DOBOS JÁNOS

SZÁMÍTÓGÉPES GRAFIKA C 16-ON ÉS C PLUS/4-EN

ORSZÁGOS PEDAGÓGIAI INTÉZET

VORKER®

DOBOS JÁNOS

Számítógépes grafika C 16-on és C Plus/4-en

Főszerkesztő:
dr. Szűcs Barna

Szerkesztő:
Borbola István

A kiadvány szerzője:
Dobos János

Bírálta:
dr. Bohus Mihály

ISBN 963 682 160 7 összkiadás
ISBN 963 682 170 4

Kiadja az Országos Pedagógiai Intézet Számítástechnikai Programirodája
Felelős kiadó: Szabolcsi Miklós

TARTALOMJEGYZÉK

A SOROZAT TAGJAI

Az Országos Pedagógiai Intézet Számítástechnikai Programirodájának gondozásában 1987/88-ban 12 számítástechnikai témakörhöz készült oktatási anyagok C 16-os és C Plus/4-es Commodore házi számítógépekre:

1. Algormitmusok, játékok:

A kiadvány tömören, de közérthetően, szemléletes példákon keresztül világít rá a probléma—algoritmus—program kapcsolatra, elvi és gyakorlati tanácsokat ad a jól felépített programok készítéséhez. A hozzá tartozó kazetta tizenkét, önmagában is alkalmazható, számítástechnikai és matematikai tanulságokkal teli programot tartalmaz. A programok jórészt listázhatók, így példát mutatnak egy színvonalas programozási stílus kialakítására, a gyakorlottabb felhasználóknak lehetőséget adnak arra, hogy más programok készítésénél is hasznosítsák az itt bemutatott programozási fogásokat. *Ajánlott:* ált. isk. 7., 8. osztályosainak és középiskolásoknak.

2. A számítógép és környezete:

Mindazoknak készült, akik szeretnék összekötni saját számítógépüket a külvilággal. 8 db különböző interfész és periféria tervezése kapcsán ismerteti meg az olvasót az interfész tervezés és illesztés fogásaival. Útmutató és 8 kapcsolási rajz, kész nyomtatott ármköri tervrajz. *Ajánlott:* ált. isk. és középiskolai tanároknak, számítástechnikai amatőröknek, 8. osztályos és középiskolai tanulóknak, tanárjelölt főiskolai és egyetemi hallgatóknak.

3. Programmodulok:

A felhasználó 55 db szubrutint, programmodult, ill. önálló felhasználói programot kap. Ezek egy része gépi kódú rutin, melyet az átlagos felhasználó nem tud megírni, de beépíthető munkájába. A felhasználói programok ötletet adhatnak hasonló programok megírásához. Az egyes rutinok, programok alkalmazási lehetőségeinek, működésének leírását útmutató tartalmazza. *Ajánlott:* ált. iskolai tanároknak, tanulóknak, kezdő amatőr programozóknak.

4. Adatkezelés számítógéppel C—16-ra és C Plus/4-re:

A témakör feldolgozását segítő útmutató az adatkezelés, adatfeldolgozás alapfogalmaival és folyamatával foglalkozik. Megismertet az adatfeldolgozó rendszerek fejlesztésével és a rendszerek alkalmazásbavételével. A feldolgozást 5 program és 2 minta-adatállomány segíti. *Ajánlott:* ált. és középiskolai tanároknak, fakultációs tananyagként, továbbképzési anyagként.

5. Szövegfeldolgozás számítógéppel:

A témakör feldolgozásához a felhasználó megkapja a Nemzetközi ABC szövegszerkesztő programot, amely magyar, cirill, német, lengyel, görög karakterkészletet és különleges matematikai jeleket (gyökjel, integráljel, alsó és felső index stb.) is tartalmaz. A szövegek és különleges karakterek nyomtathatók — akár fűzött kézírást imitálva is. A kézírásos nyomtatás főleg alsó tagozatban számíthat sikerre. A szövegszerkesztő minden olyan szolgáltatást tud, amely a tudományos munkához, vagy a tanári munkához kell. A feldolgozást útmutató és bemutató programok is segítik. *Ajánlott:* tanároknak, kutatóknak, íróknak, adminisztrátoroknak, ill. fakultációk anyagaként.

6. Számítógépes grafika C 16-on és C Plus/4-en:

Az útmutató és a mintaprogramok a felhasználói kézikönyvből meg nem tanulható különleges grafikus lehetőségekkel ismertetik meg az olvasót. A többszínű háttérszín üzemmód mellett a bittérképes színes grafikák elkészítésének fortélyait is megtanulhatja a felhasználó. Az útmutató és a mintaprogramok áttanulmányozása után az olvasó saját maga is képes lesz a játékprogramokból ismert színes és mozgó figurák „szellemecskék" programozására. *Ajánlott:* ált. isk. 7., 8. osztályosainak, középiskolásoknak és a grafika programozását most kezdő programozóknak.

7. Hangkeltés C 16-os és C Plus/4-es számítógéppel:

Az anyaghoz az útmutató mellett bemutató programok is tartoznak. Ezek áttanulmányozása alkalmassá teszi a felhasználót egy és többszólamú dallamok programozására, az interrupt alatti programozási lehetőségek kihasználására. A programok és dallamok az ált. iskolai ének-zene oktatáshoz igazodnak, így felhasználhatók számítógépes motivációra. *Ajánlott:* ált. isk. 7., 8. osztályosoknak, középiskolásoknak, valamint ált. iskolai énektanároknak, kezdő programozóknak.

8. Számítógépes fogások, trükkök C 16-ra és C Plus/4-re:

Az útmutató és a hozzá tartozó 16 program azok számára mutatja be a C 16-on és a C Plus/4-en alkalmazható fogásokat, akik a BASIC programozás alapjain túljutottak. A közölt rendszerváltozók, memóriacímek és gépi kódú programok megismertetik az olvasót az alkalmazások különleges lehetőségeivel. *Ajánlott:* kezdő progcamozóknak, általános és középiskolásoknak, tanároknak.

9. Számítógép és videó:

Az anyaghoz útmutató, videófilm és bemutató programok tartoznak. A videófilm a hozzá tartozó számítógépes programokkal egy megvalósítható rendszert és egy példát mutat be a képmagnó és a házi számítógép összekapcsolására, az oktatásban történő alkalmazására. Az anyag az elveken túl egy interfész elkészítésével is megismerteti az olvasót. Elsősorban tanároknak, oktatástechnikai szakembereknek ajánlott!

10. Számítógéppel vezérelt mérések:

Az útmutató megismerteti az olvasót a legfontosabb méréstani fogalmakkal, a mérések tervezésének, elvégzésének, értékelésének elméleti és műszaki-, méréstani alapjaival. A mérőberendezések és mérési összeállítások jelátvivő funkcionális egységeként a Tudományszervezési és Informatikai Intézetnél kapható TechnoMIR moduláris interfészrendszer berendezéseit mutatja be és használja fel az anyag. A témakörhöz tartozó 21 db professzionális program egyrészt az oktatómunkában és a bemutató kísérletekben alkalmazható számítógép-vezérlésű intelligens műszert szimulál (pl. tároló oszcilloszkópot, 8 csatornás logikai analizátort, digitális multimétert stb.) másrészt az iskolai kísérletekhez szükséges mérési összeállításokat vezérli. *Ajánlott:* általános és középiskolai tanároknak, speciális szakképző intézeteknek, pedagógusképző intézeteknek bemutatási és oktatási célra, pedagógiai intézeteknek a továbbképzéshez.

11. Számítógépes irányítástechnika:

Az anyaghoz útmutató, az útmutatóban ismertetett berendezésekhez 11 db mintaprogram tartozik. Ezek áttanulmányozása megismerteti az olvasót a számítógépes vezérlések és szabályozások alapfogalmaival, a mechatronikus modelleket működtető berendezések (kapcsolómodulok, interfészek) elkészítésével. A mintaprogramok elemzése példát mutat az irányítási algoritmusok programozására. Az irányító rendszerek jelátvivő funkcionális egységeként a Tudományszervezési és Informatikai Intézetnél kapható TechnoMIR interfészrendszer elemeit, érzékelő-, erősítő berendezéseknek és beavatkozó szerveknek, berendezéseknek pedig saját készítésű, a felhasználó által is reprodukálható eszközöket használ az anyag. Ezek leírásait és elkészítési módját is tartalmazza. *Ajánlott:* az általános és középiskolai technika tárgy oktatásához, fakultációkhoz, szakköri feldolgozáshoz.

12. Számítógépek összekapcsolása, helyi oktatóhálózatok:

Az anyag egy hazánkban eddig nyomtatásban még meg nem jelent területtel, a Commodore házi számítógépek (és a VIDEOTON TV COMPUTER) összekapcsolásának gyakorlati megvalósításával és e számítógépekből összeállított helyi oktatóhálózatok programozásával foglalkozik. Megismerteti a felhasználóval a számító-

gépek közötti kapcsolat hardver és szoftver eszközeit, a kapcsolatteremtés és az adatátvitel alapfogalmait. Megépített és reprodukálható kapcsolásokkal és kész felhasználói programokkal segíti, hogy az anyagot feldolgozó olvasó saját maga is összekapcsolhasson számítógépeket. A megszerzett ismeretek segítségével a minimális elektronikai ismerettel és készüléképítési gyakorlattal rendelkező felhasználó is meg tud valósítani számítógépek közötti adatátvitelt. Az útmutató második része az oktatásban bevált és sikerrel alkalmazott, a szegedi VORKER Kisszövetkezet által kifejlesztett TC-NET oktatóhálózatokkal, azok programozási fogásaival foglalkozik. Az anyag 6 db kapcsolást, 1 db nyomtatott áramköri rajzot és 12 db professzionális felhasználói programot tartalmaz. *Ajánlott:* általános és középiskolai tanároknak, 3., 4. éves középiskolai tanulóknak, számítástechnikai amatőröknek, középfokú szakképző intézeteknek, tanárképző intézeteknek, pedagógiai intézeteknek, továbbképző intézeteknek stb.

Valamennyi itt felsorolt anyag (útmutató és program) valamint a TC-NET megrendelhető a szegedi VORKER Kisszövetkezettől.

Cím:
VORKER Kisszövetkezet
6701 SZEGED, Pf. 711.
Tel.: H 06-62-*26-144*
Telex: 82-688

A VORKER Kisszövetkezet vállalja — megrendelés alapján — a felsorolt témakörökben bemutatók, alap- és továbbképzések tartását, szervezését is.

ELŐSZÓ

Az olvasó a szegedi Tarjánvárosi IV. számú Általános Iskolában fejlesztett 12 pedagógiai program (tanári kézikönyv, szoftver és/vagy beültetési rajzok) egyikét tartja a kezében. A munkában több mint hatvanan vettek részt: általános és középiskolai tanárok, főiskolai, egyetemi oktatók, tudományos intézetek munkatársai.)

Az Országos Pedagógiai Intézet Számítástechnikai Programirodája és a szegedi iskola közös vállalkozása összefügg a számítástechnikai alapműveltség körülhatárolására irányuló erőfeszítésekkel. Kétségtelen, hogy a számítástechnikai-informatikai alapismeretek és az alapműveltség kapcsolatának feltárása elsősorban teoretikus jellegű munkát igényel. Az 1970-es években induló tartalmi korszerűsítés tapasztalatai azonban arra hívják fel a figyelmünket, hogy az iskolák nélkül, az iskolai hagyományok és a pedagógia öntörvényeinek mellőzésével nem hozhatók létre olyan dokumentációk, amelyeket a gyakorlat magáénak érez, s amelyekkel a pedagógusok többsége közösséget vállal.

Az alapműveltség témakörében már eddig is több ranulmány látott napvilágot. Viszonylag kevesebb szó esett azonban arról, hogy miként lehet pedagógiailag szervezett ismeretrendszerré formálni az itthon is és külföldön is már megfogalmazott elképzeléseket (amelyek között meglepően szoros összefüggés lelhető fel).

Az elképzelésekről szóló következő rövid ismertetés nem tekinthető állásfoglalásnak abban a kérdésben, hogy mi a jobb: ha a számítástechnikai-informatikai alapismeretekhez a diákok az *óratervben,* különféle tantárgyakban jutnak hozzá; *szakköri vagy fakultációs* programok keretében, esetleg *külön tantárgyként* biztosítják ezt számukra.

A fejlesztők munkáját a következő feltételezések irányították:

1. A számítástechnikai-informatikai ismeretek és az ez iránti érdeklődés nehezen tagolható az eddig megszokott módon, vagyis életkor és iskolatípus szerint. Az elkészült anyagoknak tehát egyaránt szolgálniok kell az általános iskolákat és a középfokú iskolákat.
2. Az iskolák fogadókészsége eltérő (pl. más-más a felszerelés, a tanárok felkészültsége, a szoftverellátottság stb.). Ez aligha teszi lehetővé az eddig megszokott, a mindenki számára kötelező tantervi előírásokat, sémákat. Az igényekhez tehát csak a többféle lehetőség egyidejű bemutatásával kerülhetünk közelebb.
3. A 12 pedagógiai program és ezek járulékai alapul szolgálhatnak egy majdan országos érvényű informatika tantárgy kidolgozásához. A témák ilyen rendszere a már ma megvalósítható iskolai tevékenységeket tükrözi. Az informatika több más

témájáról az iskolák jórésze számára ma még csupán leírás adható (pl. országos, nemzetközi hálózatok). A későbbiekben elképzelhető, hogy az iskolákban is lehetőség lesz az informatika teljes skálájának bemutatására.

4. A 12 rész között biztosíthatók „átjárások": a részek egymáshoz illeszthetők és kombinálhatók egymással.
5. A pedagógiai programokban megfogalmazott elképzelések érvényesítése érdekében az elinduláshoz szükséges a megfelelő segédletek elkészítése.
6. A pedagógiai programokhoz készült járulékok (beültetési rajzok, szoftverkészlet) egyik részének tantárgyi alkalmazásokra kell épülnie (pl. matematika, mérések), másik részük viszont az informatikának a mindennapi életben megjelenő gyakorlatát tükrözze (pl. adatkezelés szövegszerkesztés).

Reméljük, hogy munkánk segíteni fogja a pedagógiai gyakorlatot és ezáltal hozzájárul e téma elméleti kérdéseinek tisztázásához.

Örömmel és tisztelettel fogadjuk munkánk olvasójának, használójának észrevételeit.

