7–8 '90 ЮЛИ – АВГУСТ ГОДИНА ШЕСТА

КВ

КОМПЮТЪР ЗА ВАС

Софтуер на годината

Българска BBS

Intel 80486

Боледува ли нашата електроника

Обмен между Правец-82 и Правец-8Д

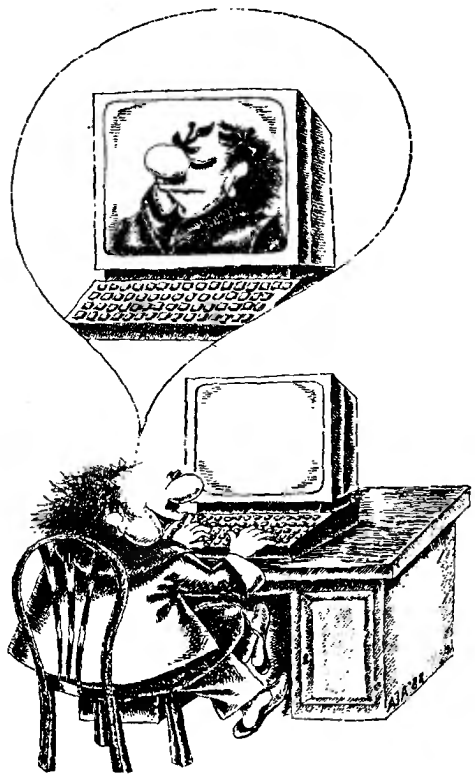
МикроТЕКСТ II — ИЗДАТЕЛ

ФИРМА „ПРОГРАМНИ ПРОДУКТИ И СИСТЕМИ”
ТК „НАЦИОНАЛЕН ПРОГРАМЕН И ПРОЕКТЕН ФОНД”

НАСТОЛНА ИЗДАТЕЛСКА СИСТЕМА

я днес работят:
Българска телевизия, Издателство „Народна култура”, Институт по икономика и изация на селското стопанство, вестник „Футбол”, сп. „Компютър за вас“, в. “Нап- ам”, ТИ „ВЗУМЛ” - гр. Пловдив, ДП „В.Андреев” - гр. Перник
чад 200 организации.

Предлагаме Ви нови програмни продукти,



които разширяват възможностите на системата:

- към многофункционалната графика – **МАГ**:
  - нов модул **ЩРИХ**, с който бързо и удобно се създават висококачествени текстово-графически изображения;
  - нов модул **ДИАГРАМА**, с който бързо, леко и удобно се създават и представят карти, знаци, проекти и планове (диаграми);
- нова версия на **СКЕНЕР**, която дава възможност за редактиране на сканираното изображение (изрязване на част от него, копиране, залепване, обрязване, увеличаване или намаляване);
- нов набор от шрифтове – **ТАЙМС** и **УНИВЕРС** в размерности от 6 до 48 пункта и начертания — нормален, получер, курсив и получер курсив;
- възможности за директно експониране чрез фотонаборен автомат *LASER COMP.*

Тези програмни продукти работят с популярния програмен пордукт *„ИЗДАТЕЛ”*.
На всички фирми, издателства, институти и други, които желаят да сами да подготвят своите модерни и прецизно оформени издания, напомняме, че това което им трябва е

НАСТОЛНА ИЗДАТЕЛСКА СИСТЕМА

Посетете нашия комбинат или вижте резултатите на клиентите ни, за да се убедите в това.
При желание на клиента ние оказваме пълно съдействие за внедряване на **Настолната издателска система.**

За справки и по-подробна информация:
София, ТК „НППФ”, бул. „Бр. Бъкстон” бл. 207А
тел. 55-70-28

форума представляват дискусиите между специалистите по някои теоретични аспекти на компютърната вирусология. Такива бяха споровете за това, дали е теоретично възможно създаването на програма, която да открива всички възможни вируси (оказа се, че, уви – не е възможно); заслужава ли си да се използва криптографски слаб метод за пресмятане на контролни суми, като например CRC (оказа се, че да – заслужава си, ако се спазват някои елементарни правила, като това всеки потребител да използва собствен полином за CRC контрол и по време на проверката паметта на компютъра да е гарантирано свободна от вируси); какво е приложението на криптографските системи с обществен ключ в борбата срещу вирусите; етично ли е студентите да бъдат обучавани как се правят вируси и антивирусни програми, като това, разбира се, се прави в строго контролирана среда; и много, много други.

Форумът има огромно значение за своевременния обмен на информация между специалистите, както и за ефективното достигане на тази информация до обикновените потребители. На участващия в него той предлага неописуемото чувство на приобщеност към хора, които са разпръснати по целия свят (аз се свързах с хора от Австралия, Тайван, Израел, ФРГ, Англия, Исландия, САЩ, Бразилия, Чили) и които се вълнуват от твоите проблеми, които имат твоите интереси. Вероятно нещо такова са изпитали първите радиолюбители, когато са открили, че могат да се свързват помежду си.

При това връзката, която се осигурява, е миого по-стабилна от радиовръзката, много по-бърза от пощата и много по-удобна от телефона. При обмен на съобщения със САЩ получавах отговор още същия ден – поради изоставането на САЩ в часовите пояси – по обяд. Когато бях готов със съобщението си (след като съм го съставил въз основа на получената през нощта поща), там беше още ранна сутрин. Съответният абонат беше токущо дошъл на работа или пък скоро щеше да дойде. Тъй като обикновено електронната поща се преглежда сутрин, той имаше възможност веднага да се запознае със съобщеннето ми, да състави отговор и да ми го изпрати, преди в Мюнхен да настъпи вечерта. По-голямо беше закъснението при обмен на съобщения с Тайван или Израел – в тези случаи аз бях изостаиалият в часовите пояси и кореспондентите ми отговаряха едва на другия ден – след като са прегледали сутрешната си поща. Същевременно комуникацията по електронната поща има това предимство, че не струва нищо (на абоната) и за разлика от телефона по нея могат да се предават и програми.

Но най-приятна от всичко е възможността на равна нога да участваш в борбата срещу вирусите съвместно с известни специалисти от целия свят. През тези три месеца ми се удаде само една такава възможност.

Появи се нов вирус, заразяващ .COM и .EXE файлове. Място на появата – Тайван. Тайванският кореспондент – човек, сравнително нов за форума и незапознат с методите му за работа и с някои етични предпоставки, изпаднал в паника и разпратил копие от заразена програма (CHKDSK) на всички абонати на VALERT (около 600 души, повечето от които – неспециалисти), придружена с отчаяна молба за помощ.

На втория ден сутринта David Chess (служител на IBM) изпраща от Ню Йорк, САЩ, съобщение с основните свойства на вируса, получени в резултат на проучване на дизасемблерния му листинг (той не се е осмелил или е нямал време да прави експерименти с работещия вирус). Същия ден вечерта пишещият тези редове изпрати съобщение от Мюнхен, ФРГ, с което потвърди основните резултати на David Chess и съобщи за някои нови свойства, наблюдавани при експерименти с живия вирус и потвърдени от изследването на кода му.

На третия ден кореспондент от Чилийския университет нанесе поправка в едно от твърденията на David Chess и изказа хипотеза за начин за лекуването на вируса. По същото време John McAfee (Калифорния, САЩ) съобщи, че поредната версия на вирусотърсачката му (SCANV58) вече е в състояние да открива и този вирус. Там (и в придружаващата програмата документация) той неправилно смята, че вирусът е дълъг 1559 байта.

На другия ден аз изтъкнах несъстоятелността на хипотезата на кореспондента от Чили и коригирах John McAfee, че вирусът е 1554 байта дълъг, въпреки че може да добави още до 16 байта към заразените файлове (името на вируса е вече коригирано в документацията на версия 60 на вирусотърсачката). Същата вечер Fridrik Skulason от Университета в Рейкявик, Исландия, съобщи, че е създал програма, която успешно лекува заразените с този вирус файлове. Разпращането на тази програма до всички пострадали беше извършено напълно професионално – тя просто беше поставена в един възел на мрежата, където се намираше компютър, предназначен да архивира и разпространява на всички желаещи обществено достъпен (public domain) софтуер. Така само за четири дни светът беше избавен от поредния вирус.

Темповете, при които се води подобна борба у нас, наистина будят съжаление. Човек може само да си задава въпроса, защо у нас, една, общо взето, активно компютризираща се страна все още няма електронна поща, след като в една Турция например има, и при това – доста добре развита. (Използвам като пример Турция, защото ни е близка съседка. Същото обаче може да се каже и за такива страни, като Индия, Чили, страните от Латинска Америка, Тайван и др.)

Докога ще стоим настрана от цивилизования свят? Досега България беше известна предимно като страна на софтуерни пирати. Сега тя е известна и като страна, в която активно се правят комнютърни вируси – Eddie, наричан още Dark Avenger, TP05, наричаи и TP39, наричаи Yankee Doodle, са български вируси, които са добре известни иа света. Известно е това, че са български. Докога у нас правенето на вируси ще се разглежда като вид спорт, а не като престъпление срещу обществото, каквото е всъщност? Докога борбата срещу това явление ще се води на любителски и обществени начала, вместо професионално? Кога най-сетне у нас ще се появи субсидирана от държавата организацня за борба с вирусите – организация, каквато имат редица страни като САЩ, Англия, ФРГ, Австралия и др.?

Наистина сп. „Компютър за вас“ не е място за политически дискусии въпреки активната политизация на все по-широки области и слоеве от нашето общество. Аз самият също не проявявам никакъв интерес към политиката. Обаче въпросите, които зададох по-горе, съм готов да задам (и няма да престана да задавам) на всяко правителство. ■

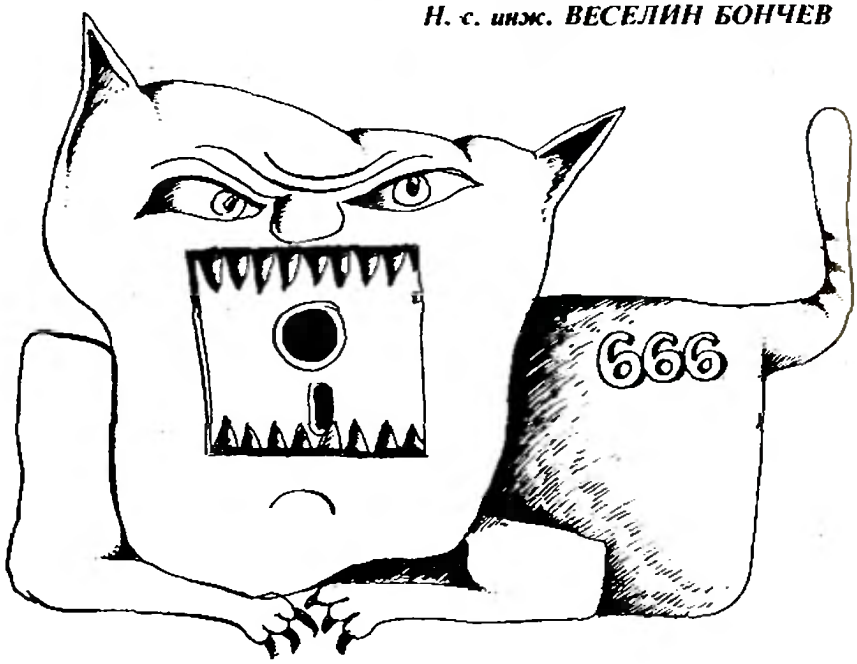
НОВОТО на вирусния фронт

Н. с. инж. ***ВЕСЕЛИН БОНЧЕВ***

По време на тримесечното ми отсъствие от България през месеците декември, януари, февруари и началото на март, донякъде изпуснах от контрол ситуацията с компютърните вируси. Веднага след като се прибрах, честотата на телефонните обаждания на хора, търсещи консултация по въпроса, достигна и дори надмина тази от времето, когато за първи път публикувах телефоните си. Единствената добра новина е, че на практика всички обаждания се отнасяха *за вируси, които моят пакет* антивирусни програми[1] не е в състояние да лекува. Нека да се надяваме, че той е успял да се справи поне с предишните вируси. Два от новите вируси се бяха появили непосредствено преди моето заминаване, когато нямах време дори да започна да създавам средства за борба срещу тях. Освен това през същия период се появиха доста нови вируси и мутации на предишните.

Но нека да започнем подред.

Вирусът V512



Много читатели ми писаха, за да ми поясняват, че подписът „666", съдържащ се в този вирус, е т. нар. „Число на Звяра", т. е. на Сатаната. Аз не съм могъл да се сетя за това, защото не познавам *Библията, нито пък гледам филми* с подобна тематика. Както и да е, в бъдеще можем да наричаме този вирус именно така — Числото на *Звяра* (The Number of the Beast). Тъй като обаче това е доста дълго, на места ще продължавам да използвам по-краткото V512.

В КВ.1—2.90 бях писал, че този вирус едва ли ще се разпространи, защото предизвиква „увисване" при начално зареждане на операционната система и заразен команден интерпретатор. Естествено това предизвика десетки телефонни обаждания на хора, които твърдяха, че са заразени именно от този вирус, *защото компютърът им* „увисвал при извършване на начално зареждане". Най-забавното от цялата история е, че аз отново съм сгрешил или, по-точно, изложеното в статията ми не отговаря напълно на истината.

Работата се състои в това, че вирусът отлично разбира кога се извършва начално зареждане и се опитва да стартира заразения команден интерпретатор още от първия път. Обаче това не се извършва съвсем коректно — когато се опитва да отвори файла, съдържащ злополучния команден интерпретатор, вирусът погрешно *предполага, че дължината на името* му не превишава 16 знака. А това е така само ако този файл се намира в основната директория — най-често срещаният случай. При това положение компютър, заразен с този вирус, ще работи изрядно (с някои малки изключения) и незабележимо ще разпространява заразата.

Обаче на компютъра, на който работя, файлът COMMAND.COM съвсем не се намира в главната директория! Именно това беше причината за увисването, което наблюдавах.

Малките изключения в изрядната работа на заразения компютър, за които споменах по-горе, се отнасят за програмите от типа на DISKCOPY и DISKCOMP (във всеки случай за тях знам със сигурност). При стартиране на заразена DISKCOPY и указването, че трябва да се копира например от устройство A: на устройство B:, програмата извежда странното съобщение, че указаното устройство е non-removable, т. е., че е твърд диск. Причината за това е,

[1] *Разпространявани безплатно от списание „Компютър за вас" — Б.Р.*

че вирусът разрушава при работата си стойността в регистър AX. А, както е известно, при задаване на аргументи на една програма в AH трябва да стои логическият номер на дисковото устройство, зададено чрез първия от аргументите (или OFFh, ако дисковото устройство е невалидно). Повечето програми не използват тази информация (тя е отживелица от времето на операционната система CP/M), но както е добре известно, системните програми на PC-DOS са удивително консервативни.

През споменатия в началото на статията тримесечен период у нас се появиха поне още три мутации на този вирус. И трите (аз ги наричам съответно V512-B, V512-C и V512-D) очевидно са писани от същия автор. При разглеждане на кода просто се вижда как той се е опитвал да напъха още повече трикове в програмата си. Обаче мястото е наистина прекалено малко. За да го увеличи, той е пожертвал дори подписа си (и трите мутации не съдържат низа „666"), но това не е помогнало. В една от тях той е изпуснал проверката за версията на операционната система (хватката с INT 2 Fh, описана в КВ.1—2.90, работи само за PC-DOS 3.30). В друга се отказал да следи дали при опита за заразяване не е възникнала грешка и т. н. И при трите мутации пълната файлова спецификация на командния интерпретатор се извлича коректно, независимо в коя директория е поставен той. А една от тях дори не разрушава стойността на регистър AX за радост на програмата DISKCOPY. Както и да е, усилията на автора на вируса са напразни — разполагам с програма, която е в състояние да лекува файловете независимо от това, с кое от мутациите са заразени. Хубавото е, че програмата е в състояние да лекува и от бъдещи мутации на този вирус — стига основните принципи на заразяване да се запазят (впрочем, ако те бъдат променени, ще става дума вече за нов вирус, а не за мутация).

Друга разлика и на трите мутации от оригиналната версия е фактът, че вече се заразяват файлове със спецификация „*.?OM", а не тази със спецификация „*.CO?". Вероятно авторът на вируса е взел предвид, че в последния случай има опасност да се зарази файлът PRESTO.COD на продукта „Престо!", което може да го направи неработоспособен. Освен това вече не е необходимо заразената програма да се стартира два пъти, за да тръгне на незаразена система — тя тръгва още от първия път.

Вирусът Eddie

В началото на годината авторът на този вирус е имал наглостта да започне да разпространява изходния текст на вируса си. Текстът е пълен със саркастични коментари по мой адрес и със забележки, че на места програмата е пълна с глупости и недомислия — нещо, което и аз винаги съм твърдял и което е очевидно за всеки, който си е направил труда да дизасемблира вируса и да изследва поведението му.

Очевидно Мрачният отмъстител се е надявал, че сега куцо и сакато ще започне да използва текста на вируса му, за да прави на негова основа нови вируси. Всъщност пожелание в този дух се среща съвсем явно в текста. Да се надяваме, че това няма да се случи.

Освен това този човек е пуснал в разпространение и една програма, наречена DOCTOR[2] (удивителна липса на въображение при избора на име), която лекува вируса. Програмата е дълга точно 20 000 байта (старата любов на Мрачния отмъстител към кръглите числа), шифрована е (по изключително тривиален начин) и в нея се съдържат някои весели хватки, които, разбира се, не са документирани никъде. По-интересното е, че в нея се съдържат и двата вируса — Eddie и V2000 (по-подробно за втория — след малко). Друг любопитен момент е, че когато се стартира с опция /h, програмата извежда два екрана текст, в който се описва историята на създаването на вируса Eddie. В текста отново се съдържат ехидни забележки по мой адрес и по адрес на инженерите изобщо. Самият текст е написан на също толкова неграмотен английски, колкото е неграмотен и българският на Мрачния отмъстител — нещо, в което лесно можете да се убедите, ако прочетете публикуваното му в КВ.3—4.90 анонимно писмо до мен.

В текста се отправя и едно предизвикателство към потребителите — да се опитат да познаят защо вирусът се нарича именно Eddie. Ще се опитам да отговоря на тази загадка. По всичко изглежда, че Мрачният отмъстител е почитател на музикалния стил метъл, и по-точно — на групата Iron Maiden. За това недвусмислено говорят текстовете, които се срещат в неговите вируси — „Somewhere in time", „Only the Good die young", та даже и „The Number of the Beast" са все песни на тази група. Е добре, а точно Eddie се нарича един скелет, който е избран за талисман на групата. Пръв за това се досети Fridrik Skulason от Исландия.

В заключение ще ви кажа, че не ви съветвам да използвате тази програма, за да се лекувате от вируси. Впрочем и вие едва ли ще се решите да го направите — антивирусна програма, правена от създател на вируси — това звучи повече от подозрително.

Вирусът V2000

Още един български вирус. Ако се съди по стила на програмиране, автор и на това „произведение" е отново Мрачният отмъстител. Впрочем той сам си признава това в програмата DOCTOR. Там той го нарича версия 1,4 на Eddie. Показателен е и фактът, че редица западни програми, предназначени за откриване на вируса Eddie (този вирус е разпространен и на Запад), указват вируса V2000 като негова мутация. Други косвени улики са дължината на вируса (Мрачният отмъстител обича кръглите числа), името Диана П. (този път написано на кирилица), както и три низа, разположени в началото, средата и края на вируса.

Тези три низа са съответно „Only the Good die young...", „Диана

[2] *Известната антивирусна програма DOCTOR на Валери Трифонов, която се разпространява от нашето списание, има същото название, но друга дължина. Между двете едноименни програми няма нищо общо. — Б.Р.*

П." и „Copyright (c) 1989 by Vesselin Bontchev". Освен това низът „Диана П." се среща още веднъж — на края на вируса.

Държа да подчертая, че злощастната Диана П. няма нищо общо с момичето, дало името на софтуерната фирма „Диана" (ако не се лъжа, нейното име е Диана М.). Напълно безпочвени са слуховете, че този вирус и вирусът Eddie са разработени от тази фирма. Що се отнася до низа, който ме набеждава като автор на този вирус, той сам по себе си е достатъчно голяма подлост. Мрачният отмъстител обаче не се е ограничил с това. На практика част от този низ („Vesselin Bontchev") се използва от вируса. Той следи дали някоя от стартираните програми не го съдържа и ако това е така, компютърът „увисва".

У нас са известни поне две модификации на този вирус. Според мен и двете са дело на един и същ автор — Мрачния отмъстител. В първата модификация низът „Only the Good die young..." е заменен с низа „Copy me – I want to travel". Освен това има една малка промяна в кода на вируса, която между другото съдържа низа „666". Очевидно Мрачният отмъстител много държи да бъде смятан и за автор на вируса „Числото на Звяра". Втората модификация е съвсем незначителна промяна на първата — началната буква от думата „Copy" е заменена от C на Z.

Вирусът заразява както .COM, така и .EXE файлове, като увеличава дължината им с 2000 байта (не 2 Кбайта!). .EXE файловете първо се допълват, така че дължината им да стане кратна на 16 байта и чак след това към тях се добавя дългият 2000 байта вирус. .EXE файловете се разпознават по първите им два байта, като се прави проверка както за MZ, така и за ZM.

Подобно на Eddie (и поради същите причини), този вирус може да зарази двата скрити файла на операционната система (ако разширенията им са .COM), което ще доведе до това, че нападнатият компютър ще престане да извършва начално зареждане (разбира се — до откриването на вируса).

Вирусът остава резидентен в паметта чрез манипулация на контролните блокове на паметта, като заделя 4 Кбайта памет. Инсталацията в паметта се извършва по малко по-различен начин от вируса Eddie — в смисъл че не съдържа толкова грешки и предизвиква по-малко проблеми. Този вирус, както и Eddie, няма да работи на версия на PC-DOS, по-малка от 3.0. За сметка на това поддържа се и версия 4.0 на тази операционна система. Също както при Eddie файловете се заразяват както при стартиране, така и при копиране и/или разглеждане. Командният интерпретатор се заразява първи — механизмът е същият както при Eddie.

Този вирус предприема редица мерки, затрудняващи откриването му. Той не само претърсва контролерите, за да намери адреса на програмата, която управлява дисковете (както Eddie), но и използва функция 13h на INT 2Fh, за да получи адреса на оригиналната програма за обслужване на INT 13h (както „Числото на Звяра"). Освен това се прехващат и функциите на PC-DOS за търсене на файлове (както това се прави във V651, при това се съдържат и същите грешки), така че командата DIR да не показва увеличение в дължината на заразените файлове. За щастие CHKDSK (и редица програми за работа с файлове — PCTools, Files, Norton Comander) е в състояние да забележи измамата.

Борбата с програмите, съдържащи моето име, се е видяла недостатъчна на автора на вируса. Той е насочил злонамерените си усилия и към напълно невинните потребители. Вирусът разрушава информация на същия принцип, както и Eddie — всяко шестнайсето стартиране на заразена програма води до записване на един сектор върху случайно място от диска. Съответните два брояча — от 0 до 16 и за последен атакуван сектор, се поддържат в boot-сектора на същото място, както е при Eddie. Същият е и алгоритъмът за избор на случаен сектор. Променен е само начинът, по който се извършва записът. Вече не се използва INT 26h (нещо, което лесно може да бъде прехванато). Вместо това вирусът се обръща директно към програмата за управление на съответното устройство (device driver) на операционната система. Разбира се, адресът ѝ се получава чрез използването на недокументирана функция на PC-DOS. Този метод, уви, не може да бъде прехванат от резидентна програма. Наистина той се свежда от PC-DOS до извикване на INT 13h, но вирусът междувременно е променил и вектора на това прекъсване с оригиналния — да не забравяме, че този вектор се открива още при инсталирането на вируса в паметта.

Едва ли е нужно да подчертавам, че вирусът има собствена програма за обработка на критичните грешки, така че да не се появяват съобщенията на PC-DOS, например при опит за заразяване на механично защитена срещу запис дискета. Това, както и някои други свойства (да не се виждат в паметта от програмата MARMEM, да заразяват както .COM, така и .EXE файлове, да преодоляват резидентните защити и т. н.) стаиха едва ли не стандарт за създаваните в нашата страна вируси.

Вирусът V-277

Този вирус е от поредицата V-847/V-345/V-299 (по света наричат тези вируси Amstrad или Pixel). Просто да се чуди човек защо е било необходимо за основа да се взема един толкова глупав вирус. Както и да е, засега това е най-късият вирус в света за компютри от типа IBM PC. (Най-късият вирус в света въобще е дълъг само 11 байта, два от които са интервали. Писан е на езика на командния интерпретатор shell за операционната система Unix.) Дължината на V-277 (т. е. това, с което увеличава заразените файлове) е само 277 байта. Между другото по света е прието това да се нарича инфективна дължина на вируса.

Вирусът представлява почти пълно копие на V-299 (заедно със всичките му недостатъци), но дългото съобщение в него е премахнато и е заменено от малка програма, която се изпълнява с вероятност 1/2 и която предизвиква PARITY ERROR. Този вирус е изключително рядко срещан, така че, моля ви, не ми се обаждайте всеки път, когато видите съобщение за тази грешка на екрана на компютъра си.

Продължава в следващия брой

СОФТУЕР на годината 1989

Инж. ГЕОРГИ МИРЧЕВ

Повечето от редакциите на големите и авторитетни списания ежегодно провеждат конкурси-класации на програмни продукти и компютърна техника. Така те утвърждават една от най-важните си функции:

– да дават безпристрастна оценка на качеството и постиженията в съответната област и с това да способстват за развитието на софтуера и хардуера;

– компетентно и авторитетно да подпомагат потребителите при техния избор.

Това е възможно, разбира се, при условие, че има пазар – производители, които се конкурират, за да привлекат действителните купувачи, т.е. тези които плащат със собствените си пари и съответно знаят да ги ценят.

Такива конкурси провежда и западногерманското списание Чип. Тук ще ви запознаем с резултатите от петото поредно издание на конкурса за софтуер, някои от които ще съпоставим с по-предния, за 1988 г. конкурс. Крайните оценки се формират по точкова система, като освен журналисти-специалисти от списанието-организатор, в журито участват и журналисти от други специализирани компютърни списания. В класацията за софтуер на 1989 г. са участвали редакциите на Personal Computing (САЩ), ASCII Magazine (Япония), Practical Computing (Великобритания), Komputer (Полша), Impulzus (Унгария), Micros (Испания), CHIP Italien (Италия) и Svet (Югославия).

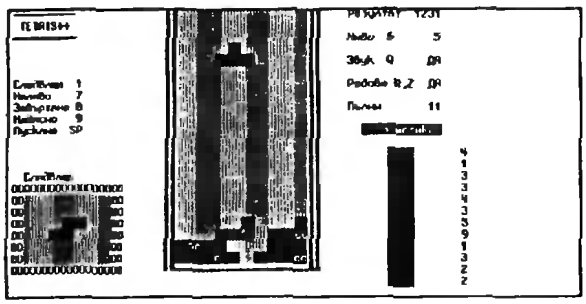
Програмните продукти са разделени в четири групи и се оценяват по своите функционални възможности, качество, удобства при обслужването, съответствие на новите технически стандарти и т.н.

Първото и по-общо впечатление е, че характерният за предните години бум в появата на нови фирми-разработчици е започнал да намалява, по-малко са и новите продукти. За сметка на това фирмите, утвърдили вече популярни и широкоразпространени пакети програмни продукти продължават работата по тяхното по-нататъшно усъвършенстване. В резултат на това едва ли не всяка година се появява поредната версия. Ето защо много от премираните продукти участват в конкурса с петата си версия, а Autocad даже и с десета версия. По същата причина повече от половината отличени продукти, макар и с известни различия в подреждането, бяха в челната класация и за 1988 г.

Забавен софтуер

Tetris	125 т.
Lazy Larry II	110 т.
Populos	100 т.
Rack'em	100 т.
Tenka Toitsu	100 т.
F 19 Stealth	75 т.
Sim City	50 т.
Microsoft Flight	40 т.

За втора поредна година класацията при игрите се оглавява от популярния Тетрис. Тази интелигентно замислена, създадена в САЩ игра, очевидно продължава да увлича младите, пък и не само тях, със своя стратегически характер, изисквания за съобразителност и бързи рефлекси. Тетрис е добре познат и у нас във версии за шестнайсет и осем битови компютри, има и разработен



вариант за домашния компютър Правец-8Д. Като тема за размисъл си струва да се спомене, че цената му в ФРГ е около 200 марки.

Лари, чиито две версии са също познати у нас, изисква добро владеене на английски език, включително и на жаргонни изрази, което естествено ограничава броя на неговите почитатели. Навярно това е и причината за второ място на играта.

Бизнес-софтуер

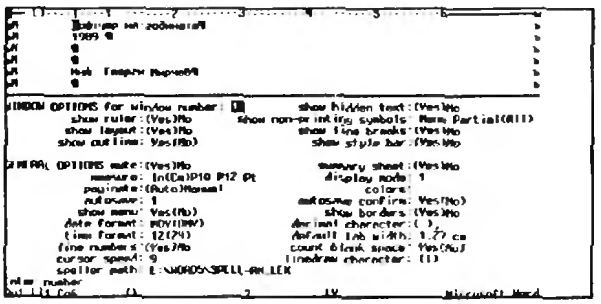
Word Perfect	160 т.
Lotus, v.3.0	135 т.
Lotus, v.2.01	100 т.
Excel	75 т.
Word 5.0	75 т.
Ventura	55 т.
Viewlink	50 т.
Wingz	50 т.

На първите шест места са две текстообработки, три електронни таблици и настолна *издателска система. Любопитното е*, че в сравнение с класацията от по-предната година, сега липсва типичен представител на третата голяма група – базите данни. Вместо това напред е излязла настолна издателска система.

Голям скок прави Word Perfect 5.0. Същата версия през 1988 г. е била едва на деветото място, докато по-предната версия WORD 4.0 заемала седмото място.

Word Perfect 5.0 ярко демонстрира тенденцията в развитието на съвременните текстообработки за интегриране на текст с графика. С възможностите си за изобразяване на екрана на точното оформление на страниците той все по-плътно се доближава до специализираните настолни издателски системи. Продуктът притежава собствен речник за *автоматична правописна корекция* с около 120 000 думи, който може да се обогатява от потребителя. Усъвършенстван е и достъпът и работата с файловете на диска.

Всъщност това са двете текстообработки, които не от вчера си оспорват световното господство. Съкрушителната разлика в набраните точките вероятно се дължи на факта, че WORD 5.0 е от по-скоро на пазара и през второто полугодие на 1989 г. чуждоезичните му версии още не са били готови и той е бил по-слабо познат в страните, чиито списания участват в оценката.



Макар, че това е тема за специална подробна статия, определено може да се каже, че WORD 5.0 едва ли отстъпва на Word Perfect 5.0, а по много показатели очевидно го и превъзхожда – има значително по-богат правописен речник, синонимният му речник съдържа 220 000 думи срещу 80 000 на Word Perfect 5.0, неговата ориентация към управление чрез менюта (включително и с мишка) го прави по-удобен за работа, операциите зареждане и записване на файл, търсене и прескачане в текста, заместване на думи той извършва значително по-бързо и т.н.

Така че едва ли трябва много да съжаляваме, че установилата се традиция е отредила лидираща позиция на фамилията WORD у нас.

Инженерен и научно-изследователски софтуер

Autocad, v. 10.0	180 т.
Mathematica	150 т.
Microstation	80 т.
Fastcad, v. 2.05	75 т.
OR-Cad	60 т.
Cadkey	50 т.
Pointline	50 т.
PC Animator	50 т.

За втора година пред класацията се оглавява от КАД-КАМ програмен продукт, като за 1988 г. Autocad, v. 9.0 е заемал второ място. Придвижването напред е признание на значителното усъвършенстване на новата

версия. Любопитно е да се отбележи, че първата версия на този най-популярен в областта си продукт е от 1984 г.

В сравнение с предшествениците си десетата поред версия предлага значително подобрени възможности за обслужване от оператора – най-вече чрез опростяване на най-често изпълняваните операции. Значително са усъвършенствани такива функции като увеличаване или намаляване изображението на разрези, преместване на прозорци, промяна вида на изгледа, улеснен е достъпът до командите за чертане чрез менюта и др.. Всичко това значително съкращава времето за конструиране на детайлите.

Като известен недостатък на продукта се посочва, че за работа с него е необходим копроцесор, а и цената му е доста висока – 12 000 марки. Същевременно е единодушна оценката, че продуктът напълно си струва парите.

Служебни и помощни програми

Известно е, че операционната система MS-PC DOS не е особено щедра в дружелюбността си към потребителите. Не случайно българският вариант на ръководството за работа с ДОС 3.3 е томче с дебелина 3 см.

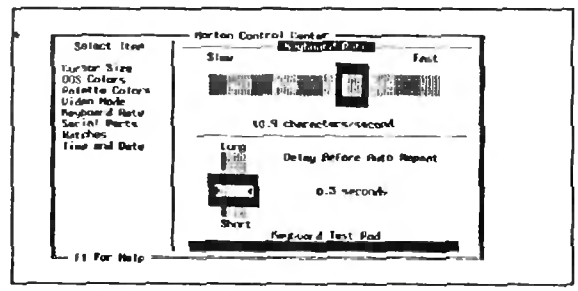
На помощ обаче идват множество помощни програми, които не само улесняват диалога „потребител-операционна система", но и съдържат повече или по-пълен набор от инструментариума, необходим за поддържане на файловото стопанство.

Най-популярни сред популярните пакети са Norton Utilities и PC-Tools, чиито последни версии съвсем закономерно и оглавяват класацията.

Norton Utilities, Version 4.5	140 т.
PC-Tools, Version 5.0	125 т.
Laplink III	80 т.
DOS-Man	70 т.
Lotus Magellan	70 т.
Turbo-Pascal	70 т.
Quick-Pascal	30 т.
Misrosoft C	30 т.

Първите два продукта са много добре познати у нас – както новите, така и по-предните им версии.

Norton Utilities 4.5 представлява пакет от програми, които могат да се изпълняват поотделно или да се стартират от обединеното меню на Norton Integrator (NI/DO за PC XT съвместими компютри). Повечето от функциите на досега съществуващите програми са разширени и допълнени, включително пакетът е пригоден да работи под управление на новия DOS 4.0. Добавени са и нови програми:



– **Norton Disk Doctor (NDD)** – програма за тестване на диска, за възстановяване на информацията след неволно форматиране, за възстановяване на повредени сектори (доколкото това е възможно), за „ремонтиране" на сектора за първоначално зареждане на операционната система, за замяна на дисковата операционна система и т.н.

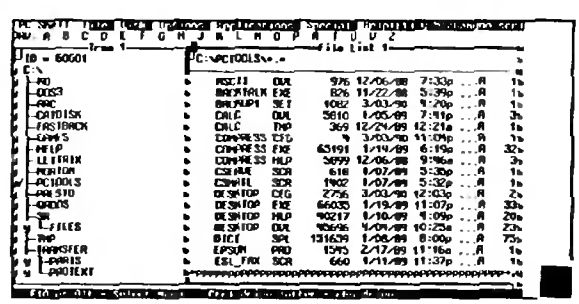
– **Norton Control Center (NCC)** – за настройване на серийните портове на компютъра, за промяна дебелината на курсора и бързодействието на клавиатурата, избиране на цветовата палитра на екрана.

– **Save Format** – усъвършенствана програма за форматиране на дискети и дискове с всички известни до момента стандарти, притежава режим за форматиране, при което се запазва възможността загубената информация да бъде възстановена (с QU), режим за „бързо форматиране" и т.н.

– **Batch Enhancer (BE)** – за създаване на батфайлове с разширени възможности;

– **File Data and Time (FD)** – промяна на датата и часът като атрибути на файловете.

Най-характерното отличие на PC-Tools 5.0 е графичната надстройка на ДОС (PC SHELL), чрез която се облекчава работата с продукта. Интерес представлява и програмата за защита на файловете от нежелано прочитане.



