PEDAGÓGIAI MŰHELY

11.

Gudenus László

BEVEZETÉS A COMMODORE PLUS/4-ES GÉPI KÓDÚ PROGRAMOZÁSÁBA I.

PEST MEGYEI PEDAGÓGIAI INTÉZET

1989

Gudenus László

BEVEZETÉS A COMMODORE PLUS/4-ES GÉPI KÓDÚ PROGRAMOZÁSÁBA

Pest Megyei Pedagógiai Intézet

1989

Lektorálta:
S o k Z o l t á n

ISSN 0238 3217
ISBN 963 01 9696 4
I. 963 01 9697 2

Kiadja a Pest Megyei Pedagógiai Intézet
Felelős kiadó: Dr.Szivi József igazgató
Felelős szerkesztő: Pap János
Készült a Megyei Pedagógiai Intézet sokszorosításában
1989-ben 2000 példányban
Felelős vezető: Márkus István
Munkaszán: /1989. Budapest

T A R T A L O M J E G Y Z É K

Bevezetés 2

1. Előismeretek 4

2. A processzor 7

3. Irjunk a képernyőre! 12

4. Aritmetika 25

5. Csillagok, villogás 37

6. Számoljunk tovább! 48

7. Mi mindenre használhatjuk? 63

Befejezés 73

Utasítástáblázat 74

Irodalom 77

ICL - segédlet 78

BEVEZETES

Az általános iskolákban fellelhető számítógépek között kétségkívül a Commodore+4-es a leggyakoribb. Ez a gép - sok más kiváló tulajdonsága mellett - rendelkezik egy, a ROM-ban tárolt gépi kódú monitorral, amelyet előbb-utóbb mindenki kipróbál, s ezzel megteszi az első lépéseket a gép belső felépítésének, a gépi kódú programozás elemeinek megismerése felé. Hiába hajt azonban sokakat a tudásvágy, keveseknek sikerül számottevő eredményt elérni - s ez tanárokra-diákokra egyaránt érvényes - : néhány sikertelen kísérlet után, a gép látszólag érthetetlen viselkedése, a segédeszközök, az értő társ hiánya miatt általában hamar felhagynak a próbálkozással, s felfoghatatlannak, megtanulhatatlannak tartva, örökre búcsút mondanak a gépi kódú programozásnak. Ennek ellenére számos példa bizonyítja, hogy már az általános iskolások között is jónéhány profi művelője akad ennek a területnek.

A gépi kódú programozással való ismerkedéshez kiválóan alkalmasak a Commodore-gépek processzorai (amelyek egyébként programozás szempontjából tökéletesen egyenrangúak): egyszerű felépítés, kisszámú utasítás, bőséges címzésmód jellemzi őket.Az utasítások jelentését, a jelzőbitekre gyakorolt hatását a függelékben találhatjuk; a további szükséges információkat (rendszerváltozók, TED-regiszterek, ROM-lista stb.) - a gép belső felépítésének pontos és részletes ismertetésével együtt - tartalmazzák azok a könyvek is, amelyeket az irodalomjegyzékben felsoroltunk, így ebben a füzetben csak a legszükségesebb tudnivalókat ismételjük meg.

A gépi kódú programozásra különösen igaz, hogy elsajátításának leghatékonyabb, mondhatni kizárólagos útja a feladatmegoldás, a gyakorlás. Ehhez kívánunk segítséget nyújtani. A gépi kódú programozással foglalkozó könyvek általában azt a felépítési módot választják, hogy felsorolják az összes utasítást, a jelzőbitekre gyakorolt hatásukkal, a lehetséges címzésmódokkal stb. együtt, említenek esetleg 1-2 példát ezekre, majd magára hagyva a tanulni vágyót, rögtön nagyobb lélegzetű feladatra térnek át. Ez kétségkívül korrekt tárgyalási mód, viszont figyelmen kívül hagyja, hogy egy, az átlagosnál jóval nehezebben elsajátítható témáról van szó. Aki az ehhez vezető, akadályokkal zsúfolt úton halad, az lépten-nyomon nehézségekbe ütközik, s ha menet közben nem jut elegendő sikerélményhez, bizony nagyon könnyen kedvét veszítheti, s hamarosan visszafordul. Mi tehát nem ezt az utat követjük, hanem feladatokon keresztül mutatjuk be nem csupán az utasításokat, hanem a gépi kódú programozásra leginkább jellemző sajátosságokat is, de nem törekszünk - mert nem is törekedhetünk - ebben a füzetben teljességre. Alig néhány utasítással már rövid, de működő programokat készítünk, s ezt az utasításkört fokozatosan bővítve, egyre nagyobb és nagyobb feladatokat oldunk meg. Nem leszünk

tekintettel tehát sem az utasítások szokásos értelemben vett osztályozására, sem pedig a feladatok tematikus csoportosítására.

A programok írásához, fordításához az elterjedt JCL assembler programot használtuk; ennek leírása pld. [10]-ben fellelhető. A feladatok megoldásához néhány jótanács: soha ne induljunk el úgy, hogy rögtön a gépbe pötyögjük éppen megszületó gondolatainkat, fordítjuk, majd futtatjuk; ez gépi kódó programozásnál különösen veszélyes: akár egész munkánk is kárba veszhet. Először mindig tervezzük meg a programot, fogalmazzuk meg az algoritmust (ehhez kiváló segítséget nyújt a pld. [6]-ban ismertetett algoritmusleíró nyelv), majd ezt követi a kódolás, a szintaktikus ellenőrzés, a mentés (lemezre vagy kazettára), és csak ezután jöhet a fordítás, majd a belövés.

Tudjuk, hogy az azonos feladatot végző BASIC ill. gépi kódó program közül ez utóbbi sokszorta gyorsabb és lényegesen kevesebb tárat igényel. Mégsem javasoljuk, hogy mindent gépi kódban próbáljunk megírni, mert sokkal nehezebb és fáradságosabb munka vár ránk. Csak akkor nyúljunk ehhez az eszközhöz, ha feltétlenül szükséges, azaz valami különleges szempont (gyorsaság, pontos időzítés, a gép belsejéhez való hozzáférés stb.) miatt a BASIC már alkalmatlan. Feladataink nem mindig felelnek meg ennek a követelménynek, mert jó részüket BASIC-ből is meg tudnánk oldani, mégis érdemes foglalkozni velük, nemcsak a gépi kódó program sebességének érzékelése, hanem a gépi kódó programozás technikájának elsajátítása szempontjából is.

A megoldások kiválasztásánál általában a tanulás segítésére törekedtünk, nem pedig arra, hogy a feladatra (valamilyen értelemben) optimális megoldást közöljünk. Mivel a gépi kódó programozás egyik jellemzője, hogy ugyanazt a feladatot sokféleképpen is megoldhatjuk, mindig próbáljunk más, jobb (pld. egyszerűbb, kisebb helyet elfoglaló vagy még gyorsabb) megoldást adni.

1. ELŐISMERETEK

Feltétlenül szükség van bizonyos előismeretekre a feladatok megértéséhez, megoldásához. Ezek elsősorban a következőket jelentik:

- az alapvető programozási struktúrák (elágazás, ciklus) ismeretét, a programozásban szerzett jártasságot (lehetőleg a C+4 BASIC nyelvén),
- a gép memóriájához való hozzáférés (PEEK-POKE) lehetőségeit,
- a bináris és a hexadecimális számrendszerek ismeretét,
- a beépített monitor parancsainak használatát.

Ezeknek az ismereteknek a megszerzéséhez, bővítéséhez az irodalomjegyzékben felsorolt művek jó szolgálatot tehetnek. Az ellenőrzéshez néhány feladattal mi is segítséget nyújtunk:

1.1 A következőkben a decimális számokat külön nem jelezzük, a bináris számokat mindig %-jellel, a hexadecimális számokat pedig mindig $-jellel kezdjük. Alakítsuk át az alábbi számokat mindkét másik számrendszerbe:

12	17	164
274	255	1989
%1	%1001	%1011110
%10000000	%10101010	%11111111
$01	$10	$12
$1c	$76	$ff
$123	$1234	$1989
$0c00	$ff4f	$ffff

Figyeljük meg, mennyire egyszerű a kettes és a tizenhatos számrendszer közötti átváltás!

1.2 Végezzük el a következő műveleteket (a számok decimálissá alakítása nélkül):

%0+%0=	%0+%1=	%1+%1=
%11+%1=	%111+%1=	%101+%1=
%1010+%1001=	%1011+%1001=	%11-%1=
%100-%1=	%101-%11=	%1010-%111=
$6+$4=	$6+$6=	$a+$b=
$75+$67=	$9a+$6c=	$fa+$6=
$100-$1=	$100-$ac=	$121-$f8=
$f0+24=	200-$d7=	$ff+1=
$ff+%1=	12-%1000111=	%11110000-$ad=

1.3 Határozzuk meg a következő bináris számok kétszeresét (szintén bináris formában):

%10 %1001 %10111011
%10101010

1.4 Vizsgáljuk meg, hogy a 3. feladatban kapott eredmények milyen összefüggést mutatnak az eredeti értékekkel! Mit tudunk mondani az eredményként kapott bináris számok 'hosszáról'? A számok tárolásához egy-egy (8 bites) byte-ot használunk. El fog-e mindegyik szám kétszerese is férni egy byte-on? Mit kell tennünk, hogy ne veszítsünk információt?

1.5 Határozzuk meg a következő bináris számok felét (szintén bináris formában):

%10 %1001 %10111011
%10101010

1.6 Vizsgáljuk meg, hogy az 5. feladatban kapott eredmények milyen összefüggést mutatnak az eredeti értékekkel! Mit tudunk mondani az eredményként kapott bináris számok 'hosszáról'? A számok tárolásához egy-egy (8 bites) byte-ot használunk. El fog-e mindegyik szám fele is férni egy byte-on? Mit kell tennünk, hogy ne veszítsünk információt?

1.7 A képernyőtár kezdete $0c00, az 'a' betű képernyőkódja pedig 1. Milyen BASIC-paranccsal tudjuk közvetlenül a képernyő bal felső sarkába írni az 'a' betűt? (Vigyázzunk a kisbetű-nagybetű üzemmódra!)

1.8 Írjunk olyan BASIC-programot, amely az előző feladat mintájára teleírja a képernyőt *-gal!

1.9 Írjunk olyan BASIC-programot, amely mindenfajta átalakítás elvégzésére alkalmas a bináris, a decimális és a hexadecimális számrendszer között! Segítségével ellenőrizzük az 1. feladatra adott megoldásainkat!

1.10 A memória 198-as című byte-ja 64-et mutat, ha nem nyomunk a billentyűzethez, viszont (a Ctrl, a shift és a control kivételével) minden billentyű lenyomására más és más értéket. Írjunk olyan BASIC-programot, amellyel kideríthető, hogy melyik gomb lenyomására mennyi ez az érték! (Célszerű egy, a feladatot végző végtelen ciklust futtatni, miközben nyomogatjuk az egyes billentyűket.)

1.11 Írjunk olyan BASIC-programot, amely folyamatosan a képernyőre írja a $a3-$a4-$a5 byte-ok tartalmát (itt helyezkedik el a gép belső órája; a leggyorsabban az $a5-ös byte jár). A program írja ki a TI rendszerváltozó mindenkori tartalmát is! Vessük össze a két értéket! Mit tapasztalunk, ha a $ff00-$ff01 számláló értékeit próbáljuk kiíratni?

1.12 Mi a 64 kilobyte-nyi tárterület legfelső byte-jának hexadecimális címe? Hány bit kell ennek bináris ábrázolásához?

A MONITOR parancs hatására a gép ROM-jában lévő TEDMON monitor szolgáltatásait vehetjük igénybe. Közülük a C, F, H, L, M, S, T, V és X funkciókat még a gépi kódú utasítások ismerete nélkül is használhatjuk. Segítségükkel oldjuk meg az alábbi feladatokat:

1.13 Töltsük fel a képernyő felső sorát a '?'-jellel!

1.14 Töltsük fel az egész képernyőt a '!'-jellel!

1.15 Váltsuk át a képernyőn látható összes karakter színét zöldre (a színmemória $0800-n kezdődik)! A színt a színmemória megfelelő byte-jának 0.-3. (hátsó 4) bitje szabályozza.

1.16 Váltsuk át a képernyőn látható összes karaktert a lehető legfényesebbre! A fényességet a színmemória megfelelő byte-jának 4.-6. bitje szabályozza.

1.17 Váltsuk át a képernyőn látható összes karaktert villogóra! A villogást a színmemória megfelelő byte-jának 7. (legfelső) bitje szabályozza.

1.18 Keressük meg, hol helyezkednek el a tárban a hibaüzenetek!

1.19 Írjuk ki a képernyőre a tár $8000-től kezdődő $300 byte-ját a I parancs segítségével! Írjuk át a 'COMMODORE' feliratot 'DECIMALIS'-ra, majd a C parancs segítségével határozzuk meg a képernyőtárban az átírt hely címét!

1.20 Van-e a gép ROM-jában olyan hely, amelyre a monogrammunkat égették be?

1.21 Írjuk ki a képernyőre az M parancs segítségével a ROM $818e-től kezdődő néhány byte-ját! Itt a BASIC-kulcsszavakat találjuk, de nincs közöttük elválasztójel. Miről ismerhető fel mégis, hogy melyiküktől-meddig tart?

A fenti másoló, feltöltő, összehasonlító stb. funkciók végrehajtása nagyon jól érzékelteti, hogy a gépi kódban megírt TEDMON nagyságrendekkel gyorsabb, mint a BASIC-ben írt programok. Az előismeretek már lényegében elegendők, kezdjünk tehát hozzá a processzorral való ismerkedéshez!

2. A PROCESSZOR

Számítógépünk 'lelke' a mikroprocesszor: sok más feladata mellett ő bonyolítja le az adatforgalmat a tár egyes részei között, végzi a logikai ill. számítási műveleteket s ezek eredményétől, továbbá a program előírásaitól függően vezérli a program futását. A Commodore+4 7501-es ill. 8501-es processzorral készült; ezek egymással s a cég korábbi termékeivel (a 6502-es ill. a 6510-es processzorral) programozás szempontjából pontosan megegyeznek. Mivel a gépi kódú programozás nem más, mint a processzor programozása, feltétlenül szükséges a processzor regisztereinek alapos ismerete.

A processzor regiszterei 8 bitesek, csupán egy 16 bites regisztere van, a programmutató (PC). Mint tudjuk, a program a tárban helyezkedik el, s az éppen soron következő utasításra (amely az 1.12 feladat eredménye alapján a 64 kilobyte-os címtartomány miatt éppen 16 bit segítségével határozható meg) mutat ez a bizonyos regiszter. Tartalma egy-egy utasítás végrehajtása közben általában az utasítás hosszával növekszik meg, s így automatikusan a következő utasítás kezdetére mutat, de általa oldható meg a program másutt való folytatása is: ilyenkor egyszerűen a kívánt címet töltjük a PC regiszterbe, s a program ettől a címtől kezdve folytatódik. A TEDMON-ba lépve rögtön láthatjuk (de az R parancs segítségével később is bármikor lekérdezhetjük) a PC regiszter tartalmát.

A 8 bites regiszterek közül a leggyakrabban az akkumulátorral (AC) fogunk találkozni. Sokszor használjuk majd a két ún. indexregisztert is (jelük: XR ill. YR), amelyeknek közel azonos szerepük van.

A veremmutató (SP) szerepének megértéséhez szólnunk kell a veremről, amely (többek között) a szubrutinok hívásakor azok visszatérési címét tartalmazza. Képzeljük el, hogy meghívunk egy szubrutint, abból egy másik szubrutint (ezt esetleg többször is megismételve - ezeket hívjuk egyébként egymásba ágyazott szubrutinoknak). Az első hívásakor a visszatérési címet meg kell jegyeznünk, de a visszatérés előtt hívjuk a másodikat, természetesen ennek a visszatérési címét is el kell tennünk. A másodjára hívott szubrutin befejeztével felhasználjuk az ő visszatérési címét; ez után folytatjuk, majd befejezzük az elsőként hívottat, s ekkor kell csak ennek a visszatérési címe. Általánosan is igaz, hogy egymásba ágyazott szubrutinok hívásakor az utolsóként elraktározott visszatérési címre lesz először szükségünk, s ezt a tárolási módot a legegyszerűbben a verem segítségével valósíthatjuk meg. A verem a memória 1. lapján, a $100...$1ff címeken helyezkedik el, s csak a $100 felőli végén férhetünk hozzá; mindig csak egyesével tehetjük bele ill. vehetjük ki belőle az elemeket. Az SP regiszter mindenkor azt a címet mutatja,

ahová berakáskor a következő elem kerül. Mivel a veremben az összes hely címe $1-gyel kezdődik, felösleges volna ezt is tárolni, ezért csak a cím alsó 8 bitjét mutatja az SP. Tehát például: ha SP=$f0, akkor tudjuk, hogy a verem $1f1...$1ff része foglalt, az első üres hely címe, ahová adatot tehetünk, $1f0; ha pedig (pld. egy szubrutin befejezésekor) a veremből egy adatot kell kivennünk, azt most a $1f1 címen találjuk. Ismételten felhívjuk a figyelmet arra, hogy a vermet nem csupán a szubrutinok visszatérési címének, hanem tetszőleges adatoknak a tárolására használjuk.

Már csak egyetlen, de nagyon fontos regiszterről kell beszélnünk: a feltételregiszterről (állapotregiszter, SR). Ez a legutóbb végrehajtott utasítás eredményéről szolgáltat bizonyos információkat, mivel az utasítások zöme állítja az SR bitjeit (ezeket jelzőbiteknek is nevezzük). A jelzőbitek sorban a következők:

a bit sorszáma:	7	6	5	4	3	2	1	0
a jelzőbit neve:	N	V		B	D	I	Z	C

Jelentésük:

N NEGATIV jelző: értéke akkor 1, ha a művelet eredményének 7. bitje 1.

V OVERFLOW (túlcsordulás) jelző: értéke 1, ha a művelet eredményeképpen átvitel történt a 6. bitről a 7. bitre.

B BREAK jelző: értéke 1, ha a bekövetkezett megszakítást nem a hardware, hanem a program hozta létre.

D DECIMÁLIS jelző: értéke 1, ha a számítási műveletet nem 16-os, hanem 10-es számrendszerben akarjuk elvégezni.

I INTERRUPT (megszakítás) jelző: értéke 1, ha a megszakítást letiltottuk.

Z ZERO jelző: értéke akkor 1, ha a művelet eredménye 0.

C CARRY (átvitel) jelző: értéke akkor 1, ha a művelet eredménye 8 biten nem tárolható.

Az 5. bitet nem használjuk, értéke mindig 1. Az SR-et egybyte-os hexadecimális számként olvasva egy 0 és 255 közötti értéket kapunk, amelyből az egyes jelzőbitek állapotára vissza tudunk következtetni.

Menjünk a TEDMON-ba! Bejelentkezéskor kiírja a regiszterek tartalmát, ezeket kedvünk szerint módosíthatjuk; próbáljuk ki! Használjuk a ROM-ban lévő, $8000-en kezdődő gépi kódú programot a BASIC hidegindításához: g8000, majd RETURN. Nyugodtan próbálkozzunk más címekkel is; a gépet ezzel úgysem tudjuk elrontani!

A gépi kódú programozás eredetileg a memória byte-jainak bitenkénti beállítását jelenti. Az egyes memóriacímek tartalmától függően a processzor más-más tevékenység elvégzésére kap utasítást, a gépi kódú program tehát egy - a külső szemlélő számára áttekinthetetlen, érthetetlen - memóriaterület. Ezért a gépi kódú programozáskor a kívánt hatás eléréséhez állandóan táblázatokat kellene forgatnunk, s azok tartalmának megfelelően kellene a tár byte-jait egyenként beállítanunk. Ezt az embertelen - ráadásul tévedési lehetőségekkel zsúfolt - tevékenységet szerencsére elvégzi helyettünk egy fordítóprogram, az ún. assembler, ha a programot számára érthető formában, az ún. assembly nyelven adjuk meg . Amikor gépi kódú programot írunk, akkor tehát valójában assembly nyelven programozunk, ennek eredménye az ún. forrásprogram, amelyet az assemblerrel lefordítva kapjuk a gépi kódú programot. Assembly nyelv természetesen nem egy van, hiszen minden proceszortípushoz más és más nyelv tartozik. Mi ettől függetlenül továbbra is gépi kódú programról beszélünk, mert ez az elnevezés jobban elterjedt, s a félreértéstől nem kell tartanunk.

Néhány egyszerű feladat a fentiekkel kapcsolatban:

2.1 Határozzuk meg, melyik jelzőbit értéke 1 ill. melyiké 0, ha SR értéke

$ff	$21	$38
$66	$22	$a0
$28	$34	$aa

2.2 Írjuk át a regiszterek tartalmát: A:=01 : X:=02 : Y:=03! Használjuk a TEDMON A parancsát! Ez a beépített assembler fordító parancsa, az utána szereplő tárhelyre fordítja az ezt követő assembly nyelvű utasítást. Írjuk be:

```
a 2000 tax
a 2001 brk
```

(természetesen RETURN-nel zárva a sorokat). A fordítás eredményét máris látjuk: a gép jelzi, hogy a tár $2000-es címén $aa (a TAX kódja), a tár $2001-es címén pedig $00 (a BRK kódja) található; ezek együttesen már egy gépi kódú programot alkotnak. Ezután g2000-rel indítva ezt a programot a következő értékeket kapjuk: A=01 : X=01 : Y=03 , ugyanis a TAX utasítás áttölti az A tartalmát X-be, miközben megvizsgálja, hogy ez a szám 0-e ill. a 7. bitje 1-e, s ennek megfelelően beállítja a Z és az N jelzőbiteket.

