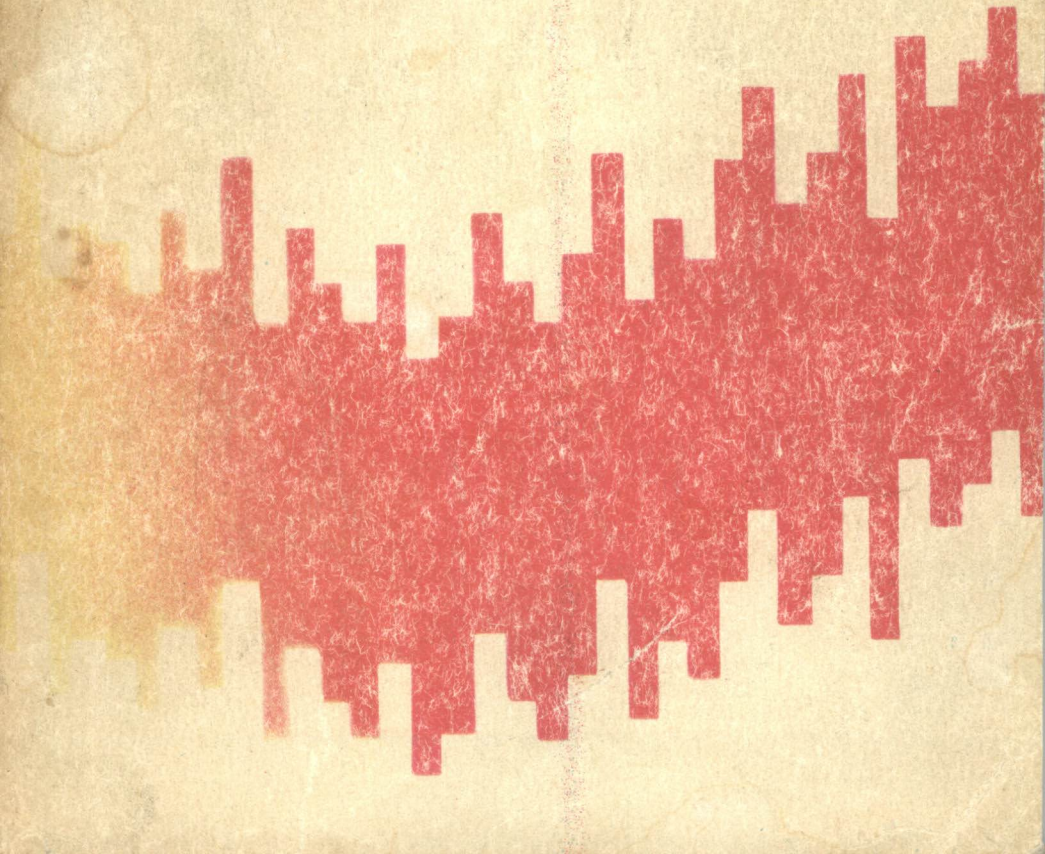
ПЕРСОНАЛЕН КОМПЮТЪР
ПРАВЕЦ 8А

П. ПЕТРОВ

АРХИТЕКТУРА, ОРГАНИЗАЦИЯ, ПРОГРАМНО ОСИГУРЯВАНЕ

ТЕХНИКА



Инж. Петър Д. Петров

ПЕРСОНАЛЕН КОМПЮТЪР ПРАВЕЦ-8А

АРХИТЕКТУРА, ОРГАНИЗАЦИЯ, ПРОГРАМНО ОСИГУРЯВАНЕ

Държавно издателство „Техника“

София, 1989

УДК 681.3

В книгата са разгледани програмните и апаратните особености на персоналния компютър Правец–8А. Описани са вградените входно–изходни устройства, организацията на системната и на допълнителната памет, работата в мониторен режим. Отделни глави са посветени на подпрограмите на програмата МОНИТОР, на командите на резидентния интерпретатор на БЕЙСИК и на устройството на компютъра. Особено внимание е отделено на разликите между персоналния компютър Правец–8А, от една страна, и компютрите Правец–82 и Правец–8М, от друга. В приложения са дадени наборът от инструкции и методите за адресиране на микропроцесора СМ630, входните точки на програмата МОНИТОР, адресите на програмноуправляемите ключове, кодовите таблици на клавиатурата, както и други сведения, необходими на потребителя при работата с компютъра.

Книгата е предназначена за широк кръг от специалисти в различни области, които използват персонални компютри и имат известен опит в програмирането на езика БЕЙСИК.

ПЕРСОНАЛЕН КОМПЮТЪР ПРАВЕЦ–8А
АРХИТЕКТУРА, ОРГАНИЗАЦИЯ, ПРОГРАМНО ОСИГУРЯВАНЕ

Автор инж. **Петър Димитров Петров**
Рецензенти: **Костадин Неделчев Коручев**
инж. **Веселин Владимиров Бончев**

Българска
Първо издание

Код 03 $\frac{9533123413}{3205–16–89}$

Изд.№ 16370

Научен редактор инж. **Любов Алексиева**
Художник **Любомир Михайлов**
Художествен редактор **Мария Димитрова**
Технически редактор **Дарина Асенова**
Коректор **Росица Стоянова**

Дадена за набор м. юни 1988 г.
Подписана за печат м. февруари 1989 г.
Излязла от печат м. март 1989 г.
Формат 60•90/16
Печ. коли 15,50
Изд. коли 15,50
УИК 16,30
Тираж 13400 + 111
Цена 2,20 лв.

Държавно издателство „**Техника**", бул. Руски 6, София
Държавна печатница „**Г. Димитров**" — Ямбол

681

Въведение

Развитието на елементната и технологичната база за производство на персонални компютри в Комбината по микропроцесорна техника (КМПТ) в гр. Правец доведе до появата на нов член на фамилията осемразредни персонални компютри Правец. След Правец–82 (1983 г.) и Правец–8М (1986 г.) в края на 1987 г. започна редовното производство и на персоналния компютър Правец–8А. И докато разликата между Правец–82 и Правец–8М бе главно количествена и се изразяваше в интегрирането на модули 16К DRAM и CP/M към системната платка на компютъра, Правец–8А се различава и количествено, и качествено от предшествениците си.

Правец–8А е универсален осемразреден персонален компютър, предназначен за индивидуално ползване от потребители с различна професия и квалификация. Той не изисква специални климатични условия за работата си, захранва се с еднофазно напрежение 220 V, а размерите му позволяват да бъде монтиран на всяко работно място. В основата на новата архитектура на компютъра са микропроцесорът СМ630 (аналог на 6502) и комплектът специализирани интегрални схеми с висока степен на интеграция СМ631, СМ632 и СМ633. Тази архитектура позволява разширяването на оперативната памет да става на страници от 64 Кбайта. Пряко следствие от това са новите текстови и графични режими с двойно по–голяма разделителна способност по хоризонталата: ТЕКСТ80 – текстово изображение с формат 80 колони на 24 реда, ГР80 – графика с двойна малка разделителна способност с формат 80 x 48 цветни блокчета в 16 цвята, и ГР560 – графика с двойна голяма разделителна способност с формат 560 x 192 точки в 16 цвята. Новост са и апаратните средства за поддържане на осембитова кодова таблица. Заедно с измененията в резидентното и системното програмно осигуряване те създават условия за поддържане на набор от знаци, включващ главни и малки букви на кирилица и латиница, цифрите от 0 до 9 и специални и управляващи символи. Разширени са и възможностите на резидентното програмно осигуряване. В управляващата програма МОНИТОР са добавени програмата МИНИАСЕМБЛЕР и функции за търсене на байт или последователност от байтове в паметта и за преобразуване на числа от десетичен в шестнайсетичен код и от шестнайсетичен в десетичен. Добавена е и подпрограма за обслужване на екрана в режим ТЕКСТ80. В резидентния интерпретатор на

БЕЙСИК са включени нови команди. Създадени са условия за работа с шестнайсетични константи. В конструктивно отношение интерес представляват комбинацията от модули, с които оперативната памет на компютъра може да се разширява до 1080 Кбайта, и новата клавиатура, предлагаща три различни стандарта на подреждане на кирилицата: два машинописни (съгласно БДС и ГОСТ) и познатото ни от предишните модели микрокомпютърно или подреждане QWERTY. Независимо от качествено новите възможности на Правец–8А той запазва пълна програмна и апаратна съвместимост с Правец–82 и Правец–8М. Това позволява част от допълнителните модули и почти всички съществуващи програми да се използват и с новия компютър.

В настоящата книга са описани апаратните и програмните възможности на персоналния компютър Правец–8А. Тя е предназначена за програмисти, конструктори на допълнителни модули за управление на входно–изходни устройства и за всички потребители, които искат да използват пълните възможности на новия компютър.

Глава 1. Основни сведения

1.1. Технически характеристики

В основата на разширените възможности на персоналния компютър Правец–8А в сравнение с Правец–82 и Правец–8М стоят новата организация на паметта, новите програмноуправляеми ключове и средствата за въвеждане на осми значещ бит от клавиатурата. Тези новости са резултат от измененията в архитектурата на компютъра и се отразяват на всичките му основни характеристики (вж. табл. 1.1).

От техническите характеристики на персоналния компютър Правец–8А на първо място трябва да се постави по–голямата оперативна памет. Макар че на системната (основната) платка оперативната памет е само 64 Кбайта – толкова, колкото има и Правец–8М, в архитектурата на компютъра са заложени апаратни средства за поддържане на двойно по–голяма оперативна памет. Това, съчетано със страничната организация на допълнителната памет, създава условия за още по–голямо увеличение на паметта. Пример за подобно решение е комбинацията от модули 128/512K RAM и RAM PLUS, които по–нататък в изложението ще означаваме с общото име МЕГАРАМ. Чрез използване на различни комбинации на памети с организация 64K x 1 или 256K x 1 (4164 или 41256) тези модули позволяват общата оперативна памет на компютъра да се увеличи до 1080 Кбайта.

При новата архитектура на компютъра Правец–8А се поддържа и по–голяма постоянна памет (вж. табл. 1.1), което от своя страна е позволило в резидентното програмно осигуряване да се направят съществени изменения. Програмата МОНИТОР е преработена изцяло. Сега тя заема не само областта $F800 – $FFFF, но и част от областта $C100 – $CFFF. Част от подпрограмите са преработени, добавени са и много нови подпрограми. Измененията са насочени предимно към осигуряване на програмно поддържане на обмена с клавиатурата – при работа с осембитова кодова таблица, и с екрана – при работа в режим ТЕКСТ80, и към облекчаване на прехвърлянето на блокове данни от основната в допълнителната памет и обратно. Създадени са условия и за предаване на управлението между програмни сегменти, разположени в основната и допълнителната памет.

Резидентният интерпретатор на БЕЙСИК също е претърпял

много изменения, най-съществените от които е възможността за работа с осембитова кодова таблица. Добавени са командите SETMOD, LINE INPUT и CONV. Създадени са условия за работа с шестнайсетични константи, за директно преобразуване на десетични числа в шестнайсетични и обратно и за симулиране на оператор PRINT USING с числени изрази. Не на последно място трябва да се отбележи, че при работа с осембитов код новата версия на резидентния БЕЙСИК приема командите и тогава, когато те са написани с малки букви.

Апаратните и програмните средства, осигуряващи въвеждането и обработването на осми значещ бит от клавиатурата, са разкрили пътя за работа с кодова таблица с 256 символа. Това е позволило да се включат две пълни азбуки. В конкретната реализация са заложени кирилицата и латиницата, но по принцип е възможно (например за нуждите на образованието) на мястото на малките букви от латиницата да се поставят буквите от гръцката азбука.

Таблица 1.1

Технически характеристики на компютрите от фамилията Правец

Параметър	Правец-82	Правец-8М	Правец-8А
1	2	3	4
1.Микропроцесор:			
– основен	6502	6502	СМ630
– копроцесор	Z80[1]	Z80	Z80[1]
2.Оперативна памет:			
– на системната платка	48K	64K	64K
– разширение			
на страници по 16 Кбайта	да	да	да
на страници по 64 Кбайта	не	не	да
– реализирани модули за разширение на паметта	128K	128K	1024K
3.Постоянна памет	12K	12K	16K
4.Режими на изображение:			
– ТЕКСТ40: 40x24 символа	да	да	да
– ТЕКСТ80: 80x24 символа	не	не	да[2]
– ГР40: 40x48 блокчета в 16 цвята	да	да	да
– ГР80: 80x48 блокчета в 16 цвята	не	не	да[2,3]
– ГР280: 280x192 точки в 8 цвята	да	да	да
– ГР560: 560x192 точки в 16 цвята	не	не	да[2,3]
– СМ40/40	да	да	да[4]
– СМ280/40	да	да	да[4]

1	2	3	4
– СМ80/80	не	не	да[2,3,4]
– СМ560/80	не	не	да[2,3,4]
5.Режими на работа:			
– 7–битова кодова таблица, главни букви кирилица и латиница	да	да	да
– 8–битова кодова таблица, малки и главни букви кирилица и латиница	не	не	да
6.Клавиатура:			
– подреждане QWERTY	да	да	да
– подреждане по БДС	не	не	да
– подреждане по ГОСТ	не	не	да
– функционални клавиши	не	не	2
7.Резидентно програмно осигуряване:			
– програма МОНИТОР	стандартна	стандартна	разширена[5]
– програма МИНИАСЕМБЛЕР	не	не	да
-интерпретатор на БЕЙСИК	стандартен	стандартен	разширен[5]
8.Операционни системи:			
– ДОС 3.3	да	да	да
– ПроДОС	да	да	да
– CP/M	да[1]	да	да[1]
– USCD	да	да	да
9.Работа с езици от високо ниво	да	да	да
10.Работа с касетофон	да	да	не

[1] С допълнителен модул CP/M.

[2] С допълнителна памет.

[3] Не се поддържа от резидентното програмно осигуряване. Архитектурата на компютъра позволява работа в такъв режим, но за ефективното му използване са необходими специализирани програми.

[4] Имената на режимите на смесено изображение се образуват като комбинация от разделителната способност на графичното и формата на текстовото изображение.

[5] Терминът "разширен" означава, че съответната версия на програмата МОНИТОР, респ. на интерпретатора на БЕЙСИК, има допълнителни команди. Той не бива да се смесва с вида на интерпретатора — ЦЕЛОЧИСЛЕН БЕЙСИК (целочислена аритметика) или РАЗШИРЕН БЕЙСИК (аритметика с плаваща запетая и графика с голяма разделителна способност).

1.2. Основна конфигурация

Персоналният компютър Правец–8А е едноплатков компютър. Той използва голяма част от конструктивните решения, утвърдени в процеса на производството на персоналния компютър Правец–8М.

Изцяло са заимствани кутията, захранващият блок и голяма част от механичните детайли. Подобно на останалите компютри от фамилията Правец и той е оформен като самостоятелно устройство, наречено системно устройство, което включва системна платка, високоговорител, клавиатура и захранващ блок. Към него се включва монохроматичен видеомонитор. Те заедно образуват т.нар. минимална конфигурация. С тази конфигурация обаче не могат да се реализират голяма част от заложените в компютъра възможности. Ето защо в основната конфигурация на компютъра (тази, която се произвежда в КМПТ в гр. Правец, и тази, с която се отъждествява Правец–8А в по–голямата част от настоящото изложение) е добавен модул МЕГАРАМ с минимум 128 Кбайта допълнителна оперативна памет и контролер с две ЗУГМД (минифлопидискови устройства). Така конфигуриран, персоналният компютър Правец–8А позволява:

– работа с програмите МОНИТОР и МИНИАСЕМБЛЕР за създаване и изпълнение на програми на машинния език на микропроцесора СМ630;

– създаване, редактиране и изпълнение на програми на БЕЙСИК;

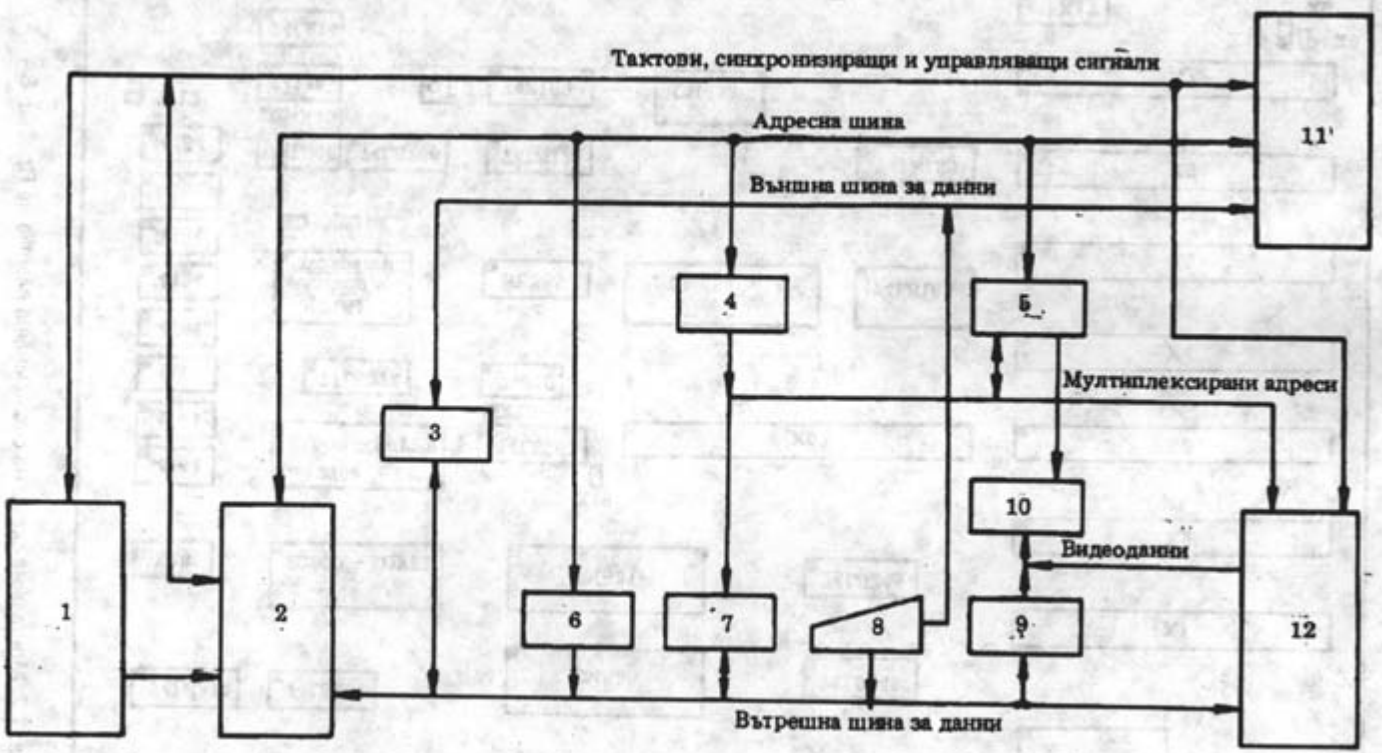
– зареждане, архивиране (съхранение) и изпълнение на програми от дисковите устройства под управление на някоя от операционните системи.

Устройството на компютъра от гледна точка на апаратната част е разгледано по–подробно в гл.7, а тук са дадени само тези сведения, които са необходими за първоначалното запознаване с него. На фиг. 1.1 е показана опростена функционална схема на системната платка. Вижда се, че това всъщност е функционално завършен компютър с микропроцесор, постоянна и оперативна памет, схеми за поддържане на обмена с входно (клавиатура) и изходно устройство (екран), съединители за допълнителни устройства и т.н.

На фиг. 1.2 е показана монтажната схема на системната платка. На тази фигура се вижда разположението и номерирането на съединителите за включване на допълнителни модули (X0 – X7), на клавиатура (X13), на видеомонитор (X8 – X10 и X14, X15) и на високоговорител (X12).

Фиг. 1.1. Опростена функционална схема на системната платка на Правец–8А

1 – схема за генериране на тактови и синхронизиращи сигнали; **2** – микропроцесор; **3** – буфер за данни; **4** – схема за управление на паметта; **5** – схема за управление на вградените входно–изходни устройства; **6** – постоянна памет; **7** – оперативна памет; **8** – схеми за управление на клавиатурата; **9** – буфер за видеоданни; **10** – схема за генериране на видеосигнала; **11** – съединители за интерфейсни модули (X1 – X7); **12** – допълнителен съединител за разширяване на паметта (X0)


Тактови, синхронизиращи и управляващи сигнали
Адресна шина
Външна шина за данни
Мултиплексирани адреси
Видеоданни
Вътрешна шина за данни
1
2
3
4
5
6
7
8
9
10
11
12

Фиг. 1.2. Монтажна схема на системната платка на Правец–8А

Правец–8А се произвежда с две различни клавиатури. И двете в зависимост от режима на работа на компютъра работят със 7– или 8–битова кодова таблица. В първия тип клавиатура се използват клавиши с ридконтакти. Тя е с подреждане QWERTY за латиницата (фиг. 1.3) и с микрокомпютърно подреждане за кирилицата (фиг. 1.3) и по механична конструкция, подреждане и схемно решение не се различава съществено от клавиатурата на Правец–8М.

Във втория тип клавиатура се използват бутони с провеждаща гума. Тази клавиатура е с по–добър дизайн и има друго схемно решение. Подреждането за латиницата е QWERTY, на специалните символи – еднакво с възприетото в Правец–16, а за кирилицата в нея са заложени три различни подреждания: едно основно – съгласно БДС за пишещи машини (фиг. 1.4), и две спомагателни – микрокомпютърно и съгласно ГОСТ за пишещи машини.

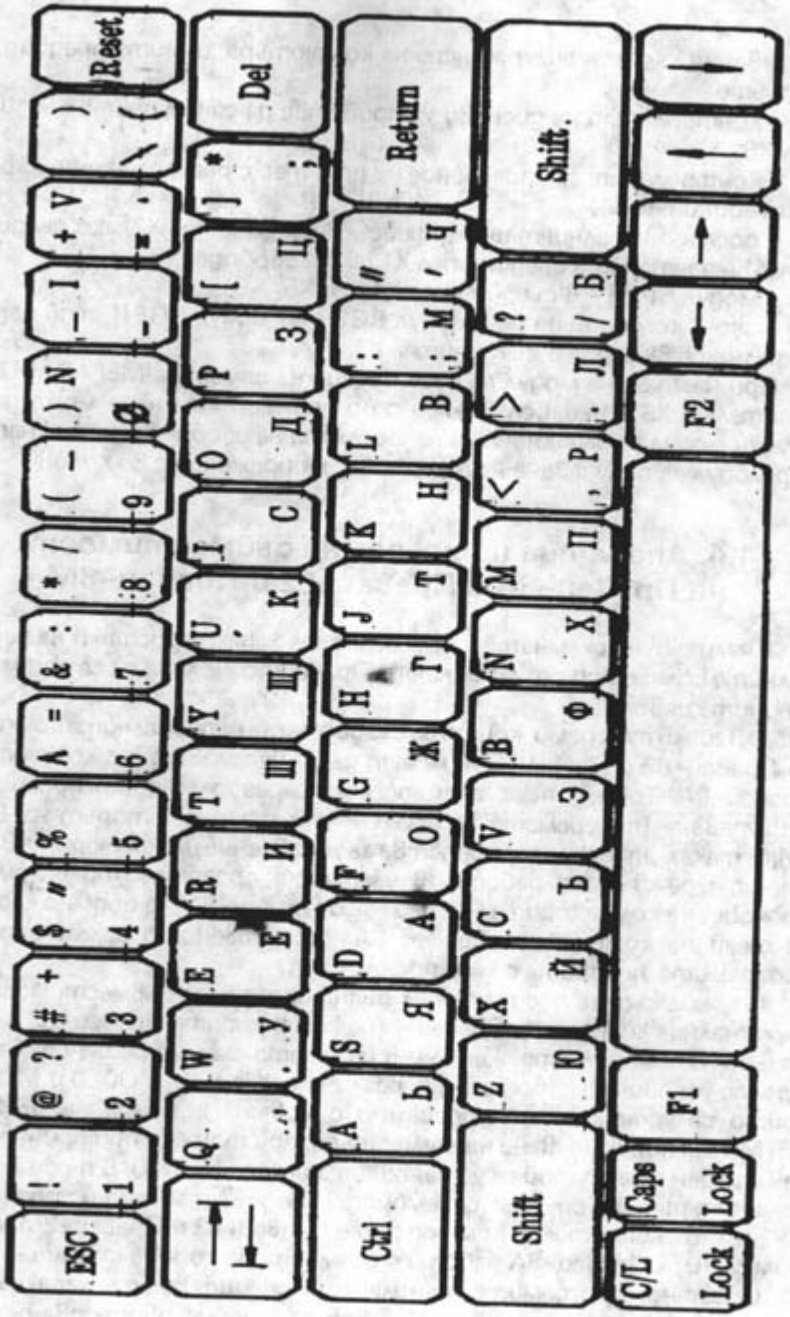
Видеомониторът е монохроматичен. Свързва се с коаксиален кабел към съединителя, монтиран на задната плоча на компютъра. Чрез допълнителен модул (модул RGB или модул RGB][) към компютъра може да се свърже и RGB монитор за цветно изображение.

1.3. Включване на допълнителни модули

Персоналният компютър Правец–8А е отворена система. Към него могат да се добавят различни устройства, така че да се получи конфигурация, удовлетворяваща нуждите на конкретно приложение. За тази цел са предвидени осемте съединителя, в които могат да се включват модули с различно предназначение. Съединителите са номерирани от X0 до X7 (вж. фиг. 1.2) и до голяма степен са равнозначни. Изключение прави съединителят X0, който няма аналог в Правец–82 и Правец–8М. На него са изведени вътрешната шина за данни, шината за видеоданни, мултиплексираните адреси и редица други сигнали, управляващи вътрешния обмен в компютъра. Предназначението на съединителя X0 е тясно свързано с новите архитектурни решения в Правец–8А. Образно казано, той представлява прозорец към шините за вътрешен обмен в компютъра и на този етап се използва изключително от модулите за разширение на паметта, какъвто е модулът МЕГАРАМ. Възможно е да се конструират и специализирани комбинирани модули за разширение на паметта и за управление на RGB монитор, които да се включват чрез съединителя X0.

Съединителите от X1 до X6 и отчасти X7 са равностойни дотолкова, доколкото на техните изводи има едни и същи сигнали. Независимо от това при работа с някои приложни програми и операционни системи (USCD, CP/M) се предявяват строги изисквания към позициите на включване на допълнителните модули. Ако в конкретната програма, която използвате или създавате, няма специални

Фиг. 1.3. Клавиатура с подреждане QWERTY на кирилицата



Фиг. 1.4. Клавиатура с машинописно подреждане на кирилицата

изисквания към конфигурацията на компютъра, имайте предвид, че е прието:

– контролерът на дискови устройства да се включва в съединител X6;

– контролерът за управление на принтер с паралелен интерфейс – в съединител X1;

– последователният интерфейс – в съединител X2, ако съединител X1 е зает, или в съединител X1, ако е свободен;

– модул CP/M – в съединител X4 или X5;

– ако в компютъра има модул RGB или модул RGB][, той задължително се включва в съединител X7.

При наличие на модул за разширение на паметта МЕГАРАМ съединителят X3 е неизползваем, но по принцип към него може да се включи модул за разширяване на формата на изображението, например модул за получаване на текстово изображение с 80 колони.

1.4. Апаратна и програмна съвместимост на Правец–8А с Правец–82 и Правец–8М

Анализът на схемните и програмните характеристики на персоналните компютри от фамилията Правец позволява да се направят следните изводи:

1. Всички програми, които са създадени за персоналните компютри Правец–82 и Правец–8М, могат да се използват и с компютъра Правец–8А. Това е така, защото както резидентното програмно осигуряване (програмата МОНИТОР и интерпретаторът на БЕЙСИК), така и апаратните средства за управление на компютъра в различните режими на работа (в частност програмноуправляемите ключове) на компютрите Правец–82 и Правец–8М са подмножество на тези на компютъра Правец–8А. Единственото изключение са програмите за работа с касетофон.

Стремежът за постигане на пълна програмна съвместимост на персоналния компютър Правец–8А с компютрите Правец–82 и Правец–8М е намерил отражение и в неговото схемно решение. Създадени са условия за работа в два различни режима – MODE0 и MODE1, които се установяват програмно с новата команда на БЕЙСИК SETMOD. При включване на компютъра, при топъл и при принудителен студен старт той се установява в режим MODE0. В този режим компютърът работи със седембитова кодова таблица и програмно е напълно съвместим с компютрите Правец–82 и Правец–8М. В режим MODE1 Правец–8А работи с осембитова кодова таблица.

Обратно, програмите, използващи специфичните характеристики на компютъра Правец–8А в кой да е от неговите два режима на работа, няма да могат да се изпълняват на компютрите Правец–82 и Правец–8М. Това са на първо място възможностите, които

предлагат допълнителната оперативна памет, допълнителните програмноуправляеми ключове, новите команди на БЕЙСИК, новите подпрограми на програмата МОНИТОР и т.н.

2. Почти всички съществуващи модули, предназначени за работа с компютрите Правец–82 и Правец–8М, могат да се използват и с компютъра Правец–8А. Тук изключенията са значително повече. Те се дължат не толкова на архитектурните, колкото на конструктивните различия на компютрите. По една или друга причина с Правец–8А не могат да се използват модулите 16K DRAM, 128K RAM и VIDEOCARD 80. Използването на модула GPIB не е целесъобразно, защото той се включва в съединител X3.

Модулите, разработени за компютъра Правец–8А, с изключение на тези, предназначени за включване в съединител X0, могат да се използват и с компютрите Правец–82 и Правец–8М.

Глава 2. Вградени входно–изходни устройства

Както при персоналните компютри Правец–82 и Правец–8М, така и при персоналния компютър Правец–8А входно–изходните устройства се делят условно на вградени и външни. Към вградените входно–изходни устройства спадат клавиатурата, схемите за генериране на видеоизображение, високоговорителят, аналоговите и цифровите входове и цифровите изходи на компютъра.

От гледна точка на програмиста програмното управление на вградените входно–изходни устройства на най–ниско ниво (машинен език или АСЕМБЛЕР) става, като програмата извършва обръщение към определени клетки от паметта.

В тази глава са описани функциите и възможностите на вградените входно–изходни устройства. Разгледан е начинът, по който те се поддържат от програмното осигуряване. Дадени са адресите на клетките, имащи отношение към тяхното управление, включително и на програмноуправляемите ключове.

2.1. Клавиатура

Основното входно устройство на персоналния компютър Правец–8А е клавиатурата. Както вече споменахме, компютърът се произвежда с две различни клавиатури (вж. фиг.1.3 и 1.4). Те се различават по подреждане, по конструкция и като схемна реализация, но функционално са абсолютно еднозначни. Единствената разлика, която е от значение за потребителя, е подреждането на клавишите за двете азбуки, респ. кодовете, които се генерират от различните клавиши. Тези кодове са различни при работа със седембитова и осембитова кодова таблица (вж. приложение 8). Клавиатурата с микрокомпютърно подреждане за кирилицата е по–близко до клавиатурите на компютрите Правец–82 и Правец–8М. По-съществените изменения са следните:

1. **Кодови таблици.** В зависимост от режима на работа на компютъра клавиатурата генерира седембитов или осембитов код (вж. табл. 2.1 и 2.2). Имената на управляващите символи, използвани в тези таблици, са възприети от големите изчислителни машини (вж. табл. 2.3), където те са тясно свързани с обмена на информация с входно–изходните устройства, в частност с телетайпите. Поради това тези имена не винаги отговарят напълно на смисъла, влаган в тях при използването им в персоналните компютри.

2. **Брой на клавишите.** Наличието на 64, респ. на 63 клавиша дава възможност за генериране на всичките 128 кода от седембитовата кодова таблица (вж. табл. 2.1). Освен това са добавени шест нови клавиша, управляващи движението на курсора:

* СТРЕЛКА НАЛЯВО (←). Този клавиш генерира код $88. Използва се за придвижване на курсора наляво.

* СТРЕЛКА НАДЯСНО (→). Генерира код $95. Използва се за придвижване на курсора надясно.

* СТРЕЛКА НАГОРЕ (↑). При натискането му се генерира код $8B. Използва се за придвижване на курсора нагоре.

* СТРЕЛКА НАДОЛУ (↓). Генерира код $8A. Използва се за придвижване на курсора надолу.

* Табулация (TAB). Този клавиш генерира код $89. Използва се подобно на табулатора в пишещите машини.

* Изтриване (DELETE или DEL). Генерира код $FF. Използва се за изтриване на последния въведен символ.

Кодовете, генерирани от клавиши СТРЕЛКА НАЛЯВО, СТРЕЛКА НАДЯСНО и СТРЕЛКА НАДОЛУ, се възприемат от резидентното програмно осигуряване и от почти всички приложни и системни програми.

Кодовете, генерирани от клавиши СТРЕЛКА НАГОРЕ, TAB и DEL, не се възприемат от резидентното програмно осигуряване, а само от някои приложни програми.

3. **Функционални клавиши** F1 и F2. Тези клавиши са свързани към цифровите входове на компютъра и тяхното действие няма пряко отношение към клавиатурата. Наличието на функционални клавиши обаче облекчава програмирането, като разтоварва използването на управляващите символи от несвойствени функции (организиране на преходи към различни сегменти на програмата и други подобни), така че те да могат да се използват изключително за управление на входно–изходните устройства.

4. **Автоматично повторение.** И в двата варианта клавиатурата е с автоматично повторение. Кодът, генериран от определен клавиш, се въвежда многократно, ако клавишът се задържи натиснат повече от 0,5 s. Тази функция е от съществено значение при програмите за текстообработка.

5. **Нулиране.** Клавиш RESET (или RST) от клавиатурата на персоналния компютър Правец–8А действа само в комбинация с клавиш CONTROL (CTRL).

* Инициализирането на компютъра или т. нар. топъл старт става с едновременното натискане на клавиши CONTROL и RESET.

* Рестартирането на компютъра или т. нар. принудителен студен старт, който е еквивалентен на изключване и повторно включване на компютъра, става с едновременното натискане на клавиши CONTROL, RESET и F1.

* Прекъсването на изпълнението на която и да е програма и безусловното преминаване под управлението на програмата МОНИТОР (т. нар. мониторен режим) става с едновременното натискане на клавиши CONTROL, RESET и F2.

6. **Клавишите за избор** на режим на работа на клавиатурата имат различно действие и са означени по различен начин в двата типа клавиатура:

* В клавиатурата с микрокомпютърно подреждане преминаването от кирилица в латиница и обратно при работа със седембитова кодова таблица става с клавиш LOCK. При работа с осембитова кодова таблица този клавиш служи за преминаване от режим на малки в режим на главни букви, а преминаването от кирилица в латиница става с клавиш КИРИЛИЦА.

* В клавиатурата с подреждане по БДС изборът на азбука (кирилица или латиница) и изборът на регистър (малки или главни букви) се извършва с клавиши C/L LOCK и CAPS LOCK съответно. Действието на тези клавиши не зависи от избора на кодова таблица.

Таблица 2.1

Седембитова кодова таблица

		$80 128	$90 144	$A0	$B0 160	176	$C0	$D0 192	208	$E0	$F0 224	240
$0	(0)	NUL	DLE		0		@	P		Ю	П	
$1	(1)	SOH	DC1	!	1		A	Q		А	Я	
$2	(2)	STX	DC2	"	2		B	R		Б	Р	
$3	(3)	ETX	DC3	#	3		C	S		Ц	С	
$4	(4)	EOT	DC4	$	4		D	T		Д	Т	
$5	(5)	ENQ	NAK	%	5		E	U		Е	У	
$6	(6)	ACK	SYN	&	6		F	V		Ф	Ж	
$7	(7)	BEL	ETB	'	7		G	W		Г	В	
$8	(8)	BS	CAN	(	8		H	X		Х	Ь	
$9	(9)	HT	EM	)	9		I	Y		И	Ъ	
$A	(10)	LF	SUB	*	:		J	Z		Й	З	
$B	(11)	VT	ESC	+	;		K	[		К	Ш	
$C	(12)	FF	FS	,	<		L	\		Л	Э	
$D	(13)	CR	GS	-	=		M	]		М	Щ	
$E	(14)	SO	RS	.	>		N	^		Н	Ч	
$F	(15)	SI	US	/	?		O	_		О	DEL	

Таблица 2.2

Осембитова кодова таблица

	$00 0	$10 16	$20 32	$30 48	$40 64	$50 80	$60 96	$70 112	$80 128	$90 144	$A0 160	$B0 176	$C0 192	$D0 208	$E0 224	$F0 240
$0 (0)	...	...	...	...	'	p	ю	n	NUL	DLE		0	@	P	Ю	П
$1 (1)	...	...	...	...	a	q	а	я	SOH	DC1	!	1	A	Q	А	Я
$2 (2)	...	...	...	...	b	r	б	р	STX	DC2	"	2	B	R	Б	Р
$3 (3)	...	...	...	...	c	s	ц	с	ETX	DC3	#	3	C	S	Ц	С
$4 (4)	...	...	...	...	d	t	g	т	EOT	DC4	$	4	D	T	Д	Т
$5 (5)	...	...	...	...	e	u	е	у	ENQ	NAK	%	5	E	U	Е	У
$6 (6)	...	...	...	...	f	v	ф	ж	ACK	SYN	&	6	F	V	Ф	Ж
$7 (7)	...	...	...	...	g	w	г	в	BEL	ETB	'	7	G	W	Г	В
$8 (8)	...	...	...	...	h	x	х	ь	BS	CAN	(	8	H	X	Х	Ь
$9 (9)	...	...	...	...	i	y	u	ъ	HT	EM	)	9	I	Y	И	Ъ
$A (10)	...	...	...	...	j	z	й	э	LF	SUB	*	:	J	Z	Й	З
$B (11)	...	...	...	...	k	{	к	ш	VT	ESC	+	;	K	[	К	Ш
$C (12)	...	...	...	...	l	|	л	э	FF	FS	,	<	L	\	Л	Э
$D (13)	...	...	...	...	m	}	м	щ	CR	GS	-	=	M	]	М	Щ
$E (14)	...	...	...	...	n	~	н	ч	SO	RS	.	>	N	^	Н	Ч
$F (15)	...	...	...	...	o	...	о	...	SI	US	/	?	O	_	О	DEL

Забележка. Кодовете, означени с три точки (...), не се генерират от клавиатурата.

Таблица 2.3

Управляващи кодове

Код	Наименование	Значение
1	2	3
$80	NUL — NULl (Blank)	Празен символ
$81	SOH — Start Of Header	Начало на идентификатор
$82	STX — Start Of Text	Начало на текст
$83	ETX — End Of Text	Край на текста
$84	EOT — End Of Transmition	Край на предаването
$85	ENQ — ENQuiry	Запитване
$86	ACK — ACKnowledge	Потвърждение
$87	BEL — BELl	Звуков сигнал
$88	BS — Back Space	Интервал назад

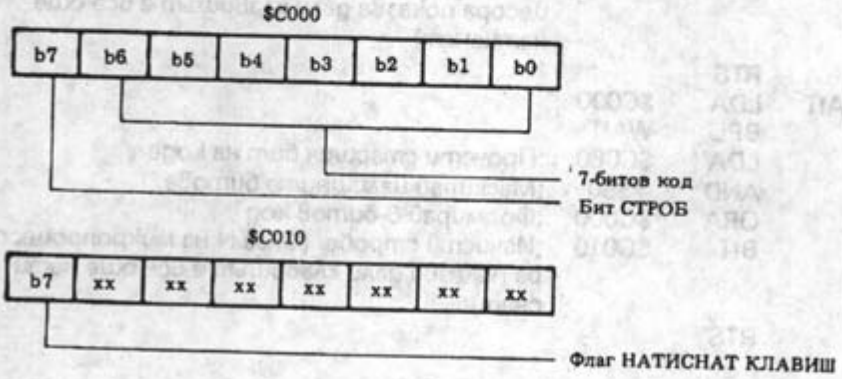
1	2	3
$89	HT — Horizontal Tab	Хоризонтална табулация
$8A	LF — Line Feed	Нов ред
$8B	VT — Vertical Tab	Вертикална табулация
$8C	FF — Form Feed	Нова страница
$8D	CR — Carriage Return	Връщане в началото на реда
$8E	SO — Shift Out	Долен регистър
$8F	SI — Shift In	Горен регистър
$90	DLE — Data Link Escape	Освобождаване на линията за обмен
$91	DC1 — Device Control 1	Първи код за управление на устройството
$92	DC2 — Device Control 2	Втори код за управление на устройството
$93	DC3 — Device Control 3	Трети код за управление на устройството
$94	DC4 — Device Control 4	Четвърти код за управление на устройството
$95	NAK — Negative AcKnowledge	Негативно потвърждение
$96	SYN — SYNchronisation	Синхронизация
$97	ETB — End of Text Block	Край на блока текст
$98	CAN — CANcel	Анулиране
$99	EM — End of Medium	Край на носителя
$9A	SUB — SUBstitute	Заместване
$9B	ESC — ESCape	Освобождаване
$9C	FS — File Separator	Разделител на файлове
$9D	GS — Group Separator	Разделител на групи
$9E	RS — Record Separator	Разделител на записи
$9F	US — Unit Separator	Разделител на томове
$FF	DEL — DELete	Изтриване

Както вече споменахме, при натискане на един и същи клавиш клавиатурата генерира различни кодове в зависимост от това, дали се работи със седембитов или осембитов код (вж. приложение 8). С какъв код ще се работи, се определя от състоянието на флаг 8-БИТОВ КОД (бит b6 на адрес $C060). Този флаг се установява или нулира с най–младшия бит на байта данни при команда за запис на адрес $C060. Когато флагът е установен, клавиатурата изпраща осембитов код. Когато флагът е нулиран, клавиатурата изпраща седембитов код, като старшият бит се смята равен на единица.

В режим MODE0 (седембитова кодова таблица) програмите четат кода, генериран от клавиатурата, от адрес $C000 (вж.фиг. 2.1). Младшите седем бита на байта, прочетен от този адрес, представ-

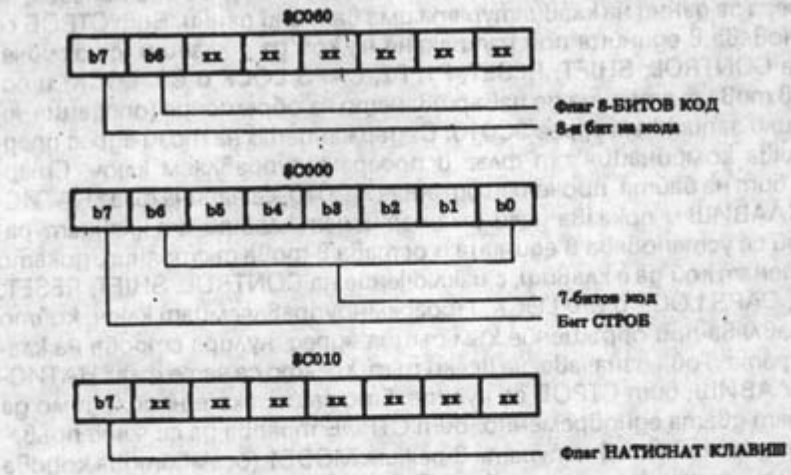
ляват кода, генериран от последния натиснат клавиш. Старшият бит на този байт се нарича бит СТРОБ. Той не е значещ (стойността му не зависи от това кой клавиш е натиснат), а само показва, че в буфера за данни на клавиатурата има валидни данни. Бит СТРОБ се установява в единица при натискане на кой да е клавиш (с изключение на CONTROL, SHIFT, RESET, F1, F2, CAPS LOCK и C/L LOCK) и остава в това състояние до извършването на обръщение (операция четене или запис) към адрес $C010. Съдържанието на този адрес представлява комбинация от флаг и програмноуправляем ключ. Старшият бит на байта, прочетен от адрес $C010, се нарича флаг НАТИСНАТ КЛАВИШ и показва дали има натиснат клавиш от клавиатурата. Той се установява в единица и остава в това състояние, докато е натиснат кой да е клавиш, с изключение на CONTROL, SHIFT, RESET, F1, F2, CAPS LOCK и C/L LOCK. Програмноуправляемият ключ, който се управлява при обръщение към същия адрес, нулира строба на клавиатурата. Това означава, че всеки път, когато се чете флаг НАТИСНАТ КЛАВИШ, бит СТРОБ се нулира. Ето защо, ако е необходимо да се четат двата едновременно, бит СТРОБ трябва да се чете пръв.

Четенето на клавиатурата в режим MODE1 (осембитова кодова таблица) е значително по-сложно. Това се дължи на факта, че осмият бит на байта (бит СТРОБ), прочетен от адрес $C000, не носи информация за кода, изпратен от клавиатурата. Ето защо за реализирането на осембитова кодова таблица е необходимо в кода, прочетен от адрес $C000, осмият бит да бъде заместен с осмия бит на кода, ге-



Фиг. 2.1. Четене на седембитов код от клавиатурата

нериран от самата клавиатура. Той постъпва в компютъра като старши бит на байта, прочетен от адрес $C060 (вж. фиг. 2.2).



Фиг. 2.2. Четене на осембитов код от клавиатурата

Сравнете следните два примерни драйвера за четене на клавиатурата:

```
WAIT    LDA     $C000   ;Бит СТРОБ = 1?
        BPL     WAIT    ;Не. Изчакай
        BIT     $C010   ;Да. Изчисти строба. (Флаг N на микропро-
                        цесора показва дали клавишът е все още
                        натиснат)

        RTS
WAIT    LDA     $C000
        BPL     WAIT
        LDA     $C060   ;Прочети старшия бит на кода
        AND     #$80    ;Маскирай излишните битове
        ORA     $C000   ;Формирай 8-битов код
        BIT     $C010   ;Изчисти строба. (Флаг N на микропроцесо-
                        ра показва дали клавишът е все още нати-
                        снат)

        RTS
```

2.2. Режими на видеоизображение

Основното изходно устройство на персоналния компютър Правец–8А е видеомониторът. В основната конфигурация на компютъра се използва монохроматичен монитор с вход за комплексен видео-

сигнал. Няма възможности за директно включване на монитор или телевизор за цветно изображение, както и на телевизор за черно-бяло изображение. Сигналът на стандартния видеоизход обаче съдържа всички компоненти, необходими за получаването на цветно изображение. За целта е необходимо към компютъра да се включи допълнителен модул, с който съществуващият сигнал или да се разложи на три сигнала (R, G и B) за директно подаване към усилвателите за управление на трите лъча на електронолъчевата тръба, или да се прекодира в сигнал по система PAL или SECAM. Така например с използването на модул RGB или RGB][към компютъра може да се свърже RGB монитор.

Компютърът Правец–8А в минимална конфигурация (без допълнителна памет) работи в общите за компютрите от фамилията Правец три режима на изображение:

– **Режим ТЕКСТ40.** Изображението е монохроматично (видеосигналът не съдържа информация за цветност). На екрана се изобразява текстова информация във формат 24 реда по 40 символа на ред.

– **Режим ГР40.** Изображението представлява графика с малка разделителна способност и се състои от 48 реда по 40 елемента, представляващи оцветени блокчета, чиито размери са два пъти по-малки от размера на един символ от режим ТЕКСТ40. Цветовата палитра има 16 цвята.

– **Режим ГР280.** Изображението представлява графика с голяма разделителна способност. То е съставено от 192 реда по 280 точки в 6 различни цвята. Цветовете са позиционно зависими (цветът, в който може да свети определена точка от екрана, зависи от нейната хоризонтална позиция).

Включването на модул с допълнителна памет разширява възможностите на видеоизхода и в основната си конфигурация компютърът поддържа три нови режима на изображение: ТЕКСТ80, ГР80 и ГР560. Вграденото програмно осигуряване позволява използването на режим ТЕКСТ80, а с потребителски програми е възможна и работа в режими ГР80 и ГР560, осигуряващи графично изображение с двойна малка и двойна голяма разделителна способност. Това, че графичните режими с двойна разделителна способност не се поддържат от резидентното програмно осигуряване, означава, че то не съдържа нито команди от високо ниво, нито подпрограми за задаване на цвета, за изчертаване на точка или линия в тези режими. По-съществено е, че са създадени апаратните средства, поддържащи графика с двойна разделителна способност. Останалото е въпрос на създаване на подходящи програми.

– **Режим ТЕКСТ80.** Изображението е монохроматично. На екрана се изобразява текст във формат 24 реда по 80 символа на ред.

Внимание! Програмното осигуряване на режим ТЕКСТ80 е различно от програмното осигуряване на режим ТЕКСТ40. То поддържа два формата: 80 x 24 и 40 x 24, като преминаването от единия в другия

става с последователностите ESC 8 и ESC 4. Независимо от това, кой е текущият формат, когато по-нататък се говори за режим ТЕКСТ80, се подразбира и програмното осигуряване за неговото поддържане.

– **Режим ГР80.** Изображението представлява графика с двойна малка разделителна способност. То е съставено от 48 реда по 80 цветни блокчета. Цветовата палитра е от 16 цвята.

– **Режим ГР560.** Изображението представлява графика с двойна голяма разделителна способност. При работа с монохроматичен монитор се приема, че то е съставено от 192 реда по 560 точки, а при работа с монитор за цветно изображение — от 192 реда по 140 точки в 16 цвята (без ограничение на цвета в зависимост от хоризонталната позиция на точката на екрана).

Графичните режими на изображение имат подклас. Това са т.нар. режими на смесено изображение. Общото за тях е, че ако условно се приравнят към текстовите режими, се оказва, че първите 20 реда са заети от съответната графика, а последните 4 реда — от текст. Форматът на текста може да бъде 40 или 80 колони, като при графичните режими с двойна разделителна способност този формат е винаги 80 колони.

Минималната конфигурация на компютъра поддържа два режима на смесено изображение:

– **Режим СМ40/40.** Изображението е комбинация от графика с малка разделителна способност и текст във формат 40 колони.

– **Режим СМ280/40.** На екрана се изобразяват графика с голяма разделителна способност и текст във формат 40 колони.

В основната конфигурация на компютъра са възможни още четири режима на смесено изображение. Първите два се поддържат и от резидентното програмно осигуряване.

– **Режим СМ40/80.** Изображението е комбинация от графика с малка разделителна способност и текст във формат 80 колони.

– **Режим СМ280/80.** На екрана се изобразяват графика с голяма разделителна способност и текст във формат 80 колони.

– **Режим СМ80/80.** На екрана се получава комбинация от графика с двойна малка разделителна способност и текст в 80 колони.

– **Режим СМ560/80.** Изображението е комбинация от графика с двойна голяма разделителна способност и текст в 80 колони.

Друга характерна особеност на генератора на видеоизображение на компютъра Правец–8А е това, че в двата текстови режима ТЕКСТ40 и ТЕКСТ80 и в съответните смесени режими на екрана се изобразяват две различни множества от символи в зависимост от това, дали се работи със седембитова или осембитова кодова таблица.

2.2.1. Изобразяване на текст

Двата текстови режима ТЕКСТ40 и ТЕКСТ80 се поддържат от различни подпрограми в програмата МОНИТОР. Двата режима се различават по формата на текстовото изображение, но матрицата (броят на точките по хоризонталата и по вертикалата), с която се изобразяват символите, не зависи от режима и от формата на изображението. Всеки символ се изобразява в рамките на поле с широчина 7 и височина 8 точки. С малки изключения самите символи са широки 5 и високи 7 точки, така че на екрана между всеки два съседни символа остава разстояние от две точки, а между два съседни реда — една точка.

Както при Правец–82 и Правец–8М, така и при Правец–8А текстовото изображение е винаги монохроматично и има три формата: НОРМАЛНО, ИНВЕРСНО и МИГАЩО ВИДЕО. При работа с БЕЙСИК тези формати се установяват с команди NORMAL, INVERSE и FLASH. За установяването на формат МИГАЩО ВИДЕО в програмата МОНИТОР няма команда, а формати НОРМАЛНО и ИНВЕРСНО ВИДЕО се установяват с команди N и I. Във формат НОРМАЛНО ВИДЕО символите са бели на черен фон, в ИНВЕРСНО ВИДЕО — черни на бял фон, а в МИГАЩО ВИДЕО алтернативно се редуват бели символи на черен фон с черни символи на бял фон.

И в двата текстови режима могат да се използват две различни символни множества. Те са записани в знаковия генератор на компютъра, като работното множество се избира с програмноуправляем ключ.

Първото символно множество (вж. табл. 2.4) се нарича стандартно или първично и по принцип не се различава от символното множество на персоналните компютри Правец–82 и Правец–8М. То осигурява съвместимостта на компютрите от фамилията и се използва само при работа със седембитовата кодова таблица (режим MODE0).

Второто символно множество се нарича алтернативно. То съдържа графичните изображения на главните букви от кирилицата и латиницата, на цифрите от 0 до 9 и специалните символи във формати НОРМАЛНО и ИНВЕРСНО ВИДЕО и на малките букви от кирилицата и латиницата във формат НОРМАЛНО ВИДЕО (вж. табл. 2.5). Предназначено е за работа с осембитовата кодова таблица (режим MODE1).

Таблица 2.4

Стандартно символно множество

Стойност	Стандартно символно множество
$00-$1F	Латиница — главни букви във формат ИНВЕРСНО ВИДЕО
$20-$3F	Цифри и специални символи във формат ИНВЕРСНО ВИДЕО
$40-$5F	Латиница — главни букви във формат МИГАЩО ВИДЕО
$60-$7F	Цифри и специални символи във формат МИГАЩО ВИДЕО
$80-$9F	Кирилица — главни букви във формат ИНВЕРСНО ВИДЕО
$A0-$BF	Цифри и специални символи във формат НОРМАЛНО ВИДЕО
$C0-$DF	Латиница — главни букви във формат НОРМАЛНО ВИДЕО
$E0-$FF	Кирилица — главни букви във формат НОРМАЛНО ВИДЕО

Таблица 2.5

Алтернативно символно множество

Стойност	Алтернативно символно множество
$00-$1F	Латиница — главни букви във формат ИНВЕРСНО ВИДЕО
$20-$3F	Цифри и специални символи във формат ИНВЕРСНО ВИДЕО
$40-$5F	Латиница — малки букви във формат НОРМАЛНО ВИДЕО
$60-$7F	Кирилица — малки букви във формат НОРМАЛНО ВИДЕО
$80-$9F	Кирилица — главни букви във формат ИНВЕРСНО ВИДЕО
$A0-$BF	Цифри и специални символи във формат НОРМАЛНО ВИДЕО
$C0-$DF	Латиница — главни букви във формат НОРМАЛНО ВИДЕО
$E0-$FF	Кирилица — главни букви във формат НОРМАЛНО ВИДЕО

Поради разширения символен набор формати ИНВЕРСНО и МИГАЩО ВИДЕО не винаги могат да се използват. Визуална представа за двете символни множества и за форматите на текстовото изображение можете да получите при изпълнение на следните програми.

```
5   Rem Символни множества в Правец-8А
10  Text: Home
20  For I = 0 To 7
30  For J = 0 To 31
40  Poke $400 + $80 * I + J, J + $20 * I
50  Next: Next
60  Setmod 0: Vtab 18: Print "Стандартно символно множество":
    Print: Print "  Натиснете кой да е клавиш";: Get A$: Print
70  Setmod 1: Vtab 18: Print "Алтернативно символно множество":
    Print: Print "  Натиснете кой да е клавиш";: Get A$: Print: Goto 60
```

```
5   Rem Режими на текстовото изображение в Правец-8А
10  Text: Home: Normal
20  Data "Режим: ТЕКСТ40", "Режим: ТЕКСТ80"
21  Data "Режим на работа: MODE0"
22  Data "Режим на работа: MODE1"
23  Data " — формат НОРМАЛНО ВИДЕО"
24  Data " — формат ИНВЕРСНО ВИДЕО"
25  Data " — формат МИГАЩО ВИДЕО"
26  Data "Стандартно символно множество"
27  Data "Алтернативно символно множество"
30  For I = 1 To 2: Read A$(I): Next
40  For I = 1 To 2: Read B$(I): Next
50  For I = 1 To 3: Read C$(I): Next
60  For I = 1 To 2: Read D$(I): Next
65  I = 1: Print  Chr$ (17): Rem Режим ТЕКСТ40
70  For J = 1 To 2: For K = 1 To 3: For L = 1 To 2
80  Setmod 0: Normal: Home: Print A$(I);C$(K): Print B$(J): Print D$(L)
90  Vtab 23: Print "Натиснете кой да е клавиш";: Get AA$: Print:
    Home
100 If J = 2 Then Setmod 1
110 If K = 2 Then Inverse
120 If K = 3 Then Flash
130 If L = 1 Then Poke $C00E,0: Rem Стандартно символно мно-
    жество
140 If L = 2 Then Poke $C00F,0: Rem Алтернативно символно
    множество
150 Print "$20-$3F:";: For M = $20 To $3F: Print Chr$ (M);: Next: Print
160 Print "$40-$5F:";: For M = $40 To $5F: Print Chr$ (M);: Next: Print
170 Print "$60-$7F:";: For M = $60 To $7F: Print Chr$ (M);: Next: Print
180 Print "$C0-$DF:";: For M = $C0 To $DF: Print Chr$ (M);: Next:
    Print
190 Print "$E0-$FF:";: For M = $E0 To $FF: Print Chr$ (M);: Next: Print
195 Vtab 23: Get AA$
200 Next: Next: Next
210 I = I + 1: If I > 2 Then  Normal: End
220 Pr# 3: Rem Режим ТЕКСТ 80
230 Goto 70
```

В осемразредните персонални компютри от фамилията Правец екранната памет е част от паметта на компютъра. Това означава, че в паметта са резервирани определени области, в които се записва информацията, изобразявана на екрана. Така в режим ТЕКСТ40 е възможно организирането на два екрана с текстова информация. Това са т.нар. първа и втора текстова страница. Те са разположени на адреси $400 — $7FF и $800 — $BFF в системната памет на компютъра. Всяка страница заема 1024 последователни адреса, независимо че

изобразяваните символи, респ. необходимите клетки от паметта на компютъра, са само 960 (24 реда по 40 символа). Съответствието между адреса, в който е записан определен символ, и позицията му на екрана е показано в табл. 2.6. Обърнете внимание, че:

– броенето на редовете и колоните на екрана започва от нула;

– в рамките на един ред от екрана съседните символи са записани на съседни адреси в паметта; така например, ако се работи в първа текстова страница, адресът на символа от 15 ред, 16 колона се получава, като към началния адрес за този ред ($7A8) се прибави номерът на колоната ($10):

$$\$7A8 + \$10 = \$7B8$$

Таблица 2.6

Съответствие между адреса на символа в паметта на компютъра и позицията му на екрана в режим ТЕКСТ40

Ред от екрана	Първа страница	Втора страница
0	$400 — $427	$800 — $827
1	$480 — $4A7	$880 — $8A7
2	$500 — $527	$900 — $927
3	$580 — $5A7	$980 — $9A7
4	$600 — $627	$A00 — $A27
5	$680 — $6A7	$A80 — $AA7
6	$700 — $727	$B00 — $B27
7	$780 — $4A7	$B80 — $BA7
8	$428 — $44F	$828 — $84F
9	$4A8 — $4CF	$8A8 — $8CF
10	$528 — $54F	$928 — $94F
11	$5A8 — $5CF	$9A8 — $9CF
12	$628 — $64F	$A28 — $A4F
13	$6A8 — $6CF	$AA8 — $ACF
14	$728 — $74F	$B28 — $B4F
15	$7A8 — $7CF	$BA8 — $BCF
16	$450 — $477	$850 — $877
17	$4D0 — $4F7	$8D0 — $8F7
18	$550 — $577	$950 — $977
19	$5D0 — $5F7	$9D0 — $9F7
20	$650 — $677	$A50 — $A77
21	$6D0 — $6F7	$AD0 — $AF7
22	$750 — $777	$B50 — $B77
23	$7D0 — $7F7	$BD0 — $BF7

В режим ТЕКСТ80 броят на символите, изобразявани на екрана (съответно необходимата екранна памет), е двойно по-голям — 1920 (24 реда по 80 символа). В този режим има само една текстова страница с обем от 2 Кбайта, но тя има по-сложна структура. Разположена е по равно в системната и в допълнителната памет от компютъра, като и на двете места заема едни и същи адреси: $400 — $7FF. Символите, изобразявани на четните колони от екрана, са записани в допълнителната, а на нечетните — в системната памет. Превключването между системната и допълнителната памет за целите на изображението се извършва от програмноуправляемите ключове в компютъра. Адресът на клетката от паметта, в която е записан символът от определена позиция на екрана, може да се определи, като се използват данните от табл. 2.7:

адрес = начален адрес за съответния ред +
+ INT (адрес на колона / 2)

Така например адресът, на който е записан символът, изобразен на 13 ред, 48 колона, е:

– в допълнителната памет (колоната е четна);
– на адрес $6BD = $6A8 + INT ($30 / 2).

Таблица 2.7

Съответствие между адреса на символа в паметта на компютъра и позицията му на екрана в режим ТЕКСТ80

Ред от екрана	Адрес на символа в системната памет	Адрес на символа в допълнителната памет
1	2	3
0	$400 — $427	$400 — $427
1	$480 — $4A7	$480 — $4A7
2	$500 — $527	$500 — $527
3	$580 — $5A7	$580 — $5A7
4	$600 — $627	$600 — $627
5	$680 — $6A7	$680 — $6A7
6	$700 — $727	$700 — $727
7	$780 — $7A7	$780 — $7A7
8	$428 — $44F	$428 — $44F
9	$4A8 — $4CF	$4A8 — $4CF
10	$528 — $54F	$528 — $54F
11	$5A8 — $5CF	$5A8 — $5CF
12	$628 — $64F	$628 — $64F
13	$6A8 — $6CF	$6A8 — $6CF
14	$728 — $74F	$728 — $74F

Продължение на таблица 2.7

1	2	3
15	$7A8 — $7CF	$7A8 — $7CF
16	$450 — $477	$450 — $477
17	$4D0 — $4F7	$4D0 — $4F7
18	$550 — $577	$550 — $577
19	$5D0 — $5F7	$5D0 — $5F7
20	$650 — $677	$650 — $677
21	$6D0 — $6F7	$6D0 — $6F7
22	$750 — $777	$750 — $777
23	$7D0 — $7F7	$7D0 — $7F7

2.2.2. Изобразяване на графика

Графика с малка разделителна способност. В режим GR40 екранът се разделя на 48 реда. На всеки ред могат да се изобразят по 40 цветни блокчета (правоъгълници) в един от цветовете, дадени в табл.2.8. Между отделните точки няма разделително поле, така че съседните точки с един и същи цвят се сливат. При работа с монохроматичен монитор на екрана ясно се различават само черният и белият цвят, а всички останали образуват области с различна щриховка.

Таблица 2.8

Цветове в режим GR40

Код	Цвят	Код	Цвят
$00	Черен	$08	Кафяв
$01	Тъмновиолетово-червен	$09	Оранжев
$02	Тъмносин	$0A	Сив 2
$03	Виолетов	$0B	Розов
$04	Тъмнозелен	$0C	Светлозелен
$05	Сив 1	$0D	Жълт
$06	Син	$0E	Морскосин
$07	Светлосин	$0F	Бял

Цветовете, получавани във всеки един от графичните режими, са апаратно зависими. Това е следствие от използването на фазова разлика за кодиране на цветовете. Ето защо различните закъснения на сигналите, съдържащи информация за цветност, при преминаването им през видеотракта водят до промяна на един или друг цвят. Особено чувствителни са оранжевият и розовият цвят. В табл. 2.9 са дадени реалните цветове, получени при работа с модул RGB][и монитор "Taxan RGBvision II".

