počaka po vsaki napaki na uporabniško soglasje, ali naj s prevajanjem preneha ali pa ga nadaljuje; ta opcija je smiselna pri odsotnosti uporabnika, tako da prevajalnik lahko opravi svoje delo do konca; napake se tu ne pojavijo na zaslonu, zato je smiselna uporaba tiskalnika; določene sintaksne napake lahko povzročijo tiskanje velikega števila sporočil.

Če naštetih opcij ne uporabimo, pa velja tole:

D - generira se popravljalni kod;
B - sporočila o napakah se ne pojavijo;
W - ni sporočil o nezdružljivosti z Janus Ado;
C - pogojno prevedene vrstice se obravnavajo kot komentarji;
L - ne generira se lista zbirke;
R - ista disketa kot jo ima vhodna zbirka;
S - ista disketa kot je za prevajalnik;
Q - prevajalnik zahteva potrditev za prenehanje prevajanja po vsaki napaki.

Primeri:

```
JANUS preizkus/Q/LX
JANUS preizkus.IKE/W
JANUS preizkus
JANUS preizkus   .IKE /B /W /L
```

5.2. Delovanje prevajalnika

Delovanje štiriprehodnega prevajalnika za Janus Ado preizkusimo na zbirki (Ada programu) z imenom QSORT1.PKG, ki je prikazana z listo 6. Med prevajalnim postopkom dobimo na zaslonu izpis, ki je prikazan z listo 7. Ta izpis ima tale pomen:

```
C   JANUS  B:QSORT1/L/SB  -- Prevedi QSORT1 in
        -- generiraj listo (L).  Zbirke naj bodo
        -- na disku B.
J A N U S - VERSION 1.4.5
COPYRIGHT (C) 1981,82 - R R SOFRWARE
8080/Z80 VERSION (CP/M-80)
SCRATCH FILES ON DISK B   -- Ta vrstica pokaže
        -- vtipkano opcijo, tako da vidimo, da
        -- jo je prevajalnik sprejel.
LISTING ON - FILENAME = B:QSORT1.PRN
INPUT FILE IS B:QSORT1.PKG  -- Vhodna zbirka.
##################
        -- Po en znak dvojnega križa se natisne
        -- za vsakih včitanih 16 vrstic.  To za-
        -- poredje dokazuje, da prevajalnik pre-
        -- vaja (pravilno deluje).
STRSPACE PTR - 854 USAGE - 26%
HASH TABLE - 136 BUCKETS USED, USAGE - 28%
        -- Ta statistika pove, koliko prevajal-
        -- niškega prostora je bilo porabljene-
        -- ga.  Če se poraba približa 100%, je
        -- treba program razdeliti na manjše mo-
        -- dule.
PARSING COMPLETED - 272 LINES FOUND
PASS II   -- Naloži se drugi prehod in se za-
        -- čne izvajati.
NO PACKAGE SPECIFICATION FOUND
        -- To pomeni, da se prevaja paket, za
        -- katerega ni bila prevedena specifika-
        -- cija.  To je lahko glavni program, ki
        -- ne more biti uporabljen v kakem dru-
        -- gem modulu.
%%%%%  -- Po en znak '%' se tiska za vsako upo-
```

```
C>TYPE B:QSORT1.PKG
PACKAGE BODY QSORT1 IS

-- OCENJEVALNI PROGRAM QUICKSORT
-- TA PROGRAM JE LE PRIMER SPLOSNE SINTAKSE JE-
--   ZIKA ADA V PRIMERJAVI Z JEZIKOM PASCAL.
-- PODOBEN PROGRAM JE V WIRTHOVEM  A + DS = P.
-- **** OPOMBE ZA PASCALSKE PROGRAMREJE JOSIJO
--   OZNAKO --** .
-- POGOJNO PREVEDLJIVE VRSTICE SO NAMENJENE O-
--   PAZOVANJU PREVEDENEGA KODA IN RAZUMEVANJU
--   DELOVANJA PROGRAMA.

    ZERO : CONSTANT := 0;
    NUMRECS : CONSTANT := 200;
    -- NAJVECJE STEVILO SORTIRANIH ZAPISOV
    TYPE ITEM IS RECORD
            DATA1,DATA2 : CHARACTER;
            KEY1 : INTEGER;
        END RECORD;
    TYPE DRECORD IS ARRAY (ZERO..NUMRECS) OF
      ITEM;
    NEWREC,DATREC : DRECORD;
    TOTAL : INTEGER;-- TOTAL NUMBER OF RECORDS
    --** DEKLARACIJE SO TAKE KOT V PASCALU.

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

    PROCEDURE GETRECS (FILREC : IN OUT DRECORD;
      SUM : IN OUT INTEGER) IS
        -- INICIALIZACIJA POLJA ZAPISOV, KI SO
        --   PODOBNI VAR PARAMETROM V PASCALU.

        ALINE : INTEGER;
    BEGIN
        FOR I IN 1..100 LOOP
            PRAGMA ARITHCHECK(OFF);
            ALINE := (I * 3377) MOD 97;
              -- UPORABI PRESTOP IN MOD ZA GE-
              --   NERIRANJE PSEVDONAKLJUCNEGA
              --   ZAPOREDJA.
            PRAGMA ARITHCHECK(ON);
            FILREC(I).KEY1 := ALINE;
            FILREC(I).DATA1 := 'A';
            FILREC(I).DATA2 := 'Z';
            PUT(I); PUT(":  "); PUT(ALINE);
              NEW_LINE;
        END LOOP;
        SUM := 100;
        --** PROCEDURA NE POTREBUJE RETURN STA-
        --**   VKOV, CE ZELIMO IZSTOPITI NA
        --**   NJENEM KONCU.
    END GETRECS;

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

    PROCEDURE FILEOUT (OUTFIL : IN DRECORD;
      SEND : IN INTEGER) IS
        -- IZPISE SORTIRANE ZAPISE NA ZASLON.

        TOT : INTEGER;
    BEGIN
        TOT := ZERO;
        WHILE TOT /= SEND LOOP
            TOT := TOT + 1;
            PUT(TOT); PUT(": ");
            PUT(OUTFIL(TOT).KEY1); NEW_LINE;
        END LOOP;
        PUT("TOTAL "); PUT(SEND); NEW_LINE;
    END FILEOUT;

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

    PROCEDURE QUICKSORT (LIST : IN OUT DRECORD;
      NUMB : IN INTEGER) IS

        MAXSUB : CONSTANT := 21;
        -- NAJMANJSA PODZBIRKA, KI JE DOVOLJENA
        --   V QSORT.
        STACKDEP : CONSTANT := 20;
        -- OBSEG SKLADA.
        TYPE INDICIES IS RECORD
        -- DELITVENI ZAPISI SKLADA.
              BEG,END : INTEGER;
            END RECORD;
        STK,I,J,LEFT,RIGH : INTEGER;
        T1REC,T2REC : ITEM;-- ZACASNI ZAPISI.
        SUBTYPE STACKPTR IS INTEGER
          RANGE 1..STACKDEP;
        TYPE TEMPARR IS ARRAY (STACKPTR) OF
          INDICIES;
        STACK : TEMPARR;
        --** VRSTNI RED DEKLARACIJ JE POTREBEN,
        --**   TAKO DA SE TIPI IN KONSTANTE
        --**   LAHKO UPORABLJAJO V DRUGIH
        --**   DEKLARACIJAH.

        FUNCTION MEDIAN (LISTNAM : IN DRECORD;
          LEF,RIT : IN INTEGER)
            RETURN INTEGER IS
        --** NACIN POSREDOVANJA PARAMETROV JE
        --**   NAVEDEN ZA DVOPICJEM.
        --** PARAMETROV NACINA IN NI POTREBNO
        --**   SPECIFICIRATI.

            MED : INTEGER;
        BEGIN
            --** IME FUNKCIJE SE NE SME UPORAB-
            --**   LJATI KOT ZACASNA SPREMEN-
            --**   LJIVKA, KER JE VRNITEV TA-
            --**   KOJSEN SKOK NA KONEC.
            MED := (LEF + RIT) / 2;
            PUT("MED="); PUT(MED); PUT(" LEF=");
              PUT(LEF); PUT(" RIT=");
            PUT(RIT); NEW_LINE; PUT("     ");
              PUT(LISTNAM(MED).KEY1);
            PUT("     "); PUT(LISTNAM(LEF).KEY1);
              PUT("     ");
            PUT(LISTNAM(RIT).KEY1);
            PUT(" SHOULD RETURN MEDIAN VALUE");
              NEW_LINE;
            IF (LISTNAM(RIT).KEY1 >
                  LISTNAM(MED).KEY1) THEN
                IF LISTNAM(MED).KEY1 >
                     LISTNAM(LEF).KEY1
                THEN RETURN(MED);
                ELSIF (LISTNAM(RIT).KEY1 >
                         LISTNAM(LEF).KEY1)
                THEN RETURN(LEF);
                ELSE RETURN(RIT);
                END IF;
            ELSIF LISTNAM(MED).KEY1 <
                    LISTNAM(LEF).KEY1
              THEN RETURN(MED);
            ELSIF LISTNAM(RIT).KEY1 <
                    LISTNAM(LEF).KEY1
              THEN RETURN(LEF);
            ELSE RETURN(RIT);
            --** FUNKCIJE POTREBUJEJO RETURN
            --**   STAVKE.
            END IF;
        END MEDIAN;

        PROCEDURE STINSERTSORT (NEWREC :
          IN OUT DRECORD; M,N : INTEGER) IS
            -- MORE EFFICENT
            -- 'M' JE ZACETNI, 'N' KONCNI POLO-
            --   ZAJ.
            LFT : INTEGER;
            -- LEVA SORTIRNA USTAVITEV.
            SAVREC,XREC : ITEM; -- ZACASNI ZAP.
        BEGIN
            SAVREC := NEWREC(M - 1);
            -- RESI ZAPIS PRED SORTIRNIM OBMOC.
            FOR RGT IN (M + 1)..N LOOP
            -- DESNA SORTIRNA USTAVITEV.
                XREC := NEWREC(RGT);
                NEWREC(M - 1) := XREC;
                LFT := RGT - 1;
                WHILE XREC.KEY1 <
                  NEWREC(LFT).KEY1 LOOP
                  -- PREKLOPNI ZAPISI.
                    NEWREC(LFT + 1) :=
                                NEWREC(LFT);
                    LFT := LFT - 1;
                END LOOP;
                NEWREC(LFT + 1) := XREC;
            END LOOP;
            NEWREC(M - 1) := SAVREC;
```

```
        -- SHRANI RESENI ZAPIS.
    END STINSERTSORT;

BEGIN
    IF NUMB < MAXSUB THEN
        STINSERTSORT(LIST,1,NUMB);
    ELSE -- ZBIRKA JE DALJSA OD MINIMALNE-
         --   GA OBSEGA PODZBIRKE.
        STK := 1;
        STACK(1).BEG := 1;
        STACK(1).EDN := NUMB;
        LOOP -- VZAMI VRHNJO ZAHTEVO IZ
             --   SKLADA.
            LEFT := STACK(STK).BEG;
            RIGH := STACK(STK).EDN;
@           PUT("STK="); PUT(STK);
@             PUT(" LEFT="); PUT(LEFT);
@           PUT(" RIGHT="); PUT(RIGH);
@             NEW_LINE;
            STK := STK - 1;-- SORTIRAJ POD-
            -- ZBIRKE MANJSE OD MAXSUB Z
            --   NEPOSREDNIM VSTAVITVENIM
            --   SORTIRANJEM.
            IF (RIGH - LEFT) < MAXSUB THEN
              STINSERTSORT(LIST,LEFT,RIGH);
            ELSE
                LOOP -- RAZDELI INTERVALE.
                    I := LEFT;
                    J := RIGH;
@                   PUT("I=LEFT="); PUT(I);
@                   PUT(" J=RIGH="); PUT(J);
@   PUT(" LOW AND HIGH MARKS OF CURRENT SORT");
@                   NEW_LINE;
                    TIREC :=
                  LIST(MEDIAN(LIST,LEFT,RIGH));
@                   PUT("TIREC.KEY1=");
@                   PUT(TIREC.KEY1);
@           PUT(" SHOULD MATCH MEDIAN VALUE");
@                   NEW_LINE;
                    LOOP
                        WHILE LIST(I).KEY1 <
                           TIREC.KEY1 LOOP
                            I := I + 1;
                        END LOOP;
                        WHILE TIREC.KEY1 <
                         LIST(J).KEY1 LOOP
                            J := J - 1;
                        END LOOP;
                        IF I <= J THEN
                            T2REC.KEY1 :=
                              LIST(I).KEY1;
                            LIST(I).KEY1 :=
                              LIST(J).KEY1;
                            LIST(J).KEY1 :=
                                T2REC.KEY1;
                            I := I + 1;
                            J := J - 1;
                        END IF;
                        EXIT WHEN I > J;
                    END LOOP;--PONOVI ZANKO
@                 PUT("DONE PARTITION - I=");
@                   PUT(I); PUT(" J=");
@                   PUT(J); NEW_LINE;
                    IF (J - LEFT) <
                            (RIGH - I) THEN
                        IF I < RIGH THEN
@                       PUT("STACK RIGHT");
                            STK := STK + 1;
                        STACK(STK).BEG := I;
                        STACK(STK).EDN := RIGH;
                        END IF;
                        RIGH := J;
    -- NADALJUJ S SORTIRANJEM LEVE PARTICIJE.
                    ELSE
                        IF LEFT < J THEN
@                        PUT("STACK LEFT");
                            STK := STK + 1;
                        STACK(STK).BEG := LEFT;
                        STACK(STK).EDN := J;
                        END IF;
                        LEFT := I;
    -- NADALJUJ S SORTIRANJEM DESNE PARTICIJE.
                    END IF;
                    EXIT WHEN LEFT >= RIGH;
                END LOOP;
                    END IF;
                    EXIT WHEN STK = ZERO;
                END LOOP; -- PONOVI ZANKO.
            END IF;
        END QUICKSORT;

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

BEGIN
    GETRECS(DATREC,TOTAL);
    PUT("**** SORT START ****"); NEW_LINE;
    FOR I IN 1..30 LOOP
        NEWREC := DATREC;
        QUICKSORT(NEWREC,TOTAL);
    END LOOP;
    PUT("**** 30 SORTS DONE ****"); NEW_LINE;
    FILEOUT(NEWREC,TOTAL);
    PUT("QSORT FINISH"); NEW_LINE;
END QSORT1;
```

Lista 6. Ta lista prikazuje na tej in prejšnji strani Ada program za hitro sortiranje. Pomen tega programa je predvsem v njegovi primerjavi z določenimi jezikovnimi konstrukti jezika Pascal. Hkrati je na tem primerku pokazana tudi uporaba prevajalnika z interakcijo na listi 7.

Lista 7. Ta lista prikazuje sporočila na konzoli pri prevajanju programa QSORT1.PKG z liste 6. Pomen posameznih sporočil je pojasnjen v tekstu članka.

```
C>JANUS B:QSORT1/L/SB
J A N U S - VERSION 1.4.5
COPYRIGHT (C) 1981,82 - R R SOFTWARE
8080/Z80 VERSION (CP/M-80)
SCRATCH FILES ON DISK B
LISTING ON - FILENAME = B:QSORT1.PRN
INPUT FILE IS B:QSORT1.PKG
##################
STRSPACE PTR - 854 USAGE - 26%
HASH TABLE - 136 BUCKETS USED, USAGE - 28%
PARSING COMPLETED - 272 LINES FOUND
PASS II
NO PACKAGE SPECIFICATION FOUND
%%%%%
COMPILER TABLE USAGE:
                  CURRENT MAX.  LIMIT
SYMBOL TABLE        5     45    400
TYPE TABLE                14     90
PROCEDURE TABLE           6     80
PARAMETER TABLE           13     90
RANGE TABLE               7    100
PARSE STACK         3     25     50
PASS II COMPLETED
PASS III - 8080
LISTING FILE B:QSORT1.PRN
*****
DATA USED = 1612
HIGHEST DATA ADDRESS = 064C
HIGHEST CODE ADDRESS = 0BEE
PASS III COMPLETED
PASS IV
THANK YOU FOR USING JANUS
```

```
        -- Številano podprogramsko telo.
COMPILER TABLE USAGE:
                CURRENT MAX. LIMIT
SYMBOL TABLE        5    45   400
TYPE TABLE               14    90
PROCEDURE TABLE           6    80
PARAMETER TABLE          13    90
RANGE TABLE               7   100
PARSE STACK         3     25    50
        -- Ta preglednica kaže uporabo različnih
        -- prevajalniških tabel. Če je katera od
        -- tabel blizu svoje meje, naj bi se za-
        -- devni objekti v programu uporabljali
        -- manj pogosto. To lahko pomeni delitev
        -- programa na več modulov. Analizni
        -- sklad kaže globino vgnezdenja posame-
        -- znih programskih konstruktov.
PASS II COMPLETED
        -- Prehod 3 se zažne nalagati in bo iz-
        -- vajan.
PASS III - 8080
LISTING FILE B:QSORT1.PRN
*****   -- Znak '*' se tiska za vsako uporablje-
        -- no podprogramsko telo programa.
DATA USED = 1612
        -- Decimalno število podatkovnih zlogov,
        -- uporabljenih v programu. To so samo
        -- statično dodeljeni paketni podatki.
        -- Podprogramski lokalni podatki se
        -- shranjujejo v skladu.
HIGHEST DATA ADDRESS = 064C
        -- To je obseg podatkovnega segmenta v
        -- heksadecimalni obliki. Ta podatek je
        -- uporaben za preizkušanje povezovalnih
        -- naslovov.
HIGHEST CODE ADDRESS = 0BEE
        -- To je vrhnji naslov ukaznega koda v
        -- heksadecimalni obliki. Ta naslov ka-
        -- že, ali je program prestopil območje.
PASS III COMPLETED
        -- Začne se nalagati prehod 4, ki se po-
        -- tem izvaja.
PASS IV
THANK YOU FOR USING JANUS
        -- Sporočilo konca delovanja prevajal-
        -- nika.
```

Pri prevajanju se sporočajo napake takoj, ko so zaznane, to je v prvih dveh prehodih; v prehodih 3 in 4 skorajda ni več teh sporočil.

5.3. Uporaba prevedenega programa

Prevajalnik izdela zbirko tipa .JRL (Janus ReLocatable); ta zbirka mora biti povezana (kombinirana) s knjižničnimi rutinami izvajalnega časa, da bi tako dobili izvršljivi program. V našem primeru bomo imeli ukazno vrstico

JLINK B:qsort1

Po povezavi bomo vtipkali za uporabo tega programa enostavno 'qsort1'. Tudi pri povezovanju imamo na razpolago različne opcije (npr. za vstavljanje knjižničnih disket), pojavljajo pa se tudi značilna sporočila o napakah pri povezovanju.

5.4. Janus diskete in razpoložljivi prostor

V listi 8 imamo sezname treh disket Janus Ade, in sicer prevajalniške, povezovalniske in storitvene s primeri. Povejmo še, kako naj bodo posamezne zbirke razporejene po posameznih diskovnih enotah.

```
A>STAT C:*.*

 RECS  BYTES  EXT ACC
    3     1K    1 R/W C:BIT.SYM
    2     1K    1 R/W C:BLKIO.SYM
    2     1K    1 R/W C:CHAINLIB.SYM
   93    12K    1 R/W C:ERROR.MSG
    6     1K    1 R/W C:IO.SYM
  177    23K    2 R/W C:JANUS.COM
  161    21K    2 R/W C:JANUS1.OVL
  149    19K    2 R/W C:JANUS2.COM
  262    33K    3 R/W C:JANUS2A.COM
  245    31K    2 R/W C:JANUS2B.COM
  322    41K    3 R/W C:JANUS3.COM
   73    10K    1 R/W C:JANUS4.COM
   29     4K    1 R/W C:JLIB80.SYM
    1     1K    1 R/W C:NAME.$$$
   13     2K    1 R/W C:OPCODE.SYM
    3     1K    1 R/W C:STANDARD.SYM
    4     1K    1 R/W C:STRLIB.SYM
    4     1K    1 R/W C:TIMELIB.SYM
    6     1K    1 R/W C:UTIL.SYM
BYTES REMAINING ON C: 36K

A>STAT B:*.*

 RECS  BYTES  EXT ACC
   25     4K    1 R/W B:BIT.ASM
    2     1K    1 R/W B:BIT.JRL
    9     2K    1 R/W B:BIT.LIB
    3     1K    1 R/W B:BIT.SYM
    7     1K    1 R/W B:BLKIO.JRL
    9     2K    1 R/W B:BLKIO.LIB
   25     4K    1 R/W B:BLKIO.PKG
    2     1K    1 R/W B:BLKIO.SYM
    8     1K    1 R/W B:CHAINLIB.JRL
    9     2K    1 R/W B:CHAINLIB.LIB
   29     4K    1 R/W B:CHAINLIB.PKG
    2     1K    1 R/W B:CHAINLIB.SYM
   43     6K    1 R/W B:IO.JRL
   13     2K    1 R/W B:IO.LIB
  113    15K    1 R/W B:IO.PKG
    6     1K    1 R/W B:IO.SYM
  631    79K    5 R/W B:JLIB80.ASM
   48     6K    1 R/W B:JLIB80.JRL
   37     5K    1 R/W B:JLIB80.LIB
   29     4K    1 R/W B:JLIB80.SYM
  157    20K    2 R/W B:JLINK.COM
    1     1K    1 R/W B:OPCODE.JRL
   25     4K    1 R/W B:OPCODE.LIB
   13     2K    1 R/W B:OPCODE.SYM
   33     5K    1 R/W B:STRLIB.JRL
   13     2K    1 R/W B:STRLIB.LIB
   41     6K    1 R/W B:STRLIB.PKG
    4     1K    1 R/W B:STRLIB.SYM
   19     3K    1 R/W B:TIMELIB.JRL
    9     2K    1 R/W B:TIMELIB.LIB
   29     4K    1 R/W B:TIMELIB.PKG
    4     1K    1 R/W B:TIMELIB.SYM
   33     5K    1 R/W B:UTIL.ASM
    3     1K    1 R/W B:UTIL.JRL
   13     2K    1 R/W B:UTIL.LIB
    6     1K    1 R/W B:UTIL.SYM
BYTES REMAINING ON B: 39K

A>STAT B:*.*

 RECS  BYTES  EXT ACC
   53     7K    1 R/W B:CROSSREF.PKG
  234    30K    2 R/W B:DISASM80.COM
  254    32K    2 R/W B:JASM80.COM
  117    15K    1 R/W B:JASM80.OVL
  133    17K    2 R/W B:JASM81.OVL
  121    16K    1 R/W B:KALAHA.PKG
   45     6K    1 R/W B:MEMCHK.COM
    7     1K    1 R/W B:PRIME.PKG
   57     8K    1 R/W B:QSORT.PKG
   49     7K    1 R/W B:READ.ME
   89    12K    1 R/W B:READSYM.COM
  158    20K    2 R/W B:SYN1.OVL
  201    26K    2 R/W B:SYNTAX.COM
BYTES REMAINING ON B: 39K
```

Lista 8. Zbirke na prevajalniški, povezovalniški in storitveni disketi Janus Ade

Če imamo dovolj diskovnega prostora, je priporočljivo, da imamo na eni enoti prevajalnik, knjižnične simbolne zbirke (.SYM), knjižnične objektne zbirke (.JRL) in povezovalnik. Ta kombinacija omogoča hitro prevajanje in povezovanje programov. Tako niso potrebne zamenjave disket pri iskanju manjkajočih zbirk.

Če se ne uporablja opcija /S, mora prevajalnik najti knjižnične simbolne zbirke (.SYM). Te zbirke morajo tako biti na prevajalniškem ali na izvirnem disku. Podobno velja tudi za lastne knjižnice (prevedene iz paketnih specifikacij). Priporočljivo je imeti tudi sistemske knjižnične zbirke na prevajalniškem disku in lastne knjižnice na izvirnem disku.

Pri uporabi opcije /S se lahko knjižnične simbolne zbirke nahajajo na posebnem disku (npr. na povezovalnem disku). Takšen disk se po potrebi vstavi v prevajalniški diskovni pogon. Če pa so te zbirke že na prevajalniškem disku, zamenjava ni potrebna. Pri uporabi opcije /S mora biti na prevajalniškem disku vsaj 2k zlogov prostora. Ta prostor se uporablja za informacijsko zbirko, ki pove prevajalniku, kam naj pogleda za druge zbirke.

Če se opcija /S ne uporabi, mora biti na prevajalniškem disku še precej prostora (16k zlogov + dvakratni obseg izvirne zbirke). Ta prostor je potreben za vmesne in končne zbirke. Pri sistemih, ki imajo diske s 400k zlogi in več, je najbolje imeti prevajalniške zbirke, knjižnične simbolne zbirke, knjižnične objektne zbirke in povezovalnik na prevajalniškem disku. Tu opcija /S ni potrebna. Podobno lahko imamo na enem disku tudi vse izvirne zbirke.

Pri sistemih z 240k - 400k zlogov na disku je priporočljivo imeti poseben prevajalniški in poseben povezovalniški disk. Prevajalniški disk vsebuje še sistemske knjižnične simbolne zbirke. Povezovalniški disk ima še sistemske knjižnične objektne zbirke. Tu opcija /S ni potrebna, v kolikor so zadevne izvirne zbirke na enem samem disku. Pri manjših diskovnih enotah se uporabi opcija /S, pri zelo majhnih (pod 160k zlogi) pa se prevajalnik razdeli v dva ali v več delov; ustrezna navodila se pri tem pojavijo med prevajanjem.

6. Sklep

Čeprav Janus Ada ni popoln prevajalnik za standardni jezik Ada, je vendarle zelo zanimiv s stališča možnosti učenja tega jezika na malih (mikro) sistemih. Ta prevajalnik je tako moč zelo uspešno uporabljati na sistemu Partner (proizvod DO Iskra Delta); ta sistem uporablja disketo in vinčestrski disk z velikim obsegom pod operacijskim sistemom CP/M Plus; prevajanje in povezovanje sta tu zelo hitra postopka in tudi omejitev izvirnih zbirk praktično ni.

Jezik Ada bo prej ko slej postal aktualen na posebnih področjih programiranja, ima pa tudi posebne lastnosti, ki so še kako koristne. V teh dneh prihaja na tržišče tudi paket za aritmetiko s plavajočo vejico, ki bo omogočil uporabo Janus Ade tudi na klasičnih - poslovnih in tehniških področjih.

T A K O V F R A N C I J I !

T A K O V F R A N C I J I !

****
Nabor za računalniško generacijo
****

Francozi so pravi posebneži v podpiranju prodora računalniške tehnologije. Intelektualci in vladajoči državniki so vselej podpirali sodoben tehnološki razvoj s posebno skrbjo za ohranjanje in nastajanje francoske računalniške industrije, izobraževanja, terminologije.

Letos v januarju je francoski publicist J.J. Servan-Schreiber sprožil idejo, da naj bi srednješolci in študentje opravljali "nacinalno računalniško služenje" kot alternativo za 12-mesečno služenje vojaškega roka. Ta nenavadna ideja je postala zanimiva tudi za vladajočo garnituro.

V marcu letos je predsednik F. Mitterrand sprožil akcijo za uresničitev Servan-Schreiberjeve ideje. Namesto služenja vojaškega roka bodo francoski študentje politehnike in posebnih gimnazij lahko izbrali poučevanje nezaposlenih mladih o elementarni računalniški tehnologiji. In v okviru tega posebnega gibanja je Mitterrand takoj ukazal mobilizacijo 12000 diplomirancev iz 110 institucij, ki bodo predavali na računalniških tečajih. Mitterrandova motivacija za to akcijo izvira iz strahu, da ima Francija že znaten zaostanek v tekmi z japonsko informacijsko tehnologijo.

V tem trenutku je težko napovedati, kakšen bo učinek tega računalniškega nabora. Izobraževalni tečaji za nezaposlene med 15 in 25 letom starosti bodo trajali le mesec ali dva, kar pa bržkone ne bo zadostovalo za obvladanje sodobne računalniške tehnologije. In nazadnje je nabor vendarle samo nabor, zato je vprašanje, ali bodo francoski računalniški rekruti dovolj motivirani, da bi obvladali delo v tem programu. Vsekakor pa velja občudovati Mitterrandovo hitrost reagiranja, ko je sprejel izziv in ga pretvoril v akcijo. Francija vsebolj spoznava, da so za postavitev sodobne računalniške industrije potrebni posebni napori in posebna podpora francoske družbe.

Seveda se ob tem postavlja tudi vprašanje, kje smo pri nas v teh naporih, kje so pobude za bolj sodobno izobraževanje v srednjih in visokih šolah, kakšni so naši programi na tem področju, ali jih sploh imamo (tudi intelektualce in družbene aktiviste)? Ali ni naša izobraževalna dejavnost na področju računalništva preveč (in pretežno)

- nestrokovno pragmatična,
- površno rutinerska,
- močno utrujena (učitelje utrujajoča),
- malenkostno pridobitniška in
- pedagoško neangažirana?

A.P.Železnikar

K A K O P A P R I N A S ?

K A K O P A P R I N A S ?

JEDAN KVANTITATIVNI POSTUPAK ZA VREDNOVANJE ORGANIZACIJE MIKROPROCESORSKIH REGISTARA

UDK: 681.322.042

JOZO J. DUJMOVIĆ,
MILETA NOVAKOVIĆ

ELEKTROTEHNIČKI FAKULTET, BEOGRAD
INSTITUT „MIHAILO PUPIN", BEOGRAD

U radu se predlaže jedan novi kvantitativni postupak za vrednovanje i komparaciju organizacije mikroprocesorskih registara. Postupak se zasniva na kombinatornoj analizi sadržaja adresabilnih procesorskih registara. Za svaku kombinaciju adresnih registara, registara podataka i univerzalnih registara predloženi postupak omogućava izračunavanje jednog globalnog kvantitativnog indikatora registarske organizacije, koji se naziva broj ekvivalentnih univerzalnih registara. Na vrednost ovog indikatora utiče kako broj tako i efikasnost organizacije adresabilnih registara. Broj ekvivalentnih univerzalnih registara se zatim koristi u okviru odgovarajućeg elementarnog kriterijuma za izračunavanje elementarne preference adresabilnih registara. Izračunata elementarna preferenca se tada može agregirati sa elementarnim preferencama ostalih atributa mikroprocesorske arhitekture koristeći LSP metodu za vrednovanje složenih sistema. Predloženi postupak je u radu ilustrovan sa komparativnom analizom 13 popularnih 8-bitnih i 16-bitnih mikroprocesora.

A QUANTITATIVE TECHNIQUE FOR THE EVALUATION OF MICROPROCESSOR REGISTER ORGANIZATION. A quantitative technique for the evaluation and comparison of microprocessor register organization is proposed. The technique is based on the combinatorial analysis of the contents of addressable registers. For each combination of address registers, data registers, and universal registers the proposed technique enables the computation of an overall quantitative indicator of register organization called the number of equivalent universal registers. This indicator is affected both by the number of available registers, and by the efficiency of their organization. The number of equivalent universal registers is then used for organizing a suitable elementary criterion for computing the elementary preference of addressable registers. Furthermore, the computed elementary preference can be aggregated with elementary preferences of other microprocessor attributes according the LSP-method for system evaluation. The proposed technique is illustrated by a comparative analysis of 13 popular 8-bit and 16-bit microprocessors.

1. UVOD

Mnoge organizacije i pojedinci suočeni su sa problemom izbora najpovoljnije mikroprocesorske arhitekture za određenu klasu primena. Pri tome osnovni problemi obuhvataju (1) ocenu mikroprocesorskih registara, (2) ocenu načina adresiranja, (3) ocenu broja adresa po instrukciji, (4) ocenu skupa mašinskih instrukcija i (5) ocenu raznih hardverskih i softverskih osobina vezanih za sistem prekida, organizaciju magistrala i ulaza/izlaza, podršku operativnom sistemu i prevodiocima, i slično. Dosadašnji pokušaji vrednovanja svodili su se najčešće na neformalne postupke bazirane na sistematskim prikazima relevantnih atributa mikroprocesorske arhitekture [1,2,3,4] . Činjeni su i pokušaji da se problem komparacije mikroprocesora u izvesnoj meri formalizuje, ali su predlagane tehnike uglavnom ostale na nivou specijalizovanog grafičkog prikaza skupova instrukcija [5], aditivnih metoda za kodovanje [6], ili jednostavnih postupaka za eliminaciju pomoću skupa relacija koje opisuju obavezne osobine koje mikroprocesorska arhitektura mora da poseduje [7].