2.3 A TAX-hoz hasonlóak a TXA, TAY és TYA utasítások; nevükből kiolvasható, hogy melyik regiszter tartalmát melyik másik regiszterbe töltik. Nincs viszont sem TXY, sem pedig TYX utasítás. Készítsünk olyan programot, amelyik X-et Y-ba tölti ill. fordítva!

2.4 Készítsünk olyan programot, amelyik:
a./ A tartalmát tölti X-be és Y-ba,
b./ X tartalmát tölti Y-be és A-ba,
c./ Y tartalmát tölti X-be és A-ba!
Van még két tagja ennek az utasításcsoportnak, a TSX és a TXS, az X regiszter és az SP közötti adatátvitelt szolgálják, de ezekkel most még nem tudunk foglalkozni.

2.5 Írjuk át Y értékét 0-ra, X értékét pedig $ff-re! Hajtsunk végre egy TYA utasítást, majd vizsgáljuk meg a jelzőbitek állását! Mit tapasztalunk, ha a TXA utasítást alkalmazzuk?

Ismerkedjünk meg a talán leggyakoribb assembly utasítással (más szóval: mnemonikkal), az LDA-val! Ez az utasítás az utána szereplő szimbólum (az ún. operandus) formájától függően egy egybyte-os értéket tölt az akkumulátorba (lehet közvetlenül egy számot vagy a tár valamelyik címének aktuális értékét). Meg kell azonban különböztetnünk, hogy pld. az egybyte-os $23 számot, vagy a $23-as memóriacímen lévő egybyte-os számot (ami történetesen éppen $a8) akarjuk-e az A-ba tölteni, ezért az első esetben az lda #$23, míg a második esetben az lda $23 utasítást használjuk, s ezek hatására az akkumulátorba $23 ill. $a8 kerül. Hogy egy bizonyos mnemonikhoz az operandust milyen módon érhetjük el, azt a lehetséges címzésmódok határozzák meg. A mi processzorunk összesen 13 különböző címzésmódot ismer, ez teszi lehetővé a viszonylag kis számú utasítás rugalmas felhasználását. Eddig már három címzésmóddal megismerkedtünk:

- a TAX és társai külön operandust nem igényelnek, hiszen egyértelműen meghatározzák, hogy mit és mivel kell végrehajtanunk; ez az ún. implied vagy beleértett címzésmód;

- az LDA #$23 utasítás operandusát közvetlenül meghatározza, ez az ún. közvetlen címzésmód;

- az LDA $23 egybyte-os címe csak azokra a memóriahelyekre tud hivatkozni, amelyek címe nem nagyobb, mint $ff. Mivel ezek a memória 0. lapján helyezkednek el, ezt a címzésmódot nullalapos (Zero Page) címzésmódnak nevezzük.

Ez utóbbinak a $ff-nél nagyobb című tárhelyek esetén a következő címzésmód felel meg:

- pld. az LDA $4320 utasításnál a cím csak két byte-on fér el, ez az ún. abszolut címzésmód.

A többi címzésmódot - hasonlóan az utasításoknál követett úthoz - akkor ismertetjük, amikor a tárgyalásban előkerülnek.

2.6 Az LDA-hoz hasonlóan használhatók az X ill. az Y regiszterbe töltéshez az LDX ill. LDY utasítások. Az eddig megismert utasítások segítségével oldjuk meg mindhárom regiszter nullázását!

Ez utóbbi, nagyon egyszerű feladat is példázza már, hogy a gépi kódú programozásban nagyon sokféle módon meg tudjuk oldani ugyanazt a feladatot, s ezek között több lényegesen különböző is van. Hogy melyiket részesítjük előnyben, az mindig a pillanatnyi helyzettől függ.

2.7 Válasszuk ki a lehetséges megoldások közül azt, amelyik a tárból a legkevesebb helyet foglalja el!

2.8 Vizsgáljuk meg konkrét értékekkel, hogy az LDA, LDX és LDY utasítások melyik jelzőbiteket állítják! Vessük össze eredményünket a függelékben található táblázatban foglaltakkal!

Néhány esetben kifejezetten szükség van arra, hogy egy bizonyos programrész futtatása előtt a jelzőbitek valamely, általunk igényelt állásban legyenek. Ilyenkor - mivel az előző utasítások által beállított helyzetük esetleges - külön bitbeállító utasításokat használunk. Ezek beleértett (implied) címzésmódú utasítások, mindegyikük egy-egy jelzőbitet állít 1-re ill. 0-ra. Felsorolásuk:

SEC SED SEI 1-re állítja rendre a C, a D ill. az I jelzőket, míg
CLC CLD CLI CLV 0-ra állítja rendre a C, a D, az I ill. a V jelzőket.

2.9 Az SR regiszter tartalma $0f. Mennyi lesz az alábbi utasítások végrehajtása után:

SEC SED CLV CLI SEI CLD CLC ?

3. ÍRJUNK A KÉPERNYŐRE!

Gépünk már bekapcsoláskor is, bár még semmit sem írtunk be, sok mindent 'tud'. A ROM-ban ugyanis - többek között - olyan gépi kódú programokat találhatunk, amelyek nagyban segítségünkre lehetnek. Ezeket mi is használhatjuk, csupán arra kell ügyelnünk, hogy pontosan kiszolgáljuk őket. Cserébe sok munkát takaríthatunk meg: nem kell ezeket megírnunk (ami biztos, hogy gyengébbre sikerülne, mint ahogy a gép alkotói, profi programozók elkészítették). Legfontosabbak közülük számunkra most azok, amelyek segítségével a képernyőre tudunk írni. Lássuk ezeket részletesen!

A CHROUT nevű, a $ffd2 címen kezdődő rutin egy karaktert ír ki; a kérdéses karakter kódját a rutin hívása előtt az A regiszterbe kell tölteni (a kódokat lásd pld. a gépkönyvben, vagy az irodalomjegyzékben szereplő könyvek függelékében!).

A PRIMM rutin (kezdőcíme $ff4f) összefüggő szöveget ír ki; a kérdéses szöveget a rutint hívó utasítás után kell a tárba tenni, a szöveg végét egy 0 byte jelzi.

A KIHA rutin (kezdőcíme $fb05) az akkumulátor tartalmát (CHROUT-tal ellentétben nem kódként, hanem) egy 0 és 255 közötti számként értelmezi, s ezt a számot írja hexadecimális formában a képernyőre (és egy space-t).

A KIHAX rutin (kezdőcíme $faff) az A-ban lévő és az X-ben lévő érték (ilyen sorrendben) egymás mellé írásával kapott kétbyte-os számot írja ki. (Úgy is mondjuk, hogy ennek a kétbyte-os számnak A a felső, X pedig az alsó byte-ja.)

A KIDXA rutin (kezdőcíme $a45f az X felső és az A alsó byte-ból kapott kétbyte-os számot írja ki decimálisan. (Vigyázat! A sorrend fordított, mint KIHAX-nál!)

Már mindent tudunk, csak azt nem, hogyan lehet egy szubrutint meghívni! Erre szolgál a JSR utasítás, amelyet csak abszolut címzésmóddal használhatunk. Például a

```
2000 jsr $ffd2
2003 brk
```

hatására a visszatérési cím a verembe kerül, és a program futása a $ffd2 címen folytatódik, egészen addig, amíg egy RTS utasítást nem talál, majd a veremben lévő címre tér vissza (és ezt természetesen ki is veszi a veremből), tehát futását a $2003 címen folytatja. A JSR - RTS utasításpár egyébként pontos megfelelője a BASIC-béli GOSUB - RETURN utasításpárnak.

3.1 Mutassunk példákat a fenti rutinok használatára! A programot pld. $6666-tól fordítsuk a tárba, majd futtassuk!

```
;put"@0:k0"
;************************************************
;*  képernyőre író rutinok használata           *
;************************************************
*=$6666
chrout=$ffd2        ;az A tartalmát kiíró rutin címe
primm=$ff4f         ;a szövegkiíró rutin címe
kiha=$fb05          ;a-ban lévő egész kiírása hexadecimálisan
kihax=$faff         ;a/x-ben lévő egész kiírása hexadec.
kidxa=$a45f         ;x/a-ban lévő egész kiírása decimálisan
cls=$93             ;képtörlés kódja
ret=$0d             ;RETURN kódja
        lda #cls
        jsr chrout  ;képtörlés
        jsr primm   ;ki szöveg:
        .byte 'ezt a szoveget irja ki a kepernyore',0
        lda '%'     ;folyamatosan egy
        jsr chrout  ;%-jel kiírása,
        lda #ret    ;majd egy
        jsr chrout  ;soremelés
        lda '@'     ;s ezután egy
        jsr chrout  ;kukac-jel kiírása
        lda #ret    ;újabb
        jsr chrout  ;soremelés
        lda #17     ;a:=17 (=$11)
        jsr kiha    ;a kiírása hex.
        lda #1      ;a:=1
        ldx #2      ;x:=2
        jsr kihax   ;ki hex.$0201 (=513)
        lda #1      ;a:=1
        ldx #2      ;x:=2
        jsr kidxa   ;ki dec. 258 (=$0102)
        rts         ;vissza.a BASIC-be
        .end
```

Egy kis kitérő azok kedvéért, akik még nem tól otthonosak a JCL assembler használatában: a lemezen található seq file-ok az ún. forrásszöveget tartalmazzák; ezt a JCL-be töltve olvashatjuk, szerkeszthetjük a szokásos módon, majd az asm parancs után fordíthatjuk is. A számunkra lényeges fordítási lehetőségek:

```
   ,c              szintaktikus ellenőrzés;
   ,x,c            a tárba,
   ,p,c            a nyomtatóra,
   .8:filenév,c    a 8-as lemezegységben lévő lemezre
```

fordítja a memóriában lévő forrásszöveget.

3.2 Töröljük le a képernyőt, majd írjuk ki rá a nevünket, két sorral lejjebb a személyi számunkat! Meg tudjuk-e úgy is oldani a feladatot, hogy a személyi szám kiírásához a KIHA, a KIHAX ill. a KIDXA rutinokat használjuk?

3.3 Irjuk ki a kurzor pillanatnyi helyzetétől öt sorral lejjebb az 1989 számot, decimális és hexadecimális formában!

3.4 Vigyük a kurzort (bárhol is van) a 6. sor 3. pozíciójára! Figyeljük a $c4-$c5 címeken lévő sor- ill. oszloppozíció-mutatókat! Hogyan hasznosíthatjuk ezeket?

3.5 Töröljük le a képernyőt, majd írjunk ki az első sorba egy 1-est, melléje pedig az 1 kódú karaktert, a második sorba egy 2-est, mellé a 2 kódú karaktert s. í. t. egészen 5-ig!

3.6 Használjuk a vezérlőkódokkal is a CHROUT rutint (kurzormozgatás, színek, inverz, villogó karakterek stb.)!

3.7 A CHROUT rutin segítségével definiáljunk ablakot! Bal felső sarka a #. sor 5. helyén, jobb alsó sarka a 15. sor 12. helyén legyen!

Innen is láthatjuk, mennyire egyszerű a dolgunk azokban az esetekben, amikor pontosan az a rutin van a tárban, amire szükségünk van (ezért is hasznos dolog a ROM-listákat forgatni!). Persze, nem mindig van ilyen szerencsénk, pld. már a következő feladatnál sem:

3.8 Irjunk a képernyő 4. sorának közepére egy csillagot!

Megoldhatnánk a problémát a 3.4 feladat segítségével is, van azonban egy sokkal egyszerűbb megoldás, hiszen pontosan ismerjük a kérdéses képernyőpozíció tárbéli címét: a kezdőponttól (3072=$0c00) 3 és fél sorral, azaz 3.5*40=140 pozícióval feljebb van. Erre a címre kell beírnunk a *-karakter kódját. Ehhez ismernünk kell az STA utasítást, amely az akkumulátor tartalmát írja a megadott címre; abszolut címzésmódot alkalmazunk:

```
;put"@0:c0"
;************************************************
;*  a 4. sor közepére ir egy csillagot          *
;************************************************
*=$6666
hely=$0c00+140      ;4. sor közepe
       lda '*'      ;akkumulátorba * kódja
       sta hely     ;a helyére irjuk
       rts          ;vissza a BASIC-be
       .end
```

Figyeljük meg, hogy a JCL megengedi decimális és hexadecimális (sőt bináris) számok együttes használatát!

A következő feladatok megoldásához segítséget nyújtanak az 1. fejezet feladatainál közölt információk:

3.9 Irjunk a képernyő bal alsó sarkába egy piros &-jelet!

3.10 Irjunk a képernyő jobb alsó sarkába egy zöld egyenlőségjelet, a legsötétebb színárnyalattal!

3.11 A TED-regiszter térképe alapján megállapíthatjuk, hogy a háttér és a keret színét az $ff15 és a $ff19 címeken változtathatjuk. Állítsuk be a keret színét a legsötétebb kékre, a háttér színét pedig a leghalványabb pirosra!

Ha egy karaktert villogóra szeretnénk állítani, a színmemória megfelelő byte-jának legfelső bitjét magasra (1-re) kell állítanunk, oly módon, hogy a többi bit változatlan maradjon. Erre kiválóan alkalmas az ORA utasítás, amely az akkumulátor tartalma és az ORA utasítás operandusa között bitenként elvégzi a logikai VAGY műveletet, s az eredményt az akkumulátorba írja. Ennek segítségével tudjuk a következő feladatot megoldani:

3.12 Irjunk a 4. sor közepére egy villogó csillagot!

```
;put"@0:c1"
;************************************************
;* a 4. sor közepére ír egy villogó csillagot   *
;************************************************
*=$6666
hely=$0c00+140          ;4. sor közepe
szín=hely-$0400         ;ua. a színmemóriában
        lda '*'         ;akkumulátorba a * kódja
        sta hely        ;a helyére írjuk
        lda szín        ;szín-érték betöltése
        ora #%10000000  ;a legmagasabb bit bekapcsolása
        sta szín        ;szín-érték visszatöltése
        rts             ;vissza a BASIC-be
        .end
```

A megoldásnál az ORA utasítást közvetlen címzésmóddal használtuk, de megoldhatjuk abszolut címzés segítségével is:

3.13 Oldjuk meg az előző feladatot az ORA utasítás abszolut címzésével!

```
;put"@0:c11"
*=$6666
hely=$0c00+140          ;4. sor közepe
szín=hely-$0400         ;ua. a színmemóriában
        lda '*'         ;akkumulátorba a * kódja
        sta hely        ;a helyére írjuk
        lda #%10000000  ;maszk a 7. bithez
        ora szín        ;a szín-érték 7. bitjének
        sta szín        ;bekapcsolása, visszatöltése
        rts             ;vissza a BASIC-be
        .end
```

3.14 Szüntessük meg az előző feladatban kiírt * villogását!

Nyilvánvaló, hogy az előbb kiírt csillag villogását a színmemória ugyanazon byte-ja legfelső bitjének kikapcsolása szünteti meg, míg a többi bitnek érintetlenül kell maradnia. Erre az AND utasítás alkalmas; ez az akkumulátor és az AND utasítás operandusa között bitenként logikai ÉS műveletet végez, s az eredményt az akkumulátorba írja. Egy megoldás a következő:

```
;put"@0:c2"
;************************************************
;* a 4. sor közepére kiírt csillag villogását   *
;* megszünteti (sys27000)                       *
;************************************************
*=27000
hely=$0c00+140          ;4. sor közepe
szín=hely-$0400         ;ua. a színmemóriában
       lda szín         ;szín-érték betöltése
       and #%01111111   ;a legmagasabb bit kikapcsolása
       sta szín         ;szín-érték visszatöltése
       rts              ;vissza a BASIC-be
      .end
```

Ezt a programot 27000-től fordíttattuk, hogy az eredeti ($6666=26214-től kezdődő) programmal egymás után többször is futtathatók legyenek.

3.15 Oldjuk meg az előző feladatot az AND utasítás abszolut címzésével!

```
;put"@0:c21"
;************************************************
;* a 4. sor közepére kiírt csillag villogását   *
;* megszünteti (sys27000)                       *
;************************************************
*=27000
hely=$0c00+140          ;4. sor közepe
szín=hely-$0400         ;ua. a színmemóriában
       lda #%01111111   ;maszk a 7. bithez
       and szín         ;a szín-érték 7. bitjének
       sta szín         ;kikapcsolása, visszatöltése
       rts              ;vissza a BASIC-be
       .end
```

3.16 Írjunk a jobb felső sarokba egy inverz space-t!

A megoldáshoz kell tudnunk, hogy egy karakter inverz voltát a képernyőtárban neki megfelelő byte legfelső bitje határozza meg.

3.17 Írjunk olyan programot, amely az előző feladatban kiírt inverz space-t normál space-szé alakítja vissza!

3.18 Írjunk olyan programot, amely a 4. sor közepén lévő karakter villogását ellenkezőjére változtatja (azaz: ha villogott, megszünteti a villogását, ha pedig nem, akkor villogtatni kezdi)!

Segítségképpen: az EOR utasítás az akkumulátor tartalma és az EOR utasítás operandusa között bitenként KIZÁRÓ VAGY logikai műveletet végez, majd az eredményt az akkumulátorba írja. (Emlékeztető: 1EOR1 = 0EOR0 = 0 ill. 1EOR0 = 0EOR1 = 1 .)

További haladásunkhoz feltétlenül szükséges néhány elágazási lehetőség ismerete, így most erre látunk néhány példát.

3.19 Írjunk olyan programot, amely az 1, 2, 3 számok közül bármely kettő közötti relációt írja a képernyőre!

Ebben a feladatban különböző számokat kell összehasonlítanunk, s ehhez a CMP utasítást fogjuk használni. Ez az utasítás – anélkül, hogy a tárban vagy a feltételregiszteren kívül bármely más regiszterben változást idézne elő – beállítja a jelzőbiteket az operandusa értékétől függően úgy, mintha az akkumulátorból kivonta volna az operandus értékét.

Használjuk az on. feltételes ugró utasítások közül a BEQ és a BPL utasításokat; a többi ilyen utasítással együtt mindketten a tárban elfoglalt helyüktől számított relatív ugrásra szolgálnak: az operandusként szereplő távolság visszafelé ugrásnál legfeljebb 126, előre ugrásnál pedig legfeljebb 129 byte lehet; ha ennél nagyobb távolságra szeretnénk segítségükkel ugrani, fordításkor hibajelzést kapunk. A BEQ utasítás a Z jelző 1-es értéke esetén, a BPL pedig az N jelző 0 értéke esetén eredményezi az adott relatív címre történő ugrást, ellenkező esetben (mintha mi sem történt volna) a program a tárban következő utasítással folytatódik.

Fontos szót ejtenünk még arról is, hogy egy, az akkumulátorban lévő, 0 és 9 közötti számot hogyan írathatunk ki a képernyőre. Egyszerű megoldás a KIHA rutin használata, mert egy ilyen számot akár decimálisan, akár hexadecimálisan írunk is ki, ugyanaz adódik. A CHROUT rutinhoz azonban – mivel ez az akkumulátorban a kiírandó karakter kódját várja – eredeti formájában nem alkalmas, a kiírás kedvéért át kell alakítanunk, majd – mivel az eredeti értékére továbbra is szükségünk lesz – a visszaalakítást is el kell végeznünk. Szerencsére a 0 és 9 közötti számok kódjai $30 és $39 közé esnek, eredeti sorrendjüknek megfelelően, így az átalakítást egy ORA '0', a visszaalakítást pedig egy EOR '0' utasítás segítségével egyszerűen megoldhatjuk.