Таблица 2.9

Цветове в режим ГР40, получени при работа с модул RGB][и монитор Taxan RGB vision II

Код	Цвят	Код	Цвят
$00	Черен	$08	Тъмнокафяв
$01	Тъмновиолетово-червен	$09	Керемиденочервен
$02	Тъмносин	$0A	Сив 2
$03	Тъмновиолетов	$0B	Светловиолетов
$04	Тъмнозелен	$0C	Светлозелен
$05	Сив 1	$0D	Жълт
$06	Син	$0E	Синьо-зелен
$07	Светлосин	$0F	Бял

Режими ГР40 и ТЕКСТ40 използват една и съща екранна памет, като и в двата режима е възможно организирането на две страници — първа и втора, които заемат адреси $400 — $7FF и $800 — $BFF. Разликата между двата режима е в това, че при графиката с малка разделителна способност всеки байт от екранната памет съдържа информация за две цветни блокчета, а не за един символ. Блокчетата са с правоъгълна форма и се изобразяват едно под друго на екрана, като общата им площ е равна на площта, заемана от един символ в режим ТЕКСТ40 (широчина седем, височина осем точки). Съответствието на адреса на клетката от паметта с позицията на блокчетата, чиито цветове се определят от нейното съдържание, е същото както с позицията на съответния символ в режим ТЕКСТ40 (вж. табл. 2.6). В режим ГР40 обаче на всеки ред от текстовата страница съответстват два реда от цветни блокчета. Всяко блокче или по-точно неговият цвят се описва с четири бита, т.е. възможни са 16 цвята (вж. табл. 2.8). Младшите четири бита, записани в клетката, определят цвета на горното, а старшите четири бита — на долното блокче от двойката. Следователно в рамките на една колона от екрана цветът на блокчетата, намиращи се на четни редове, се определя от младшите четири бита, а на тези на нечетни редове — от старшите четири бита на съответния байт. Така например, ако се работи в първа страница, адресът в паметта на блокчето от 15

ред (съответства на 7 ред от режим ТЕКСТ40, долно блокче), 16 колона се получава, като към началния адрес за този ред ($780) се прибави номерът на колоната ($10):

$780 + $10 = $790

И тъй като редът е с нечетен номер, цветът на въпросното блокче ще се определи от старшите четири бита на байта, записан в тази клетка. Съгласно табл. 2.8, ако в нея е записана например стойността $5C, това блокче ще бъде светлозелено.

Графика с голяма разделителна способност. В режим ГР280 изображението представлява графика с голяма разделителна способност. Екранът се разделя на 192 реда по 280 точки. Цветовете, които могат да се получат, са дадени в табл. 2.10. Съществено за този режим е, че цветовете са позиционно зависими, т.е. цветът, в който може да се оцвети определена точка, зависи от нейната позиция на екрана. Това дава основание при работа с монитор за цветно изображение да се смята, че разделителната способност по хоризонталата е 140, а не 280 точки. В сила са следните зависимости:

– точките от четните колони могат да светят във виолетово или синьо;

– точките от нечетните колони могат да светят в зелено или оранжево;

– ако две съседни точки от един ред светят едновременно, независимо от позицията си те светят в бяло.

Таблица 2.10

Цветове в графика с голяма разделителна способност

Позиция на точката. Състояние на съответния бит	Нулиран седми бит (основни цветове)	Установен седми бит (допълнителни цветове)
Битът е нула	Черен 1	Черен 2
Битът е единица и точката е във:		
— четна колона	Виолетов	Син
— нечетна колона	Зелен	Оранжев
— две съседни точки	Бял 1	Бял 2

Програмното осигуряване на Правец–8А поддържа две графични страници. В тях се записват данните, изобразявани на екрана в режим ГР280. Страниците са с обем по 8 Кбайта и са разположени на адреси $2000 — $3FFF и $4000 — $5FFF. Графичните страници имат по-голям обем от текстовите, защото състоянието на всяка точка от екрана се определя от състоянието на определен бит от съ-

ответната страница. Всеки байт от екранната памет определя състоянието на група от седем последователни точки от екрана, т.е. в рамките на един ред от екрана съседните групи са записани на съседни адреси в паметта. Ако се приеме, че всяка група от седем бита образува една колона, екранът се разделя на 40 колони, чиито номера представляват т. нар. номера на базови колони. Тяхната позиция съвпада с позицията на колоните в режими ТЕКСТ40 и ГР40. Състоянието на всички точки от даден ред, принадлежащи към определена базова колона, се определя от един и същи байт. Старшият бит на байта (бит 7) се използва за избор на основните или допълнителните цветове (вж. табл. 2.10). Така всеки седем хоризонтално разположени точки, чието състояние се определя от един байт на екранната памет, в зависимост от това,дали старшият бит е установен или не, могат да светят или в бяло, синьо и оранжево, или в бяло, виолетово и зелено. При това младшият бит (бит 0) управлява най–лявата точка, бит 1 — съседната отдясно и т.н. За управлението на един ред от екрана са необходими 40 байта, а за 192 реда — 7680 байта. Подобно на текстовите и при графичните страници част от областта, определена като екранна памет, не се изобразява на екрана. Съответствието между адреса на клетката от паметта и позицията на седемте точки, управлявани от нейното съдържание, е показано в табл. 2.11. Номерът на реда, записан в скоби, е т.нар. номер на базов ред и отговаря на номера на съответния ред от текстовата страница. На всеки базов ред отговарят 8 реда от графичната страница. Ако номерът на графичния ред се означи с N(i), където i = 0 — 7, то действителният ред от графичната страница може да се изчисли по формулата:

<номер на ред от графичната страница> = 8 * <номер на базов ред> + N(i)

В табл. 2.11 са дадени адресите, съответстващи на графичните редове N(0). Адресите на останалите графични редове N(i), за i = 1, 2, ..., 7, се получават от тях с добавяне на M(i), където

$$M(i) = i * \$400$$

Така например, ако се работи в първа графична страница, адресът на точката от 15 ред, 16 колона се получава, като към началния адрес за съответния базов ред (1) се прибави изместването, отговарящо на графичен ред N (7), и изместването, отговарящо на номера на базовата колона (3):

$$\$3C83 = \$2080 + 7 * \$400 + 3$$

Състоянието на самата точка се определя от бит 1 на байта, записан на адрес $3C83.

На тази основа е възможно определянето на адреса на клетката от паметта и номера на бита от тази клетка, които съответстват на точка от екрана със зададени хоризонтална (H) и вертикална (V) координата. Ако приемем, че точката от екрана, намираща се в горния му ляв ъгъл, има координати H = V = 0, то състоянието на

точка с произволни H и V координати се определя от бит n на адрес m, където:

```
m = $2000 * номер на графичната страница +
  + $80 * Int ((V Mod 64) / 8) +
  + $28 * Int (V / 64) + $400 * (V Mod 8) +
  + Int (H / 7)
n = H Mod 7
```

Таблица 2.11

Графика с голяма разделителна способност. Съответствие между адреса от паметта и позицията на екрана

Ред от екрана (Базов ред)	Адрес в екранната памет: Първа страница	Втора страница
0 (0)	$2000 — $2027	$4000 — $4027
8 (1)	$2080 — $20A7	$4080 — $40A7
16 (2)	$2100 — $2127	$4100 — $4127
24 (3)	$2180 — $21A7	$4180 — $41A7
32 (4)	$2200 — $2227	$4200 — $4227
40 (5)	$2280 — $22A7	$4280 — $42A7
48 (6)	$2300 — $2327	$4300 — $4327
56 (7)	$2380 — $23A7	$4380 — $43A7
64 (8)	$2028 — $204F	$4028 — $404F
72 (9)	$20A8 — $20CF	$40A8 — $40CF
80 (10)	$2128 — $214F	$4128 — $414F
88 (11)	$21A8 — $21CF	$41A8 — $41CF
96 (12)	$2228 — $224F	$4228 — $424F
104 (13)	$22A8 — $22CF	$42A8 — $42CF
112 (14)	$2328 — $234F	$4328 — $434F
120 (15)	$23A8 — $23CF	$43A8 — $43CF
128 (16)	$2050 — $2077	$4050 — $4077
136 (17)	$20D0 — $20F7	$40D0 — $40F7
144 (18)	$2150 — $2177	$4150 — $4177
152 (19)	$21D0 — $21F7	$41D0 — $41F7
160 (20)	$2250 — $2277	$4250 — $4277
168 (21)	$22D0 — $22F7	$42D0 — $42F7
176 (22)	$2350 — $2377	$4350 — $4377
184 (23)	$23D0 — $23F7	$43D0 — $43F7

Управление на режимите на изображение. Получаване на графика с двойна разделителна способност. Режимът на изображение в осемразредните компютри от фамилията Правец се избира с помощта на вградените програмноуправляеми ключове. Необходимостта от програмна и апаратна съвместимост между различ-

ните компютри от фамилията определя аналогичността в управлението на тези режими. По отношение на възможностите на изображението на екрана Правец–8А в минимална конфигурация с нищо не превъзхожда предшествениците си. По-големите възможности на новия компютър се проявяват при наличието на модул с допълнителна памет. Увеличеният брой на режимите на изображение е довел до увеличаване и на броя на програмноуправляемите ключове, необходими за тяхното управление (табл. 2.12).

Таблица 2.12

Ключове за управление на режимите на изображение

Ключ	Адрес и начин за нулиране	Адрес и начин за установяване	Адрес за четене на ключа	Предназначение
1	2	3	4	5
ЗАП80	$C000 Запис	$C001 Запис	$C018 Четене на D7	ЗАПИС80. Модифицира действието на ключ СТР2. Когато е установен, ключ СТР2 включва допълнителната памет, а не втора страница
80КОЛ	$C00C Запис	$C00D Запис	$C01F Четене на D7	80КОЛОНИ. Ако ключ ТЕКСТ е установен, разрешава режим ТЕКСТ80. Ако ключ ТЕКСТ е нулиран, в зависимост от състоянието на другите ключове разрешава режим ГР80 или режим ГР560
АСМ	$C00E Запис/ Четене	$C00F Запис/ Четене	$C01E Четене на D7	АЛТЕРНАТИВНО СИМВОЛНО МНОЖЕСТВО. Включва алтернативното символно множество, необходимо при работа с 8-битова кодова таблица
ТЕКСТ	$C050 Запис/ Четене	$C051 Запис/ Четене	$C01A Четене на D7	ТЕКСТ. Включва режим ТЕКСТ. Когато е нулиран, компютърът работи в графичен режим
СМ	$C052 Запис/ Четене	$C053 Запис/ Четене	$C01B Четене на D7	СМЕСЕНО ИЗОБРАЖЕНИЕ. Ако ключ ТЕКСТ е нулиран, включва смесено изображение
СТР2	$C054 Запис/ Четене	$C055 Запис/ Четене	$C01C Четене на D7	ВТОРА СТРАНИЦА. Избира втора страница на екранната памет. Действието му се модифицира, ако ключ ЗАПИС е установен. В този случай се избира екранната памет от допълнителната памет

Продължение на таблица 2.12

1	2	3	4	5
ГР.ВРС	$C056 Запис/ Четене	$C057 Запис/ Четене	$C01D Четене на D7	ГРАФИКА С ГОЛЯМА РАЗДЕЛИТЕЛНА СПОСОБНОСТ. Ако ключ ТЕКСТ е нулиран, включва режим ГР280
ЦИЗ	$C05E Запис/ Четене	$C05F Запис/ Четене	Не се чете	ЦИФРОВ ИЗХОД 3. През модула с допълнителна памет се свързва към шина FTXT (ПРИНУДИТЕЛЕН ТЕКСТ). Режимите с двойна разделителна способност са разрешени, ако ключ ЦИЗ е нулиран, а ключ 80КОЛ е установен

Следващите няколко примера илюстрират използването на програмноуправляемите ключове.

Пример 1. Установяване на режим ТЕКСТ40 с избор на първа страница

```
LDA $C051   ;Установява текстов режим
LDA $C054   ;Нулира ключ СТР2. Избира първа страница
STA $C00C   ;Установява изображение с 40 колони
```

Пример 2. Установяване на режим ТЕКСТ80 с избор на първа страница

```
LDA $C051   ;Установява текстов режим
LDA $C054   ;Избира първа страница
STA $C00D   ;Установява режим на 80-колонно изображение
```

Пример 3. Установяване на режим СМ40/80. Избор на първа страница

```
LDA $C050   ;Установява графичен режим
LDA $C053   ;Установява смесено изображение
LDA $C054   ;Избира първа страница
LDA $C056   ;Установява графика с малка
            ;разделителна способност
LDA $C05F   ;Забранява графиката с двойна
            ;разделителна способност
STA $C00D   ;Установява изображение с 80 колони
```

Пример 4. Установяване на режим ГР560. Избор на първа страница

```
LDA $C050   ;Установява графично изображение
LDA $C052   ;Изключва режима на смесено изображение
```

```
LDA $C057  ;Установява графика с голяма
           ;разделителна способност
LDA $C05E  ;Установява графика с двойна
           ;разделителна способност (ГР560)
STA $C001  ;Разрешава достъпа до допълнителната
           ;памет чрез ключ СТР2
STA $C00D  ;Установява изображение с 80 колони
```

Преди да разгледаме някои конкретни програми, реализиращи режими ГР80 и ГР560, нека се спрем на това, което ги отличава от стандартните графични режими. Основното е организацията на екранната памет. Тя е разделена на две еднакви страници, разположена е на едни и същи адреси в системната и допълнителната памет (вж. режим ТЕКСТ80) и има двойно по-голям обем в сравнение със съответния стандартен режим (ГР40 или ГР280). По-различно е и съответствието между адреса от екранната памет и позицията на екрана. Така в режим ГР80 цветът на блокчетата от колоните с четни номера се определя от съдържанието на частта от екранната памет, разположена в допълнителната памет, а на тези от нечетните колони — от екранната памет, разположена в системната памет. Ситуацията в режим ГР560 е подобна, но тук делението е на колони от седем последователни точки. И в този режим четните колони (точки 0 до 6, 14 до 20, 28 до 34 и т.н.) се определят от съдържанието на страницата, разположена в допълнителната памет, а нечетните (точки 7 до 13, 21 до 27, 35 до 41 и т.н.) — от съдържанието на страницата, разположена в системната памет. Тези особености, съчетани с използваната система за получаване на цветно изображение, изискват сложни алгоритми за определяне на връзката между адреса от екранната памет и координатите и цвета на точката на екрана. Получените програми на БЕЙСИК са бавни и подходящи само за демонстрации и обучение. За решаването на реални задачи е необходимо да се използват програми на машинен език.

Следващата програма (пример 5) демонстрира режим ГР80. Обърнете внимание на подпрограмата, която изчертава получените резултати. Тя получава координатите на блокчето H (0—78) и V (0—47) и преди да ги изчертае, ги обработва. Обработката е стандартна за този режим и включва:

1. За всяка стойност на хоризонталната координата (H) се определя къде попада блокчето и в зависимост от това с ключ СТР2 се включва системната или допълнителната памет.

2. Тъй като за изчертаването на блокчето се използват командите на БЕЙСИК от режим ГР40, параметърът H се привежда в съответствие с допустимите стойности за този режим (0 — 39), като вместо действителната стойност на хоризонталната позиция (H) се използва INT (H/2).

3. Стойността, определяща цвета на блокчетата, съответст-

ващи на допълнителната памет, се получава от стойността за текущия цвят с единична ротация надясно. В противен случай съседните блокчета имат различен цвят. Причината за този ефект се крие в метода, използван за кодиране на цветовете, и подробното му разглеждане не е предмет на настоящата книга.

Пример 5. Режим ГР80. Демонстрационна програма

```
5   Rem ГР80 демо
7   Goto 100
9   Rem Изчертаване
10  If H / 2 <> Int (H / 2) Then Poke NPG2, 0: Plot (H / 2), V: Return:
    Rem Нечетните колони са в системната памет
20  Poke PG2, 0: C = Peek (CLR): C1 = C / 2: C1% = C1: Rem
    Включване на допълнителната памет. Определяне на текущия
    цвят
30  If C1 - C1% Then C1% = C1% + 128: Rem Ротация на цвета
    с едно поле надясно
40  Poke CLR, C1%: Plot (H / 2), V: Poke CLR, C: Rem Изчертаване
    на блокчето от допълнителната памет с ротирания цвят
    и възстановяване на цвета
50  Return
99  Rem Дефиниране на константите
100 PI2    =  6.2831852: Rem 2 * PI
110 PG2    =  $C055: Rem Втора страница
120 NPG2   =  $C054: Rem Първа страница
130 STR80  =  $C001: Rem ЗАП80
140 NMXD   =  $C052: Rem Нулиране на смесения режим
150 COL80  =  $C00D: Rem 80КОЛ
160 NAN3   =  $C05E: Rem Нулиране на ЦИЗ
170 CLR    =  48: Rem Цвят в ГР40
199 Rem Установяване на режима
200 Gr: Poke STR80, 0: Poke NMXD, 0: Poke COL80, 0: Poke NAN3,
    0: Poke PG2, 0: Call - 1998: Poke NPG2, 0: Call - 1998:
    Rem Call - 1998 изчиства страницата
299 Rem Определяне координатите на блокчето
300 For I = 1 To 15: Color = I: Rem Последователно се задават
    всички възможни цветове
310 For H = 0 To 78: Rem Определяне на хоризонталната коорди-
    ната на блокчето
320 V = 23 - (8 + I) * Sin (PI2 * 79 * H): Rem Определяне на вер-
    тикалната координата на блокчето
330 Gosub 10: Next: Next
340 Goto 340: Rem За край натиснете CONTROL—RESET
```

В режим ГР560 е възможно използването на различни алгоритми за определяне на цвета на точките в зависимост от координатите

им. Два от тях са демонстрирани в примери 6 и 7. Първият е по-бърз, а вторият е по-подходящ за единични точки. Програмата от пример 8 е значително по-бърза и от двете, защото не изчислява цветовете на точките. Предназначена е за работа с монохроматичен монитор. Трябва да се отбележи, че следващите три програми са производни една от друга и от програма 5. При това:

1. Всички програми ползват едни и същи константи. Затова програмни редове с номера 99 — 160 са еднакви и за четирите програми. В програмите за графика с двойна голяма разделителна способност се дефинира още една константа (рег 180 от програма 6). Рег 7 също е общ за всички програми.

2. Програмните редове с номера 9 — 90 са запазени за подпрограми. Естествено е използваните подпрограми за графика с двойна малка разделителна способност (пример 5) и графика с двойна голяма разделителна способност (примери 6, 7 и 8) да са различни. Програмата за монохроматично изображение не определя цвета на точката и поради това не използва съответната подпрограма (редове 9 — 20 от програма 6).

3. Аналогично частта от програмата, установяваща режима на изображението (редове 199 и 200), и тази, зареждаща машинната подпрограма за изчистване на графичните страници (редове 180 и 190), са общи за програмите за графика с двойна голяма разделителна способност.

4. Редовете с номера, по-големи от 299, са индивидуални за всяка програма.

Програмите 6 и 7 изчертават 15 синусоиди в режим GR560, всяка с различен цвят и амплитуда. Използваните цветове са 16 на брой, еквивалентни са на цветовете от режим GR40 и се задават със същите кодове (0 — 15) на рег 300 от програмата. В подпрограмата за преобразуване (редове 10 — 20) кодът на цвета се преобразува от десетичен в шестнайсетичен и получените четири бита се присвояват на променливите A(0) до A(3). Съответствието между цветовете в GR40 и GR560 е такова, че точките, за които са в сила равенствата: H0 MOD 4 = 0, H0 MOD 4 = 1, H0 MOD 4 = 2 и H0 MOD 4 = 3, съответстват на A(1), A(2), A(3) и A(0). Така, ако в режим GR560 се изберат точки с номера 0 и 3 от даден ред, ще се получи цветът, отговарящ на код *A(1),*A(2),A(3),A(0) = $3 в режим GR40.

Двата алгоритъма, респ. двете програми, се различават само по метода за получаване на 4-битовия код на цвета за режим GR560 (редове от 300 нагоре в съответните програми 6 и 7). При първия метод (пример 6) всеки ред от екрана се разделя на 140 позиции, съдържащи по 4 точки. За всяка позиция в зависимост от зададения цвят CLR (отговаря на цвят от режим GR40) се изчисляват точките A(1), A(2), A(3) и A(0), които трябва да светят, за да се получи желаният цвят. При втория метод (програма 7) се поддържат 557 позиции по 4 точки всяка. За да се получи желаният цвят (зададен от CLR), тряб-

ва да се изчисли какво е отместването за всяка от тези позиции по отношение на нулевата. Нулевата позиция започва с точка, определена с A(1). Така например позиция 255 започва с A(3) и се състои от точките, определени от A(3), A(0), A(1) и A(2) в изброения ред, защото 255 MOD 4 = 3. Следователно осветяването на позиция 255 в цвят, определен с код $C (1100), се свежда до осветяването на точки 255 и 258 ($C = 1100 = A(3), A(2), *A(1), *A(0)).

Пример 6. Режим ГР560. Първи алгоритъм за определяне на цвета

```
5   Rem ГР560 демо — V1
7   Goto 100
9   Rem Преобразуване кода на цвета от шестнайсетичен в десетичен вид
10  CLR% = CLR: For I = 0 To 3: A(I) = 1: CLR = CLR% / 2: CLR% = CLR% / 2: If CLR = CLR% Then A(I) = 0
20  Next: Return
49  Rem Изчертаване на графиката
50  M14 = Int (H) / 14: M14% = M14: P14% = 14 * (M14 - M14%) + 0.5: Poke PG2, 0: If P14% > 6 Then Poke NPG2, 0: P14% = P14% - 7
60  Hplot M14% * 7 + P14%, V: Return
180 HCL = 768: Rem Начало на програма за изчистване на графичната страница
190 Data 169, 32, 76, 234, 243: For I = 0 To 4: Read A: Poke HCL + I, A: Next
199 Rem Установяване на режима на изображение. Изчистване на графичните страници
200 Hgr: Poke STR80, 0: Poke NMXD, 0: Poke COL80, 0: Poke NAN3, 0: Poke PG2, 0: Call HCL
299 Rem Определяне координатите и цвета на точките
300 For J = 1 To 15: CLR = J: Gosub 10: For H0 = 0 To 556 Step 4: V = 96 — (20 + 5 * J) * Sin (PI2 / 556 * H0)
310 If A(1) Then H = H0: Gosub 50
320 If A(2) Then H = H0 + 1: Gosub 50
330 If A(3) Then H = H0 + 2: Gosub 50
340 If A(0) Then H = H0 + 3: Gosub 50
350 Next: Next
360 Goto 360
```

Пример 7. Режим ГР560. Втори алгоритъм за определяне на цвета

```
5   Rem ГР560 демо — V2
299 Rem Определяне координатите и цвета на точките
300 For J = 1 To 15: CLR = J: Gosub 10: For H0 = 0 To 556: V = 96 - (20 + 5 * J) * Sin (PI2 / 556 * H0)
```

```
310 P4% = (H0 / 4 — Int (H0 / 4)) * 4 + 0.5
320 If P4% = 0 And A(1) Or P4% = 1 And A(2) Or P4% = 2 And A(3)
    Or P4% = 3 And A(0) Then H = H0: Gosub 50
330 If P4% = 0 And A(2) Or P4% = 1 And A(3) Or P4% = 2 And A(0)
    Or P4% = 3 And A(1) Then H = H0 + 1: Gosub 50
340 If P4% = 0 And A(3) Or P4% = 1 And A(0) Or P4% = 2 And A(1)
    Or P4% = 3 And A(2) Then H = H0 + 2: Gosub 50
350 If P4% = 0 And A(0) Or P4% = 1 And A(1) Or P4% = 2 And A(2)
    Or P4% = 3 And A(3) Then H = H0 + 3: Gosub 50
360 Next: Next
370 Goto 370
```

Редове от 9 до 20, от 49 до 60, 180, 190, 199 и 200 съвпадат с тези от програма 6.

Пример 8. Режим ГР560. Монохроматично изображение

```
5   Rem ГР560. Монохроматично изображение
299 Rem Определяне координатите на точките
300 Hcolor= 7: For H = 0 To 556: V = 96 – 96 * Sin (PI2 / 556 * H):
    Gosub 50: Next
310 Gomo 310
```

Редове от 49 до 60, 180, 190, 199 и 200 съвпадат с тези от програма 6.

2.3. Допълнителни входно–изходни устройства

Вградените входно–изходни устройства условно се делят на основни и допълнителни. Основни входно–изходни устройства са клавиатурата (вход) и мониторът (изход), а към допълнителните спадат високоговорителят (изход), цифровите входове и изходи и аналоговите входове на компютъра. От гледна точка на програмиста начинът на работа с допълнителните входно–изходни устройства не се различава от работата с програмноуправляемите ключове. Това е така, защото и тяхното управление става с обръщение към определени клетки от паметта на компютъра (вж. табл. 2.13). Тъй като някои от допълнителните устройства се управляват от тригери с броячен вход, които сменят състоянието си при всяко обръщение към тях, за предпочитане е обръщението да става с операция четене, а не с операция запис, защото при изпълнение на операция запис компютърът понякога издава два пъти един и същи адрес.

Високоговорител. Вграденият високоговорител се управлява от програмноуправляемия ключ на адрес $C030. Всяко обръщение към този адрес сменя състоянието на изхода на ключа, а в съответствие с това се изменя и сигналът, подаван на високоговорителя. Тонове с различна честота и продължителност се получават с използ-

ване на комбинации от програмни цикли с различна продължителност. Следната програма генерира тон с честота около 1 kHz.

```
        PHP             ;Запазва съдържанието на
                        ;регистъра на състоянието
        SEI
        LDY #200        ;Продължителност
LOOP    LDX #100        ;Честота 1 kHz (полупериод
                        ;100 * 5 = 500 μs)
LOOP5MS DEX
        BNE LOOP5MS     ;Цикъл 5 μs
        BIT $C030
        DEY
        BNE LOOP
        PLP             ;Възстановява съдържанието на
                        ;регистъра на състоянието
        RTS
```

В действителност поради загубеното време извън цикъла LOOP5MS полупериодът е 510 μs, а не 500 μs. По-голяма точност ще се получи, ако на реда с етикет LOOP се постави инструкцията LDX #88.

Цифрови входове. Персоналният компютър Правец–8А има три едноразредни цифрови входа, означени с ЦВХ0, ЦВХ1 и ЦВХ2. Те са предназначени за приемане на сигнали с TTL нива.

Внимание! Подаването на сигнал с ниво над 5 V на цифровите входове може да повреди компютъра.

Цифровите входове са изведени на съединителя за управление на игри. Към тях най–често се включват бутоните на ръчките за управление на игри, на мишки, но е възможно да бъдат използвани и от други електронни устройства. Така например в Правец–8А към първите два цифрови входа (ЦВХ0 и ЦВХ1) са свързани клавиши F1 и F2 от клавиатурата. Състоянието на съответния цифров вход (логическа единица или логическа нула) се чете на определен адрес (вж. табл. 2.13) по шина D7. Това означава, че само старшият бит на байта, прочетен от този адрес, е валиден, а всички останали битове са случайни.

Цифрови изходи. Компютърът Правец–8А има четири едноразредни цифрови изхода ЦИЗХ0, ЦИЗХ1, ЦИЗХ2 и ЦИЗХ3, които са изведени на съединителя за управление на игри. Напреженията, получавани от тях, са с TTL нива и могат да се използват за управление на различни електронни схеми. Товароспособността им отговаря на товароспособността на стандартен TTL LS изход. Всеки от цифровите изходи се управлява от програмноуправляем ключ, разположен на два съседни адреса от паметта на компютъра (вж. табл. 2.13). Всяко обръщение към по-малкия адрес нулира съответния цифров изход, а към по-големия — го установява. Така например инструкцията

BIT $C058 нулира цифров изход 0, а инструкция BIT $C059 установява този изход.

Стробиращ изход. Стробиращият изход на компютъра е изведен на съединителя за управление на игри. Той може да се използва там, където е необходимо синхронизирането на два или повече от цифровите изходи на компютъра. Нормално нивото на стробиращия изход е логическа единица. То става логическа нула за около 0,5μs при всяко обръщение към адрес $C040 (вж. табл. 2.13).

Внимание! При операция запис или при операция четене с индексна адресация на адрес $C040 е възможно на стробиращия изход да се получи двоен импулс.

Аналогови входове. Правец–8А има четири аналогови (потенциометрични) входа ABX0, ABX1, ABX2 и ABX3, които са изведени на съединителя за управление на игри. Те са предназначени за измерване на съпротивление в обхвата от 0 до 150 k Ω. Измерваното съпротивление R_x се свързва между потенциометричния вход и линията +5V така, че става част от времезадаваща верига, времеконстантата на която се измерва по програмен път. За целта е необходимо най–напред да се нулират времезадаващите схеми на четирите потенциометрични входа. Това става с обръщение към адрес $C070 (вж. табл. 2.13). При това положение старшият бит на байтовете, прочетени от адресите, съответстващи на аналоговите входове ($C064 до $C067), се установява в единица и остава в това състояние определено време (максимум 3 ms). Това време е правопропорционално на стойността на съпротивлението R_x, свързано към съответния аналогов вход. Ако входовете са отворени (към тях няма свързано съпротивление), съответният старши бит остава в единица неопределено дълго време.

Таблица 2.13

Адреси, свързани с допълнителните входно–изходни устройства

Адрес	Функция
1	2
$C030	Високоговорител
$C040	Стробиращ изход
$C058	Цифров изход 0 изключен
$C059	Цифров изход 0 включен
$C05A	Цифров изход 1 изключен
$C05B	Цифров изход 1 включен
$C05C	Цифров изход 2 изключен

Продължение на таблица 2.13

1	2
$C05D	Цифров изход 2 включен
$C05E	Цифров изход 3 изключен
$C05F	Цифров изход 3 включен
$C061	Цифров вход 0
$C062	Цифров вход 1
$C063	Цифров вход 2
$C064	Аналогов (потенциометричен) вход 0
$C065	Аналогов (потенциометричен) вход 1
$C066	Аналогов (потенциометричен) вход 2
$C067	Аналогов (потенциометричен) вход 3
$C070	Нулиране на аналоговите входове

Глава 3. Организация на паметта

Персоналният компютър Правец–8А е построен на основата на осемразредния микропроцесор СМ630. Адресната шина на този микропроцесор е 16-разредна, така че могат директно да се адресират 64 Кбайта памет. Тук се включват постоянната и оперативната памет на компютъра, както и паметта, резервирана за входно–изходните устройства. Стремежът за по-голяма оперативна и постоянна памет, по-сложни и по-функционални драйверни програми, съчетан с ограниченото адресно пространство на микропроцесора, налага различни устройства да ползват едни и същи адреси. Естествено това не може да става в едно и също време. Използваният метод на страничната организация на паметта осигурява приемлив компромис между бързодействие, простота на обслужването и обем на паметта. Благодарение на този метод в компютъра са създадени условия за включване на две страници по 64 Кбайта оперативна памет (системна и допълнителна), 16 Кбайта постоянна памет и 4 Кбайта памет за обслужване на входно–изходните устройства. Нещо повече в архитектурата на компютъра са заложени два различни и независими един от друг механизми за разширяване на неговата памет, основаващи се на страничната организация.

3.1. Системна памет

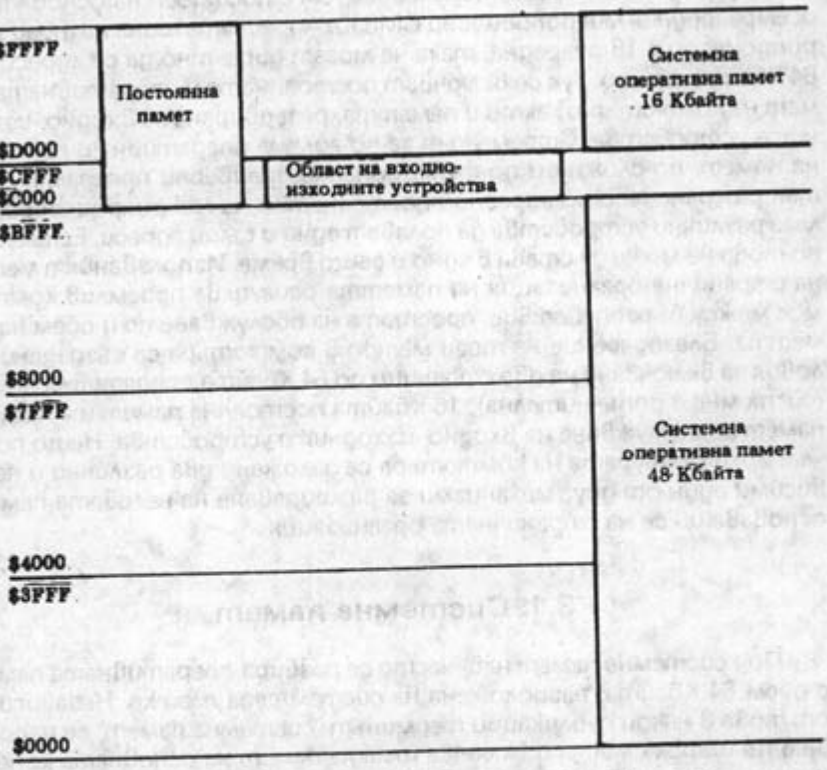
Под системна памет най–често се разбира оперативната памет с обем 64 Кбайта, разположена на системната платка. Независимо от това в някои публикации терминът "системна памет" се използва в по-широк смисъл и включва цялата памет на основната конфигурация на компютъра. Във връзка с разглеждането на организацията на паметта в настоящата книга са приети следните термини.

– *Системна памет.* С него се означава оперативната памет от 64 Кбайта, разположена на системната платка на компютъра, т.е. това е оперативната памет от минималната конфигурация на компютъра.

– *Допълнителна памет.* Така се нарича оперативната памет с обем 64 Кбайта, разположена на допълнителния модул за разширяване на паметта, включен в съединител X0.

В частност модулът МЕГАРАМ, който може да има до 1 Мбайт памет, също е със странична организация (16 страници по 64 Кбайта). Когато модул МЕГАРАМ е включен в компютъра, под допълнителна памет се разбира паметта на тази страница от 64 Кбайта, която се използва от схемите за генериране на изображение при ра-

бота в режимите с двойна разделителна способност ТЕКСТ80, ГР80 и ГР560. Почти винаги това е първата страница от 64К на модула (страница номер 0). С други думи, под допълнителна памет най–често се разбира минималната допълнителна оперативна памет, необходима за реализиране на пълните възможности на компютъра Правец–8А.



Фиг. 3.1. Карта на паметта на Правец–8А

Областта от системната, респ. допълнителната оперативна памет, заемаща адресната област $D000 — $FFFF, има специфична организация. Тя дублира постоянната памет в тази област и не може да се използва от програмите, написани на БЕЙСИК.

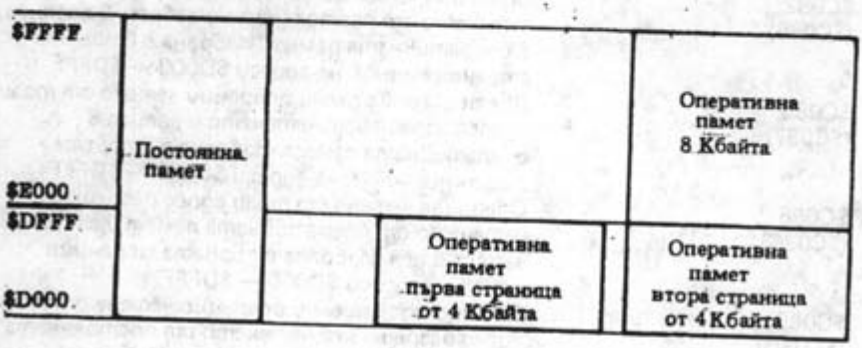
– *Обща памет.* С този термин се означава общата оперативна памет на компютъра.

– *Памет.* Това е термин, обединяващ различните типове памет, разположени на системната платка (оперативна и постоянна памет и адресна област, предназначена за използване от входно–изходните устройства).

На фиг. 3.1 е показана картата на паметта на компютъра Правец–8А. По принцип тя не се различава от картата на паметта на компютъра Правец–8М и е еквивалентна на картата на паметта на компютъра Правец–82 с включен модул 16K DRAM. Това е естествено, защото именно еднаквата организация на паметта на трите компютъра е един от основните фактори за тяхната съвместимост. Единствената разлика, на която ще се спрем по-късно, е в организацията на постоянната памет и е свързана с разширяването на областта, заемана от програмата МОНИТОР и от интерпретатора на БЕЙСИК.

3.2. Организация на областта $D000 — $FFFF

Както се вижда от фиг. 3.1, адресното пространство в областта $D000 — $FFFF (общо 12 Кбайта) се използва както от постоянната, така и от оперативната памет. На тези адреси в постоянната памет са разположени програмата МОНИТОР и интерпретаторът на БЕЙСИК. На същите адреси са разположени и 16 Кбайта оперативна памет, чиято организация отговаря на организацията на паметта на модул 16K DRAM (фиг. 3.2) и подобно на нея се използва от операционните системи USCD, CP/M и ПроДОС, от някои езици от високо ниво и някои потребителски програми. Като техническо решение това е класически пример на странична организация. На едни и същи адреси ($D000 — $FFFF) са разположени и постоянна, и оперативна памет, като част от самата оперативна памет дублира адресната област $D000 — $DFFF. Тъй като в определен момент е възможен достъпът или само до постоянната, или само до оперативната памет, ясно е, че в компютъра трябва да съществуват апаратни средства, с които да се разрешава този достъп. Такива средства



Фиг. 3.2. Карта на паметта в областта $D000 — $FFFF

са програмноуправляемите ключове и свързаните с тях адреси, показани в табл. 3.1. Тяхното управление се извършва с една или две последователни операции четене от съответния адрес, в резултат на което е възможно:

– да се разреши достъпът до постоянната или оперативната памет в областта $D000 — $FFFF;

– да се разреши или да се забрани записът в оперативната памет от областта $D000 — $FFFF;

– да се избере първата или втората страница от оперативна памет от областта $D000 — $DFFF.

Таблица 3.1

Управление на паметта от областта $D000 — $FFFF

Адрес	Име	Функция
1	2	3
$C080 ($C084)		Операция четене от този адрес разрешава четенето от оперативната памет и забранява записа в нея. Избрана е втората страница от 4K на адреси $D000 — $DFFF
$C081 ($C085)		Две последователни операции четене от този адрес разрешават четенето от постоянната и записа в оперативната памет. Избрана е втората страница от 4K на адреси $D000 — $DFFF
$C082 ($C086)		Операция четене от този адрес разрешава четенето от постоянната и забранява записа в оперативната памет. Избрана е втората страница от 4K на адреси $D000 — $DFFF
$C083 ($C087)		Две последователни операции четене от този адрес разрешават четенето и записа в оперативната памет. Избрана е втората страница от 4K на адреси $D000 — $DFFF
$C088 ($C08C)		Операция четене от този адрес разрешава четенето от оперативната памет и забранява записа в нея. Избрана е първата страница от 4K на адреси $D000 — $DFFF
$C089 ($C08D)		Две последователни операции четене от този адрес разрешават четенето от постоянната и записа в оперативната памет. Избрана е първата страница от 4K на адреси $D000 — $DFFF

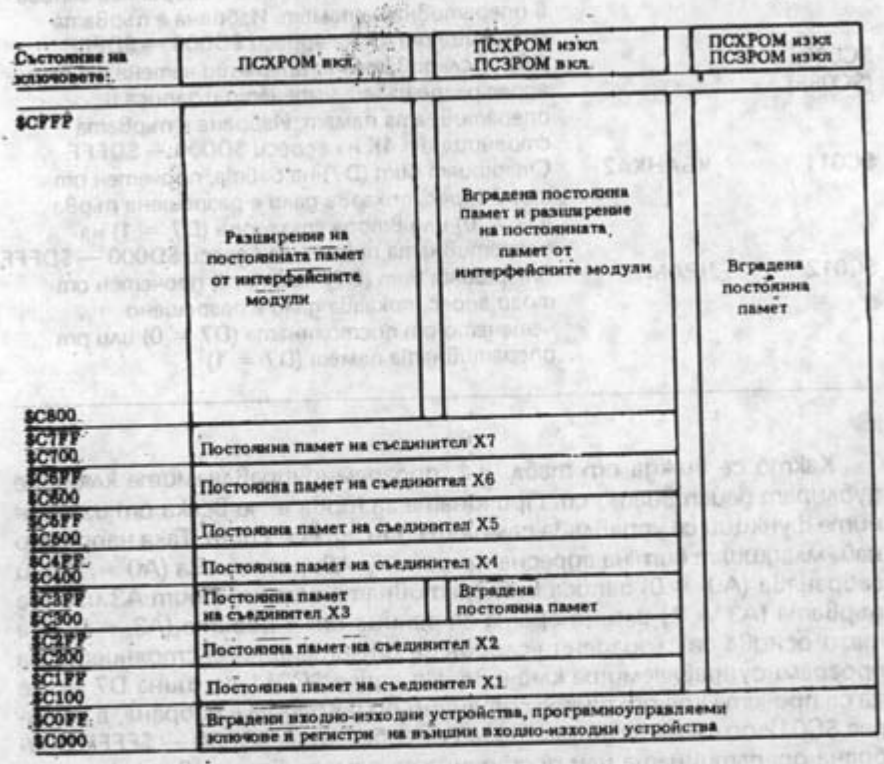
1	2	3
$C08A ($C08E)		Операция четене от този адрес разрешава четенето от постоянната и забранява записа в оперативната памет. Избрана е първата страница от 4К на адреси $D000 — $DFFF
$C08B ($C08F)		Две последователни операции четене от този адрес разрешават четенето и записа в оперативната памет. Избрана е първата страница от 4К на адреси $D000 — $DFFF
$C011	ЧБАНКА2	Старшият бит (D7) на байта, прочетен от този адрес, показва дали е разрешена първа (D7 = 0) или втора страница (D7 = 1) на оперативната памет от адреси $D000 — $DFFF
$C012	ЧРАМ	Старшият бит (D7) на байта, прочетен от този адрес, показва дали е разрешено четенето от постоянната (D7 = 0) или от оперативната памет (D7 = 1)

Както се вижда от табл. 3.1, програмноуправляемите ключове дублират действието си. Причината за това е, че всяка от изброените функции се управлява само от един адресен бит. Така например най-младшият бит на адресната шина — A0, разрешава (A0 = 1) или забранява (A0 = 0) записа в оперативната памет, а бит A3 избира първата (A3 = 1) или втората страница от 4 Кбайта (A3 = 0). На тази основа са създадени условия за отчитане на състоянието на програмноуправляемите ключове. На адрес $C011 по шина D7 може да се прочете коя от двете страници по 4 Кбайта е избрана, а на адрес $C012 по същата шина — дали от областта $D000 — $FFFF е избрана оперативната или постоянната памет. Единственият начин да се определи дали е разрешен или не записът в оперативната памет е да се направи опит за запис в съответната област.

При студен и при топъл старт на компютъра програмноуправляемите ключове се установяват така, че е разрешено четенето от постоянната, записът в оперативната памет и е разрешен достъпът до втората страница от 4 Кбайта. Обърнете внимание, че за разлика от компютрите Правец-82 и Правец-8М в компютъра Правец-8А при топлия старт програмноуправляемите ключове винаги се установяват в едно и също състояние. Ето защо в Правец-8А RESET векторът не може да се използва за предаване на управлението на програма, записана в областта $D000 — $FFFF.

3.3. Област на входно–изходните устройства

Персоналният компютър Правец–8А третира всички входно–изходни устройства като клетки от паметта с определени адреси. Ето защо в неговото адресно пространство е отделена специална област за ползване от входно–изходните устройства (вж. фиг. 3.1).



Фиг. 3.3. Област на входно–изходните устройства

Тя се нарича област на входно–изходните устройства и заема адреси $C000 — $CFFF, като допълнително се разделя на подобласти (фиг. 3.3). Организацията е следната.

* Вградените входно–изходни устройства са достъпни в областта $C000 — $C08F.

* За всеки един от седемте съединителя за включване на допълнителни модули (X1 — X7) са отделени по две подобласти. Първата е с обем 16 байта, заема адреси $C0n0 — $C0nF (n е номерът на съединителя плюс 8) и е предназначена за използване от регистрите за управление на съответните модули. Втората е с обем 256 байта и

заема адреси $Cn00 — $CnFF (n е номерът на съединителя). Обикновено на всеки от тези адреси се намира малка по обем постоянна памет, в която е записана драйверната програма на съответния интелигентен контролер.

* Най–голямата област (2 Кбайта) заема адреси $C800 — $CFFF и може да се използва от всички допълнителни модули. Обикновено тези адреси се присвояват на постоянна памет, съдържаща по-дълги драйверни програми, но е възможно това да бъде и статична оперативна памет, в която да се зареждат различни драйверни програми в зависимост от конкретното приложение на модула.

Съществена особеност на компютъра Правец–8А е, че неговата постоянна памет е с обем 16 Кбайта и е разположена в областта $C000 — $FFFF. Реално се използват 256 байта по-малко, защото областта $C000 — $C0FF е резервирана за вградените входно–изходни устройства. От фиг.3.3 се вижда, че част от постоянната памет дублира и областта $C100 — $CFFF, предназначена за използване от външните входно–изходни устройства. В тази област са записани драйверната програма за управление на изображението в режим ТЕКСТ80 и част от допълнителните команди на програмата МОНИТОР и на интерпретатора на БЕЙСИК. Достъпът до тази област, респ. до програмите, записани в нея, се разрешава с програмноуправляемите ключове, описани в табл. 3.2.

Таблица 3.2

Управление на постоянната памет на компютъра, разположена в областта на входно–изходните устройства

Адрес	Име	Функция
1	2	3
$C006	ПСХРОМ ВКЛ	Операция запис на този адрес разрешава достъпа до постоянната памет в областта $C100 — $C7FF, разположена на допълнителните модули
$C007	ПСХРОМ ИЗКЛ	Операция запис на този адрес разрешава достъпа до постоянната памет на компютъра в областта $C100 — $CFFF. На тези адреси освен драйверната програма за управление на изображението в режим ТЕКСТ80 са записани и новите функции на програмата МОНИТОР и интерпретатора на БЕЙСИК

1	2	3
$C00A	ПСЗРОМ ИЗКЛ	Операция запис на този адрес разрешава достъпа до вградената в компютъра постоянна памет в областите $C300 — $C3FF и $C800 — $CFFF. На тези адреси е записана драйверната програма за управление на изображението в режим ТЕКСТ80. Ако в допълнителния съединител X0 е включен модул за разширение на паметта, ключ ПСЗРОМ се изключва при студен старт на компютъра
$C00B	ПСЗРОМ ВКЛ	Операция запис на този адрес разрешава достъпа до постоянната памет на модула, включен в съединител X3 (адреси $C300 — $C3FF)
$C015	ЧЕТЕНЕ КЛЮЧ ПСХРОМ	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ ПСХРОМ
$C017	ЧЕТЕНЕ КЛЮЧ ПСЗРОМ	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ ПСЗРОМ

3.4. Допълнителна памет

Разширяването на паметта на компютъра Правец–8А става по два независими един от друг начина. Първият е познат от архитектурата на компютъра Правец–82 и е основан на използването на сигнал *INH, изведен на 32-ри извод на всеки от съединителите X1 — X7. Генерирането на този сигнал забранява достъпа до постоянната и оперативната памет на компютъра в адресната област $D000 — $FFFF. Това дава възможност за конструиране на модули за разширение на паметта от типа на модулите 16K DRAM и 128K RAM, използвани с Правец–82, при които допълнителната памет има една или няколко страници с организизацията, показана на фиг. 3.2 (по 16 Кбайта).

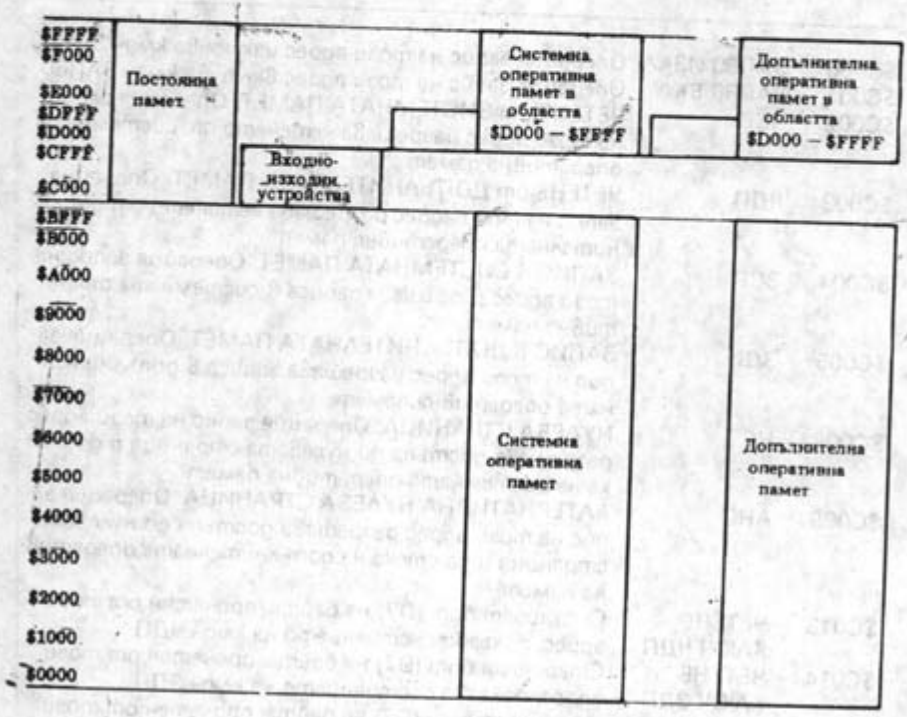
Вторият начин за разширяване на паметта се основава на новата архитектура на компютъра Правец–8А и се осъществява чрез допълнителни модули, включени в съединителя X0. Допълнителната памет е на страници по 64 Кбайта и в рамките на една страница изцяло повтаря организацията на системната памет, включително и в областта $D000 — $FFFF (фиг. 3.4). Нещо повече, достъпът до тази

област от допълнителната памет се определя от състоянието на програмноуправляемите ключове, описани в т. 3.2, и може да се забрани със сигнал *INH. Механизмът за достъп до допълнителната памет като цяло е основан на използването на програмноуправляеми ключове (табл. 3.3). Според предназначението си те условно могат да се разделят на два типа:

1. Ключове за достъп до определена област: АНС (АЛТЕРНАТИВНА НУЛЕВА СТРАНИЦА), ЗАП80, СТР2 и всички ключове за управление на достъпа до областта $D000 — $FFFF, описани в табл. 3.1.

2. Ключове за разрешаване на определен вид достъп, като ключ ЧДП (ЧЕТЕНЕ от ДОПЪЛНИТЕЛНАТА ПАМЕТ) и ключ ЗДП (ЗАПИС в ДОПЪЛНИТЕЛНАТА ПАМЕТ). Използването на тези ключове и изобщо на допълнителната памет на компютъра Правец–8А има някои особености:

– При превключване от системната към допълнителната памет и обратно състоянието на ключовете, управляващи достъпа до областта $D000 — $FFFF, не се изменя.



Фиг. 3.4. Карта на паметта на компютър Правец–8А с допълнителна памет

– Ключовете ЧДП и ЗДП управляват достъпа до допълнителната памет в областта $0200 — $BFFF.

– Ключовете ЗАП80 и СТР2, управляващи достъпа до екранната памет, имат приоритет пред ключовете ЧДП и ЗДП. Това означава, че ако ключ ЗАП80 е включен, независимо от състоянието на ключове ЧДП и ЗДП ключ СТР2 ще разрешава достъпа до тази част от екранната памет, която се намира в допълнителната памет.

– Достъпът за четене и запис до областите $0000 — $01FF и $D000 — $FFFF от допълнителната памет се разрешава от ключ АНС.

Таблица 3.3

Адреси, свързани с управлението на допълнителната памет

Адрес	Име	Функция
1	2	3
$C000	ЗАП80 ИЗКЛ	Операция запис на този адрес изключва ключ ЗАП80
$C001	ЗАП80 ВКЛ	Операция запис на този адрес включва ключ ЗАП80
$C002	ЧСП	ЧЕТЕНЕ от СИСТЕМНАТА ПАМЕТ. Операция запис на този адрес разрешава четенето от системната оперативна памет
$C003	ЧДП	ЧЕТЕНЕ от ДОПЪЛНИТЕЛНАТА ПАМЕТ. Операция запис на този адрес разрешава четенето от допълнителната оперативна памет
$C004	ЗСП	ЗАПИС в СИСТЕМНАТА ПАМЕТ. Операция запис на този адрес разрешава записа в системната оперативна памет
$C005	ЗДП	ЗАПИС в ДОПЪЛНИТЕЛНАТА ПАМЕТ. Операция запис на този адрес разрешава записа в допълнителната оперативна памет
$C008	НС	НУЛЕВА СТРАНИЦА. Операция запис на този адрес разрешава достъпа до нулевата страница и до стека на системната оперативна памет
$C009	АНС	АЛТЕРНАТИВНА НУЛЕВА СТРАНИЦА. Операция запис на този адрес разрешава достъпа до нулевата страница и до стека на допълнителната оперативна памет
$C013	ЧЕТЕНЕ КЛЮЧ ЧДП	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ ЧДП
$C014	ЧЕТЕНЕ КЛЮЧ ЗДП	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ ЗДП
$C016	ЧЕТЕНЕ КЛЮЧ АНС	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ АНС
$C018	ЧЕТЕНЕ КЛЮЧ ЗАП80	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ ЗАП80

1	2	3
$C01C	ЧЕТЕНЕ КЛЮЧ СТР2	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ СТР2
$C01D	ЧЕТЕНЕ КЛЮЧ ГР.ВРС	Старшият бит (D7) на байта, прочетен от този адрес, показва състоянието на ключ ГР.ВРС
$C054	СТР2 ИЗКЛ	Обръщението към този адрес нулира ключ СТР2
$C055	СТР2 ВКЛ	Обръщението към този адрес установява ключ СТР2. Когато ключ ЗАП80 е включен, ключ СТР2 избира вместо втора страница от системната памет първа страница от допълнителната памет
$C056	ГР.ВРС ИЗКЛ	Обръщението към този адрес нулира ключ ГР.ВРС
$C057	ГР.ВРС ВКЛ	Обръщението към този адрес установява ключ ГР.ВРС

Глава 4. Работа с програмата МОНИТОР

Управляващата програма МОНИТОР е част от резидентното програмно осигуряване на компютъра Правец–8А. Тя е записана в неговата постоянна памет и е разположена в адресните области $F800 — $FFFF и $C100 — $CFFF. Програмата МОНИТОР е написана на машинен език и е съставена от множество подпрограми, които изпълняват разнообразни функции, свързани с управлението на компютъра на най–ниско ниво. Основното предназначение на подпрограмите, включени в програмата МОНИТОР, е да осигурят стандартен интерфейс между БЕЙСИК и другите езици от високо ниво и вградените входно–изходни устройства при въвеждането на символи от клавиатурата и извеждането им на екрана в режими ТЕКСТ40 и ТЕКСТ80, при поддържане на графика с малка разделителна способност и т.н. Тези подпрограми са добре документирани и са толкова компактни и бързи, че се използват винаги когато това е възможно. Така например резидентната програма интерпретатор на БЕЙСИК, операционните системи ДОС3.3 и ПроДОС и множество системни и приложни програми ги използват директно, като извършват обръщение към началните им адреси. Програмистът на БЕЙСИК рядко изпитва остра необходимост от директна работа с програмата МОНИТОР. Но тя поддържа някои специфични функции, които са незаменими при програмирането на машинен език, при създаването на таблици на фигури за графиката с голяма разделителна способност, при търсене и отстраняване на повреди в компютъра и др.

В т.нар. мониторен режим компютърът работи под управлението на програмата МОНИТОР и по–точно на една от нейните подпрограми, наречена *команден процесор*. Тази подпрограма организира приемането, интерпретирането и изпълняването на командите на програмата МОНИТОР, които се разглеждат в настоящата глава.

4.1. Влизане в програмата МОНИТОР

При включване на компютрите от фамилията Правец те преминават под управлението на интерпретатора на БЕЙСИК. Предаването на управлението на програмата МОНИТОР или т.нар. влизане в МОНИТОР става с командата CALL от БЕЙСИК за извикване на програма на машинен език. Като параметър на тази команда се задава началният адрес на програмата МОНИТОР в десетичен (–151 или 65385) или шестнайсетичен вид ($FF69). Командата за влизане в програмата МОНИТОР се включва в програма на БЕЙСИК или се въвежда директно от клавиатурата. След нейното изпълнение оповестяващият символ на БЕЙСИК (]) се заменя от оповестяващия символ на

програмата МОНИТОР (*), а вдясно от него се появява мигащият курсор.

В персоналния компютър Правец–8А има още един начин за преминаване в МОНИТОР. Това става с едновременното натискане на клавиши CONTROL, RESET и F2.

Предаването на управлението обратно на интерпретатора на БЕЙСИК (връщането в БЕЙСИК) може да стане по един от следните начини (приемаме, че се работи с резидентния БЕЙСИК и че интерпретаторът на ЦЕЛОЧИСЛЕН БЕЙСИК не е зареден в паметта на компютъра):

– С едновременното натискане на клавиши CONTROL и RESET. Този метод е неприложим, ако RESET векторът е модифициран.

– С управляващата последователност CTRL–C и RETURN. Връщането в БЕЙСИК не оказва влияние на програмата, написана на БЕЙСИК, която се намира в паметта на компютъра.

– С управляващата последователност CTRL–B и RETURN. Връщането в БЕЙСИК е съпроводено с изтриване на програмата, написана на БЕЙСИК, която се намира в паметта на компютъра.

– С мониторната команда 3D0G. Тази команда предава управлението на адрес $3D0, където операционните системи ДОС3.3 и ПроДОС записват инструкция за преход (JMP) към резидентния интерпретатор на БЕЙСИК. Следователно този метод е приложим само ако в компютъра е заредена една от цитираните две операционни системи.

4.2. Формат на командите на програмата МОНИТОР

Командите на програмата МОНИТОР подобно на програмните редове и командите на БЕЙСИК се въвеждат от клавиатурата. Един програмен ред на програмата МОНИТОР може да има до 254 символа и винаги завършва с управляващия символ CR (код $8D). Той може да съдържа три различни типа информация: адреси, данни и команди. Адресите и данните се задават в шестнайсетичен код. Данните са еднобайтови и се представят с двуразредни шестнайсетични числа, т.е. те могат да имат стойност от $00 до $FF. Адресите са двубайтови, изменят се от $0000 до $FFFF и обикновено се представят с четириразредни шестнайсетични числа. Командите на програмата МОНИТОР се състоят от един команден символ. Това може да бъде главна буква от латинската азбука, специален или управляващ символ.

Внимание! В рамките на настоящото изложение е прието параметрите на командите на програмата МОНИТОР и на интерпретатора на БЕЙСИК, чиято стойност се задава от програмиста (съгласно изискванията на конкретната програма), да се означават с малки

букви от кирилицата (вж. по-долу). Когато имената на тези параметри са използвани в пояснителен текст, те са заградени с триъгълни скоби < >.

4.3. Извеждане на съдържанието на паметта

Формат: [*адрес_1*][*.адрес_2*]

На екрана в шестнайсетичен вид се извежда съдържанието на паметта от началния адрес – <адрес_1> до крайния адрес – <адрес_2> включително. Изображението е разделено на 9 колони. В първата колона са т.нар. базови адреси (началният адрес и всички кратни на осем адреси, по-големи от него и по-малки или равни на крайния адрес). В останалите осем колони е съдържанието на съответните клетки от паметта. В зависимост от конкретните стойности на началния и крайния адрес първият и последният ред могат да бъдат непълни. С тази команда е възможно да се провери съдържанието на една клетка от паметта, на група от осем последователни клетки или на блок с неограничена дължина (в този и следващите примери на гл.4 данните и командите, въвеждани в компютъра, са показани с получерен шрифт).

```
*FCCB.FCF3

FCCB- FC 20 1A FC 20
FCD0- D0 F8 20 53 F9 84 3B 85
FCD8- 3A 38 B0 0D 98 AA 20 4A
FCE0- F9 A9 DE 20 ED FD 20 3A
FCE8- FF A9 A1 85 33 20 67 FD
FCF0- A0 01 20 CD
*
```

Ако в командата за извеждане на съдържанието на паметта е изпуснат началният адрес, програмата МОНИТОР приема, че той е равен на последния адрес, чието съдържание е било извеждано на текущото изходно устройство (т.нар. последен прочетен адрес), плюс единица.

```
*.FD00

FCF4- FE 90 D2 AA
FCF8- F0 E2 60 A0 03 20 CD FE
FD00- 4C
*
```

Ако не се зададат нито началният, нито крайният адрес, а само се натисне клавиш RETURN, програмата МОНИТОР извежда съдържа-

ието на паметта от последния прочетен адрес плюс единица до първия кратен на осем адрес минус единица. Ако клавиш RETURN се натисне още веднъж, извежда се съдържанието на следващите осем адреса.

```
*RETURN

  41 F9 FF FF FF FF FF

*RETURN

FD08- FF FF FF FF A4 24 B1 28
*
```

Ако е изпуснат крайният адрес или ако той е по-малък или равен на началния, извежда се само съдържанието на началния адрес.

```
*FFCF.FEC5

FECF- 2C
*
```

4.4. Промяна на съдържанието на паметта

Формат: [*адрес*]:*стойност* [{ *стойност*}]

В програмата МОНИТОР има функция, която позволява да се променя съдържанието на отделни клетки или на последователност от клетки от паметта на компютъра. Това става, като след адреса на първата клетка, чието съдържание ще се променя, се постави двоеточие и се въведат новите стойности, разделени с интервал. Заедно с проверката на съдържанието на паметта това е най-често използваната функция на програмата МОНИТОР. Характерно за тези две функции е, че след тяхното изпълнение се запомнят адресът на последната клетка, чието съдържание е извеждано на текущото изходно устройство, и адресът на последната клетка, чието съдържание е променяно, плюс единица. Ето защо при повторно изпълнение на функцията за промяна на съдържанието на паметта началният адрес може да се изпусне. Например

```
*3F5:4C

*3F6:69 FF

*:4C D0 03
```

```
*3F5.3FA

3F5- 4C 69 FF
3F8- 4C D0 03
*
```

Както ще видим по-късно, на адреси $3F5 и $3F8 се записват векторите на командата & от БЕЙСИК и на командата CTRL-Y на програмата МОНИТОР. Затова разгледаният пример представлява кратка програма, която позволява предаването на управлението от интерпретатора на БЕЙСИК на програмата МОНИТОР да става с команда & вместо с команда CALL-151, а обратно (под управлението на операционните системи ДОС3.3 и ПроДОС) — с управляващата последователност CTRL-Y.