Mikroprocesorske arhitekture se mogu uspešno kvantitativno vrednovati pomoću opšte LSP metode za vrednovanje složenih sistema [8], pod uslovom da se razvije kompletan skup relevantnih atributa arhitekture, i da se za svaki atribut realizuje odgovarajući elementarni kriterijum. Prvi pokušaj u tom pravcu učinjen je u radu [9], ali je tu naglasak bio postavljen na vrednovanje načina adresiranja i skupova mašinskih instrukcija, dok metode za kvantitativno vrednovanje organizacije registara nije bilo.

U ovom radu ograničavamo se na vrednovanje adresabilnih mikroprocesorskih registara. U tom smislu ovaj rad predstavlja dopunu metodologije izložene u radu [9] i korak u pravcu realizacije celovite metodologije za vrednovanje mikroprocesorskih arhitektura. Pored toga, postupak vrednovanja koji sledi može se bez teškoća primeniti i za vrednovanje procesorskih

registara kod miniračunara i većih računara.

2. MATEMATIČKI MODEL ZA VREDNOVANJE MIKROPROCESORSKIH REGISTARA

2.1 Klasifikacija registara

Najvažniji adresabilni mikroprocesorski registri su registri podataka (Data Registers), adresni registri (Address Registers), brojač naredbi (Program Counter), ukazatelj(i) steka (Stack Pointer(s)) i registar stanja (Status Register). Svaki od njih ima specifičnu ulogu u asemblerskom programiranju. Funkcije brojača naredbi, ukazatelja steka i registra stanja su skoro iste za sve tradicionalne mikroprocesorske arhitekture. Sa druge strane, broj, organizacija i uloga registara podataka i adresnih registara može se od procesora do procesora znatno razlikovati, što utiče u velikoj meri na efikasnost asemblerskog programiranja. Dok mali broj ovih registara predstavlja ograničenje u programiranju i sprečava optimizaciju njihove upotrebe od strane viših jezika, dovoljan broj registara po pravilu olakšava realizaciju efikasnih (kratkih i brzih) programa. Ovo je naročito izraženo u slučajevima kada su raspoloživi univerzalni registri. Zbog toga je glavni problem u vrednovanju mikroprocesorskih registara napraviti pogodan analitički model za vrednovanje registara podataka i adresnih registara.

Za potrebe vrednovanja neophodna je klasifikacija adresabilnih mikroprocesorskih registara i ona je sa odgovarajućim terminima i skraćenicama prikazana na sl.1. Kod adresnih registara razlikujemo one koji sadrže apsolutnu adresu (ukazatelji podataka - Data Pointers) i relativnu adresu (indeks registri - Index Registers). Neki od njih imaju mogućnost autoinkrementiranja odnosno autodekrementiranja sadržaja zakonstantu (q) pre ili posle izvršenja instrukcije. Ukazatelj podataka koji sadrži fiksnu apsolutnu adresu naziva se bazni registar (Base Register), a ukoliko se adresa može automatski modifikovati u cilju odbrojavanja, onda takav registar nazivamo brojač podataka (Data Counter). Na sličan način se definiše fiksni indeks registar (Fixed Index) i indeksni brojač (Index Counter) (ovaj poslednji se za sada sreće samo kod većih procesora). U nastavku ovog rada razvijen je model za vrednovanje adresabilnih mikroprocesorskih registara baziran na analizi raspoloživih adresnih registara, registara podataka i univerzalnih registara koji mogu da obavljaju funkcije adresnih registara i registara podataka.

2.2 Ekvivalentni univerzalni registri

Neka je n_a broj specijalizovanih adresnih registara a n_d broj specijalizovanih registara podataka. Podrazumeva se da je broj bita u registru podataka jednak širini spoljne magistrale podataka. Veličina adresnog registra može biti jednaka širini apsolutne adrese (koja je jednaka širini adresne magistrale ili manja od nje), ili može biti i manja, ako predstavlja relativnu adresu (najčešće od 8 do 16 bita).

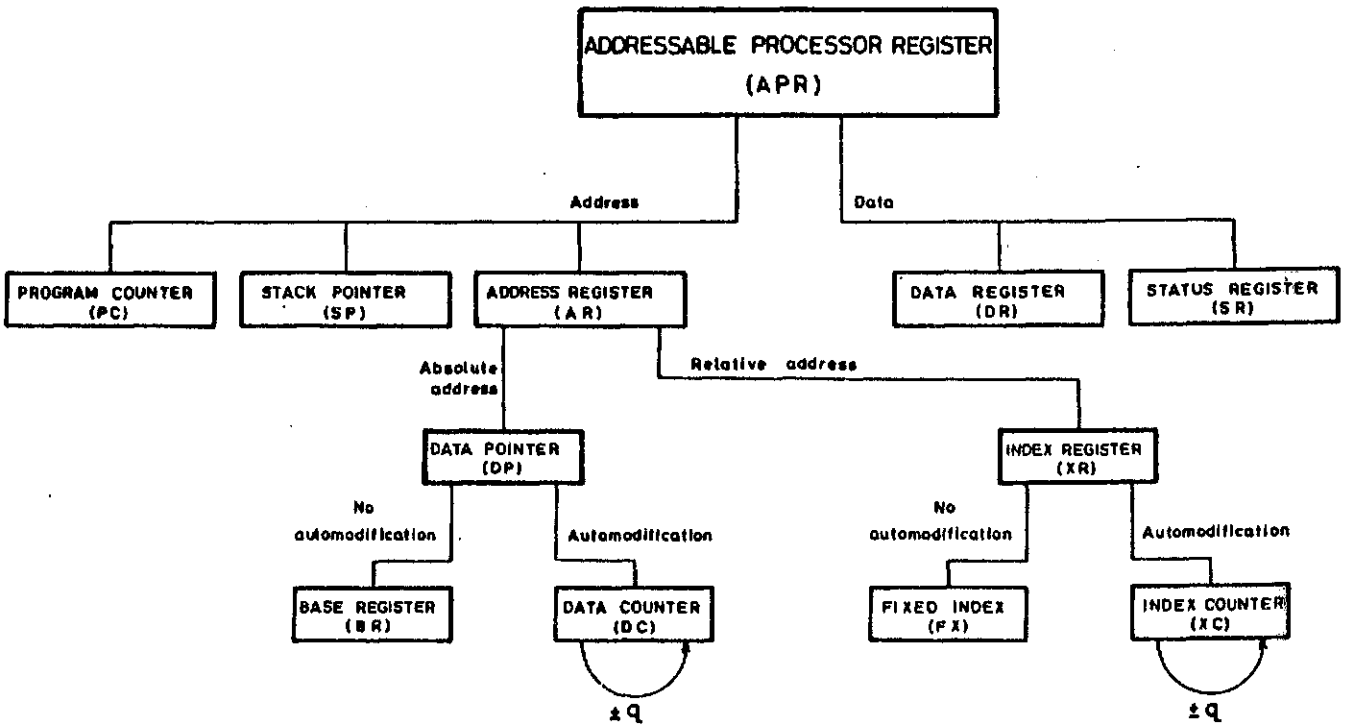
Pored specijalizovanih adresnih registara i registara podataka, ili umesto njih, neki mikroprocesori imaju univerzalne registre koji mogu sadržati adresu ili podatak. Za potrebe ovog rada smatraćemo da je registar univerzalan ako prema potrebi može obavljati funkcije ukazatelja podataka i funkcije indeks registra i ako u ulozi registra podataka može izvršiti sledeće operacije nad svojim sadržajem: (1) dvosmerni prenos podataka memorija-registar, (2) najvažnije aritmetičke, logičke i ulazno-izlazne operacije, kao i operacije pomeranja sadržaja i editovanja podataka, (3) operacije testiranja i poredjenja.

Pri vrednovanju mikroprocesorskih registara korisno je razlikovati dve grupe univerzalnih registara. Prvu grupu, univerzalne registre standardne dužine, čine registri koji mogu sadržati adrese ili podatke jednake veličine. Označimo broj ovih registara sa n_u. U drugoj grupi, koja je karakteristična za 8-bitne mikroprocesore, dužina podataka jednaka je polovini dužine adrese. Podrazumevaćemo da ovi registri mogu sadržati n_e podataka ili $n_e/2$ adresa.

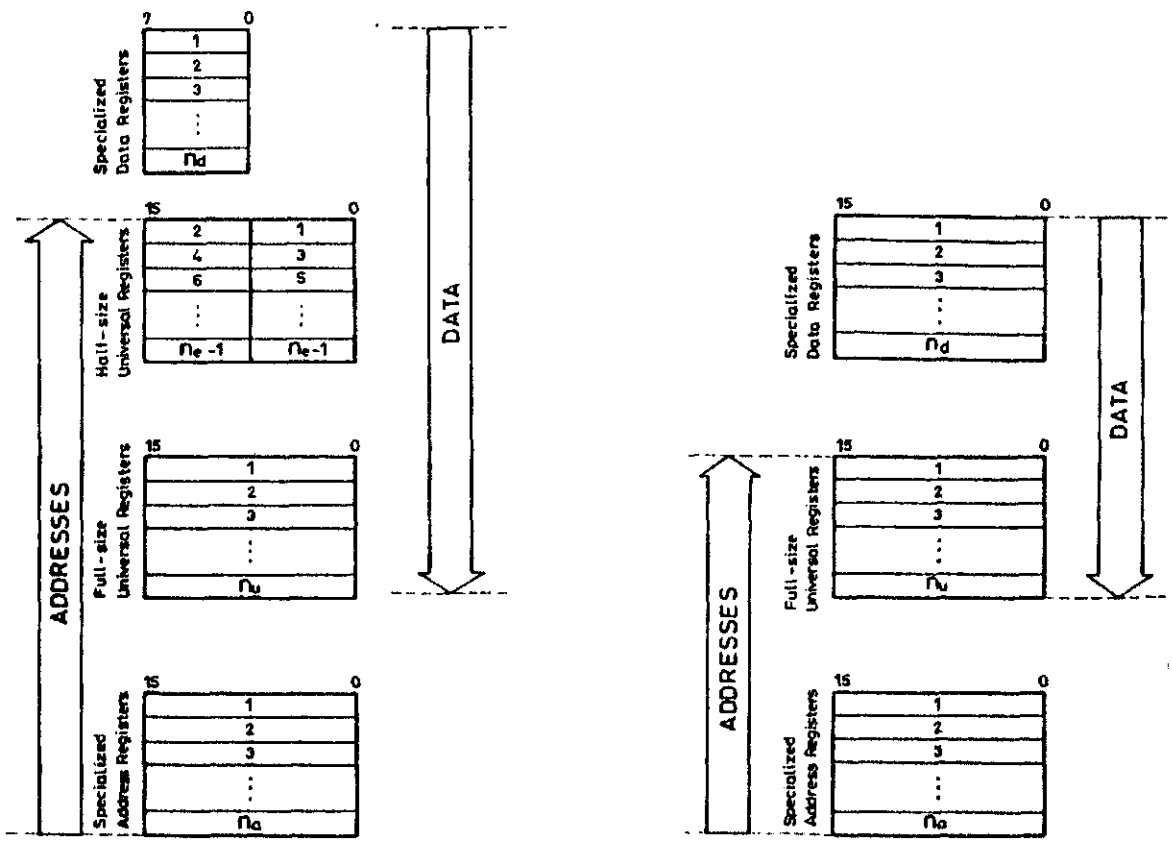
U prethodnom prikazu definisane veličine n_a, n_d, n_u i n_e predstavljaju četiri glavna ulazna elementa potrebna za vrednovanje adresabilnih registara. Sada je potrebno odrediti funkciju

$$E_{reg} = G_{reg}(n_a, n_d, n_u, n_e)$$

za izračunavanje rezultantne preference adresabilnih registara E_{reg} za date vrednosti n_a, n_d, n_u i n_e. Postupak vrednovanja se može izvršiti imajući u vidu da je osnovna funkcija adresabilnih registara da omoguće čuvanje što većeg broja raznih kombinacija adresa



Slika 1. Klasifikacija adresabilnih procesorskih registara



Slika 2. Redosled smeštanja adresa i podataka

(apsolutnih ili relativnih) i podataka.

Neka a i d označavaju broj adresa i broj podataka, respektivno, koji se mogu nalaziti u grupi adresabilnih registara. Izračunaćemo prvo broj mogućih (a,d)-kombinacija za sve tri glavne grupe:

(1) specijalizovane registre,
(2) univerzalne registre standardne (pune) dužine,
(3) univerzalne registre polovične dužine (u odnosu na standardnu dužinu).

Izračunavanje (a,d)-kombinacija se bazira na pretpostavci da su svi registri unutar bilo koje grupe ekvivalentni i da redosled uskladištenja adresa i podataka u tu grupu registara nije važan. Na primer, ako je broj univerzalnih registara $n_u=6$ i a=d=1, tada postoji 30 raznih parova adresa-podatak koji se mogu smestiti u 6 registara. Medjutim, kada se piše program, izbor odredjenog para nema uticaja na dužinu programa i/ili njegovu strukturu, pa je opravdano usvojiti da se u ovom slučaju radi o jednoj jedinoj (a,d)-kombinaciji (a,d)=(1,1). Na strukturu programa utiče jedino broj dostupnih registara pa su stoga i svi registri unutar grupe ekvivalentni.

Za grupu specijalizovanih registara broj mogućih (a,d)-kombinacija, C_s, je ograničen nejednačinama $a \le n_a$ i $d \le n_d$. Moguće kombinacije su $(0,0),\ldots,(0,n_d)$; $(1,0),\ldots,(1,n_d)$; $\ldots$; $(n_a,0),\ldots,(n_a,n_d)$. Odavde je

$$C_s = (n_a + 1)(n_d + 1)$$

Za grupu univerzalnih registara standardne dužine broj mogućih (a,d)-kombinacija, C_u, je ograničen nejednačinom $a + d \le n_u$. U ovom slučaju moguće kombinacije su $(0,0),\ldots,(0,n_u)$; $(1,0),\ldots,(1,n_u-1)$; $\ldots$; $(n_u-1,0),(n_u-1,1)$; $(n_u,0)$. Tako je

$$C_u = 1 + 2 + \ldots + (n_u + 1) = (n_u + 1)(n_u + 2)/2 \quad .$$

Za grupu univerzalnih registara polovične dužine broj mogućih (a,d)-kombinacija, C_e, je ograničen nejednačinom $2a + d \le n_e$. Kombinacije su $(0,0),\ldots,(0,n_e)$; $(1,0),\ldots,(1,n_e-2)$; $\ldots$; $(n_e/2,0)$, pa imamo da je

$$C_e = 1 + 3 + \ldots + (n_e + 1) = (n_e/2 + 1)^2 \quad .$$

Posmatrajmo sada opšti slučaj gde su sve grupe registara istovremeno dostupne. Imajući u vidu uobičajeni način korišćenja registara pri programiranju 8-bitnih mikroprocesora možemo prihvatiti sledeću pretpostavku: adrese se redom stavljaju u n_a adresnih registara, zatim (kada su svi specijalizovani adresni registri iscrpljeni) u n_u univerzalnih registara i na kraju u n_e registara polovične dužine. Podaci se prvo smeštaju u n_d registara podataka, potom u n_e i na kraju u n_u univerzalnih registara na način kako je to prikazano na sl.2. Ovakve pretpostavke ograničavaju ukupan broj mogućih (a,d)-kombinacija i odražavaju strategiju korišćenja registarskog prostora kod koje se pri uskladištenju podataka ostavlja maksimalan slobodan prostor za adrese i obrnuto, kod uskladištenja adresa ostavlja maksimalan slobodan prostor za podatke. Pored toga, ovakve pretpostavke olakšavaju postupak vrednovanja koji sledi.

Koristeći n_a specijalizovanih adresnih registara moguće je uskladištiti od 0 do n_a adresa. U isto vreme moguće je smestiti od 0 do $n_d+n_e+n_u$ podataka u preostale registre, dobijajući parcijalnu sumu od $(n_a+1)(n_d+n_e+n_u+1)$ različitih (a,d)-kombinacija. Posle ispunjavanja svih specijalizovanih adresnih registara, adresama se dalje puni n_u univerzalnih registara. Kada se jedna adresa smesti u jedan univerzalni registar, može se smestiti od 0 do $n_d+n_e+n_u-1$ podataka u preostale registre, što omogućuje $n_d+n_e+n_u$ (a,d)-kombinacija. Slično ovome, n_u adresa omogućuje smeštanje od 0 do n_d+n_e podataka, odnosno n_d+n_e+1 (a,d)-kombinacija. Parcijalna suma kombinacija je sada $(n_d+n_e+1)+\ldots+(n_d+n_e+n_u) = (2n_d+2n_e+n_u+1)n_u/2$. Na kraju, posle ispunjavanja n_a i n_d registara, ostatak adresa se može smestiti u n_e univerzalnih registara. Jedna adresa zauzima dva registra u ovoj grupi te se u preostale registre može smestiti od 0 do n_d+n_e-2 podataka. Slično tome, $n_e/2$ adresa može popuniti sve n_e registre i dopustiti smeštanje od 0 do n_d podataka. Parcijalna suma (a,d)-kombinacija je sada $(n_d+1)+(n_d+3)+\ldots+(n_d+n_e-1) = (n_d+n_e/2)n_e/2$. Zbir svih parcijalnih suma u opštem slučaju daje ukupan broj (a,d)-kombinacija:

$$C = (n_a+1)(n_d+n_e+n_u+1) + (2n_d+2n_e+n_u+1)n_u/2 + (n_d+n_e/2)n_e/2$$
$$= C_s + C_u + C_e + C_{su} + C_{se} + C_{ue} - 2 \qquad (1)$$

gde su

$$C_{su} = n_u(n_a+n_d), \quad C_{se} = n_e(n_a+n_d/2), \quad C_{ue} = n_u n_e.$$

Ako je $n_u=n_e=0$ tada je $C=C_s$, ako je $n_a=n_d=n_e=0$ onda $C=C_u$, i ako je $n_a=n_d=n_u=0$, onda je $C=C_e$. Ostali važni specijalni slučajevi formule (1) su:

$$\begin{aligned} C &= C_s + C_u + C_{su} - 1, \quad n_e = 0 \\ &= C_u + C_e + C_{ue} - 1, \quad n_a = n_d = 0 \\ &= C_s + C_e + C_{se} - 1, \quad n_u = 0 \quad . \end{aligned}$$

Prema tome, formula (1) je univerzalna, te se može primeniti na 8-bitne, 16-bitne mikroprocesore, kao i na procesore drugih, većih računara.

Skup registara se može smatrati, statistički posmatrano, ekvivalentnim skupu univerzalnih registara ako oba skupa omogućavaju jednak broj (a,d)-kombinacija. Skup registara koji sadrži n_a adresnih registara, n_d registara podataka, n_u univerzalnih registara standardne dužine i n_e univerzalnih registara polovične dužine smatramo da je statistički ekvivalentan skupu od N univerzalnih registara ako je

$$(N + 1)(N + 2)/2 = C \quad .$$

Rešenje gornje jednačine daje

$$N = [-3 + \sqrt{9 + 8(C - 1)}] / 2 \quad . \qquad (2)$$

N se može interpretirati kao broj univerzalnih registara koji su u statističkom smislu funkcionalno ekvivalentni postojećim specijalizovanim i univerzalnim registrima. Stoga ćemo N zvati broj ekvivalentnih univerzalnih registara. Ukupan broj (a,d)-kombinacija C i registara N odražavaju istovremeno broj dostupnih adresabilnih registara i fleksibilnost njihove upotrebe, pa se C i N mogu direktno primeniti u vrednovanju i poredjenju adresabilnih registara.

Osnovne osobine formula (1) i (2) su:

1. C i N su striktno rastuće funkcije n_a, n_d, n_u i n_e. Zbog toga je svako povećanje broja registara i povećanje C i N.

2. Ako je $n_a = n_d = n_e = 0$ onda je, kako se i očekuje, $N = n_u$. Za sve druge slučajeve N je jednako n_u plus inkrement koji odražava uticaj n_a, n_d i n_e.

3. Za svako $n_u \geq 0$ i svako $k \geq 0$ slučajevi $n_a = n_d = k$, $n_e = 0$ i $n_a = n_d = 0$, $n_e = 2k$ su ekvivalentni.

4. Formule (1) i (2) su nezavisne od "subjektivnih interpretacija" u sledećem smislu: ne može se postići povećanje C i N ako se univerzalni registri interpretiraju kao specijalizovani. Tada se, u stvari, C i N smanjuju.

Druge interesantne osobine formula (1) i (2) su prikazane u 30 primera tabele 1. Primeri 1-7 ilustruju osnovne uticaje odnosa broja univerzalnih i specijalizovanih registara. Na primerima 6, 8, 9 i 10 kada je $n_a+n_d=\text{const}$, vidi se da je slučaj $n_a=n_d$ bolji od slučajeva gde $n_a \neq n_d$. Zavisnost N od odnosa broja specijalizovanih registara i univerzalnih registara polovične dužine prikazan je u primerima 11-19. U primerima 11-15 podrazumeva se 10 univerzalnih registara polovične dužine, dok je u primerima 16-19 $n_a+n_d+n_e=10$ bez obzira na veličinu registara. Zahvaljujući prethodno navedenoj osobini broj 3 formula (1) i (2), u odnosu na C i N medjusobni odnos broja univerzalnih i specijalizovanih registara (primeri 20-23) je ekvivalentan medjusobnom odnosu broja univerzalnih registara standardne dužine i univerzalnih registara polovične dužine (primeri 24-27). Na kraju, primeri 5, 28, 29 i 30 prikazuju šta se dobija ako se svaka polovina univerzalnog registra može adresirati pod uslovom da je $n_a+n_e/2 = 10$.

2.3 Koeficijent univerzalnosti registara

Proces programiranja je najjednostavniji onda kada su svi registri univerzalni, tako da programer ne mora da vodi računa o tome koji registar bira za koju primenu već jedino o tome ima li slobodnih registara. Ovaj zaključak podržavaju formule (1) i (2) time što važi

$$N \leq N^* \quad , \quad N^* := n_a + n_d + n_u + n_e \quad .$$

Pri tome $N = N^*$ jedino u slučaju kada $n_a = n_d = n_e = 0$, tj, kada $N^* = n_u$. Odavde se uočava da ukupan broj raspoloživih registara N^* ima redukovanu funkcionalnost zbog specijalizacije pojedinih registara. Stoga veličina N predstavlja jednu kvantitativnu meru funkcionalnosti skupa registara, pa u nekim slučajevima može biti od interesa da se u proces vrednovanja uključi i sledeći koeficijent univerzalnosti registara

$$u_{reg} := 100\ N/N^* \quad [\%] \quad .$$

Tabela 1. Primeri izračunavanja ukupnog broja (a,d)-kombinacija i broja ekvivalentnih univerzalnih registara

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_a	0	1	0	0	0	5	0	4	3	1	0	1	1	2	2
n_d	0	1	0	0	0	5	0	6	7	9	2	0	2	0	2
n_u	2	0	0	0	10	0	0	0	0	0	0	0	0	0	0
n_e	0	0	2	4	0	0	10	0	0	0	8	8	6	6	4
C	6	4	4	9	66	36	36	35	32	20	35	34	33	30	29
N	2	1.4	1.4	2.8	10	7	7	6.9	6.5	4.8	6.9	6.8	6.6	6.3	6.1

No	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
n_a	4	3	2	1	1	2	3	4	0	0	0	0	0	0	0
n_d	4	3	2	1	1	2	3	4	0	0	0	0	0	0	0
n_u	0	0	0	0	8	6	4	2	8	6	4	2	9	4	0
n_e	2	4	6	8	0	0	0	0	2	4	6	8	2	12	20
C	40	42	42	40	64	60	54	46	64	60	54	46	76	111	121
N	7.5	7.7	7.7	7.5	9.8	9.5	8.9	8.1	9.8	9.5	8.9	8.1	10.8	13.4	14.1

Ovaj pokazatelj je prosto procenat univerzalnih registara i u principu organizacija registara je najbolja ukoliko $u_{reg} = 100\%$. Medjutim, $u_{reg} < 100\%$ još uvek ne znači neefikasnu organizaciju registara, jer ako je N dovoljno velik time je programiranje olakšano pa smanjenje vrednosti u_{reg} nema kritični značaj.

3. UTICAJ DUŽINE PODATAKA [illegible] LENTNIH UNIVERZALNIH R[illegible]

Vrednovanje i kompa[illegible] sabilnih mikroproces[illegible] zavisi od dužine po[illegible] mogu uskladištiti [illegible] likovati dve osn[illegible] sa 8-bitnim po[illegible] žine reči (1[illegible]

Rad sa [illegible] termina[illegible] uredja[illegible] podaci koj[illegible] bajtovi. U ovakvi[illegible] cima veće dužine sreće [illegible] ukoliko postoje registri i ins[illegible] sa dužim podacima onda to ne predst[illegible] o suštinsko preimućstvo posmatranog mikroprocesora.

Rad sa rečima veće dužine javlja se uvek kada mikroprocesor podržava neki od viših programskih jezika u okviru malih računara i srodnih sistema gde je korisno omogućiti direktnu podršku radu sa celobrojnim podacima dužine barem [illegible]a. U ovom slučaju se podrazumeva da [illegible]ašinski jezik obezbedjuje aritmeće operacije, prenos, pomeranje, i komparaciju podataka u dvobajtnim [illegible] memorijskim ćelijama i procesorskim [illegible]ima.

[illegible]majući u vidu različite potrebe u bajtovim primenama i primenama vezanim za duže reči [illegible]orisno je uvesti klasifikaciju procesora pomoću šifre B/D, gde B označava širinu magistrale podataka i memorijske reči, a D (ukoliko postoji) označava da postoje procesorski registri i mašinske instrukcije za prihvatanje i obradu podataka dužine D bita. Primeri ove klasifikacije u domenu mikroprocesora i miniračunara su sledeći.

TIP 8. Procesor sa 8-bitnom magistralom podataka predvidjen isključivo za 8-bitne primene. Nema mogućnosti

neposrednog dohvatanja, skladištenja u registre i obrade 16-bitnih ili dužih podataka. Primeri: 6502, 6800, 6809, Z80, 8080, 1802 itd.

TIP 8/16. Procesor sa 8-bitnom magistralom podataka predvidjen primarno za 8-bitne primene i opremljen instrukcijama dohvatanja, skladištenja u registre i obrade 16-bitnih podataka. Pri tome se programi za obradu 16-bitnih podataka ne razlikuju od programa za ekvivalentnu obradu 8-bitnih podataka; sekvencijalno dohvatanje 16-bitnih podataka usporava obradu ali je nevidljivo za programera. Primer: 8088.

TIP 16. Procesor sa 16-bitnom magistralom podataka predvidjen isključivo za 16-bitne primene. Specijalizovane instrukcije za neposredno dohvatanje, skladištenje i obradu 8-bitnih podataka ne postoje. Memorija nije bajtovski adresabilna, procesor nema 8-bitnih registara, a sve 8-bitne operacije moraju se simulirati pomoću odgovarajućih 16-bitnih operacija, što otežava programiranje i usporava obradu. Primer: IBM 1130.

TIP 16/8. Procesor sa 16-bitnom magistralom podataka predvidjen za 16-bitne i 8-bitne primene. Postoji većina specijalizovanih instrukcija za prenos iz memorije, skladištenje u procesorskim registrima, i obradu 8-bitnih podataka. Memorija je bajtovski adresabilna, pri čemu neke instrukcije dohvataju i obradjuju dva bajta, a neke jedan bajt. Primeri: 8086, 68000, Z8000, LSI-11.

Navedena klasifikacija ukazuje da vrednovanje procesorskih registara ima smisla jedino u okviru jednog tipa primena, t.j. posebno u domenu 8-bitnih primena i posebno u domenu 16-bitnih primena. Stoga jedan te isti mikroprocesor može biti različito vrednovan u ova dva domena primene. Oni 16-bitni procesori koji imaju bajt-adresabilne registre imaju u 8-bitnim primenama veći broj registara nego u 16-bitnim primenama i stoga u 8-bitnim primenama mogu dobiti bolju ocenu. Obično svi registri nisu bajt-adresabilni već samo neki od njih (uporediti npr. 8086 i Z8000). Sa druge strane, neki 16-bitni procesori nemaju bajt-adresabilne registre već jednobajtne podatke uskladištavaju u 16-bitne ili duže registre (npr. LSI-11 i 68000) na isti način kao i 16-bitne podatke. Stoga u tim slučajevima broj registara i odgovarajuća ocena ostaju isti u 8-bitnim i 16-bitnim primenama.

U slučajevima kada se vrednuje organizacija registara za procesore od kojih se očekuje da uspešno rade kako u 8-bitnom tako i u 16-bitnom režimu (ili režimu reči veće dužine) onda se za svaki procesor izračunavaju dva broja ekvivalentnih univerzalnih registara: broj ekvivalentnih univerzalnih 8-bitnih registara N_8 i broj ekvivalentnih univerzalnih 16-bitnih (ili dužih) registara N_{16}. Ako je očekivana relativna frekvencija za 8-bitne primene W_8 a za 16-bitne primene W_{16} (pri čemu $W_8 \geq 0$, $W_{16} \geq 0$, $W_8+W_{16}=1$), onda se rezultantni broj ekvivalentnih univerzalnih registara N_{reg} može računati po nekoj od težinskih stepenih sredina. Najčešće,može se primeniti aritmetička sredina

$$N_{reg} := W_8 N_8 + W_{16} N_{16}$$

ili (u slučaju strožijih uslova) geometrijska sredina

$$N_{reg} := N_8^{W_8} \cdot N_{16}^{W_{16}} .$$

Naravno, ako su u pitanju samo 8-bitne primene onda je $W_8 = 1 - W_{16} = 1$, pa sledi $N_{reg} = N_8$. Na sličan način, kod isključivih 16-bitnih primena $W_{16} = 1 - W_8 = 1$ pa je $N_{reg} = N_{16}$.

4. KRITERIJUM ZA VREDNOVANJE ORGANIZACIJE REGISTARA

Broj ekvivalentnih univerzalnih registara N_{reg} se može direktno koristiti za poredjenje organizacija mikroprocesorskih registara. Medjutim, kada se vrednovanje adresabilnih registara želi spojiti sa procesom vrednovanja drugih komponenata mikroprocesora, tada se mora izračunati odgovarajuća preferenca $E_{reg} = g_{reg}(N_{reg})$, $0 \leq E_{reg} \leq 100\%$, koja se interpretira kao procenat zadovoljenja korisnikovih zahteva. Funkcija g_{reg} mora uvek biti odraz potreba odredjene primene. Kao ilustracija, ovde se koristi jednostavna funkcija sa jednim parametrom

$$E_{reg} = 100 [1 - \exp(-N_{reg}/N_0)] \quad (3)$$

gde se parametar N_0 može lako odrediti iz bilo kog (N_{reg}, E_{reg}) para. Na primer, ako se smatra da je dovoljan broj ekvivalentnih univerzalnih registara $N_{reg} = 6$ za $E_{reg} = 75\%$ mogućih primena, tada je $N_0 = 4.328$. Ova vrednost je

primenjena za izračunavanje preferenci organizacije mikroprocesorskih registara za 13 popularnih 8-bitnih i 16-bitnih mikroprocesora prikazanih u odeljku koji sledi. Naravno, prethodna formula se bez teškoća može modifikovati ili dopuniti u saglasnosti sa konkretnim potrebama odredjenog korisnika. Pored kontinualnih funkcija, često je pogodno da se $g_{reg}(N_{reg})$ izrazi u vidu niza pravolinijskih segmenata, pri čemu je vrednovatelj dužan da uz obrazloženje definiše skup koordinata prelomnih tačaka.

5. REZULTATI VREDNOVANJA ORGANIZACIJE MIKROPROCESORSKIH REGISTARA

Radi ilustracije organizacije mikroprocesorskih registara i odredjivanja veličina n_a, n_d, n_u i n_e na sl.3 su prikazani registri tri popularna 8-bitna mikroprocesora (8088, 6800 i Z80) i tri 16-bitna mikroprocesora (8086, 68000 i LSI-11/23). Notacija i analiza registara na sl.3 podrazumeva 8-bitne primene za 8088, 6800 i Z80, i 16-bitne primene za 8086, 68000 i LSI-11/23. Pri tome se mogućnosti registra da prihvata razne podatke ustanovljava detaljnom analizom skupa mašinskih instrukcija, a uloga registara u adresiranju odredjuje se analizom skupa načina adresiranja. Rezultate prikazane na sl.3 u nastavku ćemo detaljno prodiskutovati na primeru komparativne analize 8088 i 8086.