Ennyi előkészítés után a program megértése bizonyára nem jelent problémát:

```
:put"@0:e0"
;*************************************************
;*            elágazások   : beq, bpl            *
;*             (kiírás relációjellel)            *
;*************************************************
*=$6666
primm=$ff4f         ;a szövegkiíró rutin címe
chrout=$ffd2        ;az A tartalmát kiíró rutin címe
        lda #1      ;rendre
        jsr vizsg   ;1, 2 ill. 3
        lda #2      ;vizsgálata
        jsr vizsg
        lda #3
        jsr vizsg
        rts
vizsg               ;vizsgálat: az acc-ban lévő szám
                    ;milyen relációban van rendre
                    ;1-gyel, 2-vel ill. 3-mal?
        jsr számki  ;acc értékének kiírása
        cmp #1      ;hasonlít 1-hez
        jsr kiír    ;kiírja
        jsr primm   ;ki szöveg:
        .byte ' 1 ',13,0
        jsr számki  ;A értékének kiírása
        cmp #2      ;hasonlít 2-höz
        jsr kiír    ;kiírja
        jsr primm   ;ki szöveg:
        .byte ' 2 ',13,0
        jsr számki  ;A értékének kiírása
        cmp #3      ;hasonlít 3-hoz
        jsr kiír    ;kiírja
        jsr primm   ;ki szöveg:
        .byte ' 3 ',13,0
        rts
kiír                ;kiírja a relációt
        beq egyenl  ;ha =, akkor egyenl
        bpl nagy    ;ha >, akkor nagy
        jsr primm   ;különben ki szöveg:
        .byte ' < ',0
        rts         ;majd vissza
egyenl
        jsr primm   ;ki szöveg:
        .byte ' = ',0
        rts         ;majd vissza
nagy
        jsr primm   ;ki szöveg:
        .byte ' > ',0
        rts         ;majd vissza
számki              ;az A-ban lévő számjegy kiírása
        ora '0'     ;kiíráshoz a kód módosítása
        jsr chrout  ;kiír
        eor '0'     ;módosítás vissza,
        rts         ;majd vissza
        .end
```

3.20 Oldjuk meg az előző feladatot más relatív ugrások segítségével!

```
;          elágazások    : bne, bmi                    *
;*          (szöveges kiírás)                          *
*=$6666
primm=$ff4f          ;a szövegkiíró rutin címe
chrout=$ffd2         ;az akkumulátor tartalmát kiíró rutin címe
        lda #1       ;rendre
        jsr vizsg    ;1, 2 ill. 3
        lda #2       ;vizsgálata
        jsr vizsg
        lda #3
        jsr vizsg
        rts
vizsg                ;vizsgálat: az acc-ban lévő szám
                     ;milyen relációban van rendre
                     ;1-gyel, 2-vel ill. 3-mal?
        jsr számki   ;acc értékének kiírása
        cmp #1       ;hasonlít 1-hez
        jsr kiír     ;kiírja
        jsr primm    ;ki szöveg:
        .byte '1',13,0
        jsr számki   ;acc értékének kiírása
        cmp #2       ;hasonlít 2-höz
        jsr kiír     ;kiírja
        jsr primm    ;ki szöveg:
        .byte '2',13,0
        jsr számki   ;acc értékének kiírása
        cmp #3       ;hasonlít 3-hoz
        jsr kiír     ;kiírja
        jsr primm    ;ki szöveg:
        .byte '3',13,0
        rts
kiír                 ;szöveggel kiírja a relációt
        bmi kisebb   ;ha <, akkor kisebb
        bne nagy     ;ha >, akkor nagy
        jsr primm    ;ki szöveg:
        .byte ' egyenlo ',0
        rts          ;majd vissza
kisebb
        jsr primm    ;különben ki szöveg:
        .byte ' kisebb ',0
        rts          ;majd vissza
nagy
        jsr primm    ;ki szöveg:
        .byte ' nagyobb ',0
        rts          ;majd vissza
számki               ;az acc-ban lévő számjegy kiírása
        ora '0'      ;kiíráshoz a kód módosítása
        jsr chrout   ;kiír
        eor '0'      ;módosítás visszaállítása,
        rts          ;majd vissza
        .end
```

A BNE utasítás abban az esetben eredményez relatív ugrást, ha a Z jelzőbit értéke 0, a BMI pedig akkor, ha az N bit értéke 1.

E témájában kicsi, de használati értékét tekintve igen jelentős kitérő után folytassuk 'csillagos' feladatainkat!

3.21 Írjuk végig a képernyő 3. sorát csillagokkal!

A feladat megoldása lehetne egy olyan program is, amelyben 40 különböző STA utasítás szerepel, amelyek operandusai a 3. sor egyes pozícióira mutatnak. Ez azonban nagyon csúnya, idő- és energiaigényes megoldás olyankor, amikor a ciklusszervezéshez szükséges eszközök már majdnem mindnyájan a kezünkben vannak, csupán az X regiszter egy újabb funkciójával, a nevét is adó indexelt címzéssel kell megismerkednünk. Ez a következőt jelenti:

Ha az LDA CÍM,X utasítást kapja a processzor, akkor az akkumulátorba a CÍM+X címen lévő értéket tölti (tehát: ha az X regiszter 0-t tartalmaz, akkor pont a CÍM-en, ha 1-et, akkor a CÍM+1-en stb. lévő értéket). Ha tehát több, a tárban egymás után elhelyezkedő értékkel kell egymás után ugyanazt végrehajtanunk, használhatjuk ezt a címzésmódot, s az egyikről a másikra való áttéréskor csupán X értékét kell a megadott határok között egyesével változtatni: ha növelésre van szükségünk, ez az INX, ha pedig csökkentésre, akkor a DEX utasítás segítségével könnyedén megvalósítható. Ugyanez a címzésmód a másik indexregiszterrel (az Y-nal) is használható; ekkor INY a regiszter tartalmát növelő és DEY a csökkentő utasítás. Ezek után már tényleg gyerekjáték a program elkészítése:

```
;put"@0:c3"
;************************************************
;* a 3. sort végigírja csillaggal               *
;*             (sys26214)                       *
;************************************************
*=$6666
hely=$0c00+80       ;3. sor eleje
       ldx #40      ;ennyi csillag kell egy sorba
       lda '*'      ;akkumulátorba a * kódja
ciklus
       sta hely-1,x ;a helyére írjuk
       dex          ;x:=x-1
       bne ciklus   ;ha x>0, akkor vissza
       rts          ;vissza a BASIC-be
       .end
```

Ha a kiírandó csillagok darabszámát tesszük az X regiszterbe, akkor az indexelést a kezdőcímnél 1-gyel kisebb címhez kell viszonyítanunk. Ha a kezdőcímhez akarunk viszonyítani, akkor a következő módon írjuk meg a ciklust:

3.22 Oldjuk meg az előző feladatot úgy, hogy az indexelésnél a kezdőcímhez viszonyítunk!

```
:put"@0:c4"
:************************************************
:* a 3. sort végigírja csillaggal               *
:*              (sys26214)                      *
:************************************************
*=$6666
hely=$0c00+80       ;3. sor eleje
        ldx #39     ;ennyi+1 csillag kell egy sorba
        lda '*'     ;akkumulátorba * kódja
ciklus
        sta hely,x  ;a helyére írjuk
        dex         ;x:=x-1
        bpl ciklus  ;ha x>0, akkor vissza
        rts         ;vissza a BASIC-be
        .end
```

Következő feladatainkban a 3. sorban fogunk egy csillagot balról jobbra végigfuttatni oly módon, hogy az aktuális helyétől mindig 1 karakterrel jobbra írjuk ki, s az eredeti helyét space-szel töltjük fel. Mielőtt azonban hozzáfognánk, ismerkedjünk meg az 1.10 feladatban is említett 198-as (=$c6) byte használatával!

3.23 Írjunk egy olyan ciklust, amely a 'kurzor jobbra' billentyű lenyomására vár!

Mindössze annyit kell tudnunk, hogy az említett byte értéke 51 (=$33) lesz, ha ezt a billentyűt nyomjuk le. A program:

```
:put"@0:b0"
:************************************************
:*  a kurzor jobbra billentyű lenyomására vár   *
:************************************************
*=$6666
bill=$c6            ;billentyű lenyomását jelzi
jobbra=51           ;kurzor jobbra bill.
vár                 ;várakozás
        lda bill    ;billentyűt nyomunk?
        cmp #jobbra ;kurzor jobbra?
        bne vár     ;ha nem, várjunk!
        rts         ;vissza a BASIC-be
        .end
```

3.24 Írjunk egy olyan programot, amely a 'kurzor jobbra' billentyű lenyomására engedi a 3. sorban mozogni a csillagot!

```
;put"@0:c5"
;************************************************
;*  a 3. sorba írt csillag a kurzor jobbra      *
;*  gomb nyomására elmegy a sor végéig.         *
;*  láthatatlanul gyors                         *
;************************************************
*=$6666
bill=$c6              ;billentyű lenyomását jelzi
jobbra=51             ;kurzor jobbra bill.
hely=$0c00+80         ;3. sor eleje
        lda '*'       ;akkumulátorba a * kódja
        sta hely      ;* a sor elejére
        ldx #0        ;számoljuk
ciklus                ;mozgató ciklus
vár                   ;várakozás
        lda bill      ;billentyűt nyomunk?
        cmp #jobbra   ;kurzor jobbra?
        bne vár       ;ha nem, várjunk!
        lda '*'       ;akkumulátorba a * kódja
        sta hely+1,x  ;a köv. helyre írjuk
        lda ' '       ;akkumulátorba a space kódja
        sta hely,x    ;az előző * törlése
        inx           ;x:=x+1
        cpx #39       ;utolsó volt?
        bne ciklus    ;ha nem, akkor vissza
        rts           ;vissza a BASIC-be
        .end
```

Láthatjuk, hogy a 'kurzor jobbra' gomb egyszeri lenyomása is sokkal tovább tart, mint a *-nak a soron való végigfutása, ezért - mint ahogy az a BASIC nyelvben is szokásos - egy ellenkező értelmű várakozó ciklussal 'fogjuk meg' a csillagot.

3.25 Írjunk egy olyan programot, amely a 'kurzor jobbra' és a 'kurzor balra' billentyű váltogatott lenyomására engedi a 3. sorban mozogni a csillagot!

```
;put"@0:c6"
;************************************************
;*  a 3. sorba írt csillag a kurzor jobbra ill. *
;*  a kurzor balra gombok nyomására elmegy a    *
;*  sor végéig                                  *
;************************************************
*=$6666
bill=$c6              ;billentyű lenyomását jelzi
jobbra=51             ;kurzor jobbra bill.
balra=48              ;kurzor balra bill.
hely=$0c00+80         ;3. sor eleje
        lda '*'       ;akkumulátorba a * kódja
        sta hely      ;* a sor elejére
        ldx #0        ;számoljuk
```

```
ciklus                  ;mozgató ciklus
vár1                    ;várakozás
        lda bill        ;billentyűt nyomunk?
        cmp #jobbra     ;kurzor jobbra?
        bne vár1        ;ha nem, várjunk!
vár2                    ;várakozás
        lda bill        ;billentyűt nyomunk?
        cmp #balra      ;kurzor balra?
        bne vár2        ;ha nem, várjunk!
        lda '*'         ;akkumulátorba a * kódja
        sta hely+1,x    ;a köv. helyre írjuk
        lda ' '         ;akkumulátorba a space kódja
        sta hely,x      ;az előző * törlése
        inx             ;x:=x+1
        cpx #39         ;utolsó volt?
        bne ciklus      ;ha nem, akkor vissza
        rts             ;vissza a BASIC-be
        .end
```

3.226 Az előző program rettentően darabos. Oldjuk meg úgy, hogy a várakozásokat egy-egy számláló ciklus helyettesíti! (Töltsünk 255-öt az Y regiszterbe, ezt egyesével csökkentsük, s ezt addig folytassuk, amíg el nem éri az értéke a 0-t!) Milyen mértékben lassítja ez a mozgást?

3.227 Tanulmányozzuk az 1.11 feladatban is említett belső órát, hogy ennek segítségével lassítsuk a folyamatot!

```
:put"@0:vek1"
;*************************************************
;* a belső óra legalsó byte-ja értékének megfe-  *
;*    lelő helyre ir egy *-ot                    *
;*************************************************
*=$6666
kép=$0c00               ;a képernyőtár kezdete
vek=$a5                 ;a belső óra legalsó byte-ja
bill=$c6                ;billentyű lenyomását jelzi
cls=$93                 ;képtörlés kódja
chrout=$ffd2            ;az A tartalmának kiírása
vek1                    ;vek legalsó byte-ja kijelzése
        ldy vek         ;y:=vek
        bne ír          ;nem 0, tehát írunk
        lda ' '         ;A-ba a space
ciklus  sta kép,y       ;a képre
        iny             ;y=y+1
        bne ciklus      ;még van, vissza
ír                      ;*-ot írunk
        lda '*'         ;a-ba a * kódja
        sta kép,y       ;helyére
        lda bill        ;billentyűt figyel
        cmp #$33        ;kurzor jobbra?
        bne vek1        ;nem, ismét
        rts
        .end
```

Szemmel látható, hogy minden 5. csillag kiírása után az óra 'ugrik' egyet. Ennek az a magyarázata, hogy az órát a másodpercenként 50-szer érkező megszakítás működteti, de az óra - mivel 1/60 másodpercekben méri az időt - csak úgy lesz pontos, ha minden 5 megszakításból 4 alkalommal 1-1 lépést tesz, az ötödik alkalommal viszont kettőt. Így - ha menet közben egy picit 'ingadozik' is a járása, végeredményben másodpercenként 60-nal növekszik az értéke, tehát éppen 1/60 másodpercekben méri az időt.

3.28 Módosítsuk úgy az előző programot, hogy a belső óra középső byte-ját jelezze ki!

3.29 Használjuk a *-nak a 3. sorban történő mozgása lassításához a belső órát!

Azt tapasztaljuk, hogy a belső órának még a legalsó (tehát a legygyorsabb) byte-ja is túlságosan 'lomha' a csillag mozgásának lassításához.

3.30 Próbálkozzunk egy másik órával: a TED elején lévő időzítők egyikével!

```
:put"@0:c61"
:************************************************
:*  a 3. sorba irt csillag elmegy a sor végéig  *
:************************************************
*=$6666
vekk = $ff01          ;1. időzítő felső byte-ja
hely=$0c00+80         ;3. sor eleje
        lda '*'       ;akkumulátorba a * kódja
        sta hely      ;* a sor elejére
        ldx #0        ;számoljuk
ciklus                ;mozgató ciklus
vár1                  ;várakozás
        lda vekk      ;az óra 0-n áll-e?
        bne vár1      ;ha nem, várjunk!
vár2                  ;várakozás
        lda vekk      ;az óra 0-n áll-e?
        beq vár2      ;ha igen, várjunk!
        lda '*'       ;akkumulátorba a * kódja
        sta hely+1,x  ;a köv. helyre irjuk
        lda ' '       ;akkumulátorba a space kódja
        sta hely,x    ;az előző * törlése
        inx           ;x:=x+1
        cpx #39       ;utolsó volt?
        bne ciklus    ;ha nem, akkor vissza
        rts           ;vissza a BASIC-be
        .end
```

A képernyőn mozgó csillagok tanulmányozásához feltétlenül szükséges bizonyos számításokat is végeznünk, így ezek ismertetésével folytatjuk a munkát.

4. ARITMETIKA

Ebben a részben az egy- és kétbyte-os egészek körében végezhető aritmetikai műveletekről lesz szó. Bár eléggé ritkán használatos, tegyünk említést az aritmetikai műveletek használatáról decimális üzemmódban!

4.1 Növeljük meg 5-tel decimális módban a $6650 címen lévő szám értékét!

Az összeadáshoz az ADC utasítást használjuk, amely összeadja az akkumulátor, az operandus és a C jelzőbit értékét (ez utóbbinak majd a kétbyte-os összeadásnál lesz lényeges szerepe). Ne felejtsük el a C jelzőt az összeadás előtt 0-ra állítani, mert lehet, hogy az előző műveletek eredményeképpen 1-et mutat!

```
:put"@0:a1"
;************************************************
;* aritmetika: egy adott érték hozzáadása egy   *
;* adott címen lévő számhoz, decimális módban   *
;************************************************
*=$6666
szám=$6650          ;egy egybyte-os szám címe
a1                  ;szám:=szám+5
        sed         ;d-jelző beállítása
        lda szám    ;acc=szám
        clc         ;carry=0
        adc #5      ;acc=szám+5
        sta szám    ;szám=szám+5
        cld         ;d-jelző törlése
        rts
        .end
```

4.2 Csökkentsük 3-mal decimális módban a $6650 címen lévő szám értékét!

A kivonás az SBC utasítás segítségével végezhető el; ez az akkumulátor tartalmát csökkenti az operandus és a C jelző ellentettjének (tehát 0 esetén 1-nek, 1 esetén 0-nak) az értékével. Ez utóbbinak szintén a kétbyte-os kivonásnál lesz jelentősége, ezért ügyelnünk kell, hogy a művelet megkezdése előtt C értékét 1-re állítsuk, mert lehet, hogy az előző műveletek eredményéből kifolyólag éppen 0-n áll.

```
:put"@0:a2"
;************************************************
;* aritmetika: egy adott érték kivonása egy     *
;* adott címen lévő számból, decimális módban   *
;************************************************
```

```
*=$6666
szám=$6650          ;egy egybyte-os szám címe
a2                  ;szám:=szám-3
        sed         ;d-jelző beállítása
        lda szám    ;acc=szám
        sec         ;carry=1
        sbc #3      ;acc=szám-3
        sta szám    ;szám=szám-3
        cld         ;d-jelző törlése
        rts
        .end
```

4.3 Adjunk össze két egybyte-os egész számot decimális módban, és az összeget írjuk az egyik szám helyére!

```
;put"@0:a3"
;*************************************************
;* aritmetika: két, adott címen lévő egybyte-os  *
;* szám összeadása, decimális módban             *
;*************************************************
*=$6666
egyik=$6650         ;az egyik egybyte-os szám címe
másik=$6652         ;a másik egybyte-os szám címe
a3                  ;egyik:=egyik+másik
        sed         ;d-jelző beállítása
        lda egyik   ;acc=egyik
        clc         ;carry=0
        adc másik   ;acc=egyik+másik
        sta egyik   ;egyik=egyik+másik
        cld         ;d-jelző törlése
        rts
        .end
```

4.4 Határozzuk meg két egybyte-os egész szám különbségét decimális módban, és az eredményt írjuk egy újabb byte-ra!

```
;put"@0:a4"
;*************************************************
;* aritmetika: két, adott címen lévő egybyte-os  *
;* szám kivonása, decimális módban               *
;*************************************************
*=$6666
egyik=$6650         ;az egyik egybyte-os szám címe
másik=$6652         ;a másik egybyte-os szám címe
eredm=$6652         ;az eredmény címe
a4                  ;eredm:=egyik-másik
        sed         ;d-jelző beállítása
        lda egyik   ;acc=egyik
        sec         ;carry=1
        sbc másik   ;acc=egyik-másik
        sta eredm   ;eredm=egyik-másik
        cld         ;d-jelző törlése
        rts
        .end
```

A decimális aritmetika után térjünk át az általában használatos hexadecimális aritmetikára!

4.5 Növeljünk 1-gyel egy adott egybyte-os számot!

A megoldáshoz ismernünk kell az INC utasítást, amely az operandusának megfelelő értéket 1-gyel, megnöveli.

```
;put"@0:a5"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* szám növelése 1-gyel (inkrementálás)         *
;************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a5                  ;szám:=szám+1
        inc szám
        rts
        .end
```

4.6 Növeljünk 1-gyel egy adott kétbyte-os számot!

```
;put"@0:a22"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékének 1-gyel növelése               *
;************************************************
*=$6666
szám=$6650          ;a kétbyte-os szám címe
a22                 ;szám:=szám+1
        inc szam    ;alsó byte
        bne vég     ;ha nem 0, kész
        inc szám+1  ;felső byte
vég     rts
        .end
```

A megjegyzések magukért beszélnek: a felső byte-ot csak akkor növeljük, ha az alsó 0-n áll.

4.7 Csökkentsünk 1-gyel egy adott egybyte-os számot!

```
;put"@0:a6"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* szám csökkentése 1-gyel (dekrementálás)      *
;************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a6                  ;szám:=szám-1
        dec szám
        rts
        .end
```

4.8 Csökkentsünk 1-gyel egy adott kétbyte-os számot!

```
;put"@0:a23"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékének 1-gyel csökkentése            *
;************************************************
*=$6666
szám=$6650          ;a kétbyte-os szám címe
a23                 ;szám:=szám-1
        lda szám    ;<szám=0?
        bne ugr     ;nem, ->ugr
        dec szám+1  ;>szám=>szám-1
ugr     dec szám    ;<szám=<szám-1
        rts
        .end
```

4.9 Növeljünk 13-mal egy adott egybyte-os számot!

```
;put"@0:a7"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* szám növelése egy adott értékkel (pl.13-mal) *
;************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a7                  ;szám:=szám+13
        clc         ;carry=0
        lda szám    ;acc=szám
        adc #13     ;acc=szám+13
        sta szám    ;szám=szám+13
        rts
        .end
```

4.10 Növeljük 13-mal egy adott címen lévő kétbyte-os szám értékét!

```
;put"@0:a24"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékének egy adott egybyte-os számmal  *
;* (pld. 13-mal) növelése                       *
;************************************************
*=$6666
szám=$6650          ;a kétbyte-os szám címe
a24                 ;szám:=szám+13
        clc         ;carry=0
        lda szám    ;acc=<szám
        adc #13     ;acc=<szám+13
        sta szám    ;<szám=<szám+13
        bcc ugr     ;ha nincs átvitel:->ugr
        inc szám+1  ;van,->>szám:=>szám+1
        ugr rts
        .end
```

4.11 Egy adott címen lévő egybyte-os szám 52-vel növelt értékét írjuk egy másik adott címre!

A megoldás annyira egyszerű, hogy elegendő csak a megjegyzésekkel ellátott forrásszöveget közölnünk:

```
;put"@0:a8"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* szám növelése egy adott értékkel (pl.52-vel); *
;* az eredmény egy másik címen legyen.          *
;************************************************
*=$6666
egyik=$6650          ;az egyik egybyte-os szám címe
másik=$6652          ;a másik egybyte-os szám címe
a8                   ;másik:=egyik+52
        clc          ;carry=0
        lda egyik    ;acc=egyik
        adc #52      ;acc=egyik+52
        sta másik    ;másik=egyik+52
        rts
        .end
```

4.12 Egy adott címen lévő kétbyte-os szám $1234-gyel növelt értékét írjuk egy másik adott címre!

```
;put"@0:a25"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékét növeljük egy adott kétbyte-os   *
;* számmal (pld. $1234-gyel), s az eredményt    *
;* írjuk egy másik kétbyte-os szám címére       *
;************************************************
*=$6666
egyik=$6650          ;az egyik kétbyte-os szám címe
másik=$6652          ;a másik kétbyte-os szám címe
szám=$1234           ;az adott kétbyte-os szám értéke
a25                  ;másik:=egyik+szám
        clc          ;carry=0
        lda egyik    ;acc=<egyik
        adc #<szám   ;acc=<egyik+<szám
        sta másik    ;<másik=<egyik+<szám
TPONT   lda egyik+1  ;acc=>egyik
        adc #>szám   ;acc=>egyik+>szám
        sta másik+1  ;>másik=>egyik+>szám
        rts
        .end
```

Érdemes megfigyelnünk a C bit értékének változását az EGYIK szám felső byte-jához való hozzáadáskor: a JCL SETBRK utasítása segítségével töréspontot iktathatunk be a TPONT-nál (az alsó byte-on való összeadás befejezésekor). Vizsgáljuk meg az EGYIK=$1111 ill. az EGYIK=$11ff értékek esetén a C bit állását a felső byte-ok összeadása előtt!

