HT 1080 Z/64
HT 2080 Z/64
KEZELÉSI ÚTMUTATÓ

HÍRADÁSTECHNIKA SZÖVETKEZET

HIRADÁSTECHNIKA SZÖVETKEZET
H-1519 BUDAPEST PF.268
TELEX: 226151 HTSZH

ISBN: 963 592 519 0

KÉSZÜLT: AZ LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT
GONDOZÁSÁBAN

TARTALOMJEGYZÉK

ELŐSZÓ

Ez a gépkönyv azok számára készült, akik most ismerkednek meg a számítógépek BASIC nyelvű programozásával, illetve a HT-1080Z/64 géppel. Hasznos ismereteket szerezhet az is, aki már dolgozott más számítógépekkel.

Igyekeztünk a gépkönyvet didaktikusan felépíteni, hogy a kezdő programozó hamar sikerélményekhez jusson. Ezt segítik a szép számmal elhelyezett példák is. Nagyobb mintapéldáink a géphez mellékelt demonstrációs kazettán is megtalálhatók, így azok begépelés nélkül, annak betöltésével kipróbálhatók.

A gépkönyv felépítéséről:

Négy részben foglaltuk össze a tudnivalókat a HT-1080Z/64-ről és programozásáról. Az 1. rész a gép használatának módját ismerteti a kicsomagolástól a programozás legelemibb lépéseiig. A 2. rész fejezetei szólnak részletesen magáról a BASIC-nyelvről, melyeket igyekeztünk olvasmányosan (könnyen emészthetően), ugyanakkor precizen megfogalmazni. Az ezt követő részben elsősorban a gyakorlott programozóknak szántunk hasznos tudnivalókat. (E rész keretében esik szó pl.: a memória és a képernyő felosztásáról, a karakterkészletről és a hibajelzésekről.)

Végül a függelékben kapott helyet egy - reményünk szerint - jól használható "szótár", a számítástechnikai szókincsben nagyon gyakran előforduló fogalmakról.

A gépkönyv könnyebb használatát szerettük volna előmozdítani azzal, hogy a szövegben bizonyos nyomdatechnikai (kiemelési) konvenciókat alkalmaztunk. Így a szövegből jól kiugróan, vastag nagy betűkkel találhatók a KULCSSZAVAK, az utasítás vagy fogalom DEFINÍCIÓK, a fontos magyarázó részeket ritkán szedtük. Az egyértelműséget hivatott szolgálni a billentyűk sajátos megkülönböztetése a kulcsszavaktól: vastag, aláhúzott nagybetűk jelentik a BILLENTYŰKET.

Az előszó lezárásaként - a számítástechnikai körökben közismert - alapigazsággal kívánjuk biztatni a HT-1080Z/64 minden kedves használóját: "Az öntevékeny kísérletezgetést nem pótolhatja semmilyen gépkönyv sem!"

1. A HT-1080Z/64 használata

1.1. ÖSSZEÁLLÍTÁS ÉS BEKAPCSOLÁS

Műszaki adatok

Központi egység : Z80 CPU

Órafrekvencia : 1,78 MHz

Tároló - RAM : 64 kbyte dinamikus
- ROM : 12 kbyte EPROM (BASIC interpreter)
2 kbyte EPROM (gépi kódú monitor)

Megjelenítés - alfanumerikus : 16 sor x 64 karakter
Karakterkészlet : magyar ékezetes betűkkel és speciális szimbólumokkal kiegészített ASCII.
Karakterkép : 5 x 7-es pontmátrix
- szemigrafikus : 48 sor x 128 pont

Megjelenítő kimenet : - video 2V p-p (negativ szinkronjel) 75 OHM-os lezáráson
- VHF (OIRT 2. csatorna) 200 mV p-p 75 OHM-os lezáráson

Háttértároló : - beépített kompakt kazettás adatrögzítő
szalagsebesség : 4,75 cm/s
- külső magnócsatlakozás : szükséges bemenőszint 1 V p-p, kimenő szint 300 mV p-p,távkapcsoló terhelhetősége: 0.5 A max 6 V DC-nél

Beépített hanggenerátor : 3 csatornás, tisztahang + zaj

Csatlakozási lehetőségek : - Z80 bus
- 2 x 8 programozható be - kimenet
- hanggenerátor kimenet

Hálózati tápfeszültségigény : 220 V ± 10%, 50 Hz

Teljesítményfelvétel : 25 W max.

Környezeti adatok : üzemi hőmérséklet 0...+ 45 Celsius fok
relatív páratartalom 80% max.
tárolási hőmérséklet -25... +45 Celsius fok
relatív páratartalom 95% max.

Érintésvédelmi osztály : II. (jelölése ⧈)
(Itt hívjuk fel a figyelmet arra, hogy a számítógéphez csatolt egyéb eszközöknek is II. érintésvédelmi osztályúaknak kell lenni!)