4.5. Копиране на съдържанието на област от паметта

Формат: адрес 3<адрес_1.адрес_2M

Съдържанието на областта от паметта на компютъра с начален адрес *<адрес_1>* и краен адрес *<адрес_2>* се копира (премества) в областта с начален адрес *<адрес_3>*. Данните в изходната област не се променят. Разгледайте следната последователност от команди на програмата МОНИТОР:

1. От начален адрес $5000 се записва кратка програма на машинен език:

```
*5000:A9 00 85 3C 85 42 A9 F8 85 3D 85
43 A9 FF 85 3E 85 3F 20 2C FE
*
```

2. Проверява се дали програмата е записана правилно:

```
*5000.5014

5000- A9 00 85 3C 85 42 A9 F8
5008- 85 3D 85 43 A9 FF 85 3E
5010- 85 3F 20 2C FE
*
```

3. Програмата се премества на нов адрес ($306):

```
*306<5000.5014M
*
```

4. Проверява се резултатът от изпълнението на функцията за копиране.

```
*306.31A

0306- A9 00
0308- 85 3C 85 42 A9 F8 85 3D
0310- 85 43 A9 FF 85 3E 85 3F
0318- 20 2C FE
*
```

5. Проверява се има ли изменения в оригиналната програма:

```
*5000.5014

5000- A9 00 85 3C 85 42 A9 F8
5008- 85 3D 85 43 A9 FF 85 3E
5010- 85 3F 20 2C FE
*
```

Ако в командата за копиране крайният адрес < *адрес_2* > е по-малък или равен на началния < *адрес_1* >, копира се съдържанието само на една клетка:

```
*306<5014.5000M

*306.31A

0306- FE 00
0308- 85 3C 85 42 A9 F8 85 3D
0310- 85 43 A9 FF 85 3E 85 3F
0318- 20 2C FE
*
```

Копирането започва от най-малкия адрес. Ето защо, ако двете области се припокриват, се променя съдържанието и на изходната област. Това се използва за запълване на дадена област с определен байт или последователност от байтове.

```
*2000:FF

*2001<2000.201EM

*2000.201F

2000- FF FF FF FF FF FF FF FF
2008- FF FF FF FF FF FF FF FF
2010- FF FF FF FF FF FF FF FF
2018- FF FF FF FF FF FF FF FF
*
```

или:

```
*2000:AA 55

*2002<2000.201DM

*2000.201F

2000- AA 55 AA 55 AA 55 AA 55
2008- AA 55 AA 55 AA 55 AA 55
2010- AA 55 AA 55 AA 55 AA 55
2018- AA 55 AA 55 AA 55 AA 55
*
```

Запълването на област от паметта с определен байт или последователност от байтове има следните особености:

1. В началния адрес на зададената област се записва байтът или комбинацията от байтове, с които тя ще се запълва.

2. Началният адрес на областта, която се запълва, трябва да е по-голям от действителния начален адрес с броя на байтовете от използваната комбинация.

3. Крайният адрес на изходната област е по-малък от действителния краен адрес с броя на байтовете от използваната комбинация.

Копирането на области от паметта се използва често за преместване на текстови или графични изображения от едната страница в другата, при работа с оперативната памет в областта $D000 — $FFFF, при използване на компютъра като развойна система за кодиране, прекодиране и за програмиране на постоянни памети и т.н. Така копирането на частта от програмата МОНИТОР, намираща се на адреси $F800 — $FFFF, в оперативната памет може да стане със следната последователност от команди (за значението на адреси $C081 и $C083 вж. гл. 3):

```
*C081

C081- A0
*C081

C081- A0
*F800<F800.FFFFM

*C083

C083- C6
*C083

C083- A0
*
```

След изпълнението на тези команди резидентната програма МОНИТОР и интерпретаторът на БЕЙСИК се изключват, разрешават се операции четене и запис от оперативната памет в областта $D000 — $FFFF и компютърът преминава под управление на записаното в оперативната памет копие на програмата МОНИТОР.

4.6. Сравняване на съдържанието на две области от паметта

Формат: адрес 3<адрес_1.адрес_2V

Съдържанието на областта, определена с началния и крайния си адрес (*<адрес_1>* и *<адрес_2>* съответно), се сравнява със съдържанието на област със същия обем, зададена с началния си адрес *<адрес_3>*. Ако се открият разлики, на текущото изходно устройство се извеждат съответните адреси и тяхното съдържание. В първата колона са адресите от изходната област, във втората колона — съдържанието на тези адреси, а в третата колона — съдържанието на съответния адрес от областта, определена с *<адрес_3>*. Например

***306<5000.5014V**

5000– A9 (FE)
*

Сравняването на двете области започва от началния адрес на изходната област и продължава до крайния ѝ адрес. Обърнете внимание, че при откриване на разлика се отпечатват адресът и неговото съдържание от изходната област, а съдържанието на съответния адрес от областта, с която се извършва сравнението, се дава в скоби.

4.7. Извеждане и промяна на съдържанието на регистрите на микропроцесора

Формат: CTRL–E

Когато компютърът работи, съдържанието на регистрите на микропроцесора се изменя непрекъснато независимо от това, коя е управляващата програма. Ето защо неправилно е да се смята, че мониторната команда CTRL–E показва моментното съдържание на регистрите на микропроцесора. В действителност тя показва какво е било тяхното съдържание, когато машинната програма е спряла при изпълнение на инструкция BRK. При изпълнение на тази инструк-

ция програмата МОНИТОР записва съдържанието на регистрите на микропроцесора (в следния ред: A, X, Y, P, S) в пет последователни клетки, започвайки от адрес $45. При изпълнение на команда G, с която управлението се предава на програма на машинен език, програмата МОНИТОР зарежда в същия ред регистрите на микропроцесора и едва тогава предава управлението на първата инструкция на тази програма. В този смисъл командата CTRL–E позволява да се провери, респ. да се измени съдържанието на петте последователни адреса, започвайки от адрес $45, а действителното изменение на съдържанието на регистрите на микропроцесора става едва при изпълнение на команда G.

```
*CTRL–E
A=98 X=86 Y=D8 P=00 S=B7
*:10 20 30 40
*CTRL–E
A=10 X=20 Y=30 P=40 S=B7
*
```

4.8. Изпълнение на програма на машинен език

Формат: адрес G

Командата G зарежда регистрите на микропроцесора със съдържанието на петте последователни клетки, започващи от адрес $45, и с инструкция JSR предава управлението на първата инструкция от програмата.

Като пример въведете и изпълнете дадената тук програма. Тя използва подпрограмата, започваща от адрес $FE2C, за да изкопира съдържанието на областта от паметта с начален адрес, записан в клетки $3C и $3D, и краен адрес, записан в клетки $3E и $3F, в областта, чийто начален адрес е записан в клетки $42 и $43. В резултат съдържанието на първа графична страница се копира във втора графична страница.

```
*300:A9 00 85 3C 85 42 A9 20 85 3D A9 FF
85 3E A9 FF 85 3F A9 40 85 43 20 2C FE 60
*300G
*
```

4.9. Извикване на интерпретатора на БЕЙСИК без инициализация

Формат: CTRL–C

Тази команда предава управлението на интерпретатора на БЕЙСИК, без да изтрива програмата, написана на БЕЙСИК, от паметта на компютъра. Нейното изпълнение е еквивалентно на изпълнението на команда E003G, както и на изпълнението на команда 3D0G, когато в компютъра е заредена операционната система ДОС3.3 или ПроДОС. В този случай преминаването в БЕЙСИК става през операционната система.

4.10. Извикване на интерпретатора на БЕЙСИК с инициализация

Формат: CTRL–B

Тази команда предава управлението на интерпретатора на БЕЙСИК с инициализация. Това означава, че ако в паметта на компютъра има програма, написана на БЕЙСИК, тя се изтрива. Изпълнението ѝ е еквивалентно на изпълнението на команда E000G.

4.11. Шестнайсетично събиране и изваждане

Формат: стойност_1 + стойност_2
стойност_1 – стойност_2

Тези команди дават възможност за събиране и изваждане на еднобайтови шестнайсетични числа, като се пренебрегва преносът, респ. заемът. Например:

```
*15+2
=17
*18+9
=21
*FA+A
=4
*33–4
=2F
*5–A
=FD
*
```

Използват се при работа с програмата МИНИАСЕМБЛЕР за изчисляване на операндите на инструкциите за преход.

4.12. Избор на входно устройство

Формат: nCTRL–K

Тази команда е еквивалентна на командата от БЕЙСИК IN#n, където **n** = 0–7 е номерът на съответния съединител. Тя свързва програмния вход на компютъра към входното устройство, чийто контролер е включен в зададения съединител. При изпълнението на командата на адреси $38 и $39 се записва началният адрес на драйверната програма, обслужваща съответното входно устройство. Ако **n** > 0, на адрес $38 се записва $00, а на адрес $39 — $C0 + **n**. Клавиатурата (основното входно устройство) се свързва към програмния вход на компютъра с команда 0CTRL–K. В този случай на адрес $38 се записва $1B, а на адрес $39 — $FD. Тези два байта са началният адрес на подпрограмата KEYIN от програмата МОНИТОР, която обслужва клавиатурата (вж. гл. 5).

4.13. Избор на изходно устройство

Формат: nCTRL–P

Тази команда е еквивалентна на командата от БЕЙСИК PR#n. Тя пренасочва изходния поток (свързва програмния изход) на компютъра към изходното устройство, чийто контролер е включен в зададения съединител. При нейното изпълнение на адреси $36 и $37 се записва началният адрес на драйверната програма, обслужваща съответното изходно устройство. Ако **n** > 0, на адрес $36 се записва $00, а на адрес $37 — $C0 + **n**. Основното изходно устройство (видеомонитора) се свързва към програмния изход на компютъра с команда 0CTRL–P, при което на адрес $36 се записва $F0, а на адрес $37 — $FD. Това е началният адрес на подпрограмата COUT1, която обслужва екрана.

4.14. Потребителска команда

Формат: CTRL–Y

С тази команда управлението се предава на програмата с начален адрес $3F8, т.е. тя е равносилна на командата $3F8G. Нормално на този адрес се записва инструкция за преход (JMP) към началото на същинската програма. Това може да бъде някоя от подпрограмите на резидентното програмно осигуряване, на операционните системи ДОС3.3 и ПроДОС или потребителска програма на машинен език. Един прост пример е изпълнението на командата на дисковата операционна система CATALOG от програмата МОНИТОР.

***3F8: 4C 6E A5**

4.15. Преобразуване на шестнайсетични числа в десетични

Формат: шестнайсетично число **$**

Командата преобразува четирибайтови шестнайсетични числа в десетичния им еквивалент. Не съществува в компютрите Правец-82 и в Правец-8М.

***FF$**

@00255
***A$**

@00010
***300$**

@00768
*

4.16. Преобразуване на десетични числа в шестнайсетични

Формат: @десетично число

Командата @ преобразува десетичните числа от 0 до 65535 в шестнайсетични. Ако десетичното число е по-голямо от 65535, полученото шестнайсетично число е равно на MOD 65535 от десетичното. Такава команда не съществува в Правец-82 и Правец-8М.

***@13**

$000D

***@65535**

$FFFF

***@65546**

$000B
*

4.17. Дезасемблиране на съдържанието на паметта

Формат: [*адрес*]L

Командата L дезасемблира 20 инструкции, започвайки от зададения адрес. Инструкциите се представят с трибуквен мнемоничен код. Невалидните кодове на инструкции се отбелязват с три въпросителни знака (???). Относителните адреси в инструкциите за преход се преобразуват в абсолютни. В резултат на това различните видове адресиране се означават в получения листинг по следния начин:

Непосредствено	#$xx
Абсолютно	$xxxx
Относително	$xxxx
Индиректно	($xxxx)
Индексно	$xxxx,X
	$xxxx,Y
Индексно–индиректно	($xx,X)
Индиректно–индексно	($xx),Y
Нулева страница	$xx
Нулева страница по X	$xx,X
Нулева страница по Y	$xx,Y

Повторното въвеждане на команда L дезасемблира следващите 20 инструкции, без да е необходимо отново да се задава началният адрес. На един команден ред от програмата МОНИТОР може да се въведе повече от една команда L.

Като пример нека разгледаме една по–сложна програма, която симулира оператора от ЦЕЛОЧИСЛЕН БЕЙСИК — AUTO, в РАЗШИРЕН БЕЙСИК. Тя се изпълнява с команда CALL $300. След като се въведе номерът на първия ред от програмата на БЕЙСИК, следващите редове автоматично получават номер с 10 по–голям от предишния. От програмата се излиза с управляващата последователност CTRL–X, а автоматичното извеждане на номерата на програмните редове може да се възобнови с команда CALL $305.

Въведете програмата:

```
*300:A9 00 8D 7A 03 A9 10 85 38 A9 03 85
39 4C EA 03 48 AD 7A 03 F0 17 68 91 28
BD 7B 03 E0 05 30 04 EE 7A 03 60 C9 B0

*:30 3C C9 BA 10 38 60 68 20 1B FD C9 95
D0 02 B1 28 E0 05 10 03 9D 7B 03 C9 98
F0 25 C9 8D D0 1D 8A 48 A2 03 FE 7B 03
```

```
*:BD 7B 03 C9 BA D0 08 A9 B0 9D 7B 03 CA
10 EE 68 AA A9 8D CE 7A 03 60 20 2D FF
A9 FF 8D 7A 03 A9 1B 85 38 A9 FD 85 39
20 EA 03 A9 98 60 FF B0 B0 B0 B1 B0 A0

*
```

Дезасемблирайте програмата, като обърнете внимание на различните видове адресиране и на инструкциите за преход. Отбележете и това, че с команда L не може да се фиксира крайният адрес на дезасемблираната област.

```
*300L

0300-  A9 00     LDA   #$00
0302-  8D 7A 03  STA   $037A
0305-  A9 10     LDA   #$10
0307-  85 38     STA   $38
0309-  A9 03     LDA   #$03
030B-  85 39     STA   $39
030D-  4C EA 03  JMP   $03EA
0310-  48        PHA
0311-  AD 7A 03  LDA   $037A
0314-  F0 17     BEQ   $032D
0316-  68        PLA
0317-  91 28     STA   ($28),Y
0319-  BD 7B 03  LDA   $037B,X
031C-  E0 05     CPX   #$05
031E-  30 04     BMI   $0324
0320-  EE 7A 03  INC   $037A
0323-  60        RTS
0324-  C9 B0     CMP   #$B0
0326-  30 3C     BMI   $0364
0328-  C9 BA     CMP   #$BA

*L

032A-  10 38     BPL   $0364
032C-  60        RTS
032D-  68        PLA
032E-  20 1B FD  JSR   $FD1B
0331-  C9 95     CMP   #$95
0333-  D0 02     BNE   $0337
0335-  B1 28     LDA   ($28),Y
0337-  E0 05     CPX   #$05
0339-  10 03     BPL   $033E
```

```
033B-  9D 7B 03   STA   $037B,X
033E-  C9 98      CMP   #$98
0340-  F0 25      BEQ   $0367
0342-  C9 8D      CMP   #$8D
0344-  D0 1D      BNE   $0363
0346-  8A         TXA
0347-  48         PHA
0348-  A2 03      LDX   #$03
034A-  FE 7B 03   INC   $037B,X
034D-  BD 7B 03   LDA   $037B,X
0350-  C9 BA      CMP   #$BA
```

*LL

```
0352-  D0 08      BNE   $035C
0354-  A9 B0      LDA   #$B0
0356-  9D 7B 03   STA   $037B,X
0359-  CA         DEX
035A-  10 EE      BPL   $034A
035C-  68         PLA
035D-  AA         TAX
035E-  A9 8D      LDA   #$8D
0360-  CE 7A 03   DEC   $037A
0363-  60         RTS
0364-  20 2D FF   JSR   $FF2D
0367-  A9 FF      LDA   #$FF
0369-  8D 7A 03   STA   $037A
036C-  A9 1B      LDA   #$1B
036E-  85 38      STA   $38
0370-  A9 FD      LDA   #$FD
0372-  85 39      STA   $39
0374-  20 EA 03   JSR   $03EA
0377-  A9 98      LDA   #$98
0379-  60         RTS
037A-  FF         ???
037B-  B0 B0      BCS   $032D
037D-  B0 B1      BCS   $0330
037F-  B0 A0      BCS   $0321
0381-  00         BRK
0382-  00         BRK
0383-  00         BRK
0384-  00         BRK
0385-  00         BRK
0386-  00         BRK
0387-  00         BRK
0388-  00         BRK
```

```
0389-  00        BRK
038A-  00        BRK
038B-  00        BRK
038C-  00        BRK
038D-  00        BRK
038E-  00        BRK
038F-  00        BRK
0390-  00        BRK
*
```

4.18. МИНИАСЕМБЛЕР

Програмата МОНИТОР на компютъра Правец–8А съдържа програма, наречена МИНИАСЕМБЛЕР. Тя дава възможност направо от клавиатурата да се въвеждат програми на машинен език, като се използват мнемоничните кодове на инструкциите на микропроцесора (вж. т. 4.17). За разлика от "пълнокръвните" асемблиращи програми, каквито са MERLIN, BIG MAC и др., в МИНИАСЕМБЛЕР не могат да се използват етикети и коментари, а само действителни адреси и стойности. При това асемблирането на програмата се извършва непосредствено след въвеждането на всеки ред.

Влизането в МИНИАСЕМБЛЕР става с команда T. Оповестяващият символ на програмата МОНИТОР (*) се заменя от оповестяващия символ на МИНИАСЕМБЛЕР (!).

Излизането от МИНИАСЕМБЛЕР може да стане по няколко начина. Така например връщането в програмата МОНИТОР може да стане с команда $ или с едновременното натискане на клавиши CONTROL, RESET и F2, а преминаването в БЕЙСИК — с едновременното натискане на клавиши CONTROL и RESET.

Форматът на командата в МИНИАСЕМБЛЕР е

[*адрес*:|] *мнемоничен_код операнд*

Въвеждането на програма на машинен език с помощта на МИНИАСЕМБЛЕР започва с установяване на програмния брояч. Ето защо още на първия команден ред на МИНИАСЕМБЛЕР задължително се задава началният адрес <*адрес*> на програмата. Този адрес съвпада с началото на асемблираната програма и еднозначно определя използваната област от паметта. Началният адрес се запомня от програмата МИНИАСЕМБЛЕР и ако асемблираната програма заема последователни адреси в паметта на компютъра, въвеждането на адресите на следващите инструкции не е необходимо. Достатъчно е вместо съответния адрес да се постави интервал. Въвеждането на програмата продължава инструкция по инструкция, като между адреса на инструкцията (ако има такъв) и нейния мнемоничен код се поста-

вя двоеточие, а между мнемоничния код и операнда — интервал. Командният ред завършва с RETURN, след което мнемоничният код на инструкцията се преобразува в шестнайсетичен код и заедно с операнда се записва в паметта на компютъра, започвайки от <*адрес*>. След това програмата МИНИАСЕМБЛЕР дезасемблира получения код, заменя с него въведения ред и извежда оповестяващия символ на следващия ред от екрана. Ако в командния ред има грешка, той се игнорира, а под символа, в който е открита грешката, се появява символът ^.

Работата с програмата МИНИАСЕМБЛЕР има следните особености:

1. Използват се само шестнайсетични числа. Поради това символът $, с който се означават шестнайсетичните операнди, не е задължителен. По тази причина инструкциите ADC $0305 и ADC 305 например са еквивалентни.

2. Непосредственото адресиране се различава от абсолютното по символа #. При него операндът съвпада с втория байт на инструкцията. Така например инструкцията LDA #$3F не означава регистър A да се зареди със стойността, записана на адрес $003F, а означава регистър A да се зареди със стойността $3F.

3. Относителното адресиране се преобразува в абсолютно. В инструкциите за преход, които използват относително адресиране, е необходимо да се зададе абсолютният адрес на прехода, който не трябва да се различава с повече от 127 от адреса на текущата инструкция.

Следващият пример показва как с помощта на програмата МИНИАСЕМБЛЕР в компютъра може да се въведе кратка програма на машинен език. При изпълнението ѝ екранът се изчиства и на него се появяват въведеният символ и неговият шестнайсетичен код.

```
*T

!300: JSR $FD35
0300-  20 35 FD   JSR    $FD35
! STA 340
0303-  8D 40 03   STA    $0340
! JSR FDED
0306-  20 ED FD   JSR    $FDED
! LDA #AD
0309-  A9 AD      LDA    #$AD
! JSR FDED
030B-  20 ED FD   JSR    $FDED
! LDA #A0
030E-  A9 A0      LDA    #$A0
! JSR FDED
0310-  20 ED FD   JSR    $FDED
```

```
! LDA 340
0313-  AD 40 03   LDA    $0340
! JSR FD DA
0316-  20 DA FD   JSR    $FDDA
! LDA #8D
0319-  A9 8D      LDA    #$8D
! JSR FDED
031B-  20 ED FD   JSR    $FDED
! BNE 300
031E-  D0 E0      BNE    $0300
!
```

В компютрите Правец–82 и Правец–8М програмата МИНИАСЕМБЛЕР не е част от резидентното програмно осигуряване. Тя е част от интерпретатора на ЦЕЛОЧИСЛЕН БЕЙСИК и може да се използва, когато той е зареден в компютъра и е активен.

4.19. Търсене на низ

Търсенето на низ с дължина до четири байта в паметта на компютъра се извършва на два етапа:

1. Задаване на низа. На петте последователни адреса $45–$49 се записват съответно дължината на низа минус единица и самият низ.
2. Търсене на низа.

Формат: адрес_1.адрес_2S

Зададеният низ се търси в областта, определена от началния адрес <*адрес_1*> и крайния адрес <*адрес_2*>. На екрана се извеждат адресите, на които е намерен търсеният низ. Например

```
*45: 1 00 C0
*D000.FFFFS

D859-
FB7D-
FB89-
*
```

Функцията за търсене на низ не съществува в компютрите Правец–82 и Правец–8М.

4.20. Управление на формата на текстовото изображение

В програмата МОНИТОР има две команди N и I, с които форматът на текстовото изображение се установява във формат НОРМАЛНО, респ. ИНВЕРСНО ВИДЕО. За да добиете представа за тяхното действие, изпълнете следната програма:

```
*300.31F I 300.31F N 300L
```

Глава 5. Подпрограми на програмата МОНИТОР

В предната глава бяха разгледани част от новите възможности на програмата МОНИТОР, свързани с използването ѝ в директен режим. Тук се разглеждат подпрограмите на програмата МОНИТОР от гледна точка на програмиста на АСЕМБЛЕР. Направената класификация е условна, защото много подпрограми са универсални и е трудно да се фиксира най-типичното им приложение. В скоби след всяка подпрограма е даден началният ѝ адрес.

Разликата в резидентното програмно осигуряване на компютъра Правец-8А, от една страна, и на компютрите Правец-82 и Правец-8М, от друга, е многопосочна. Това е резултат преди всичко от новата организация и новите схемни решения, намерили място в Правец-8А, за чието поддържане е необходимо и ново програмно осигуряване. Програмата МОНИТОР е разширена. Освен стандартната адресна област $F800 — $FFFF тя заема и част от областта $C100 — $CFFF, в която се намират подпрограмите за поддържане на 80-колонно изображение и за интерпретиране на новите команди на програмата МОНИТОР и на интерпретатора на БЕЙСИК. Това означава, че тази област се ползва не само за поддържане на режим ТЕКСТ80, но и когато в компютъра няма допълнителна памет. Достъпът до постоянната памет в областта $C100 — $CFFF е сравнително сложен, не защото няма апаратни средства за това, а защото е свързан с блокиране на областта на външните входно-изходни устройства. Това изисква в програмното осигуряване да се предвидят специални мерки за организация на използването на тази област. Влизането в подпрограмите или техните сегменти, разположени в областта $C100 — $CFFF, е унифицирано. То се осъществява през две входни точки GOTOXY (адрес $FBB4) и GOTOKOZE (адрес $FECD). И в двата случая регистър Y се използва за предаване на номера на изпълняваната функция (вж. табл. 5.1 и 5.2).

Таблица 5.1

Входна точка GOTOKOZE (адрес $FECD)

Параметър в регистър Y	Функция
1	Програма МИНИАСЕМБЛЕР
2	Преобразуване на шестнайсетични числа в десетични (HEX → DEC)
3	Преобразуване на десетични числа в шестнайсетични (DEC → HEX)

Таблица 5.2

Входна точка GOTOXY (адрес $FBB4)

Параметър в регистър Y	Функция
0	Сегменти от подпрограмата CLREOP
1	Сегменти от подпрограмата HOME
2	Сегменти от подпрограмата SCROLL
3	Сегменти от подпрограмата CLREOL
4	Сегменти от подпрограмата CLREOLZ
5	Сегменти от подпрограмата INIT/RESET
6	Сегменти от подпрограмата KEYIN
7	Сегменти от подпрограмата FIXESC
8	Сегменти от подпрограмата SETWND

5.1. Подпрограми за запазване и възстановяване на съдържанието на регистрите на микропроцесора

Повечето от подпрограмите на програмата МОНИТОР разрушават съдържанието на някой от регистрите на микропроцесора. Това налага честото използване на подпрограми за запазване и възстановяване на неговото състояние.

Подпрограма IOSAVE ($FF4A). Записва съдържанието на регистрите на микропроцесора в паметта на компютъра съгласно табл. 5.3.

Подпрограма IOREST ($FF3F). Възстановява съдържанието на регистрите на микропроцесора съгласно табл. 5.3.

Таблица 5.3

Регистри на микропроцесора

Адрес	Име	Предназначение
$45	ACC	Временно съхранява съдържанието на регистър A
$46	XREG	Временно съхранява съдържанието на регистър X
$47	YREG	Временно съхранява съдържанието на регистър Y
$48	STATUS	Временно съхранява съдържанието на програмния брояч
$49	SPNT	Временно съхранява съдържанието на указателя на стека

5.2. Стандартни входно–изходни вектори

Клетките от оперативната памет на компютъра, които се използват за предаване на управлението от една на друга подпрограми, се наричат вектори. Типичен пример за използването на вектори е организацията на стандартния вход и изход на компютъра. И наистина от листинга на програмата МОНИТОР лесно се установява, че първото нещо, което правят основните подпрограми за изчитване и въвеждане на символ — COUT и RDKEY, е индиректен преход по съдържанието на определени клетки от оперативната памет. Това са съответно изходният вектор CSW (адреси $36 — $37) и входният вектор KSW (адреси $38 — $39). Ролята на този преход става ясна, като се провери съдържанието на посочените клетки.

```
]Call-151
*36.37
*0036- F0 FD
*38.39
*0038- 1B FD
```

Така в режим ТЕКСТ40, когато се използват стандартните входно–изходни устройства (клавиатура и екран) и когато в компютъра няма заредена операционна система (ДОС3.3 или ПроДОС), изход-

ният вектор CSW сочи началния адрес ($FDF0) на подпрограмата за извеждане на един символ на екрана COUT1, а входният вектор KSW — началния адрес ($FD1B) на подпрограмата за обслужване на входа от клавиатурата KEYIN. Следователно извеждането на определен символ на екрана на компютъра може да стане, като кодът на този символ се зареди в регистър A и се извика подпрограмата COUT. Например

```
LDA  #$E3
JSR  $FDED
```

Естествен е въпросът, защо е необходимо да се извиква подпрограмата COUT, когато същият резултат ще се получи и ако се извиква направо подпрограмата COUT1?

```
LDA  #$E3
JSR  $FDF0
```

Отговорът на този въпрос е тясно свързан с идеята за използване на вектори. Представете си, че след като сте написали програма на АСЕМБЛЕР и с използването на подпрограмата COUT1 сте я ориентирали изцяло към екрана, трябва да пренасочите изхода на компютъра към друго устройство, например принтер. Този нагледен проблем ще наложи цялостна преработка на вашата програма, докато при използването на подпрограмата COUT пренасочването на програмния изход се осъществява съвсем просто, като във вектор CSW се запише началният адрес на съответния драйвер. Ако принтерът се управлява от интелигентен контролер (контролер, който има постоянна памет със собствена драйверна програма), то изходният вектор съдържа нейния начален адрес във вида Cn00, където **n** е номерът на съединителя. При това смяната на изходния вектор се извършва автоматично от команда PR# n, с която се инициализира контролерът. Следователно концепцията за използване на вектори е в основата на универсалността на входно–изходните подпрограми, защото позволява чрез смяна на драйверната програма стандартният вход и изход на компютъра да се пренасочват към различни входно–изходни устройства. Така например двата текстови режима TEXT40 и TEXT80 използват различни входно–изходни подпрограми. Ето защо при работа в режим TEXT80 векторът CSW сочи началния адрес ($C307) на подпрограмата BASICOUT, а векторът KSW — началния адрес ($C305) на подпрограмата BASICIN.

Трябва да се знае, че операционните системи ДОС3.3 и ПроДОС поддържат собствена връзка със стандартните входно–изходни програми. Ето защо под тяхно управление векторите CSW и KSW съдържат адресите на съответните входно–изходни подпрограми на операционната система. Така при работа с ДОС3.3 векторът CSW

съдържа $9EBD, а векторът KSW — $9E81. Съдържанието на съответните вектори при работа с ПроДОС е съответно $B84B и $B84E. За яснота на изложението по-нататък няма да се спираме на отражението на операционните системи върху работата на програмата МОНИТОР.

5.3. Изходни подпрограми

Подпрограма COUT ($FDED). Подпрограмата COUT е стандартната изходна подпрограма на програмата МОНИТОР. Тя е предназначена за извеждане на един символ към текущото изходно устройство и се извиква, след като кодът на символа се зареди в регистър А. Чрез изходния вектор CSW управлението се предава на текущата изходна програма. Най-често това е подпрограмата COUT1 или BASICOUT. Подпрограмата COUT се използва от почти всички изходни подпрограми на компютъра (вж. описанието на подпрограмите COUTZ, CROUT, CROUT1, PRERR, PRBLNK, PRBLK2, PRBYTE, PRHEX и т.н.). В този смисъл резидентното програмно осигуряване, обслужващо изхода на компютъра, може да се разглежда като програма с множество входни точки, тяло, в което се извършва обединяването на различните входове (подпрограмата COUT), и множество драйверни програми, които се свързват към тялото чрез изходния вектор CSW (вж. фиг. 5.1).

Подпрограма OUTPORT ($FE95). Подпрограмата OUTPORT зарежда в изходния вектор CSW началния адрес на съответната драйверна програма, който е от вида $Cn00 (n = 0 – 7). При n = 0 изходният вектор сочи подпрограмата SETVID. Преди да се извика подпрограмата OUTPORT, необходимо е в регистър А да се зареди номерът на съответния съединител. Съответства на командата PR# n на БЕЙСИК и на командата nCTRL-P на програмата МОНИТОР.

При изпълнението на подпрограмата OUTPORT се променя съдържанието на регистри А, X и Y.

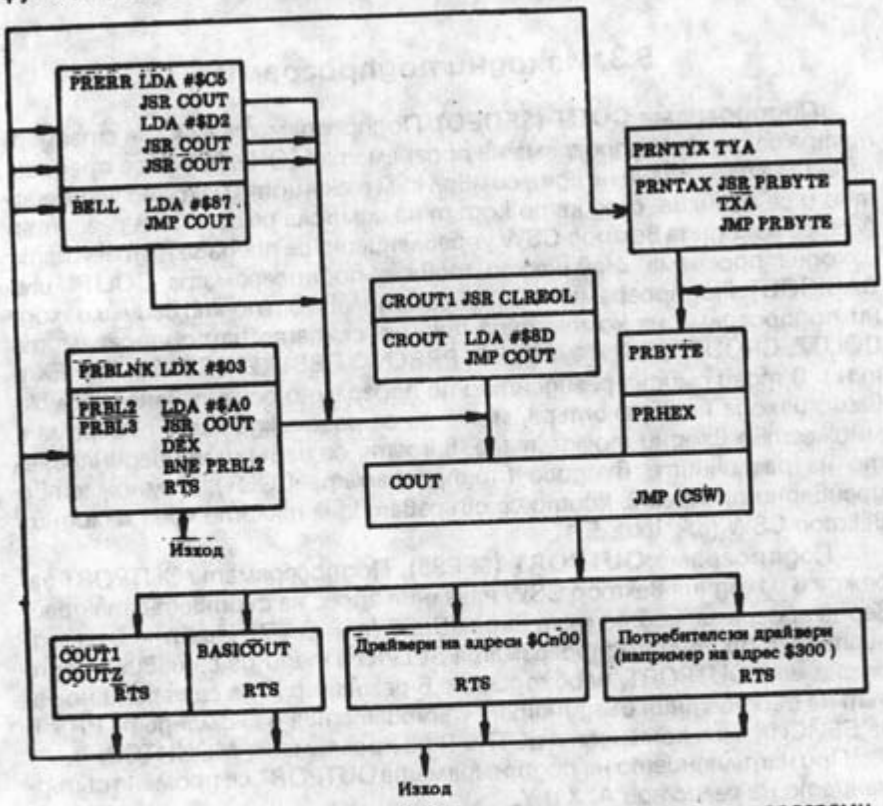
Подпрограма SETVID ($FE93). Подпрограмата SETVID зарежда изходния вектор CSW с началния адрес на подпрограмата COUT1. Нейното изпълнение съответства на изпълнението на командата PR# 0 на БЕЙСИК, на командата 0CTRL-P на програмата МОНИТОР и на последователността от инструкции:

```
LDA #$00
JSR OUTPORT
```

При изпълнението на подпрограмата SETVID се променя съдържанието на регистри А, X и Y.

Подпрограма COUT1 ($FDF0). Подпрограмата COUT1 е основната подпрограма за управление на изхода към екрана в режим

ТЕКСТ40. Тя се извиква, след като кодът на символа се зареди в регистър А. Ако този код е код на буква, цифра или специален символ, подпрограмата COUT1 го извежда на екрана. Ако акумулаторът е съдържал управляващ символ, тя или изпълнява някоя от специалните функции, описани в табл. 5.6, или игнорира управляващия символ.



Фиг 5.1. Опростена схема на връзките между изходните подпрограми

Символите се извеждат на текущата позиция на курсора, а той се измества с една позиция надясно. Ако курсорът е достигнал десния край на реда в рамките на текстовия прозорец, той се премества на най–лявата позиция от следващия ред. Ако курсорът е достигнал долния десен ъгъл на текстовия прозорец (последната позиция от последния ред), изображението се премества с един ред нагоре (т.нар. операция СКРОЛ) и курсорът се измества на най–лявата позиция на новия последен ред. За изчисляване на текущата позиция на курсора подпрограмата COUT1 използва съдържанието на клетките, дадени в табл. 5.4.

Таблица 5.4

Клетки, използвани от подпрограмата COUT1 за изчисляване на позицията на курсора

Адрес	Име	Значение
$20	WNDLFT	Хоризонтална позиция на лявата страна (началото) на текстовия прозорец
$21	WNDWDTH	Широчина на текстовия прозорец
$22	WNDTOP	Вертикална позиция на първия ред от текстовия прозорец
$23	WNDBTM	Вертикална позиция на последния ред от текстовия прозорец
$24	CH	Хоризонтална позиция на курсора
$25	CV	Вертикална позиция на курсора

Сумата от широчината на текстовия прозорец и хоризонталната позиция на лявата му страна не трябва да надхвърля широчината на екрана (40 или 80 колони). Освен това винаги когато се сменят размерите на текстовия прозорец, текущата позиция на курсора трябва да попада в новия прозорец. В противен случай е възможно изпращаните символи от подпрограмите COUT1, респ. BASICOUT, да попаднат извън екранната памет, а това може да разруши изпълняваната програма.

Допустимите стойности на параметрите на текстовия прозорец са дадени в табл. 5.5.

Таблица 5.5

Допустими параметри на текстовия прозорец

Параметър	Адрес HEX(DEC)	Нормално при 40/80 колони	Минимално	Максимално при 40/80 колони
WNDLFT	$20 (32)	0/0	0	39/79
WNDWDTH	$21 (33)	40/80	0	40/80
WNDTOP	$22 (34)	0/0	0	23/23
WNDBTM	$23 (35)	24/24	1	24/24

Подпрограмата COUT1 може да изобразява на екрана малки и главни букви от кирилицата и латиницата, цифрите от 0 до 9 и специ-

алните символи във формати НОРМАЛНО и ИНВЕРСНО ВИДЕО. По принцип съществува и формат МИГАЩО ВИДЕО, но използването му е съпроводено с ред ограничения и не се препоръчва (вж. гл. 2). Така например при работа със стандартното символно множество във формат МИГАЩО ВИДЕО вместо главни букви от кирилицата се получават мигащи специални символи и цифри, а вместо мигащи главни букви от латиницата алтернативно се появяват главни букви от кирилица и латиница в инверсен формат.

Таблица 5.6

Формати на режимите на изображение ТЕКСТ40 и ТЕКСТ80

Код на символа	Символно множество	
	Стандартно	Алтернативно
$00-$1F	Главни букви латиница в ИНВЕРСНО ВИДЕО	Главни букви латиница в ИНВЕРСНО ВИДЕО
$20-$3F	Цифри и специални символи в ИНВЕРСНО ВИДЕО	Цифри и специални символи в ИНВЕРСНО ВИДЕО
$40-$5F	Главни букви латиница в МИГАЩО ВИДЕО	Малки букви латиница в НОРМАЛНО ВИДЕО
$60-$7F	Цифри и специални символи в МИГАЩО ВИДЕО	Малки букви кирилица в НОРМАЛНО ВИДЕО
$80-$9F	Главни букви кирилица в ИНВЕРСНО ВИДЕО	Главни букви кирилица в ИНВЕРСНО ВИДЕО
$A0-$BF	Цифри и специални символи в НОРМАЛНО ВИДЕО	Цифри и специални символи в НОРМАЛНО ВИДЕО
$C0-$DF	Главни букви латиница в НОРМАЛНО ВИДЕО	Главни букви латиница в НОРМАЛНО ВИДЕО
$E0-$FF	Главни букви кирилица в НОРМАЛНО ВИДЕО	Главни букви кирилица в НОРМАЛНО ВИДЕО

Форматът на символа на екрана се определя както от използваното символно множество (стандартно или алтернативно), така и от състоянието на старшите два бита от неговия код (вж. табл. 5.6). Когато подпрограмата COUT1 се използва за изпращане на символ към екрана, тя маскира двата старши бита от байта на символа, като извършва логическа операция AND със стойността, записана на адрес $32. Това е т.нар. флаг INVFLG, който за формат НОРМАЛНО ВИДЕО съдържа $FF, за формат ИНВЕРСНО ВИДЕО — $3F, а за формат МИГАЩО ВИДЕО — $7F. Този флаг се установява с командите NORMAL, INVERSE и FLASH на БЕЙСИК, с командите N и I на програмата МОНИТОР или директно с подпрограмите SETNORM и SETINV.

Управляващите символи не се извеждат на екрана. Обикновено те се използват за стартиране на някоя от функциите на програмното осигуряване. Конкретно подпрограмата COUT1 разпознава само шест управляващи символа и игнорира всички останали (вж. табл. 5.7).

Таблица 5.7

Управляващи символи, разпознавани от подпрограма COUT1

Символ	Означение	Код	Действие
CTRL-C	ETX	$83	Разпознава се само в режим СТОП НА ЛИСТИНГА, т.е. след приемане на управляващия символ DC3. В този случай подпрограмата COUT1 изчаква следващия символ (като стробът на клавиатурата се нулира) и ако това не е управляващ символ, записва ETX в регистър A. Това позволява спирането на програмата, написана на БЕЙСИК, да става и в режим СТОП НА ЛИСТИНГА
CTRL-G	BEL	$87	Генерира тон с честота около 1000 Hz през високоговорителя в продължение на 0,1 s
CTRL-H	BS	$88	Премества курсора с една позиция наляво. Премества го в края на горния ред, ако се е намирал в началото на реда
CTRL-J	LF	$8A	Премества курсора с един ред надолу. Ако е необходимо, преди това премества изображението с един ред нагоре
CTRL-M	CR	$8D	Премества курсора в началото на следващия ред от текстовия прозорец. Ако е необходимо, преди това премества изображението с един ред нагоре
CTRL-S	DC3	$93	След като приеме управляващия символ CR, подпрограмата COUT1 проверява дали не е въведен и управляващият символ DC3. Ако го открие, спира, без да изпрати CR към екрана (режим СТОП НА ЛИСТИНГА) и не продължава, докато не се натисне друг клавиш. Тогава подпрограмата COUT1 изпраща задържания символ CR и продължава нормално. Функцията СТОП НА ЛИСТИНГА се изпълнява само в директен режим

Подпрограма BASICOUT ($C307). Подпрограмата BASICOUT е част от програмното осигуряване на компютъра, предназначено за поддържане на режим ТЕКСТ80 (80- или 40-колонно изображение). То-

ва е основната подпрограма за управление на изхода към екрана в този режим. По функции и по последователност на извършваните действия тя е напълно еквивалентна на подпрограмата COUT1.

От разликите между подпрограмите BASICOUT и COUT1 произтича и разликата между възможностите на двата текстови режима. И тъй като за реализирането на режим ТЕКСТ80 са необходими не само програмни, но и апаратни средства, подпрограмата BASICOUT трябва да се разглежда в контекста на конфигурацията на компютъра.

Режим ТЕКСТ80 е една от характерните особености на персоналния компютър Правец–8А. С помощта на допълнителни модули не е проблем да се осъществи 80-колонно изображение и в предишните модели на фамилията, но при Правец–8А това става на друг принцип. Без да се впускаме в излишни подробности, достатъчно е да знаем, че за реализирането на режим ТЕКСТ80 е необходимо в компютъра да има допълнителна оперативна памет, включена към съединител X0. Това може да бъде модул МЕГАРАМ (в произволна конфигурация) или специално разработен за целта модул. След включване на захранването програмното осигуряване на режим ТЕКСТ80 остава пасивно дори при наличие на такъв модул в компютъра. Установеният режим е ТЕКСТ40 и се работи със стандартното символно множество. Активирането на режим ТЕКСТ80, респ. предаването на управлението на програмното осигуряване, което го поддържа, става с команда PR# 3 от БЕЙСИК или с команда 3CTRL-P от програмата МОНИТОР. Обърнете внимание, че това са командите, с които би се активирал всеки модул, включен в съединител X3. Следователно модулите, включени в допълнителния съединител X0, имат приоритет пред модулите, включени в съединител X3. Това се гарантира с подходящо установяване на програмноуправляемите ключове за избор на постоянна памет в областта $C300 — $C3FF и е причина излизането от режим ТЕКСТ80 да не може да стане само с команда PR#0.

Внимание! Дезактивирането на програмното осигуряване, поддържащо режим ТЕКСТ80, не бива да става с команда PR# 0 или еквивалентната команда 0CTRL-P на програмата МОНИТОР, защото те не изключват съответните програмноуправляеми ключове. Вместо това в директен режим трябва да се използва управляващата последователност ESC CTRL–Q, а в програмен режим да се изпраща CTRL–U чрез подпрограмата COUT.

Три са основните различия между подпрограмите BASICOUT и COUT1:

1. Максималните стойности на параметрите на текстовия прозорец (вж. табл. 5.5).

2. Формат МИГАЩО ВИДЕО на текстовите режими. Тъй като в режим ТЕКСТ80 се използва само алтернативното символно множество, подпрограмата BASICOUT не поддържа формат МИГАЩО ВИ-

ДЕО и за разлика от подпрограмата COUT модифицира само най–старшия бит от кода на символа (вж. табл. 5.6).

3. Разпознаваните управляващи символи. Подобно на подпрограмата COUT1 и подпрограмата BASICOUT не отпечатва управляващите символи, но във връзка с по-големите възможности за редактиране разпознава значително по-голяма част от тях (сравнете табл. 5.7 и 5.8).

Таблица 5.8

Управляващи символи, разпознавани от подпрограма BASICOUT

Символ	Означение	Код	Действие
1	2	3	4
CTRL-C	ETX	$83	Разпознава се само в режим СТОП НА ЛИСТИНГА, т.е. след приемането на управляващия символ DC3. В този случай подпрограмата BASICOUT изчаква следващия символ и ако това не е управляващият символ ETX, нулира строба на клавиатурата. Ако изпълняваната програма е написана на БЕЙСИК, това позволява тя да бъде спряна и в режим СТОП НА ЛИСТИНГА
CTRL-G	BEL	$87	Генерира тон с честота около 1000 Hz през високоговорителя в продължение на 0,1 s
CTRL-H	BS	$88	Премества курсора с една позиция наляво. Ако той се намира в началото на реда, премества го на най–дясната позиция от горния ред
CTRL-J	LF	$8A	Премества курсора с един ред надолу. Ако е необходимо (курсорът е на последния ред), преди това премества изображението с един ред нагоре
CTRL-K	VT	$8B	Изчиства екрана от текущата позиция на курсора до края. Изпълнява се само в програмен режим
CTRL-L	FF	$8C	Премества курсора в горния ляв ъгъл на текстовия прозорец. Изчиства текстовия прозорец. Изпълнява се само в програмен режим
CTRL-M	CR	$8D	Премества курсора в началото на следващия ред на текстовия прозорец. Ако е необходимо, преди това премества изображението с един ред нагоре
CTRL-N	SO	$8E	Установява формат НОРМАЛНО ВИДЕО. Изпълнява се само в програмен режим
CTRL-O	SI	$8F	Установява формат ИНВЕРСНО ВИДЕО. Изпълнява се само в програмен режим

Продължение на таблица 5.8

1	2	3	4
CTRL-Q	DC1	$91	Установява режим ТЕКСТ40. Изпълнява се само в програмен режим
CTRL-R	DC2	$92	Установява режим ТЕКСТ80. Изпълнява се само в програмен режим
CTRL-S	DC3	$93	След като приеме управляващия символ CR, подпрограмата BASICOUT проверява дали не е въведен и управляващият символ DC3 и ако го открие, спира, без да изпрати CR към екрана (режим СТОП НА ЛИСТИНГА). Изпълнението на програмата продължава, когато се натисне друг клавиш. Едва тогава задържаният управляващ символ CR се изпраща към екрана. Функцията СТОП НА ЛИСТИНГА се изпълнява само в директен режим
CTRL-U	NAK	$95	Дезактивира програмното осигуряване, поддържащо режим ТЕКСТ80. Изпълнява се само в програмен режим
CTRL-V	SYN	$96	Премества изображението с един ред надолу на екрана. Не променя позицията на курсора. Изпълнява се само в програмен режим
CTRL-W	ETB	$97	Премества изображението с един ред нагоре на екрана. Не променя позицията на курсора. Изпълнява се само в програмен режим
CTRL-Y	EM	$99	Премества курсора в горния ляв ъгъл на текстовия прозорец. Изпълнява се само в програмен режим
CTRL-Z	SUB	$9A	Изчиства реда, на който се намира курсорът. Изпълнява се само в програмен режим
CTRL-\	FS	$9C	Премества курсора с една позиция надясно. Премества го в началото на следващия ред, ако той се намира в десния край на текстовия прозорец. Изпълнява се само в програмен режим
CTRL-]	GS	$9D	Изчиства реда от екрана от текущата позиция на курсора до десния край на текстовия прозорец. Изпълнява се само в програмен режим
CTRL-^	RS	$9E	Не се поддържа в БЕЙСИК. В ПАСКАЛ премества курсора на позиция, определена от стойността на следващите два байта минус 32

Подпрограма COUTZ ($FDF6). Подпрограмата COUTZ може да се разглежда като алтернативен вход в програмата COUT1. Тя се различава от подпрограмата COUT1 единствено по това, че игнорира съдържанието на флаг INVFLG.

Подпрограма CROUT ($FD8E). Изпраща управляващия символ CR към текущото изходно устройство. Зарежда $8D в регистър A и предава управлението на подпрограмата COUT.

След изпълнението на подпрограмата CROUT в регистър A има код $8D.

Подпрограма CROUT1 ($FD8B). Подпрограмата CROUT1 изчиства реда от текущата позиция на курсора до десния край на текстовия прозорец и изпраща управляващия символ CR към текущото изходно устройство. За целта извиква подпрограмата CLEOL, а след това предава управлението на подпрограмата CROUT. Изпълнението на подпрограмата CROUT1 променя съдържанието на регистри A и Y.

Подпрограма PRERR ($FF2D). Подпрограмата PRERR използва подпрограмата COUT, за да изпрати съобщението ERR и управляващия символ BEL към текущото изходно устройство. Тя има следния вид:

```
FF2D    PREER   LDA  #$C5 ;Изпрати символа E към изхода
                JSR  COUT
                LDA  #$D2 ;Изпрати два символа R към
                JSR  COUT ;изхода
                JSR  COUT
FF3A    BELL    LDA  #$87 ;Изпрати CTRL-G към изхода
                JMP  COUT
```

При изпълнението на подпрограмата PRERR се променя съдържанието на регистър A.

Подпрограма BELL ($FF3A). Подпрограмата BELL използва подпрограмата COUT, за да изпрати управляващия символ BEL към текущото изходно устройство. Тя започва от адрес $FF3A (вж. подпрограма PRERR). При изпълнението ѝ се променя съдържанието на регистър A.

Подпрограма PRBLNK ($F948). Подпрограмата PRBLNK изпраща три интервала (код $A0) към текущото изходно устройство чрез подпрограма COUT.

Изпълнението на подпрограмата PRBLNK разрушава съдържанието на регистри A и X. Обикновено записва $A0 в регистър A и $00 в регистър X.

Подпрограма PRBL2 ($F94A). Подпрограмата PRBL2 изпраща от 1 до 256 интервала към текущото изходно устройство. Това е алтернативна входна точка на подпрограмата PRBLNK. Тя се извиква, след като броят на интервалите се зареди в регистър X ($1 — $FF). Например:

```
LDX #$30
JSR PRBL2
```

Когато X = 0, броят на интервалите, изпратени към текущото изходно устройство, е 256.

Изпълнението на подпрограмата PRBL2 разрушава съдържанието на регистри A и X. Обикновено записва $A0 в регистър A и $00 в регистър X.

Подпрограма PRBYTE ($FDDA). Подпрограмата PRBYTE извежда съдържанието на регистър A като двуразредно шестнайсетично число на текущото изходно устройство. Извиква се, след като числото се зареди в регистър A. Изпълнението ѝ променя съдържанието на регистър A.

Подпрограма PRHEX ($FDE3). Подпрограмата PRHEX извежда младшите четири бита от съдържанието на регистър A като едноразредно шестнайсетично число. Тя е алтернативен вход на подпрограмата PRBYTE. При изпълнението ѝ се променя съдържанието на регистър A.

Подпрограма PRNTAX ($F941). Като използва подпрограмата PRBYTE, подпрограмата PRNTAX извежда съдържанието на регистри A и X като четириразредно шестнайсетично число (старшите два разреда са в регистър A, а младшите два — в регистър X). При нейното изпълнение се променя съдържанието на регистър A.

Подпрограма PRNTYX ($F940). Подпрограмата PRNTYX е алтернативна входна точка в подпрограмата PRNTAX. Тя извежда съдържанието на регистри Y и X като четириразредно шестнайсетично число. Старшите два разреда са в регистър Y, а младшите два — в регистър X. Изпълнението ѝ променя съдържанието на регистър A.

5.4. Входни подпрограми

Обща, обединяваща всички останали входни подпрограми на програмата МОНИТОР, е подпрограмата RDKEY. Подобно на подпрограмата COUT и тя използва вектор за предаване на управлението на различни входни устройства. Този вектор се нарича стандартен входен вектор, означава се с KSW и се намира на адреси $38 — $39. В зависимост от режима на изображение подпрограмата RDKEY, използвайки входния вектор KSW, предава управлението:

– в режим TEKCT40 — на подпрограмата KEYIN;

– в режим TEKCT80 — на подпрограмата BASICIN;

– при получаване на данни от входно–изходно устройство — на съответната входна драйверна програма с начален адрес от вида $Cn00 (където n е номерът на съответния съединител) или на потребителската драйверна програма, управляваща обмена.

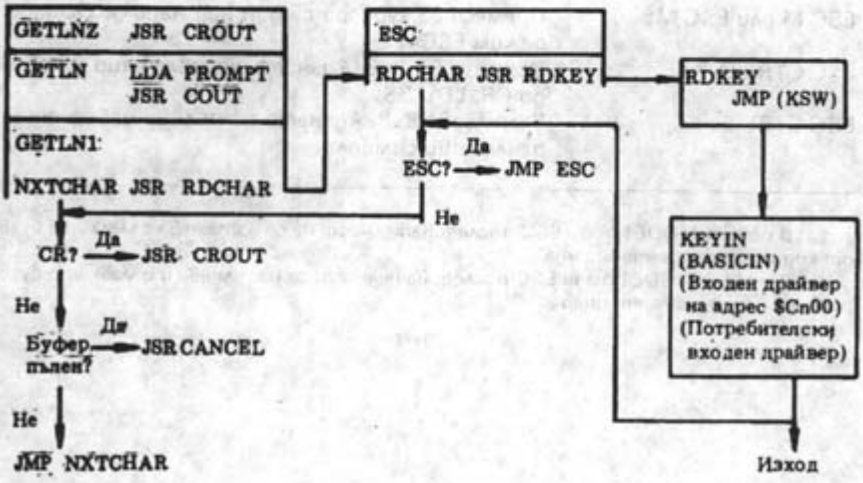
На фиг. 5.2 е показана опростена схема на връзките между входните подпрограми.

Подпрограма RDKEY ($FD0C). Подпрограмата RDKEY е основната подпрограма за въвеждане на един символ от текущото вход-

но устройство. Най-напред тя симулира мигащ курсор на позицията, определена от съдържанието на клетка CH, като маскира двата старши бита на кода на символа, намиращ се на тази позиция. След това чрез входния вектор KSW предава управлението на съответната подпрограма. Това най-често е подпрограмата KEYIN или BASICIN, но може да бъде и програмата за управление на интелигентно входно устройство, включено към някой от съединителите на компютъра, или потребителска програма. След изпълнението си подпрограмата RDKEY записва кода на въведения символ в регистър A.

Подпрограма RDKEY1 ($FD18). Подпрограмата RDKEY1 е алтернативна входна точка на подпрограмата RDKEY. Изпълнява същите функции като подпрограмата RDKEY, но не поддържа собствен курсор. Предпочита се пред подпрограмата RDKEY при обмен с външни входно-изходни устройства, защото предава управлението директно на съответната обслужваща програма.

Подпрограма RDCHAR ($FD35). Подпрограмата RDCHAR е алтернативна подпрограма за въвеждане на един символ. Тя използва стандартната входна подпрограма RDKEY за получаване на символа. Ако въведеният символ не е ESC, записва го в регистър A. Ако въведеният символ е ESC, тя се обръща повторно към подпрограмата RDKEY за получаване на следващия символ и след това предава този символ на подпрограмите за обработка на т.нар. ESC-последователности (вж. табл. 5.8 и 5.9).



Фиг. 5.2. Опростена схема на връзките между входните подпрограми

Таблица 5.9

ESC-последователности в режим ТЕКСТ40

Последователност	Функция
ESC @	Изчиства текстовия прозорец, поставя курсора в горния му ляв край и излиза от режим ESCAPE. Еквивалентна е на команда HOME от БЕЙСИК
ESC A или ESC А[1]	Премества курсора с една позиция надясно и излиза от режим ESCAPE
ESC B или ESC Б[1]	Премества курсора с една позиция наляво и излиза от режим ESCAPE
ESC C или ESC Ц[1]	Премества курсора с един ред надолу и излиза от режим ESCAPE
ESC D или ESC Д[1]	Премества курсора с един ред нагоре и излиза от режим ESCAPE
ESC E или ESC Е[1]	Изчиства екрана до края на текущия ред и излиза от режим ESCAPE
ESC F или ESC Ф[1]	Изчиства екрана до долния десен ъгъл на текстовия прозорец и излиза от режим ESCAPE
ESC I или ESC И[2]	Премества курсора с един ред нагоре. Остава в режим ESCAPE
ESC J или ESC Й[2]	Премества курсора с една позиция наляво. Остава в режим ESCAPE
ESC K или ESC К[2]	Премества курсора с една позиция надясно. Остава в режим ESCAPE
ESC M или ESC М[2]	Премества курсора с един ред надолу. Остава в режим ESCAPE
ESC CTRL-D	Ограничава въвежданите управляващи символи до CR, LF и BS
ESC CTRL-E	Разрешава въвеждането на пълния набор от управляващи символи

[1] В режим MODE1 тази ESC-последователност не се изпълнява с малките букви от кирилицата и латиницата.

[2] В режим MODE1 тази ESC-последователност се изпълнява и с малките букви от кирилицата и латиницата.

Таблица 5.10

ESC-последователности, изпълнявани от програмното осигуряване, поддържащо 80-колонно изображение

Последователност	Функция
1	2
ESC 4	Ако програмното осигуряване на режим ТЕКСТ80 е активно, установява формат 40 КОЛОНИ, записва началния адрес на подпрограмата BASICOUT в изходния вектор CSW и на подпрограмата BASICIN във входния вектор KSW, възстановява нормалните размери на текстовия прозорец и излиза от режим ESCAPE
ESC 8	Ако програмното осигуряване на режим ТЕКСТ80 е активно, установява формат 80 КОЛОНИ, записва началния адрес на подпрограмата BASICOUT в изходния вектор CSW и на подпрограмата BASICIN във входния вектор KSW, възстановява нормалните размери на текстовия прозорец и излиза от режим ESCAPE
ESC @	Изчиства текстовия прозорец, поставя курсора в горния му ляв край и излиза от режим ESCAPE Еквивалентна е на команда HOME от БЕЙСИК
ESC A или ESC А[1]	Премества курсора с една позиция надясно и излиза от режим ESCAPE
ESC B или ESC Б[1]	Премества курсора с една позиция наляво и излиза от режим ESCAPE
ESC C или ESC Ц[1]	Премества курсора с един ред надолу и излиза от режим ESCAPE
ESC D или ESC Д[1]	Премества курсора с един ред нагоре и излиза от режим ESCAPE
ESC E или ESC Е[1]	Изчиства екрана до края на текущия ред и излиза от режим ESCAPE
ESC F или ESC Ф[1]	Изчиства екрана до долния десен ъгъл на текстовия прозорец и излиза от режим ESCAPE
ESC I или ESC И[2]	Премества курсора с един ред нагоре. Остава в режим ESCAPE
ESC J или ESC Й[2]	Премества курсора с една позиция наляво. Остава в режим ESCAPE
ESC K или ESC К[2]	Премества курсора с една позиция надясно. Остава в режим ESCAPE
ESC M или ESC М[2]	Премества курсора с един ред надолу. Остава в режим ESCAPE

1	2
ESC CTRL-D	Ограничава въвежданите управляващи символи до CR, LF и BS
ESC CTRL-E	Разрешава въвеждането на пълния набор от управляващи символи
ESC CTRL-Q	Дезактивира програмното осигуряване, поддържащо режим ТЕКСТ80, установява началния адрес на подпрограмата COUT в изходния вектор CSW и на подпрограмата KEYIN във входния вектор KSW, възстановява нормалните размери на текстовия прозорец и излиза от режим ESCAPE

[1] В режим MODE1 тази ESC-последователност не се изпълнява с малките букви от кирилицата и латиницата.

[2] В режим MODE1 тази ESC-последователност се изпълнява и с малките букви от кирилицата и латиницата.

Подпрограма GETLN ($FD6A). Това е основната входна подпрограма за въвеждане на низ. С последователни обръщения към подпрограмата RDKEY получава до 255 символа от текущата входна програма (обикновено KEYIN или BASICIN). От подпрограмата GETLN се излиза при получаване на управляващ символ CR. При нейното изпълнение въведените символи се връщат във входния буфер, а числото, показващо техния брой — в регистър X. Тъй като обръщенията към подпрограмата RDKEY се извършват през подпрограмата RDCHAR, GETLN поддържа всички стандартни функции за редактиране, познати от работата с програмата МОНИТОР и с интерпретатора на БЕЙСИК. Тези функции са достъпни и при всички програми, които използват подпрограмата GETLN.

Първото, което прави подпрограмата GETLN, е зареждането и извеждането на съдържанието на клетка PROMPT (адрес $33) на текущото изходно устройство. В тази клетка се намира оповестяващият символ на програмата, използваща за вход подпрограмата GETLN. Той показва, че съответната програма очаква да получи низ от текущото входно устройство. Прието е различните програми да използват различни оповестяващи символи (вж. табл. 5.11). Така потребителят се ориентира по-лесно коя е управляващата програма в момента. Програмите, написани на машинен език, могат да имат произволен оповестяващ символ. Не е така обаче с програмите, написани на БЕЙСИК. Там оповестяващият символ е фиксиран и интерпретаторът на БЕЙСИК и програмата МОНИТОР го възстановяват всеки път, когато очакват въвеждането на низ.

Таблица 5.11

Оповестяващи символи

Оповестяващ символ	Управляваща програма
*	МОНИТОР
!	МИНИАСЕМБЛЕР
>	Интерпретатор на ЦЕЛОЧИСЛЕН БЕЙСИК
]	Интерпретатор на РАЗШИРЕН БЕЙСИК
?	Оператор INPUT от БЕЙСИК

След като чрез подпрограмата COUT изведе оповестяващия символ на текущото изходно устройство, подпрограмата GETLN извиква подпрограмата RDCHAR и очаква символ от текущото входно устройство. Ако полученият символ е NAK, подпрограмата GETLN извлича следващия символ от текущата позиция на курсора върху екрана. Останалите символи се записват във входния буфер. След това се проверява дали въведеният символ не е CR и ако е така, GETLN го изпраща към текущото изходно устройство и връща управлението на извикващата програма. Всички останали символи се маскират с флаг INVFLG и се извеждат на текущото изходно устройство (т.нар. ехо), след което се правят проверки за:

– управляващ символ BS — курсорът, поддържан от GETLN, се премества с една позиция назад;

– управляващ символ CAN — отхвърля въведения низ, като изпраща символи \ и CR към текущото изходно устройство и рестартира подпрограмата GETLN.

Ако в буфера вече има 248 или повече символи, подпрограмата GETLN изработва звуков сигнал през вградения високоговорител, преди отново да извика подпрограмата RDCHAR. Ако буферът се препълни, преди да се получи управляващ символ CR, въведеният низ се отхвърля и подпрограмата GETLN се стартира отново.

Обърнете внимание, че за разлика от Правец–82 и Правец–8М в Правец–8А с флаг INVFLG се маскират всички символи, изпращани към текущото изходно устройство, включително и ехото от входния поток. Тази промяна в програмата МОНИТОР на Правец–8А се отразява на действието на операторите на БЕЙСИК INVERSE и FLASH.

Подпрограма GETLNZ ($FD67). Тази подпрограма е алтернативна входна точка на подпрограмата GETLN. Преди да предаде управлението на GETLN за отпечатване на оповестяващия символ, изпраща управляващ символ CR към текущото изходно устройство.

Подпрограма GETLN1 ($FD6F). Подпрограмата GETLN1 също е алтернативна входна точка на подпрограмата GETLN, но за разлика от нея извежда оповестяващия символ само ако е въведен управляващият символ CAN или буферът е препълнен.

Подпрограма INPORT ($FE8B). Това е подпрограма за избор на входно устройство. Записва началния адрес на текущата драйверна програма във входния вектор KSW. Извиква се, след като номерът на съединителя е зареден в регистър A. Адресът се записва във входния вектор във вида Cn00, където **n** = 0 — 7 е номерът на съответния съединител. Изпълнението на подпрограмата INPORT е еквивалентно на изпълнението на команда IN# n от БЕЙСИК. За **n** = 0 то е еквивалентно на изпълнението на подпрограмата SETKBD.