4.13 Egy adott címen lévő egybyte-os szám 27-tel csökkentett értékét írjuk az adott címre!

```
;put"@0:a9"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os   *
;* szám csökkentése egy adott értékkel           *
; (pld. 27-tel)                                  *
;************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a9                  ;szám:=szám-27
        sec         ;carry=1
        lda szám    ;acc=szám
        sbc #27     ;acc=szám-27
        sta szám    ;szám=szám-27
        rts
        .end
```

4.14 Egy adott címen lévő egybyte-os szám 31-gyel csökkentett értékét írjuk egy másik adott címre!

```
;put"@0:a10"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os   *
;* szám csökkentése egy adott értékkel (pl. 31-  *
;* gyel); az eredmény egy másik címen legyen.    *
;************************************************
*=$6666
egyik=$6650           ;az egyik egybyte-os szám címe
másik=$6652           ;a másik egybyte-os szám címe
a10                   ;másik:=egyik-31
        sec           ;carry=1
        lda egyik     ;acc=egyik
        sbc #31       ;acc=egyik-31
        sta másik     ;másik=egyik-31
        rts
        .end
```

4.15 Egy adott címen lévő kétbyte-os szám 27-tel csökkentett értékét írjuk az adott címre!

A kétbyte-os szám egybyte-ossal való csökkentésekor a felső byte értéke 1-gyel vagy 0-val csökkenhet aszerint, hogy az alsó byte-ok különbségének képzésekor keletkezett-e átvitel vagy sem. Ez a C bit állásától függő elágazást jelent.

```
;put"@0:a26"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékének egy adott egybyte-os számmal   *
;* (pld. 27-tel) csökkentése                     *
;************************************************
```

```
*=$6666
szám=$6650          ;a kétbyte-os szám címe
a26                 ;szám:=szám-27
        sec         ;carry=1
        lda szám    ;acc=<szám
        sbc #27     ;acc=<szám-27
        sta szám    ;<szám=<szám-27
TPONT   bcs ugr
        dec szám+1  ;>szám:=>szám-1
        ugr rts
        .end
```

Itt is - mint a 4.12 feladatnál - figyeljük meg különböző esetekben a TPONT törésponban a C bit értékét!

4.16 Egy adott címen lévő kétbyte-os szám $1234-gyel csökkentett értékét írjuk egy másik adott címre!

```
;put"@0:a27"
;*************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékét csökkentsük egy adott kétbyte-os *
;* számmal (pld. $1234-gyel), s az eredményt     *
;* írjuk egy másik kétbyte-os szám címére        *
;*************************************************
*=$6666
egyik=$6650         ;az egyik kétbyte-os szám címe
másik=$6652         ;a másik kétbyte-os szám címe
szám=$1234          ;az adott kétbyte-os szám értéke
a27                 ;másik:=egyik-szám
        sec         ;carry=1
        lda egyik   ;acc=<egyik
        sbc #<szám  ;acc=<egyik-<szám
        sta másik   ;<másik=<egyik-<szám
        lda egyik+1 ;acc=>egyik
        sbc #>szám  ;acc=>egyik->szám
        sta másik+1 ;>másik=>egyik->szám
        rts
        .end
```

A továbbiakban - különösebb magyarázat nélkül - ezekkel rokon feladatokat és megjegyzésekkel bőségesen ellátott megoldásokat közlünk.

4.17 Egy adott címen lévő egybyte-os számot növeljünk meg egy másik címen lévő egybyte-os számmal, s az eredményt írjuk az első szám helyére!

```
;put"@0:a11"
;*************************************************
;* aritmetika: egy adott címen lévő egybyte-os   *
;* számhoz adjuk hozzá egy másik címen lévő,     *
;* ugyancsak egybyte-os szám értékét.            *
;*************************************************
```

```
*=$6666
egyik=$6650          ;az egyik egybyte-os szám címe
másik=$6652          ;a másik egybyte-os szám címe
a11                  ;egyik:=egyik+másik
        clc          ;carry=0
        lda egyik    ;acc=egyik
        adc másik    ;acc=egyik+másik
        sta egyik    ;egyik=egyik+másik
        rts
        .end
```

4.18 Egy adott címen lévő kétbyte-os számot növeljünk meg egy másik címen lévő kétbyte-os számmal, s az eredményt írjuk az első szám helyére!

```
;put"@0:a28"
;*************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékét növeljük meg egy másik kétbyte-os*
;* számmal!                                      *
;*************************************************
*=$6666
egyik=$6650          ;az egyik kétbyte-os szám címe
másik=$6652          ;a másik kétbyte-os szám címe
a28                  ;egyik:=egyik+másik
        clc          ;carry=0
        lda egyik    ;acc=egyik
        adc másik    ;acc=egyik+másik
        sta egyik    ;egyik=egyik+másik
        lda egyik+1  ;acc=>egyik
        adc másik+1  ;acc=>egyik+>másik
        sta egyik+1  ;>egyik=>egyik+>másik
        rts
        .end
```

4.19 Egy adott címen lévő egybyte-os számot növeljünk meg egy másik címen lévő egybyte-os számmal, s az eredményt írjuk egy harmadik címre!

```
:put"@0:a12"
;* aritmetika: egy adott címre írjuk be két,     *
;* ugyancsak adott címeken lévő egybyte-os szám *
;* összegének értékét.                           *
*=$6666
egyik=$6650          ;az egyik egybyte-os szám címe
másik=$6652          ;a másik egybyte-os szám címe
eredm=$6652          ;az eredmény címe
a12                  ;eredm:=egyik+másik
        clc          ;carry=0
        lda egyik    ;acc=egyik
        adc másik    ;acc=egyik+másik
        sta eredm    ;eredm=egyik+másik
        rts
        .end
```

4.20 Egy adott címen lévő kétbyte-os szám értékét növeljük meg egy másik címen lévő kétbyte-os számmal, s az eredményt írjuk egy harmadik címre!

```
;put"@0:a29"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékét növeljük meg egy másik kétbyte-os*
;* számmal, s az eredményt írjuk egy ugyancsak  *
;* adott helyre!                                *
;************************************************
*=$6666
egyik=$6650         ;az egyik kétbyte-os szám címe
másik=$6652         ;a másik kétbyte-os szám címe
eredm=$6654         ;az eredmény címe
a29                 ;eredm:=egyik+másik
       clc          ;carry=0
       lda egyik    ;acc=<egyik
       adc másik    ;acc=<egyik+<másik
       sta eredm    ;<eredm=<egyik+<másik
       lda egyik+1  ;acc=>egyik
       adc másik+1  ;acc=>egyik+>másik
       sta eredm+1  ;>eredm=>egyik+>másik
       rts
       .end
```

4.21 Egy adott címen lévő egybyte-os szám értékét csökkentsük egy másik címen lévő egybyte-os számmal, s az eredményt írjuk egy harmadik címre!

```
;put"@0:a13"
;************************************************
;* aritmetika: egy adott címre írjuk be két,    *
;* ugyancsak adott címeken lévő egybyte-os szám *
;* különbségének értékét.                       *
;************************************************
*=$6666
egyik=$6650         ;az egyik egybyte-os szám címe
másik=$6652         ;a másik egybyte-os szám címe
eredm=$6652         ;az eredmény címe
a13                 ;eredm:=egyik-másik
       sec          ;carry=1
       lda egyik    ;acc=egyik
       sbc másik    ;acc=egyik-másik
       sta eredm    ;eredm=egyik-másik
       rts
       .end
```

4.22 Egy adott címen lévő kétbyte-os szám értékét csökkentsük egy másik címen lévő kétbyte-os számmal, s az eredményt írjuk egy harmadik címre!

```
:put"@0:a30"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékét csökkentsük egy másik kétbyte-os *
;* számmal, s az eredményt írjuk egy ugyancsak  *
;* adott helyre!                                *
;************************************************
*=$6666
egyik=$6650         ;az egyik kétbyte-os szám címe
másik=$6652         ;a másik kétbyte-os szám címe
eredm=$6654         ;az eredmény címe
a30                 ;eredm:=egyik-másik
        sec         ;carry=1
        lda egyik   ;acc=<egyik
        sbc másik   ;acc=<egyik-<másik
        sta eredm   ;<eredm=<egyik-<másik
        lda egyik+1 ;acc=>egyik
        sbc másik+1 ;acc=>egyik->másik
        sta eredm+1 ;>eredm=>egyik->másik
        rts
        .end
```

4.23 Egy adott címen lévő egybyte-os szám értékét növeljük meg egy másik címen lévő egybyte-os számmal és még 32-vel, s az eredményt írjuk az első szám helyére!

```
:put"@0:a14"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* számhoz adjunk hozzá egy másik címen lévő,   *
;* ugyancsak egybyte-os számot és egy adott     *
;* számot (pld. 32-t)!                          *
;************************************************
*=$6666
egyik=$6650         ;az egyik egybyte-os szám címe
másik=$6652         ;a másik egybyte-os szám címe
a14                 ;egyik:=egyik+másik+32
        clc         ;carry=0
        lda egyik   ;acc=egyik
        adc másik   ;acc=egyik+másik
        clc         ;carry=0
        adc #32     ;acc=egyik+másik+32
        sta egyik   ;egyik=egyik+másik+32
        rts
        .end
```

4.24 Egy adott címen lévő egybyte-os szám értékét növeljük meg egy másik címen lévő kétbyte-os számmal, majd csökkentsük 21-gyel, s az eredményt írjuk egy harmadik címre!

```
:put"@0:a15"
```

```
;************************************************
;* aritmetika: egy adott címre írjuk be két,    *
;* ugyancsak adott címeken lévő szám összegének *
;* egy adott számmal (mondjuk 21-gyel) csökken- *
;* tett értékét!                                *
;************************************************
*=$6666
egyik=$6650          ;az egyik egybyte-os szám címe
másik=$6652          ;a másik egybyte-os szám címe
eredm=$6654          ;az eredmény címe
a15                  ;eredm:=egyik+másik-21
        clc          ;carry=0
        lda egyik    ;acc=egyik
        adc másik    ;acc=egyik+másik
        sec          ;carry=1
        sbc #21      ;acc=egyik+másik-21
        sta eredm    ;eredm=egyik+másik-21
        rts
        .end
```

Egyetlen dologra lesz még szükségünk ahhoz, hogy a képernyőre tovább írjunk+rajzoljunk: ez pedig a kétbyte-os egészek összehasonlítása. Nézzünk erre is egy példát!

4.25 Határozzuk meg, melyik nagyobb két, adott címen lévő kétbyte-os szám közül (pld. < ill. >= relációkkal)!

```
;put"@0:e12"
;************************************************
;* aritmetika: határozzuk meg két adott címen lé-*
;* vő kétbyte-os szám között a nagyságrendi vi- *
;* szonyt!                                      *
;************************************************
*=$6666
egyik=$6650          ;az egyik kétbyte-os szám címe
másik=$6654          ;a másik kétbyte-os szám címe
vizsg                ;ha egyik<másik, akkor ki:'<'
                     ;különben ki:'>='
        lda egyik+1  ;felső byte-ok
        cmp másik+1  ;összehas.
        bcc kisebb   ;egyik<másik, kisebb
        bne nagyeg   ;egyik>másik, nagyeg
        lda egyik    ;alsó byte-ok
        cmp másik    ;összehas.
        bcc kisebb   ;egyik<másik, kisebb
nagyeg               ;egyik>=másik
        jsr $ff4f    ;ki: szöveg
        .byte '>=',0
        rts          ;vissza
kisebb               ;egyik<másik
        jsr $ff4f    ;ki: szöveg
        .byte '<',0
        rts          ;vissza
        .end
```

Végezetül gyakorlásképpen néhány feladat:

4.26 Oldjuk meg egy a 4.25 feladatot, hogy '<=' ill. '>' relációt vizsgáljon a program!

4.27 Oldjuk meg egy a 4.25 feladatot, hogy válasszuk szét a '<', az '=' és a '>' eseteket!

4.28 Adott két egybyte-os szám a címével: EGYIK és MÁSIK. Írjunk olyan programot, amely közülük a kisebbiket EGYIK-be, a nagyobbikat pedig MÁSIK-ba teszi!

4.29 Oldjuk meg az előző feladatot abban az esetben, ha EGYIK és MÁSIK egyaránt kétbyte-os számokat jelölnek!

4.30 Adott két egybyte-os szám a címével: EGYIK és MÁSIK. Írjunk olyan programot, amely összegüket EGYIK-be, a különbségüket pedig MÁSIK-ba teszi!

4.31 Oldjuk meg az előző feladatot abban az esetben, ha EGYIK és MÁSIK egyaránt kétbyte-os számokat jelölnek!

4.32 Adott a memóriában két, egybyte-os számokból álló 60-60 elemű sorozat. A sorozatok kezdőcímei: EGYIK és MÁSIK. Írjunk olyan programot, amely a sorozatok elemeit rendre összeadja, s az így kapott 60 számot sorban lerakja a tár ÖSSZEG címétől kezdve! (Az egyszerűség kedvéért tételezzük fel, hogy minden összeg elfér 1 byte-on.)

4.33 Oldjuk meg az előző feladatot abban az esetben, ha mindkét sorozat kétbyte-os számokat tartalmaz!

4.34 Mekkora számokat tudunk négy byte-on ábrázolni? Írjunk programot, amely két négybyte-os szám összegét képezi!

4.35 Írjunk programot, amely két négybyte-os szám különbségét képezi!

4.36 Írjunk programot, amely két négybyte-os szám közül kiválasztja a nem nagyobbikat!

4.37 Írjunk programot, amely a memóriában felcserél két négybyte-os számot!

5. CSILLAGOK; VILLOGAS ...

Ismét visszatérünk a képernyőhöz: már meg tudunk oldani olyan feladatokat, amelyekben apróbb-nagyobb számításokat is el kell végezni. Elsőként egy olyan feladatot oldunk meg, amelyben egy igen gyakori címzésmódot is használunk (bár nem ez a mostani alkalmazása a legnagyobb előnye, hanem pld. 4.32-ben) az ún. indirekt indexelt címzést, amellyel egy, a tár nullalapján elhelyezett címhez képest Y-nal hátrébb lévő értékhez tudunk hozzáférni. Ez így eléggé bonyolultnak tűnik, de lássunk egy példát! Tudvalévő, hogy a tár $2b-$2c címén mindig a BASIC-program kezdetének helyét jelző mutató van, értéke általában $1001. Tegyük fel, hogy az Y regiszter tartalma $33! Ekkor az LDA ($2b),Y utasítás hatására az akkumulátorba töltődik a $2b-n található címnél Y-nal nagyobb című byte tartalma, tehát (mivel $1001+$33 = $1034) ez egyenértékű az LDA $1034 utasítással. Hogy akkor miért nem ezt használjuk? Egyrészt azért, mert az Y index értékét igen könnyen tudjuk növelni ill. csökkenteni, ezáltal könnyedén tudunk pld. adattáblázatokban mozogni, másrészt pedig sok esetben a tár más és más területétől kezdődő, ugyanolyan szerkezetű adatok feldolgozását könnyen el tudjuk végezni (a nullalapon lévő egyetlen cím átírásával).

5.1 Írjuk tele a képernyő 10. oszlopát csillagokkal!

```
:put"@0:c7"
;************************************************
;* a 10. oszlopot végigírja csillaggal          *
;************************************************
*=$6666
zp=$e0              ;tárhely a nullalapon
hely=$0c00+10       ;10. oszlop teteje
        lda #<hely  ;kezdőcím
        sta zp      ;nullalapra
        lda #>hely  ;töltése
        sta zp+1
        ldx #24     ;ennyi+1 csillag kell egy oszlopba
        ldy #0      ;indexeléshez nullázzuk
ciklus
        lda '*'     ;akkumulátorba a * kódja
        sta (zp),y  ;a helyére irjuk
        clc         ;zp:=zp+40
        lda #40     ;,azaz egy sorral lejjebb
        adc zp
        sta zp
        bcc ugrik
        inc zp+1
ugrik   dex         ;x:=x-1
        bpl ciklus  ;ha x>0, akkor vissza
        rts         ;vissza a BASIC-be
        .end
```

Még egy apróság: ha magunk készítette mutatót helyezünk a nullalapra, figyeljünk rá, nehogy oda írjuk, ahol más, a rendszer által is használt értékek vannak! Egy RAM-térképből pillanatok alatt kiderül (ld. pld. [1]-ben), hogy a $d8-$e8 rész üres, így a $e0-$e1 byte-okat nyugodtan használhatjuk.

Most már mindent tudunk a következő feladat megoldásához:

5.2 Rajzoljunk csillagokból egy keretet a képernyőre!

```
;put"@0:c8"
;************************************************
;*     a keretet végigírja csillaggal           *
;************************************************
*=$6666
zp=$e0                  ;tárhely a nullalapon
helyb=$0c00             ;a bal szélső oszlop teteje
helyj=helyb+39          ;a jobb szélső oszlop teteje
helya=helyb+960         ;az alsó sor eleje
        ldx #37         ;ennyi+1 csillag kell egy sorba
                        ;(a sarkokat az oszlopokból írjuk)
        lda '*'         ;akkumulátorba a * kódja
sorcik                  ;sortöltő ciklus
        sta helyb+1,x   ;felső sorba írjuk
        sta helya+1,x   ;alsó sorba írjuk
        dex             ;x:=x-1
        bpl sorcik      ;ha x>0, akkor vissza
        lda #<helyb     ;bal felső sarok
        sta zp          ;címének
        lda #>helyb     ;nullalapra
        sta zp+1        ;töltése
        jsr oszlop
        lda #<helyj     ;jobb felső sarok
        sta zp          ;címének
        lda #>helyj     ;nullalapra
        sta zp+1        ;töltése
        jsr oszlop
        rts             ;vissza a BASIC-be
oszlop                  ;oszloptöltő ciklus
        ldx #24         ;ennyi+1 csillag kell egy oszlopba
        ldy #0          ;indexeléshez nullázzuk
oszcik
        lda '*'         ;akkumulátorba a * kódja
        sta (zp),y      ;a helyére írjuk
        clc             ;zp:=zp+40
        lda #40         ;,azaz egy sorral lejjebb
        adc zp
        sta zp
        bcc ugrik
        inc zp+1
ugrik   dex             ;x:=x-1
        bpl oszcik      ;ha x>0, akkor vissza
        rts             ;vissza a BASIC-be
        .end
```

5.3 Írjunk az előző feladatra rövidebb programot!

```
;put"@0:c9"
;************************************************
;*      a keretet vegigírja csillaggal          *
;************************************************
*=$6666
zp=$e0              ;tárhely a nullalapon
helyb=$0c00         ;a bal szélső oszlop teteje
helyj=helyb+39      ;a jobb szélső oszlop teteje
helya=helyb+960     ;az alsó sor eleje
        ldx #39     ;ennyi+1 csillag kell egy sorba
        lda '*'     ;akkumulátorba a * kódja
sorcik              ;sortöltő ciklus
        sta helyb,x ;felső sorba írjuk
        sta helya,x ;alsó sorba írjuk
        dex         ;x:=x-1
        bpl sorcik  ;ha x>0, akkor vissza
        lda #<helyj ;jobb felső sarok
        sta zp      ;címének
        lda #>helyj ;nullalapra
        sta zp+1    ;töltése
        ldx #24     ;ennyi csillag kell egy oszlopba
oszcik
        ldy #0      ;indexeléshez nullázzuk
        lda '*'     ;akkumulátorba * kódja
        sta (zp),y  ;a helyére írjuk
        iny
        sta (zp),y  ;a helyére írjuk
        clc         ;zp:=zp+40
        lda #40     ;,azaz egy sorral lejjebb
        adc zp
        sta zp
        bcc ugrik
        inc zp+1
ugrik
        dex         ;x:=x-1
        bne oszcik  ;ha x>0, akkor vissza
        rts         ;vissza a BASIC-be
        .end
```

5.4 Az előző két megoldás között leginkább az INY utasítás használatában van különbség. Mi teszi ezt lehetővé, és mennyiben 'rövidebb' a második program az elsőnél?

5.5 Azonos módon használjuk-e az 5.1, az 5.2 és az 5.3 feladatokban az STA (zp),Y utasítást?

5.6 Írjunk olyan programot, amely egy 12*6 oldalú téglalapot ír a képernyő közepére #-jelekből!

5.7 Írjunk olyan programot, amely a keretből indulva egyre csökkenő oldalhosszúságú téglalapokat rajzol különböző jelekből!

5.8 Írjunk olyan programot, amely a képernyőt teleírja csillaggal!

