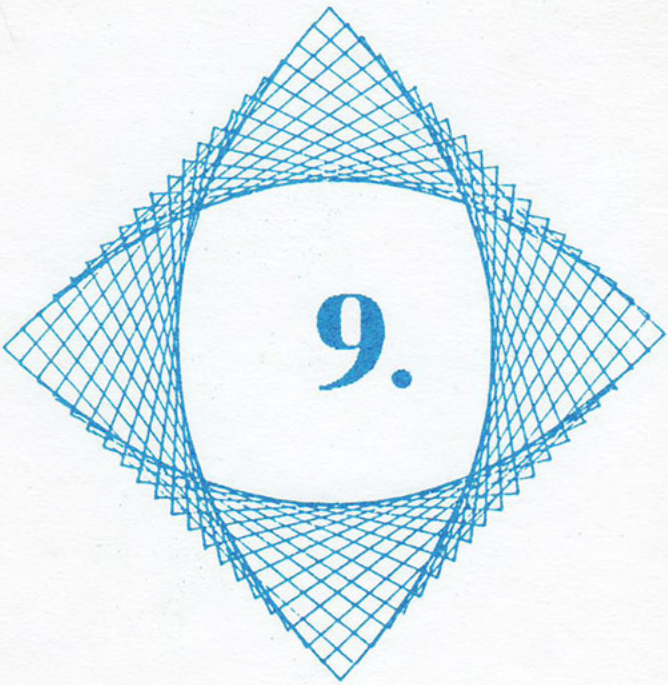
SZÁMÍTÁSTECHNIKAI OKTATÁSI FÜZETEK

Gyetvai Károly

ASSEMBLY ÉS GÉPI SZINTŰ PROGRAMOZÁS COMMODORE 16, PLUS/4 SZÁMÍTÓGÉPEN



1991

Gyetvai Károly

ASSEMBLY ÉS GÉPI SZINTŰ PROGRAMOZÁS COMMODORE 16, PLUS/4 SZÁMÍTÓGÉPEN

Füzet sorszáma: 9.

1991.

Szerző:

Gyetvai Károly
Mechwart András Gépipari Műszaki Középiskola
Debrecen

Lektorálta:

Huszár Gábor
Neumann János Közgazdasági Szakközépiskola
Budapest

Ez a füzet az ÉGSZI - SCOLA Alapítvány szervezésében készült.

TARTALOMJEGYZÉK

I. BEVEZETÉS

A Commodore 16 és Plus/4 számítógépek nagyon fejlett BASIC értelmezővel rendelkeznek. Mégis előfordulhatott már mindenkivel, hogy lassúnak érezte a BASIC program végrehajtását, vagy a megoldandó feladatához nem talált megfelelő BASIC eszközt. Ilyenkor kell választanunk a gépi kódú vagy az assembly szintű programozás eszközeit. Füzetünkkel azoknak kívánunk segítséget adni, akik arra adták a fejüket, hogy megismerkednek a gépi kódú és assembly szintű programozással.

Feltételezzük, hogy aki ebbe belevág, az már ismeri az adott géptípus BASIC nyelvű programozását. Megismert olyan számítástechnikai alapfogalmakat, mint: algoritmus, kód, program, alprogram, függvény, bit, bájt, bináris-, és hexadecimális számrendszer, fixpontos-, és lebegőpontos számábrázolás, logikai műveletek, digitális és analóg számítógépek elvi felépítése stb. Az assembly tanulmányok megkezdése előtt célszerű ezeket az ismereteket felidézni.

A digitális számítógép központi vezérlő egységét és az aritmetikai, logikai műveleti egységet magába integráló processzor logikai felépítésével, a regiszterek és jelzőbitek ismertetésével kezdjük anyagunkat. Megbeszéljük a memória és a perifériák főbb kérdéseit is. Ezután sorra vesszük a processzorunk utasításait, a címzési módokat. Az assembly programozás két segédeszközét; a MONITOR programot és az assemblert megtanuljuk használni. Ezután már írhatunk egyszerű assembly programokat. Alapfeladatokat tartalmazó fejezetünkben több egyszerű algoritmust ismertetünk. A számítógép ROM memóriájában lévő BASIC rutinok és KERNAL rutinok használatát is bemutatja füzetünk. A gépi kódú programjaink és a BASIC nyelvű programjaink kapcsolatát is tárgyaljuk. A feladatgyűjteményben sok-sok feladattal, programmal ellenőrizhetjük megszerzett tudásunk mélységét.

A füzet közreadásakor feltételezzük, hogy az assembly programozás alapjaival szervezett oktatás formájában (tanítási óra, szakkör, fakultáció, tanfolyam stb.) ismerkedik az olvasó. Tehát a tanári magyarázatok nélkülönözhetetlenek a füzet feldolgozásához. Az önálló tanulás és programírás segítésére a függelékben közölt táblázatokat szánjuk. Az érdeklődőbb olvasónak egy, a témát teljességében átfogó szakirodalom ajánlással kedveskedünk.

Az anyag tárgyalása előtt szeretnénk rögzíteni, hogy a füzetben leírtak a Commodore 16 és a Commodore Plus/4 típusú számítógépre vonatkoznak csak. Aki más típusú Commodore gépre szeretné adaptálni a leírtakat, annak több helyen (címek, értékek) módosítania kell az anyagot. Az utasításkészlet, a címzési módok, a KERNAL rutinok és még sok egyéb más változtatás nélkül érvényes pl. a Commodore 64 típusú gépre is.

Végül eredményes tanulást, jó programozást kívánunk!

II.HARDVER ALAPISMERETEK

1. REGISZTEREK

AC akkumulátor regiszter: 8 bites kitüntetett munkaregiszter. Az aritmetikai és a logikai müveletek az egyik operendusukat ezen tárolóból veszik és a müvelet eredményét is ide irják vissza. A legtöbb utasitás használja az AC-t.

XR X indexregiszter : 8 bites munkaregiszter, általános felhasználási területe az indexelés.

YR Y indexregiszter : 8 bites munkaregiszter, szerepe ugyanaz mint az XR-é.

SR státuszregiszter : 8 bites, a számitógép mindenkori állapotát jelző un. állapotregiszter. A gépi utasitások végrehajtása után az eredmény paramétereitől függően állitódnak a regiszter jelzöbitjei, a FLAG-ek. Az egyes bitek jelölése: N, V, 1, B, D, I, Z, C. Jelentésük leirását alább közöljük.

SP Stack Pointer : 8 bites, a veremtár mutatója. A legutóljára beirt adat címének alsó félbájtját tartalmazza. A cím felsö félbájtja mindig $01. A verem induló címe $01FF, tehát az SP induló értéke $FF.

PC Program Counter : 16 bites, un. programszámláló regiszter. Tartalma mindig az aktuálisan végrehajtandó utasitás utasitáskódjának tárbeli címe (2 bájt!).

2. AZ ALLAPOTREGISZTER FLAG-JEI

N NEGATIV
1 ha egy müvelet eredménye nagyobb mint 127,
0 ha ugyanez kisebb vagy egyenlö 127-tel.
(az eredmény 7. bitjét jelenti)

V OVERFLOW
1 ha az összeadási vagy kivonási müvelet közben túlcsordulás történt,
0 ha nem volt túlcsordulás.

B BREAK
1 ha a megszakitást a BREAK utasitás okozta,
0 különben.

D DECIMALIS
1 ha az aritmetikai müveletek operandusait BCD számokként akarjuk kezelni,
0 ha ugyanezeket hexadecimális számoknak tekintjük.

I INTERRUPT
1 megszakitás letiltva,
0 megszakitás engedélyezve.

Z ZERO
1 ha a müvelet eredménye zérus volt,
0 ha az eredmény nem volt zérus.

C CARRY
1 ha az aritmetikai müvelet eredménye 255-nél nagyobb,
0 különben.

3.MEMÓRIAKEZELÉS

A processzor mellett a számítógép másik nagyon fontos egysége a memória. Az adatok, programok tárolására szolgáló memória szinte mindegyik assembly utasításban szerepel valamilyen módon, ezért az utasítások részletezése elött tekintsük át a memóriakezelés alapjait!

BÁJT, CÍM, NULLASLAP

A memória legkisebb címezhetö egysége a bájt. Mint tudjuk egy bájt 8 bitböl áll, igy az egy bájtba irható érték 0 és 255 közé esik (hexadecimális számrendszerben $00 és $FF). Az egyes bájtokat azonositó sorszámokat címeknek nevezzük. A 64 K-s memóriájú (64 * 1024 = 65536 bájt) tárak esetén a címek 0 és 65535 közé esnek (hexadecimális számrendszerben $0000 és $FFFF). A továbbiakban egy cím általános jelölésére a $cccc alakot használjuk. Amennyiben a cím elsö két számjegye (haxadecimális alak!) nulla, akkor a cím általános alakja $cc lesz. Az ilyen címmegadást a processzor értelmezi, a programok és a végrehajtási idö rövidebb lesz. Összesen 256 db olyan bájt van a memóriában, amelynek a címét igy rövidithetjük. Azt mondjuk, hogy ezek a bájtok a nullaslapon helyezkednek el.

SZÓ, CÍM TAROLASA

A memóriának a bájtnál nagyobb egysége a szó. Egy szó két darab egymást követö bájtot jelent, igy kiválóan alkalmas egy cím tárolására. A szó elsö bájtjának címét, azaz a szó címét a továbbiakban ($cccc) vagy ($cc) fogja jelezni, a zárójellel megkülönböztetve a bájt címétöl. A szó két bájtja un. bájtfordított sorrendben tartalmaz memória címet. Tehát a cím alsó felét (LB) a szó elsö, a cím felsö felét (HB) a szó második bájtja tartalmazza. Egy példával megvilágitva: legyen a $D0 cimü bájt tartalma $33, a $D1 cimüé pedig $40. Ekkor a ($D0) cimü szóban tárolt cim $4033 lesz.

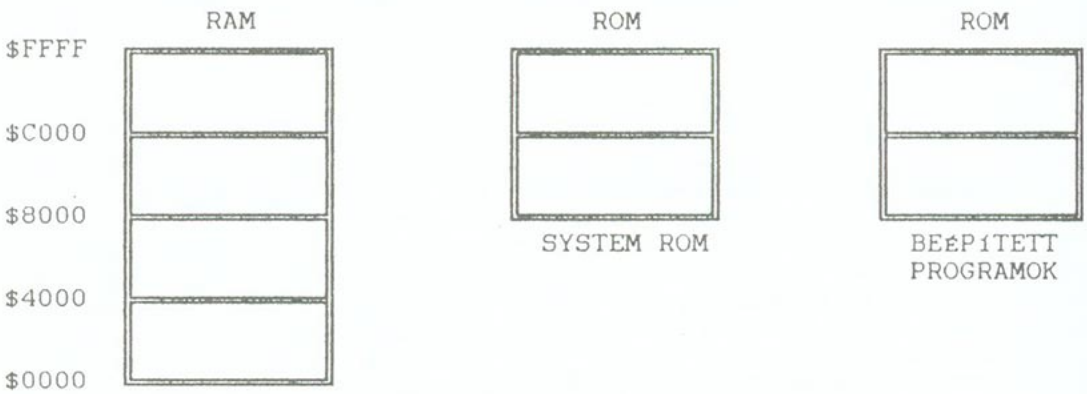
TARFELOSZTAS

A Commodore 16 és a Plus/4 memóriája csak méretben tér el egymástól. Az elöbbi tipus 16 K RAM (irható és olvasható memória) az utóbbi 64 K RAM területtel rendelkezik. A ROM (csak olvasható tár) mindkét tipusnál 32 K méretü (a Plus/4 beépitett 4 programját nem idevéve).
Egy vázlatos tártérkép tehát az alábbiak szerint adható meg:

RAM	$0000 - $07FF	rendszerváltozók
	$0800 - $0BFF	színmemória
	$0C00 - $0FFF	képernyömemória
	$1000 - $3FFF	BASIC programok szabad területe, C 16-os
	$1000 - $FFFF	BASIC programok szabad területe, C Plus/4-s
ROM	$8000 - $CFFF	BASIC ROM
	$D000 - $D7FF	karaktergenerátor
	$D800 - $FFFF	KERNAL rutinok

MEMÓRIALAPOZAS

Grafikusan ábrázolva a memória egységeket, méginkább szembetünö, hogy több tárterületen is van azonos címü bájt az alapkiépitésü C Plus/4 tipusú számítógépben:



Felvetödik a kérdés, hogy pl. a $D200 cím melyik területre vonatkozik? Alapállapotban az iró utasítások mindig a RAM területre vonatkoznak, míg az olvasásnál a $8000 felett a ROM-ot olvashatjuk, alatta a RAM-ot (ld. MONITOR program). Ezt az alapértelmezést változtathatjuk meg a RAM-ROM lapozással. A RAM-ot úgy tudjuk felülre lapozni, ha az $FF3F címü bájtba valamilyen értéket beírunk (mindegy, hogy mit). A ROM felülre lapozásához az $FF3E címü tárhely tartalmát kell megváltoztatnunk (szintén mindegy, hogy mit írunk ide). A RAM-ROM lapozás természetesen nem érinti a a tár alsó felét, hiszen ott mindig RAM terület található. Ugyancsak nem változtatható meg az $FD00-$FF3F terület funkciója sem, amelyik az I/O eszközökhöz rendelt speciális tárterület.

Itt említjük meg, hogy az assembly programozás egyik segédeszköze, a késöbbiekben ismertetésre kerülö TEDMON MONITOR program ennél egyszerübben valósítja meg a RAM-ROM lapozást. Ha a tár $07F8 címü bájtjában a 7. bit 1, akkor a MONITOR a "magas címek" esetén a RAM-ot, ellenkezö esetben a ROM-ot olvassa.