Mikroprocesor Intel 8088 po svojoj unutrašnjoj strukturi i skupu mašinskih instrukcija predstavlja 16-bitni mikroprocesor. Medjutim, zbog svoje 8-bitne spoljne magistrale podataka i softverskih mogućnosti rada u 8-bitnim primenama opravdano je da se 8088 analizira u grupi 8-bitnih mikroprocesora, a njegov 16-bitni ekvivalent, 8086, u grupi 16-bitnih mikroprocesora. Prema ranijoj klasifikaciji 8088 je procesor tipa 8/16, a 8086 je tipa 16/8. Interesantno je uporediti organizaciju registara za 8086 i 8088 uočavajući kako se jedan isti skup registara može koristiti na dva različita načina. Registri AX i CX kod 8086 sadrže isključivo podatke dok su BX i DX, isto kao i BP, SI i DI univerzalni registri (u smislu kako je to definisano u tački 2.2). Odatle je za 8086 $n_d = 2$ i $n_u = 5$ pa prema formulama (1) i (2) sledi

$$C_1 = n_d + n_u + 1 + (2n_d + n_u + 1)n_u/2 = 33$$

$$N_{16} = (-3 + \sqrt{9 + 8 \cdot 32}\,)/2 = 6.64 \quad .$$

Registri AX, CX, BX i DX su bajt-adresabilni, tako da se u 8-bitnoj varijanti kod 8088 pojavljuju četiri registra podataka (AH, AL, CH i CL), tj. $n_d=4$. Registri BX i DX mogu i sada da sadrže 16-bitne adrese, a podaci koji se u njih mogu uskladištiti su 8-bitni, tako da se radi o univerzalnim registrima polovične dužine (BH, BL, DH i DL), pa imamo $n_e=4$. Konačno, BP, SI i DI nisu bajtovski adresabilni pa predstavljaju univerzalne registre pune dužine, tj. $n_u=3$. Koristeći formule (1) i (2) sada se dobija

$$C = n_d+n_u+n_e+1 + (2n_d+2n_e+n_u+1)n_u/2 + (n_d+n_e/2)n_e/2 = 54$$

$$N_8 = (-3 + \sqrt{9 + 8 \cdot 53}\,)/2 = 8.90 \quad .$$

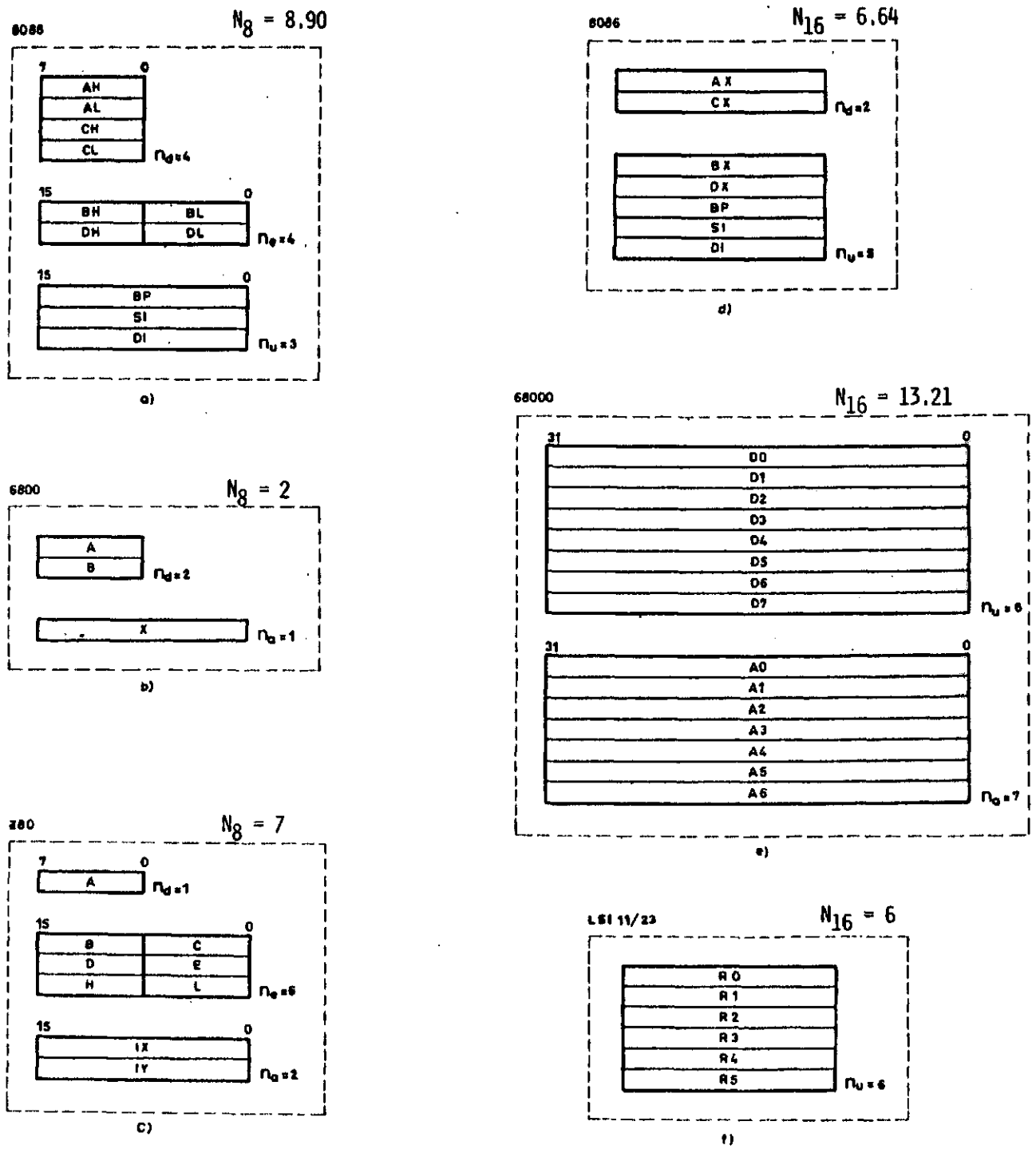
Kao što se i očekivalo $N_8 > N_{16}$. Ako predpostavimo $W_8=W_{16}=0.5$ i primenimo aritmetičko usrednjavanje sledi

$$N_{reg} = (N_8 + N_{16})/2 = 7.77 \quad .$$

Budući da su u ovom slučaju N_8 i N_{16} bliske vrednosti odabrani tip usrednjavanja ne utiče bitno na rezultat (npr. u slučaju geometrijskog usrednjavanja $N_{reg}= 7.69$).

Ukoliko registri 16-bitnog mikroprocesora nisu bajt-adresabilni, a funkcija registara je ista za 8-bitne i 16-bitne primene, onda veličine n_a, n_d, n_u i n_e ostaju neizmenjene, pa je $N_8 = N_{16}$. Na taj način se npr. za LSI-11/23 dobija $N_{reg} = N_8 = N_{16} = n_u = 6$.

Kod 6800 postoje samo dva akumulatora i jedan bazni registar (proizvodjač ga naziva indeks registrom), što je ekvivalentno sa dva univerzalna registra. Ovaj primer ilustruje da se sve procesorske funkcije mogu realizovati i sa veoma skromnim brojem registara, što je bio čest slučaj i kod miniračunara starijeg datuma. Naravno, u ovakvoj situaciji programer nailazi na niz ograničenja koja otežavaju njegov rad i umanjuju rezultantnu efikasnost programiranja. Kod savremenije koncipiranih mikroprocesora, kao što je Z80, broj registara je znatno povećan, a njihova funkcionalnost je dodatno poboljšana uvodjenjem univerzalnih registara. Prema sl.3 Z80 ima ukupno 9 registara koji su ekvivalentni sa 7 univerzalnih registara, u kategoriji 8-bitnih primena ovo predstavlja jedan od veoma dobrih rezultata. Konačno, 68000 je mikroprocesor tipa 16/8/32 kod koga je proizvodjač predvideo 7 adresnih registara i 8 registara podataka.



Slika 3. Primeri odredjivanja veličina n_a, n_d, n_u, n_e, N_8 i N_{16} za neke 8-bitne i 16-bitne mikroprocesore

Kako registri podataka mogu prilikom indeksiranja obavljati i ulogu adresnih registara to je u slučajevima gde je ovo od interesa moguće interpretirati D0,...,D7 kao univerzalne registre. To je i učinjeno na sl. 3 tako da je dobijeno 13.21 ekvivalentnih univerzalnih registara (u protivnome, za n_a=7 i n_d=8 sledelo bi N_{16}=10.51).

Detaljni prikaz postupka vrednovanja adresabilnih registara za 13 popularnih 8-bitnih i 16-bitnih mikroprocesora izložen je u tabeli 2. Za izračunavanje E_{reg} korišćena je formula (3) sa parametrom N_o=4.328. Pored podataka neophodnih za izračunavanje E_{reg} u tabeli 2 su dati i drugi važni podaci koji karakterišu prikazane

mikroprocesore i koji se mogu primeniti ako se proces vrednovanja proširi i na ove karakteristike. U drugom delu tabele 2 u kolonama pod nazivom FUNCTIONAL ORGANIZATION OF REGISTERS nalaze se podaci o adresabilnim registrima mikroprocesora. U kolonama NUMBER OF EQUIVALENT UNIVERSAL REGISTERS i PREFERENCE RATING OF ADDRESSABLE REGISTERS nalaze se finalni rezultati vrednovanja. Mikroprocesori su u tabeli poredjani po opadajućim vrednostima preference adresabilnih registara u okviru svake od grupa.

U koloni DATA BUS WIDTH data je širina spoljne magistrale podataka. U koloni ADDRESS SPACE je data veličina direktno adresabilnog memorijskog prostora bez upotrebe pomoćnog hardvera i softvera za njegovo proširenje. Veličine operanada sa kojima mogu mikroprocesori raditi date su u koloni AVAILABLE SIZES OF OPERANDS. Naznačene su one veličine operanada koje su veličine bajta ili veće. Operandi veličine 1 i 4 bita nisu uzimani u obzir. Pored toga nisu uzimane u obzir ni dvostruke veličine operanada koje su rezultat operacije množenja. Podaci o prekidima su dati u kolonama VECTORED INTERRUPTS i NUMBER OF PROGRAM PRIORITY LEVELS. U koloni NUMBER OF PROGRAM PRIORITY LEVELS naveden je ukupan broj nivoa prioriteta na kojima se mogu izvršavati mašinske instrukcije ne uzimajući u obzir kako se na te nivoe dolazi: hardverskim ili softverskim prekidom, ili direktnim postavljanjem nivoa prioriteta iz nekog od programa.

U kolonama FUNCTIONAL ORGANIZATION OF REGISTERS dat je prikaz najvećeg mogućeg broja registara sa svakom od navedenih funkcija. Na primer, Z8002 i 68000 ne mogu istovremeno imati oba ukazatelja steka, već im je u jednom trenutku dostupan samo jedan, sistemski ili korisnički stek. I pored toga u opštem slučaju se oba steka mogu bez ograničenja alternativno koristiti, pa su u ovom slučaju u koloni STACK POINTERS naznačena dva steka. Kako u mikroprocesoru mora postojati brojač naredbi, a u slučaju mikroprocesora 1802 COSMAC bilo koji od adresnih registara može biti korišćen kao brojač naredbi, to je u ovom slučaju maksimalan broj adresnih registara morao biti smanjen za jedan. Kolona NUMBER OF STATUS FLAGS ne daje samo broj indikatora unutar statusne reči mikroprocesora, već ukupan broj unutrašnjih indikatora dostupnih programeru.

Podela registara u delu PHYSICAL ORGANIZATION OF REGISTERS je izvršena shodno definicijama n_a n_d, n_u i n_e registara na način kako je to ilustrovano slikom 3. Za razliku od funkcionalne organizacije registara ovde su date vrednosti koje odgovaraju broju registara u upotrebi prilikom programiranja. One su manje ili jednake maksimalnim vrednostima i odgovaraju registarskoj arhitekturi koju vidi programer u jednom trenutku. U nastavku slede objašnjenja kako su odredjene vrednosti n_a, n_d, n_u i n_e tamo gde su takva objašnjenja neophodna.

Zilog Z80. Ovaj mikroprocesor ima dva skupa registara, primarni i sekundarni, koji se koriste alternativno. U kolonama maksimalnog broja registara oni su uzeti u obzir, dok u kolonama fizičke organizacije registara nisu. Razlog je što programer u jednom momentu koristi samo jedan od dva skupa dok drugi koristi za opsluživanje prekidnih rutina, privremeno čuvanje sadržaja registara, itd.

RCA CDP1802 COSMAC. Karakteristika ovog mikroprocesora je izrazito neuravnotežena organizacija adresabilnih registara. Šesnaest 16-bitnih registara se mogu koristiti samo kao adresni. Pored njih postoji jedan registar podataka - 8-bitni akumulator. Svaki od 16 adresnih registara može imati ulogu brojača naredbi. Ako se od preostalih 15 jedan koristi kao ukazatelj steka, a dva za opsluživanje prekidne rutine, ostaje korisnih 12 registara za programera. Ta vrednost je i navedena u koloni n_a.

Motorola MC6809. U koloni n_a ne uzimaju se u obzir ukazatelji steka. Medjutim, dva postojeća ukazatelja steka mogu se istovremeno upotrebiti kao adresni registri. Da bi se ovo uzelo u obzir vrednosti navedene u kolonama DATA POINTERS i INDEX REGISTERS su za dva veće od vrednosti u koloni n_a.

Zilog Z8002. Od 16 univerzalnih registara samo registar R0 ne može sadržati relativnu adresu prilikom indeksiranja te je u koloni FIXED INDICES broj registara umanjen za jedan. Iz istog razloga kao i u slučaju 6809 u kolonama DATA POINTERS i INDEX REGISTERS dodat je ukazatelj steka, što nije učinjeno u n_u koloni. U koloni STACK POINTERS navedeni su sistemski i korisnički ukazatelj steka i pored toga što se ne mogu simultano koristiti.

Motorola MC68000. Važi isti komentar kao u slučaju Z8002.

Texas Instruments TMS9900. Programirajući ovaj mikroprocesor programer može da koristi 16 univerzalnih registara koji predstavljaju deo memo-

Tabela 2. Vrednovanje organizacije registara za 13 8-bitnih i 16-bitnih mikroprocesora

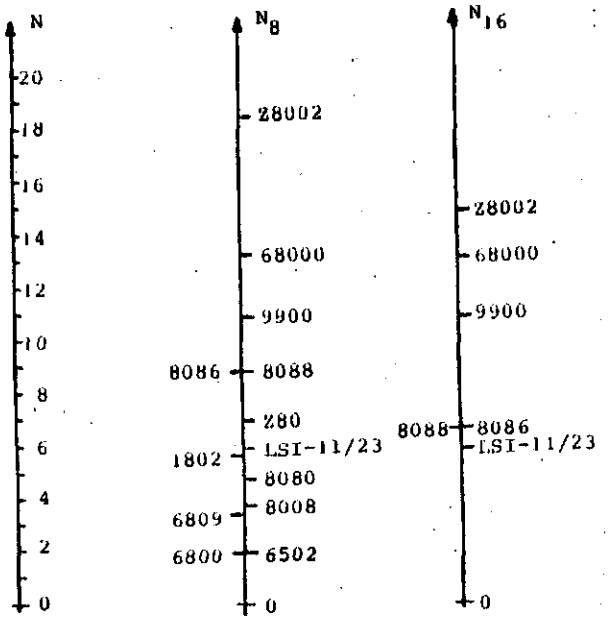
PROCESSOR	GENERAL CHARACTERISTICS					FUNCTIONAL ORGANIZATION OF REGISTERS (MAXIMUM NUMBER OF REGISTERS WHICH CAN OPERATE IN EACH GROUP)								PHYSICAL ORGANIZATION OF REGISTERS						
						ADDRESS REGISTERS						DATA AND STATUS REGISTERS								
								DATA POINTERS		INDEX REGISTERS										
	DATA BUS WIDTH [bits]	ADDRESS SPACE [kilobytes]	AVAILABLE SIZES OF OPERANDS [bytes]	VECTORED INTERRUPTS	NUMBER OF PROGRAM PRIORITY LEVELS	PROGRAM COUNTERS	STACK POINTERS	BASE REGISTERS	DATA COUNTERS	FIXED INDICES	INDEX COUNTERS	DATA REGISTERS	NUMBER OF STATUS FLAGS	NUMBER OF SPECIALIZED ADDRESS REGISTERS (n_a)	NUMBER OF SPECIALIZED DATA REGISTERS (n_d)	NUMBER OF FULL-SIZE UNIVERSAL REGISTERS (n_u)	NUMBER OF HALF-SIZE UNIVERSAL REGISTERS (n_e)	NUMBER OF EQUIVALENT UNIVERSAL REGISTERS (N_{reg})	PREFERENCE RATING OF ADDRESSABLE REGISTERS (E_{reg})	
8088	8	1 MB	1;2	Yes	4	1	1	5	2	4	0	11	9	0	4	3	4	8.90	87.22	$N_{reg} = N_8$
Z80	8	64	1;2	Yes	3	1	1	8	0	0	0	14	6	2	1	0	6	7.00	80.15	
1802	8	64	1	No	2	16	15	15	15	0	0	1	7	12	1	0	0	5.72	73.38	
8080	8	64	1;2	Yes	2	1	1	3	0	0	0	7	5	0	1	0	6	4.84	67.35	
8008	8	16	1	Yes	2	1	1	1	0	0	0	7	4	0	5	0	2	3.81	58.58	
6809	8	64	1;2	No	4	1	2	5	5	5	0	2	9	3	2	0	0	3.42	54.67	
6800	8	64	1;2	No	3	1	1	1	0	0	0	2	6	1	2	0	0	2.00	37.00	
6502	8	64	1	No	3	1	1	0	0	2	0	1	7	2	1	0	0	2.00	37.00	
Z8002	16	64	1;2;4	Yes	5	1	2	16	16	15	0	16	10	7	0	0	16	18.51	98.61	
8086	16	1MB	1;2	Yes	4	1	1	5	2	4	0	11	9	0	4	3	4	8.90	87.22	
68000	16	16MB	1;2;4	Yes	8	1	2	8	8	16	0	8	7	7	0	8	0	13.21	95.27	$N_{reg} = N_8 = N_{16}$
9900	16	64	1;2	Yes	16	1	0	16	16	15	0	16	7	0	0	11	0	11.00	92.13	
LSI-11/23	16	64	1;2	Yes	8	1	2	7	7	7	0	6	5	0	0	6	0	6.00	75.00	
Z8002	16	64	1;2;4	Yes	5	1	2	16	16	15	0	16	10	0	0	15	0	15.00	96.87	$N_{reg} = N_{16}$
8086	16	1MB	1;2	Yes	4	1	1	5	2	4	0	7	9	0	2	5	0	6.64	78.43	
8088	8	1MB	1;2	Yes	4	1	1	5	2	4	0	7	9	0	2	5	0	6.64	78.43	

rije koji ima ulogu registara. Iako je u principu maksimalan broj skupova ovih registara ograničen isključivo veličinom raspoložive memorije, u kolonama funkcionalne organizacije registara dat je samo jedan skup. Skok na podprogram obavlja uz pomoć prebacivanja na sledeći skup univerzalnih registara (Context Switch princip). Budući da postoji autoinkrementno i autodekrementno adresiranje to 9900 može da radi i bez posebnog registra ukazatelja steka. Registar R0 ne može učestvovati u indeksiranom adresiranju, te je u toj koloni ukupan broj registara za jedan manji. Prilikom programiranja neki od 16 registara imaju posebne funkcije, pa je stvarni raspoloživi broj univerzalnih registara 11.

U delu tabele 2 koji se odnosi na 8-bitne registre prikazane su vrednosti za N_8, a u delu tabele koji se odnosi na 16-bitne registre prikazane su veličine N_{16}. Međutim, svi navedeni 16-bitni mikroprocesori mogu se uspešno koristiti i u 8-bitnim primenama. U slučaju 68000, 9900 i LSI-11/23 u 8-bitnim primenama koristi se samo jedna polovina odgovarajućih 16-bitnih registara. Na taj način n_a, n_d, i n_u ostaju neizmenjeni u 8-bitnim i 16-bitnim primenama, pa $N_{reg} = N_8 = N_{16}$. U slučaju Z8002 u 16-bitnim primenama $N_{16}=n_u=15$, dok u 8-bitnim primenama 7 16-bitnih registara se koriste za adrese, a 8 16-bitnih registara može se koristiti za adrese ili za 16 8-bitnih podataka. Odatle sledi $n_a=7$ i $n_e=16$, pa je C=200, $N_8=18.51$ i za 8-bitne primene dobijamo $E_{reg}=98.61\%$ što je posebno navedeno u tabeli 2. Što se tiče 8086, saglasno analizi prikazanoj na slici 3, u 8-bitnim primenama on se ponaša kao 8088 i za njega važi $N_{16}=6.64$ i $N_8=8.90$. Prema tome, u tabeli 2 mikroprocesori su grupisani zavisno od toga da li je $N_8 \neq N_{16}$ ili $N_8=N_{16}$, tako da se mogu obrazovati rang liste posebno za 8-bitne primene i posebno za 16-bitne primene. Ove rang liste grafički ilustruje slika 4.

6. ZAKLJUČAK

U ovom radu prikazan je jedan kombinatorni postupak za vrednovanje organizacije adresabilnih procesorskih registara. Prikazani su primeri vrednovanja 8-bitnih i 16-bitnih mikroprocesora, ali se predložena metoda može bez ikakvih izmena primeniti i za vrednovanje organizacije registara kod miniračunara i velikih računara. Vrednovanje organizacije registara je važan element u vrednovanju procesorskih arhitektura. Pri tome se rezultati vrednovanja organizacije registara mogu agregirati sa rezultatima vrednovanja broja adresa po instrukciji, načina adresiranja, skupova mašinskih instrukcija i drugih elemenata za procenu. Postupak agregacije može se širiti i dalje prerastajući u kompletne kvantitativne modele za vrednovanje računarskog hardvera i računara kao celine.



Slika 4. Rangiranje mikroprocesora prema broju ekvivalentnih univerzalnih registara.

7. REFERENCE

[1] Theis, D.J., Microprocessor and microcomputer survey. Datamation, December 1974, pp.90-101.

[2] Osborne, A. and Kane, G., Osborne 4 & 8-bit microprocessor handbook. Osborne 1981.

[3] Osborne, A. and Kane, G., Osborne 16-bit microprocessor handbook. Osborne/McGraw-Hill 1981.

[4] Mimner, R. and Deluigi, B., 16-Bit-Prozessoren im Vergleich. Elektronik, No.5,6, and 7 (1981).

[5] Barton, M. and Dagless, E., Graphical approach to microprocessor comparison. Microprocessors V.1,No.6,1977.

[6] Microcomputer Technique, Inc., Microprocessors Scorecard. New Logic Handbook Vol.1, No.1, Sept. 1974.

[7] Cole, T.A., A collected guide to microsystem selection. Rome Air Development Center, Report No. RADC-TR-77-28, January 1977.

[8] Dujmović, J.J., and Elnicki, R., A DMS cost/benefit decision model: mathematical models for data management system evaluation, comparison and selection. National Bureau of Standards Publication No. NBS-GCR-82-374, 1981.

[9] Dujmović, J.J., Evaluation and comparison of 8-bit and 16-bit microprocessors. Seminar held at the Informatica'82 Congress.

INFORMATICA 1/1983

MODELIRANJE RADA JEDINICA MEMORIJE SA DIREKTNIM DOSTUPOM (DASD)

UDK: 681.3:519.876.5

PETAR KNEŽEVIĆ,
MIROSLAVA LAPAINE

NIKOLA TESLA, ZAGREB

Razmatran je model ulazno/izlaznog podsistema jedinica sa direktnim dostupom. U modelu se primjenom metode simulacije promatra promjena srednjeg vremena posluživanja ulazno/izlaznih zahtjeva variranjem veličina koje na to utječu. Model dozvoljava istovremeno nezavisno variranje karakteristika svake pojedine jedinice i promatranje efekta na zauzeće kanala i vrijeme odgovora drugih jedinica.

MODELLING OF DIRECT ACCESS STORAGE DEVICES (DASD) ACTIVITY. A model of the input/output subsystem of direct access storage devices was considered. Simulation methods were applied to observe the changes in the mean servicing time of I/O demands caused by variations in the quantities which influence it. The model allows the simultaneous and independent varying of the characteristics of each particular unit and the observation of its effects on channel activity and the response time of other units.

UVOD

Tokom zadnjih godina došlo je do nagle preorjentacije načina rada na kompjuterskim sistemima. Korisnici kompjuterskih sistema se sve više orjentiraju na interaktivni rad te se kao jedno od osnovnih mjerila efikasnosti rada kompjuterskog sistema, promatra vrijeme odgovora sistema na zahtjeve za obradom inicirane sa terminala.

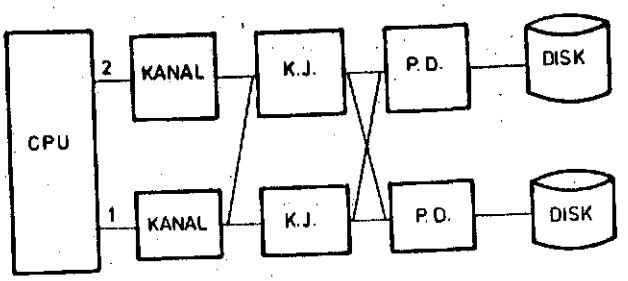
Najveći djelovi operativnog sistema, programi i podaci korisnika se nalaze na jedinicama sa direktnim dostupom i istraživanje puta transakcije ili komande inicirane od terminala kroz sistem i natrag na terminal pokazalo je da je kritičan dio puta ulazno/izlazni podsistem jedinica sa direktnim dostupom na kojima se nalaze podaci i programi.

Detaljna analiza cjelokupnog sistema zahtjevala bi kompletnu informaciju o sistemu (hardver, softver, aplikacije korisnika, karakterizacija opterećenja itd.) te je zbog potrebnih dugotrajnih ulaganja u razvoj kompleksnog modela vrlo skupa. Zbog toga se razmatranja u ovom radu ograničavaju na promatranje ulazno/izlaznog podsistema. Mjerenje efikasnosti rada i planiranja proširenja i rekonfiguracije ulazno/izlaznog podsistema pretpostavlja stvaranje modela podsistema pomoću kojeg će se moći ispitati efekt promjene opterećenja ulazno/izlaznog podsistema na parametre koji predstavljaju mjerila efikasnosti rada.

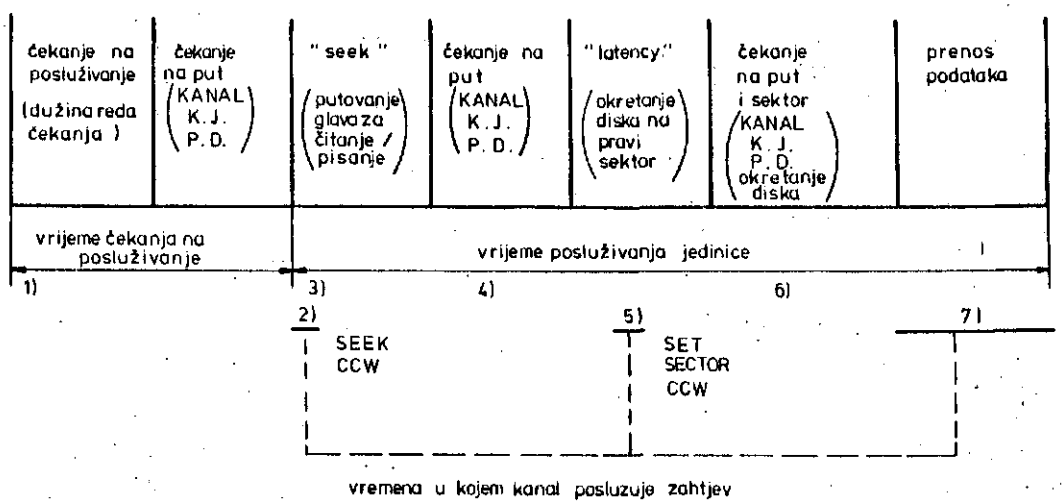
U literaturi je jako puno radova koji korištenjem različitih tehnika stvaraju na različitim nivoima detalja modele ulazno/izlaznog podsistema (empirijske, analitičke, simulacijske metode). U ovom radu se koristilo metodom simulacije. Da bi se lakše moglo usporedjivati rezultate dobivene iz modela i iz konkretnog sistema jedan dio izlaza iz simulatora po svom obliku je identičan obliku izlaza koji se dobiju programskim monitorima, razvijenim od strane proizvodjača, koji u odredjenim vremenskim intervalima snimaju status sistema i na osnovu tih snimanja mogu prikazati postotak zauzeća, srednje vrijeme posluživanja i čekanja u redu za odredjene resurse /vidi napr. (2), (3) /.

Ovdje provedeno modeliranje pretpostavlja konfiguracije u kojima postoji jedinstveni put od CPU-a do jedinice. Višestruki putevi preko više CPU-a ovdje nisu razmatrani i taj je problem obradjen u (1). Sigurno je da su i ta razmatranja interesantna, ali ona vrijede samo za jako velike instalacije kakve su ipak rjedje zastupljene. Dugotrajnim mjerenjima i proučavanjem realnog sistema (IBM 3031) dobivena je velika količina podataka koji su omogućili da se u model ide sa ulaznim vrijednostima koji odgovaraju stvarnim situacijama. Zbog toga se u ovdje razradjenom modelu pretpostavlja da je na svim jedinicama diskova koji pripadaju jednom nizu spojenom preko prednjeg diska (P.D.) na kontrolnu jedinicu samo vrijeme okretanja jednako dok se, za razliku od prethodnih modela, svi ostali interesantni podaci variraju od jedinice do jedinice, jer samo takva slika odgovara realnoj situaciji. Današnje ulazno/izlazni podsistemi srednje velikih i velikih konfiguracija imaju 6, 12 ili više U/I kanala. Pretpostavlja se da algoritmi za dodjelu puta unutar operativnog sistema nastoje približno jednako opteretiti svaki put od CPU-a do jedinica tako da ista razmatranja važe za svaki mogući put.

Sl. 1.



Sl. 2.



Sl. 3.

Kako je dodjela puta disk jedinici po principu FCFS. (first come first served) uglavnom napuštena i razmatrana je detaljno npr. u (8) u ovom se modelu pretpostavljao SLTF (shortest latency time first) koji se zbog svojih prednosti danas daleko češće susreće. Zbog toga prezentacija modela počinje sa kratkim opisom rada ulazno/izlaznog podsistema da se uoči svi nužni detalji koji su kasnije mjereni da bi se došlo do realnih vrijednosti i uzelo ih u obzir u modelu. Nakon toga su opisane i prikazane neke mogućnosti kroz modeliranje rada sistema simulatorom. Sugestije za korištenje i poboljšanje mogućnosti i točnosti simulatora dane su na kraju rada.

1. OPIS RADA ULAZNO/IZLAZNOG PODSISTEMA

Tipičan put koji se treba u izvodjenju jednog ulazno/izlaznog zahtjeva prema jedinicama sa direktnim dostupom preći je dan na slici 1. Ove komponente su prisutne kod IBM sistema S/370, 30XX i 43XX. Kod većih instalacija sa jednim procesorom ili više procesora taj put može biti znatno kompliciraniji i tada u razmatranju mogućeg puta učestvuje znatno više faktora istovremeno. U ovom radu se pretpostavlja da preko kanala do jedinice postoji jedinstveni put. Tipovi disk jedinica koje se u ovom radu analiziraju su tzv. RPS (rotational position sensing) koje omogućuju znatno bolje korištenje kanala zato što zauzimaju kanal samo u vrijeme prenosa komandi i prenosa podataka. Vrijeme zauzimanja kanala, kontrolne jedinice, prednjeg diska, i jedinice diska zbog prenosa komandi iako je vrlo malo (oko 0,5 ms) u ovom modelu je uzeto u obzir zato što kod većeg opterećenja ukupno vrijeme zauzeća kanala komandama nije zanemarivo i to povećava točnost modela. Faze koje se dešavaju kod izvodjenja jednog U/I zahtjeva su prikazane na slici 3. Zbog lakšeg praćenja i analize faze su numerirane sa 1) - 7).