Elsőként a képernyő 4 lapjának párhuzamos töltésével oldjuk meg a feladatot:

```
;put"@0:c91"
;***********************************************
;*     a képernyőt teleírja csillaggal         *
;*             (sys26214)                      *
;***********************************************
*=$6666
kép1=$0c00          ;a képernyő 1. lapjának kezdete
kép2=$0d00          ;a képernyő 2. lapjának kezdete
kép3=$0e00          ;a képernyő 3. lapjának kezdete
kép4=$0f00          ;a képernyő 4. lapjának kezdete
tele1               ;csillaggal teleírja a képernyőt
                    ;laponként
       lda '*'      ;a * képernyőkódja az acc-ba
       ldx #0       ;index 0-ról indul
indul               ;ciklus kezdete
       sta kép1,x   ;* az 1. lapra
       sta kép2,x   ;* a 2. lapra
       sta kép3,x   ;* a 3. lapra
       sta kép4,x   ;* a 4. lapra
       inx          ;x:=x+1
       bne indul    ;ha x<256, akkor vissza
       rts
       .end
```

Rögtön látható, hogy azzal, hogy mind a 4 lapot teleírtuk, fölösleges munkát is végeztünk, mert ezzel összesen 1024 csillagot helyeztünk el, a képernyőre viszont csak 25*40, azaz 1000 karakter fér, így 24 csillagot a képernyő után lévő tárterületre írtunk. Ez problémát nem okoz, viszont a programozás szempontjából lényeges könnyítést jelent, hogy nem kell a jobb alsó saroknál megállni.

5.9 Írjunk olyan programot, amely az 5.8 feladatot egy nullalapos mutató segítségével oldja meg!

Ez a megoldás egy nullalapos címre teszi le a képernyő aktuális lapjára mutató értéket, majd az Y regiszter értékének növelésével végigfut a lapon, ennek végeztével pedig a lapmutató értékét (azaz a felső byte-ot) növeli 1-gyel, s mindezt 4-szer csinálja végig, ezzel lefedi az egész képernyőt.

```
;put"@0:c92"
;***********************************************
;*     a képernyőt teleírja csillaggal         *
;*             (sys26214)                      *
;***********************************************
```

```
*=$6666
kép=$0c00           ;a képernyő kezdete
zp=$e0              ;kétbyte-os mutató a nullalapon
tele2               ;csillaggal teleírja a képernyőt
                    ;laponként, nullalapos mutató segítségével
        lda #<kép   ;a nullalapos
        sta zp      ;mutató beállítása
        lda #>kép   ;a képernyő
        sta zp+1    ;kezdetére
        lda '*'     ;a * képernyőkódja az acc-ba
        ldx #4      ;4-szer csináljuk
        ldy #$00    ;index 0-ról indul
indul               ;ciklus eleje
        sta (zp),y  ;* a mutatott helyre
        iny         ;y:=y+1
        bne indul   ;ha y<256, akkor vissza
        inc zp+1    ;új lap a képernyőn
        dex         ;x:=x-1
        bne indul   ;ha x>0, akkor vissza
        rts
        .end
```

5.10 Írjunk olyan programot, amely az 5.8 feladatot egy nullalapos mutató segítségével oldja meg, de nem a már teleírt lapok számától, hanem az elért cím értékétől függően áll le!

```
;put"@0:c93"
;*************************************************
;*      a képernyőt teleírja csillaggal          *
;*              (sys26214)                       *
;*************************************************
*=$6666
kép=$0c00           ;a képernyő kezdete
zp=$e0              ;kétbyte-os mutató a nullalapon
tele3               ;csillaggal teleírja a képernyőt
                    ;egy nullalapos mutató segítségével
        lda #<kép   ;a nullalapos
        sta zp      ;mutató beállítása
        lda #>kép   ;a képernyő
        sta zp+1    ;kezdetére
        ldy #0      ;index 0-ról indul
kezd                ;külső ciklus (laponként)
        lda '*'     ;a * képernyőkódja A-ba
indul               ;belső ciklus (egy lapon belül)
        sta (zp),y  ;* a mutatott helyre
        iny         ;y:=y+1
        bne indul   ;ha y<256, akkor vissza
        inc zp+1    ;új lap a képernyőn
        lda zp+1    ;a lapmutató
        cmp #$10    ;elérte a #$10-et?
        bne kezd    ;ha még nem, akkor vissza
        rts
       .end
```

Nyilvánvalóan akkor van a leállásra szükség, ha a mutató értéke eléri a $1000-et, mert ez már nem adhat az Y-nal való indexelés után a képernyőre eső értéket; a kép jobb alsó sarkának a címe ugyanis 4071, s ez a legnagyobb.

5.11 Írjunk olyan programot, amely az 5.8 feladatot egy nullalapos mutató segítségével oldja meg, de az indexeléshez az Y regiszter értékét csak formailag használja (passzív Y)!

A 'passzív Y' azt jelenti, hogy a program futása közben Y értéke végig 0, nem változtatjuk, magyarul: csak a nullalapon lévő cím határozza meg a kérdéses képernyőcímet. Hogy akkor miért van rá szükség? Azért, mert nincs nem indexelt, nullalapos indirekt címzésmód, s ezzel a látszólag felöslegesen cipelt Y-nal egyszerűen megírhatók az ilyen ciklusok.

```
;put"@0:c94"
;************************************************
;*      a képernyőt teleírja csillaggal         *
;*              (sys26214)                      *
;************************************************
*=$6666
kép=$0c00           ;a képernyő kezdete
zp=$e0              ;kétbyte-os mutató a nullalapon
tele4               ;csillaggal teleírja a képernyőt
                    ;egy nullalapos mutató segítségével,
                    ;(passzív) y-nal indexelve
        lda #<kép   ;a nullalapos
        sta zp      ;mutató beállítása
        lda #>kép   ;a képernyő
        sta zp+1    ;kezdetére
        ldy #0      ;0 az indexeléshez
kezd                ;külső ciklus (laponként)
        lda '*'     ;a * képernyőkódja A-ba
indul               ;belső ciklus (egy lapon belül)
        sta (zp),y  ;* a mutatott helyre
        inc zp      ;mutató alsó byte-ja
        bne indul   ;ha lapon belül van, akkor vissza
        inc zp+1    ;új lap a képernyőn
        lda zp+1    ;a lapmutató
        cmp #$10    ;elérte a #$10-et?
        bne kezd    ;ha még nem, akkor vissza
        rts
        .end
```

5.12 Oldjuk meg ugyanezt a feladatot, de most passzív X-szel indexelve!

Ez a másik nullalapos indexelési lehetőség, a neve indexelt indirekt címzés, amely nevében hasonlít a fejezet elején ismertetett címzésmódhoz, de nem csupán formailag, hanem lényegében is eltér attól. Egyrészt - mint nevéből is kitűnik - itt először az indexelést kell figyelembe venni, s csak

ezután az indirekciót (tehát azt, hogy ez az érték csak egy cím, a tényleges operandus ott található, ahová ez a cím mutat). Másrészt: míg az előző címzésmódnál csak az Y regiszter, itt csak az X regiszter használható. Ennek a címzésmódnak a megvilágítására is lássunk egy példát!

Tegyük fel, hogy a $d8 helytől kezdve a következő értékek vannak a tárban: $12 $34 $56 $78 $9a $bc $de $f0 , és tegyük fel azt is, hogy az X regiszter tartalma $04! Ekkor az LDA ($d8, X) utasítás a tár $bc9a címén lévő értéket tölti az akkumulátorba, ugyanis először történik az indexelés: a $d8-hoz képest $04-gyel hátrább lévő (tehát a $dc-n kezdődő) cím, tehát $bc9a a mérvadó, s - most jön az indirekció - a $bc9a-n lévő érték kerül az akkumulátorba. Ha X értéke történetesen 0, akkor az akkumulátorba a $3412 rekesz tartalma kerül. Ez a címzésmód rendkívül ritka, a ROM-listában egyszeri alkalmazásával sem találkoztunk, s itt is csak a bemutatása kedvéért mutatunk rá példát.

```
;out"@0:c95"
;*************************************************
;*      a képernyőt teleírja csillaggal          *
;*              (sys26214)                       *
;*************************************************
*=$6666
kép=$0c00           ;a képernyő kezdete
zp=$e0              ;kétbyte-os mutató a nullalapon
tele5               ;csillaggal teleírja a képernyőt
                    ;egy nullalapos mutató segítségével,
                    ;de most (ugyancsak passziv) x-szel
indexelve
        lda #<kép   ;a nullalapos
        sta zp      ;mutató beállítása
        lda #>kép   ;a képernyő
        sta zp+1    ;kezdetére
        ldx #0      ;0 az indexeléshez
kezd                ;külső ciklus (laponként)
        lda '*'     ;a * képernyőkódja A-ba
indul               ;belső ciklus (egy lapon belül)
        sta (zp,x)  ;* a mutatott helyre
        inc zp      ;mutató alsó byte-ja
        bne indul   ;ha lapon belül van, akkor vissza
        inc zp+1    ;új lap a képernyőn
        lda zp+1    ;a lapmutató
        cmp #$10    ;elérte a #$10-et?
        bne kezd    ;ha még nem, akkor vissza
        rts
        .end
```

5.CSILLAGOK; VILLOGÁS ...

A következőkben arra látunk egy egyszerű példát, hogy egy program hogyan képes önmagát átírni, módosítani. Ez tipikus gépi kódú sajátosság, de nagyon óvatosan bánjunk vele, mert kellemetlen meglepetések érhetnek bennünket!

5.13 Készítsünk olyan, önmagát módosító programot, amely a képernyőt feltölti csillagokkal!

```
:put"@0:c96"
;************************************************
;*      a képernyőt teleírja csillaggal         *
;*          vigyázz! módosítja magát!           *
;*   csak egyszer futtasd, különben baj lesz!   *
;*              (sys26214)                      *
;************************************************
*=$6666
kép=$0c00           ;a képernyő kezdete
tele6               ;csillaggal teleírja a képernyőt
                    ;egy magát átíró program segítségével
        lda '*'     ;a * képernyőkódja az A-ba
mut                 ;ciklusok kezdete
        sta kép     ;* a mutatott helyre
        inc mut+1   ;mutató alsó byte-ja növelése
        bne mut     ;ha még nem 0, tovább
        inc mut+2   ;felső byte-ot is
        ldx mut+2   ;növeljük
        cpx #$10    ;elérte a #$10-et?
        bne mut     ;ha még nem, akkor vissza
        rts
        .end
```

Ezzel a fogással jócskán lerövidítettük programunkat. A lényeg a STA KÉP sorban van; ez a 3 byte-os utasítás 2. és 3. byte-ján (a megszokott alacsony-magas sorrendben) mindig azt a képernyőcímet tartalmazza, ahová a következő * kerül, s ezt a címet módosítjuk az INC MUT+1 ill. szükség esetén az INC MUT+2 utasításokkal, amíg a cím el nem éri a $1000-t.

Ez a program ilyen formájában azonban óriási veszélyt rejt magában. Első futtatása előtt ugyanis a STA KÉP utasítás STA $0c00 alakú, de a futás után STA $1000 alakú lesz! Másodszori indítása után tehát először a $1000, majd a $1001, a $1002 ... s. í. t. címre írja a * karakter kódját, egyesével növelve a MUT+1 és a MUT+2 értékét, s ezzel (egészen addig, amíg $6666-hoz érve saját magát is tönkre nem teszi) a tár minden byte-ját $2a-ra, a * kódjára állítja! Tökéletesen alkalmas arra, hogy minden eddigi munkánkat tönkretegye! Óvatosan bánjunk tehát ezzel az eszközzel! De mi akkor a megoldás? Az, hogy az önmagát átíró program futása előtt a szükséges kezdőértékre állítjuk az ilyen, menet közben módosuló értékeket.

5.14 Készítsünk az 5.13 feladatra olyan megoldást, amely többször egymás utáni futtatásra is alkalmas!

```
:put"@0:c97"
:************************************************
:*      a képernyőt teleírja csillaggal         *
:*  ez is módosítja magát, de ez már ügyesebb!  *
:*              többször is futtathatod!        *
:*             (sys26214)                       *
:************************************************
*=$6666
kép=$0c00           ;a képernyő kezdete
tele7               ;csillaggal teleírja a képernyőt
                    ;egy magát átíró program segítségével
        lda #<kép   ;a programban lévő
        sta mut+1   ;mutató beállítása
        lda #>kép   ;a képernyő
        sta mut+2   ;kezdetére
        lda '*'     ;a * képernyőkódja az A-ba
mut                 ;ciklusok kezdete
        sta kép     ;* a mutatott helyre
        inc mut+1   ;mutató alsó byte-ja növelése
        bne mut     ;ha még nem 0, tovább
        inc mut+2   ;felső byte-ot is
        ldx mut+2   ;növeljük
        cpx #$10    ;elérte a #$10-et?
        bne mut     ;ha még nem, akkor vissza
        rts
        .end
```

5.15 Készítsünk olyan, önmagát módosító programot, amely a képernyő minden karakterének villogását megváltoztatja (azaz: a villogók megszűnnek, a nem villogók pedig elkezdenek villogni)!

A villogtatásért (mint már a 3.18 feladatnál láttuk) a színmemória byte-jainak legfelső bitje a felelős, ezt kell tehát a színmemória minden byte-ja esetén ellenkezőjére változtatni. Ez az EOR #%10000000 utasítás segítségével valósítható meg, de ez a változást csak az A-ban hozza létre. A színmemória byte-jait tehát egyenként be kell töltenünk az A-ba, ott el kell végeznünk az EOR műveletet, majd az A-t vissza kell töltenünk a színmemóriába. Ha az előbbihez hasonló, magát módosító programmal dolgozunk, akkor a módosítandó cím két helyen is szerepel: először az LDA, másodszor pedig az STA utasítás operandusaként. Pontosan kiszámítható, hogy a címek a MUT+1-MUT+2, ill. a MUT+6-MUT+7 címeken vannak. Még egy megjegyzés: az aktuális címet addig változtatjuk, míg el nem éri a színmemória mögött lévő képernyőmemória kezdőcímét, $0c00-t. Ezek után nézzük a megoldást:

```
;put"@0:c99"
;************************************************
;* a képernyő minden villogó karaktere villogá- *
;* sát megszünteti, a nem villogókat pedig vil- *
;* logóvá teszi.                                *
;*              (sys26214)                      *
;************************************************
*=$6666
szín=$0800              ;a színmemória kezdete
vill                    ;villogást fordít
        lda #<szín      ;a programban lévő
        sta mut+1       ;mutató beállítása
        sta mut+6       ;a másik helyen is
        lda #>szín      ;a színmemória
        sta mut+2       ;kezdetére
        sta mut+7       ;a másik helyen is
mut                     ;ciklusok kezdete
        lda szín        ;ciklusok kezdete
        eor #%10000000  ;a 7. bit negálása
        sta szín        ;* a mutatott helyre
        inc mut+1       ;mutató alsó byte-ja növelése
        inc mut+6       ;a másik helyen is
        bne mut         :ha még nem 0, tovább
        inc mut+2       ;felső byte-ot is
        inc mut+7       ;a másik helyen is
        ldx mut+2       ;növeljuk
        cpx #$0c        ;elérte a #$0c-t?
        bne mut         ;ha még nem, akkor vissza
        rts
        .end
```

A továbbiakban néhány, az eddigi ismeretekkel összefüggő feladat önálló megoldását javasoljuk:

5.16 Rajzoljunk a képernyő közepére egy 10*10-es, piros karikákból álló négyzetet, s az átlóit húzzuk meg kék x-ekkel!

5.17 Készítsünk olyan programot, amely az F1 billentyű megnyomására letörli a képernyő alsó, az F2-ére pedig a felső felét!

5.18 Készítsünk olyan programot, amely az F1 billentyű megnyomására letörli a képernyő bal, az F2-ére pedig a jobb oldalát!

5.19 Készítsünk olyan programot, amely a * billentyű megnyomására felcseréli a képernyő alsó és felső felét!

5.20 Készítsünk olyan programot, amely a @ billentyű megnyomására felcseréli a képernyő bal és jobb oldalát!

5.21 Készítsünk olyan programot, amely az F1 billentyű megnyomására a képernyőt vízszintes középvonalára tükrözi!

5.22 Készítsünk olyan programot, amely az F1 billentyű megnyomására a képernyőt függőleges középvonalára tükrözi!

5.23 Készítsünk olyan programot, amely az F1 billentyű megnyomására a képernyő minden karakterét az eredeti inverzébe váltja!

5.24 A fekete-fehér képernyőn nem látjuk a háttérrel azonos fényességű karaktereket. Készítsünk olyan programot, amely az F1 billentyű megnyomására az ilyen karakterek fényességét megváltoztatja!

5.25 Készítsünk olyan programot, amely a 'kurzor balra' billentyű megnyomására a képernyőt 1 oszloppal balra lépteti úgy, hogy a jobb szélső oszlop helyére üres oszlopot ír!

5.26 Készítsünk olyan programot, amely a 'kurzor jobbra' billentyű megnyomására a képernyőt 1 oszloppal jobbra lépteti úgy, hogy a bal szélső oszlop helyére üres oszlopot ír!

5.27 Készítsünk olyan programot, amely a 'kurzor fel' billentyű megnyomására a képernyőt 1 sorral felfelé lépteti úgy, hogy az alsó sor helyére üres sort ír!

5.28 Készítsünk olyan programot, amely a 'kurzor le' billentyű megnyomására a képernyőt 1 sorral lefelé lépteti úgy, hogy a felső sor helyére üres sort ír!

5.29 Öldjuk meg az előző 4 feladatot úgy, hogy a kilépő sort ill. oszlopot az ellenkező oldalon újra beléptetjük!

5.30 Módosítsuk a képernyő tartalmát úgy, hogy minden nagybetűt kisbetűre cserélünk!

5.31 Módosítsuk a képernyő tartalmát úgy, hogy minden kisbetűt nagybetűre cserélünk!

5.32 Módosítsuk a képernyő tartalmát úgy, hogy minden nagybetűt kisbetűre és ugyanakkor minden kisbetűt nagybetűre cserélünk!

5.33 Írjuk tele a képernyőt sakktábla-szerűen felváltva zöld és barna négyzetekkel (az inverz space kiváló erre a célra)!

5.34 Készítsünk olyan 'sakktáblát', amelyen minden barna mezőt két zöld követ! Vigyázzunk a laphatároknál!

6. SZÁMOLJUNK TOVÁBB!

A következőkben a szorzással, az osztással és a négyzetgyökvonással összefüggő feladatokat tanulmányozunk.

6.1 Szükségünk van egy, a memória adott címén lévő egybyte-os szám kétszeresére, eredeti helyén. Írjunk olyan programot, amely ezt összeadás segítségével megoldja!

Ez így kissé nehézkes, főként nagyobb számok esetén, de szerencsére van egyszerűbb módja is. Az 1.3-1.6 feladatokban láthattuk, hogy a bináris alakban ábrázolt számok kétszeresét (ugyanúgy, mint a tízes számrendszerben ábrázolt számok tízszeresét) egyszerűen egy 0 számjegy hozzáírásával kaphatjuk meg; ezzel lényegében a számban minden számjegyet az eredetinél eggyel magasabb helyiértékre helyeztünk, s az utolsó helyiértékre (az egyesek helyére) 0-t írtunk. A szám ezáltal 'hosszabb' lesz, ábrázolásához az eddiginél 1-gyel több bitre van szükségünk, s ha a rendelkezésünkre álló ábrázolási tartomány korlátozott (mint például egy egybyte-os szám esetén), akkor erről az 1 bitről külön kell gondoskodnunk. Egybyte-os számok 2-vel való szorzásakor mindezeket egyetlen , az 1 bittel balra léptető ASL utasítás elvégzi, tehát: a legfelső (bal szélső, azaz 7-es) bitet a C-be írja, a többi bitet egy hellyel balra tolja, majd a legalsó (jobb szélső, azaz 0-s) bitbe 0-t ír. A többi, ehhez hasonló funkciót ellátó utasítással egyetemben több módon címezhető, így pld. a regiszterekbe töltés nélkül, saját helyén is elvégezhetjük egy szám megkétszerezését.

6.2 Oldjuk meg a 6.1 feladatot az ASL utasítás segítségével!

```
;put"@0:a16"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* szám megduplázása                            *
;************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a16                 ;szám:=szám*2
        asl szám
        rts
        .end
```

Kétbyte-os számok esetén nyilvánvalóan ugyanezt kell kétszer végrehajtanunk, egy 'apró' eltéréssel. Az alsó byte-tal ugyanazt kell tennünk, mint az előbb, viszont a felső byte 0. bitjébe értelemszerűen nem 0-t kell írnunk, hanem az alsó byte léptetésekor onnan a C jelzőbitbe került, eredetileg legfelső bitjét. A felső byte kialakításához tehát a következőket kell megvalósítanunk: a C-ben lévő értéket a 0.

bitbe írjuk, ezzel egyidejűleg minden bitet egy hellyel balra tolunk, és a 'kicsorduló' 7. bitet (mint ASL-nél) a C jelzőbitbe tesszük. Ezt a funkciót úgy is szemléltethetjük, hogy a C biten 'keresztül' balra forgatjuk a byte tartalmát, ezért az ezt megvalósító ROL utasítást balra forgató utasításnak is nevezzük.

6.3 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám kétszeresét, eredeti helyén!

```
:put"@0:a31"
:************************************************
:* aritmetika: egy adott címen lévő kétbyte-os  *
:* szám értékét szorozzuk meg 2-vel!            *
:************************************************
*=$6666
szám=$6650          ;a szám  címe
a31                 ;szám:=szám*2
        asl szám    ;alsó byte
        rol szám+1  ;felső byte
        rts
        .end
```

6.4 Az előző megoldás értelemszerű alkalmazásával határozzuk meg egy, a memória adott címén lévő négybyte-os szám kétszeresét, eredeti helyén!

Nagyon egyszerű a 2 hatványaival való szorzás programozása is. Lássunk erre is példákat!