VEREMMEMÓRIA

A memória speciális területével, a veremmemóriával már a BASIC nyelvü programozás során a GOSUB, RETURN utasítások tárgyalásánál megismerkedtünk. Emlékeztetöül elmondjuk, hogy a veremtár olyan speciális terület, amelynél mindig a legutóljára beírt adat után írhatunk csak, olvasni is a legutóljára elhelyezett értéket lehet. A verem kezelését a veremmutató (SP regiszter) könnyiti meg. Értéke mindig a soronkövetkezö üres veremhely címének alsó fele. A veremtár $0100-tól $01FF-ig terjedö tárterület, igy a cím felsö fele mindig $01, amit a rendszer automatikusan az SP regiszter tartalmához illeszt.
A veremtár föbb alkalmazási területei az assembly programozás során is az alprogramhivás, a vezérlésátadás és a megszakitások kezelése.

4. INPUT/OUTPUT ESZKÖZÖK

A digitális számítógép elvi sémájában említett I/O vezérlöröl még nem szóltunk. A külvilággal való kapcsolatteremtés sokrétü voltáról egyszerüen meggyözödhetünk, ha egy pillantást vetünk a számítógépünk hát-, és oldallapjára. Itt az alábbi csatlakozókat találjuk:

monitor csatlakozó	(2 db: VIDEO és RF)
kazettás egység csatlakozója	(CASETTE)
soros busz	(SERIAL)
felhasználói kapu	(USER PORT)
memóriabövítés helye	(MEMORY EXPANSION)
botkormány csatlakozók	(2 db: JOY0 és JOY1)

Nem látszik kívülröl, de a billentyüzet is egy csatlakozón keresztül kommunikál a számítógéppel. Vegyük sorra az egyes I/O eszközökre vonatkozó legalapvetöbb ismereteket!

BILLENTYÜZET

Az elsödleges input eszközünk 66 db billentyüt kezel, közülük a RESTORE nem 'programozható', a SHIFT LOCK pedig a baloldali SHIFT-re van kötve. A 64 db billentyüt egy 8*8-as drótháló fölött képzeljük el. A billentyü él, ha éppen az alatta lévö helyen érintkezik a sor és oszlop drót.

A billentyüzet mátrix az alábbi:

	0	1	2	3	4	5	6	7
0	DEL	3	5	7	9	->	fel	1
1	RET	W	R	Y	I	P	*	HOM
2	\	A	D	G	J	L	];	CTRL
3	HELP	4	6	8	0	<-	le	2
4	F1	Z	C	B	M	>.	ESC	SPC
5	F2	S	F	H	K	[:	=	C=
6	F3	E	T	U	O	-	+	Q
7	@	SHIFT	X	V	N	<,	?/	STOP

A billentyüzet programozására kényelmes KERNAL rutinok állnak a rendelkezésünkre: egyetlen billentyü lenyomását figyelö SCNKEY rutin ($FF9F), a billentyüpuffert olvasó GETIN rutin ($FFE4), több billentyü egyidejü lenyomását a $DB70 címen kezdödö rutin vizsgálja. A KERNAL rutinok leírására a továbbiakban kerül sor.

MONITOR

A monitor, vagy kevésbé szerencsés esetben a TV készülék karakteres és grafikus üzemmódban használható. Mint tudjuk karakteres üzemmódban a képernyöre 25 sorban, soronként 40 karaktert írhatunk. A használható karakterek mindegyike egy 8*8-as méretü pontmátrixban helyezkedik el. Nagyítóval szemlélve a képernyöt jól látszik a karakter felépítése.

A használható karakterek alakját is a ROM egy összefüggő területe, az un. karaktergenerátor határozza meg. Egy-egy karakternek 8 bájt örzi a leírását az alábbiak szerint.

Legyen a vizsgált karakter az 'A' és a '2' !

cím	érték	bitminta	cím	érték	bitminta
$D008	$18	...11...	$D190	$3C	..1111..
$D009	$3C	..1111..	$D191	$66	.11..11.
$D00A	$66	.11..11.	$D192	$06	.....11.
$D00B	$7E	.111111.	$D193	$0C	....11..
$D00C	$66	.11..11.	$D194	$30	..11....
$D00D	$66	.11..11.	$D195	$60	.11.....
$D00E	$66	.11..11.	$D196	$7E	.111111.
$D00F	$00	....	$D197	$00	....

Figyeljük meg, hogy az utolsó bájt értéke mindig 0, a sorok közötti üres sáv miatt. Hasonlóan minden bájt 0. bitje is 0, a karakterek közötti távolság miatt. A beépített karaktergenerátor mérete 2K (2048:8=256 db különböző jel kódolására elegendő hely), kezdöcíme pedig $D000. A karakterek a képernyö kódjaik szerinti rendben követik egymást.

Ha saját tervezésü karaktereket akarunk használni, akkor a RAM-ba kell másolnunk a karaktergenerátort. Itt lehet módosítani a megfelelö pontmátrixot. Természetesen közölni kell a rendszerrel, hogy a saját karaktergenerátorát vagy az általunk készíttetet akarjuk használni.

A képernyö minden egyes karakterhelyéhez a képernyömemória és a színmemória egy-egy bájtja tartozik (ld. Függelék).

KAZETTAS EGYSÉG

Az adatok és programok tárolására szolgáló egység és a számítógép közötti adatforgalom éppen úgy pufferezett, mint a billentyüzet kezelése. A pufferezési technika lényege, hogy a számítógép és a periféria közötti adatforgalomban egy közbülsö tárterület, a puffer vesz részt. A számítógép is és a perifériális eszköz is a pufferrel van kapcsolatban, ide irják és innen olvassák az adatokat. A puffer szerepe a processzor és a periféria müködési sebessége közötti különbség kiegyenlítése.
A kazettás egység puffer területe: $0333-$03F2.
A kazettás berendezés egységszáma 1.
Itt kell megemlíteni, hogy a számítógép meghibásodásának leggyakoribb oka az, ha müködés közben csatlakoztatjuk a géphez a kazettás egységet, vagy kihúzzuk a magnetofon csatlakozóját a gépböl.

SOROS BUSZ

A mágneslemezes egység (egységszáma 8, 9, 10, 11 lehet), és a nyomtató (egységszáma 4 vagy 5 lehet) az un. soros buszon keresztül csatlakozik a számítógéphez. A soros buszra helyezett egységek (egyidöben természetesen több is) háromféle funkciót tölthetnek be:

Vezérlö: kiosztja a 'hallgató' vagy a 'beszélö' szerepet. Ilyen funkciója általában csak a számítógépnek lehet.

Beszélö: olyan funkció, amikor az eszköz képes adatokat adni a soros buszra. Értelemszerüen egyidöben csak egy eszköz lehet ilyen állapotban.

Hallgató: olyan funkció, amikor az eszköz képes fogadni az adatokat a soros buszról.

A soros adatátvitel lényege, hogy az átviendö adatbájt bitjei egymás után haladnak az egyetlen vonalon. Természetesen az adatátvitel ideje igy jelentösen nagyobb, mintha párhuzamosan 8 vonalon futna egy bájt tartalma. Mivel a soros buszra több egység is rá van kötve, ezért mindegyiket egy-egy azonosító eszközszámmal láttak el.

BOTKORMANYOK

A Commodore Plus/4 számítógépet két speciális, 8 pólusú csatlakozóval láttak el, amelyekhez botkormányok csatlakoztathatók. A botkormányok állapotának kiolvasására a JOY BASIC függvényen kívül gépi kódú rutinok is használhatók. A botkormányokkal kapcsolatos memóriahelyeket és jelentéstartalmukat közöljük:

	0. botkormány	1. botkormány
kiválasztás:	$FF08 2. bit	$FF08 1. bit
tüzgomb:	$FF08 4. bit	$FF08 4. bit
iránykapcsoló:	$FF08 0.-3. bit	$FF08 0.-3. bit

FELHASZNALÓI KAPU

A USER PORT jelölésü csatlakozóra kivezetett jelek elsösorban a számítógép funkcióinak a felhasználó általi bövítésére szolgálnak (pl. vezérlési, szabályozási, érzékelési, mérési feladatok megoldása stb.). Itt kaptak helyet egy 8 vonalas párhuzamos kapuáramkör kivezetései és a szabványos RS 232 illesztö is. Müködésük megértése mélyebb ismereteket igényel. Számunkra most elegendö tudni, hogy a ROM-ban sok rutin szolgál a felhasználói kapu programozására.

MEMÓRIA BÖVÍTÉSE

A memória bövítése két okból szokásos: vagy a C 16 alapgépet bövítjük ki RAM memóriával, vagy pedig valamilyen kész programot tartalmazó ROM memóriabövitöt használunk a C 16 vagy a Plus/4 esetében. Mindkét tipusú bövítésnél alapszabály, hogy a bövitö modult csak a gép kikapcsolt állapotában illeszthetjük a géphez, vagy vehetjük le arról! Ha a Plus/4 számítógéphez ROM memóriát bövítünk, akkor a gép már 128 K memórával rendelkezik.

5. MEGSZAKÍTASOK

A BASIC nyelvü programozás során használtuk a számítógép belsö óráját. Az óra léptetése érdekében az éppen futó programunk másodpercenként hatvanszor megszakadt, csak mi ezt nem vettük észre. A számítógéphez kapcsolt külsö eszközök is kérhetik a processzor müködésének felfüggesztését, azaz megszakítást kérhetnek. A megszakításnak egyszerüen fogalmazva az a szerepe, hogy az éppen futó programtól függetlenül, vele párhuzamosan ellásson bizonyos feladatokat. Mivel a gépben csak egy processzor van, ezért egyidöben vagy csak az aktuális program fut, vagy a megszakításkérés feldolgozása történik. Alapvetöen háromféle megszakításkezelést alkalmaznak a Commodore számítógépekben:

1. külsö eszköz megszakításkérése
2. futó program megszakításkérése
3. RESET gomb megszakításkérése

Nézzük meg hogyan kezeli a processzor az egyes tipusú megszakításokat!

KÜLSÖ ESZKÖZ MEGSZAKÍTASKÉRÉSE

A külsö eszköz a processzor un. IRQ vezetékén kér megszakítást. A kérés hatására a processzor

- befejezi az aktuális utasítás végrehajtását
- ha az I jelzöbit értéke 1, akkor nem foglalkozik a kéréssel
- ha az I jelzöbit értéke 0, akkor a verembe menti az SR és a PC regiszterek értékeit
- az I jelzöbitet 1-re állítja, igy letiltja a további megszakításokat
- beolvassa a PC-be a megszakítást kezelö alprogram kezdöcímét, és végrehajtja azt az RTI utasításig. Az utasítás hatására a veremböl kiolvassa a PC és az SR elmentett értékeit.
- a PC tartalmának megfelelöen folytatja tevékenységét

FUTÓ PROGRAM MEGSZAKÍTASKÉRÉSE

Az IRQ-hoz hasonlóan futó programból is kérhetünk megszakítást. Erre szolgál a BRK utasítás, amelyet az I jelzöbit állításával nem tudunk letiltani. Ilyenkor a PC az elmentése elött kettövel növekszik. További különbség az elözö pontban leírtakhoz képest az, hogy a BRK végrehajtásakor a B jelzöbit értéke 1 lesz. Ha a megszakítást a BRK okozta, akkor a rutin végrehajtása után a MONITOR programba ugrik a rendszer. Az assembly programjainkban nagyon sokszor alkalmazzuk a programok befejezésére a BRK utasítást.

RESET GOMB MEGSZAKÍTASKÉRÉSE

A megszakítások harmadik módja, amit a gép jobboldalán található RESET gombot benyomva aktivizálhatunk. A gépet lényegében alapállapotba helyezzük ezzel. A processzor megszakítja müködését, majd végrehajtja az $FFFC címen kezdödö RESET rutint.