1) Prvo čekanje nakon iniciranja U/I zahtjeva od strane CPU-a je čekanje na slobodnu jedinicu i na slobodan put do jedinice. To vrijeme čekanja je općenito funkcija od:
   - vremena u kojem je jedinica zauzeta zbog posluživanja prethodnog ulazno/izlaznog zahtjeva i ovisi o funkciji raspodjele U/I zahtjeva te vremenu posluživanja zahtjeva od strane disk jedinice
   - vremena čekanja puta zbog toga što je neki resurs (kanal, kontrolna jedinica, prednji disk) zauzet. Općenito se kod višestrukih puteva kao na sl. 2 mora promatrati odvojeno zauzimanje svakog od ovih resursa. U ovakvom modelu (sl.1) se može pretpostaviti da su oni istovremeno približno jednako zauzeti. Ovo vrijeme čekanja ovisi o vremenu posluživanja zahtjeva na ovim resursima i korištenju resursa (vjerojatnosti da je resurs zauzet).

2) Kada je put do disk jedinice slobodan inicira se komanda kojom se traži pomak glava za čitanje/pisanje na odredjeni cilindar.

3) Nakon poslane komande kanal i ostali resursi na tom putu se oslobadjaju i ostaje nadalje u posluživanju tog zahtjeva zauzeta disk jedinica. Prosječno vrijeme za tu operaciju se kod proračuna uzima kao polovica maksimalnog vremena. To vrijeme za različite jedinice iznosi od 20-30 ms. Zna se medjutim, da to vrijeme ovisi o razmještaju i broju istovremeno aktivnih datoteka te o smještaju VTOC-a (Volume Table of COntens) na disku i često je

značajno različito od polovice vremena maksimalnog trajanja "seek"-a.

4) Nakon što je završio "seek" disk jedinica signalizira završetak te operacije prema kanalu. Da bi se to obavilo mora biti slobodan prethodno definirani put kroz sve resurse. To vrijeme je očito funkcija korištenja kanala i vremena posluživanja zahtjeva.

5) Jedinica se ponovo spojila na kanal signalom o završetku "seek"-a i šalje se komanda za postavLjanje na odredjeni sektor. Statistički to vrijeme iznosi pola vremena potrebnog za okretaj diska.

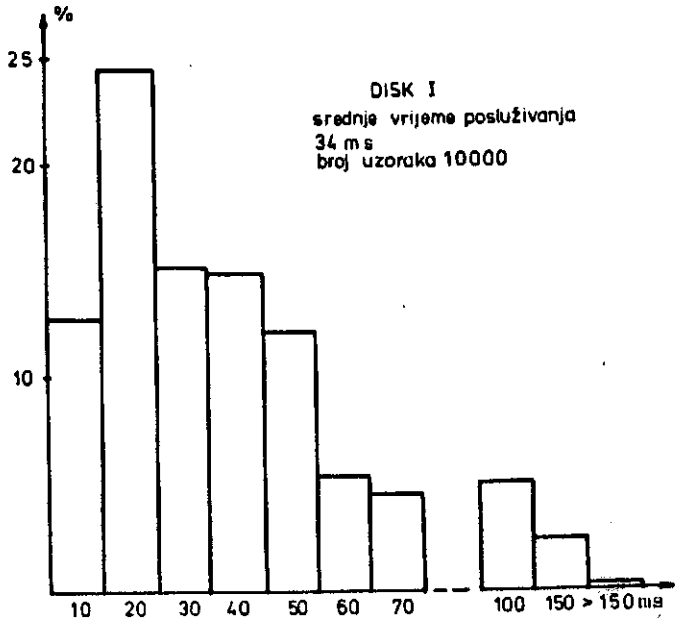
6) Kad se dosegao odgovarajući sektor jedinica pokušava da se nanovo spoji na CPU preko definiranog puta i to vrijeme opet ovisi o korištenju puta i vremenu posluživanja zahtjeva.

7) Sve dok kompletan put nije slobodan sistem je u fazi 6 koja traje općenito n okretaja diska. Interesantno je razmotriti koja je vjerojatnost da nakon što se jedinica nije uspjela spojiti na CPU u prvom okretaju spoji u nekom od slijedećih okretaja. Očito da to ovisi o broju pokušaja da se jedinica spoji i može se naći da je jednako vremenu potrebnom za okretanje diska pomnoženom sa postotkom zauzeća i podijeljenom sa (100 - postotak zauzeća).

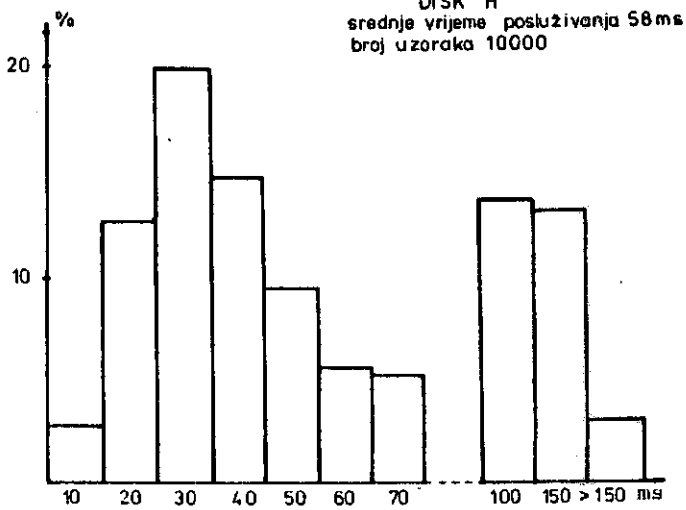
Uspješnim povezivanjem jedinice preko definiranog puta sa CPU-om počinje transfer podataka. Vrijeme za ovu operaciju ("search id" i "transfer") ovisi o pristupnoj metodi (ISAM, SAM, DAM), dužini bloka podataka na disku i načinu alociranja (staza, cilindar te zbog toga načinu prenosa: više staza, lanac CCW-ova itd.)

KONSTRUKCIJA MODELA I DOBIVANJE ULAZNIH PODATAKA

Proučavanjem rada različitih resursa uključenih u promatrani podsistem i odredjivanjem njihovih medjusobnih odnosa i veza problem je specificiran. Sada treba specificirati koji su



Sl. 4.



Sl. 5.

disk jedinica	broj uzoraka	ukupno vrijeme za U/I	prosj. vrijeme posl.	max. vrijeme posluž.	% poslu-	50	75	100	125	150	175	200	225	250	300	>
DISK A	11733	375.888	0.032	0.360		81.47	10.21	4.38	2.0	0.92	0.4	0.21	0.12	0.14	0.07	0.03
						81.47	91.69	96.07	98.08	99.00	99.4	99.61	99.74	99.88	99.96	99.99
DISK B	36785	1161.701	0.031	0.738		85.79	8.76	3.06	1.25	0.54	0.22	0.14	0.06	0.04	0.04	0.04
						85.79	94.55	97.62	98.87	99.41	99.64	99.79	99.86	99.90	99.95	99.99
DISK C	3025	139.888	0.042	0.531		77.71	9.61	3.37	2.31	1.85	1.81	1.61	0.76	0.36	0.19	0.36
						77.71	87.33	90.71	93.02	94.87	96.69	98.31	99.07	99.43	99.63	99.99
DISK D	41212	1514.527	0.036	0.375		80.99	11.13	3.55	3.01	0.79	0.27	0.10	0.04	0.01	0.03	0.01
						80.88	92.13	95.69	98.70	99.50	99.77	98.88	99.92	99.94	99.98	99.99
DISK E	14118	463.332	0.032	0.351		81.27	10.82	3.82	2.08	0.98	00.32	0.24	0.15	0.14	0.08	0.04
						81.27	92.10	95.92	92.01	99.00	99.32	99.56	99.72	99.87	99.95	99.94
DISK F	87178	5484.884	0.062	0.799		52.43	16.38	12.15	8.44	4.96	2.68	1.32	0.68	0.37	0.33	0.21
						52.43	68.81	80.97	89.41	94.37	97.06	98.38	99.06	99.94	99.78	99.99
DISK G	8142	143.213	0.017	0.339		93.83	3.66	0.90	0.71	0.52	0.07	0.02	0.02	0.14	0.06	0.02
						93.83	97.49	98.40	99.11	99.64	99.71	99.74	99.76	99.91	99.97	99.99

Tabela 1.

podaci potrebni da bi se moglo napraviti model sistema kojim će se prema specificiranom nivou detaljizacije proučavati fenomeni koji se u ovakvom podsistemu pojavljuju.

Podaci koji se moraju znati su:

- broj zahtjeva za ulazno/izlaznu operaciju u sekundi λ_i i = 1,2, ... N
- broj disk jedinica spojenih na jedan put N
- prosječno vrijeme potrebno za putovanje glava za čitanje/pisanje S_i i=1,...,N
- prosječna dužina prenesenog bloka podataka kroz kanal t_i
- brzina okretanja disk jedinice l_i

Jedina je l_i konstantna veličina za neki model disk jedinice, a ostale se veličine mijenjaju i ovise o konkretnoj situaciji.

Da bi se dobili točni podaci koji karakteriziraju neki odredjeni U/I podsistem vršeno je niz mjerenja različitim softverskim sredstvima. Budući da je potrebno dobiti što točnije podatke da se može izvršiti validacija i podešavanje točnosti rada simulatora korištena su programska sredstva koja zapisuju sve događaje iz nekih klasa događaja u sistemu (vidi (4)). Rezultat snimanja rada disk jedinica spojenih na jedan kanal dan je u Tabeli 1. Uočljivo je da postoji prilična razlika u srednjim vremenima posluživanja disk jedinica što je posljedica prije spomenutih različitih karakteristika i smještaja datoteka na njima te pristupnih metoda i načina rada pojedinih komponenti operativnog sistema koje s njima rade. Na slici 4 i 5 su dane frekvencije vremena posluživanja izmjerene za 2 diska.

Iz ovakvih mjerenja dobiveni su realni podaci kao posljedica karakteristike opterećenja pa se podsistem može promatrati kod maksimalnog, prosječnog, tipičnog itd. opterećenja. Ulaskom u model sa takvim podacima mjeri se efekt promjene i djelovanja na pojedine komponente kod različitih opterećenja i mogućih konfiguracija. Simulator se u principu može upogoniti sa podacima dobivenim mjerenjem tako da se koriste izmjereni događaji (trace-driven) ili se može na osnovu mjerenja naći funkcija raspodjele vjerojatnosti događaja i na taj ga način upogoniti. Često se koristi i kombiniranje te dvije metode. Ovdje dan model se upogonjava sa generiranjem slučajnih brojeva po funkciji raspodjele vjerojatnosti koja najbliže opisuje realni sistem. Sigurno da je interesantno napraviti i upravljanje realnim događajima te komparacijom mjeriti točnost pretpostavljenih funkcija raspodjele.

Izračunavanjem 0,90% nivoa pouzdanosti za pretpostavljene vrijednosti odredilo se broj potrebnih uzoraka koji se trebaju generirati. Isti postupak se primjenjuje i za mjerenje realnog sistema i potrebni broj promatranja. U prikazanim tabelama taj je broj uzoraka znatno iznad potrebnih i sigurno leži u danom intervalu pouzdanosti.

U modelu je pretpostavljeno da zahtjevi za posluživanje za neki disk I, 1 I N dolaze po Poissonovoj razdiobi sa srednjim brojem zahtjeva u jedinici vremena λ pa se vjerojatnost da će k zahtjeva doći u vremenskom intervalu opisuje sa:

$$P_K(T) = \frac{(\lambda \tau)^k}{k!} e^{-\lambda \tau} \quad \lambda > 0,\ k \geq 0, \tau \geq 0$$

Pretpostavljeno je nadalje da je vrijeme za postavljanje glava za čitanje/pisanje ("seek") dano po eksponencijalnoj razdiobi, raspodjela vremena potrebnog za okretanje diska ("latency" po uniformnoj razdiobi $[0,R]$ te vrijeme transfera po eksponencijalnoj razdiobi.

U modelu su, da sumariziramo, uzete još i slijedeće pretpostavke:

- diskovi rade po SLTF disciplini
- uzeto je u obzir trajanje komandi koje kod velikih opterećenja ipak nisu tako zanemarive u ukupnom opterećenju (prethodni modeli ne uzimaju komande u obzir)
- za vrijeme trajanja komande put je zauzet i obratno, kad je put zauzet transferom podataka, ne može se poslati komanda.
- u fazama dok je kanal zauzet (vidi sliku 3) postoji medjusobno djelovanje diskova na vrijeme posluživanja. Vrijeme koje se potroši u 6) ovisi o broju puta koliko se ponavlja i o vjerojatnosti da je put slobodan.
- pretpostavljeno je da su "seek", "latency" i posluživanje kanala za svaki zahtjev za posluživanje nezavisni.

DOBIVENI REZULTATI

Napomenuto je da su izlazni rezultati koji se dobivaju iz simulatora prilagodjeni obliku koji se dobiva iz programskih monitora na realnom sistemu. Iako je na ovom prostoru nemoguće pokazivati sve varijacije koji se mogu promatrati izabran je za promatranje sistem koji ima na kanal preko kontrolne jedinice spojena 4 diska. Za svaki disk su izabrane različite kombinacije trajanja "seek"-a, transfera podataka i "latency" faze. Promatralo se efekt porasta opterećenja po svakom disku (različit za svaki disk) na dužinu reda čekanja na posluživanje, prosječnu duljinu reda i prosječno vrijeme odgovora (posluživanja)toy diska. Uz to se može vidjeti kako se opterećenje pojedinih diskova reflektira na opterećenje kanala. Izračunat je broj potrebnih zahtjeva uz nivo pouzdanosti 0.90 i interval pouzdanosti od oko 5% srednje vrijednosti dužine reda. Nakon toga se promatrao efekt povećanja broja diskova na istom kanalu sa 4 na 6 i 8 diskova spojenih u jednom nizu. Za svaki niz je promatrana promjena opterećenja na prije dane veličine. Kako se iz toga za procjenu efikasnosti rada odredjene konfiguracije mogu promatrati s obzirom na dane kriterije različiti efekti, i za sve mogućnosti ovdje nema prostora kao ilustracija rada, prikazana je na nekoliko dijagrama promjena vremena posluživanja jedinica kod istog opterećenja kanala na različite diskove (sa različitim "seek"-om, transferom podataka) i različitim brojem disk jedinica.

U tabelama se prosječna dužina reda čekanja označila sa L_q, a prosječno vrijeme odgovara sa t_s. Ostale veličine su prije objašnjene.

	BROJ ZAHTJEVA	% RASPOD. REDA ČEKANJA 0	1	2	3	4+	L_q	90%	% ZAUZEĆA	t_s[ms]	λ_1	s_1[ms]	t_1[ms]	$l_1/2$[ms]
DISK 1	1343	98.1	1.9	0.0	0.0	0.0	0.019 ±	0.006	2.35	23.71	1	10	4	8
DISK 2	1340	96.6	3.4	0.1	0.0	0.0	0.036 ±	0.008	3.38	23.81	1	20	4	8
DISK 3	2661	94.0	5.8	0.2	0.0	0.0	0.062 ±	0.008	7.45	39.41	2	20	8	8
DISK 4	2656	96.0	3.8	0.2	0.0	0.0	0.041 ±	0.007	5.49	28.51	2	10	8	8
KANAL		89.1	10.3	0.5	0.0	0.0	0.11 ±	0.006	4.61					
DISK 1	1342	95.8	4.1	0.1	0.0	0.0	0.044 ±	0.009	4.89	25.63	2	10	4	8
DISK 2	1340	93.1	6.6	0.3	0.0	0.0	0.072 ±	0.012	7.06	37.77	2	20	4	8
DISK 3	2662	86.7	12.1	1.1	0.1	0.0	0.145 ±	0.012	15.17	43.18	4	20	8	8
DISK 4	2656	91.6	7.6	0.7	0.0	0.0	0.091 ±	0.010	11.29	31.04	4	10	8	8
KANAL		78.4	20.0	1.6	0.0	0.0	0.23 ±	0.008	9.23					
DISK 1	1342	91.4	8.2	0.4	0.0	0.0	0.089 ±	0.014	10.23	28.38	4	10	4	8
DISK 2	1340	85.4	12.9	1.5	0.1	0.0	0.163 ±	0.019	14.57	42.38	4	20	4	8
DISK 3	2662	72.4	21.4	5.2	0.8	0.2	0.350 ±	0.020	31.36	51.46	8	20	8	8
DISK 4	2656	81.3	15.1	3.1	0.4	0.1	0.228 ±	0.016	23.39	35.98	8	10	8	8
KANAL		58.6	34.3	6.6	0.5	0.0	0.49 ±	0.011	18.42					
DISK 1	1343	85.3	13.2	1.6	0.0	0.0	0.163 ±	0.015	16.37	32.48	6	10	4	8
DISK 2	1340	76.0	19.7	3.3	1.0	0.0	0.294 ±	0.025	23.15	49.23	6	20	4	8
DISK 3	2661	55.7	27.7	11.2	3.5	1.9	0.689 ±	0.031	49.04	67.00	12	20	8	8
DISK 4	2656	69.8	21.6	6.4	1.6	0.7	0.421 ±	0.024	36.68	43.25	12	10	8	8
KANAL		38.5	44.2	15.2	2.1	0.0	0.81 ±	0.014	27.73					
DISK 1	1342	76.6	20.0	2.9	0.4	0.0	0.272 ±	0.024	23.89	38.01	8	10	4	8
DISK 2	1340	65.9	24.5	7.3	1.9	0.4	0.465 ±	0.034	33.01	57.89	8	20	4	8
DISK 3	2662	37.5	27.0	16.2	9.0	10.3	1.367 ±	0.050	67.21	95.91	16	20	8	8
DISK 4	2656	54.8	26.8	11.2	4.9	2.3	0.744 ±	0.034	51.22	55.45	16	10	8	8
KANAL		22.3	43.7	27.9	6.0	0.0	1.18 ±	0.015	36.84					
DISK 1	1341	65.8	25.1	7.1	1.9	0.2	0.457 ±	0.033	33.59	48.48	10	10	4	8
DISK 2	1341	54.1	28.9	11.6	4.0	1.5	0.706 ±	0.043	44.18	71.71	10	20	4	8
DISK 3	2662	14.4	14.8	10.4	8.2	52.2	6.020 ±	0.201	87.98	309.47	20	20	8	8
DISK 4	2656	39.0	26.6	14.9	8.8	10.8	1.375 ±	0.053	66.34	78.24	20	10	8	8
KANAL		9.0	36.6	40.3	14.1	0.0	1.59 ±	0.015	46.02					

Tabela 2.

	BROJ ZAHTJEVA	% RASPOD. REDA ČEKANJA 0	1	2	3	4+	L_q	90%	% ZAUZEĆA	t_s[ms]	λ_1	s_1[ms]	t_1[ms]	$l_1/2$[ms]
DISK 1	1512	98.2	1.8	0.0	0.0	0.0	0.018 ±	0.005	2.33	23.53	1	10	4	8
DISK 2	1480	97.3	2.6	0.1	0.0	0.0	0.028 ±	0.007	3.37	34.96	1	20	4	8
DISK 3	2974	93.0	6.6	0.3	0.0	0.0	0.073 ±	0.008	7.33	39.43	2	20	8	8
DISK 4	2998	95.9	3.9	0.1	0.0	0.0	0.042 ±	0.006	5.50	28.49	2	10	8	8
DISK 5	1567	97.4	2.6	0.0	0.0	0.0	0.026 ±	0.007	2.46	24.28	1	10	4	8
DISK 6	1469	96.9	3.0	0.1	0.0	0.0	0.031 ±	0.007	3.29	34.51	1	20	4	8
KANAL		84.8	14.2	1.0	0.1	0.0	0.16 ±	0.006	5.57					
DISK 1	1512	96.6	3.3	0.1	0.0	0.0	0.034 ±	0.007	4.95	25.94	2	10	4	8
DISK 2	1480	93.0	6.6	0.3	0.0	0.0	0.073 ±	0.012	7.06	38.52	2	20	4	8
DISK 3	2974	86.9	11.3	1.5	0.2	0.1	0.152 ±	0.012	15.12	43.87	4	20	8	8
DISK 4	2998	91.2	8.0	0.7	0.1	0.0	0.097 ±	0.009	11.38	31.48	4	10	8	8
DISK 5	1567	95.3	4.7	0.1	0.0	0.0	0.048 ±	0.009	5.21	26.63	2	10	4	8
DISK 6	1469	93.1	6.4	0.5	0.0	0.0	0.074 ±	0.012	6.80	37.21	2	20	4	8
KANAL		70.3	25.4	4.0	0.2	0.0	0.34 ±	0.009	11.15					

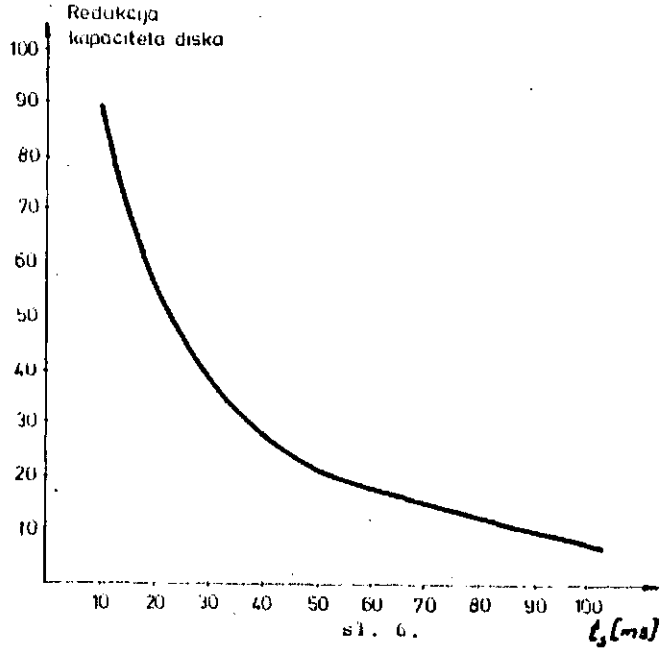
	BROJ ZAHTJEVA	% RASPOD. REDA ČEKANJA 0	1	2	3	4+	L_q	90%	% ZAUZEĆA	t_s[ms]	λ_1	s_1[ms]	t_1[ms]	$1_1/2$[ms]
DISK 1	1512	91.5	8.1	0.3	0.0	0.0	0.088	± 0.013	10.55	29.32	4	10	4	8
DISK 2	1480	85.5	12.8	1.5	0.1	0.0	0.162	± 0.018	14.70	43.42	4	20	4	8
DISK 3	2974	71.2	21.7	5.6	1.1	0.4	0.379	± 0.020	31.79	54.17	8	20	8	8
DISK 4	2998	79.6	16.6	2.9	0.8	0.1	0.251	± 0.016	23.78	37.18	8	10	8	8
DISK 5	1567	89.2	9.6	1.0	0.1	0.0	0.121	± 0.015	11.45	31.53	4	10	4	8
DISK 6	1469	86.2	12.3	1.4	0.1	0.0	0.154	± 0.017	14.47	42.67	4	20	4	8
KANAL		44.2	38.7	14.3	2.5	0.3	0.76	± 0.012	22.30					
DISK 1	1512	83.6	15.2	1.1	0.1	0.0	0.177	± 0.017	17.53	34.42	6	10	4	8
DISK 2	1480	77.0	18.2	4.2	0.4	0.1	0.284	± 0.014	23.40	50.43	6	20	4	8
DISK 3	2974	53.5	27.6	12.1	4.4	2.3	0.753	± 0.031	50.20	71.89	12	20	8	8
DISK 4	2998	66.9	23.7	6.6	2.0	0.8	0.461	± 0.023	37.71	45.58	12	10	8	8
DISK 5	1567	82.1	15.2	2.5	0.2	0.0	0.207	± 0.020	19.13	37.72	6	10	4	8
DISK 6	1469	77.1	18.9	3.5	0.5	0.1	0.277	± 0.024	23.89	51.74	6	20	4	8
KANAL		23.6	38.4	26.8	9.1	2.0	1.28	± 0.015	33.47					
DISK 1	1512	76.1	19.4	3.8	0.7	0.1	0.292	± 0.024	26.43	42.60	8	10	4	8
DISK 2	1480	67.4	22.8	7.4	1.9	0.5	0.454	± 0.033	33.91	61.92	8	20	4	8
DISK 3	2974	31.9	26.9	16.9	10.3	14.0	1.657	± 0.055	70.85	114.47	10	20	8	8
DISK 4	2998	50.0	26.6	12.9	5.9	4.6	0.918	± 0.037	55.24	65.50	10	10	8	8
DISK 5	1567	72.6	22.4	3.9	0.9	0.3	0.339	± 0.026	27.64	44.14	8	10	4	8
DISK 6	1469	66.6	23.6	7.2	2.5	0.2	0.462	± 0.033	34.26	62.51	8	20	4	8
KANAL		9.5	28.4	33.8	20.5	7.8	1.90	± 0.017	44.57					
DISK 1	1512	70.3	22.6	5.7	1.2	0.3	0.386	± 0.029	31.60	48.22	9	10	4	8
DISK 2	1480	61.1	26.4	8.5	3.0	1.1	0.567	± 0.036	39.46	68.20	9	20	4	8
DISK 3	2972	17.7	17.9	13.9	11.7	38.7	3.785	± 0.129	83.68	219.90	18	20	8	8
DISK 4	2998	39.2	25.9	14.7	8.6	11.5	1.374	± 0.048	64.43	83.65	18	10	8	8
DISK 5	1568	66.8	23.7	7.1	1.5	0.9	0.463	± 0.034	33.12	52.22	9	10	4	8
DISK 6	1470	60.8	27.1	8.6	3.0	0.5	0.553	± 0.034	40.86	69.86	9	20	4	8
KANAL		4.7	20.0	34.0	27.9	13.3	2.27	± 0.017	50.13					

Tabela 3.

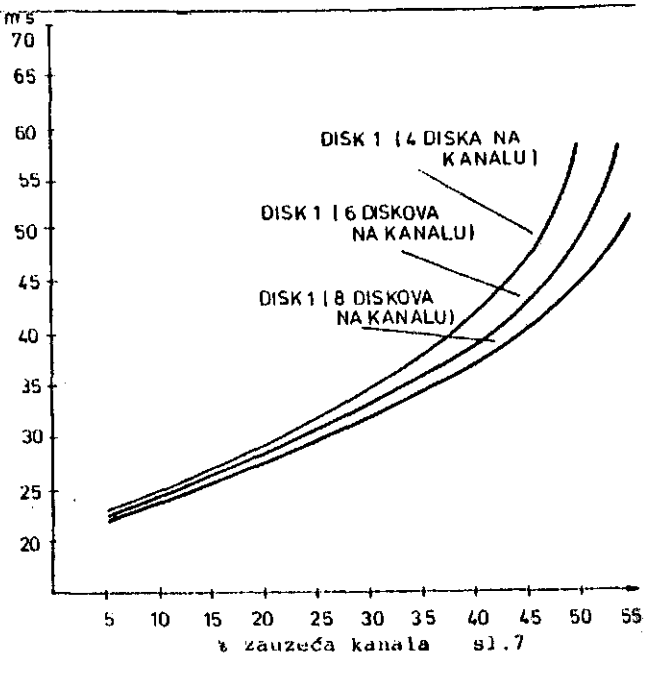
	BROJ ZAHTJEVA	% RASPOD. REDA ČEKANJA 0	1	2	3	4+	L_q	90%	% ZAUZEĆA	t_s[ms]	λ_1	s_1[ms]	t_1[ms]	$1_1/2$[ms]
DISK 1	1322	98.3	1.6	0.1	0.0	0.0	0.017	± 0.006	2.38	24.82	1	10	4	8
DISK 2	1358	97.3	2.7	0.1	0.0	0.0	0.028	± 0.007	3.47	35.34	1	20	4	8
DISK 3	2702	93.3	6.2	0.6	0.0	0.0	0.073	± 0.009	7.75	40.84	2	20	8	8
DISK 4	2652	95.7	4.2	0.1	0.0	0.0	0.043	± 0.006	5.45	28.77	2	10	8	8
DISK 5	1321	98.5	1.4	0.2	0.0	0.0	0.017	± 0.006	2.43	25.46	1	10	4	8
DISK 6	1316	96.8	3.2	0.0	0.0	0.0	0.032	± 0.008	3.38	35.33	1	20	4	8
DISK 7	2706	93.5	6.1	0.3	0.0	0.0	0.068	± 0.008	7.72	40.51	2	20	8	8
DISK 8	2623	96.0	4.0	0.1	0.0	0.0	0.041	± 0.006	5.59	29.65	2	10	8	8
KANAL		76.0	21.1	2.7	0.2	0.0	0.27	± 0.008	9.20					
DISK 1	1322	96.2	3.6	0.2	0.0	0.0	0.039	± 0.009	5.18	28.20	2	10	4	8
DISK 2	1358	94.2	5.4	0.4	0.0	0.0	0.063	± 0.011	7.24	38.51	2	20	4	8
DISK 3	2702	86.0	12.1	1.7	0.1	0.0	0.160	± 0.012	16.30	46.85	4	20	8	8
DISK 4	2652	90.9	8.4	0.6	0.1	0.0	0.099	± 0.010	11.67	32.82	4	10	8	8
DISK 5	1321	95.6	4.2	0.0	0.0	0.0	0.046	± 0.010	5.32	28.98	2	10	4	8
DISK 6	1316	92.2	7.3	0.5	0.0	0.0	0.084	± 0.013	7.19	39.95	2	20	4	8
DISK 7	2706	86.1	12.4	1.3	0.1	0.0	0.156	± 0.013	16.16	46.08	4	20	8	8
DISK 8	2623	91.4	8.0	0.6	0.0	0.0	0.092	± 0.010	11.77	33.36	4	10	8	8
KANAL		54.5	34.2	9.6	1.5	0.2	0.59	± 0.009	18.42					

	BROJ ZAHTJEVA	% RASPOD. REDA ČEKANJA 0	1	2	3	4+	L_q	90%	% ZAUZEĆE	t_s[ms]	λ_i	s_i[ms]	t_i[ms]	$1_i/2$[ms]
DISK 1	1322	93.2	6.6	0.2	0.0	0.0	0.070 ±	0.012	8.45	31.63	3	10	4	8
DISK 2	1358	89.4	9.6	1.0	0.0	0.0	0.117 ±	0.016	11.55	43.11	3	20	4	8
DISK 3	2702	76.5	19.2	3.4	0.9	0.0	0.289 ±	0.018	25.60	53.93	6	20	8	8
DISK 4	2652	84.6	13.3	1.8	0.3	0.0	0.178 ±	0.014	18.81	37.47	6	10	8	8
DISK 5	1321	93.1	6.4	0.5	0.1	0.0	0.075 ±	0.013	8.40	31.57	3	10	4	8
DISK 6	1316	87.3	11.4	1.3	0.0	0.0	0.140 ±	0.017	11.53	44.66	3	20	4	8
DISK 7	2706	78.5	17.5	3.3	0.6	0.1	0.263 ±	0.018	25.29	52.34	6	20	8	8
DISK 8	2623	85.1	13.5	1.4	0.1	0.0	0.165 ±	0.011	18.79	37.98	6	10	8	8
KANAL		36.2	38.6	18.3	5.7	1.2	0.97 ±	0.011	27.58					
DISK 1	1322	89.0	10.2	0.8	0.0	0.0	0.117 ±	0.017	12.33	35.79	4	10	4	8
DISK 2	1358	85.2	13.0	1.5	0.2	0.0	0.168 ±	0.019	16.37	47.80	4	20	4	8
DISK 3	2702	65.4	24.6	7.2	2.2	0.6	0.479 ±	0.025	36.29	65.01	8	20	8	8
DISK 4	2652	76.4	19.2	3.6	0.7	0.1	0.288 ±	0.019	26.80	43.87	8	10	8	8
DISK 5	1321	89.0	10.2	0.8	0.0	0.0	0.117 ±	0.015	12.44	36.77	4	10	4	8
DISK 6	1316	82.2	15.4	2.1	0.3	0.0	0.204 ±	0.021	15.99	48.66	4	20	4	8
DISK 7	2706	68.8	22.4	6.8	1.8	0.1	0.420 ±	0.022	35.50	61.44	8	20	8	8
DISK 8	2623	77.6	18.3	3.5	0.5	0.0	0.271 ±	0.018	26.95	44.38	8	10	8	8
KANAL		21.8	36.2	25.6	12.0	4.4	1.42 ±	0.014	36.79					
DISK 1	1322	78.1	17.2	3.6	0.9	0.2	0.278 ±	0.026	23.47	52.18	6	10	4	8
DISK 2	1359	70.3	22.6	5.9	1.1	0.1	0.383 ±	0.029	29.81	67.36	6	20	4	8
DISK 3	2701	38.9	26.0	14.9	7.8	12.3	1.421 ±	0.054	62.65	122.90	12	20	8	8
DISK 4	2652	54.7	26.4	11.3	4.8	2.8	0.757 ±	0.034	49.44	73.58	12	10	8	8
DISK 5	1321	77.9	17.6	3.6	0.8	0.2	0.278 ±	0.024	23.35	51.31	6	10	4	8
DISK 6	1316	69.2	21.0	7.9	1.4	0.5	0.429 ±	0.034	29.69	72.67	6	20	4	8
DISK 7	2706	40.8	28.5	15.8	8.0	6.9	1.152 ±	0.041	62.31	107.44	12	20	8	8
DISK 8	2623	55.7	28.1	10.2	3.9	2.1	0.691 ±	0.031	48.01	68.77	12	10	8	8
KANAL		4.7	15.9	25.8	25.7	27.9	2.69 ±	0.018	55.23					
DISK 1	1322	67.3	22.6	6.6	2.0	1.5	0.483 ±	0.038	33.17	73.31	7	10	4	8
DISK 2	1359	59.2	25.8	9.9	3.2	1.9	0.634 ±	0.041	40.28	90.93	7	20	4	8
DISK 3	2703	19.9	17.1	14.2	12.7	36.1	3.075 ±	0.088	80.11	226.15	14	20	8	8
DISK 4	2651	39.2	26.7	14.6	8.7	10.8	1.373 ±	0.053	63.84	108.24	14	10	8	8
DISK 5	1321	66.2	22.0	8.3	2.1	1.4	0.512 ±	0.041	31.82	70.63	7	10	4	8
DISK 6	1316	59.9	25.9	10.1	2.1	2.1	0.622 ±	0.043	37.98	87.70	7	20	4	8
DISK 7	2705	22.0	19.1	15.5	11.6	31.8	2.870 ±	0.088	79.77	215.01	14	20	8	8
DISK 8	2623	38.6	28.6	15.3	8.8	8.7	1.284 ±	0.048	63.70	105.09	14	10	8	8
KANAL		1.2	5.9	15.5	24.3	53.0	3.60 ±	0.018	64.36					

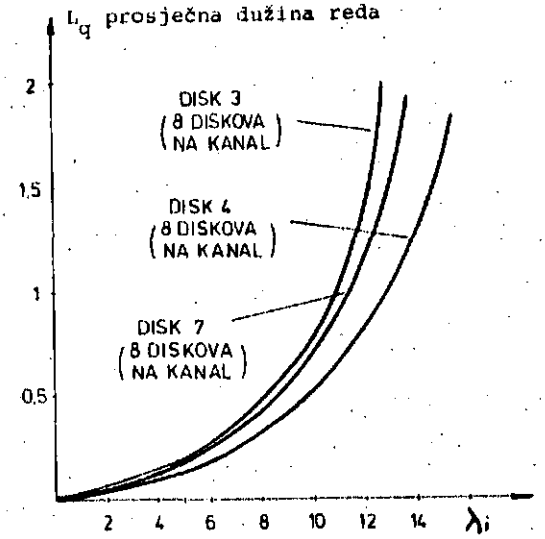
Tabela 4.