ИВАН БОНЕВ

32-битов микропроцесор Intel 80486

През последните няколко години персоналните компютри *се развиха изключително бързо.* Промениха се и се разшириха сферите на тяхното приложение. *Основните движещи сили зад* този напредък бяха развитието на полупроводниковата технология *и програмното осигуряване.* Преди 5–6 години малко минимашини можеха да се похвалят с памет, по-голяма от 2 Мбайта, и производителност над 1–2 MIPS (милиона инструкции в секунда). Сега тези характеристики са съвсем нормални за един среден персонален компютър. Повишаването на производителността на персоналните компютри досега винаги бе следствие на появата на нови по-бързи микропроцесори. Преди три години старите съперници „Моторола" и „Интел" съобщиха за първите си 32-битови процесори – MC68030 и I80386. Тези два процесора окончателно премахнаха разделителната линия между персоналните компютри, работните станции и минимашините. След по-малко от три години, през лятото на 1989 г., отново същите фирми съобщиха за нови два процесора, които този път заплашват да вкарат съществуващите минимашини в музеите – процесорите MC68040 и I80486. Тъй като все още не разполагам с достатъчно информация за новия процесор на „Моторола", в тази статия ще ви запозная със структурата и възможностите на I80486.

32-битовият микропроцесор I80486 е най-новият член на фамилията Intel 8086. Той е вторият 32-битов процесор на „Интел" след познатия I80386. В архитектурата на процесора са изразени най-новите тенденции в развитието на микропроцесорите. Ултрависоката степен на интеграция с позволила на Интел да реализира в кристала 32-битов процесор (CPU) и копроцесор (FPU), устройство за управление на паметта (MMU), кеш-контролер (Cache) и 8 Кбайта кеш-памет. Интегралната схема съдържа около 1,2 милиона транзистора и е реализирана с CHMOS IV технология, благодарение на което консумира само 6 W при 33 MHz тактова честота. I80486 се произвежда в керамичен PGA корпус със 168 извода.

Новият процесор е програмно съвместим със I80386 и I80387, което позволява използването му със съществуващите операционни системи MS-DOS, OS/2, Windows 386, Xenix и Unix System V/386. Характерно за I80486 с това, че в него са съчетани две, на пръв поглед несъвместими, концепции: CISC (Complex Instruction Set Computer – компютър с комплексен набор инструкции) и RISC (Reduced Instruction Set Computer – компютър с намален набор инструкции). RISC компютрите разполагат с малък набор (няколко десетки) прости инструкции, които обаче се изпълняват за един процесорен такт, докато CISC компютрите имат богат набор инструкции (неколкостотин), чието изпълнение може да заема десетки процесорни цикли. Инженерите на „Интел" са успели да оптимизират така командите на I80486, че най-често употребяваните от тях да се изпълняват за един процесорен цикъл. По тази причина I80486 е около три пъти по-бърз от I80386, който работи на същата тактова честота, и има производителност, характерна за RISC процесор.

Архитектура на I80486

I80486 е построен на базата на 32-битова архитектура и съдържа процесор, процесор за изчисление с плаваща точка, устройство за управление на паметта, кеш-контролер и 8 Кбайта кеш-памет, която се използва както за инструкции, така и за данни. Наборът инструкции на 80486 е напълно съвместим с тези на 80386 и 80387 (съответно с 8086/87 и 80286/287), като са добавени нови инструкции за нови видове приложения и за управление на новите функции на процесора.

Вградените кеш-контролер и 8 Кбайта кеш-памет позволяват да се използват относително бавни динамични памети (с голямо време на достъп), без да се забавя процесорът от въвеждането на цикли за изчакване. Кеш-контролерът позволява изграждането на многослойни кеш-системи, които позволяват по-голяма гъвкавост при изграждането на основната памет. Нов е специалният механизъм, който осигурява надеждна работа на кеш-контролера в многопроцесорни системи. Специална схема следи постоянно външните шини и при нужда обновява съдържанието на кеш-паметта, ако друг процесор е променил данни в нея.

Устройството за управление на паметта (MMU) се състои от устройство за сегментиране и устройство за страниране. Чрез механизма за сегментация се осигурява управление на логическото адресно поле с възможност за преместване на кода и данните, както и осигуряването на зони за глобални ресурси. Страницирането на паметта е слой под нивото на сегментация и то е напълно прозрачно за нея. Страницирането може да се изключва програмно. Тези два

механизма осигуряват реализирането на системи с виртуална памет.

Паметта е организирана като един или повече сегменти с променлива дължина. Всеки сегмент може да бъде с дължина до четири Гбайта и може да има атрибути за позиция, дължина, тип (стек, код или данни) и код за разрешения вид достъп. В многозадачен режим всяка задача може да борави с не повече от 16381 такива сегмента. По този начин на всяка задача се осигурява достъп до 64 Тбайта виртуална памет.

Сегментационният механизъм поддържа четири нива на привилегии, които служат за изолиране на отделните задачи и операционната система. При опит за нарушаване на правата сработва хардуерен механизъм, чрез който може да се осигури проектирането на много надеждни операционни системи.

Както по-старите си събратя I80286 и I80386, новият процесор има два режима на работа – реален и защитен. В реален режим процесорът работи като ултрабърз 8086. Този режим се използва при стартирането на процесора и подготовката на структурите от данни за преминаване към защитен режим. Защитеният режим на работа поддържа вече посочените схеми за привилегии и сегментация, както и виртуална памет. От защитен режим процесорът може да бъде превключен и в един специализиран защитен режим – виртуален 8086 процесор. Този особен режим е задача, която работи под контрола на защитения режим и която може да съжителства с други здачи, като напълно запазва съвместимостта с 8086/88 процесора.

Доскоро недостъпният копроцесор I80387 вече е вграден в самия процесор. Той работи паралелно на аритметично-логическото устройство и може да извършва аритметични действия, включително и изчисленията на трансцендентни функции върху множество типове данни.

I80486 има и вградени възможности за самотестване и хардуерни средства за настройка. При включването на процесора автоматично се тестват вътрешните регистри, кеш-паметта и работата на аритметично-логическото устройство. Резултатите от теста са достъпни в един от регистрите. В четири регистъра за настройка може да се зададат адреси на точки на прекъсване.

Регистри на I80486

I80486 съдържа всички регистри на I80386 и I80387. Регистрите могат да бъдат разделени на четири групи.

Базови регистри

Регистри на базовата архитектура	Брой	Ширина
– регистри с общо предназначение	8	32
– програмен брояч	1	32
– флагов регистър	1	32
– сегментни регистри	6	16

Тези регистри са специфични за всяка отделна задача и се зареждат автоматично при превключването на задачите. 32-битовите регистри са означени EAX, EBX, ECX и т. и. Тези регистри могат да се адресират като 8- и 16-битови с познатите им имена в 8086. Горните 16 бита от регистрите не могат да се адресират отделно. Регистрите могат да участвуват в 8- , 16- и 32-битови операции. Индексните регистри са 32-битови и позволяват линейното адресиране на четири Гбайта памет. Новост в I80386 и I80486 са двата допълнителни сегментни регистъра FS и GS.

Системни регистри

Системни регистри	Брой	Ширина
– контролни регистри	3	32
– системни адресни регистри	4	48

Системните регистри служат за управлението на вградения кеш-контролер, аритметичния копроцесор и механизмите за сегментация и страниране. Тези регистри са достъпни само на програмите, изпълнявани в ниво на привилегия 0 (най-високото ниво). Те включват три контролни регистъра (CR0, CR1 и CR2) и четири регистъра за сегментация (GDTR, IDTR, TR и LDTR).

Регистри на копроцесора

Регистри на копроцесора	Брой	Ширина
– регистри за данни	8	80
– регистър за маркери	1	16
– статус регистър	1	16
– указатели на иструкции и данни	2	48
– контролен регистър	1	16

Регистрите на вградения аритметичен копроцесор са напълно идентични с тези на I80387. 80-битовият регистър за данни е разделен на мантиса (64 бита), експонента (15 бита) и знак (1 бит). Регистрите за данни са достъпни чрез стеков механизъм.

Регистри за настройка и тестване

Регистри за настройка и тестване	Брой	Ширина
Адрес на точка за прекъсване	4	32
Статус на точката за прекъсване	1	32
Регистър за контрол на точки за прекъсване	1	32
Кеш тест данни	1	32
Кеш тест статус	1	32
TLB тест контрол	1	32
TLB тест статус	1	32

Първите шест регистъра служат на механизма за подпомагане на настройката на програми. Потребителят може да зареди линейните адреси на четири точки на прекъсване и посочи в контролния регистър кои от тях в кой режим на достъп до паметта са активни. Процесорът хардуерно следи за появата на тези адреси и генерира прекъсване при появата им на адресната шина. Тест регистрите служат за проверка на кеш- контролера, кеш-паметта и буфера за паралелна обработка на инструкциите.

Производителност

Новият процесор на фирмата „Интел" само 24 часа след официалното си обявяване бе показан „присаден" на специална платка в компютъра PS/2 Model 70 на фирмата IBM.

Инженерите на IBM са изпробвали I80486 с DOS, OS/2 и AIX (Unix операционната система на IBM). Първоначалните проверки са показали удвояване на скоростта спрямо I80386, който работи със същата тактова честота. В момента „Интел" произвежда серийно 25 MHz и 33 MHz варианти на I80486. До края на 1990 г. трябва да се появи процесор с 40 MHz тактова честота, а до две години — 50 и 60 MHz вариаити. Според „Интел" производителността на 25 MHz варианта е 37000 Драйстоуна за секунда (Dhrystone benchmark) или 15 VAX MIPS. Образци на I80486 персонални компютри, показани на CeBit'90 в Хановер, са *демонстрирали* удивителните 120 MHz AT по Landmark Speed Test. (Производителността на компютъра в този тест се оценява спрямо производителността на стандартен PC/AT 6 MHz компютър. Цифрата 120 означава, че тестваният компютър има производителност, еквивалентна на IBM PC/AT с тактова честота 120 MHz).

Горепосочената производителност може да предизвика завист *в множество мини и да даде* големи машини. В коментарите на няколко авторитетни източника се споменава, че „I80486 персоналният компютър може да прогори дупка на бюрото Ви...". Несъмнено е дошла ерата на персоналните суперкомпютри, но проблемът с

програмното осигуряване

все още остава открит. Разглеждахме прекрасните възможности на I80486 процесора да работи в защитен режим, с виртуална памет и т. н. Нека обаче си спомним, че I80286 и I80386 процесорите също поддържат тези възможности. Не се оправдаха очакванията за Microsoft OS/2 — операционната система, която трябваше да реши проблемите с многозадачните си възможности и виртуалната си памет. Все още OS/2 не може да получи широко разпространение, а тя е писана за I80286 процесора. Липсва и необходимата широка гама от приложни програми, която да използва възможиостите *на OS/2. Появата на високопро*изводителния I80486 може би ще стимулира разпространението на Unix подобните операционни системи. И така, до появата на операционната система и приложни пакети за новите процесори, ние, потребителите на персонални компютри, ще трябва да коригираме фаталните си асемблерски грешки с изключване на компютъра *и да махаме всички* резидентни програми, за да осигурим на капризните програми свещените 640 Кбайта памет. А дотогава 32-битовият персонален компютър ще е нещо като състезателен автомобил от Формула-1, управляван от любител шофьор. ■

Тази статия е написана в резултат на активна експлоатация на MS DOS 4. 01 в продължение на почти шест месеца. Тя има за цел да помогне на потребителите на тази версия, които не разполагат с документация за нея, както и на потребителите на MS DOS 3. 30, които имат желание или намерение да преминат към работа на 4. 0.

Повече за DOS 4.01

Инж. АЛЕКСАНДЪР ТОМОВ

Да се разкаже за операционна система с обем само на потребителската документация (User's guide and user's reference) около *600 страници в рамките на една* статия е невъзможно, дори и при максимална лаконичност. Затова в тази статия е направен опит за сравнение на версия 4.01 с добре позната 3.30, за която има литература и на български, като са посочени разликите и новите възможности на MS DOS 4.01, които са по-съществени.

Първото, което прави впечат*ление, е значително нарасналият* обем на самата операционна система (разпространява се на 6 дискети по 360 Кбайта) и документацията ѝ (заедно с описанието на системата SHELL и вградения GM — BASIC ръководството на потребителя надхвърля 1000 страници). Освен това съществено е повишена дружелюбността на системата,

изразена както в процедурата за инсталиране, в което има и елементи на генериране, така и във вградения потребителски графичен интерфейс, реализиран чрез SHELL – надстройката.

Значително с нараснала гъвкавостта на системата при инсталиране – предлагат се алтернативни варианти на съотношението между резидентно заеманата оперативна памет и възможностите: избор на драйвери за печатащи устройства в зависимост от типа и броя им (за съжаление малцина у нас разполагат с тях), автоматично стартиране на SHELL-надстройката от файла AUTOEXEC или стандартното му завършване и издаване на системен известяващ знак (промпт), избор на кодови таблици, формат за дата и време, паричен знак и десетичен разделител, приети в съответната страна, „опипване" на конфигурацията и настройка според нея (например според монтирания видеоадаптор се инсталира съответният драйвер) и др.

Инсталиращата процедура генерира по един примерен AUTOEXE.BAT и CONFIG.SYS, като запазва съществуващите, ако има такива.

Кои са все пак новите възможности, предоставяни на потребителя на MS DOS 4.01? В документацията производителят посочва следното:

1. Премахнато е ограничението за обем на раздел на твърдия диск от 32 Мбайта. По-подробно за това може да се прочете в [4].

2. Активно използване на разширенията на оперативната памет – expanded от 640 Кбайта до 1 Мбайт и extended над 1 Мбайт (само за системи, базиранн на I80286/386/486) за разполагане на файлови буфери, данни за FASTOPEN, RAM и CACHE драйвове.

3. Включена е нова команда MEM, която извежда карта на оперативната памет с резидентно разположените модули на DOS и потребителски програми заедно със съпътстваща информация.

4. Разширена е поддръжката на специфичната национална информация (добавени са четири държави, но за съжаление България отново не фигурира в списъка на „уважените"), като незначителна модификация на някои модули или замяната им със съответните модули на SPS DOS 3.30 позволява да се избегнат неудобствата.

5. Усилена е поддръжката на EGA и VGA видеоконтролерите – преработени са модулите MODE и GRAPHICS за нормална работа с EGA и VGA видеоконтролери.

6. Модифицирана е командата FASTOPEN за използване на допълнителната (expanded) памет.

7. Добавен е параметър „/p" във формата на командата DEL за изискване на допълнително потвърждаване на изтриване на файл при изпълнение на командата.

Общо съществено са изменени и усилени следните команди:

APPEND – добавена е възможност за изключване на ключа „/x" и индикация на обработваните (необработваните) файлове;

ATTRIB – измененията не променят начина на използване;

BACKUP, RESTORE – промените ги правят несъвместими с тези от по-стари версии;

COUNTRY – поддържат се допълнително четири държави;

DEL – добавен е параметър „/p" за изискване на допълнително потвърждаване при изтриване;

FASTOPEN – добавен е параметър „/x" за разполагане в допълнителната (expanded) памет на данните и е възможно посочване на броя на фрагментите на файла.

FDISK – добавена е възможност за работа с том или раздел с обем над 32 Мбайта;

NLSFUNC – вж. COUNTRY;

REPLACE – добавен е ключ „/u" за подмяна само на тези файлове от присмащия справочник, които са по-стари от тези в справочника-източник;

SELECT – процесът на инсталиране е значително променен.

„Усилени" са следните драйвери:

ANSI.SYS – добавен е ключ „/x" за работа със 101-клавишни и други разширени клавиатури;

DISPLAY.SYS – модифициран е за поддръжка на VGA видеоконтролер в пълните му възможности;

DRIVER.SYS – не се поддържат 8-инчови устройства, които явно са излезли от употреба, но в DIVPARM е добавен ключ „/1" за поддръжка на допълнителни 3,5-инчови устройства, в случай че ROM – BIOS на компютъра не ги поддържа, както е в някои по-стари модели IBM AT и съвместими с тях;

PRINTER.SYS – напълно се поддържат някои по-нови модели принтери.

Добавени са и нови драйвери като:

XMAZEMS.SYS – драйвер за използване на допълнителната (expanded) памет (EMS) по стандарт LIM;

HIMEM.SYS – драйвер за използване на разширената (extended) памет (XMS);

EMM386.SYS – драйвер за емулиране на допълнителната (expanded) в разширената (extended) памет в конфигурации с процесори I80386/486;

SMARTDRV.sys – драйвер за работа с кеш-паметта за ускоряване на входно-изходните операции при работа с твърд диск.

Опитът на продължителна експлоатация на новата версия с машини XT (4,77 MHz) и AT (6 – 12 MHz) позволява да се направят следните изводи:

1. Новата версия 4.01 на MS DOS е качествен скок в областта на еднозадачните операционни системи. Разликите между версиите 4.01 и 3.30 са значителни в сравнение с разликите между 3.30 и 3.20 или 3.10.

2. Появата на MS DOS 4.01 е отговорът на Microsoft на появата на нов хардуер: системи AT, AT 386/486 със значително по-мощни процесори и обем на оперативната памет над 640 Кбайта, периферия – 3,5-инчови флопидискови устройства, твърди дискове с обем значително над 32 Мбайта.

3. За съжаление от съображения на програмна съвместимост не е преодоляна границата на 640 Кбайта оперативна памет, макар че усъвършенстването използване на разширената памет expanded и extended значително ускорява работата на задачи с голям брой входно-изходни операции (т. е. повишава производителността на системата) и разтоварва стандартно достъпната оперативна памет от 640 Кбайта.

4. Промененият размер на елементите на FAT за твърди

дискове с размер на раздела по-голям от 32 Мбайта може да предизвика проблеми при работа с някои програми, които организират собствен (без да ползват функциите на DOS) достъп до диска; някои вируси например са безсилни в среда на DOS 4.01.

5. Опасенията от грешки, изказани в [4], се оказаха, поне засега, неоснователни — в продължение на 6 месеца активна работа MS DOS 4.01 не предизвика никакви проблеми, освен с някои функции на PCTOOLS—4.21 и Norton Utilities, заради променения размер на елементите на FAT. Препоръчително с използването на PCTOOLS—5.01, който освен това с по-мощен и поддържа мишка. Не са *наблюдавани никакви проблеми* с кирилизиращи програми за клавиатура, видеоконтролери и принтери.

6. Значителното усложняване и „усилване" на MS DOS 4.01 в сравнение с предишните версии отнема на компютъра повече време за служебни операции, като забавянето става чувствително при XT (4,77 MHz) конфигурации (авторът не е експериментирал с 10—12 MHz XT), затова *използването на* 4.01 в повечето случаи е оправдано при по-мощни компютри, като AT и други, базирани на процесори Intel 80286/386/486 с голяма оперативна памет.

Още повече че голяма част и от по-значителните предимства на новата версия се реализират именно при такива конфигурации. Нещо повече, за тях тя е желателна, защото използва по-пълно мощния хардуер и дава значителни *предимства пред* DOC 3.30 и по-старите версии.

Литература

1. Microsoft MS DOS 4.01. User's guide and user's reference.
2. Microsoft MS DOS 3.30. User's guide and user's reference.
3. Microsoft MS DOS 4.01 Shell. User's guide and user's reference.
4. Г. Балански. Първо запознанство с IBM PC DOS 4. 0. КВ.04.89.

МНЕНИЕ

Боледува ли нашата електроника

Прочетох с интерес увод-ната статия на инж. Г. Балански в последния брой на списанието (б. р. — КВ. 1—2. 90). Като се присъединявам към оценката за отчайващата изостаналост в елементната ни база, искам да изразя мнението си, че ние не можем да развиваме електроника само с разработката на програмно осигуряване, пък дори и то да е с високо качество. Проблемите на производството на интегрални схеми у нас отразяват в умален вид проблемите на цялата ни промишленост — монополизъм, липса на сериозна инвестиционна програма, волунтаризъм при вземането на решения и ниско технологично ниво. Поради изключително динамичното развитие на технологията на производство на интегрални схеми в световен мащаб обаче изоставането у нас в тази област е наистина отчайващо. Досега инженерите по електроника, хората, които са свързали своето настояще и бъдеще с нейното развитие, нямаха възможност да изразяват своето мнение за проблемите на електрониката и пътищата за тяхното решаване. Смятам, че сп. „Компютър за вас" може да допринесе много за това, ако открие една нова рубрика „Проблеми на електронната ни промишленост", в която да дава думата на специалисти и ръководители от определени области от електрониката, като отделя място и за писма на читатели.

Ст.ас.инж. ОГНЯН ЖЕЛЕЗОВ

Редакцията винаги е подкрепяла, а сега това е още по-необходимо, предложения като съдържащите се в писмото на инж. Железов. Готови сме за постоянно да откриваме не само рубрика „Проблеми на електрониката ни промишленост", но и „Проблеми на софтуерната ни индустрия и пазар" и т. н. Списанието ще публикува с предимство всички статии, които биха допринесли за *развитието на компютризацията у нас.*

Очакваме мнението на специалистите, а от **нас — добро заплащане за добре свършената работа.**

КОМПЮТЪР ЗА ВАС

Обяви

Фиг. 1. Главно меню на информационна система преди активиране на Маестро

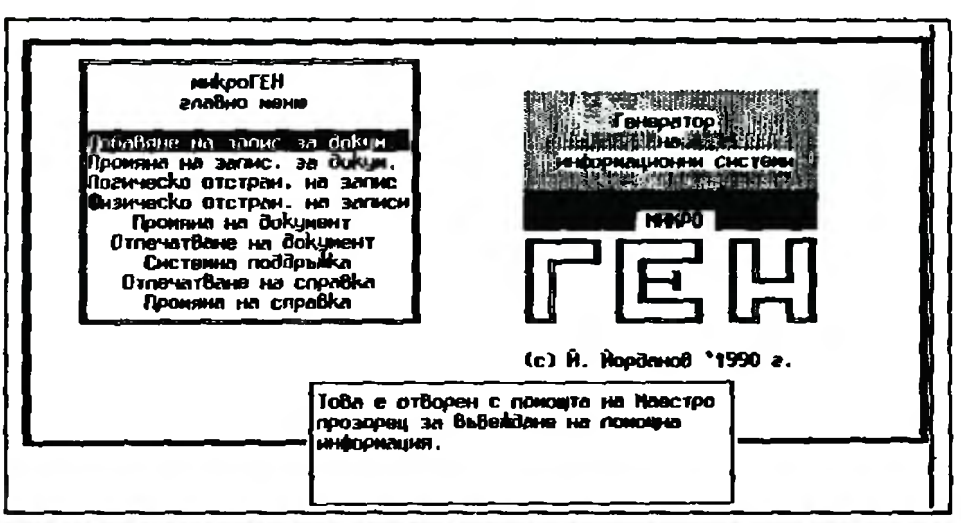
Помогни си сам

или

ПОЛЗАТА ОТ МАЕСТРО

ЙОРДАН ЙОРДАНОВ

16



Фиг. 2. Прозорец с помощна информация, създаден с Маестро

Повикът за професионализъм при производството на програмни продукти и услуги отдавна е факт. Факт поради множеството фирми и организации, които търсят „място под слънцето". Факт, който се налага от високо вдигнатата летва от производителите на Запад, от големия интелектуален заряд, влаган в този вид стока, и от нейното предназначение за масовия потребител.

Първоначално и на Запад програмистите бяха хора с манталитета на работещи за себе си, т. е. те правеха програмни продукти, за които знаеха, че ще се експлоатират от хора също с познания в тяхната професия. Но с широкото разпространение на персоналните компютри нещата коренно се промениха. И водещите производители на програмни продукти за микрокомпютри бързо се ориентираха в новата обстановка — осигуряване на обилна помощна информация за работа с програмния продукт, подробно (до нивото на детски буквар) ръководство за работа, т. нар. hot line – дежурство на специалист на определен телефон на разположение на потребителя и др.

Всичко това доведе до няколко важни за потребителя и особено за производителя на програмни продукти ефекта:

— намалява се времето, необходимо за усвояване на програмния продукт от крайния потребител, което пести човекоресурси и от двете страни;

— увеличава се доверието между производител и потребител, което намалява страха и разширява свободата на потребителя за идеи за нови разработки;

— премахва се почти изцяло необходимостта от заделяне на интелект от потребителя за използване на програмния

продукт като инструмент в своята работа;

— дава се възможност на производителя да излиза с приложения във все нови и нови области на човешката дейност, без за това да му е необходимо да си изгражда „междинни" специалисти (например метеоролог — програмист). Просто помощната информация се пише от специалист — потребител, а програмата от специалист — програмист.

Как стоят нещата по повдигнатите въпроси у нас?

Ниската цена на труда на програмиста и липсата на ефективни закони за защита на програмните продукти от неправомерно разпространение първоначално ориентираха нашите програмисти към т. нар. адаптация на западни продукти. Бързо, евтино и без много мислене. Но бумерангът много бързо се върна обратно.

Невъзможността от качествен превод на западните продукти вместо приближаване на продукта до потребителя го отдалечи. Съгласете се, че българинът трудно може да се досети какво означават множеството преводи на английската дума cut (режи), при това съкратени до три букви например РЖИ. Да не говорим за отговора на въпроса „Съгласни ли сте Y/N". Изобщо вместо приобщаване на клиента с последващо разширяване на пазара по негови инициативи се получи точно обратният ефект.

Напоследък обаче има шансове нещата да се променят. Първата ластовичка вече като че ли дойде. Името ѝ е Микросистеми. Тази фирма като че ли първа се сети за могъщата сила на рекламата. Ръководствата на нейните програмни продукти вече по нищо не отстъпват на ръководствата на западни продукти. Показателен е примерът с ръководството на МикроТЕКСТ II. Добра е идеята новите продукти, преди да се пуснат на пазара, да се дават на специалисти — потребители, които „да си поиграят" с тях, както стана с Шампион и Престо.

Сега фирмата е решила да отговори на острата потребност от доближаване на програмните продукти и на другите производители до „ухото" на българския потребител. Това може да стане с най-новия им продукт **Маестро.**

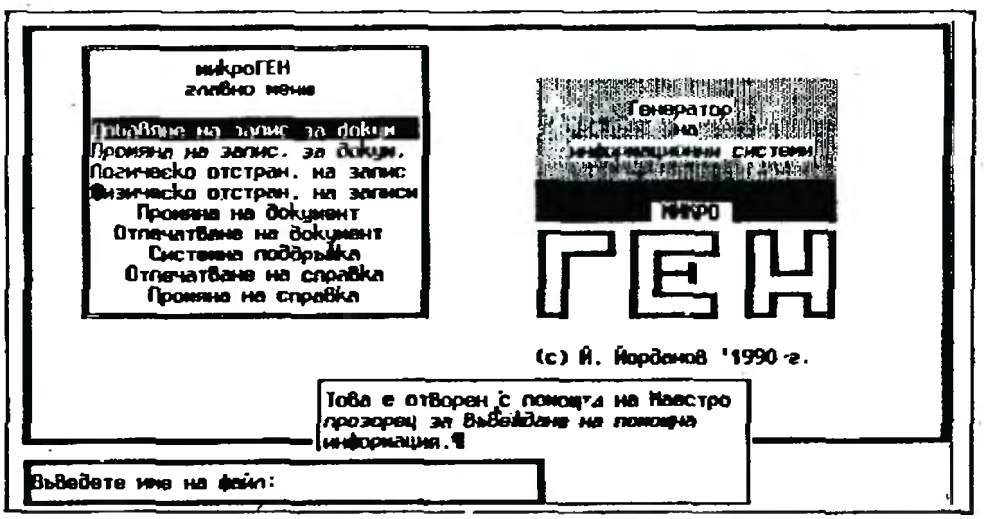
Всеизвестно е, че най-широко разпространените офис системи у нас са създадени на базата на Микрофайл 16, dBase и компилаторите Кобра и Clipper.

Именно към тях **Маестро** *позволява с лекота да се изграждат подобни системи с помощна информация, съставени от екрани и менюта. Помощната информационна система се създава независимо от процеса на създаване на обработващите програми. Това позволява на производителя да въвежда паралелизъм в работата си или да поръча написването на помощната система на външен професионалист в областта на прилагане на готовия продукт. Това е и шанс за тези, които искат да направят по-дружелюбни вече готовите си програмни продукти.*

Освен това **Маестро** *предоставя възможност за „закачване" на текстови блокове с помощна информация на всяка стъпка от работата на основната програма — избор от меню, въвеждане на стойност или текст или подаване на команда. Продуктът притежава свой редактор за въвеждане на текстовете и „рисуване" на екраните и менютата.*

Предвидена е възможност за „зареждане" на текстове, подготвени с какъв да е редактор, за да се ограничава този, който ще пише помощната информация.

За специалистите в разработването на информационни системи ще кажа, че **Маестро** *е един интелигентен генератор на файлове Help.prg, който се активизира по подразбиране с натискане на клавиша F1.*

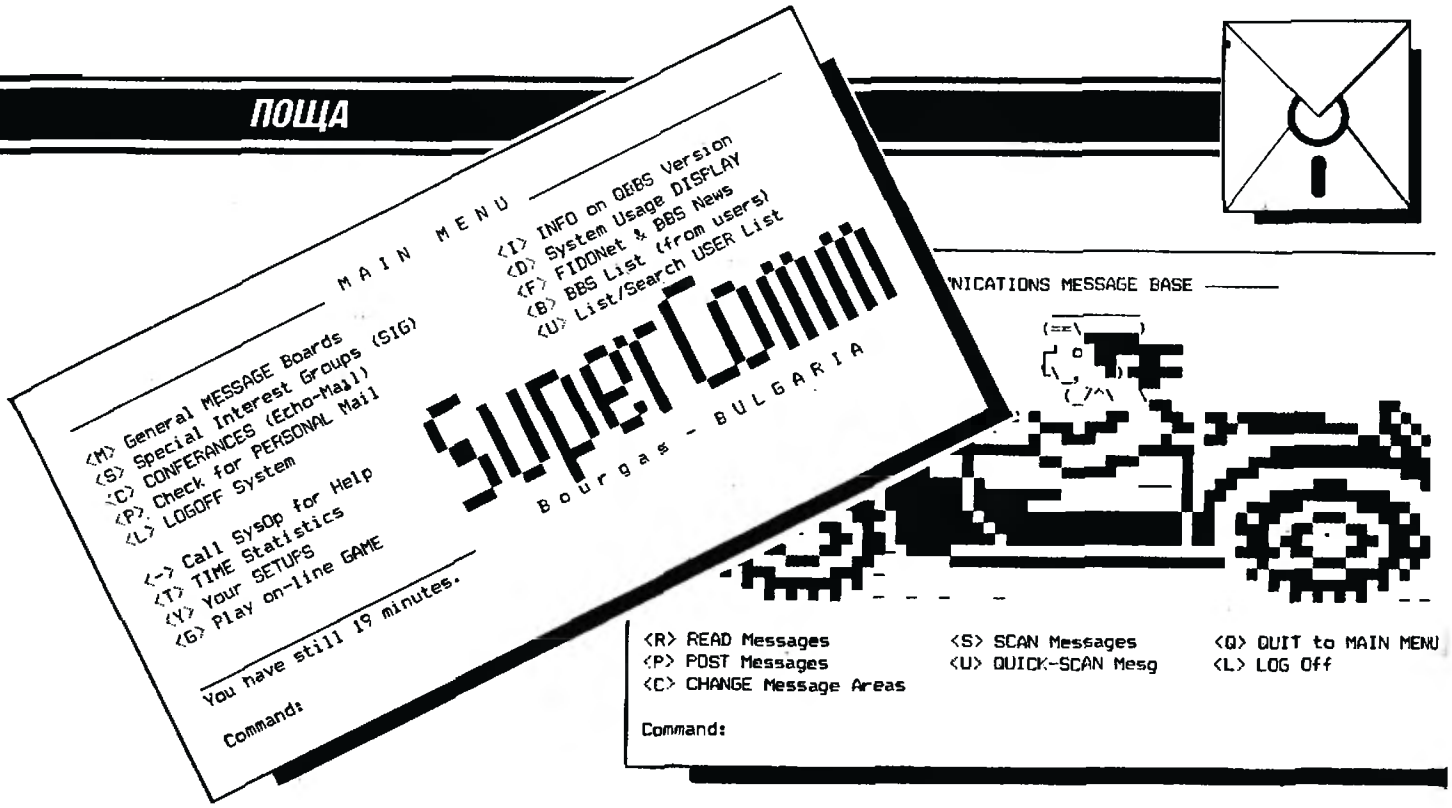


Фиг. 3. Създаване на втори прозорец с информация

Не знам колко са разработчиците на системи у нас на езицита на фирмата „Борланд"—Турбо Паскал и Турбо Си, но подобна система за тях се предлага от фирмата „Blaise Computing". Тя също дава възможност към вече направена програма да се добави интелигентна помощна информационна система.

На останалите разработчици бих препоръчал, ако на друго, то поне създаването на помощни файлове с Norton Guide на „Peter Norton Computing". Вярно трябва малко повече труд, но бъдете сигурни — инвестициите ще се възвърнат.

■



Драги колеги,

Предлагам на вашето внимание кратка информация за първата действаща в България от началото на 1990 г. любителска компютърна система от типа BBS (Bulletin Board System), аналогична на многото такива системи, които действат в САЩ и в Западна Европа. За повече подробности можете да прегледате януарския брой на сп. „BYTE" от тази година, където е направена равносметка за неколкогодишно бързо развитие на системите BBS (по-специално в САЩ) и на бъдещите им перспективи. Това са изцяло любителски системи, персонални или на групи любители, като например една от най-големите BBS в САЩ е тази на Боб Махони, който има 96 (!) телефонни линии (модемни канала) в сутерена на жилището си и получава над 3000 обаждания (респ. потребители) на ден.

Една от организираните по този начин любителски компютърни мрежи е FidoNet (подобни са и мрежите *LCRNet, AlterNet и други). FidoNet обаче е най-голямата световна любителска компютърна мрежа и географски е разделена на три зони:*

Зона 1 — Северна Америка

Зона 2 — Европа

Зона 3 — Тихия океан и Далечния изток.

Зоните са разделени на райони, които от своя страна са разделени на мрежи, а всяка индивидуална BBS е възел в границите на една мрежа.

С FidoNet всичко започва през юни 1984 г. в САЩ, когато Том Дженингс и неговият приятел Джон Мейдил, които живеят в отдалечени щати, решават да установят пряк начин за комуникация помежду си. Така започват да се предават и първите съобщения в двете посоки, пресичайки цялата страна. Скоро и други оператори на подобни BBS се заинтересуват от връзките между различните щати и се присъединяват към двамата ентусиасти. През август 1984 г. FidoNet вече има 30 възела (nodes), а през пролетта на 1985 г. — около 200! Днес тя обхваща целия свят и има няколко хиляди абонати.

Според консултациите с координаторите на FIDO в Европа, досега е правен опит за такава система в ЧСФР, но дори в началото на 1990 г. в страните на Източна Европа все още няма регистрирани членове на FidoNet. Поради проявения интерес от страна на колегите от Парижката регионална мрежа FIDO към евентуално създаване на BBS у нас и благодарение на оказаната ми безвъзмездна помощ от тяхна страна днес са вече факт ПЪРВИТЕ СТЪПКИ и у нас по създаването на любителска компютърна мрежа, интегрирана в мрежата FidoNet на страните от Западна Европа и САЩ.