III. UTASíTASOK, CíMZéS

1. UTASíTASKéSZLET

ADATMOZGATó UTASíTASOK

LDA op	: egy érték beolvasása az AC-ba	AC <-- op
LDX op	: egy érték beolvasása az XR-be	XR <-- op
LDY op	: egy érték beolvasása az YR-be	YR <-- op
STA op	: az AC tartalmának kiírása a memóriába	op <-- AC
STX op	: az XR tartalmának kiírása a memóriába	op <-- XR
STY op	: az YR tartalmának kiírása a memóriába	op <-- YR
TAX	: az AC tartalmát beírja az XR-be	XR <-- AC
TXA	: az XR tartalmát beírja az AC-ba	AC <-- XR
TAY	: az AC tartalmát beírja az YR-be	YR <-- AC
TYA	: az YR tartalmát beírja az AC-ba	AC <-- YR
TXS	: az XR tartalmát beírja az SP-be	SP <-- XR
TSX	: az SP tartalmát beírja az XR-be	XR <-- SP
PLA	: a veremtárból írás az AC-ba	AC <-- VT
PLP	: a veremtárból írás az SR-be	SR <-- VT
PHA	: az AC tartalmának írása a veremtárba	VT <-- AC
PHP	: az SR tartalmának írása a veremtárba	VT <-- SR

ARITMETIKAI UTASíTASOK

ADC op	: AC értékének, az operandusnak és a C bitnek az összegét az AC-ba írja	AC <-- AC+op+C
SBC op	: AC értékéből kivonja az operandust és a C bit ellentetjét, az eredményt az AC-be írja	AC<-AC-op-(1-C)
INC op	: az operandust eggyel növeli	op <-- op+1
INX	: az XR értékét eggyel növeli	XR <-- XR+1
INY	: az YR értékét eggyel növeli	YR <-- YR+1
DEC op	: az operandust csökkenti eggyel	op <-- op-1
DEX	: az XR értékét csökkenti eggyel	XR <-- XR-1
DEY	: az YR értékét csökkenti eggyel	YR <-- YR-1

LOGIKAI, ELTOLASI UTASíTASOK

AND op	: logikai ÉS müvelet az AC és az operandus között. Eredmény az AC-ban.	AC <- AC AND op
BIT op	: az operandus tesztelése az AC-ral, logikai ÉS müvelet az operandus és az AC között. Az eredményt nem írja sehová sem, csak a Z-t állítja.	N <-- op 7. bit V <-- op 6. bit
ORA op	: logikai VAGY müvelet az AC és az operandus között. Eredmény az AC-ban.	AC <-- AC OR op
EOR op	: logikai KIZARó VAGY müvelet az AC és az operandus között. Eredmény az AC-ban.	AC <- AC XOR op
ASL op	: bitenkénti léptetés balra, leesö bit a C bitbe kerül, a jobbról hiányzó bit értéke 0 lesz.	<------- C<-!76543210!<-0

```
LSR op : bitenkénti léptetés jobbra, a leesö      ------->
         bit a C-be kerül, a balról hiányzó  0->¦76543210¦->C
         bit értéke 0 lesz
ROL op : bitforgatás balra a C biten át             <-------
                                                C<-¦76543210¦<-C

ROR op : bitforgatás jobbra a C biten át          ------->

                                                C->¦76543210¦->C
```

ÁLLAPOTREGISZTER UTASÍTÁSAI

```
SEC    : a C (átvitel) bit beállítása 1-re              C <-- 1
SED    : a D (decimális mód) bit beállítása 1-re        D <-- 1
SEI    : az I (megszakítás) bit beállítása 1-re         I <-- 1
CLC    : a C (átvitel) bit nullázása                    C <-- 0
CLD    : a D (decimális mód) bit nullázása              D <-- 0
CLI    : az I (megszakítás) bit nullázása               I <-- 0
CLV    : a V (túlcsordulás) bit nullázása               V <-- 0
```

FELTÉTELES ELÁGAZÓ UTASÍTÁSOK

```
BCS op : ugrás, ha a C bit  1
BEQ op : ugrás, ha a Z bit  1
BMI op : ugrás, ha az N bit 1
BVS op : ugrás, ha a V bit  1
BCC op : ugrás, ha a C bit  0
BNE op : ugrás, ha a Z bit  0
BPL op : ugrás, ha az N bit 0
BVC op : ugrás, ha a V bit  0
```

A feltételes elágazó utasítás operandusa mindig egy abszolult cím, ahová ugrunk. Bizonyos okok miatt az 'ugrás' távolsága korlátozott: elöre $7E, visszafelé $81 a távolság a feltételes elágazó utasítástól mérve.

ÖSSZEHASONLÍTÓ UTASÍTÁSOK

```
CMP op : az AC  hasonlítása az operandussal,      AC < = > op
         az SR  bitjeinek állítása
CPX op : az XR  hasonlítása az operandussal,      XR < = > op
         az SR  bitjeinek állítása
CPY op : az YR  hasonlítása az operandussal,      YR < = > op
         az SR  bitjeinek állítása
```

UGRÓ UTASÍTÁSOK

```
JMP op : a megadott címre történö feltétel nélküli ugrás
JSR op : alprogram hívása feltétel nélkül
RTS    : visszatérés az alprogramból a hívás utáni helyre
RTI    : visszatérés a megszakítást kezelö alprogramból
```

EGYÉB UTASÍTÁSOK

```
NOP    : üres utasítás
BRK    : szoftveres megszakítás utasítása
```

2. CÍMZÉSI MÓDOK

A Commodore számítógépekbe épített mikroprocesszornak viszonylag kevés müveletet írhatunk elö, számszerint 56 db mnemonik van. A változatos címzési módoknak köszönhetöen viszont az alkalmazható különbözö utasítások száma 150. A címzési módok sokszínüsége talán nehéz tudománynak tünik, de részletes megértése után a használható utasítások kárpótolnak minket. Néhány egyszerü példával szemléltetve megadjuk a használható címzési módokat. Azt, hogy melyik utasítás melyik címzési móddal használható, a függelékben található táblázatból olvashatjuk ki.

KÖZVETLEN #ee 2 bájt

Az operandust a mnemonik után írjuk # jellel bevezetve. Az operandus lehet egy szám (hexadecimális, bináris, oktális vagy decimális számrendszerben megadva), de lehet egy ASCII karakter is.

Pl.:	LDA #$41	A9,41	tölts az AC-ba $41-t!
	CPY #20	C0,14	hasonlítsd össze az YR-t 20-szal!
	EOR #$DB	49,DB	végezd el a kizáró-vagy logikai müveletet az AC tartalma és a $DB szám között! Az eredményt írd vissza az AC-ba!

ABSZOLULT cccc 3 bájt

A mnemonik után szereplö tárcím vagy annak tartalma lesz a müvelet alanya.

Pl.:	LDA $3A0E	AD,0E,3A	töltsd az AC-ba a $3A0E címü tárhely tartalmát!
	STX $400B	8E,0B,40	írd ki az XR tartalmát a tár $400B címü helyére!
	LSR $30AC	4E,AC,30	végezd el a bitenkénti léptetést jobbra a $30AC címü tárhely tartalmával!
	JMP $3000	4C,00,30	ugorj a $3000 címü helyre! Lényegében a PC regisztert állítja át ez az utasítás.

NULLASLAPOS ABSZOLULT cc 2 bájt

Megegyezik az abszolult címzéssel, de a tárcímek itt $00cc alakúak, így az utasítás is rövidebb lesz.

Pl.:	LDA $90	A5,90	töltsd az AC-ba a $0090 címü tárhely tartalmát (az ST rendszerváltozót)!
	ORA $98	05,98	végezd el a VAGY müveletet az AC és a $0098 címü tárhely tartalma között. Az eredményt írd az AC-ba!
	STY $DA	84,DA	írd ki az YR tartalmát a $00DA címü tárhelyre!

IMPLIED, BELEÉRTETT 1 bájt

Sok utasításnak a mnemonikjából következik, hogy mivel kell az adott müveletet elvégezni. Az ilyen utasításoknál tehát nincs címrész.

Pl.:	CLC	18	nullázd az SR C bitjét!
	PHA	48	az AC tartalmát írd a verembe!
	TXS	9A	az XR értékét helyezd el az SR-ben!

X REGISZTERREL INDEXELT ABSZOLULT cccc,X 3 bájt

Az operandus tényleges címét úgy számítjuk ki, hogy a [címrész]-ben megadott címhez hozzáadjuk az XR aktuális értékét. Ha a [címrész] = $4A02 az XR tartalma pedig $C9, akkor a tényleges cím $4ACB lesz.

Pl.:	AND $30A0,X	3D,A0,30	végezd el az ÉS logikai müveletet az AC tartalma és azon memóriahely tartalma között, melynek címe az XR tartalmának és $30A0-nak az összege! Az eredményt az AC-ba írd vissza!
	CMP $3000,X	DD,00,30	hasonlítsd össze az AC-ral azon tárhely tartalmát, amelynek címe az XR tartalmának és $3000-nak az összege! Az SR-t megfelelöen állítsd be!

Y REGISZTERREL INDEXELT ABSZOLULT cccc,y 3 bájt

Analóg az elözöhöz, csak a cím kiszámításánál az YR aktuális értékét kell figyelembe venni.

Pl.:	LDA $4000,Y	B9,00,40	töltsd be az AC-ba azt az értéket, amelynek címe az YR tartalmának és $4000-nak az összege!

X REGISZTERREL INDEXELT NULLASLAPU cc,X 2 bájt

Az operandus tényleges címét itt is egy összeadás adja meg; az XR értékéhez adjuk hozzá a [címrész] -ben megadott nulláslapú értéket.

Pl.:	DEC $D8,X	D6,D8	csökkentsük eggyel azon tárhely tartalmát, amelynek a címe $D8 és az XR tartalmának az összege!

Y REGISZTERREL INDEXELT NULLASLAPU cc,Y 2 bájt

Analóg az elözövel, csak itt az YR tartalma adja meg az eltolási számot.

Pl.:	STX $DC,Y	96,DC	az XR tartalmát írjuk ki a tár azon helyére, amelynek a címe $DC és az YR aktuális tartalmának összege!

LDX $D0,Y B6,D0 másold be az XR-be azt a tártartalmat, amelynek a címét úgy kapod meg, hogy $D0-hoz hozzáadod az YR tartalmát!

INDIREKT (cccc) 3 bájt

A címet a [címrész]-ben megadott című szó tartalmazza. A rendszer tehát kiolvassa a cccc és a cccc+1 című tárhelyek tartalmát (pl.: $40 és $3D), majd elöállítja a tényleges címet (a példa szerint $3D40-át).

Pl.: JMP ($4000) 6C,00,40 átadja a vezérlést a $3A45 címre, ha $4000 : $45 és $4001 : $3A volt.

INDIREKT Y REGISZTERREL INDEXELT (cc),Y 2 bájt

Az indirekt és az Y regiszterrel indexelt címzési mód együttese. Elöbb meghatározzuk az indirekt címet, majd ehhez adjuk hozzá az indexregiszter (csak YR lehet) tartalmát. Ez a címzési mód csak a nulláslappal kapcsolatban használható.

Pl.: LDA ($D9),Y B1,D9 az AC-ba tölti a $3079 tárhely tartalmát, ha $00D9 : $45
$00DA : $30
YR : $34,
hiszen $3079 = $3045 + $34.

ADC ($E0),Y 71,E0 az AC-ba helyezi a $40CD címen lévö számnak, az AC tartalmának és a C bitnek az összegét, ha $00E0 : $AA
$00E1 : $40
YR : $23,
hiszen $40CD = $40AA + $23.

X REGISZTERREL INDEXELT INDIREKT (cc,X) 2 bájt

Az X'regiszter értékét hozzáadjuk a zárójelbeli címhez, az így kapott című szó adja meg a tényleges címet, alsó/felsö félcím formájában. Ez a címzési mód is csak a nulláslappal kapcsolatban használható.

Pl.: SBC ($B2,X) E1,B2 utasítás által meghatározott cím $42A3, ha XR : $23
$00D5 : $A3
$00D6 : $42,
hiszen $23 + $B2 = $D5. A $42A3 címen lévö számot és a C bit negátját vonja ki az AC-ból a gép, az eredményt visszaírja az AC-ba.

STA ($D0,X) 81,D0 kiírja az AC tartalmát a $4000 címre, ha XR : $11
$00E1 : $00
$00E2 : $40,
hiszen $11 + $D0 = $E1.

IV. SZOFTVER ESZKÖZÖK

1. A TEDMON MONITOR PARANCSAI

A TEDMON MONITOR-ba lépés kulcsszava MONITOR, vagy M és SHIFT O.

A -ASSEMBLE- A [cim] [mnemonik] [operandus]

Egy assembly sort lefordít gépi szintre, a kódot a tárba írja, az adott kezdöcímtöl kiindulva. Címet csak az elsö alkalommal kell beírni, azután automatikusan megadja a gép a következö címet. Ha a fordításnál hibát talál, akkor a sor végére egy kérdöjelet helyez el a rendszer.

Példa: A 3000 STA $4000

C -COMPARE- C [cím1] [cím2] [cím3]

összehasonlítás a tár két összefüggö területe között. Az egyik terület [cim1]-töl [cim2]-ig terjed, a másik terület elejét a [cim3] adja meg. Azonosság esetén nincs visszajelzés, eltérés esetén az eltérö értékek címét adja vissza a rendszer.

Példa: C D000 D400 3000

D -DISASSEMBLE- D [cím1] [cím2]

A fordítás inverze, a [cím1]-töl kezdödö tártartalmakat gépikódú utasításoknak vélve kiírja az assembly mnemonikot, és cím részt. Ha a tártartalom nem értelmezhetö utasítás résznek, akkor ??? jelzést kapunk. A visszafordítás a [cím2]-ig történik, annak hiányában 20 db bájtra korlátozódik a tevékenység. A kapott assembly listában javításokat végezhetük a kurzormozgató billentyük és a RETURN segítségével.

Példa: D F400 F4A0

F -FILL- F [cím1] [cím2] [érték]

összefüggö tárterület feltöltése azonos értékkel (pl.nullázás). A terület határait a [cím1] és [cím2] paraméterek adják meg.

Példa: F 3000 30FF 00

G -GO- G [cím]

Gépikódú, a tárban a [cím]-töl kezdve elhelyezkedö program végrehajtásának elindítása. Ha a [cím] rész nincs megadva, akkor a PC regiszter tartalma a mérvadó.

Példa: G 3000

H -HUNT- H [cím1] [cím2] [adatok]

A tár egy összefüggö területén ([cím1]-[cím2]) keres egy adategyüttest. Ha talál ilyet, akkor kiírja a kezdöcímet, ha nem, akkor nem jelez semmit.

Példa: H 3000 30FF 86 45

L -LOAD- L "file név",[egységszám]

Mágneses adathordozóról betölti a tárba az adott nevü file-t. Az egységszám a kazettás berendezésnél 1, a lemezes egységnél 8 és 11 közötti érték. Egységszám elhagyása esetén a kazettás berendezésröl olvas a rendszer. A betöltési cím megegyezik azzal a címmel, ahonnan kimentettük az állományt.

Példa: L "program12",8

M -MEMORY- M [cím1] [cím2]

Tártartalom megjelenítése a képernyön hexadecimális és karakteres alakban. A kurzormozgató és a RETURN billentyükkel lehetöségünk van módosításra is (RAM).