6.5 Határozzuk meg egy, a memória adott címén lévő egybyte-os szám nyolcszorosát, eredeti helyén!

```
:put"@0:a17"
:************************************************
:* aritmetika: egy adott címen lévő egybyte-os  *
:* szám 8-cal szorzása                          *
:************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a17                 ;szám:=szám*8
        asl szám    ;szám=2*szám (kétszeres)
        asl szám    ;szám=2*szám (négyszeres)
        asl szám    ;szám=2*szám (nyolcszoros)
        rts
        .end
```

6.6 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám nyolcszorosát, eredeti helyén!

A 6.5 mintájára egyszerű a megoldás. Mi most mégis a 6.3-ban készített a31 rutin felhasználását mutatjuk be, egy háromszor lefutó ciklus alkalmazásával, mert ez példát szolgáltat a 2 magasabb hatványaival való szorzás mmegvalósítására.

```
;put"@0:a32"
;*************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékét szorozzuk meg 8-cal!             *
;*************************************************
*=$6666
szám=$6650          ;a szám  címe
a32                 ;szám:=szám*8
        ldx #3      ;index indul
ugr     jsr a31     ;szám duplázása
        dex         ;kell még?
        bne ugr     ;ha igen, ->vissza
        rts         ;vissza a BASIC-be
a31                 ;szám:=szám*2
        asl szám    ;alsó byte
        rol szám+1  ;felső byte
        rts         ;vissza a hívás helyéhez
        .end
```

6.7 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám 32-szeresét, eredeti helyén!

A nem 2-hatvánnyal történő szorzás léptetéssel-forgatással való megoldása általában a szorzó értékétől függően, más-más megoldást igényel. Erre mutatunk most néhány példát.

6.8 Határozzuk meg egy, a memória adott címén lévő egybyte-os szám tízszeresét, eredeti helyén!

A szám 10-szeresét a szám kétszeresének és nyolcszorosának összegéből határozhatjuk meg.

```
;put"@0:a18"
;*************************************************
;* aritmetika: egy adott címen lévő egybyte-os   *
;* szám 10-zel szorzása                          *
;*************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a18                 ;szám:=szám*10
        lda szám    ;acc=szám
        asl a       ;acc=2*szám
        sta szám    ;szám=2*szám
        asl a       ;acc=4*szám
        asl a       ;acc=8*szám
        clc         ;carry=0
        adc szám    ;acc=10*szám
        sta szám    ;szám=10*szám
        rts
        .end
```

6.9 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám tízszeresét, s az eredményt írjuk egy másik címre!

```
;put"@0:a33"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékét szorozzuk meg 10-zel, s az ered- *
;* ményt írjuk egy ugyancsak adott címre!        *
;************************************************
*=$6666
szám=$6650          ;a szám  címe
eredm=$6652         ;az eredmény címe
másik=$6654         ;egy üres kétbyte-os hely a tárban
a33                 ;eredm:=szám*10
        jsr a31     ;szám duplázása
        lda szám    ;acc=<szám
        sta másik   ;<másik=<szám
        lda szám+1  ;acc=>szám
        sta másik+1 ;>másik=>szám
        jsr a31     ;szám duplázása
        jsr a31     ;szám duplázása
        jsr a29     ;eredm=szám+másik
        rts         ;vissza a BASIC-hez
a29                 ;eredm:=szám+másik
        clc         ;carry=0
        lda szám    ;acc=<szám
        adc másik   ;acc=<szám+<másik
        sta eredm   ;<eredm=<szám+<másik
        lda szám+1  ;acc=>szám
        adc másik+1 ;acc=>szám+>másik
        sta eredm+1 ;>eredm=>szám+>másik
        rts         ;vissza a hívás helyéhez
a31                 ;szám:=szám*2
        asl szám    ;alsó byte
        rol szám+1  ;felső byte
        rts         ;vissza a hívás helyéhez
        .end
```

6.10 Határozzuk meg egy, a memória adott címén lévő egybyte-os szám 15-szörösét, eredeti helyén!

Itt és a következő feladatoknál több lehetőséget mutatunk a megoldásra (az igénybe vett tárhelyek számától, a verem használatától, a 15-szörös érték kialakításától stb. függően). Ezek alapján újabb, jobb megoldásokat is készíthetünk.

Most a 15-szörös értéket az 1-, a 2-, a 4- és a 8-szoros érték összegéből képezzük; a tárból csak egy helyet veszünk igénybe, az eredetit; a részeredmények átmeneti tárolására a vermet használjuk.

```
;put"@0:a19"
;************************************************
;* aritmetika: egy adott címen lévő egybyte-os  *
;* szám 15-tel szorzása                         *
;************************************************
*=$6666
szám=$6650          ;az egybyte-os szám címe
a19                 ;szám:=szám*15
        lda szám    ;acc=szám
        asl a       ;acc=2*szám
        pha         ;verembe 2*szám
        asl a       ;acc=4*szám
        pha         ;verembe 4*szám
        asl a       ;acc=8*szám
        clc         ;carry=0
        adc szám    ;acc=9*szám
        sta szám    ;szám=9*szám
        pla         ;acc=4*szám
        clc         ;carry=0
        adc szám    ;acc=13*szám
        sta szám    ;szám=13*szám
        pla         ;acc=2*szám
        clc         ;carry=0
        adc szám    ;acc=15*szám
        sta szám    ;szám=15*szám
        rts
        .end
```

6.11 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám 15-szörösét, eredeti helyén!

Most a szám 15-szörösét a 16-szorosának és a számnak a különbségéből alakítjuk ki; az eredeti tárhelyen kívül még két másikat veszünk igénybe. Két, már előzőleg megírt rutinra is építünk: a 4.22 feladat a30-as rutinjára, amely két, adott címeken lévő kétbyte-os szám különbségét egy harmadik címre írja, és az a31-esre (ld. 6.3 feladat), amely egy adott helyen szereplő kétbyte-os számot eredeti helyén megdupláz. A vermet adatok tárolására nem használjuk.

```
;put"@0:a34"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékét szorozzuk meg 15-tel!            *
;************************************************
*=$6666
szám=$6650          ;a szám címe
eredm=$6652         ;egy üres kétbyte-os hely a tárban
másik=$6654         ;egy másik üres kétbyte-os hely a tárban
a34                 ;szám:=szám*15/i.
        lda szám    ;acc=<szám
        sta másik   ;<másik=<szám
        lda szám+1  ;acc=>szám
        sta másik+1 ;>másik=>szám
        jsr a31     ;szám duplázása (*2)
        jsr a31     ;szám duplázása (*4)
        jsr a31     ;szám duplázása (*8)
        jsr a31     ;szám duplázása (*16)
        jsr a30     ;eredm=szám-másik
        lda eredm   ;acc=<eredm
        sta szám    ;<szám=<eredm
        lda eredm+1 ;acc=>eredm
        sta szám+1  ;>szám=>eredm
        rts         ;vissza a BASIC-hez
a31                 ;szám:=szám*2
        asl szám    ;alsó byte
        rol szám+1  ;felső byte
        rts         ;vissza a hívás helyére
a30                 ;eredm:=szám-másik
        sec         ;carry=1
        lda szám    ;acc=<szám
        sbc másik   ;acc=<szám-<másik
        sta eredm   ;<eredm=<szám-<másik
        lda szám+1  ;acc=>szám
        sbc másik+1 ;acc=>szám->másik
        sta eredm+1 ;>eredm=>szám->másik
        rts         ;vissza a hívás helyére
        .end
```

6.12 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám 15-szörösét, és írjuk egy adott címre!

A 15-szörös érték kialakítását 6.10-zel egyezően végezzük; az eredeti tárhelyen kívül még két másikat veszünk igénybe. Most az a31-es rutin (kétbyte-os szám helyben duplázása) mellett az a29-est is (ld. 4.20 feladat) használjuk: ez utóbbi két, adott címeken lévő kétbyte-os szám összegét egy harmadik címre írja. A vermet adatok tárolására most sem használjuk.

```
;put"@0:a35"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os   *
;* szám értékét szorozzuk meg 15-tel, s az ered- *
;* ményt írjuk egy ugyancsak adott címre!        *
;************************************************
*=$6666
szám=$6650          ;a szám címe
eredm=$6652         ;az eredmény helye a tárban
másik=$6654         ;egy üres kétbyte-os hely a tárban
a35                 ;eredm:=szám*15/i.
        lda szám    ;acc=<szám
        sta másik   ;<másik=<szám
        lda szám+1  ;acc=>szám
        sta másik+1 ;>másik=>szám
        jsr a31     ;szám=2*szám
        jsr a29     ;eredm=szám+másik (=3*)
        jsr a31     ;szám=2*szám (4*)
        lda eredm   ;acc=<eredm
        sta másik   ;<másik=<eredm
        lda eredm+1 ;acc=>eredm
        sta másik+1 ;>másik=>eredm
        jsr a29     ;eredm=szám+másik (=7*)
        jsr a31     ;szám=2*szám (8*)
        lda eredm   ;acc=<eredm
        sta másik   ;<másik=<eredm
        lda eredm+1 ;acc=>eredm
        sta másik+1 ;>másik=>eredm
        jsr a29     ;eredm=szám+másik (=15*)
        rts
a31                 ;szám:=szám*2
        asl szám    ;alsó byte
        rol szám+1  ;felső byte
        rts
a29                 ;eredm:=szám+másik
        clc         ;carry=0
        lda szám    ;acc=<szám
        adc másik   ;acc=<szám+<másik
        sta eredm   ;<eredm=<szám+<másik
        lda szám+1  ;acc=>szám
        adc másik+1 ;acc=>szám+>másik
        sta eredm+1 ;>eredm=>szám+>másik
        rts
        .end
```

6.1. Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám 15-szörösét, és írjuk egy adott címre!

```
:put"@0:a36"
:*************************************************
:* aritmetika: egy adott címen lévő kétbyte-os   *
:* szám 15-szörösét írjuk egy adott címre!       *
:*************************************************
*=$6666
szám=$6650          ;a szám címe
eredm=$6652         ;az eredmény helye a tárban
a36                 ;eredm:=szám*15/ii.
        lda szám+1  ;acc=>szám
        pha         ;>1*
        lda szám    ;acc=<szám
        pha         ;<1*
        jsr a31     ;szám=2*szám (2*)
        lda szám+1  ;acc=>szám
        pha         ;>2*
        lda szám    ;acc=<szám
        pha         ;<2*
        jsr a31     ;szám=2*szám (4*)
        lda szám+1  ;acc=>szám
        pha         ;>4*
        lda szám    ;acc=<szám
        pha         ;<4*
        jsr a31     ;szám=2*szám (8*)
        clc         ;c=0
        pla         ;<4*
        adc szám    ;<12*
        sta szám    ;<12*
        pla         ;>4*
        adc szám+1  ;>12*
        sta szám+1  ;>12*
        clc         ;c=0
        pla         ;<2*
        adc szám    ;<14*
        sta szám    ;<14*
        pla         ;>2*
        adc szám+1  ;>14*
        sta szám+1  ;>14*
        clc         ;c=0
        pla         ;<1*
        adc szám    ;<15*
        sta eredm   ;helyére
        pla         ;>1*
        adc szám+1  ;>15*
        sta eredm+1 ;helyére
        rts         ;vissza a BASIC-hez
a31                 ;szám:=szám*2
        asl szám    ;alsó byte
        rol szám+1  ;felső byte
        rts         ;vissza a hívás helyére
        .end
```

Mint látjuk, a 15-szörös érték kialakítását itt is 6.10-zel egyezően végezzük; csak az eredeti tárhelyet vesszük igénybe, de a vermet használjuk, s csak az a31-es rutinra építünk.

Az eddigi tapasztalatok alapján rövidebb megoldást is tudunk mutatni:

6.11 Határozzuk meg egy, a memória adott címén lévő egybyte-os szám 15-szörösét, eredeti helyén!

A verem használatával és ciklusok alkalmazásával, más tárhely igénybevétele nélkül működik a következő program:

```
;put"@0:a37"
;**************************************************
;* aritmetika: egy adott cimen lévő kétbyte-os    *
;* szám értékét szorozzuk meg 15-tel!             *
;**************************************************
*=$6666
szám=$6650          ;a szám címe
a37                 ;szám:=szám*15/ii.
        ldx #3      ;index indul
ugra    lda szám+1
        pha         ;felső byte
        lda szám
        pha         ;alsó byte
        asl szám
        rol szám+1  ;duplázás
        dex         ;x:=x-1
        bne ugra    ;ha még kell, vissza
        ldx #3      ;index indul
ugrb    clc         ;carry=0
        pla         ;alsó byte
        adc szám
        sta szám    ;helyére
        pla         ;felső byte
        adc szám+1
        sta szám+1  ;helyére
        dex         ;x:=x-1
        bne ugrb    ;ha még kell, vissza
        rts
        .end
```

Térjünk át az osztásra! Ehhez - a szorzásnál mondottak értelemszerű alkalmazásával - szükségünk van a jobbra léptető LSR és a jobbra forgató ROR utasításokra, amelyek (az eltérő mozgatási iránynak megfelelő módosítással) mindenben megfelelnek az ASL ill. a ROL utasításoknak. Világos tehát, hogy egy egybyte-os szám jobbra léptetésekor a szám új értéke az eredeti fele lesz, a 2-vel való osztáskor keletkezett maradék pedig a C bitbe kerül.

6.15 Felezzünk meg egy, a memória adott címén lévő egybyte-os számot!

```
:put"@0:a20"
:*************************************************
:* aritmetika: egy adott címen lévő egybyte-os   *
:* szám megfelezése                              *
:*************************************************
*=$6666
szám=$6650          :az egybyte-os szám címe
a20                 :szám:=szám/2
        lsr szám
        rts
        .end
```

6.16 Felezzünk meg egy, a memória adott címén lévő egybyte-os számot, s a C jelzőbit állásától függően írjuk ki a képernyőre, hogy az eredeti szám páros volt-e vagy páratlan!

6.17 Felezzünk meg egy, a memória adott címén lévő kétbyte-os számot!

```
:put"@0:a38"
:*************************************************
:* aritmetika: egy adott címen lévő kétbyte-os   *
:* szám értékét osszuk el 2-vel!                 *
:*************************************************
*=$6666
szám=$6650          :a szám  címe
a38                 :szám:=szám/2
        lsr szám+1  :felső byte
        ror szám    :alsó byte
        rts
        .end
```

6.18 Határozzuk meg egy, a memória adott címén lévő egybyte-os szám nyolcadrészét!

Nyilvánvalóan csak az egész osztásról lehet szó az LSR utasítás ismételt alkalmazásával.

```
:put"@0:a21"
:*************************************************
:* aritmetika: egy adott címen lévő egybyte-os   *
:* szám 8-cal osztása                            *
:*************************************************
*=$6666
szám=$6650          :az egybyte-os szám címe
a21                 :szám:=szám/8
        lsr szám    :szám=szám/2 (fele)
        lsr szám    :szám=szám/2 (negyede)
        lsr szám    :szám=szám/2 (nyolcada)
        rts
        .end
```

6.19 Határozzuk meg egy, a memória adott címén lévő egybyte-os szám nyolcadrészét, és írjuk ki az osztási maradékot is!

Figyeljük az egyes jobbra léptetések után keletkező maradékokat!

6.20 Határozzuk meg egy, a memória adott címén lévő kétbyte-os szám nyolcadrészét!

```
:put"@0:a39"
:************************************************
:* aritmetika: egy adott címen lévő kétbyte-os  *
:* szám értékét osszuk el 8-cal!                *
:************************************************
*=46666
szám=$6650          :a szám  címe
a39                 :szám:=szám/8
        jsr a38     :szám=szám/2 (fele)
        jsr a38     :szám=szám/2 (negyede)
        jsr a38     :szám=szám/2 (nyolcada)
        rts         :vissza a BASIC-hez
a38                 :szám:=szám/2
        lsr szám+1  :felső byte
        ror szám    :alsó byte
        rts         :vissza a hívás helyére
        .end
```

6.21 Határozzuk meg két, a memória adott címén lévő egybyte-os szám szorzatát, s az eredményt írjuk egy adott címre! Vigyázzunk, a szorzat kétbyte-os is lehet!

Ennek, a bármilyen két egybyte-os szám összeszorzására alkalmas programnak az elkészítéséhez vegyük észre, hogy két szám, EGYIK és MÁSIK szorzata a következőképpen számítható:

EGYIK*MÁSIK = EGYIK , ha MÁSIK=1
EGYIK*MÁSIK = (2*EGYIK) * (MÁSIK/2) , ha MÁSIK páros
EGYIK*MÁSIK = EGYIK + EGYIK*(MÁSIK-1) , ha MÁSIK páratlan.

Ezek alapján MÁSIK értékét (akár páros, akár páratlan) egyre csökkentve eljuthatunk a végeredményhez (természetesen közben EGYIK-et is változtatjuk), de csak összeadást, 2-vel való szorzást ill. páros MÁSIK esetén ugyancsak 2-vel való osztást és dekrementálást alkalmazva. Az algoritmus:

```
EREDM:=0
Ciklus, amíg MÁSIK>0
   Ha MÁSIK páros, akkor EGYIK:=2*EGYIK
                         MÁSIK:=MÁSIK/2
                 különben EREDM:=EREDM+EGYIK
                         MÁSIK:=MÁSIK-1
   Elágazás vége
Ciklus vége
```

A kódolt program:

```
;put"@0:a40"
;*************************************************
;* aritmetika: egy adott címen lévő egybyte-os   *
;*szám értékét szorozzuk meg egy másik egybyte-os*
;* számával, s az eredményt írjuk egy ugyancsak  *
;* adott helyre (a szorzat kétbyte-os is lehet!) *
;*************************************************
*=$6666
egyik=$6650             ;az egyik egybyte-os szám címe
másik=$6652             ;a másik egybyte-os szám címe
eredm=$6654             ;az eredmény címe (két byte!)
a40                     ;eredm:=egyik*másik
        lda #0          ;eredm nullázása
        sta eredm
        sta eredm+1
        sta egyik+1     ;egyik felső byte-ja nullázása
cikl                    ;ciklus, amíg másik>0
        lda másik       ;másik vizsgálata
        beq vége        ;ha =0, -> vége
        and #%00000001  ;másik páros-e?
        bne ptl         ;ha nem, ->ptl
páros                   ;másik páros esetén
        asl egyik       ;egyik=2*egyik : alsó,
        rol egyik+1     ;majd felső byte
        lsr másik       ;másik=másik/2 :másik felezése
        jmp cikl        ;páros ág vége
ptl                     ;másik páratlan esetén
        clc             ;carry=0
        lda eredm       ;eredm:=eredm+egyik
        adc egyik
        sta eredm       ;alsó byte
        lda eredm+1
        adc egyik+1
        sta eredm+1     ;felső byte
        dec másik       ;másik:=másik-1
        jmp cikl        ;páratlan ág vége
vége                    ;másik=0 esetén
        rts             ;vissza a BASIC-hez
        .end
```

↵ . 22 Ehhez hasonló gondolatmenet alkalmazásával készítsünk hatványozó programot, amely kiszámítja EGYIK↑MÁSIK értékét!

Figyeljük meg, hogy

EGYIK↑MÁSIK = 1 , ha MÁSIK=0
EGYIK↑MÁSIK = (EGYIK↑2) ↑ (MÁSIK/2) , ha MÁSIK páros
EGYIK↑MÁSIK = EGYIK * [EGYIK↑(MÁSIK-1)] , ha MÁSIK páratlan
!

Ennek alapján készítsük el az algoritmust, majd a programot.

6.23 Két, adott címen lévő kétbyte-os szám hányadosát írjuk egy adott címre, az osztáskor keletkezett maradékot pedig az első szám helyére (maradékos osztás)!

```
;put"@0:a41"
;************************************************
;* aritmetika: egy adott címen lévő kétbyte-os  *
;* szám értékét osszuk el egy másik kétbyte-os  *
;* számával, s az eredményt írjuk egy ugyancsak *
;* adott helyre!                                *
;************************************************
*=$6666
egyik=$6650          ;az egyik kétbyte-os szám címe
másik=$6652          ;a másik kétbyte-os szám címe
eredm=$6654          ;az eredmény címe (két byte!)
a41                  ;eredm:=int(egyik/másik)
                     ;(azaz: a hányados)
                     ;egyik:=egyik-másik*eredm
                     ;(azaz: a maradék)
        lda #0       ;a hányados nullázása
        sta eredm    ;alsó, majd
        sta eredm+1  ;felső byte
kivon                ;eredmény számlálása
        inc eredm    ;alsó byte
        bne ugri     ;ha nem 0, kész
        inc eredm+1  ;felső byte növelése
ugri                 ;egyik:=egyik-másik
        sec          ;carry=1
        lda egyik    ;acc=<egyik
        sbc másik    ;acc=<egyik-<másik
        sta egyik    ;<egyik=<egyik-<másik
        lda egyik+1  ;acc=>egyik
        sbc másik+1  ;acc=>egyik->másik
        sta egyik+1  ;>egyik=>egyik->másik
vizsg                ;egyik<0?
        bcs kivon    ;még nem, tovább
had                  ;igen, egyszer vissza kell adni
                     ;eredm:=eredm-1
        lda eredm    ;<eredm=0?
        bne ugrd     ;nem, ->ugrd
        dec eredm+1  ;>eredm=>eredm-1
ugrd
        dec eredm    ;<eredm=<eredm-1
                     ;egyik:=egyik+másik
        clc          ;carry=0
        lda egyik    ;acc=<egyik
        adc másik    ;acc=<egyik+<másik
        sta egyik    ;<egyik=<egyik+<másik
        lda egyik+1  ;acc=>egyik
        adc másik+1  ;acc=>egyik+>másik
        sta egyik+1  ;>egyik=>egyik+>másik
        rts
        .end
```

A megjegyzésekből világos az algoritmus: EGYIK=ből addig vonjuk ki MÁSIK-at, amíg EGYIK negatívvá nem válik, majd egyszer 'visszaadjuk', s a műveletek számából ismerjük a hányadost, EGYIK-ben pedig éppen a maradékot találjuk.