Системата-ми в Бургас се нарича SuperComm BBS. Имам голямото желание да помогна на всички, които обичат компютрите (любители и професионалисти), в по-преките им контакти, общуване, обмен на информация, програми и т. н. Системата има вградени възможности за максимум 200 електронни пощи за съобщения, въпроси и отговори от всеки регистриран потребител (всяка поща е с отделна тематика или с отделно предназначение; има и Echo-Mail, Net-Mail). Системата ми е напълно безплатна и достъпна за всички любители (добронамерени и честни). Системата поддържа детайлен дневник на дейността на всеки включил се в нея потребител и дава възможност за максимум 32000 нива на достъп до отделните менюта. Всеки може (в зависимост от нивото си) да сваля или качва файлове (download/upload, които го интересуват или за които смята, че ще бъдат интересни и за други потребители на системата. Обществения достъп на тази софтуерна библиотека, разбира се, позволява да се разпространяват само програмни продукти от тип FREEWARE/SHAREWARE, но не и комерсиализиран софтуер (ако някой все пак качи такъв, системният оператор ще го изтрие от диска на системата). Достъпни засега са

около 8–10 протокола за обмен – Xmodem, Windowed_Xmodem, Ymodem, Batch_Ymodem, Zmodem, Lynx, Kermit, BiModem и ASCII.

Засега могат да се обслужват потребители на компютрите IBM PC/XT/AT и PS/2, Apple, Commodore, Amiga, MicroPDP и MicroVAX. Но самата система е изградена на базата на компютър IBM – Turbo AT/286 (и IBM/2 като допълнителен), специализирано програмно осигуряване (около 5–6 Мбайта, (C) 1989/90 САЩ) и интелигентен сериен асинхронен телефонен модем, който работи по стандартите (CCITT или Bell):

– v21: 300/300 bd, FDX, 8/N/1;
– v22: 1200/1200 bd, FDX, 8/N/1;
– v23: 1200/75 bd, FDX, 8/N/1.

Тези, които притежават компютри, съвместими с IBM/PC, могат безпроблемно да използват и графичните (ANSI) възможности на системата SuperComm BBS (8-битовата аскитаблица IBM/MIK и възможностите, които предоставя драйверът ANSI.SYS от IBM DOS – PC).

За свързване към системата SuperComm BBS са нужни поне терминал (или компютър със сериен интерфейс RS–232–C и необходимата му телекомуникационна програма, като например IBM–PC с програми като ProComm, CrossTalk, Telix, Телемост–16 и др.) и, разбира се, телефонен модем, включен към терминала (или компютъра) и към телефонната мрежа. Ако модемът съответства на някой от посочените стандарти и е правилно настроен, то остава само да се избере телефонен номер (056) 287–301. След второто позвъняване там автоматично ще отговори модемът на SuperComm BBS, като трябва да се изчака появяването на първите съобщения от системата.

Голямата трудност при въвеждането на такива групови комуникационни системи у нас идви единствено от факта, че другите народи в Европа и САЩ използват латиница, а само ние – кирилица. Още повече че у нас имаме 7- и 8-битови аскитаблици с несъвместимо разполагане на кирилицата. По този начин за различните разпространени у нас компютри (дори само да се посочат Правец–82 и Правец–16) единствената обща част от азбуката с оригиналните аскитаблици са само големите букви: от A до Z.

Като се има предвид, че системата SuperComm BBS се изгражда за първи път у нас като опит за редовен абонат (възел) на FidoNet в Европа и че от международна гледна точка е необходима комуникация на английски език, то всички менюта в системата са на английски език. Разбира се, самите съобщения в електронните пощи могат да се пишат, като се използват графичните символи от 8-битовата IBM аскитаблица (т. е. с кирилица, но само за притежателите на компютри, съвместими с IBM PC, и знаков генератор по таблицата MIK).

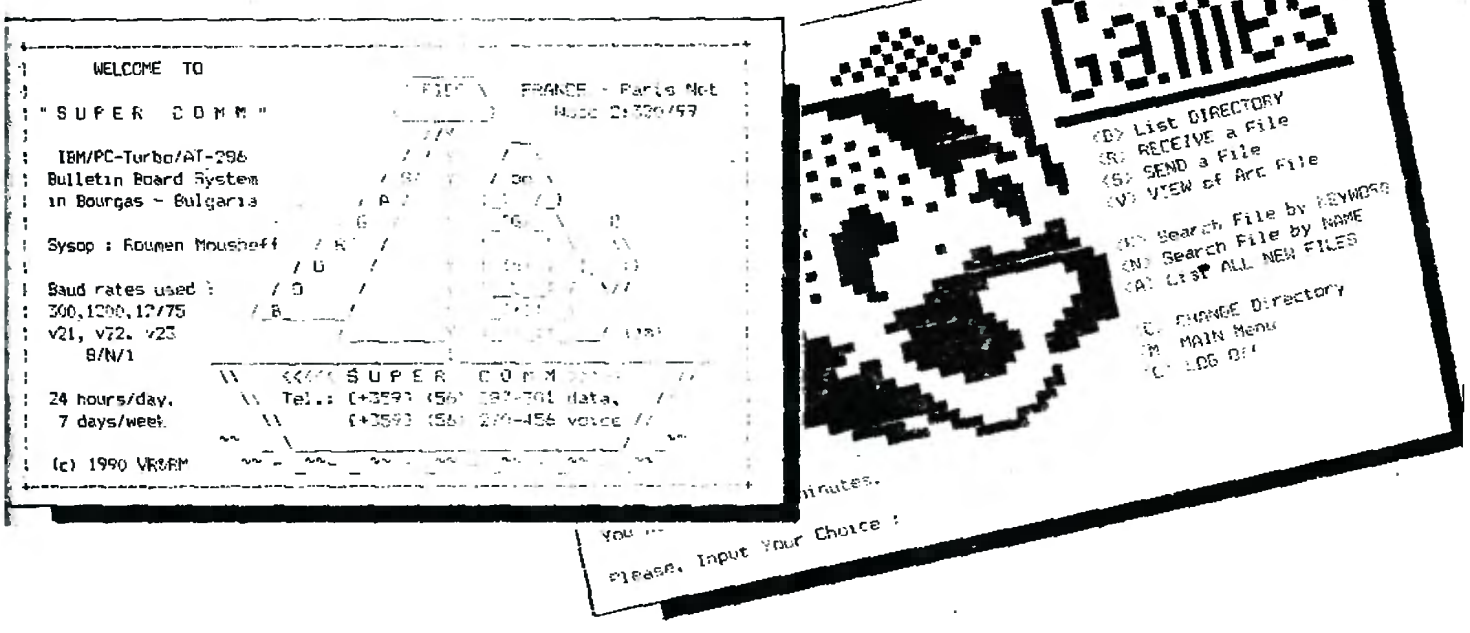
Не съм имал досега време и възможност за популяризиране на системата и тя е все още неизвестна у нас. Но ако към нея има интерес и се желае диалогът да бъде и на български език, то изображенията се генерират съвсем ясно и това не е проблем (но пак повтарям – само по таблицата MIK, като за Правец–82 и другите компютри със 7-битова аскитаблица изображенията няма да могат да се четат).

Системата SuperComm BBS предлага почти всички възможности на подобните BBS в Западна Европа. И бих се радвал, ако съобщите за нея на всички любители на компютрите у нас. Ако проявите интерес към системата, бих могъл да ви изпратя по-подробна информация за нея и да запозная интересуващите се читатели с големите възможности, които предлага сферата на обществените комуникации. Разбира се, най-доброто запознаване с една система става, когато човек се включи веднъж в нея. Ето защо приветствам всички потенциални потребители на системата с „Добре дошли!" и им пожелавам успешна работа.

Напомням, че системата SuperComm BBS в Бургас работи непрекъснато за всички любители на компютри.

С уважение към всички колеги от трудната (у нас) област на информатиката:

Инж. РУМЕН МУШЕВ
системен оператор на **SuperComm BBS**
8000 — Бургас ул. 9 септември 77
дом. тел. (056) 8-75-45
служ. тел. (056) 270-456



Инж. ВЛАДИМИР БОЧЕВ

Как да използваме PKARC[1]

PKARC, или така наречения архиватор, е една от разпространените програми за сгъстяване на файлове. Тя използва описаните в предишните броеве на списанието методи Lempel-Ziv-Welch и Huffman за сгъстяване на данни, както и цикличен код с пораждащ полином CRC16 за надеждно съхранение на данните във файловете.

Описанието на програмата, предназначено за потребителя, е доста обемисто, затова тук ще се опитам да извлека само онова, което е необходимо на широкия потребител за правилна и надеждна работа. Например няма да описвам различните начини за постигане на съвместимост с по-стари версии, тъй като надали някой ги ползва (версията, за която тук става дума, е 3.5).

Като начало, нека да стартираме програмата така: C > pkarc/h. Ако след командата не се зададат параметри, на екрана на дисплея се извежда кратко справочно меню. Различните опции могат да се използват както поединично, така и в комбинации.

Значението им е следното:

– a – прибавя нови файлове към съществуващ вече архив или създава нов архив и въвежда в него посочените файлове. Ако не се подадат имена на файлове за въвеждане в архива, това ще се възприеме като команда за архивиране на текущата директория. Имената на файловете трябва да са разделени с интервал и тези файлове трябва да съществуват.

– u – подновява файлове в архива, като подновяването става само ако некомпресираният файл е с по-нова дата от файла със същото име в архива. При създаване на нов архив или прибавяне на нови файлове действието е аналогично на предишната опция.

– f – опресняване на архива. По принцип почти същото като опция „u". Най-често се използва така C > pkarc f archive.arc a:*.* По този начин ще се претърсят всички файлове от диск A: и когато се намери файл с по-късна дата на създаване от съществуващия вече в архива, се извършва подновяване.

m – действа аналогично на опция „a", като освен това изтрива всички подадени за въвеждане оригинали. Изтриването става само след напълно успешно (без съобщения за грешки) архивиране на подадените оригинали.

– d – изтрива от архива определените на командния ред на програмата файлове.

– g – защитава даден архивиран файл от нежелано извличане от архива с помощта на ключова дума. Например: C > pkarc agkey archive.arc file. По посочения начин към архива archive.arc ще прибавим файла file, който впоследствие ще може да бъде извлечен само ако се въведе правилната ключова дума, в случая „key". Опцията „g" трябва да се въвежда последна и след нея трябва да се напише без интервали ключовата дума. Това е отчасти опасна опция, тъй като ключовата дума лесно може да се забрави.

– v – тази опция дава информация за някои технически параметри на архива. За всеки файл се посочват неговата дължина, методът, по който е извършено сгъстяването, дължината след сгъстяването, с колко процента даденият файл е станал по-малък, датата и времето на създаване на файла и накрая изчислената стойност CRC за файла.

c, x – тези две опции са създадени, за да може потребителят да въведе собствена информация в архива под формата на текст (коментар за даден файл или архив). Разрешената дължина на потребителския текст е 32 знака. Опция „c" позволява въвеждането на коментар за даден файл, докато опция „x" важи за архива като цяло. Тези опции могат да се използват както самостоятелно, така и записани след опциите „a, u, f, m, v". В първия случай може да се прибави или промени даден коментар, а във втория потребителят „се подканва" да въведе съответен текст (коментар). Ако първа е опцията „v", то коментарите се появяват на мястото на информацията за дължина, метод на сгъстяването, дължина след сгъстяването и фактор на сгъстяването в проценти.

Дотук стана дума само за архиватора. Дадената информация е напълно достатъчна за правилна работа. За да можем да извличаме файлове в оригиналния им вид, ще ни бъде необходима и програмата PKXARC. Извикването на програмата е същото както по-горе, PKXARC, опции, имена на файлове. Следва описание на тези опции.

– e, – x – тези опции са вградени, т. е. не е необходимо да се въвеждат. Въведени са само за съвместимост с други архиватори. Ако обаче посочените опции са въведени, то те трябва да са единствени (не могат да се комбинират с други). Предназначението им е да извличат файлове от архива и да ги възстановяват в нормален (несгъстен) вид.

– r – обикновено PKXARC пита дали да се възстанови даден файл, ако на мястото, където трябва да се възстанови файлът, вече има такъв със същото име. С тази опция желаните файлове се възстановяват, като е възможно да се запишат върху файл със същото име (ако има такъв).

– c – Файловете се извличат на екрана на компютъра

– p – Извлича файловете върху принтер, като след всеки

[1] *Това е първата архивираща програма, с която започна бумът на архиваторите у нас. След това последваха PKPAK, ZIP, ZOO и т. н. Практически указания за тях очаквайте в следващите броеве – Б. Р.*

файл принтерът се позиционира на нова страница.

– v – предназначението е същото *като при PKARC.*

– t – проверява архива, като преизчислява CRC кодовете и ги сравнява със записаните.

– g – предназначението и начинът *на използване са същите,* както и при PKARC. Последната програма от този „комплект" е PKSFX. *Това всъщност* не е програма, която може директно да се изпълни, а модул, към който се прибавя съществуващ вече архив. „Прибавянето" се извършва по следния начин: C>copy /b pksfx.pgm + archive.arc archive.exe. Създаденият файл е вече изпълним и резултатът от действието с „саморазархивиране", archive.arc е вашият архив, а „/b" означава копирането да се извърши изцяло (без „/b" ще копира до първия символ ^Z, т. е. край на текстов файл).

Някой може да запита, и то с право – как така pksfx не беше програма и изведнъж с едно copy стана програма? Работата е там, че pksfx.pgm всъщност е EXE файл, само че просто не е предназначен да бъде пускан самостоятелно. Ако се преименува с окончание EXE, той може да се изпълни. Лошото е, че сега, макар и бегло, навлизам в елементи от технологията на създаване на вируси. Каквото и „да лепнете" след един EXE файл, той ще продължава да бъде работоспособен.

Няма да казвам защо и дали има изключения. С показаната по-горе команда copy зад pksfx просто се залепва архивът. А pksfx върши същата работа като pkxarc (затова за него са валидни опциите на pkxarc без „v"), само че върху онашката си. Как да намери мястото, от което започва истинският архив, е, общо взето, проста работа, тъй като pksfx „знае" собствената си дължина и дължината на самовъзстановяващ се файл.

ОПИТ

„ЧЕРНО-БЯЛ" ЧАСОВНИК

В излязлата неотдавна книга „Правец-8Д – професионални приложения" на Б.Захариев и Й. Йорданов се натъкнах на една много интересна програма – „Часовник за реално време". Веднага я въведох и... (о, разочарование!) на черно-белия ми телевизор не се виждаше абсолютно нищо.

Позволих си да направя няколко промени в програмата, поставяйки удобството над всичко. Повечето от собствениците на домашния Правец използват именно черно-бели телевизори. За да не вървят и те по трънливия път на пробите и грешките, искам да споделя с тях своя опит.

За да можете да използвате пълноценно часовника, достатъчно е в директен режим да въведете следните команди:

POKE#96AF,#10 – установява PAPER0 при първоначално стартиране на програмата;

POKE#96B4,7 – установява INK7 при първоначално стартиране;

POKE#973F,#10 – установява черен фон на цифрите на часовника;

POKE#9740,7 – установява бяло мастило на цифрите на часовника;

POKE#972C,#8D – при изпълнението на този оператор часовникът се изобразява в средата на горния ред на екрана (режим TEXT).

Последният оператор е много удобен, тъй като дава възможност във всеки момент да се знае в какъв регистър е компютърът – латиница или кирилица. В оригиналния вариант на програмата мястото, където нормално се извежда съобщението ЛАТ!, е заето от часовника.

След така направените промени програмата може да се запише с командата

CSAVE „CLOCK", A#96A0, E#97A8, AUTO.

ПОЩА

АЛИС'90

Радваме се на проявената от вас инициатива по обсъждане на проблема за безконечните ленти за печатащи устройства. Това доведе до намирането и публикуването на решения, които са може би най-добрите за любителски условия и затова поздравяваме техните автори.

На вниманието на всички, на които се налага по-често да подменят износени ленти, предлагаме услугите на нашата фирма, създадена в началото на годината.

Производството на фир мата ни обхваща безконечните ленти от всякакъв тип: нормални и мьобиус (ляво и дясно усукване) със стандартни дължини 6, 10, 15, 20 m и стандартни широчини 8, 9, 13 mm в цветове черно и черно-червено.

За делови контакти:

КФ „АЛИС 90"
БУРГАС — 8000
ул. Първи май 19
тел. (056) 4-55-42
инж. Атанасов

КФ „АЛИС 90"
ПЛОВДИВ — 4000
ул. Сашо Смирнов 5
тел. (032) 44-23-37
Мария Киндерова

ТРИК ЗА МИКРОТЕКСТ II

МикроТЕКСТ II – ИЗДАТЕЛ

Инж. ГЕОРГИ БАЛАНСКИ

Текстовите файлове, създадени с някоя от текстообработващите програми от семействата МикроТЕКСТ и ДОКС, са особено подходящи за графично оформяне и отпечатване с настолната издателска система ИЗДАТЕЛ. Запазват се знаците за край на абзац и пренос на нов ред, табулациите, шрифтът – получер или курсив, подчертаването на текста, горните и долни индекси. Запазват се само шрифтовете, зададени чрез директно форматиране с въвеждане на команди от командното меню или чрез комбинация за бързо форматиране (вж. справочника КВ.11-12.89). Шрифтовете, получени чрез маски, не се запазват и затова те трябва да се заменят с директно форматиране. Това става най-лесно със следната макрокоманда.

```
·1···1·····2·····3·····4·····5·····6··
<shift·f10><esc>фш<tab·6>н<enter><esc>фа<tab·7>н<enter>
```

В сравнение със специализираните текстообработки ИЗДАТЕЛ разполага със значително по-богат набор от графически знаци. Допълнителните знаци обаче не могат да се изписват директно от клавиатурата, а това става чрез въвеждане на кода им при едновременно натиснат клавиш ALT. Понеже редакторските възможности на настолната система са ограничени, значително по-удобно е тези кодове предварително да се въведат с текстообработката. Така например за отпечатване на буквата ѝ (и с ударение) трябва да се въведе кодът <210>, ограден със скоби. Пълният набор от знаци и техните кодове за активната в момента таблица с широчини (в OUTLINE.WID няма алтернативен знаков набор) могат да се видят, като се отпечата главата CHARSET.CHP, която се записва в справочника TYPESET по време на инсталирането на ИЗДАТЕЛ.

Във всеки текст, който се подготвя за ИЗДАТЕЛ, трябва да се извършат поне две промени:

– Да се отделят дългите от късите тирета. Тиретата, които трябва да станат дълги, се заменят с кода <197>. Те служат за по-силно, в сравнение със запетаята, интонационно-синтактично отделяне на части от изречението, поставят се и на мястото на изпуснати предлози и съюзи;

– Еднотипните кавички на текстообработката се заменят с „отварящи кавички"(леви) („) пред думата и съответно – със „затварящи кавички" (десни)(") след нея.

Често допускани грешки в текста, които проличават след отпечатването му, са:

– Въвеждане на повече от един интервал между думите;

– Въвеждането на интервали между последната дума от изречението и знака за край на абзац (¶);

– Въвеждане на интервали преди запетайка.

В първия и третия случай се получават забележимо големи интервали, а при втория – знакът за край на абзац често пъти бива пренесен самостоятелно на долния ред, поради което се получава празен ред.

Доработването на текста може да стане с последователно използване на функцията „заменяне".

При МикроТЕКСТ II обаче това може да стане несравнимо по-бързо и лесно чрез автоматична замяна с помощта на макрокоманда.

Дадената по-долу макрокоманда извършва автоматично следното:

1. Заменя всички поредици „интервал-кавичка" с кода за „отварящa кавичка"

2. Заменя останалите кавички с кода на „затварящa кавичка".

3. Заменя всички поредици „интервал-тире" с кода за дълго тире.

```
[····!····1·····2·····3·····4·····5·····6····]···
     <ctrl·
pgup><esc>з<space>"<tab><space>^<192><tab><space><enter><esc>з"<t
ab>^<201><tab><space><enter><esc>з<space>-
<tab><space>^<197><tab><space><enter><esc>з^^p-
<tab>^^p^<197><tab><space><enter><esc>з<space>^^p<down>^^p<tab><s
pace><enter><esc>з<space·
2><tab><space><tab><space><enter><esc>з<space>,<tab>,<tab><space>
<enter>
```

4. Заменя всички поредици „край на абзаците" с кода на знака за край на абзац, последван от кода на дълго тире.

5. Премахва интервала в поредицата „интервал-знак за край на абзац".

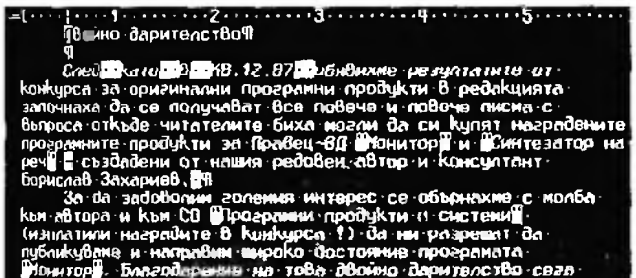
6. Заменя поредица от два интервала с един.

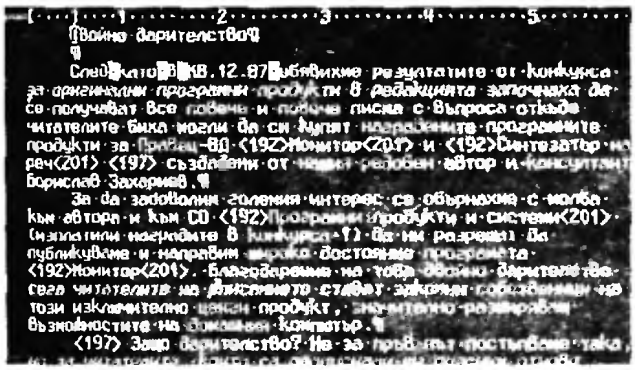
7. Заменя поредица „интервал-запетая" със *запетая*.

Резултатът от изпълнението на макрокомандата се вижда на двете илюстрации. За по-голяма прегледност е избран режим за видимост на всички знаци на екрана, а знаците, които ще бъдат заменени, са с получер шрифт.

В случай че между думите са оставяни повече от два интервала (също и преди знаците за край на абзац или преди запетайки), *макрокомандата се стартира повторно.*

Тя се записва в речника на МикроТЕКСТ II по един от следните два начина:





I

1. *Показаното на илюстрацията съдържание* на макрокомандата се написва с текстообработката.

2. Маркерът се придвижва върху първия знак, натиска се **F6** и чрез придвижване на *маркера с клавишите-стрелки текстът на* макрокомандата се избира.

3. Избраният текст се записва в речника с командата **Копие** от командното меню.

4. Въвежда се подсказващо предназначението *на макрокомандата име, например*

MT_IZDATEL.mac^<ctrl M>I.

Кодът **ctrl+M I** служи за бързо стартиране *на макрокомандата.*

II

При втория начин се изпълнява описаната в *макрокомандата последователност от* действия, които МикроТЕКСТ II автоматично записва. За целта преди и след въвеждането на командите се натиска комбинацията **Shift+F3.**

Името на макрокомандата се записва, както това вече бе обяснено.

Тъй като компютърът не е надарен с интелект, макрокомандата може да сгреши при некоректно написан текст. Така например тя различава дългото от обикновеното тире единствено по оставения пред него интервал. По същия начин се разпознават и кавичките, които трябва да бъде отпечатани като „отварящи".

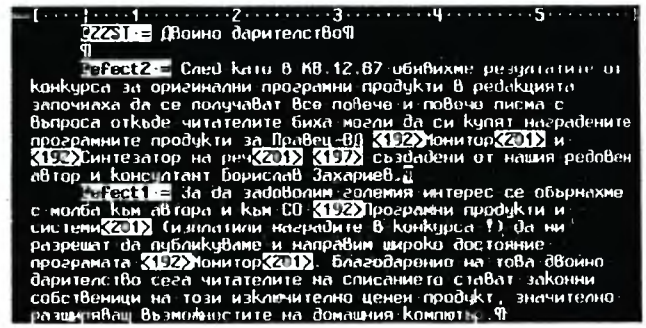
*Затова, след изпълнение на макрокоманда*та, е добре текстът да бъде прегледан и ако е необходимо, да се направят корекции. Това при всички случаи става многократно по-бързо, отколкото ако се работи по традиционния начин.

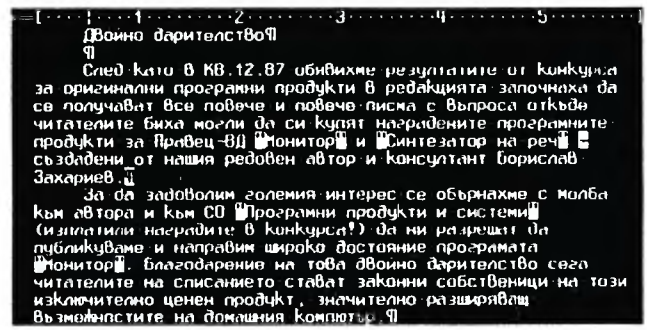
Често пъти се налага и противоположното действие – текстови файлове, обработвани с ИЗДАТЕЛ, „да се почистят" от допълнително въведените кодове, за да могат да бъдат отпечатани на обикновен принтер, т.е. да се възстанови първоначалният им вид.

В този случай, наред с възстановяването на обикновените кавички и уеднаквяването на всички тирета, трябва да бъдат изтрити и кодовете на маските за форматиране на абзаци, използвани от издателската система. Това се прави автоматично с дадената по-долу макрокоманда. Тъй като кодовете имат различна дължина, вместо функцията „замяна" се използва малко по-усложнен метод за намиране и изтриване на кодовете. Те се разпознават по това, че започват със знака @ и завършват със знак за равенство, ограден с интервали (например @ **efect** =).

В макрокомандата е включен операторът за условен преход **WHILE ... ENDWHILE**. Целта е да се осигури цикличното изпълнение на поредицата от команди, включени между **WHILE** и **ENDWHILE**, докато зададеното в оператора условие е изпълнено (то е „истина"). Щом условието престане да бъде „истина", се изпълнява стъпката, следваща оператора **ENDWHILE**.

В нашия случай, след като приключи с възстановяването на кавичките и тиретата, МикроТЕКСТ II търси знака @ и всеки път, когато го намери, се изпълняват командите, включени между **WHILE** и **ENDWHILE**. Той намира края на кода на маската, избира го и го изтрива. След като всички кодове на маски бъдат изтрити, макрокомандата преустановява действието си.






```
N       <ctrl-pgup>^p<esc>з^(197)<tab>-
<tab><space><enter><esc>з^(192)<tab>"<tab><space><enter><esc>з^(2
01)<tab>"<tab><space><enter><ctrl-
pgup><space><left><del><esc>н@<enter>◀while-
открит▶<esc>н<space>=<space><enter><right><left><f6><home><del>◀e
ndwhile▶<ins>◆
```

Макрокомандата се записва по първия от двата описани по-горе метода.

По-наблюдателните читатели ще забележат, че в макрокомандата са включени и няколко излишни на пръв поглед действия – натискане на клавиша **Ins** след завършване на цикъла, въвеждането на празен интервал и неговото прехвърляне в буфера на текстообработката. Това се прави, за да се възстанови изтритата първа буква от последния, започващ с маска абзац, както и за осигуряване на коректното поведение на макрокомандата в случай, че текстът не съдържа кодове на маски (макар че това е рядко срещан случай) или пък, когато първият абзац започва с код на маска.

Забележки:

1. Втората версия на настолната издателска система Ventura Publisher поставя автоматично полиграфическите кавички, а като срещне последователност от две къси тирета – заменя ги с едно дълго.
2. Макрокомандата за възстановяване на изходния текст е тествана върху работна версия на незавършения МикроТЕКСТ II. В случай че тя не работи коректно с търговския продукт, трябва да се провери дали не е променено името на запазената променлива, която проверява резултата при търсене с командата **Намери**.

Потребителите на различни дискови операционни системи го назовават с различни имена – директория, справочник, каталог, файлова система, файлстор. Независимо как я наричате, тази структура от данни е най-важната във вашия компютър – тя се грижи за файловете на гъвкавите и твърдите дискове.

В няколко поредни броя ще разгледаме организацията и особеностите на справочниците в най-разпространените дискови операционни системи за 8- и 16-битови микрокомпютри: ДОС 3.3, ПроДОС, CP/M, MS/DOS, Макинтош, UNIX и др.

Статиите са подготвени от ВЕСЕЛА НИКОЛОВА по *материали на списание Byte*.

Почти всичко ЗА СПРАВОЧНИКА

CP/M

През 1979 г. бе пусната втората версия на операционната система *CP/M*. *Последвалата CP/M 2.2* се превърна в главна операционна система за процесорите 8080 и Z80.

CP/M все още се радва на множество почитатели и ако имитацията е най-пряката форма на ласкателство, то CP/M би *трябвало да се изчерви* от удоволствие – първата версия на PC–ДОС не е ни повече, ни по-малко вариант на CP/M за *процесора 8088*.

Най-доброто във файловата система на CP/M е нейната адаптивност: тя може да поддържа както 5 1/4-инчови едностранни дискети, така и твърди дискове с големина до Гбайт. Лошото е, че тази адаптивност често изисква някои сложни изчисления при инсталирането на системата.

Пространството в диск под управлението на CP/M е разделено на отделни блокове. Файловете нарастват скокообразно на блокове. Също както в ПроДОС, блоковете в CP/M се състоят от няколко физически сектора върху диска. Разликата е там, че в CP/M тези блокове не са с еднаква големина при различните дискове. Оттук произтича и гъвкавостта на CP/M – възможност за избор на оптималния размер на блоковете в зависимост от обема на съответния твърд диск и индивидуалните нужди на потребителя.

Блокът с параметрите на диска (БПД) е структурата от данни в CP/M, която описва геометрията на диска (*вж.* табл. 1). Избирането на стойностите за елементите на БПД се определя в зависимост от два фактора: *броя на файловете, които предвиждате* да поместите върху диска, и следната дължина на всеки файл.

От *потребителска гледна точка* файловата система в CP/M се основава на единичния запис. Всяка програма разглежда даден файл *като* сбор от 128-байтови записи. Това се е запазило от времето, когато дължината на един физически сектор за дисковете, които работят с CP/M, е бил точно 128 байта. Много често се употребява и терминът „стандартен запис". Когато става дума за CP/M, стандартният запис е точно 128 байта. БПД носи информацията, необходима за изчисляване на броя на стандартните записи в един блок.

Структурата на справочника в CP/M не е свързана, както това е в описаните досега операционни системи (КВ.5–6.90). *Справочникът на диска* се пази на части в последователни блокове, които започват на точно определена пътечка на диска. *Местоположението на тази* начална пътечка от справочника е записано в БПД като отстъп от пътечка 0. Това позволява да се *запази място* за *справочника* след пътечките за операционната система и преди блоковете с данни за файлове.

Такава справочна организация има своите предимства и недостатъци. Последователните записи в справочника са по-лесно достъпни. Неудобството е, че броят на блоковете, заделени за справочника, винаги е точно фиксиран. Ако справочникът се състои от 8 блока, той ще заема точно толкова, дори ако върху целия диск има само един файл.

Всеки стандартен запис на справочника може да съдържа до 4 елемента за файлове (виж. табл. 2). Във всеки елемент има полета за име и тип на файла, както и карта за данни. Картата за данни е набор указатели към адресите, където се съхраняват данните за всеки файл. В зависимост от конфигурацията на диска указателят може да е с дължина 1 или 2 байта, но винаги сочи към един блок с данни. Ако дискът е съставен от по-малко от 256 блока, указателите са с дължина 1 байт. Двубайтовите указатели позволяват CP/M да поддържа дискове, съдържащи до 65 535 блока. Следователно всеки елемент в справочника сочи към максимум 8 или 16 блока с файлови *данни*.

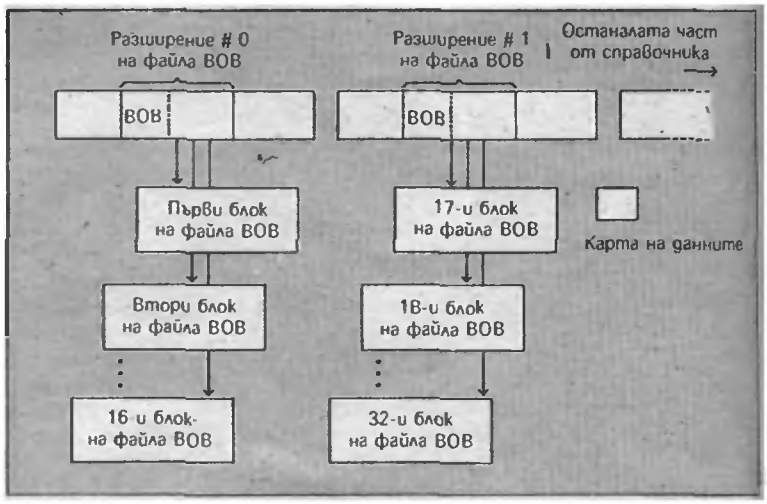
Когато даден файл нарасне дотолкова, че един елемент от справочника не може да помести *всички данни за него*, CP/M създава т. нар. разширен елемент (extend entry). Това е елемент от справочника, подобен на първия, *като* се различава от него само по стойността на байта за разширение и съдържанието на картата за данни. В примера, илюстриран на фигурата, файлът BOB с нараснал толкова, че една карта с данни не е достатъчна да побере цялото му съдържание. Операционната система създава още един елемент в справочника, като поставя в байта за разширение номер 1.

Таблица 1. Блок с параметрите на диска в CP/M. Тази структура от данни (съхраявана в паметта) позволява на ОС да определи физическите параметри на диска и да съотнася логическите записи към физическите сектори.

Байт	Описание
0 – 1	Брой на 128-байтовите записи в пътечка
2	Фактор на отместване. Посочва колко пъти трябва да бъде отместен надясно номерът на записа, за да се получи номерът на блока
3	Блокова маска. Операция логическо И между този байт и номера на записа дава отместването на записа вътре в неговия блок
4	Маска за разширение. Позволява от логическия номер на запис да се изчисли физическото разширение
5 – 6	Номер на последния блок върху диска
7 – 8	Максимален брой елементи в справочника (първият елемент е означен с 0)
9 – 10	Началните 16 бита от вектора за запълване. Използват се за предварително определяне на блоковете на справочника при създаване на вектора за запълване
11 – 12	Размер на контролната област, която позволява на CP/M да определи дали дискът е бил сменен. Тази област обикновено съдържа по 1 контролен байт за всеки 128 байта от справочника
13 – 14	Първоначален отстъп. Брой на пътечките (започва се от 0), запазени за операционната система и за данните за разделяне на диска

Таблица 2. Елементи на справочник в CP/M

Байт	Описание
0	Потребителски код
1 – 8	Име на файл (8 байта)
9 – 11	Тип на файла. Старшият бит на всеки от тези байтове е запазен като атрибутивен бит. Байт 9 – старшият бит определя файл за четене Байт 10 – старшият бит определя системен файл Байт 11 – старшият бит определя в този елемент (с цел архивиране)
12	Номер на разширението
13	Запазен байт
14	Байт за препълване на разширенията
15	Брояч за записи. Сочи броя на 128-байтовите записи, описани в този елемент
16 – 31	Карта за данни. Това поле може да бъде от 16 еднобайтови елемента или от 8 двубайтови



Картата за данни на този елемент сочи към допълнителните блокове на файла BOB. Ако файлът продължи да нараства, CP/M ще създава все нови и нови разширения, докато достигне максимума – 32. Някои версии на CP/M позволяват и повече разширения, като за целта използват байт 14 за препълване на разширенията.

При нарастването на файловете скокообразно според големината на блоковете може да се предположи, че средно около половината от последния блок на всеки файл ще остане празна. Това лесно се променя, като се избере по-малка дължина за *блоковете. Но когато файловете* нарастват, а блоковете са със сравнително малка дължина, системата ще трябва да създава множество разширени елементи в справочника. По-лошото е, че в действителност няма механизъм, който да осигурява записването на допълнителни елементи във възходящ ред в справочника. Освен това големият брой елементи в справочника означава и нарастване на броя на обръщенията към диска.

Може би сте забелязали, че дисковете под управлението на CP/M нямат карта на застостта. Всъщност съществува т. нар. вектор за запълване – своеобразна карта на застостта, която се съхранява не върху диска, а в *оперативната памет.* Всеки път, когато операционната система работи с избран диск, тя прочита целия справочник, за да създаде в паметта вектор за запълване. Разбира се, при нарастването или изтриването на файлове системата актуализира вектора.