Példa: M 3000 30FF

R -REGISTERS- R

A regiszterek jelzéseit és tartalmukat megjeleníti a képernyön.

S -SAVE- S "file név",[eszköz szám],[cím1],[cím2]

Egy összefüggö tárterület ([cím1]-[cím2]) kiírása az adott egységszámú berendezésre. Az állomány PRG tipusjelzöt kap, nevét az utasításban kell megadni. Az egységszámok leírását lásd a LOAD-nál!

Példa: S "program12",8,3000,30FF

T -TRANSFER- T [cím1] [cím2] [cím3]

Tárterület másolása a tár egy másik helyére. A másolandó rész határait az elsö két cím, az új terület elejét a harmadik cím határozza meg.

Példa: T D000 D800 4000

V -VERIFY- V "file név",[egységszám]

Gépi kódú program kimentésének ellenörzésére szolgáló utasítás. A mágneses adattároló adott nevü állományát hasonlítja össze a tár tartalmával. A tárterület kezdöcímét az állományból olvassa ki a rendszer.

Példa: V "program12",8

X -EXIT- X

Kilépés a MONITOR programból a BASIC rendszerbe.

2. AZ ASS-16 ASSEMBLER PROGRAM HASZNALATA

Az ASS-16 assembler eredetileg Commodore 16 tipusú számítógépre írt fordítóprogram, de használható a Commodore Plus/4 -en is, s alig tér el a Commodore 64-re írt PROFI-ASS 64 programtól. Elöször be kell tölteni az ASS-16 fordítóprogramot, majd a BASIC-ben szokásos szerkesztési módokkal írhatjuk az assembly forrásprogramunkat. Az elsö nem BASIC utasítás elött szerepelnie kell a fordítás indítását elrendelö SYS 12224 utasításnak. A kész programunkat a BASIC-beli SAVE, LOAD parancsokkal rögzíthetjük az adathordozóra vagy tölthetjük be a tárba. A tárban lévö programot a RUN paranccsal elindítva a BASIC utasítások végrahajtása megkezdödik, ill. a fordítás elindul. Célszerü egy sorba egy assembly utasítást írni, de a ':' karakter több utasítást is elválaszthat egymástól. Egy sor alakja a következö:

[sorszám] [cimke] [mnemonik] [operandus] [megjegyzés]

ahol [sorszám] azonos a BASIC-ben alkalmazható sorszámmal,
[cimke] 8 karakterre azonosító egyedi név, amelyik csak betüvel kezdödhet. (Használata nem kötelezö.)
[mnemonik] a processzor utasításkészletének megfelelöen,
[operandus] a címzési módok valamelyikének megfelelö alkalmazás, változatos alakban felírva:

```
decimális szám:        LDA #223
bináris szám:          LDA #%10001110
hexadecimális szám:    LDA #$4D
ASCII jel:             LDA #"%"
kifejezés:             LDA $3000+200
szimbólum:             LDA TAR1
programszámláló:       BCC *+27,
```

[megjegyzés] ';' karakterrel kezdödö jelsorozat, amelyik a program jobb olvashatóságát segiti.

Tehát egy minta:

```
   200  IDE LDA TAROLO ; az AC-ba tölt az átmeneti pufferböl
```

Az operandusban alkalmazható müveletek felsorolása:

```
A+B összeadás              A-B kivonás          A*B szorzás
A!B VAGY                   A&B ÉS               A^B KIZARÓ VAGY
A<B A értéke B db bittel balra tolva
A>B A értéke B db bittel jobbra tolva
< A a kétbájtos cím alsó bájtja
> A a kétbájtos cím felsö bájtja
```

A FORDíTó DIREKTíVAI

*= programszámláló regisztert vezérlö direktíva

A PC regiszter feltöltésére ill. olvasására használható kód.
Alakja: *= [operandus] vagy [cimke]

Példák:
```
 200    *= $3000 ; a gépi kódú program elejének meghatározása

 300    CIM1 *= $3C00 ; egy terület kezdöcímének kijelölése
 301    CIM2 *= *+128 ; az elözö terület alapján másik jelölése

 400    BCC *+20 ; feltételes ugrás a PC+20 címre
```

<- értékadó direktíva

Szimbólum feltöltése értékkel úgy, hogy azt késöbb meg lehessen változtatni.

Példák:
```
 200 ciklv <- $01   ; cilusváltozó kezdöértékének beállítása

 300 ciklv <- ciklv+2 ; ciklusváltozó növelése a lépésközzel
```

.BYTE direktíva

A direktíva kifejezések értékeit helyezi el a tárba, az érvényes címtöl kezdve, folytonosan. Lényegében DATA mezö létrehozását eredményezi.

Példa:
```
 200 TAR .BYTE 0,0,0,0,0; egy tárterületre 5 db zérus érték
                          elhelyezése, a tár elejét a TAR szó
                          jelzi
```

.ASC direktíva

Szöveges adatok tárbeli elhelyezésére szolgáló direktíva (ASCII kód).

Példa:
```
 200 TAR2 .ASC "TELAPO" ; a TELAPO szöveget elhelyezi a TAR2-
                          vel jelzett címtöl kezdödöen
```

.WORD direktíva

Bármilyen 2 bájtos adat elhelyezése a tárba, fordított bájtállásban.

Példa:
```
 200 TAR3 .WORD *;a PC tartalmának mentése a TAR3-mal jelzett
                 memória helyre
```

.END direktíva

A fordítóprogramnak jelzi, hogy a forrásprogramnak vége van. Ekkor a fordítás eredménye vagy a tárban, vagy a fordítónak elöírtaknak megfelelö háttértárolón található meg.

Példa:
```
 1000 VEGE .END; a forrásprogram vége
```

.BAS direktíva

A fordítás megszakítása, és a BASIC interpreter müködésbe helyezése a funkciója. Ha ismét assemblálni akarunk, akkor alkalmazni kell a SYS 12230 utasítást.

Példa:
```
1000 vege .END; a forrásprogram vége
1010      .BAS; vissza a BASIC-be
1020      REM * assembly program-23
```

.GOTO direktíva

A fordítás során az assembly program néhány sorának fordítását elhagyhatjuk (pl.: program 'belövés') ezen direktíva alkalmazásával. Az azonosító után írt sorszámon fog folytatódni a fordítás.

Példa:
```
300 KI .GOTO 400 ; 300-400 -ig nem fordítja az assembly prog-
                   ramot
```

.OPT direktíva

A fordítóprogram számára írja elö, hogy a fordítás eredményét hová, milyen alakban helyezze el. Az egyes parancsokat OPCIóK - kal lehet elöírni. Ha egy direktívában több opciót alkalmazunk, akkor azokat a , karakternek kell elválasztania.

P opció	: a képernyöre kérjük a listát
P[szám] opció	: a nyomtatóra kérjük a listát, ahol a [szám] a logikai file-szám
OO opció	: a tárnak a *= direktívával meghatározott helyére kerüljön a tárgyprogram
O opció	: a forrásprogram utáni szabad területre kerüljön a tárgyprogram
O[szám]	: a fordítás nem a tárba, hanem egy elözöleg megnyitott állományba történik. A [szám] az állományhoz rendelt file száma
N opció	: a .OPT direktíva elözöleg elrendelt opcióinak hatása megszünik, újakat lehet elöírni

AZ ASS-16 ASSEMBLER SYNTAX HIBAÜZENETEI

0 : utasítás nélkül szereplö cimke
1 : érvénytelen operációs kód
2 : érvénytelen címzési mód
3 : nem megengedett karakter
4 : zárójel hiba
5 : nem megengedett kifejezés
6 : hiányzó ',' karakter
7 : a direktíva nem létezik
8 : a szimbólum érvénytelen
9 : a müveleti kód igy nem címezhetö

V. ASSEMBLY ALAPFELADATOK

A fejezetben olyan egyszerü alapfeladatokat közlünk, amelyekkel az assembly szintü programozás során gyakran találkozunk. A feladat megfogalmazása után a megoldó alprogram következik, az egyes utasítások funkciójának megadásával. Magyarázatot csak akkor közlünk, ha a megoldó program nehezen érthetö. A fejezetben ismertetett feladatokhoz hasonlóakat tartalmaz a Feladatgyüjtemény is.

EGYBAJTOS ARITMETIKA

1. FELADAT: adjuk össze a $34 és az $AC számokat, az eredményt a $3000 címre irjuk be!

```
$2000 LDA #$34        ; egyik szám kiirása $D0-ra
      STA $D0
      LDA #$AC        ; másik szám az AC-ba
      CLC             ; a C bit nullázása
      ADC $D0         ; az összeadás elvégzése
      STA $3000       ; az összeg kiirása a kivánt helyre
      RTS
```

2. FELADAT: vonjuk ki $D3-ból az $A4-t, az eredményt irjuk be az $E0 cimü helyre!

```
$3000 LDA #$A4        ; a kivonandó kiirása $2000-ra
      STA $2000
      SEC             ; a C bit 1-re állitása
      LDA #$D3        ; a kisebbitendö beállitása
      SBC $2000       ; az aritmetikai müvelet elvégzése
      STA $E0         ; a különbség kiirása a kivánt helyre
      RTS
```

KÉTBAJTOS ARITMETIKA

1. FELADAT: Adjuk össze az $1234 és az $ABCD számokat, az összeget a ($3000) cimü szóba irjuk ki bájtfordított sorrendben!

```
$2000  LDA #$12      ; az elsö szám  elhelyzése a ($3100) szó-
       STA $3100     ; ba
       LDA #$34
       STA $3101
       LDA #$AB     ; a második szám elhelyezése a ($3200) cí-
       STA $3200    ; mü szóba
       LDA #$CD
       STA $3201
       CLC          ; a C bit nullázása
       ADC $3101    ; a számok alsó bájtjainak összeadása
       STA $3001    ; az összeg kiirása a szó 2. bájtjába
       LDA $3200    ; a számok felsö bájtjainak összegzése
       ADC $3100
       STA $3000    ; az összeg kiirása a szó 1. bájtjába
       RTS
```

A két összeadás között a C bitet nem szabad nullázni, nehogy a belsö átvitel értéke elvesszen!

2. FELADAT: vonjuk ki az $ABCD-böl az $1234 számot, az eredményt a ($D0) címü szóba bájtfordított sorrendben írjuk!

```
$3000  LDX #$34     ; a kivonandó kiírása a ($2000) szóba
       STX $2000
       LDX #$12
       STX $2001
       LDA #$CD     ; a kisebbítendö alsó bájtja az AC-ba
       SEC          ; a C bit 1-re állítása
       SBC $2000    ; alsó bájtok kivonása
       STA $D0      ; eredmény alsó bájtja a helyére
       LDA #$AB     ; a kisebbítendö felsö bájtja az AC-ba
       SBC $2001    ; felsö bájtok kivonása
       STA $D1      ; eredmény felsö bájtja a helyére
       RTS
```

A két kivonási müvelet között nem szabad a C bit értékét módosítani!

EGYBAJTOS SZAM SZORZASA 2 HATVANYAIVAL

1. FELADAT: szorozzuk meg a $03-t nyolccal, az eredményt az $E0-ra írjuk ki!

```
$2000  LDA $03      ; szorzandó az AC-ba
       ASL          ; szorzás 2-vel
       ASL          ; szorzás 2-vel
       ASL          ; szorzás 2-vel
       STA $D0      ; szorzat a kívánt helyre
       RTS
```

2. FELADAT: osszuk el az $A4-t néggyel, az eredményt az XR-be kérjük!

```
$2000  LDA #$A4     ; osztandó az AC-ban
       LSR          ; osztás 2-vel
       LSR          ; osztás 2-vel
       TAX          ; eredmény az XR-be
       RTS
```

ELAGAZASOK

1. FELADAT: vizsgáljuk meg a $D0 címü tárhely tartalmát! Ha nem nulla, akkor csökkentsük eggyel a tártartalmat, máskülönben növeljük meg eggyel azt!

```
$2000  LDX #$00     ; a kulcs érték az XR-be
       CPX $D0      ; a hasonlítás
       BNE $2020    ; ha nem nulla, akkor ugrás a csökkentésre
       INC $D0      ; növelés és vége
       RTS
       ;
       ;
$2020  DEC $D0      ; csökkentés és vége
       RTS
```

CIKLUSOK SZERVEZÉSE

1. FELADAT: Előírt lépésszámú, hátultesztelö ciklus.
8 darab A betüt írassunk ki a képernyö elsö sorába!

```
$2000   JSR $C567     ; képernyö letörlése
        LDA #$01      ; az A betü kódja
        LDX #$00      ; ciklusváltozó kezdöértéke
$2007   STA $0C00,X   ; ciklusmag
        INX           ; ciklusváltozó növelése a lépésközzel
        CPX #$08      ; végértekhez hasonlítás
        BNE $2007     ; ciklusmag újbóli végrehajtása
        RTS           ; kilépés a ciklusból
```

2. FELADAT: Nem előírt lépésszámú, hátultesztelö ciklus.
Keressük meg a nulláslap elsö olyan bájtjának a címét, amelyikben az $FC van. Az illetö bájt címét az AC tartalmazza a program végén!

```
$2000   LDX #$00      ; a nulláslap aktuális bájtjának címe
$2002   LDY $00,X     ; ciklusmag
        INX
        CPY $FC       ; kilépési feltétel vizsgálata
        BNE $2002     ; ciklusmag ujra lejátszása
        DEX           ; a ciklus utáni utasítások
        TXA
        RTS
```

MASZKOLASI FELADATOK

Mint tudjuk a memória legkisebb címezhetö egysége a bájt. Ezt tudjuk olvasni, írni közvetlenül. Az un. maszkolási feladatokkal az egyes bitek értékeit is tudjuk olvasni, illetve írni. A feladatosztályban a logikai alapmüveleteket használjuk fel.