Dr. Szűcs Barna

1. FINOMFELBONTÁSÚ GRAFIKA

1.1 Bevezetés

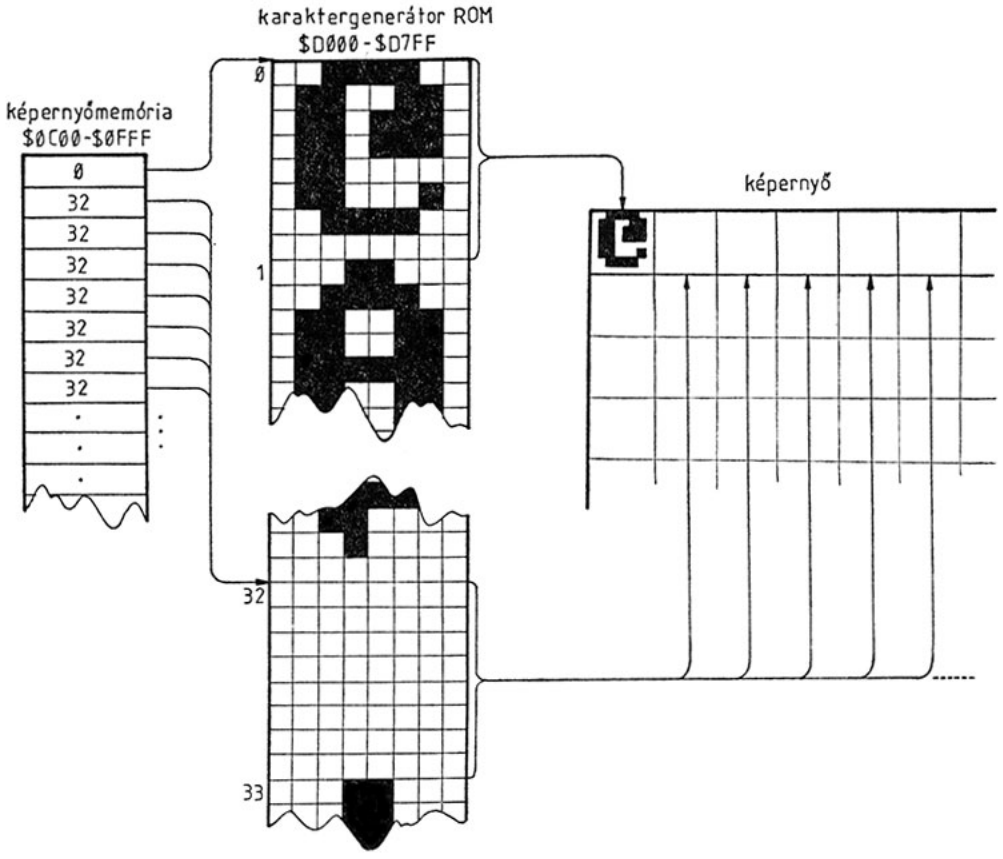
Mivel a BASIC nyelv megalkotásakor, 1959-ben még nemigen volt szó számítógépes grafikáról (legalábbis e nyelv szintjén), a szabvány BASIC szótár nem ismer grafikus utasításokat. Ezért minden számítógépet előállító gyár saját fejlesztésű grafikus utasításkészlettel egészíti ki gépe programnyelvét. Így van ez a Commodore 16 és Commodore Plus/4 számítógépek esetében is. Az itt használt utasításokat más gépek egyszerűen gépelési hibának tekintik.

Az említett két gép grafikus lehetőségeivel mindenképpen érdemes foglalkozni a fakultáció keretében. Mivel a gyerekek nagy része vizuális típus, ezért egyrészt jobban leköti a figyelmüket minden olyan feladat, aminek eredménye rögtön látható, másrészt a programozási ismeretek alapköveinek lerakásánál grafikus programokkal az eredmény sokkal „kézzelfoghatóbb" látvány alapján, mintha puszta számokkal kellene foglalkozniuk.
Ennek a fejezetnek a célja nemcsak a grafikus utasítások felsorolása, hanem részben a programozási munkához igyekszik segítséget adni, részben a számítógépes grafika **iskolai feldolgozásának módszereivel kíván foglalkozni.**

1.1.1. Grafikus üzemmódok

A grafikus üzemmódok tárgyalása előtt röviden a karakteres üzemmódról. A gép bekapcsolásakor ezzel találkozunk. Ilyenkor csak a billentyűzeten lévő jelek használhatók, nincs lehetőség grafikus utasítások használatára. A képernyő 25 sorra és 40 oszlopra, összesen tehát 25*40=1000 pozícióra osztott. E pozíciók mindegyike a memória egy-egy bájtjának felel meg, ezt az 1000 bájt hosszúságú memóriaterületet (3072—4071) nevezzük képernyőmemóriának. Ezek a bájtok természetesen nem tartalmazhatják egy-egy karakter képét. A karakterek alakja a memória egy meghatározott részén ($D000—$D7FF), az úgynevezett karaktergenerátor ROM-ban helyezkedik el. A képernyőmemória bájtjainak értékei mutatók, amelyek a karaktergenerátorból a megfelelő karaktert hívják. Ha például a képernyő bal felső sarkára egy „@" jelet szeretnénk írni, nincs más dolgunk, mint a választott pozíciónak megfelelő bájt (3072) értékéül adjuk a kívánt jel karaktergenerátorbeli sorszámát (0): POKE3072,0. A képernyőmemória többi bájtjának értéke — feltéve, hogy egyébként üres a képernyő — 32 lesz, ez ugyanis a „SPACE" (üres) karakter

sorszáma. Próbáljuk ki például ezt: a PRINT PEEK(3300) parancs eredményeként 32 adódik.



1.1. ábra
A karakteres üzemmód működési elve

Arról, hogy a megjelenített karakter milyen színű legyen, a színmemória megfelelő bájtja gondoskodik. A színmemória szintén 1000 bájt hosszúságú és a képernyőmemória bájtjaival kölcsönösen egyértelmű megfeleltetésben áll. A memória 2048-tól 3047-ig terjedő részét foglalja el.

A karakteres üzemmódról bővebbet a Karaktermód-lehetőségek című fejezetből tudhatunk meg.

Ahhoz, hogy karakteres képernyőn valamit megjelenítsünk, természetesen nem kell ilyen bonyolult utat bejárnunk, a PRINT, vagy még inkább a kényelmes CHAR utasítás könnyen kezelhetővé teszi a képernyőt. A CHAR utasítással ugyanis a képernyő bármely pozícióját „megcélozhatjuk", csak a kiszemelt hely oszlop, illetve sorszámát kell megadnunk. Például a CHAR1,17,12, „C Plus-4" parancs a képernyő közepére (17. oszlop, 12. sor) írja számítógépünk nevét. Egy hiányossága ennek az

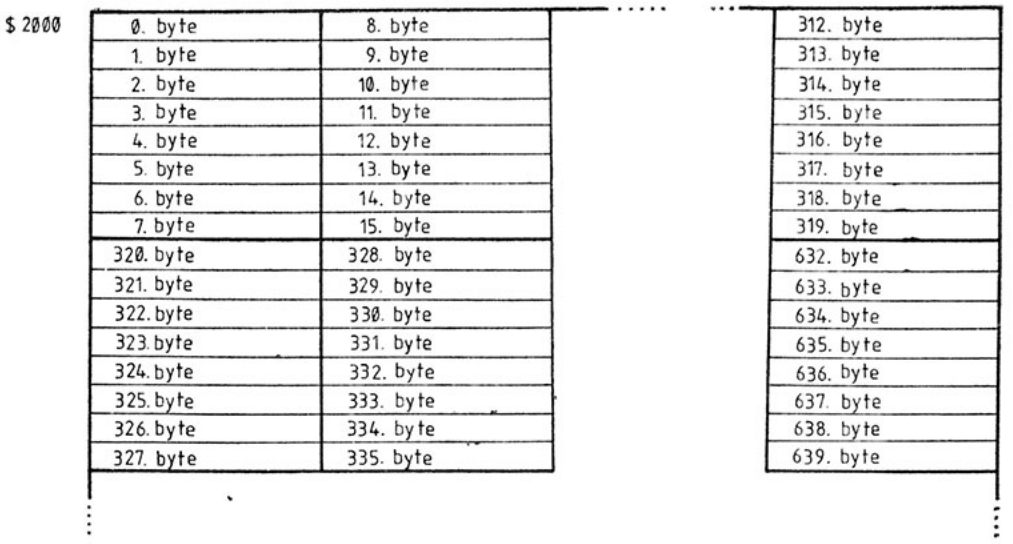
utasításnak, hogy csak sztringkifejezésekkel foglalkozik, így egy számítás eredményét nem írhatjuk ki vele közvetlenül. Mint minden akadály, természetesen ez is elhárítható: alakítsuk az eredményt rögtön sztringgé.

```
10 CHAR1,16,10,""
20 INPUT "SUGAR";R
30 T=R*R*π
40 CHAR1,16,12,"TERÜLET "+STR$(T)
```

Ebben a mintaprogramban a 10-es sor látszólag értelmetlen, ugyanis nem ír semmit a képernyőre. Nem is ez a célja, hanem az utána következő INPUT helyét határozza meg. A 40-es sor a STR$ függvénnyel a numerikus T értéket sztringgé alakítja. A karakteres képernyőn jelenthet gondot az úgynevezett idézőjeles üzemmód. Ilyenkor a vezérlőbillentyűk (kurzormozgatók, CLEAR/HOME, színbeállító billentyűk stb.) funkciójukat nem hajtják végre, ehelyett az idézőjelen belül vezérlőkarakterek jelennek meg egészen addig, míg az idézőjelet be nem zárjuk. Aki programozott már, azzal valószínűleg előfordult, hogy elátkozta ezt az egyébként kényelmes funkciót, mert nem tudott belőle kikecmeregni másként, csak RETURN gombbal kilépve a sorból és az egészet újrakezdve. Kínálkozik azonban egy egyszerűbb, barátságosabb módszer is: az Esc és 0 billentyűk egymás utáni lenyomása megszünteti az idézőjel üzemmódot.

Bár a most tárgyalt karakteres üzemmóddal jelentkezik a gép bekapcsoláskor, előfordulhat, hogy grafikus képernyőről szeretnénk visszajutni ide. Erre szolgál a GRAPHIC Ø utasítás. Emellett bármilyen hiba fordul is elő, a gép automatikusan erre az üzemmódra vált.

Finomfelbontású grafika. Ebben az üzemmódban a képernyő 320×200 pontra osztott. Ezt az teszi lehetővé, hogy a megjelenítendő pontok és a memóriabitek között kölcsönösen egyértelmű megfeleltetés áll fenn. Amikor a nagyfelbontású képernyőt nézzük, akkor tulajdonképpen a memória egy meghatározott területét látjuk. Az „1" állapotú bitek írásszínnel, a „Ø" állapotúak háttérszínnel jelennek meg. A finomfelbontású képernyő normál esetben a memória $2000-tól $3FFF-ig terjedő részét jelnti a következő elrendezésben:

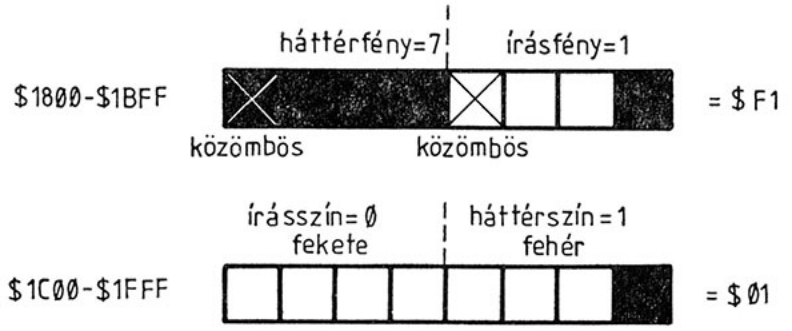


1.2. ábra
A finomfelbontású képernyő felosztása

Ez a finomfelbontású képernyőnek nevezett „ablak" természetesen a memória bármely területére „rányitható" az $FF12=65298, ún. bázisregiszter megfelelő bitjeinek beállításával. Erre a lehetőségre a Néhány grafikai fogás című részben még visszatérünk.

Mivel ez az üzemmód egy karakterhelynyi területre nem egy, hanem nyolc egymás utáni bájttal hivatkozik, így nyolcszor akkora helyet foglal a memóriában, mint a karakteres képernyő. (Közel 8 kbájtot, ami egy bővítés nélküli C—16-os gépen tetemes mennyiség.)

A finomfelbontású képernyőn a színekkel korántsem bánhatunk olyan gavallérosan, mint a képpontokkal. Bármelyik 8×8-as terület ugyanis mindössze kétféle színű lehet: a „Ø" bitnek megfelelő háttérszínű és az „1" bitnek megfelelő, úgynevezett írásszínű. Az ezek színére és fényességére vonatkozó információt egy-egy 1ØØØ bájt nagyságú memóriarész hordozza. A fényességet meghatározó terület helye $18ØØ-tól $1BFF-ig terjed, a színeket hordozó területé pedig a rákövetkező, $1CØØ-tól $1FFF-ig. Ha például COLORØ,2 paranccsal a hátteret fehérre állítjuk, COLOR1,1-gyel pedig az írásszínt feketére, akkor a következőképpen festenek a fényességet és színt meghatározó memóriaterület bájtjai:



1.3. ábra
Háttér- és írásszín állítása

Ha MONITOR-ba váltva belépillantunk a két memóriába (M 18ØØ ill. M 1CØØ monitorparanccsal), akkor valóban minden bájton ugyanazt az előbb meghatározott értéket találjuk. Normál felhasználás esetén kényelmes a COLOR utasítással háttérszínt állítani, de ha egy képernyőn egyszerre többfélét szeretnénk használni, ez az út már nem járható. Ilyen esetekben a kiválasztott 8×8-as pozíciónak megfelelő szín- illetve fényességmeghatározó bájt átírásáról magunknak kell gondoskodnunk.

```
10 GRAPHIC1,1
20 DRAW1,Ø,ØTO3Ø,3Ø
3Ø CIRCLE1,16Ø,3Ø,25
4Ø FOR I=DEC("1CØØ") TO DEC("1C4F"):REM KET KEPERNYOSOR
5Ø POKEI,DEC("16"):REM '1'-ES IRAS, '6'-OS HATTERSZIN
6Ø POKEI-1Ø24,DEC("47"):REM '4'-ES HATTER, '7'-ES IRASFENY
7Ø NEXT I
8Ø GETKEY A$:GRAPHICØ
```

Ez a kis program bemutatja ezt a lehetőséget. A 2Ø-as és 3Ø-as sorokban egy szakasz, illetve egy kör megrajzolása történik. A következő ciklus írja át a képernyő felső két sorának szín- és fényértékeit. A színérték $16, azaz az írásszín 1-es fehér, a háttérszín 6-os kék lesz. A fényesség $47, ez azt jelenti, hogy a háttér 4-es fényességű, az írás pedig a legvilágosabb 7-es. Az eredmény az, hogy a megrajzolt alakzatok új, kék sávba eső része fehér lesz, míg a többi továbbra is fehér alapon fekete.
A finomfelbontású képernyő beállítása már a program 1Ø-es sorában megtörtént a GRAPHIC 1 utasítással. Ha ezek után szöveges képernyőre térünk vissza (GRAPHIC Ø), a képernyőmemória továbbra is erre a célra marad lefoglalva, így ha később erre a képre még szükségünk lenne GRAPHIC 1,Ø utasítással visszahívhatjuk. Ha azonban „tiszta lappal" szeretnénk indulni, akkor GRAPHIC 1,1-et kell begépelnünk.
Többszínű grafika. Ez a finomfelbontású grafika egy változata. Nagyobb színválasztási lehetőséget kínál a vízszintes felbontás rovására. A színeket nem önálló pontok, hanem bitpárok hordozzák. A négyféle lehetőség (ØØ, Ø1, 1Ø, 11) mindegyike másféle

színt jelenthet, tehát egy 8×8-as pozíción belül négyféle szín használható egyszerre. Mivel most minden pont tulajdonképpen egy pontpár, a felbontás csak 160×200-as. Bekapcsolása GRAPHIC 3 utasítással történik. A felhasználni kívánt színek legegyszerűbben COLOR utasítással állíthatók be:

COLOR0	háttérszín
COLOR1	„1"-es szín
COLOR2	„2"-es szín
COLOR3	„3"-as szín

A gép a színinformációkat másképp tárolja, mint az előbbi grafikus módnál megismertük. A háttérszínt és a „3"-as színt egy-egy háttérregiszter hordozza. $FF15= =65301 regiszterben a háttér színét raktározza, a $FF16=65302 regiszter pedig a „3"-as színét. Természetesen ezáltal e színek csak az egész képernyőn egyszerre változtathatók meg. Az „1"-es és „2"-es szín beállítása történik azon a memóriaterületen, amelyet az előbbiekben a háttér és az írás színének meghatározására használtunk. Ezek 8×8-as blokkonként külön állíthatók, tehát egyes pozíciókon akár más-más értéket is felvehetnek. Beállításuk ugyanúgy történik, mint az előbb. A színregiszterek értékadásáról a Bővített háttérszínmód című részben esik szó bővebben.
A karakteres és grafikus képernyők gépi megvalósításának különbözőségéből fakadóan a finomfelbontású képernyőn közvetlenül szöveg csak CHAR utasítással jeleníthető meg. A grafikus üzemmódok bővítési lehetősége az osztott képernyő, mely lehetőséget ad finomfelbontású grafika és karakteres mód egyidejű használatára. Ilyenkor az alsó 40 bitsor 5 karaktersorra változik át. A felső megmaradt grafikus területre csak CHAR utasítással, az alsó szöveges részre pedig csak PRINT utasítással írhatunk.