Подпрограма SETKBD ($FE89). Установява клавиатурата като текущо входно устройство. Записва входната точка на подпрограмата KEYIN във входния вектор KSW. Изпълнението ѝ е еквивалентно на изпълнението на команда IN# 0 от БЕЙСИК. То е еквивалентно и на последователността

```
LDA #$00
JSR INPORT
```

Подпрограма KEYIN ($FD1B). Подпрограмата KEYIN е стандартната подпрограма за обслужване на клавиатурата в режим ТЕКСТ40. Тя изпълнява едновременно три функции:

1. Поддържа собствен мигащ курсор с формата на шахматно поле, като алтернативно записва на текущата позиция на курсора код $FF и кода на символа от тази позиция (най–често интервал).

2. Непрекъснато изменя съдържанието на генератора на случайни числа, разположен на адреси $4E — $4F.

3. Проверява клавиатурния буфер за валиден код и при натискане на клавиш изчиства строба на клавиатурата, маха курсора от екрана и записва кода на въведения символ в регистър A. В зависимост от режима на работа (MODE0 или MODE1) полученият код е седемили осембитов.

Подпрограма BASICIN ($C305). Подпрограмата BASICIN обслужва клавиатурата, когато програмното осигуряване, поддържащо 80-колонно изображение, е активно. По функции тя е еквивалентна на подпрограмата KEYIN. Мигащият курсор, поддържан от подпрограмата BASICIN, се различава от този, поддържан от подпрограмата KEYIN, и представлява правоъгълник във формат ИНВЕРСНО ВИДЕО. В режим ESCAPE той се заменя от знака плюс (+) също във формат ИНВЕРСНО ВИДЕО. В зависимост от установения режим на работа (MODE0 или MODE1) подпрограмата BASICIN поддържа седемили осембитов код.

5.5. Подпрограми за обслужване на екрана в текстов режим

Характерно за подпрограмите, обслужващи екрана в текстов режим, е, че голяма част от тях могат да се използват както в режим ТЕКСТ40, така и в режим ТЕКСТ80. В действителност функциите по поддържане на екрана в двата текстови режима се изпълняват от различни подпрограми, като тези, които реализират една и съща функция, са свързани с обща входна точка. Коя от двете едноименни подпрограми ще се изпълни се определя единствено от установения текстов режим. Това, от една страна, опростява програмирането, защото програмата става независима от избора на текстов режим, но от друга, води до усложняване на съответните подпрограми и на връзките между тях. И тъй като новите подпрограми са и повече, и по-дълги в сравнение с тези на Правец–82 и Правец–8М и не могат да се поместят изцяло на определеното им място в областта $F800 — $FFFF, се налага някои от тях (INIT, HOME, CLREOP, CLREOL, CLEOLZ, SCROLL) да се разположат частично в областта $F800 — $FFFF, частично в областта $C100 — $CFFF. Връзките между отделните подпрограми допълнително се усложняват и от необходимостта от включване и изключване на вградената постоянна памет в областта $C100 — $CFFF и от автоматично определяне на установения текстов режим.

Към подпрограмите, управляващи екрана в текстов режим, спадат и някои от подпрограмите, разгледани в предишните параграфи (COUT1, COUTZ, BASICOUT, OUTPORT, SETVID, ESC и др.). Условно всички те могат да се разделят на няколко подкласа.

1. Подпрограми за управление движението на курсора: VTAB, VTABZ, TABV, BS, UP, ADVANCE, STOADV, CR, LF, BASCALC и др. Изпълнението им води до относително (спрямо текущата позиция) или абсолютно (спрямо началото на екрана или началото на текстовия прозорец) преместване на курсора. В общия случай позицията на курсора на екрана се определя от съдържанието на четири клетки от нулевата страница от паметта на компютъра:

* **CV** (адрес $25) — абсолютна вертикална позиция на курсора. Не зависи от параметрите на текстовия прозорец, а

се определя единствено от поредния номер на реда от екрана. Първият ред е с номер $0, а последният е с номер $17.

* **CH** (адрес $24) — относителна хоризонтална позиция на курсора по отношение на началото (лявата страна) на текстовия прозорец.

* **BASL,H** (адреси $28 — $29) — базов адрес на ред от екрана (адресът на клетката от екранната памет, който съответства на първия символ от даден ред на екрана). В зависимост от това, дали при отчитане на началото на реда се взема предвид лявата страна на

текстовия прозорец, базовият адрес е относителен или абсолютен. Абсолютният адрес се изчислява от подпрограмата BASCALC по съдържанието на клетка CV, а относителният — от подпрограмата VTAB по съдържанието на клетки WNDLFT и CV.

* **WNDLFT** (адрес $20) — хоризонтално отместване на лявата страна на текстовия прозорец.

Адресът на курсора в екранната памет е сума от съдържанието на клетки BASL,H и CH, а това означава, че BASL,H трябва да се преизчислява при всяка смяна на CV или WNDLFT.

Внимание! Програмното осигуряване на режим TEKCT80 записва хоризонталната позиция на курсора в клетка CH80 (адрес $57B). Поради това подпрограмите ADVANCE, BS, STORADV, CR и др., чието действие е свързано с промяна на хоризонталната позиция на курсора, не функционират нормално в режим TEKCT80.

2. Подпрограми за форматиране на екрана: INIT, SETTXT, SETNORM, SETINV, SETWND, SETIFLG. Тези подпрограми са предназначени за установяване на текстов режим, за установяване на формата на текстовото изображение (НОРМАЛНО или ИНВЕРСНО ВИДЕО) и за нормализиране на размерите на текстовия прозорец.

3. Подпрограми за изчистване на целия екран или на част от него: HOME, CLREOL, CLEOLZ, CLREOP. В тази група са и подпрограмите, които преместват изображението на екрана с един ред нагоре или надолу (операция СКРОЛ) — SCROLL и SCROLLDN.

4. Подпрограми за отпечатване на текст на текущото изходно устройство: COUT1, COUTZ, BASICOUT.

Подпрограма BASCALC ($FBC1). Извиква се, след като поредният номер на реда от екрана (0 — 23) се зареди в регистър A. Изчислява базовия адрес на този ред и го записва в клетки BASL,H (осъществява преобразуване на поредния номер на реда в адрес от екранната памет). Този адрес отговаря на първата позиция на реда от екрана и не зависи от параметрите на текстовия прозорец. Всички следващи адреси в рамките на същия ред се получават с индексна адресация по съдържанието на регистър Y. Подпрограмата BASCALC се извиква от почти всички подпрограми, използвани за управление на екрана в текстов режим. При нейното изпълнение се променя съдържанието на регистър A.

Подпрограма VTAB ($FC22). Това е основната подпрограма за управление на движението на курсора. Извиква подпрограмата BASCALC и по съдържанието на клетки CV и WNDLFT изчислява базовия адрес. Записва получената стойност в клетки BASL,H и така установява вертикалната позиция на курсора. Обърнете внимание, че докато подпрограмата BASCALC изчислява абсолютния базов адрес, подпрограмата VTAB изчислява базовия адрес в рамките на текстовия прозорец. Ето защо подпрограмите, управляващи тексто-

вия режим в рамките на текстовия прозорец, при изчисляване на базовия адрес не се обръщат директно към подпрограмата BASCALC, а използват подпрограмата VTAB. За целта, преди да се извика подпрограмата VTAB, в клетка CV се записва вертикалната позиция на курсора или директно се използва алтернативната входна точка VTABZ. Например:

```
LDA #$V     ;V = $0 — $17 — номер на ред от екрана
STA CV
JSR VTAB
   или:
LDA #$V
JSR VTABZ
```

В програмата МОНИТОР на Правец–8А не съществува подпрограма за установяване на хоризонталната позиция на курсора. В зависимост от установения режим такава подпрограма се симулира по следните начини:

```
– в режим ТЕКСТ40:
LDA #$H     ;H = $0 — $27 — хоризонтална позиция
STA CH      ;CH показва хоризонталната позиция
            ;на курсора (адрес $24)

— в режим ТЕКСТ80:
LDA #$H     ;H = $0 — $4F — хоризонтална позиция
STA $057B
```

При изпълнението на подпрограмата VTAB се променя съдържанието на регистър А.

Подпрограма VTABZ ($FC24). Това е алтернативна входна точка на подпрограмата VTAB. По съдържанието на регистър А и на клетка WNDLFT изчислява базовия адрес и го записва в BASL,H. Не записва съдържанието на регистър А в клетка CV. Извиква се, след като номерът на реда от екрана се зареди в регистър А.

```
LDA #$V     ;V = $0 — $17
JSR VTABZ
```

При изпълнението на подпрограмата VTABZ се променя съдържанието на регистър А.

Подпрограма TABV ($FB5B). И тази подпрограма е алтернативна входна точка на подпрограмата VTAB. Записва съдържанието на регистър А в клетка CV и безусловно предава управлението на под-

програмата VTAB. Извиква се, след като номерът на реда от екрана се зареди в регистър A. Изпълнението ѝ променя съдържанието на този регистър.

```
LDA #$V     ;V = $0 — $17
JSR TABV
```

Подпрограма UP ($FC1A). Премества курсора с един ред нагоре, без да променя хоризонталната му позиция. Курсорът не се премества, ако се намира на най–горния ред от текстовия прозорец. Изпълнението на подпрограмата UP променя съдържанието на регистър A.

Подпрограма BS ($FC10). Подпрограмата BS премества курсора с една позиция наляво. Ако курсорът е в началото на реда, тя го премества в края на горния ред, като преизчисляването на вертикалната му позиция се извършва от подпрограмата UP.

Изпълнението на подпрограмата BS променя съдържанието на регистър A.

Подпрограмата BS не може да се използва в режим ТЕКСТ80.

Подпрограма ADVANCE ($FBF4). Увеличава съдържанието на клетка CH с единица. Ако е достигнат краят на реда в рамките на текстовия прозорец, извиква подпрограмата CR.

При изпълнението на подпрограмата ADVANCE се променя съдържанието на регистър A.

Подпрограмата ADVANCE не може да се използва в режим ТЕКСТ80.

Подпрограма STORADV ($FBF0). Подпрограмата STORADV, без да анализира съдържанието на регистър A, го записва на текущата позиция на курсора. Използва подпрограмата ADVANCE за преместване на курсора с една позиция надясно.

Изпълнението на подпрограмата STORADV променя съдържанието на регистри A и Y.

Подпрограмата STORADV не може да се използва в режим ТЕКСТ80.

Подпрограма LF ($FC66). Премества курсора с един ред надолу, без да променя хоризонталната му позиция. За целта съдържанието на клетка CV се увеличава с единица и новата стойност се сравнява с долния край на текстовия прозорец. Ако (CV) ≥ (WNDBTM), изображението на екрана се премества с един ред нагоре. Ако (CV) < (WNDBTM), извиква се подпрограмата VTABZ, която преизчислява новия адрес на курсора.

Изпълнението на подпрограмата LF разрушава съдържанието на

регистър A, а при преместване на изображението с един ред нагоре се разрушава и съдържанието на регистър Y.

Подпрограма CR ($FC62). Премества курсора в началото на следващия ред от екрана. За целта записва нула в клетка CH (премества курсора в началото на текущия ред) и извиква подпрограмата LF, която премества курсора с един ред надолу.

Изпълнението на подпрограмата CR разрушава съдържанието на регистър A, а при преместване на изображението с един ред нагоре се разрушава и съдържанието на регистър Y.

Подпрограмата CR не може да се използва в режим ТЕКСТ80.

Подпрограма SETTXT ($F839). Установява екрана в текстов режим и използва подпрограмата SETWND за нормализиране на параметрите на текстовия прозорец.

Изпълнението на подпрограмата SETTXT променя съдържанието на регистър A.

Подпрограма SETWND ($FB48). Подпрограмата SETWND нормализира размерите на текстовия прозорец. При изпълнението ѝ се променя съдържанието на регистър A.

Подпрограма SETIFLG ($FE86). Записва съдържанието на регистър Y в клетка INVFLG и така установява формата на изображението (НОРМАЛНО, ИНВЕРСНО или МИГАЩО ВИДЕО). Извиква се, след като стойността на маската за съответния формат се зареди в регистър Y.

$FF — НОРМАЛНО ВИДЕО
$7F — МИГАЩО ВИДЕО
$3F — ИНВЕРСНО ВИДЕО

Например:

```
LDY #$7F
JSR SETIFLG
```

Подпрограма SETNORM ($FE84). Установява формат НОРМАЛНО ВИДЕО. Използва подпрограмата SETIFLG за установяване на флаг INVFLG.

При изпълнението на подпрограмата SETNORM се променя съдържанието на регистър Y.

Подпрограма SETINV ($FE80). Установява формат ИНВЕРСНО ВИДЕО. Използва подпрограмата SETIFLG за установяване на флаг INVFLG.

При изпълнението на подпрограмата SETINV се променя съдържанието на регистър Y.

Подпрограма INIT ($FB2F). Нулира съдържанието на клетка STATUS (адрес $48). Изключва режим ГР280. Нулира програмноуправляемия ключ СТР2. Установява текстов режим. Установява нормалните размери на текстовия прозорец.

При изпълнението на подпрограмата INIT се променя съдържанието на регистър А.

Подпрограма CLREOL ($FC9C). Изчиства реда от текущата позиция на курсора до десния край на текстовия прозорец. Позицията на курсора се определя от хоризонталната му координата и от базовия адрес BASL,H.

Изпълнението на подпрограмата CLREOL разрушава съдържанието на регистри А и Y, а курсорът запазва позицията си.

Подпрограма CLEOLZ ($FC9E). Изчиства текущия ред от екрана. Започва от позицията, определена от базовия адрес BASL,H и съдържанието на регистър Y, и завършва до десния край на текстовия прозорец. Извиква се, след като в регистър Y се укаже позицията, откъдето трябва да започна изтриването. Например:

```
LDY H       ;H = $0 — $24 за 40 колони
            ;H = $0 — $4F за 80 колони
JSR CLEOLZ
```

Изпълнението на подпрограмата CLEOLZ разрушава съдържанието на регистри А и Y, а курсорът запазва позицията си.

Подпрограма CLREOP ($FC42). Изчиства екрана от текущата позиция на курсора до края на текстовия прозорец. Изпълнението ѝ разрушава съдържанието на регистри А и Y, а курсорът запазва позицията си.

Подпрограма HOME ($FC58). Позиционира курсора в началото на текстовия прозорец (горния му ляв край) и изчиства екрана. За целта нулира клетките, съдържащи хоризонталната и вертикалната координата на курсора,и извиква подпрограмата CLREOP.

Изпълнението на подпрограмата HOME променя съдържанието на регистри А и Y.

Подпрограма SCROLL ($FC70). Премества изображението, ограничено от рамките на текстовия прозорец, с един ред нагоре. Съдържанието на най–горния ред от текстовия прозорец се губи, а най–долният ред се изчиства.

Изпълнението на подпрограмата SCROLL разрушава съдържанието на регистри A и Y, а курсорът запазва позицията си.

Подпрограма SCROLLDN ($CCAA). Премества изображението, намиращо се в текстовия прозорец, с един ред надолу. Изпълнява се само под управление на програмното осигуряване, поддържащо 80-колонно изображение. Например:

```
STA $C007   ;Разрешава достъпа до резидентната постоянна па-
            ;мет на адреси $C100 – $CFFF
JSR SCROLLDN
STA $C006   ;Разрешава достъпа до постоянната памет от об-
            ;ластта $C100 – $CFFF, разположена на допълнител-
            ;ните модули
```

5.6. Подпрограми за поддържане на графика с малка разделителна способност (режим ГР40)

Подпрограмите за поддържане на изображението в режим ГР40 са много близки до съответните команди в БЕЙСИК. Те включват подпрограми за установяване на цвят, за изчистване на целия екран или част от него, за изобразяване на блокче, на хоризонтална и на вертикална линия в текущия цвят и за определяне на цвета на зададено блокче от екрана. Към тях условно се причисляват и подпрограмите за установяване на режим ГР40 и за нормализиране на текстовия прозорец. На фиг. 5.3 в опростен вид са показани връзките между отделните подпрограми, поддържащи графиката с малка разделителна способност. Не е трудно да се установи, че основна е подпрограмата от най–ниско ниво GBASCALC, която изчислява базовия адрес за всеки ред от графичното изображение (адресът на клетката от екранната памет, отговаряща на началото на съответния ред от екрана). Подпрограмите NXTCOL и SETCOL, които установяват текущия цвят, не са свързани с останалите.

Подпрограма SETGR ($FB40). Започва от адрес $FB40 и последователно изпълнява следните функции:

1. Установява графичен режим. Обърнете внимание, че подпрограмата SETGR не установява нито вида на графичния режим (ГР40 или ГР280), нито графичната страница. И графичният режим, и страницата трябва да са установени предварително.
2. Установява режим на смесено изображение.
3. Извиква подпрограмата CLRTOP, която изчиства първите 40 реда от екрана в режим ГР40. Следователно, ако подпрограмата

SETGR бъде извикана след установяването на режим ГР280, тя няма да изчисти съответната графична страница.

4. Преминава в подпрограма SETWND.

За установяването на целия екран в режим ГР40 няма готова подпрограма, но такава може да се симулира по следния начин:

```
LDA $C050   ;Установява графичен режим
LDA $C052   ;Забранява режима на смесено изображение
LDA $C056   ;Установява графика с малка разделителна способ-
            ;ност
LDA $C054   ;Избира първа страница
```

Изпълнението на подпрограмата SETGR променя съдържанието на регистри A и Y.

Подпрограма NEXTCOL ($F85F). Извлича кода на текущия цвят за режим ГР40 от клетка COLOR (адрес $30) и го увеличава с три, след което преминава в подпрограма SETCOL.

```
COLOR = (COLOR + 3) MOD 16
```

При изпълнението на подпрограмата NEXTCOL се променя съдържанието на регистър A.

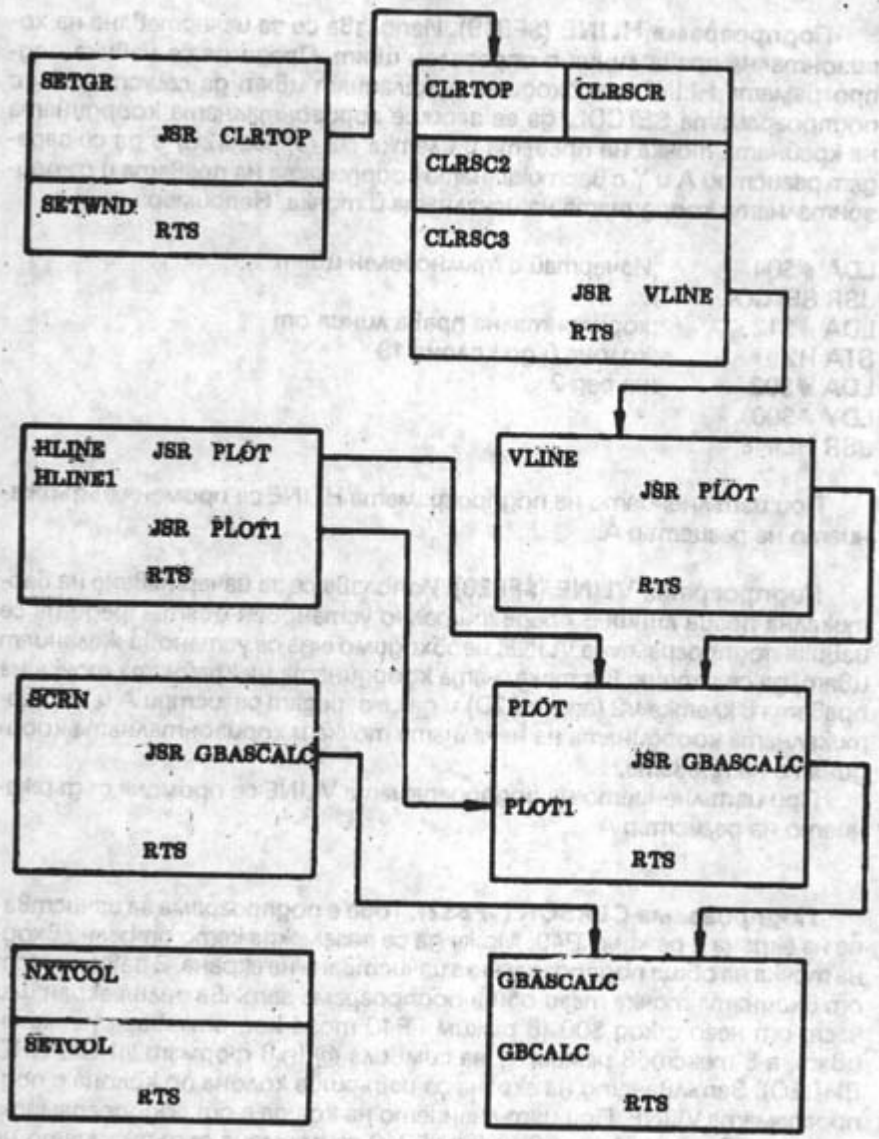
Подпрограма SETCOL ($F864). Установява текущия цвят в режим ГР40. Преди нейното извикване е необходимо кодът на цвета да се зареди в регистър A (A = 0 — 15). Подпрограмата SETCOL преобразува този код така, че двете половинки на байта да са еднакви, и го записва в клетка COLOR. Така например след изпълнение на следната програма на адрес $30 има $66:

```
LDA #$06
JSR SETCOL
```

Изпълнението на подпрограмата SETCOL води до промяна на съдържанието на регистър A.

Подпрограма GBASCALC ($F847). Преобразува координатите на блокче от екрана в адрес от екранната памет. Извиква се, след като вертикалната координата на блокчето се зареди в регистър A (A = 0 — 23). Изчислява и записва в GBASL,H (адреси $38 — $39) базовия адрес. Това е адресът на клетката от екранната памет, отговарящ на най–лявото блокче от съответния ред на екрана. Използва се от почти всички подпрограми, свързани с режим ГР40 (вж. фиг. 5.3). При изпълнението на подпрограмата GBASCALC се променя съдържанието на регистър A.

Подпрограма PLOT ($F800). Изчертава единично блокче в установения цвят. Извиква се, след като хоризонталната и вертикалната координата на блокчето са заредени в регистри Y (Y = 0 — 39) и A (A = 0 — 47). Използва подпрограмата GBASCALC за изчисляване



Фиг. 5.3. Опростена схема на връзките между подпрограмите, поддържащи режим ГР40

на базовия адрес на съответния ред от екрана. Определя дали блокчето попада на четен или на нечетен ред от екрана и в зависимост от това записва $0F или $F0 в клетка MASK (адрес $2E). При изпълнението ѝ се променя съдържанието на регистър A.

Подпрограма HLINE ($F819). Използва се за изчертаване на хоризонтална права линия с определен цвят. Преди да се извика подпрограмата HLINE, необходимо е желаният цвят да се установи с подпрограмата SETCOL, да се запише хоризонталната координата на крайната точка на правата в клетка H2 (адрес $2C) и да се заредят регистри A и Y с вертикалната координата на правата и с хоризонталната координата на началната ѝ точка. Например:

```
LDA #$04        ;Изчертай с тъмнозелен цвят
JSR SETCOL      ;
LDA #$12        ;хоризонтална права линия от
STA H2          ;колона 0 до колона 18
LDA #$02        ;на ред 2
LDY #$00
JSR HLINE
```

При изпълнението на подпрограмата HLINE се променя съдържанието на регистър A.

Подпрограма VLINE ($F828). Използва се за изчертаване на вертикална права линия в предварително установен цвят. Преди да се извика подпрограмата VLINE, необходимо е да се установи желаният цвят, да се запише вертикалната координата на крайната точка на правата в клетка V2 (адрес $2D) и да се заредят регистри A и Y с вертикалната координата на началната точка и хоризонталната координата на правата.

При изпълнението на подпрограмата VLINE се променя съдържанието на регистър A.

Подпрограма CLRSCR ($F832). Това е подпрограма за изчистване на екрана в режим ГР40. Може да се разглежда като отделна входна точка на обща подпрограма за изчистване на екрана. В зависимост от входната точка тази обща подпрограма запълва целия екран или част от него с код $00 (в режим ГР40 този код отговаря на черен цвят, а в текстов режим — на символа @ във формат ИНВЕРСНО ВИДЕО). Запълването на екрана се извършва колона по колона с подпрограмата VLINE. При изпълнението на коя да е от подпрограмите за изчистване на екрана в режим ГР40 се променя съдържанието на регистри A и Y.

Подпрограмата CLRSCR изчиства целия екран (48 реда).

Подпрограма CLRTOP ($F836). Тя е алтернативна входна точка на програмата за изчистване на екрана. Използва се в режим ГР40 за изчистване на първите 40 реда от екрана.

Подпрограма CLRSC2 ($F838). Тази подпрограма също е алтернативна входна точка на програмата за изчистване на екрана. Изчиства екрана от най–горния ред до реда, определен от съдържанието на регистър Y.

Подпрограма CLRSC3 ($F83C). И тя е алтернативна входна точка на програмата за изчистване на екрана. Изчиства горния ляв ъгъл на екрана. Започва от точката с координати H = V = 0 и продължава до реда и колоната, определени от съдържанието на регистър Y и на клетка V2.

Подпрограма SCRN ($F871). Използва се за определяне на цвета на единичното блокче с координати, зададени със съдържанието на регистри A и Y. Извиква се, след като вертикалната и хоризонталната координата на блокчето се заредят в регистри A и Y. След завършване на подпрограмата SCRN кодът на цвета на блокчето е записан в регистър A.

5.7. Подпрограми за работа с допълнителната памет

В програмата МОНИТОР на персоналния компютър Правец–8А има две подпрограми, предназначени за облекчаване на работата с допълнителната памет. Те позволяват допълнителната памет да се използва, без да е необходимо да се превключват програмноуправляемите ключове, описани в гл.3. И двете подпрограми имат по две равностойни входни точки. Това са съответно адреси $C311 и $C363 за подпрограмата AUXMOVE и адреси $C314 и $C3B0 за подпрограмата XFER.

Подпрограма AUXMOVE ($C311). Подпрограмата AUXMOVE копира блокове от данни от системната в допълнителната памет или обратно — от допълнителната в системната памет. За да се използва тази подпрограма, е необходимо в двойките клетки на нулевата страница A1, A2 и A4 да се заредят съответно началният и крайният адрес на блока данни, който ще се копира, и началният адрес на областта, в която ще се извърши копирането. Посоката на трансфера (от системната в допълнителната памет или обратно) се определя от състоянието на флаг C (ПРЕНОС) от регистъра на състоянието на микропроцесора (вж. табл. 5.12).

Тъй като предаването на параметри към подпрограмата AUXMOVE се извършва чрез клетки от нулевата страница на паметта на компютъра, тя не може да се използва за копиране на блокове данни в нулева и първа страница и в областта $D000 — $FFFF, т.е. в областите, превключвани с програмноуправляемия ключ AHC.

Таблица 5.12

Предаване на параметри към подпрограмата AUXMOVE

Име	Адрес	Параметър
A1H	$3D	Старши байт на началния адрес на блока, който ще се копира
A1L	$3C	Младши байт на началния адрес на блока, който ще се копира
A2H	$3F	Старши байт на крайния адрес на блока, който ще се копира
A2L	$3E	Младши байт на крайния адрес на блока, който ще се копира
A4H	$43	Старши байт на началния адрес на областта, в която се извършва копирането
A4L	$42	Младши байт на началния адрес на областта, в която се извършва копирането
Флаг С		Флаг С на регистъра на състоянието на микропроцесора. Ако този флаг е единица, трансферът се извършва от системната към допълнителната памет. Ако той е нула, трансферът се извършва от допълнителната към системната памет

Ето и една примерна последователност за предаване на необходимите параметри към подпрограмата AUXMOVE и за нейното изпълнение. За повече подробности относно използването на клетките от нулевата страница вж. приложение 2.

```
LDA #$STARTH    ;Запис на началния адрес на областта,
STA $A1H        ;чието съдържание ще се премества, на
LDA #$STARTL    ;адреси A1H, A1L
STA $A1L        ;
LDA #$ENDH      ;Запис на крайния адрес на областта,
STA $A2H        ;чието съдържание ще се премества, на
LDA #$ENDL      ;адреси A2H, A2L
STA $A2L        ;
LDA #$DESTH     ;Запис на началния адрес на новата
STA $A4H        ;област на адреси A4H, A4L
```

```
LDA #$DESTL     ;
STA $A4L        ;
SEC             ;Определяне на посоката на
(CLC)           ;преместване. SEC — от системната в допълни-
                ;телната памет. CLC — от допълнителната в
                ;системната памет
STA $C007       ;Разрешаване на достъпа до резидентната
                постоянна памет в областта $C100 — $CFFF
JSR $C311       ;Извикване на подпрограмата AUXMOVE.
STA $C006       ;Забрана на по-нататъшния достъп до
                ;резидентната постоянна памет в
                ;областта $C000 — $CFFF
```

Подпрограма XFER ($C314). Подпрограмата XFER се използва за предаване на програмното управление на компютъра от програмни сегменти, разположени в системната памет, на сегменти в допълнителната памет и обратно. Обръщението към подпрограмата се предхожда от установяването на три параметъра:

– начален адрес на програмата, на която се предава управлението;

– посока на предаване на управлението (от системната към допълнителната памет или обратно);

– коя от двете нулеви страница, респ. кой от двата стека, ще се използва след предаването на управлението.

Началният адрес на програмата, която трябва да поеме управлението на компютъра, се зарежда на адреси $3ED — $3EE. Посоката на предаване на управлението се определя от флаг С на регистъра на състоянието на микропроцесора, а това, коя нулева страница ще се използва от тази програма — от флаг V (ПРЕПЪЛВАНЕ) на същия регистър (вж. табл. 5.13).

След като се установят параметрите за предаване на програмното управление на компютъра, извикването на подпрограмата XFER трябва да стане с инструкция за безусловен преход (JMP), а не с инструкция за извикване на подпрограма (JSR). Подпрограмата XFER записва съдържанието на регистър А и началния адрес на новата управляваща програма в текущия стек, установява програмноуправляемите ключове съгласно зададените параметри и с инструкция за безусловен преход предава управлението на новата програма.

Таблица 5.13

Параметри, използвани от подпрограмата XFER

Име	Адрес	Параметър
	$3ED	Младши байт на адреса на новата управляваща програма
	$3EE	Старши байт на адреса на новата управляваща програма
Флаг С		Ако флаг С на регистъра на състоянието на микропроцесора е 0, управлението се предава от програма в допълнителната на програма в системната памет, а ако този флаг е единица — от програма в системната на програма в допълнителната памет
Флаг V		Ако флаг V на регистъра на състояние на микропроцесора е 0, новата управляваща програма използва нулевата страница и стека от системната памет, а ако този флаг е единица — от допълнителната памет

Следващата програма е един пример за подготвяне на параметрите и за извикване на подпрограмата XFER.

```
LDA #$STARTH      ;Зареждане на началния адрес на
STA $3EE          ;програмата, която ще поемe
LDA #$STARTL      ;управлението, на адреси $3ED — $3EE
STA $3ED          ;
SEC               ;Определяне посоката на предаване на
(CLC)             ;управлението. SEC — от програма в
                  ;в системната на програма в
                  ;допълнителната памет. CLC — от
                  ;програма в допълнителната на
                  ;програма в системната памет
BIT $FF58         ;Избор на нулева страница и стек
(CLV)             ;BIT $FF58 — нулева страница и стек
                  ;от допълнителната памет
                  ;(съдържанието на адрес $FF58 е $69)
                  ;CLV — нулева страница и стек от
                  ;системната памет
STA $C007         ;Разрешава достъпа до подпрограмата
                  ;XFER в резидентната постоянна памет
JMP $C314         ;Извикване на подпрограмата
STA $C006         ;Забрана на достъпа до резидентната
                  ;постоянна памет
```

Внимание! Подпрограмата XFER не подготвя нулевата страница и стека за предаване на управлението.

5.8. Други подпрограми на програмата МОНИТОР

В програмата МОНИТОР има още редица входни точки на подпрограми или на сегменти от подпрограми: за обслужване на вградените входно–изходни устройства, за изключване на режим ТЕКСТ80, за използване на командите от командния процесор на програмата МОНИТОР, за вход в програмата МОНИТОР и в програмата МИНИАСЕМБЛЕР и т.н. Само малка част от тези подпрограми могат да се изпълняват директно. Повечето са от типа на т.нар. симулирани подпрограми и изискват предварителното установяване на един или няколко параметъра.

Подпрограма BELL1 ($FBD9). Това е една от трите входни точки на основната подпрограма за управление на вградения високоговорител. Подпрограмата BELL1 се използва за обработване на управляващия символ BEL. Тя проверява съдържанието на регистър A и ако то е равно на $87, предава управлението на подпрограмата BELL1A.

Подпрограма BELL1A ($FBDD). Тази подпрограма е алтернативна входна точка на подпрограмата BELL1 ($FBD9), но за разлика от нея подпрограмата BELL1A не проверява съдържанието на регистрите, а директно изработва сигнал с продължителност 0,1 s и честота 1 kHz и го изпраща към вградения високоговорител.

При изпълнението на подпрограмата BELL1A или на свързаната с нея подпрограма BELL2 се променя съдържанието на регистри A и Y.

Подпрограма BELL2 ($FBE4). В зависимост от съдържанието на регистър Y тя изпраща тонове с различна честота към вградения високоговорител. Подпрограмата BELL2 се извиква, след като параметърът, необходим за нейното изпълнение, се зареди в регистър Y. В случая това е стойност, отговаряща на желания тон. На ниските тонове отговаря стойност 0, а на високите — 255. Например:

```
LDY #$C0          ;Около 1 kHz
JSR BELL2
```

Подпрограма WAIT ($FCA8). Това е основната подпрограма за изработване на времезакъснения с предварително зададена продължителност. Параметърът в регистър A, който се предава към подпрограмата WAIT, определя продължителността на времезакъснението:

```
LDA #$N          ;N = 0 — 255
JSR $FCA8
```

Времезакъснението, получавано с подпрограмата WAIT, се изчислява по зависимостта

$$TW = 0{,}5T0\ (26 + 27N + 5N^2)$$

където TW е времезакъснението (в микросекунди), внасяно от подпрограмата WAIT; N е съдържанието на регистър A, а T0 = 0,982425 μs е продължителността на микропроцесорния цикъл.

Трябва да се има предвид, че стойността на времезакъснението, внасяно от подпрограмата WAIT, не може да се изчисли точно, защото продължителността на микропроцесорния цикъл T0 не е постоянна величина (един от всеки 64 цикъла е удължен с 0,140346 μs).

Подпрограма PREAD ($FB1E). Определя състоянието на аналоговите входове на компютъра (вж. гл.2). По принцип се използва за отчитане на положението на потенциометрите от ръчките за управление на игри, но може да намери приложение навсякъде, където има преобразуване на съпротивление в код.

Четирите аналогови входа ABX0, ABX1, ABX2 и ABX3 са изведени на съединителя за управление на игри и могат да се използват за измерване на съпротивление в обхвата от 0 до 150 kΩ. Измереното съпротивление R_x се свързва между съответния аналогов вход и линията +5V така, че да стане част от времезадаващата верига, времеконстантата на която се измерва от подпрограмата PREAD. Тази подпрограма се извиква, след като номерът на аналоговия вход се зареди в регистър X. Например:

```
LDX #$02         ;ABX2
JSR PREAD
```

Подпрограмата PREAD най-напред нулира времезадаващите вериги на четирите аналогови входа. След това на всеки 12 микропроцесорни цикъла тя проверява състоянието на старшия бит, прочетен от съответния вход (адреси $C064 — $C067). Ако този бит е все още установен, съдържанието на регистър Y се увеличава с единица. В крайна сметка стойността, натрупана в регистъра, е пропорционална на стойността на измереното съпротивление.

При изпълнението на подпрограмата PREAD се променя съдържанието на регистри A и Y.

Подпрограма QUIT ($CDAA). Изключва режим TEKCT80 и установява режим TEKCT40. За целта установява пълните размери на тек-

ствия прозорец, съответстващи на 40-колонно изображение, преобразува изображението от 80 в 40 колони, поставя мигащия курсор в началото на последния ред от екрана и изключва алтернативното символно множество. Променя вектори KSW и CSW (извиква подпрограмите SETVID и SETKBD), в резултат на което се разкъсват връзките с текущата операционна система (ДОС3.3 или ПроДОС). Това налага след изпълнението на подпрограмата QUIT да се възстановят връзките с операционната система, а при работа в режим MODE1 — да се включи отново и алтернативното символно множество. Например:

```
STA $C007       ;Разрешава избора на резидентната
                ;постоянна памет
JSR $CDAA       ;Извиква подпрограмата QUIT
STA $C006       ;Изключва резидентната постоянна памет
STA $C00F       ;Включва алтернативното символно
                ;множество
JSR $03EA       ;Свързва ДОС3.3
   или:
JSR $9A17       ;Свързва ПроДОС
```

Изпълнението на подпрограмата QUIT разрушава съдържанието на всички регистри на микропроцесора.

Подпрограма GETNUM ($FFA7). Преобразува от две- до четириразреден шестнайсетичен низ в едно- или двубайтово шестнайсетично число и го записва в многоцелевия псевдорегистър от нулевата страница A2L,A2H (адреси $3E — $3F). Подпрограмата GETNUM изисква предварително във входния буфер да се зареди шестнайсетичният низ с установен старши бит, неговият край да се маркира с празен низ (код $00), а регистър Y да се нулира. След изпълнението на подпрограмата младшият байт на шестнайсетичното число е записан в клетка A2L, а старшият — в клетка A2H. Ако полученото шестнайсетично число е едноразредно, в клетка A2H има нула.

При изпълнението на подпрограмата GETNUM се променя съдържанието на регистри A, X и Y.

Подпрограма INSTDSP ($F8D0). Дезасемблира инструкцията на микропроцесора (с дължина от един до три байта), чийто начален адрес се намира в клетките от нулевата страница PCL, PCH (адреси $3A — $3B), извежда я с помощта на подпрограмата COUT на текущото изходно устройство и записва нейния формат в клетка FORMAT (адрес $2E) и нейната дължина в клетка LENGHT (адрес $2F). Подпрограмата INSTDSP съответства на команда L на програмата МОНИТОР, но за разлика от нея дезасемблира само една инструкция. Тя се извиква, след като началният адрес на инструкцията се запише

на адреси PCL, PCH. Така например инструкцията с начален адрес $F800 се разпечатва с програмата:

```
LDA #$00
STA $3A
LDA #$F8
STA $3B
JSR $F8D0
```

Внимание! При работа с тази подпрограма под управление на ДОС3.3 и при отворен файл всяко извикване на програмата на ДОС RWTS завършва с конфликт в нулевата страница.

Изпълнението на подпрограмата INSTDSP променя съдържанието на регистри A, X и Y.

Подпрограма LIST ($FE5E). Отговаря на команда L на програмата МОНИТОР. Дезасемблира 20 последователни инструкции на микропроцесора и чрез подпрограмата COUT ги извежда на текущото изходно устройство. Изисква адресът на първата инструкция да е зареден в клетки PCL,PCH, а регистър X да е нулиран. Например:

```
LDA #$STARTL          ;Младшият байт на адреса на
STA $PCL              ;първата инструкция в PCL, а
LDA #$STARTH          ;старшият байт — в PCH
STA $PCH
LDX #$0
JSR $FE5E
```

Изпълнението на подпрограмата LIST или на свързаната с нея подпрограма LIST2 променя съдържанието на регистри A, X и Y.

Подпрограма LIST2 ($FE63). Това е алтернативна входна точка на подпрограмата LIST. В зависимост от съдържанието на регистър A дезасемблира от 1 до 255 инструкции, започвайки от инструкцията, чийто начален адрес се намира в клетки PCL,PCH. Например:

```
LDA #$STARTL
STA $PCL
LDA #$STARTH
STA $PCH
LDA #$64              ;Дезасемблирай 100 инструкции
JSR $FE63
```

Подпрограма REGDSP ($FAD7). Отговаря на команда CTRL-E на програмата МОНИТОР. Използва подпрограмите CROUT и COUT, за

да изпрати управляващия символ CR и да изведе съдържанието на регистрите на микропроцесора на текущото изходно устройство.

Изпълнението на подпрограмата REGDSP или на свързаната с нея подпрограма REGDSP1 променя съдържанието на регистри A и X.

Подпрограма REGDSP1 ($FADA). Това е алтернативна входна точка на подпрограмата REGDSP, но за разлика от нея не изпраща управляващия символ CR към текущото изходно устройство.

Подпрограма MOVE ($FE2C). Копира (премества) съдържанието на блок от паметта. Отговаря на командата M от програмата МОНИТОР. Преди изпълнението на подпрограмата MOVE е необходимо:

1. Да се установи началният адрес на оригинала (областта, която ще се копира) в многоцелевия псевдорегистър от нулевата страница A1L,A1H (адреси $3C — $3D).
2. Да се установи крайният адрес на оригинала в многоцелевия псевдорегистър от нулевата страница A2L,A2H.
3. Да се установи началният адрес на копието (областта, в която ще се копира) в многоцелевия псевдорегистър от нулевата страница A4L,A4H (адреси $42 — $43).
4. Да се нулира регистър Y.

Например:

```
LDA #$STARTL        ;Младшият байт на началния адрес на
STA $A1L            ;оригинала в A1L, а старшият байт -
LDA #$STARTH        ;в A1H
STA $A1H
LDA #$ENDL          ;Младшият байт на крайния адрес на
STA $A2L            ;оригинала в A2L, а старшият байт -
LDA #$ENDH          ;в A2H
STA $A2H
LDA #$DESTL         ;Младшият байт на началния адрес на
STA $A4L            ;копието в A4L, а старшият байт — в
LDA #$DESTH         ;A4H
STA $A4H
LDY #$00            ;Нулиране на регистър Y
JSR $FE2C           ;Извикване на подпрограмата MOVE
```

Изпълнението на подпрограмата MOVE разрушава съдържанието на регистър A и нулира регистър Y.

Подпрограма PRA1 ($FD92). Изпраща управляващия символ CR, съдържанието на многоцелевия псевдорегистър A1L,A1H (като четириразредно шестнайсетично число) и тире (знак минус) към текущото изходно устройство. Извиква се от подпрограмите VERIFY,

XAM, XAM8, LIST, LIST2 и др. Нейното изпълнение променя съдържанието на регистри A, X и Y.

Подпрограма VERIFY ($FE36). Сравнява съдържанието на две области от паметта. Действието ѝ е напълно еквивалентно на действието на команда V от програмата МОНИТОР. Подпрограмата VERIFY използва изходните подпрограми на програмата МОНИТОР PRA1, CROUT, PRNTYX и COUT, за да отпечати откритите при сравнението разлики в три колони:

1. Адресът на клетката от изходната област (оригинала), за която е открита разлика.
2. Съдържанието на тази клетка.
3. Съдържанието в скоби на съответната клетка от областта, с която се извършва сравнението.

За изпълнението на подпрограмата VERIFY е необходимо да се установят четири групи параметри (вж. подпрограма MOVE):

1. Началните адреси на двете области, които ще се сравняват; на оригинала — в регистър A1L,A1H, а на областта, с която ще сравнява — в регистър A4L,A4H.
2. Крайният адрес на оригинала в регистър A2L,A2H.
3. Да се нулира регистър Y.

Изпълнението на подпрограмата VERIFY разрушава съдържанието на регистър A и нулира регистър Y.

Подпрограма XAM ($FDB3). Използва изходните подпрограми PRA1, PRBYTE и COUT, за да изведе съдържанието на определена област от паметта на текущото изходно устройство. Еквивалентна е на командата SSSS.EEEE от програмата МОНИТОР, където SSSS е началният, а EEEE — крайният адрес на областта. За изпълнението на подпрограмата XAM е необходимо тези адреси да са установени предварително в псевдорегистрите A1L,A1H и A2L,A2H. Например:

```
LDA #$STARTL        ;Младшият байт на началния адрес -
STA $A1L            ;в A1L, старшият байт — в A1H
LDA #$START
STA $A1H
LDA #$ENDL          ;Младшият байт на крайния адрес -
STA $A2L            ;в A2L, а старшият байт — в A2H
LDA #$ENDH
STA $A2H
JSR $FDB3
```

Изпълнението на подпрограмата XAM или на свързаната с нея подпрограма XAM8 разрушава съдържанието на регистър A и нулира регистър Y.

Подпрограма XAM8 ($FDA3). Това е алтернативна входна точка на подпрограмата за извеждане на екран или на печат съдържанието на паметта. За разлика от подпрограмата XAM подпрограмата XAM8 извежда на текущото изходно устройство съдържанието най-много на осем последователни клетки от паметта на компютъра. Започва от адрес SSSS и завършва с адрес (SSSS + 7) MOD 8, където SSSS е адресът, определен от съдържанието на многоцелевия псевдорегистър A1L,A1H, а (SSSS + 7) MOD 8 е най-малкият възможен адрес, кратен на осем и по-голям от адреса SSSS.

Подпрограма GO ($FEB6). Отговаря на командата G на програмата МОНИТОР. Предава управлението на подпрограмата, чийто начален адрес е записан в псевдорегистъра A1L,A1H. За целта:

1. Извиква подпрограмата A1PC, която зарежда програмния брояч PCL,PCH от псевдорегистъра A1L,A1H.
2. Извиква подпрограмата IOREST, която зарежда регистрите на микропроцесора (без указателя на стека) от клетки ACC, XREG, YREG и STATUS.
3. Изпълнява инструкция JMP по съдържанието на програмния брояч PCL,PCH.

Подпрограмата GO се извиква, след като се установят следните парамери:

1. Началният адрес на програмата в псевдорегистъра A1L, A1H.
2. Съдържанието на регистър A в клетка ACC.
3. Съдържанието на регистър X в клетка XREG.
4. Съдържанието на регистър Y в клетка YREG.
5. Съдържанието на регистъра на състоянието на микропроцесора в клетка STATUS.

Изпълнението на подпрограмата GO променя съдържанието на регистри A, X, Y и P.

Входна точка MONZ ($FF69). Това е основната входна точка в командния процесор на програмата МОНИТОР. Нейното използване е еквивалентно на използването на командата от БЕЙСИК CALL -151.

Входна точка MON ($FF65). Това е алтернативна входна точка в командния процесор на програмата МОНИТОР. Изчиства десетичния режим на работа на микропроцесора, извиква подпрограмата BELL и преминава в MONZ.

Входни точки MINIASM ($FEEE) и MINIASM1 ($FCE9). Две равностойни входни точки в програмата МИНИАСЕМБЛЕР.

5.9. Обслужване на прекъсванията в Правец–8А

В най–общия смисъл прекъсването е метод, с който, при настъпване на определено събитие, управлението на компютъра временно се предава от една програма на друга. Прекъсванията се делят на две големи групи — апаратни и програмни, като апаратните прекъсвания могат да бъдат маскируеми, немаскируеми и т.н. Обща характеристика на прекъсванията е, че техният механизъм е заложен още в архитектурата на микропроцесора. Следователно това, какви прекъсвания съществуват в даден компютър, зависи както от вида на използвания микропроцесор, така и от архитектурата на самия компютър.

В персоналния компютър Правец–8А се поддържат и четирите вида прекъсвания, характерни за микропроцесора СМ630. Три от тях са апаратни, а четвъртото — програмно:

– RESET — начално (общо) нулиране;
– NMI — немаскируемо прекъсване;
– IRQ — маскируемо прекъсване;
– инструкция BRK; с тази инструкция по програмен път се симулира прекъсване от типа IRQ.

Прекъсването тип RESET не е обикновено прекъсване, защото не завършва с предаване на управлението на прекъснатата програма. Независимо от това то има много общи черти с "класическите" типове прекъсвания IRQ и NMI и затова се разглежда заедно с тях.

Преди да се спрем по-подробно на различните типове прекъсвания, е необходимо да изясним някои принципни положения.

Заявка за прекъсване. Като правило използването на прекъсвания в компютъра Правец–8А е свързано с обмена на данни с входно–изходните устройства. При прекъсванията (с изключение на RESET) се спира изпълнението на текущата програма, обслужва се устройството, поискало прекъсването, и се възобновява прекъснатата програма. Така например, ако определено входно–изходно устройство, работещо с прекъсвания, е готово да приеме или да предаде байт или блок от данни, то изпраща заявка за прекъсване. Това става, като неговият контролер промени състоянието на съответната линия за прекъсване: извод 29 на съединители X1 — X7 за NMI прекъсване или извод 30 на същите съединители за IRQ прекъсване.

Прието е RESET прекъсването да се стартира единствено от клавиатурата. Независимо от това сигналът *RESET постъпва и на извод 31 на съединителите X1 — X7 и при общото апаратно нулиране на системата се използва и от контролерите на входно–изходните устройства.

Приемане на заявката за прекъсване. От съединители X1 — X7 или по линия *RESET заявката за прекъсване постъпва на входове *NMI, *IRQ или *RESET на микропроцесора, където се приема.

Последователност на прекъсване. Последователността от

действия, които микропроцесорът изпълнява след приемане на заявката, се нарича последователност на прекъсване. Тази последователност е различна за различните видове прекъсвания и с изключение на RESET прекъсването е насочена към подготвяне на условията, необходими за възобновяване на прекъснатата програма.

Обслужване (обработка) на заявката за прекъсване. След като микропроцесорът изпълни последователността на прекъсване, управлението се предава на програмата за неговото обслужване.

Вектори на прекъсване. Използват се за предаване на управлението на подпрограмите за обслужване на прекъсвания. Адресите, на които се намират векторите на различните типове прекъсвания, са специфични за използвания микропроцесор. Те са фиксирани и не могат да се променят, но адресите, към които сочат, се определят от програмното осигуряване на компютъра (вж. табл. 5.14).

Таблица 5.14

Вектори на прекъсвания в Правец–8А

Тип на прекъсването	Адрес на вектора	Посока на вектора
RESET	$FFFC — $FFFD	$FA62
NMI	$FFFA — $FFFB	$03FB
IRQ/BRK	$FFFE — $FFFF	$FA40

Приоритет на прекъсванията. Последователността на приемане и обслужване на едновременно издадените заявки за прекъсвания от различен тип се определя от приоритета на прекъсванията. Най–висок приоритет има прекъсването RESET. В низходящ ред следват прекъсванията NMI, BRK и IRQ.

5.9.1. Прекъсване тип RESET

Основната задача на това прекъсване е да постави компютъра в предварително строго и еднозначно дефинирано състояние и да стартира управляващата програма. RESET се изпълнява при включване на захранването, а при включен компютър — с едновременното натискане на клавиши CONTROL и RESET. В зависимост от това, как е осъществено то, в персоналния компютър Правец–8А се различават няколко вида прекъсвания тип RESET — студен старт, топъл старт, принудителен студен старт и топъл старт с предаване на управлението на програмата МОНИТОР. Независимо от вида си този тип прекъсване винаги има две независими една от друга страни:

апаратна, която е обща за всички видове прекъсвания тип RESET и се нарича начално (общо) нулиране или само нулиране, и програмна, която е специфична за различните видове RESET прекъсвания.

Начално нулиране. Началното нулиране на компютъра е следствие от активирането на сигнала *RESET и се свежда до нулиране на микропроцесора, схемата за управление на входно–изходните устройства, програмноуправляемия ключ MODE, определящ режима на работа на компютъра, клавиатурата и всички контролери на входно–изходни устройства, свързани към линията *RESET.

При нулиране микропроцесорът изпълнява следната последователност:

1. Управляващата линия R/*W (четене/запис) се установява в логическа единица (четене).

2. Извличат се три невалидни байта от върха на стека (при нулиране стекът винаги съдържа невалидна информация).

3. Установява се в единица флаг I (ЗАБРАНЕНО IRQ — бит 2 на регистъра на състоянието на микропроцесора) и с това се забраняват прекъсванията тип IRQ.

4. В програмния брояч на микропроцесора се зарежда съдържанието на RESET вектора.

5. Управлението се предава на програмата за обработка на RESET прекъсвания.

Схемата за управление на паметта, респ. ключовете за управление на паметта, също се нулират при начално нулиране на компютъра, но това не става директно (схемата не е свързана към сигнал *RESET), а косвено — в резултат от изпълнението на последователността на прекъсване. Схемата за управление на паметта непрекъснато следи състоянието на адресната шина на микропроцесора и когато разпознае трите последователни обръщения към стека, следвани от обръщение към адрес $FFFC (младши байт на RESET вектора), си изработва собствен сигнал за нулиране.

В резултат на началното нулиране компютърът се установява в режим MODE0 (седембитова кодова таблица), а клавиатурата — в режим латиница. Избира се стандартното символно множество. Разрешен е достъпът до системната оперативна памет и четенето от постоянната памет в областта $D000 — $FFFF. Оперативната памет в същата област е разрешена за запис, като на адреси $D000 — $DFFF е избрана втора страница. Ако има допълнителна оперативна памет, достъпът до нея е забранен, но съединителят X3 е резервиран за ползване от програмното осигуряване за поддържане на режим ТЕКСТ80.

Подпрограма RESET ($FA62). Подпрограмата RESET е общата подпрограма за обслужване на RESET прекъсвания в персоналния компютър Правец–8А. Нейното изпълнение дава следните резултати:

1. Преминава се в режим на двоична аритметика.

2. Екранът се установява във формат НОРМАЛНО ВИДЕО.

3. Извиква се подпрограмата INIT, която установява текстов режим и нормализира размерите на текстовия прозорец.

4. Стандартният изход на компютъра се свързва с видеомонитора.

5. Стандартният вход на компютъра се свързва с клавиатурата.

6. Инициализират се цифровите изходи на компютъра, като ЦИЗХ0 и ЦИЗХ1 се установяват в логическа нула, а ЦИЗХ2 и ЦИЗХ3 — в логическа единица.

7. Проверява се състоянието на цифровия вход ЦВХ1. Ако то е логическа единица (функционален бутон F2 е натиснат), микропроцесорът изпълнява инструкция JMP $FF69 — безусловен преход към подпрограмата MONZ. Това е т.нар. топъл старт с предаване на управлението на програмата МОНИТОР.

8. Ако цифровият вход ЦВХ1 е в състояние логическа нула, проверява се състоянието на цифровия вход ЦВХ0. Ако то е логическа единица, съдържанието на адреси $3F3 и $3F4 се разрушава и компютърът изпълнява принудителен студен старт.

9. С обръщение към адрес $CFFF се забранява достъпът до постоянните памети от областта $C800 — $CFFF, разположени на контролерите на допълнителните входно–изходни устройства.

10. Нулира се клавиатурният буфер.

11. Повторно се преминава в режим на двоична аритметика.

12. Извиква се подпрограмата BELL (адрес $FF3A).

13. Проверява се дали т.нар. вторичен RESET вектор, който се намира на адреси $3F2 — $3F3 и определя къде ще се предаде управлението след обработване на RESET прекъсването, е валиден. За целта се сравняват съдържанието на флаг ВКЛЮЧЕНО (адрес $3F4) и на клетка $3F3. Ако съдържанието на флаг ВКЛЮЧЕНО е равно на резултата от логическата операция ИЗКЛЮЧВАЩО ИЛИ върху съдържанието на клетка $3F3 и стойността $A5, се изпълнява топъл старт. В противен случай се изпълнява студен старт.

С това общият клон на подпрограмата за обработване на прекъсването тип RESET завършва.

Студен старт на компютъра. Студеният старт на компютъра се изпълнява при включване на захранването, при едновременното натискане на клавиши CONTROL, RESET и F1 (принудителен студен старт) или при едновременното натискане на клавиши CONTROL и RESET и невалиден флаг ВКЛЮЧЕНО. Студеният старт протича в следната последователност:

1. Изчиства се екранът и в неговия горен край се изписва надписът ПРАВЕЦ.

2. Актуализира се съдържанието на вторичния BREAK вектор (адреси $3F0 и $3F1).

3. Установяват се вторичният RESET вектор и флагът ВКЛЮЧЕ-

НО, с което се подготвят условия за изпълнение и на топъл старт. Във вторичния RESET вектор се записва $E000 — адресът на входа в резидентния интерпретатор на БЕЙСИК с инициализация (отговаря на изпълнението на команда CTRL-B от програмата МОНИТОР).

4. Проверява се дали в компютъра има контролер на дисково устройство, като се започва от съединител X7. Ако такъв контролер се открие, студеният старт завършва, като управлението се предава на драйверната програма за обслужване на дисковото устройство.

5. Ако в компютъра не е включен контролер на дисково устройство, съдържанието на вторичния RESET вектор се модифицира, така че да сочи $E003 — адресът на входа в резидентния интерпретатор на БЕЙСИК без инициализация (отговаря на изпълнението на команда CTRL-C от програмата МОНИТОР) и студеният старт завършва с преминаване в БЕЙСИК без инициализация.

Топъл старт. Топлият старт се изпълнява при едновременното натискане на клавиши CONTROL и RESET, при условие че флаг ВКЛЮЧЕНО е валиден, т.е. ако е изпълнено условието

($3F4) = ($3F3) EOR $A5

Тази разновидност на RESET прекъсването дава възможност управлението да се предаде на вторичния RESET вектор. И тъй като той е разположен в оперативната памет и е възможно да се модифицира програмно, управлението практически може да се предаде на всяка програма, намираща се в паметта на компютъра. Нормално топлият старт се използва за рестартиране на управляващата програма. Това може да бъде резидентният интерпретатор на БЕЙСИК (дисковата операционна система не е заредена), активният интерпретатор на БЕЙСИК (дисковата операционна система е заредена и е модифицирала RESET вектора) или потребителска програма, която е модифицирала RESET вектора.

При смяна на RESET вектора е необходимо да се промени и флаг ВКЛЮЧЕНО. В противен случай обработката на RESET прекъсването завършва със студен старт. Актуализирането на този флаг може да стане с подпрограма SETPWRC (адрес $FB6F).

Следният пример показва как топлият старт може да се свърже с команда RUN от БЕЙСИК (адрес $D566).

```
LDA #$66
STA $3F2
LDA #$D5
STA $3F3
JSR SETPWRC
RTS
```

5.9.2. Прекъсвания тип NMI

Заявка за NMI прекъсване се изработва от контролерите на входно–изходните устройства и се възприема от микропроцесора по отрицателния преден фронт на сигнала *NMI. След приемането на заявката микропроцесорът извършва следните действия:

1. Завършва изпълнението на текущата инструкция.
2. Запазва съдържанието на програмния брояч в стека, като първо записва старшия, а после и младшия байт.
3. Запазва съдържанието на регистъра на състоянието в стека.
4. Забранява прекъсванията IRQ, като вдига флаг I от регистъра на състоянието на микропроцесора.
5. Зарежда младшия байт на програмния брояч от адрес $FFFA и старшия байт — от адрес $FFFB, като по този начин предава управлението на програмата, започваща от адрес $3FB (вж. табл. 5.14).

В програмното осигуряване на компютъра Правец–8А няма подпрограма за обработване на NMI прекъсване. Няма и апаратни средства за блокиране на нови NMI прекъсвания, докато се обработва първото.

При създаването на допълнителни модули, използващи NMI прекъсване, и на програми за тяхното поддържане трябва да се има предвид общата схема, по която се изпълняват този тип прекъсвания:

1. Съответният модул подава заявка за NMI прекъсване, като установява линия *NMI в състояние логическа нула (ниско ниво).
2. Подпрограмата за обслужване на NMI прекъсване се изпълнява, като се забранят нови прекъсвания NMI и IRQ (на линия *NMI се поддържа ниско ниво, а флаг I е установен в единица).
3. След обслужване на заявката модулът установява високо ниво на линия *NMI.
4. Възстановява се съдържанието на регистъра на състоянието и се възобновява изпълнението на прекъснатата програма.

5.9.3. Прекъсване тип IRQ/BRK

Издаването на заявка за IRQ прекъсване, нейното приемане и обслужване протича по следната обща схема:

1. Контролерът на входно–изходно устройство издава заявка за IRQ прекъсване, като установява ниско ниво на линия *IRQ.
2. Забраняват се нови прекъсвания IRQ и управлението се предава на съответната подпрограма за обслужване на този тип прекъсване.
3. Обслужващата програма сигнализира на контролера, че прекъсването е прието и той установява високо ниво на линия *IRQ. Така се разрешават апаратно (но не и програмно!) нови IRQ прекъсвания.
4. В зависимост от конкретното приложение обслужващата програма разрешава нови IRQ прекъсвания, като нулира флаг I. Ней-

ното изпълнение завършва с възстановяване на съдържанието на регистъра на състоянието, след което се възобновява изпълнението на прекъснатата програма.

В персоналния компютър Правец–8А заявката за IRQ прекъсване се изработва от контролер на входно–изходно устройство и се възприема от микропроцесора по ниско ниво на сигнала *IRQ. След нейното приемане микропроцесорът извършва следните действия:

1. Завършва изпълнението на текущата инструкция.

2. Проверява състоянието на флаг I. Ако този флаг не е установен, микропроцесорът започва да изпълнява последователността на IRQ прекъсване.

3. Запазва съдържанието на програмния брояч в стека, като първо записва старшия, а после и младшия байт.

4. Нулира бит 4 (флаг BREAK — ПРОГРАМНО ПРЕКЪСВАНЕ) на регистъра на състоянието на микропроцесора и запазва съдържанието на регистъра в стека. По-късно флаг BREAK се използва от общата подпрограма за обслужване на прекъсване IRQ/BRK за определяне на типа на прекъсването (IRQ или BRK).

5. Забранява следващите IRQ прекъсвания, като вдига флаг I в регистъра на състоянието.

6. Зарежда младшия байт на програмния брояч от адрес $FFFE и старшия байт от адрес $FFFF и предава управлението на подпрограмата, започваща от адрес $FA40 (вж. табл. 5.13).

Инструкция BRK на микропроцесора СМ630 е инструкция за генериране на програмно прекъсване. BRK прекъсването по същество представлява програмна симулация на IRQ прекъсване. Двете прекъсвания имат общ вектор на прекъсване, респ. обща подпрограма за обслужване (вж. табл. 5.14), която определя типа на прекъсването единствено по състоянието на флаг BREAK. Последователността от действия на микропроцесора при BRK прекъсване се различава от тази при IRQ:

1. Независимо от състоянието на флаг I микропроцесорът започва да изпълнява последователността на прекъсване.

2. Увеличава съдържанието на програмния брояч с две и го запазва в стека, като първо записва старшия, а после и младшия байт.

3. Установява флаг BREAK на регистъра на състоянието на микропроцесора и запазва съдържанието на регистъра в стека.

4. Забранява следващите IRQ прекъсвания, като вдига флаг I в регистъра на състоянието.

5. Зарежда младшия байт на програмния брояч от адрес $FFFE и старшия байт от адрес $FFFF и предава управлението на подпрограмата IRQ (адрес $FA40 — вж. табл. 5.14).

След като обслужването на IRQ/BRK прекъсването се поеме от подпрограмата IRQ, то продължава в следната последователност:

1. Съдържанието на регистър А се запазва в клетката от нулевата страница ACC (адрес $45). Оттук следва, че програмите, зако-

то съдържанието на този адрес е критично, не могат да работят с IRQ прекъсвания, а съответно и с контролери на входно–изходни устройства, използващи такива прекъсвания. Един от възможните начини да се разреши този проблем е заменянето на разположената в постоянната памет подпрограма IRQ с алтернативна програма за обработване на прекъсвания IRQ/BRK, разположена на същите адреси, но в оперативната памет.