1. FELADAT: állapítsuk meg a $3000 címü tárhely legmagasabb bitjének értékét, a bit értékét írjuk az XR-be!

```
$2000   LDA #$80      ; maszk beírása              1000 0000
        AND $3000     ; bitvizsgálat               bbbb bbbb
        STA $D0       ; az eredmény vizsgálata     b000 0000
        LDX #$80
        CPX $D0
        BNE $2020     ; ha a vizsgált bit 0, akkor ugrás
        LDX $#01      ; a vizsgált bit 1 volt
        RTS
        ;
        ;
$2020   LDX #$00      ; a vizsgált bit 0 volt
        RTS
```

2. FELADAT: a $3000 címü bájt legalacsonyabb bitjét nullázzuk ki de a többi értékét ne változtassuk!

```
$2000   LDA #$FE    ; a maszk beírása az AC-ba     1111 1110
        AND $3000   ; a bit 0-ra állítása          bbbb bbb0
        STA $3000
        RTS
```

VI. LEBEGÖPONTOS ARITMETIKA GÉPI KÓDBAN

A számítógépes feladatmegoldások jó részében matematikai müveletek sokaságát végeztetjük el a számítógéppel. Most azt írjuk le, hogy hogyan dolgozik a gép a valós számokkal. Mint tudjuk a valós számokat a Commodore BASIC 5 bájton ábrázolja: első bájt a bináris kitevő 128-as többlettel, a további négy bájton a bináris mantissza szerepel, a legmagasabb bit az elöjelé. A ROM-ban a lebegöpontos számokkal kapcsolatban több eszköz is található, de mind közül a legfontosabb az un. FAC (Floating point ACcumulátor), melynek címe $61-$65. A $61 a kitevö bájt, a többi 4 db a mantisszáé. Az aritmetikai müveletek majdnem mindegyike használja a FAC-t. További ilyen lebegöpontos valós számok kezelésére alkalmas tárhelyek is ismertek: ARG $69-$6D (FAC2) és külön név nélkül: $57-$5B ill. $5C-$60. Az aritmetikai rutinok valós operandusait elhelyezhetjük a RAM tetszöleges, de összefüggö területén is. Ilyenkor az operandus kezdöcímét az AC, és az YR tartalmazza (AC - alsó, YR - felsö bájt). Ezt a tényt (YA) jelöléssel fogjuk leírni a továbbiakban. Természetesen a ROM-ban is elhelyeztek valós számkonstansokat (pl. $A00F címen kezdödik a négyzetgyök 2 értéke). Azt is külön jelöljük majd, hogy az (YA) a ROM, vagy a RAM területre vonatkozik-e. Az aritmetikai rutinok belépési címeit tartalmazó táblázatban minden további lényeges információ megtalálható a lebegöpontos aritmetikáról. Most egy példaprogramon keresztül mutatjuk be a rutinok alkalmazásának technikáját.

Feladat: szorozzuk össze a π-t egy, a billentyüzetröl beadott egész számmal!

Program:

```
100 SYS 12224
110 .OPT P,OO
120 *= $2000
130 JSR $885A     ;INPUT A BILLENTYÜZETRÖL
140 LDA $3B       ;BASIC MUTATÓ A VEREMBE
150 PHA
160 LDA $3C
170 PHA
180 LDA #0        ;A SZÖVEG ELEJE A BASIC MUTATÓBA
190 STA $3B
200 LDA #$02
210 STA $3C
220 JSR $0479     ;CHRGOT RUTIN HÍVÁSA
230 JSR $A37F     ;ASCII KONVERZIÓ A FAC-BA
240 LDA #$39      ;A π ALSÓ CÍME
250 LDY #$94      ;A π FELSÖ CÍME
260 JSR $A0DC     ;ARG= (YA) ROM-BÓL
270 JSR $A07B     ;FAC=ARG*FAC
280 PLA           ;A BASIC MUTATÓ VISSZAALLÍTÁSA
290 STA $3C
300 PLA
310 STA $3B
320 LDY #1        ;ELÖKÉSZÍTÉS
330 JSR $A469     ;FAC KIÍRÁSA ASCII ALAKBAN
340 RTS
350 .END
```

VII. A GYAKRABBAN HASZNALT KERNAL RUTINOK LEíRASA

A KERNAL rutinok az operációs rendszer nagyon fontos részét jelentik. Az input/output müveletek, az óraidözítés, a memória kezelés programozásakor a programozó kész rutinokkal dolgozhat. A KERNAL programok alkalmazása nagy figyelmet kíván. Az egyes rutinok hívása elött vagy egy másik rutint kell alkalmazni, vagy az egyes regiszterekbe kell alkalmas értékeket elhelyezni. A KERNAL rutinok egy szabványos ugrótáblázaton (minden Commodore gépen azonos) keresztül érhetök el a legkényelmesebben. Az ugrótáblázatban egy-egy JMP utasítás található, amellyel elérhetjük a tényleges programrészletet.

FIGYELEM: a rutinok leírásánál megadjuk a belépési címet, az elökészítö rutin(ok) azonosítóit, a használt regiszter(ek) jelét!

OPEN $FFC0 (SETLFS, SETNAM) [AC, XR, YR]
Egy file megnyitására szolgáló program. A SETLFS (a file paramétereket beállító), a SETNAM (file nevét beállító) elökészítö rutin használata az OPEN elött kötelezö!

SETLFS $FFBA (-) [AC, XR, YR]
A file paramétereket beállító rutin. A logikai file-számot az AC az egységszámot az XR míg a megnyitási módot az YR tartalmazza a rutin meghívása elött.

SETNAM $FFBD (-) [AC, XR, YR]
A file nevét beállító rutin. Elözöleg az AC-ba kell tölteni a név karaktereinek a számát. Az XR, YR tartalmazzák a név elsö karakterének tárbeli címét (LB,HB).

CHKOUT $FFC9 (OPEN) [AC, XR]
Az output csatorna kijelölésére szolgáló alprogram. Elözöleg az OPEN rutinnal megnyitjuk az állományt, majd a logikai file-számot az XR-be töltjük. Ezután hívható a CHKOUT.

CHROUT $FFD2 (OPEN, CHKOUT) [AC]
Egy karaktert helyez el az elözöleg OPEN-nel megnyitott, CHKOUT-tal outputra állított csatornára. Ha a CHKOUT elökészítö rutin elmarad, akkor a képernyöre küldi a karaktert. A karakter kódját az AC-ból olvassa ki a rendszer.

CHKIN $FFC6 (OPEN) [AC, XR]
Ugyanaz mint a CHKOUT, csak itt input csatornát definiálunk. Alapértelmezésben a billentyüzet az input eszköz.

CHRIN $FFCF (OPEN, CHKIN) [AC, XR]
Az input csatornáról egyetlen karaktert olvas be az AC-ba.

READST $FFB7 (-) [AC]
Az AC-ba tölti a $90 címü rendszerváltozó (az ST) értékét. Az I/O müveletek során esetlegesen elöforduló hibákat vagy az adatállomány végének elérését lehet így figyelni.

CLRCHN $FFCC (-) [AC, XR]
Visszaállítja az I/O csatornák kezdeti (default) értékét. Azaz az aktuális input eszköz a billentyüzet (0), az aktuális output eszköz a képernyö (3) lesz.

CLOSE $FFC3 (-) [AC, XR, YR]
Nyitott file lezárására szolgáló program. A logikai file-számot elözöleg az akkumulátorba kell tölteni.

PLOT $FFF0 (-) [AC, XR, YR]
Ha a C bit 1, akkor a kurzor pozícióit tölti az indexregiszterekbe (XR a vízszintes, YR a függöleges koordinátákat fogja tartalmazni). Ha a C bit 0, akkor a kurzor beállítására szolgál a rutin. Elözöleg a megfelelö indexregisztereket be kell állítani.

PRINT $FF4C (PLOT) [AC, XR]
A kurzor aktuális helyére (ld. PLOT) kiír egy karaktert. A karakter kódját elözöleg az AC regiszterbe kell tölteni. Ez a rutin mindig a képernyöre ír, függetlenül attól, hogy mi az aktuális output eszköz. A vezérlökódokat a szokásos módon kezeli.

TALK $FFB4 (-) [AC]
A soros buszra helyezett eszközt 'beszélö' állapotba hozza. Az eszköz egységszámát elözöleg az AC-ba kell tölteni. Az eszköz ezután input berendezésként fog funkcionálni.

TKSA $FF96 (TALK) [AC]
Megnyitási mód elrendelése az input eszköznek. Az eszközt elözöleg a TALK rutinnal 'beszélö' állapotba kell hozni. A megnyitási mód számát az AC-ból olvassa ki a gép.

ACPTR $FFA5 (TALK, TKSA) [AC, XR]
A soros buszról beolvas a gép egy bájtot az AC-ba. A buszon kell egy 'beszélö' eszköznek lenni (TALK, TKSA)!

UNTLK $FFAB (-) [AC]
A soros buszon lévö, elözöleg 'beszélö' állapotba hozott eszköz alapállapotba ('figyelö') helyezése.

LISTEN $FFB1 (-) [AC]
A soros buszra füzött valamelyik eszköz 'hallgató' állapotba helyezése. Az eszköz egységszámát elözöleg az AC-ba kell írni. A rutin lefutása után az eszköz output berendezésként fog müködni.

SECOND $FF93 (LISTEN) [AC]
A 'hallgató' állapotba hozott eszköznek a megnyitási mód számát ezen rutin alkalmazásával tudjuk elöírni. Az AC-ból veszi a számot a rendszer.

CIOUT $FFA8 (LISTEN, SECOND) [AC]
A soros buszon keresztül egy bájtot küld ki az elözöleg 'hallgató' állapotba helyezett (LISTEN, SECOND) eszköznek. Az output adatot az AC-ba kell tölteni a rutin meghívása elött!

UNLSN $FFAE (-) [AC]
A soros buszra történö adatforgalmazás befejeztével ezen rutin kiüríti a puffert, és megszünteti minden eszköz 'hallgató' státuszát.

GETIN $FFE4 (OPEN, CHKIN) [AC, XR, YR]
Egyetlen bájt beírása az AC-ba a billentyüzetröl vagy az RS 232 csatornáról. Ha a puffer üres, akkor 0 kerül az akkumulátorba. Ha az input csatorna a kazettás egység vagy a soros busz, akkor a megfelelö CHRIN rutin kerül végrehajtásra.

SCNKEY $FF9F (-) [AC, XR, YR]
Billentyüzetet kezelö rutin. Ha a megszakítást letiltottuk és a billentyüzetröl várunk adatot, akkor ezt a rutint használjuk. Meghíváskor a lenyomott billentyü kódja a pufferbe kerül, és a a visszatéréskor ez az érték van az akkumulátorban. Az AC $FF lesz, ha nincs lenyomva semmi.

RESTOR $FF8A (-) [AC, XR, YR]
Visszaállítja az operációs rendszer $0312-$0331 területen lévö vektorait.

RDTIM $FFDE (-) [AC, XR, YR]
Az óra beolvasására szolgáló rutin. A memóriában a $A3-$A5 területen lévö óra aktuális értékét helyezi el a munkaregiszterekbe az alábbiak szerint:
AC : $A5 XR : $A4 YR : $A3.

SETTIM $FFDB (-) [AC, XR, YR]
Az elözö rutin fordítottja, a munkaregiszterek értékeinek megfelelöen beállítja az órát. A regiszter : memória megfeleltetést az RDTIM rutinnál leírtuk.

UDTIM $FFEA (-) [AC, XR]
Szintén az órával kapcsolatos rutin, eggyel növeli az óra értékét. Alapállapotban a megszakítási rendszer másodpercenként 60-szor hívja meg az UDTIM rutint. A rutin figyeli a STOP billentyüt is.

KERNAL HIBAKEZELÉSE

Hiba esetén vagy az AC-ba kerül a hibakód, vagy maga az ST rendszerváltozó értéke tájékoztat a hiba jellegéröl.

HIBAKÓDOK AZ AC-BAN

0 a program futását a STOP billentyü szakította meg
1 sok file-t nyitottunk meg egyszerre
2 a file már nyitva van
3 a file nincs megnyitva
4 nem találja a file-t a rendszer
5 az egység nincs jelen
6 nem INPUT file
7 nem OUTPUT file
8 hiányzik a file neve
9 illegális egységszám

VIII. GÉPI KÓDÚ PROGRAMOK BEÉPÍTÉSE BASIC KÖRNYEZETBE

Legtöbbször nem gépi kódú programokat futtatunk, hanem a BASIC nyelvü programunkba integrálunk gépi kódú betéteket. A gépi kódú alprogramjaink minden esetben az RTS utasítással fejezödnek be, igy az alprogram végrehajtása után a BASIC nyelvü hivó részre kerül a vezérlés vissza. A gépi kódú alprogramok hívása kétféleképpen történik: SYS utasítással vagy USR függvénnyel.

SYS UTASÍTAS

Szintaxisa: SYS decimális cím.
Hatására átadódik a vezérlés a címben meghatározott helyre. Az itt kezdödö gépi kódú alprogram végrehajtása megkezdödik. Az alprogramból való visszatérés címét a rendszer automatikusan megörzi.
Pl. legyen a gépi kódú programunk az alábbi:

```
$2000  A9 01      LDA #$01
       8D 0C 00   STA $0C00
       60         RTS
```

ezt az alprogramot beépithetjük az alábbi BASIC programba:

```
100  SCNCLR
110  SYS 8192
120  END
```

A SYS utasítással lehetöségünk van a processzor regisztereinek értéket átadni a BASIC programból, még a gépi kódú rutin hívása elött. Az értékeket a POKE utasítással kell elhelyezni az alábbiak szerint:

AC	$07F2	azaz	2034
XR	$07F3		2035
YR	$07F4		2036
SR	$07F5		2037.