От пръв поглед личи, че структурата на справочника в CP/M не е йерархична – не могат да се създават подсправочници. Но с възможно да се установят няколко алтернативни справочника на един и същ диск, като се използва потребителският код (байт 0 във всеки елемент). Този байт може да приема стойности от 0 (по подразбиране) до 31 и по този начин служи за разделяне на файловете в справочника на 32 потребителски области. Преместването от област в област се осъществява чрез командата USER. Ако въведете USER 4, ще можете да виждате и да работите само с файловете, чийто потребителски код е *установен на* 4.

Всяко разглеждане на операционната система CP/M би било непълно, ако не се спомене за разширението към нея – ZCPR3. Готовият пакет на ZCPR3 съдържа множество помощни програми, сред които заслужава да се отбележи възможността на ZCPR3 да поддържа имена на справочници. ZCPR3 позволява да се прикрепи към определено дискетно устройство име и потребителски номер в байт 0. Така A15 (устройство A, потребителска област 15) може да получи име ROOT, A7 да бъде WORDS и т. н. Придвижването между справочниците става с командата CD (Change Directory – смяна на справочник). С малко повече умение ZCPR3 може да създаде илюзията, че се работи с йерархични справочници в по-сложна файлова система.

Б. Р. Повече информация за операционната система CP/M можете да намерите в КВ.7 – 8, 9 – 10, 11 – 12. 88.

Тънкостите на МЕРЛИН

① ОРЛИН ВЪЛЧЕВ
Инж. БОРИСЛАВ ЗАХАРИЕВ

След излизането на пазара на новия домашен компютър Правец-8С много наши читатели ни пишат с молба да продължаваме да публикуваме указания за работа с разпространените програмни продукти, за които липсват ръководства. Резултатите от анкетата „Обектив" ни подсказват, че нашите читатели са доволни от поредиците, посветени на програмните продукти Локсмит 6.0 и Bag of Tricks.

Голям е интересът и към редактор-асемблера Мерлин. В КВ.9 и 10–11.87 и КВ.1.88 вече поместихме три статии с първоначални указания за работа с РЕД-АС – редактор-асемблер, който по своите функционални възможности и начин на *използване е аналогичен на МЕРЛИН.* От този брой започваме публикуването на нова поредица, която ще ви запознае с някои от *тънкостите при използването на този програмен* продукт.

АДРЕСИРАНЕ

Редактор-асемблерът Мерлин разпознава всички стандартни мнемонични кодове* на микропроцесора 6502. За улеснение са добавени и мнемоничните кодове BLT (Branch if Less Than – преход при по-малко) и BGE (Branch if Greater or Equal – преход при по-голямо или равно). Тези мнемонични кодове се асемблират така както BCC и BCS – чрез кодовете $90 и $B0. Ето примери за тринайсетте начина на адресиране при микропроцесора 6502:

* код – код на операция

1. Адресиране с подразбиране на акумулатора

За разлика от други асемблери при този начин на адресиране Мерлин не изисква и не приема операнд.

```
ASL
```

2. Адресиране с подразбиране на регистър

```
DEY
```

3. Непосредствено адресиране (непосредствен операнд)

```
SBS  #число – $10
LDX  #'A' + $40
LDA  #* : 2
```

4. Относително адресиране спрямо програмния брояч (PC)

```
BMI  LOOP
BGE  КРАЙ + 3
```

5. Пряко адресиране в нулевата страница

```
INC  4
```

6. Пряко адресиране в нулевата страница с индексиране по X

```
EOR  $30,X
```

7. Пряко адресиране в нулевата страница с индексиране по Y

```
ROR  240,Y
```

8. Пряко пълно адресиране

```
BIT  STROBE
```

9. Пряко пълно адресиране с индексиране по X

```
STA  ЕКРАН + $100,X
```

10. Пряко пълно адресиране с индексиране по Y

```
CMP  ДАННИ + 64,Y
```

11. Косвено пълно адресиране

```
JMP  ($3FE)
```

12. Косвено пълно адресиране с предварително индексиране по X

```
AND  ($1C,X)
```

13. Косвено пълно адресиране с последващо индексиране по Y

```
STA  (2),Y
```

Не съществува разлика в синтаксиса *между адресирането* в нулевата страница и пълното адресиране.

Асемблерът използва нулева страница навсякъде, където това е възможно. Ако се срещне инструкцията STA – $280 и стойността на *адрес* е $300, се получава обектен код: 85 80. Използвано е адресиране в нулевата страница, защото изчисленият адрес е еднобайтов ($80 = = $300 – $280). Генерираният код на STA е съответният за този тип адресиране: $85.

При косвеното адресиране с индексиране по X и Y стойността на използвания израз трябва да е в границите от 0 до $FF. В противен случай се издава съобщение за грешка.

Ако трябва да се използва пълно адресиране при изчислен еднобайтов адрес (адрес в нулевата страница), достатъчно е да се добави някакъв знак (без D) към мнемоничния код на инструкцията.

Пример:
STA 6 се асемблира 85 06 (2 байта)

STA' 6 генерира 8D 06 00 (3 байта)

ДИРЕКТИВИ

Директива на даден асемблер е команда към асемблера за някакво действие, което не е задължително да генерира обектен код. Понякога неправилно се използва терминът псевдооператор (от английската дума pseudoopcode).

1. Директива EQU (EQUals) или =

етикет EQU израз
етикет = израз

Тази директива служи за именуване на израз. Стойността на *израз* може да е външен за програмата адрес или често употребявана константа. Изразът може да е сложна функция на адреси и имена, които се променят често. Добре е всички директиви EQU да се намират в началото на програмата. Това изискване е задължително при дефиниране на етикети в нулевата страница.

Примери:

```
ИМЕ       EQU  $8C00
НАЧАЛО    EQU  ИМЕ+$100
ДЪЛЖИНА   =    КРАЙ-НАЧАЛО
КРАЙ      =    *
```

В последния пример КРАЙ приема стойността на програмния брояч, т. е. адреса на следващата инструкция, която генерира обектен код. Особеност на Мерлин е, че в този случай =* може дори да липсва, т. е. да съществува само етикетът.

2. Директива ORG (ORiGin)

ORG *израз*

Установява началния адрес, от който ще се изпълнява програмата. Ако липсва, се подразбира HIMEM на Мерлин ($8000). Обикновено програмата съдържа една директива ORG и тя е в началото. Ако са употребени няколко директиви ORG, първата устаиовява адреса на зареждане на програмата. Това понякога се налага, ако адресът на зареждане и изпълнение ие съвпадат. При Мерлии изрази от вида

```
ORG *-1
```

са недопустими!

3. Директива OBJ (OBJect)

OBJ *израз*

Установява адреса, от който ще се разполага обектният код при асемблирането. Ако тази директива липсва, се подразбира HIMEM на Мерлин ($8000). Всъщност рядко възниква нужда от употребата на OBJ, а начинаещите програмисти трябва да го избягват. Ако например OBJ е установен в областта на самия Мерлин (ако не е в рам-картата), при асемблирането ще настъпят непредвидени неща поради разрушаването на Мерлин. Обектният код ще бъде разрушен, ако OBJ е областта на таблицата на имената, установявана със SYM. Използването на OBJ е оправдано, ако се пести записът на обектния код върху диска и повторното му зареждане в паметта за евентуални проби.

4. Директива PUT

PUT *файл* [,Ss][,Dd]

Тази директива предизвиква четенето на текстовия файл с име T.*файл* и вмъкването на обектния код от асемблирането му в главния код. Всъщност се вмъква само обектният код, докато самият текстов файл се разполага в паметта непосредствено след основния файл. Асемблирането на прекалено големи файлове става възможно, като се разбият на по-малки и се извикват с PUT в реда на разбирането. След името на файла могат да се зададат номер на слот и флопидисково устройство.

Пример:

```
ORG $5000
OBJ $5000
PUT EXAMPLE,D2
PUT MAIN,D1
PUT FILE,D2
```

Така последователно ще се прочетат и асемблират файловете T.EXAMPLE, T.MAIN и T.FILE от съответните флопидискови устройства. Съществуват две ограничения при употребата на PUT – файлът, извикан чре PUT, не трябва да съдърж макродефиниции или друг директиви PUT. Тези ограни чения се преодоляват, като всичк макродефиниции и директив PUT са в основния файл.

5. Директива VAR (VARiable)

VAR *израз*[;*израз*[;*израз*...]]

Мерлин поддържа осем сис темни променливи]1 до]8 С директивата VAR могат да се установят техните стойности като първият израз задава стойност на]1, вторият на]2 и т. н.

Пример:

```
Var $88;БРАВО;FILL
```

Така]1 ще приеме стойност $88,]2 – стойността на БРАВО а]3 – стойността на FILL Употребата на VAR е наложителна преди четенето на файл с PUT, който съдържа системни променливи.

Пример:

Нека файлът T.FILE съдържа

```
LDA ]1
STA ]2
RTS
```

Нека основният файл съдържа:

```
        ORG $300
FROM    =   $1000
TO      =   $2000
        ...
        VAR FROM;TO
        PUT FILE
        ...
```

Написаното по-горе е равно силно на:

```
ORG  $300
...
LDA  $1000
STA  $2000
RTS
```

6. Директива SAV (SAVe)

SAV *файл* [,Ss][,Dd]

Текущият обектен код с записва на диска с посоченото име. В директивата могат да се посочат номерът на слота и на флопидисковото устройство Директивата SAV може да с

среща на няколко места в програмата и това позволява да се асемблират доста големи файлове в съчетание с PUT.

След всяко изпълнение на SAV адресът на разполагане на обектния код приема началната си стойност ($8000, ако не е променен чрез директивата OBJ). Това не се отразява на обектния код, записан на диска.

Пример:

```
ORG $1000
...
SAV ПЪРВИ,D1
ORG $2000
...
SAV ВТОРИ,D1
ORG $3000
...
SAV ТРЕТИ,D2
```

7. Директива DSK (DiSK)

DSK *файл* [,Ss] [,Dd]

Тази директива насочва генерирането на обектен код направо върху диска. След името на *файла може да се посочи номер* на слот и дисково устройство. DSK се използва при асемблирането на изключително големи програми. При средно големи програми се предпочита SAV като по-бърза и по-надеждна директива. Поради специфичното действие на DSK (целият обектен код не е в паметта) директивата CHK (контролна сума) губи смисъл и се пренебрегва от DSK.

8. Директива END

Това е рядко употребявана директива. Тя информира транслатора, че останалата част от програмата трябва да се пренебрегне. Всички стикети след END не се разпознават.

СЛЕДВА

КС5

Умножение — цяла част и остатък

За процесор 8086 да се напише на асемблер програмен модул, който да умножава съдържанието на регистър AX с числото 1,234 и резултатът да се запише по следния начин:
цялата част — в AX,
остатък — в DX.

СТОЯН КАЦАРОВ
ИВАН ГАНЕВ
Пловдив

КООРДИНАТАТА Y

Решение на КС—4

Уважаеми читатели, сигурно сте се убедили от личен опит, че у нас много трудно можете да направите справка в справочник — това е неприятна и неблагодарна работа, свързана със загуба на време и нерви. А като награда ще *ви разочарованието — информа*цията, която търсите, колкото и обикновена да ни изглежда, обикновено липсва в справочника. Затова, ако не сте намерили там как се прекодира номер на ред в адрес от графичната страница, не се *отчайвайте. Спомнете си народ*ната мъдрост (защо вратът на вълка е дебел) и се захващайте за работа.

Откъде да започнем? Най-простият начин, който не изисква никакви допълнителни умения освен работата с монитора на „Правец 82", е да намерим съответствието адрес-номер на ред. То е еднозначно обратимо (естествено!) и по него можем да намерим обратното, което ни трябва. Знаейки, че всички адреси *от първа графична страница са* от 2000 до 3FFF, превключваме в този графичен режим (например с команда HGR) и започваме да пишем по един байт FF (една бяла чертичка за по-добра видимост) от адрес 2000 нататък. Не след *дълго ще се убедим, че:*

- на един ред има 40 (десетично) байта — екранът е разделен на три хоризонтални ивици от по 64 (десетично) реда;
- всяка третина е разделена на 8 ивици от по 8 реда.

„Разделен на ивици" означава, че байтовете („чертичките"), разположени един под друг в съответните ивици, се намират на еднакво разстояние помежду си в паметта. Така например всеки два *байта, чийто адрес се различава с* 40 (десетично), са разположени един под друг на еднакви по номер редове, считано от началото на съответната третина (редове 0 и 64, 1 и 65 или 64 и 128 и т. н.). Аналогично съответните редове от *ивиците от по 8 са на* разстояние 128 байта помежду си, а съседните редове от всяка такава ивица — на 1024 байта разстояние.

Е, сега вече можем да пристъпим към съставяне на съответ*ствието. Както сме се убедили,* младшите три бита на адреса на началото на всеки ред са нули. Старшите три бита на този адрес са 001 за първа графична страница и 010 за втора. Останалите 10 бита съответстват на *осемте бита на номера на реда.* Старшите три от тези 10 съответстват на младшите три от номера (съседните редове на разстояние 1024). Следващата по-младша тройка от адреса съответства на по-старшата тройка *от номера (ивиците от по 8 са на* 128 байта помежду си).

Най-досадно е съответствието между старшите два бита на номера на реда и останалите 4 бита от използваните 10 на адреса. То се отнася до байтовете от тре*тините, отстоящи на 40 в па*метта. Разглеждайки трите възможни комбинации за старшите два бита на номера, а именно 00, 01 и 10, на които отговарят младши битове на адреса съответно 0000000, 0101000 и 1010000, виждаме, че двата бита от номера са разположени в четири бита от адреса, като са дублирани през един.

Цялото съответствие става съвсем ясно от картинката:

Битове в номер на ред от старши към младши:

ABCDEFGH

Битове в адреса, съответстващ на този номер:

OXXFGHCDEABABOOO,

където XX=01 за първа страница и XX=10 за втора. Сега остава „само" да напишем програмата, която „да разбърква" подходящо битовете от номера и да ги „слага" в адреса. В програмата, отговаряща на условието на задачата, са използвани следните означения: YC – адрес в нулевата страница, на който се намира еднобайтов номер на ред. ADR – адрес в нулевата страница, където се получава резултатът — двубайтов адрес, съответен на началото на реда, чийто номер е в YC. PG – адрес в програмата, чието съдържание е 20 (шестн.), ако програмата да връща като резултат адрес в първа графична и 40 (шестн.) съответно за втора страница. W – адрес на еднобайтово работно поле в нулевата страница.

Ето и самата програма:

```
      LDA YC
      ASL A
      ASL A
      STA W
      AND #$1C
      STA ADR+1
      EOR W
      ASL A
      ROL A
      ROL A
      ORA ADR+1 PG  EQU *+1
      ORA #$20
      STA ADR+1
      LDA YC
      ROR A
      AND #$E0
      STA ADR
      AND #$60
      LSR A
      LSR A
      ADC ADR
      STA ADR
```

Вече виждам раздразнението в очите ви „Ама къде е коментарът?". Е, няма все да се водим „за ръчичка" — особено когато програмата е доволно проста! Пък и не е редно да ви отнемам удоволствието сами да проследите „танца на битовете" от номера на реда до адреса.

Приятно забавление!

АВГУСТ СТОЯНОВ

Тест „Като Слънце"
Регистрация след КС3

БАЗА ДАННИ I²

Информация за информатиците в България

1.	GIBBY software	
2.	JSP	В. Търново
3.	Александър Георгиев Николов	
4.	Александър Георгиев Паталенски	
5.	Ангел Димитров Димитров	Пловдив
6.	Ангел Димитров Кавлаков	София
7.	Ангелин Бахчеванов	Г. Оряховица
8.	Антон Георгиев Николов	Ямбол
9.	Атанас Д. Атанасов	София
10.	Бенчо Иванов Ангелов	София
11.	Борис Чернокожев	Бургас
12.	Борислав Ад. Стойков	Варна
13.	Валентин Шопов	Благоевград
14.	Валентина Иванова Николова	
15.	Валери Василев Найденов	София
16.	Валери Иванов Милетиев	Михайловград
17.	Валери Нешков Йорданов	Бургас
18.	Васил Борисов Даскалов	Варна
19.	Васил Иванов Попов	Варна
20.	Владимир Г. Шумански	с. Безмер
21.	Владимир П. Иванов	Левски
22.	Владимир Петров	София
23.	Гарабед Капрнел Атамян	София
24.	Георги Атанасов Георгиев	Пещера
25.	Георги Иванев Цонев	В. Търново
26.	Георги К. Георгиев	Стара Загора
27.	Георги Марков	Варна
28.	Георги Рафаилов Върбанов	Благоевград
29.	Георги Станчев Георгиев	Толбухин
30.	Десислав Б. Капламаджиев	Варна
31.	Деян Георгиев Разсадов	Варна
32.	Димитър А. Борисов	Търговище
33.	Димитър Бож. Неделчев	Н. Загора
34.	Димитър Маринов Симеонов	
35.	Дончо Николов Николов	Сливен
36.	Дончо Николов Николов	Сливен
37.	Емия Дочевски	Пловдив
38.	Енчо Топалов	София
39.	Живко Господинов	Стара Загора
40.	Златин Георгиев Георгиев	София
41.	Зоран Веселинов Цветков	
42.	Ивайло Йорданов Бялков	
43.	Иван К. Гайларски	Хисар
44.	Иван Николов Николов	Стара Загора
45.	Иван Павлов Митов	Сливен
46.	Иван Панайотов Иванов	В. Търново
47.	Иван Петков Стоев	Казанлък
48.	Иван Христов Данов	Троян
49.	Илиян Венелинов Цачев	Варна
50.	Илиян Иванов Иванев	Плевен
51.	Калин Каменов	София
52.	Калин Станоев Пенев	
53.	Калин Стоев Пенев	
54.	Кирил Христов Стойменов	София
55.	Кирил Христов Стойчевски	София
56.	Коста Стефчов Костов	София
57.	Красен Тонев	
58.	Красимир Койчев	Казанлък
59.	Красимир Т. Божанов	Варна
60.	Красимир Т. Стефанов	Провадия
61.	Кръстю Гошев Мушкаров	Благоевград
62.	Лъчезар Йорданов Недев	Г. Оряховица
63.	Лъчезар Йорданов Недев	Г. Оряховица
64.	Лъчезар Костадинов Христов	
65.	Любомир Дапков Иванов	Нова Загора
66.	Любомир Тодоров Ноев	Русе
67.	Любомир Тодоров Ноев	Русе
68.	Мартин Иванов Чаушев	Златица
69.	Миросла Йорданов Костадинов	
70.	Мирослав Николов Маринов	Шумен
71.	Надя Петрова Пенчева	София
72.	Недялко Иванов	Пловдив
73.	Недялко Пасков	Пловдив
74.	Никола Желязков Жечков	Силистра
75.	Николай Валентинов Попов	София
76.	Николай Георгиев Николов	Ямбол
77.	Николай Д. Юрганджиев	Бургас
78.	Николай Минчев	Варна
79.	Николай Паиков Петров	Толбухин
80.	Николай Трифонов Семов	Варна
81.	Огнян Иванов Железов	Варна
82.	Павел Стефанов Атанасов	Варна
83.	Пенко Иванов Ковачев	София
84.	Петър Георгиев Зоровски	Смолян
85.	Петър Георгиев Петров	Пазарджик
86.	Петър Хараланов	Шумен
87.	Пламен Н. Трифонов	Михайловгр.
88.	Пламен Тодоров Джонев	София
89.	Радосдав Велчев Раданов	Разград
90.	Радостин Младенов	Провадия
91.	Росен Димитров Добринов	Варна
92.	Светлозар Емилов Керемидарски	
93.	Светозар Минчев Йонов	Русе
94.	Станислав Бончев Кирилов	Варна
95.	Станчо Станев	Варна
96.	Стефан Данчев	София
97.	Стефан Лъвчитопов	Габрово
98.	Стойчо Добрев Стойков	София
99.	Теодор Георгиев Тончев	Русе
100.	Теодор Георгиев Тончев	Русе
101.	Тодор Димитров Митев	Варна
102.	Тодор Славов Славов	
103.	Янислав Г. Карагьозов	Варна
104.	Ясен Ризов	Шумен

ЗАДОЧЕН КОНКУРС ПО ИНФОРМАТИКА

Конкурсна задача

5 за 1990 г.

Въобразяваме микропроцесор има една клетка акумулатор, и достъп до достатъчно голяма памет, в която клетките са номерирани последователно. Езикът за програмиране от асемблерски тип за този микропроцесор разполага със следните инструкции:

load i – зарежда акумулатора със стойността на i-тата клетка от паметта;

sto i – зарежда i-тата клетка от паметта със стойността на акумулатора;

add i – събира стойността на акумулатора и стойността на клетка i, като резултатът остава в акумулатора;

mul i – умножава стойността на акумулатора и стойността на клетка i, като резултатът остава в акумулатора.

Напишете програма за истински компютър, която да генерира редица от инструкции на описания въобразяваем език, предназначена за пресмятане на стойността на алгебричен израз, съставен от променливите X1, X2, ..., X9, чрез действията събиране и умножение, *като могат да се използват и* скоби.

Например за израза (X1+X2)*(X3+X4) редицата от инструкции може да бъде следната:

load 1 add 2 sto 5 load 3 add 4 mul 5

Предполага се, че стойностите на променливите са предварително заредени в първите няколко последователни клетки от паметта и че крайният резултат остава в акумулатора.

ЕМИЛ КЕЛЕВЕДЖИЕВ

Решението на конкурсната задача трябва да се състои от текста на програмата, описание на начина на работата с нея и обосновка на използвания алгоритъм. Желателно е програмата да е записана на дискета за микрокомпютър. Дискетата ще ви бъде върната по пощата. За това трябва да я снабдите с подходяща опаковка и с отделен плик, на който да е написан вашият адрес.

Срокът за изпращане е два месеца след излизане на списанието. Адресът е:

1113 София
ул. Акад. Г. Бончев, бл. 8
Институт по математика с ИЦ при БАН
КС за ТНТМ
За Конкурса по информатика

Резултати от задача 7 за 1989 г.

Да припомним условието на задачата:

Даден е списък, който съдържа не повече от 100 думи. Две думи ще наричаме сходни, ако едната от тях може да се получи от другата чрез едно от следните действия:

– разменяне местата на две съседни букви;

– заменяне на една буква с някаква друга;

– вмъкване или дописване (отпред или отзад) на нова буква;

– премахване на една буква.

Да се състави програма, която при въведена нова дума да отпечата всички думи от дадения списък, които са сходни с нея.

Общо бяха получени 11 работи. Първа награда от 10 точки и 50 лв. се присъжда на Красимир Генов от Плевен, който е предложил програма, написана на Паскал. Втора награда от 9 точки и 30 лв. получава Лъчезар Христов от София с програма, написана на Си. Трета награда се разделя от Георги Билчев от Русе, който е изпратил решение на Пролог, и от Павел Бойчев от Стара Загора с програма на Паскал. Двамата получават по 8 точки и по 10 лв.

Останалите участници получават следните *точки за общото годишно класиране: Георги* Георгиев от София, Иво Маринчев от Бургас и Петьо Делчев от Пловдив – по 7 т.; Генади Жилевски от София, Георги Георгиев от Тутракан, Динко Господинов от София и Свилен Цанов от Силистра – по 6 т.

Предложената програма за решаване на задачата е написана на Турбо Паскал 5.0. Списъкът от думите се представя чрез масива w, който може да съдържа до сто думи, всяка

съставена от най-много wlen на брой знакове. Актуалният брой на думите се пази в променливата n. Процедурата input е един възможен вариант за въвеждане на списъка от думите. Тук под дума всъщност разбираме произволен текстов низ (string). Чрез пробягване по целия списък процедурата find намира всички думи, които са сходни с въведената дума nw. За целта се използва функцията similar, която играе основна роля в програмата. Чрез нея се сравняват две думи wa и wb за сходност. Първият фрагмент от текста на тази функция служи за преглеждане на двете думи от ляво на дясно, знак по знак. Така се намира коя е първата позиция, в която двете думи се различават. Всички предишни знакове се изтриват. Вторият фрагмент на функцията similar чрез „дългия" оператор if..else.. открива дали думите са сходни. Използвани са стандартните за Турбо Паскал процедура

delete (var s : string; n1, n2 : integer)

и функция

copy (s:string; n1, n2:integer) : string.

Първата премахва n2 на брой (или максимално възможните, ако няма толкова) знакове от низа s, като започва от знака с номер n1, а втората връща низ, който се състои от n2 на брой знакове (или пак максимално възможните, ако няма толкова), взети от s, като започва от знака с номер n1. Тази процедура и функция (delete и copy) може би работят различно дълго време (това знаят точно създателите на компилатора на Турбо Паскал) в зависимост от това, дали се изтрива (копира) начална или крайна част на текстовия низ. За да направи сравнение, читателят може да реализира „огледален" вариант на similar, при който сравняването на думите wa и wb да започва не от първите им знакове, а от последните.

ЕМИЛ КЕЛЕВЕДЖИЕВ

```
program _7_89;

const
 wlen=20;
 empty='';

var
 w : array[1..100] of string[wlen];
 n : integer;

procedure init;
begin
 writeln('Въведете думите от списъка');
 writeln('( празен низ - за края )');
 n:=0;
 repeat
  n:=n+1;
  write('->'); readln(w[n]);
 until w[n]=empty;
 n:=n-1;
end{init};

function similar(wa,wb:string):boolean;
begin
 while (wa<>empty) and (wb<>empty) and (wa[1]=wb[1]) do
  begin
   delete(wa,1,1);
   delete(wb,1,1);
  end;

 if      (wa=empty) and (wb=empty)
  then similar:=false
 else if copy(wa,2,length(wa))=copy(wb,2,length(wb))
  then similar:=true
 else if wa=copy(wb,2,length(wb))
  then similar:=true
 else if wb=copy(wa,2,length(wa))
  then similar:=true
 else if (wa[1]=wb[2]) and (wa[2]=wb[1]) and
         (copy(wa,3,length(wa))=copy(wb,3,length(wb)))
  then similar:=true
 else similar:=false;
end{similar};

procedure find;
 var i:integer;
     nw : string;
begin
 writeln('Въведете дума за сравняване');
 write('->'); readln(nw);
 writeln('Подобни са:');
 for i:=1 to n do if similar(nw,w[i]) then writeln(w[i
end{find};

begin
 init;
 find;
end.
```

Резултати от задача 8 за 1989 г.

Условието на задачата беше следното:

Даден е лабиринт, който представлява правоъгълна таблица от клетки, като във всяка клетка е записано числото нула или едно. Да се състави програма, която намира как може да се стигне от една зададена клетка до друга, като се минава само през клетки, съдържащи нули, и така, че след всяка посетена клетка да се отива:

а) в някоя от четирите и съседни (отляво, отдясно, отгоре или отдолу) клетки;

б) във всичките и осем съседни клетки.

Първа награда за тази задача се присъжда на *Павел Бойчев от Стара Загора, който най-ясно* е описал използвания алгоритъм и с приложия някои сполучливи похвати при програмирането. *Той получава 10 точки и парична награда от 50* лв. Втора награда печели Красимир Генов от Плевен – 9 точки и 30 лв. Трета награда се дава *на Румен Ножделов от Пазарджик – 8 точки* и 20 лв.

Останалите участници получават следните *точки: Генади Жилевски от София, Георги* Билчев от Русе, Георги Константинов Георгиев от София, Лъчезар Христов от София и Мирослав Велев от Русе – по 7 т., Георги Ангелов Георгиев от Тутракан, Илия Косев от Пловдив – по 6 т., и Свилен Цанов от Силистра – 5 точки.

От получените 11 работи повече от половината реализират класическия алгоритъм, известен с името „вълна", за намиране на път (и то най-късия) между две клетки в лабиринт. Този алгоритъм е описан в статията „Път през лабиринта" от А. Закарян и Д. Шишков в КВ.9–10.88. В статията всъщност се съдържа и решението на тази конкурсна задача. Само един от участниците – Красимир Генов, цитира статията, но той сполучливо е видоизменил публикуваното там.

Основната идея на алгоритъма „вълна" е следната: записваме числото 1 в крайната клетка, до която трябва да стигнем; във всички съседни празни клетки записваме 2; същото правим с празните съседи на клетките, в които вече сме записали 2, като там записваме 3 и така продължаваме, докато се появи число в началната клетка. Сега, за да намерим търсения път, трябва да тръгнем от началната клетка, като последователно преминаваме в клетки, в които е записано число, по-малко с 1 от числото в предходната клетка. Така непременно стигаме до крайната клетка, в която беше записано числото 1. Необходими са две забележки. Първата е, че понятието съседна клетка се разбира по два различни начина, според подточките а) и б) на задачата. Втората забележка се отнася за правия ход на алгоритъма. Ако при опит да запишем число, по-голямо с 1, в съседна празна клетка, се окаже, че такава клетка означава, че задачата няма решение.

В приложената програма на Турбо Паскал 5.0, лабиринтът се пази в масива maze, с актуални размери maxRow редове и maxCol стълбове. Във всяка клетка на този масив се записва числото 0, ако клетката е празна, и числото 255 – в противен случай.

По време на работата на програмата, съгласно алгоритъма „вълна", в празните клетки се записват числа от 1 нагоре. При обратния ход на този алгоритъм пътят се маркира с числото 254. Въвеждането на данните при тази програма е реализирано както в работата на Павел Бойчев: задаване на лабиринта като масив – константа в текста на програмата. Това се осъществява чрез процедурата input, която задава още и размерите на лабиринта – maxRow и maxCol, началната и крайната точка на търсения път – Xbeg и Xend, а така също и признак, дали програмата ще работи по точка а) или по точка б) на условието – в зависимост от това, дали maxDir=4 или maxDir=8. Двойният цикъл в края на тази процедура извършва всъщност прекодиране на единиците в числото 255. Читателят може винаги да видоизмени процедурата input в зависимост от вкуса си. За извеждане на данните върху екрана на компютъра е предвидена процедурата show. Основната дейност в програмата се извършва от процедурите wave и back, като първата извършва правия ход на вълната, а втората – обратния. И в двете процедури намирането на съседите става чрез спомагателния масив – константа dir. В процедурата wave клетките от лабиринта, които са фронт на вълната, са записани в едномерния масив front с големина frontSize. От този масив итеративно се формира новият фронт на вълната newFront с големина newFrontSize. Итерациите спират, щом като се достигне до клетката – цел или се установи, че това не е възможно. В първия случай управлението в програмата се предава чрез излизане (exit) от процедурата и се отива към процедурата back, а във втория – програмата спира чрез halt. Процедурата back чрез преглеждане на съседните клетки на всяка текуща клетка намира търсения път, в който записаните числа намаляват от клетка в клетка с единица. Накрая процедурата show изобразява намерения път, като го маркира със звездички на екрана.

ЕМИЛ КЕЛЕВЕДЖИЕВ


```
program _8_89;

const
 dir : array[1..8,1..2] of integer =
 ((0,1),(-1,0),(0,-1),(1,0),(1,1),(1,-1),(-1,-1),(-1,1));

 maxRow0=50;
 maxCol0=50;

type
 cell = array[1..2] of byte;

var
 maze : array[1..maxRow0,1..maxCol0] of byte;
 maxRow,maxCol,maxDir : integer;
 Xbeg,Xend : cell;
 front,newFront: array[1..2*maxRow0+2*maxCol0] of cell;
 level,frontSize,newFrontSize : integer;

procedure input;
const
  m : array[1..8] of string[24]=

('111111111111111111111111',
 '100000000000010000000001',
 '101100111110111101111001',
 '101000000010001000100001',
 '101111101000100010000101',
 '111010101010101010100101',
 '100010001000100010000101',
 '111111111111111111111111');

var i,j : integer;

begin
```

```
  maxRow:=8;
  maxCol:=24;

  Xbeg[1]:=2;
  Xbeg[2]:=2;
  Xend[1]:=maxRow-1;
  Xend[2]:=maxCol-1;

  maxDir:=4; {=8}

   for i:=1 to maxRow do
    for j:=1 to maxCol do
     if m[i][j]='1' then maze[i][j]:=255 else maze[i][j]:=0;
end{input};



   procedure show;
    var i,j : integer;
   begin
     for i:=1 to maxRow do
      begin
       for j:=1 to maxCol do
        case maze[i][j] of
         0   : write(' ':3);
         254 : write(' * ':3);
         255 : write('###');
         else write(maze[i][j]:3);
        end;
       writeln;
      end;
     readln;
   end{show};

   procedure wave;
    var k,t : integer;
        x : cell;
        ex : boolean;
   begin
    maze[Xend[1],Xend[2]]:=1;
    front[1]:=Xend;
    frontSize:=1;
    level:=1;
    repeat
     newFrontSize:=0;
     ex:=true;
     for k:=1 to frontSize do
       for t:=1 to maxDir do
        begin
         x[1]:=front[k,1]+dir[t,1];
         x[2]:=front[k,2]+dir[t,2];
         if maze[x[1],x[2]]=0 then
          begin
           maze[x[1],x[2]]:=level+1;
           if (x[1]=Xbeg[1]) and (x[2]=Xbeg[2]) then exit;
           newFrontSize:=newFrontSize+1;
           newFront[newFrontSite]:=x;
           ex:=false;
          end;
        end;
     front:=newFront;
     frontSize:=newFrontSize;
     level:=level+1;
    until ex;
    show;
    writeln('Няма решение');
    halt;
   end{wave};



    procedure back;
     var k:integer;
         x,xn : cell;
    begin
     x:=Xbeg;
     maze[x[1],x[2]]:=254;
     repeat
      k:=1;
      repeat
       xn[1]:=x[1]+dir[k,1];
       xn[2]:=x[2]+dir[k,2];
       k:=k+1;
      until maze[xn[1],xn[2]]=level;
      maze[xn[1],xn[2]]:=254;
      level:=level-1;
      x:=xn;
     until level=0;
    end{back};

    begin
     input;
     show;
     wave;
     show;
     back;
     show;
    end.
```

ВРЕМЕННО КЛАСИРАНЕ

Първите десет участници след първите осем конкурсни задачи за 1989 г.:

1.	Павел Бойчев	61 m.
2.	Красимир Генов	59 m.
3.	Георги Билчев	58 m.
4.	Илия Косев	52 m.
5.	Свилен Цанов	45 m.
6.	Мирослав Велев	43 m.
7.	Георги Константинов Георгиев	41 m.
8.	Тодор Митев	40 m.
9.	Лъчезар Христов	33 m.
10.	Стефан Роглев	32 m.

Из архива на националните конкурси по програмиране

ТРИЪГЪЛНА ДЪСКА

Задача за съставяне на алгоритъм и програма

МАНОЛ ИЛИЕВ
Лауреат на втори национален конкурс по програмиране

Триъгълна дъска е съставена от еднакви равностранни триъгълници, наречени *полета*. Броят на полетата е 100. Числата 1, 1, 2, 2, 3, 3, ..., 49, 49, 50, 50 са написани върху полетата, като върху всяко поле е написано по едно число. Две полета са *съседни*, ако имат обща страна. Първоначално върху зададени полета е поставен по един *пул*. Пуловете са бели и черни. Разположението на числата по полетата и първоначалното разположение на пуловете са начални данни за алгоритъма. Извършват се последователни стъпки, като всяка стъпка се състои от задължително преместване на всички пулове на съседни полета, ако има такива, по следните правила:

1. Бял пул се придвижва в това съседно поле, в което е записано най-голямо число (независимо *дали в полето се намира друг пул*). *При наличие* на две такива полета изборът е произволен.

2. Черен пул се придвижва в това съседно поле, в което е записано най-малко число (независимо дали в полето се намира друг пул). При наличие на две такива полета изборът е произволен.

При тези движения е възможно два или три пула да попаднат в едно и също поле. Такива конфликти се решават по следния начин:

1. Едноцветни пулове се сливат в един пул от същия цвят.