2. От стека се извлича съдържанието на регистъра на състоянието в момента на приемане на прекъсването и се записва в регистър A. Възстановява се съдържанието на стека такова, каквото е било при извикване на подпрограмата IRQ.

3. По състоянието на флаг BREAK се определя типът на прекъсването (IRQ или BRK) и в зависимост от това или се извършва преход по съдържанието на вторичния вектор IRQ от трета страница ($3FE — $3FF), или се предава управлението на подпрограмата BREAK (адрес $FA4C).

С това общата част на подпрограмата IRQ за двата вида прекъсвания завършва. До този момент съдържанията на регистри X и Y и на указателя на стека все още не са запазени в клетките от нулевата страница. Все още не са разрешени и нови IRQ прекъсвания. Съдържанието на регистър A в момента на прекъсването е запазено в клетка ACC, а на програмния брояч и на регистъра на състоянието — в стека.

Обработката на IRQ прекъсването продължава от програмата за обработка на прекъсвания, определена от вторичния IRQ вектор. При студен старт в този вектор се записва адресът на входната точка MON ($FF65) в програмата МОНИТОР. Неговото съдържание се запазва и след зареждане на ДОС3.3 и ако то не се измени от потребителя, прекъсванията тип IRQ завършват с преминаване в програмата МОНИТОР.

При BRK прекъсване управлението се предава на подпрограмата BREAK. Тя извлича от стека съдържанието, което регистърът на състоянието е имал в момента на прекъсването; при това съдържанието на указателя на стека се увеличава с единица. След това извиква подпрограмата IOSAVE1 ($FF4C), с което съдържанието на указателя на стека се намалява с две. Подпрограмата IOSAVE1, която е алтернативна входна точка на подпрограмата IOSAVE, запазва съдържанието на регистри X и Y, на регистъра на състоянието и на указателя на стека в клетки XREG, YREG, STATUS и SPNT от нулевата страница. Изпълнението на подпрограмата BREAK продължава с извличане на съдържанието на програмния брояч от стека и записването му в клетки PCL и PCH от нулевата страница и завършва с преход по съдържанието на вторичния BREAK вектор от адреси $3F0 — $3F1.

В този момент съдържанието на всички регистри на микропроцесора е запазено в нулевата страница на паметта. Програмният

брояч сочи инструкцията, чийто адрес е с две по-голям от този на изпълнената инструкция BRK, а стойността на указателя на стека е с единица по-малка от тази в момента на прекъсване.

След студен старт и след начално зареждане на ДОС3.3 вторичният BREAK вектор сочи началния адрес на подпрограмата OLDBRK ($FA59). Като краен резултат от нейното изпълнение инструкцията, определена от съдържанието на PCL,PCH, и съдържанието на регистрите на микропроцесора се извеждат на текущото изходно устройство и управлението се предава на програмата МОНИТОР. Ако потребителят иска да ползва своя програма за обслужване на BRK прекъсване, той трябва да запише началния ѝ адрес във вторичния BREAK вектор.

Глава 6. Резидентен интерпретатор на БЕЙСИК

6.1. Особености на резидентния интерпретатор на БЕЙСИК в Правец–8А

Резидентното програмно осигуряване на персоналния компютър Правец–8А освен програмата МОНИТОР включва и интерпретатор на БЕЙСИК с плаваща запетая, който в настоящото изложение се нарича или само БЕЙСИК, или РАЗШИРЕН БЕЙСИК (за разлика от нерезидентния ЦЕЛОЧИСЛЕН БЕЙСИК). Интерпретаторът на БЕЙСИК за компютъра Правец–8А представлява модификация на познатата ни от Правец–82 и Правец–8М версия на този език. Съществуващите разлики могат да се обособят в няколко групи.

1. В зависимост от режима на работа на компютъра (MODE0 или MODE1) може да се работи със седембитова или осембитова кодова таблица.

2. Множеството на използваните цели числа е разширено със стойността -2^{15}.

3. Дефиницията за константа е разширена, като в числените константи са включени и шестнайсетични константи. Шестнайсетичните константи могат да се използват навсякъде, където е допустимо използването на цели числа в обхвата от -2^{15} до $2^{16}-1$. Те се предхождат от символа $. Например $3, $AA, $F3C5 са шестнайсетични константи.

Множеството от символи, които могат да се използват в символните константи, е разширено, като в него са включени и малките букви от латиницата и кирилицата.

4. Дефиницията за израз е разширена, като освен символни и числени изрази и изрази за съотношение включва и форматирани и шестнайсетични изрази.

Форматираният израз се получава, като резултатът от изчисляването на обикновен числен израз се представи във вида

ААА[–]ВВВ.СС

където <ААА> са водещите интервали; <ВВВ> е цялата част, а <СС> — дробната част на резултата, получен при изчисляване на израза.

Изразът се нарича форматиран, защото при дефинирането му се фиксират:

* Общата дължина на израза. Тя представлява сума от броя на полетата за водещите интервали, за знака на израза (едно поле за знак минус и нула полета за знак плюс), за разредите на цялата и дробната част на израза и за десетичната точка. Минималната дължина на форматирания израз е 1 поле, а максималната – 10 полета.

* Броят на разредите в дробната част на израза. Задава се с цяло неотрицателно число, по–малко от броя на полетата в общата дължина на израза. Това означава, че ако при изчислението на числения израз се получи дробна част, чиято дължина е различна от определената за форматирания израз, тя трябва да се приведе към последната със закръгление или с добавяне на незначещи нули след десетичната точка (вж. оператор PRINT).

* Броят на водещите интервали и на разредите на цялата част на израза се определят от стойността на израза и от неговия знак и по същество също са фиксирани.

Форматираният израз се описва по следния начин:

форм.израз ::= *числ.израз_1* :*числ.израз_2* :*числ.израз_3*

където <*числ. израз_3*> е численият израз, който ще се форматира; <*числ. израз_1*> — целочислен израз в границите от 1 до 10, който определя общата дължина на форматирания израз; <*числ. израз_2*> — целочислен неотрицателен израз, по–малък от <*числ. израз_1*>, определящ броя на полетата на форматирания израз след десетичната точка.

Форматираните изрази намират приложение при финансово–счетоводни изчисления, където всички резултати трябва да се привеждат към т.нар. паричен формат (два знака след десетичната точка). Те могат да се използват и при форматиране на изхода към екрана или печатащото устройство.

Форматираните изрази не могат да се включват директно в числени и символни изрази и в изрази за съотношение. Те са средство за форматиране на изхода на компютъра и се използват само с оператор PRINT (т.нар. форматиран PRINT) и функция CONV.

Шестнайсетичният израз е ново понятие, характерно за интерпретатора на БЕЙСИК за компютър Правец–8А. Той представлява числен израз, резултатът от изчислението на който се представя в шестнайсетичен код. Използват се два различни формата: байтов (резултатът се представя като двуразредно шестнайсетично число) и адресен (резултатът се представя като четириразредно шестнайсетично число). Шестнайсетичният израз се описва по следния начин:

шестн.израз ::=!*байт.израз* | #*адр.израз*

където *байт.израз* ::= *чисел.израз* (изменя се в границите от 0 до 255); *адр.израз* ::= *чисел.израз* (изменя се в границите от -2^{15} до $2^{16}-1$).

Шестнайсетичните изрази не могат да се включват директно в числени и символни изрази и изрази за съотношение. Те са средство за форматиране на изхода на компютъра и се използват само с оператор PRINT.

5. Тъй като компютърът Правец–8А не работи с касетофон, съответните оператори LOAD, RECALL, SAVE, SHLOAD и STORE са премахнати. Те не са част от запазените думи на БЕЙСИК и при опит за изпълнението им компютърът издава съобщение за грешка: ?SYNTAX ERROR.

6. Добавени са три нови оператора: LINE INPUT, SETMOD и CONV.

7. Възможностите за форматиране при работа с оператор PRINT са разширени в две посоки: симулация на PRINT USING за числени променливи и отпечатване на цели числа в шестнайсетичен вид.

Ако при сравнението на двете версии на интерпретатора се изключат операторите за поддържане на обмена с касетофон, може да се твърди, че новата версия на програмния език БЕЙСИК е разширение на предишната и следователно всички програми, създадени за персоналните компютри Правец–82 и Правец–8М, могат да се използват и с Правец–8А.

В настоящата глава накратко са разгледани предназначението и синтаксисът на операторите и функциите на интерпретатора на БЕЙСИК за Правец–8А. В изложението са използвани следните конвенции и термини.

* На места двата термина *оператор* и *функция* се обединяват с термина *команда*. Това не е съвсем коректно, защото в най–широк смисъл командата е оператор или функция, изпълнявана в директен режим. Впоследствие това значение на термина е трансформирано и много често оператори като RUN, LIST, NEW и др., които се използват предимно в директен режим, се наричат команди.

* Ако изрично не е упоменато, се приема, че разглежданият оператор или функция може да се използва както в програмен, така и в директен режим. Друг е въпросът дали използването им и в двата режима е оправдано.

* Символът | се използва като разделител на алтернативни възможности. Той не е част от формата на командата.

* Големите скоби { } се използват за означаване на многократно повтарящи се параметри. Те не са част от формата на командата.

* Квадратните скоби [ ] означават опция. Те не са част от формата на командата, а затвореният в тях параметър не е задължителен.

* Номерът, под който операторите, функциите или последователностите от оператори и функции се поставят в програмата, се нарича номер на ред и се означава с *#_на_ред*. Номерата на редове се изменят от 0 до 63999.

* Специални символи. Множеството на специалните символи включва управляващите символи (вж. табл. 2.3) и препинателните знаци:

! " # $ % ' ( ) : * - = ^ ; + , < . > / ?

Специалните символи са неразделна част от формата на командата.

* Главни букви от латиницата. Командите на БЕЙСИК и имената на променливи се въвеждат с главни латински букви. Те са част от командата и трябва да се използват съобразно нейния формат.

Внимание! При работа с осембитова кодова таблица интерпретаторът приравнява малките латински букви от командите на БЕЙСИК и от имената на променливи към главни. Така например командата PRINT може да се въведе и като Print, print, pRInt и т.н. Аналогично името на променливата START може да се напише и като Start, start, STarT и т.н.

* В описанието на формата на командите на интерпретатора на БЕЙСИК се използват следните съкращения:

съобщ – всеки низ, затворен в кавички;

конст – произволна числена или символна константа;

пром – всяка числена, целочислена или символна променлива;

числ.пром – всяка числена променлива;

реал.пром – всяка реална променлива;

инд.пром – всяка индексна числена, целочислена или символна променлива;

израз – сяка константа, променлива или израз, а също и всяка валидна комбинация от тях;

симв.израз – всяка символна константа, символна променлива или символен израз;

числ.израз – всяка числена константа, числена променлива или числен израз;

форм.израз – всеки форматиран израз;

шестн.израз – всеки шестнайсетичен израз;

адрес – числен израз, променлива или константа, която се оценява като адрес на клетка от паметта; адресите на клетките от паметта се изменят от –32767 до 32767 или от 0 до 65535, като с адреси **n** и 2^{16}–**n** се извършва обръщение към една и съща клетка (**n** < 2^{15});

кол – номер на колона от екрана в режими ТЕКСТ40 и ГР40; представя се с числен израз, чиято стойност е в границите от 0 до 39;

гр.кол – номер на колона от екрана в режим ГР280; представя се с числен израз, чиято стойност е в границите от 0 до 279;

ред – номер на ред от екрана в режим ГР40; представя се с числен израз, чиято стойност е в границите от 0 до 47;

гр.ред – номер на ред от екрана в режим ГР280; представя се с числен израз, чиято стойност е в границите от 0 до 191.

6.2. Оператори и функции на интерпретатора на БЕЙСИК

Функция ABS. Изчислява абсолютната стойност на число или числен израз.

Формат: ABS (*числ.израз*)

Функция ASC. Определя ASCII кода на даден символ.

Формат: ASC (*симв.израз*)

На променливата се присвоява стойност, отговаряща на ASCII кода на символа (число в границите от 1 до 255). Ако символният израз съдържа повече от един символ, функцията ASC определя ASCII кода на първия символ от израза.

Ако първият символ от низа е CTRL–@ (ASCII код 0), компютърът издава съобщение за грешка: ?SYNTAX ERROR.

Ако низът е празен (нулев низ), компютърът издава друг тип съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Функция ATN. Изчислява аркустангенс от аргумента на функцията.

Формат: ATN (*числ.израз*)

Ъгълът се получава в радиани и е в границите от $-\pi/2$ до $\pi/2$, където $\pi = 3.14159266$.

Оператор CALL. Предава управлението на подпрограмата, написана на машинен език, чийто адрес е зададен като параметър на оператора CALL.

Формат: CALL *адрес*

Функция CHR$. Определя символа, отговарящ на зададен ASCII код.

Формат: CHR$ (*числ.израз*)

Символът се задава с целочислената стойност на аргумента, интерпретирана като ASCII код. Ако стойността на числения израз не е в границите 0 – 255, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

CHR се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът $.

Оператор CLEAR. Нулира всички числени променливи и всички елементи на числени масиви. Присвоява нулев низ на всички символни променливи и елементи от символни масиви.

Формат: CLEAR

Изпълнението на оператор CLEAR е еквивалентно на изключване и повторно включване на компютъра, съчетано с повторно зареж-

дане на програмата в неговата памет. Ако оператор CLEAR е част от изпълнявана програма, тя ще продължи да се изпълнява, ако нулирането на променливите не е оказало влияние на програмната логика.

Оператор COLOR. Установява работен цвят за графиката с малка разделителна способност..

Формат: COLOR = *числ.израз*

След изпълнението на оператор COLOR цветът, използван от оператори PLOT, VLIN и HLIN, е фиксиран и може да се смени само с изпълнението на нов оператор COLOR.

Числеиият израз <*числ.израз* >, с който се определя цветът, може да има стойност от 0 до 255, независимо че цветовете са само 16 (вж. табл. 2.6). Ако получената стойност е реална, тя автоматично се преобразува в целочислена. Стойностите, по-големи от 15, повтарят основните 16 цвята. Така например 3, 19, 35 и т.н. определят един и същи цвят.

Оператор COLOR се използва в режими ГР40, СМ40/40 и СМ40/80. Той няма ефект, ако се използва в режим ГР280. Използван в текстов режим, той определя какви ще бъдат символите, извеждани на екрана от операторите PLOT, HLIN и VLIN (вж. оператор HLIN).

Ако оператор COLOR се изпълни преди оператор GR, той не оказва влияние върху избора на работен цвят.

COLOR се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът =.

Оператор CONT. Възобновява изпълнението на прекъсната програма. Програмата продължава с изпълнение на следващата команда.

Формат: CONT

Оператор CONT се използва, ако програмата е прекъсната с операторите STOP, END или с управляващата последователност CTRL–C. Неговото използване не се препоръчва, ако изпълнението на програмата е прекъснато с едновременното натискане на клавиши CONTROL и RESET.

Ако по време на изпълнение на оператор INPUT програмата е била прекъсната с CTRL–C, тя не може да продължи с команда CONT.

Ако няма прекъсната програма, ако програмата е била модифицирана след прекъсването и/или ако компютърът е издавал съобщение за грешка след прекъсването на програмата, изпълнението на оператор CONT води до издаване на съобщението: ?CAN'T CONTINUE ERROR.

Функция CONV. Присвоява стойността на форматиран израз на символна променлива.

Формат: CONV *симв.пром = числ.израз_1: числ.израз_2 :числ.израз_3*

Дясната страна на равенството представлява форматиран израз. Обърнете внимание, че във формата на функция CONV липсва апострофът от описанието на форматирания израз.

Ако някой от числените изрази *<числ.израз_1>*, *<числ.израз_2>* или *<числ.израз_3>* не удовлетворява изискванията за форматиран израз, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Внимание! Използването на функция CONV е единственият начин за запазване на стойността, получена при изчисляването на форматиран израз.

Функция COS. Изчислява косинус от ъгъла, използван като аргумент на функцията. Ъгълът се задава в радиани.

Формат: COS (*числ.израз*)

Оператор DATA. Създава списък от елементи, които по-късно се присвояват на една или повече променливи с оператор READ.

Формат: DATA *конст* [{,}][{,*конст*}]

Операторът DATA може да се постави където и да е в програмата. Данните, въведени с него, не се присвояват на променливите, докато не се изпълни операторът READ.

Символните константи – параметри на оператор DATA, обикновено се затварят в кавички. В такъв случай те не могат да съдържат управляващите символи CTRL–M и CTRL–X и кавички.

Ако символната константа не е затворена в кавички, тя не може да съдържа CTRL–M, CTRL–X, двоеточие и запетая, а интервалите в началото на низа се игнорират. Кавичките се възприемат, ако не са предшествани от интервал.

Ако част от константите, използвани като параметри на оператор DATA, липсват, на съответната числена променлива се присвоява нула, а на съответната символна променлива – нулев низ.

Операторът DATA се използва само в програмен режим. Ако този оператор се изпълни в директен режим, не се издава съобщение за грешка, но въведените елементи не са достъпни за оператора READ.

Функция DEF FN. Дефинира функция в БЕЙСИК.

Формат: DEF FN *име* (*реал.пром*) = *числ.израз_1*

Функцията се дефинира с името си *<име>* и числения израз *<числ.израз_1>*, а впоследствие се извиква от програмата с команда: FN *име* (*числ.израз_2*). Едва тогава стойността на променливата *<реал.пром>* се определя от стойността на числения израз <числ.израз_2>. Необходимо е стойностите на всички променливи, участващи в този израз, да са изчислени или дефинирани предварително (вж. функция FN).

Численият израз, с който се дефинира функцията, трябва да е част от един единствен програмен ред, като това не означава, че ве-

че дефинирани функции не могат да бъдат част от <*числ.израз_1*>. По този начин могат да се дефинират функции с произволна сложност.

Името на функцията <*име*> може многократно да се използва в програмата. Така функцията може динамично да се предефинира.

Особености:

1. Дефинирането на функции не може да става в директен режим. В директен режим е възможно единствено обръщението към вече дефинирани функции, ако междувременно не е изпълнен оператор NEW или CLEAR.

2. Функцията не може да бъде рекурсивна, т.е. тя не може директно или индиректно да се обръща към самата себе си.

Оператор DEL. Изтрива определена група програмни редове.

Формат: DEL *#_на_ред_1* , *#_на_ред_2*

Всички програмни редове, с номера, по–големи или равни на <*#_на_ред_1* > и по–малки или равни на <*#_на_ред_2* >, се изтриват от програмата, намираща се в паметта на компютъра.

След оператор DEL трябва да има два номера на програмни редове, разделени със запетая. Нито един от тях не може да бъде отрицателно число, а вторият номер на програмен ред трябва да е по–голям или равен на първия. Ако двата номера са равни, се изтрива най–много един програмен ред. Ако в програмата не съществува ред с номер, равен на <*#_на_ред_1* >, изтриването започва от програмния ред със следващ възходящ номер. Ако в програмата не съществува ред с номер, равен на <*#_на_ред_2* >, изтриването приключва с програмния ред със следващ низходящ номер.

Ако оператор DEL се използва в програмен режим, определените редове се изтриват и програмата спира. Нейното изпълнение не може да продължи с команда CONT.

Оператор DIM. С оператор DIM се резервира място за масиви в паметта на компютъра.

Формат: DIM *пром* (*инд* [{,*инд* }]) [*пром* (*инд* [{,*инд* }])]

Масивите, както и променливите, са три типа: целочислени, реални и символни. Типът на масива се определя от неговото име. Ако променливата <*пром* >, участваща във формата на командата, е целочислена променлива, дефинираният масив е целочислен. Ако тя е реална или символна – масивът е реален, респ. символен. Елементите, от които е изграден масивът, са от един и същи тип и съответстват на типа на масива.

Масивите могат да бъдат едномерни, двумерни и т.н. Размерността на масива се определя от броя на индексите, а големината му – от общия брой на елементите в масива. И големината, и размерността на масивите, които могат да се използват при работа с интерпретатора на РАЗШИРЕН БЕЙСИК, са ограничени от свобод-

ната памет на компютъра. Така например размерността на използваните масиви не може да надхвърля 88. Дефинирането на 89–мерен масив или на масив, който по някакъв друг начин надхвърля свободната памет на компютъра, води до издаването на съобщение за грешка: ?OUT OF MEMORY ERROR.

Достъпът до елементите на масива се извършва, като се използват индексни променливи. Името на индексната променлива, стойността на индексите и техният брой трябва да отговарят на дефиницията на масива. Ако се извърши обръщение към масив с индекс, по–голям от дефинирания, или към масив с неправилна размерност, компютърът издава съобщението: ?BAD SUBSCRIPT ERROR.

Ако се извърши обръщение към елемент от недефиниран масив, интерпретаторът на БЕЙСИК приема, че дължината на този масив е 11 по всички размери.

Не се допуска повторно дефиниране на масива. При опит да се направи това компютърът издава съобщението: ?REDIM'D ARRAY ERROR.

Оператор DRAW. По предварително създадена таблица (вж. приложение 7) изчертава фигура в графика с голяма разделителна способност (режими ГР280, СМ280/40, СМ280/80).

Формат: DRAW *числ.израз* [AT *гр.кол* , *гр.ред*]

Фигурата, чийто номер е определен от целочислената стойност на числения израз, се изчертава в текущия цвят в предварително определен мащаб и ъгъл на завъртане (вж. оператори HCOLOR, SCALE и ROT). Изчертаването започва от точка с координати, определени от целочислените стойности на числените изрази <*гр.ред* > и <*гр.кол* >. Ако те не са дефинирани, се използват координатите на последната точка, изчертана с команда DRAW, XDRAW или HPLOT. Номерът на фигурата, зададен с <*числ.израз* >, може да се изменя от 0 до 255, но не трябва да надхвърля броя на фигурите в таблицата.

Командата DRAW не трябва да се използва, ако в паметта на компютъра няма таблица на фигури, защото това може да доведе до блокиране на компютъра, до изчертаване на случайни фигури върху неговия екран или до изтриване на част от програмата.

Оператор END. Безусловно спира изпълнението на програмата.

Формат: END

В зависимост от структурата на програмите, написани на БЕЙСИК, в тях може и да липсва оператор END.

След изпълнението на оператор END компютърът не издава съобщение.

Функция EXP. Константата E = 2,71828183 (основа на натуралните логаритми) се повдига на степен, определена от аргумента на функцията.

Формат: EXP (*числ.израз*)

Оператор FLASH. Установява формат МИГАЩО ВИДЕО в режими ТЕКСТ40 и ТЕКСТ80.

Формат: FLASH

Съобщенията за грешки, съобщенията, изпращани към екрана с оператор PRINT, и символите, извеждани на екрана като ехо от клавиатурата, се изобразяват във формат МИГАЩО ВИДЕО. Не се засягат съобщенията, изпращани като ехо от оператор INPUT, и изобразените вече символи.

Оператор FLASH изменя ASCII кодовете на символите, изпращани към текущото изходно устройство. По тази причина след установяването на формат МИГАЩО ВИДЕО не трябва да се правят записи в текстови файлове на дисковото устройство.

Използването на оператор FLASH е ограничено от разширения символен набор и не се препоръчва.

Функция FN. Извлича функция, дефинирана от потребителя.

Формат: FN *име* (*числ.израз_2*)

С <*име*> е означено името на променливата, използвано при дефиниране на функцията, а <*числ.израз_2* > се изчислява и неговата стойност се присвоява навсякъде, където променливата се среща в дефиницията на функцията (вж. DEF FN).

При опит за използване на функция FN преди да е изпълнен оператор DEF FN компютърът издава съобщение за грешка: ?UNDEF'D FUNCTION ERROR.

Функцията не може да бъде рекурсивна, т.е. тя не може да се обръща към самата себе си или към други функции, които се обръщат към нея.

Оператори FOR – TO – STEP. Изпълнява се цикъл от инструкции, който се повтаря дотогава, докато една автоматично променяща се променлива не достигне определена стойност. Началото на цикъла се задава с операторите FOR – TO, а краят му – с оператор NEXT.

Формат: FOR *реал.пром* = *числ.израз_1* TO *числ.израз_2* [STEP *числ.израз_3*]

При първото изпълнение на оператор FOR на реалната променлива се присвоява стойността на първия числен израз. Изпълняват се операторите от тялото на цикъла (между операторите FOR и NEXT) и към стойността на променливата се прибавя или стойността на третия числен израз, или единица (ако оператор STEP липсва).

Получената стойност се сравнява със стойността на втория числен израз и ако двете стойности са равни, цикълът завършва и управлението се предава на оператора, поставен след оператор NEXT. В противен случай цикълът се изпълнява още веднъж и процесът на сравнение се повтаря.

Организирането на FOR – NEXT цикъл има следните особености:

1. Цикълът се изпълнява поне веднъж.

2. Началната и крайната стойност на променливата, както и нейното нарастване се определят еднократно с трите числени израза още при изпълнението на първия оператор FOR. Това означава, че ако стойността на който и да е от тези изрази се измени по време на изпълнението на цикъла, това няма да окаже никакво влияние върху броя на циклите.

3. Броят на изпълняваните цикли може да се изменя динамично, ако стойността на променливата се изменя в тялото на цикъла.

4. Не се препоръчва цикълът да се започва в основната програма и да завършва в подпрограма.

5. Допуска се да се организират до 10 цикъла в цикъл, ако всеки цикъл има различна променлива и всеки вграден цикъл изцяло се съдържа в обхващащия го външен цикъл.

6. Използването на оператор FOR в директен режим се допуска, ако целият цикъл е въведен на един програмен ред. Ако оператор NEXT липсва, операторите от тялото на цикъла ще се изпълнят само веднъж.

Команда FP. Това е команда за преминаване от ЦЕЛОЧИСЛЕН в РАЗШИРЕН БЕЙСИК.

Формат: FP

Командата FP е команда от операционната система, а не от интерпретатора на БЕЙСИК. Тя е тясно свързана с програмата на БЕЙСИК и затова се разглежда тук. По принцип тази команда се използва, ако интерпретаторът на ЦЕЛОЧИСЛЕН БЕЙСИК е зареден в компютъра, но това не е задължително. И в двата случая изпълнението ѝ е еквивалентно на студен старт на интерпретатора на РАЗШИРЕН БЕЙСИК: програмата, написана на БЕЙСИК, се изтрива от паметта на компютъра, установяват се формат НОРМАЛНО ВИДЕО и максимална скорост на обмен с входно–изходните устройства, изключва се режимът на проследяване, установен с команда TRACE, и се възстановяват нормалните стойности на LOMEM и HIMEM (вж. оператори HIMEM: и LOMEM:).

Команда FP се използва само в директен режим.

Функция FRE. Дава броя на байтовете свободна памет, които могат да се използват от програмата.

Формат: FRE (*числ.израз*)

Приема се, че свободна е тази памет, която се намира под об-

ластта, запазена за символни променливи, и над областта, запазена за масиви. Ако обемът на свободната памет е по-голям от 32767 байта, функция FRE дава отрицателно число. Действителният обем свободна памет се получава, като към това число се прибави 65536.

Ако при изпълнение на програмата даден низ се модифицира, неговата стара стойност остава в паметта на компютъра. Функция FRE изтрива неизползваните низове от съответната област. Ето защо се препоръчва в програмите, които работят с низове, периодично да се изпълнява функция от вида A = FRE (0).

Числеиият израз, участващ във формата на функцията, не се използва от нея, но се изчислява и ако е невалиден, се генерира съобщение за грешка.

Оператор GET. Приема един символ от входния поток. Когато символите се въвеждат от клавиатурата, не е необходимо натискането на клавиш RETURN. Приетият символ не се извежда на екрана.

Формат: GET *пром*

При изпълнение на оператор GET програмата спира и изчаква въвеждането на един символ от клавиатурата.

Ако като параметър на оператор GET е използвана символна променлива, въведеният символ се присвоява на нея. Ако това е управляващият символ CTRL–@, на символната променлива се присвоява нулев низ.

Ако за параметър на оператор GET е използвана числена променлива, разпознават се само числата от 0 до 9. При въвеждането на какъвто и да е друг символ на числената променлива се присвоява стойност нула. При това:

* ако е въведен знак минус, знак плюс, точка, интервал, CTRL–@ или главната латинска буква E, програмата продължава, без да се издава съобщение за грешка;
* ако въведеният символ е запетая или двоеточие, програмата спира и компютърът издава съобщение за грешка: ?EXTRA IGNORED;
* ако това е някой от останалите символи, програмата спира и компютърът издава друг тип съобщение за грешка: ?SYNTAX ERROR.

Оператор GET не може да се използва в директен режим.

Оператор GOSUB. Изпълнява се безусловен преход към началото на подпрограма. След изпълнението на оператор RETURN управлението се връща на командата, поставена непосредствено след оператор GOSUB.

Формат: GOSUB *#_на_ред*

Оператор GOSUB се използва за извикване на подпрограми. Входната точка на подпрограмата (нейното логическо начало) трябва да съвпада с програмния ред, определен с <*#_на_ред*>.

Оператор GOSUB може да се постави където и да е в програма-

та. Това означава, че преминаването към подпрограма може да стане от която и да е точка на основната програма.

Подпрограмите могат да извикват други подпрограми или рекурсивно да се обръщат към самите себе си. Допуска се вграждането на 25 подпрограми една в друга.

Излизането от подпрограмата става с оператор RETURN. Допуска се да се използва и оператор GOTO, ако преди това се изпълни оператор POP.

Оператор GOTO. Това е оператор за безусловен преход към определен ред на програмата.

Формат: GOTO *#_на_ред*

Изпълнението на програмата продължава с първата команда от указания програмен ред. Ако в програмата не съществува такъв ред, компютърът издава съобщение за грешка: ?UNDEF'D STATEMENT ERROR.

Оператор GR. Установява режим на графика с малка разделителна способност (режим СМ40/40 или режим СМ40/80).

Формат: GR

При изпълнение на команда GR частта от екрана, съответстваща на първите 20 реда от текстовия екран, се преобразува в поле за графика с малка разделителна способност, черният цвят се установява като работен цвят (COLOR = 0) и графичното поле се изчиства (запълва се с черен цвят). Последните 4 реда от екрана остават в текстов режим, а курсорът се премества в текстовия прозорец. Установеният текстов режим не се променя. С допълнителни команди е възможно да се променя формата на текста в рамките на текстовия прозорец (40 или 80 колони), без това да влияе на графиката с малка разделителна способност. Така например, ако програмното осигуряване за поддържане на 80-колонно изображение не е активно, с команда GR се установява режим СМ40/40. Команда PR#3 активира това програмно осигуряване и установява режим СМ40/80. По-нататък превключването на двата смесени режима става с последователностите ESC 4 и ESC 8.

Преминаването от графика с голяма в графика с малка разделителна способност не променя установения текстов режим, но и то има своите особености. Ако преминаването става от втора графична страница, включва се втора, а не първа текстова страница. Тази страница съвпада с началото на програмата, написана на БЕЙСИК, и обикновено не съдържа смислена информация. Проблемът може да се избегне, като преди команда GR се издаде команда TEXT.

Ако след команда GR се изпълни команда POKE –16302, 0, екранът се установява в режим ГР40. Възстановяването на текстовия прозорец, респ. на установения смесен режим, може да стане с команда POKE –16301, 0.

Оператор HCOLOR. Установява работния цвят в графика с голяма разделителна способност (режими ГР280, СМ280/40 и СМ280/80).

Формат: HCOLOR= *числ.израз*

До смяната на работния цвят, което може да стане само с изпълнението на друг оператор HCOLOR, всички команди HPLOT и DRAW се изпълняват във вече установения цвят. Стойността на числения израз е в границите 0 – 7, което отговаря на осемте налични цвята в този режим (вж. табл. 2.10). Ако стойността на числения израз е извън допустимите граници, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Оператор HCOLOR не се използва при графиката с малка разделителна способност.

Оператор HGR. Установява графика с голяма разделителна способност (режими СМ280/40 или СМ280/80) и избира първа графична страница.

Формат: HGR

В зависимост от установения текстов режим (ТЕКСТ40 или ТЕКСТ80), командата HGR установява или режим СМ280/40, или режим СМ280/80. При нейното изпълнение първите 20 реда от текстовия екран се преобразуват в поле за графика с голяма разделителна способност с размери 280 на 160 точки, графичното поле се изчиства (запълва се в черно), а установеният цвят не се изменя. Избира се първа графична страница. Последните четири реда остават в установения текстов режим и образуват текстов прозорец. Съдържанието на текстовата страница не се променя, но от нея се виждат само четири реда. Курсорът не се премества в текстовия прозорец, а остава там, където е. Както при графиката с малка разделителна способност, така и тук промяната на текстовия режим не оказва влияние на графичното поле. Ако програмното осигуряване за поддържане на 80-колонно изображение не е активно, с команда HGR се установява режим СМ280/40. Команда PR# 3 активира това програмно осигуряване и установява режим СМ280/80. Превключването на двата смесени режима става с последователностите ESC 4 и ESC 8.

При необходимост превръщането на текстовия прозорец в поле за графика с голяма разделителна способност може да стане с команда POKE –16302, 0.

Работата в режим ГР280 изисква внимателно да се планира използването на паметта на компютъра. Така например, ако програмата, написана на БЕЙСИК, е много дълга, тя може да заеме цялата графична страница или част от нея. Това означава, че при изпълнение на команда HGR част от програмата ще се изтрие. Това може да се избегне, като долната граница на областта от паметта, предназначе-

на за използване от програми, написани на БЕЙСИК, се премести с команда LOMEM: $2000 над първа графична страница.

Оператор HGR2. Установява режим ГР280 и избира втора графична страница.

Формат: HGR2

Целият екран се преобразува в поле за графика с голяма разделителна способност с размери 280 на 192 точки. Екранът се изчиства (запълва се в черно). Текущият цвят, текстовият режим и съдържанието на текстовата страница не се променят. Въведените от клавиатурата команди не се виждат на екрана, но се изпълняват правилно. Съобщенията от компютъра също не се виждат на екрана.

При работа във втора графична страница не създавайте текстов прозорец с команда POKE –16301, 0, защото на екрана ще се появи част от втора текстова страница, която не е достъпна за програмата, написана на БЕЙСИК.

Оператор HIMEM: Записва в клетка HIMEM максималния адрес от оперативната памет, достъпен за програмата, написана на БЕЙСИК, и нейните променливи.

Формат: HIMEM: *числ.израз*

Използва се за запазване на място в паметта на компютъра за програми на машинен език, за таблици на фигури, за защита на графичните страници и т.н.

Стойността на числения израз трябва да бъде в границите от –65535 до 65535, като отрицателните числа се интерпретират в допълнителен код. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Първоначално HIMEM се установява на най–големия адрес, достъпен за програмата, написана на БЕЙСИК ($C000). Ако програмата използва низове, те се записват под този адрес.

Операционните системи ДОС3.3 и ПроДОС се зареждат непосредствено под областта, резервирана за входно–изходни устройства, като при това изместват надолу HIMEM, а съответно и областта, използвана от низовете. Текущата стойност на HIMEM зависи от операционната система и от броя на файловете, с които се работи. Тя е записана на адреси $73 – $74 и може да се провери със следната команда:

```
Print # Peek ($74) * 256 + Peek ($73)
```

Установената стойност за HIMEM не се изменя от командите NEW, RUN и CLEAR и при топъл старт на компютъра.

Ако HIMEM се установи под LOMEM, компютърът издава съобщение за грешка: ?OUT OF MEMORY ERROR.

Ако при установяването на HIMEM и LOMEM не се остави дос-

татъчно място за самата програма, компютърът издава друг тип съобщение за грешка: ?PROGRAM TOO LARGE.

Всяко отваряне или затваряне на файл под управление на операционната система ПроДОС динамично премества HIMEM с около 1 Кбайт.

HIMEM се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът двоеточие (:).

Оператор HLIN. Изчертава хоризонтална права линия с предварително установен цвят в режим ГР40.

Формат: HLIN *кол_1* , *кол_2* AT *ред*

Хоризонталната линия се изчертава от <*кол_1* > до <*кол_2* > на зададения ред. Използваният цвят е определен от последния изпълнен оператор COLOR. Ако екранът е в текстов режим или ако съществува текстов прозорец и зададеният ред е с номер, по-голям от 39, операторът HLIN ще отпечата хоризонтален ред от символи, а не цветна линия.

Параметрите <*кол_1* >, <*кол_2* > и <*ред*> могат да се изменят в границите 0 – 39. Те се задават с целочислени или реални изрази, като последните се преобразуват в целочислени. Ако някой от параметрите на оператор HLIN не е в допустимите граници, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Операторите HLIN, VLIN и PLOT са предназначени за поддържане на графиката с малка разделителна способност, но могат да се изпълняват и в текстов режим, респ. в рамките на текстовия прозорец в смесените режими СМ40/40 и СМ40/80. В такъв случай те отпечатват символи, които се определят от позицията на екрана (четен или нечетен ред от графиката с малка разделителна способност), от установения цвят, от избраното символно множество и т.н. Анализирайте резултатите от изпълнението на следната програма:

```
10      Setmod 0
20      A$ = "  Основно символно множество"
30      Gosub 100
40      Setmod 1
50      A$ = "Алтернативно символно множество"
60      Gosub 100
70      End
100     Home
110     Htab 3: Print A$
120     For I = 0 To 15: Color = I
130     Vtab 4: Htab 6: Print "Четен ред,
        Color =";I:2:0:I
140     Vtab 6: Htab 4: Print "Нечетен ред,
        Color =";I:2:0:I
```

```
150     Plot 33,6: Plot 33,11
160     Vtab 12: Print "Съседни редове,
        Color=";'2:0:I
170     Hlin 30,35 At 24
180     For J = 0 To 15: Color= J
190     Vtab 13: Htab 17: Print "Color=";'2:0:J
200     Hlin 30,35 At 25
210     For K = 1 To 3000: Next
220     Next : Next
230     Return
```

Операторите PLOT, HLIN и VLIN не поддържат графиката с двойна малка разделителна способност и не отчитат наличието на екранна памет в допълнителната оперативна памет. Ето защо, ако някой от тях се изпълни в режим TEKCT80, символите се отпечатват само на нечетните колони от екрана, които се намират в основната оперативна памет. За да добиете визуална представа за поведението на оператори PLOT, HLIN и VLIN в режим TEKCT80, добавете следния програмен ред и изпълнете показаната програма още веднъж.

```
5   Print Chr$ (4); "Pr# 3"
```

Оператор HOME. В текстов режим изчиства текстовия прозорец и позиционира курсора в горния му ляв ъгъл.

Формат: HOME

Оператор HPLOT. Изчертава цветна точка, права или начупена линия в режим ГР280.

Формати:

HPLOT	*гр.кол, гр.ред*
HPLOT TO	*гр.кол, гр.ред*
HPLOT	*гр.кол_1, гр.ред_1* TO *гр.кол_2, гр.ред_2* [{TO *гр.кол_3, гр.ред_3*}]

Първият формат на командата се използва за поставяне на цветна точка на определена позиция на екрана. Цветът на тази точка се определя от последния изпълнен оператор HCOLOR.

Вторият формат на оператор HPLOT се използва за изчертаване на права линия от последната изчертана точка до точката с координати, определени от числените изрази за <*гр.кол*> и <*гр.ред*>. И в този случай цветът на линията се определя от последния изпълнен оператор HCOLOR. Ако от момента на установяване на режим ГР280 не е изчертавана точка или линия, операторът HPLOT се игнорира.

Третият формат се използва за изчертаване на права линия между две точки с определени координати или на начупена линия, свързваща серия от точки. Цветът на линията, както и цветът на всич-

ки сегменти на начупената линия, е един и същ и се определя от последния изпълнен оператор HCOLOR. Частта от линията, която попада в текстовия прозорец, се изчертава правилно, но не се вижда на екрана. Елиминирането на текстовия прозорец с команда POKE – 16302, 0 прави тази линия видима.

Команда HPLOT трябва да се издава след установяване на режим ГР280. В противен случай е възможно част от програмата, намираща се в паметта на компютъра, да бъде изтрита.

Оператор HTAB. Позиционира курсора на определена колона от текущия ред.

Формат: HTAB *числ.израз*

Курсорът се премества наляво или надясно на колоната, определена от стойността на числения израз, без да изтрива символите, през които преминава. Изразът може да бъде целочислен или реален (реалните изрази се интерпретират като целочислени), а стойността му може да се изменя в границите 0 – 255. Броенето на колоните започва от текущия ред на екрана, като най–лявата колона има номер 1. Последната колона от реда има номер 40, а първата колона от следващия ред – номер 41. Ако стойността на числения израз е равна на 0, курсорът се премества на 256–ата позиция. Ако стойността е по–голяма от 255 или по–малка от 0, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Промяната на размерите на текстовия прозорец променя функционирането на оператор HTAB.

Оператор HTAB не може да се използва пълноценно в режим ТЕКСТ80. Вместо него се използва командата

POKE $57B, *кол*

Оператори IF – THEN. Оператори IF – THEN, самостоятелно или заедно с оператор GOTO, се използват за образуване на т. нар. конструкция за условен преход, с която, ако е изпълнено дадено условие, управлението се предава на определен ред на програмата.

Формати:

IF *израз* THEN *оператор [:{оператор}]*
IF *израз* THEN *#_на_ред*
IF *израз* GOTO *#_на_ред*
IF *израз* THEN GOTO *#_на_ред*

При първия формат, ако условието, определено от израза, е удовлетворено, се изпълнява операторът, поставен непосредствено след оператора THEN (на същия програмен ред). В противен случай се изпълнява първият оператор от следващия програмен ред, като всички оператори, поставени след оператора THEN, се игнорират.

Останалите три формата представляват класическа конструкция за условен преход. При изпълнение на условието, дефинирано с *<израз >*, управлението се предава на програмния ред зададен с *<#_на_ред >*. Ако на същия програмен ред след оператори IF – THEN

има и такъв, с който трябва да се извърши безусловен преход (оператор GOTO), той трябва да бъде последен оператор в реда и трябва да има следния формат: GOTO *#_на_ред* . Ако след него има поставени други оператори, те никога няма да се изпълнят.

Най–често използваните изрази с оператори IF – THEN са изразите за съотношение. Ако с тях се сравняват символни изрази, стойността им се оценява по ASCII кодовете на включените в тях символи. Низове се сравняват символ по символ, докато се получи несъвпадение. В този момент низът, имащ символ с по–голям ASCII код в позицията на несъвпадение, се оценява като по–голям. Ако не се получи несъвпадение, по–дългият низ се приема за по–голям.

Изразът, използван с операторите IF – THEN, може да бъде и числен израз. Условието се смята за изпълнено, ако стойността на числения израз не е нула. Ако тя е нула, изпълнението на програмата продължава с първия оператор от следващия програмен ред.

С оператори IF – THEN може да се използва и символен израз. Трябва да се има предвид, че изпълнението на повече от два–три такива оператора в една програма ще генерира съобщение за грешка: ?FORMULA TOO COMPLEX ERROR.

Използването на конструкцията за условен преход с оператори IF – THEN има следните особености:

1. Ако последният символ, различен от интервал, преди оператор THEN е буквата A от латинската азбука, тя се комбинира с първата буква от оператор THEN и образува запазената дума AT. Това може да се избегне, като целият израз, използван с оператори IF – THEN (включително и буквата A), се затвори в скоби.

2. Ако на същия програмен ред, на който са операторите IF – THEN, е необходимо да се постави цикъл FOR – NEXT, той изцяло трябва да се съдържа на въпросния ред.

3. На един програмен ред може да се организира повече от една конструкция за условен преход, ако всички оператори IF – THEN са разположени на този ред. Добрият стил на програмиране обаче изисква вградените оператори IF – THEN да се заменят с логически изрази. Сравнете:

```
10 If A$ = "X" Then If B = 2 Then If C > D Then 50
10 If A$ = "X" And B = 2 And C > D Then 50
```

Оператор IN#. Свързва програмния вход на компютъра с контролера, включен в съответния съединител.

Формат: IN# n

където n = 1 – 7 е целочислена константа.

След зареждане на ДОС3.3 или ПроДОС команда IN# се разглежда като команда от операционната система и включването ѝ в програмата става с оператор PRINT и управляващата последователност CTRL–D. Например

```
100 Print Chr$ (4); "In# 6"
```

IN се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът #.

Оператор INPUT. Приема символите, въвеждани от текущото входно устройство, оценява ги и присвоява получените стойности на списък от числени и/или символни променливи.

Формат: INPUT ["*съобщение* ";] *пром* [,{*пром* }]

След изпълнение на оператор INPUT на текущата позиция на курсора се отпечатва въпросителен знак. Това е индикация, че предстои въвеждането на стойности на променливи. Въпросителният знак липсва, ако в оператора е включено <*съобщение*>. То представлява символна константа, която се отпечатва непосредствено преди въвеждането на стойността на първата променлива от списъка. Когато с оператор INPUT се присвояват стойности на повече от една променлива, всяка стойност може да се въвежда на различен ред от екрана, като различните стойности се отделят с натискане на клавиш RETURN. Компютърът отговаря с отпечатването на два въпросителни знака (??) за всяка променлива след първата. Стойностите на променливите от списъка могат да се въвеждат и на един ред, ако разделянето им става със запетая, а не с натискане на клавиш RETURN.

Ако се въведат невалидни символи (например букви вместо числена стойност), компютърът издава съобщението ?REENTER и оператор INPUT се изпълнява повторно. Ако в неговия формат е включено съобщение, то се отпечатва още веднъж и въвеждането на данни започва отначало.

Числените стойности трябва да се въвеждат само с валидни символи. Това са цифрите от 0 до 9, знак плюс, знак минус, десетична точка, символът E и допълнителен знак плюс и знак минус, т.е. всички символи, участващи в експоненциалната форма на представяне на числата. За всички числени променливи от списъка трябва да се въвежда някаква стойност.

При въвеждане на низове освен запетаята и управляващия символ CR като терминатори (символи, с които се означава краят на низа) могат да се използват кавичките и двоеточията. Това донякъде усложнява нещата. Така например, ако въвежданият низ започва с кавички, всички символи (включително запетаите и двоеточията) до следващите кавички или управляващ символ CR са част от този низ. Ако низът не започва с кавички, всички символи, включително кавичките, до следващата запетая, двоеточие или RETURN се присвояват на една променлива. Ако с един оператор INPUT се въвеждат два или повече низове, те трябва да се отделят със запетаи.

Ако се натисне клавиш RETURN вместо да се въведе низ, на съответната променлива се присвоява нулев низ. Символите, въведени след двоеточие, се игнорират, ако низът не започва с кавички.

Оператор INPUT не може да се използва в директен режим.

Команда INT. Това е команда за преминаване от РАЗШИРЕН в ЦЕЛОЧИСЛЕН БЕЙСИК.

Формат: INT

Команда INT е команда на операционната система, а не на БЕЙСИК. Използва се за преминаване от РАЗШИРЕН в ЦЕЛОЧИСЛЕН БЕЙСИК и изпълнението ѝ е еквивалентно на студен старт на интерпретатора на ЦЕЛОЧИСЛЕН БЕЙСИК (вж. команда FP). Ако той не е зареден, компютърът издава съобщение за грешка: ?LANGUAGE NOT AVAILABLE. Ако се направи опит за изпълнение на команда INT, когато дисковата операционна система не е заредена, компютърът издава съобщение за грешка: ?SYNTAX ERROR.

Команда INT се използва само в директен режим.

Функция INT. Определя целочислената стойност на числен израз.

Формат: INT (*числ.израз*)

Оператор INVERSE. В текстов режим установява екрана във формат ИНВЕРСНО ВИДЕО.

Формат: INVERSE

След изпълнение на оператор INVERSE ехото от клавиатурата и всички съобщения, включително съобщенията за грешки, се изобразяват на екрана в ИНВЕРСНО ВИДЕО (черни букви на бял фон). Не се променят вече изведените съобщения, както и ехото към екрана, изпращано при изпълнението на оператор INPUT.

Оператор INVERSE модифицира ASCII кодовете на символите, изпращани към текущото изходно устройство, включително и тези, изпращани към дисковото устройство. Затова не бива да се прави запис в текстов файл, когато текстовото изображение е във формат ИНВЕРСНО ВИДЕО.

Поради разширения набор от символи използването на команда INVERSE е ограничено.

Функция LEFT$. Отделя определен брой символи от началото на низа.

Формат: LEFT$ (*симв.израз* , *числ.израз*)

Броят на символите се определя от целочислената стойност на числения израз. Последната трябва да е в границите 1 – 255, а дължината на символния израз не трябва да надхвърля 255 символа. Ако стойността на числения израз е по–голяма от дължината на низа, функция LEFT$ извлича целия низ.

LEFT се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът $.

Функция LEN. Определя дължината на низ.

Формат: LEN (*симв.израз*)

Преброява символите в низа, включително интервалите и управляващите символи. Ако символният израз има повече от 255 символа, което е възможно при обединяване на низове, компютърът издава съобщение за грешка: ?STRING TOO LONG ERROR.

Оператор LET. Оператор LET = или само = се използва за присвояване на стойност на определена променлива.

Формат: [LET] *пром = израз*

На променливата се присвоява стойността, получена при изчисляването на израза.

Оператор LINE INPUT. Приема до 255 символа от текущото входно устройство и ги присвоява на определена символна променлива.

Формат: LINE INPUT *симв.пром*

Оператор LINE INPUT допълва възможностите на оператор INPUT, като позволява на дадена символна променлива да се присвоява низ с произволно съдържание. След изпълнение на оператор LINE INPUT курсорът остава на текущата позиция и компютърът изчаква въвеждането на един ред (до 255 символа) от текущото входно устройство.

Във формата на оператор LINE INPUT не може да се включва съобщение. С него може да се присвоява стойност само на една символна променлива.

Оператор LIST. Извежда съдържанието на програмата или част от нея на текущото изходно устройство.

Формати:

LIST *#_на_ред_1* [,|– *#_на_ред_2*]
LIST *#_на_ред_1* [,|–]
LIST [*#_на_ред_1*] ,|– *#_на_ред_2*

Ако в оператор LIST не са включени параметри, се отпечатва и/или извежда на екрана листингът на цялата програма.

Ако в оператора е включен само първият номер на ред, се отпечатва само този ред (ако съществува).

Ако в оператора са включени номерата на два реда, се извеждат всички редове от програмата, включени между тях. Ако в програмата не съществува програмен ред, указан с < *#_на_ред_1* >, отпечатването започва от реда със следващ възходящ номер. Аналогично, ако редът с < *#_на_ред_2* > не съществува, листингът завършва до реда със следващ низходящ номер.

Разделянето на номерата на програмните редове става или със запетая, или със знак минус.

Възможно е листингът на програмата да започне от нейното начало и да завърши с определен ред (вторият формат) или да започне от определен ред и да завърши с нейния край (третият формат).

Особености:

1. Операторът LIST не може да се използва с променливи и изрази на мястото на параметрите, определящи номерата на редовете.

2. При отпечатването на програмата операторът LIST добавя допълнителни интервали около имената на променливите и запазените думи, правейки листинга по-четлив. Дължината на реда от програмата се изчислява, преди операторът LIST да е добавил допълнителни интервали. Това означава, че е възможно да се постигне ефективно удължаване на програмния ред, ако при неговото въвеждане се изпускат всички несъществени интервали.

3. В интерпретатора на БЕЙСИК за персоналния компютър Правец-8А операторът LIST е преработен изцяло. При неговото изпълнение командите и съобщенията от програмата, написана на БЕЙСИК, се извеждат към изходното устройство по различен начин в двата режима на работа на компютъра. Ако текущият режим е MODE0, те се отпечатват с главни букви на латиница и кирилица независимо от това, в какъв режим е бил компютърът при въвеждане на програмата. Ако текущият режим е MODE1, командите на БЕЙСИК се отпечатват с малки букви на латиница, с изключение на първата буква, която е главна, независимо от това, в какъв режим (или на какъв компютър) е създавана програмата. Съобщенията в програмата не се променят.

Въведете следващата програма в режим MODE1 и изпълнете посочените действия. Анализирайте листинга и изпълнението ѝ в двата режима на работа на компютъра. Обърнете внимание, че:

1. В режим MODE1 имената на променливите и командите на БЕЙСИК и на дисковата операционна система могат да се въвеждат и с малки букви.

2. В режим MODE0 малките букви се изписват като главни, но вътрешното им представяне остава различно.

3. При запазване на програмата на дискета в режим MODE0 малките букви в командите на операционната система се преобразуват в главни.

```
]SETMOD1
]10 rem LIST Демо
]20 a$ = "absdef..."
]30 b$ = "ABSDEF..."
]40 c$ = "абвгде..."
]50 d$ = "АБВГДЕ..."
]60 print a$;b$;c$;d$
]70 print chr$(4); "save DEMO"

]list
10  Rem  LIST Демо
20  A$ = "absdef..."
```

```
30  B$ = "ABSDEF..."
40  C$ = "абвгде..."
50  D$ = "АБВГДЕ..."
60  Print A$;B$;C$;D$
70  Print Chr$(4); "save DEMO"

]run

absdef...ABSDEF...абвгде...АБВГДЕ...

]load DEMO
]list
10  Rem  LIST Демо
20  A$ = "absdef..."
30  B$ = "ABSDEF..."
40  C$ = "абвгде..."
50  D$ = "АБВГДЕ..."
60  Print A$;B$;C$;D$
70  Print Chr$(4); "save DEMO"

]setmod0
]LIST
10  REM  LIST ДЕМО
20  A$ = "ABSDEF..."
30  B$ = "ABSDEF..."
40  C$ = "АБВГДЕ..."
50  D$ = "АБВГДЕ..."
60  PRINT A$; B$; C$; D$
70  PRINT CHR$(4); "SAVE DEMO"

]RUN

ABSDEF...ABSDEF...АБВГДЕ...АБВГДЕ...

]SETMOD1
]load DEMO
]list
10  Rem  LIST Демо
20  A$ = "absdef..."
30  B$ = "ABSDEF..."
40  C$ = "абвгде..."
50  D$ = "АБВГДЕ..."
60  Print A$;B$;C$;D$
70  Print Chr$(4); "SAVE DEMO"
```

Функция LOG. Изчислява натурален логаритъм от числен израз.
Формат: LOG (*числ.израз*)

Ако числеиият израз е нула или отрицателно число, компютърът отпечатва съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Оператор LOMEM:. Установява в клетка LOMEM най–малкия адрес на клетка от паметта, достъпна за променливите от програмата, написана на БЕЙСИК.
Формат: LOMEM: *числ.израз*

При студен старт на интерпретатора на БЕЙСИК LOMEM се установява на $800. Всеки път, когато се добавя нов програмен ред или се редактира съществуващ ред, LOMEM се изменя. Изтриването на програмата с команда NEW също изменя LOMEM. Това означава, че ако е необходимо да се измести началото на програмата, написана на БЕЙСИК, към по–големите адреси, то трябва да стане след студен старт в БЕЙСИК или след изпълнение на команда NEW (преди да е заредена нова програма).

Стойността на числения израз трябва да бъде в границите от –65535 до 65535, като отрицателните числа се интерпретират като допълнителен код. Ако числеиият израз е извън тези граници, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Установяването на LOMEM на стойност, по–висока от HIMEM или по–ниска от текущата стойност на LOMEM, генерира съобщение за грешка: ?OUT OF MEMORY ERROR.

Текущата стойност на LOMEM е записана на адреси $69 – $6A и може да се отпечати с командата

```
Print # Peek ($6A) * 256 + Peek ($69)
```

LOMEM се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът двоеточие (:).

Функция MID$. Извлича определен брой символи от даден низ.
Формат: MID$ (*симв.израз, числ.израз_1*
[,*числ.израз_2*])

Броят на символите се определя от целочисленатa стойност на втория числен израз, а първият символ (началото) – от първия числен израз.

Ако вторият числен израз липсва, функция MID$ извлича частта от низа, заключена между първия символ (определен от <*числ.израз_1* >) и неговия край. Ако дължината на низа е по–малка от стойността на първия числен израз <*числ.израз_1* >, извлича се нулев низ. Ако от символа, определен със стойността на <*числ.израз_1*>, до края на низа има по–малко символи, отколкото са зададени с

<числ.израз_2 >, функция MID$ извлича всички символи до края на низа.

Дължината на символния низ не трябва да надхвърля 255 символа, а стойностите на двата числени израза *<числ.израз_1 >* и *<числ.израз_2 >* трябва да са в границите 1 – 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

MID се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът $.

Оператор NEW. Изтрива текущата програма, написана на БЕЙСИК, и нейните променливи от паметта на компютъра.

Формат: NEW

Изпълнението на оператор NEW установява LOMEM на $800, но не изменя текущите стойности на HIMEM, COLOR и HCOLOR.

Оператор NEXT. Това е оператор за край на цикъл от типа FOR – NEXT.

Формат: NEXT [*пром*] [,{*пром* }]

При изпълнение на оператор NEXT индексната променлива на цикъла *<пром >* се изменя със стойност, определена от съответния оператор STEP. След това в зависимост от параметрите, въведени с FOR, програмата или продължава с оператора, следващ непосредствено след оператор NEXT, или изпълнява операторите от тялото на цикъла още веднъж (за повече подробности вж. оператор FOR).

Ако при изпълнението на оператор NEXT не съществува активен оператор FOR, отговарящ на зададената индексна променлива, компютърът издава съобщение за грешка: ?NEXT WITHOUT FOR ERROR.

Ако с оператор NEXT се въвеждат няколко индексни променливи, те трябва да се дадат в подходящ ред (цикълът, който е разкрит последен, трябва да се затвори пръв). В противен случай компютърът отпечатва съобщение за грешка: ?NEXT WITHOUT FOR ERROR.

Операторът NEXT може да се използва, без да се указва името на индексната променлива, за която се отнася. В такъв случай той се изпълнява по-бързо и се отнася за последния разкрит и все още активен цикъл.

При използване на оператор NEXT в директен режим той може да предизвика преход към оператор FOR, който е изпълнен в програмен режим и все още е активен.

Оператор NORMAL. Установява формат НОРМАЛНО ВИДЕО (бели букви на тъмен фон) на текстовото изображение.

Формат: NORMAL

Оператор NOTRACE. Преустановява действието на оператор TRACE.

Формат: NOTRACE

Ако преди това не е изпълнен оператор TRACE, оператор NOTRACE се игнорира.

Оператор ONERR GOTO. Предава управлението на програмата на определен ред, ако при изпълнението ѝ възникне грешка.

Формат: ONERR GOTO *#_на_ред*

Изпълнението на оператор ONERR GOTO установява специален флаг в компютъра и ако впоследствие, при изпълнението на програмата, се получи грешка, управлението се предава на програмния ред, въведен с оператор ONERR GOTO. Ето защо този оператор трябва да се изпълни, преди да се генерира съобщение за грешка, най–добре в самото начало на програмата.

Различните типове грешки имат различен код (вж. приложение 6). Кодът на последната грешка се записва от интерпретатора на РАЗШИРЕН БЕЙСИК на адрес $DE, а на адрес $DF се поддържа стойността на указателя на стека до момента на възникване на грешка в програмата. Тези две стойности се използват от програмите за обработка на грешките при оценка на възникналата грешка и при възобновяване на нормалното изпълнение на програмата (вж. оператор RESUME).

При определени ситуации операторът ONERR GOTO не функционира правилно:

1. Компютърът блокира, ако възникнат две последователни грешки при изпълнение на оператор GET и ако подпрограмата за обработка на грешките завършва с оператор RESUME, а не с оператор GOTO.

2. В програми, използващи оператор PRINT, или програми, при които операторът TRACE е активен, четиридесет и третата грешка, непричинена от изпълнението на оператор INPUT, води до преход в програмата МОНИТОР. Ако подпрограмата за обработка на грешките обаче завършва с оператор GOTO, а не с оператор RESUME, преходът в мониторен режим се извършва на осемдесет и седмата грешка.

Оператор ON GOSUB. Оператор за условен преход към една от няколко подпрограми. Определянето на подпрограмата се извършва в зависимост от текущата стойност на даден числен израз.

Формат: ON *чисел.израз* GOSUB *#_на_ред* [,{*#_на_ред* }]

Ако целочислената стойност на числения израз е 1, програмата извършва преход към подпрограмата, започваща от реда, определен с първия номер на ред. Ако стойността е 2, преходът е към подпрограмата, започваща от реда, определен с втория номер на ред, и т.н. Ако стойността на числения израз е 0 или число, по–голямо от броя на редовете, използвани като параметри на оператор ON GOSUB, управлението се предава на оператора непосредствено след него.

Оператор RETURN, с който завършва всяка подпрограма, също връща управлението на този оператор.

Стойността на числения израз, използван за параметър на оператор ON GOSUB, трябва да бъде в границите 0 – 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Оператор ON GOTO. В зависимост от текущата стойност на определен израз изпълнява преход към една от няколко възможни точки на програмата.

Формат: ON *чисел.израз* GOTO *#_на_ред* [,{*#_на_ред* }]

Управлението се предава на програмния ред, определен от първия номер на ред, ако целочислената стойност на числения израз е 1; на програмния ред, определен от втория номер на ред, ако тази стойност е 2 и т.н. Ако стойността на израза е 0 или число, по–голямо от броя на редовете, използвани като параметри, управлението се предава на оператора непосредствено след ON GOTO. Резултатът от изчислението на числения израз трябва да бъде в границите 0 – 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Функция PDL. Определя положението на потенциометрите на ръчките за управление на игри.

Формат: PDL (*чисел.израз*)

Получената стойност е число между 0 и 255 и се определя от позицията на потенциометъра на ръчката за управление на игри, определен от стойността на числения израз. Ръчките са номерирани от 0 до 3. Ако стойността на числения израз е по–малка от 0 или по–голяма от 255, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR. Ако тя е от 4 до 255, функцията PDL извлича непредсказуема стойност в границите от 0 до 255. Често това е съпроводено със странични явления, като "бипване" на високоговорителя, преминаване в графичен режим и др.

Ако се изпълнят две функции PDL една след друга, стойността, извлечена от втората функция, може да е повлияна от стойността, извлечена от първата функция, т.е. може да е невалидна. Ето защо се препоръчва между две функции PDL да се изпълняват няколко други команди, например един празен цикъл FOR – NEXT.

Функция PEEK. Извлича съдържанието на определена клетка от паметта.

Формат: PEEK (*чисел.израз*)

Полученото число е десетичен еквивалент на байта, записан на зададения със стойността на числения израз адрес.

Оператор PLOT. В режим ГР40 предизвиква светенето на определено блокче от екрана с предварително определен цвят.

Формат: PLOT *кол, ред*

Цветът на блокчето се определя от последния изпълнен оператор COLOR, а позицията му на екрана – от параметрите *<кол >* и *<ред >*. Те могат да се изменят от 0 до 39, респ. от 0 до 47. Блокчето с координати 0,0 отговаря на горния ляв ъгъл на екрана. Блокчетата от редовете с номера от 40 до 47 попадат в текстовия прозорец, ако той съществува.

Ако оператор PLOT се използва в текстов режим или ако блокчето попада в рамките на текстовия прозорец, на екрана се изобразява символ, а не цветно блокче. Позицията на символа се определя от параметрите *<ред >* и *<кол >* на оператор PLOT. Тъй като един символ заема площ колкото две цветни блокчета, една и съща позиция от екрана се задава с две различни стойности на параметъра *<ред >*. ASCII кодът на символа се определя от последния изпълнен оператор COLOR и от това дали позицията на блокчето отговаря на горната или долната половина от позицията на символа (вж. оператор HLIN).