Pl. az elözö feladathoz hasonlóan most is egy A betü jelenik meg a képernyö bal felsö sarkában:

gépi kódú rutin

```
$2000   8D 0C 00   STA $0C00
        60         RTS
```

BASIC program

```
100 SCNCLR
110 POKE 2034.1
120 SYS 8192
130 END
```

Megjegyzendö, hogy a visszatéréskor a regiszterek tartalmát a rendszer bemásolja a fent említett tárhelyekre.

USR FÜGGVÉNY

Szintaxisa: USR (paraméter), ahol a paraméter tetszöleges aritmetikai kifejezés lehet. Vigyázat az USR függvény és nem utasítás! A függvény alkalmazása elött a meghívandó gépi kódú rutin kezdöcímét a ($0501) címü szóba kell beírni bájtfordított sorrendben.

Ha az USR paraméterét is át akarjuk adni a hívott gépi kódú rutinnak, akkor a rutint a JSR $9DE4 utasítással kell kezdeni. Ennek hatására az USR paraméterének felsö bájtja az AC-ba, míg alsó bájtja az YR-be másolódik. Ekkor természetesen a paraméter 0 és 65535 közötti érték lehet csak. Ha a gépi kódú rutint az RTS helyett a JMP $9A76 utasítás zárja le, akkor az USR függvény értéke az AC/YR regiszterpár utoló értéke lesz, bájtfordított értelemben. A fentieket megvilágítja egy egyszerü példa:

Gépi kódú program:

```
$2000  20 E4 9D   JSR $9DE4    ; paraméter átvétele
       C8         INY
       99 00 0C   STA $0F1F,Y
       4C 76 9A   JMP $9A76    ; AC/YR értékének visszaadása
```

BASIC program:

```
10 SCNCLR:X=0
20 POKE 1281,0  : REM a cím alsó fele
30 POKE 1282,32 : REM a cím felsö fele
40 X=X+USR(512) : REM AC-ba 2, YR-be 0 kerül a rutin elött
50 PRINT X      : REM az X az AC/YR utolsó értékét örzi
60 END
```

IX. BASIC RUTINOK BEÉPíTÉSE ASSEMBLY KÖRNYEZETBE

Az elözö feladat fordítottja, amikor az assembly programunkba építünk be BASIC rutinokat a ROM-ból. A BASIC rutinok használata nagy figyelmet kíván. Az alábbiakban bemutatjuk a CIRCLE utasítás beépítését egy assembly programba. Az utasítás belépési címe $C01E (ld. függelék). A rutin meghívása elött a verembe írjuk a BASIC mutató értékét ($3B, $3C), majd a rajzolás után visszaállítjuk azt az eredeti értékre.
A kör paraméterei: színkód 1, középpont koordinátái 100,100 sugár 50.

```
100 SYS 12224
110 .OPT P,OO
120 *= $1600
130 LDA $3B                ; BASIC MUTATO ELMENTESE
140 PHA
150 LDA $3C
160 PHA
170 LDA #< PAR             ; A BASIC MUTATÓ ATíRASA
180 STA $3B
190 LDA #> PAR
200 STA $3C
210 JSR $0479              ; CHRGOT RUTIN HíVASA
220 JSR $C01E              ; CIRCLE RUTIN HíVASA
230 PLA                    ; A BASIC MUTATÓ VISSZAALLíTASA
240 STA $3C
250 PLA
260 STA $3B
270 RTS                    ; RUTIN VÉGE
280 PAR .ASC "1,100,100,50"
290 .BYTE 0
300 .END
```

Fordítás után a GRAPHIC 1,1:SYS 5632 paranccsal hívható a rutin.

X. FELADATOK, PROGRAMOK

ADATMOZGATÓ UTASÍTÁSOK

1. Töltsük fel közvetlen címzéssel az AC-t, majd a TEDMON MONITOR parancsainak alkalmazásával nézzük meg az AC és az SR tartalmát! Próbáljunk ki több értéket is (pl. $32,$00,$FF)!

2. Töltsük fel közvetlen címzéssel az AC, az XR ,az YR tárhelyeket, majd a TEDMON MONITOR R parancsot alkalmazzuk!

3. A TEDMON MONITOR M parancsával írjunk három értéket a $3000-tól kezdödö helyekre, majd abszolult címzést alkalmazva a három értéket írjuk be rendre az AC az XR és az YR tárhelyekre! A müvelet helyességéröl az R parancs alkalmazásával gyözödjünk meg!

4. Töltsük fel közvetlen címzéssel az AC-t az XR és az YR tárhelyeket, majd ilyen sorrendben írjuk ki azok tartalmát a tár $3000 -tól kezdödö összefüggö területére! A müvelet helyességéröl most is a TEDMON MONITOR M parancsával gyözödhetünk meg.

5. Az alábbi program elemzésével töltsd ki a táblázatot! A [cim] jelzésü fejlécben azokat a címeket alkalmazd, amelyekbe a program végrehajtása során érték kerül. Az értékadásnak megfelelö programsoroknál írd be az értékeket a táblázatba!

[cím]:	AC	XR	YR	....	....	....	....
LDA #$00							
STA $57							
LDA #$0E							
STA $58							
LDY #$04							
LDA #$41							
STA $0E00							
STA ($57),Y							

6. Az utasítások táblázata alapján állítsd elö a fenti program gépi kódját! A megoldásodat a TEDMON MONITOR A assembláló parancsával ellenörizheted.

7. Az alábbi gépi kódú programot az utasítások táblázata segítségével írd át assembly alakba (disassemblálás)! Megoldásod helyességéröl a számítógép használatával gyözödj meg (TEDMON MONITOR D parancs)! Mi a program funkciója?

```
$2000    A9,41
$2002    A2,F0
$2004    9D,00,0E
$2007    00
```

8. Add meg az alábbi BASIC program assembly megfelelöjét!

```
100 POKE 64,A
110 POKE 32768,X
120 POKE 64+X,A
```

9. Az elözö feladatban közölt BASIC programhoz hasonlóan add meg az alábbi assembly programrészlet megfelelöjét!

```
LDX $CC
STA $BB,X
LDA $2000,Y
LDA ($FF),Y
LDA ($FF,X)
```

10. Melyik a hibás assembly utasítás az alábbiak közül?

```
LDX $FF,X
STX $3000,X
LDY $BB,X
LDA ($D0,Y)
LDX ($D0),Y
TXY
TYS
LDY #$A00B
```

11. Add meg az alábbi BASIC utasításoknak megfelelö assembly sorokat!

```
POKE 32+X,PEEK(20)
POKE 65535,PEEK(15+X)
POKE 32768+X,PEEK(X)
```

12. Használjuk a veremmemóriát a következö feladatok megoldásánál!

a., Cseréljük fel az YR és az AC tartalmát!
b., Helyezzük el a verembe a BASIC szöveg mutató ($3B,$3C) értékét!
c., Állítsuk vissza a BASIC szöveg mutatót a veremböl!
d., A veremmutató értékét mentsük ki a nullászlap $D0 rekeszébe!
e., írjuk az AC-ba az állapotregiszter értékét!

13. Megírandó az az assembly programrészlet, amellyel a számítógép AC, XR, YR, SR regiszterinek az értékei a verembe menthetöek a megadott sorrendben.

14. Oldjuk meg az elözö feladat inverzét, azaz olvassuk ki a verembe helyezett értékeket! A beírás sorrendjét is az elözö feladatban találjuk.

15. Mi az alábbi assembly program funkciója?

```
$3000    PHA
$3001    TXA
$3002    PHA
$3003    TYA
$3004    TAX
$3005    PLA
$3006    TAY
$3007    PLA
$3008    BRK
```

ARITMETIKAI UTASíTASOK

1. Az alábbi utasítások végrehajtása után mi lesz az SR N és Z bitjének az értéke?

```
a.,  LDX #$FF        b.,  LDY #$AA        c.,  LDY #$7F
     INX                  STY $3000            INY
                          DEC $3000
```

2. Növeljük meg az AC aktuális értékét eggyel! Az egyik megoldásban ne használjuk egyik indexregisztert sem, a másik megoldásban ezt a korlátozó feltételt ne vegyük figyelembe!

3. Ha a számítógép elvégzi az ADC #$0A müveletet, akkor mi lesz az SR N,Z,C jelzöbitjeinek az értéke? Az összeadás elött az AC és a C bit értékei az alábbiak voltak:

```
a., AC=$00  C=0      b., AC=$FF  C=1      c., AC=$7F  C=1
d., AC=$26  C=0      e., AC=$36  C=1      f., AC=$F6  C=0
```

Határozd meg az AC új értékét is!

4. Oldjuk meg az elözö feladatot úgy is, hogy az elvégzendö müvelet assembly utasítása SBC #$06 legyen!

5. Mi lesz az AC és az SR N,Z,C jelzöbitek értéke az alábbi assembly programok végrehajtása után?

```
a., LDA #$4D          b., LDA #$4D
   CLC                   ADC #$4C
   ADC #$4C              BRK
   BRK

c., LDA #$FA          d., LDA #$FA
   CLC                   SEC
   ADC #$5A              ADC #$5A
   BRK                   BRK
```

6. Mi az alábbi assembly programok funkciója?

```
a.,  LDX #$76          b.,  LDA #$17
     STX $E0               STA $D0
     LDX #$01              LDA #$4A
     STX $E1               STA $D1
     LDA #$B5              SEC
     CLC                   LDA $D0
     ADC $E0               SBC #$A1
     STA $3000             STA $D0
     LDA #$01              LDA $D1
     ADC $E1               SBC #$22
     STA $3001             STA $D1
     BRK                   BRK
```

7. írd meg azt az assembly programot, amelyik összeadja a 345 és a 234 decimális számokat! Egy másik program segítségével határozd meg a két szám különbségét is!

LOGIKAI, ELTOLASI UTASíTASOK

1. Mi lesz az AC és az SR N,Z bitjeinek az értéke az alábbi müveletek elvégzése után?

```
a., LDA #$37        b., LDA #$D2        c., LDA #$B7
    AND #$A3            ORA #$4C            EOR #$3D
```

2. Végeztetsd el a számítógéppel a logikai és, a vagy, és a kizáró vagy müveleteket a 247 és a 119 között assembly programokkal!

3. A megfelelö logikai müveletek alkalmazásával oldd meg az alábbi feladatokat!
   a., A $FF06 regiszter 3. bitjét töröljük, a többi marad!
   b., Nullázzuk az AC-t!
   c., Állítsuk 1-re a $FF07 regiszter 3. bitjét!
   d., A $FF0F regiszter legmagasabb bitjét negáljuk, a többit hagyjuk változatlanul!

4. Az alábbi program elemzése után töltsük ki az alábbi táblázat pontozott helyeit!

cím	utasítás	AC	XR	YR	$008C	N	Z	C
$2000	LDA #$4A	...	...	...	....	..	..	..
$....	TAX	...	...	...	....	..	..	..
$....	INX	...	...	...	....	..	..	..
$....	STX $8C	...	...	...	....	..	..	..
$....	INC $8C	...	...	...	....	..	..	..
$....	EOR $41,X	...	...	...	....	..	..	..
$....	BRK							

5. Az alábbi programok végrahajtása után mi lesz az SR N, V, Z bitjeinek az értéke?

```
a., LDA #$FE        b., LDX #$7F        c., LDY #$22
    STA $D0             STX $D0             TYA
    BIT $D0             LDA #$F7            INY
    BRK                 BIT $D0             STY $D0
                        BRK                 BIT $D0
                                            BRK
```

6. írjuk, meg azt az assembly programot, amellyel a 24 decimális számot 2-vel megszorozhatjuk!

7. Osszuk el a 64 decimális értéket kettövel! Az assembly programban az eltolási és/vagy forgatási utasításokat használjuk!

8. Mi lesz az AC tartalma az alábbi programok hatására?

```
a.,  $1FFE   LDA  #$1A          b.,  $3000   LDA  #$C9
     $2000   STA  $D0                $3002   CLC
     $2002   ASL                     $3003   ROR
     $2003   CLC                     $3004   ASL
     $2004   ADC  $D0                $3005   SEC
     $2006   ASL                     $3006   LSR
     $2007   BRK                     $3007   ROL
                                     $3008   BRK
```

ÖSSZEHASONLíTó, FELTéTELES ELAGAZTATó, UGRó UTASíTASOK

1. Elemezve a programokat, töltsd ki a táblázatok pontozott helyeit!

a.,
```
$2000  LDX  #$4D
$2002  CPX  $3000
$2005  BRK
```

$3000	N	Z	C
$6F	..	..	..
$1A	..	..	..
$FF	..	..	..

b.,
```
$2000  LDA $D0
$2002  CMP #$3C
$2004  BRK
```

$D0	N	Z	C
$7B	..	..	..
$09	..	..	..
$FF	..	..	..

2. Add meg az SR N, Z, C jelzöbitjeit, ha egy CMP utasítással összehasonlítjuk egymással az alábbi két-két számot!
   a., 128-t 127-el, b., 127-t 128-al, c., 128-at 128-al
   írd meg a feladatot megoldó assembly programokat is!