2. Разноцветни пулове (два или три) унищожават текущото поле и го правят *невалидно* (т. е. полето става недостъпно за пуловете). Пуловете от невалидното поле се премахват.

Свързаната област е множество от валидни полета, за всяко от които съседните му полета са или невалидни, или са от същата област. Между две полета от свързана област съществува път.

Алгоритъмът прекратява работата си след изпълнението на K стъпки (числото K е начална данна на алгоритъма).

НАПИШЕТЕ ПРОГРАМА, която реализира описания алгоритъм за движение на пуловете и:

а) пресмята броя на свързаните области в последната позиция;

б) проверява дали последната позиция е *повторение на вече възникнала позиция* (две позиции са еднакви, ако съдържат едни и същи невалидни полета и пуловете в двете позиции са еднакво разположени).

ОПИШЕТЕ СЛОВЕСНО: кодировката на позициите, проверката за свързана област и проверката за повторение на позиции.

РЕШЕНИЕ НА ЕЗИКА ФОРТРАН

Правите, които минават през върховете на полетата и са успоредни на една фиксирана страна на дъската, я разделят на 10 слоя, съдържащи съответно 1, 3, 5, ..., 19 полета. Ще смятаме, че полетата на дъската са номерирани с числата от 1 до 100 последователно в една и съща посока за всички слоеве, като единственото поле от първия слой има номер 1, трите полета от втория слой имат номера 2, 3 и 4 и т. н. Полето с номер i ($1 \leqslant i \leqslant 100$) ще означаваме с F_i. За всяко поле F_i със S_{i1}, S_{i2} и S_{i3} означаваме номерата на полетата, съседни на F_i. Ако F_i има само едно или две съседни полета, то едно или две от числата S_{i1}, S_{i2} и S_{i3} определяме като нули.

За получаването на числото S_{ij} има няколко възможности:

– *да се изчисляват наново всеки път, когато* някой пул трябва да бъде преместен от F_i в съседно поле;

– да се изчислят предварително;

– да бъдат зададени явно в оператор DATA.

При използването на език с достатъчно богати средства за макропрограмиране елегантно решение е да се състави макропрограма, която *поражда числата* S_{ij} по време *на компилация*. Следващата програма изчислява S_{ij} ($1 \leqslant i \leqslant 100$, $i \leqslant j \leqslant 3$) по време на изпълнение:

```
C   N - брой на полетата в текущия ред:
C   L - номер на първото поле в текущия ред:
C   I - номер на поредното поле в текущия ред:
C   M = (-1)**(I-L)*N.
C
      SUBROUTINE INITS (S)
      INTEGER S(100,3)
      L = 1
      DO 10 N=1,19,2
       M=N
        L=L+N-2
        DO 10 I=L,L+N-1
         IF(I.GT.L+1) S(I,1)=I-1
         IF(I.LT.L+N) S(I,2)=I+1
         IF(I+M.LT.100) S(I,3)=I+M+1
10      M=-M
      RETURN
      END
```

За всяко поле F_i с CODE [illegible] числото записано върху F_i [illegible] и нула, когато е невалидно. За представяне на броя и разположението на пуловете ще използваме следните означения:

np_i – брой на пуловете от цвят j (1 – бял, 2 – черен);

p_{ij} – номер на полето, заето от i-тия пул от цвят j.

Номера на полето, в което се придвижва един пул p* от полето F_i ($1 \leqslant i \leqslant 100$) в някое от съседните на F_i полета, ще означаваме с next (p*). Този номер се определя измежду числата S_{ij} ($1 \leqslant j \leqslant 3$), за които $S_{ij} \neq 0$ и $CODE_{sij} \neq 0$, така че $CODE_{sij}$ е минимално или максимално (в зависимост от цвета на пула). Ако изборът е невъзможен, то пулът остава на мястото си.

```
C   K    - номер на полето, в което се намира пулът:
C   SG   - белег за цвят (1 - бял; -1 - черен);
C   I    - брояч за обхождане на съседни полета:
C   NEXT - номер на поле, в което пулът ще се придвижи.
C
      FUNCTION NEXT(K,CODE,S,SG)
      INTEGER CODE(100),S(100,3),SG
      NEXT=K
      DO 10 I=1,3
       IF(S(K,I).EQ.0.OR.CODE(S(K,I)).EQ.0) GOTO 10
       IF(NEXT.EQ.K.OR.SG*(CODE(NEXT)-CODE(S(K,I)))
     *                    LT.0) NEXT=S(K,I)
10    CONTINUE
      RETURN
      END
```

За решаване на конфликтите, които възникват при изпълнение на поредната стъпка, на всяко поле от дъската F_i ($1 \leqslant i \leqslant 100$) се съпоставя състояние t_i, дефинирано по следния начин:

$t_i = 0$, ако от началото на текущата стъпка до този момент в полето F_i не е преместен нито един пул;

$t_i = 1$, ако в текущата стъпка в полето F_i са преместени няколко пула (поне един), първият от които е бял;

$t_i = -1$, ако в текущата стъпка в полето F_i са преместени няколко пула (поне един), първият от които е черен.

Броят на невалидните полета, възникнали от началото на текущата стъпка до момента, е означен с ninv, а номерата на самите невалидни полета – с inv_1, inv_2, ..., inv_{ninv}. В края на всяка стъпка в новопоявилите се невалидни полета се записват нули ($CODE_{ninv} \leftarrow 0$, $1 \leqslant i \leqslant ninv$).

Изпълнението на всяка стъпка се състои в последователно придвижване на всеки от пуловете p* от поле с номер p_{ij} ($1 \leqslant j \leqslant 2$, $1 \leqslant i \leqslant np_j$) в поле с номер (next (p*)). При това са възможни три случая:

1. Новото поле е празно ($t_{next(p*)} = 0$). В този случай се запомня новата позиция на пула ($p_{ij} \leftarrow$ next (p*)) и в $t_{next(p*)}$ се записва цвета на полето (1 или –1).

2. В новото поле вече е преместен пул със същия цвят ($t_{next(p*)} = 3 - 2j$). В този случай пулът p* се премахва ($p_{ij} \leftarrow p_{npij}$ и $np_j \leftarrow np_j - 1$).

3. В новото поле вече е преместен пул с другия цвят ($t_{next(p*)} = 2j - 3$). В този случай се отбелязва възникването на ново невалидно поле (ninv ← ninv + 1 и $inv_{ninv} \leftarrow$ next (p*)) и пулът се премахва както във втория случай.

Тези операции се изпълняват от подпрограмата MOVE, която осъществява една стъпка от движението на пуловете върху дъската.

```
C   J      - брояч за цветовете (1 - бял, 2 - черен);
C   I      - номер на пул от цвета J (1≤I≤NP(J));
C   SIGN(J) - белег за цвета J (1 - бял, -1 - черен);
C   N      - поле, в което се придвижва I-тият пул;
C   MOVE   - флаг за повторение на позиция (0 или 1).
C
      FUNCTION MOVE (S,CODE,P,NP)
      INTEGER S(100,3),CODE(100),P(100,2),NP(2)
      INTEGER T(100), INV(50), SIGN(2)
      DATA SIGN/1,-1/
      NINV=0
      MOVE=1
      DO 10 N=1,100
10    T(N)=0
      DO 100 J=1,2
       I=0
20     I=I+1
30     IF(I.GT.NP(J)) GOTO 100
```

```
      IF(CODE(P(I,J)).EQ.0) GOTO 60
      N=NEXT(P(I,J),CODE,S,SIGN(J))
      IF(T(N)*SIGN(J)) 50,40,60
40    IF(NEXT(N,CODE,S,SIGN(J)).NE.P(I,J)) MOVE=0
      P(I,J)=N
      T(N)=SIGN(J)
      GOTO 20
50    NINV=NINV+1
      INV(NINV)=N
60    P(I,J)=P(NP(J),J)
      NP(J)=NP(J)-1
      GOTO 30
100   CONTINUE
      IF(NINV.EQ.0) RETURN
      MOVE=0
      DO 200 I=1,NINV
200   CODE(INV(I))=0
      RETURN
      END
```

В хода на изпълнение на текущата стъпка подпрограмата MOVE следи за повторение на позицията. Тя проверява дали при изпълнението на още една стъпка ще възникне отново позицията преди текущата стъпка. В такъв случай тя получава стойност 1, а иначе – 0. Проверката се извършва от оператора с етикет 40:

40 IF(NEXT(N, CODE, S, SIGN(J)). NE. P(I, J))
MOVE = 0

Основание за такова решение са следващите няколко леми.

ЛЕМА 1. Ако един пул последователно е посетил полета F_{i_0}, F_{i_1},..., F_{i_n}, от които първото и последното съвпадат, а всички междинни полета са различни от тях и междувременно *не са възникнали нови невалидни* полета, то $n \leqslant 4$.

Доказателство. Подобно на обикновената шахматна дъска полетата от нашата игрална дъска могат да се оцветят в два цвята – бял и черен, *така* че всеки две съседни полета да са оцветени в различни цветове. От това следва, че полетата F_{i_1}, F_{i_3}, F_{i_5},... са оцветени различно от първото поле. Следователно n е четно число. За всяко число k от интервала $[0, n-2]$ е изпълнено неравенството $CODE_{ik} \leqslant CODE_{i(k+2)}$, ако пулът е бял, или обратното неравенство $CODE_{ik} \geqslant CODE_{i(k+2)}$, ако пулът е черен, поради предположението, че *пулът се е придвижил от полето* $F_{i(k+1)}$ в полето $F_{i(k+2)}$, а не във F_{ik}. И в двата случая се получава, че редицата $CODE_{i_0}$, $CODE_{i_2}$,..., $CODE_{i_n} = CODE_{i_0}$ е монотонна. Следователно върху полетата F_{i_0}, F_{i_2},..., $F_{i_{n-2}}$ е записано едно и също число, а това е възможно само когато $n-2 \leqslant 2$.

ЛЕМА 2. При предположенията на Лема 1 *и при начина на определяне на следващо поле*, реализиран в подпрограмата NEXT, $n \leqslant 2$.

Доказателство. Лесно се съобразява, че ако $n = 4$, то $i_1 = i_3$. В този случай се оказва, че пулът във втората стъпка се е придвижил от полето F_{i_0} в полето F_{i_2}, а в четвъртата – от същото поле F_{i_2} в F_{i_3}. При избрания от нас алгоритъм на придвижване на пуловете това е невъзможно, тъй като *номерът на следващото поле* се определя по един и същ начин в различните стъпки и зависи само от цвета на пула и полето, в което се намира, ако междувременно не се появят нови невалидни полета. Следователно $n < 4$ и понеже n е четно, то $n \leqslant 2$.

ЛЕМА 3. Ако едно разположение на пуловете върху дъската се повтаря след определен брой стъпки, то това разположение се повтаря и след не повече от две стъпки.

Доказателство. Нека h е произволен пул от началната позиция, а F_{i_0} е полето, в което е разположен. Означаваме с $F_{i_1}, F_{i_2}, \ldots$ полетата, в които h се намира при всяко следващо *повторение на позицията*. Тъй като дъската е крайна, то някои от полетата в тази редица ще се повторят. Нека a е най-малкото неотрицателно число, за което е вярно, че F_{i_a} се среща поне два пъти в редицата. Ако допуснем, че $a > 0$, ще се окаже, че съществува и друго поле освен $F_{i(a-1)}$, такова, че пулът, разположен в него при следващото повторение на позицията, се е *придвижил в полето* F_{i_a}. *Но тогава тези два* пула ще се слеят в един и повторението на първоначалната позиция е невъзможно. От полученото противоречие следва, че $a = 0$, т. е. всеки пул от началната позиция след известен брой стъпки се връща в полето, от което е тръгнал. Съгласно Лема 2 това ще се случи след един или два хода, следователно цялата *позиция ще се повтори след не повече от два хода*.

Следствие. Ако позицията p_1 се получава в една стъпка от позицията p_0, то необходимо и достатъчно условие p_1 да се повтори след известен брой стъпки с за всеки пул p^* от p_0 да е изпълнено условието next (next (p^*)) = p^*.

На това следствие се основава методът, по който подпрограмата MOVE следи за повторението на позицията.

За да пресметнем броя на свързаните области в последната позиция, ни е необходим метод за обхождане на полетата от една свързана област. Да започнем с произволно валидно *поле* F_i. *За да обходим и маркираме всички* полета от тази област, ще използваме работен стек. Първо проверяваме полетата, съседни на F_i, и в тези от тях, които още не са посетени ($CODE_{ij} > 0$, $j = 1, 2, 3$), записваме – 1, а номерата им вкарваме в стека. След това извличаме от стека поредният елемент и с него извършваме същата процедура. Когато стекът се изпразни, *всички полета от първата свързана област ще* бъдат маркирани. Сега трябва да търсим ново непосетено поле, да увеличим броя на свързаните области и да продължим по същия начин, докато бъдат маркирани всички валидни полета на дъската.

По този начин ще се окаже, че за всяка свързана област отново обхождаме полетата от дъската. По-добро решение е предварително да се вкарат в стека номерата на всички валидни полета и при извличането им да се проверява дали са вече маркирани. Ако не – увеличаваме броя на свързаните области. Във всички случаи маркираме неговите непосетени съседи и вкарваме номерата им в стека. Тази операция повтаряме, докато стекът се изпразни.

```
C   STACK  - работен стек;
C   NS     - указател на стека;
C   I      - номер на поредното поле;
C   J      - брояч за обхождане на съседните полета;
```

ХАНОЙСКИТЕ КУЛИ

Итеративно решение

от **СИМЕОН ПЪНЕВ**
студент II курс
информатика
ФМИИ при СУ
„Св. Климент Охридски“

РЕКУРСИЯ ИЛИ ИТЕРАЦИЯ

Задачата за Ханойските кули е една от типичните, давани на начинаещите програмисти за упражнение по рекурсия. На времето с мен също не направиха изключение. Тогава първият ми порив бе да намеря „обикновено“ решение, но не успях. Заинтересувах се от проблема, говорих с други програмисти и следях публикациите по въпроса, но дълго време не намерих и намек за това, че е възможно итеративно решение. Първото, в което не се реализираше пряко рекурсия, беше разгледано и в статията, публикувана в КВ.1—2.89. Може да се спори за това, доколко решението съдържа някаква „скрита“ рекурсия или „имитира“ стека ѝ (както предполагам и за моето), или самата реализация е добра; лично на мен не ми харесва начинът, по който е получено — чрез детайлно разписване на действията и доста утежнения и тромав метод с двоичните числа.

Придържам се към мнението, че само рекурсивно определените обекти и процеси изискват рекурсивни решения, а останалите „нетипични“ задачи имат, и то по-добри итеративни решения. Смятам, че настоящият алгоритъм е лишен от посочените недостатъци и е добър аргумент в негова подкрепа.

Няма да се спирам отново на описанието на задачата, ще отбележа само, че с N съм означил броя на дисковете, а със „стълб“ — оста, заедно с дисковете върху нея.

```
C   NAREAS - брой на свързаните области.
C
      FUNCTION NAREAS(CODE,S)
      INTEGER CODE(100),S(100,3)
      INTEGER STACK(200)
      DO 10 I=1,100
       IF(CODE(I).EQ.0) GOTO 10
       NS=NS+1
       STACK(NS)=I
10    CONTINUE
      NAREAS=0
20    IF(NS.EQ.0) RETURN
      I=STACK(NS)
      NS=NS-1
      IF(CODE(I).GT.0) NAREAS=NAREAS+1
      CODE(I)=-1
      DO 30 J=1,3
       IF(S(I,J).EQ.0.OR.CODE(S(I,J)).LE.0) GOTO 30
       CODE(S(I,J))=-1
       NS=NS+1
       STACK(NS)=S(I,J)
30    CONTINUE
      GOTO 20
      END
```

Подпрограмата NAREAS е типичен пример за моделиране на рекурсия в нерекурсивен език и за възможностите, които такова моделиране дава за повишаване на ефективността.

Основната програма е оформена като подпрограма, която получава началната позиция чрез своите аргументи и получава като стойност броя на свързаните области. Подпрограмата спира една стъпка преди позицията да се повтори, като на K се присвоява броят на извършените стъпки.

```
C   K      - брой стъпки:
C   I      - номер на поредната стъпка:
C   MOVING - брой свързани области.
C
      FUNCTION MOVING(CODE,P,NP,K)
      INTEGER CODE(100), P(100,2), NP(2)
      INTEGER S(100,3)
      DATA S /300*0/
      CALL INITS(S)
      DO 10 I=1,K
       IF(MOVE(S,CODE,P,NP).EQ.0) GOTO 10
        K=I
        GOTO 20
10    CONTINUE
20    MOVING=NAREAS(CODE,S)
      RETURN
      END
```

ИДЕЯТА

Идеята е следната. Нека за даденото множество от номера на стълбове, подредени по големина, додефинираме операциите „стъпка напред" (или „избор на следващия елемент") и „стъпка назад" (или „избор на предишния елемент"), реализирани например от pred и succ на Паскал, така че

СТЪПКА НАЗАД (1) = 3 И СТЪПКА НАПРЕД (3) = 1.

Ще отбележа, че

(1) СТЪПКА НАПРЕД (СТЪПКА НАПРЕД(X)) = СТЪПКА НАЗАД(X) и обратно.

Сега ще опиша елементарното действие на алгоритъма (итерацията) и ще дам нужните пояснения:

Нека А е стълбът, от който ще местим. Тогава:

1. Движение с НУЖНАТА СТЪПКА, докато намерим стълб Б, на който можем да преместим съгласно правилата. (Или ако използваме (1), това може да се преформулира, като намирането на Б става чрез СТЪПКА в НУЖНАТА посока, АКО е възможно (т.е. местенето удовлетворява правилата), ИНАЧЕ — чрез СТЪПКА в ОБРАТНАТА посока.)

2. Прехвърляне на диска от А на Б.

3. Изобр на следващия стълб, от който ще местим.

По т. 1: При нечетен брой дискове НУЖНАТА посока е НАЗАД, а при четен — НАПРЕД.

По т. 3: Нека сме преместили диск от стълб А на стълб Б. Очевидно е, че ако това местене е било правилно, следващото няма да започва от него, а от някой от оставащите стълбове. И тъй като те са точно два, ясно е, че трябва да бъде един от тях. Според правилата може да се мести от стълба, който има по-малък диск на върха. (Тези, които се задълбочат повече в тази стъпка от алгоритъма, ще забележат, че в зависимост от посоката на неравенството между стълб А и В понякога се получава детерминизъм — възможното местене е единствено.) Номерата на тези два стълба се намират удобно чрез СТЪПКА НАПРЕД (Б) и СТЪПКА НАЗАД (Б).

Това елементарно действие се повтаря, докато всички дискове се съберат на стълб 3 (или $2^n - 1$ пъти, ако знаем формулата за броя на местенията).

```
program Hanoi_towers ;
  uses CRT ;
  var pilon : array[1..3,0..10] of integer ;
   ukaz  : array[1..3] of integer ;
   i,ot,na,step,pred,sled,N : integer ;
   c : char ;

  Begin
   Clrscr ;
   write('Въведете броя на дисковете <1-10> ');
   readln(N);writeln;
   pilon[2,0]:=N+1;pilon[3,0]:=N+1;              { Начално                }
   for i := 0 to N do pilon[1,i]:=N+1-i ;        { установяване на        }
   ukaz[1]:=N;ukaz[2]:=0;ukaz[3]:=0 ;            { дисковете в/у стълбовете }
   ot:=1 ;                                       { Начален стълб          }
   if(odd(N)) then step:=-1                      { Избор на стъпка:       }
              else step:= 1 ;                    { -1 : назад, 1 : напред }
   for i := 1 to trunc(exp(N*ln(2))) - 1 do  { или    }
    begin                                    { repeat }
     na:=(ot+2+step)mod 3 + 1 ;                  { Избор на стълба,}
     if(pilon[na,ukaz[na]]<pilon[ot,ukaz[ot]])   { върху който ще  }
       then na:=(na+2+step)mod 3+1;              { се прехвърля    }
     ukaz[na]:=ukaz[na]+1 ;                      { Прехвърляне     }
     pilon[na,ukaz[na]]:=pilon[ot,ukaz[ot]] ;    { на диск между   }
     ukaz[ot]:=ukaz[ot] - 1 ;                    { стълбовете      }
     writeln(i,'. Прехвърляне от ',ot,' на ',na);
      c:=readkey ;
      pred:=(na+1)mod 3 + 1 ;                    { Намиране на            }
      sled:= na mod 3 + 1 ;                      { другите два стълба     }
      if(pilon[pred,ukaz[pred]]<pilon[sled,ukaz[sled]])  { Избор на       }
       then ot:=pred                             { следващия пилон,       }
       else ot:=sled                             { от който ще се прехвърля }
    end   { или until(ukaz[3]:=N) }
  End.
```

ДОКАЗАТЕЛСТВО НА АЛГОРИТЪМА

Нека разгледаме случая N = 1. Съответната СТЪПКА е НАЗАД и задачата е решена. Да отбележим важното обстоятелство, че при „неправилна стъпка" дискът отива на стълб 2. Обобщено това означава, че при „неправилна стъпка" дисковете се подреждат на следващия, спрямо този, от който сме тръгнали, а при „правилна" — на предния.

Ще докажем решението по индукция:

1. За случая N = 1 това е очевидно (вече проверихме).

2. Допускаме, че решението е вярно за N = n.

3. Ще покажем, че той е верен за n + 1.

Стъпката, избрана за n + 1, е „неправилна" за n. Следователно от т. 2 първите n диска ще се подредят на стълб 2 (дискът „n + 1" очевидно не влияе върху тяхното движение). Следващото действие, независимо от стъпката, е прехвърлянето на диска „n + 1" от стълб 1 на стълб 3. Изборът на следващия стълб, от който ще местим, пада на 2 и оттам с „неправилната" стъпка дисковете отново се преместват на следващия, т. е. 3, с което доказателството е завършено. (Оттук може да се получи и формулата за броя на прехвърлянията.)

РЕАЛИЗАЦИЯ

В програмата стълбовете

са реализирани чрез двумерния масив pion. Той играе ролята на стек, в който указателят ukaz сочи върхния guck, а стойностите, записани в него, са номерата на дисковете. Ролята на дъската играят ограничителните стойности N+1, записани на ниво 0. Надявам се, никой не се заблуждава, че стекът тук играе ролята не на имитиране на рекурсията, а на „физическото" местене на дисковете.

Двете СТЪПКИ са обединени чрез формулата (X+2+step) mod 3+1, където стойност −1 на променливата step отговаря на СТЪПКА НАЗАД, а+1 — на СТЪПКА НАПРЕД. В ограничаването на елементарните действия съм използвал цикъл FOR, макар че по-коректна е употребата на REPEAT... UNTIL (ukaz [3]=N). Броят на дисковете е ограничен до 10 единствено от съображения за времето за извеждане на съобщенията и тяхното четене.

Програмата може да бъде написана и по друг, вероятно по-добър начин. Аз я използвам главно за илюстрация на моето твърдение. За мен бе по-важна итеративността на решението — едно елементарно действие от типа „стремеж към нещо", повтарящо, докато бъде получен крайният резултат. Но и в този си вид при стартиране тя работи със същата скорост, както рекурсивният вариант. (При 13 диска и двете програми работят около 4 минути и половина на компютър Правец—16. Тъй като засичах времето с обикновен часовник, не успях да забележа разлика между тях.)

Всъщност, ако се погледне по отношение на доказателството на алгоритъма, както и като четливост и лекота на възприемане, програмата се доближава най-много до него, може би защото не използва различни хитрости от по-ниско равнище, а имитира „физическите" действия, което е по-близо до човека и неговото мислене, отколкото до машината. В този смисъл алгоритъмът е подходящ и за реално решаване на задачата, а не само за компютърни програми. ■

АСЕМБЛЕР И IBM–PC

Инж. ВЛАДИМИР БОЧЕВ

Ползата от писането на асемблер едва ли има нужда да бъде изтъквана по какъвто и да е начин. Дори и непрофесионални програмисти често прибягват до програмиране на асемблер по една или друга причина. Какво ни предлага асемблерът? Отговорът е: голяма скорост на изпълнение и малки размери на програмите. Както и едно „вторично" действие: по-пълно опознаване на машината, а то винаги е от полза. Най-разпространеният у нас компютър, който е еднакво добър както за обучение, така и за писане на най-различни приложни програми, е несъмнено IBM–PC във всичките му разновидности. Според някои американски експерти това е компютър (заедно с операционната система естествено), който идеално „пасва" за писане на програми на асемблер. Тук ще ви запозная с някои похвати при съставянето на програми на асемблер за IBM–PC, някои от които се използват от водещи професионални програмисти в САЩ. Да започнем с нещо просто, като например преместване на байтове от едно място в паметта на друго;

```
Начални условия          :CX = брой на байтовете
DS:SI = място в паметта, където е поредицата от байтове
ES:DI = място в паметта, където ще се откопира низът
        shr     cx,1     ;брой на 16-битови думи
  rep   movsw            ;прехвърляне на 16-битови думи
        jnc     endmov   ;флагът за пренос е нула при четен
                         ;брой байтове
        movsb            ;само при нечетен брой байтове
endmov:
```

Следващият фрагмент сравнява два низа, като при равенство в регистър AX се записва 0, 1, ако низ1 < низ2 и −1 в обратния случай.

```
Начални условия: CX = дължината на кой да е от низовете
 (Тук трябва да направя следното допуточнение: този
 програмен фрагмент да се използва само при равни по
 дължина низове или когато единственото, което ни
 интересува, е съвпадението на знакове.)
    DS:SI = низ1,   ES:DI = низ2
          xor     ax,ax    ;AX = 0
  rep     cmpsb
          je      exit
          sbb     ax,ax    ;флагът за пренос е установен
                           ;от cmpsb
          cmc
          adc     ax,0
  exit:
```

А сега нещо просто, но много често използвано превръщане от шестнайсетично число в аскикод.

```
Начално условие :AL = число от 0 до F шестнайсетично
add     al,90h   ;пренос в диапазона 90 - 9Fh
daa              ;90 - 99h или 00 - 05h + пренос
adc     al,40h   ;0D0h - 0D9h + пренос или 41h - 46h
daa              ;30 - 39h или 41h - 46h , а това са
                 ;аскикодовете на 0 - 9 или на A - F
```

Абсолютна стойност се намира със следния фрагмент:

```
Начални условия :AX = число със знак (допълнителен код)
cwd             ;разширяване на знака
xor     ax,dx   ;инвертиране на AX, ако числото е било
                ;отрицателно
sub     ax,dx   ;AX - (-1), ако знакът е бил минус
```

Сега ще си позволя да дам един съвет за тестване на индивидуални битове, който в общи линии е известен още от процесора 6502. Много често особено при работа с периферни устройства се налага да се тестват по няколко бита, предварително заредени в регистър. Най-удобно е те да бъдат последните няколко от най-старшите битове. Например, ако от дадено периферно устройство ще тестваме 2 бита и условието е в зависимост от двата, то най-добре е те да са битове 6 и 7.

Ето защо. С една инструкция преместваме регистъра наляво. Най-старшият бит отива във флага за пренос, а флагът за знак директно отразява състоянието на бит 7, който сега е зает (след преместването) от бит 6. Освен това флагът за препълване ще се установи в 1, ако преносът и знакът се различават. Ясно е, че само с едно преместване вляво имаме на разположение всичките 4 възможни комбинации от два бита, след което може директно да се изпълняват необходимите за конкретния случай инструкции за условен преход. Има и още един начин за тестване този път едновременно на 4 бита. Това става с използването на инструкцията SAHF, която прехвърля битове 0,4,6 и 7 от регистър AH съответно във флаговете за пренос, четност, нула и знак. Следващият фрагмент позволява да се избере преход към дадена обработваща функция в зависимост от номера ѝ, записан в регистър CX (етикетът Dispatch, който се среща по-долу, сочи към таблица с начални адреси).

За малък брой функции (до 10) може да се използва:

```
        jcxz    function0
        dec     cx
        jz      function1
        dec     cx
        jz      function2
        .
        .
        .
```

Тъй като областта на условните преходи е ограничена до 128 байта, то ще бъде добре първата инструкция на всяка функция да е JMP (преход) към самата обработваща програма. Ако използваме предварително образувана таблица (Dispatch сочи към нея) и номерът на функцията е в регистър BX, то можем да използваме следната поредица от инструкции:

```
        shl     bx,1        ;всеки адрес е двубайтов, ако е
                            ;в рамките на един сегмент
        jmp     Dispatch[bx]
```

По-долу е показан един интересен вариант на първия метод за избор на функция:

```
                jcxz    function0
                loop    notfunc1
function1:      .
                .
                .
notfunc1:       loop    notfunc2
function2:      .               .
                .
notfunc2:       loop    notfunc3
function3:      .               .
```

И накрая един истински трик, който напомня самомодифициращите се програми, заклеймени като най-опасния начин на програмиране. Случва се дадена подпрограма да има няколко входни точки. Тези входни точки могат да се различават помежду си само по една инструкция, която може да служи за инициализиране на даден регистър с определена стойност, която пък може да служи по-нататък като флаг за това, откъде е била извикана подпрограмата. Например:

```
entry1:         mov     al,0
                jmp     body
entry2:         mov     al,1
                jmp     body
entry3:         mov     al,-1
body:           .
                .
                .
```

За по-голямо бързодействие и по-компактен код (това наистина прилича на баснята за великия програмист, който в оставащите 256 байта от паметта на „Voyager" вместил програма за разпознаване на спътницита на Сатурн, а пишейки на асемблер за някоя голяма машина, използвал самомодифициращ се код, за да спести 10 наносекунди) горният пример може да се представи така:

```
SKIP2F  MACRO
        db      3Dh  ;Код на инструкцията CMP AX, стойност
        ENDM         ;Този макрос ще промени флаговете
entry1: mov     al,0
        SKIP2F       ;Следващите два байта се третират от
                     ;процесора като непосредствена
                     ;стойност
entry2: mov     al,1
        SKIP2F       ;Вижи предният коментар
entry3: mov     al,-1
body:   .
```

Хитростта тук се състои в това, че макар и генерираният обектен код да дава следните инструкции към процесора:

```
mov     al,1
cmp     ax,01B0h   ;Данните са самата инструкция
                   ;mov al,1
cmp     ax,0FFB0h  ;А сега mov al,-1
body:   .
        .
        .
```

Адресите на входните точки са вътре в самата инструкция CMP. И те сочат непосредствените данни, които от своя страна са солиден (а и желан от нас, поне в този пример) операционен код. Променят се само флаговете. Трябва да е ясно, че такъв начин на работа е възможен само с макроси, които засенчват истинските инструкции. Ето и един пълен макрос с аргумент, който показва какво ще се промени, регистър или както по-горе флаговете.

```
SKIP2   MACRO   ModReg
IFIDNI  <ModReg>,<f>        ;Тест - дали желаем промяна
                            ;само на флаговете
        db      3Dh         ;код на CMP AX, стойност
ELSE
?_i     =       0           ;локална променлива на макроса
        IRP     Reg,<ax,cx,dx,bx,sp,bp,si,di>
IFIDN   <ModReg>,<Reg>      ;В списъка ли е регистърът?
        db      0B8h + ?_i  ;0B8h е основен опкод на
                            ;инструкцията MOV
        EXITM
ELSE
?_i     =       ?_i + 1
ENDIF
        ENDM
.errnz  ?_i EQ 8            ;сигнализация за грешка, ако
                            ;неправилно сме задали
                            ;параметъра на макроса
ENDIF
        ENDM                ;Край на макроса SKIP2

Този макрос може да се използува така:
SKIP2   f                   ;Променя само флаговете
SKIP2   ax                  ;Ще промени AX, но ще запази
                            ;флаговете
```

В показания по-горе конкретен случай, който се среща достатъчно често, печалбата е 2 байта и 3 такта на процесора 80286, тъй като се премахват JMP инструкциите.

Малка библиотека за ТУРБО ПАСКАЛ

Много програмисти, които работят на Турбо Паскал, са се сблъсквали с трудностите при използването на адресна аритметика, тъй като тя не е присъща на Паскал.

Предлагам за колегите една малка библиотека за Турбо Паскал, която много помага да се преодолява тази трудност. Тя се нарича „Unit Pointers" и се състои от два файла: Pointers. pas и Pointers. asm. Pointers. pas съдържа дефинициите на функциите и процедурите, необходими за компилирането им като .TPU файлове. Pointers. asm съдържа асемблерския текст на функциите и процедурите. Асемблерският файл се компилира с Турбо Асемблер и създаденият обектен файл се свързва при компилирането на Pointers. pas.

Библиотеката (създаденият от компилатора на Турбо Паскал файл Pointers. tpu) е работоспособна и надеждна за следните версии:
Турбо Паскал 4.0,
Турбо Паскал 5.0,
Турбо Паскал 5.5.

Н.с. инж. ВЕСЕЛИН ВЪЛЧЕВ

```
Unit  Pointers;

Interface

  Function AddPtr(P:Pointer;Inc:Word):Pointer;
          { Return pointer = Add P with Inc }
  Function SubPtr(P:Pointer;Dec:Word):Pointer;
          { Return pointer = Sub P with Dec }
  Procedure IncPtr(Var P:Pointer;Incr:Word);
                  { Increment P with Incr }
  Procedure DecPtr(Var P:Pointer;Decr:Word);
                  { Decrement P with Decr }

Implementation

{$L Pointers.obj}

    Function AddPtr;              External;
    Function SubPtr;              External;
    Procedure IncPtr;             External;
    Procedure DecPtr;             External;

end.  { Pointers }
```

```
;---------------------------------------------------------------
;
; Small Library for address arithmetic for Turbo Pascal
;
; Written by Vesselin Valchev
;                                              04.02.1990
;
; Compiling : TASM Pointers
;---------------------------------------------------------------
CODE    SEGMENT   BYTE   PUBLIC
        ASSUME    CS:CODE
        PUBLIC    AddPtr,SubPtr,IncPtr,DecPtr   ;  Make them known
pOfs        EQU       8       ; Offset for pointer offset
pSeg        EQU      10       ; Offset for pointer segment
Value       EQU       6       ; Offset for value inc/dec
;---------------------------------------------------------------
; NormPtr - procedure for pointer normalization
;---------------------------------------------------------------
NormPtr     PROC    NEAR
            MOV     CX,AX
            AND     AX,000FH  ; Offset mask
            SHR     CX,1      ; Rotate
            SHR     CX,1
            SHR     CX,1
            SHR     CX,1
            ADD     DX,CX
            RET
NormPtr     ENDP
;---------------------------------------------------------------
; AddPtr - function for Add pointer
;
; Pascal declaration:
;   function AddPtr(P:pointer;Inc:Word):pointer;
;---------------------------------------------------------------
AddPtr      PROC    FAR
            PUSH    BP              ;Save BP
            MOV     BP,SP           ;Set up stack frame
            MOV     AX,[BP][pOfs]   ;Offset part of address
            MOV     DX,[BP][pSeg]   ;Segment part of address
            ADD     AX,[BP][Value]  ;Offset + value
            JNC     NoOverflowAdd   ;Jump if not overflow
            ADD     DX,1000H        ;If yes overflow
NoOverflowAdd:
            CALL    NormPtr         ;Normalize
            POP     BP              ;Restore BP
            RETF    6               ;Remove parameters and return
AddPtr      ENDP
;---------------------------------------------------------------
; SubPtr - function for Sub pointer
;
; Pascal declaration:
;   function SubPtr(P:pointer;Inc:Word):pointer;
;---------------------------------------------------------------
SubPtr      PROC    FAR
            PUSH    BP              ;Save BP
            MOV     BP,SP           ;Set up stack frame
            MOV     AX,[BP][pOfs]   ;Offset part of address
            MOV     DX,[BP][pSeg]   ;Segment part of address
            SUB     AX,[BP][Value]  ;Offset + value
            JNC     NoOverflowSub   ;Jump if not overflow
            SUB     DX,1000H        ;If yes overflow
NoOverflowSub:
            CALL    NormPtr         ;Normalize
            POP     BP              ;Restore BP
            RETF    6               ;Remove parameters and return
SubPtr      ENDP
;---------------------------------------------------------------
; IncPtr - procedure for Increment pointer
;
; Pascal declaration:
;   procedure IncPtr(Var P:pointer;Incr:Word);
;---------------------------------------------------------------
IncPtr      PROC    FAR
            PUSH    BP              ;Save BP
            MOV     BP,SP           ;Set up stack frame
            PUSH    DS              ;Save DS
            LDS     BX,[BP][pOfs]   ;DS = Data segment Pascal
            MOV     AX,[BX]         ;BX = Offset Ptr offset
            MOV     DX,[BX][2]      ;DX = Segment Ptr
            ADD     AX,[BP][Value]  ;AX = Increment value
            JNC     NoOverflowInc   ;Jump if not overflow
            ADD     DX,1000H        ;If yes overflow
NoOverflowInc:
            CALL    NormPtr         ;Normalize
            MOV     [BX],AX         ;Save offset pointer
            MOV     [BX][2],DX      ;Save segment pointer
            POP     DS              ;Restore DS
            POP     BP              ;Restore BP
            RETF    6               ;Remove parameters and return
IncPtr      ENDP
;---------------------------------------------------------------
; DecPtr - procedure for Decrement pointer
;
; Pascal declaration:
;   procedure DecPtr(Var P:pointer;Decr:Word);
;---------------------------------------------------------------
DecPtr      PROC    FAR
            PUSH    BP              ;Save BP
            MOV     BP,SP           ;Set up stack frame
            PUSH    DS              ;Save DS
            LDS     BX,[BP][pOfs]   ;DS = Data segment Pascal
            MOV     AX,[BX]         ;BX = Offset Ptr offset
            MOV     DX,[BX][2]      ;DX = Segment Ptr
            SUB     AX,[BP][Value]  ;AX = Decrement value
            JNC     NoOverflowDec   ;Jump if not overflow
            SUB     DX,1000H        ;If yes overflow
NoOverflowDec:
            CALL    NormPtr         ;Normalize
            MOV     [BX],AX         ;Save offset pointer
            MOV     [BX][2],DX      ;Save segment pointer
            POP     DS              ;Restore DS
            POP     BP              ;Restore BP
            RETF    6               ;Remove parameters and return
DecPtr      ENDP
CODE        ENDS
            END
```

СОРТИРАНЕ НА ФАЙЛ

ХРИСТОС КИЛИАС
Атина

Предлагам откъс и програма от книгата на Христос Килиас „Файлове в Бейсик и приложенията им", Атина 1985 г., предложена за превод на български език и издаване на издателство „Техника". Авторът е широко известен на гръцките читатели освен с книгите си по Бейсик и като главен редактор на многотомна енциклопедия по информатика и компютърни технологии.