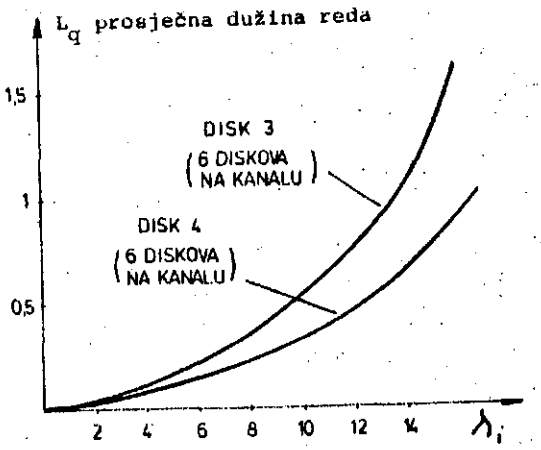
sl. 6.



sl.7



Sl. 8.



Sl. 9.

ZAKLJUČAK

Rezultati dobiveni modelom omogućuju proučavanje važnih svojstava i pojava u U/I podsistemu jedinica sa direktnim dostupom. U modelu se nije išlo za postizanjem potpune teoretske točnosti već se u prvi plan stavila funkcionalnost modela. Uostalom, potpuna teoretska točnost nije postignuta niti jednim analitičkim modelom kojim se rješavala ova klasa problema i to zbog nesavršenosti matematičkog aparata s jedne strane i zbog kompleksnosti sistema i pojava kada se problem pokušava rješiti bez aproksimacija na dovoljno velikom nivou detalja.
S obzirom na dobivene rezultate i svrhu modela (procjena efikasnosti rada neke konfiguracije i proučavanje medjusobne ovisnosti važnih parametara) model u potpunosti zadovoljava postavljene ciljeve i znatno je precizniji od softverskih monitora prisutnih u sistemu. U principu model je pogodan i za komercijalnu upotrebu i daje dovoljno precizne podatke za procjenu performansi i planiranje kapaciteta U/I podsistema. Daljnji rad na modelu može omogućiti poboljšanja za:

- ispitivanje ponašanja U/I podsistema sa višestrukim putevima
- upravljanje simulatora sa podacima dobivenim direktno iz softverskih monitora koji zapisuju svaki dogadjaj u sistemu (za sad se to radi za operativni sistem MVS)
- korištenje rezultata iz ovog modela za konstrukciju modela koji bi na višem nivou dao odgovore na pitanja vezana uz rad cjelokupnog puta zahtjeva za obradom iniciranog od terminala ili kao "batch" posla.

LITERATURA

1. Bard Y.,: "A Model of Shared DASD and Multipathing", CACM, no.10, vol.23,1980, str. 564-572.
2. IBM:"OS/VS2 MVS Resource Measurement Facility (RMF) Reference and User's Guide", form no. SC 28-0922, 1980.
3. IBM: "OS/VS2 SPL: Initialization and Tuning Guide", form no. GC 28-1029, 1979.
4. IBM: "OS/VS2 SPL: Service Aids", form no. GC28-0674, 1979.
5. IBM: "Reference Manual for IBM 3830 Storage Control Model 2", form no. GA26-1617, 1977.
6. IBM: "Reference Manual for IBM 3350 Direct Access Storage", form. no. GA26-1638, 1977.
7. Knežević P.,: "Prilog informacijskoj analizi sustava za automatsko projektiranje procesorski upravljanih komutacija", doktorska disertacija, Zagreb, 1982.
8. Wilhelm N.C.,: "A General Model for the Performance of Disk Systems", JACM, no.1, vol. 24, 1977, str. 14-31.
9. Ziegler K.,: "DASD Configuration and Sharing Considerations", IBM Complex Systems, form no. GG22-9052, 1978.

PROŠIRENO UPRAVLJANJE MEMORIJOM ZA RAČUNAR DELTA 340

MILOVAN V. JEFIĆ
ISKRA DELTA, LJUBLJANA

UDK: 681.3:181.4

Proširenо upravljanje memorijom je izvedeno kao posebna celina, koja se pridodaje postojećim instalacijama DELTA 340 (PDP 11/34), te se tako povećava područje adresiranja sa 256 KB na 4 MB. Na takav način rašírena memorija poveća sposobnosti celoga sistema sa obzirom na overhead, koji se javlja zbog čestog komuniciranja sa diskom.

EXTENDED MEMORY MANAGEMENT (EMM) is an upgrade which may be fitted to an Delta 340 processor to increase its addressing range from 256 KB to 4 MB. This enhancement greatly increases system efficiency as more memory may be fitted to reduce the overhead of job swapping to disc and allow extra memory resident routines.

1. UVOD

Standardni DELTA 340 (PDP 11/34) procesor ima 16 bitnu reč, koja omogućava direktno adresiranje 64 KB memorije. U koliko želimo adresirati šire područje, moramo uvesti pojam relokacije, t.j. sa dodavanjem relokacijske konstante omogućujemo 16 bitnoj adresi adresiranje šireg adresnog prostora.
Standardni DELTA 340 procesor ima relokacijsku konstantu ograničenu na 12 bitova sa čim se omogućava 18 bitno adresiranje. Ako sada dodamo još 4 bita relokacijskom faktoru, možemo generisati 22 bitnu adresu sa kojom pokrivamo 4 MB adresni prostor.

To je osnovni princip proširenog upravljanja memorijom. U koliko želimo omogućiti standardnom 18 bitnom adresiranju dostup do dodatne memorije, potrebno je izvršiti i odgovarajuću kontrolu registara (sl. 1).

2. VIRTUALNO ADRESIRANJE

Proširenо upravljanje memorijom (EMM) se omogućava ako u registru MMR0 aktiviramo multi-bit, tada normalna 16 bitna adresa nije interpretirana kao direktna fizična adresa, nego kao virtualna adresa, koja nadrži informacije za konstrukciju nove 18 ili 22 bitne fizične adrese.
Ako pa u registru MMR3 aktiviramo bit 4 na EMM je omogućeno 22 bitno relociranje. Informacija koja sadrži virtualna adresa se kombinira sa relokacijskom i opisnom informacijom, koja je sadržana u Active Page registrima (APR) te se dobije 18/22 bitna adresa na sabirnici.
Virtualni adresni prostor je podeljen na stranice veličine 32 do 4096 reči i svaka stranica se posebno relocira.

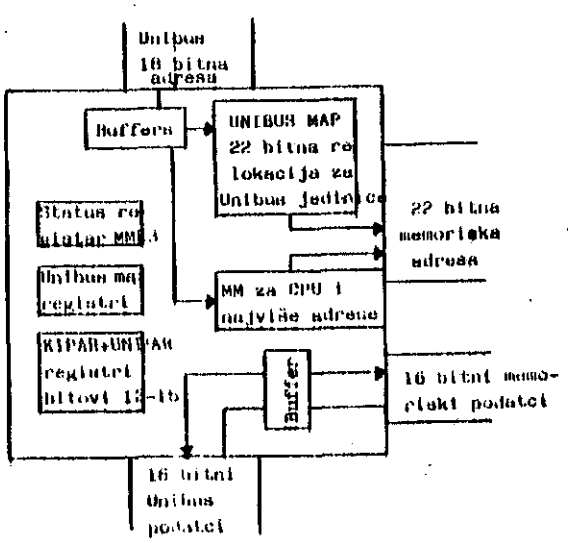
3. VIŠESTRUKI ADRESNI PROSTOR

Postoje dva kompleta (APR) registara PAR/PDR kao sastavni dio procesora DELTA 340. Jedan set je za osnovni način (Kernel) i jedan set za korisnički (User) način rada.
Koji od načina rada je upotrebljen, vidi se iz aktivnosti 14 i 16 bita u procesorskoj reči stanja (PSW).

PSW bit	15	14	
	0	0	Kernel način
	0	1	Nevažeća stanja
	1	0	
	1	1	User način

4. ACTIVE PAGE REGISTRI (APR)

DELTA 340 ima dva seta po 8 APR-ov. Svaki APR sadrži Page Address Register (PAR) i Page Descriptor Register (PDR). Ovi registri se uvjek upotrebljavaju u paru i sadrže svu informaciju potrebnu za locitanje i opis tekuće aktivne stranice za svaki način rada. Koja grupa PAR/PDR registara je selektirana, određuju bitovi 13, 14 i 15 šesnaest bitne virtualne adrese.



Slika 1.

Virtualna adresa	Bit 15	Bit 14	Bit 13	PAR/PDR Set
000000 - 017776	0	0	0	0
020000 - 037776	0	0	1	1
040000 - 057776	0	1	0	2
060000 - 077776	0	1	1	3
100000 - 117776	1	0	0	4
120000 - 137776	1	0	1	5
140000 - 157776	1	1	0	6
160000 - 177776	1	1	1	7

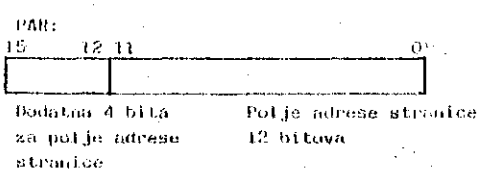
5. PAR/PDR ADRESE

Tačno su određene I/O adrese za svaki PAR i PDR:

PAR/PDR Set	Kernal		User	
	PAR	PDR	PAR	PDR
0	772340	772300	777640	777600
1	772342	772302	777642	777602
2	772344	772304	777644	777604
3	772346	772306	777646	777606
4	772350	772310	777650	777610
5	772352	772312	777652	777612
6	772354	772314	777654	777614
7	772356	772316	777656	777616

6. PAGE ADDRESS REGISTER (PAR)

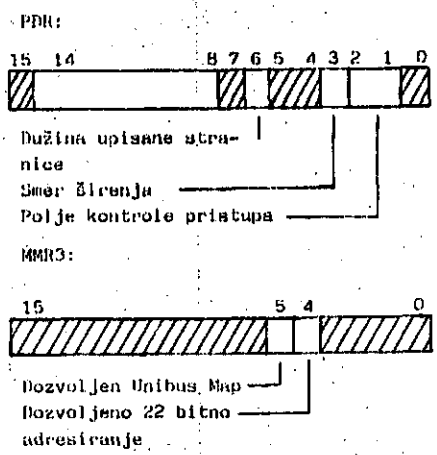
PAR u procesoru DELTA 340 sadrži 16 bitno Page Address polje (relokacioni faktor), sa kojim je specificirana bazična adresa stranice. Pri radu sa 18 bitnom adresom (bit 4 MMR3 nije aktivan) se upotrebi samo donjih 12 bitova PAR-a, a u slučaju 22 bitnog načina se upotrebi svih 16 bitova kao relokacioni faktor.



7. PAGE DESCRIPTOR REGISTER (PDR)

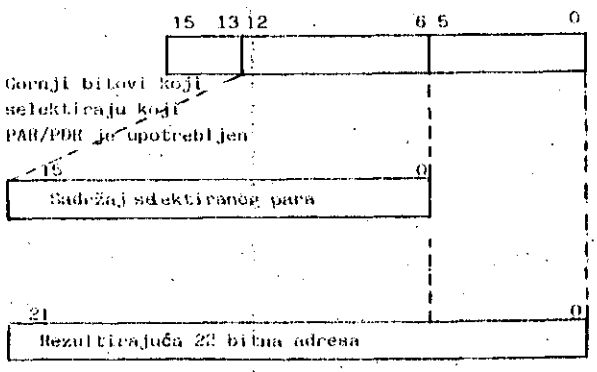
Registri opisa stranica (PDR) se ne menjaju sa proširenim upravljanjem memorije i predstavljaju četiri scratchpad memorije, koje sadrže informaciju o širenju stranica, dužini stranice i kontroli pristupanja.

Adresa MMR3 registra je 772516 i on se nalazi na EMM modulu, a dozvoljava 22 bitno ili Unibus mapiranje.
Sa aktiviranjem bita 4 se omogući proširено upravljanje memorijom i CPU razširi adresiranje na 18 ili 22 bita.
Ukoliko se aktivira bit 5, tada Unibus mapiranim jedinicama se omogući dostup do dodatne memorije. Oba bita se izbrišu sa inicijalizacijom sabirnice (Unibus).



Kada je aktiviran EMM bitovi 13 - 15, virtualne adrese su upotrebljeni za određivanje koji od osam PAR/PDR registara je iskorišten za izabrani način rada.
Bitovi virtualne adrese 6 - 12 se saberu sa sadržajem PAR registra, a bitovi 0 - 5 se dodaju nepromenjeni i dobijen rezultat je 22 bitna adresa.

Pretvaranje virtualne u fizičku adresu:



8. EMM UNIBUS MAP

8.1 Funkcija mapiranja

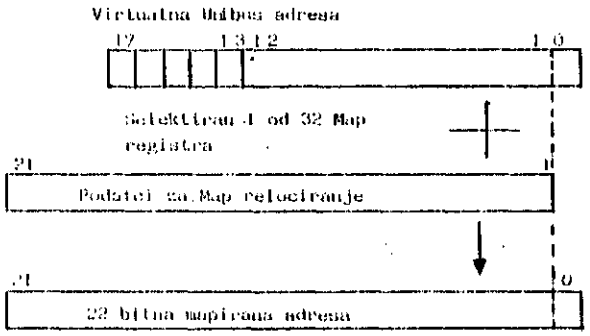
Unibus map omogućava da jedinice na Unibus-u komuniciraju sa fizičkom memorijom. 18 bitna adresa na Unibus-u se prevede na 22 bitnu adresu pomoću logike za relociranje. Relokaciska logika se aktivira postavljanjem bita 5 u MMR3. Najviših 4 KB od 256 KB Unibus adresnog prostora je rezervirano za CPU i I/O registre i ne locira se sa mapiranjem.

8.2 Zabranjeno relociranje

Kada bit 5 u MMR3 ostane neaktivan je relociranje onemogućeno i 18 bitnoj Unibus adresi se doda na najvišim mjestima 4 nule za izgradnju 22 bitne adrese, niži 18 bitova ostane nepromjenjeno.

8.3 Relociranje

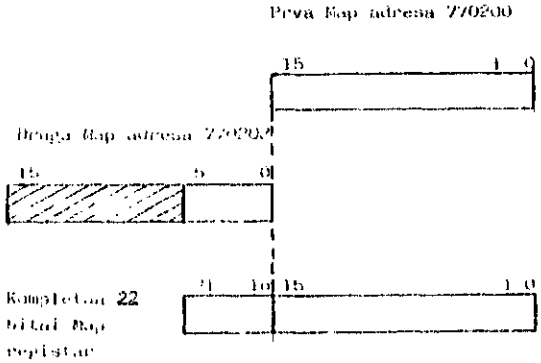
MM logika gleda takodje na 18 bitnu adresu kao virtualnu. Najviših 5 bitova služi za izbor jednog 22 bitnog relokacionog registra. Sadržaj izabranog registra se sabere sa bitovima 1 - 12 Unibus adrese te se dobije 22 bitna fizična adresa. U toku tog procesa je map bit 0 stalno nula, jer se sve izvršava na nivou reči.
Kombinacija bitova 13 - 17 daje podatak za adresiranje 32 registra i kada su ti bitovi postavljeni u 1, je map relokacija onemogućena i tada je prepredjeno relociranje I/O stranica od 760000 do 777776.



8.4 Map registri

Svaki 22 bitni map registar je kombinacija dva 16 bitna registra i zauzima jednu od adresa između 770200 i 770372.

Bitovi 1 - 15 i 0 - 5 se u svakom registru mogu izmenično upisati.



9. ZAKLJUČAK

Prošireno upravljanje sa memorijom omogućava naslednje sistemske karakteristike:

Memorijski adresni prostor	4 MB
Adresni načini	Virtualno 16 bitno, fizično 18/22 bitno
Načini rada	Osnovni i korisnički
Stack pokazivači	2, jedan za svaki način
Memorisko relociranje	16 stranica (8 za svaki način)
Dužina stranice	32 do 4096 reči
Zaštita memorije	Nema pristupa, samo čitanje, čitanje/unašanje

10. LITERATURA

1. Peter D. Vogt: Virtual memory extension for an existing *minicomputer*, *Computer Design*, July 1981

2. R. Colin Johnson: Microsystems exploit *mainframe* methods, Electronics, August 11, 1981

3. PDP 11/34 (DELTA 340) Processor Handbooks

A TWO-STEP METHOD FOR FINDING ROOTS

D.B. POPOVSKI

SOCIETY OF SCIENCE AND ART-BITOLA, YUGOSLAVIA

UDK: 681.3:51

In this paper a two-step method for solving single real variable equations is presented. The method is derived on the basis of a two-point Hermite approximation by the rational function $y(x)=(x-a)/(bx^2+cx+d)$. Proposed method in each iteration step requires two function evaluations and one evaluation of the first two derivatives and has an asymptotic convergence rate 6.541.

JEDNA DVOKORACNA METODA ZA NALAZENJE KORENA. *U radu je prezentirana jedna dvokoraćna metoda sa rešavanje jednačina sa jednom realnom nepoznatom. Metoda je izvedena na bazi dvotačkaste Hermite-ove aproksimacije racionalnom funkcijom $y(x)=(x-a)/(bx^2+cx+d)$. Predložena metoda u svakoj iteraciji zahteva dva izračunavanja funkcije i jedno izračunavanje njenih prvih dva izvoda i ima red konvergencije 6.541.*

Popovski [1] proposed a method for solving single real variable equations of the form

$$f(x)=0$$

in which he uses a two-point Hermite approximation by the rational function

$$y(x)=(x-a)/(bx^2+cx+d) \qquad (1)$$

whose parameters a,b,c and d are determined from the conditions

$$y(x_{i-2})=f(x_{i-2})$$
$$y^{(j)}(x_{i-1})=f^{(j)}(x_{i-1}) \quad (j=0,1,2)\ .$$

His method

$$x_i=x_{i-1}+p_{i-1}r_{i-1}/(([f'^2(x_{i-1})-0.5f(x_{i-1})\cdot f''(x_{i-1})]p_{i-1}q_{i-1}-r_{i-1}) \equiv \Phi_1(x_{i-1},x_{i-2}) \qquad (2)$$

where

$$p_{i-1}=x_{i-1}-x_{i-2}$$
$$q_{i-1}=p_{i-1}f(x_{i-2})$$
$$r_{i-1}=([f(x_{i-1})-f(x_{i-2})]f(x_{i-1})-f'(x_{i-1})q_{i-1})f(x_{i-1})$$

in each iteration step requires one evaluation of the function and of each of its first two derivatives and has an asymptotic convergence rate 3.303.

Using a two-point Hermite approximation by rational function (1) in which the parameters a, b, c and d are determined from the conditions

$$y^{(j)}(x_{i-2})=f^{(j)}(x_{i-2}) \quad (j=0,1,2)$$
$$y(x_{i-1})=f(x_{i-1})$$

we may easily obtain the method

$$x_i=x_{i-1}+p_{i-1}r_{i-1}/(([f(x_{i-2})-f(x_{i-1})]f(x_{i-2})+f'(x_{i-2})q_{i-1})f(x_{i-2})-r_{i-1}) \equiv \Phi_2(x_{i-1},x_{i-2}) \qquad (3)$$

where

$$p_{i-1}=x_{i-1}-x_{i-2}$$
$$q_{i-1}=p_{i-1}f(x_{i-1})$$
$$r_{i-1}=[f'^2(x_{i-2})-0.5f(x_{i-2})f''(x_{i-2})]p_{i-1}q_{i-1}$$

Method (3), as well as method (2), in each iteration step requires one evaluation of the function and of each of its first two derivatives, but has asymptotic convergence rate only 3.

Combining (2) and (3) we may obtain a two-step method,

$$\begin{aligned} \omega_i &= \phi_1(x_{i-1}, \omega_{i-1}) \\ x_i &= \phi_2(\omega_i, x_{i-1}) \end{aligned} \qquad (4)$$

which requires two evaluations of the function and one evaluation of each of its first two derivatives and has asymptotic convergence rate 6.541.

To prove the asymptotic convergence rates of all the above mentioned methods we use Herzberger's matrix method [2] according to which the order of a single-step method

$$x_i = \phi(x_{i-1}, x_{i-2}, \cdots, x_{i-n})$$

is spectral radius of the matrix M with elements $m_{1,k} = I_{i-k}$ $(k=1(1)n)$, where I_{i-k} is the amoung of informations required at the point x_{i-k}, $m_{j,j-1}=1$ $(j=2(1)n)$ and $m_{j,k}=0$ otherwise. The order of a multi-step method

$$\Phi = \phi_1 * \phi_2 * \cdots * \phi_s$$

is spectral radius of the matrix

$$M = M_1 M_2 \cdots M_s .$$

In our case, for method (2) we have matrix

$$M_1 = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}$$

with spectral radius $(3+\sqrt{3})/2 \doteq 3.303$, for method (3) matrix

$$M_2 = \begin{bmatrix} 1 & 3 \\ 1 & 0 \end{bmatrix}$$

with spectral radius 3 and for method (4) matrix

$$M = M_1 M_2 = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 9 \\ 1 & 3 \end{bmatrix}$$

with spectral radius $(7+\sqrt{37})/2 \doteq 6.541$

ACKNOWLEDGEMENT

I wisch to thank the Association of Scientific Activities - Skopje for its support.

REFERENCES

[1] D.B.Popovski: *A Method for Solving Equations*, Proc. 15th Yug.Int.Symp. on Computer Technology and Problems of Informatics - Informatica'81, Ljubljana 5-6 October 1981, 1 212.

[2] J.Herzberger: *Über Matrixdarstellungen für Iterationsverfahren bei nichtlinearen Gleichungen*, Computing 12(1974),215-222.

INFORMATICA 1/1983

ALGORITMI ZA ISKANJE REZERVIRANIH BESED

M. GAMS

UDK: 519.688

INSTITUT „JOŽEF STEFAN“, JAMOVA 39, LJUBLJANA

V članku so opisani algoritmi za iskanje rezerviranih besed. Osnovna algoritma sta binarno iskanje in iskanje z razpršenimi tabelami. V članku so opisane izboljšave teh dveh osnovnih algoritmov. Podana je tudi njihova časovna in prostorska analiza.

ALGORITHMS FOR SEARCHING FOR RESERVED WORDS. This article describes algorithms for searching for reserved words. Basic algorithms are binary search and hash search. This article describes basic algorithms with their improvements and provides time and memory requirements analysis.

1. Uvod

V računalniških programih pogosto srečamo rezervirane besede. To so besede, ki imajo posebeh, rezerviran pomen. Take besede srečamo v prevajalnikih, urejevalnikih, operacijskih sistemih itd. Rezervirane besede so del teksta, npr. programa v programskem jeziku, zato morajo računalniški programi vsebovati postopke za iskanje rezerviranih besed. Osnovna algoritma za iskanje rezerviranih besed sta binarno iskanje in iskanje z razpršenimi tabelami. Z izboljšavami teh dveh algoritmov je mogoče doseči precejšne časovne prihranke.

2. Opis algoritmov

Najprej bomo opisali osnovna algoritma in nato njune izboljšave. Algoritme bomo opisovali v Pascalu. Kjer pa bomo prekoračili izrazne možnosti tega jezika, se to posebej označeno. V tem poglavju bomo govorili tudi o kompleksnosti algoritmov. S tem pojmom bomo označili povprečno število primerjanj neznane besede z rezerviranimi.

2.1. Binarno iskanje

Opis algoritma:

```
lower := 1; higher := bottom;
found := false;
while (lower<=higher) and not found do
begin
  middle := (lower + higher) div 2;
  if words[middle] > unknownWord
    then higher := middle
    else if words[middle] < unknownWord
      then lower := middle
      else found := true
end;
```

```
                WORDS
              ------------
lower =>      |AND       |
              |ARRAY     |
              |...       |
              |...       |
              |...       |
middle =>     |...       |
              |...       |
              |...       |
              |...       |
              |...       |
higher =>     |WITH      |
              ------------
```

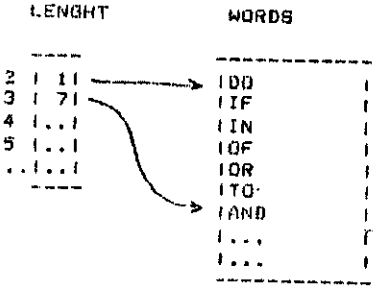
Opombe:

- na večini obstoječih Pascalovih prevajalnikov operatorja "<" in ">" nista implementirana za primerjanje sestavljenih tipov,
- rezervirane besede v tabeli "WORDS" morajo biti urejene po abecednem redu,
- spremenljivki "lower" in "higher" morata biti pred začetkom izvajanja binarnega iskanja prirejeni na vrh, oziroma dno tabele.

Kompleksnost algoritma za binarno iskanje je reda velikosti log(n), kjer je n število rezerviranih besed. Ker je rezerviranih besed običajno nekaj deset, je log(n) < 5 ali 6.

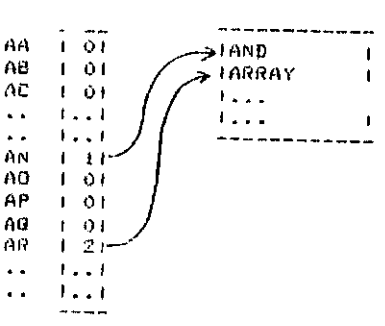
Najbolj običajen trik za pohitritev iskanja je grupiranje rezerviranih besed v skupine. Eden primerov za grupiranje v skupine je po dolžini besed, tako da so znotraj iste skupine besede iste dolžine, urejene po abecednem redu. Algoritem za binarno iskanje je potrebno samo malenkostno spremeniti. Binarno iskanje vršimo v tabeli "WORDS" tako, da

Pred izvajanjem osnovnega algoritma priredimo spremenljivkama "lower" in "higher" vrednosti iz vnaprej generirane tabele, ki vsebuje indekse začetka in konca skupin besed iste dolžine. Na primer, če je "l" dolžina neznane besede, bo "lower := lenght[l]" in "higher := lenght[l+1] - 1". Primer za jezik Pascal je shematično prikazan malo naprej.



Za realne primere se kompleksnost algoritma približno dvakrat zmanjša in je med 2 in 3.

Drug način grupiranja je po prvi ali po prvih dveh črkah. V prvem primeru je tabela za določanje indeksov v tabeli WORDS razdeljena na 26 in v drugem na 676 delov. Dodatno potrošimo 26 ali 676 celic spomina, iskanje neznane besede pa ima kompleksnost med 2 in 1. Možne so še razne kombinacije, npr. grupiranje po dolžini in po prvi črki itd.



2.2. Iskanje z razpršenimi tabelami

Osnovni algoritem iskanja z razpršenimi tabelami temelji na funkciji "h", ki preslika v našem primeru besede v indeks tabele. Kompleksnost takih algoritmov je običajno kar konstanta. Kompleksnost je 1 kadar je funkcija h injektivna (h(w1) <> h(w2)). Običajno imajo razpršitvene funkcije lastnosti, ki so v povprečju blizu injektivnosti, tako da se možno približajo kompleksnosti 1. Druga pomembna lastnost razpršitvenih funkcij je to, da običajno razpršijo podatke v precej večjo tabelo, oziroma da je v tabeli precej praznih mest. Odtod tudi ime "razpršene tabele".

Kadar je razpršitvena funkcija injektivna, navadno rečemo, da je "popolna" (perfect). Kadar je razpršitvena funkcija surjektivna, rečemo, da je "minimalna". Minimalna razpršitvena funkcija ima v našem primeru tabelo "WORDS" polno zasedeno.

V zadnjih letih je nekaj odkritij povečalo zanimanje za minimalne ali skoraj minimalne popolne razpršitvene funkcije [1,2,3]. V [1] je opisana minimalna razpršitvena funkcija, implementirana na rezerviranih besedah jezika Pascal. Razpršitvena funkcija je oblike

```
h(Word) = value[Word[1]] + lenght +
          value[Word[lenght]]
```

"lenght" je dolžina neznane besede "Word"
"Word[i]" je i-ta črka besede "Word"
"value[letter]" je izbrana vrednost črke "letter"

Obitno funkcija ne more biti popolna, kadar obstajata dve rezervirani besedi enake dolžine, ki imata paroma enaki prvi in zadnji črki. V [2] je pokazano, da obstaja še več drugačnih protiprimerov, torej postopek iskanja razpršitvenih funkcij na zgornji način ne jamči rešitve. V [3] je omenjena metoda, ki nima te slabosti, vendar je tako zgenerirana funkcija časovno zahtevnejša.

Problem iskanja razpršitvene funkcije je problem prirejanja vrednosti posameznim črkam. Denimo, da imamo samo 20 različnih začetnih in končnih črk rezerviranih besed, ter da vsaki lahko priredimo največ 20 različnih vrednosti. Kompleksnost preprostega algoritma, ki preračuna vse variante, je 20**20 (dvajset na dvajset), torej praktično nezvedljivo. Napišimo sa:

```
repeat
  if overlap then letter := pred(letter)
    else begin letter := succ(letter);
               value[letter]:= -1
    end;
  repeat
    value[letter] := value[letter] + 1;
    checkOverlapping(overlap,value,words)
  until not overlap or
        (value[letter])>maxValue)
until not overlap and (letter = 'z')
```

Osnovni algoritem lahko pohitrimo z raznimi triki. Namesto zaporednega izbiranja črk začnemo z najbolj pogostimi. Nadaljne izboljšave niso objavljene v literaturi, čeprav so v praksi najbrž uporabljane. Najbolj učinkovito pohitritev imenujem "superbacktrack". Kadar se dve rezervirani besedi prekrivata tako, da jo ne moremo ločiti s spreminjanjem vrednosti tekoče črke, moramo spremeniti vrednost črke, ki nastopa v prirejanju in smo ji zadnji določili vrednost. Poglejmo si primer:

```
value[letter]
---------
e |  0
r |  1
t |  4
f |  ?   <== tekoča črka

w1 = for
w2 = file

h(w1)= value[f] + 3 + value[r] = value[f] + 4
h(w2)= value[f] + 4 + value[e] = value[f] + 4
```

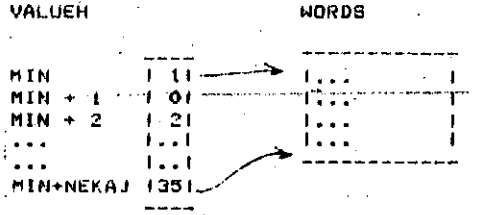
"Superbacktrack" v takem primeru poveča vrednost črke "r" za 1.