3. A BASIC feladatok assembly megfelelöjét és az assembly sorok BASIC megfelelöjét írd le!

a., IF AC=24 THEN GOTO [cima]

b.,
```
CMP #$4A
BCC [cimb]
```

c., IF YR>=34 THEN GOTO [cimc]

d.,
```
CPX #$FD
BEQ [cimd]
```

e., IF XR<100 THEN GOTO [cime]

f.,
```
CPY #$88
BCS [cimf]
```

4. A feltételes ugrás UTASíTASát rendszerint egy összehasonlító utasítás elöz meg (valamelyik REGISZTER értékét hasonlítja a megcímzett OPERANDUS-hoz és felülírja az SR valamelyik JELZöbitjét). Az alábbi táblázat elsö sorát segítségül kitöltöttük, a többi beírása legyen a te feladatod! Egy utasítás csak egyszer szerepel a táblázatban!

UTASíTAS	UGRAS HA	REGISZTER...OPERANDUS	JELZö=éRTéK	
BCS		nagyobb egyenlö	C	1
BEQ		.....	..	..
...		nem egyenlö	..	..
...		kisebb	C	
BMI		.....	..	..
...		.....	N	0

5. írd meg azt az assembly programot, amelyik a nulláslapon szereplö elsö olyan címet meghatározza, amelynek a tartalma 128. A címet írjuk be a verembe!

6. írassuk ki a képernyöre az ABC betüit úgy, hogy minden karaktert egy-egy szóköz válasszon el egymástól!

7. A $3000-$3040 tárterületen található $00 értékeket számoljuk le, az eredményt írjuk be az XR-be! Mind a BASIC, mind az assemly programot készítsük el!

8.Állapítsd meg az alábbi programok funkcióját!

```
a.,  $2000   LDX #$00          b.,  $3000  LDA #$48
     $2002   LDY #$00               $3002  STA $2000
     $2004   LDA $3000,X            $3005  LSR $2000
     $2007   CMP #$00               $3008  LDX $2000
     $2009   BNE $2040              $300B  CPX #$01
     $200B   INX                    $300D  BNE $3005
     $200C   CPX #$31               $300F  BCS $3014
     $200E   BNE $2004              $3011  BRK
     $2010   BRK                    $3012  LDY #$41
     .....                          $3014  STY $0D00
     $2040   INY                    $3017  JMP $3011
     $2043   STY $4000
     $2044   JMP $200B
```

9.A TEDMON MONITOR A parancsának használata során lemaradt a feliratról az utasítás operandusa, amelynek beírása a te feladatod. Az SR milyen bitállásánál történik az ugrás?

```
a., $2000  D0  29  BNE $....      jelzö:...  értéke:...
b., $2040  B0  EB  BCS $....      jelzö:...  értéke:...
c., $200A  70  F5  ... .....      jelzö:...  értéke:...
```

10.Az elözö feladathoz hasonlóan 'elromlott a TEDMON MONITOR A funkciójának kijelzöje '. Egészítsd ki a sorokat a jelzöbitek nevének és értékének feltüntetésével együtt!

```
a., $2000  ..  ..  BMI $206C      jelzö:...  értéke:...
b., $2040  ..  ..  ... $202B      jelzö: C   értéke: 1
c., .....  ..  AA  ... $20F0      jelzö: N   értéke; 0
```

11.Szorozzuk össe a 12-t 8-al, a szorzatot a $3000 címre tároljuk le! A szorzást ismételt összeadással valósítsuk meg!

12.Osszuk el a 49-t 15-el! A hányados egész részét az XR-ben a maradékot az AC-ban tároljuk! Az osztást ismételt kivonással relizáljuk az assembly programunkban!

13.Mi az alábbi assembly programok funkciója?

```
a.,  $2000  LDX #$00          b.,  $2000  LDX #$FF
     $2002  LDA $00,X               $2002  LDA $00,X
     $2004  AND #$40                $2004  AND #$20
     $2006  BEQ $2020               $2006  BNE $2020
     $2008  INX                     $2008  DEX
     $2009  BNE $2002               $2009  BNE $2002
     $200B  JMP $2023               $200B  BRK
     .....                          .....
     $2020  STX $3000               $2020  STX $3000
     $2023  BRK                     $2023  JMP $200B
```

14.Add meg az alábbi BASIC programrészletnek megfelelö assembly programocskát!

```
2000 FOR Y= 255 TO 0 STEP -1
2010   IF PEEK(Y)=100 THEN AC=255:END
2020 NEXT Y
2030 AC=0
2040 END
```

15. Milyen müveletet végeznek az AC tartalmán az alábbi assembly programok?

a.,
```
$3000 LDX #$08
$3002 ASL
$3003 ROR $D0
$3005 DEX
$3006 BNE $3002
$3008 LDA $D0
$300A BRK
```

b.,
```
$3000  LDX #$00
$3002  LDY #$08
$3004  ROL
$3005  BCC $3008
$3007  INX
$3008  DEY
$3009  BNE $3004
$300B  BRK
```

16. írassuk ki a képernyöre a számítógép hibaüzeneteit! Ez úgy történik, hogy az XR-be a soron következö hibakódot helyezve meghívjuk a $8683-n kezdödö szubrutint.

17. A ROM $818E címétöl kezdve helyezkednek el a BASIC kulcsszavak. írassuk ki ezeket a képernyöre a $FFD2-töl kezdödö alprogram segítségével! Ez a szubrutin azt az ASCII jelet viszi ki a képernyö aktuális helyére, amelynek a kódja az AC-ban van a rutin hívásakor (ld. KERNAL RUTINOK). A kulcsszavak kiírásának az jelenti a végét, hogy a tárterületröl beolvasott érték $00 lesz.

18. A $FFD2 címen kezdödö szubrutin segítségével írjuk ki a képernyöre a számítógépünk grafikus jeleit!

19. A képernyö utolsó elötti sorában egy díszítösort szeretnénk elhelyezni, amelyik a felváltva szereplö '*' és a '-' karakterekböl áll. A képernyö bal felsö karakterhelyének címe $0C00.

20. A képernyö tabulátormezöinek elsö karakterhelyeire az 'A', az utolsó helyeire a 'Z' karaktereket írassuk ki!

KERNAL RUTINOK, BASIC RUTINOK HASZNALATA

1. Adjuk meg az alábbi BASIC programoknak megfelelö assembly megoldásokat. Használjuk a ROM KERNAL rutinjai közül a megfelelöeket!

```
100 REM a. feladat
110 SCNCLR
120 INPUT A$
130 SCNCLR
140 A$=LEFT$(A$,1)
150 FOR X=0 TO 39 STEP 2
160     PRINT TAB(X);A$
170 NEXT X
180 END
```

```
100 REM b. feladat
110 SCNCLR
120 INPUT A$
130 OPEN 6,4,7
140 PRINT#6,A$
150 CLOSE 6
160 END
```

Mi az egyes programok funkciója?

2. A közölt mintaprogramoknak megfelelö BASIC programokat írd meg, csupán az assembly listák alapján!

a.,program

```
$2000 LDA #$93
$2002 JSR $FFD2
$2005 LDY #$00
$2007 JSR $FFCF
$200A STA $3000,Y
$200D INY
$200E CMP #$0D
$2010 BNE $2007
$2012 STY $2F00
$2015 LDA #$93
$2017 JSR $FFD2
$201A LDX #$00
$201C LDY $2F00
$201F DEY
$2020 LDA $3000,Y
$2023 JSR $FFD2
$2026 DEY
$2027 INX
$2028 CPX $2F00
$202B BNE $2020
$202D RTS
```

b.,program

```
$2000 LDA #$93
$2002 JSR $FFD2
$2005 LDY #$00
$2007 JSR $FFCF
$200A STA $3000,Y
$200D INY
$200E CMP #$0D
$2010 BNE $2007
$2012 STY $2F00
$2015 LDA #$06
$2017 LDX #$04
$2019 LDY #$07
$201B JSR $FFBA
$201E LDA #$00
$2020 JSR $FFBD
$2023 JSR $FFC0
$2026 LDX #$06
$2028 JSR $FFC9
$202B LDY #$00
$202D LDA $3000,Y
$2030 JSR $FFD2
$2033 INY
$2034 CPY $2F00
$2037 BNE $202D
$2039 JSR $FFCC
$203B LDA #$06
$203E JSR $FFC3
$2041 BRK
```

3. Az alábbi programot az ASS-16 assembler rendszerprogrammal fejlesztettük ki. Add meg a program funkcióját! Írd meg azt a BASIC programot, amelyik ugyanezt a feladatot oldja meg! A hely szükössége miatt megosztottuk a lapot.

```
100 SYS 12224                 :    250 JSR SETNAM
110 .OPT P,OO                 :    260 JSR OPEN
120 SETLFS =$FFBA             :    270 LDX #15
130 CLOSE  =$FFC3             :    280 JSR CHKOUT
140 SETNAM =$FFBD             :    290 LDA #"I"
150 OPEN   =$FFC0             :    300 JSR CHROUT
160 CHKOUT =$FFC9             :    310 JSR CLRCHN
170 CLRCHN =$FFCC             :    320 LDA #15
180 CHROUT =$FFD2             :    330 JSR CLOSE
190        *=$2000            :    340 RTS
200 LDA #15                   :    350 .END
210 LDX #8                    :
220 LDY #15                   :
230 JSR SETLFS                :
240 LDA #0                    :
```

4. Oldd meg assembly nyelven is az alábbi BASIC programot!

```
100 INPUT A$
110 OPEN 3,1,1,"ADAT91"
120 PRINT#3,A$
130 CLOSE 3
140 END
```

5. Egy, a billentyüzetröl beadott számmal osszuk el a négyzetgyök kettöt! Használjuk a FAC-ot!

6. Egy, a billentyüzetröl beadott számhoz adjuk hozzá a π konstanst!

7. Számítsuk ki egy egész szám ötödik hatványát!

8. Határozzuk meg egy természetes szám természetes kitevöjü hatványának értékét!

9. Állítsunk elö 20 darab véletlen egész számot a [0;9] intervallumban. Az alábbi aritmetikai rutinokat célszerüen használhatjuk a feladat mogoldása során:

$A70E	FAC=RND(0)	$A162	FAC=10*FAC
$A358	FAC=INT(FAC)	$A469	FAC --> ASCII

10. Elemezzük az alábbi programot! A pontozott vonalakra írj!

```
100  SYS 12224
110  .OPT P,OO
120  *=$2000
130  LDY #1          ;CIKLUS VALTOZó KEZDöéRTéKéNEK BEALLíTASA
140  IDE STY $E0     ;.....
150  JSR $9A81       ;FAC <-- Y ARITMETIKAI RUTIN
160  LDA #$39        ; π ALSó CíME
170  LDY #$94        ;.....
180  JSR $A072       ;.........ARITMETIKAI RUTIN
190  JSR $AA77       ;.........ARITMETIKAI RUTIN
200  LDY #1          ;ELöKéSZíTéS
210  JSR $A469       ;.........ARITMETIKAI RUTIN
220  LDY $E0         ;.....
230  INY             ;.....
240  CPY #11         ;.....
250  BNE IDE         ;.....
260  RTS
270  .END
```

11. Az alábbi program a soros busz programozására mutat példát. A program elemzésekor állapítsd meg az I/O eszközt, az I/O müvelet irányát és tárgyát! Most is osztott irásmódot alkalmazunk.

```
100 SYS 12224                 :  230 LDY #0
110 .OPT P,OO                 :  240 CIKLUS LDA SZOVEG,Y
120 LISTEN = $FFB1            :  250 CMP #48
130 SECOND = $FF93            :  260 BEQ VEGE
140 CIOUT  = $FFD2            :  270 JSR CIOUT
150 UNLS   = $FFAE            :  280 INY
160        *= $2000           :  290 JMP CIKLUS
170 LDA #0                    :  300 VEGE LDA #13
180 STA $90                   :  310 JSR CIOUT
190 LDA #4                    :  320 JSR UNLSN
200 JSR LISTEN                :  330 RTS
210 LDA #$67                  :  340 SZOVEG .ASC "MARIKA0"
220 JSR SECOND
```

írd meg a programot BASIC nyelven is!

XI. FÜGGELÉK

1. ALFANUMERIKUS KARAKTEREK KÉPERNYÖ ÉS ASCII KÓDJAI

Terjedelmi korlátok miatt csak az un. alfanumerikus karakterek kódjait tartalmazza a táblázatunk. Emlékeztetöül: a képernyöhöz rendelt memória kezdöcíme $0C00, az utolsó bájt címe pedig $0FE7.
Ha egy karakterhely pozíciói [sor, oszlop], ahol 0 <= sor <= 24 és 0 <= oszlop <= 39, akkor a 3072+40*sor+oszlop képlettel határozhatjuk meg az illetö helyhez tartozó címet.