Стефан Стоянов
ИМ с ИЦ при БАН

Тази програма сортира произволен файл с пряк достъп. Тя получава от потребителя съответните параметри и създава сортиран изходен файл с местата на записите. Програмата няма ограничения за големината на входния файл. Изискват се само N. (KL +2) байта за помощни файлове и 3N байта — за изходния файл, където N е броят на записите, а KL — е дължината на ключа за сортировка.


```
10 REM <<< SORT >>>
20 REM ********************* ИНИЦИАЛИЗИРАНЕ *******************
30 DEFINT A-Z
40 DIM Z.LEN(4),Z.POS(4)
50 DIM Q$(4001),STACK(50):MAXREC=4000:QUICKM=10
60 CO$=CHR$(0):CZ6$=CHR$(26)
70 IQ=1:IQ1=IQ+1:IQ2=IQ+2
80 K1=1:FLAG=0
100 '
110 REM ******************* ГЛАВНА ПРОГРАМА *******************
120 '
130 GOSUB 280                    ' ПАРАМЕТРИ НА СОРТИРОВКАТА
140 GOSUB 490                    ' ОТВАРЯНЕ НА ВХОДНИЯ ФАЙЛ
150 '
160 PRINT TIME$;"  ";"НАЧАЛО НА СОРТИРОВКАТА"
170 '
180 GOSUB 580                    ' ЗАПЪЛВАНЕ НА БУФЕРА
190 GOSUB 710                    ' СОРТИРАНЕ НА БУФЕРА
200 IF K1>MR THEN GOSUB 1360     ' СЛИВАНЕ
210 IF FLAG=0 THEN 180           ' КЪМ НАЧАЛОТО
220 GOSUB 1000                   ' СЪЗДАВАНЕ НА ИЗХОДНИЯ ФАЙЛ
230 '
240 PRINT "          СОРТИРАНИ ЕЛЕМЕНТИ=";K-1
250 PRINT TIME$;"  КРАЙ НА ИЗПЪЛНЕНИЕТО"
260 END
270 '
280 REM ***** ВЪВЕЖДАНЕ НА ПАРАМЕТРИТЕ НА СОРТИРОВКАТА ******
290 CLS
300 PRINT "ВЪВЕДЕТЕ ПАРАМЕТРИТЕ НА СОРТИРОВКАТА"
310 PRINT "-------------------------------------"
320 INPUT "ВХОДЕН ФАЙЛ      :",Z.IF$
330 INPUT "ДЪЛЖИНА НА ЗАПИСА:",Z.RL
340 INPUT "ИЗХОДЕН ФАЙЛ     :",Z.OF$
350 INPUT "БРОЙ ПОДПОЛЕТА-КЛЮЧОВЕ:",Z.NKS
360 FOR I=1 TO Z.NKS
370   PRINT "*** ПОДПОЛЕ #";I;" ***"
380   INPUT "   НАЧАЛО :",Z.POS(I)
390   INPUT "   ДЪЛЖИНА:",Z.LEN(I)
400 NEXT I
410 PRINT :INPUT "OK (Y/N) :";Z$
420 IF Z$="N" THEN 290
430 IF Z$<>"Y" THEN BEEP:GOTO 410
440 '
450 KL=0:FOR I=1 TO Z.NKS:KL=KL+Z.LEN(I):NEXT I
    ' ДЪЛЖИНА НА КЛЮЧА
460 FREMEM!=FRE(0)-2000
461 ' ЗА КОМПИЛИРАНИ ПРОГРАМИ СЕ ЗАМЕСТВА С:
    FREMEM!=FRE(0)
462 MR=INT(FREMEM!/(KL+5))
463 ' ЗА КОМПИЛИРАНИ ПРОГРАМИ СЕ ЗАМЕСТВА
    : MR=INT(FREMEM!/(KL+6))
465 IF MR>MAXREC THEN MR=MAXREC
470 RETURN
480 '
490 REM ************* ОТВАРЯНЕ НА ВХОДНИЯ ФАЙЛ **************
500 OPEN "R",#IQ,Z.IF$,Z.RL
510 IF Z.RL>255 THEN Z.RL=255
520 FIELD#IQ,1 AS F.F1$
530 FIELD#IQ,Z.RL AS F.F$
540 GET#IQ,1:IF NOT EOF(IQ) THEN 560
550 CLOSE#IQ:KILL Z.IF$:PRINT "ПРАЗЕН ФАЙЛ":END
560 RETURN
570 '
580 REM ** ЧЕТЕНЕ НА КЛЮЧОВЕТЕ И ЗАПЪЛВАНЕ НА ТАБЛИЦАТА Q$ **
590 FOR I=K1 TO MR+K1-1
600   GET#IQ,I
610   IF EOF(IQ) OR F.F1$=CO$ THEN CLOSE#IQ:FLAG=1:GOTO 680
620   CLE$=""
630   FOR J=1 TO Z.NKS
640     CLE$=CLE$+MID$(F.F$,Z.POS(J),Z.LEN(J))
650   NEXT J
660   Q$(I-K1+1)=CLE$+MKI$(I)
670 NEXT I
680 SIZE=I-K1:K1=I
690 RETURN
700 '
710 REM ***************** (QUICK)  SORT Q$ ********************
```

```
720 IF SIZE<=QUICKM THEN 900
730 FOR P=1 TO 50:STACK(P)=0:NEXT P:P=0
740 TOP=1:BOTTOM=SIZE:Q$(0)=SPACE$(KL)
    :Q$(SIZE+1)=STRING$(KL,"M")
750 REM ПРЕНАРЕЖДАНЕ НА ПОДМАСИВ
760 I=TOP:J=BOTTOM+1:QQ$=Q$(TOP)
770 I=I+1:IF Q$(I)<QQ$ THEN 770
780 J=J-1:IF Q$(J)>QQ$ THEN 780
790 IF J>I THEN SWAP Q$(I),Q$(J):GOTO 770
800 SWAP Q$(TOP),Q$(J)
810 GOSUB 860                          ' ПОСТАВЯНЕ В СТЕК
820 IF J-TOP>QUICKM THEN BOTTOM=J-1:GOTO 750
830 IF P=0 THEN 900                    ' ВМЪКВАНЕ
840 REM ИЗВАЖДАНЕ ОТ СТЕКА
850 BOTTOM=STACK(P):P=P-1:TOP=STACK(P):P=P-1:GOTO 780
860 REM ПОСТАВЯНЕ В СТЕКА
870 IF BOTTOM-J<=QUICKM THEN RETURN
880 P=P+1:STACK(P)=J+1:P=P+1:STACK(P)=BOTTOM
890 RETURN
900 REM INSERTION SORT
910 FOR J=1 TO SIZE
920   I=J-1:QQ$=Q$(J)
930   IF QQ$>=Q$(I) THEN 960
940   Q$(I+1)=Q$(I):I=I-1
950   IF I>0 THEN 930
960   Q$(I+1)=QQ$
970 NEXT J
980 RETURN
990 '
1000 REM ************ СЪЗДАВАНЕ НА СОРТИРАН ФАЙЛ **************
1010 OPEN "R",#IQ2,Z.OF$,3        ' ОТВАРЯНЕ НА ИЗХОДНИЯ ФАЙЛ
1020 GET#IQ2,1:IF EOF(IQ2) THEN 1050
1030 CLOSE#IQ2:KILL Z.OF$
1040 OPEN "R",#IQ2,Z.OF$,3
1050 FIELD#IQ2,1 AS F.F$,2 AS F.ZPD$
1060 '
1070 IF K1<=MR  THEN GOSUB 1280 ELSE GOSUB 1200
1080 RETURN
1090 '
1100 REM *************** ОТВАРЯНЕ НА WORK1 ******************
1110 OPEN "R",#IQ1,"WORK1.&&&",KL+2
1120 FIELD#IQ1,1 AS W11$
1130 FIELD#IQ1,KL+2 AS W1$
1140 RETURN
1150 REM *************** ОТВАРЯНЕ НА WORK2 ******************
1160 OPEN "R",#IQ2,"WORK2.&&&",KL+2
1170 FIELD#IQ2,1 AS W21$
1180 FIELD#IQ2,KL+2 AS W2$
1190 RETURN
1200 REM ******* СЪЗДАВАНЕ НА ИЗХОДЕН ФАЙЛ ОТ WORK1 *********
1210 K=0:GOSUB 1100                    ' ОТВАРЯНЕ НА WORK1
1220 K=K+1:GET#IQ1,K
1230 IF W11$=C26$ THEN 1250
1240 LSET F.ZPD$=RIGHT$(W1$,2):PUT#IQ2,K:GOTO 1220
1250 LSET F.F$=C26$:LSET F.ZPD$=C0$:PUT#IQ2,K
1260 CLOSE:KILL "WORK1.&&&"
1270 RETURN
1280 REM ********* СЪЗДАВАНЕ НА ИЗХОДЕН ФАЙЛ ОТ Q$ **********
1290 LSET F.F$=C0$
1300 FOR K=1 TO SIZE
1310   LSET F.ZPD$=RIGHT$(Q$(K),2):PUT#IQ2,K
1320 NEXT K
1330 LSET F.F$=C26$:LSET F.ZPD$=C0$:PUT#IQ2,K
1340 CLOSE
1350 RETURN
```

```
1360 REM *************** СЛИВАНЕ НА Q$ И WORK1 ****************
1370 IF SIZE=0 THEN RETURN
1380 GOSUB 1100                        ' ОТВАРЯНЕ НА WORK1
1390 GET#IQ1,1:IF EOF(IQ1)  THEN GOSUB 1600:RETURN
1400 GOSUB 1150                        ' ОТВАРЯНЕ НА WORK2
1410 K=0:I=1:J=1:GET#IQ1,1
1420 IF Q$(I)<=W1$ THEN 1430 ELSE 1460
1430 LSET W2$=Q$(I):K=K+1:PUT#IQ2,K:I=I+1
1440 IF I>SIZE THEN 1500
1450 GOTO 1420
1460 LSET W2$=W1$:K=K+1:PUT#IQ2,K
 470 J=J+1:GET#IQ1,J
1480 IF EOF(IQ1) OR W11$=C26$ THEN 1540
1490 GOTO 1420
1500 LSET W2$=W1$:K=K+1:PUT#IQ2,K
1510 J=J+1:GET#IQ1,J
1520 IF EOF(IQ1) OR W11$=C26$ THEN 1560
1530 GOTO 1500
1540 LSET W2$=Q$(I):K=K+1:PUT#IQ2,K:I=I+1
1550 IF I<=SIZE THEN 1540
1560 LSET W21$=C26$:PUT#IQ2,K+1
1570 CLOSE#IQ1,#IQ2
1580 KILL "WORK1.&&&":NAME "WORK2.&&&" AS "WORK1.&&&"
1590 RETURN
1600 REM **** ИЗ ЗА ПЪРВОНАЧАЛНО ЗАПИСВАНЕ НА Q$ В WORK1 ****
1610 FOR I=1 TO SIZE
1620 LSET W1$=Q$(I):PUT#IQ1,I
1630 NEXT I
1640 LSET W11$=C26$:PUT#IQ1,I
1650 CLOSE#IQ1:RETURN
```

Главната част от процедурата се извършва в редове 110—260. Първо се въвеждат параметрите на сортировката (подпрограма 280—480), а такива са името и дължината на записа на подлежащия на сортиране файл, името на изходния файл и броят на подполетата, които съставят ключа. За всяко от тях се въвеждат началната позиция и дължината му. Всички подполета от тип идентификатор трябва да са съставени от букви и цифри и да започват с буква. В ред 450 се намира дължината KL, а в следващия ред — максималната големина на помощната таблица Q (буфер). Тази големина се пресмята, като се изхожда от намиращата се на разположение свободна памет и не превишава стойността MAXREC. В подпрограмата 490—570 се отваря входният файл.

Работата на следващите подпрограми може да се повтори, ако големината на файла е значителна (> MR).

В подпрограмата 580—690 се чете частта от файла от позиции K1 до MR + K1 (първоначално K1 = 1), съставя се ключът на всеки запис и се извежда в таблицата Q заедно с мястото на записа.

В края на тази процедура променливата SIZE съдържа броя на ключовете в таблицата Q $ (равен на MR освен последния път), а K1 става MR+1.

Продължава сортирането на таблицата Q $ (подпрограма 710—990). За тази цел се използва ефективният алгоритъм quick sort [виж. (1), стр. 140].

Ако броят на записите във файла е по-малък от MK, програмата приключва работата по създаване на изходния файл. В противен

случай се създава един временен файл от ключове — файлът WORK1. Това става в подпрограмата 1360 — 1590.

При първото обръщение към тази подпрограма се извършва просто прехвърляне на съдържанието на Q$ в WORK1. От следващото обръщение обаче подпрограмата извършва всеки път сливане на Q$ със (винаги) сортирания файл WORK1, създавайки WORK2. В края на процедурата WORK2 се преименува на WORK1.

Горната процедура се повтаря, докато се достигне краят на файла. Тогава променливата FLAG става 1 (ред 610) и продължава последният етап от създаването на изходния файл (подпрограмата 1000 — 1090).

Тук възможните случаи са два. Броят на записите е по-малък от MK — тогава сортираното множество от ключове се намира в таблицата Q$ и оттук се създава изходният файл. В противен случай тази роля се изпълнява от WORK1 — тогава създаването на изходния файл се извършва от него. Последният има дължина на записа 3 байта, от които последните два съдържат номера на записа в изходния файл. Към края му се прибавя един запис, чийто първи байт съдържа стойността CHR$ (26). Тази стойност ще бъде като EOF в проложната програма, която ще използва създадения по-горе сортиран файл. ■

ПОПРАВКА. В КВ.5-6.90 в материала „Скрол наляво“ от рубриката „Отговори“ по вина на редактора не е поместен листингът на програмата. Затова се извиняваме за пропуска, публикуваме листинга в този брой.

```
10 ' #400 ГОРНА ЛИНИЯ (0...199)
12 ' #401 ДОЛНА ЛИНИЯ (0...199)
14 ' #402 ЛЯВА ГРАНИЦА (0...39)
16 ' #403 ДЯСНА ГРАНИЦА (0...39)
18 ' CALL #404 - ИЗВЪРШВА СКРОЛ
20 L=100:PRINT"ЗАРЕЖДАНЕ";
30 FOR I=#400 TO #47F STEP 8
40 S=0:FOR J=0 TO 7:READ A$
50 A=VAL("#"+A$):S=S+A:POKEI+J,A:NEXT
60 READA$:A=VAL("#"+A$):IF A=S THEN90
70 PRINT CHR$(7):PRINT
80 PRINT"ГРЕШКА В РЕД":L:LIST 100-
90 L=L+10:PRINT".";:NEXT:PRINT:LIST-18
100 DATA 00,C7,00,27,AD,1F,02,D0,28C
110 DATA 01,60,AD,01,04,38,ED,00,238
120 DATA 04,18,69,01,85,76,AD,00,22E
130 DATA 04,A2,00,86,72,0A,26,72,240
140 DATA 0A,26,72,0A,26,72,85,70,239
150 DATA A6,72,86,71,0A,26,72,0A,2BB
160 DATA 26,72,18,65,70,85,70,A5,31F
170 DATA 72,65,71,69,A0,85,71,AC,3F3
180 DATA 03,04,A2,00,86,72,B1,70,2C2
190 DATA 85,75,29,C0,85,74,A5,75,3F6
200 DATA 29,20,85,73,A5,75,0A,29,28E
210 DATA 3E,05,74,05,72,91,70,A5,2D4
220 DATA 73,18,2A,2A,2A,2A,85,72,22A
230 DATA 88,CC,02,04,10,D8,18,A5,2FF
240 DATA 70,69,28,85,70,A5,71,69,375
250 DATA 00,85,71,C6,76,D0,C0,60,422
```

■

СОФТУЕР

Н. с. инж. МИХАИЛ РАЗВИГОРОВ

В КВ.06.87 бе представена програма за изобразяване на повърхнини, предназначена за Правец—82, с плотер Микроника 297. Но плотерът е сравнително скъпо устройство и не се използва навсякъде. От друга страна, в практиката масово навлязоха 16-битовите микрокомпютри със значително по-големи възможности. Затова има смисъл да се използва следната програма, която позволява да се изобразяват тримерно повърхнини с Правец-16 и програмно съвместимите с тях, без да е необходим плотер. Програмата реализира алгоритъма, описан в КВ.6.87, като са прибавени някои допълнителни възможности:

- начертаване на координатните оси и означаване на мащабите по тях;
- изчисляване на минимални и максимални стойности на функцията, което служи за правилно определяне на мащаба и получаване на качествено тримерно изображение;
- възможност за получаване на огледален образ на графиката на функцията.
- Направени са и някои структурни изменения на програмата, за да се увеличи бързодействието ѝ.

ТРИМЕРНО

ОПИСАНИЕ НА МЕТОДА

Нека е дадена функция с две променливи от вида:
$z = f(x, y)$
в областта:
$x \in (x_{min}, x_{max})$, $y \in (y_{min}, y_{max})$.

Тримерното изобразяване в равнината на повърхнини, зададени аналитично с две независими променливи, може да се реализира чрез последователно начертаване на сечения на повърхнината с равнина, успоредна на равнината XOZ. Положението на секущата равнина се изменя в зададения интервал на изменение на независимата променлива y. Първото сечение се определя от равнина, успоредна на равнината XOZ и пресичаща оста OY в точка $y = y_{min}$. Следващите сечения се получават чрез равнини, пресичащи оста y в точки $y = y_{min} + y$. Секущата равнина се измества успоредно със стъпка y, докато се достигне точка $y = y_{max}$. Броят на сеченията се определя от необходимата плътност на изображението. По-големият брой сечения определя по-качествено тримерно изобразяване на повърхнината.

ОПИСАНИЕ НА АЛГОРИТЪМА

Последователността на изпълнение на алгоритъма е следната:

1. Въвеждат се видът на функцията, минималните и максималните стойности на двете независими променливи x_{min}, x_{max}, y_{min}, y_{max}, броят на стойностите n, за които ще се изчислява функцията в интервала на променливите, броят на сегментите ns, с които ще се изобразява функцията.
2. Определят се нарастванията по оста y, които определят отместването на сегментите един спрямо друг
$y = (y_{max} - y_{min}) / ns$.
3. Полага се $yt = y_{min}$.

4. Изчислява се функцията за n стойности на променливата x в зададения интервал, при фиксирана стойност на променливата y = yt

5. Начертава се първата линия (сегмент), която се приема за базова. При изобразяването на следващите сегменти се определят предварително видимите части, разположени над и под базовата. Това се постига чрез сравняване на текущите стойности на функцията с максималните респективно минималните стойности, фиксирани до този момент. На всяка стъпка се актуализират максималните и минималните стойности, които се използват за следващото определяне на видими и невидими части.

6. Определя се новата фиксирана стойност на независимата променлива yt (yt = yt + y).

7. Проверява се условието

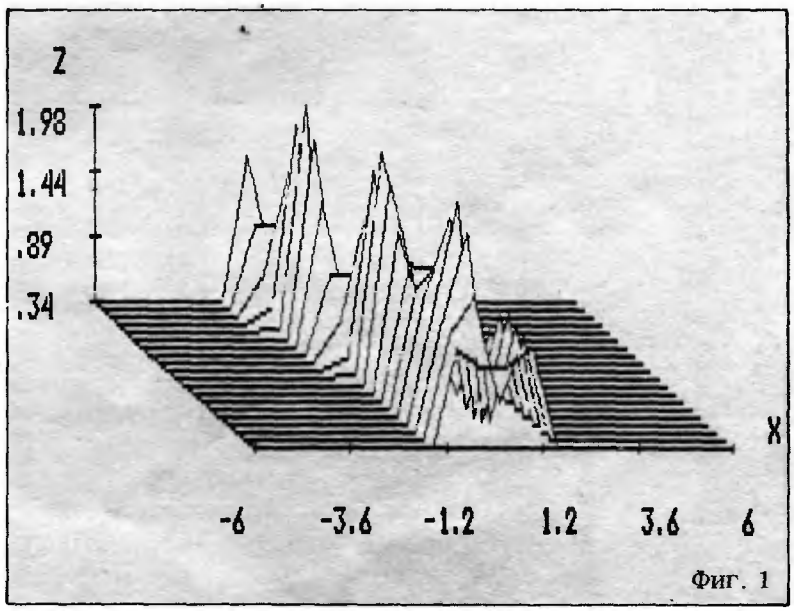
$yt \leq y_{max}$

При изпълнение на това условие се преминава към стъпка 4. В противен случай се преминава към следващата стъпка.

8. Край.

ОПИСАНИЕ НА ПРОГРАМАТА

Програмата изобразява тримерно повърхнини на функции с две променливи.



Фиг. 1

ИЗОБРАЗЯВАНЕ НА ПОВЪРХНИНИ

За работа с програмата е необходимо да се зададе повърхнината във вида Z = F (X, Y) на програмен ред 290 и да се въведат следните входни величини:

N — брой на точките, за които ще се изчислява функцията;

NF — брой на кривите, чрез които ще се изобрази повърхнината;

XMIN — минимална стойност на независимата променлива X;

XMAX — максимална стойност на независимата променлива X;

YMIN — минимална стойност на независимата променлива Y;

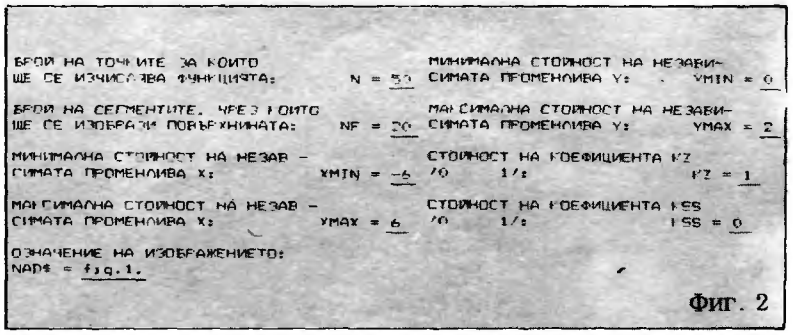
YMAX — максимална стойност на независимата променлива Y;

KZ е управляваща променлива. Приема стойност 0 или 1 и определя квадранта от координатната система, в който се изобразява функцията. При KZ = 1 се изобразява огледалният образ на графиката на функцията;

KSS — управляваща променлива. Приема стойност 0 или 1 и определя дали да се изобразява координатната система. При KSS = 1 координатната система не се изобразява;

NA — означение на изобразяваната функция (надпис на чертежа).

Функцията се изобразява върху видеомонитора. За копиране на изображението върху принтер (hard copy) е необходимо да се натиска едновременно клавишите Schift и PrtSc при предварително стартиран модул GRAPHICS. COM.



Фиг. 2

ПРИМЕР

Да се изобрази функцията, зададена със следния аналитичен израз:

$z(x, y) = 0{,}34 + 1{,}9\{e(-10(x+y) + 0{,}5e(-10(x + y - 1)\}$

при ограничения за x и y:

$-6 \leq x \leq 6,\ 0 \leq y \leq 2.$

На всяка стъпка да се изчисляват по 50 стойности на функцията в зададения интервал по x (n=50). Повърхнината да се изобрази с 20 криви (ns=20).