Avtor je opazil še en koristen recept. Ko zasledujemo izvajanje algoritma, opazimo, da so nekatera zaporedja črk zelo neprijetna. Takrat program spreminja vrednosti črk korak za korakom brez "superbacktracka". Takrat je najbolje prekiniti izvajanje in spremeniti vrstni red prirejanja vrednosti črkam.

Tako je za generiranje minimalne popolne razpršitvene funkcije za Pascalove rezervirane besede avtorjev program potrošil le nekaj sekund. Poglejmo si primer, ko vnaprej izberemo najmanjšo možno vrednost razpršitvene funkcije 3:

	VALUE	WORDS
1	(E) 0	
2	(P) 0	
3	(D) 0	END
4	(T) 1	ELSE
5	(N) 3	TYPE
6	(O) 8	PACKED
7	(R) 6	NOT
8	(F) 11	THEN
9	(L) 11	PROCEDURE
10	(A) 15	DO
11	(C) 19	TO
12	(I) 19	RECORD
13	(M) 23	REPEAT
14	(V) 25	DOWNTO
15	(W) 14	FILE
16	(B) 21	OR
17	(G) 19	NIL
18	(H) 15	AND
19	(S) 31	WHILE
20	(U) 20	FOR
21	(Y) 17	OF
22	(J) 0	FUNCTION
23	(K) 0	CASE
24	(Q) 0	IN
25	(X) 0	CONST
26	(Z) 0	MOD
27	------	LABEL
28		DIV
29		BEGIN
30		PROGRAM
31		GOTO
32		IF
33		WITH
34		VAR
35		SET
36		UNTIL
37		ARRAY

Za Pascalove rezervirane besede znamo enostavno in hitro določiti minimalno popolno razpršitveno funkcijo. Kaj pa če imamo v rezerviranih besedah tudi besede kot "odr" in "car" ali "fi" in "if"? Očitno moramo izbrati drugačno obliko razpršitvene funkcije. Nato poskusimo generirati tabelo "value". Ker nam postopek ne jamči rešitve, prav lahko poskušamo zaman. V takih primerih je smiselno najprej iskati popolno razpršitveno funkcijo, ki ni minimalna. Tabela "WORDS" je zdaj dimenzije N + NEKAJ, kjer je N število rezerviranih besed. Ko najdemo eno rešitev, lahko zmanjšujemo "NEKAJ", dokler še najdemo rešitev. Da ne bi trošili preveč spomina, si pomagamo še z naslednjim trikom:



Tako potrošimo še N + NEKAJ celic spomina.

3. Podrobnejša časovna in prostorska analiza algoritmov

Do sedaj smo govorili le o povprečnem številu potrebnih primerjanj neznane besede z rezerviranimi. To smo označili s "kompleksnostjo algoritma". Sledi podrobnejša analiza algoritmov. Za eno operacijo bomo šteli eno primerjanje, eno prirejanje, eno računsko operacijo (+,*..), en dostop polja itd. Te operacije časovno niso enakovredne, zato so ocene okvirne in odvisne od računalnika. Privzeli bomo še, da en znak zavzema eno besedo spomina, da je rezervirana beseda sestavljena iz 10 znakov in da je vseh rezerviranih besed 32.

Ocenimo število osnovnih operacij za primerjanje besed med seboj:

```
while (a[i]=b[i]) and (i<dimWord) do
        i := i + 1;
```

Imamo:

```
        1 prirejanje (:=)
        1 rač.op.    (+)
        1 log.op.    (and)
        2 primerjanji (=,<)
        2 dostopa polja ([i])
        +1 za kontrolni konstrukt "while"
       ---------------------------------
        8 osnovnih operacij
```

Za enaki besedi načeloma porabimo toliko primerjanj kot je dimenzija besede (maksimalna dovoljena dolžina). Za različni besedi porabimo bistveno manj primerjanj, v povprečju bomo za naše ocene vzeli 1,5.

A - neznana beseda je rezervirana
B - neznana beseda ni rezervirana

ALGORITEM	ŠTEVILO OPERACIJ A	ŠTEVILO OPERACIJ B	CELIC SPOMINA
binarno	180	125	320
+ po dolžini	140	60	330
+ po prvi črki	120	55	346
+ po prvi čr. + dolž.	80	20	580
+ po prvih dveh črkah	55	10	896
minim.pop. razpr. f.	55	12	346
popolna razpr. f.	57	14	385

Opombe:

- ocene so okvirne, ker so zelo odvisne od oblike rezerviranih besed itd.

- konkretne izvedbe algoritmov lahko malo pohitrimo z enostavnimi triki, ki so v časovni analizi upoštevani. Npr. kadar v povprečju dostikrat prirejamo enake besede, bomo primerjanje naredili takole:

```
while (unKnownWord[i] = WORDS[indeks,i]) and
      (WORDS[index,i] <> ' ') do
        i := i + 1;
```

Tako primerjanje precej pohitrimo.

- v oceni je bila privzeta taka oblika razpršitvene funkcije kot za Pascalove rezervirane besede.

- v praktični uporabi algoritmov, npr. pri prevajalnikih, odpade večji del potrošenega časa na branje, tako da so prihranki manjši.

Praktični rezultati:

Skupina študentov (M.Bradeško, B.Oremuž, M.Hrvatin, T.Kokalj) je merila razmerje med osnovnim binarnim iskanjem in popolno razpršitveno funkcijo za Pascalove rezervirane besede. Razmerje se je sukalo od 1:6 do 1:9 glede na način generiranja testnih besed od samih rezerviranih do samih nerezerviranih. To razmerje ustreza teoretično izračunanemu.

Rad bi se zahvalil tudi I. Mozetiču in N. Lavrač za konstruktivne pripombe.

Zaključek analize:

Natančna analiza algoritmov za iskanje rezerviranih besed da določeno prednost popolnim minimalnim ali skoraj minimalnim razpršitvenim funkcijam. Algoritem z razdelitvijo rezerviranih besed po prvih dveh črkah je časovno celo malenkost hitrejši, vendar troši bistveno več spominskega prostora. Popolne razpršitvene funkcije imajo le to dodatno neprijetnost, da moramo najti obliko razpršitvene funkcije inda moramo prirediti vrednosti črkam. To pa je običajno delo nekaj dni.

4. Zaključek

Praktični in teoretični rezultati dajejo prednost uporabi algoritmov z minimalnimi ali skoraj minimalnimi razpršitvenimi funkcijami za iskanje rezerviranih besed. Sama uporaba razpršitvene funkcije je 5 do 10 krat hitrejša od osnovnega binarnega iskanja. Zaradi podaljšanega branja je v praktičnih primerih prihranek precej manjši, vendar še vedno upoštevanja vreden. Zato je smiselno pohitriti obstoječo programsko opremo s popolno razpršitveno funkcijo.

Literatura:

1. R.J.Cichelli: Minimal Perfect Hash Functions made simple, CACM, Vol.23, Num.1, Januar 1980, str. 17-19.

2. G.Jaeschke, G.Osterburg: On Cichelli's Minimal Perfect Hash Function Method, CACM, Vol.23, Num.12, December 1980, str. 728-729.

3. G. Jaeschke: Reciprocal Hashing: A Method For Generating Minimal Perfect Hashing Functions, CACM, Vol.24, Num.12, December 1981, str. 829-833.

AUTOMATSKO UPRAVLJANJE LETEĆOM TESTEROM UVODJENJEM JEDNOSMERNOG MOTORA I MIKRORAČUNARA

MIROSLAV S. MILIĆEVIĆ
FABRIKA PROFILA, ALEKSINAC

UDK;621.931–52:681.3

U radu se projektuje sistem za automatsko upravljanje letećom testerom kod odsecanja profila u pokretu. Za glavni pogon se uvodi jednosmerni motor, reduktor, zupčanik i zupčasta letva koja je kruto povezana sa testerom. Regulacija brzine se izvodi trofaznim tiristorskim regulatorom koji napaja jednosmerni motor. Za automatizaciju procesa projektuje se mikroračunar na bazi osmobitnog mikroprocesora 8080. Na osnovu projektovanog hardware-a mikroračunara uradjena je software-ska podrška. Zadatak novoprojektovanog sistema je povećanje tačnosti odsecanja i podizanje nivoa automatizacije.

AUTOMATIC CONTROL FOR THE FLYING CUT-OFF DEVICE BY THE INTRODUCTION OF THE DIRECT-CURRENT MOTOR AND MICROCOMPUTER. In the paper there is projected the system for automatic regulation of the flying cut-off device for the moving profile. For the main drive there is introducted the direct-current motor, reductor, gear and gear-lathe which is rigidly connected with cut-off device. The speed is regulated by the three-phase thyristor regulator which supplies the DC motor. For the process automation there is projected a microcomputer on the base of 8080 8 bit microprocessor. On the base of the projected hardware microcomputer there is made a software support. The task for the newly projected system is better accuracy of the cutting and increasing of the automation level.

1. UVOD

Izrada čeličnih profila se obavlja na tehnološkim linijama za proizvodnju profila. Postupak izrade započinje tako što se čelična traka oblikuje valjcima za formiranje. Iza formiranja profili se uzdužno zavaruju generatorom. Zavarivanje je induktivno a generator osciluje na radiofrekvencijama. Posle zavarivanja profili se kreću preko preloznog stola gde se var normalizuje. Na sekciji za kalibrisanje se vrši dodatno oblikovanje profila valjcima za kalibrisanje. Proizvodnja profila je kontinualna a pogon je izveden preko jednosmernih motora gde se regulacija brzine obavlja tiristorskim regulatorima.

Proizvodnja profila se završava odsecanjem u pokretu, na unapred odabranu dužinu, što se realizuje testerom za odsecanje. Postojeći sistem za odsecanje u pokretu se sastoji od elektronskog upravljanja, hidrauličnih komponenti i potrebne mehaničke op-

reme. Odsecanje se obavlja tako što je rezni alat kruto povezan sa hidrauličnim cilindrom, čijom se translacijom upravlja promenom protoka fluida što ima za posledicu različite brzine. Sam trenutak odsecanja nastaje kada se generiše signal ventilu za aktiviranje hidrauličnog cilindra za odsecanje.

Rad sistema za odsecanje u pokretu se odlikuje posedovanjem radnog ciklusa. Radni ciklus se sastoji od ubrzavanja, sinhronizacije sa brzinom proizvoda, odsecanja i vraćanja reznog alata u početni mirni položaj, odakle startuje nov radni ciklus nailaskom potrebne komande. Pri samom odsecanju potrebno je da se sinhronišu brzine testere i profila, kada se ima rez pod pravim uglom. Tačnost dužina odsečenih profila zavisi od vremena preuranjenja polaska. Ovo vreme je takvo, da se generiše komanda za polazak pre anuliranja brojača, da bi se u trenutku sinhronizacije brzina, rezni alat našao upravo iznad teorijske tačke za odsecanje.

Odsecanje profila u pokretu je razradjivano u [1,...,5] . Cilj je bio povećati kvalitet odsecanja gde dolaze ugao odsecanja i izabrana dužina za odsecanje. Razvojem ovakvih sistema težilo se povećanju produktivnosti proizvodnje profila. Na postojećem sistemu za odsecanje primećeni su sledeći nedostaci: netačnost kola za preuranjeni polazak, nelinearnost frekventno naponskog konvertora i rad sistema u otvorenoj povratnoj sprezi.

Za upravljanje translacijom reznog alata su instalirani servo ventil i servo pumpa promenljivog protoka. Često puta, zbog problema nečistoće fluida, dolazilo je do zastoja i smanjenja produktivnosti. Problem temperaturno različitih režima se nepovoljno odražavao na rad sistema. Zbog navedenih nedostataka, koji su potvrdjeni u praksi na sistemu, dolazilo je do većih zastoja u proizvodnji a dužine i ugao odsecanja, često puta, nisu bili u okviru dozvoljenih tolerancija.

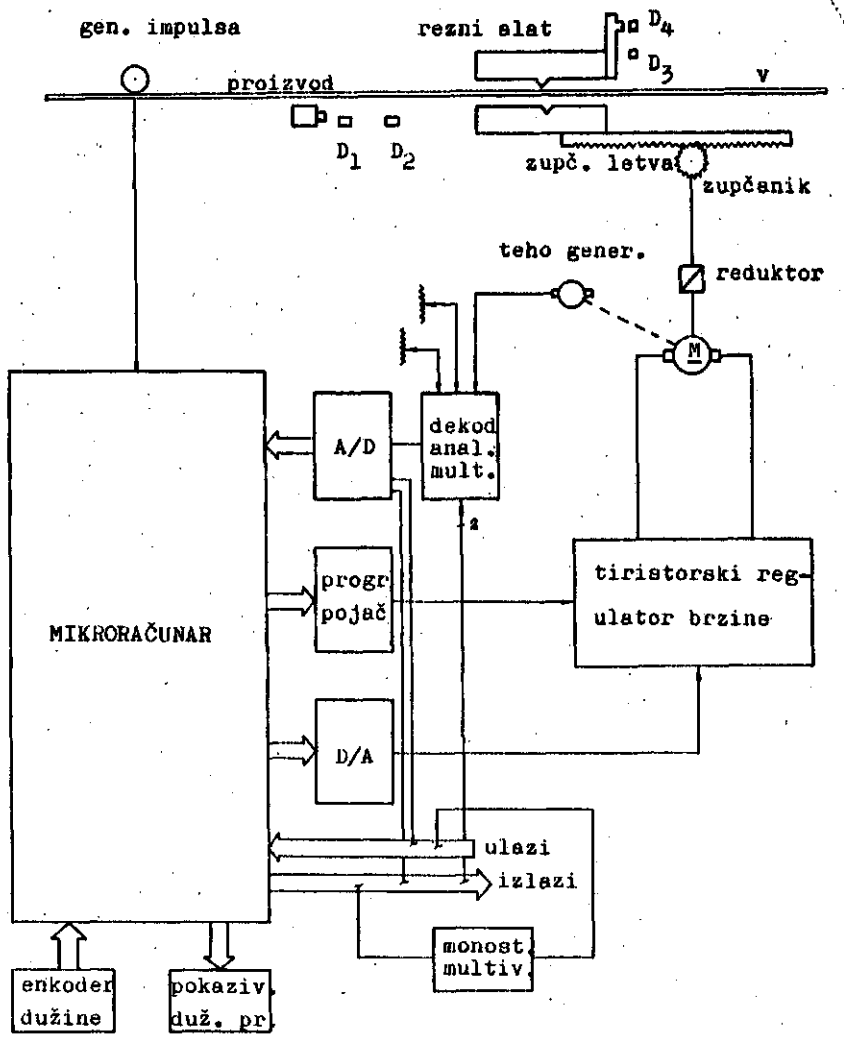
Iz napred navedenih razloga prišlo se projektovanju novog sistema automatskog upravljanja koji će imati za cilj da poboljša rad i poveća produktivnost tehnoloških linija. Umesto postojeće elektronike projektuje se mikroračunar koji se bazira na mikroprocesoru 8080. Servo ventil, servo pumpa i hidraulični cilindar, za translaciju, se zamenjuju jednosmernim motorom sa odvojenom pobudom. Regulacija brzine jednosmernog motora se izvodi trofaznim punoupravljivim tiristorskim regulatorom. Zbog promene smera uvodi se dvostruki, antiparalelno spojen, tiristorski regulator. Osovina motora se spaja reduktorom čiji izlaz ide na zupčanik koji pokreće zupčastu letvu a za koju je kruto povezan sistem za odsecanje. Merenjem brzine motora omogućava se uvodjenje povratne sprege, što sve ima za zadatak da se adaptivno upravlja brzinom motora u odnosu na brzinu proizvoda. Cilj novoprojektovanog sistema je da poveća tačnost i produktivnost rada celog sistema za odsecanje u pokretu.

2. OPIS NOVOPROJEKTOVANOG SISTEMA

Zbog poboljšanja odsecanja u pokretu, na tehnološkim linijama za proizvodnju čeličnih profila, uvodi se nov sistem automatskog upravljanja dat na sl.1. Za pogon reznog alata uvodi se zupčasta letva, zupčanik, reduktor i jednosmerni motor. Zupčasta letva je kruto vezana za rezni alat, tako da zajedno transliraju u smeru napred ili nazad. Jednosmerni motor se napaja iz antiparalelnog punoupravljivog trofaznog tiristorskog regulatora brzine. Pobuda motora je konstantna, čime se poseduje konstantni moment.

Položaj reznog alata se indicira graničnim beskontaktnim prekidačima D_1, D_2, D_3 i D_4. Brzina reznog alata je u direktnoj srazmernosti sa brzinom motora, i ona se meri tahogeneratorom. Brzina proizvoda, koji se odseca, meri se optičkim generatorom impulsa. Dužina proizvoda se meri brojanjem impulsa optičkog generatora impulsa.

Za automatsko upravljanje odsecanjem uvodi se i mikroračunar. Mikroračunar se spreže sa procesom preko I/O jedinica, D/A i A/D konvertora. Biranje željene dužine proizvoda za odsecanje se obavlja biračima koji su locirani na pultu. Trenutna dužina se pokazuje na displeju za pokazivanje dužine. Postupak je automatizovan tako da se algoritam izvršava programski. Mikroračunar očitava postavljenu dužinu za odsecanje, meri dužinu i brzinu proizvoda i programski izdaje komande izvršnim organima. Sinhronizacija brzine reznog alata i proizvoda se postiže merenjem brzine motora i uporedjivanjem sa brzinom proizvoda generiše se potrebno upravljanje tiristorskom regulatoru brzine. Radni ciklus započinje ubrzavanjem do referentne vrednosti, zatim se obavlja odsecanje,



sl.1.

zaustavljanje i vraćanje reznog alata na početnu poziciju. Povratna brzina se može izabrati na potenciometru, kao i minimalna čime se ostvaruje potreban pritisak za držanje na odbojnike.

Za pogon ćemo izabrati jednosmerni motor tip SEVER - OIM 250 M-1s čije su karakteristike:

nominalni napon U_n=400V,
nominalna snaga P_n=80kW,
nominalna struja I_n=217A,
napon pobude U_p=200V,
struja pobude I_p=2,8A, (2.1)
nominalna brzina n_n=1560o/m,
otpornost rotora R_a=0,11Ω,
induktivnost rotora L_a=2,7mH i
moment inercije J_m=1,15kgm^2 .

Za redukciju brzine izabraćemo reduktor tip SEVER - Z82, čije su karakteristike:

prenosni odnos i=12,39 ,
moment M=4200 Nm i (2.2)
moment inercije J_r=0,16 kgm^2.

Ako se izabere maksimalna brzina reznog alata, tada prema maksimalnom broju obrtaja, ima se za poluprečnik zupčanika

r = 13,27 cm. (2.3)

Neka je masa reznog alata 2500 kg, tada se dobija za moment inercije tereta

J_t = 43,7 kgm^2 . (2.4)

Svodjenjem momenta inercije tereta dobija se

$$J_o = 0{,}2846 \text{ kgm}^2. \qquad (2.5)$$

Posle sračunavanja za otporni moment tereta se dobija

$$M = 70{,}88 \text{ Nm} . \qquad (2.6)$$

Smenom poznatih dobija se vrednost momenta motora

$$M_m = 489{,}74 \text{ Nm} . \qquad (2.7)$$

Što se momenata tiče izabrani motor odgovara, a u procesu radiće sa momentom ubrzanja

$$M_u = 418{,}86 \text{ Nm}. \qquad (2.8)$$

Ukupni moment inercije jednak je zbiru svedenog momenta inercije tereta na osovinu motora, momenta inercije motora i reduktora, čija je vrednost

$$J = 1{,}56 \text{ kgm}^2 . \qquad (2.9)$$

Da bi odredili prenosnu funkciju jednosmernog motora, potrebno je izračunati konstante motora. Električna konstanta je

$$T_a = \frac{L_a}{R_a} = 24{,}54 \text{ ms} . \qquad (2.10)$$

Na osnovu poznatih relacija dobijaju se konstante

$$k_e = 2{,}3 \frac{V \text{ rad}}{s} \quad \text{i} \qquad (2.11)$$

$$k_m = 2{,}26 \frac{Nm}{A} . \qquad (2.12)$$

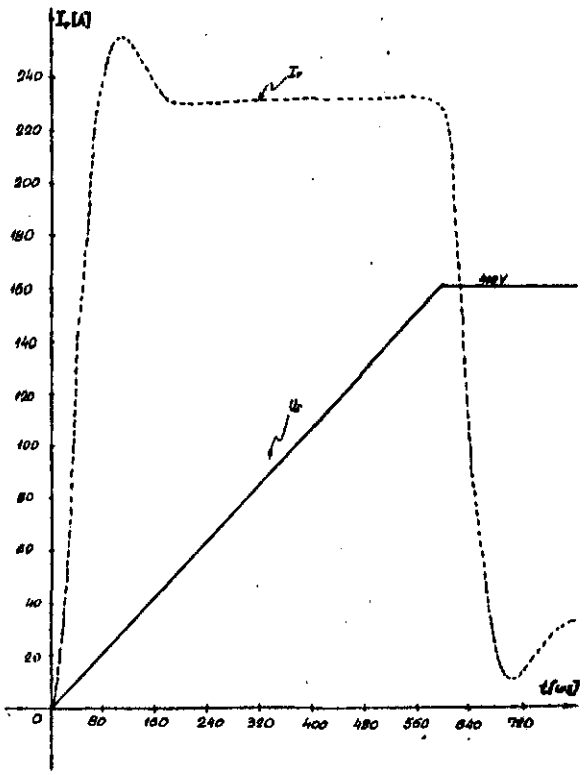
Za mehaničku vremensku konstantu ima se

$$T_m = \frac{JR_a}{k_e k_m} = 33{,}05 \text{ ms}. \qquad (2.13)$$

Kod jednosmernog motora, pri ubrzavanju sa teretom, ograničavajući faktor je struja rotora. Poznata je relacija za struju

$$I_r = U_r \frac{sT_m}{R_a(s^2T_aT_m+sT_m+1)} + \\ + \frac{M_t}{k_m} \frac{1}{s^2T_aT_m+sT_m+1} , \qquad (2.14)$$

gde su I_r struja rotora, U_r napon rotora a M_t moment opterećenja. Ako se napon U_r menja po zakonu, kao na sl.2, tada struja ima oblik



sl.2.

$$i_r = \left[200{,}3(u(t)-u(t-0{,}6))+31{,}4\right] \cdot \\ \cdot \left[1-1{,}227e^{-20{,}37t}\sin(28{,}6t+0{,}95)\right] , \qquad (2.15)$$

čiji grafik je prikazan na sl.2. Sa sl.2 zaključujemo da struja rotora dostiže vrednosti veće od maksimalne, što ukazuje da se mora raditi sa strujnim ograničenjem. Rad sa strujnim ograničenjem dovodi do toga da je brzina motora linearna u vremenu što se pokazuje i relacijom

$$n = \frac{30M_u}{\pi J} . t . \qquad (2.16)$$

Iz relacije (2.16) se može izračunati vreme zaleta motora do nominalne brzine, i ono za naš slučaj iznosi

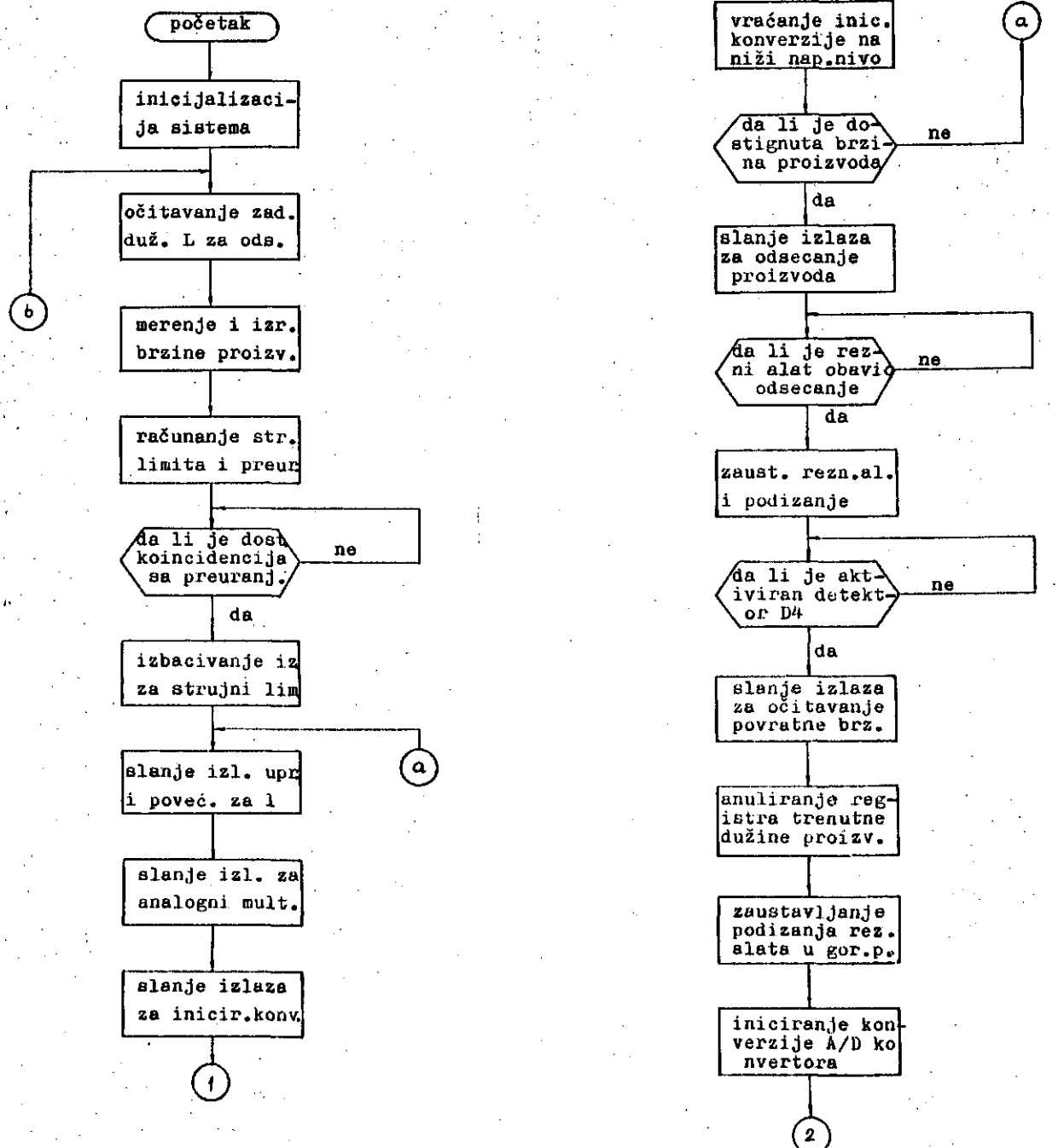
$$t_z = 665 \text{ ms} . \qquad (2.17)$$

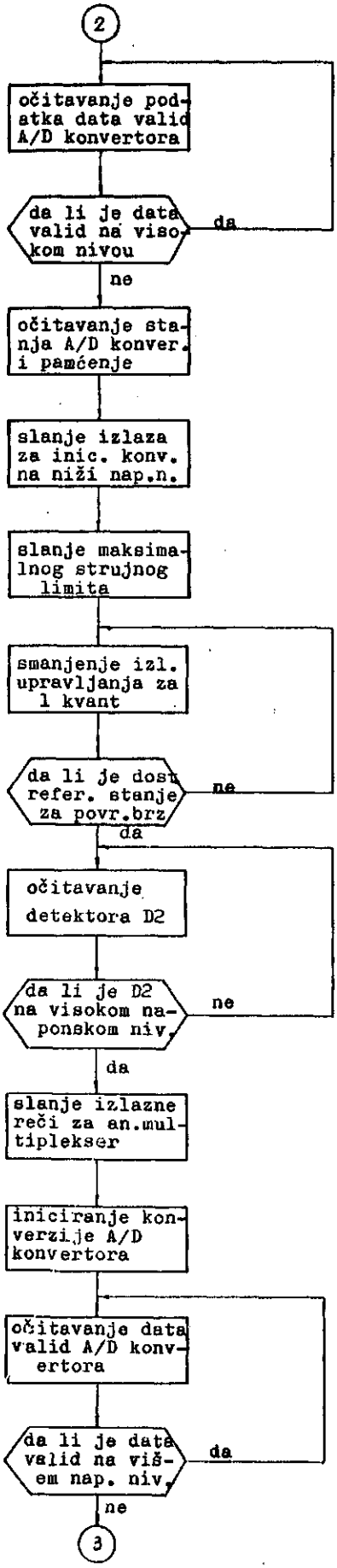
Budući da je vreme zaleta manje od odziva, kada se koristi hidraulični servo sistem, to zaključujemo da izabrani motor odgovara za pogon reznog alata. Za strujno ograničenje se biraju vrednosti proporcionalne brzini proizvoda, tako da se za isto vreme sinhronišu brzine reznog alata i proizvoda. Na ovaj način se dobijaju vrednosti preuranjenja kao linearne zavisnosti brzine, što nije slučaj sa prvobitnim rešenjem.

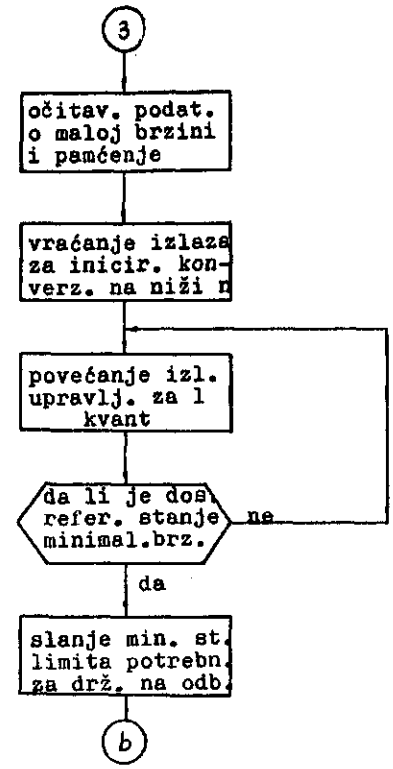
3. PROJEKTOVANJE MIKRORAČUNARA

Da bi se automatizovao sistem odsecanja profila u pokretu potrebno je projektovati mikroračunar. Novoprojektovani sistem sa mikroračunarem je prikazan na sl.1. Koristeći ulazne podatke mikroračunar programski generiše komandne signale vodeći proces odsecanja automatski.

Algoritam rada sistema je prikazan na sl.3.







sl.3.

Opisaćemo blok šemu rada sistema sa sl.3. Na početku se obavlja inicijalizacija sistema. Zatim se očitava zadata dužina proizvoda za odsecanje. Za jedan vremenski interval, odredjen trajanjem kvazistabilnog perioda multivibratora, broje se impulsi optičkog generatora impulsa čiji rezultat predstavlja brzinu proizvoda. Dužina se meri brojanjem impulsa za šta se koristi sistem prekida.