GRAPHIC2,1: vágott finomfelbontás
képernyőtörléssel,
GRAPHIC4,0: vágott többszínű képernyő
törlés nélkül.

Ha a grafikus képernyőre nincs tovább szükség, és az általa elhasznált nagy mennyiségű memóriát másra szeretnénk használni, nem elegendő GRAPHIC0 utasítással visszatérni karakteres módba. Az eredeti állapot visszaállításának egyetlen módja (a kikapcsoláson kívül) a GRAPHIC CLR utasítás, amely újra felszabadítja ezt a területet.

1.1.1.1. Iskolai feldolgozás

Grafikai témájú iskolai foglalkozások sorában első lépés természetesen az üzemmódok ismertetése. Nem célszerű azonban a gyerekekkel az összes lehetőség megtanítása, használni először úgyis csak az egyszínűt fogják.
Először próbálkozzanak osztott képernyő beállításával, ekkor ugyanis lehetőség van a beírt parancsok közvetlen ellenőrzésére, míg teljes grafikus képernyő esetén kénytelenek „vakon" gépelni.

A C—16-os gép F1 funkcióbillentyűjét leütve a GRAPHIC utasítás jelenik meg automatikusan. E lehetőség kihasználása segíti a gyors üzemmódváltást. Vigyázat, a C Plus/4 számítógépre mindez nem igaz, ha egy C—16-oshoz szokott gyerek nem kellő figyelemmel használja az F1 gombot, hamar a szövegszerkesztőben találja magát és akkor nagyon csodálkozik.
Még egy könnyítési lehetőség kínálkozik: grafikus képernyőről szövegesre tér át automatikusan a gép, ha hibát talál. Ezt tudva egyszerűen visszatérhetünk karakteres képernyőnkre, ha szándékos gépelési hibát ejtünk (pl. egy betű leütése után RETURN).

1.1.2. *Grafikus utasítások*

Ha a finomfelbontású, más néven bittérképes üzemmódban megrajzolni kívánt ábráinkhoz a bekapcsolandó bitek helyét magunknak kellene kiszámolni, akkor szinte reménytelen vállalkozás lenne, akár egy szakasz elkészítése is. Szerencsére a kibővített BASIC interpreter alkalmas ezen számítások gyors elvégzésére és az ábrák képernyőrevitelére.
LOCATE x,y. A finomfelbontású képernyőn levő grafikus kurzor (idegen nevén pixel kurzor) helyét állítja be úgy, hogy látható nyomot nem hagy. A grafikus kurzor alaphelyzetben a bal felső sarokban (0,0 pontban) található. Az összes rajzoló utasítás rajzolás közben magával viszi, de a LOCATE utasítással ezektől függetlenül is állítható. Nézzük a következő rövid példát:

```
10 GRAPHIC1,1
20 FOR Y=0 TO 200 STEP 10
30 LOCATE 160,100
40 DRAW1 TO 200,Y
50 NEXT Y
```

A 40-es sorban a DRAW utasítást szakaszrajzolásra használjuk, de a szakasz kezdőpontja nincs megadva. Ilyenkor a gép a grafikus kurzor pillanatnyi helyét veszi, amit minden alkalommal a 30-as sorban a képernyő közepére állítunk. Az eredmény egy közös pontból kiinduló sugárnyaláb.
DRAW színtípus, paraméterek
Alapvető utasítás, ugyanis ez teszi lehetővé különálló pontok megrajzolását, így ügyes programba ágyazva akár az összes többit helyettesíthetné. Alkalmas még — az előbb már említettük — szakasz, sőt irányított szakasz rajzolására is.
Pont rajzolásához paraméterként elegendő egy koordinátapár. Szakasz megrajzolásához a végpontok koordinátáit kell megadni, de közéjük a TO szócska kerül. (A TO egyébként tetszőleges sokszor ismételhető, így mód van töröttvonal rajzolására is.)

Különleges lehetőség az irányított szakasz rajzolása. A LOCATE utasítással beállított grafikus kurzortól kezdődően adott hosszúságú, adott irányú szakasz húzható. Az előbbi példánkat egy kicsit átalakítva:

```
10 GRAPHIC1,1
20 FOR SZ=0 TO 360 STEP 20
30 LOCATE 160,100
40 DRAW1 TO 60;SZ
50 NEXT SZ
```

Ebben a programban a szög a ciklusváltozó. A középpont körül 10 fokonként 60 egység hosszúságú sugarakat rajzol balra körülforogva. (A 0 fok az „északi" irány.) Az előbbi példával összevetve látható, hogy a kétféle szakaszrajzoló funkció külsőleg csak a pontosvesszőben tér el egymástól.

A színtípus mindig a rajzoláshoz használandó, előzőleg COLOR utasítással beállított színt jelenti. Ha értéke 1, akkor írásszínnel, ha 0, akkor háttérszínnel rajzol (törlés). A 2-es és 3-as szín csak többszínű módban használható.

Rajzoljunk DRAW utasítással téglalapot!

DRAW1,120,80TO200,80TO200,120TO120,120TO120,80

Ugyanezt az eredményt elérhetjük a DRAW utasítás elkerülésével is:

BOX színtípus, paraméterek

Ez az egyetlen utasítás segít bennünket a téglalaprajzolásban. Paraméterként a bal felső sarok és a jobb alsó sarok koordinátáit kell megadni. Rajzoljuk meg az előbbi téglalapot így is:

BOX1,120,80,200,120

Lehetőség van a téglalap elforgatására vagy befestésére. Ehhez a sarkokat jelentő számpárok után kell írni az elforgatás szögét, ha a téglalapot be is akarjuk festeni, akkor pedig még egy egyest is. Ez utóbbiak akár el is hagyhatók.

BOX1,120,80,200,120,30,1

Ez az utasítás az előbbi téglalapot rajzolja, csak 30 fokkal elforgatva és befestve.

CIRCLE színtípus, paraméterek

Nagyon sokféle paramétert használhat, de természetesen ezek nagyrésze elhagyható. Lehetőséget ad kör, ellipszis, sokszög rajzolására, mód van ezek tetszőleges elforgatására, ívek megrajzolására. A paraméterezés részletesen a következő:

CIRCLE1,X,Y,R1,R2,SZ1,SZ2,SZ,S

A paraméterek jelentése:

X — középpont vízszintes koordinátája
Y — középpont függőleges koordinátája
R1 — vízszintes sugár

R2 — függőleges sugár
SZ1 — ívhúzáskor ettől a szögtől kezd rajzolni
SZ2 — ívhúzáskor eddig a szögig rajzol
SZ — középpont körüli elforgatás szöge
S — sokszög rajzolásakor ez a szög határozza meg, hogy a szomszédos csúcsok milyen szögben látszanak a középpontból nézve.

Ha az X,Y koordinátákat elhagyjuk, a grafikus kurzor pillanatnyi helyzetét tekinti a gép. Kör rajzolásakor R2 elhagyható.
Rajzoljunk a képernyő közepére egy szabályos hatszöget:

CIRCLE1,160,100,60,,,,,60

Ebben a példában S értéke 60 fok, tehát a körív 60 fokonként elhelyezkedő pontjainak összekötésével adódik szabályos hatszög. (Például négyzet rajzolásához S értékéül 90 fokot, szabályos háromszöghöz 120 fokot kell választanunk.)
További mintapéldák az Iskolai feldolgozás című részben találhatók.

PAINT színtípus,x,y

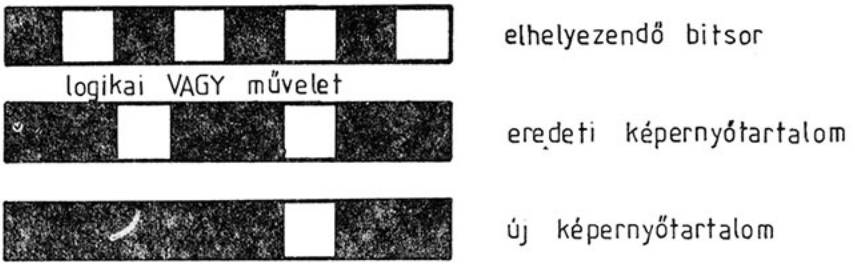
Zárt terület befestésére alkalmas. Egyetlen kikötés van: az x,y koordinátapár a befestendő terület belsejébe essen! Többszínű grafika esetén ügyelni kell arra, hogy csak a saját színtípusának megfelelő határvonalat veszi figyelembe, ha ez nem zárt, könnyen „kifolyhat" a festés nem várt helyekre is.

SSHAPE sztringváltozó, paraméterek

A grafikus képernyő egy tetszőleges téglalap alakú darabját elraktározza egy sztringváltozóban. A paraméterek a kiválasztott téglalap bal felső, illetve jobb alsó sarkának koordinátáit jelentik. Méretei nem korlátlanok, úgy kell a téglalap koordinátáit megadni, hogy a sztring hossza 255-nél nagyobb ne legyen.

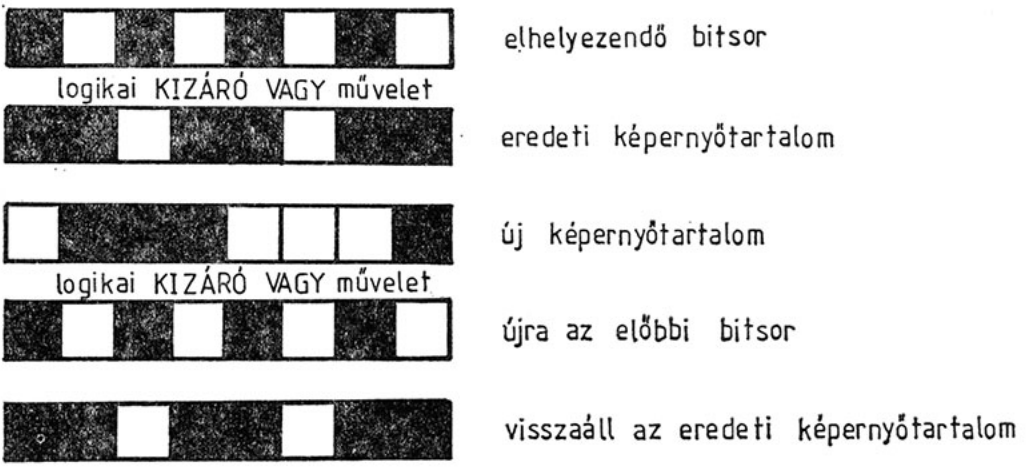
GSHAPE sztringkifejezés, paraméterek

Az SSHAPE utasítással elraktározott téglalap alakú képet tölthetjük vissza a képernyő tetszőleges helyére. A paraméterekben a kép bal felső sarkának koordinátáit kell megadni, valamint a másolás módját. Ezek a következők lehetnek:
0 — másolás. A sztringváltozóban tárolt képet leteszi, függetlenül az adott képernyőterület előző tartalmától:
1 — inverz megjelenítés. A tárolt kép ‚negatívja' kerül a képernyőre. A letakart terület előző tartalma itt is elvész.
2 — logikai vagy. Az egymásra kerülő bitek logikai „vagy" művelet eredményeként adják a keletkező kép bitjeit: az eredmény csak akkor 0, ha mindkét bit 0, minden más esetben 1. Gyakorlatilag a hatás olyan, mintha üveglapon levő képet helyeznénk a képernyő egy adott darabjára.



3 — logikai és. Az egymásra kerülő bitek a logikai „és" művelettel hasonlítódnak össze: az eredmény csak akkor 1, ha mindkét bit 1 értékű volt, minden más esetben Ø. Magyarul: csak az a része jelenik meg a képünknek, ami eredetileg nem üres területre kerül.

4 — logikai kizáró vagy. Az egymásra kerülő bitek ott, és csak ott eredményeznek 1 értékűt, ahol a két bit logikai értéke különböző. Haszna ennek a másolási módnak, hogy az általa lerakott képek úgy eltüntethetők, hogy az eredeti képernyőtartalom megmarad.