6.24 Határozzuk meg egy adott címen lévő kétbyte-os szám négyzetgyökének egész részét!

```
:put"@0:a42"
;************************************************
:* aritmetika: határozzuk meg egy adott címen    *
;* lévő kétbyte-os szám négyzetgyökét!           *
;************************************************
*=$6666
szám=$6650          :a kétbyte-os szám címe
eredm=$6652         :az eredmény címe
egyik=$6654         :az egyik segédváltozó
másik=$6656         :a másik segédváltozó
a42                 :eredm:=int(sqr(szám))
        lda #0      :az eredmény
        sta eredm   ;nullázása
        lda #1      :1-re állítjuk
        sta egyik   :egyiket és
        sta másik   :másikat
cikl                :ciklus eleje
vizsg               :ha szám<egyik, akkor vége
        lda szám+1  ;felső byte-ok
        cmp egyik+1 ;összehasonlítása
        bcc vége    ;szám<egyik, vége
        bne tovább  ;szám>egyik, tovább
        lda szám    :alsó byte-ok
        cmp egyik   :összehasonlítása
        bcc vége    ;szám<egyik, vége
tovább              ;szám>=egyik:ciklus tovább
        inc eredm   :eredm:=eredm+1 - csak 1 byte!
                    ;másik:=másik+2
        clc         :carry=0
        lda másik   :acc=<másik
        adc #2      :acc=<másik+2
        sta másik   :<másik=<másik+2
        bcc ugr     :ha nincs átvitel:->ugr
        inc másik+1 ;van, >másik:=>másik+1
ugr                 :egyik:=egyik+másik
        clc         :carry=0
        lda egyik   :acc=egyik
        adc másik   :acc=egyik+másik
        sta egyik   :egyik=egyik+másik
        lda egyik+1 ;acc=>egyik
        adc másik+1 :acc=>egyik+>másik
        sta egyik+1 :>egyik=>egyik+>másik
        jmp cikl    ;vissza a ciklus elejére
vége    rts
        .end
```

6. SZÁMOLJUNK TOVÁBB!

Két segédváltozóra lesz szükségünk, EGYIK-re és MÁSIK-ra. Olyan EREDM értéket keresünk, amelyre igaz, hogy

EREDM↑2 <= SZÁM < (EREDM+1)↑2 .

Felhasználjuk azt az (egyébként könnyen igazolható) összefüggést, amely szerint a k. négyzetszám az első k páratlan szám összegével egyenlő. Az EREDM szám értéke 1-gyel kisebb annál, amelyik természetes számnál tartunk a vizsgálódásban (gondoljuk meg: a kilépés után épp erre van szükségünk), MÁSIK az aktuális páratlan számot jelenti, EGYIK pedig a négyzetszámot (EREDM+1 négyzetét), ami tehát nem más, mint az eddigi páratlan számok összege. Ennek ismeretében algoritmusunk a következő:

```
EREDM:=0
EGYIK:=1
MÁSIK:=1
Ciklus, amíg EGYIK<=SZÁM
   EREDM:=EREDM+1
   MÁSIK:=MÁSIK+2
   EGYIK:=EGYIK+MÁSIK
Ciklus vége
```

6.25 A 6.22 feladat megoldásához szükséges a négyzetreemelés. Írjuk át úgy, hogy a fenti algoritmust használjuk!

7. MI MINDENRE HASZNALHATJUK?

Ebbe a fejezetbe olyan aprób feladatokat gyűjtöttünk össze, amelyek az eddigieken tól példát mutatnak a gép különböző lehetőségeihez való hozzáférésre.

7.1 Tegyük a képernyőtárba a képernyő bal felső sarkától a tárban a SZöVEG cimtől kezdve lerakott, 0 byte-tal záródó szöveget!

A feladatban mindössze az a kis probléma van, hogy a szöveg a tárban ASCII-kódban van, a képen pedig képernyőkódban kell lennie. de ez egy apró átalakitás segitségével megoldható.

```
:put "@0:ke0"
:************************************************
:*  a tárban, ascci kódban tárolt szöveget      *
:*  képernyőkóddá alakitva irja a képernyőre    *
:************************************************
*=$6666
kép = $0c00
        ldy #0          ;index indul
start
        lda szöveg,y    ;szöveg aktuális byte-ja
        beq vége        ;ha =0, ->vége
        and #$3f        ;képernyőkód
        sta kép,y       ;képre teszi
        iny             ;köv. byte-ra mutat
        jmp start       ;vissza, tovább
vége    rts             ;vissza a BASIC-be
szöveg .byte 'jo napot!',0
        .end
```

Itt jegyezzük meg, hogy amikor a PRINT BASIC-utasítás segitségével irunk szöveget a képre, akkor valami ehhez hasonló módon történik az átalakítás.

7.2 Tegyük a tárba a SZöVEG címtől kezdve a képernyő bal felső sarkában lévő, *-gal záródó szöveget!

Itt az előző feladatnak éppen a fordítottjáról van szó, mert képernyőkódban lévő szöveget kell ASCII-kódóvá alakítani. Ilyen feladat adódik az INPUT BASIC-utasítás végrehajtásakor is.

```
:put "@0:ke1"
:************************************************
:*  a képernyő kezdetén tárolt, legfeljebb 256  *
:*  karakter hosszó, csillaggal záródó szöveget *
:*       ascii kódóvá alakitva irja a tárba,    *
:*         a 'szöveg' cimtől kezdődően          *
:************************************************
```

```
*=$6666
kép = $0c00          ;képernyő kezdete
       ldy #0        ;index indul
start
       lda kép,y     ;a szöveg aktuális byte-ja
       cmp '*'       ;csillag-e?
       beq vége      ;ha igen, ->vége
       ora #$40      ;ascii kód
       sta szöveg,y  ;a memóriába írja
       iny           ;köv. byte-ra mutat
       jmp start     ;vissza, tovább
vége
       lda #0        ;a szöveg végére
       sta szöveg,y  ;0-t ir
       rts           ;vissza a BASIC-be
szöveg *=*+255       ;hely lefoglalása
       .end
```

7_3 Írjuk ki eredeti sorrendjükben az abc betűit a képernyő bal felső sarkától kezdődően!

```
;put"@0:abc1"
;*****************************************************
;*  az egyes képernyőkódoknak megfelelő betűket      *
;*        abc sorrendben a képernyőre írja           *
;*****************************************************
*=$6666
kép=$0c00            ;a képernyőtár kezdete
abc1                 ;betűk abc-ben a képre
       ldy #1        ;az első betű kódja
ciklus               ;a ciklus kezdete
       tya           ;a:=y
       sta kép-1,y   ;a képernyőre
       iny           ;y:=y+1
       cpy #27       ;van még?
       bmi ciklus    ;igen, vissza
       rts           ;nincs, vége
       .end
```

Figyeljük meg: a TYA utasításra azért van szükség, mert nincs STY KÉP-1,Y utasítás, azaz saját magával nem indexelhető az indexregiszterből másoló utasítás (sőt, az X regiszter is csak nullalapos címzés esetén indexelhető Y-nal)!

7_4 Írjuk ki az előzőhöz hasonló módon az összes karaktert a képernyő kezdetétől!

Ez az előzőtől csak annyiban különbözik, hogy Y értékét egy egész lapon keresztül kell futtatnunk.

```
:put"@0:abc2"
:*************************************************
:* az egyes képernyőkódoknak megfelelő karakte-  *
:*   reket növekvő sorrendben a képernyőre írja  *
:*************************************************
*=$6666
kép=$0c00           :a képernyőtár kezdete
abc2                :karakterek abc-ben a képre
        ldy #0      :az első karakter
ciklus              ;a ciklus kezdete
        tya         ;a:=y
        sta kép,y   :a képernyőre
        iny         :y:=y+1
        bne ciklus  :ha még tart, vissza
        rts         ;már nincs, vége
        .end
```

7.35 Írjuk ki a BASIC-kulcsszavakat a képernyő kezdetétől!

Mint az 1.21 feladatban láttuk, a kulcsszavak a $818e-től $8381-ig terjedő területen helyezkednek el a tárban, így feladatunk ennek a tartománynak a képernyőre másolása. Ez nem egészen két lapnyi terület, s mint a képernyő csillagokkal való feltöltésekor láttuk, érdemesebb egy-két utasítást néhányszor esetleg feleslegesen végrehajtatnunk, ha a program ezáltal egyszerűbbé válik. Most is inkább ezt a módszert választjuk.

```
:put"@0:abc3"
:*************************************************
:*a BASIC-kulcsszavakat - tárbeli sorrendjükben  *
:*a képernyőre írja                              *
:*************************************************
*=$6666
képe=$0c00              :a képernyőtár eleje
kte=$818e               :a kulcsszótáblázat kezdete
ktv=$8382               :a kulcsszótáblázat vége+1
hossz=ktv-kte           :a kulcsszótáblázat hossza
képv=képe+hossz         ;a képernyőtár vége
abc3                    :kulcsszavak a képre
        ldy #0          :index indul
ciklus                  ;a ciklus kezdete
        lda kte,y       :1 byte a táblázat első feléből
        sta képe,y      :a képernyőre
        lda ktv-$100,y  ;1 byte a táblázat 2. feléből
        sta képv-$100,y ;a képernyőre
        dey             :y:=y-1
        bne ciklus      :még van, vissza
        rts             :nincs, vége
        .end
```

7.6 Írjunk az 1. időzítő (#ff00) alsó byte-jából egy 0 és 7 közötti 'véletlenszámot'!

A kapott érték nyilván nem véletlenszám, hiszen az időzítő meghatározott rendszer alapján veszi fel különböző értékeit (egyesével csökken az értéke), mégis véletlennek hat, mert véletlenszerű időpillanatokban fordulunk hozzá.

A programból való kilépést a STOP billentyű lenyomását figyelő elágazás segítségével oldottuk meg.

```
:put"@0:v0"
;************************************************
:*  egy 0...7 közötti véletlenszámot ír ki      *
;************************************************
*=$6666
stop=$91            :a stop billentyűt figyeli
timer=$ff00         :az 1. időzítő alsó byte-ja
cls=$93             :képtörlés kódja
chrout=$ffd2        :1 byte kiírása
        lda #cls
        jsr chrout  :a képernyő törlése
ciklus
        lda timer   :'véletlen' byte
        and #07     :alsó 3 bitje
        ora #$30
        jsr chrout  ;kiírása
        lda stop    ;volt stop?
        bmi ciklus  ;nem, tovább
        rts         ;vissza a BASIC-be
        .end
```

7.7 Írjuk át az előző feladatot az 1. időzítő felső byte-jára ill. a belső óra ($a3-$a4-$a5, ld. 1.11 feladat) egyes byte-jaira! Mit tapasztalunk a 'véletlenszámok' eloszlásával kapcsolatban?

7.8 Írjuk ki az 1. időzítő (#ff00) alsó byte-jának értékét folyamatosan a képernyő első 256 byte-jára, de csak abban az esetben, ha az éppen egy számjegy kódja!

Tulajdonképpen a gyorsan csökkenő számláló értékeit 'elhúzva' írunk a képernyőre. Hogy a látvány számunkra is érzékelhetővé váljék, beiktattunk egy lassító ciklust. A programból a 'kurzor jobbra' billentyű segítségével tudunk kilépni.

```
;put"@0:vek2"
;************************************************
;* a képernyőre írja az 1. időzítő alsó byte-ját,*
;*    ha az éppen egy számjegy kódja             *
;************************************************
```

```
*=$6666
kep=$0c00           ;a képernyőtár kezdete
timer1=$ff00        ;1. időzítő alsó byte-ja
bill=$c6            ;billentyű lenyomását jelzi
vek2                ;timer1 a képre (elhúzva)
        ldy #0      ;index indul
ciklus lda timer1
        cmp #$30    ;0-t megelőzi?
        bmi ciklus  ;igen, vissza
        cmp #$3a    ;9 után jön?
        bpl ciklus  ;igen, vissza
        sta kep,y   ;számjegy a képre
        ldx #0      ;lassító
var     dex         ;ciklus
        bne var
        iny         ;y:=y+1
        bne ciklus  ;még van, tovább
        lda bill    ;billentyűt figyeli
        cmp #$33    ;kurzor jobbra?
        bne vek2    ;nem, ismét
        rts
        .end
```

7.9 Írjuk át az előző feladatot az 1. időzítő felső byte-jára ill. a belső óra ($a3-$a4-$a5, ld. 1.11 feladat) egyes byte-jaira! Mit tapasztalunk a 'véletlenszámok' eloszlásával kapcsolatban?

7.10 Készítsünk olyan programot, amely kiírja, hogy éppen melyik grafikus módban vagyunk!

A grafikus módok a $83-as címről olvashatók ki, ennek a byte-nak az értéke az aktuális grafikus mód függvényében ugyanis az alábbi:

```
00000000 karakteres képernyő
00100000 nagyfelbontású grafika
01100000 nagyfelbontású grafika + karakteres képernyő
10100000 többszínű grafika
11100000 többszínű grafika + karakteres képernyő.
```

Világosan látszik tehát, hogy a legfelső bit mutatja a többszínű grafika, az 5. bit általában a grafikus üzemmód, a 6. bit pedig az osztott képernyő bekapcsolt voltát. Ezeket a biteket kell tehát megvizsgálnunk. Ennek legegyszerűbb módja: a BIT utasítás használata. Ez az utasítás a kérdéses byte 7. bitjét N-be, 6. bitjét V-be másolja, és Z értékét az akkumulátor és a kérdéses byte közötti bitenkénti AND művelet eredményétől függően a szokásos módon állítja be. Mivel (Z, N és C szerint) többszörös elágazásról lesz szó, s ezeket csak egyenként tudjuk lekérdezni, kézenfekvő, hogy az első vizsgálat után – a további elágazást biztosítandó – mentsük el a jelzőbitek értékét. A feltételregiszter verembe mentését a PHP, a veremből töltését pedig a PLP utasítás hajtja végre.

```
;put"@0:g1"
;*************************************************
;*   grafika: irjuk ki a képernyőre, hogy épp  *
;*   melyik grafikus módban vagyunk!           *
;*************************************************
*=$6666
primm=$ff4f          ;szövegkiíró rutin
grmod=$83            ;grafikus mód jelző
g1                   ;melyik grafikus módban vagyunk?
        lda #$ff     ;A feltöltése
        bit grmod    ;a jelző vizsgálata
        bmi kar      ;karakteres
        php          ;elmentjük a jelzőbiteket
        bmi multi    ;többszínű grafika
                     ;nagyfelbontású grafika
        jsr primm    ;különben ki szöveg:
        .byte 'nagyfelbontasu ',0
        jmp grat     ;tovább
multi                ;többszínű grafika
        jsr primm    ;multi: ki szöveg:
        .byte 'tobbszinu ',0
grat                 ;grafikus módok
        jsr primm    ;ki szöveg:
        .byte 'grafika ',0
        plp          ;jelzőbitek vissza
        bvc vége     ;ha v=0, nincs szöveg, vége
        jsr primm    ;különben ki szöveg:
        .byte 'es ',0
kar                  ;szöveg van
        jsr primm    ;ki szöveg:
        .byte 'karakteres képernyő ',0
vége    rts
        .end
```

7.11 Írjuk ki a grafikus képernyőre a ROM-ban lévő karakterkészletet!

Tudvalévő, hogy a gépben lévő két karakterkészlet $d000-$dfff-ig tart, így ezt a területet kell átmásolnunk a grafikus képernyő $2000-től kezdődő területére.
Előtte még néhány értéket be kell állítanunk: a TED $ff06, $ff12 és $ff14 regiszterének értékét a grafikus módba való átíréshez és a szín-, fényerő- és képmemória kezdetének beállításához. Felhasználjuk az SCNCLR BASIC-utasítás gépben lévő rutinját, s az INNEN és az IDE címkék után következő LDA és STA utasítások operandusainak folyamatos átírásával biztosítjuk a kérdéses tárterület lefedését.

```
;put"@0:g2"
;*************************************************
;*   grafika: irjuk ki a grafikus képernyőre a   *
;*   rom-ban lévő karakterkészletet!             *
;*************************************************
```

```
*=$6666
scnclr=$c566            ;scnclr belépési pontja
kép=$2000               ;a graf. képernyő kezdete
kargen=$d000            ;a karakterkészlet kezdete
grmod=$83               ;grafikus mód jelző
g2                      ;karakterkészlet
        lda $ff06       ;áttérés
        ora #%00100000  ;bittérképes
        sta $ff06       ;üzemmódra
        lda $ff12       ;a bittérkép kezdete
        and #%00000011  ;és a rom/ram váltó
        ora #%00001000  ;nullázása, $20 az új
        sta $ff12       ;bittérkép kezdete
        lda ##$18       ;szín- és fényerőmemória
        sta $ff14       ;kezdete: $1800
        lda ##$60       ;nagyfelbontású grafika
        sta grmod       ;+szöveg beállítása
        jsr scnclr      ;képtörlés
        lda #<kargen    ;a programban lévő
        sta innen+1     ;mutató beállítása
        lda #>kargen    ;a karakterkészlet
        sta innen+2     ;kezdetére
        lda #<kép       ;a programban lévő
        sta ide+1       ;mutató beállítása
        lda #>kép       ;a képernyő
        sta ide+2       ;kezdetére
innen                   ;a karaktergenerátor aktuális
        lda kargen      ;byte-jából
ide                     ;a kép aktuális byte-ja
        sta kép         ;helyre
        inc ide+1       ;mutatók alsó byte-
        inc innen+1     ;jainak növelése
        bne innen       ;ha még nem 0, tovább
        inc ide+2       ;felső byte-
        inc innen+2     ;okat is növeljük
        ldx innen+2     ;karaktergenerátor
        cpx ##$d8       ;elérte a ##d8-at?
        bne innen       ;ha még nem, tovább
        rts
        .end
```

7.12 Sok játékprogram - igényeinek megfelelően - saját karakterkészletet használ, s ezt a RAM-ban tárolja (leggyakrabban a $1000 ill. a $3800 címtől kezdődően). Készítsünk ezeket a karakterkészleteket a grafikus képernyőre másoló programokat!

7.13 Írjunk olyan programot, amely minden billentyű lenyomására hangjelzést ad, mégpedig különböző billentyűk esetén különböző magasságú hangot!

A feladat megoldásához a már ismert $c6-os billentyűjelző byte-ot használjuk. A lenyomott billentyű $c6-ban érzékelt kódját megnégyszerezzük, majd az így kapott értéket írjuk a

hanggenerátorba. A programból kilépni a 'kurzor jobbra' gomb lenyomásával tudunk.

```
:put"@0:b1"
:*************************************************
:* a billentyűk lenyomására az 1. hanggenerator  *
:* hangjelzést ad                                *
:*************************************************
*=$6666
bill=$c6                ;billentyű lenyomását jelzi
hangv=$ff11             ;hangvezérlő regiszter
hangla=$ff0e            ;1. hanggen. 0...7. bit
b1                      ;a billentyűk lenyomására hangot ad
        lda bill
        cmp #$33        ;kurzor jobbra?
        beq vége        ;igen, kiszáll
        cmp #$40        ;más billentyűt nyomunk?
        beq ki          ;nem, kikapcsol
        asl a           ;szorzás
        asl a           ;4-gyel
        sta hangla      ;frekvencia beírása
        lda hangv       ;igen,
        ora #%00011111  ;1. gen. be, hangerő max.
        sta hangv       ;megszólal
        jmp b1          ;ismét, tovább
ki                      ;hang kikapcsolása
        lda hangv
        and #%11101111  ;1. gen. ki, hangerő max.
        sta hangv       ;elhallgat
        jmp b1          ;ismét, tovább
vége    rts             ;vissza a BASIC-be
        .end
```

7.14 Utolsó kidolgozott példánk egy egyszerű megszakítás-módosítás: írjunk olyan programot, amely (a Q, a control ill. a shift kivételével) bármelyik billentyű lenyomásának idejére a felső sort csillagokkal írja tele! A programtól elvárjuk, hogy gyakorlatilag bármikor aktív legyen, tehát pld. egy BASIC-program vagy egy (nem letiltott megszakítás alatt futó) gépi kódú program futása közben is használhassuk.

A $0314 címen lévő megszakításvektor - amely alapesetben a $ce0e címre mutat - átírásával a MEGSZ címre küldjük a megszakítást, ahol elvégeztetjük vele a kívánt feladatot, majd annak végeztével továbbküldjük a $ce0e címre. Ezért minden megszakításkor a gép elvégzi eredeti feladatait, csak előtte még a mi kérésünknek is eleget tesz.