Koristeći se podatkom o brzini, izračunava se strujno ograničenje i vreme preuranjenja. Onda se ispituje da li je došlo do koincidencije sa preuranjenjem, i ako jeste izlazi se iz petlje izbacivanjem izlaza za strujni limit. Nakon ovog koraka uvečava se izlazno upravljanje za 1 kvant. Algoritam se nastavlja slanjem izlaza za analogni multiplekser i za iniciranje konverzije A/D konvertora. Ispitivanjem stanja A/D konvertora, ako je data valid na visokom nivou, ostaje se u petlji, inače se očitava izlaz. Izlaz A/D konvertora predstavlja brzinu motora. Onda se ispituje da li je dostignuta brzina proizvoda, i ako nije ostaje se u petlji, inače algoritam se nastavlja. Sledeći programski korak je slanje izlaza za odsecanje proizvoda, i kada je završeno odsecanje alat se podiže u gornji položaj. Podizanje u gornji položaj prestaje onda kada je aktivi-

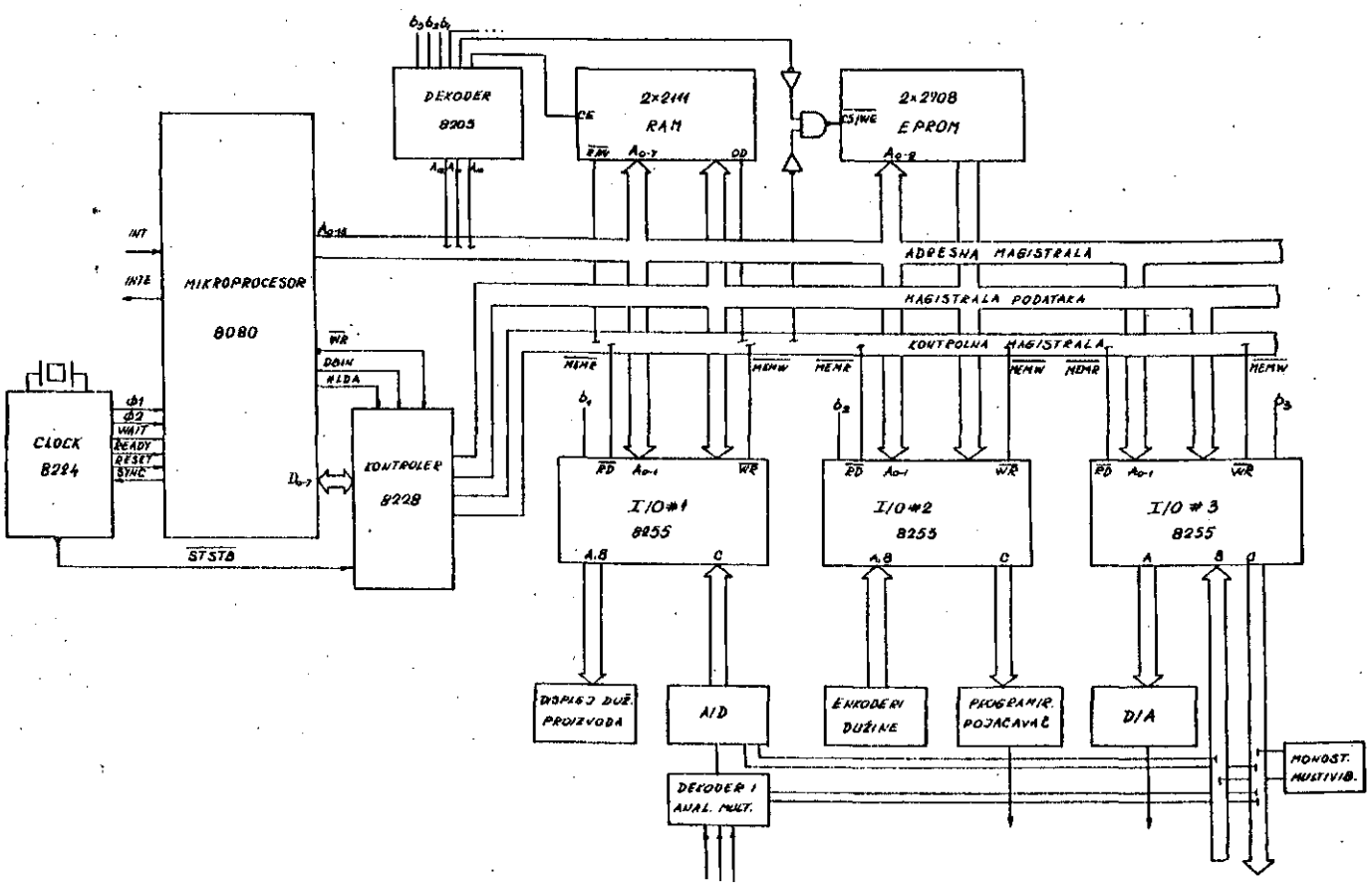
ran detektor D_4.

Po izvršenom odsecanju anulira se registar trenutne dužine proizvoda i šalje se izlaz za očitavanje povratne brzine. Povratna brzina se dobija iniciranjem konverzije A/D konvertora, i po zadovoljenju uslova za očitavanje, očitavanjem izlaza konvertora. Posle ovog očitavanja šalje se maksimalan izlaz za strujno ograničenje.

Izlazno upravljanje se smanjuje za jedan kvant sve dotle dok se ne dostigne referentna vrednost za povratnu brzinu. Zatim se ispituje da li je aktiviran detektor D_2 za usporenje pri povratku reznog alata nazad. Kada je isti aktiviran, izlazi se iz petlje slanjem izlazne reči za analogni multiplekser. Kada je inicirana konverzija, po ispunjenju uslova, očitava se podatak o maloj brzini. Pošto je očitan podatak o maloj brzini, povećava se izlazno upravljanje za jedan kvant sve do dostizanja referentne vrednosti. Onda se šalje minimalan strujni limit potreban za držanje na odbojnike, a algoritam se vraća na početak ponovnim očitavanjem dužine proizvoda za odsecanje.

Mikroračunar koji izvršava programske korake blok šeme sa sl.3, predstavljen je na sl.4. Ovaj mikroračunar se bazira na osmobitnom mikroprocesoru 8080. Za vremensko vodjenje upotrebljava se clock generator 8224. Koristi se i kontroler iz ove familije 8228. Za memorije se upotrebljava EPROM od 2 kbyte-a (2x2708) i RAM memorija od 256 byte-a (2x 2111). Za odabiranje jedinice koristi se dekoder 8205, koji za ulaze uzima sa adresne magistrale i to A_{12}, A_{11} i A_{10}. Sistem poseduje i tri programirajuće ulazno izlazne jedinice tipa 8255 preko kojih se mikroračunar povezuje sa procesom.

Izlazi I/O#1 jedinice (A,B) se vode na displej gde se pokazuje trenutna dužina proizvoda a (C) je ulaz od A/D konvertora. Koristi se jedan A/D konvertor, gde se konvertuju podaci o brzini motora, povratnoj i minimalnoj brzini, što je omogućeno primenom dekodera i analognog multipleksera. Dekoder



sl. 4.

se koristi zbog korišćenja dva ulaza a potrebno je imati bar tri izlaza za uključenje analognog multipleksera preko koga idu analogni podaci za konvertovanje u A/D konvertoru.

Izlazi iz enkodera dužine su priključeni na ulaze (A,B) ulazno izlazne jedinice I/O#2. Izlaz (C) pomenute jedinice se vodi na programirajući pojačavač, čiji izlaz predstavlja strujno ograničenje za tiristorski regulator brzine. Pri većim brzinama proizvoda izlazi programirajućeg pojačavača su veći, tako da se motor, a s njim i rezni alat, ubrzavaju većim ubrzanjem. Idući ka manjim brzinama, smanjuje se ovo ubrzanje, tako da se sinhronizacija brzina postiže za isto vreme. Predjeni put reznog alata, a s njim i preuranjenje polaska, su linearne zavisnosti u funkciji promene brzine proizvoda koji se odseca. Kod prvobitno instaliranog sistema ove zavisnosti su bile nelinearne, što je otežavalo njihovo tačno izračunavanje, a kasnije i praktično izvodjenje.

D/A konvertor se napaja iz (A) od ulazno izlazne jedinice I/O#3, čiji izlaz predstavlja upravljanje za tiristorski regulator brzine. Ostali ulazi u mikroračunar kao što su podaci sa graničnih prekidača, A/D konvertora i monostabilnog multivibratora su predstavljeni ulazom (B) pomenute jedinice. Izlaz (C) ove jedinice, predstavlja izlaznu reč gde su izlazi predstavljeni izlazom za iniciranje konverzije A/D konvertora, monostabilnog multivibratora i za generisanje potrebnih komandi pri odsecanju proizvoda.

Sistem prekida se organizuje počev od adrese ØØØØH, i on ima rezervisanu memoriju sve do adrese ØØ3FH. Potprogram MNOZ, za množenje dva jednobajtna broja sa rezultatom od dva byte-a, smešten je počev od adrese ØØ4ØH. Za pretvaranje brojeva iz BCD koda u binarni sastavljen je potprogram PRETV, i on je smešten počev od adrese ØØ8ØH. Glavni program je smešten sa početnom adresom ØØCØH. EPROM memorija zauzima adrese od ØØØØH - Ø7FFH. RAM memorija ima adrese od Ø8ØØH - Ø8FFH, tako da je stek smešten počev od najveće adrese.

Ulazno izlazne jedinice imaju sledeće adrese:

PORT1 - ØCØØH,
PORT2 - ØCØ1H,
PORT3 - ØCØ2H,
PORT4 - 1ØØØH,
PORT5 - 1ØØ1H,
PORT6 - 1ØØ2H,
PORT7 - 18ØØH,
PORT8 - 18Ø1H i
PORT9 - 18Ø2H.

Na osnovu instrukcionih dijagrama tokova uradjeni su potprogrami i glavni program, u simboličkom jeziku, za mikroračunar i isti su prikazani u tabeli T_1.

```
PORT1   EQU   ØCØØH
PORT2   EQU   ØCØ1H
PORT3   EQU   ØCØ2H
PORT4   EQU   1ØØØH
PORT5   EQU   1ØØ1H
PORT6   EQU   1ØØ2H
PORT7   EQU   18ØØH
PORT8   EQU   18Ø1H
PORT9   EQU   18Ø2H
STEK    EQU   Ø8FFH
REL     EQU   Ø8ØØH
RBR     EQU   Ø8Ø2H
RPR     EQU   Ø8Ø3H
RLI     EQU   Ø8Ø4H
RLP     EQU   Ø8Ø5H
RDU     EQU   Ø8Ø7H
RTR     EQU   Ø8Ø9H
RBRM    EQU   Ø8ØAH
VPOV    EQU   Ø8ØBH
VDR     EQU   Ø8ØCH
CONST   EQU   Ø8ØDH
ADR     EQU   Ø8ØEH
STOR    EQU   Ø8ØFH

        ORG   ØØH
        PUSH  PSW
        PUSH  B
        PUSH  D
        PUSH  H
        JMP   PREK1
        HLT
PREK1:  LHLD  RDU
        INX   H
        SHLD  RDU
        MOV   A,L
        DAA
        MOV   L,A
        MOV   A,H
        ACI   ØH
        DAA
        MOV   H,A
        SHLD  PORT1
        EI
        POP   H
        POP   D
        POP   B
        POP   PSW
        RET

        ORG   4ØH
MNOZ:   MOV   E,L
        MVI   D,ØH
        MOV   A,H
        LXI   H,ØH
        MVI   B,8D
LOOP1:  DAD   H
        RAL
        JNC   LOOP2
        DAD   D
LOOP2:  DCR   B
        JNZ   LOOP1
        RET
```

T_1

```
        ORG   8ØH
PRETV:  MVI   D,4D
        MVI   C,16D
        MVI   B,1ØD
JUMP1:  MVI   A,255D
        SUB   C
        MOV   E,A
        LDA   ADR
        ANA   C
        JZ    JUMP2
        LDA   ADR
        ANA   E
        ADD   B
        STA   ADR
JUMP2:  STC
        CMC
        MOV   A,C
        RAL
        MOV   C,A
        STC
        CMC
        MOV   A,B
        RAL
        MOV   B,A
        DCR   D
        JNZ   JUMP1
        RET

        ORG   CØH
        LXI   H,Ø8ØØH
        LXI   SP,STEK
        MVI   D,FFH
SKOK1:  MVI   M,ØH
        INX   H
        DCR   D
        JNZ   SKOK1
        MVI   A,ØH
        STA   PORT6
        STA   PORT7
        STA   PORT9
PET:    MVI   A,ØH
        STA   RBR
        LHLD  PORT4
        MOV   A,L
        STA   ADR
        CALL  PRETV
        LDA   ADR
        STA   STOR
        MOV   A,H
        STA   ADR
        CALL  PRETV
        LDA   ADR
        MOV   H,A
        MVI   L,1ØØD
        CALL  MNOZ
        MVI   D,ØH
        LDA   STOR
        MOV   E,A
        DAD   D
        SHLD  REL
        MVI   A,4ØH
        STA   PORT9
SKOK2:  LDA   PORT8
        ANI   16D
        JNZ   SKOK2
        LDA   RBR
        INR   A
        STA   RBR
SKOK3:  LDA   PORT8
        ANI   16D
        JNZ   SKOK3
        LDA   PORT8
        ANI   64D
        JNZ   SKOK2
        LDA   RBR
        STA   PORT6
        MOV   L,A
        LDA   CONST
        MOV   H,A
        CALL  MNOZ
        SHLD  RPR
        XCHG
SKOK4:  LHLD  RDU
        DAD   D
        SHLD  RLP
        MOV   B,H
        MOV   C,L
        LHLD  REL
        MOV   A,H
        CMP   B
        JNZ   SKOK4
        MOV   A,L
        CMP   C
        JNZ   SKOK4
SKOK5:  LDA   RTR
        STA   PORT7
        INR   A
        STA   RTR
        MVI   A,Ø1H
        STA   PORT9
SKOK6:  LDA   PORT8
        ANI   2ØH
        JNZ   SKOK6
        LDA   PORT3
        STA   RBRM
        MVI   A,Ø1H
        STA   PORT9
        LDA   RBRM
        MOV   E,A
        LDA   RBR
        CMP   E
        JNZ   SKOK5
        MVI   A,3ØH
        STA   PORT9
SKOK7:  LDA   PORT8
        ANI   Ø4H
        JZ    SKOK7
        MVI   A,Ø8H
        STA   PORT9
SKOK8:  LDA   PORT8
        ANI   Ø8H
        JZ    SKOK8
        MVI   A,ØH
        STA   PORT9
        MVI   L,ØH
        MVI   H,ØH
        SHLD  RDU
        MVI   A,Ø2H
        STA   PORT9
        MVI   A,Ø6H
        STA   PORT9
SKOK9:  LDA   PORT8
        ANI   2ØH
        JNZ   SKOK9
        LDA   PORT3
        STA   VPOV
        MVI   A,ØH
        STA   PORT9
        MVI   A,FFH
        STA   PORT6
SKO1Ø:  LDA   RTR
        DCR   A
        STA   RTR
        STA   PORT7
        MOV   E,A
        LDA   VPOV
        CMP   E
        JNZ   SKO1Ø
SKO11:  LDA   PORT8
        ANI   Ø2H
        JZ    SKO11
        MVI   A,Ø3H
        STA   PORT9
        MVI   A,Ø7H
        STA   PORT9
SKO12:  LDA   PORT8
        ANI   2ØH
        JNZ   SKO12
        LDA   PORT3
        STA   VDR
        MVI   A,ØH
```

T_1 nast.

```
        STA   PORT9
SKOl3:  LDA   RTR
        INR   A
        STA   RTR
        MOV   E,A
        STA   PORT7
        LDA   VDR
        CMP   E
        JNZ   SKOl3
        LDA   VDR
        STA   PORT6
        JMP   PET
        END
```

T_1 nast.

Na osnovu dinamičke analize dolazi se do zaključka, da izabrana konfiguracija sistema sa mikroračunarem odgovara svojoj nameni.

4. ZAKLJUČAK

U radu se projektuje sistem za automatsko upravljanje odsecanjem profila u pokretu. Predlaže se uvodjenje jednosmernog motora i mikroračunara koji će ovaj proces voditi automatski.

Na početku rada se opisuje tehnološki postupak za izradu čeličnih profila. Posebna pažnja se obraća odsecanju profila u pokretu. Analiziraju se nedostaci prvobitno instaliranog sistema kroz davanje opisa radnog ciklusa. Korišćenjem iskustava dolazi se do zaključka da treba projektovati nov sistem čiji bi zadatak bio da se poveća tačnost i pouzdanost rada sistema za odsecanje.

Za pogon leteće testere, a prema podacima procesa, usvaja se jednosmerni motor sa odvojenom pobudom, reduktor, zupčanik i zupčasta letva. Na osnovu konkretnih podataka analizira se strujni režim rada motora pri zaletanju. Dolazi se do zaključka, da pri ubrzavanju tereta, motor mora raditi u režimu sa strujnim ograničenjem. Uvodi se ubrzanje motora proporcionalno brzini proizvoda, tako da se dostizanje brzine vrši za isto vreme. Zbog režima sa strujnim ograničenjem, brzina motora je linearna funkcija po vremenu. Odavde proizilazi da je preuranjenje polaska linearna funkcija koja se na jednostavan način odredjuje a što nije bio slučaj sa ranijim rešenjima.

Radi automatizacije odsecanja na letećoj testeri, projektuje se mikroračunar koji se povezuje sa procesom preko A/D, D/A konvertora i ulazno izlaznih jedinica. Za regulaciju brzine, pored mikroračunara, uvodi se i tiristorski regulator brzine. Brzina motora se očitava i uvodi u računar, a na osnovu merene brzine proizvoda, generiše se upravljanje sa tendencijom da leteća testera se sinhroniše sa brzinom profila koji se odseca.

Za usvojenu konfiguraciju hardware-a mikroračunara, realizovan je software programima napisanim u simboličkom jeziku.

Zadatak novopredloženog sistema je da se leteća testera adaptivno upravlja u odnosu na brzinu proizvoda. Sistem ima te odlike da će se znatno povećati tačnost odseacanja proizvoda i produktivnost pri proizvodnji. Pored ovih performansi, može se reći da se podiže nivo automatizacije na tehnološkim linijama za proizvodnju čeličnih profila.

LITERATURA

[1] Davis R. Maurice G., Electronically controlled pumps, Natinal Conference on Fluid Power, v.XVIII, Chicago, 1964

[2] Nettel D. F., Cut to length devices, Measurement and Control, May 1964,London

[3] Racine Vickers - Armstrongs, Electronically controlled axial piston pumps, RVA Bulletin S 204

[4] Vickers, Electro-Hydraulic Drives for Flying cut-off Applicatins, vol. L792/42, Swindon

[5] The Oilgear Company, Tipical die accelerator system with Digital Control

[6] Wester J. G., Software Design for Microprocessors, Texas Instruments,London

[7] Lilen H., Du microprocesseur au microordinateur, Editions Radio,Paris, 1977

[8] Lilen H., Programation des microprocesseurs, Editions Radio, Paris, 1979

[9] Intel 8080 Microcomputer Systems, User's Manual, 1975

[10] Milićević M., Automatska regulacija snage generatora za induktivno zavarivanje pomoću mikroračunara, Automatika br.5-6, Zagreb, 1981

[11] Milićević M., Adaptivno upravljanje odsecanjem metalnih profila u pokretu primenom mikroračunara, III Jugoslovensko savetovanje "Automatizacija u crnoj metalurgiji", Sarajevo, 1981

[12] Milićević M.,Automatizacija uredjaja za odsecanje profila u pokretu primenom mikroračunara, Automatika br.5-6, Zagreb, 1982

UPORABNI PROGRAMI

Program, ki izlista CP/M dodeljevalne skupine na 8" dikseti

```
**************************************
* Informatica UP 10                  *
* CP/M Allocation Groups Program     *
* marec 1983                         *
* modificiral A. P. Železnikar       *
* sistem CP/M, Iskra Delta PARTNER   *
**************************************
```

1. Področje uporabe

Iz različnih razlogov želimo večkrat vedeti, kakšna je razporeditev podatkov na disketi v CP/M sistemu. Pri sistemu CP/M 2.2 sta prvi dve stezi rezervirani za operacijski sistem, vendar lahko s primernim znanjem tudi to organizacijo spremenimo. Ostale steze na disketi so na razpolago uporabniku.

Osemcolska disketa (takšna kot jo dobimo od uporabniške skupine ali od prodajalca programske opreme) ima 26 sektorjev na posamezni stezi in 77 stez na eni strani. Na teh disketah se uporablja tkim. preskočni faktor za sosednje logične sektorje, kar povzroči razliko med fizičnimi in logičnimi sektorji na posamezni stezi. Logični sektorji se namreč dodeljujejo fizično v naslednjem zaporedju:

1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21,
2, 8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22.

Pri tem je osem sosednjih logičnih sektorjev povezanih v tkim. dodeljevalno skupino, v kateri se tako nahaja 8 X 128 = 1024 zlogov.

Dodeljevalne skupine začnejo na stezi 2, vsaka skupina ima po osem logičnih sektorjev, tako da lahko pišemo vobče za skupino

$$G = 0; X, \quad X = (0, 1, \ldots, 7)$$

ki obsega sektorje steze T = 2 s fizičnimi sektorji 1, 7, 13, 19, 25, 5, 11, 17. Skupina 3 se nahaja na dveh stezah, in sicer na T = 2 s sektorjema 16 in 22 in na stezi T = 3 s preostalimi sektorji (1, 7, 13, 19, 25, 5). X označuje logične sektorje v posamezni dodeljevalni skupini.

Skočni faktor za sosednje fizične sektorje se uporablja zaradi časovne rezerve, ki je potrebna pri posameznih sistemih med dvema sektorjema, da bi se lahko izvajal vpis na disketo in branje iz nje.

Na listi 1 je prikazan izpis neposredno iz diskete, ko imamo primer dveh dodeljevalnih skupin, ene iz imenika (directory) in druge z ASCII podatki. Prikazani primer kaže npr. zbirko LISTGRPS.BAK (to je prav tekst tega uporabnega programa) v imeniku in začetek tega teksta

```
i*JD
G=00:01, T=2, S=2, PS=7                    število sektorjev
00  00525546 20202020  20434F4D 0000005C  *.RUN      COM...\*
10  15161718 191A1B1C  1D1E1F20 00000000  *.......... .... *
20  004C4953 54475250  5342414B 000000[12] *.LISTGRPSBAK....*
30  [21252700] 00000000  00000000 00000000  *!%'.............*
40  E54C4953 54475250  5342414B 00000009  *ELISTGRPSBAK....*
50  22260000 00000000  00000000 00000000  *"&.............*
60  004C4953 54475250  53424153 00000009  *.LISTGRPSBAS....*
70  23240000 00000000  00000000 00000000  *#$.............*
```

→ta skupina ima zasedena le dva sektorja

```
i*JD
G=00:02, T=2, S=3, PS=13
00  004C4953 54475250  53444F43 0000001D  *.LISTGRPSDOC....*
10  28292A2B 00000000  00000000 00000000  *()*+............*
20  E5E5E5E5 E5E5E5E5  E5E5E5E5 E5E5E5E5  *EEEEEEEEEEEEEEEE*
30  E5E5E5E5 E5E5E5E5  E5E5E5E5 E5E5E5E5  *EEEEEEEEEEEEEEEE*
40  E5E5E5E5 E5E5E5E5  E5E5E5E5 E5E5E5E5  *EEEEEEEEEEEEEEEE*
50  E5E5E5E5 E5E5E5E5  E5E5E5E5 E5E5E5E5  *EEEEEEEEEEEEEEEE*
60  E5E5E5E5 E5E5E5E5  E5E5E5E5 E5E5E5E5  *EEEEEEEEEEEEEEEE*
70  E5E5E5E5 E5E5E5E5  E5E5E5E5 E5E5E5E5  *EEEEEEEEEEEEEEEE*
```

Sektor: 128 zlogov

```
i*JD
G=01:00, T=12, S=5, PS=25
00  20202020 2050726F  67726168 20206B69  *     PROGRAMI, KI*
10  20697A6C 69737461  2043502F 4D206A6F  * IZLISTA CP/M DO*
20  64656C6A 65766A6C  6E650D0A 20202020  *DELJEVALNE..    *
30  20202020 20202020  20736B75 706E6E65  *        SKUPINE*
40  206E6120 38222064  696D7365 74690D0A  * NA 8" DISKETI..*
50  0D0A0D0A 2A2A2A2A  2A2A2A2A 2A2A2A2A  *....**********
60  2A2A2A2A 2A2A2A2A  2A2A2A2A 2A2A2A2A  **********
70  2A2A2A2A 2A2A2A2A  0D0A2A20 496E666F  **********..* INFO*
```

Skupina 21 ima 8 sektorjev : 1024 zlogov

```
i*JD
G=01:01, T=12, S=6, PS=5   [1A]je znak konca ASCII zbirke
00  7261c174 69636120  55504154 30202020  *IMATICA UP 10   *
10  20202020 20202020  20202020 202A0D0A  *              *..*
20  2A204350 2F4D2041  6C6C6F63 6174696F  ** CP/M ALLOCATIO*
30  6E204772 6F757073  2050726F 6772616D  *N GROUPS PROGRAM*
40  20202020 0D0A2A20  6D617265 63203139  *    *..* MAREC 19*
50  38332020 20202020  20202020 20202020  *83              *
60  20202020 20202020  202A0D0A 2A206D6F  *          *..* MO*
70  64696669 63697261  6C204128 2E502E20  *DIFICIRAL A. P. *
```

Lista 1. Prikazana lista je dobljena s posebnim programom, ki zmore brati posamezne sektorje iz steze diskete.

Na levi listi smo odčitali dva sektorja iz tkim. CP/M imenika in dva zaporedna sektorja s podatki zbirke v imeniku.

V imeniku se nahaja med drugim tudi zbirka LISTGRPS.BAK (vrstica 30 prvega sektorja v listi), ki zaseda skupine 21, 25 in 27H na disketi, kot je razvidno iz vrstice 40 prvega sektorja liste.

Skupina 21, in sicer njena prva dva sektorja, je prikazana s sektorjema 3 in 4 v levi listi. Tu je shranjen prav naslov tega prispevka in v nadaljnih skupinah celoten tekst. Vsi znaki so pri tem napisani z velikimi črkami (teleprinter).

Drugi sektor liste je zaseden v imeniku samo z eno zbirko, in sicer z LISTGRPS.DOC; ostale tri pozicije so še prazne (vrstice 20 do 70 drugega sektorja v listi). Te vrstice so napolnjene z znakom 'e5'.

Zbirka LISTGRPS.BAK v vrstici 40 prvega sektorja liste pa je neveljavna (dejansko je bila zbrisana) in njen začetni znak v vrstici 40 je 'e5' namesto '00' pri veljavni zbirki. Lista nazorno prikazuje del organizacije podatkov na CP/M disketi.

v dodeljevalni skupini 21H. Podatek za začetek te skupine smo prebrali iz imenika (skupina G = 0, logični sektor = 2, pozicija = 30H v listi 1).

2. Kratek opis programa

Na listi 2 imamo program za listanje tabele, v kateri se nahajajo podatki o dodeljevalni skupini in pripadajoči stezi/stezah, logičnih in fizičnih sektorjih. Dodeljevalne skupine v listi 1 se navajajo v heksadecimalni obliki, v listi 2 pa jih izpisujemo decimalno.

V listi 2 imamo v vrstici 20 for zanko za read stavek s podatki v vrsticah 260 in 265; s tem se zmanjša dolžina programa. Preostali del programa izračunava formulo za dodeljevalne skupine, steze, logične in fizične sektorje. Program je napisan v jeziku CBASIC2 oziroma EBASIC.

3. Izvajanje programa

V listi 3 je prikazano izvajanje programa z liste 2. Izpis se ujema z osnovnim izpisom (brez listanja sektorja) v listi 1. Ta lista kaže tako prostorsko organizacijo diskete v sistemu CP/M 2.2. Seveda lahko to organizacijo po želji spremenimo (vobče bo organizacija drugačna pri dvostranski, 5" disketi in pri vinčestrskem disku, ko bodo npr. sektorji daljši, dodeljevalne skupine spremenjene itd.). V spremenjenih razmerah bomo brez težav spremenili program v listi 2.

```
A>TYPE LISTGRPS.BAS
1 REM TA PROGRAM IZLISTA PRIREDITVE STEZ IN
2 REM     SEKTORJEV POSAMEZNIM DODELITVENIM
3 REM     SKUPINAM NA STANDARDNI CP/M DISKETI.
4 REM     VREDNOSTI SO DECIMALNE.
5 REM PRI IZPISU Z VRSTICNIM TISKALNIKOM SE
6 REM     USTREZNO SPREMENIJO VRSTICE 65, 66,
7 REM     67, 90, 150 IN 151 (LPRINTER).
8 REM PROGRAM JE NAPISAN V JEZIKU CBASIC2.
9 REM
10 DIM D(104)
20 FOR I=1 TO 104
30 READ D(I)
35 IF I-26=0 OR I-52=0 OR I-78=0 THEN RESTORE
40 NEXT I
50 G=0
60 T=2
65 PRINT "G - CP/M ALLOCATION GROUP, T - TRACK,"
66 PRINT "LS - LOGICAL SECTOR, "
67 PRINT "PS - PHYSICAL SECTOR"
68 LS=1
70 B=1
80 C=B+7
90 PRINT: PRINT "G = ";
91 S=1
100 FOR I=B TO C
110 IF I<=104 THEN A=3
120 IF I<=78 THEN A=2
130 IF I<=52 THEN A=1
140 IF I<=26 THEN A=0
150 PRINT TAB(5);G;":";S-1;"  T = ";T+A;
151 PRINT "  LS = "; LS;"  PS = ";D(I)
152 S=S+1: LS=LS+1
153 IF LS=27 THEN LS=1
160 NEXT I
170 IF C=104 THEN GOTO 220
180 B=B+8
190 C=C+8
200 G=G+1
210 GOTO 90
220 T=T+4
230 IF T>76 THEN GOTO 300
240 G=G+1
250 GOTO 70
260 DATA 1,7,13,19,25,5,11,17,23,3,9,15,21
265 DATA 2,8,14,20,26,6,12,16,24,4,10,16,22
300 END
```

Lista 2. Ta kratek program, ki je napisan v jeziku CBASIC2, izlista tabelo dodeljevalnih skupin, stez na disketi, logičnih in fizičnih sektorjev, kot je razvidno iz liste 3. Pri nekaterih raziskavah podatkov na disku ali disketi, je taka tabela zelo priročna. Program lahko modificiramo v primeru drugačne organizacije podatkov na disketi, tako da ustrezno spremenimo parametre v programu leve liste. Tako lahko dobimo popolnoma drugačno tabelo npr. za organizacijo podatkov na vinčestrskem disku ali na dvostranskih disketah z dvojno gostoto zapisa.

```
B>CRUN2 A:LISTGRPS

CRUN VER 2.07P
G - CP/M ALLOCATION GROUP, T - TRACK,
LS - LOGICAL SECTOR, PS - PHYSICAL SECTOR

G =  0 : 0   T =  2   LS =  1   PS =  1
     0 : 1   T =  2   LS =  2   PS =  7
     0 : 2   T =  2   LS =  3   PS = 13
     0 : 3   T =  2   LS =  4   PS = 19
     0 : 4   T =  2   LS =  5   PS = 25
     0 : 5   T =  2   LS =  6   PS =  5
     0 : 6   T =  2   LS =  7   PS = 11
     0 : 7   T =  2   LS =  8   PS = 17

G =  1 : 0   T =  2   LS =  9   PS = 23
     1 : 1   T =  2   LS = 10   PS =  3
     1 : 2   T =  2   LS = 11   PS =  9
     1 : 3   T =  2   LS = 12   PS = 15
     1 : 4   T =  2   LS = 13   PS = 21
     1 : 5   T =  2   LS = 14   PS =  2
     1 : 6   T =  2   LS = 15   PS =  8
     1 : 7   T =  2   LS = 16   PS = 14

G =  2 : 0   T =  2   LS = 17   PS = 20
     2 : 1   T =  2   LS = 18   PS = 26
     2 : 2   T =  2   LS = 19   PS =  6
     2 : 3   T =  2   LS = 20   PS = 12
     2 : 4   T =  2   LS = 21   PS = 18
     2 : 5   T =  2   LS = 22   PS = 24
     2 : 6   T =  2   LS = 23   PS =  4
     2 : 7   T =  2   LS = 24   PS = 10

G =  3 : 0   T =  2   LS = 25   PS = 16
     3 : 1   T =  2   LS = 26   PS = 22
     3 : 2   T =  3   LS =  1   PS =  1
     3 : 3   T =  3   LS =  2   PS =  7
     3 : 4   T =  3   LS =  3   PS = 13
     3 : 5   T =  3   LS =  4   PS = 19
     3 : 6   T =  3   LS =  5   PS = 25
     3 : 7   T =  3   LS =  6   PS =  5
```

Lista 3. To listo generira progrma z liste 2, seveda potem, ko je bil predhodno preveden in nato izvajan. Ta lista se izpiše do vključno 77. sektorja, če programa prej ne prekinemo. Z določeno modifikacijo lahko dosežemo, da se lista izpiše na disketo, tako da jo pri potrebi lahko čitamo (npr. z ukazom TYPE).