Az SSHAPE-GSHAPE utasításpár főleg akkor hasznos, ha a grafikus képernyőre készítendő feliratainkhelyét a rögzített karakterpozícióktól függetlenül „bit-pontossággal" szeretnénk megadni. Az alábbi példaprogram a választott, maximum 3Ø

karakter hosszúságú szöveget írja többször a képernyőre fél karakterhelynyi elmozdulásokkal:

```
10 PRINT"MIT IRJAK A KEPERNYORE?"
20 INPUT X$
30 IF LEN(X$)>30 THEN 10
40 GRAPHIC1,1:REM GRAFIKUS KEPERNYO BE
50 CHAR1,0,0,X$:REM SZOVEG A KEPERNYORE
60 SSHAPEA$,0,0,LEN(X$)*8,7:REM KEP ELRAKTAROZASA
70 GSHAPEA$,0,0,4:REM KEP LETORLESE
80 J=50
90 FOR I=10 TO 50 STEP 4
100 GSHAPEA$,I,J,0:REM KEP KIIRASA
110 J=J+8
120 NEXT I
130 GETKEYA$:GRAPHIC0:REM SZOVEGES KEPERNYO BE
```

CHAR színtípus, paraméterek, sztringkifejezés, inverzjelzés
A grafikus képernyőre való írást teszi lehetővé (a PRINT utasítás eredménye ugyanis láthatatlan marad). Paraméterek gyanánt egy koordinátapárt kell megadnunk, amely a szöveg első karakterének helyét adja meg a szöveges képernyőnél megszokott 40×25-ös felbontás szerint.
Néhány dologra kell ügyelni használata során. Először is a karakteres üzemmódban használható vezérlő karakterek (színállítás, inverz jelzés, villogás jelzés) nem használhatók. Hatástalanok maradnak, ráadásul képük megjelenik a képernyőn, ami enyhén szólva zavaró. A színek változtatására a COLOR utasítás szolgál. Ha pedig inverz módban szeretnénk föliratot készíteni, ezt a sztringkifejezés utáni 1-es jelzi. Másik kényelmetlenség az, hogy az utasítás a grafikus képernyőre csak az első 128 karaktert (nagybetűs karakterkészletet) tudja használni. Ez azért kellemetlen, mert hiába van ékezetes betűkészletünk, az nem használható. E hiányosság kiküszöböléséről később lesz szó.

1.1.2.1. Iskolai feldolgozás

Az összes grafikus utasítás koordinátapárokkal dolgozik. Ezért ahhoz, hogy a gyerekek ezeket használni tudják, elengedhetetlen a koordinátarendszer igen alapos, készségszintű ismerete. Az is igaz viszont, hogy maga a számítógépes grafika nyújt kiváló gyakorlási lehetőséget a koordinátarendszerben való tájékozódási készség elmélyítésére.
Az anyag feldolgozásához először tehát mutassuk be a képernyőt, mint olyan koordinátarendszert, amelynek 0,0 pontja a bal felső sarok. A koordinátapárok első értéke a vízszintes, második értéke függőleges irányban értendő. Első próbálkozáshoz megfelelő utasításnak kívánkozik a DRAW pontrajzoló funkciója. Ezzel akár maguk

a gyerekek mérhetik föl a képernyő méreteit. Segítségül szolgálhat a képernyőn való tájékozódás gyakorlására a mellékletben szereplő Torpedo című program.
Sorrendben másodikként a CIRCLE utasítás javasolható. Körrajzoláshoz ugyanis mindössze eggyel (a sugár megadásával) kell több adat, mint pontrajzoláshoz. A kör már alkalmas arra, hogy egyszerű ciklusokkal, ahol például a sugár változik, vagy a középpont egyik koordinátája, látványos mintákat hozzanak létre. Nagyon jól szemmel kísérhető a ciklus kezdő- illetve végértékének, vagy a növekmény értékének (STEP) változtatásával előálló eredmények különbsége:

```
10 GRAPHIC2,1
20 FOR R=10 TO 60 STEP 2
30 CIRCLE1,160,100,R
40 NEXT R
50 GETKEYA$:GRAPHIC0
```

Az utolsó sor egy billentyű leütése után visszaállítja a gépet karakteres üzemmódra. A következő lépés ellipszis rajzolása lehet. Itt is lehetőség van egyszerű ciklus írására. Például változtassuk az egyik tengely hosszát, minden egyéb maradjon változatlan:

```
10 GRAPHIC1,1
20 FOR R=1 TO 100 STEP 5
30 CIRCLE1,160,100,50,R
40 NEXT R
50 GETKEYA$:GRAPHIC0
```

Ha még az elforgatást is meg szeretnénk tanítani, akkor szép mintát rajzolhatunk ennek a lehetőségnek felhasználásával:

```
10 GRAPHIC 2,1
20 FOR SZ=0 TO 360 STEP 10
30 CIRCLE1,160,100,80,40,,,SZ
40 NEXT SZ
50 GETKEYA$:GRAPHIC0
```

Bármely program bővíthető a *ciklust vezérlő értékek INPUT-tal való megadásával.* Így rugalmasan kezelhető, egyszerű mintarajzoló programokhoz jutnak a gyerekek. Nem muszáj túl sokat magyarázni, a gyerekek fantáziája úgyis hamarosan nekilendül és olyan változtatásokat produkál, amik esetleg eszünkbe se jutottak volna.
A BOX utasítás és a PAINT utasítás maradhat a végére. Mindkettő remek alkalmat nyújt a képernyőn való tájékozódás gyakorlására. Különösen a PAINT utasítás követel a gyerekektől nagy odafigyelést, ugyanis egy rosszul „célzott" PAINT mindent átfest, csak azt nem, amit kellene.

Ha az SSHAPE-GSHAPE utasításpárt nem tanítjuk meg mindenkivel teljes mélységében, nem történik probléma. *Gyakorlásképpen írassunk programot, amely futballpálya alaprajzát készíti el.* A program megírása előtt készítsenek négyzetrácsos papíron tervet, amelyen kiszámíthatják a szakaszvégpontok koordinátáit.

1.1.3. *Grafikus függvények*

Négy függvény áll rendelkezésre a grafikus képernyő vizsgálatára.

RGR (kifejezés)

Az éppen érvényben levő grafikus üzemmódnak megfelelő értéket adja vissza. A kifejezés értéke tetszőleges, úgynevezett álargumentum. Próbáljuk ki:
PRINT RGR(Ø) eredményéül Ø-t kaptunk, feltéve hogy szöveges képernyőn dolgoztunk. Váltsunk például osztott képernyős grafikus üzemmódra: GRAPHIC2,1. Ha most adjuk ki az előbbi parancsot, 2-t kapunk eredményül.

RCLR (kifejezés)

A kifejezésben megadott színtípus színértékét adja vissza. A kifejezésben szerepeltethető színtípusok a következők:
Ø — háttérszín
1 — írásszín
2 — „2"-es szín (többszínű üzemmódnál)
3 — „3"-as szín (többszínű üzemmódnál)
4 — keret színe

RLUM (kifejezés)

A kifejezésben megadott színtípus fényességének értékét adja vissza. Kifejezés gyanánt ugyanazok az értékek szerepeltethetők, mint az előbb. Vizsgáljuk meg például az utóbbi két függvény segítségével, hogy milyen színű és fényességű az eredeti keret:

PRINT RCLR(4),RLUM(4)

Az eredményül kapott számpár: 15 és 6. Tehát a keret 15-ös sötétkék 6-os fényességgel.

RDOT (kifejezés)

A leghasznosabb grafikus függvény. Segítségével kereshetjük meg a grafikus kurzor helyét a képernyőn, vagy ami még fontosabb, ezzel vizsgálhatjuk meg bármely képpont állapotát (bekapcsolt, vagy kikapcsolt).

RDOT(Ø): a grafikus kurzor x koordinátája.

RDOT(1): a grafikus kurzor y koordinátája.

A grafikus mód bekapcsolása után mindkét függvényérték Ø (bal felső sarok). Állítsuk a grafikus kurzort kísérletképpen középre: LOCATE160,100! Ezután megvizsgálva a két függvényértéket a következőket tapasztaljuk: RDOT(Ø)=160 és RDOT(1)=100.
RDOT(2): a grafikus kurzor pillanatnyi helyének színtípusa. Ha a függvényértéke Ø, akkor a vizsgált pont háttérszínű (üres), ha az érték 1, akkor írásszínű. Többszínű üzemmódban értékül kaphatunk még 2-t, illetve 3-at, feltéve, hogy a függvénnyel vizsgált pont 2-es, vagy 3-as színű.

```
10 GRAPHIC2,1
20 DRAW1,160,100
30 CIRCLE1,160,100,50
40 INPUT "MELYIK PONTOT VIZSGALOD";X,Y
50 LOCATEX,Y
60 PRINT"EZ A PONT";RDOT(2);"ERTEKU"
70 GOTO 40
```

Ez a program bemutatja e függvény működését. A 10-es sorban bekapcsolt osztott grafikus képernyő közepére a 20-as sorban egy pontot, a 30-as sorban pedig egy 50 egység sugarú kört rajzol. A 40-es sorban a vizsgálandó pont koordinátáit kéri, majd LOCATE függvénnyel beállítja a grafikus kurzort a kívánt helyre. A 60-as sorban történik az értékelés és az eredmény kiírása. Próbáljuk ki a következő koordinátapárokat:

160,100 — az eredmény 1 lesz. (a középpont írásszínű)

159,100 — ez közvetlenül a középponttól balra eső bit.

Mivel háttérszínű, biztos Ø értéket kapunk.

160,50 — a körív középpont feletti pontja írásszínű, tehát a kapott érték 1.

1.1.3.1. Iskolai feldolgozás

A grafikus függvények közül az utolsóval, az RDOT függvénnyel érdemes fakultációs foglalkozáson foglalkozni. Ha az argumentumának az értéke 2, akkor a grafikus kurzor helyének színtípusát adja, tehát *alkalmas „találatjelzésre"*. Ezen a lehetőségen alapszik a mellékelt kazettán levő Torpedó című játék. Ha ugyanis LOCATE x,y utasítással a vizsgálandó pontra állítjuk a kurzort, akkor egy egyszerű feltétellel megvizsgálhatjuk, hogy van-e ott valami. Ha RDOT(2)=Ø, akkor a pont háttérszínű, vagyis üres. Érdekes játékprogramot írhatnak a gyerekek kevés segítséggel. Az alábbi program — többszörösen — sikerrel kipróbált. A játék lényege, hogy a képernyő bal alsó sarkában levő képzeletbeli ágyúval kell eltalálnia a célt a kilövési szög és a kezdősebesség megadásával. A program kirajzolja a lövedék röppályáját, és ha találatot észlel, akkor ezt jelzi.

A 70-es sor számítja át a fokokban megadott szöget radiánba. A 80-as és 100-as sorban levő képleteket el kell árulnunk előre. Ezek végzik a röppálya kiszámítását. A találat vizsgálata a 120, 130-as sorokban történik. (Az y helyett mindenütt 160—y szerepel, hogy a 0,0 pont alulra kerüljön.)
Nagyon fontos tudnivaló a RDOT (2) függvénnyel kapcsolatban az, hogy mindig LOCATE utasításnak kell megelőznie. Bár mindegyik grafikus utasítás állítja a grafikus kurzort (pl. a DRAW is), de egyúttal értékét is megváltoztatja, így vizsgálata fölösleges lenne (mindig ugyanazt az eredményt kapnánk).
A program:

```
10 COLOR4,1:COLOR0,1:COLOR1,2
20 VOL 7
30 GRAPHIC2,1
40 CHAR1,INT(RND(0)*10)+28,19,"■"
50 INPUT"SZOG";SZ
60 INPUT"SEBESSEG";V
70 SZ=π/180*SZ
80 X1=COS(SZ):Y1=SIN(SZ)
90 FOR T=0 TO 10 STEP 0.05
100 X=T*V*X1:Y=T*V*Y1-9.81*T*T
110 IF Y<0 OR X>320 THEN 160
120 LOCATEX,160-Y
130 IF RDOT(2)=1 AND X>200 THEN 170
140 DRAW1,X,160-Y
150 NEXT T
160 GOTO 50
170 SOUND3,300,10
180 PRINT"ELTALALTAD!"
190 PRINT"NYOMJ MEG EGY BILLENTYUT!"
200 GETKEYA$:RUN
```

A program természetesen még a gyerekek fantáziájától függően tovább bővíthető. Például rajzolja meg az ágyút, adjon ki hangot a lövés pillanatában és repülés közben, legyen a cél a levegőben stb.

1.1.4. *Néhány grafikai fogás*

A grafikus utasításoknál említés történt arról, hogy a CHAR utasítás a finomfelbontású képernyőn csak a nagybetűs karakterkészletet használhatja, így a beépített ékezetes betűk (amelyek a kisbetűs készletben vannak) nem érhetők el. Problémánkon segít, ha tudjuk, hogy a 740-es memóriacím tartalmazza a grafikus képernyőn használt **karakterkészlet kezdőcímet.** Normál állapotban ez $D0=208. ($D000 címnél kezdődik a karaktergenerátor.) A kisbetűs készlet kezdőcíme $D400, ha innen

akarjuk venni karaktereinket, akkor az említett 740-es címre $D4=212-t kell írni (POKE 740,212). Ha saját tervezésű karaktert is használni szeretnénk, akkor a Karakterdefiniálás című fejezetben leírtak alapján kell eljárnunk, majd az új karakterkészlet kezdőcímének felső két helyiértékén levő számot kell beírni a 740-es címre (tízes számrendszerben). Például, ha karakterkészletünk $3000-nál kezdődik akkor $30=48-at. Figyelem! Ez a változtatás csak a grafikus képernyőre vonatkozik, karakteres képernyőre nem.
Másik érdekes lehetőséget rejt magában az a tény, hogy a gép nemcsak az eleve kijelölt területet ($2000—$3FFF) tudja grafikus képernyőként értelmezni, hanem $2000 bájtonként a memória bármely területét. **Hogy a memória mely részét lássuk a képernyőn, azt a $FF12=65298 bájt 3—5 bithármasa mutatja meg.**

1.4. ábra
A grafikus képernyőterület állítása

A számunkra érdekes bithármas jelentése:

értéke:	grafikus képernyőként értelmezett terület kezdete:
000	$0000
001	$2000 (normál állapot)
010	$4000
011	$6000
100	$8000
101	$A000
110	$C000
111	$E000

Mire jó mindez? Például arra, **grafikus animációt** készítsünk, azaz úgy jelenítsük meg ábráinkat egymás után, hogy a megrajzolást magát ne lássuk, csak a kész eredményt. Az eljárás nagyon egyszerű. Miután grafikus üzemmódra váltottunk, írjuk át az $FF12 regiszter értéket úgy, hogy például a $6000-tól lássuk a memoriát (POKE 65298,220). Ha most adunk ki például egy körrajzoló utasítást, ugyanazt a csíkos képernyőt látjuk továbbra is. Pedig a rajz elkészült, de az eredeti helyén. Ahhoz, hogy megláthassuk, az egész képernyőmemóriát ($2000—$4000) át kell másolni a $6000 címtől kezdődő részre. Ha erre van gyors gépi kódú rutinunk, akkor SYS utasítással hívva, egy szemvillanás alatt megjelenik a kép. Ha ezt az eljárást egy

egyszerű BASIC programba építjük be, akkor először be kell töltenünk a megfelelő helyre a memóriaterületet másoló rutint, utána az előbb ismertetett műveletsor következik. Ezek után bármit rajzoltathatunk a program további részében, az eredményt csak akkor látjuk meg, ha utána mindig kiadjuk az áttöltő rutin kezdőcímét tartalmazó SYS utasítást.
A képernyő területét átmásoló rutin:

```
. 1500  A0 20     LDY #$20     ;kulso ciklus kezdoertek
. 1502  A2 FF     LDX #$FF     ;belso ciklus kezdoertek
. 1504  BD 00 20  LDA $2000,X  ;eredeti kepernyoteruletrol
. 1507  9D 00 60  STA $6000,X  ;uj teruletre
. 150A  CA        DEX          ;X=X-1
. 150B  D0 F7     BNE $1504    ;belso ciklusra
. 150D  AD 00 20  LDA $2000    ;0-dik
. 1510  8D 00 60  STA $6000
. 1513  EE 06 15  INC $1506
. 1516  EE 09 15  INC $1509
. 1519  EE 0F 15  INC $150F
. 151C  EE 12 15  INC $1512
. 151F  88        DEY          ;Y=Y-1
. 1520  D0 E0     BNE $1502    ;kulso ciklusra
. 1522  A9 20     LDA #$20     ;eredeti allapot vissza
. 1524  8D 06 15  STA $1506
. 1527  8D 0F 15  STA $150F
. 152A  A9 60     LDA #$60
. 152C  8D 09 15  STA $1509
. 152F  8D 12 15  STA $1512
. 1532  60        RTS
```

A program BASIC betöltője:

```
10 FOR I=0 TO 50
20 READ A:POKE DEC("1500")+I,A
30 NEXT I
40 DATA 160,32,162,255,189,0,32,157,0,96
50 DATA 202,208,247,173,0,32,141,0,96
60 DATA 238,6,21,238,9,21,238,15,21,238
70 DATA 18,21,136,208,224,169,32,141,6
80 DATA 21,141,15,21,169,96,141,9,21,141,18,21,96
```

A rutint a SYS DEC("1500") utasítás hívja. Ezt a most ismertetett grafikai lehetőséget a mellékelt kazetta „Mozgó grafika" című rövid bemutatóprogramja illusztrálja.