Maga a program megírása nem jelent különösebb nehézséget, csupán arra kell vigyáznunk, hogy a megszakításvektor átírásának idejére tiltsuk le a megszakítást, mert könnyen elszállhat a programunk!

```
;put"@0:msz"
;************************************************
;*  megszakítás: bármikor - akár egy basic vagy  *
;*  gépi kódú program futása közben is - ,       *
;*  (a c=, a control ill. a shift kivételével)   *
;*  bármelyik billentyű lenyomásának idejére a   *
;*  felső sort csillagokkal írja tele.           *
;************************************************
*=$6666
kép=$0c00           ;a képernyőtár kezdete
mszv=$0314          ;a megszakításvektor helye
mszf=$ce0e          ;az eredeti megszakítás helye
bill=$c6            ;billentyűzet-jelző
mi                  ;példa a megszakítás átírására
        sei         ;megszakítás tiltása
        lda #<megsz ;megszak.
        sta mszv    ;vektor
        lda #>megsz ;átírása
        sta mszv+1
        cli         ;megszakítás engedélyezése
        rts         ;vissza a BASIC-be
megsz               ;új megsz.
        lda bill    ;billentyűt
        cmp #$40    ;nyomunk-e?
        beq nem     ;nem nyomunk, ->nem
        igen        ;billentyűt nyomunk
        lda '*'     ;* kódja az a-ba
töltés              ;töltés az adott karakterrel
        ldx #39     ;ennyi+1 hely van egy sorban
cik     sta kép,x   ;képre
        dex         ;x:=x-1
        bpl cik     ;ha még van, vissza
        jmp mszf    ;megszakítás tovább
nem                 ;nem nyomunk billentyűt
        lda ' '     ;space kódja az a-ba
        bne töltés  ;töltés - most space-szel
                    ;mindig ugrik!
      .end
```

7.15 Oldjuk meg megszakításból a 7.13 feladatot!

Befejezésül néhány vegyes feladat:

7.16 Fordítsuk meg egy byte bitjeinek sorrendjét!

7.17 Generáljunk véletlenszerűen öt lottószámot, s írjuk ki őket a képernyőre nagyság szerinti sorrendben!

7.18 Határozzuk meg az első 100 természetes szám összegét úgy, hogy az összeadáskor keletkező részletösszegeket is kiírja!

7.MI MINDENRE HASZNÁLHATJUK?

7.19 Készítsünk egy táblázatot a képernyőre! Tartalmazza 1-20-ig a természetes számokat, négyzetüket és köbüket!

7.20 Készítsünk egy másik táblázatot a képernyőre! Tartalmazza 1-20-ig a természetes számokat és faktoriálisukat!

7.21 Írjunk olyan programot, amely kiírja egy adott címen lévő számnak egy másik címen lévő számnál kisebb összes többszörösét!

7.22 Írjunk olyan programot, amely kiírja az 2 és 3 első 10 hatványát!

7.23 Írjunk olyan programot, amely kiírja az első 20 Fibonacci-számot!

7.24 Írjunk olyan programot, amely a gép hanggenerátorának minden lehetséges hangját annyi ideig hallatja, amíg lefut egy VÁR ciklus (ld. 7.7 feladat)!

BEFEJEZES

Senki ne higgye, még ha idáig el is jutott, hogy már mindent tud a gépi kódu programozás rejtelmeiről, hiszen még éppen csak a kezdeteknél tart! Még az sem biztos, hogy az összes utasítást és címzésmódot ismeri, mivel nem tudtunk mindenre példát mutatni (bár a táblázat véleményünk szerint elegendő információt tartalmaz a továbbhaladáshoz). Azt azonban elmondhatja, hogy a gépi kódó programozás elsajátításának legnehezebbjén már túl van, s már csupán szorgalom kérdése, hogy mennyire lesz 'profi' ezen a területen. A további munka nagyfoku önállóságot feltételez, s ehhez ötleteket meríthetünk az irodalomjegyzékben felsorolt művekből. Ezen kivül tervezünk egy ujabb kötetet is, amelyben néhány, az itt ismertetetteknél nagyobb lélegzetű program teljes megoldásával segitenénk az egyéni és a szakköri munkát egyaránt.

Befejezésképpen álljon itt egy összefoglaló táblázat a címzésmodokról:

	Név	Jel	Példa
1.	Közvetlen	#érték	LDA #34
2.	Abszolut	abs	JSR $ffff4
3.	Nullalapos	zp	LDA $c6
4.	Akkumulátor		LSR
5.	Beleértett		SEC
6.	Indexelt indirekt	(zp,X)	CMP ($2b,X)
7.	Indirekt indexelt	(zp),Y	AND (83),Y
8.	Nullalapos, X ind.	zp,X	LDY $18,X
9.	Nullalapos, Y ind.	zp,Y	LSR $2b,Y
10.	Abszolut, X ind.	abs,X	EOR $1234,X
11.	Abszolut, Y ind.	abs,y	SBC $4344,Y
12.	Indirekt	(abs)	JMP $8000
13.	Relativ	táv	BNE $3354

UTASITASTABLAZAT

NEV	CIMZES	JELZOBITEK NV BDIZC	LEIRAS
adc	(zp,x)	** **	a:=a+m+c
adc	zp	** **	a:=a+m+c
adc	#érték	** **	a:=a+érték+c
adc	abs	** **	a:=a+m+c
adc	(zp),y	** **	a:=a+m+c
adc	zp,x	** **	a:=a+m+c
adc	abs,y	** **	a:=a+m+c
adc	abs,x	** **	a:=a+m+c
and	(zp,x)	* *	a:=a and m
and	zp	* *	a:=a and m
and	#érték	* *	a:=a and érték
and	abs	* *	a:=a and m
and	(zp),y	* *	a:=a and m
and	zp,x	* *	a:=a and m
and	abs,y	* *	a:=a and m
and	abs,x	* *	a:=a and m
asl	zp	* **	balra léptetés 1 bittel m-ben
asl		* **	balra léptetés 1 bittel a-ban
asl	abs	* **	balra léptetés 1 bittel m-ben
asl	zp,x	* **	balra léptetés 1 bittel m-ben
asl	abs,x	* **	balra léptetés 1 bittel m-ben
bcc	táv		c=0 esetén relatív ugrás
bcs	táv		c=1 esetén relatív ugrás
beq	táv		z=1 esetén relatív ugrás
bit	zp	** *	n:=m7, v:=m6, a and m szerint z
bit	abs	** *	n:=m7, v:=m6, a and m szerint z
bmi	táv		n=1 esetén relatív ugrás
bne	táv		z=0 esetén relatív ugrás
bpl	táv		n=0 esetén relatív ugrás
brk		1 *	belső megszakítás
bvc	táv		v=0 esetén relatív ugrás
bvs	táv		v=1 esetén relatív ugrás
clc		0	c:=0
cld		0	d:=0
cli		0	megszakítás engedélyezése
clv		0	v:=0
cmp	(zp,x)	* **	jelzőbitek állítása a-m szerint
cmp	zp	* **	jelzőbitek állítása a-m szerint
cmp	#érték	* **	jelzőbitek áll. a-érték szerint
cmp	abs	* **	jelzőbitek állítása a-m szerint
cmp	(zp),y	* **	jelzőbitek állítása a-m szerint
cmp	zp,x	* **	jelzőbitek állítása a-m szerint
cmp	abs,y	* **	jelzőbitek állítása a-m szerint
cmp	abs,x	* **	jelzőbitek állítása a-m szerint
cpx	#érték	* **	jelzőbitek áll. x-érték szerint
cpx	zp	* **	jelzőbitek állítása x-m szerint
cpx	abs	* **	jelzőbitek állítása x-m szerint
cpy	#érték	* **	jelzőbitek áll. y-érték szerint
cpy	zp	* **	jelzőbitek állítása y-m szerint

NEV	CIMZES	JELZOBITEK NV	BDIZC	LEIRAS
cpy	abs	*	**	jelzőbitek állítása y-m szerint
dec	zp	*	*	m:=m-1
dec	abs	*	*	m:=m-1
dec	zp,x	*	*	m:=m-1
dec	abs,x	*	*	m:=m-1
dex		*	*	x:=x-1
dey		*	*	y:=y-1
eor	(zp,x)	*	*	a:=a eor m
eor	zp	*	*	a:=a eor m
eor	#érték	*	*	a:=a eor érték
eor	abs	*	*	a:=a eor m
eor	(zp),y	*	*	a:=a eor m
eor	zp,x	*	*	a:=a eor m
eor	abs,y	*	*	a:=a eor m
eor	abs,x	*	*	a:=a eor m
inc	zp	*	*	m:=m+1
inc	abs	*	*	m:=m+1
inc	zp,x	*	*	m:=m+1
inc	abs,x	*	*	m:=m+1
inx		*	*	x:=x+1
iny		*	*	y:=y+1
jmp	abs			ugrás abs-ra
jmp	(abs)			ugrás az abs-on lévő címre
jsr	abs			abs-on kezdődő szubrutin hívása
lda	(zp,x)	*	*	a:=m
lda	zp	*	*	a:=m
lda	#érték	*	*	a:=érték
lda	abs	*	*	a:=m
lda	(zp),y	*	*	a:=m
lda	zp,x	*	*	a:=m
lda	abs,y	*	*	a:=m
lda	abs,x	*	*	a:=m
ldx	#érték	*	*	x:=érték
ldx	zp	*	*	x:=m
ldx	abs	*	*	x:=m
ldx	zp,y	*	*	x:=m
ldx	abs,y	*	*	x:=m
ldy	#érték	*	*	y:=érték
ldy	zp	*	*	y:=m
ldy	abs	*	*	y:=m
ldy	zp,x	*	*	y:=m
ldy	abs,x	*	*	y:=m
lsr	zp	*	**	jobbra léptetés 1 bittel m-ben
lsr		*	**	jobbra léptetés 1 bittel a-ban
lsr	abs	*	**	jobbra léptetés 1 bittel m-ben
lsr	zp,x	*	**	jobbra léptetés 1 bittel m-ben
lsr	abs,x	*	**	jobbra léptetés 1 bittel m-ben
nop				üres utasítás
ora	(zp,x)	*	*	a:=a or m
ora	zp	*	*	a:=a or m
ora	#érték	*	*	a:=a or érték

NEV	CIMZES	JELZOBITEK NV	BDIZC	LETRAS
ora	abs	*	*	a:=a or m
ora	(zp),y	*	*	a:=a or m
ora	zp,x	*	*	a:=a or m
ora	abs,y	*	*	a:=a or m
ora	abs,x	*	*	a:=a or m
pha				verembe: a
php				verembe: feltételregiszter
pla		*	*	a: veremből
plp		**	*****	feltételregiszter: veremből
rol	zp	*	**	balra forgatás 1 bittel m-ben
rol		*	**	balra forgatás 1 bittel a-ban
rol	abs	*	**	balra forgatás 1 bittel m-ben
rol	zp,x	*	**	balra forgatás 1 bittel m-ben
rol	abs,x	*	**	balra forgatás 1 bittel m-ben
ror	zp	*	**	jobbra forgatás 1 bittel m-ben
ror		*	**	jobbra forgatás 1 bittel a-ban
ror	abs	*	**	jobbra forgatás 1 bittel m-ben
ror	zp,x	*	**	jobbra forgatás 1 bittel m-ben
ror	abs,x	*	**	jobbra forgatás 1 bittel m-ben
rti		**	*****	visszatérés megszakításból
rts				visszatérés szubrutinból
sbc	(zp,x)	**	**	a:=a-m-(1-c)
sbc	zp	**	**	a:=a-m-(1-c)
sbc	#érték	**	**	a:=a-érték-(1-c)
sbc	abs	**	**	a:=a-m-(1-c)
sbc	(zp),y	**	**	a:=a-m-(1-c)
sbc	zp,x	**	**	a:=a-m-(1-c)
sbc	abs,y	**	**	a:=a-m-(1-c)
sbc	abs,x	**	**	a:=a-m-(1-c)
sec			1	c:=1
sed			1	d:=1
sei			1	megszakítás tiltása
sta	(zp,x)			m:=a
sta	zp			m:=a
sta	abs			m:=a
sta	(zp),y			m:=a
sta	zp,x			m:=a
sta	abs,y			m:=a
sta	abs,x			m:=a
stx	zp			m:=x
stx	abs			m:=x
stx	zp,y			m:=x
sty	zp			m:=y
sty	abs			m:=y
sty	zp,x			m:=y
tax		*	*	x:=a
tay		*	*	y:=a
tsx		*	*	x:=sp
txa		*	*	a:=x
txs				sp:=x
tya		*	*	a:=y

IRODALOM

[1] Babán Gábor - Masa István:
Gépi kódú programozás kezdőknek és haladóknak
C16 és PLUS/4 számítógépre
Novotrade Rt. 1987.

[2] Vadnai Szabolcs:
Programozói kézikönyv
Commodore+4
Novotrade Rt. 1987.

[3] C16-Plus/4 programozói útmutató
Novotrade Rt. 1987.

[4] Tóth Viktor:
A Commodore 16-os belső felépítése
Novotrade Rt. 1986.

[5] Gáspár Dénes - Gyenes Tamás:
a Plus/4-es belső felépítése
Novotrade Rt. 1988.

[6] Szlávi Péter - Zsakó László:
Módszeres programozás
Műszaki Könyvkiadó, 1986.

[7] Erdős Iván:
Commodore PLUS/4, C-16, C-116 ROM-lista
LSI ATSZ. 1986.

[8] Ury László:
BASIC és felhasználói kézikönyv
Commodore C-16, C-116
LSI ATSZ. 1986.

[9] Lothar Englisch:
Gépi kódú programozás a Commodore 64-esen
Data Becker - Novotrade, 1986.

[10] Szabó Szilárd - Tóthné Máriássy Éva:
Játékok és felhasználói programok
C-16 - PLUS/4
LSI ATSZ, 1986.

[11] Számítástechnika középfokon
OMIKK, 1987.

[12] Számítástechnikai feladatok 2000-ig
OMIKK, 1988.

JCL-SEGEDLET

Ebben a függelékben röviden összefoglaljuk a BASIC- és gépi kódú programok készítésekor igen hasznos JCL fejlesztőrendszer témánkkal összefüggő tulajdonságait. Részletesebb (bár hibáktól nem mentes) leírás található pld. [10]-ben.

Az eredeti programcsomag C-64-es és C-128-as gépekre is használható, C+4-en a +4SYS nevű programfile-ja fut, amely a $1001-$5000 közötti helyet foglalja el a tárból. BOOT-tal való töltés után automatikusan, ha csak a +4SYS-t töltjük be, akkor RUN-nal indul. RESET utáni újraindítása SYS 4111-gyel lehetséges.

Témánkhoz kapcsolódva: a program leginkább a lemezkezeléshez, az assembly nyelvű forrásszövegek szerkesztéséhez, fordításához ill. beléveséhez nyújt segítséget. A rendszerben az összes BASIC-kulcsszó továbbra is (eredeti jelentésének megfelelően) használható, de újak is rendelkezésünkre állnak; ezeket a WORDS parancsra a rendszer a képernyőre írja. Rövidítéseknél az új kulcsszavak elsőbbséget élveznek (pld. SCROLL - SCRATCH).

A lemezkezelést segítő lehetőségeknél a '@' ill. a '>' jel egyenrangúan használható. Magában bármelyikük a hibacsatorna lekérdezésével a lemezmeghajtó állapotát (BASIC-ben DS$-t) írja a képernyőre. Ha egy karakterrel együtt alkalmazzuk, akkor ezt a karaktert küldi ki a parancscsatornán. Így pld. a @N, @R, @C, @S, @I, @V parancsok rendre formattáláshoz, átnevezéshez, másoláshoz, törléshez, inicializáláshoz ill. validáláshoz használhatók. Nagy haszna van a @$ parancsnak: az F3 billentyűt helyettesíti (DIRECTORY), a funkcióbillentyűket a program ugyanis másra használja. Programfile-ok betöltése a

'/filenév' paranccsal (a mentési címtől függetlenül) a BASIC terület elejére, a

'%filenév' paranccsal a mentési címre, a

'%filenév,új cím' paranccsal pedig a (decimális formában megadott) új címre történik (ugyancsak a mentési címtől függetlenül).

A lemezen lévő programfile-ok terjedelme a SIZE''filenév'' paranccsal olvasható ki.

Az assembly nyelvű program (a továbbiakban forrásszövegként említjük) szerkesztése az EDIT parancs beírása után lehetséges. A memóriában lévő forrásszöveget szekvenciális file-ként lemezre a PUT''filenév'', kazettára pedig a PUTC''filenév'' parancs menti. A forrásszöveg betöltése a GET''filenév'' ill. GETC''filenév'' segítségével történik. A TYPE''filenév'' a kérdéses file tartalmát a képernyőre listázza, a memóriában lévő file megsértése nélkül.

A forrásszöveg egy sora (például) a következőképpen néz ki:

```
1010 indul  lda #$01          ;megjegyzés
```

Egy sor részei:

SORSZÁM 1010 az egyes utasítások sorrendjét határozza meg; csak a szerkesztést segíti (nem lehet pld. a BASIC-beli GOTO-nál szokásos módon hivatkozni rá).

CÍMKE indul egy legfeljebb 6 karakteres, betűvel kezdődő (de esetleg számokat is tartalmazó), az utasításoktól különböző szimbolikus név. Ennek segítségével hivatkozhatunk a tár azon rekeszének a címére, amelyre fordításkor a megcímkézett utasítás kerül. Használata nem kötelező.

UTASÍTÁS lda egy utasítás a processzor utasításkészletéből, ill. egy ún. pszeudoutasítás (direktíva). Ez utóbbiakról később lesz szó.

OPERANDUS #$01 az adott utasításhoz szükséges cím vagy érték meghatározója (akár szimbolikus név is lehet). A decimális számokat külön jelzés nélkül írjuk, a binárisok elé %, a hexadecimálisok elé $-jelet írunk; a karaktereket aposztrofok közé tesszük. Bizonyos utasítások (pld. CLC, ROR, TXA stb.) esetén értelemszerűen szükségtelen, ezért elmarad.

MEGJEGYZÉS a sornak a ; után következő része; szintén nem kötelező.

Az utasításon egyes mezőit szóközökkel kell elválasztani egymástól. A forrásszöveg szerkesztéséhez hasznos parancsok:

HD és DH decimális-hexadecimális átalakítás

FIND egy adott szövegrész keresése. A
FIND/szöveg/
hatására a program a képernyőre írja a file azon sorait, amelyben a szöveg előfordul.

CHANGE egy adott szövegrész másikra cserélése. A
CHANGE/mit/mire/
hatására a file-ban a 'mit' szöveget mindenütt a 'mire' szövegre cseréli.

NUM a forrásszöveg sorainak átszámozását végzi.

EXPAND hatására a forrásszöveg listázáskor tabulátorán, jól átlekinthető formában kerül a képre.

SCROLL — a forrásszöveg listázását segítő parancs. Lehetővé teszi a teljes szövegben történő mozgást a kurzormozgató gombok segítségével. A funkcióbillentyűk hatása:

- F1 — a képernyőn lévő oldalt újra listázza,
- F2 — a kurzort tartalmazó sortól kezdődő oldalt listázza,
- F3 — a képen látható részt követő oldalt listázza

DO — végrehajtja az első sorban ;-vel kezdődő parancssort.

CBM-ASCII — segítségükkel választhatjuk ki a megfelelő nyomtatót.

A pszeudoutasítások vagy direktívák olyan utasítások, amelyek nem a processzornak, hanem a fordítóprogramnak szólnak, a fordítást vezérlik. A legfontosabbak a következők:

= értékadás; az előtte szereplő szimbólum értéke az őt követő szám lesz.

* a programszámláló értékét határozhatjuk meg vele.

.end a forrásprogram végét jelzi.

.byte az őt követő egy vagy több szám (szimbólum) értékét sorrendben a tárba írja.

.word az őt követő kétbyte-os számot címként értelmezve (azaz fordított sorrendben) a tárba írja.

A forrásprogram szerkesztését követően az ASM parancs segítségével kezdhetünk a fordításhoz. Az egyes opciók:

s fordítás a képernyőre
p fordítás a nyomtatóra
m a lefordított program a lemezre kerül .mod kiegészítéssel
x fordítás a memóriába
c fordítás a memóriából.

Szintaktikus ellenőrzést a c opcióval végezhetünk. A lefordított program belövését, tesztelését, javítását segíti a SETBRK parancs. Segítségével a forrásszöveg átírása ill. újabb fordítása nélkül - töréspont iktatható be a gépi kódú programba, majd a CLRBRK segítségével a program ugyanilyen egyszerűen visszaállítható eredeti állapotába.

A PEDAGÓGIAI MŰHELY SOROZATBAN MEGJELENT KIADVÁNYOK JEGYZÉKE:

1. Mikus Katalin:
   Ki? Miről? Kinek?
   Téma- és előadóajánlás továbbképzések szervezéséhez 1987

2. Szöllősi Zsuzsanna:
   A filozófiaoktatás helyzete Pest megyében az 1986/87. tanévben végzett felmérés tanulságai alapján 1988

3. Az osztályfőnöki nevelőmunka tapasztalatainak összegzése
   Összeállította: Dr.Deák Béláné, Oláh Mihályné 1988

4. Az általános iskolai fakultáció szerepe a differenciált képességfejlesztésben /Az 1987-ben rendezett százhalombattai országos tanácskozás rövidített közlése/
   Szerkesztette: Bottlik István, dr.Szabó Imre 1988

5. Fürstné dr. Kólyi Erzsébet:
   Munkaközösségek az iskolában 1988

6. Bázisintézmények Pest megyében 1988
   Összeállította: Mikus Katalin

7. Muskovits Lászlóné:
   Középtávú pedagógiai program 1989

8. Humanisztikus iskola
   Szerkesztő: Benda József-Matula Józsefné 1989

9. Farkas Károly-Kőrösné Mikis Márta:
   Játszd el a teknőcöt!... 1989

10. Németh Ferenc:
    Számítástechnika fakultáció 1989