NOVICE IN ZANIMIVOSTI

PROBLEMI INTELIGENCE

1. Uvod

Z načrti za reševanje problemov pete računalniške generacije je stopila v ospredje tudi problematika o razumevanju delovanja možganov in možganskih (informacijskih) procesov. Nenavaden, izjemen, poseben, predvsem pa nerazumljen pojav človekove inteligence kot skupka paralelnih informacijskih procesov v zapleteni nevronski mreži je postal osrednji predmet preučevanja in raziskav na področjih nevrofiziologije, nevropsihologije, nevrofarmakologije, informatike in računalništva.

Možgani so funkcionalno in fizično radeljeni na dve polobli, ki sta medseboj povezani z obsežnim svežnjem živčnih vlaken (200 do 300 milijonov), imenovanim corpus callosum. Ta sveženj je bil pri živalih in človeku večkrat eksperimentalno ali zdravstveno prekinjen (prerezan) s kirurškim posegom, tako da sta ostali polobli brez naravne medsebojne povezave. Ljudje z ločenima poloblama pa se obnašajo tako, kot da imajo dvoje možganov, ki delujejo neodvisno in ne vedo eni za druge.

Pri teh eksperimentih in posegih je postalo očitno, da so funkcionalne značilnosti leve in desne poloble bistveno različne, da je pri človeku moč razločevati tkim. leve in desne inteligence in da deluje povezan sistem leve in desne poloble bistveno drugače kot sistem, kjer je medsebojna informacijska povezava oslabljena ali celo prekinjena. Ob tem se je naravno pojavilo vprašanje, kje je sedež človekove zavesti, kako je ta spremenjena v prekinjenem sistemu in naposled, ali je moč razumeti informacijsko delovanje možganov z možganskim podsistemom, npr. z levo možgansko poloblo (pri desničarjih).

Vprašanje razumevanja delovanja možganov z levo poloblo se je pojavilo zaradi posebnih (zavestnih) lastnosti te poloble, v kateri je sedež (so središča) tkim. človekovega eksaktnega mišljenja. To vprašanje je postalo ključno za reševanje problematike pete računalniške generacije, v kateri bo potrebno na določene načine razrešiti probleme inteligence, ki naj bi postala glavna značilnost te tehnološke generacije. Ali je mogoče razrešiti te probleme z mišljenjem leve poloble, z nakopičenim eksaktnim znanjem? Kaj se pravzaprav skriva v obeh poloblah in je med njima funkcionalno (informacijsko) bistveno različno?

2. Razlike med levo in desno možgansko poloblo

S stališča možganov in njihovih procesov kot informacijskega sistema so raziskovalci možganov in človekovega obnašanja opazili bistvene razlike (različne lastnosti) v obdelavi informacij leve in desne poloble. Te včasih nasprotne značilnosti so prikazane v tabeli 1.

Pri tem velja povedati nekaj besed o serijskih in paralelnih informacijskih procesih v možganih. Serijska in paralelna obdelava v tabeli 1 se nanaša na integralno informacijo, na dovolj velik, informacijsko zaokrožen (podatkovni, podinformacijski) kompleks. Glede na delovanje nevronske mreže je v možganih obdelava takega kompleksa tudi v levi polobli paralelna, vendar je v danem trenutku v tej polobli v ospredju (v stanju pozornosti) en sam tak kompleks. S tega vidika deluje desna polobla diskurzivno, obravnava, primerja in ocenjuje hkrati več kompleksov in izvaja tako višje funkcije nad pojavljajočimi se informacijskimi produkti.

Tabela 1

Leva polobla	Desna polobla
* serijska obdelava informacij	* paralelna obdelava informacij
* linearna	* nelinearna
* časovno zaporedna	* časovno neodvisna
* paketna	* večprocesorska
* prekinitve z mehanizmom sklada	* nakljućno izvajanje (izvrševanje)
* besedna/simbolna	* vzorčna/slikovna
* neintuitivna	* visoko intuitivna
* točna, verna	* netočna, neverna
* strukturirano spominska	* asociativno spominska
* zbirno korelacijska	* istočasno večpodatkovno korelacijska
* s stopničastim učenjem	* z nezaporednim, nelogičnim učenjem
* čutilno odvisna	* čutilno neodvisna
* itd.	* itd.

Nekateri raziskovalci dojemajo (razumevajo) človekovo mišljenje kot definitivno nerazumljivo (nedoumljivo, nepojmljivo) za človekovo mišljenje, ker mora biti vsako razumevanje mišljenja neglede na obsežnost (problematike) podmnožica množice mišljenja (podobni aksiomatično-sistemski paradoksi so znani na področju "leve" logike, matematike, eksaktnih ved). Prav tak način razmišljanja je značilen za levo (zavestno) poloblo. Vendar je mogoče dognati določene sestavine inteligence tudi brez potrebnega znanja ali pojasnjevanja, kako delujejo. Te sestavine ali sposobnosti so npr. v naraščajočem zaporedju tele:

- izračunavanje,
- sovisnost,
- asociacija,
- sklepanje in
- ekstrapolacija (ugotavljanje izven obsega znanega).

Rezultati raziskav kažejo, da je leva polobla biološki primerek nekakšnega von Neumannovega (serijskega, linearnega) računalnika, kot je razvidno iz tabele 1. Desna polobla je še vedno zavita v skrivnost, vendar se dozdeva, da se v njej bolj razvijajo in obravnavajo gestalti, oblike, slike kot pa posamezni (izolirani) predmeti. Na nesrečo je večina zavestnega mišljenja besedne (stavčne, simbolne) narave, jezikovna središča (centri za jezikovno abstakcijo) pa se nahajajo v levi polobli. Iz tega izhaja, da je zelo težko opisati funkcije desne poloble s pojmi (terminologijo) leve poloble, ker so verbalni procesi (koncepti) zapleteno povezani s serijskim von Neumannovim računalnikom leve poloble.

Dojemanje delovanja desne poloble z delovanjem

leve poloble je de facto povezano z dekompozicijo paralelnih informacijskih procesov desne poloble v serijske informacijske procese leve poloble. Ali je ta dekompozicija vselej mogoča, to je osnovno vprašanje. Pri napredovanju računalniške tehnologije bi človek želel uporabljati predvsem obratno dekompozicijo: iz koncipiranih serijskih procesov (algoritmov), nastalih v levi polobli, bi želel pridobiti ekvivalentne (hitrejše) paralelne procese. Z logiko leve poloble je moč brez težav formulirati serijske procese, računalnik pa naj bi te procese inteligenčno (in seveda hitrostno) oplemenitil tako, da bi jih preslikal v paralelne procese (v logiko desne poloble). Iz tega bi bilo moč sklepati in se priučevati (skozi stoletja in tisočletja), kako deluje logika desne poloble.

Praktično vse to pomeni, da bo računalnik pete generacije preslikal serijski postopek v postopek paralelnega (inteligentnega) večprocesorskega sistema. Drugačna (nižja) funkcijska zmogljivost leve poloble je biološki primerek (dokaz), ki kaže, da je mogoče iz desne poloble prenesti le bistveno majhen del njene aktivnosti v levo polobl0. Polobli sta v normalnih možganih bistveno funkcionalno specializirani in vprašanje je, ali je moč specializacijo leve poloble dopolniti (naučiti) z deli specializacije desne poloble.

3. Procesi v desni polobli

Informacijski procesi desne poloble so splošno neverbalni. Primeri takih procesov so slikarski dosežki, simfonije, intuitivni prebliski; kolikšen je v njih delež procesov leve poloble? Poezija je verjetno lahko proces obeh polobel (seveda samo, če je zares izvirna, ustvarjalna, umetniško dobra). Leva polobla prispeva sistemske (šolske, modne, logične, okoliške, konformne) komponente, zavestno osredotočene. Kakovost abstraktnih asociacij, navidezno nelogičnega sklepanja in ustvarjalne ekstrapolacije pa prihaja iz desne poloble. Iz leve poloble prihajajo tako logično togo utrjene strukture mišljenja.

Čeprav desna polobla vobče nima zmogljivosti leve poloble, pa si te v posebnih pogojih lahko pridobi. Kadar pride do poškodb v levi polobli v zgodnji mladosti, te poškodbe navzven ne bodo opazne (tudi v primeru odstranitve leve poloble), ker lahko desna polobla zadovoljivo prevzame funkcije leve poloble. Leva polobla pa si nekaterih sposobnosti desne poloble ne more pridobiti. Te sposobnosti so npr.:

- reševanje problemov
- opravljavljanje nalog, ki zahtevajo neverbalno procesiranje
- presojanje in vrednotenje prostorskih sovisnosti
- razpoznavanje slik in vzorcev, ki so preveč zapleteni ali subtilni za verbalno opisovanje
- uporaba podobnostnih, šibko kategoriziranih funkcij
- itd.

Vse to kaže, da obstajajo v desni polobli funkcije, ki jih verbalno ni moč opisati in tudi njihovih zmogljivosti in lastnosti ni moč dobro razumeti.

Evolucijski pritisk je v skladu z razvojem v smeri višjih življenskih oblik (pogojev, navad) dejansko razdelil (specializiral) možgane v dva korezidentna računalna sistema. Danes še ni moč natanko kvantificirati in napovedati funkcij desne poloble, ni jih mogoče izraziti s pojmi leve poloble z izjemo izredno pomešanih in raznoličnih filozofskih jezikov tipa zen.

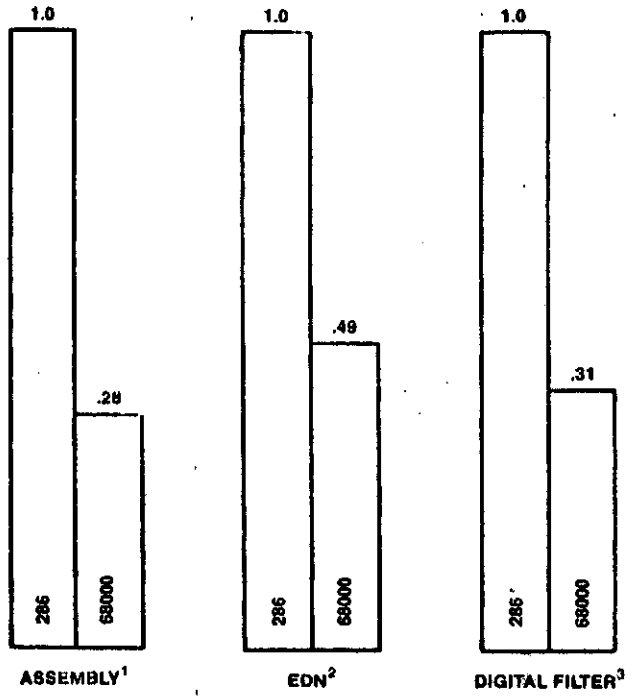
Desnomožganski računalnik lahko npr. izvrši funkcije A, B, C, D v poljubnem zaporedju in dobi še vedno pravilen odgovor. Izvrši lahko le A, B, -, D in sklepa o funkciji C oziroma o njenem učinku, ko so prisotni vsi štirje elementi. Podobno pripeljejo hkratne rešitve do novih tvorb, ki jih lahko označimo kot simfonije, slikarije, relativnostne teorije itd. Desna polobla uporablja tudi svojska časovna merila, tako da prihodnost lahko prehiteva sedanjost in se sedanjost razširja v prihodnost. Manifestacije v desni polobli so večkrat podobne neverjetnosti (telepatija, parapsihološki pojavi).

A. P. Železnikar

****
PRIMERJAVE PROCESORJA 80286
****

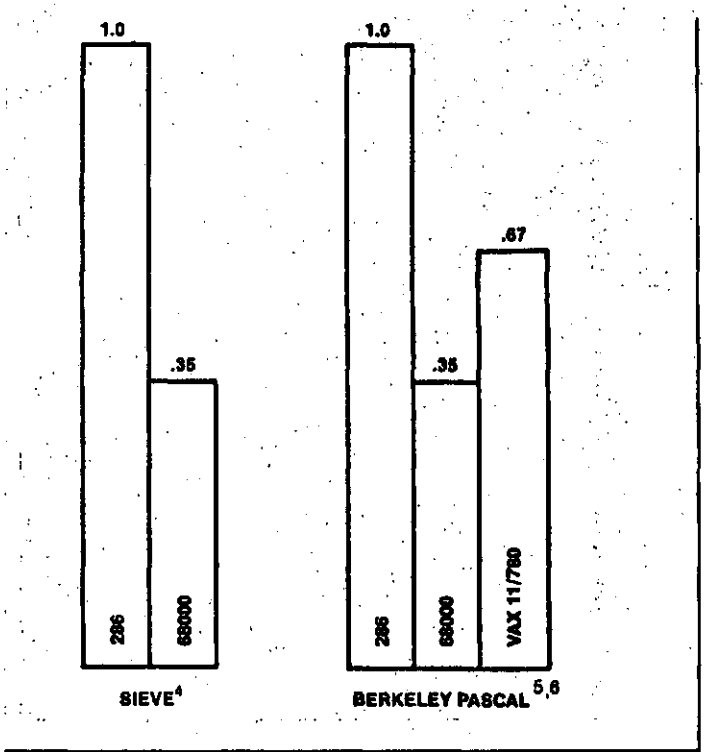
Procesor Intel 80286 se v zadnjem času večkrat primerja z drugimi mikroprocesorji pa tudi s procesorjem VAX 11/780. Te primerjave so zanimive, saj so primeri tako izbrani, da so vselej v prid mikroprocesorja 80286. Imamo tole:

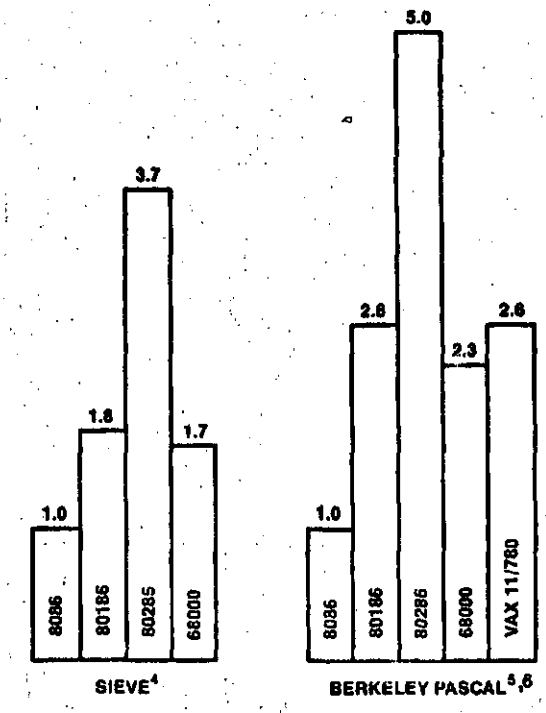
Protected-Mode System Level Comparisons 8 MHz, 150 nS Memory



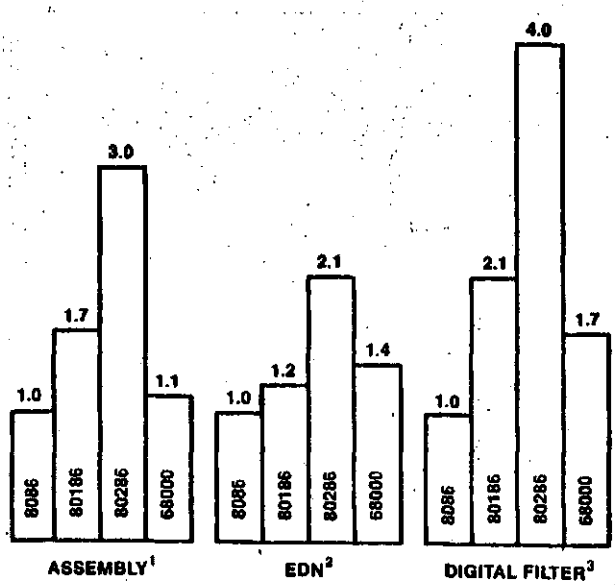
Opombe:

1. "16-Bit Microprocessor Benchmark Report," Intel Corporation, 1981.
2. "16-Bit Microprocessor Benchmarks," EDN, Sept 1981.
3. "Digital Filter Implementation on 16-Bit Microprocessors,", IEEE Micro, Feb 1981.
4. "A High Level Language Benchmark," Byte, Sept 1981.
5. "A Performance Evaluation of the Intel iAPX 432," Computer Architecture News, June 1982
6. 68000 z 68451 MMu potrebuje 2 čakalni stanji pri taktu 8MHz.





CPU to CPU Benchmark Comparisons
8 MHz 0 Wait States



Opombe:

1. - 5. so enake kot prej.
6. 8086 v primeru Berkeley Pascal je umerjen iz 5MHz, z dvema čakalnima stanjema

Iz prikazanih primerjav z različnimi evaluacijskimi postopki in vidiki pa se kaže npr. permanentna premoč v zmogljivosti procesorja 80286 nad procesorjem 68000. Ta premoč pa je verjetno še veliko večja v razpoložljivi sistemski in uporabniški programski opremi ter tudi v integrirani materialni podpori.

A.P.Železnikar

****
NOVI JAPONSKI MIKRORAČUNALNIKI
****

Prej ko slej bodo Japonci imeli prvovrstne osebne računalnike, kot je pokazala mednarodna razstava v jeseni prejšnjega leta (Japan Data Show '82, Oct 19-22, 1982, Tokyo). Ta razstava je pokazala spekter prenosnih in osebnih mikroračunalniških sistemov, v katerih prevladujejo mikroprocesorji podjetja Intel, in sicer 8088 in 8086. Pojavili so se tudi novi domači računalniki, ki pa za naš pregled niso relevantni.

Poglejmo le najprivlačnejše razstavljene mikrosisteme.

Japonski 16-bitni mikroračunalniki

Japonci so razstavili 20 novih 16-bitnih mikroračunalnikov, od katerih si bomo posebej ogledali osebne mikrosisteme.

National Mybrain 3000 je osebni računalnik podjetja Matsushita, ki se v ZDA prodaja pod imeni Panasonic, National in Quasar. Ima 4 ločene enote: izredno bogato opremljeno tastaturo, sistemsko enoto, videoprikazovalnik za barvno grafiko visoke ločljivosti in diskovne enote. Procesor je 8088 s 96K zlogi RAMa, z dodatnimi 32K zlogi video RAMa in 16K zlogi ROMa. Video RAM se lahko poveča na 128K zlogov. Video grafika ima ločljivost 640 krat 400 točk. Dobavljivi upogljivi diski so 3 colski, 5 in 1/4 colski in 8 colski, vsi v ločenih vertikalnih enotah. Sistem ima RS-232C in IEEE-488 vrata. Operacijski sistem je MS-DOS ali CP/M-86 z vrsto programirnih jezikov (programska oprema je v celoti ameriška!). Pri dve upogljivih diskih (s po 160K zlogi), 80-kolonskem igličnem tiskalniku in z jezikom Basic je cena tega sistema cca $3000.

Mitsubishi Multi 16 je mikroračunalnik "v enem kosu" in ga prodajajo tudi v ZDA, tastatura je ločena. Sistem uporablja procesor 8088 in po želji koprocesor 8087, 128K RAM, ki je razširljiv na 576K zlogov, barvno grafiko s 640 krat 400 točkami, en 300K zložni upogljivi disk in

operacijski sistem CP/M-86. Sistem je moč aparaturno razširiti.

Toshiba Pasopia 16 je osebni računalnik, ki se v ZDA prodaja pod oznako T300. Ima več video monitorjev za različne ločljivosti (320 X 200 do 640 X 500 točk). Enote so med seboj ločene, upogljivi diski (5 colski) dosežejo že 640K zlogov. Tiskalnik je 80-kolonski, uporabijo se lahko tudi 8 colski upogljivi diski (v tem je prednost ločenih enot). Procesor je 8088 s koprocesorjem 8087, s 4K ROMom, 192K RAMom, ki je razširljiv do 512K zlogov. Doda se lahko tudi video RAM od 128 do 256K zlogov. Operacijski sistem je MS-DOS.

Tesbac UX-300 (Toshiba) uporablja procesor 68000, ki je podoben procesorju 8086. Ta sistem lahko uporablja vinčestrski disk 10M zlogov, ima do 512K RAMa, cena je $9300.

NEC PC-9800 in APC (Advanced Personal Computer) sta presenečenji podjetja NEC. PC-9800 je pravi 16-bitni osebni računalnik s procesorjem 8086, dočim se APC prodaja v ZDA pod oznako N5200 in uporablja tudi procesor 8086. APC ima 128K zložni RAM, dva 8 colska diska z 1M zlogi, monokromatični minitor s 640 X 475 točkami, s ceno $2800č njegova ameriška različica se prodaja za $3900. PC-9800 ima 128K do 640K RAM, 96K ROM z jezikom NBASIC-86 in s 96K video RAMom. Monitor ima ločljivost 640 X 400 točk. S 128K RAMa in dvema 8 colskima diskoma (po 1M vsak) je njegova cena $2600 na Japonskem.

Hitachi BASIC Master 16000 in PT-1 osebni terminal sta 16-bitna osebna računalnika. Prvi uporablja procesor 8088, MS-DOS in 320K RAMa v standardni konfiguraciji. Video ločljivost znaša 640 X 400 točk. Pet colska diska imata po 340K zlogov, cena pa je $1850. Tudi PT-1 uporablja MS-DOS, dva 8 colska diska (po 1M) in video ločljivost 720 X 520. PT-1 je predviden za pisarniško avtomatizacijo v lokalni mreži.

Sanyo MBC-55 je vrhunski osebni računalnik s procesorjem 8088, zelo kompakten, s tankima 5 colskima diskoma (po 160K). Pri prodaji v ZDA bo zanj mogoče uporabljati enake module kot za IBMov PC. MBC-55 uporablja operacijska sistema CP/M-86 in MS-DOS, pomnilnik je razširljiv od 64K do 256K. Sistem ima 4K nalagalni ROM in procesor 8087. Njegova cena v ZDA bo pri minimalni konfiguraciji $1000 brez video monitorja. To bo močna konkurenca osebnemu računalniku IBM, saj bo MBC-55 sprejemljiv za družinski proračun.

Sord M-343 je le eden izmed številni mikroračunalnikov, ki jih izdeljuje podjetje Sord (npr. M5 domači računalnik ali prenosni mikroračunalnik M23P). Ta sistem ima izredno kvalitetno barvno grafiko in procesorje 8086, 8087 in Z80A. Sistem je v enem ohišju in lahko ima 1176K zlogov RAMa, 4 V/I vrata, video ločljivost 640 X 400 točk. Z80A je predviden kot povezava v mreži. Ima dva 8 colska diska s po 1,2M zlogi, 80 programskih tipk na tastaturi in tri dodatna podnožja (konektorje) za vodilo S-100. Uporablja različne operacijske sisteme (6 po številu, med njimi CP/M-86, MS-DOS, UCSD p-sistem, PIPS). Takšne programske integracije nima zaenkrat noben komercialni ameriški osebni računalnik.

Anritsu Packet II Hy Personal Computer je edini med eksponati s procesorjem 68000 in brzkone edini s tem procesorjem v svojem razredu. Ima dva diskovna pogona (5 colska s 150K vsak), ima 256K zložni RAM, dodati je moč celo A/D pretvornike. Ta računalnik je namenjen tehničnim uporabnikom.

AI Electronics AI-M16 uporablja procesor 8086 in 8089 V/I procesor, ima možnost uporabe 8087 (matematični koprocesor), vsebuje koledarsko uro in RAM od 256K do 1M zlogov. Uporablja javni standard IEEE-796 (Multibus). Nadalje ima 16K nalagalni ROM in 8K znakovni generator plus 128K kanji ROM. Uporablja 8 colske upogljive diske in 5 colski vinčestrski disk. M-16 ima vrsto operacijskih sistemov: Genix, CP/M-86, Concurrent CP/M-86, MP/M-86, MS-DOS, UCSD p-sistem itd. Njegovi sistemski jeziki so med drugimi LISP, PL/I in C.

Seiko 9500 in 8600 sta 16-bitna mikroračunalnika, predvidena za japonsko in ameriško tržišče. 9500 je super osebni mikroračunalnik, zgrajen v enem ohišju, z vodoravno postavljenimi diskovnimi enotami. Ima vrsto procesorjev: 8086 z 8087 koprocesorjem in še dva 8088 procesorja, ki ju uporablja za V/I in druge krmilne komunikacije. Območje RAMa sega od 256K do 512K zlogov, operacijski sistem je RMX/86, ločljivost barvne grafike znaša 512 X 480 točk. 8600 je predviden za prodajo v ZDA, je manjši in ima bolj dognan izgled.

Japonski 8-bitni mikroračunalniki

Sharp X1 je osembitni mikroračunalnik s procesorjem Z80A, z barvnim monitorjem, 64K RAMom, 4K video RAMom in 6K znakovnim RAMom, dodati pa je mogoče še 48K video RAMa. Ta računalnik ima tračno kasetno enoto, uro realnega časa, mali tiskalnik.

Sony SMC-70 uporablja Z80A procesor, v prihodnosti pa bo imel procesor 8086 z MS-DOSom. V ZDA se prodaja kot poslovni sistem. Ima 3-colske diske.

Japonski prenosni mikroračunalniki

Na razstavi v Tokiu so se pojavili 4 zanimivi prenosni mikroračunalniki.

Aval AVC-777J2 je v enem kosu, uporablja operacijski sistem CP/M 2.2 z Z80A, 64K RAMa in 16K video RAMa. Vgrajen je 5-colski monitor, dve 5-colski diskovni enoti (po 600K vsaka), 5-colski termični tiskalnik, priključnice za zunanje diske in zunanje vodilo itd. Cena je $3320, teža pa 12,5 kg.

Aval AVC-666 je razvojni sistem s CP/M 2.2, z dvema 5-colskima diskovnima enotama, podoben sistemu 777J2. Ima priključnico za zunanji video prikazovalnik, je veliko lažji in cenejši ($2500).

Sord M23P je pravi prenosni mikroračunalnik s procesorjem Z80A, 128K RAMa in 80-znakovnim in 8-vrstičnim prikazovalnikom s tekočimi kristali. Ta računalnik omogoča delo na poti, doma pa je moč priključiti navaden ali barvni monitor. Računalnik uporablja dve diskovni minienoti s 600 obrati na sekundo s po 580K zlogi. Njegova teža je 9 kg, cena $2200.

Epson HC-20 se prodaja v ZDA pod oznako HX-20, je manj zmogljiv kot M23P, cena je $800, portabilnost je odlična.

Novi domači računalniki

Da ne bo nesporazuma, omenimo takoj, da so domači mikroračunalniki namenjeni za uporabo doma (v domu) in da je njihova uporabnost in cena prilagojena potrebam in finančnim zmogljivostim.

National JR-200 osebni računalnik je cenen proizvod podjetja Matsushita, ima procesor 6802, 16K ROMa in 32 K RAMa. Cena je okoli $300.

JR-100 ima procesor 6802, 8K ROMa, 16K RAMa, manjšo tastaturo, pri ceni $210.

Sord M5 ima procesor Z80A, 8K ROMa, 4K sistemskega RAMa, 16K grafičnega RAMa. Programska oprema se prodaja na kasetah ali v ROMih. Ta računalnik ima VF izhod (za TV), cena je $187.

Sanyo PHC-25 sodi med boljše domače računalnike s ceno $264. Velikost tega računalnika je enaka velikosti tastature; ima pa 24K-zložni Basic v ROMu in 22K-zložni RAM. Kot video prikazovalnik uporablja televizor in kaseto kot sekundarni pomnilnik. PHC-20 je cenejša izvedba tega domačega računalnika.

Novi ročni računalniki

Sanyo PHC-8000 je računalnik, ki ga držimo v roki. Njegov procesor je NSC-800 v CMOS izvedbi, ima pa 24K-zložni ROM in 4K zložni RAM. Prikazovanje se opravlja na enovrstičnem zaslonu iz tekočih kristalov. Cena je $263. Nanj je moč priključiti zunanji video monitor, mikro kasetni zapisovalnik in dodati še 14K-zložni ROM in 22K-zložni RAM. Mali tiskalnik je v ohišju oziroma vse skupaj v torbici.

Toshiba Pasopia Mini ima 8-bitni CMOS procesor, 4K-zložni RAM, 20K-zložni ROM s 16K-zložnim jezikom Basic. Prikazovalnik je iz tekočih kristalov, enovrstični. Osnovna cena je $210, dodati pa je mogoče še 12K-zložni RAM za $113. Priključi se lahko tudi zunanji tiskalnik, cena zunanjega kasetnika je $170. Tako počasi in zanesljivo naraste cena!

NEC PC-2001 ročni računalnik je med najbolj prijaznimi. Uporablja 8-bitni mikroprocesor uPD7907 v CMOS izvedbi, s taktno frekvenco 4 MHz, 36K-zložnim ROMom in 16K-zložnim RAMom. Ima tudi serijska vrata in dvovrstični prikazovalnik s tekočimi kristali. Cena je $225.

Podjetniške usmeritve

Težnje japonskih elektronskih podjetij se kažejo v proizvodnji popolnih mikroračunalniških linij, od ročnih do zelo zmogljivih namiznih računalnikov. Podjetje NEC izdeluje npr. ročni računalnik PC-2001, domači računalnik PC-6000, osebni računalnik PC-8000 (z Z80 procesorjem), močnejši osebni računalnik PC-8800 (tudi z Z80), osebni računalnik PC-9800 (s procesorjem 8086) in visoko zmogljivi osebni računalnik N5200 (procesor 8086).

Toshiba proizvaja npr. ročni računalnik Pasopia Mini, osebni računalnik Pasopia (s procesorjem Z80), 8-bitni namizni računalnik T200 z dvema 5-colskima upogljivima diskoma, 8-bitni namizni računalnik T250 z vgrajenima 8-colskima diskoma in osebni računalnik Pasopia 16 s procesorjem 8088. Obe podjetji pa proizvajata tudi tkim. tastaturne računalnike, kot je npr. Epson HC-20 in prenosne mikroračunalnike.

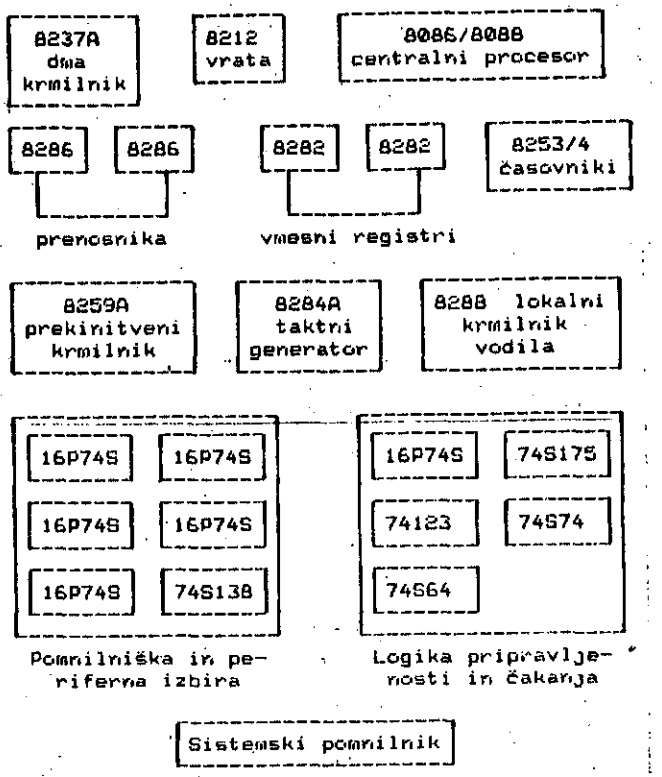
V ZDA ima le Hewlett-Packard širšo paleto, deloma tudi DEC, za IBM pa to že ne velja več. Gre torej za bistveno različni strategiji osvajanja novih mikroračunalniških tržišč in kdo bo uspešnejši (Japonci ali Američani), postaja danes vse bolj očitno.

A. P. Železnikar

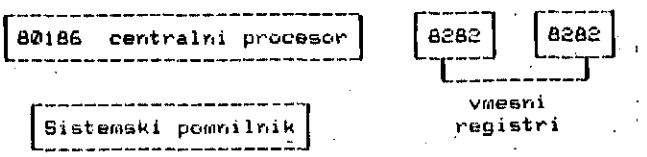
****
RAZLIKA MED 8086 IN 80186
****

Razlika med tema odličnima mikroprocesorjema je najbolj nazorno vidna na spodnji sliki. Gre predvsem za bistveno razliko v arhitekturah.

Procesor 8086 s svojo ožjo okolico (podporo):



Ekvivalentna konfiguracija s procesorjem 80186:



A. P. Železnikar