2. KARAKTERDEFINIÁLÁS

2.1. Bevezetés

Ha az ember egy kicsit jobban belejön a programírásba, hamar úgy érzi, hogy a gép lehetőségei nem követik a fantáziáját. Játékprogramok vagy érdekesebbnek ígérkező oktatóprogramok például nehezen képzelhetők el a C—16 ,vagy C PLUS/4 számítógépbe épített betűkkel és grafikus karakterekkel. Először bizonyos matematikai jelek hiányoznak, mint a négyzetgyök jel, integráljel, bizonyos hatványok, később jó lenne esetleg cirill betűs készlet, vagy saját tervezésű kis figurák. Memóriatakarékossági szempontoknak felelne meg, ha esetleges térképeinket, finomfelbontású rajzainkat nem grafikus képernyőn valósítanánk meg, hanem ügyesen elkészített saját karakterekből állítanánk össze. Az ilyen jellegű munkákhoz nyújt segítséget ez a fejezet.

2.1.1. *Alapismeretek*

A gép két, egyenként 128 jelből álló karakterkészlettel dolgozik. Az egyik nagybetűs, a másik a kisbetűs készlet. *Normál karaktermód* esetén egyszerre a képernyőn csak az egyiket használhatjuk. Átválthatunk egyikről a másikra a *C= és SHIFT* gombok együttes megnyomásával, vagy programfutás közben **kisbetűre vált a PRINT CHR$(14), nagybetűre a PRINT CHR$(142)** utasítás. A véletlen átváltás megakadályozására is lehetőség van, **a PRINT CHR$(8) utasítás hatástalanítja C= SHIFT billentyűpárt. A tiltás feloldása a PRINT CHR$(9) utasítással lehetséges.**
A két karakterkészlet a karaktergenerátor ROM-ban helyezkedik el $D000-tól $D7FF-ig. Mint a neve is mutatja (ROM), csak olvasható memóriáról van szó, tehát helyben való átírásáról nem lehet szó. Ahhoz, hogy bizonyos karaktereket vagy akár az összeset saját ízlésünk szerint átalakítsuk, az egész jelkészletet át kell telepíteni egy olyan RAM jellegű memóriarészre, amit azután másra már nem használunk. Természetesen a változásról, az új helyre költözésről a gépet külön értesíteni kell, de ennek módjára majd még kitérünk. Az új karakterterületen azután már szabadon variálhatjuk a készletünket akár programfutás közben is.
Minden karakter egy 8×8-as pontrácsként van elhelyezve a memóriában. Minden nyolc pontból álló sor egy bájtban van elraktározva, és mivel egy betű nyolc ilyen sorból áll, egy karakter képét nyolc egymás utáni bájt hordozza. Egy betű átírásához tehát nyolc bájt értékét kell megváltoztatni. A különféle jelek képe azután úgy jele-

nik meg a képernyőn, hogy a bájtok „*1*" *értékű bitjei írás színűek*, míg a „*0*" *értékűek háttér színűek* lesznek.
Például az „A" betűt nézzük meg a memóriában.
Gépeljük be a gépbe a következőket:

MONITOR; áttérés monitorba
M D000; memória listázása $D000-tól

```
MONITOR

   PC  SR AC XR YR SP
 02FF 00 FF 02 FF F8

>D000 3C 66 6E 6E 60 62 3C 00 :
>D008 18 3C 66 66 7E 66 66 00 :
>D010 7C 66 66 7C 66 66 7C 00 :
>D018 3C 66 60 60 60 66 3C 00 :
>D020 78 6C 66 66 66 6C 78 00 :
>D028 7E 60 60 78 60 60 7E 00 :
>D030 7E 60 60 78 60 60 60 00 :
>D038 3C 66 60 6E 66 66 3C 00 :
>D040 66 66 66 7E 66 66 66 00 :
>D048 3C 18 18 18 18 18 3C 00 :
>D050 1E 0C 0C 0C 0C 6C 38 00 :
```

Mivel az „A" betű a készlet második tagja, ezért képe a memóriában a $D008 címtől kezdődik. *Váltsuk át* az ott következő nyolc bájt értékét *kettes számrendszerbe*:

```
$18= 00011000
$3C= 00111100
$66= 01100110
$66= 01100110
$7E= 01111110
$66= 01100110
$66= 01100110
$00= 00000000
```

2.1. ábra
Az „A" betű képe a memóriában

Máris kirajzolódik a betű képe. Karakterdefiniálásnál éppen fordítva fogunk eljárni, tervünket *tízes* vagy **tizenhatos számrendszerbe váltva** a memória megfelelő helyére beírjuk.

2.1.2. *Karakterkészlet átmásolása*

Mielőtt saját tervezésű karaktereinket el szeretnénk helyezni a memóriában, előbb a ROM-ból a BASIC területtől elrabolt RAM területre kell átmásolni. Két lehetőség kínálkozik. Az egyik az, hogy **monitorban egyetlen utasítással** elintézzük a másolást. Ennek előnye a gyorsaság, hátránya, hogy programból nem használható. Másik lehetőség egy **gépi kódú rutin használata.**
Nézzük előbb az elsőt, az egyszerűbbet:
Tegyük föl, hogy készletünk új helyéül megfelel $3000 és $37FF közötti terület. Természetesen ha majdani programunk ezt a területet használná, az elrontaná a karakterek képét, ezért ha a veszély fennáll, lehet máshova is másolni.
Teendőnk a következő:

```
MONITOR          ; áttérés monitorba

T D000 D7FF 3000 ; memóriaterület másolása
```

Máris megnézhetjük munkánk eredményét, ha megnézzük a $3000-tól kezdődő memóriaterületet:

```
M 3000           ; memória listázása
```

Ha semmit sem tévesztettünk el, akkor itt is ugyanazt találjuk most már, mint az előbb a $D000 címnél. Térjünk vissza a BASIC-be:

```
X                ; visszatérés
```

Helytakarékosságból elképzelhető, hogy például csak a kisbetűs készletre lévén szükségünk, csak azt másoljuk át. Ilyenkor a másolást $D400-tól $D7FF-ig elég megtenni. (Csak nagybetű esetén $D000—$D3FF.)
Ahhoz, hogy gépünk tudomást vegyen az új helyen lévő karakterkészletről, **két regiszter értékét meg kell változtatnunk.** Az egyik a $FF12=65298 című bájt, amely többek közt arról tájékoztatja a számítógépet, hogy a használandó karakterek RAM-ból, vagy ROM-ból származnak-e. Eredeti értéke 196, írjuk ezt át 192-re (POKE 65298,192). A másik regiszter az $FF13=65299 című bájt, mely azt tartja számon, hogy a memóriában hol a karaktergenerátor kezdete. **A címnek csak a két felső helyiértéke tárolódik,** az utolsó kettőt automatikusan 0-nak veszi. Normál állapotban értéke $D0=208 ($D000 két felső helyiértéke). Hogy az átmásolt karaktereinket megtalálja a gép, írjuk át a regiszter tartalmát $30=48-ra (POKE 65299,48). Érdemes a két regisztert közvetlenül egymás után átírni, különben a képernyőn furcsa jelenségeket észlelünk. Egyébként is vigyázni kell, ugyanis bármilyen hibát vétünk ezután, a hibajelzés mellé még a képernyő be is mákosodik, úgyhogy tovább nem is lehet rajta dolgozni. Ennek oka, hogy hiba után az $FF12 regiszter értéke eredeti értékére állítódik vissza, míg az $FF13 nem. Ez a kényelmetlenség természetesen kiküszöbölhető, hamarosan szó lesz róla.

A másik lehetőség a karakterkészlet átmásolására egy gépi kódú rutin használata. Előnye, hogy a programokba beépíthető, így használata kényelmes. A következő példa feltételezi, hogy karakterkészletünk számára megint megfelel a $3000 címtől kezdődő memóriarész. A rutin közvetlenül a jövendőbeli karakterkészlet mögött foglal helyet, $3800 címtől kezdődően.

```
. 3800  A0 08     LDY #$08     ;kulso ciklus kezdoertek
. 3802  A2 FF     LDX #$FF     ;belso ciklus kezdoertek
. 3804  BD 00 D0  LDA $D000,X  ;karaktergenerator ROM-bol
. 3807  9D 00 30  STA $3000,X  ;RAM-ba
. 380A  CA        DEX          ;X=X-1
. 380B  D0 F7     BNE $3804    ;belso ciklus elejere
. 380D  AD 00 D0  LDA $D000    ;0-dikat is
. 3810  8D 00 30  STA $3000
. 3813  EE 06 38  INC $3806
. 3816  EE 09 38  INC $3809
. 3819  EE 0F 38  INC $380F
. 381C  EE 12 38  INC $3812
. 381F  88        DEY          ;Y=Y-1
. 3820  D0 E0     BNE $3802    ;kulso ciklus elejere
. 3822  60        RTS
```

Természetesen a program megfelelő címeinek átírásával bármilyen igényeknek megfelelően átalakítható.
A gépi rutin BASIC betöltője:

```
10 FOR I=0 TO 34
20 READ A
30 POKE DEC('3800')+I,A
40 NEXT I
50 SYS DEC('3800'):REM INDITAS
60 DATA 160,8,162,255,189,0,208,157,0,48,202
70 DATA 208,247,173,0,208,141,0,48,238,6,56,238
80 DATA 9,56,238,15,56,238,18,56,136,208,224,96
```

A gépi kódú rutin a két DATA sorban van elhelyezve, innen töltődik be a ciklus segítségével a megfelelő tárcímtől kezdődően.
A betöltés megtörténte után az elindítás a feladata az 50-es sorban levő SYS utasításnak. Természetesen az így áttöltött karakterkészletünket sem érhetjük el a megfelelő regiszterek átírása nélkül, tehát a betöltőprogram végére kerülhet a következő sor:

90 POKE 65298,192:POKE 65299,48

Egy hiányossága még mindig van az új helyen levő karakterkészlettel működő rendszernek. Az előbb említett képernyő „elmákosodásról" van szó, amely már a legelső *szintaktikus hibánál* is bekövetkezik. „Kikecmeregni" belőle csak a két POKE uta-

sítás újbóli begépelésével lehet (természetesen „vakon",ugyanis a képernyő ilyenkor teljesen használhatatlan).
Ez a ránk leselkedő veszély elhárítható még egy rövid gépi kódú rutin segítségével:

```
. 3830  A9 3B    LDA #$3B  ;hibakezelo cime also
. 3832  8D 14 03 STA $0314
. 3835  A9 38    LDA #$38  ;hibakezelo cime felso
. 3837  8D 15 03 STA $0315
. 383A  60       RTS
. 383B  A9 C0    LDA #$C0  ;hibakezelo kezdete
. 383D  8D 12 FF STA $FF12
. 3840  A9 30    LDA #$30
. 3842  8D 13 FF STA $FF13
. 3845  4C 0E CE JMP $CE0E ;megszakitaskezelesre
```

A rutin két részből áll. Az első megadja a gépnek azt a címet, ahova megszakítás esetén ugrani kell ($383B). Így kezdődik ugyanis a program második része, amely nem tesz mást, mint a két regiszter ($FF12=65298 és $FF13=65299) értéket visszaírja az általunk megadott új értékekre ($C0=192 és $30=48).
Ez a rutin is BASIC-ből betölthető a megfelelő memóriaterületre:

```
10 FOR I=0 TO 23.
20 READ A
30 POKE DEC("3830")+I,A
40 NEXT I
50 SYS DEC("3830")
60 DATA 169,59,141,20,3,169,56,141,21,3,96,169
70 DATA 192,141,18,255,169,48,141,19,255,76,14,206
```

A megfelelő helyre töltött gépi kódú programot a 60-as sorban levő SYS utasítás indítja el. Az indítást követően semmi különöset nem tapasztalunk, de ha ezek után például gépelési hibát ejtünk, egy apró villanás jelzi, a programocska működik, és nem hagyja tönkremenni a képet.
Befejezésül álljon itt egy komplett áttöltő rutin, amely egy, az eddig ismertetetteken felüli szolgáltatást is nyújt: a sztringterület kezdetét jelző mutatót is megfelelően, a karakterkészlet elé állítja, így nem fordulhat elő, hogy a programfutás közben egyre szaporodó sztringek hátulról ellepjék a karaktereinket.

```
. 3BA0  A0 04     LDY #$04     ;kulso ciklus kezdoertek
. 3BA2  A2 FF     LDX #$FF     ;belso ciklus kezdoertek
. 3BA4  BD 00 D4  LDA $D400,X  ;karaktergenerator ROM-bol
. 3BA7  9D 00 3C  STA $3C00,X  ;RAM-ba
. 3BAA  CA        DEX          ;X=X-1
. 3BAB  D0 F7     BNE $3BA4    ;belso ciklus elejere
. 3BAD  AD 00 D4  LDA $D400    ;0-dikat is
. 3BB0  8D 00 3C  STA $3C00
. 3BB3  EE A6 3B  INC $3BA6
. 3BB6  EE A9 3B  INC $3BA9
. 3BB9  EE AF 3B  INC $3BAF
. 3BBC  EE B2 3B  INC $3BB2
. 3BBF  88        DEY          ;Y=Y-1
. 3BC0  D0 E0     BNE $3BA2    ;kulso ciklus elejere
. 3BC2  A9 A0     LDA #$A0     ;sztringterulet vege also
. 3BC4  85 37     STA $37
. 3BC6  A9 3B     LDA #$3B     ;sztringterulet vege felso
. 3BC8  85 38     STA $38
. 3BCA  A9 C0     LDA #$C0     ;karakterek a RAM-bol
. 3BCC  8D 12 FF  STA $FF12
. 3BCF  A9 3C     LDA #$3C     ;uj karakterkeszlet kezdete
. 3BD1  8D 13 FF  STA $FF13
. 3BD4  A9 DF     LDA #$DF     ;hibakezelo cime also
. 3BD6  8D 14 03  STA $0314
. 3BD9  A9 3B     LDA #$3B     ;hibakezelo cime felso
. 3BDB  8D 15 03  STA $0315
. 3BDE  60        RTS
. 3BDF  A9 C0     LDA #$C0     ;hibakezelo
. 3BE1  8D 12 FF  STA $FF12
. 3BE4  A9 3C     LDA #$3C
. 3BE6  8D 13 FF  STA $FF13
. 3BE9  4C 0E CE  JMP $CE0E    ;megszakitaskezelesre
```