Резултатът от изпълнението на контролния пример е показан на фиг. 1. Начинът на въвеждане на входните данни е на фиг. 2. Подчертаните знакове се въвеждат.

```
10  '
20  ' ПРОГРАМА ЗА ТРИМЕРНО ИЗОБРАЗЯВАНЕ НА
    ФУНКЦИИ
30  '
40 DIM X(350), Z(350), Z1(350), XH(500)
50 DIM P1Z(2, 500), P2Z(2, 500), P3Z(2,
    500), H(500)
60 NGP = 0: NPG1 = -3: MXDP = 500
70 JPR = 0: YYMIN = 1.1 ^ 5: YYMAX = -
    YYMIN: XXX3 = 1.1 ^ 5
80 PI = 3.14159
90 KEY OFF: CLS : LOCATE 1. 1
100 PRINT "БРОЙ НА ТОЧКИТЕ, ЗА КОИТО"
110 INPUT ; "ЩЕ СЕ ИЗЧИСЛЯВА
    ФУНКЦИЯТА:       N = ", N
120 LOCATE 4, 1: PRINT "БРОЙ НА СЕГМЕНТИТЕ,
    ЧРЕЗ КОИТО"
130 INPUT ; "ЩЕ СЕ ИЗОБРАЗИ ПОВЪРХНИНАТА:
    NF = ", NF
140 LOCATE 7, 1: PRINT "МИНИМАЛНА СТОЙНОСТ
    НА НЕЗАВИ-"
150 INPUT ; "СИМАТА ПРОМЕНЛИВА X:
    XMIN = ", XIN
160 LOCATE 10, 1: PRINT "МАКСИМАЛНА СТОЙНОСТ
    НА НЕЗАВИ-"
170 INPUT ; "СИМАТА ПРОМЕНЛИВА X:
    XMAX = ", XAX
180 LOCATE 1, 44: PRINT "МИНИМАЛНА СТОЙНОСТ
    НА НЕЗАВИ-"
190 LOCATE 2, 44: INPUT ; "СИМАТА ПРОМЕНЛИВА
    Y:      YMIN = ", YIN
200 LOCATE 4, 44: PRINT "МАКСИМАЛНА СТОЙНОСТ
    НА НЕЗАВИ-"
210 LOCATE 5, 44: INPUT ; "СИМАТА ПРОМЕНЛИВА
    Y:      YMAX = ", YAX
220 LOCATE 7, 44: PRINT "СТОЙНОСТ НА
    КОЕФИЦИЕНТА KZ"
230 LOCATE 8, 44: INPUT ; "/0
    1/:              KZ = ", KZ
240 LOCATE 10, 44: PRINT "СТОЙНОСТ НА
    КОЕФИЦИЕНТА KSS"
250 LOCATE 11, 44: INPUT ; "/0
    1/:             KSS = ", KSS
260 LOCATE 13, 1: PRINT "ОЗНАЧЕНИЕ НА
    ИЗОБРАЖЕНИЕТО:"
270 INPUT ; "NAD$ = ", NAD$
290  DEF FNP (XX, YY) = .34 + 1.9 * (EXP(-10
    * (XX + YY) ^2) + .5 * EXP(-10 * (XX ^2
    + YY ^2 - 1) ^2))
310  EPS = 10 ^ -6
320 '
330 ' Определяне максимума и минимума на
    функцията
340 '
350 NN = 20: ZIN = 1 * 10 ^ 5: ZAX = -ZIN:
    XST = (ABS(XAX)
   .+ ABS(XIN)) / NN
360 YST = (ABS(YAX) + ABS(YIN)) / NN: YY =
    YIN
370 FOR I = 1 TO NN
380 XX = XIN
390 FOR J = 1 TO NN
400 ZZ = FNP(XX, YY): IF ZAX < ZZ THEN ZAX =
    ZZ
410 IF ZIN > ZZ THEN ZIN = ZZ
420 XX = XX + XST: NEXT J
430 YY = YY + YST: NEXT I
440 '
450 ' Определяне нарастъците по осите X и Y
460 ' и определяне на мащабите
470 '
480 N1 = -N: FSN = NF - 1
490 XDLTA = (XAX - XIN)
500 DXIN = XDLTA / 3 / FSN
510 ZDLTA = (ZAX - ZIN)
520 IF KZ = 0 THEN ZDLTA = -ZDLTA
530 DZIN = ZDLTA / 2 / FSN
540 KOR = 1: IF KZ = 0 THEN KOR = -1
550 MX = 400 / (XAX - XIN + DXIN * FSN)
560 MY = KOR * 115 / (ABS(ZAX - ZIN) +
    ABS(DZIN * FSN))
570 MYD = 100 + KOR * (ABS(MY * (ZAX - ZIN)
    / 2)) + KOR * (ABS(MY * (DZIN * FSN)/2))
    - KOR * ABS(ZIN * MY)
580 MXD = (634 - MX * (XIN + XAX - DXIN *
    FSN)) / 2 + 20
590 CLS : SCREEN 2: LINE (1, 1)-(634, 198),
    1, B
600 FVN = N - 1
610 STPX = (XAX - XIN) / FVN
620 XST = STPX
630 YDLTA = (YAX - YIN) / FSN
640 XX = XIN - XST
650 FOR I = 1 TO N
660 XX = XX + XST
670 X(I) = XX + DXIN: NEXT I
680 YY = YIN
690 XKKD = -DXIN: ZKKD = -DZIN
700 G2N = NGP: G3N = NPG1
710 MAXDM = MXDP
720 '
730 ' Дефиниране на функция, която определя
```

```
    ординатата
740 ' на точка с абциса XFN и лежаща на
    права,
750 ' съединяваща точките (XT1,Y1) и
    (XT2,Y2)
760 '
770 DEF FNL (XFN, XT1, Y1, XT2, Y2) = Y1
    (XFN - XT1)
    * (Y2 - Y1) / (XT2 - XT1)
780 IF KZ = 1 THEN ZIN = ZAX
790 FOR KKK = 1 TO NF
800 XKKD = XKKD + DXIN
810 ZKKD = ZKKD + DZIN
820 XX = XIN - STPX
830 FOR J = 1 TO N
840 XX = XX + STPX
850 ZZ = FNP(XX, YY)
860 ZZ = ZZ + ZKKD
870 AA = 2 * ZIN
880 IF KZ = 1 THEN ZZ = -ZZ
890 Z(J) = ZZ: Z1(J) = -ZZ
900 IF KZ = 0 THEN Z(J) = -ZZ: Z1(J) = ZZ
910 X(J) = X(J) - DXIN
920 P1Z(1, J) = Z(J): P1Z(2, J) = Z1(J)
930 NEXT J
940 N4P = N: N5P = G2N: N6P = NF: L = 1
950 '
960 ' Обръщение към подпрограма за
    изчертаване
970 ' на линиите,които са видими и се
    намират
980 ' над базовата линия
990 '
1000 GOSUB 1700
1010 N = N4P: G2N = N5P
1020 N4P = N1: N5P = G3N: N6P = 0: L = 2
1030 '
1040 ' Обръщение към подпрограма за
    изчертаване
1050 ' на линиите, които са видими и се
    намират
1060 ' под базовата линия
1070 '
1080 GOSUB 1700
1090 N1 = N4P: G3N = N5P
1100 YY = YY + YDLTA
1110 NEXT KKK
1120 '
1130 ' Изчертаване на координатните оси
1140 '
1150 YYY2 = YYMAX: YYY5 = YYMIN
1160 IF KZ = 0 THEN YYY2 = YYMIN: YYY5 =
     YYMAX
1170 YYY3 = YYY2 + KOR * FSN * DZIN
1180 YYY4 = YYYP + KOR * FSN * DZIN
1190 IF KZ = 1 THEN YYY3 = YYY2 - KOR * FSN *
     DZIN
1200 YYY1 = YYY2
1210 XXX3 = XXX2 - FSN * DXIN
1220 PSET (XXX1 * MX + MXD, YYY1 * MY + MYD),
     0
1230 LINE -(XXX2 * MX + MXD, YYY2 * MY + MYD)
1240 LINE -(XXX3 * MX + MXD, YYY3 * MY + MYD)
1250 LINE -(XXX3 * MX + MXD, YYY5 * MY + MYD)
1260 '
1270 ' Градуировка на оста X
1280 '
1290 XXXD = (XXX1 - XXX2) / 5
1300 FOR I = 1 TO 6
1310 PSET (XXX1 * MX + MXD, YYY1 * MY + MYD -
     1)
1320 LINE -(XXX1 * MX + MXD, YYY1 * MY + MYD
     + 1)
1330 XXX1 = XXX1 - XXXD: NEXT I
1340 '
1350 ' Нанасяне на мащаба по оста X
1360 '
1370 MIX = 3: MIY = 2
1380 LY = INT((YYY2 * MY + MYD) / 8) + MIX:
     SX = INT((XXX2 * MX + MXD) / 8) - 1
1390 SXD = INT((XXXD * MX) / 8) + 1: XP = XIN
1400 FOR I = 1 TO 6: LOCATE LY, SX
1410 XPP = (INT(XP * 100)) / 100: PRINT XPP:
     XP = XP + (XAX - XIN) / 5: SX = SX +
     SXD: NEXT I
1420 LOCATE LY - MIX, SX - SXD + 3: PRINT "X"
1430 '
1440 ' Градуировка на оста Z
1450 '
1460 YYYD = (YYY3 - YYY5) / 3: YYYI = YYY3
1470 FOR I = 1 TO 4
1480 PSET (XXX3 * MX + MXD - 3, YYYI * MY +
     MYD)
1490 LINE -(XXX3 * MX + MXD + 3, YYYI * MY +
     MYD)
1500 YYYI = YYYI - YYYD
1510 NEXT I
1520 '
1530 ' Нанасяне на мащаба по оста Z
1540 '
1550 YP = YYY2: SY = INT((YYY3 * MY + MYD)/8
     + .5) + 1: LY = INT((XXX3 * MX + MXD)/
```

```
     8) - 6: SYD = INT((YYYD * MY)/8 + .5)
1560 FOR I = 1 TO 4
1570 LOCATE SY, LY: YPP = INT(YP * 100):YPP =
     YPP / 100
1580 IF KZ = 1 THEN YPP = -YPP
1590 PRINT YPP
1600 YP = YP - (YYY3 - YYY5) / 3
1610 SY = SY - SYD
1620 NEXT I
1630 '
1640 ' Написване на изображението
1650 '
1660 LOCATE SY + SYD - MIY, LY + 4: PRINT "Z"
1670 LOCATE 23, 5: PRINT NAD$
1680 LOCATE 25, 1
1690 END
1700 '
1710 ' Подпрограма за изчертаване на линиите
1720 ' на отделните сектори на повърхнината
1730 ' с отстраняване на невидимите части
1740 '
1750 IF MAXDM <= 0 THEN RETURN
1760 FIPOT = 1
1770 IF N4P <= 0 THEN N4P = -N4P: FIPOT = 0
1780 IF N5P > 0 THEN 2300
1790 IF (N4P + 4) <= MAXDM THEN 1860
1800 MAXDM = -MAXDM
1810 RETURN
1820 '
1830 ' При SIGN=1 се чертае максимум,
1840 ' при SIGN=-1 - минимум
1850 '
1860 SIGN = 1
1870 IF N5P < -1 THEN SIGN = -1
1880 FM1 = N6P - 1
1890 '
1900 ' Стойностите на масивите P2Z и P3Z
1910 ' определят изчертаваните линии.
1920 ' Стойността им се актуализира на
1930 ' всяка стъпка. Новите стойности са
1940 ' визуалния максимум (минимум) на
1950 ' функцията относно изчертаните
1960 ' до този момент линии.
1970 '
1980 INDEXT = 3
1990 FOR KM = 1 TO N4P
2000 P2Z(L, INDEXT) = X(KM)
2010 P3Z(L, INDEXT) = P1Z(L, KM)
2020 INDEXT = INDEXT + 1
2030 NEXT KM
2040 N5P = N4P + 4
2050 P2Z(L, 1) = -FM1 * DXIN + XIN - ABS(XIN
       ABS(P2Z(L, 3)) - 1!
2060 P2Z(L, 2) = P2Z(L, 3) - EPS
2070 N9 = N4P + 3: N8 = N4P + 2
2080 P2Z(L, N9) = P2Z(L, N8) + EPS
2090 ZZ = ZIN
2100 IF SIGN < 0 THEN ZZ = -ZIN - 50! * ZDLTA
2110 P3Z(L, 1) = ZZ: P3Z(L, 2) = ZZ
2120 P3Z(L, N9) = ZZ
2130 P3Z(L, N5P) = ZZ
2140 IF FIPOT <> 1 THEN 2270
2150 QA = X(1): QB = P1Z(L, 1)
2160 YYMAX = QB: YYMIN = QB
2170 PSET (QA * MX + MXD, QB * MY + MYD), 0
2180 FOR KO = 2 TO N4P
2190 QA = X(KO): QB = P1Z(L, KO)
2200 LINE -(QA * MX + MXD, QB * MY + MYD)
2210 IF YYMAX < QB THEN YYMAX = QB
2220 IF YYMIN > QB THEN YYMIN = QB
2230 NEXT KO
2240 XXX1 = X(N4P)
2250 XXX2 = X(1)
2260 YYYP = P1Z(L, 1)
2270 RELING = XDLTA / ZDLTA
2280 P2Z(L, N5P) = SIGN
2290 RETURN
2300 SIGN = P2Z(L, N5P)
2310 P2Z(L, N5P) = X(N4P)
2320 KB = 2
2330 IF P2Z(L, KB) = X(1) THEN 2360
2340 IF P2Z(L, KB) < X(1) THEN KB = KB + 1:
     GOTO 2330
2350 IF P2Z(L, KB) > X(1) THEN KB = KB - 1
2360 IF KB >= MAXDM THEN 3810
2370 FOR M1 = 1 TO KB
2380 XH(M1) = P2Z(L, M1)
2390 H(M1) = P3Z(L, M1): NEXT M1
2400 IG = KB + 1: XH(IG) = X(1)
2410 H(IG) = FNL(X(1), P2Z(L, KB), P3Z(L,
     KB), P2Z(L, IG), P3Z(L, IG))
2420 IDEXG = KB: INDEXT = 1
2430 ZS = X(1)
2440 F1 = H(IG) - P1Z(L, 1)
2450 IT = 2
2460 KB = IG
2470 IF H(IG) >= P1Z(L, 1) THEN 2520
2480 IF KB >= MAXDM THEN 3810
2490 KB = IG + 1
2500 H(KB) = P1Z(L, 1)
2510 XH(KB) = ZS + EPS
2520 LAST = 0: XC = ZS
```

```
2530 IF P2Z(L, IG) < X(IT) THEN 2580
2540 IWHICH = 0: X2 = X(IT)
2550 F2 = FNL(X2, P2Z(L, IG - 1), P3Z(L, IG -
     1), P2Z(L, IG), P3Z(L, IG))
2560 F2 = F2 - P1Z(L, IT)
2570 IT = IT + 1: GOTO 2610
2580 X2 = P2Z(L, IG): IWHICH = 1
2590 F2 = P3Z(L, IG) - FNL(X2, X(IT - 1),
     P1Z(L, IT - 1), X(IT), P1Z(L, IT))
2600 IG = IG + 1
2610 SW = F1 * F2
2620 IF SW > 0 THEN 2710
2630 IF X2 = XC THEN X2 = X2 + EPS
2640 SLOPE = (F2 - F1) / (X2 - XC)
2650 JGG = IG - 1 - IWHICH
2660 JTT = IT - 2 + IWHICH
2670 IF ABS(SLOPE * RELING) > 10 ^ -6 THEN
     2690
2680 ZB = X2: GOTO 2800
2690 ZB = XC - F1 / SLOPE
2700 GOTO 2800
2710 XC = X2: F1 = F2
2720 IF IT <= N4P THEN 2530
2730 LAST = 1: ZB = X(N4P): JGG = 2
2740 MZ = IDEXG + JGG - 1
2750 IF P2Z(L, MZ) = ZB THEN 2780
2760 IF P2Z(L, MZ) < ZB THEN JGG = JGG + 1:
     MZ = MZ + 1: GOTO 2750
2770 IF P2Z(L, MZ) > ZB THEN JGG = JGG - 1:
     MZ = MZ - 1
2780 JGG = IDEXG + JGG - 1
2790 JTT = N4P - 1
2800 ZZ = .99 * ZS + .01 * ZB
2810 K1 = 2
2820 LS = INDEXT + K1 - 1
2830 IF X(LS) = ZZ THEN 2860
2840 IF X(LS) < ZZ THEN K1 = K1 + 1: LS = LS
     + 1: GOTO 2830
2850 IF X(LS) > ZZ THEN K1 = K1 - 1: LS = LS
     - 1
2860 K2 = 2
2870 JS = IDEXG + K2 - 1
2880 IF P2Z(L, JS) = ZZ THEN 2910
2890 IF P2Z(L, JS) < ZZ THEN K2 = K2 + 1:JS =
     JS + 1: GOTO 2880
2900 IF P2Z(L, JS) > ZZ THEN K2 = K2 - 1
2910 K1 = K1 + INDEXT - 1
2920 K2 = K2 + IDEXG - 1
2930 IF (X(K1 + 1) = X(K1)) THEN 2950
2940 A = FNL(ZZ, X(K1), P1Z(L, K1), X(K1 +
     1), P1Z(L, K1 + 1))
2950 IF P2Z(L, K2 + 1) = P2Z(L, K2) THEN 2970
2960 AW = FNL(ZZ, P2Z(L, K2), P3Z(L, K2),
     P2Z(L, K2 + 1), P3Z(L, K2 + 1))
2970 IF A > AW THEN 3140
2980 IF (KB + JGG - IDEXG) >= MAXDM THEN 3810
2990 IF IDEXG = JGG THEN 3050
3000 MJ = IDEXG + 1
3010 FOR I = MJ TO JGG
3020 KB = KB + 1
3030 XH(KB) = P2Z(L, I)
3040 H(KB) = P3Z(L, I): NEXT I
3050 KB = KB + 1: XH(KB) = ZB
3060 H(KB) = FNL(ZB, P2Z(L, JGG), P3Z(L,
     JGG),P2Z(L, JGG + 1), P3Z(L, JGG + 1))
3070 IDEXG = JGG: INDEXT = JTT
3080 GOTO 3430
3090 '
3100 ' Ако функцията е видима в интервала
3110 ' ZS,ZB, актуализира се максимумът
3120 ' (минимумът) и се изчертава
3130 '
3140 NRAPH = JTT - INDEXT + 2
3150 IF (KB + NRAPH - 1) > MAXDM THEN 3810
3160 N2 = KB
3170 IF NRAPH = 2 THEN 3220
3180 J1 = INDEXT + 1
3190 FOR I = J1 TO JTT
3200 KB = KB + 1: XH(KB) = X(I)
3210 H(KB) = P1Z(L, I): NEXT I
3220 KB = KB + 1: XH(KB) = ZB
3230 H(KB) = FNL(ZB, X(JTT), P1Z(L, JTT),
     X(JTT + 1), P1Z(L, JTT + 1))
3240 IF N6P <> 0 THEN 3290
3250 FOR I = N2 TO N2 + NRAPH - 1
3260 H(I) = -H(I): NEXT I
3270 IF FIPOT <> 1 THEN 3420
3280 IF NRAPH - 1 < 3 AND N2 < 4 THEN 3420
3290 PSET (XH(N2) * MX + MXD, H(N2) * MY +
     MYD), 0
3300 FOR IIS = N2 TO N2 + NRAPH - 1
3310 LINE -(XH(IIS) * MX + MXD, H(IIS) * MY +
     MYD)
3320 '
3330 ' Определяне на абсолютната стойност на
3340 ' максимума и на минимума
3350 '
3360 IF YYMAX < H(IIS) THEN YYMAX = H(IIS)
3370 IF YYMIN > H(IIS) THEN YYMIN = H(IIS)
3380 IF XXX3 > XH(N2) THEN XXX3 = XH(N2):
     YYY4 = H(N2)
3390 NEXT IIS
```

```
3400 IF N6P <> 0 THEN 3420
3410 FOR I = N2 TO N2 + NRAPH - 1: H(I) =
     -H(I): NEXT I
3420 INDEXT = JTT: IDEXG = JGG
3430 IF LAST = 1 THEN 3540
3440 XC = X2: F1 = F2: ZS = Zb
3450 '
3460 ' След изчертаване на функцията в
3470 ' интервала ZS,ZB се търси следващият
3480 ' интервал за изобразяване, ако не е
3490 ' достигната последната дефинирана
3500 ' стойност на X
3510 '
3520 IF IT <= N4P THEN 2530
3530 GOTO 2730
3540 IF P2Z(L, N5P) <= P2Z(L, N5P - 1) THEN
     N5P = N5P - 1
3550 IF P2Z(L, N5P) <= X(N4P) THEN 3660
3560 IF (KB + 3 + N5P - JGG) > MAXDM THEN
     3810
3570 XH(KB + 1) = XH(KB) + EPS
3580 KB = KB + 1
3590 IF P2Z(L, JGG + 1) = P2Z(L, JGG) THEN
     3610
3600 H(KB) = FNL(X(N4P), P2Z(L, JGG), P3Z(L,
     JGG), P2Z(L, JGG + 1), P3Z(L, JGG + 1))
3610 JPGG1 = JGG + 1
3620 FOR J = JPGG1 TO N5P
3630 KB = KB + 1
3640 XH(KB) = P2Z(L, J)
3650 H(KB) = P3Z(L, J): NEXT J
3660 N5P = KB + 2
3670 IF N5P > MAXDM THEN 3810
3680 '
3690 ' В масивите P2Z и P3Z се запомнят
3700 ' стойностите на текущия максимум
3710 ' (минимум) на функцията
3720 '
3730 FOR I = 1 TO KB
3740 P3Z(L, I) = H(I)
3750 P2Z(L, I) = XH(I): NEXT I
3760 P3Z(L, KB + 1) = ZIN + ZKKD
3770 IF SIGN < 0 THEN P3Z(L, KB + 1) = -ZIN
     50! * ZDLTA + ZKKD
3780 P3Z(L, N5P) = P3Z(L, KB + 1)
3790 P2Z(L, N5P) = SIGN
3800 RETURN
3810 MAXDM = -MAXDM
3820 P2Z(L, N5P) = SIGN
3830 RETURN
```

ОТГОВОРИ

С тази статия отговаряме на въпросите, с които се обърнаха към нашата рубрика студентът Атанас Богданов от Варна и Илия Илиев от Хасково.

Интерпретаторът на Бейсик за Правец—8Д поддържа апарат за работа с числа с плаваща точка. Този апарат е съвкупност от подпрограми в постоянната памет. Тези подпрограми извършват различни аритметични действия с т. нар. акумулатори с плаваща точка. Като стана дума за акумулаторите, ще поясня, че това са всъщност две области от оперативната памет с дължина от шест байна. Като всички останали клетки от паметта те могат да бъдат използвани и за други цели. Например програмата Монитор (КВ.9—10.88) използва адресното пространство на акумулаторите за работни клетки. Обърнете особено внимание на този факт, ако работите с монитора и желаете да изпробвате подпрограмите за работа с числа с плаваща точка. Не че програмите няма да се изпълняват, но резултатът в акумулаторите ще бъде неверен. При работа с Монитора трябва резултатът да бъде съхранен на подходящо място в паметта.

Ето адресите на двата акумулатора с плаваща точка:

```
#D0 .. #D5  Първи акумулатор - ACC1
#D6 .. #DD  Втори акумулатор - ACC2
```

Обърнете внимание, че всеки от акумулаторите е с дължина шест байта, докато за съхранението на число с плаваща точка на произволно място в паметта са необходими пет байта. Това е така, защото шестият байт от акумулаторите е само за знака на числото. Така при акумулаторите „се губи" един байт, но се работи по-бързо. При съхраняването на числата в паметта е ясно защо се предпочита петбайтовият формат.

Искам да отбележа, че литературата, от която съм черпил информация (Clefs pour L ORIC), съдържаше четири съществени грешки. Грешки има и в книгата „Правец-8Д — професионални приложения" на Б. Захариев и Й. Йорданов, С., ДИ „Техника", 1989, така че реших сам да проверя всички подпрограми и да напиша два примера. Така че можете да разчитате на приведената по-долу информация. C< ADR и >ADR ще означавам съответно младши и старши байт на адрес от паметта. Така, ако ADR = #1234, то <ADR = #34, а >ADR =#12. C (ADR) ще означавам съдържанието на пет поредни байта от паметта от начален адрес ADR, а (ACC1) или (ACC2) означава съдържанието (шест байта) на съответния акумулатор.

АКУМУЛАТОРИТЕ С ПЛАВАЩА ТОЧКА

```
АДРЕС   ПАРАМЕТРИ       ДЕЙСТВИЕ
=================================================
#0021   ACC1            ACC1 = USR(ACC1)
#02FB   ACC1            ACC1 = &(ACC1)

#DE7B   A = <ADR        ACC1 = (ADR)
        Y = >ADR        Зареждане на ACC1 от паметта

#DEAD   ACC1            (ADR) = ACC1
        X = <ADR        Запазване на ACC1 в паметта
        Y = >ADR

#DEE5   ACC1            ACC2 = ACC1
#DED5   ACC2            ACC1 = ACC2

#D499   Y=мл.байт NUM   ACC1 = NUM
        A=ст.байт NUM   Преобразува цялото число със
                        знак NUM в реално число с
                        плаваща точка.
                        Резултатът остава в ACC1

#D92C   ACC1            NUM = ACC1
                        Преобразува реалното число
                        от ACC1 в двубайтово
                        целочислено число NUM.
                        В #D4 остава мл. байт на NUM,
                        а в #D3 - ст. байт на NUM.

#E0D5   ACC1            Преобразува числото от ACC1 в
                        ASCII поредица от адрес #100.

#DF21   ACC1            ACC1 = SGN(ACC1)
#DFBD   ACC1            ACC1 = INT(ACC1)
#DF49   ACC1            ACC1 = ABS(ACC1)
#E22E   ACC1            ACC1 = SQR(ACC1)
#E34F   ACC1            ACC1 = RND(ACC1)
#DCAF   ACC1            ACC1 = LN(ACC1)
#E2AA   ACC1            ACC1 = EXP(ACC1)
#E38B   ACC1            ACC1 = COS(ACC1)
#E392   ACC1            ACC1 = SIN(ACC1)
#E3DB   ACC1            ACC1 = TAN(ACC1)
#E43F   ACC1            ACC1 = ATN(ACC1)
#DDD4   ACC1            ACC1 = LOG(ACC1)
#DE77                   ACC1 = PI

#DB22   ACC1            ACC1 = (ADR) + ACC1
        A = <ADR
        Y = >ADR

#DBDB   ACC1            ACC1 = (ADR) - ACC1
        A = <ADR
        Y = >ADR

#DCED   ACC1            ACC1 = (ADR) * ACC1
        A = <ADR
        Y = >ADR

#DDE4   ACC1            ACC1 = (ADR) / ACC1
        A = <ADR
        Y = >ADR

#E235   ACC2            ACC1 = ACC2 ^ (ADR)
        A = <ADR        Основа на степента в ACC2!
        Y = >ADR
```

Ето примерна програма за генериране на цяло случайно число в диапазона от 0 до 1000. Ако изпълните командата DOKE#2FC; #400 и заредите програмата от начален адрес #400, ще можете да извеждате резултата с макрооператора &, като параметърът в скобите е без значение. Например PRINT &(0) ще изведе случайно число в избрания обхват.

```
        ORG  $400
RANGE   =    1000       ;ОБХВАТ НА СЛУЧАЙНИТЕ ЧИСЛА
        LDY  #<RANGE    ;МЛ. БАЙТ НА RANGE
        LDA  #>RANGE    ;СТ. БАЙТ НА RANGE
        JSR  $D499      ;ACC1 = 1000 (REAL)
        LDX  #<TEMP     ;МЛ. АДРЕС НА TEMP
        LDY  #>TEMP     ;СТ. АДРЕС НА TEMP
        JSR  $DEAD      ;(TEMP) = ACC1
                        ;СЛЕДВАЩИТЕ ТРИ ИНСТРУКЦИИ
                        ;СА ИЗЛИШНИ, ТЪЙ КАТО В
                        ;ACC1 ИМА ЧИСЛО > 0
        LDA  #0         ;RND ИЗИСКВА ПАРАНЕТЪР > 0,
        LDY  #1         ;НЕКА ИЗБЕРЕМ ЧИСЛОТО 1
        JSR  $D499      ;ACC1 = 1 (REAL)
        JSR  $E34F      ;ACC1 = RND(1)
        LDA  #<TEMP
```

```
        LDY  #>TEMP
        JSR  $DCED      ;ACC1 = (TEMP) * ACC1
        JMP  $DFBD      ;ACC1 = INT(ACC1)
TEMP    DS   5          ;ПЕТ БАЙТА ЗА ЕДНО
                        ;РЕАЛНО ЧИСЛО
```

Вторият пример демонстрира един от начините за изчисляване на израза SIN (PI/6) *100, като резултатът остава в ACC1. Нека предварително с инструкцията

MEM DS 5

е дефинирана област от пет байта с име MEM, която ще съхранява междинните резултати.

```
        JSR  $DE77      ;ACC1 = PI
        LDX  #<MEM
        LDY  #>MEM
        JSR  $DEAD      ;(MEM)=ACC1=PI
        LDA  #0
        LDY  #6         ;КОНСТАНТА 6
        JSR  $D499      ;ACC1 = 6
        LDA  #<MEM
        LDY  #>MEM
        JSR  $DDE4      ;ACC1=(MEM)/6=PI/6
        JSR  $E392      ;ACC1=SIN(ACC1)
        LDX  #<MEM
        LDY  #>MEM
        JSR  $DEAD      ;(MEM)=ACC1=SIN(PI/6)
        LDA  #0
        LDY  #100       ;КОНСТАНТА 100
        JSR  $D499      ;ACC1 = 100
        LDA  #<MEM
        LDY  #>MEM
        JSR  $DCED      ;ACC1=(MEM)*ACC1
        ...             ;ACC1=SIN PI/6)*100
        ...             ;ОСТАНАЛАТА ЧАСТ
        ...             ;ОТ ПРОГРАМАТА
```

Инж. БОРИСЛАВ ЗАХАРИЕВ

Още един начин за превръщане на машинни програми в оператори DATA при Правец–8Д

От две години притежавам Правец-8Д. Много ми харесва, че в списание „Компютър за вас" често отговаряте на писма. Направи ми впечатление, че някои програми са придружени с таблица на контролни суми. Струва ми се, че много бихте улеснили всички, които въвеждат програми на Правец-8Д, ако и те са придружени с контролни суми.

Владимир Христов
Ловеч – 5500
ЖК „Маркс и Енгелс"
бл. 206, вх.А, ап. 8

Дисковата операционна система ДОС 8Д дава възможност за превръщането на област от паметта на Правец 8Д в оператори DATA. Така машинни програми или данни се превръщат в част от програма на Бейсик. Предлаганата програма генерира програмни редове DATA с по осем еднобайтови числа във всеки ред, като добавя контролна сума за реда. При грешно въведено число контролната сума позволява да се локализира грешката с точност до осем числа един ред. Извежда се съобщението „ГРЕШКА В РЕД XXX". При генерирането на операторите DATA се създава временен текстов файл с име TEMP.TXT, който после може да бъде изтрит.

Освен съдържанието на програмните редове с DATA временният файл съдържа още програмния ред 30, в който са посочени началният и крайният адрес на областта за преобразуване. Файлът съдържа също и числата от 0 до 12, които изтриват програмните редове с тези номера. В крайна сметка в паметта на компютъра остава програма на Бейсик, която след стартиране предизвиква зареждане на машинната програма или данните в паметта там, където е необходимо.

Първият програмен ред с DATA е с номер 100, но може да бъде зададен и друг номер, като се промени началната стойност на променливата L. Междуредовата стъпка е 10, но също може да бъде зададена друга стойност. Тези промени трябва да се извършат както в генериращата, така и в зареждащата част на програмата.

Инж. БОРИСЛАВ ЗАХАРИЕВ

```
 0 L=100:F$="FILE.TMP":B$=CHR$(2)
 1 INPUT"НАЧАЛО";B:INPUT"КРАЙ  ";E
 2 LPRINT B$"MON O":LPRINT B$"OPEN"F$
 3 LPRINTB$"CLOSE":LPRINTB$"DELETE"F$
 4 LPRINTB$"OPEN"F$:LPRINTB$"WRITE"F$
 5 FOR I=0 TO 12:LPRINT I:NEXT
 6 LPRINT"30 FOR I="HEX$(B)" TO
   "HEX$(E)" STEP 8"
 7 FOR I=B TO E STEP 8:S=0:LPRINT
   L;"DATA ";
 8 FOR J=0 TO 7:N=PEEK(I+J):S=S+N
 9 LPRINT RIGHT$(HEX$(N+256),2)",";
10 NEXT:LPRINT RIGHT$(HEX$(S+4096),3)
11 L=L+10:NEXT:LPRINT:LPRINTB$"CLOSE"
12 LPRINTB$"MON C":LPRINTB$"-"F$:END
20 L=100:PRINT"ЗАРЕЖДАНЕ":
40 S=0:FOR J=0 TO 7:READ A$
50 A=VAL("&"+A$):S=S+A:POKE I+J,A:NEXT
60 READA$:A=VAL("&"+A$):IF A=S THEN90
70 PRINT CHR$(7):PRINT
80 PRINT"ГРЕШКА В РЕД":L:LIST 100-
90 L=L+10:PRINT".":NEXT
```

СЕРИЕН ОБМЕН между ПРАВЕЦ–82 И ПРАВЕЦ–8Д

Разполагам с модул за последователен интерфейс RS 232 Board, производство на завода в Правец. Моля за кратка информация за начина за използване на този модул при прехвърляне на данни от Правец-82 към Правец-16.

Ст. инж. Ст. Джиев
Бургас
ЖК „Толбухин", бл. 4, вх.А, ет. 4

Проблемът за прехвърлянето на данни между Правец—82 и Правец—16 е бил неведнократно разглеждан на страниците на списанието. За тези, които разполагат със серийната интерфейсна платка RS 232C (за повече информация виж КВ.8.87), напомняме статията на Константин Коручев „Почти всичко за прехвърлянето на данни между Правец—82 и Правец—16" от КВ.1—2.88. Решение за директен обмен на данни между Правец—82 и Правец—16 или техни модификации в подходяща проводникова среда е предложено в статията на Август Стоянов „Прехвърляне на файлове през принтерския порт", КВ.9—10.89. Дискетите с файловете на програмите за директно прехвърляне са на разположение на всички, които дойдат в редакцията да си ги изкопират.

Притежавам компютър Правец—8С. Нямаме описание за работа с компютъра. Моля ви, ако имате възможност, изпратете ми ръководство за работа с него.

Петър ИВ. Батачки
ул. Явор 12
гара Елен Пелин

Ръководство за работа с Правец—8С не е издавано. На всички наши читатели, които се интересуват от начина на работа и *програми за домашния компютър Правец—8С, предлагаме* списък на публикуваните в списанието материали по темата. Пълната програмна съвместимост между компютрите Правец—8А и Правец—8С, както и сходствата в архитектурата и организацията им, прави доста полезна и поредицата статии, посветени на модела Правец—8А. Новият домашен компютър на КМНТ — Правец, работи и с интегрирания програмен продукт Парис (списък на публикациите за този пакет ще намерите в КВ. 3—4. 90).

1. Правец—8А. КВ. 1, 3, 4, 6, 7, 8, 9, 10—11. 87.
2. 64 Кбайта РАМ за Правец—8Е и Правец—8А. КВ. 1—2. 88
3. Подпрограмни на монитора. КВ. 4—5, 6, 7—8. 88
4. Към: Усъвършенстване на Правец—8Е. КВ. 7—8. 88
5. Бейсик за Правец—8А. КВ. 9—10. 88
6. *Правец—8С. КВ. 1—2. 90*
7. Отново за Правец—8С. КВ. 3—4. 90
8. КВ. 5—6. 90.

Освен това добра работа в много случаи могат да ви свършат книгите от поредицата „Микрокомпютърна техника за всички" на ДИ „Техника". Специално ви препоръчваме книгата „Правец—8А" на П. Петров; С., „Техника", 1989.

Предлаганата програма е предназначене за двупосочен *асинхронен обмен на данни* по сериен канал между компютрите Правец—82 и Правец—8Д. Функциите на серийните адаптори са осъществени програмно като драйверни програми, а схемата на връзка между компютрите е максимално опростена — всичко три съединителни проводника. При това достигнатата скорост на обмен надвишава 10 кбода. Предвидена е възможност за обмен на блокове *двоична (или текстова)* информация и на текстове на бейсикови програми.

Схемата на свързване на двата компютъра е следната (фиг. 1): изводите STB и ACK от куплунга за принтер на Правец—8Д са свързани съответно към изводите ЦВХ2 и ЦИЗХО от куплунга за игри на Правец—82. Третият проводник е *електрическа маса*. При тази схема на свързване е избегната опасността от конфликт *при обмена*, тъй като всеки от свързващите информационни проводници се използва само в една посока — или само като вход, или само като изход.

Правец—82			Правец—8Д	
4	ЦВХ2	←	1	STB
15	ЦИЗХО	→	19	ACK
8	GND	—	2	GND

фиг. 1

Протоколът на обмен е следният: един стартов бит, осем информационни бита (при текстове на бейсикови програми — седем информационни бита), един бит контрол по четност и стопов бит. Времето, необходимо за предаване на един бит, е около 64 ms.

Изпращането на данни винаги се предхожда от контролен блок с дължина 6 байта, който съдържа данни за типа на изпращания файл (машинен или бейсиков), адреса на разполагане и дължината на файла (машинен или бейсиков), адреса на разполагане и дължината на файла (при машинни файлове) и един байт контролна сума.

Обменът на бейсикови програми съдържа някои особености:

● Поради несъответствие в кодовете на бейсиковите оператори на двата компютъра се налага преобразуването на бейсиковите програми в текстови файлове при предаване и обратно преобразуване в бейсикови програми при приемане. Първото преобразуване се извършва от модифициран оператор LIST от предаващата страна, а второто — чрез интерпретатора на приемащия компютър (входният текстов файл през пренасочен входен вектор се подава директно на интерпретатор).

● Различното време, необходимо на интерпретатора за обработка на различните знакове (напр. CR) и на различно дългите програмни редове, налага нуждата предаващият компютър да очаква сигнал „готовност за приемане" преди изпращането на поредния знак.

● Трябва да се взема под внимание разликата в дължината на програмния ред при Правец—82 и Правец—8Д — съответно 255 и 80 знака.

Чрез командите CALL # B000 (CALL 5000) се създават новите бейсикови оператори ICLOAD и ICSAVE (&L и &S). Означенията извън скобите са на Правец—8Д, а тези в скобите — за Правец—82.

Новите команди имат следния формат:

ICSAVE (&S) — изпраща бейсиковата програма, която се намира в паметта;

ICSAVE, A *нач. адрес*, E *кр. адрес*, N *нач. адрес2*

(&S, A *нач. адрес*, E *кр. адрес*, N *нач. адрес2*) — изпраща двоичен файл, който се намира от *нач. адрес* до *кр. адрес* и го разполага от *нач. адрес2* в приемащия компютър;

ICLOAD (&L) — приема файл (независимо от вида).

Операторът &S с параметър (&S, C) пренебрегва интервалите при предаване на бейсикова програма от Правец—82 към Правец—8Д. В посока към Правец—82 интервалите винаги се пренебрегват. И в двата случая това не се отнася за интервалите, заключени между кавички.

Новите бейсикови оператори могат да се използват както в директен, така и в програмен режим.

Внимание! При обмена винаги първо се стартира операторът за приемане (приемащата страна).

ЗЛАТИН ГЕОРГИЕВ

ПРОГРАМА ЗА ПРАВЕЦ–8Д

```
20 L=100:PRINT"ЗАРЕЖДАНЕ";
30 FOR I=#B000 TO #B352 STEP 8
40 S=0:FOR J=0 TO 7:READ A$
50 A=VAL("#"+A$):S=S+A:POKE I+J,A:NEXT
60 READA$:A=VAL("#"+A$):IF A=S THEN90
70 PRINT CHR$(7):PRINT
80 PRINT"ГРЕШКА В РЕД";L:LIST 100-
90 L=L+10:PRINT".";:NEXT:CALL #B000
100 DATA A9,34,8D,F5,02,A9,B2,8D,449
110 DATA F6,02,60,AE,11,B3,A0,00,36A
120 DATA AD,0C,03,29,FE,8D,0C,03,27F
130 DATA 2C,01,03,A9,02,2C,0D,03,117
140 DATA F0,FB,AD,0C,03,49,01,8D,37E
150 DATA 0C,03,2C,01,03,B4,01,A0,164
160 DATA 02,88,D0,FD,A4,01,84,01,381
170 DATA A0,04,88,D0,FD,A4,01,A9,447
180 DATA 02,2D,0D,03,4A,4D,0C,03,0E5
190 DATA 8D,0C,03,4A,90,02,B0,01,229
200 DATA C8,66,00,AD,01,03,EA,CA,393
210 DATA D0,DC,B4,01,A0,03,8B,D0,42C
220 DATA FD,A4,01,98,4D,0C,03,0A,2A0
230 DATA 4D,0D,03,29,02,D0,02,38,192
240 DATA 60,A5,00,49,FF,18,60,A5,36A
250 DATA 08,C5,0A,A5,09,E5,0B,E6,35B
260 DATA 08,D0,02,E6,09,60,78,A9,34A
270 DATA 00,85,07,A9,0B,8D,11,B3,28E
280 DATA 20,0B,B0,90,09,20,C4,B0,308
290 DATA 20,BA,B0,4C,03,C0,A0,00,339
300 DATA 91,08,45,07,85,07,20,77,208
310 DATA B0,90,E5,20,0B,B0,B0,E5,495
320 DATA C5,07,F0,06,20,CB,B0,4C,3A9
330 DATA 9B,B0,AD,0C,03,29,FE,8D,3B8
340 DATA 0C,03,58,60,A9,12,A0,B3,2D5
350 DATA 4C,B0,CC,A9,25,A0,B3,4C,435
360 DATA B0,CC,A9,3A,A0,B3,4C,B0,4AE
370 DATA CC,A9,EF,20,00,03,8D,00,321
380 DATA 03,60,A9,10,0D,00,03,8D,1B9
390 DATA 00,03,60,85,00,AD,11,B3,259
400 DATA 85,01,A0,00,20,D9,B0,EA,3B9
410 DATA EA,46,00,90,06,20,E2,B0,378
420 DATA C8,B0,08,20,D9,B0,EA,EA,4FA
430 DATA A2,04,CA,D0,FD,E4,00,C6,4E7
440 DATA 01,D0,E6,EA,98,4A,90,05,418
450 DATA 20,D9,B0,B0,04,20,E2,B0,40F
460 DATA EA,A2,08,CA,D0,FD,20,E2,52D
470 DATA B0,60,78,A9,00,85,07,A9,366
480 DATA 08,8D,11,B3,A0,00,B1,08,2B2
490 DATA 48,20,EB,B0,68,45,07,85,33C
500 DATA 07,20,77,B0,90,EE,A5,07,378
510 DATA 20,EB,B0,3B,60,A9,00,85,3A1
520 DATA 09,85,0B,A9,02,85,0B,A9,27A
530 DATA 06,85,0A,60,A9,FF,85,02,324
540 DATA 38,AB,51,B3,ED,4F,B3,48,420
550 DATA AD,52,B3,ED,50,B3,A8,68,4B2
560 DATA 18,65,03,85,05,98,65,04,20B
570 DATA 85,06,20,4D,B1,20,2A,B1,2A4
580 DATA 20,8B,B1,A2,0F,CA,D0,FD,4A4
590 DATA 4C,2A,B1,A2,03,BD,4F,B3,388
600 DATA 95,08,CA,10,FB,60,A2,03,374
610 DATA B5,03,95,0B,CA,10,F9,60,3B8
620 DATA 20,4D,B1,20,B6,B0,A5,02,31B
630 DATA C9,FF,F0,0C,A2,00,C9,01,430
640 DATA D0,03,4C,BE,B1,4C,C0,B1,44B
```