Оператор POKE. Записва байт данни в определена клетка от паметта на компютъра.

Формат: POKE *числ.израз_1 , числ.израз_2*

Адресът на клетката от паметта се определя от стойността на първия числен израз, а байтът данни – от стойността на втория числен израз. Последният е цяло число в границите 0 – 255.

Оператор POP. Премества указателя на стека с една позиция нагоре.

Формат: POP

Оператор POP променя посоката на предаване на управлението при излизане от подпрограма. Неговото изпълнение премества указателя на стека с една позиция нагоре, така че ако след оператор POP се изпълни оператор RETURN, управлението вместо на оператора, следващ последния изпълнен оператор GOSUB, се предава на този, следващ предпоследния изпълнен оператор GOSUB.

Ако общият брой на изпълненията в програмата оператори POP и RETURN надхвърля броя на изпълненията оператори GOSUB, компютърът издава съобщение за грешка: ?RETURN WITHOUT GOSUB ERROR.

Функция POS. Извлича текущия номер на колоната, на която се намира курсорът.

Формат: POS (*числ.израз*)

Численият израз, използван с функцията, не оказва влияние на

нейното изпълнение. Той може да има произволна, но валидна стойност.

Изпълнението на функция POS дава цяло число между 0 и 39, което показва позицията на курсора спрямо началото на екрана. Нулата отговаря на най–лявата колона от текстовия прозорец.

Особености:

1. Промяната на размерите на текстовия прозорец променя и резултата, получен при изпълнение на функция POS.

2. За функции POS и SCRN колоните на екрана са номерирани от 0 до 39, а за функции HTAB и TAB – от 1 до 40.

3. Функция POS не може да се използва в режим TEKCT80 и когато програмното осигуряване за поддържане на 80–колонно изображение е активно. В този случай тя може да се симулира с функцията PEEK ($57B).

Оператор PR#. Свързва програмния изход на компютъра с контролера на входно–изходното устройство, включен към определен съединител.

Формат: PR# n

Тук n е целочислена константа в границите 1 – 7 и отговаря на номера на съединителя.

След зареждане на операционната система (ДОС3.3 или ПроДОС) оператор PR# става част от нея и изпълнението му в програмен режим изисква команди от вида

```
Print Chr$(4);"PR# n"
```

PR се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът #.

Оператор PRINT. Изпраща символи към текущото изходно устройство.

Формат: PRINT|? {[*израз* [{;|,} [{*израз* }]}]] [,|;]}

Представеният формат е опит за обобщение на различните формати на оператор PRINT. Така например най–външните големи скоби показват, че той може да се изпълнява и без аргумент, като резултатът се свежда до изпращане на управляващите символи CR и LF към текущото изходно устройство.

Ако след оператор PRINT са поставени един или повече изрази, форматът на изхода се определя в зависимост от това, дали отделните изрази са разделени със запетая или с точка и запетая. Ако изразите са разделени с точка и запетая, техните стойности се извеждат (отпечатват) една след друга без интервал между тях. Ако изразите са разделени със запетая, екранът се разделя на определен брой полета (извършва се т.нар. табулиране) и стойностите на изразите се извеждат в тези полета.

Ако списъкът от изрази не завършва със запетая или с точка и запетая, след последния израз се издават управляващите символи CR и LF. Ако списъкът завършва с точка и запетая, първият символ, изведен от следващия оператор PRINT, се позиционира непосредствено след последния символ, изведен от текущия оператор PRINT, т.е. издаването на тези управляващи символи се потиска.

Изразите могат да бъдат поставени и един след друг, без да се разделят със запетая или точка и запетая. Това е еквивалентно на използването на точка и запетая.

Като параметри на оператор PRINT могат да се използват числени, символни, форматирани и шестнайсетични изрази. Те се оценяват и ако е необходимо, се изчисляват. Получените резултати се отпечатват, както следва:

1. Ако резултатът от изчислението на числения израз е по-малък по абсолютна стойност от 0,1 или има повече от 9 значещи цифри пред десетичната точка, той се представя в експоненциална форма. Във всички останали случаи този разултат се представя стандартно. Отрицателните числа се предхождат от знак минус (–), а пред положителните числа няма знак.

2. Низовете се извеждат без изменения.

3. Форматираните изрази се извеждат на екран или отпечатват след изчисляване на участващите в тях числени изрази. Изпълнете и анализирайте следните програми:

```
10  A = Int (8 * Rnd (1))
20  B = Rnd (1)
30  Print '10:2: B * 10 ^ A
40  Goto 10
```

```
10  For I = 0 To 8
20  Print 10:I: 6,66666666
30  Next
```

4. Шестнайсетичните изрази се извеждат след превръщане на десетичните числа в шестнайсетични. Например:

```
10  Pr# 3
20  For I = 0 To 127
30  Print '3:0:I;" – $";!!;" ";
40  Next
```

В режим ТЕКСТ80 табулирането със запетая в оператор PRINT не функционира нормално. Извеждат се само първите две стойности, а останалите се губят.

Оператор READ. Присвоява стойностите, въведени с оператори DATA, на списък от променливи.

Формат: READ *пром* [,{*пром* }]

Двойката оператори READ и DATA се "синхронизират" с показалец, определящ коя от стойностите, въведени с оператор DATA, ще се присвои на променливата от текущия оператор READ. Преди стартиране на програмата и след изпълнение на оператор RESTORE показалецът сочи първата стойност от списъка на променливите. След всяко изпълнение на оператор READ на текущата променлива, определена с оператор READ, се присвоява текущата стойност от списъка, въведен с оператор DATA, и показалецът се премества с една позиция надолу в списъка. Променливите могат да бъдат от произволен тип, но трябва да съвпадат с типа на съответната стойност от списъка. На символна променлива може да се присвои и числена стойност, но при опит да се присвои низ на числена променлива се генерира съобщение за грешка: ?SYNTAX ERROR.

Ако броят на променливите, въведени с оператор READ, е по-голям от броя на стойностите, въведени с оператор DATA, компютърът издава съобщение за грешка: ?OUT OF DATA ERROR.

Оператор READ може да се изпълнява и в директен режим при условие, че програмата, записана в паметта на компютъра, съдържа достатъчно стойности, обявени със съответни оператори DATA. В противен случай компютърът издава съобщение за грешка: ?OUT OF DATA ERROR. Ако операционната система е заредена в паметта на компютъра, изпълнението на оператор READ в директен режим се потиска от съответната команда на ДОС3.3 или ПроДОС и компютърът издава съобщение за грешка: NOT DIRECT COMMAND.

Оператор REM. Вмъква коментари в програмата.

Формат: REM *коментар*

В така показания формат <*коментар*> е всяка последователност от символи, която може да се побере на текущия програмен ред. Коментарите, включени в програмата с оператор REM, се извеждат (отпечатват) в листинга ѝ, но се игнорират при нейното изпълнение. Оператор REM се поставя или на отделен ред от програмата, или като последен оператор от реда. Ако на програмния ред след него има други оператори, те се интерпретират като коментар.

Оператор RESTORE. Нулира показалеца в списъка на стойностите, въведени с оператор DATA (установява го на първата стойност от списъка).

Формат: RESTORE

След изпълнение на оператор RESTORE следващият оператор READ извлича първата стойност от списъка на променливите, създаден с оператор DATA.

Оператор RESUME. Предава управлението на програмата в на-

чалото на програмния ред, при изпълнението на който се е получила грешка.

Формат: RESUME

Оператор RESUME се използва в края на подпрограмата за обработка на грешки, и то само ако предварително е активиран с оператор за условен преход ONERR GOTO. Ако оператор RESUME се изпълни, без да е генерирана грешка в хода на изпълнение на програмата, резултатите са непредсказуеми.

Особености:

1. Оператор RESUME не може да се използва в директен режим.

2. Оператор RESUME не винаги възстановява коректно стека. Този проблем се решава, като в програмата за обработване на получената грешка се включи и обръщение към следната подпрограма (вж. и оператор ONERR GOTO):

```
PLA
TAY
PLA
LDX $DF
TXS
PHA
TYA
PHA
RTS
```

Оператор RETURN. Предизвиква преход към оператора, следващ непосредствено след последния изпълнен оператор GOSUB (вж. оператор POP).

Формат: RETURN

Функция RIGHT$. Извлича символи от края на определен низ.

Формат: RIGHT$ (*симв.израз* , *числ.израз*)

Броят на символите, които се извличат от края на символния израз, се определя от стойността на числения израз. Тази стойност трябва да е между 1 и 255, а дължината на низа, получен при оценката на символния израз, не трябва да надхвърля 255 символа. Ако стойността на числения израз е по–голяма от дължината на низа, функцията извлича целия низ.

RIGHT се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът $.

Функция RND. Извлича случайно число.

Формат: RND (*числ.израз*)

Генерира случайно реално число, по–голямо или равно на нула и по–малко от единица. В зависимост от стойността на числения израз са възможни следните три случая:

1. Ако числениият израз е положителен, функция RND всеки път генерира различно число освен ако преди това не е стартирана периодично повтаряща се поредица.

2. Периодично повтаряща се поредица се стартира, когато функция RND се използва с отрицателен аргумент. Всяка отрицателна стойност на числения израз стартира различна поредица, която продължава дори ако по-късно се изпълни функция RND с положителен аргумент.

3. Функцията RND с нулев аргумент извлича последното генерирано случайно число. Това свойство на функцията не се влияе от командите CLEAR и NEW.

Оператор ROT. Установява ъглъла на завъртане на фигурите, изчертавани с оператори DRAW и XDRAW в графика с голяма разделителна способност.

Формат: ROT = *числ.израз*

След команда ROT = 0 фигурата се изчертава с нейната начална ориентация. При всяко увеличаване на стойността на числения израз, използван за параметър на оператора ROT, с 16 фигурата се завърта на 90°. Така команда ROT = 32 завърта фигурата на 180°, а команда ROT = 64 отново я изчертава с началната ѝ ориентация. Ако стойността на числения израз е по-голяма от 63, тя се преизчислява с MOD 64.

Когато размерът на фигурата е установен с команда SCALE = 1, ефективно се разпознават само четири стойности на числения израз, включен в оператор ROT. Това са стойностите ROT = 0, ROT = 16, ROT = 32 и ROT = 48. Ако командата е била SCALE = 2, разпознаваните стойности са осем; при команда SCALE = 3 те са 16 и т.н. Максималният брой разпознавани стойности на оператор ROT е 64. Междинните стойности се интерпретират след приравняване към най-близката по-малка разпознавана стойност.

Стойността на числения израз, използван като параметър на оператор ROT, трябва да е в границите от 0 до 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Думата ROT се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът равно (=).

Оператор RUN. Стартира програмата, намираща се в паметта на компютъра, започвайки от определен ред или от реда с най-малък номер.

Формат: RUN [*#_на_ред*]

Ако в програмата не съществува ред с такъв номер, компютърът издава съобщение за грешка: ?UNDEF'D STATEMENT ERROR.

Оператор SCALE. Установява размера на фигурата, изчертавана с оператори DRAW и XDRAW в графика с голяма разделителна способност.

Формат: SCALE = *чисд.израз*

Размерът на фигурата, описана в таблицата на фигурите, се умножава с целочислената стойност на числения израз. Ако командата е SCALE = 1, фигурата се изчертава така, както е дефинирана. Ако командата е SCALE = 2, тя се изчертава в двоен размер и т.н. Ако командата е SCALE = 255, размерът на фигурата се увеличава 255 пъти в сравнение с оригинала.

Числеиият израз, използван като параметър на оператор SCALE, трябва да бъде в границите от 0 до 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Думата SCALE не се разпознава като запазена дума в БЕЙСИК, ако първият символ след нея, различен от интервал, не е символът равно (=).

Функция SCRN. В режим ГР40 определя цвета на блокче по неговите координати.

Формат: SCRN (*кол, ред*)

Ако се работи в текстов режим или съществува текстов прозорец и зададеното блокче е в него, кодът на цвета се определя от кода на символа, намиращ се на определената от параметрите на функцията позиция. Ако блокчето е на четен ред, цветът се определя от младшите четири бита на кода на символа, записан в съответната клетка от екранната памет, а ако е нечетен – от старшите четири бита на този код. Това дава възможност да се изчисли кодът на символа, заемащ определена позиция на екрана. Например кодът на символа, намиращ се на позиция с координати H,V, се определя с израза:

$$Scrn\ (H, 2 * V) + 16 * Scrn\ (H, 2 * V + 1)$$

където H се изменя от 0 до 39, а V – от 0 до 23.

При определяне на координатите на блокчето трябва да се има предвид следното:

1. Ако параметърът <*кол*> е в границите 0 – 39, функцията SCRN дава кода на цвета на блокчето с координати <*кол*>, <*ред*>.

2. Ако параметрите <*кол*> и <*ред*> са в границите 40 – 47 и 0 – 31 съответно, функцията SCRN дава кода на цвета на блокчето с координати (<*кол*> – 40), (<*ред*> + 16).

3. Ако параметрите <*кол*> и <*ред*> са в границите 40 – 47 и 32 – 47 съответно, функция SCRN дава код на блокче, което е извън екрана.

Особености:

1. Ако функция SCRN се използва в режим ГР280, полученият код

не се отнася до графичната страница с голяма разделителна способност, а до съответната текстова страница.

2. Думата SCRN се разпознава като запазена дума в БЕЙСИК, ако първият различен от интервал символ след нея е малка лява скоба.

3. В режим ТЕКСТ80 функция SCRN не действа правилно в рамките на текстовия прозорец. Подобно на останалите команди, поддържащи изображението с малка разделителна способност (вж. HLIN, PLOT и т.н.), при нейното изпълнение не се отчита наличието на екранна памет в допълнителната оперативна памет.

Оператор SETMOD. Установява режим на работа със седем- или осембитова кодова таблица.

Формат: SETMOD 0|1

Команда SETMOD 0 установява режим MODE0 (седембитова кодова таблица), а команда SETMOD 1 – режим MODE1 (осембитова кодова таблица).

Функция SGN. Определя дали стойността на даден числен израз е положителна, отрицателна или е равна на нула.

Формат: SGN (*числ.израз*)

Изпълнението на функцията SGN дава +1, ако численият израз е положителен, –1, ако изразът е отрицателен, и 0, ако той е равен на нула.

Функция SIN. Изчислява синуса от определен ъгъл.

Формат: SIN (*числ.израз*)

Численият израз се оценява в радиани и от получената стойност се изчислява синус.

Функция SPC. Премества курсора с определен брой позиции надясно.

Формат: SPC (*числ.израз*)

Функция SPC премества курсора с толкова позиции надясно, колкото е стойността на числения израз. Броенето започва от текущата позиция на курсора за разлика от функция TAB, при която броенето започва от най–лявата колона на екрана.

Функцията SPC се използва заедно с оператор PRINT за изпращане на определен брой интервали, като символите, през които преминава курсорът, се изтриват.

SPC се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът малка лява скоба.

Оператор SPEED. Променя скоростта на обмен с текущото изходно устройство.

Формат: SPEED = *числ.израз*

Стойността на числения израз определя скоростта, с която данните се изпращат към изходното устройство. Тя може да се изменя от 0 (най–ниска скорост) до 255 (най–висока скорост).

SPEED се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът равно (=).

Функция SQR. Изчислява квадратен корен от положителен числен израз.

Формат: SQR (*числ.израз*)

Ако числеиият израз има отрицателна стойност, компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Функцията SQR се изпълнява по–бързо от еквивалентното повдигане на степен 0,5.

Оператор STOP. Спира изпълнението на програмата, написана на БЕЙСИК.

Формат: STOP

Компютърът преминава от програмен в директен режим на работа. На екрана се появява съобщението BREAK IN #_на_ред, където <#_на_ред> определя програмния ред, при изпълнението на който програмата е спряла.

Функция STR$. Преобразува числена стойност в низ.

Формат: STR$ (*числ.израз*)

Стойността на числения израз се преобразува в низ. Символите в низа са същите, каквито биха били, ако се отпечатат с командата PRINT *числ.изр*.

Ето защо:

```
STR$ (2/3) = ".666666667"
STR$ (2468013579) = "2.46801358E+09"
```

Ако числеиият израз надхвърля допустимите граници за реални числа, компютърът издава съобщение за грешка: ?OVERFLOW ERROR.

STR се разпознава като запазена дума в БЕЙСИК само ако първият символ след нея, различен от интервал, е символът $.

Функция TAB. Премества курсора на определена колона от екрана.

Формат: TAB (*числ.израз*)

Установява хоризонталната позиция на курсора, която се използва от следващия оператор PRINT. Числеиият израз трябва да е в границите 0 – 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Особености:

1. За функция TAB колоните са номерирани от 1 до 255. Ако стойността на числения израз е по–голяма от широчината на един ред от изходното устройство (40 за екрана), курсорът се премества на

следващия ред и броенето продължава (вж. оператор HTAB). Ако стойността на израза е нула, курсорът се премества на 256-а колона. Стойности извън обхвата 0 – 255 причиняват генериране на съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

2. Функция TAB се използва съвместно с оператор PRINT, за да се премести курсорът на позицията, определена от стойността на числения израз, ако тя е вдясно от текущата му позиция. Курсорът не се премества, ако текущата му позиция е вдясно от колоната, определена от стойността на числения израз, т.е. той не се връща назад.

3. Функция TAB извежда интервали при преместването на курсора надясно. Това означава, че символите, през които курсорът преминава, се изтриват от екрана.

4. Функция TAB не може да се използва в режим ТЕКСТ80. В този режим тя действа подобно на функция SPC и премества курсора спрямо неговата текуща позиция, а не спрямо лявата колона от екрана. В режим ТЕКСТ80 следните две команди са еквивалентни:

```
PRINT SPC (10)
PRINT TAB (11)
```

Функция TAN. Изчислява тангенс от зададен ъгъл.

Формат: TAN (*числ.израз*)

Числениият израз се оценява в радиани и от получената стойност се изчислява тангенс.

Оператор TEXT. След работа в графичен или смесен режим на изображение команда TEXT възстановява текстовия режим.

Формат: TEXT

След изпълнението на команда TEXT оповестяващият символ и курсорът се преместват на последния ред от екрана и текстовият прозорец възстановява нормалните си размери.

Команда TEXT не изчиства екрана или, по-точно казано, не изчиства екранната памет, използвана в режима на графика с малка разделителна способност. И тъй като тази екранна памет съвпада с паметта, използвана като първа текстова страница, след изпълнението на команда TEXT в режими ГР40, ГР80 или в съответните смесени режими първите 20 реда от екрана остават запълнени с невалидна информация.

Оператор TRACE. Отпечатва на екрана номера на програмния ред, на който се намира изпълняваният оператор.

Формат: TRACE

Този оператор е средство за откриване на грешки в програмите. Недостатък е, че неговото изпълнение води до смесване на съобще-

нията, извеждани на екрана от програмата и от оператор TRACE (номерата на изпълнените програмни редове).

Действието на оператор TRACE може да се прекрати само с оператор NOTRACE.

Функция USR. Извършва преход към подпрограма на машинен език, като същевременно записва определена стойност в акумулатора с плаваща запетая.

Формат: USR (*числ.израз*)

На адреси $A – $C се записва инструкция за преход към началото на подпрограмата на машинен език. Числеият израз се изчислява и стойността му се записва в акумулатора с плаваща запетая, намиращ се на адреси $9D – $A3.

Функция VAL. Преобразува низ в число.

Формат: VAL (*симв.израз*)

Преобразува низа, определен от символния израз, в число. Стойността на израза се приема за нула, ако първият символ не е цифра. В противен случай изразът се преобразува символ по символ до достигането на недопустим символ (допустими са цифрите от 0 до 9, интервалът, десетичната точка, знакът плюс или минус в началото на низа, а при експоненциална форма на представяне на числата и буквата E, допълнителният знак плюс или минус).

Ако символният израз е получен от сливането на два или повече низа с обща дължина над 255 символа, компютърът издава съобщение за грешка: ?STRING TOO LONG ERROR.

Ако получената числена стойност надхвърля границите на допустимите стойности за реалните числа, генерира се друг тип съобщение за грешка: ?OVERFLOW ERROR.

Оператор VLIN. Изчертава вертикална права линия в графика с малка разделителна способност.

Формат: VLIN *peg_1, peg_2* AT *кол*

Правата линия е разположена между < *peg_1* > и < *peg_2* > на определената колона. Цветът ѝ се определя от последния изпълнен оператор COLOR.

Особености:

1. Ако компютърът е в текстов режим или поне един от редовете попада в текстовия прозорец при работа в режим на смесено изображение (CM40/40 или CM40/80), цялата линия или част от нея се отпечатва като символи, а не като цветни блокчета (вж. HLIN).

2. Оператор VLIN не функционира нормално, ако текстовият прозорец е в режим TEKCT80 (вж. оператор HLIN).

Оператор VTAB. Позиционира курсора на определен ред от текущата колона.

Формат: VTAB *ред*

Курсорът се премества нагоре или надолу по текущата колона до зададения ред, без да променя отпечатаните символи. Редовете са номерирани от 1 до 24.

Оператор WAIT. Спира изпълнението на програмата, написана на БЕЙСИК, и изчаква, докато определена клетка от паметта на компютъра не приеме определена стойност.

Формат: WAIT *адрес, числ.израз_1* [,*числ.израз_2*]

Оператор WAIT проверява съдържанието на клетката, определена с параметъра <*адрес*> за определено подреждане на нули и единици. Подреждането се задава от двоичната стойност на <*числ.израз_2*>, а двоичната стойност на <*числ.израз_1* > определя кои битове на посочената клетка да се вземат под внимание и кои да се игнорират. При това, ако даден бит на <*числ.израз_1* > е единица, съответният бит на байта, записан на зададения адрес, се проверява.

При изпълнение на команда WAIT програмата спира и не продължава, докато поне една двойка битове – от паметта и от <*числ.израз_2*>, определени с <*числ.израз_1* >, не се изравнят.

Ако във формата на оператор WAIT не е зададен <*числ.израз_2*>, неговата стойност се приема за нула.

Стойностите на двата числени израза трябва да са в границите от 0 до 255. В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Ако с оператор WAIT се зададе несъществуващ адрес от паметта или адрес на входно–изходно устройство, което не отговаря, компютърът блокира. От това положение може да се излезе само с едновременното натискане на клавиши CONTROL и RESET.

Оператор XDRAW. По зададена таблица изчертава фигура в графика с голяма разделителна способност. Ако оператор XDRAW се използва повторно със същите параметри, фигурата се изтрива.

Формат: XDRAW *числ.израз* [AT *гр.кол, гр.ред*]

От таблицата на фигурите (вж. приложение 7) се извлича фигурата, чийто номер е определен от числения израз, и се изчертава, като се започва от точката, зададена с координати <*гр.кол* >, <*гр.ред* >. Всяка точка на фигурата е с цвят, допълнителен на текущия цвят на екрана за тази точка (допълнителни са цветовете с кодове 0 и 3, 1 и 2, 4 и 7, 5 и 6). Размерите на фигурата и нейната ориентация се задават предварително с оператори SCALE и ROT.

Използването на оператор XDRAW дава възможност за лесно изтриване на начертаната фигура. Ако оператор XDRAW се изпълни четен брой пъти с едни и същи параметри, изображението на екрана не се изменя.

Ако началната точка на фигурата не се зададе с оператор XDRAW,

изчертаването ѝ започва от точката, използвана от последния изпълнен оператор DRAW, XDRAW или HPLOT.

Номерът на фигурата от таблицата трябва да бъде в границите от 0 до 255. Допустимите стойности за числените изрази, с които се задават редът и колоната, трябва да бъдат в съответните граници (0 – 191 за реда и 0 – 279 за колоната). В противен случай компютърът издава съобщение за грешка: ?ILLEGAL QUANTITY ERROR.

Команда &. Предава управлението на програмата на машинен език, започваща от адрес $3F5.

Формат: &

Обикновено на адрес $3F5 се записва инструкция за безусловен преход (JMP) към началото на програмата на машинен език.

Операционната система DOC3.3 записва на адрес $3F5 инструкция за безусловен преход към адрес $FF58 (JMP $FF58), където има записана инструкция RTS.

Глава 7. Схемна реализация

Персоналният компютър Правец–8А е компактно устройство, което включва изчислителна част (микропроцесор с постоянна и оперативна памет), клавиатура и схеми за нейното поддържане, схеми за генериране на видеосигнал, цифрови входове и изходи, аналогови входове, интерфейсни съединители за включване на допълнителни модули, високоговорител и захранващ блок. Тук е разгледана функционалната схема на компютъра и са дадени сведения, необходими на потребителя — как се използва паметта от външните входно–изходни устройства, как са изведени сигналите на изводите на съединителите, какво е значението на мостчетата от системната платка, каква е товароспособността на захранването и т.н.

7.1. Функционална схема

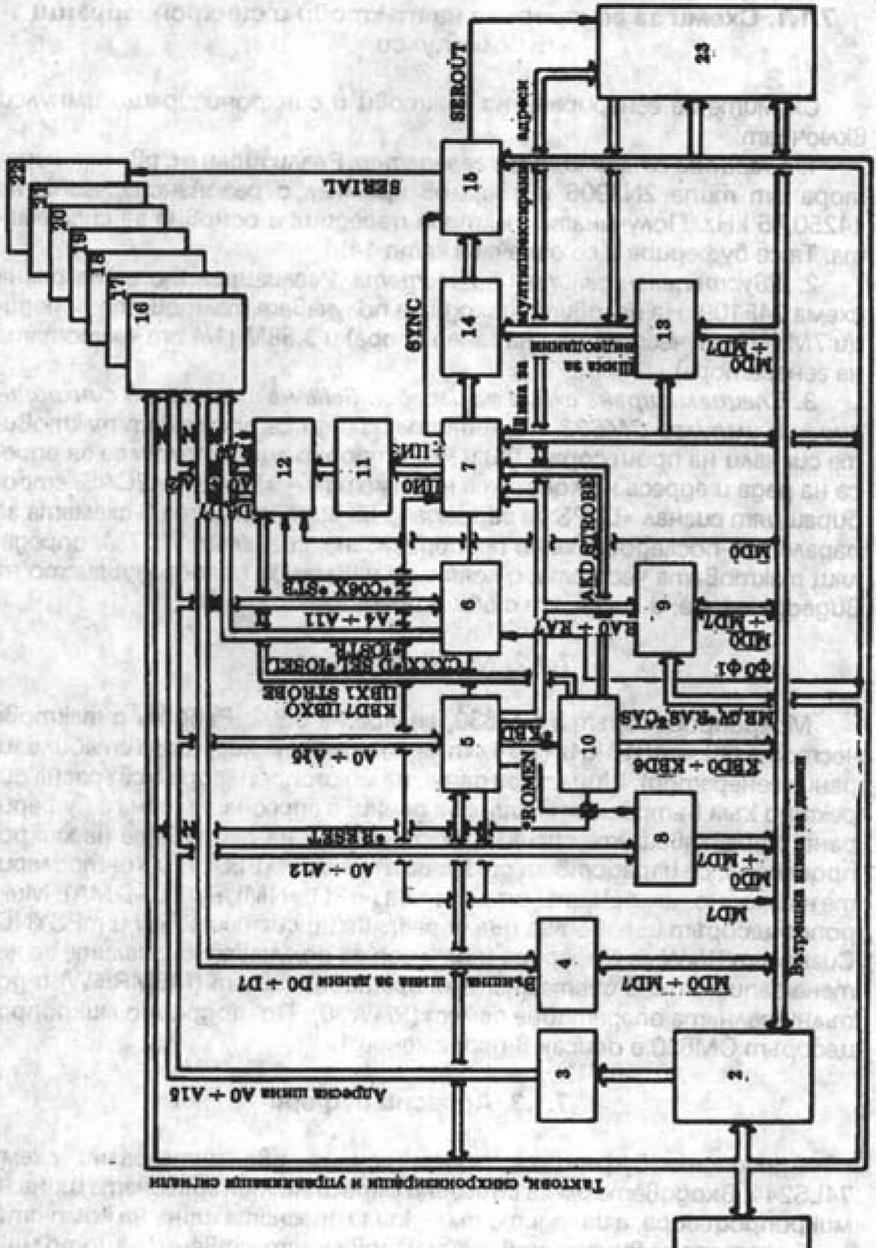
На фиг. 7.1 е дадена функционалната схема на системната платка на компютъра. Показани са основните функционални блокове, различните връзки между тях (управляващи и синхронизиращи сигнали, адреси, външна и вътрешна шина за данни, видеоданни и мултиплексирани адреси) и съединителите за входно–изходни устройства.

Звездичката пред имената на сигналите означава, че активното им състояние е логическа нула.

Сигналите от вида CXXX, C0XX и т.н. се получават при дешифриране на сигналите от адресната шина и се активират при обръщение към съответните адресни области.

Фиг. 7.1. Функционална схема на системната платка

1 – схема за генериране на тактови и синхронизиращи сигнали; **2** – микропроцесор; **3** – адресни буфери; **4** – буфер за данни; **5** – схема за управление на паметта; **6** – адресни декодери; **7** – схема за управление на вградените входно–изходни устройства; **8** – постоянна памет; **9** – оперативна памет; **10** – схеми за управление на клавиатурата; **11** – съединител за свързване на ръчки за управление на игри; **12** – цифрови и аналогови входове на компютъра; **13** – буфер за видеоданни; **14** – знаков генератор; **15** – преобразувател-смесител; **16 ÷22** – съединители за интерфейсни модули (X1 – X7); **23** – допълнителен съединител (X0)


Тактови, синхронизиращи и управляващи сигнали
Адресна шина A0 ÷ A15
Външна шина за данни D0 ÷ D7
MD0 ÷ MD7
MD7
A0 ÷ A12
*RESET
A0 ÷ A15
KBD0 ÷ KBD6
KBD7 ЦВХ0
ЦВХ1 STROBE
A4 ÷ A11
*C06X *STB
RA0 ÷ RA7
AKD STROBE
φ0 φ1
Вътрешна шина за данни
Шина за видеоданни
SYNC
SERIAL
SEROUT
адреси
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

7.1.1. Схеми за генериране на тактови и синхронизиращи импулси

Схемите за генериране на тактови и синхронизиращи импулси включват:

1. *Кварцово стабилизиран генератор.* Реализиран е с два транзистора от типа 2N3906 и кварцов кристал с резонансна честота 14250,45 kHz. Получената тактова поредица е основна за системата. Тя се буферира и се означава като 14M.

2. *Двустъпален делител на честота.* Реализиран е с интегрална схема 74F109. На неговите изходи се получават тактовите поредици 7M (1/2 от честотата на генератора) и 3.58M (1/4 от честотата на генератора).

3. *Специализирана схема за изработване на тактови и синхронизиращи импулси СМ633.* На нейните изходи се получават тактовите сигнали на процесора – Ф0 и Ф1, стробиращите импулси за адреса на реда и адреса на колоната на паметта – *PRAS и *PCAS, стробиращият сигнал *LDPS за зареждане на видеоданните в схемата за паралелно–последователно преобразуване, сигналът VID7M, определящ тактовата честота, с която се извършва преобразуването на видеоданните, и сигналът с общо предназначение Q3.

7.1.2. Микропроцесор

Микропроцесорът е СМ630, аналог на 6502. Работи с тактова честота Ф0 = 1018 kHz (1/14 от честотата на кварцово стабилизирания генератор). Шината за данни на микропроцесора е свързана директно към вътрешната шина за данни, а адресната шина е буферирана. Управляващите сигнали, постъпващи на входовете на микропроцесора, се изработват от клавиатурата (*RESET) и контролерите на входно–изходните устройства (*IRQ, *NMI, RDY и *DMA). Микропроцесорът изработва два управляващи сигнала R/*W и mPSYNC. Сигналът R/*W се буферира и от него се получават сигналите за четене/запис от и в системната оперативна памет (MEMR/*W) и допълнителната оперативна памет (R/*W80). По–подробно микропроцесорът СМ630 е описан в приложение 1.

7.1.3. Адресни буфери

Адресните буфери са реализирани с две интегрални схеми 74LS244. Входовете им са свързани директно към адресната шина на микропроцесора, а изходите им — към адресната шина на компютъра. Буферите са винаги отворени и това е естествено, защото микропроцесорът извършва адресирането на всички устройства в компютъра. Единственото изключение е режимът за пряк достъп до паметта — DMA. В този режим външното устройство, осъществява-

що пряк достъп до паметта на компютъра, издава сигнал *DMA, който блокира входния такт на микропроцесора, адресните буфери и сигнала R/*W (четене/запис).

7.1.4. Буфери за данни

Шината за данни на персоналния компютър Правец–8А е разделена на две.

Вътрешна шина за данни. Към тази шина са свързани микропроцесорът, оперативната и постоянната памет, схемата за управление на паметта, схемата за управление на вградените входно–изходни устройства, буферът на клавиатурата, буферът за видеоданни, програмноуправляемият ключ за избор на вътрешна кодова таблица и съединителят X0.

Външна шина за данни. Към нея са включени устройствата, ползващи съединителите X1 – X7, цифровите и аналоговите входове на компютъра и старшият бит (бит 7) на данните от клавиатурата.

Вътрешната и външната шина за данни са разделени от буфера за данни (интегрална схема 74LS245). При операция четене буферът е винаги отворен, а при операция запис е отворен само по време на Ф0. И в двата случая посоката на обмен се определя от сигнала MD IN/*OUT. Този сигнал се изработва от схемата за управление на паметта така, че буферът за данни да е отворен от външната към вътрешната шина за данни само при положение, че се извършва операция четене от адреси $C060 – $C06F и $C100 – $CFFF (програмноуправляеми ключове ПСЗРОМ и ПСХРОМ са включени).

Трябва да се отбележи, че когато сигналът *INH е активен, адресната област $D000 – $FFFF е част от областта, свързана със съединители X1 – X7, и микропроцесорът не може да адресира нито постоянната, нито оперативната памет от системната платка, намираща се на тези адреси.

7.1.5. Схема за управление на паметта

Управлението на паметта се осъществява от специализирана интегрална схема с голяма степен на интеграция (ГИС) от типа СМ631. Като използва управляващите и тактовите сигнали R/*W, *DMA, *INH, Ф0, Q3 и *PRAS, тя дешифрира адреса, издаден от микропроцесора, и в зависимост от състоянието на вградените в нея програмноуправляеми ключове изработва следните сигнали:

– MD IN/*OUT. Този сигнал определя дали устройствата, отговарящи на издадения адрес, са свързани към вътрешната или външната шина за данни. Управлява буфера за данни.

– *ROMEN1 и *ROMEN2. Тези сигнали се издават, когато се извършва операция четене от постоянната памет на компютъра. Сигнал *ROMEN2 се генерира при обръщение към областта $E000 – $FFFF

(сигнал *INH е неактивен), а сигнал *ROMEN1 – при обръщение към една от следните области:

$C100 – $C2FF	(програмноуправляем ключ ПСХРОМ е изключен);
$C300 – $C3FF	(програмноуправляемите ключове ПСХРОМ и ПСЗРОМ са изключени);
$C400 – $C7FF	(програмноуправляем ключ ПСХРОМ е изключен);
$C800 – $CFFF	(програмноуправляем ключ ПСЗРОМ е изключен, извършено е обръщение към областта $C300 – $C3FF и не е издаван адрес $CFFF);
$D000 – $DFFF	(сигнал *INH е неактивен).

– CXXX. Този сигнал се издава, когато има обръщение към външните входно–изходни устройства или към тези от вградените входно–изходни устройства, които са свързани към външната шина за данни. Използва се като разрешение за схемите за дешифриране на адреса, които изработват сигнали *I/O STROBE, *C0XX, *I/O SELn, *DEV.SELn, *C07X, *C06X и *STB.

– *KBD. Служи като строб за четене на клавиатурата.

– *CASEN. Разрешава достъпа до оперативната памет на системната платка.

– *EN80. Разрешава достъпа до допълнителната оперативна памет, свързана към съединителя X0. Отваря буфера за данни на модула, включен в този съединител, и разрешава постъпването на сигнала R/*W80.

– RA0 – RA7. На тези изходи се получават мултиплексираните адреси, необходими за адресиране на оперативната памет по време на процесорния цикъл Ф0. По време на цикъла Ф1 за опресняване на екрана тези изходи са в състояние на висок импеданс, а сигналите RA0 – RA7 се изработват от схемата за управление на вградените входно–изходни устройства.

Внимание! По шината за мултиплексирани адреси се подават адресите за достъп и за опресняване на оперативната памет. Достъпът се извършва по време на процесорния такт Ф0 от микропроцесора или от устройство, работещо в режим DMA. Опресняването на оперативната памет се извършва едновременно с опресняването на екрана по време на Ф1 от схемата за управление на входно–изходните устройства и по–точно от схемите за управление на видеоизображението, вградени в нея. По този начин в рамките на един процесорен цикъл на шината за мултиплексирани адреси постъпват 4 адреса:

– адрес на ред на оперативната памет – издава се от схемите за генериране на видеоизображение;

– адрес на колона на оперативната памет – издава се от схемите за генериране на видеоизображение;

– адрес на ред на оперативната памет – изработва се в схемата за управление на паметта от адреса, издаден от микропроцесора;

– адрес на колона на оперативната памет – изработва се в схемата за управление на паметта от адреса, издаден от микропроцесора.

В схемата за управление на паметта са вградени следните програмноуправляеми ключове: ПСХРОМ, ПСЗРОМ, ЗАП80, ЧСП, ЗСП, АНС, СТР2, ГР.ВРС, БАНКА2. Те се установяват, респ. нулират, с издаване на съответния адрес (вж. приложение 4), а състоянието им може да се прочете на линия MD7. Освен тях в схемата за управление на паметта се намират и ключовете за управление на паметта в областта $D000 – $FFFF (вж. табл. 3.1).

Внимание! Ключове ЗАП80, СТР2 и ГР.ВРС са дублирани в схемата за управление на вградените входно–изходни устройства.

7.1.6. Управление на вградените входно–изходни устройства

Схемите за управление на вградените входно–изходни устройства осигуряват четенето на символите, въвеждани от клавиатурата, и свързаното с това установяване на вътрешна кодова таблица, управлението на вградения високоговорител, издаването на сигнали на цифровите изходи на компютъра, възприемане на информацията, постъпваща на цифровите и аналоговите входове на компютъра, установяването на режима на изображение. Условно към тези схеми се включват и схемите за изработване на видеосигнала. Тези задачи се изпълняват от няколко функционално свързани групи схеми.

Схема за управление на вградените входно–изходни устройства. Това е специализирана ГИС от типа СМ632 с многостранно предназначение. Тя използва управляващите и тактовите сигнали – R/*W, *COXX, Ф0, Q3 и *PRAS, и като дешифрира издадения от микропроцесора адрес, задейства някой от вградените в нея програмноуправляеми ключове ЗАП80, 80КОЛ, АСМ, ТЕКСТ, СМ, СТР2, ГР.ВРС, ЦИЗХ0, ЦИЗХ1, ЦИЗХ2, ЦИЗХ3, ВГ. Тези ключове управляват режимите на видеоизображение, избора на символно множество, изхода към високоговорителя и цифровите изходи на компютъра. Състоянието на някои от тях може да се прочете по линия MD7 (вж. приложение 4).

Схемата за управление на вградените входно–изходни устройства обработва сигналите от клавиатурата STROBE и AKD и в зависимост от тяхното състояние управлява автоматичното повторение на натиснатия клавиш на клавиатурата (установява програмноуправляем ключ СТРОБ и флаг НАТИСНАТ КЛАВИШ).

В тази схема са вградени и схемите за генериране на синхросместа и схемите за генериране на адресите, необходими за опресняване на екрана, респ. на оперативната памет на компютъра.

На изходите на схемата за управление на вградените входно–изходни устройства се получават следните сигнали.

– RA0 – RA7. Това са мултиплексираните адреси, необходими за адресиране на оперативната памет по време на цикъла за нейното опресняване, респ. опресняването на екрана (Ф1). По време на процесорния цикъл схемата за управление на вградените входно–изходни устройства приема адресите A0 – A5 и A7, издадени от микропроцесора, по шина RA0 – RA7.

– *RA9 и *RA10. Тези два сигнала се използват при определяне на формата на текстовото изображение (НОРМАЛНО или ИНВЕРСНО ВИДЕО). В режим графика те повтарят двата старши бита на видеоданните VID6 и VID7.

– *CLRG. Това е сигнал, който разрешава преминаването на сигнала за цветова синхронизация по система NTSC към видеосмесителя.

– *SYNC. Това е синхросместа, необходима за изработване на комплексния видеосигнал.

– SEGA, SEGB и VC. В текстов режим тези сигнали образуват триразреден брояч, състоянието на който определя развивката на матрицата на символа. В режим ГР40 сигнал VC определя кое от двете цветни блокчета, дефинирани от байта видеоданни, се изобразява на екрана.

– SPCR. Управлява вградения високоговорител.

– GR. Показва, че компютърът е в режим графика.

– *WNDW. Този сигнал забранява постъпването на преобразуваните видеоданни във видеосмесителя. Активен е по време на обратния ход на лъча по редове и кадри.

– *80VID. Разрешава режимите с двойна разделителна способност – ТЕКСТ80, ГР80 и ГР560.

– H0. Това е младшият бит на брояча, определящ хоризонталната позиция на символа върху екрана.

– ЦИЗХ0, ЦИЗХ1, ЦИЗХ2 и ЦИЗХ3. Това са сигналите, получавани на цифровите изходи на компютъра.

Цифрови входове. Персоналният компютър Правец–8А има три цифрови входа, означени с ЦВХ0, ЦВХ1 и ЦВХ2. Към първите два от тях са свързани функционалните клавиши F1 и F2. Състоянието на цифровите входове се възприема от процесора по линия D7 след преминаване през мултиплексора (интегрална схема 74LS251), който е достъпен на адреси C06X.

Аналогови (потенциометрични) входове. Правец–8А, подобно на предшествениците си има четири потенциометрични входа. Тяхното състояние се възприема от процесора по линия D7 след преминаване през споменатия мултиплексор. Нулирането на аналоговите входове се извършва от сигнал *PDL RESET (адресна област $C070 – $C07F).

Схеми за управление на клавиатурата. Тези схеми включват:

Буфер, реализиран с интегрална схема 74LS374. В този буфер по сигнал STROBE (изработван от клавиатурата) и при активен сигнал *ENKBD се записват младшите седем бита от кода на натиснатия клавиш. Процесорът чете съдържанието на буфера на адрес $C000 (по този адрес схемата за управление на паметта изработва сигнал *KBD).

Буфер, реализиран с един D–тригер от интегрална схема 74LS74. В него по сигнал STROBE се записва осмият бит на кода, генериран от клавиатурата. Този бит се чете от процесора по линия D7 на адрес $C060.

Програмноуправляем ключ, определящ текущата кодова таблица (седем– или осембитова). Реализиран е с втория D–тригер от интегралната схема 74LS74. Неговото състояние се установява с операция запис на адрес $C06X и се определя от младшия бит на байта данни. Процесорът може да чете състоянието на този ключ по линия D6 на същия адрес.

Схеми за изработване на видеосигнала и за управление на режимите на изображение. В тази група освен схемите за генериране на основните тактови сигнали на компютъра (14M, VID7M, *LDPS и Ф0) и схемата за управление на вградените входно–изходни устройства се включват:

Буфер за видеоданни. Реализиран е с интегрална схема 74LS374. Разделя шината за видеоданни от вътрешната шина за данни. В него по предния фронт на сигнала Ф0 се записва информацията, намираща се на вътрешната шина за данни. По задния фронт на сигнала Ф0 съдържанието на този буфер постъпва на шината за видеоданни, а оттам и на входа на знаковия генератор. Шината за видеоданни е шина с три състояния. Тя е изведена и на съединител X0, откъдето на входовете на знаковия генератор постъпват данните от текстовата, респ. графичната страница, намираща се в допълнителната оперативна памет.

Знаков генератор. Предназначението му е в зависимост от режима на изображение (определен от управляващите сигнали GR, SEGA, SEGB и VC) да преобразува видеоданните в текстово или графично изображение. Съдържа два независими един от друг набора от символи, които се превключват с мостчето X20.

Схема за преобразуване на паралелните данни от знаковия генератор (интегрална схема 74LS166). Преобразува байта, получен на изхода на знаковия генератор в резултат от преобразуването на видеоданните, от паралелен в последователен код. Байтът, получен на изхода на знаковия генератор, се зарежда в преобразувателя по сигнал *LDPS, а скоростта на преобразуване се определя от тактовия сигнал VID7M, който повтаря една от двете поредици – 7M или 14M.

Видеосмесител. Реализиран е с два транзистора от типа 2T3851 и 2T3608. Служи за смесване на данните, преобразувани в последова-

телен код (SERIAL), със синхросместа (*SYNC) и за буфериране на видеоизхода.

Схема за управление на вградения високоговорител. Тя представлява нискочестотен усилвател, реализиран с двойката транзистори 2Т3608 и 2Т6551 и предназначен за усилване на сигнал SPCR, изработен в схемата за управление на вградените входно–изходни устройства.

7.1.7. Оперативна памет

Частта от оперативната памет на компютъра, разположена на системната платка, има обем 64 Кбайта и е реализирана с 8 интегрални схеми от типа 4164, свързани към вътрешната шина за данни. Това е динамична памет, която изисква един пълен цикъл на опресняване на всеки 2 ms. За да не се намалява бързодействието на компютъра, опресняването на паметта е съчетано с опресняването на екрана и се извършва от схемата за управление на вградените входно–изходни устройства по време на цикъла Ф1, т.е. независимо от работата на процесора. Процесорът адресира паметта по време на Ф0, като мултиплексирането на адреса се извършва в схемата за управление на паметта. Адресните стробове за реда – *RAS, и за колоната – *CAS, всъщност са терминираните сигнали *PRAS и *PCAS, изработени от схемите за генериране на тактови и синхронизиращи импулси. Сигналът за четене/запис MEMR/*W се получава чрез буфериране на общия за компютъра сигнал R/*W.

7.1.8. Постоянна памет

Изградена е с две схеми от типа 2764, има обем 16 Кбайта и е свързана към вътрешната шина за данни. Съдържа резидентното програмно осигуряване на компютъра. Предвидена е възможност резидентното програмно осигуряване да се дублира с програмно осигуряване, създадено от потребителя. Това може да се осъществи, като интегралните схеми от типа 2764 се заменят със схеми от типа 27128, имащи двойно по–голям обем. В такъв случай работното програмно осигуряване се избира с мостчето X19.

7.2. Използване на адресното пространство от входно–изходните устройства

Входно–изходните устройства използват адресното пространство $C000 – $CFFF. Вградените (клавиатура, видеомонитор, цифрови входове и изходи, аналогови входове, високоговорител и програмноуправляеми ключове) са разположени на адреси $C000 – $C08F, а външните ползват адресното пространство $C090 – $CFFF. Де-

шифрирането на адресите, използвани от входно–изходните устройства, се извършва от:

1. Схемата за управление на паметта. Разпознава обръщението към входно–изходно устройство в областта $C000 – $CFFF (сигнал CXXX), към клавиатурата на адрес $C000 (сигнал *KBD) и към вградените в нея програмноуправляеми ключове.

2. Схемата за управление на вградените входно–изходни устройства. Разпознава обръщението към вградените в нея програмноуправляеми ключове. Директно управлява високоговорителя на адрес $C030 (сигнал SPCR), цифровите изходи на компютъра на адреси $C05A – $C05F (сигнали ЦИЗХ0 – ЦИЗХ3) и бит СТРОБ на клавиатурата на адрес $C010.

3. Схемите за адресна дешифрация (74LS138, 74LS154 и част от 74S10). Изработват следните сигнали:

– при обръщение към адресната област $C000 – $C0FF – сигнал *C0XX (определя областта на вградените входно–изходни устройства);

– при обръщение към адресната област $C040 – $C04F – сигнал *STB (строб за цифровите изходи на компютъра);

– при обръщение към адресната област $C060 – $C06F – сигнал *MUX.SEL (разрешение за мултиплексора, през който се отчита състоянието на цифровите и аналоговите входове на компютъра и на бит 7 от данните на клавиатурата);

– при обръщение към адресната област $C070 – $C07F – сигнал *PDL.RESET (нулиране на аналоговите входове на компютъра);

– при обръщение към адресната област $C080 – $CFFF – някой от сигналите *DEV.SELn, *I/O SELn или *I/O STROBE (вж. по–долу).

Адресното пространство, определено за външните входно–изходни устройства, е разделено на три области.

Първата област е индивидуална за всеки интерфейсен модул и е с обем 16 байта. Тя се намира на адреси $C0m0 – $C0mF и се разрешава със сигнали *DEV.SELn, където **n** = 1 – 7 е номерът на съответния съединител, а **m** = **n** + 8. В тази област се намират адресите на входно–изходните и управляващите регистри на интерфейсните модули.

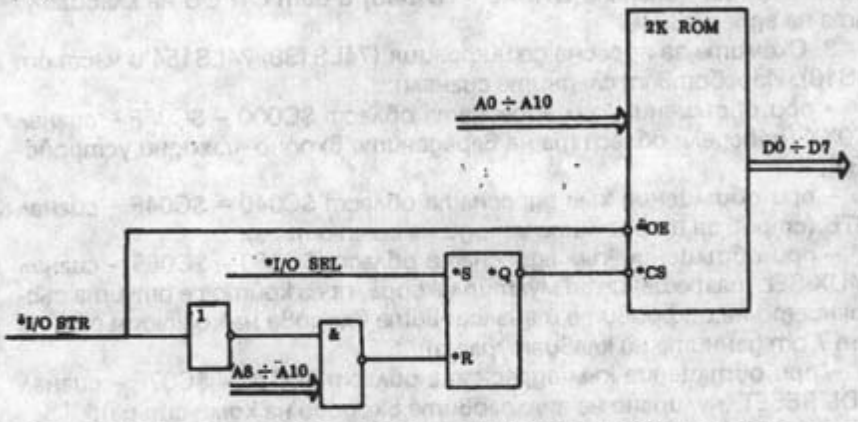
Втората област също е индивидуална за всеки интерфейсен модул, но има по–голям обем – 256 байта. Тя се намира на адреси $Cn00 – $CnFF и се разрешава със сигнали *I/O SELn, където **n** = 1 – 7 е номерът на съответния съединител. Обикновено в това адресно пространство е разположена постоянна памет, в която е записана драйверната програма на интерфейсния модул, но е възможно и съвместното му използване от драйверната програма и от входно–изходните регистри на модула.

Третата област е обща за всички интерфейсни модули. Тя се състои от 2 Кбайта, намира се на адреси $C800 – $CFFF и се разрешава

със сигнал *I/O STROBE. Обикновено се използва от драйверната програма на интерфейсния модул. Съвместното използване на едно и също адресно пространство от интерфейсните модули създава условия за конфликти, особено ако се работи с прекъсвания. Това налага спазването на определени конвенции:

1. Достъпът до драйверната програма на интерфейсния модул, разположена в адресната област $C800 – $CFFF, се разрешава със сигнал *I/O SEL (вж. фиг. 7.2).

2. При обръщение към адрес $CFFF всички интерфейсни модули трябва да забраняват достъпа до паметта от тази адресна област. Допуска се и непълна дешифрация на този адрес (вж. фиг. 7.2).



Фиг. 7.2. Примерна схема за достъп до областта $C800 – $CFFF

3. На адрес $7F8 от системната оперативна памет се записва байт данни от вида $Cn (n = 1 – 7 е номерът на съединителя). Във всеки момент от време този байт показва кой от интерфейсните модули използва адресната област $C800 – $CFFF.

Тези три конвенции за използване на общото адресно пространство от интерфейсните модули са наложени от апаратни съображения. Към тях се добавят и конвенциите, наложени от операционните системи: за използване на незаетите клетки от текстовата страница, за начално зареждане, за байтовете за идентификация на типа на модула и т.н.

7.3. Мостчета за конфигуриране

Във вида, в който компютърът се доставя от завода производител, той не се нуждае от конфигуриране. Съществуващите мостчета са предназначени за промяна на някои от характеристиките на компютъра, и то от потребители с висока квалификация.

Мостче X18. Неговото включване установява логическа единица на цифров вход 2 (ЦВХ2).

Мостче X16. Неговото включване забранява сигнал *DMA.

Мостче X19. Дава възможност на потребителя да добави и използва алтернативно резидентното програмно осигуряване. За целта е необходимо постоянната памет да се замени с нова, имаща два пъти по-голям обем (две интегрални схеми от вида 27128) и в двете ѝ половини да се запишат основното и алтернативното програмно осигуряване.

Мостче X20. Служи за избор на един от двата знакови генератора, вградени в компютъра.

7.4. Съединители на системната платка на персоналния компютър Правец–8А

7.4.1. Допълнителен съединител X0

Съединителят X0 е предназначен за разширяване на паметта и на формата на изображението. Разпределението на сигналите по неговите изводи е показано в табл. 7.1.

Всички изходи, изведени на съединителя X0, имат товароспособност 2 TTL LS товара.

Таблица 7.1

Съединител X0

Извод	Сигнал	Предназначение
1	2	3
1	3.58M	Синхросигнал с честота 3.58 MHz. Използва се от схемите за генериране на видеоизображение
2	VID7M	Тактов сигнал, определящ скоростта на преобразуване на паралелните данни от изхода на знаковия генератор в паралелен код, респ. разделителната способност на изображението по редове
3	*SYNC	Синхросмес за синхронизация по редове и кадри
4	*PRAS	Строб за адреса на реда на оперативната памет
5	VC	Старши бит (бит 2) на триразредния брояч, определящ вертикалната развивка на матрицата на символа на екрана
6	*PDL.RESET	Сигнал за нулиране на аналоговите входове на компютъра

1	2	3
7	*WNDW	Определя времето от развивката, през което се изпраща видеоинформация към екрана
8	SEGA	Младши бит (бит 0) на триразредния брояч, определящ вертикалната развивка на матрицата на символа на екрана
9	RA7	Старши бит (бит 7) на шината за мултиплексирани адреси
10	RA1	Бит 1 на шината за мултиплексирани адреси
11	*ROMEN1	Разрешение за достъп до резидентната постоянна памет в областта $D000 – $CFFF
12	*ROMEN2	Разрешение за достъп до резидентната постоянна памет в областта $E000 – $FFFF
13	RA4	Бит 4 на шината за мултиплексирани адреси
14	RA5	Бит 5 на шината за мултиплексирани адреси
15	VID7	Старши бит (бит 7) на шината за видеоданни
16	MD7	Старши бит (бит 7) на вътрешната шина за данни
17	MD6	Бит 6 на вътрешната шина за данни
18	VID6	Бит 6 на шината за видеоданни
19	VID5	Бит 5 на шината за видеоданни
20	MD5	Бит 5 на вътрешната шина за данни
21	MD4	Бит 4 на вътрешната шина за данни
22	VID4	Бит 4 на шината за видеоданни
23	Ф0	Фаза 0 на процесорния такт
24	*CLRGAT	Сигнал за синхронизация по цветност в система NTSC
25	*80VID	Разрешение за режим на 80-колонно изображение
26	*EN80	Разрешение за достъп до допълнителната памет
27	*ALTVID	Видеосигнал от алтернативен източник
28	*SEROUT	Изход на схемата за преобразуване на паралелните видеоданни в последователен код
29	*ENVID	Установяването на този сигнал в логическа единица блокира видеоинформацията и екранът става бял. Това позволява подаването на видеосигнал от алтернативен източник по шина *ALTVID
30	5V	Шина +5V
31	GND	Шина ЗЕМЯ
32	14M	Основен тактов сигнал с честота 14250,45 kHz
33	*PCAS	Строб за адреса на колоната на оперативната памет

Продължение на таблица 7.1

1	2	3
34	*LDPS	Строб на паралелно–последователния преобразувател. По отрицателния фронт на този сигнал данните, получени на изхода на знаковия генератор, се зареждат в преместващия регистър
35	R/*W80	Сигнал четене/запис за допълнителната памет
36	Ф1	Фаза 1 на процесорния такт
37	*CASEN	Разрешение за строба на адреса на колоната на оперативната памет от системната платка. Този сигнал е логическа единица по време на достъпа до допълнителната памет, включена в съединител X0
38	VID3	Бит 3 на шината за видеоданни
39	MD3	Бит 3 на вътрешната шина за данни
40	MD2	Бит 2 на вътрешната шина за данни
41	VID2	Бит 2 на шината за видеоданни
42	VID1	Бит 1 на шината за видеоданни
43	MD1	Бит 1 на вътрешната шина за данни
44	MD0	Младши бит (бит 0) на вътрешната шина за данни
45	VID0	Младши бит (бит 0) на шината за видеоданни
46	RA6	Бит 6 на шината за мултиплексирани адреси
47	H0	Младши бит (бит 0) на брояча, определящ хоризонталната позиция на символа върху екрана
48	RA3	Бит 3 на шината за мултиплексирани адреси
49	RA2	Бит 2 на шината за мултиплексирани адреси
50	ЦИЗХ3	Цифров изход 3
51	RA0	Младши бит (бит 0) на шината за мултиплексирани адреси
52	R/*W	Сигнал четене/запис
53	Q3	Тактов сигнал с честота 2 MHz и общо предназначение
54	SEGB	Бит 1 на триразредния брояч, определящ вертикалната развивка на матрицата на символа на екрана
55	*FRCTXT	Когато този сигнал е логическа нула, честотата на паралелно–последователното преобразуване е 14 MHz дори ако сигналът GR е активен
56	*RA9	Сигнал за управление на знаковия генератор
57	*RA10	Сигнал за управление на знаковия генератор
58	GR	Разрешение за работа в графичен режим
59	7M	Тактов сигнал с честота 7 MHz
60	*ENTMG	Установяването на състояние логическа единица по тази линия блокира тактовите сигнали на системата

7.4.2. Съединители за включване на допълнителни модули X1 – X7

С малки изключения (сигнали *I/O SELn и *DEV.SELn) тези съединители са свързани в паралел. Малко по-особен е съединителят X7, тъй като на него са изведени сигналите, необходими за получаване на цветно изображение (3.58M, 14M, *SERIAL, GR и SYNC). Разпределението на сигналите от шината на компютъра по изводите на тези съединители е показано в табл. 7.2. С изключение на специално указаните, изходите, изведени на съединители X1 – X7, имат товароспособност 2 TTL LS товара.

Таблица 7.2

Съединители X1 – X7

Извод	Сигнал	Предназначение
1	2	3
1	*I/O SEL	Нормално този сигнал е логическа единица. Става логическа нула, когато процесорът издаде адрес от областта $CnXX, където n е номерът на съответния съединител. Тази линия има товароспособност 10 TTL LS товара
2–17	A0 – A15	Адресна шина. Това е шина с три състояния и товароспособност 5 TTL LS товара. Адресът става валиден по време на Ф1 и остава такъв и по време на Ф2
18	R/*W	Сигнал четене/запис. Линия с три състояния, сигналът по която е валиден едновременно с адресите
19	*SYNC	Синхросмес. Постъпва само на съединител X7
20	*I/O STR	Нормално този сигнал е логическа единица. Става логическа нула, когато процесорът издаде адрес от областта $C800 – $CFFF. Тази линия има товароспособност 4 TTL LS товара
21	RDY	Вход на микропроцесора. Подаването на ниско ниво на тази линия по време на Ф1 спира работата на процесора, а на адресната шина се запазва издаденият адрес
22	*DMA	Сигнал за директен достъп до паметта. Издаването му от интерфейсния модул по време на Ф1 блокира достъпа на процесора до адресната шина

1	2	3
		Внимание! На този извод на съединителя X7 се подава видеоинформацията в последователен код – сигнал SERIAL
23	INT OUT	Изход на веригата за определяне на приоритета на прекъсванията. Обикновено се свързва към извод 28 (INT IN)
		Внимание! На този извод на съединителя X7 се подава сигнал GR
24	DMA OUT	Изход на веригата за определяне на приоритета за директен достъп до паметта. Обикновено се свързва към извод 27 (DMA IN)
		Внимание! На този извод на съединителя X7 се подава сигнал 14M
25	5V	Шина + 5V
26	GND	Шина ЗЕМЯ
27	DMA IN	Вход на веригата за определяне на приоритета на директния достъп до паметта. Обикновено се свързва към извод 24 (DMA OUT)
28	INT IN	Вход на веригата за определяне на приоритета на прекъсванията. Обикновено се свързва към извод 23 (INT OUT)
29	*NMI	Вход за немаскируемо прекъсване на процесора. Подаването на логическа нула по тази линия стартира цикъл за обработка на прекъсване
30	*INT	Вход за маскируемо прекъсване на процесора. Подаването на логическа нула по тази линия стартира цикъл за обработка на прекъсване само ако флагът за забрана на прекъсванията (I) не е установен
31	*RESET	Общо нулиране на системата
32	*INH	Подаването на ниско ниво на тази линия по време на Ф1 забранява достъпа до паметта на системната платка
33	–12V	Шина –12V
34	–5V	Шина –5V
35	3.58M	Сигнал за синхронизация по цветност в система NTSC. Свързан е само към съединител X7
36	7M	Системен тактов сигнал с честота 7 MHz
37	Q3	Тактов сигнал с честота 2 MHz и общо предназначение
38	Ф1	Фаза 1 на процесорния такт

1	2	3
39	mPSYNC	Микропроцесорът установява този сигнал в състояние логическа единица по време на първия цикъл от всяко четене (извличане) на инструкция от паметта на компютъра
40	Ф0	Фаза 0 на процесорния такт
41	*DEV.SEL	Нормално този сигнал е логическа единица. Става логическа нула по време на Ф0, когато процесорът издаде адрес от областта $C0nX, където n е номерът на съответния съединител плюс 8. Тази линия има товароспособност 10 TTL LS товара
42–48	D7–D0	Външна шина за данни *Внимание!* Това е двупосочна шина с три състояния. Информацията на нея става валидна по време на Ф0 и остава валидна до падането на Ф0. Тази шина има товароспособност 1 TTL LS товар
50	12V	Шина +12V

7.4.3. Свързване на видеомонитор

По принцип видеомониторът се свързва към съединителя тип BNC, монтиран на задната плоча на компютъра, но е предвидена и друга възможност. В задния десен ъгъл на системната платка, в близост до съединителя X7, се намират щифтовите съединители X8 – X10 (вж. фиг.1.2). Те се използват като допълнителни видеоизходи. Разпределението на сигналите по техните изводи е показано в табл. 7.3.

Таблица 7.3

Съединители X8 – X10

Извод	Сигнал	Значение
1	2	3
X8 – 1	VIDEO OUT	Буфериран видеосигнал. Еквивалентен е на сигнала, получаван на съединителя тип BNC
X8 – 2	GND	Шина ЗЕМЯ
X9 – 1	VIDEO OUT	Небуфериран видеосигнал. Свързан е паралелно към извод 2 на съединителя X10
X10 – 1	GND	Шина ЗЕМЯ

Продължение на таблица 7.3.

1	2	3
X10 – 2	VIDEO OUT	Небуфериран видеосигнал. Използва се при свързване на модулатор, на модул RGB и т.н
X10 – 3	– 5V	Шина –5V
X10 – 4	12V	Шина + 12V

7.4.4. Съединител за свързване на ръчки за управление на игри

Ръчките за управление на игри се включват към компютъра в гнездовия съединител X11. Разпределението на сигналите на неговите изводи е показано в табл. 7.4.