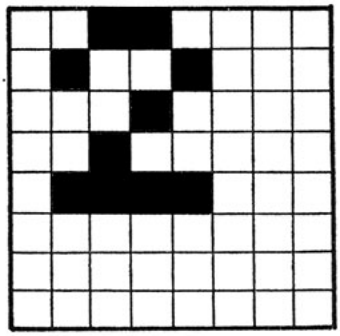
A rutin BASIC-ből betölthető ezzel a programmal:

```
10 FOR I=0 TO 76
20 READ A
30 POKE DEC("3BA0")+I,A
40 NEXT I
50 SYS DEC("3BA0")
60 DATA 160,4,162,255,189,0,212,157,0,60,202
70 DATA 208,247,173,0,212,141,0,60,238,166,59
80 DATA 238,169,59,238,175,59,238,178,59,136,208
90 DATA 224,169,160,133,55,169,59,133,56,169,192
100 DATA 141,18,255,169,60,141,19,255,169,223,141
110 DATA 20,3,169,59,141,21,3,96,169,192,141
120 DATA 18,255,169,60,141,19,255,76,14,206,96
```

Ez a betöltő rész bármilyen nagyobb programba beépíthető modulként, az általa betöltött gépi kódú rutin ugyanis további „gondozást" nem igényel. Áttölti a kisbetűs készletet új helyére (A nagybetűsre úgysem tudnánk váltani a hibakezelő rész miatt), automatikusan beállítja a két regiszter ($FF12=65298, $FF13=65299) értéket, a sztringterület kezdetét a karakterkészlet elé beállítja, és gondoskodik az „elmákosodás" elleni védekezésről.

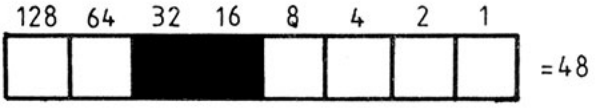
2.1.3. *Új karakterek tervezése*

Tegyük fel, hogy munkánkban elkerülhetetlen például a négyzetreemelés jelölése. Ilyen jel viszont nincs az eredeti karakterkészletben, ezért nekünk kell azt megtervezni. Első dolgunk egy 8×8-as üres rács rajzolása. Ebbe rajzoljuk bele a kisméretű 2-es szám képét:



2.2. ábra
Saját karakter tervezése

Ahogy már volt szó róla, minden sor a memóriában egy nyolcbites bájt, a nyolc sor mindegyikét *tízes számrendszerre át kell váltanunk,* hogy a memóriába írhassuk a megfelelő értékeket. Ehhez segítséget nyújt, ha a bitek fölé megjegyezzük a nekik megfelelő *helyiértéket:*



2.3. ábra
A kettes számrendszer helyiértékei

Amelyik négyzetet beszíneztük, a neki megfelelő bit értéke „1" lesz, az üresen maradottaké pedig „Ø". Az előbbi sor értéke tehát 32+16=48. A teljes karakter sorainak értéke:

```
00110000=  48
01001000=  72
00010000=  16
00100000=  32
01111000= 120
00000000=   0
00000000=   0
00000000=   0
```

2.4. ábra
Sorok átváltása tízes számrendszerbe

Egyetlen feladat maradt hátra, az új jel elhelyezése a memóriában. Válasszuk ki azt a karaktert, amelyiket nélkülözni tudnánk. Legyen ez a „£" jel. Az ő helyét fogjuk elfoglalni. Táblázatból utána nézhetünk, hogy a „£" jel a 28-as számú, tehát a 29-dik a memóriában (van 0-dik is, a „@"). Számítsuk ki ennek alapján a memóriabeli kezdőcímet. $3000=12288 és **minden karakter nyolc bájt helyet foglal el,** tehát a keresett cím: **12288+28*8=12512.** Innen kezdődően töltsük az előbb kiszámított értékeket:

```
10 FOR I=0 TO 7
20 READ A
30 POKE 12512+I,A
40 NEXT I
50 DATA 48,72,16,32,120,0,0,0
```

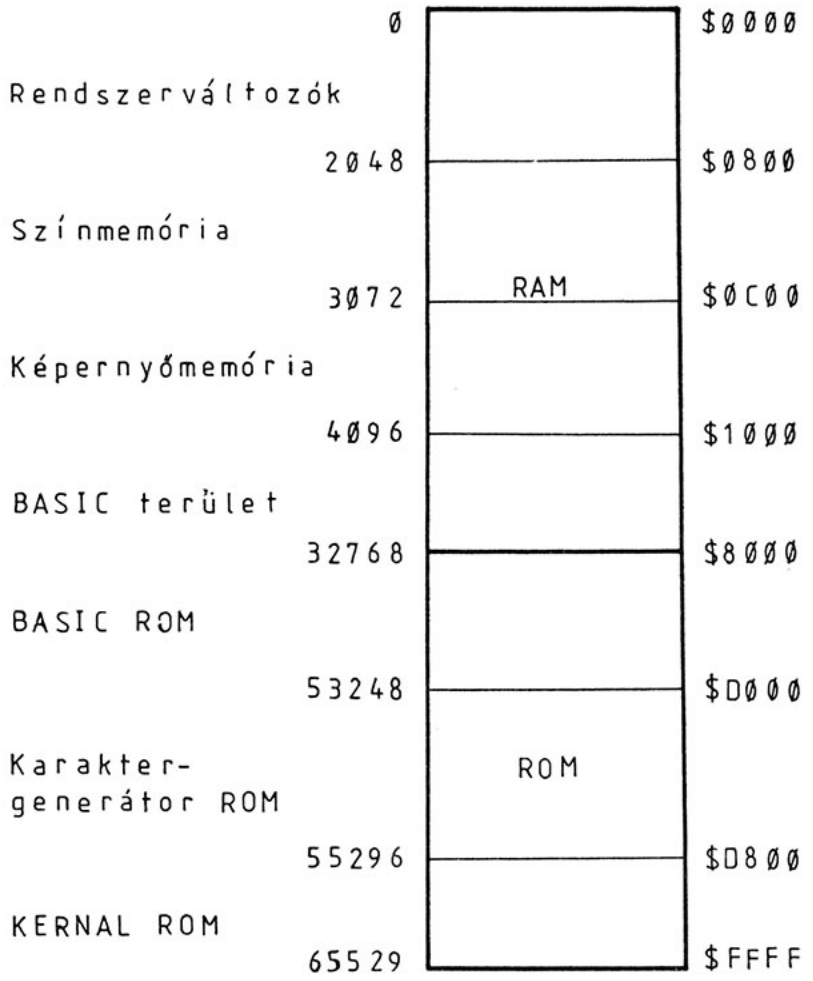
Természetesen előzőleg kellett a karakterkészlet áttöltését elvégezni. Ha ezek után a „£" billentyűt leütjük, a gombon levő jel helyett a saját karakterünk képét látjuk a képernyőn megjelenni.
Lehetővé válik például négyzetméter, négyzetdeciméter stb. képernyőreírására. A további lehetőségeknek csak a fantázia végessége szab határt. Gyakorlásképpen érdemes megpróbálkozni esetleg a köbméter megjelenítésével, új betűkészlettel, vagy kis játékfigurákkal.

2.1.3.1. Iskolai feldolgozás

A karakter újradefiniálás első pillantásra nem látszik könnyű feladatnak, érdemes mégis elmélyedni benne és gyerekeknek is megtanítani. Már a lehetőség felvillantása is izgalmas: változtassuk meg a karaktereket, csináljunk játékfigurákat stb.
Ahhoz, hogy a gyerekek megértsék a teendőket a következőket kell már előre ismerniük:

— *számítógép belső felépítésének alapjait,* (processzor, memória, bájt, bit)
— *kettes számrendszert* (általában a számrendszereket) — ez általános iskola alsós tananyag,
— *alapvető BASIC utasításokat.*

A továbbiak megtanítására a következő sorrend javasolható:
A számítógép memóriafelosztásának vázlatos ismertetése. Érdemes minél több rajzzal illusztrálni mondandónkat (ez érvényes a továbbiakra is), ugyanis csak így válik a gyerekek számára viszonylag könnyen befogadhatóvá.



2.5. ábra
Memóriafelosztás

A memóriafelosztás ismertetése közben nem árt természetesen néhány szót a **tizenhatos számrendszerről** is ejteni. Például előnye, hogy nagy számokat is meglehetősen rövid formában írhatunk le vele. A tízes számrendszerbe való átváltásukra nem érdemes időt fordítani, amelyik gyereket érdekli, úgyis megtanulja magától. Elég, ha nagyságrendi viszonyokat meg tud állapítani tizenhatos számrendszerbeli számok között.

ROM (Read Only Memory): csak olvasható memróia és **RAM (Random Access Memory):** írható-olvasható memória közötti különbség.

Ki kell emelni azt, hogy az eredeti karakterek a ROM memóriában vannak, így megváltoztatásuk ott nem lehetséges, ezért kell átjuttatni az egészet valahová a RAM területre. Itt érdemes ezt rögtön meg is tenni a gyerekekkel úgy, ahogy az előbb már volt róla szó (monitorban a „T" paranccsal). Így ugyanis munkájuk eredményét rögtön megnézhetik.

Tudassuk a géppel is, hogy változás történt Itt kell szót ejtenünk a két **vezérlőregiszterről,** ami épp a **karaktergenerátor-terület** címét tartja számon. Az előbbi memóriatérkép „KERNAL ROM" részén belül található ez a bizonyos két bájt, az $FF12 = =65298, illetve az $FF13=65299 címen. Elég első nekifutásra a két regiszter szerepét ismertetni és elárulni új értékeit, kiszámításukra később is kitérhetünk.

Most már minden előkészület megvolt, hozzáfoghatunk az átíráshoz. Ehhez azonban még meg kell ismerniük a karakterek tárolásának módját: minden betű egy 8×8-as rács be-, illetve kikapcsolt pontokkal. Egy sor — egy bájt, tehát a karakterenkénti nyolc sor ábrázolásához nyolc bájtra van szükség. Érdemes itt felrajzolni négyzetrácsos táblára például az „A" betű képét és soronként *átszámolni tízes számrendszerbe.* Emlékeztetőül persze nem árt a bitek fölé a *kettes számrendszer helyiértékeit felírni,* ez gyerekjátékká teszi az átszámítást: egyszerűen a bekapcsolt pontok helyiértékeit kell összeadni.

Ezután már mindenki *önállóan tervezhet a füzetbe karaktereket* és *át is számíthatja a sorait,* ha a 8×8-as rácsot megrajzolja.

Utolsó lépcsőfok a saját karakterek memóriábajuttatása. Ehhez érdemes egy nyúlfarknyi BASIC programot segítségül hívni. Ezt a programot kis segítséggel akár maguk is megírhatják a gyerekek, ha a POKE utasítás jelentését előbb természetesen eláruljuk.

Ebben a programban előbb megadjuk a karakterkészlet új helyének kezdőcímét (tizenhatos számrendszerben), majd azt a karaktert, amelyet át szeretnénk definiálni. Ezután a program kiszámítja a karakter helyét a memóriában (a korábban már ismertetett számítással) és az INPUT-tal beírt számoknak megfelelően sorról sorra elhelyezi a memóriában. Az éppen átírás alatt levő betűt POKE utasítással a képernyőre írtuk, így szemmel követhetjük átalakulását.

```
10 KEY1,"POKE65298,196:POKE65299,208"+CHR$(13)
20 PRINT"□"
30 INPUT"HOL KEZDODIK A KARAKTERKESZLET";X$
40 X=DEC(X$):REM KARAKTERKESZLET KEZDETE
50 C$=LEFT$(X$,2):C=DEC(C$)
60 POKE65298,192:POKE65299,C:REM KARAKTEREK A RAM-BOL
70 INPUT"HANYADIK KARAKTER HELYERE";H
80 PRINT"□"
90 POKE3092,H:REM KARAKTER A KEPERNYORE
100 K=X+H*8:REM A KARAKTER KEZDOCIME
110 REM MEMORIABAIRAS
120 FOR I=0 TO 7
130 PRINTI;"-DIK SOR";:INPUT A
140 POKE K+I,A
150 NEXT I
160 INPUT"AKARSZ MEG EGYET (I/N)";V$
170 IF V$="I" THEN 70
180 END
```

A második program még egyszerűbb, *egy ciklusban olvassa be az értékeket* és helyezi el a memóriában. Csak a 0-dik karaktert írja át.
A 10-es, illetve a második programban a 30-as sor az *F1 funkcióbillentyűt írja át.* Ha hiba miatt a kép „bemákosodik", az F1 billentyű megnyomásával megmenthető a helyzet, ugyanis visszaállítja a TED regiszterek eredeti értékét.

```
10 PRINT"□"
20 POKE65298,192:POKE65299,48
30 KEY1,"POKE65298,196:POKE65299,208"+CHR$(13)
40 K=DEC("3000")
50 POKE 3092,0:REM A '@' JEL KEPERNYOREIRASA
60 FOR I=0 TO 7
70 INPUT A
80 POKE K+I,A
90 NEXT I
```

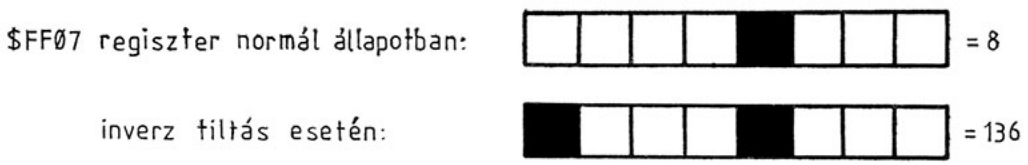
A karakterkészlet-áttöltő gépi kódú rutint, ennek BASIC betöltőjét, vagy a másik rutint, amely a mákosodás ellen nyújt védelmet, csak azoknak a gyerekeknek érdemes megmutatni, akik új karaktereket használó programot szeretnének készíteni önállóan.
A karakterdefiniálás megtanítására szánt idő a témában való elmélyülés mértékétől függ. Egy jól megszervezett kétórás foglalkozásba a teljes feladat elfér. Ha több időt szánunk rá, akkor az utolsó, ötödik lépcsőfok kerülhet feldolgozásra esetleg több foglalkozáson, egészen addig, míg például képesek lesznek (a jobbak) egy cirill betűs készlet elkészítésére.