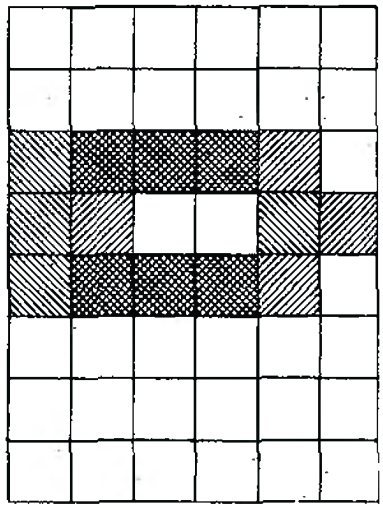
```
650 DATA 20,96,B1,4C,86,B0,A2,02,3BD
660 DATA 86,04,A2,22,86,03,A9,07,287
670 DATA 8D,11,B3,78,A9,D7,8D,3C,412
680 DATA 02,A9,B1,8D,3D,02,60,BE,316
690 DATA 4D,B3,8C,4E,B3,20,09,B0,436
700 DATA 20,0B,B0,90,06,20,C4,B0,305
710 DATA 4C,21,B2,48,20,E7,B0,68,381
720 DATA 38,6A,0B,29,7F,C9,11,D0,2FC
730 DATA 04,68,4C,21,B2,C9,22,D0,346
740 DATA 0B,48,45,03,85,03,68,D0,258
750 DATA 10,C9,20,D0,0C,A5,03,25,2A2
760 DATA 04,F0,04,28,4C,DD,B1,A9,343
770 DATA 20,AC,4E,B3,AE,4D,B3,28,3A3
780 DATA 60,A9,00,0B,20,E2,B0,A9,36C
790 DATA 7B,8D,3C,02,A9,EB,8D,3D,3A1
800 DATA 02,28,5B,60,20,E0,00,C9,2B3
810 DATA 86,D0,06,20,E7,00,4C,A0,37A
820 DATA B1,C9,B7,D0,0B,20,E2,00,40B
830 DATA DD,09,4C,FC,B2,20,D2,B0,475
840 DATA 4C,03,C0,C9,2C,D0,F6,20,3EA
850 DATA E2,00,C9,41,F0,03,4C,4D,378
860 DATA B2,20,E2,00,20,53,EB,8C,39B
870 DATA 4F,B3,8D,50,B3,20,E8,00,39A
880 DATA C9,2C,D0,EA,20,E2,00,C9,47A
890 DATA 45,D0,E3,20,E2,00,20,53,36D
900 DATA EB,8C,51,B3,8D,52,B3,20,42A
910 DATA E8,00,C9,2C,D0,D0,20,E2,47F
920 DATA 00,C9,4E,D0,C9,20,E2,00,3B2
930 DATA 20,53,EB,84,03,85,04,4C,2B7
940 DATA 5C,B1,A5,9A,85,CE,A5,9B,40F
950 DATA B5,CF,4C,66,C7,C9,0A,F0,490
960 DATA 1B,8C,4E,B3,8E,4D,B3,AA,3E0
970 DATA A9,02,2C,0D,03,F0,FB,2C,2FE
980 DATA 01,03,8A,20,EB,B0,AC,4E,343
990 DATA B3,AE,4D,B3,60,78,A9,07,3E9
1000 DATA 8D,11,B3,A9,64,8D,56,02,343
1010 DATA A9,AD,8D,3F,02,A9,B2,8D,40C
1020 DATA 40,02,20,16,C8,3B,6E,F2,2DB
1030 DATA 02,20,A2,B2,20,F0,CB,A9,3FA
1040 DATA 11,20,AD,B2,4E,F2,02,20,2F2
1050 DATA 2F,C8,5B,60,A9,FE,2D,0C,38F
1060 DATA 03,8D,0C,03,A9,00,B5,02,1CF
1070 DATA 20,4D,B1,20,2A,B1,4C,CD,332
1080 DATA B2,08,0D,0A,0E,BF,D0,C1,32F
1090 DATA D2,C9,D4,B9,A0,C5,D2,D2,651
1100 DATA 0D,0A,0E,07,00,0D,0A,0E,051
1110 DATA BF,C3,CB,C5,C3,CB,D3,D5,645
1120 DATA CD,A0,C5,D2,D2,0D,0A,0E,3FB
1130 DATA 07,00,0D,0A,0E,BF,D3,D9,297
1140 DATA CE,D4,C1,0B,A0,C5,D2,D2,644
1150 DATA 0D,0A,0E,07,00,01,35,D5,137
1160 DATA D0,C1,D2,55,55,55,55,55,40C
```

ПРОГРАМА ЗА ПРАВЕЦ-82

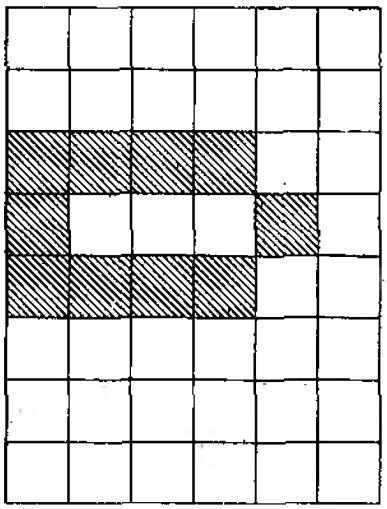
```
5000- A9 00 85 73 A9 50 85 74
5008- A9 6B 8D F6 03 A9 53 8D
5010- F7 03 4C A2 51 A0 00 AE
5018- E5 53 86 FE A9 80 2C 63
5020- C0 D0 FB A2 06 CA D0 FD
5028- A2 05 CA D0 FD E4 00 2C
5030- 63 C0 38 D0 03 1B 90 02
5038- C8 EA 66 FF A2 02 CA D0
5040- FD C6 FE D0 E3 A2 04 CA
5048- D0 FD 98 4A 6A 29 80 4D
5050- 63 C0 18 D0 01 38 A5 FF
5058- 60 AC E5 53 8C E7 53 A0
5060- 00 2C 58 C0 A2 05 CA D0
5068- FD 4A 90 06 2C 59 C0 C8
5070- B0 05 2C 58 C0 EA EA A2
5078- 08 CA D0 FD EA EA CE E7
5080- 53 D0 E6 98 4A 90 05 2C
5088- 59 C0 B0 04 2C 58 C0 EA
5090- A2 0C CA D0 FD 2C 59 C0
5098- 60 A9 00 85 CF A9 08 8D
50A0- E5 53 2C 59 C0 A0 00 B1
50A8- 3C 48 20 59 50 68 45 CF
50B0- 85 CF A2 0F 20 5C 51 20
50B8- 8A FC 90 E9 A5 CF 20 59
50C0- 50 2C 59 C0 60 A9 FF 85
50C8- F9 38 A5 3E E5 3C 48 A5
50D0- 3F E5 3D A8 68 18 65 42
50D8- 85 FC 98 65 43 85 FD A5
50E0- 42 85 FA A5 43 85 FB A2
50E8- 03 B5 3C 48 CA 10 FA 20
50F0- 32 51 20 99 50 A2 00 68
50F8- 95 3C E8 E0 04 D0 F8 A2
5100- 0A 20 60 51 4C 99 50 A9
5108- 00 85 CF A9 08 8D E5 53
5110- 20 15 50 90 03 4C 6E 51
5118- A0 00 91 3C 45 CF 85 CF
5120- 20 BA FC 90 EB 20 15 50
5128- B0 EB C5 CF F0 03 4C 75
5130- 51 60 A9 00 85 3D 85 3F
5138- A9 F9 85 3C A9 FD 85 3E
5140- 60 20 32 51 20 07 51 A5
5148- F9 C9 FF F0 03 4C D6 51
5150- A2 03 B5 FA 95 3C CA 10
5158- F9 4C 07 51 CA D0 FD 60
5160- BE EB 53 A2 00 CA D0 FD
5168- CE EB 53 D0 F6 60 A9 EC
5170- A0 53 4C 79 51 A9 F4 A0
5178- 53 20 3A DB 4C 2D FF A9
5180- B2 85 36 A9 51 85 37 A9
5188- FF 85 21 A9 07 8D E5 53
5190- 2C 59 C0 20 3C 52 A9 91
5198- 20 ED FD A9 28 85 21 20
51A0- 93 FE AD EA 03 C9 4C F0
51A8- 06 2C 81 C0 4C 03 E0 4C
51B0- EA 03 8D EB 53 8C E9 53
51B8- BE EA 53 48 A9 80 2C 63
51C0- C0 D0 FB 68 A2 08 20 5C
51C8- 51 20 59 50 AD EB 53 AC
51D0- E9 53 AE EA 53 60 78 2C
51D8- 59 C0 A9 07 8D E5 53 A9
51E0- A2 8D E6 53 A9 EF 85 38
51E8- A9 51 85 39 4C A2 51 BE
51F0- EA 53 8C E9 53 2C 58 C0
51F8- 2C 59 C0 20 15 50 90 06
5200- 20 6E 51 4C 2F 52 38 6A
5208- C9 91 D0 03 4C 2F 52 C9
5210- A2 D0 0A 48 4D E6 53 8D
5218- E6 53 68 D0 0B C9 A0 D0
5220- 07 2D E6 53 D0 CF A9 A0
5228- AE EA 53 AC E9 53 60 20
5230- B9 FE AE EA 53 AC E9 53
5238- A9 A0 58 60 A5 67 85 9B
5240- A5 68 85 9C AD FF D6 C9
5248- A6 F0 03 4C 95 52 A9 FF
5250- 85 50 85 51 A0 01 B1 9B
5258- F0 38 20 58 D8 20 FB DA
5260- C8 B1 9B AA C8 B1 9B C5
5268- 51 D0 04 E4 50 F0 02 B0
5270- 21 84 85 20 24 ED A9 20
5278- 20 A6 F7 A4 85 C8 20 9C
5280- F7 C8 AA D0 F9 A8 B1 9B
5288- AA C8 B1 9B 85 9C 86 9B
5290- D0 C2 4C FB DA A9 FF 85
5298- 50 85 51 A0 01 B1 9B F0
52A0- 44 20 58 D8 20 FB DA C8
52A8- B1 9B AA C8 B1 9B C5 51
52B0- D0 04 E4 50 F0 02 B0 2D
52B8- 84 85 20 24 ED A9 20 A4
52C0- 85 29 7F 20 5C DB A5 24
52C8- C9 21 90 07 20 FB DA A9
52D0- 05 85 24 C8 B1 9B D0 1D
52D8- A8 B1 9B AA C8 B1 9B 86
52E0- 9B 85 9C D0 B6 A9 0D 20
52E8- 5C DB 60 EA EA C8 D0 02
52F0- E6 9E B1 9D 60 10 CC 38
52F8- E9 7F AA B4 85 A0 D0 B4
5300- 9D A0 CF B4 9E A0 FF CA
5308- F0 07 20 ED 52 10 FB 30
5310- F6 A9 20 20 5C DB 20 ED
5318- 52 30 05 20 5C DB D0 F6
5320- 20 5C DB A9 20 D0 98 A9
5328- 00 F0 02 A9 01 85 F9 20
5330- 32 51 20 99 50 4C 7F 51
5338- 20 B1 00 C9 24 F0 06 20
5340- B7 00 4C 0C DA A2 00 B6
5348- 50 86 51 20 B1 00 49 30
5350- C9 0A 90 07 69 08 C9 7A
5358- B0 01 60 A2 03 0A 0A 0A
5360- 0A 0A 26 50 26 51 CA 10
5368- F8 30 E0 20 B7 00 C9 4C
5370- D0 09 20 B1 00 20 41 51
5378- 4C A2 51 C9 53 D0 63 20
5380- B1 00 D0 06 20 27 53 4C
5388- A2 51 C9 2C D0 54 20 B1
5390- 00 C9 43 D0 09 20 B1 00
5398- 20 2B 53 4C A2 51 C9 41
53A0- D0 40 20 38 53 A5 50 85
53A8- 3C A5 51 85 3D 20 B7 00
53B0- C9 2C D0 2E 20 B1 00 C9
53B8- 45 D0 27 20 38 53 A5 50
53C0- 85 3E A5 51 85 3F 20 B7
53C8- 00 C9 2C D0 15 20 B1 00
53D0- C9 4E D0 0E 20 38 53 A5
53D8- 50 85 42 A5 51 85 43 20
53E0- C5 50 4C A2 51 0B 00 00
53E8- 00 00 00 00 D0 C1 D2 C9
53F0- D4 D9 A0 00 C3 CB C5 C3
53F8- CB D3 D5 CD A0 00 00
```

Плавно изместване на фигури в режим HIRES

ХРИСТИСЛАВ НИКИТОВ
ПЕТЪР КОНЯРОВ



Фиг. 2. Същата фигура, изместена една точка надясно



Фиг. 1. Фигура, която ще се придвижва

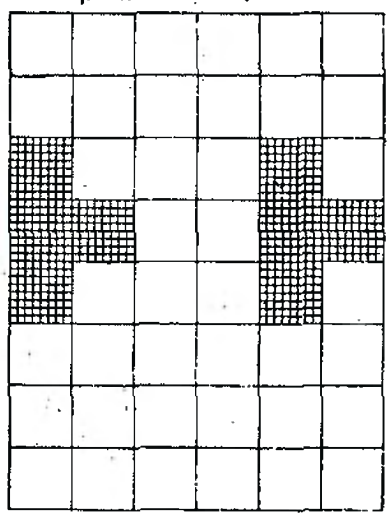
В основата на предлагания от нас метод, по който малки фигури могат плавно да се движат по екрана в режим HIRES, лежи възможността да се прекодира наборът от знакове на компютъра Правец-8Д.

Следващите разсъждения се отнасят за една матрица с размери 6 × 8 точки, т. е. с големината на един стандартен знак. Аналогично става движението на по-големи фигури (от няколко матрици). При преместването с една точка на фигурата в желаната посока при „обикновено" движение (т. е. първо се изтрива старото положение, след това се начертава новото) се получава мигане, което не е приятно и може да се избегне.

За целта е достатъчно да не се трие цялата фигура. Просто върху нея се начертава друга фигура, съставена по такъв начин, че тя елиминира определени точки от старата и добавя нови, така че в резултат се получава основното изображение, изместено в желаната посока. Начертаването става с параметър f=2 (размяна на цветовете на фона и мастилото). Преместващата фигура (или знак) може да се състави по следния начин:

1. Начертава се фигурата или част от нея. Ако тя ще се

Преместващ знак



Фиг. 3. Съставяне на дефиницията на преместващия знак

Дефиниция на преместващия знак

000000 = 0
000000 = 0
100010 = 34
110011 = 51
100010 = 34
000000 = 0
000000 = 0
000000 = 0

Началният адрес на знака А (латиница) е 46600. Следователно:

POKE 46600,0
POKE 46601,0
POKE 46602,34
POKE 46603,51
POKE 46604,34
POKE 46605,0
POKE 46606,0
POKE 46607,0

придвижва във вертикална посока, трябва да е свободен един от крайните редове, а при вертикална посока – една от крайните колони (фиг. 1).

2. Върху нея, но, изместена в посока към празната колона *или ред, се начертава същата* фигура или част от нея, като и двете трябва да се съберат в матрица 6 × 8 точки (фиг. 2).

3. Точките от двете фигури, които не се застъпват, дават искания знак (точките, които се застъпват, се пренебрегват). Полученият преместващ знак се пренася на чисто (фиг. 3).

4. Всеки ред от матрицата се представя като двоично число (светеща точка е единица, а несветеща – нула), което се превръща в десетично или шестнайсетично число.

5. Получените осем числа се записват в дефиницията на някой знак от стандартната или от алтернативна азбука (в примера на мястото на главно латинско А).

Остава да поясним как се осъществява самото движение. Начертава се основното изображение. После върху него с командата CHAR и параметър f = 2 се позиционира преместващият знак (знакове) и в резултат се вижда как фигурата се измества в посоката, където е празната колона *или ред. По-нататък движението* продължава, като преместващият знак се позиционира вече върху *новоизместеното изображение.*

Чрез преместващия знак фигурата може да се движи и в обратна посока, но този път той се позиционира не върху фигурата, а, изместен с една точка, в обратна посока. Свободният ред или колона (в зависимост от посоката на движението – вертикална *или хоризонтална*) са нужни, защото при преместването на фигурата тя няма да се *изтрие цялата.*

■

```
   ДЕМОНСТРАЦИОННА ПРОГРАМА

0 CLS:PRINTCHR$(17):PAPER0:INK7
10 CLS:FOR I=46600 TO
   46615:READA:POKEI,A:NEXT'ЗАРЕЖДА ДВЕТЕ
   ФИГУРИ
20 DATA 0,0,34,51,34,0,0,0'ПРЕМЕСТВАЩ ЗНАК
30 DATA 0,0,60,34,60,0,0,0'ОСНОВНА ФИГУРА
35 HIRES
40 X=100:Y=92:I=1:CURSETX,Y,3:CHAR
   65,0,1'ЧЕРТАЕ ОСНОВНАТА ФИГУРА
50 REPEAT'РЕДОВЕ 50-80 ДВИЖАТ ФИГУРАТА
60 GOSUB 110
70 IF X=100 OR X=140 THEN GOSUB 100
80 UNTIL KEY$=CHR$(13)'RETURN-КРАЙ
90 END
100 I=I*(-1):X=X+I'ОБРЪЩА ПОСОКАТА НА
   ДВИЖЕНИЕ
110 CURSET X,Y,3:CHAR65,0,2:X=X+I'ВЕЧЕ СЕ
   ЧЕРТАЕ САМО С ПРЕМЕСТВАЩИЯ ЗНАК
120 RETURN
```


Още 16 Кбайта РОМ

Инж. ВЛАДИСЛАВ ГЕОРГИЕВ

В Правец–8Д е предвидена възможността постоянната памет да бъде реализирана както с две интегрални схеми 2764, така и само с една 27128. *Във втория случай* върху печатната платка остават свободни места за *монтиране на* ИС 10 и ИС 11.

Свободното място позволява лесно да се вгради допълнителна постоянна памет с обем 16 Кбайта, реализирана на ИС 10 *тип* 27128. Тъй като адресните входове на епром ИС от серията 27 са високоомни, а изводите за данни са *с три състояния, то* включването на още една ИС няма да доведе до претоварване на адресната шина на микропроцесора и на шината за данни.

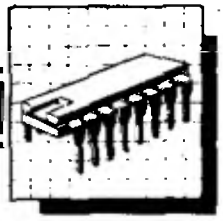
Двете постоянни памети ще *се намират на едни и същи* адреси в картата на паметта (#C000 – #FFFF), като ще се превключват алтернативно по програмен път. За целта е удобно да се използва свободният PB5 на гъвкавия интерфейсен адаптор (VIA 6522), тъй като порт B се ползва за системни нужди.

Промените, които трябва да се направят по платката на персоналния компютър, са следните:

1. Монтиране на ИС 11 *тип* 74LS00.

2. Снемане на мостчето J1, което се намира непосредствено до ИС 11.

3. *Монтиране на* цокъл за ИС с 28 извода на мястото за ИС 10 и поставяне в него на ИС *тип* 27128 в същата посока като ИС 9.



4. Прекъсване на печатния проводник, който свързва изводи 10 и 12 на ИС 11 с извод 23 на ИС 5 (R6502P). Последният се намира на страна спойки.

5. Свързване с проводник откъм страна спойки на изводи 10 и 12 на ИС 11 с извод 15 на ИС и 6 (PB5 на VIA 6522).

6. Свързване на резистор със стойност 10 KΩ от извод 10 на ИС 11 до извод 14 на същата ИС (+5V).

Желателно е също така да се прекъсне връзката на извод 27 на ИС 10 с извод 28 на същата ИС и да се свърже с извод 27 на ИС 9.

Това ще позволи входът ROMDIS от съединителя за разширение да е валиден независимо коя от двете ИС е избрана.

Избирането на допълнителните 16 Кбайта от постоянната памет може да става от машинни програми, като на адрес #0300 (ORB) се запише байт с пети бит, равен на нула (XX0X XXXX). Ако петият бит е равен на единица, ще бъде избрана основната постоянна памет.

В основната постоянна памет съществуват две подпрограми, които водят до нежелано избиране на допълнителната памет. Затова е необходимо да се промени съдържанието на следните клетки от оригиналния ром:

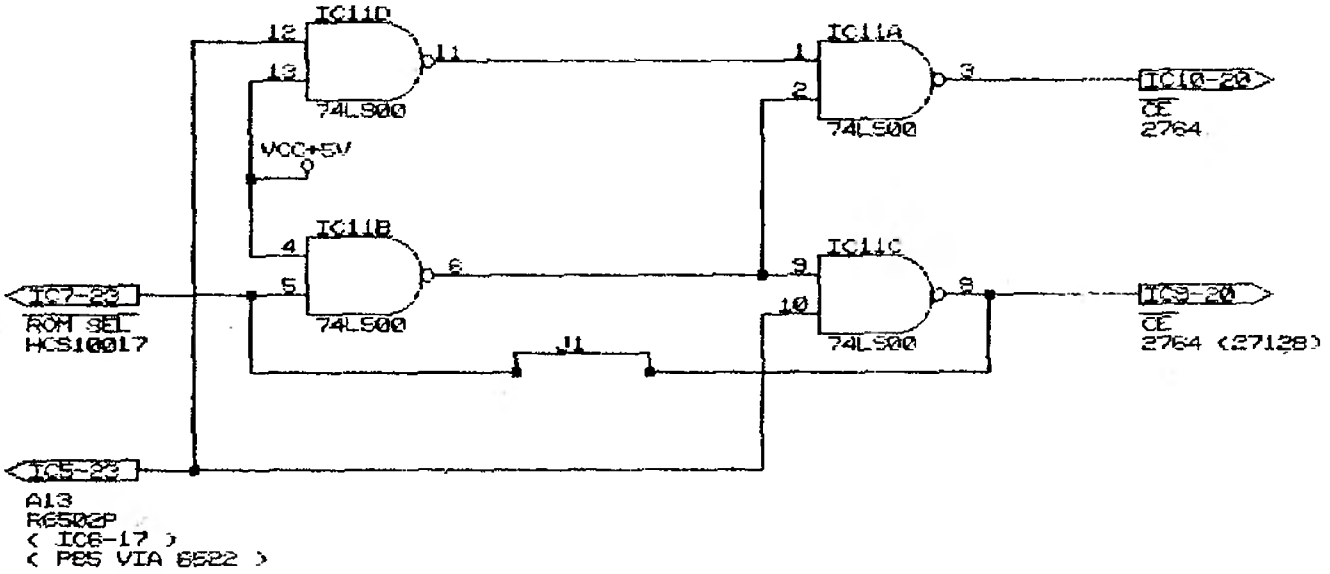
Адрес	Стара стойност	Нова стойност
277D	40	60
39B0	F7	B7
39B2	02	00
39B5	B7	F7
39B7	00	02

Тези промени практически не се отразяват на работата на компютъра. Първата промяна се отнася до подпрограмата за инициализиране на VIA6522 за работа с касетофон, а останалите четири — до подпрограмата INITVIA за начално инициализиране. Първата подпрограма записва на адрес #0300 стойността #40, с което включва релето, ако има вградено такова в компютъра.

Промяната на записваната стойност на #60 (0110 0000) освен включването на релето от PB6 запазва високото изходно ниво на PB5. Подпрограмата INITVIA се променя така, че първо да се зареди ORB и след това DDRB.

Това е необходимо, защото след първоначалното стартиране на компютъра всички регистри на VIA6522 са нулирани и ако първо се инициализира PB5 като изход, на него ще се установи ниско ниво.

След като направите описаните промени, ще разполагате с допълнителни 16 Кбайта постоянна памет и остава да решите проблема, какви програми ще разположите в нея.



ФИРМА НАЦИОНАЛЕН МЛАДЕЖКИ ЦЕНТЪР ПО ИНФОРМАТИКА „ИНФОРКОМ“

Пловдив 4000, ул. Н. Й. Вапцаров 20,

тел. (032) 26-79-93 факс: 802674

ПРЕДМЕТ НА ДЕЙНОСТ

1. РАЗРАБОТВАНЕ НА ИНФОРМАЦИОННИ ПРОЕКТИ, ПРОГРАМНО-АПАРАТНИ СИСТЕМИ, ПРОГРАМНИ ПРОДУКТИ, В ТОВА ЧИСЛО: СИСТЕМЕН (БАЗОВ) СОФТУЕР, ПРИЛОЖЕН СОФТУЕР (ОБУЧЕНИЕ, УПРАВЛЕНИЕ, ФИНАНСИ, МАШИННА ГРАФИКА), СОФТУЕР ЗА ПЕРСОНАЛНИ И ПРОФЕСИОНАЛНИ МИКРОКОМПЮТРИ, МИНИ И ГОЛЕМИ ЕИМ.
2. РАЗРАБОТВАНЕ И ВНЕДРЯВАНЕ НА НАУЧНОИЗСЛЕДОВАТЕЛСКИ И ТЕХНОЛОГИЧНИ ПРОДУКТИ.
4. ОБРАБОТКА НА ИНФОРМАЦИЯ И ИНФОРМАЦИОННО ОБСЛУЖВАНЕ.
5. ТЪРГОВСКА ДЕЙНОСТ В БЪЛГАРИЯ И ЗАД ГРАНИЦА.
6. ОБУЧЕНИЕ ЗА РАБОТА С ЕИМ.
7. ПОВИШАВАНЕ КВАЛИФИКАЦИЯТА НА СПЕЦИАЛИСТИ.
8. ПРОИЗВОДСТВО НА ТЕХНИЧЕСКИ ПОСОБИЯ В ОБЛАСТТА НА ЕЛЕКТРОНИКАТА.
9. КОНСУЛТАНТСКА И ЕКСПЕРТНА ДЕЙНОСТ В ОБЛАСТТА НА СОФТУЕРНОТО И ХАРДУЕРНОТО ПРОИЗВОДСТВО.
10. ПЕРСОНАЛНИ И ПРОФЕСИОНАЛНИ МИКРОКОМПЮТРИ – ДОСТАВКА, МОНТАЖ И ИНЖЕНЕРИНГ ПО ВАША ЗАЯВКА.
11. УНИВЕРСАЛНИ ЛОКАЛНИ МРЕЖИ „МИКРОРИНГ“ С 16-БИТОВИ МИКРОКОМПЮТРИ – ДОСТАВКА, МОНТАЖ, ИЗГРАЖДАНЕ, ОБУЧЕНИЕ НА КАДРИ.
12. МОДЕМИ ICA-110 (ПЛАТКОВИ) ЗА ДАЛЕЧНА ТЕЛЕКОМУНИКАЦИЯ ПО СТАНДАРТНА ТЕЛЕФОННА ЛИНИЯ ЗА МИКРОКОМПЮТРИ IBM PC/XT/AT.
13. ЕКСПРЕСНО КОМПЛЕКСНО ПРОЕКТИРАНЕ НА ГРАДОУСТРОЙСТВЕНИ, ПРОМИШЛЕНИ, ЖИЛИЩНИ, ЗДРАВНИ, СПОРТНИ, УЧЕБНИ, ТЪРГОВСКИ ОБЕКТИ И СЪОРЪЖЕНИЯ.

ФИРМАТА НМЦИ „ИНФОРКОМ“ С 10 ПОДЕЛЕНИЯ В БЪЛГАРИЯ ВИ ПРЕДЛАГА:

- *взаимноизгодно СЪТРУДНИЧЕСТВО!*
- *делови КОНТАКТИ!*
- *възможности за НОВИ ПАЗАРИ и успешен БИЗНЕС!*
- *маркетинг за ВАШИТЕ СТОКИ и УСЛУГИ!*
- *организиране на смесени производства и предприятия!*
- *финансово кадрово и ресурсно осигуряване на ВАШИТЕ ИДЕИ!*
- *НАШИТЕ ВЪЗМОЖНОСТИ са и ВАШИ ВЪЗМОЖНОСТИ!*

НЕ ЗАБРАВЯЙТЕ – НМЦИ „ИНФОРКОМ“ Е НА ВАШЕ РАЗПОЛОЖЕНИЕ!

ОПЪЛЧЕНЕЦ

Това е развлекателна игра, която стимулира развитието на съобразителността и съсредоточеността на играча, тъй като изисква ловкост, наблюдателност и бързина на действията. Предназначена е за играчи с различна степен на умения и има различни нива на трудност. Целта на играта е да не се допусне някое от многобройните човечета, които нападат стената, да се изкачи по нея. Ако опълченецът, който защитава стената, е достатъчно бърз и съобразителен, щом набере определен брой точки, армията на човечетата се изчерпва. Програмата съдържа демонстрация, от която се вижда как трябва да се защитава стената.

За управление на опълченеца се използват следните клавиши: лява стрелка — движение наляво; дясна стрелка — движение надясно; интервал — хвърля камък; 3 — взема камък.

ВЛАДИМИР КЮЧУКОВ

```
5 CLS:POKE&2FD,0
10 PRINT@5,12;"ИСКАТЕ ЛИ ДЕМОНСТРАЦИЯ
   (D/N)?":PRINTCHR$(20):GETDE$
20 DIMB(33):K=100:L=1:CO=25:X=20
30 CLS:INK1:PAPER3:POKE&26A,10:
   POKE&24E,4
40 FORI=1TO48:READT:POKE46599+I,T:NEXT
50 FORI=5TO34:FORQ=10TO26:PLOTI,Q,"A":
   NEXTQ,I
60 PLOT5,8,"AA":PLOT33,8,"AA":PLOT
   20,8,"A":PLOT5,9,"A":PLOT34,9,"A"
70 PRINT@13,3;CHR$(4);CHR$(27);
   CHR$(68);CHR$(27);"J ОПЪЛЧЕНЕЦ
   "CHR$(4);
80 PRINT@0,0;"ТОЧКИ 0":
   PRINT@18,0;"НИВО 1":
   PRINT@30,0;"ОБЩО 0...."
90 PRINT@X,9;"D"
100 F=0
110 IFDE$="d"THENGOSUB430ELSEJ$=KEY$
120 IFJ$=CHR$(8)ANDX>6THENX=X-1:
   PRINT@X+1,9;" "
130 IFJ$=CHR$(9)ANDX<33THENX=X+1:
   PRINT@X-1,9;" "
140 IFJ$="z"AND(X=6ORX=20ORX=33)
   THENR=1
150 IFJ$=" "ANDR=1THENR=0:PRINT@X,9;
   "D":GOTO240
160 IFR=1THENPRINT@X,9;"E"ELSEPRINT@X,
   9;"D"
170 E=E+1:IFE=20ORDE$="d"THENC=
   INT(RND(1)*26+7):E=0ELSE110
180 IFB(C)=0ANDF>KTHEN110
190 CO=CO+(B(C)=0)
200 IFB(C)=16THEN330ELSEB(C)=B(C)+1
210 IFB(M)<B(C)THENM=C
220 PRINT@C,26-B(C);"C";:PRINT@C,
   26-B(C)+1;"B";
230 GOTO110
240 CO=CO+ABS(B(X)>0)
250 SC=SC+B(X):F=F+B(X):B(X)=0
260 FORI=11TO26:PRINT@X,I;"F";:
   PRINT@X,I-1;"A";:NEXT:PRINT@X,26;
   "A";
270 ZAP
280 IFCO<>25THEN310
290 K=K+15:L=L+1:PRINT@22,0;L;
   PRINT@10,2;"ВИЕ ОТИВАТЕ В ";L;
   " НИВО"
300 EXPLODE:WAIT200:PRINT@10,2;
                    "
305 PRINT@5,0;"     ":GOTO100
310 PRINT@5,0;F:PRINT@34,0;SC
320 GOTO170
330 S=SGN(X-C)
340 FORI=CTOXSTEPS:PRINT@I,9;"C";
   PRINT@I-S,9;" "
350 PLAY1,0,1,10:WAIT10:NEXT
360 FORI=11TO26:PRINT@X,I;"D";
365 PRINT@X,I-1;"A";:MUSIC1,3,1+
   INT(.7*(27-I)),9:WAIT15:NEXT
370 PRINT@X,I-1;"A";:BNOAT
380 WAIT50:CLS:PRINT@10,12;
   "НОВА ИГРА ? (D/N)"
385 POKE&2DF,0:GETK$:PRINTCHR$(20)
390 IFK$="d"THENRUN
400 POKE&24E,32:CALLDEEK(&FFFA)
410 DATA63,33,33,33,33,33,33,63,18,
   18,18,18,18,30,18,18,18
415 DATA18,18,12,12,18,12,30,18,12,
   12,30,45,45,12,18,51
420 DATA63,63, 5,45,30,12,18,51,0,
   12,30,63,30,12,0,0
430 IFR=0THEN450
440 IFX=MTHENJ$=" ":RETURN:ELSEJ$=
   CHR$(9+(X>M)):RETURN
450 IFX=6ORX=20ORX=33THENJ$="z"
   ELSEJ$=CHR$(9+(X<14OR(X>20
   ANDX<26)))
460 RETURN
```

Бележка на научния консултант

За сравнително малкия си обем играта предлага интересна интрига. Използвани са възможностите на Правец—8Д за псевдографика. Имам следната незначителна забележка. Избраните клавиши за управление не винаги са удобни за всички. Чрез съответни изменения в програмни редове 120, 130, 140 и 150 могат да се изберат клавиши според предпочитанията на играещия.

Инж. БОРИСЛАВ ЗАХАРИЕВ

Списание „Компютър за вас“

ТАЛОН

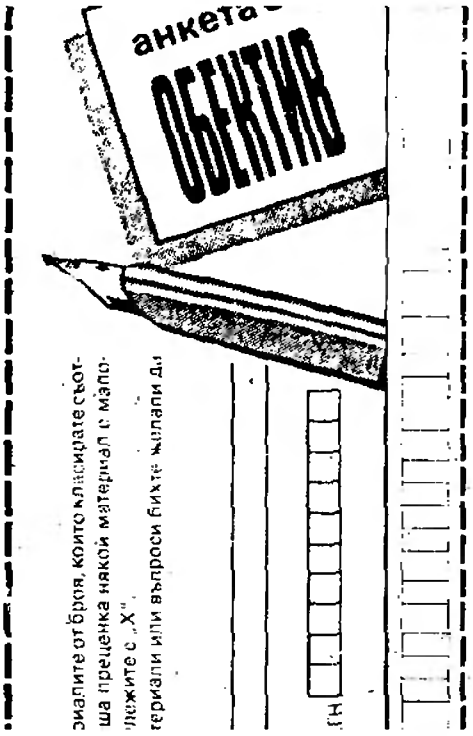
за безплатно записване на програмни продукти

1. Пакет антивирусни програми, В. Бончев, версия към 23.07.90 г. – 1 дискета.
2. Програма Bootchk (антивирусна), И. Досев.
3. Пакет антивирусни програми Hunter, А. Томов – 1 дискета.
4. Редактор за екрани Scrat, Н. Николов.
5. Интегриран програмен продукт ПАРИС – текстообработка, база данни, електронна таблица – 2 дискети.

Важи с попълнен талон на анкетата „Обектив“.

ЧЕТЕТЕ В СЛЕДВАЩИЯ БРОЙ

- Вирусни новини
- Обектно ориентирано програмиране
- Добрата стара *MS DOS 3.30*
- Тънкостите на Мерлин — част II
- Състезателни задачи
- Фортран конвертор
- Техника на форматирането с маски
- Верижно форматиране на дискети
- DIRLIST
- Компютърен тайнопис
- Прехвърляне на графична страница
- Игри за Правец-8Д



Съдържание Содержание

Стр.

NORTON COMMANDER 2

РАБОТА СЪС СПРАВОЧНИЦИТЕ

Управление

Ctrl – U	Разменя местата на двата панела
Ctrl – O	Скрива/показва панелите
Ctrl – P	Скрива/показва неактивния панел
Ctrl – F1	Скрива/показва левия панел
Ctrl – F2	Скрива/показва десния панел
Ctrl – L	Показва/скрива информация за ОП и текущия справочник
Tab	Смяна на активния панел (може и с Ctrl – I)
Ctrl – R	Повторно четене на справочника
Enter	Отваря/затваря справочник

Придвижване на маркера

Home	В началото на списъка (на съответния справочник)
End	В края на списъка
← →	Придвижване от колона в колона при триколонен режим
↑ ↓	Един ред нагоре (надолу) или прелистване на списъка
Ctrl – \	Преминаване в главния справочник
Alt – клавиш	Бързо избиране на файл по началната буква на името му (буквата на клавиша)

Избиране на файлове

Ins . ←	Избиране/(отказване от избора) на файл
+ (от цифровия блок)	Избиране на група файлове
– (от цифровия блок)	Отказване от избора

Избиране/(отказване от избора) на отделен или на група файлове
Прелистване на списъка (маркерът се намира в горния/долния край на панела и в него се появява стрелка) и едновременно избиране на файловете
Изпълнение на посочената команда (вж. клавиши Shift – F)
Отваряне на менюто за настройка на продукта (при посочена горна рамка на панела)

Изпълненне на посочената команда (вж. клавиши F)
Отваряне/затваряне на справочник (бързо двукратно натискане)
Стартиране на посочения файл (бързо двукратно натискане)
Прелистване на списъка (маркерът се намира в горния/долния край на панела и в него се появява стрелка)
Отваряне на менюто за настройка на продукта (при посочена горна рамка на панела)

Следва

ВСИЧКИ ЦЕНИ РАСТАТ!

Цените днес растат неприлично бързо.
Само цената на Правец-8 спадна три пъти за шест години...

1990 — Правец-8С
цена — 1560 лв.
+ само на 1 платка
+ има 64К ОП в повече
+ има ПроДОС в повече
+ и защото струва 2630 лв по-малко...

1984 — Правец-82
Цена — 4190 лв.
* общо на 7 платки