Kicsomagolási és üzembehelyezési utasítás

1. Távolítsuk el a csomagoló anyagokat és győződjünk meg az árban foglalt tartozékok meglétéről. Célszerű a készülék dobozát és a készülék két végére húzott müanyaghab profilt ép állapotban megőrizni, mivel ez a későbbi biztonságos tároláshoz és szállításhoz jól használható.

2. Ha a készüléket huzamosabb ideig hideg, páradús környezetben tároltuk, üzembehelyezés előtt várjuk meg, míg a készülék felveszi a használati környezet hőmérsékletét és felületén kondenzáció nem észlelhető.

3. Ezután csatlakoztassuk a monitort vagy TV készüléket, szükség esetén külső magnetofont vagy egyéb perifériát a számítógéphez. Ügyeljünk arra, hogy valamennyi rendszerelem érintésvédelmi szempontból kifogástalan legyen, továbbá a 2. pontban leírt hőmérséklet kiegyenlítődés és kondenzációmentesség ezekre is teljesüljön.

4. Csatlakoztassuk a rendszerelemeket a hálózathoz.

5. Kapcsoljuk be a megjelenítő eszközt,szükség esetén válasszuk ki a megfelelő üzemmódot (TV készüléknél TV/video átkapcsoló).

6. Kapcsoljuk be a számítógépet.
Ha a megjelenítő egység video monitor, a BASIC interpreter bejelentkező szövegének látszania kell. Ha ez nem teljesül, kisérelјük meg a monitor fényerejét növelni. Ha ez eredményre vezet, a számítógép üzemkész.
Ha a megjelenítő egység TV készülék, a számítógép bekapcsolt állapotában hangoljuk a TV készüléket az OIRT 2 (CCIR 3) csatorna közelébe. A TV készülék hangolóegységével állítsuk be a legjobb képélességet. Itt jegyezzük meg, hogy sok TV készüléknél és monitornál a fényerő és kontrasztszabályozó kezelőszervek állásától nagy mértékben függ a képcső fókusza, így a kép élessége. Ezért az optimális megjelenítés beállításához gyakran szükséges a fényerő és a kontraszt szabályozása is.

7. A 2 x 8 bites be - kimenet csatlakozón és a buszcsatlakozón a csatlakozást módositani csak a számítógép kikapcsolt állapotában szabad. Az utóbbit használaton kívül célszerű a mellékelt fedőlappal lefedni.

8. A számítógépet kikapcsolás után ismételten bekapcsolni csak 15 másodperc elteltével szabad. Ha ezt figyelmen kívül hagyjuk, a számítógép olyan állapotba kerül, amely csak kikapcsolással, majd legalább 15 másodperc múlva való ismételt bekapcsolással szüntethető meg.

ÖSSZEÁLLÍTÁS ÉS BEKAPCSOLÁS

Az árban foglalt tartozékok

1 db 1 m hosszúságú, külső magnetofonhoz csatlakozó kábel, mindkét végén 5 pólusú DIN csatlakozóval szerelve.

1 db 3 m hosszúságú koaxiális kábel, egyik végén Hirschmann TOST 10-es, a másik végén Hirschmann KOST 1-es csatlakozóval szerelve.

1 db Demonstrációs kazetta

2 db Hálózati biztositék (tartalék) 500 mA

1 db 20 pólusú csatlakozó hüvely szerelvénnyel

1 db Kezelési útmutató

1.2. BILLENTYŰZET

A HT-1080Z/64-en az összes magyar ékezetes betű használatára lehetőség van, a kis és a nagybetűkre is. A nagy és a kisbetűket a számítógép csak kiterjesztett üzemmódban képes különbözőképpen megjeleníteni, illetve az ékezetes betűk is csak így jelenhetnek meg ékezetesként (ld. 2.19., 3.4. Fejezet). Kiterjesztett üzemmódban továbbá lehetőség van arra, hogy egy billentyűt tartósan lenyomva tartva, a számítógép a lenyomás idejéig ismételje a karaktert.

A HT-1080Z/64-en található néhány speciális billentyű, ezek funkciója:

NEW LINE	-	Minden beírt sor végét ez a billentyű jelzi.
SHIFT	-	Lenyomásával egyidejűleg nyomott billentyűk "felső" karaktere lesz érvényes, illetve nagybetű helyett kisbetű kerül be. A kisbetűre váltás akkor is megtörténik, ha a képernyőn a betű nem kisbetűnek látszik.
BREAK	-	Futó BASIC program működését megszakitja, listázás közben pedig leállítja a listázást.
CLEAR	-	Letörli a képernyőt és a kurzort a bal felső sarokba állítja.
←	-	Az utoljára beírt karaktert törli.
→	-	A kurzort a következő 8-cal osztható sorszámú oszlopra állitja.
↓	-	A kurzort a következő sor elejére állítja.
SHIFT+ ←	-	A teljes beírt sor törlése.
SHIFT+ →	-	Lenyomása után a képernyőn minden kiírt karaktert a következőtől szóközzel választ el (ritkítás). Képernyőtörléssel vagy NEW paranccsal lehet hatástalanítani.
SHIFT+ @	-	A futás vagy a listázás megszakad, s tetszőleges billentyű lenyomására folytatódik, illetve sor beírásakor az ékezetes betűk és a speciális jelek beírásához szükséges.