A mellékletben szereplő „Tájékozódás a koordinátarendszerben" olyan játékos oktatóprogram, amely — a betűk kivételével — újradefiniált karaktereket használ. Mintaként érdemes bizonyos részeit megvizsgálni (például karakterkészlet-áttöltés, új karakterek definiálása, képernyőre írása, illetve a „takarás" megvalósítása, amikor két karakter egymás „fölött" elhalad).
Ha valaki kedvet kapott és további karakterlehetőségek is érdeklik, olvassa el a „Karaktermód-lehetőségek" című fejezetet, ezen belül a többszínű karakterek készítéséről szóló részt.

3. KARAKTERMÓD-LEHETŐSÉGEK

3.1 Normál karaktermód

A képernyőmemória minden egyes bájtja egy-egy karakter megjelenítésére alkalmas. Az ezer bájt mindegyike tulajdonképpen **mutató,** amely a megfelelő bájt helyén megjelenítendő *karakter karaktergenerátor ROM-beli helyére mutat.* A bájtok nyolc bitje közül azonban csak az alsó hét szerepel karaktermutatóként, a nyolcadik bit az inverz mód jelzője. Ha ennek értéke „1", akkor *inverz módban,* ha „Ø", akkor normál módban jelennek meg a karakterek a képernyőn. Mivel a karaktermutató csak hétbites, egyszerre $2^7 = 128$ karakterrel gazdálkodhatunk. Ha az egyik 128-as készletről a másikra akarunk váltani, a C= és SHIFT együttes lenyomásával ezt megtehetjük, de így az egész képernyő egyszerre átvált.
Az inverz mód rovására azonban lehetőség van a teljes, 256 karakter (kis- és nagybetűs) készlet egyszerre való megjelenítésére. Ehhez az egyik **vezérlő regiszter** ($FFØ7) hetes, **inverz jelző bitjét** kell „1"-re váltani. (POKE 65287,136). Átállítás után a képernyőmemória bájtjainak nyolcadik bitjei elvesztik inverzjelző funkciójukat és szintén mutatóként funkcionálnak. Így válik lehetővé egyszerre $2^8 = 256$ féle karakter használata. A C=SHIFT billentyűpár hatástalanná válik, inverz módra váltás esetén (Control-Rvs on) a kisbetűs készlet használható, de közben a nagybetűs készlet is megmarad.

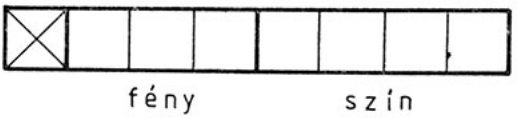


3.1. ábra
Inverz tiltás beállítása

3.2. Bővített háttérszínmód:

A képernyőkép szebb kialakításához lehetőség nyílik többféle háttérszín egy időben való használatára. Az új üzemmód beállításához a másik vezérlő regiszter ($FFØ6) hatos bitjét kell „1"-re állítani (POKE 65286, 91). A bővített háttérszínüzemmód jobb lehetőségeinek ára az, hogy a kurzor eltűnik, a használható karakterek száma pedig 64-re csökken.

A használható karakterek száma csökkenésének oka az, hogy a képernyőmemória bájtjainak felső két bitje ebben az üzemmódban a háttérszín meghatározására szolgál, a megmaradó hat bit mindössze $2^6=64$ lehetőséget kínál.
A hatos és hetes bitek a következőképpen határozzák meg a háttérszínt:
„ØØXXXXXX": **normál háttérszín,** az egész képernyőn egyforma. Ezt a színt az **$FF15 színregiszter** értéke határozza meg.
„Ø1XXXXXX": *SHIFT-tel együtt megnyomott billentyűk* hatására a hatos bit „1"-re vált és a grafikus karakterek helyett újra az első 64 betű jelenik meg **1. háttérszínnel.** Az 1. háttérszínt az **$FF16 színregiszter** értéke határozza meg.
„1ØXXXXXX": *inverz üzemmódra váltva* a hetes bit vált automatikusan „1"-re, de inverz megjelenítés helyett ismét az első 64 betű használható 2. háttérszínnel. A **2. háttérszín** beállításához az **$FF17 színregisztert** kell állítani.
„11XXXXXX": *inverz módban SHIFT-tel megnyomott billentyűk* hatására a hatos és hetes bit is „1"-re vált. Így is csak az első 64 karakter jelenik meg, de **3. háttérszínnel.** Ennek értékét az **$FF18 színregiszter** tartalmazza.
A négy színregiszternek való értékadás egyformán történik. Mindössze annyit kell megjegyeznünk, hogy **a nyolcbites regiszter alsó négy bitje a színt, a következő három bit pedig a fényerőt határozza meg:**



3.2. ábra
Színregiszterek felosztása

A színbeállításhoz rendelkezésre álló négy bit $2^4=16$ féle lehetőséget kínál, tehát tizenhatféle különböző szín közül választhatunk.
A fényerősség-beállítás három bitje $2^3=8$ variációt jelent, így **minden színhez nyolcféle fényesség állítható.**

Színkódok:	Ø	fekete	8	narancs
	1	fehér	9	barna
	2	piros	10	sárgászöld
	3	cianzöld	11	rózsaszín
	4	ibolya	12	kékeszöld
	5	zöld	13	világoskék
	6	kék	14	sötétkék
	7	sárga	15	világoszöld

Állítsuk be például az 1. háttérszínt 5-ös fényességű, 13-as színkódú világoskékre. Ehhez nincs más dolgunk, mint az $FF16 (65302) színregiszterbe a megfelelő értéket beírni POKE utasítással. Az érték kiszámítása egyszerű:

fényerő (5) világoskék (13)

érték = színkód + 16 · fényerő
Jelen esetben:
érték = 13 + 16 · 5 = 93

3.3. ábra
Színregiszter értékének beállítása

Ahhoz tehát, hogy az 1. háttérszín a tervezett módon, világoskéken jelenjen meg, POKE 65302,93 utasítást kell begépelni. Ezután már használhatjuk is ezt a háttérszínt.
Mintaprogram:

```
10 Poke 65286,91 :rem bovitett hatterszin be
20 Poke 65301,113:rem teljes hatter feher
30 Poke 65302,93 :rem 1. hatterszin vilagoskek
40 Poke 65303,66 :rem 2. hatterszin Piros
50 Poke 65304,63 :rem 3. hatterszin zold
60 Print "feher hatter"       :rem normal
70 Print "VILAGOSKEK HATTER":rem shift
80 Print "■Piros hatter■"    :rem rvs on
90 Print "■ZOLD HATTER■"     :rem rvs on+shift
```

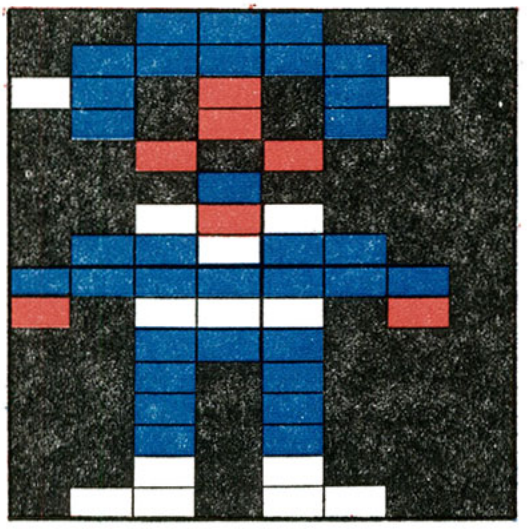
3.3 Többszínű karaktermód

A három lehetséges karakter üzemmód közül talán ez a leglátványosabb, de használatához némileg több előismeretre van szükség az előbbieknél. Mégis érdemes megpróbálkozni vele, mert igazán érdekes képernyőképet szinte csak ezzel lehet elérni. Karakterenként négyféle szín megjelenítésére nyílik lehetőség azon az áron, hogy a vízszintes felbontás a felére csökken.
Fontos tudni, hogy a beépített karaktereket nem a többszínű mód számára tervezték, ezért többszínű megjelenítésükre nincs lehetőség. Kénytelenek leszünk tehát többszínű megjelenítésre szánt karaktereinket magunk megtervezni. Nézzük először a felbontás csökkenéséből adódó problémát. A nagy „A" betű képe például ilyen a karaktergenerátor ROM-ban:

3.4. ábra
Az „A" betű képe a memóriában

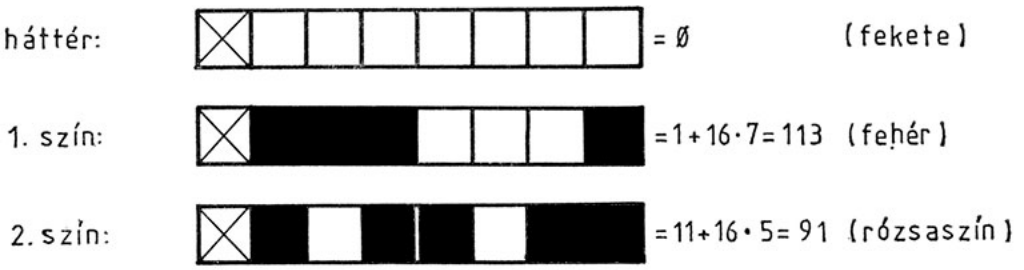
A 8×8-as négyzetben minden bit önmagában, egyedül hordoz információt: ha a bit értéke „0", akkor a neki megfelelő pont háttérszínű lesz a képernyőn, ha „1", akkor írásszínű. Mivel egy bitérték más nem lehet csak „0" vagy „1", ezért normál karaktermódban egy képernyőpozícióban csak kétféle szín jelenhet meg, háttér- és írásszín. Ha a színezési lehetőségeinket 4 féle színre szeretnénk bővíteni, akkor 4 féle állapotot kell tudnunk megvalósítani. Lehetőségként kínálkozik, hogy a biteket ne egyesével tekintsük, hanem páronként, **egy bitpár értéke ugyanis éppen 4 féle lehet: ØØ, Ø1, 1Ø, 11.** Ha mindegyik bitpár típushoz egy-egy színt rendelünk, akkor máris négy színnel gazdálkodhatunk.

Csakhogy ilyen kétbites „téglácskákból" építkezve karaktereinkben a rendelkezésre álló 8×8-as rácson egymás mellé mindössze 4 „tégla" fér el, tehát az eredeti 8×8-as finomfelbontás helyett mindössze 4×8-assal gazdálkodhatunk. Ezt a minőségromlást úgy ellensúlyozhatjuk, hogy saját tervezésű figuráinkat nem egy, hanem legalább négy karakter nagyságúra készítjük. Példánkban is egy ilyen szerepel.



3.5. ábra
Saját tervezésű többszínű karakternégyes

Jelenítsük meg ezt a figurát a képernyőn! A rajzon látható, hogy egyszerre 4 karakter helyét foglalja el, tehát négy karaktert kell majd újradefiniálnunk. A lehetséges 4 szín mindegyike megjelenik mind a négy pozícióban, tehát más lehetőségünk nincs mint a többszínű üzemmód.
Felhasznált színeink a következők:
— **Háttérszín:** fekete. Azok a bitpárok lesznek feketék, amelyek értéke „ØØ".
— **1. szín:** fehér. Azok a bitpárok fehérek, amelyek értéke „Ø1".
— **2. szín:** rózsaszín. Azok a bitpárok lesznek rózsaszínek, amelyek értéke „1Ø".
— **3. szín:** kék. Az „11" értékű bitpárok lesznek ilyen színűek.
A háttérszínt ($FF15=65301), az 1-es és 2-es színregisztereket ($FF16=65302, $FF17=65303) a bővített háttérszínmódnál megismert szabályok alapján kell beállítani: Ø—3 bit szín (16 féle), 4—6 bit fényerő (8 féle).



3.6. ábra
A színregiszterek beállítása

A 3. színt nem színregiszter hordozza, hanem az úgynevezett **„attributumbájt".** Ennek beállítása előtt szükséges egy rövid kitérőt tenni.
A *képernyőmemória* 1ØØØ bájt hosszúságú, 3Ø72-től 4Ø72-ig. Mindegyik bájt a képernyő egy-egy pozícióját jelenti. Mindegyik memóriacímhez hozzárendelődik egy — az előbb említett — **attributumbájt,** amely **meghatározza a megjelenített karakter színét.** (Normál üzemmódban is így van.) Az attributumbájtok alkotják a *színmemóriát,* amely természetesen szintén 1ØØØ bájt hosszúságú. Minden képernyőpozíció alatt 1Ø24 bájttal található a hozzárendelt színt hordozó attributumbájt. Például a képernyő bal felső sarkát jelentő 3Ø72 cím színét a 3Ø72—1Ø24=2Ø48 címen levő érték határozza meg.
Többszínű karaktermódban a **3. színt,** az „11" bitpár színét **határozza meg az attributumbájt,** ezért ez a szín bármely pozíción más lehet. A 3. szín meghatározásán kívül van még egy nagyon fontos szerepe ennek a bájtnak. Bekapcsolt többszínű üzemmód esetén az attributumbájt 3-as (alulról a negyedik) bitje határozza meg, hogy a neki megfelelő karakter több színben jelenjen meg, vagy nem. Ha értéke „1", akkor többszínű, ha „Ø" akkor normál, egyszínű karakter jelenik meg a helyén. Ez a bit teszi lehetővé, hogy a képernyőn egyszerre jelenhessenek meg többszínű figurák és az eredeti betűk.

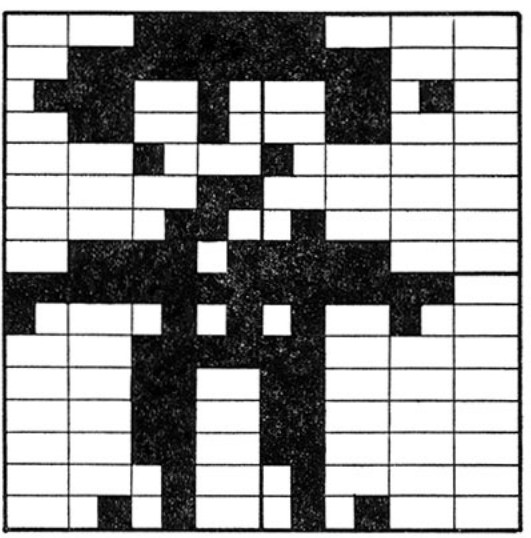
Mivel az attributumbájt 3-as bitje ilyen felhasználást nyert, a színmeghatározásra csak 3 bit maradt (Ø—2), így $2^3=8$ szín beállítására van lehetőség. A fényerőt a színregiszterek mintájára a 4—6 bithármas határozza meg.
Példánkban szereplő 3. szín beállítása tehát:

3.7. ábra
Az attributumbájt beállítása

A 3. színnek megfelelő érték meghatározása után már csak arra kell nagyon gondosan ügyelni a felhasználás során, hogy a színt hordozó attributumbájt mindig együtt mozogjon a karakterrel (1Ø24 bájttal lejjebb), máskülönben nagy gonddal készült karaktereinket nem fogjuk színesen látni. Egy kérdés maradt hátra, hogyan helyezzük el karaktereinket a memóriában?
Először kódoljuk a képet:

Háttérszínű — jelen esetben fekete — téglalapocskáknak a „ØØ" bitpár felel meg.
1. színű — most fehér — téglácskákat a „Ø1" bitpár határozza meg.
2. színű — rózsaszín — téglácskáknak az „1Ø" bitpárok felelnek meg.
3. színű — kék — téglalapocskákat az „11" bitpárok határozzák meg.