Таблица 7.4

Съединител X11

Извод	Сигнал	Предназначение
1	5V	Шина + 5V
2	ЦВХ0	Цифров вход 0
3	ЦВХ1	Цифров вход 1
4	ЦВХ2	Цифров вход 2
5	*PDL RESET	Сигнал, нулиращ аналоговите входове на компютъра
6	АВХ0	Аналогов вход 0
7	АВХ2	Аналогов вход 2
8	GND	Шина ЗЕМЯ
9	–	Не се използва
10	АВХ1	Аналогов вход 1
11	АВХ3	Аналогов вход 3
12	ЦИЗХ3	Цифров изход 3
13	ЦИЗХ2	Цифров изход 2
14	ЦИЗХ1	Цифров изход 1
15	ЦИЗХ0	Цифров изход 0
16	–	Не се използва

7.4.5. Свързване на високоговорителя

Високоговорителят се свързва към съединителя X12, разположен в десния преден край на системната платка (вж. фиг. 1.2).

7.4.6. Свързване на клавиатурата

Клавиатурата се свързва към компютъра с лентов кабел. На клавиатурата и на системната платка са монтирани цокли тип DIL 16. Изводите на съединителя X13, който е монтиран на системната платка и към който се свързва клавиатурата, и съответстващите им сигнали са дадени в табл. 7.5.

Таблица 7.5

Съединител за свързване на клавиатурата

Извод	Сигнал	Предназначение
1	5V	Шина +5V
2	STROBE	По предния фронт на импулса, получен по тази линия, се записват валидни данни от клавиатурата в клавиатурния буфер
3	*RESET	Сигнал за общо нулиране. Изработва се в клавиатурата
4	AKD	Този сигнал е активен при натиснат клавиш
5	KBD 5	Бит 5 на кода, изпращан от клавиатурата
6	KBD 4	Бит 4 на кода, изпращан от клавиатурата
7	KBD 6	Бит 6 на кода, изпращан от клавиатурата
8	GND	Шина ЗЕМЯ
9	ЦВХ0	Функционален клавиш F1
10	KBD 2	Бит 2 на кода, изпращан от клавиатурата
11	KBD 3	Бит 3 на кода, изпращан от клавиатурата
12	KBD 0	Младши бит (бит 0) на кода, изпращан от клавиатурата
13	KBD 1	Бит 1 на кода, изпращан от клавиатурата
14	SFTSW	Сигнал, изработван в системната платка. Определя вида на вътрешната кодова таблица, използвана от компютъра, и конфигурира клавиатурата за работа с осембитова кодова таблица
15	KBD 7	Старши бит (бит 7) на кода, изпращан от клавиатурата
16	ЦВХ1	Функционален клавиш F2

7.4.7. Свързване на захранващия блок

Използва се съединител тип "Жигули". Изводите на съединителя, съответните напрежения и товароспособността на отделните източници са дадени в табл. 7.6.

Внимание! Броят на модулите, които могат да се включват към осемразредните персонални компютри от фамилията Правец, расте непрекъснато. Основното изискване, което трябва да се спазва при разширяване на системата, е общата консумирана мощност на всички модули да не надвишава допустимите натоварвания на захранващите източници. В табл. 7.7 е дадена консумацията на някои от по–разпространените модули. Консумацията на модул FDC е измерена при работа с едно дисково устройство. Трябва да се има предвид, че консумацията на дискови устройства, произведени от различни производители, е различна.

Таблица 7.6

Свързване на захранващия блок

Извод	Сигнал	Забележка
1	5V / 4,0A	Броенето на изводите започва от шлица в щифтовия съединител, монтиран на системната платка
2	GND	
3	–12V / 0,25A	
4	–5V / 0,25A	
5	12V / 1,5A	

Таблица 7.7

Консумация на модулите на компютъра

Модул	5V	12V	–5V	–12V
1	2	3	4	5
Системна платка	0,96A	0,01A	0,04A	–
Модул RGB][	0,20A	–	–	–
Pug–контактна клавиатура	0,05A	–	–	–

Продължение на таблица 7.7

1	2	3	4	5
Модул 128/512K RAM				
с 64 Кбайта	0,32A	–	–	–
със 128 Кбайта	0,42A	–	–	–
с 256 Кбайта	0,37A	–	–	–
с 320 Кбайта	0,47A	–	–	–
с 512 Кбайта	0,50A	–	–	–
Модул RAM PLUS				
с 64 Кбайта	0,05A	–	–	–
със 128 Кбайта	0,10A	–	–	–
с 256 Кбайта	0,07A	–	–	–
с 320 Кбайта	0,12A	–	–	–
с 512 Кбайта	0,13A	–	–	–
Модул FDC	0,45A	0,70A	–	–
Модул PPI	0,20A	–	–	–
Модул CP/M	0,30A	–	–	–
Модул RS232C	0,19A	0,04A		0,01A

Приложение 1. Микропроцесор СМ630

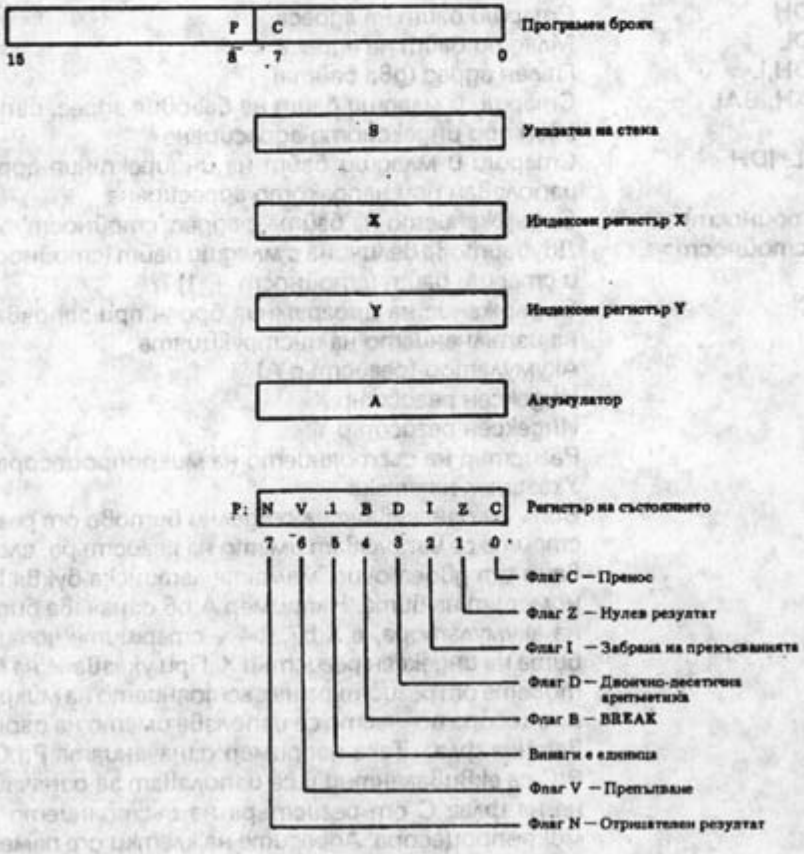
СМ630 е осемразреден микропроцесор, функционален аналог на популярния микропроцесор 6502 на фирмата Rockwell. Тук са разгледани програмният модел на микропроцесора, видовете адресиране и наборът от инструкции. Възприети са следните означения:

КОП	Код на операцията
ОР	Операнд
ЕА	Ефективен адрес. Това е адресът, на който се намира операндът ОР. При работа с регистрите на микропроцесора операндът може и да няма ефективен адрес
ADH	Старши байт на адреса
ADL	Младши байт на адреса
ADH,L	Пълен адрес (два байта)
BAH, BAL	Старши и младши байт на базовия адрес, използван при индексното адресиране
IDL, IDH	Старши и младши байт на индиректния адрес, използван при непрякото адресиране
(стойност)	Съдържанието на байта с адрес 'стойност'
<стойност>	Двубайтова величина с младши байт (стойност) и старши байт (стойност + 1)
PC	Съдържание на програмния брояч при започване на изпълнението на инструкцията
A	Акумулатор (регистър А)
X	Индексен регистър X
Y	Индексен регистър Y
P	Регистър на състоянието на микропроцесора
S	Указател на стека
b	Бит. За означаване на отделни битове от регистрите се използват името на регистъра, следвано от двоеточие, малката латинска буква b и номерът на бита. Например A:b6 означава бит 6 на акумулатора, а X:b7...b4 – старшите четири бита на индексен регистър X. При указване на битовете от регистъра на състоянието на микропроцесора по-често се използва името на съответния флаг. Така например означенията P:b0 и P:C са еквивалентни и се използват за означаване на флаг С от регистъра на състоянието на микропроцесора. Адресите на клетки от паметта се означават в шестнайсетичен вид. Например $C00:b6...b0

—> u <—	Преместване към ...
AND	Логическа операция И
OR	Логическа операция ИЛИ
XOR	Логическа операция ИЗКЛЮЧВАЩО ИЛИ

П1.1. Програмен модел

Програмният модел на микропроцесора СМ630 е показан на фиг. П1.1. Процесорът разполага с 16-битов програмен брояч (PC), 8-битов указател на стека (регистър S), два 8-битови индексни регистъра (X и Y), един 8-битов акумулатор (регистър A) и един 8-битов регистър на състоянието (регистър P). Той може да адресира дирек-



Фиг. П1.1. Програмен модел на микропроцесор СМ630

тно 64 Кбайта памет, разделена условно на 256 страници по 256 байта. Нулевата страница (адреси $0000 – $00FF) може да се адресира чрез специални адресни режими. В първа страница е разположен стекът на микропроцесора, като указателят на стека съдържа само младшата част на адреса на стека.

П1.2. Методи на адресиране

Акумулаторно адресиране – ACC. Инструкциите са еднобайтови и са предназначени за модифициране на съдържанието на акумулатора. Изпълняват се за два микропроцесорни цикъла (вж. табл. П1.1).

Акумулаторното адресиране се използва с инструкции LSR, ASL, ROL и ROR.

Таблица П1.1

Акумулаторно адресиране

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната инструкция завършва. Програмният брояч се увеличава с единица
2	PC + 1	КОП 1	Извлича се следващият КОП. Текущата инструкция се декодира. Програмният брояч не се променя
3	PC + 1	КОП 1	Повторно се извлича следващият КОП. Текущата инструкция се изпълнява. Програмният брояч се увеличава с единица
4	PC + 2		Извлича се втори байт. Резултатът от изпълнението на текущата инструкция се записва в акумулатора. Интерпретира се новият КОП

Неявно адресиране – IMP. Инструкциите са еднобайтови и са предназначени за работа с вътрешните регистри на микропроцесора. Операндът не е даден в явен вид, а се подразбира. Затова този метод на адресиране често се нарича адресиране с подразбиране на регистър. С тези инструкции се нулират, респ. установяват, флаговете на регистъра на състоянието (CLC, CLD, CLI, CLV, SEC, SED, SEI), увеличава се или се намалява съдържанието на индексните регистри с единица (INX, INY, DEX, DEY), прехвърля се съдържанието на един ре-

гистър в друг (TAX, TAY, TSX, TXA, TXS, TYA) и т.н. С изключение на инструкциите, свързани с промяна на съдържанието на стека (BRK, PHA, PHP, PLA, PLP, RTI, RTS), всички инструкции за неявно адресиране се изпълняват в два микропроцесорни цикъла (вж. табл. П1.2).

През първия цикъл на микропроцесора се извлича кодът на операцията и съдържанието на програмния брояч се увеличава с единица.

През втория микропроцесорен цикъл се извлича следващият байт от паметта на компютъра. Тъй като изпълняваната инструкция е еднобайтова, това е безсмислена операция. Този байт не се използва, а инкрементирането на програмния брояч се потиска.

През третия цикъл изпълнението на текущата инструкция завършва, а микропроцесорът повторно издава следващия адрес. Едва сега следващият байт се интерпретира като код на нова инструкция.

Инструкциите за неявно адресиране имат следните особености:

1. Инструкциите PHA и PHP се изпълняват за три микропроцесорни цикъла. След като през втория микропроцесорен цикъл кодът на операцията се декодира, през третия цикъл съдържанието на акумулатора, респ. на регистъра на състоянието, се записва в стека и указателят на стека се намалява с единица.

2. Инструкции PLA и PLP се изпълняват за четири микропроцесорни цикъла. След като през втория микропроцесорен цикъл кодът на операцията се декодира, през третия цикъл указателят на стека се увеличава с единица, като се извършва една "празна" операция четене от стека. През петия цикъл (първия цикъл от изпълнението на новата инструкция) данните се извличат от стека, като се извършва повторно четене от стека, този път от действителния адрес.

3. Инструкцията RTS се изпълнява за шест микропроцесорни цикъла, като операциите през първите три от тях съвпадат с тези за инструкциите PLA и PLP. През четвъртия и петия цикъл се извличат съответно младшият и старшият байт на програмния брояч, а през шестия цикъл с една "празна" операция четене новата стойност на програмния брояч се актуализира.

4. Инструкцията RTI също се изпълнява за шест микропроцесорни цикъла. Операциите в първите три са същите като при инструкции PLA и PLP, а в следващите три цикъла се извличат съответно стойността на регистъра на състоянието и младшият и старшият байт на програмния брояч.

Таблица П1.2

Неявно адресиране

Цикъл	Адрес на шина	Програмен брояч	Шина за данни	Изпълнявани действия
1	PC	PC + 1	КОП	Извлича се КОП
2	PC + 1	PC + 2	КОП 1	Новият КОП се игнорира. Старият КОП се декодира
3	PC + 1	PC + 2	КОП 1	Повторно се извлича кодът на новата операция. Операцията, определена от стария КОП, се изпълнява

Непосредствено адресиране – IMM. Извършва се с двубайтови инструкции, които се изпълняват за два микропроцесорни цикъла (вж. табл. П1.3). Първият байт на инструкцията еднозначно определя кода на операцията и метода на адресирането, а вторият байт е стойност, зададена от програмиста.

Таблица П1.3

Непосредствено адресиране

Цикъл	Адрес на шина	Програмен брояч	Шина за данни	Изпълнявани действия
1	PC	PC + 1	КОП	Извлича се КОП
2	PC + 1	PC + 2	Данни	Извличат се данните. КОП се декодира
3	PC + 2	PC + 3	КОП 1	Извлича се новият КОП. Операцията, определена от стария КОП, се изпълнява

Непосредственото адресиране е най–простият и бърз начин за обработка на константи. Използва се с инструкции ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA и SBC.

Пряко адресиране – ABS. При него се използват трибайтови инструкции, които се изпълняват за четири микропроцесорни цикъла (вж. табл. П1.4). Първият байт съдържа кода на операцията, а след-

ващите два байта – съответно младшия и старшия байт на ефективния адрес (EA).

Таблица П1.4

Пряко адресиране

Цикъл	Адрес на шина	Програмен брояч	Шина за данни	Изпълнявани действия
1	PC	PC + 1	КОП	Извлича се КОП
2	PC + 1	PC + 2	ADL	Извлича се младшият байт на ефективния адрес. Кодът на операцията се декодира
3	PC + 2	PC + 3	ADH	Извлича се старшият байт на ефективния адрес, след което адресът се формира
4	ADH,L	PC + 3	Данни	Извличат се данните, записани на ефективния адрес
5	PC + 3	PC + 4	КОП 1	Извлича се новият КОП. Операцията, определена от стария КОП, се изпълнява

Смята се, че прякото адресиране е "първичният и най–нормален" метод за адресиране на клетки от паметта на компютъра, а всички останали методи са негови производни. Използва се с почти всички инструкции: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, JMP, JSR, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY.

Прякото адресиране има следните особености:

1. Инструкцията JMP завършва с извличането на ефективния адрес, т.е. тя се изпълнява за три, а не за четири цикъла.

2. Инструкциите, които модифицират съдържанието на паметта (ASL, DEC, INC, LSR, ROL, ROR), се изпълняват за шест, а не за четири цикъла. През петия микропроцесорен цикъл инструкцията се изпълнява, а през шестия цикъл полученият резултат се записва обратно в паметта, като се използва същият ефективен адрес.

3. Инструкцията JSR се изпълнява за шест микропроцесорни цикъла, но по друга схема. След като във втория цикъл декодирането на КОП завърши и процесорът разпознае инструкция JSR, в третия цикъл се извършва подготовка за нейното изпълнение. През четвъртия и петия цикъл старшият и младшият байт на програмния брояч се записват в стека и едва през шестия цикъл се извлича ADH – старшият байт на ефективния адрес.

Пряко адресиране в нулевата страница – ZP. Прякото адресиране в нулевата страница е специален метод за адресиране, предназначен за ускоряване на работата на микропроцесора. Инструкци-

ите са по-къси и се изпълняват по-бързо. Те са двубайтови, като първият байт съдържа кода на операцията, а вторият – ефективния адрес в нулевата страница (микропроцесорът автоматично установява ADH на нула), и се изпълняват за три микропроцесорни цикъла (вж. табл. П1.5).

Таблица П1.5

Пряко адресиране в нулевата страница

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната операция завършва. Програмният брояч се увеличава с единица
2	PC + 1	ADL	Извлича се младшият байт на ефективния адрес. Кодът на операцията се декодира. Програмният брояч се увеличава с единица
3	00,PCL	Данни	Извличат се данните. Програмният брояч не се променя
4	PC + 2	КОП 1	Извлича се новият КОП. Операцията, определена от стария КОП, се изпълнява

Прякото адресиране в нулевата страница се използва с инструкциите: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY.

Както при прякото адресиране, така и при прякото адресиране в нулевата страница инструкциите за модифициране на съдържанието на клетка от паметта (ASL, DEC, LSR, ROL, ROR) изискват два микропроцесорни цикъла повече.

Относително адресиране – REL. Относителното адресиране се използва с инструкциите за условен преход BCC, BEQ, BMI, BNE, BPL, BCS, BVC, BCS. Това са двубайтови инструкции, първият байт на които съдържа кода на операцията, а вторият – отместването спрямо текущото съдържание на програмния брояч. Времето за тяхното изпълнение е от два до четири цикъла в зависимост от това, дали условието за прехода е удовлетворено и дали се извършва преминаване от една страница на паметта в друга (вж. табл. П1.6 до П1.8).

Трябва да се има предвид, че текущото съдържание на програмния брояч е равно на началния адрес на инструкцията за условен преход плюс две, т.е. съвпада с началния адрес на инструкцията, поста-

вена непосредствено след инструкцията за условен преход. Отместването може да бъде положително или отрицателно, в резултат на което ефективният адрес е в границите от PC – 126 до PC + 129.

Таблица П1.6

Относително адресиране (условието за преход не е изпълнено)

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната инструкция завършва и програмният брояч се увеличава с единица
2	PC + 1	Отместване	Извлича се отместването. Текущата инструкция се декодира и програмният брояч се увеличава с единица
3	PC + 2	КОП 1	Извлича се следващият КОП. Проверяват се флаговете и ако условието не е удовлетворено, програмният брояч се увеличава с единица

Таблица П1.7

Относително адресиране (условният преход се изпълнява без преминаване от една страница в друга)

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната инструкция завършва и програмният брояч се увеличава с единица
2	PC + 1	Отместване	Извлича се отместването. Текущата инструкция се декодира и програмният брояч се увеличава с единица
3	PC + 2	КОП 1	Извлича се КОП, който не се изпълнява. Проверяват се флаговете и ако условието е удовлетворено, отместването се прибавя към PCL
4	PC'	КОП 2	Извлича се следващият КОП. Формира се PC' – новата стойност на програмния брояч, след което съдържанието му се увеличава с единица

Таблица П1.8

Относително адресиране (условният преход се изпълнява с преминаване от една страница в друга)

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Завършва се изпълнението на предишната инструкция и програмният брояч се увеличава с единица
2	PC + 1	Отместване	Извлича се отместването. Текущата инструкция се декодира и програмният брояч се увеличава с единица
3	PC + 2	КОП 1	Извлича се КОП, който не се изпълнява. Флаговете се проверяват и ако условието е удовлетворено, се изчислява PC', като отместването се прибавя към PCL. Проверява се флаг ПРЕНОС
4	PC'	Невалидни данни	Извличат се данни, които се игнорират. При наличие на пренос извлеченият байт се игнорира. Формира се PCH, респ. PC''
5	PC''	КОП 2	Извлича се кодът на следващата операция. Издава се PC'' – новата стойност на програмния брояч, и той се увеличава с единица

Индексно адресиране – ABS,X и ABS,Y. Инструкциите, използващи този метод на адресиране, са трибайтови. Първият байт съдържа кода на операцията, а следващите два байта – т.нар. базов адрес. Ефективният адрес се получава, като към базовия адрес се прибави съдържанието на индексния регистър. В зависимост от използвания индексен регистър съществуват два вида индексно адресиране: по X и по Y, които протичат по една и съща схема. Инструкциите се изпълняват за четири или пет микропроцесорни цикъла в зависимост от това, дали има преминаване от една страница на паметта в друга или не (вж. табл. П1.9 и П1.10).

Индексното адресиране по X се използва с инструкции ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA, а индексното адресиране по Y – с инструкции ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC, STA.

Таблица П1.9

Индексно адресиране без преминаване от една страница на паметта в друга

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната операция завършва. Програмният брояч се увеличава с единица
2	PC + 1	BAL	Извлича се младшият байт на базовия адрес. Кодът на операцията се декодира. Програмният брояч се увеличава с единица
3	PC + 2	BAH	Извлича се старшият байт на базовия адрес. Изчислява се BAL + X и понеже няма пренос, директно се формира ефективният адрес. Програмният брояч се увеличава с единица
4	BAH,BAL + X	Данни	Извличат се данните, записани на ефективния адрес
5	PC + 3	КОП 1	Извлича се новият КОП. Изпълнява се операцията, определена със стария КОП

Таблица П1.10

Индексно адресиране с преминаване от една страница на паметта в друга

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Завършва изпълнението на предишната операция. Програмният брояч се увеличава с единица
2	PC + 1	BAL	Извлича се младшият байт на базовия адрес. Декодира се КОП. Програмният брояч се увеличава с единица
3	PC + 2	BAH	Извлича се старшият байт на базовия адрес. Изчислява се BAL + X. Програмният брояч се увеличава с единица
4	BAH,BAL + X	Данни	Извличат се и се игнорират данните, записани на адрес BAL + X. Изчислява се BAH + 1
5	BAH + 1, BAL + X	Данни	Извличат се данните
6	PC + 3	КОП 1	Извлича се новият КОП. Изпълнява се операцията, определена със стария КОП

Индексно адресиране в нулевата страница – ZP,X и ZP,Y. Индексното адресиране в нулевата страница, подобно на прякото адресиране в нулевата страница, е предназначено за ускоряване работата на микропроцесора. Инструкциите са двубайтови и се изпълняват за четири микропроцесорни цикъла (вж. табл. П1.11). Първият байт съдържа кода на операцията, а вторият – базовия адрес в нулевата страница.

С изключение на инструкции LDX и STX, които разбираемо използват индексиране по съдържанието на регистър Y, индексното адресиране в нулевата страница се изпълнява винаги с регистър X. Използва се от инструкции ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA, STY.

Ако при изчисляването на ефективния адрес се получи пренос, той се пренебрегва.

Таблица П1.11

Индексно адресиране в нулевата страница

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната инструкция завършва. Програмният брояч се увеличава с единица
2	PC + 1	ADL	Извлича се младшият байт на базовия адрес. Кодът на операцията се декодира. Програмният брояч се увеличава с единица
3	00,BAL	Невалидни данни	Извличат се и се игнорират невалидни данни от адрес 00,BAL. Формира се ефективният адрес в нулевата страница – BAL + X
4	00,BAL + X	Данни	Извличат се данните от ефективния адрес в нулевата страница
5	PC + 2	КОП 1	Извлича се новият КОП. Изпълнява се операцията, зададена със стария КОП

Непряко адресиране – IND. Непрякото адресиране е метод, при който ефективният адрес е предварително изчислен и записан на определен адрес в паметта. Използва се единствено от инструкция JMP. Това е трибайтова инструкция, първият байт на която съдържа кода на операцията, а следващите два байта – съответно младшия и старшия байт на т.нар. индиректен адрес, на който е записан ефективният адрес. Инструкцията се изпълнява за пет микропроцесорни цикъла. През първия, втория и третия цикъл се извличат съответно кодът на операцията, младшият и старшият байт на ин-

директния адрес. През четвъртия и петия цикъл от индиректния адрес се извличат съответно младшият и старшият байт на ефективния адрес.

Адресът, на който се намира старшият байт на ефективния адрес, се определя, като младшият байт на индиректния адрес се увеличи с единица. Ако се получи пренос, той се игнорира. Това означава, че когато се изпълняват инструкции от типа JMP ($XXFF), младшият байт на ефективния адрес се извлича от адрес $XXFF, а старшият – от адрес $XX00 вместо от адрес $XX + 100.

Индексно непряко адресиране – (IND, X). При този метод индиректният адрес не се задава направо, а се получава след модифициране на базовия адрес. Инструкциите са двубайтови. Първият байт съдържа кода на инструкцията, а вторият – базов адрес от нулевата страница.

Индиректният адрес също е адрес от нулевата страница. Той се получава, като към базовия адрес, включен във формата на инструкцията, се прибави съдържанието на регистър X. Ако се получи пренос, той се игнорира.

Индексното непряко адресиране се изпълнява с инструкции ADC, AND, CMP, EOR, LDA, ORA, SBC, STA.

Изпълнението на инструкциите при този метод на адресиране изисква шест микропроцесорни цикъла (вж. табл. П1.12).

Таблица П1.12

Индексно непряко адресиране

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната операция завършва. Програмният брояч се увеличава с единица
2	PC + 1	BAL	Извлича се младшият байт на базовия адрес. Кодът на операцията се декодира. Програмният брояч се увеличава с единица
3	00,BAL	Данни	Извличат се и се игнорират данните от адрес 00,BAL. Формира се BAL + X – младши байт на индиректния адрес
4	00,BAL+X	ADL	Извлича се младшият байт на ефективния адрес. Изчислява се BAL + X + 1
5	00,BAL+X+1	ADH	Извлича се старшият байт на ефективния адрес
6	ADH,ADL	Данни	Извличат се данните, записани на ефективния адрес
7	PC + 2	КОП 1	Извлича се новият КОП. Изпълнява се операцията, зададена със стария КОП

Непряко индексно адресиране – (IND), Y. Този метод комбинира непрякото и индексното адресиране. Инструкциите са двубайтови. Първият байт съдържа кода на операцията, а вторият – индиректен адрес от нулевата страница. Базовият адрес се определя от съдържанието на двете клетки с адреси IDL и IDL + 1. Ефективният адрес се получава от сумирането на базовия адрес със съдържанието на регистър Y.

Времето за изпълнение на инструкциите при непряко индиректно адресиране е пет или шест цикъла в зависимост от това, дали при сумирането на младшия байт на базовия адрес със съдържанието на регистър Y се получава преминаване от една страница в друга (вж. табл. П1.13 и П1.14).

Непряко индексно адресиране ползват инструкции ADC, AND, CMP, EOR, LDA, ORA, SBC, STA.

Таблица П1.13

Непряко индексно адресиране без преминаване от една страница в друга

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната операция завършва. Програмният брояч се увеличава с единица
2	PC + 1	IAL	Извлича се младшият байт на индиректния адрес. Кодът на операцията се декодира. Програмният брояч се увеличава с единица
3	00,IAL	BAL	Извлича се младшият байт на базовия адрес. Изчислява се IAL + 1
4	00,IAL + 1	BAH	Извлича се старшият байт на базовия адрес. Изчислява се BAL + Y. Няма пренос
5	BAH,BAL + Y	Данни	Извличат се данните, записани на ефективния адрес
6	PC + 2	КОП 1	Извлича се новият КОП. Изпълнява се операцията, зададена със стария КОП

Таблица П1.14

Непряко индексно адресиране с преминаване от една страница в друга

Цикъл	Адресна шина	Шина за данни	Изпълнявани действия
1	PC	КОП	Извлича се КОП. Изпълнението на предишната операция завършва. Програмният брояч се увеличава с единица
2	PC + 1	IAL	Извлича се младшият байт на индиректния адрес. Кодът на операцията се декодира. Програмният брояч се увеличава с единица
3	00,IAL	BAL	Извлича се младшият байт на базовия адрес. Изчислява се IAL + 1
4	00,IAL + 1	BAH	Извлича се старшият байт на базовия адрес. Изчислява се BAL + Y и се установява, че има пренос
5	BAH,BAL + Y	Данни	Извличат се и се игнорират данните, записани на адрес BAH,BAL + Y
6	BAH + 1,BAL + Y	Данни	Извличат се данните, заисани на ефективния адрес
7	PC + 2	КОП 1	Извлича се новият КОП. Изпълнява се операцията, зададена със стария КОП

П1.3. Набор от инструкции

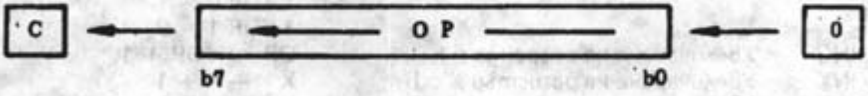
Инструкциите, изпълнявани от микропроцесора, са описани в табл. П1.15, а техните кодове при различните видове адресиране са показани в табл. П1.16.

Таблица П1.15

Набор инструкции на микропроцесор СМ630

Мнемоничен код	Значение	Описание
1	2	3
ADC	Събиране с пренос	A <— OP + A + P:C
AND	Логическа операция И	A <— OP AND A, 0 AND 0 = 0, 0 AND 1 = 0, 1 AND 0 = 0, 1 AND 1 = 1

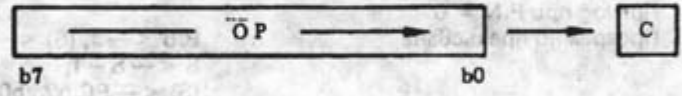
1	2	3
ASL	Аритметично преместване наляво	вж. фиг. П1.2



Фиг. П1.2. Аритметично преместване наляво

BCC	Преход при P:C = 0	
BCS	Преход при P:C = 1	
BEQ	Преход при P:Z = 1	
BIT	Проверка на адрес	OP AND A, P:N <— OP:b7, P:N <— OP:b6
BMI	Преход при P:N = 1	
BNE	Преход при P:Z = 0	
BPL	Преход при P:N = 0	
BRK	Програмно прекъсване	P:B <— 1, (S) <— P, S <— S – 1, (S) <— PC:b7...b0, S <— S – 1, (S) <— PC:b15...b8 S <— S – 1, P <— $20
BVC	Преход при P:V = 0	
BVS	Преход при P:V = 1	
CLC	Изчистване на преноса	P:C <— 0
CLD	Изчистване на десетичния режим	P:D <— 0
CLI	Разрешение на прекъсванията	P:I <— 0
CLV	Изчистване на препълването	P:V <— 0
CMP	Сравняване на акумулатора с операнда	A – OP, променя се P
CPX	Сравняване на регистър X с операнда	X – OP, променя се P
CPY	Сравняване на регистър Y с операнда	Y – OP, променя се P
DEC	Намаляване на операнда с 1	OP <— OP – 1
DEX	Намаляване на регистър X с 1	X <— X – 1
DEY	Намаляване на регистър Y с 1	Y <— Y – 1

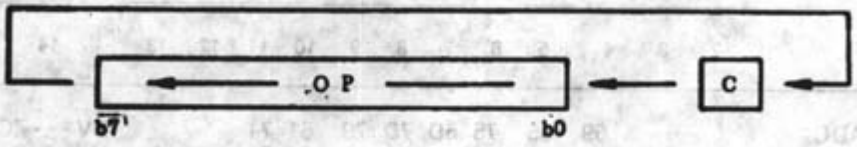
1	2	3
EOR	Логическа операция ИЗКЛЮЧВАЩО ИЛИ на акумулатора и операнда	A <— A Xor OP, 0 XOR 0 = 0, 0 XOR 1 = 1, 1 XOR 0 = 1, 1 XOR 1 = 0
INC	Увеличаване на операнда с 1	OP <— OP + 1
INX	Увеличаване на регистър X с 1	X <— X + 1
INY	Увеличаване на регистър Y с 1	Y <— Y + 1
JMP	Безусловен преход	PC <— EA
JSR	Преход към подпрограма	(S) <— PC:b15...b8, S <— S – 1, (S) <— PC:b7...b0, S <— S – 1, PC <— EA
LDA	Зареждане на акумулатора	A <— OP
LDX	Зареждане на регистър X	X <— OP
LDY	Зареждане на регистър Y	Y <— OP
LSR	Логическо преместване надясно	Вж. фиг. П1.3



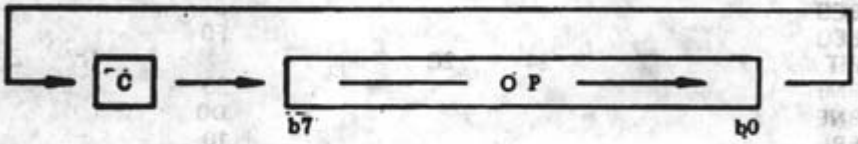
Фиг. П1.3. Логическо преместване надясно

NOP	"Празна" операция	
ORA	Операция логическо ИЛИ на акумулатора и операнда	A <— A OR OP, 0 OR 0 = 0, 0 OR 1 = 1, 1 OR 0 = 1, 1 OR 1 = 1
PHA	Запис на акумулатора в стека	(S) <— A, S <— S – 1
PLA	Възстановяване на акумулатора от стека	S <— S + 1, A <— (S)
PHP	Запис на регистър P в стека	(S) <— P, S <— S – 1
PLP	Възстановяване на регистър P от стека	S <— S + 1, P <— (S)

1	2	3
ROL	Ротация наляво	вж. фиг. П1.4
ROR	Ротация надясно	вж. фиг. П1.5



Фиг. П1.4. Ротация наляво



Фиг. П1.5. Ротация надясно

RTI	Връщане от прекъсване	S <— S + 1, P <— (S), S <— S + 1, PC:b7...b0 <— (S), S <— S + 1, PC:b15...b8 <— (S)
RTS	Връщане от подпрограма	S <— S + 1, PC:b7...b0 <— (S), S <— S + 1, PC:b15...b8 <— (S)
SBC	Изваждане на операнда от акумулатора с пренос	A <— A – OP – 1 – C
SEC	Установяване на преноса	P:C <— 1
SED	Установяване на десетичен режим	P:D <— 1
SEI	Забрана на прекъсванията	P:I <— 1
STA	Запис на акумулатора	OP <— A
STX	Запис на регистър X	OP <— X
STY	Запис на регистър Y	OP <— Y
TAX	Преместване на A в X	X <— A
TAY	Преместване на A в Y	Y <— A
TSX	Преместване на S в X	X <— S
TXA	Преместване на X в A	A <— X
TXS	Преместване на X в S	S <— X
TYA	Преместване на Y в A	A <— Y

Таблица П1.16

Кодове на инструкциите при различните видове адресиране

Мнемоничен код	Вид адресиране												Флагове[1]
	ACC	IMP	IMM	ZP	ZX	ABS	A,X	A,Y	IX	IY	REL	IND	
1	2	3	4	5	6	7	8	9	10	11	12	13	14
ADC			69	65	75	6D	7D	79	61	71			NV - - - ZC
AND			29	25	35	2D	3D	39	21	31			N - - - - Z -
ASL	0A			06	16	0E	1E						N - - - - ZC
BCC											90		
BCS											B0		
BEQ											F0		
BIT				24		2C							NV - - - Z -
BMI											30		
BNE											D0		
BPL											10		
BRK		00											
BVC											50		
BVS											70		
CLC		18											- - - - - - C
CLD		D8											- - - D - - -
CLI		58											- - - - I - -
CLV		B8											- V - - - - -
CMP			C9	C5	D5	CD	DD	D9	C1	D1			N - - - - ZC
CPX			E0	E4		EC							N - - - - ZC
CPY			C0	C4		CC							N - - - - ZC
DEC				C6	D5	CE	DE						N - - - - Z -
DEX		CA											N - - - - Z -
DEY		88											N - - - - Z -
EOR			49	45	55	4D	5D	59	41	51			N - - - - Z -
INC				E6	F6	EE	FE						N - - - - Z -
INX		E8											N - - - - Z -
INY		C8											N - - - - Z -
JMP						4C						6C	
JSR						20							
LDA			A9	A5	B5	AD	BD	B9	A1	B1			N - - - - Z -
LDX[2]			A2	A6	B6	EA	BE						N - - - - Z -
LDY			A0	A4	B4	AC	BS						N - - - - Z -
LSR	4A			46	56	4E	5E						N - - - - ZC
NOP		EA											
ORA			09	05	15	0D	1D	19	01	11			N - - - - Z -
PHA		48											
PLA		08											N - - - - Z -
PHP		68											

1	2	3	4	5	6	7	8	9	10	11	12	13	14
PLP		28											NVBDIZC
ROL	2A			26	36	2E	3E						N - - - - ZC
ROR	6A			66	76	6E	7E						N - - - - ZC
RTI		40											NVBDIZC
RTS		60											
SBC			E9	E5	F5	ED	FD	F9	E1	F1			NV - - - ZC
SEC		38											- - - - - - C
SED		F8											- - - D - - -
SEI		78											- - - - I - -
STA				85	95	8D	9D	99	81	91			
STX[2]				86	96	8E							
STY				84	94	8C							
TAX		AA											N - - - - Z-
TAY		A8											N - - - - Z -
TSX		BA											N - - - - Z -
TXA		8A											N - - - - Z -
TXS		9A											
TYA		98											N - - - - Z -
Дължина в байтове	1	1	2	2	2	3	3	3	2	2	2	3	

[1] Флагове, които се променят при изпълнение на инструкцията.
[2] Индексното адресиране в нулевата страница се извършва по съдържанието на регистър Y, а не на регистър X.

Приложение 2. Използване на паметта от програмата МОНИТОР

Паметта на персоналния компютър Правец–8А се дели на три основни категории: оперативна памет (RAM), постоянна памет (ROM) и област, използвана или предназначена за използване от входно–изходните устройства. Организацията на паметта е разгледана в гл.3. В гл.7 е показано използването на паметта от външните входно–изходни устройства, а в приложение 4 – адресите, на които са разположени вградените входно–изходни устройства и програмно-управляемите ключове. В това приложение се разглежда използването на паметта на компютъра от програмата МОНИТОР.

В табл. П2.1 е показано как е разпределена паметта на компютъра. Обърнете внимание, че свободната памет на компютъра, която се използва от всички програми, включително и от тези, написани на БЕЙСИК, е разположена под и над двете графични страници. За програмите, написани на БЕЙСИК, горната и долната граница на свободната памет се следят от показалците LOMEM и HIMEM. В областта $D000 – $FFFF има и друга свободна оперативна памет, но тя не може да се използва от програмите, написани на БЕЙСИК.

Таблица П2.1

Разпределение на паметта

Област	Тип на паметта	Предназначение
1	2	3
$0000 – $00FF	RAM	Нулева страница. Използва се от резидентното програмно осигуряване и от всички системни програми
$0100 – $01FF	RAM	Системен стек
$0200 – $02FF	RAM	Входен буфер
$0300 – $03FF	RAM	Вектори на програмата МОНИТОР. Част от тази област ($300 – $3EF) е свободна и може да се използва от малки програми, написани на машинен език
$0400 – $07FF	RAM	Първа текстова страница (съответно първа страница за графика с малка разделителна способност)

Продължение на таблица П2.1

1	2	3
$0800 – $0BFF	RAM	Втора текстова страница (съответно втора страница за графика с малка разделителна способност). Начало на програмата, написана на БЕЙСИК
$0C00 – $1FFF	RAM	Свободна памет
$2000 – $3FFF	RAM	Първа графична страница[1]
$4000 – $5FFF	RAM	Втора графична страница[1]
$6000 – $BFFF	RAM	Свободна памет. На адреси $9600 – $BFFF се зарежда ДОС
$C000 – $C08F	Вх–Изх	Вградени входно–изходни устройства и програмноуправляеми ключове (вж. приложение 4)
$C080 – $C0FF	Вх–Изх	Входно–изходни и управляващи регистри на интерфейсните модули (по 16 адреса за всеки от съединителите X1 – X7)
$C100 – $C7FF	Вх–Изх	Драйверни програми на интерфейсните модули (по 256 байта за всеки от съединителите X1 – X7)
	ROM	Част от разширението на резидентното програмно осигуряване
$C800 – $CFFF	Вх–Изх	Допълнителна памет за външните входно–изходни устройства. Използва се от всички модули
	ROM	Част от разширението на резидентното програмно осигуряване
$D000 – $FFFF	ROM	Резидентно програмно осигуряване: интерпретатор на БЕЙСИК и програма МОНИТОР
	RAM	Оперативна памет. Използва се от някои системни (ПроДОС) и потребителски програми и от програми на машинен език. Не се използва от програмите, написани на БЕЙСИК

[1] Ако не се вземат специални мерки за защита, областта на графичните страници се използва от програмите, написани на БЕЙСИК.

При първоначалното инициализиране (студен старт), при рестартиране (топъл старт), при обработка на прекъсвания и при нормална работа на компютъра програмата МОНИТОР използва адреси от нулева, първа, втора, трета страница и от първа текстова страница. Тук се дава обща представа за използването на паметта от програмата МОНИТОР.

Използване на нулевата страница. Използването на нулевата страница от програмата МОНИТОР е показано в табл. П2.2. Клетките с адреси $00 и $01 се използват при инициализиране на системата. Клетките с адреси $1F – $49 се използват при изпълнението

на основни функции на компютъра, като управление на текстовия прозорец, запазване състоянието на регистрите, работа с програмата МИНИАСЕМБЛЕР, дезасемблиране на програма, написана на машинен език, и др.

Таблица П2.2

Използване на нулевата страница от програмата МОНИТОР

Адрес	Клетка	Предназначение
1	2	3
$00	LOC0	Тези две клетки се използват от програмата
$01	LOC1	МОНИТОР при първоначалното зареждане. Чрез тях с помощта на индиректно адресиране последователно се проверяват съединители X7–X1 за включен контролер на флопидисково устройство. При откриване на такъв контролер по тяхното съдържание, което е от вида $00, $Cn, се извършва индиректен JMP към началото на драйверната програма
$1F	YSAV2	Използва се от програмното осигуряване, поддържащо 80-колонно изображение, за съхранение на съдържанието на регистър Y
$20	WNDLFT	Лява колона на текстовия прозорец. Стойността, записана на този адрес в режим TEXT40, е в границите 0 – 39, а в режим TEXT80 – в границите 0 – 79. Използва се от подпрограмите, изчисляващи BASL,H, като VTABZ, която установява BASL,H в съответствие с CV и WNDLFT. Ако стойността на WNDLFT се установи от потребителя, тя не става ефективна, докато не се изпълни някоя от подпрограмите, актуализиращи съдържанието на BASL,H, като VTABZ и др.
$21	WNDWDTH	Широчина на текстовия прозорец. В режим TEXT40 се изменя от 1 до 40–(WNDLFT), а в режим TEXT80 – от 1 до 80–(WNDLFT). При изпращане на символ към екрана от подпрограмата COUT, той се записва на позицията, определена от (BASL) и (CH). Стойността, записана в CH, се увеличава с единица и се сравнява с (WNDWDTH), за да се определи дали позицията на курсора не е вдясно от най–дясната колона на текстовия прозорец. По аналогичен начин необходимостта от подаване на нов ред се проверява и от подпрограмите за промяна на хоризонталната позиция на

1	2	3
		курсора. Клетките WNDWDTH, WNDTOP и WNDBTM се използват при операция СКРОЛ и от подпрограмите за изчистване на целия екран или част от него
$22	WNDTOP	Номер на най–горния ред от текстовия прозорец. Изменя се от 0 до 22 при текстов режим и от 20 до 22 при смесените режими. Показва от кой ред на екрана трябва да започне операция СКРОЛ, респ. къде трябва да се постави курсорът след завършване на операция HOME
$23	WNDBTM	Това е най–долният ред на текстовия прозорец. Изменя се от (WNDTOP) + 1 до 24. Използва се от подпрограмите CR и LF, за да се определи дали е достигнат последният ред от екрана, т.е. дали е необходима операция СКРОЛ
$24	CH	Хоризонтална позиция на курсора в режим ТЕКСТ40. Показва относителното изместване на курсора спрямо WNDLFT. Изменя се от 0 до (WNDWDTH)–1. Практически се използва от всички подпрограми за поддържане на екрана в режим ТЕКСТ40, като тези за извършване на операция СКРОЛ, за изчистване на екрана, за модифициране на хоризонталната позиция на курсора, за извеждане на символ на екрана, за отпечатване на ехото от клавиатурата, за поддържане на мигащ курсор и т.н.
$25	CV	Вертикална позиция на курсора върху екрана в режим ТЕКСТ40. Изменя се от 0 до 23 и отчита номера на реда спрямо началото на екрана, а не спрямо WNDTOP. Използва се от подпрограмите за вертикално преместване на курсора, от подпрограмите за изчисляване на BASL,H, от подпрограмите за изчистване на екрана и т.н. Стойността на CV се установява, като желаната стойност се зареди в регистър A и се извика подпрограмата TABV. Това може да стане и с команда POKE и подпрограма VTAB. В действителност при извеждане на символ на екрана номерът на реда се определя от BASL,H, а не от CV. Извикването на подпрограмите VTAB или TABV е необходимо само за предварително установяване на BASL,H в съответствие със стойностите на CV и WNDLFT
$26	GBASL	Адрес на клетка от екранната памет в режим

1	2	3
$27	GBASH	ГР40. Чрез подпрограмата PLOT се използва от почти всички подпрограми за поддържане на графичен режим с малка разделителна способност. Използва се и от подпрограмата SCRN при определяне на цвета на определено блокче от екрана. При изпълнение на подпрограмата GBASCALC съдържанието на регистър A се преобразува от номер на ред от екрана в адрес на клетка от екранната памет и се записва в GBASL,H (вж. и клетка MASK)
$28	BASL	Базов адрес на текущия ред в текстов режим.
$29	BASH	Съдържанието на тези клетки се определя от стойността на CV, респ. CV80, и WNDLFT. В режим ТЕКСТ40 то се установява от подпрограмите BASCALC и VTAB. Използва се при операция СКРОЛ, при редактиране (подпрограма GETLN) за извличане на следващия символ от екранната памет, при поддържане на мигащ курсор (подпрограма RDKEY) и т.н.
$2A	BAS2L	Тези два байта се използват за временно запазване на текущата стойност на BASL,H при операция СКРОЛ
$2B	BAS2H	
$2C	H2 LMNEM	Когато се използва от подпрограмите за поддържане на графика с малка разделителна способност, се бележи с H2 и съдържа координатата на крайната точка на линията, изчертавана с подпрограмата HLIN. Съдържанието ѝ се изменя от 0 до 39 и се установява, преди да се извика подпрограма HLIN. Когато се използва от подпрограмите за дезасемблиране, се бележи LMNEM. Подпрограмата INSTDSP записва пакетирания трибуквен мнемоничен код на инструкцията в клетки LMNEM и RMNEN
$2D	V2 RMNEM	Когато се използва от подпрограмите за поддържане на графика с малка разделителна способност, се бележи с V2 и съдържа координатата на долния край на вертикалната линия, изчертавана с подпрограма VLIN. Изменя се от 0 до 47 в режим ГР40 и от 0 до 39 в режим СМ40/40. Установява се, преди да е извикана подпрограмата VLIN. Индиректно се използва и от всички подпрограми за изчистване на екрана в режим ГР40. Използва се заедно с LMNEM от подпрограмата INSTDSP за записване на мнемоничния код на дезасемблираната инструкция

1	2	3
$2E	MASK FORMAT	Когато се използва от подпрограмите за поддържане на графика с малка разделителна способност (директно от подпрограма PLOT и индиректно от HLIN, VLIN, CLRSCR, CLRTOP и т.н.), се бележи с MASK. Съдържа $0F или $F0 в зависимост от това, дали се отнася за горното или долното поле на блокчето, посочено с GBASL,H. Когато се използва от подпрограмата GETFMT, респ. от подпрограмите от по–високо ниво – INSTDSP, INSTDSP1 и INSTDSP2, които извикват GETFMT, се бележи с FORMAT, съдържа формата на дезасемблираната инструкция и се използва при отпечатване на мнемоничния ѝ код. Клетка FORMAT се използва и от програмата МИНИАСЕМБЛЕР
$2F	LENGHT	Това поле се използва от подпрограмата GETFMT, респ. от INSTDSP за съхранение на дължината на дезасемблираната инструкция, от подпрограмата PCADJ за изчисляване на новата стойност на PCL,H и от програма МИНИАСЕМБЛЕР
$30	COLOR	Текущ цвят в режим ГР40. Установява се с подпрограмите SETCOL, NEXTCOL, CLRSCR3. Съвместно с MASK се използва от подпрограмата PLOT, респ. от HLIN, VLIN, CLRSCR, CLRTOP и т.н. при изчертаване на блокче, на хоризонтална или вертикална линия и при изчистване на екрана. Ако клетка COLOR се установява от подпрограма на потребителя, е необходимо кодът на цвета предварително да се умножи с $11
$31	MODE	Използва се от командния процесор на програмата МОНИТОР за управление на операциите при въвеждането на интервал след шестнайсетично число. Така например въвеждането на шестнайсетичен адрес, последван от двоеточие, установява MODE така, че при по–нататъшната обработка на входния ред интервалът означава края на шестнайсетичното число, което трябва да се запише в паметта на компютъра. Стойността на MODE се установява по различен начин от двоеточието, точката, знака плюс и знака минус
$32	INVFLG	Маска, използвана от подпрограмата COUT1 за получаване на формати НОРМАЛНО (INVFLG = $FF), ИНВЕРСНО (INVFLG = $3F) и МИГАЩО ВИДЕО (INVFLG = $7F). Установява се от подпрограмите SETNORM, SETINV и SETIFLG

1	2	3
$33	PROMPT	Оповестяващ символ на текущата управляваща програма. Подпрограмата GETLN го извежда на текущото изходно устройство при подготовката за въвеждане на един ред символи. В монитурен режим оповестяващият символ (*) се установява от подпрограмата MON
$34	YSAV	Използва се за временно запазване на съдържанието на регистър Y от командния процесор на програмата МОНИТОР, от програмата МИНИАСЕМБЛЕР, от програмата за преобразуване на десетични числа в шестнайсетични и др. Така например командният процесор на програмата МОНИТОР използва регистър Y като индексен регистър при сканиране на входния ред. При декодирането на команда съдържанието на регистър Y се запазва в YSAV, преди да се извърши преход към избраната обслужваща подпрограма. При връщането в командния процесор стойността на регистър Y се възстановява от YSAV, след което управлението отново се предава на подпрограмата за сканиране на входния ред NXTITM
$35	YSAV1	Използва се за временно запазване на съдържанието на регистър Y от подпрограмите за изход към екрана COUT1 и COUT2, от подпрограмата за изчистване до края на текущия ред от екрана CLEOLZ, респ. от подпрограмите от по–високо ниво, използващи CLEOLZ
$36 $37	CSWL CSWH	Изходен вектор. Сочи съответната програма за обслужване на текущото изходно устройство (за допълнителни подробности вж. гл.5)
$38 $39	KSWL KSWH	Входен вектор. Сочи съответната програма за обслужване на текущото входно устройство (за допълнителни подробности вж. гл.5)
$3A $3B	PCL PCH	Двубайтово поле за временно запазване на съдържанието на програмния брояч. Използва се от редица подпрограми на програмата МОНИТОР, като LIST, INSTDSP, BREAK, програмата МИНИАСЕМБЛЕР и др. Така например то се използва от програмата МИНИАСЕМБЛЕР за запомняне на мястото, където трябва да се запише следващата инструкция. При обработка на BREAK прекъсване в PCL,H се записва адресът на следващата инструкция. При изпълнение на команда L в монитурен режим (подпрограма LIST) съдържанието

1	2	3
		на PCL,H се увеличава така, че да сочи следващата инструкция. При изпълнение на команда G (подпрограма GO) това поле се използва за индиректен преход (JUMP INDIRECT)
$3C $3D	A1L A1H	Многофункционален псевдорегистър. Използва се от подпрограмата AUXMOVE, от подпрограмата за преобразуване на шестнайсетични числа в десетични, от програмата МИНИАСЕМБЛЕР и от почти всички подпрограми на командния процесор на програмата МОНИТОР, като MOVE, VERIFY, XAM, LIST, подпрограмите за събиране и изваждане на шестнайсетични числа и т.н. Така например, когато програмата МОНИТОР започне да обработва определена команда, клетка MODE се нулира. При сканиране на входния ред шестнайсетичните числа се записват първоначално в A2L,H, а след това се прехвърлят в A1L,H и A3L,H дотогава, докато съдържанието на клетка MODE е равно на нула. Ако при сканирането на входния ред се срещне знак плюс, знак минус, двоеточие или точка, клетката MODE се модифицира по съответен начин, а в A1L,H се запазва числото, намиращо се непосредствено преди срещнатата команда
$3E $3F	A2L A2H	Многофункционален псевдорегистър. Подобно на поле A1L,H и поле A2L,H се използва от редица подпрограми, като AUXMOVE, подпрограмата за преобразуване на десетични числа в шестнайсетични, програмата МИНИАСЕМБЛЕР, и голяма част от подпрограмите на командния процесор на програмата МОНИТОР. Така например при обработка на мониторните команди това поле се използва за приемане на шестнайсетичните числа от входния буфер. Когато при сканирането на въведения ред се разпознае команда, поле A2L,H съдържа последния въведен параметър. До този момент съдържанието на клетка MODE е равно на нула и всяка стойност, записана в A2L,H се копира в A1L,H и A3L,H. При разпознаването на символа < съдържанието на A2L,H се записва в A4L,H и A5L,H. Поле A2L,H се използва и като показалец, показващ края на областта от паметта, чието съдържание се извежда на екрана. Използва се също за показалец, показващ края на операциите M

1	2	3
		и V (подпрограми MOVE и VERIFY). То съдържа умалителя при изваждане и второто събираемо при събиране на шестнайсетични числа. Поле A2L,H съдържа началния адрес на подпрограмата, обслужваща съответния съединител, при изпълнението на команди от вида nCTRL–P или nCTRL–K
$40 $41	A3L A3H	Многофункционален псевдорегистър. При сканиране на входния ред от командния процесор поле A3L,H също както и поле A1L,H копира съдържанието на поле A2L,H. То се използва като работна област от подпрограмите за отпечатване на съдържанието на регистрите на микропроцесора REGDSP и REGDSP1
$42 $43	A4L A4H	Многофункционален псевдорегистър. Използва се от подпрограмата AUXMOVE, от програмата МИНИАСЕМБЛЕР, от командния процесор и т.н. Така например полета A4L,H и A5L,H се зареждат от поле A2L,H, когато при сканиране на въведения ред при обработка на мониторна команда се срещне символът <. Поле A4L,H се използва като показалец на приемащото поле при изпълнение на команда M (подпрограма MOVE) и като показалец на второто поле при изпълнение на команда V (подпрограма VERIFY)
$44 $45	A5L A5H	Многофункционален псевдорегистър. При определени условия дублира съдържанието на A2L,H. Освен от командния процесор се използва и от програмата МИНИАСЕМБЛЕР
$45 $46 $47 $48 $49	ACC XREG YREG STATUS SPNT	Петбайтово поле за запазване на съдържанието на регистрите на микропроцесора. Използва се от подпрограмите IOSAVE и IOREST. Съдържанието му се извежда на текущото изходно устройство от подпрограмата REGDSP при изпълнение на инструкция BRK и на команда CTRL–E. Регистрите на микропроцесора (с изключение на указателя на стека) се зареждат от това поле, преди да се извърши преходът към адреса, определен от командата G (подпрограма GO). Използването на това петбайтово поле има следните особености: Съдържанието на указателя на стека се запазва в поле SPNT, но никога не се възстановява. Съдържанието на регистър A се записва в поле ACC от общата част на подпрограмата за обра-

1	2	3
		ботка на прекъсвания тип IRQ и BREAK. При изпълнение на инструкция BRK съдържанието на всички регистри без регистър А се запазва от подпрограмата за обработка на BREAK прекъсване (чрез подпрограма IOSAVE1). При обработка на IRQ прекъсване програмата МОНИТОР не запазва съдържанието на останалите регистри
$4E $4F	RNDL RNDH	Двубайтово поле, използвано при генерирането на случайни числа. Неговото съдържание се променя непрекъснато от подпрограмата KEYIN, докато тя сканира клавиатурата за въведен символ

Използване на първа страница. Първа страница ($100 – $1FF) е апаратният стек на компютъра. Програмата МОНИТОР използва тази страница дотолкова, доколкото в нея има инструкции от типа PHA, PLA, PHP, PLP, JSR, RTS и т.н. Тя не установява указателя на стека нито при студен, нито при топъл старт, нито в какъвто и да е друг случай.

Използване на втора страница. Втора страница ($200 – $2FF) е входният буфер на компютъра. Подпрограмата GETLN записва въведените символи в този буфер, като използва съдържанието на регистър X като индекс. В резултат, когато управлението се върне на извикващата програма, въведените символи са записани от адрес $200 нагоре. Последният записан символ е управляващият символ CR (код $8D).

Използване на трета страница. Трета страница ($300 – $3FF) в по–голямата си част е свободна. Ето защо тя много често се използва от малки програми, написани на машинен език. Необходимо е да се подчертае, че не цялата трета страница е свободна. Така например операционната система ДОС3.3 ползва адреси $3D0 – $3EF. Програмата МОНИТОР също ползва някои адреси от третата страница (вж. табл. П2.3). За повече подробности вж. т.5.9.

Таблица П2.3

Използване на третата страница от програмата МОНИТОР

Адрес	Предназначение
$300 – $3EF	Не се използва от програмата МОНИТОР
$3F0 – $3F1	Вторичен BREAK вектор
$3F2 – $3F3	Вторичен RESET вектор. Използва се при топъл старт на компютъра
$3F4	Флаг ВКЛЮЧЕНО. Неговата стойност се използва като индикатор за това, дали изпълняваната RESET последователност е първата след включване на захранването (студен старт) или не. Ако при изпълнение на операцията ИЗКЛЮЧВАЩО ИЛИ между съдържанието на това поле и стойността $A5 се получи число, равно на съдържанието на поле $3F3, флагът е валиден и компютърът изпълнява топъл старт. В противен случай се изпълнява студен старт. Съдържанието на флаг ВКЛЮЧЕНО се актуализира с подпрограма SETPWRC
$3F5 – $3F7	Вектор на команда & от БЕЙСИК
$3F8 – $3FA	Вектор на команда CTRL–Y
$3FB – $3FD	Вторичен NMI вектор
$3FE – $3FF	Вторичен IRQ вектор

Използване на специалните клетки на първа текстова страница. В рамките на първа текстова страница (адреси $400 – $7FF) съществуват клетки, незаети от екранната памет. Те се намират на адресите, показани в табл. П2.4, и се използват от драйверните програми на интелигентните контролери на периферни устройства. Тъй като програмното осигуряване за поддържане на режим ТЕКСТ80 симулира контролер, включен към съединител X3, то ползва клетките от текстовата страница, предназначени за използване от съответния контролер (Вж. табл. П2.5).

Таблица П2.4

Клетки в първа текстова страница, използвани от контролерите на периферни устройства

Основен адрес	Съединител X1	X2	X3	X4	X5	X6	X7
$478	$479	$47A	$47B	$47C	$47D	$47E	$47F
$4F8	$4F9	$4FA	$4FB	$4FC	$4FD	$4FE	$4FF
$578	$579	$57A	$57B	$57C	$57D	$57E	$57F
$5F8	$5F9	$5FA	$5FB	$5FC	$5FD	$5FE	$5FF
$678	$679	$67A	$67B	$67C	$67D	$67E	$67F
$6F8	$6F9	$6FA	$6FB	$6FC	$6FD	$6FE	$6FF
$778	$779	$77A	$77B	$77C	$77D	$77E	$77F
$7F8	$7F9	$7FA	$7FB	$7FC	$7FD	$7FE	$7FF

Таблица П2.5

Използване на клетките от първа текстова страница от програмата МОНИТОР

Адрес	Клетка	Предназначение
1	2	3
$478	TEMP1	Клетка за временно съхранение на съдържанието на регистър А
$47B	OLDCH	В тази клетка се записва старата хоризонтална позиция на курсора
$4FB	MODE	Режим на работа. Няма нищо общо с установяването на 7– или 8–битов код и с клетка MODE от нулевата страница. Отделните битове на байта, записан в тази клетка, представляват флагове, които се поддържат и интерпретират по различен начин от различните системни програми
$57B	CH80	Хоризонтална позиция на курсора в режим ТЕКСТ80
$5FB	CV80	Вертикална позиция на курсора в режим ТЕКСТ80
$67B	CHAR	Клетка, през която се извършва прехвърлянето на символите от програмното осигуряване за поддържане на 80–колонно изображение

1	2	3
$6FB	XCOORD	Използва се от операционната система USCD за определяне на начина на влизане в областта $C100 – $CFFF
$77B	OLDBASL	Когато компютърът работи под управлението на операционната система USCD, в поле OLDBASL,H се записва стойността на базовия адрес BASL,H
$7F8	MSLOT	При първоначално зареждане програмата МОНИТОР записва на този адрес номера на съединителя, в който е открит контролер на флопидиско-во устройство във вида $Cn. За подобни цели се използва и при IRQ прекъсване
$7FB	OLDBASH	Вж. OLDBASL

Приложение 3. Подпрограми на програмата МОНИТОР

В табл. П3.1 началните адреси (входните точки) на подпрограмите са подредени по възходящ ред на адресите. За повече подробности относно предназначението на отделните подпрограми вж. гл.5.