JEL	KÉPK.	ASC	JEL	KÉPK.	ASC	JEL	KÉPK.	ASC
@	$00	$40	T	$14	$54	+	$2B	$2B
A	$01	$41	U	$15	$55	,	$2C	$2C
B	$02	$42	V	$16	$56	-	$2D	$2D
C	$03	$43	W	$17	$57	.	$2E	$2E
D	$04	$44	X	$18	$58	/	$2F	$2F
E	$05	$45	Y	$19	$59	0	$30	$30
F	$06	$46	Z	$1A	$5A	1	$31	$31
G	$07	$47	[	$1B	$5B	2	$32	$32
H	$08	$48	]	$1D	$5D	3	$33	$33
I	$09	$49	SPC	$20	$20	4	$34	$34
J	$0A	$4A	!	$21	$21	5	$35	$35
K	$0B	$4B	"	$22	$22	6	$36	$36
L	$0C	$4C	#	$23	$23	7	$37	$37
M	$0D	$4D	$	$24	$24	8	$38	$38
N	$0E	$4E	%	$25	$25	9	$39	$39
O	$0F	$4F	&	$26	$26	:	$3A	$3A
P	$10	$50	'	$27	$27	;	$3B	$3B
Q	$11	$51	(	$28	$28	<	$3C	$3C
R	$12	$52	)	$29	$29	=	$3D	$3D
S	$13	$53	*	$2A	$2A	>	$3E	$3E

2. SZÍNKÓDOK

Lehetöségünk van az egyes karakterek színeit megválasztani. Minden karakterhelynek a színmemóriában egy-egy bájt felel meg, amelynek az értéke megadja a karakter színét, a villogást és az inverz megjelenítést is. Az összefüggö színmemória kezdöcíme $0800, az utolsó használt bájt címe $0BE7. A színmemória egy bájtjának jelentése:

7.bit: 1 esetén villogás bekapcsolva,
0 esetén villogás kikapcsolva
4.-6.bit: a fényerö értéke (0-7)
0.-3.bit: a szín kódja (0-15) az alábbi táblázat szerint:

0:fekete	1:fehér	2:vörös	3:encián
4:bíbor	5:zöld	6:kék	7:sárga
8:narancs	9:barna	A:sárgászöld	B:rózsaszín
C:kékeszöld	D:világoskék	E:sötétkék	F:világoszöld

3. UTASíTASOK GéPI KóDJA

MNE.	#ee	cc	cc,X	cc,Y	cccc	cccc,X	cccc,Y	(cc,X)	(cc),Y	imp	(cccc)
ADC	69	65	75		6D	7D	79	61	71		
AND	29	25	35		2D	3D	39	21	31		
ASL		06	16		0E	1E				0A	
BCC										90	
BCS										B0	
BEQ										F0	
BIT		24			2C						
BMI										30	
BNE										D0	
BPL										10	
BRK										00	
BVC										50	
BVS										70	
CLC										18	
CLD										D8	
CLI										58	
CLV										B8	
CMP	C9	C5	D5		CD	DD	D9	C1	D1		
CPX	E0	E4			EC						
CPY	C0	C4			CC						
DEC		C6	D6		CE	DE					
DEX										CA	
DEY										88	
EOR	49	45	55		4D	5D	59	41	51		
INC		E6	F6		EE	FE					
INX										E8	
INY										C8	
JMP					4C						6C
JSR					20						
LDA	A9	A5	B5		AD	BD	B9	A1	B1		
LDX	A2	A6		B6	AE		BE				
LDY	A0	A4	B4		AC	BC					
LSR		46	56		4E	5E				4A	
NOP										EA	
ORA	09	05	15		0D	1D	19	01	11		
PHA										48	
PHP										08	
PLA										68	
PLP										28	
ROL		26	36		2E	3E				2A	
ROR		66	76		6E	7E				6A	
RTI										40	
RTS										60	
SBC	E9	E5	F5		ED	FD	F9	E1	F1		
SEC										38	
SED										F8	
SEI										78	
STA		85	95		8D	9D	99	81	91		
STX		86		96	8E						
STY		84	94		8C						
TAX										AA	
TAY										A8	
TSX										BA	
TXA										8A	
TXS										9A	
TYA										98	

4. AZ UTASíTASOK HATASA AZ ALLAPOTREGISZTER BITJEIRE

JELMAGYARAZAT: 1 a kérdéses flag értéke 1 lesz

0 a kérdéses flag értéke 0 lesz

+ a kérdéses flag állítódik, értékét a müvelet eredménye határozza meg

. az utasítás hatására a kérdéses flag értéke nem változik

MNE.	N	V	1	B	D	I	Z	C	MNE.	N	V	1	B	D	I	Z	C
ADC	+	+	.	.	.	.	+	+	AND	+	.	.	.	.	.	+	.
ASL	+	.	.	.	.	.	+	+	BCC	.	.	.	.	.	.	.	.
BCS	.	.	.	.	.	.	.	.	BEQ	.	.	.	.	.	.	.	.
BIT	+	+	.	.	.	.	+	.	BMI	.	.	.	.	.	.	.	.
BNE	.	.	.	.	.	.	.	.	BPL	.	.	.	.	.	.	.	.
BRK	.	.	.	+	.	1	.	.	BVC	.	.	.	.	.	.	.	.
BVS	.	.	.	.	.	.	.	.	CLC	.	.	.	.	.	.	.	0
CLD	.	.	.	.	0	.	.	.	CLI	.	.	.	.	.	0	.	.
CLV	.	0	.	.	.	.	.	.	CMP	+	.	.	.	.	.	+	+
CPX	+	.	.	.	.	.	+	+	CPY	+	.	.	.	.	.	+	+
DEC	+	.	.	.	.	.	+	.	DEX	+	.	.	.	.	.	+	.
DEY	+	.	.	.	.	.	+	.	EOR	+	.	.	.	.	.	+	.
INC	+	.	.	.	.	.	+	.	INX	+	.	.	.	.	.	+	.
INY	+	.	.	.	.	.	+	.	JMP	.	.	.	.	.	.	.	.
JSR	.	.	.	.	.	.	.	.	LDA	+	.	.	.	.	.	+	.
LDX	+	.	.	.	.	.	+	.	LDY	+	.	.	.	.	.	+	.
LSR	+	.	.	.	.	.	+	+	NOP	.	.	.	.	.	.	.	.
ORA	+	.	.	.	.	.	+	.	PHA	.	.	.	.	.	.	.	.
PHP	.	.	.	.	.	.	.	.	PLA	+	.	.	.	.	.	+	.
PLP	+	+	+	+	+	+	+	+	ROL	+	.	.	.	.	.	+	+
ROR	+	.	.	.	.	.	+	+	RTI	+	+	+	+	+	+	+	+
RTS	.	.	.	.	.	.	.	.	SBC	+	+	.	.	.	.	+	+
SEC	.	.	.	.	.	.	.	1	SED	.	.	.	.	1	.	.	.
SEI	.	.	.	.	.	1	.	.	STA	.	.	.	.	.	.	.	.
STX	.	.	.	.	.	.	.	.	STY	.	.	.	.	.	.	.	.
TAX	+	.	.	.	.	.	+	.	TAY	+	.	.	.	.	.	+	.
TSX	+	.	.	.	.	.	+	.	TXA	+	.	.	.	.	.	+	.
TXS	.	.	.	.	.	.	.	.	TYA	+	.	.	.	.	.	+	.

5. FONTOSABB MEMÓRIACÍMEK

A NULLAS LAP RENDSZERVÁLTOZÓI

002B - 002C	BASIC program kezdetének címe
002D - 002E	BASIC változók kezdetének címe
002F - 0030	BASIC tömbök kezdetének címe
0033 - 0034	füzérek kezdetének címe
0037 - 0038	BASIC RAM felsö határa
0061 - 0066	a FAC területe
0069 - 006E	az ARG területe
006F	ARG-FAC elöjelegyezés
0090	az ST rendszerváltozó
0098	a beviteli készülék száma
0099	a kiviteli készülék
009D - 009E	a program vége mutató
00A3 - 00A5	az óra
00D0 - 00E8	szabad terület a programozóknak

EGYÉB CÍMEK

0100 - 01FF	rendszer verem területe
0200 - 0258	input puffer
02E4	karaktergenerátor címének felsö bájtja
0300 - 0331	BASIC és KERNAL ugró vektorok
0332 - 03F2	kazettapuffer
0479	CHRGOT rutin belépési címe
0527 - 0530	billentyüzet puffer
0552 - 0558	CPU REGISZTEREK (PC, SR, AC, XR, YR, SP)
05F5 - 06EB	szabad terület, ha nincs bövitö program
07F8	MONITOR RAM/ROM kapcsoló
07FE - 07FF	nem használt terület
0800 - 0BFF	szín memória
0C00 - 0FE7	képernyö memória
1000 - FCFF	BASIC RAM grafika meghívása nélkül

KERNAL RUTINUK BELÉPÉSI PONTJAI

FF81	CINT	FF84	IOINIT	FF87	RAMTAS
FF8A	RESTOR	FF8D	VECTOR	FF90	SETMSG
FF93	SECOND	FF96	TALK	FF99	MEMTOP
FF9C	MEMBOT	FF9F	SCNKEY	FFA2	SETTMO
FFA5	ACPTR	FFA8	CIOUT	FFAB	UNTLK
FFAE	UNLSN	FFB1	LISTEN	FFB4	TALK
FFB7	READST	FFBA	SETLFS	FFBD	SETNAM
FFC0	OPEN	FFC3	CLOSE	FFC6	CHKIN
FFC9	CHKOUT	FFCC	CLRCHN	FFCF	CHRIN
FFD2	CHROUT	FFD5	LOAD	FFD8	SAVE
FFDB	SETTIM	FFDE	RDTIM	FFE1	STOP
FFE4	GETIN	FFE7	CLALL	FFEA	UDTIM
FFED	SCREEN	FFF0	PLOT	FFF3	IOBASE

6. ARITMETIKAI RUTINOK BELÉPÉSI CÍMEI

ADATATVITEL

A281	FAC=ARG
A291	ARG=FAC
A0DC	ARG=(YA) ROM
A107	ARG=(YA) RAM
A221	FAC=(YA) ROM
A21F	FAC=(YA) RAM
A259	(YX)=FAC RAM
9A81	Y a FAC-ba
9DE4	FAC AY-ba
9A76	AY a FAC-ba
A37F	ASCII a FAC-ba
A469	FAC ASCII-be

FÜGGVÉNYEK

A2BE	FAC=SIN(FAC)
A2DD	FAC=ABS(FAC)
A358	FAC=INT(FAC)
A5E4	FAC=SQR(FAC)
AA70	FAC=COS(FAC)
AA77	FAC=SIN(FAC)
AAC0	FAC=TAN(FAC)
AB1A	FAC=ATN(FAC)
A0E1	FAC=LOG(FAC)
A70E	FAC=RND(0)
A660	FAC=EXP(FAC)
A01E	FAC=LOG(FAC)

MÜVELETEK

9E9E	FAC=ARG+FAC
9E9B	FAC=(YA)+FAC (RAM)
9E87	FAC=ARG-FAC
A06C	FAC=(YA)-FAC (ROM)
A627	FAC=-FAC
A07B	FAC=ARG*FAC
A078	FAC=(YA)/FAC (RAM)
A162	FAC=10*FAC
A072	FAC=(YA)/FAC (ROM)
A194	FAC=(YA)/FAC (RAM)
A183	FAC=FAC/10
A5EE	FAC=ARG emelve a FAC-ra
95F8	FAC=FAC OR ARG
95FB	FAC=FAC AND ARG
9469	FAC=NOT FAC
A2E0	FAC és (YA) hasonlítása

ÁLLANDÓK

9FF0	1
A179	10
98C6	32768
A44E	100.000.000
A5A3	1/2
A014	-1/2
AAF6	1/4
9439	π
AAE0	2*π
AAE9	π/2
A00F	SQR(2)
A00A	1/SQR(2)
A019	LN(2)
E3A2	C1984COMMODORE
CD32	ARE YOU SURE?

BASIC UTASÍTASOK BELÉPÉSI PONTJAI

8CDA	END	ADCA	FOR	9294	NEXT	8DB0	DATA
90EE	INPUT#	9108	INPUT	969B	DIM	914F	READ
8E7C	LET	8D4D	GOTO	8BBC	RUN	8DE1	IF
8C9A	RESTORE	8D2C	GOSUB	8D83	RETURN	8E1B	ON
A7F3	LOAD	A7DE	SAVE	A7F0	VERIFY	9E12	POKE
8FE0	PRINT#	9000	PRINT	8A98	CLR	A84D	OPEN
A85A	CLOSE	90B8	GET	8E0B	ELSE	B849	SOUND
B8BD	VOL	C5C3	GRAPHIC	B8D1	PAINT	B9D4	CHAR
BAE2	BOX	C01E	CIRCLE	BD35	GSHAPE	BE29	SSHAPE
C4D9	DRAW	C50F	LOCATE	C51A	COLOR	C567	SCNCLR
B557	DO	B603	LOOP	AE5A	DELETE	B729	KEY
B42B	TRAP	B652	TRON	B655	TROFF	C8BC	DIRECTORY
FF52	MONITOR	8A79	NEW	8E0B	REM	9A9D	DEF
A7B5	SYS	B5AC	EXIT	C5B8	SCALE	8AFF	LIST

XII. IRODALOMAJANLAT

1. Czigler Zoltán : A Commodore 64 programozásának gyakorlata 4. kötet
Gépi kódú programozás
(SZAMALK, Bp., 1989.)

2. Babán Gábor-
Masa István : Gépi kódú programozás kezdöknek és haladóknak C 16 és Plus/4 számítógépre
(Novotrade Rt., Bp., 1988.)

3. Gáspár Dénes-
Gyenes Tamás : A Plus 4 belsö felépítése
(Novotrade Rt., Bp., 1988.)

4. Erdös István : Commodore Plus/4, C-16, C-116 ROM lista
(LSI, Bp., 1986.)

5. Vadnai Szabolcs : Commodore Plus/4 programozói zsebkönyv
(Novotrade Rt., Bp., 1986.)

6. W. BESENTHAL-
J. Muus : Plus/4 kézikönyv az összes tudnivalóval
(Novotrade Rt., Bp., 1989.)

7. Obádovics J. Gyula : Számítástechnika C 64
(Novotrade Rt., Bp., 1989.)

8. Varga Antal : C 16, Plus/4 programozói útmutató
(Novotrade Rt., Bp., 1987.)

9. Farkas Zoltán
Bálint Balázs : Commodore 64 file-kezelés és input-output
(LSI, Bp., 1988.)