Ennek alapján a négy karakter kettes számrendszerben:



3.8. ábra
Karakternégyesünk kettes számrendszerben

Számítsuk át ezeket az értékeket tízes számrendszerbe:

```
00001111= 15
00111111= 63
01110010= 114
00110010= 50
00001000= 8
00000011= 3
00000110= 6
00111101= 61

11000000= 192
11110000= 240
00110100= 52
00110000= 48
10000000= 128
00000000= 0
01000000= 64
11110000= 240

11111111= 255
10000101= 133
00001111= 15
00001100= 12
00001100= 12
00001100= 12
00000100= 4
00010100= 20

11111100= 252
01001000= 72
11000000= 192
11000000= 192
11000000= 192
11000000= 192
01000000= 64
01010000= 80
```

3.9. ábra
Átszámítás tízes számrendszerbe

Ezeket az értékeket kell majd a karaktermemória megfelelő bájtjai helyére írni POKE utasítással, miután a karaktereket a ROM-ból a RAM-ba másoltuk.
Befejezésül álljon itt egy BASIC program mintaként, amely az imént megtervezett karakternégyest elhelyezi a képernyő közepén.
Előzőleg másoljuk át a karaktereket:

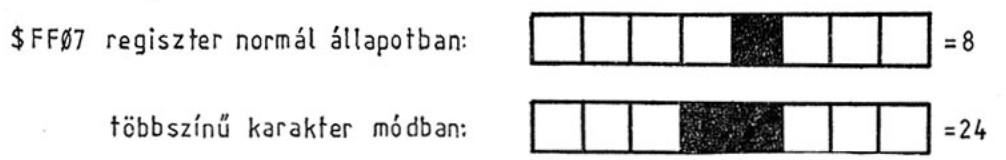
```
MONITOR              ; áttérés monitorba
T D000 D7FF 3000     ; karakterkészlet átmásolása $3000 kezdőcímmel
X                    ; visszatérés BASIC-be
```

És most a BASIC program:

```
1 DATA 15,63,114,50,8,3,6,61
2 DATA 192,240,52,48,128,0,64,240
3 DATA 255,133,15,12,12,12,4,20
4 DATA 252,72,192,192,192,192,64,80
5 COLOR4,1:COLOR0,1:COLOR1,2
6 PRINT"□"
10 FOR I=0 TO 31
20 READ A:POKE12800+I,A
30 NEXT I
40 POKE65298,192:POKE65299,48:REM KARAKTEREK A RAM-BOL
50 POKE65287,24:REM TOBBSZINU MOD BE
60 REM SZINEK BEALLITASA
70 POKE65301,0:POKE65302,113:POKE65303,91
80 REM KARAKTEREK A KEPERNYORE
90 H=3492:SZ=H-1024
100 POKEH,64:POKESZ,62
110 POKEH+1,65:POKESZ+1,62
120 POKEH+40,66:POKESZ+40,62
130 POKEH+41,67:POKESZ+41,62
140 GETKEYA$
150 POKE65287,8:REM TOBBSZINU MOD KI

READY.
```

A program 50-es sorában történik a többszínű karakterüzemmód bekapcsolása, a 150-esben pedig kikapcsolása. Látható, hogy ennek a módnak is az $FF07=65287 című vezérlő regiszter a felelőse úgy, ahogy az inverz üzemmódé is. A regiszter 4-es bitje a **Multi Color Mode** *(többszínű üzemmód)* jelzője. Ha értéke „1", akkor a többszínű mód be van kapcsolva. Ilyenkor a kurzor eltűnik, mint a bővített háttérszínmódnál is tapasztalható volt.



3.10. ábra
Többszínű karakter mód beállítása

3.3.1. *Iskolai feldolgozás*

A bővített háttérszínmód, illetve a többszínű karaktermód tanításához — hacsak a tanulócsoport ezt nem igényli — nem érdemes a dolog hardver hátterére külön kitérni. Elég, ha ebből annyit tanulnak meg a gyerekek, ami a sikerélményig elvezető munkához szükséges. Konkrétan a színbeállításokról van szó, ami nem valósítható meg egy kevés ilyen irányú előismeret nélkül.

3.3.1.1. Bővített háttérszínmód

Bővített háttérszínmód tárgyalására elég egy órát szánni. Bevezetésül mutassuk meg példán a lehetőségeket. Nem árt szót ejteni a várható kényelmetlenségekről (nincs kurzor, csak az első 64 karakter használható). A tárgyalást érdemes rögtön egy próbával kezdeni: *állítsuk át a vezérlőregiszter értékét* (POKE 65286,91) és *írassunk a képernyőre szöveget inverz módra váltva, vagy SHIFT gombbal.* Különböző háttérszínű szövegeket láthatunk (anélkül, hogy előre beállítottuk volna őket).
Következő lépcsőfok a színregiszterek beállítása lehet a gyerekek saját ízlése szerint. A használható négyféle háttérszínhez tartozó négy színregiszter értékének beállításához **a regiszterek bitjeinek szerepére is kitérhetünk** (alsó négy bit: szín, következő három bit: fényerő). Egyszerűsíthető a probléma a regiszterbe POKE utasítással írandó érték kiszámítására szolgáló képlet ismertetésével *(érték=színkód+16*fényerő).*

3.3.1.2. Többszínű karaktermód

Többszínű karaktermód tárgyalását mindenképpen meg kell előznie a karakterújradefiniálás megismerésének. Ha ugyanis képesek a gyerekek saját karaktereket tervezni és létrehozni, akkor itt nem ütközhetnek igazi problémába. Ha igazán biztosak szeretnénk lenni a sikerben, érdemes előbb a bővített háttérszínmóddal is megismerkedni. A *színregiszterek állítása* ugyanis ott már előfordul, csak éppen sokkal egyszerűbb.
Ha a gyerekek készségszinten elsajátították a *karakterdefiniálás,* valamint a *színregiszterátírás* technikáját, a következő feldolgozási mód javasolható:
Először négyzetrácsos papíron rajzoljuk meg a 8×8-as alapot (lehet egyszerre négy karakternyi figurát is tervezni, mint a példabeli is). Utána *osszuk fel a* bemutatott módon *kétrácsnyi téglalapokra!* Itt kell kitérni a bitpárok színmeghatározó szerepére. *Tervezzék meg a gyerekek saját figurájukat.* Hívjuk fel a figyelmet, hogy négyféle szín használható és egy téglalapocska csak egyféle színű lehet. Lehetőleg először mindenki ugyanazt a négy színt használja a későbbi keveredés elkerülése érdekében. Ha a tervek elkészültek, rendeljünk minden színhez egy-egy bitpárt a lehetséges négy közül (a háttér a „ØØ" legyen!).
Ezután jön a legtöbb figyelmet igénylő rész, egy újabb kockás lapon *váltsuk át a színes téglalapokat bitpárokká.* Így a tervezett figura egy kicsit formátlanná válik.
A további teendők már ismerősek lesznek a gyerekek számára: *Írjuk át a kettes számrendszerbeli nyolcbites sorokat tízes számrendszerbelivé* (segít itt is a helyiértékek jelölése), majd egy ügyes ciklussal olvassuk be rendre az értékeket az előzőleg RAM területre mentett karakterkészletbe.
Egy nagy probléma van még hátra, *a karakterek elhelyezése a képernyőn.* Csak POKE utasítással közvetlenül a képernyőmemóriába írt karakterek jelenhetnek meg többszínűen, ugyanis a színmemóriába a karakterrel párhuzamosan elhelyezett „attributumbájt"-tal együtt működik az eljárás. A képernyőmemória 3Ø72-től 4Ø71-ig tart.

Az egyes karakterpozícióknak megfelelő színmemória helyek 1024 bájttal vannak lejjebb. Ha tehát karakterhelynek kiválasztjuk a képernyő közepén levő 3492 című bájtot, akkor a hozzá tartozó „attributumbájt" címe 3492—1024=2468 lesz. **Az attributumbájt értékének meghatározásakor arra kell feltétlenül ügyelni, hogy a bájt hármas bitje** (alulról a negyedik) **„1" értékű legyen,** ez a bit felelős ugyanis a megfelelő pozíció többszínű megjelenítéséért. (Beállítása máskülönben megegyezik a színregiszterekével, annyi különbséggel, hogy itt csak az alsó három bit határozza meg a színt.)

A leírásban szereplő rövid mintaprogram segítségül szolgálhat az első próbálkozásokhoz, a mellékelt kazetta programja pedig némi bővítési lehetőséghez adhat öltetet: a karakteranimációhoz, amit célszerű először egyszínű karakterekkel kikísérletezni.

Programok leírása

Torpedo. Egyszerű játékprogram, melyben a képernyőn átvitorlázó hajót kell a lövés koordinátáinak megadásával eltalálni. A koordináták megegyeznek a finomfelbontású képernyő koordinátáival, tehát a program gyakorlási lehetőséget nyújt a képernyőn való tájékozódás gyakorlására.

Óra. Bemutatóprogram, mely egy mutatós órát szimulál. Az óra és perc megadása után a beállított értéktől méri az időt.

Ugribugri. A többszínű grafikus mód bemutatására szánt program. A ciklusváltozó és egy tetszőleges számhármas beírása után a gép egy érdekes mintát rajzol a képernyőre három színnel. A kész képen a 3-as (eredetileg piros) szín a kurzormozgató billentyűkkel tetszőlegesen változtatható.

Kockaforgató. A Néhány grafikai fogás című részhez kapcsolódó bemutatóprogram. A kiválasztott tengely körül 90 fokkal elforgat egy térbeli kockát.

Függvénytranszformáció. Az általános iskola 8.-os, hasonló című anyagrészéhez nyújt segítséget. A tanult négy alapfüggvény tetszőleges transzformációi jeleníthetők meg rugalmasan változtatható koordinátarendszerben. A funkcióbillentyűk jelentése:

F1 — új függvény, új koordinátarendszerben.
F2 — új függvény az előző koordinátarendszerben.
F3 — az előző függvény az előző koordinátarendszerben (természetesen új transzformáció lehetőségével).
HELP — egy jól használható koordinátarendszer adatai.

Koordinátarendszer. Játékos oktatóprogram a koordinátarendszerben való tájékozódás gyakorlására. A cél a kulcsok összeszedése a lehető legrövidebb időn belül. Lépni csak vízszintesen, vagy függőlegesen, a koordináták megadásával lehet.

Tájékozódás az erdőben. 5. osztályban a tizedes törtekről tanultak önálló gyakorlására szolgáló játékos oktatóprogram. Az erdőben szétszórt három feladatot kell sorban összeszedni. Az összes feladat megoldása után — amit a gép pontoz — a bal felső sarokban levő nyílra eljutva értékeléssel fejeződik be a program. Az erdőben vigyázni kell a rejtett, láthatatlan csapdákra, ha belelépünk, egy életünkbe kerül. Ha a továbbjutás lehetőségét a fák kikerülhetetlenül elzárják, az „F" billentyűvel mód van a tőlünk balra eső fa kivágására egy élet elvesztése árán. A játék joystickkel is játszható.

Cirill abc. A SHIFT-tel lenyomott billentyűk a cirill abc betűit adják. A program SYS 15281-gyel indul, az új karakterkészlet helye: $3C00—$3FFF.

Karakterkészlet 2. A SHIFT és C= gombokkal lenyomott billentyűk új, játék- illetve oktatóprogramok írásához jól használható karaktereket adnak (hatványjelek, néhány görög betű, gót betűkészlet, játékfigurák stb.). Az új karakterkészlet helye: $3C00—$3FFF. SYS 15281-gyel indul.

Karakterszerkesztő 1. Lehetőséget ad saját karakterkészlet előállítására. A kész karakterkészlet elmenthető és később bármikor felhasználható.

Multicolor Demo. A többszínű karaktermód lehetőségét mutatja be.

Karakterszerkesztő 2. Lehetőséget biztosít többszínű karakterek készítéséhez. A színek és az újradefiniálandó karakternégyes kiválasztása után a funkcióbillentyűkkel megtervezett figura a választott karakterek helyére kerül a memóriában. A betűket utoljára, vagy ha lehet, egyáltalán ne definiáljuk át, mert olvashatatlanná válnak a feliratok. A kész karakterkészlet itt is elmenthető.

Mindegyik program „load" utasítással tölthető be és RUN parancsra indul.

TÁRGYMUTATÓ

88-1451 — Szegedi Nyomda
Felelős vezető: Surányi Tibor igazgató

A VORKER Kisszövetkezet ajánlja a Tisztelt Megrendelők figyelmébe az alábbi szoftver termékeket és — szolgáltatásokat.

1. C 16, C Plus/4, C 64 és Spectrum gépekre kifejlesztett oktató és játékprogramokat, amelyek Magyarországon a legolcsóbbak. Ezek közül az idegen és magyar nyelv, a matematika, a kémia, a biológia és a fizika tantárgyi oktatóprogramok segítik a pedagógusok és a tanulók munkáját.

2. Egy- és kétszemélyes játékok, valamint totóprogramok segítik a szabadidő jobb eltöltését.

3. IBM kompatibilis gépekre vállaljuk a munkaügyi és bérszámfejtő rendszerünk adaptálását.

Különleges hardver ajánlatunk: a világon egyedülálló fejlesztésünk a TC-NET+4 számítógép-interfész, amellyel 16 db C 16 illetve C Plus/4 számítógépet kapcsolhatunk lokális hálózatba. Ez két fontos célt szolgál: egyrészt az összes gép ilyen módon használhat egyetlen mágneslemezt és nyomtató egységet, másrészt megvalósítja számítástechnikai vonalon azokat az előnyöket, amelyeket a nyelvoktatás területén egy nyelvi labor biztosít. Ily módon számítástechnikai kabinet hozható létre kis anyagi ráfordítással.

Várjuk érdeklődésüket!

VORKER® Vállalatközi
Organizációs Ipari Szolgáltató
és Kereskedelmi Kisszövetkezet
Szeged, Pf: 711.
Telex: 82-688
Telefon: (62) 26-144
(62) 25-479