Таблица П3.1

Входни точки на програмата МОНИТОР

Адрес	Име	Предназначение
1	2	3
$C305	BASICIN	Когато програмното осигуряване, поддържащо 80-колонно изображение, е активно, клавиатурата се обслужва от подпрограмата BASICIN. По функции тя е еквивалентна на подпрограмата KEYIN
$C307	BASICOUT	Обслужва екрана в режим ТЕКСТ80. Маскира символа, намиращ се в регистър А, със съдържанието на флаг INVFLG и го извежда на мястото на текущата позиция на курсора. Премества курсора с една позиция надясно. Еквивалентна е на подпрограма COUT1 в режим ТЕКСТ40
$C311	AUXMOVE	Прехвърля блокове от данни и програми между системната и допълнителната памет
$C314	XFER	Предава програмното управление на компютъра между програми или програмни сегменти, разположени в системната и допълнителната памет
$CCAA	SCROLLDN	В режим ТЕКСТ80 премества изображението с един ред надолу
$CDAA	QUIT	Изключва режим ТЕКСТ80 и установява режим ТЕКСТ40
$F800	PLOT	В режим ГР40 изчертава единично блокче в установения цвят
$F819	HLIN	В режим ГР40 изчертава хоризонтална права линия в установения цвят
$F828	VLIN	В режим ГР40 изчертава вертикална права линия в установения цвят
$F832	CLRSCR	В режим ГР40 изчиства целия екран, като запълва екранната памет с код $00. Ако се изпълни в режим ТЕКСТ40, запълва екрана със символа @ в ИНВЕРСНО ВИДЕО

1	2	3
$F836	CLRTOP	В режим ГР40 изчиства първите 40 реда от екрана
$F838	CLRSC2	В режим ГР40 изчиства част от екрана. Започва от първия ред (с номер 0) и завършва с реда, определен от съдържанието на регистър Y
$F83C	CLRSC3	В режим ГР40 изчиства част от екрана, като започва от горния му ляв ъгъл и свършва до реда и колоната, определени от съдържанието на регистър Y и на клетка V2 (адрес $2D)
$F847	GBASCALC	В режим ГР40 изчислява базовия адрес на клетка от екранната памет
$F85F	NEXTCOL	Променя установения цвят съгласно зависимостта COLOR = (COLOR + 3) MOD 16
$F864	SETCOL	В режим ГР40 установява текущия цвят в съответствие със съдържанието на регистър A
$F871	SCRN	В режим ГР40 определя кода на цвета на единичното блокче, чиито координати са определени от съдържанието на регистри A и Y
$F8D0	INSTDSP	Дезасемблира инструкцията на микропроцесора (с дължина от един до три байта), чийто начален адрес се намира в клетките от нулевата страница PCL, PCH (адреси $3A – $3B), извежда я с помощта на подпрограмата COUT на текущото изходно устройство и записва нейния формат в клетка FORMAT (адрес $2E) и нейната дължина в клетка LENGTH (адрес $2F). Съответства на команда L на програмата МОНИТОР, но за разлика от нея дезасемблира само една инструкция
$F940	PRNTYX	Отпечатва съдържанието на регистри Y и X като четириразредно шестнайсетично число
$F941	PRNTAX	Отпечатва съдържанието на регистри A и X като четириразредно шестнайсетично число
$F948	PRBLNK	Изпраща три интервала към текущото изходно устройство
$F94A	PRBL2	Изпраща от 1 до 256 интервала към текущото изходно устройство
$FA40	IRQ	Обща входна точка на подпрограма за обслужване на прекъсвания тип IRQ и BREAK. В зависимост от типа на прекъсването или се изпълнява преход по съдържанието на вторичния IRQ вектор, или управлението се предава на подпрограмата BREAK
$FA4C	BREAK	Подпрограма за обслужване на програмни прекъсвания. Завършва с преход по съдържанието на вторичния BREAK вектор

1	2	3
$FA59	OLDBRK	Обслужва програмни прекъсвания. Като краен резултат от нейното изпълнение инструкцията, определена от съдържанието на PCL, PCH, и съдържанието на регистрите на микропроцесора се извеждат на текущото изходно устройство и управлението се предава на програмата МОНИТОР
$FA62	RESET	Обща програма за обслужване на RESET прекъсвания
$FAD7	REGDSP	Отговаря на команда CTRL–E на програмата МОНИТОР. Използва подпрограмите CROUT и COUT, за да изпрати управляващия символ CR и да изведе съдържанието на регистрите на микропроцесора на текущото изходно устройство
$FADA	REGDSP1	Алтернативна входна точка на подпрограмата REGDSP. За разлика от нея не изпраща управляващия символ CR към текущото изходно устройство (не подава нов ред)
$FB1E	PREAD	Определя състоянието на аналоговите входове на компютъра
$FB2F	INIT	Установява текстов режим и нормализира размерите на текстовия прозорец
$FB39	SETTEXT	Установява екрана в текстов режим
$FB40	SETGR	Установява графичен режим за първите 40 реда от екрана. Последните 4 реда остават в текстов режим
$FB48	SETWND	Възстановява нормалните размери на текстовия прозорец
$FB5B	TABV	Алтернативна входна точка на подпрограмата за установяване на вертикалната позиция на курсора
$FB6F	SETPWRC	Актуализира флаг ВКЛЮЧЕНО и така разрешава изпълнението на топъл старт
$FBC1	BASCALC	Изчислява базовия адрес на текущия ред
$FBD9	BELL1	Обработва управляващия символ BEL. Проверява съдържанието на регистър A и ако то е равно на $87, предава управлението на подпрограмата BELL1A
$FBDD	BELL1A	Алтернативна входна точка на подпрограмата BELL1. Без да проверява съдържанието на регистрите, изработва сигнал с продължителност 0,1 s и честота 1 kHz и го изпраща към вградения високоговорител
$FBE4	BELL2	Алтернативна входна точка на подпрограмата

1	2	3
		BELL1. В зависимост от съдържанието на регистър Y изпраща тонове с различна честота към вградения високоговорител
$FBF0	STORADV	В режим TEKCT40 записва съдържанието на регистър A на текущата позиция на курсора. Премества курсора с една позиция надясно
$FBF4	ADVANCE	В режим TEKCT40 увеличава съдържанието на клетката, показваща хоризонталната позиция на курсора, с единица. Сравнява новото съдържание с широчината на текстовия прозорец и ако курсорът е достигнал края на реда, извиква подпрограма CR
$FC10	BS	В режим TEKCT40 премества курсора с една позиция наляво. Ако курсорът се намира в началото на текстовия прозорец, премества го в края на горния ред
$FC1A	UP	Премества курсора с един ред нагоре
$FC22	VTAB	Основна входна точка на подпрограмата за вертикално позициониране на курсора
$FC24	VTABZ	Алтернативна входна точка на подпрограмата за вертикално позициониране на курсора
$FC42	CLREOP	Изчиства екрана от текущата позиция на курсора до долния десен ъгъл на текстовия прозорец
$FC58	HOME	Изчиства текстовия прозорец и поставя курсора в неговия горен ляв ъгъл
$FC62	CR	В режим TEKCT40 премества курсора в началото на следващия ред от екрана
$FC66	LF	Премества курсора с един ред надолу, без да променя хоризонталната му позиция
$FC70	SCROLL	Премества изображението с един ред нагоре
$FC9C	CLROEL	Изчиства един ред от екрана, започвайки от текущата позиция на курсора
$FC9E	CLEOLZ	Изчиства част от текущия ред на екрана. Започва от позицията, определена от сумата на съдържанието на клетка BASL (базовия адрес) и на регистър Y, и завършва в десния край на текстовия прозорец
$FCA8	WAIT	Изработва времезакъснения с предварително зададена продължителност
$FCE9	MINIASM1	Входна точка в програмата МИНИАСЕМБЛЕР
$FD0C	RDKEY	Основна стандартна входна подпрограма за въвеждане на един символ. Поставя мигащ курсор на текущата позиция и чрез входния вектор KSW извиква съответната входна подпрограма (обикновено KEYIN или BASICIN), която записва кода на въведения символ в регистър A

1	2	3
$FD18	RDKEY1	Алтернативна входна точка на подпрограмата RDKEY. Изпълнява същите функции, но не поддържа курсора
$FD1B	KEYIN	Обслужва клавиатурата в режим ТЕКСТ40. Чете клавиатурния буфер и докато изчаква натискането на клавиш, поддържа генератора на случайни числа на адреси $4E – $4F. При натискането на клавиш изтрива курсора от екрана и записва кода на въведения символ в регистър А
$FD35	RDCHAR	Алтернативна подпрограма за въвеждане на един символ. Използва стандартната входна подпрограма RDKEY за получаване на символа. Обработва въведените ESC–последователности
$FD67	GETLNZ	Алтернативна входна точка на подпрограмата GETLN. Преди да предаде управлението на GETLN, подпрограмата GETLNZ изпраща управляващия символ CR към текущото изходно устройство
$FD6A	GETLN	Основна входна подпрограма за въвеждане на един ред символи (до 255). Извиква се, след като оповестяващият символ е зареден на адрес $33. Подпрограмата GETLN връща въведения ред във входния буфер ($200 – $2FF), а числото, определящо неговата дължина – в регистър X
$FD6F	GETLN1	Алтернативна входна точка на подпрограмата GETLN. За разлика от нея не отпечатва оповестяващия символ
$FD8B	CROUT1	Изчиства екрана от текущата позиция на курсора до края на реда и след това изпраща управляващия символ CR към текущото изходно устройство
$FD8E	CROUT	Изпраща управляващия символ CR към текущото изходно устройство
$FD92	PRA1	Изпраща управляващия символ CR, съдържанието на многоцелевия псевдорегистър A1L,A1H (като четириразредно шестнайсетично число) и тире (знак минус) към текущото изходно устройство
$FDA3	XAM8	Алтернативна входна точка на подпрограмата за извеждане на съдържанието на паметта. За разлика от подпрограмата XAM извежда съдържанието на една до осем последователни клетки от паметта на компютъра
$FDB3	XAM	Използва изходните подпрограми PRA1, PRBYTE и COUT, за да изведе съдържанието на определе-

1	2	3
		на област от паметта на текущото изходно устройство. Еквивалентна е на командата за извеждане на съдържанието на област от паметта от програмата МОНИТОР
$FDDA	PRBYTE	Извежда съдържанието на регистър А в шестнайсетичен код на текущото изходно устройство
$FDE3	PRHEX	Извежда младшите четири бита от съдържанието на регистър А като едноразредно шестнайсетично число
$FDED	COUT	Като използва изходния вектор CSW, извиква текущата подпрограма за обслужване на изхода на компютъра (обикновено COUT1 или BASICOUT). Извежда символа, чийто код се намира в регистър А
$FDF0	COUT1	Обслужва екрана в режим ТЕКСТ40. Маскира символа, намиращ се в регистър А, със съдържанието на флаг INVFLG и го извежда на мястото на текущата позиция на курсора. Премества курсора с една позиция надясно. Еквивалентна е на подпрограма BASICOUT в режим ТЕКСТ80
$FDF6	COUTZ	Действието ѝ е същото като COUT1, но пренебрегва състоянието на флаг INVFLG
$FE2C	MOVE	Копира съдържанието на блок от паметта. Отговаря на командата М от програмата МОНИТОР
$FE36	VERIFY	Сравнява съдържанието на две области от паметта. Еквивалентна е на команда V от програмата МОНИТОР
$FE5E	LIST	Отговаря на команда L на програмата МОНИТОР. Дезасемблира 20 последователни инструкции на микропроцесора и чрез подпрограмата COUT ги извежда на текущото изходно устройство
$FE63	LIST2	Алтернативна входна точка на подпрограмата LIST. В зависимост от съдържанието на регистър А дезасемблира от 1 до 255 инструкции, започвайки от инструкцията, чийто начален адрес се намира в клетки PCL, PCH
$FE80	SETINV	Установява формат ИНВЕРСНО ВИДЕО
$FE84	SETNORM	Установява формат НОРМАЛНО ВИДЕО
$FE86	SETIFLG	Записва съдържанието на регистър Y в клетка INVFLG
$FE89	SETKBD	Записва входната точка ($FD1B) на подпрограмата за обслужване на клавиатурата KEYIN във входния вектор KSW. Еквивалентна е на командата на БЕЙСИК IN# 0

1	2	3
$FE8B	INPORT	Записва началния адрес на текущата драйверна програма, обслужваща входа на компютъра, във вида $Cn00 във входния вектор KSW. Еквивалентна е на командата на БЕЙСИК IN# n
$FE93	SETVID	Записва входната точка на подпрограмата COUT1, обслужваща екрана в режим ТЕКСТ40, в изходния вектор CSW. Еквивалентна е на командата на БЕЙСИК PR# 0
$FE95	OUTPORT	Записва началния адрес на текущата драйверна програма, обслужваща изхода на компютъра, във вида $Cn00 в изходния вектор CSW. Еквивалентна е на командата на БЕЙСИК PR# n
$FEB6	GO	Отговаря на командата G на програмата МОНИТОР. Предава управлението на подпрограмата, чийто начален адрес е записан в псевдорегистъра A1L,A1H
$FEEE	MINIASM	Входна точка в програмата МИНИАСЕМБЛЕР
$FF2D	PRERR	Изпраща съобщението ERR и управляващия символ BEL към текущото изходно устройство
$FF3A	BELL	Изпраща управляващия символ CTRL–G (BEL) към текущото изходно устройство
$FF3F	IOREST	Възстановява съдържанието на регистрите на микропроцесора от клетките на нулевата страница ACC, XREG, YREG, STATUS и SPNT
$FF4A	IOSAVE	Записва съдържанието на регистрите на микропроцесора в клетките на нулевата страница ACC, XREG, YREG, STATUS и SPNT
$FF4C	IOSAVE1	Алтернативна входна точка на подпрограмата IOSAVE. Запазва съдържанието на регистри X и Y, на регистъра на състоянието и на указателя на стека в клетки XREG, YREG, STATUS и SPNT от нулевата страница
$FF65	MON	Алтернативна входна точк в командния процесор на програмата МОНИТОР. Преди да предаде управлението на MONZ, изчиства десетичния режим на работа на микропроцесора и извиква подпрограмата BELL
$FF69	MONZ	Основна входна точка в командния процесор на програмата МОНИТОР. Използването ѝ е еквивалентно на използването на командата от БЕЙСИК CALL –151
$FFA7	GETNUM	Преобразува от две– до четириразреден шестнайсетичен низ в едно– или двубайтово шестнайсетично число и го записва в многоцелевия псевдорегистър от нулевата страница A2L,A2H (адреси $3E – $3F)

Приложение 4. Вградени входно–изходни устройства и програмноуправляеми ключове

Областта от адресното пространство на компютъра, предназначена за използване от вградените входно–изходни устройства и програмноуправляемите ключове, звема 144 байта (адреси $C000 – $C08F). Разпределението на тази област е показано в табл. П4.1, а използването ѝ има следните особености:

1. Обръщението към някои от вградените входно–изходни устройства се извършва на повече от един адрес. Независимо от това използвайте само адресите, показани в табл. П4.1.

2. На някои адреси има повече от едно устройство. Ето защо обръщението към адресите в разглежданата област трябва да се извършва само с командата, означена в табл. П4.1.

3. За повече подробности относно вградените входно–изходни устройства и предназначението на програмноуправляемите ключове вж. гл.2 и 3.

Таблица П4.1

Вградени входно–изходни устройства и програмноуправляеми ключове

Адрес	Функция
1	2
$C000	1. Данни от клавиатурата. Младшите седем бита, прочетени от този адрес, съдържат кода, генериран от клавиатурата. Старшият бит е бит СТРОБ 2. Операция запис на този адрес нулира ключ ЗАП80
$C001	Операция запис на този адрес установява ключ ЗАП80
$C002	Операция запис на този адрес установява ключ ЧСП (нулира ключ ЧДП)
$C003	Операция запис на този адрес установява ключ ЧДП
$C004	Операция запис на този адрес установява ключ ЗСП (нулира ключ ЗДП)
$C005	Операция запис на този адрес установява ключ ЗДП
$C006	Операция запис на този адрес нулира ключ ПСХРОМ
$C007	Операция запис на този адрес установява ключ ПСХРОМ
$C008	Операция запис на този адрес установява ключ НС (нулира ключ АНС)

1	2
$C009	Операция запис на този адрес установява ключ АНС
$C00A	Операция запис на този адрес нулира ключ ПСЗРОМ
$C00B	Операция запис на този адрес установява ключ ПСЗРОМ
$C00C	Операция запис на този адрес нулира ключ 80КОЛ
$C00D	Операция запис на този адрес установява ключ 80КОЛ
$C00E	Операция запис на този адрес нулира ключ АСМ
$C00F	Операция запис на този адрес установява ключ АСМ
$C010	Флаг НАТИСНАТ КЛАВИШ и програмноуправляем ключ за нулиране на бит СТРОБ 1.Старшият бит на байта, прочетен от този адрес, показва състоянието на флаг НАТИСНАТ КЛАВИШ 2.Обръщението към този адрес нулира бит СТРОБ на клавиатурата
$C011	Старшият бит на байта, прочетен от този адрес, показва дали е разрешена първа (D7 = 0) или втора страница (D7 = 1) на оперативната памет от областта $D000 – $DFFF
$C012	Старшият бит на байта, прочетен от този адрес, показва дали е разрешено четенето от постоянната (D7 = 0) или от оперативната памет (D7 = 1) в областта $D000 – $FFFF
$C013	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ЧДП
$C014	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ЗДП
$C015	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ПСХРОМ
$C016	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ АНС
$C017	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ПСЗРОМ
$C018	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ЗАП80
$C019	Старшият бит на байта, прочетен от този адрес, показва състоянието на сигнала за гасене на обратния ход на лъча по кадри
$C01A	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ТЕКСТ
$C01B	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ СМ
$C01C	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ СТР2
$C01D	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ ГР.ВРС
$C01E	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ АСМ
$C01F	Старшият бит на байта, прочетен от този адрес, показва състоянието на ключ 80КОЛ

1	2
$C030	Всяко обръщение[1] към този адрес алтернативно променя състоянието на схемата за управление на високоговорителя
$C040	Всяко обръщение[1] към този адрес изработва един стробиращ импулс
$C050	Обръщението към този адрес нулира ключ ТЕКСТ (установява ключ ГР)
$C051	Обръщението към този адрес установява ключ ТЕКСТ
$C052	Обръщението към този адрес нулира ключ СМ
$C053	Обръщението към този адрес установява ключ СМ
$C054	Обръщението към този адрес установява ключ СТР1 (нулира ключ СТР2)
$C055	Обръщението към този адрес установява ключ СТР2
$C056	Обръщението към този адрес установява ключ ГР.НРС (нулира ключ ГР.ВРС)
$C057	Обръщението към този адрес установява ключ ГР.ВРС
$C058	Обръщението към този адрес нулира (изключва) цифров изход ЦИЗХ0
$C059	Обръщението към този адрес установява (включва) цифров изход ЦИЗХ0
$C05A	Обръщението към този адрес нулира (изключва) цифров изход ЦИЗХ1
$C05B	Обръщението към този адрес установява (включва) цифров изход ЦИЗХ1
$C05C	Обръщението към този адрес нулира (изключва) цифров изход ЦИЗХ2
$C05D	Обръщението към този адрес установява (включва) цифров изход ЦИЗХ2
$C05E	Обръщението към този адрес нулира (изключва) цифров изход ЦИЗХ3
$C05F	Обръщението към този адрес установява (включва) цифров изход ЦИЗХ0
$C060	Старши бит на данните от клавиатурата и флаг 8–БИТОВ КОД 1. Старшият бит на байта, прочетен от този адрес, е старши бит (D7) на данните от клавиатурата 2. Бит 6 (D6) на байта, прочетен от този адрес, показва състоянието на флаг 8–БИТОВ КОД 3. Операция запис на този адрес установява (D0 = 1) или нулира (D0 = 0) флаг 8–БИТОВ КОД
$C061	Старшият бит на байта, прочетен от този адрес, показва състоянието на цифров вход ЦВХ0
$C062	Старшият бит на байта, прочетен от този адрес, показва състоянието на цифров вход ЦВХ1
$C063	Старшият бит на байта, прочетен от този адрес, показва състоянието на цифров вход ЦВХ2
$C064	Старшият бит на байта, прочетен от този адрес, показва състоянието на аналогов вход АВХ0
$C065	Старшият бит на байта, прочетен от този адрес, показва състоянието на аналогов вход АВХ1

1	2
$C066	Старшият бит на байта, прочетен от този адрес, показва състоянието на аналогов вход ABX2
$C067	Старшият бит на байта, прочетен от този адрес, показва състоянието на аналогов вход ABX3
$C070	Обръщението към този адрес нулира аналоговите входове на компютъра
$C080	Една операция четене от този адрес разрешава четенето от оперативната памет от областта $D000 – $FFFF и забранява записа в нея. Избрана е втората страница от 4 Кбайта
$C081	Две последователни операции четене от този адрес разрешават четенето от постоянната и записа в оперативната памет от областта $D000 – $FFFF. Избрана е втората страница от 4 Кбайта
$C082	Една операция четене от този адрес разрешава четенето от постоянната и забранява записа в оперативната памет от областта $D000 – $FFFF. Избрана е втората страница от 4 Кбайта
$C083	Две последователни операции четене от този адрес разрешават четенето и записа в оперативната памет от областта $D000 – $FFFF. Избрана е втората страница от 4 Кбайта
$C088	Една операция четене от този адрес разрешава четенето от оперативната памет от областта $D000 – $FFFF и забранява записа в нея. Избрана е първата страница от 4 Кбайта
$C089	Две последователни операции четене от този адрес разрешават четенето от постоянната и записа в оперативната памет от областта $D000 – $FFFF. Избрана е първата страница от 4 Кбайта
$C08A	Една операция четене от този адрес разрешава четенето от постоянната и забранява записа в оперативната памет от областта $D000 – $FFFF. Избрана е първата страница от 4 Кбайта
$C08B	Две последователни операции четене от този адрес разрешават четенето и записа в оперативната памет от областта $D000 – $FFFF. Избрана е първата страница от 4 Кбайта

[1] Обръщението към тези адреси да се извършва с операция четене, а не с операция запис, защото при определени условия изпълнението на операция запис води до двукратно издаване на един и същи адрес.

Приложение 5. Запазени думи в БЕЙСИК

Персоналният компютър Правец–8А интерпретира запазените думи от табл. П5.1 като команди в БЕЙСИК независимо от това, в каква последователност или в каква комбинация с други думи ги среща. Единственото изключение е, когато запазените думи са част от низ, затворен в кавички, или част от съобщение след оператор REM или DATA.

Внимание!

1. Интерпретаторът на БЕЙСИК разпознава запазените думи независимо от това дали са написани с малки или с главни латински букви.

2. Запазените думи не могат да участват в имена на променливи и масиви.

3. На всяка от запазените думи в паметта на компютъра съответства само един байт. Този байт е с установен старши бит (вж. табл П5.1) и се нарича код на командата на БЕЙСИК.

4. Всички команди на БЕЙСИК се представят само с една запазена дума. Изключение прави операторът LINE INPUT, който се представя с двойката запазени думи LINE и INPUT.

5. Интерпретаторът на РАЗШИРЕН БЕЙСИК не разпознава правилно запазената дума TO, ако последният символ, различен от интервал, преди нея е малката или главната латинска буква A или ако буквите T и O са разделени с един или повече интервали.

Таблица П5.1

Запазени думи в БЕЙСИК

Дума	Код	Дума	Код	Дума	Код
1	2	3	4	5	6
ABS	212	AND	205	ASC	230
AT	197	ATN	225	CALL	140
CHR$	231	CLEAR	189	COLOR=	160
CONT	187	CONV	183	COS	222
DATA	131	DEF	184	DEL	133
DIM	134	DRAW	148	END	128
EXP	221	FLASH	159	FN	194
FOR	129	FRE	214	GET	190
GOSUB	176	GOTO	171	GR	136

1	2	3	4	5	6
HCOLOR=	146	HGR	145	HGR2	144
HIMEM:	163	HLIN	142	HOME	151
HPLOT	147	HTAB	150	IF	173
IN#	139	INPUT	132	INT	211
INVERSE	158	LEFT$	232	LEN	227
LET	170	LINE	182	LIST	188
LOG	220	LOMEM:	164	MID$	234
NEW	191	NEXT	130	NORMAL	157
NOT	198	NOTACE	156	ON	180
ONERR	165	OR	206	PDL	216
PEEK	226	PLOT	141	POKE	185
POP	161	POS	217	PRINT	186
PR#	138	REM	178	RESTORE	174
RESUME	166	RETURN	177	RIGHT$	233
RND	219	ROT=	152	RUN	172
SCALE=	153	SCR(	215	SETMOD	154
SGN	210	SIN	223	SPC(	195
SPEED=	169	SQR	218	STEP	199
STOP	179	STORE	168	STR$	228
TAB(	192	TAN	224	TEXT	137
THEN	196	TO	193	TRACE	155
USR	213	VAL	229	VLIN	143
VTAB	162	WAIT	181	XDRAW	149
&	175				

Приложение 6. Съобщения за грешки в интерпретатора на БЕЙСИК

Резидентният интерпретатор на БЕЙСИК поддържа осемнадесет различни съобщения за грешки. Всеки тип съобщение за грешка има определен код, който се нарича код на грешката. Когато при изпълнение на програмата, написана на БЕЙСИК, интерпретаторът открие грешка, той записва номера на програмния ред, при изпълнение на който е открита грешката, на адреси $DA – $DB, анализира типа на грешката и записва нейния код на адрес $DE и извежда съответното съобщение за грешка на текущото изходно устройство. Съобщението за грешка е предшествано от въпросителен знак (?), а след него е даден номерът на програмния ред, при изпълнението на който е генерирано съобщението.

Ако преди да се открие грешка в програмата, написана на БЕЙСИК, е изпълнен оператор ONERR GOTO (старшият бит от съдържанието на клетка $D8 е установен), кодът на грешката и номерът на реда се записват в съответните клетки на нулевата страница, но съобщение за грешка не се издава. Вместо това управлението се предава на програмата за обработка на грешки, началният адрес на която е зададен с оператора ONERR GOTO.

В таблица П6.1 са дадени кодовете и съобщенията за грешки и кратко описание на ситуацията, при която се генерират. Номерът N на програмния ред, при изпълнението на който се е получило съобщение за грешка, се изчислява с израза

N = PEEK ($DA) + 256 * PEEK ($DB)

Таблица П6.1

Кодове и съобщения за грешки

Код	Съобщение за грешка
1	2
$00	NEXT WITHOUT FOR ERROR. Генерира се при опит за изпълнение на оператор NEXT, на който не съответства активен оператор FOR
$10	SYNTAX ERROR. Показва, че е открита грешка в правописа на командата или на програмния ред или грешка, която не се описва от останалите съобщения за грешки
$16	RETURN WITHOUT GOSUB ERROR. Показва, че сумата от изпълнените оператори POP и RETURN надхвърля броя на изпълнените оператори GOSUB

1	2
$2A	OUT OF DATA ERROR. Генерира се при изпълнение на оператор READ и показва, че списъкът от данни, въведени с оператор DATA, е изчерпан
$35	ILLEGAL QUANTITY ERROR. В най–общ смисъл показва, че стойността на параметъра (константа, променлива или израз), използван с даден оператор или функция, е извън допустимите граници
$45	OVERFLOW ERROR. Показва, че полученото число е по–голямо от най–голямото или по–малко от най–малкото число, с което може да се работи в БЕЙСИК
$4D	OUT OF MEMORY ERROR. Показва, че или програмата, написана на БЕЙСИК, не се побира в свободната памет на компютъра (програмата е много дълга, броят на използваните променливи е много голям, LOMEM е установен прекалено високо или HIMEM – прекалено ниско и т.н.), или стекът е препълнен (броят на включените един в друг цикли от типа FOR – NEXT е по–голям от 10, на вложените една в друга скоби – от 36, на включените една в друга подпрограми – от 24 и т.н.)
$5A	UNDEF'D STATEMENT ERROR. Генерира се при опит за обръщение към несъществуващ ред на програмата
$6B	BAD SUBSCRIPT ERROR. Показва, че масивът е с погрешно зададена размерност или че индексната променлива е извън допустимите граници
$7B	REDIM'D ARRAY ERROR. Генерира се при опит за повторно дефиниране на масив
$85	DEVISION BY ZERO ERROR. Показва, че стойността на делителя (константа, променлива или израз) е нула
$A3	TYPE MISMATCH ERROR. Показва, че има неправилна употреба на числени, респ. символни изрази
$B0	STRING TOO LONG ERROR. Показва, че дължината на символен низ надхвърля 255 символа
$BF	FORMULA TOO COMPLEX ERROR. Генерира се, когато се изпълняват повече от две конструкции за условен преход от типа IF – THEN, условието на които е зададено със символен низ
$E0	UNDEF'D FUNCTION ERROR. Генерира се при опит за обръщение към недефинирана функция
$FE	REENTER. Показва, че стойността на променливата, въведена с оператор INPUT, не съответства на типа на използваната променлива
$FF	BREAK. Генерира се при изпълнение на оператор STOP или при прекъсване на изпълнението на програмата с управляващата последователност CTRL–C
	ILLEGAL DIRECT ERROR. Това съобщение няма код на грешка. Генерира се в директен режим на работа с компютъра и показва, че е направен опит за изпълнение на някой от операторите GET или INPUT или на функцията DEF FN

Приложение 7. Таблици на фигури в РАЗШИРЕН БЕЙСИК

В резидентния интерпретатор на РАЗШИРЕН БЕЙСИК има две функционално различни групи от оператори за поддържане на графично изображение с голяма разделителна способност. Двете групи са универсални (в смисъл, че позволяват изчертаването на фигури с произволна сложност), но се различават по начина на изчертаване на фигурите. Към първата група се причислява операторът HPLOT, с който е възможно изчертаване на точки и прави с предварително зададени координати. С този оператор фигурата се изчертава еднократно точка след точка и права след права. Втората група включва операторите DRAW, XDRAW, ROT и SCALE, които могат да се използват само с предварително дефинирани фигури. Фигурите се описват (кодират) в таблица на фигурите. Тази таблица се записва в паметта на компютъра като част от програмата, написана на БЕЙСИК, или независимо от нея, а фигурите, описани в нея, могат да се използват многократно. Това позволява една и съща фигура да се изчертава на различни позиции (оператори DRAW и XDRAW), с различна големина (операторът SCALE), различна ориентация (операторът ROT) и различен цвят (операторът HCOLOR). Макар че създаването на таблици на фигури е бавен и трудоемък процес, тяхното използване е неизбежно при решаването на проблеми, които изискват многократно изчертаване на една и съща фигура на различни позиции на екрана: при текстовите редактори, които работят в графичен режим и използват разширено символно множество; програмите, използващи графичната страница за едновременно изобразяване на графика и текст, компютърната анимация и др.

Описание на фигурите. Описанието на фигурата, необходимо за създаването на таблицата на фигурите, започва със съставяне на последователност от микроинструкции, наречени вектори, които управляват хипотетичния писец, чертаещ фигурата. Векторите задават посоката на движение (нагоре, надясно, надолу и наляво) и действието (чертае или не чертае) на писеца. Осемте възможни комбинации (посока и действие) се кодират с три бита (вж. табл. П7.1). Младшите два бита от кода на вектора показват посоката на преместване на писеца, а старшият бит (бит 2) – извършваното действие.

Таблица П7.1

Кодове на векторите за описание на фигури

Посока на преместването	Код на вектора	
	Не чертае	Чертае
Нагоре	000	100
Надясно	001	101
Надолу	010	110
Наляво	011	110

Описанието на фигурата продължава, като получените трибитови кодове се групират (опаковат) в байтове. В зависимост от описанието на конкретната фигура в един байт могат да се включат кодовете на два или три вектора. За целта осемте бита на байта се разделят на три групи (вж. табл. П7.2), като двете групи са пълни (имат по три бита), а третата е непълна (приема се, че липсващият старши бит е нула). В тази група могат да се поставят само кодовете на векторите, съответстващи на просто преместване (без чертане) наляво (код 001), надолу (код 010) и надясно (код 011). Поставянето на 00 в трета група няма отношение към вектора, означаващ просто преместване нагоре (код 000), а се използва за запълване на байта.

При опаковането на векторите съществуват още две ограничения, свързани с използването на код 000:

– съставният байт не може да завършва с код 000 (в първа група), ако това не е последен байт в описанието на фигурата;

– кодът 000 във втора група не означава просто преместване нагоре, ако в трета група има код 00.

Краят на описанието на фигурата се маркира с байт, съдържащ нули във всички групи.

Таблица П7.2

Формиране на съставния байт

Бит	b7 b6	b5 b4 b3	b2 b1 b0
Група	Трета	Втора	Първа

Съставяне на таблицата на фигурите. След като се опишат всички фигури, се пристъпва към съставяне на таблицата на фигурите (вж. табл. П7.2). Тя съдържа:

– един байт, който показва колко са фигурите в таблицата;

– последователност от двубайтови показалци (за всяка фигура), които показват какво е отместването на описанието на фигурата спрямо началото на таблицата;

– описанията на всички фигури.

Таблица П7.3

Таблица на фигурите

Номер на байта	Значение на байта
0	Брой на фигурите в таблицата (между 0 и 255)
1	Не се използва. Обикновено е нула
2	Младши байт на отместването на описанието на първата фигура по отношение на началото на таблицата
3	Старши байт на отместването на описанието на първата фигура по отношение на началото на таблицата
4	Младши байт на отместването на описанието на втората фигура по отношение на началото на таблицата
5	Старши байт на отместването на описанието на втората фигура по отношение на началото на таблицата
...	
2n + 2	Младши байт на отместването на описанието на *n*–тата фигура по отношение на началото на таблицата
2n + 3	Старши байт на отместването на описанието на *n*–тата фигура по отношение на началото на таблицата
2n + 4	Описание на първата фигура
...	
	Описание на втората фигура
...	
	Описание на *n*–тата фигура

Въвеждане на таблицата на фигурите в паметта на компютъра. След като се състави таблицата на фигурата и се определи нейната дължина, тя трябва да се въведе в паметта на компютъра. Същевременно трябва да се вземат мерки за защита на областта, заета от таблицата, от програмата, написана на БЕЙСИК, и от нейните променливи. Процедира се в следната последователност:

– преди да е изпълнена каквато и да е команда от програмата, написана на БЕЙСИК, използваща символни низове, с команда PRINT PEEK ($73) + 256 * PEEK ($74) се определя текущата стойност на HIMEM;

– изчислява се новата стойност на HIMEM, като от текущата стойност се изважда дължината на таблицата на фигурите;

– на адреси $73 и $74 се записват съответно младшият и старшият байт на новата стойност на HIMEM;

– с програма, написана на БЕЙСИК, или директно в монитoрен режим таблицата на фигурите се въвежда в защитената област от паметта (областта, ограничена от старата и новата стойност на HIMEM);

– адресът на началото на таблицата на фигурите се указва на интерпретатора на БЕЙСИК, като неговият младши и старши байт се запишат съответно на адреси $EC и $ED.

Таблицата на фигурите може да запише на дискета като двоичен файл и да се използва от различни програми.

Приложение 8. Съответствие между клавишите и кодовете, генерирани от тях

Таблица П8.1

Кодове, генерирани от клавишите на клавиатура с микрокомпютърно подреждане при работа със седембитов код

Клавиш	Нормално		С клавиш SHIFT		С клавиш CONTROL		С клавиши SHIFT и CTRL	
	Код	Символ	Код	Символ	Код	Символ	Код	Символ
1	2	3	4	5	6	7	8	9
ESCAPE	9B	ESC	9B	ESC	9B	ESC	9B	ESC
1 !	B1	1	A1	!	B1	1	A1	!
2 "	B2	2	A2	"	B2	2	A2	"
3 #	B3	3	A3	#	B3	3	A3	#
4 $	B4	4	A4	$	B4	4	A4	$
5 %	B5	5	A5	%	B5	5	A5	%
6 &	B6	6	A6	&	B6	6	A6	&
7 '	B7	7	A7	'	B7	7	A7	'
8 (	B8	8	A8	(	B8	8	A8	(
9)	B9	9	A9	)	B9	9	A9	)
0 _	B0	0	DF	_	B0	0	9F	US
: *	BA	:	AA	*	BA	:	AA	*
- =	AD	-	BD	=	AD	-	BD	=
^ Ч	DE	^	FE	Ч	9E	RS	9E	RS
TAB	89	HT	89	HT	89	HT	89	HT
Q Я	D1	Q	F1	Я	91	DC1	91	DC1
W В	D7	W	F7	В	97	ETB	97	ETB
E Е	C5	E	E5	Е	85	ENQ	85	ENQ
R Р	D2	R	F2	Р	92	DC2	92	DC2
T Т	D4	T	F4	Т	94	DC4	94	DC4
Y Ъ	D9	Y	F9	Ъ	99	EM	99	EM
U У	D5	U	F5	У	95	NAK	95	NAK
I И	C9	I	E9	И	89	HT	89	HT
O О	CF	O	EF	О	8F	SI	8F	SI
P П	D0	P	F0	П	90	DLE	90	DLE
@ Ю	C0	@	E0	Ю	80	NUL	80	NUL
\ Э	DC	\	FC	Э	9C	FS	9C	FS
RETURN	8D	CR	8D	CR	8D	CR	8D	CR

1		2	3	4	5	6	7	8	9
A	А	C1	A	E1	А	81	SOH	81	SOH
S	С	D3	S	F3	С	93	DC3	93	DC3
D	Д	C4	D	E4	Д	84	EOT	84	EOT
F	Ф	C6	F	E6	Ф	86	ACK	86	ACK
G	Г	C7	G	E7	Г	87	BEL	87	BEL
H	Х	C8	H	E8	Х	88	BS	88	BS
J	Й	CA	J	EA	Й	8A	LF	8A	LF
K	К	CB	K	EB	К	8B	VT	8B	VT
L	Л	CC	L	EC	Л	8C	FF	8C	FF
;	+	BB	;	AB	+	BB	;	AB	+
[	Ш	DB	[	FB	Ш	9B	ESC	9B	ESC
]	Щ	DD	]	FD	Щ	9D	GS	9D	GS
Z	З	DA	Z	FA	З	9A	SUB	9A	SUB
X	Ь	D8	X	F8	Ь	98	CAN	98	CAN
C	Ц	C3	C	E3	Ц	83	ETX	83	ETX
V	Ж	D6	V	F6	Ж	96	SYN	96	SYN
B	Б	C2	B	E2	Б	82	STX	82	STX
N	Н	CE	N	EE	Н	8E	SO	8E	SO
M	М	CD	M	ED	М	8D	CR	8D	CR
,	<	AC	,	BC	<	AC	,	BC	<
.	>	AE	.	BE	>	AE	.	BE	>
/	?	AF	/	BF	?	AF	/	BF	?
↑		8B	VT	8B	VT	8B	VT	8B	VT
DELETE		FF	DEL	FF	DEL	FF	DEL	FF	DEL
ИНТЕРВАЛ		A0	SP	A0	SP	A0	SP	A0	SP
←		88	BS	88	BS	88	BS	88	BS
→		95	NAK	95	NAK	95	NAK	95	NAK
↓		8A	LF	8A	LF	8A	LF	8A	LF

Кодове, генерирани от клавишите на клавиатура с микрокомпютърно подреждане при работа с осембитов код

	Латиница						Кирилица					
	Нормално	С клавиш SHIFT	С клавиш CTRL	С клавиши SHIFT и CTRL			Нормално	С клавиш SHIFT	С клавиш CTRL	С клавиши SHIFT и CTRL		
Клавиш	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ESCAPE	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC
1 !	B1	1	A1	!	B1	1	A1	!	B1	1	A1	!	B1	1	A1	!
2 "	B2	2	A2	"	B2	2	A2	"	B2	2	A2	"	B2	2	A2	"
3 #	B3	3	A3	#	B3	3	A3	#	B3	3	A3	#	B3	3	A3	#
4 $	B4	4	A4	$	B4	4	A4	$	B4	4	A4	$	B4	4	A4	$
5 %	B5	5	A5	%	B5	5	A5	%	B5	5	A5	%	B5	5	A5	%
6 &	B6	6	A6	&	B6	6	A6	&	B6	6	A6	&	B6	6	A6	&
7 '	B7	7	A7	'	B7	7	A7	'	B7	7	A7	'	B7	7	A7	'
8 (	B8	8	A8	(	B8	8	A8	(	B8	8	A8	(	B8	8	A8	(
9)	B9	9	A9	)	B9	9	A9	)	B9	9	A9	)	B9	9	A9	)
0 _	B0	0	DF	_	B0	0	9F	US	B0	0	DF	_	B0	0	9F	US
: *	BA	:	AA	*	BA	:	AA	*	BA	:	AA	*	BA	:	AA	*
- =	AD	-	BD	=	AD	-	BD	=	AD	-	BD	=	AD	-	BD	=
^ Ч	5E	~	DE	^	9E	RS	9E	RS	7E	ч	FE	Ч	9E	RS	9E	RS
TAB	89	HT	89	HT	89	HT	89	HT	89	HT	89	HT	89	HT	89	HT
Q Я	51	q	D1	Q	91	DC1	91	DC1	71	я	F1	Я	91	DC1	91	DC1
W В	57	w	D7	W	97	ETB	97	ETB	77	в	F7	В	97	ETB	97	ETB
E Е	45	e	C5	E	85	ENQ	85	ENQ	65	е	E5	Е	85	ENQ	85	ENQ
R Р	52	r	D2	R	92	DC2	92	DC2	72	р	F2	Р	92	DC2	92	DC2
T Т	54	t	D4	T	94	DC4	94	DC4	74	т	F4	Т	94	DC4	94	DC4
Y Ъ	59	y	D9	Y	99	EM	99	EM	79	ъ	F9	Ъ	99	EM	99	EM
U У	55	u	D5	U	95	NAK	95	NAK	75	у	F5	У	95	NAK	95	NAK
I И	49	i	C9	I	89	HT	89	HT	69	и	E9	И	89	HT	89	HT
O О	4F	o	CF	O	8F	SI	8F	SI	6F	о	EF	О	8F	SI	8F	SI
P П	50	p	D0	P	90	DLE	90	DLE	70	п	F0	П	90	DLE	90	DLE
@Ю	40	'	C0	@	80	NUL	80	NUL	60	ю	E0	Ю	80	NUL	80	NUL
\ Э	5C	|	DC	\	9C	FS	9C	FS	7C	э	FC	Э	9C	FS	9C	FS
RETURN	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR
A А	41	a	C1	A	81	SOH	81	SOH	61	а	E1	А	81	SOH	80	SOH

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
S С	53	s	D3	S	93	DC3	93	DC3	73	с	F3	С	93	DC3	93	DC3
D Д	44	d	C4	D	84	EOT	84	EOT	64	g	E4	Д	84	EOT	84	EOT
F Ф	46	f	C6	F	86	ACK	86	ACK	66	ф	E6	Ф	86	ACK	86	ACK
G Г	47	g	C7	G	87	BEL	87	BEL	67	г	E7	Г	87	BEL	87	BEL
H Х	48	h	C8	H	88	BS	88	BS	68	х	E8	Х	88	BS	88	BS
J Й	4A	j	CA	J	8A	LF	8A	LF	6A	й	EA	Й	8A	LF	8A	LF
K К	4B	k	CB	K	8B	VT	8B	VT	6B	k	EB	К	8B	VT	8B	VT
L Л	4C	l	CC	L	8C	FF	8C	FF	6C	л	EC	Л	8C	FF	8C	FF
; +	BB	;	AB	+	BB	;	AB	+	BB	;	AB	+	BB	;	AB	+
[Ш	5B	{	DB	[	9B	ESC	9B	ESC	7B	ш	FB	Ш	9B	ESC	9B	ESC
] Щ	5D	}	DD	]	9D	GS	9D	GS	7D	щ	FD	Щ	9D	GS	9D	GS
Z З	5A	z	DA	Z	9A	SUB	9A	SUB	7A	з	FA	З	9A	SUB	9A	SUB
X Ь	58	x	D8	X	98	CAN	98	CAN	78	ь	F8	Ь	98	CAN	98	CAN
C Ц	43	c	C3	C	83	ETX	83	ETX	63	ц	E3	Ц	83	ETX	83	ETX
V Ж	56	v	D6	V	96	SYN	96	SYN	76	ж	F6	Ж	96	SYN	96	SYN
B Б	42	b	C2	B	82	STX	82	STX	62	б	E2	Б	82	STX	82	STX
N Н	4E	n	CE	N	8E	SO	8E	SO	6E	н	EE	Н	8E	SO	8E	SO
M М	4D	m	CD	M	8D	CR	8D	CR	6D	м	ED	М	8D	CR	8D	CR
, <	AC	,	BC	<	AC	,	BC	<	AC	,	BC	<	AC	,	BC	<
. >	AE	.	BE	>	AE	.	BE	>	AE	.	BE	>	AE	.	BE	>
/ ?	AF	/	BF	?	AF	/	BF	?	AF	/	BF	?	AF	/	BF	?
↑	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT
DELETE	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL
ИНТЕР-ВАЛ	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP
←	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS
→	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK
↓	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF

Кодове, генерирани от клавишите на клавиатура с подреждане по БДС при работа със седембитов код

	Латиница						Кирилица					
	Нормално	С клавиш SHIFT	С клавиш CTRL	С клавиши SHIFT и CTRL			Нормално	С клавиш SHIFT	С клавиш CTRL	С клавиши SHIFT и CTRL		
Клавиш	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ESCAPE	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC
1 !	B1	1	A1	!	B1	1	A1	!	B1	1	A1	!	B1	1	A1	!
2 @ '?	B2	2	C0	@	B2	2	80	NUL	B2	2	AF	?	B2	2	80	NUL
3 # '+	B3	3	A3	#	B3	3	A3	#	B3	3	AB	+	B3	3	A3	#
4 $ '*	B4	4	A4	$	B4	4	A4	$	B4	4	A2	*	B4	4	A4	$
5 %	B5	5	A5	%	B5	5	A5	%	B5	5	A5	%	B5	5	A5	%
6 ^ '=	B6	6	DE	^	B6	6	9E	RS	B6	6	BD	=	B6	6	9E	RS
7 & ':	B7	7	A6	&	B7	7	A6	&	B7	7	BA	:	B7	7	A6	&
8 * '/	B8	8	AA	*	B8	8	AA	*	B8	8	AF	/	B8	8	AA	*
9 ('_	B9	9	A8	(	B9	9	A8	(	B9	9	DF	_	B9	9	A8	(
0) 'N	B0	0	A9	)	B0	0	A9	)	B0	0	CE	N	B0	0	A9	)
- _ 'I = + '.	AD	-	DF	_	AD	-	9F	US	AD	-	C9	I	AD	-	9F	US
V	BD	=	AB	+	BD	=	AB	+	AE	.	D6	V	BD	=	AB	+
\ '( )	DC	\	DC	\	9C	FS	9C	FS	A8	(	A9	)	9C	FS	9C	FS
TAB	89	HT	89	HT	89	HT	89	HT	89	HT	89	HT	89	HT	89	HT
Q ,	D1	Q	D1	Q	91	DC1	91	DC1	AC	,	AC	,	91	DC1	91	DC1
W У	D7	W	D7	W	97	ETB	97	ETB	F5	У	F5	У	97	ETB	97	ETB
E Е	C5	E	C5	E	85	ENQ	85	ENQ	E5	Е	E5	Е	85	ENQ	85	ENQ
R И	D2	R	D2	R	92	DC2	92	DC2	E9	И	E9	И	92	DC2	92	DC2
T Ш	D4	T	D4	T	94	DC4	94	DC4	FB	Ш	FB	Ш	94	DC4	94	DC4
Y Щ	D9	Y	D9	Y	99	EM	99	EM	FD	Щ	FD	Щ	99	EM	99	EM
U К	D5	U	D5	U	95	NAK	95	NAK	EB	К	EB	К	95	NAK	95	NAK
I С	C9	I	C9	I	89	HT	89	HT	E3	С	E3	С	89	HT	89	HT
O Д	CF	O	CF	O	8F	SI	8F	SI	E4	Д	E4	Д	8F	SI	8F	SI
P З	D0	P	D0	P	90	DLE	90	DLE	FA	З	FA	З	90	DLE	90	DLE
[Ц	DB	[	DB	[	9B	ESC	9B	ESC	E3	Ц	E3	Ц	9B	ESC	9B	ESC
] ; *	DD	]	DD	]	9D	GS	9D	GS	BB	;	AA	*	9D	GS	9D	GS

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
DELETE	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL
A Ь	C1	A	C1	A	81	SOH	81	SOH	F8	Ь	F8	Ь	81	SOH	81	SOH
S Я	D3	S	D3	S	93	DC3	93	DC3	F1	Я	F1	Я	93	DC3	93	DC3
D A	C4	D	C4	D	84	EOT	84	EOT	E1	A	E1	A	84	EOT	84	EOT
F O	C6	F	C6	F	86	ACK	86	ACK	EF	O	EF	O	86	ACK	86	ACK
G Ж	C7	G	C7	G	87	BEL	87	BEL	F6	Ж	F6	Ж	87	BEL	87	BEL
H Г	C8	H	C8	H	88	BS	88	BS	E7	Г	E7	Г	88	BS	88	BS
J T	CA	J	CA	J	8A	LF	8A	LF	F4	T	F4	T	8A	LF	8A	LF
K H	CB	K	CB	K	8B	VT	8B	VT	EE	H	EE	H	8B	VT	8B	VT
L B	CC	L	CC	L	8C	FF	8C	FF	F7	B	F7	B	8C	FF	8C	FF
; : 'М	BB	;	BA	:	BB	;	BA	:	ED	М	ED	М	BB	;	BA	:
' * Ч	A7	'	A2	*	A7	'	A2	*	FE	Ч	FE	Ч	A7	'	A2	*
RETURN	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR
Z Ю	DA	Z	DA	Z	9A	SUB	9A	SUB	E0	Ю	E0	Ю	9A	SUB	9A	SUB
X Й	D8	X	D8	X	98	CAN	98	CAN	EA	Й	EA	Й	98	CAN	98	CAN
C Ъ	C3	C	C3	C	83	ETX	83	ETX	F9	Ъ	F9	Ъ	83	ETX	83	ETX
V Э	D6	V	D6	V	96	SYN	96	SYN	FC	Э	FC	Э	96	SYN	96	SYN
B Ф	C2	B	C2	B	82	STX	82	STX	E6	Ф	E6	Ф	82	STX	82	STX
N X	CE	N	CE	N	8E	SO	8E	SO	E8	X	E8	X	8E	SO	8E	SO
M П	CD	M	CD	M	8D	CR	8D	CR	F0	П	F0	П	8D	CR	8D	CR
, < 'Р	AC	,	BC	<	AC	,	BC	<	F2	Р	F2	Р	AC	,	BC	<
. > 'Л	AE	.	BE	>	AE	.	BE	>	EC	Л	EC	Л	AE	.	BE	>
/ ? 'Б	AF	/	BF	?	AF	/	BF	?	AF	Б	BF	Б	AF	/	BF	?
ИНТЕР-																
ВАЛ	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP
←	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS
→	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK
↑	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT
↓	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF

Кодове, генерирани от клавишите на клавиатура с подреждане по БДС при работа с осембитов код

	Латиница					Кирилица				
	Нормално	С клавиш SHIFT	С клавиш CTRL	С клавиши SHIFT и CTRL		Нормално	С клавиш SHIFT	С клавиш CTRL	С клавиши SHIFT и CTRL	
Клавиш	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
1	2	3	4	5	6	7	8	9	10	11
ESCAPE	9B	ESC	9B	ESC	9B	ESC	9B	ESC	9B	ESC
1 !	B1	1	A1	!	B1	1	A1	!	B1	1
2 @ " ?	B2	2	C0	@	B2	2	80	NUL	B2	2
3 # ' +	B3	3	A3	₽	B3	3	A3	₽	B3	3
4 $ ' *	B4	4	A4	$	B4	4	A4	$	B4	4
5 %	B5	5	A5	%	B5	5	A5	%	B5	5
6 ^ ' =	B6	6	DE	^	B6	6	9E	RS	B6	6
7 & :	B7	7	A6	&	B7	7	A6	&	B7	7
8 * /	B8	8	AA	*	B8	8	AA	*	B8	8
9 (_	B9	9	A8	(	B9	9	A8	(	B9	9
0) N	B0	0	A9	)	B0	0	A9	)	B0	0
- _ | = +	AD	-	DF	_	AD	-	9F	US	AD	-
' : V	BD	=	AB	+	BD	=	AB	+	AE	.
\ ' ( )	5C	|	DC	\	9C	FS	9C	FS	A8	(
TAB	89	HT	89	HT	89	HT	89	HT	89	HT
Q ,	51	q	D1	Q	91	DC1	91	DC1	AC	,
W У	57	w	D7	W	97	ETB	97	ETB	75	у
E Е	45	w	C5	E	85	ENQ	85	ENQ	65	е
R И	52	r	D2	R	92	DC2	92	DC2	69	и
T Ш	54	t	D4	T	94	DC4	94	DC4	7B	ш
Y Щ	59	y	D9	Y	99	EM	99	EM	7D	щ
U К	55	u	D5	U	95	NAK	95	NAK	6B	к
I С	49	i	C9	I	89	HT	89	HT	63	с
O Д	4F	o	CF	O	8F	SI	8F	SI	64	д
P З	50	p	D0	P	90	DLE	90	DLE	7A	з
[Ц	5B	{	DB	[	9B	ESC	9B	ESC	63	ц
] ' ; *	5D	}	DD	]	9D	GS	9D	GS	BB	;
DELETE	fF	DEL	FF	DEL	FF	DEL	FF	DEL	FF	DEL

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A ь	41	a	C1	A	81	SOH	81	SOH	78	ь	F8	Ь	81	SOH	81	SOH
S Я	53	s	D3	S	93	DC3	93	DC3	71	я	F1	Я	93	DC3	93	DC3
D А	44	d	C4	D	84	EOT	84	EOT	61	а	E1	А	84	EOT	84	EOT
F О	46	f	C6	F	86	ACK	86	ACK	6F	о	EF	О	86	ACK	86	ACK
G Ж	47	g	C7	G	87	BEL	87	BEL	76	ж	F6	Ж	87	BEL	87	BEL
H Г	48	h	C8	H	88	BS	88	BS	67	г	E7	Г	88	BS	88	BS
J Т	4A	j	CA	J	8A	LF	8A	LF	74	т	F4	Т	8A	LF	8A	LF
K Н	4B	k	CB	K	8B	VT	8B	VT	6E	н	EE	Н	8B	VT	8B	VT
L В	4C	l	CC	L	8C	FF	8C	FF	77	в	F7	В	8C	FF	8C	FF
; : 'М	BB	;	BA	:	BB	;	BA	:	6D	м	ED	М	BB	;	BA	:
' " Ч	A7	'	A2	"	A7	'	A2	"	7E	ч	FE	Ч	A7	'	A2	"
RETURN	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR	8D	CR
Z Ю	5A	z	DA	Z	9A	SUB	9A	SUB	60	ю	E0	Ю	9A	SUB	9A	SUB
X Й	58	x	D8	X	98	CAN	98	CAN	6A	й	EA	Й	98	CAN	98	CAN
C Ъ	43	c	C3	C	83	ETX	83	ETX	79	ъ	F9	Ъ	83	ETX	83	ETX
V Э	56	v	D6	V	96	SYN	96	SYN	7C	э	FC	Э	96	SYN	96	SYN
B Ф	42	b	C2	B	82	STX	82	STX	66	ф	E6	Ф	82	STX	82	STX
N Х	4E	n	CE	N	8E	SO	8E	SO	68	х	E8	Х	8E	SO	8E	SO
M П	4D	m	CD	M	8D	CR	8D	CR	70	п	F0	П	8D	CR	8D	CR
, <'Р	AC	,	BC	<	AC	,	BC	<	72	р	F2	Р	AC	,	BC	<
. >'Л	AE	.	BE	>	AE	.	BE	>	6C	л	EC	Л	AE	.	BE	>
/ ? 'Б	AF	/	BF	?	AF	/	BF	?	AF	б	BF	Б	AF	/	BF	?
ИНТЕР-ВАЛ	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP	A0	SP
←	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS	88	BS
→	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK	95	NAK
↑	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT	8B	VT
↓	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF	8A	LF

Съдържание

Въведение 3

Глава 1. Основни сведения 5
1.1. Технически характеристики 5
1.2. Основна конфигурация 7
1.3. Включване на допълнителни модули 11
1.4. Апаратна и програмна съвместимост на Правец–8А с Правец–82 и Правец–8М 14

Глава 2. Вградени входно–изходни устройства 16
2.1. Клавиатура 16
2.2. Режими на видеоизображение 22
2.2.1. Изобразяване на текст 25
2.2.2. Изобразяване на графика 30
2.3. Допълнителни входно–изходни устройства 41

Глава 3. Организация на паметта 45
3.1. Системна памет 45
3.2. Организация на областта $D000 — $FFFF 47
3.3. Област на входно–изходните устройства 50
3.4. Допълнителна памет 52

Глава 4. Работа с програмата МОНИТОР 56
4.1. Влизане в програмата МОНИТОР 56
4.2. Формат на командите на програмата МОНИТОР . . . 57
4.3. Извеждане на съдържанието на паметта 58
4.4. Промяна на съдържанието на паметта 59
4.5. Копиране на съдържанието на област от паметта . . 60
4.6. Сравняване на съдържанието на две области от паметта 63
4.7. Извеждане и промяна на съдържанието на регистрите на микропроцесора 63
4.8. Изпълнение на програма на машинен език 64
4.9. Извикване на интерпретатора на БЕЙСИК без инициализация 65
4.10. Извикване на интерпретатора на БЕЙСИК с инициализация 65
4.11. Шестнайсетично събиране и изваждане 65
4.12. Избор на входно устройство 66
4.13. Избор на изходно устройство 66

4.14. Потребителска команда 66
4.15. Преобразуване на шестнайсетични числа в десетични 67
4.16. Преобразуване на десетични числа в шестнайсетични 67
4.17. Дезасемблиране на съдържанието на паметта 68
4.18. МИНИАСЕМБЛЕР 71
4.19. Търсене на низ 73
4.20. Управление на формата на текстовото изображение 74

Глава 5. Подпрограми на програмата МОНИТОР 75
5.1. Подпрограми за запазване и възстановяване на съдържанието на регистрите на микропроцесора 76
5.2. Стандартни входно–изходни вектори 77
5.3. Изходни подпрограми 79
5.4. Входни подпрограми 88
5.5. Подпрограми за обслужване на екрана в текстов режим 95
5.6. Подпрограми за поддържане на графика с малка разделителна способност (режим ГР40) 101
5.7. Подпрограми за работа с допълнителната памет . . 105
5.8. Други подпрограми на програмата МОНИТОР 109
5.9. Обслужване на прекъсванията в Правец–8А 116
5.9.1. Прекъсване тип RESET 117
5.9.2. Прекъсвания тип NMI 121
5.9.3. Прекъсване тип IRQ/BRK 121

Глава 6. Резидентен интерпретатор на БЕЙСИК 125
6.1. Особености на резидентния интерпретатор на БЕЙСИК в Правец–8А 125
6.2. Оператори и функции на интерпретатора на БЕЙСИК 129

Глава 7. Схемна реализация 166
7.1. Функционална схема 166
7.1.1. Схеми за генериране на тактови и синхронизиращи импулси 168
7.1.2. Микропроцесор 168
7.1.3. Адресни буфери 168
7.1.4. Буфери за данни 169
7.1.5. Схема за управление на паметта 169
7.1.6. Управление на вградените входно–изходни устройства 171
7.1.7. Оперативна памет 174
7.1.8. Постоянна памет 174
7.2. Използване на адресното пространство от входно–изходните устройства 174

7.3. Мостчета за конфигуриране 176
7.4. Съединители на системната платка на персоналния компютър Правец–8А 177
7.4.1. Допълнителен съединител X0 177
7.4.2. Съединители за включване на допълнителни модули X1 – X7 180
7.4.3. Свързване на видеомонитор 182
7.4.4. Съединител за свързване на ръчки за управление на игри 183
7.4.5. Свързване на високоговорителя 184
7.4.6. Свързване на клавиатурата 184
7.4.7. Свързване на захранващия блок 185

Приложение 1. Микропроцесор СМ630 187
П1.1. Програмен модел 188
П1.2. Методи на адресиране 189
П1.3. Набор от инструкции 200

Приложение 2. Използване на паметта от програмата МОНИТОР 206

Приложение 3. Подпрограми на програмата МОНИТОР 219

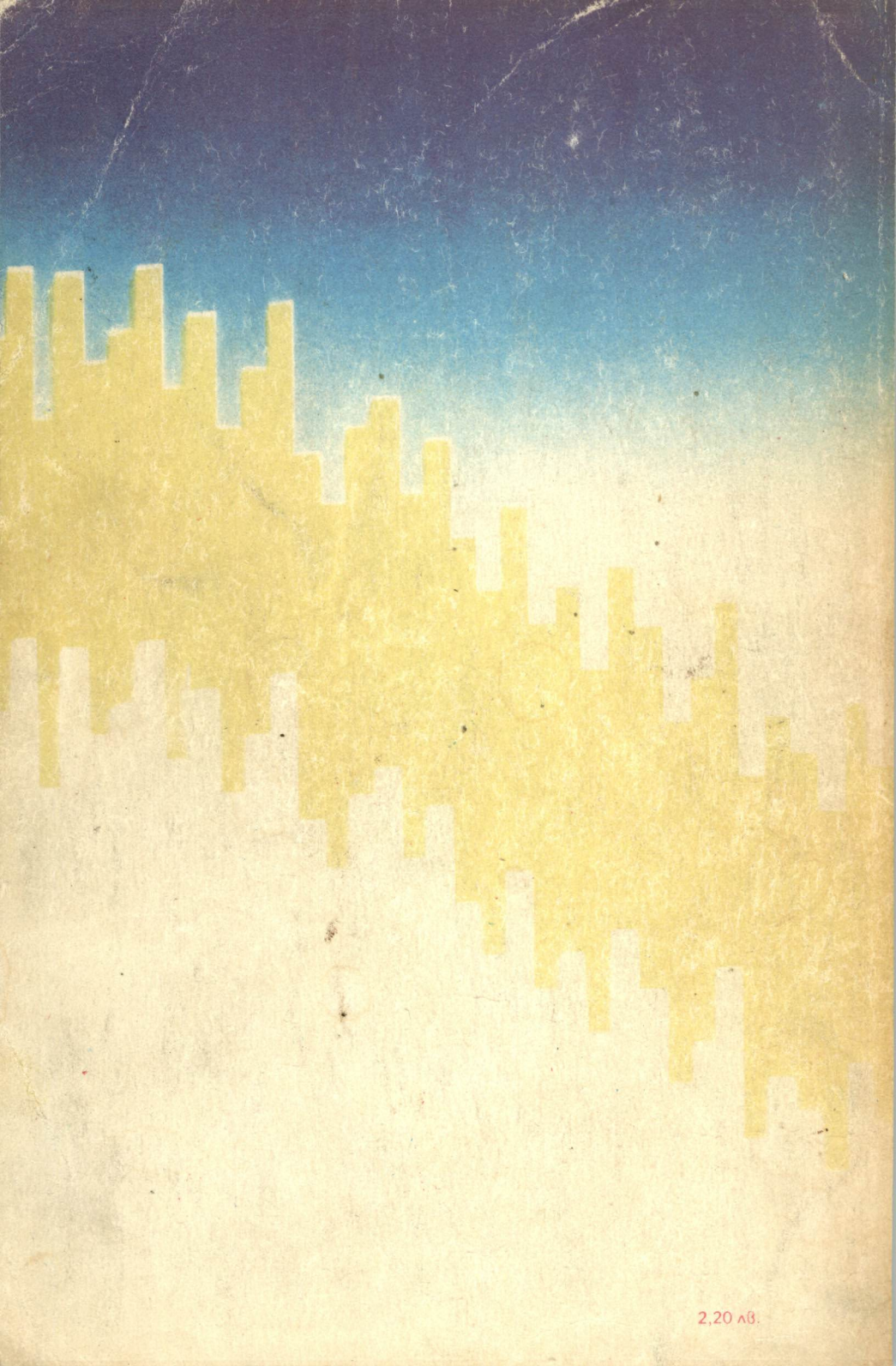
Приложение 4. Вградени входно–изходни устройства и програмноуправляеми ключове 226

Приложение 5. Запазени думи в БЕЙСИК 230

Приложение 6. Съобщения за грешки в интерпретатора на БЕЙСИК 232

Приложение 7. Таблици на фигури в РАЗШИРЕН БЕЙСИК 234

Приложение 8. Съответствие между клавишите и кодовете, генерирани от тях 238



2,20 лв.