BILLENTYŰZET

A HT-1080Z/64 hátlapján található billentyűk:

RESET - Megszakítja az éppen folyó tevékenységet, parancs üzemmódba tér át.

VIDEO CUT - Benyomott állapotában minden sorból csak 32 karakter látszik, egyébként pedig 64.

Megjegyzés: A 3.4. fejezetben részletes ismertetőt adunk a képernyő szerkezetéről, használatáról.

A hagyományos billentyűk fölött elhelyezkedő billentyűk:

PAGE - Ha a **VIDEO CUT** billentyű be van nyomva, akkor kiengedett állapotában a sorok első 32 karaktere, benyomott állapotában pedig az utolsó 32 karaktere látszik.

F1 - Benyomott állapotában a kazettás magnó motorját a magnó kezelőszerveivel közvetlenül, egyébként pedig csak a gép vezérlésével lehet indítani vagy leállítani.

PROGRAMBETÖLTÉS KAZETTÁRÓL - FUTTATÁS

1.3 PROGRAMBETÖLTÉS KAZETTÁRÓL - FUTTATÁS

Program betöltéséhez először meg kell ismerkedni a beépített kazettás magnó kezelésével. Ha a kazettás magnót kézzel kell vezérleni, akkor le kell nyomni az **F1** billentyűt, s ezután használhatók a kazettás magnó vezérlő billentyűi.

REW **RECORD** **STOP-EJECT** **PLAY** **FF**

Az egyes billentyűk funkciói:

REW Gyors hátratekerés.

RECORD Lenyomása esetén töröl a magnó a kazettáról.

STOP-EJECT Lenyomására az előzőleg kiválasztott funkció véget ér, majd újabb lenyomására kivehető a kazetta a magnóból.

PLAY Lenyomására a magnó normál sebességgel elindul, ha a **RECORD** billentyű le van nyomva, akkor törli a szalagot, ha nincs, akkor csak végigolvassa.

FF Gyors előretekerés.

Ha az **F1** billentyűt felengedjük, akkor a motor vezérlését átveszi a számítógép.

Kész programokat b e t ö l t h e t ü n k kazettáról a következő paranccsal:

CLOAD

A parancs beírása után le kell nyomni a **NEW LINE** billentyűt!

Ennek hatására a **HT-1080Z/64** elindítja a beépített kazettás magnót, s keresi a kazettán az aktuális hely után kezdődő programot. Ha megtalálta, akkor megjelenik a képernyő jobb felső sarkában k é t c s i l l a g, amelyből a második olvasás közben v i l l o g. A betöltés befejeztével ujra megjelenik a sor elejét mutató >- jel. Ha a magnó túlhaladt a program elején, s a két csillag nem jelent meg, akkor a **RESET** gomb benyomásával megállítható a megkezdett tevékenység, s a kazetta visszatekerése után ujra kezdhető. Gyakran előfordul az a hiba, hogy a beolvasandó program előtt van még valami a kazettán, s a beolvasást úgy kezdjük, hogy a magnó még ebből is érzékel néhány jelet,s ettől "megzavarodik" (ez a két csillag kimerevedésével érzékelhető). Ilyenkor a pontos beállítást javasoljuk, illetve saját program felvétele előtt mindig célszerű némi ü r e s (letörölt) h e l y e t hagyni a program előtt.

Ha a programot sikeresen betöltöttük, akkor működésre bírni a következő paranccsal lehet:

RUN

Természetesen ez után is le kell nyomni a NEW LINE billentyűt. Hatására a program működni kezd, s az esetleges kérdéseire kell válaszolni. A válasz beírásának végét szintén a NEW LINE billentyű jelzi.

1.4 PARANCSOK ÉS UTASÍTÁSOK

A HT-1080Z/64 kétféle üzemmódban használható.Az egyikben a beírt karaktersorozatot azonnal megpróbálja végrehajtani, a másikban pedig megjegyzi és csak külön felszólításra hajtja végre. Az első üzemmódot közvetlen üzemmódnak nevezzük, a másodikat pedig programozott üzemmódnak.

A HT-1080Z/64 műveletei is két csoportra oszthatók, az egyik csoport magával a programmal végez műveletet, a masik pedig a feldolgozandó adatokkal. Az előbbieket parancsoknak, az utóbbiakat pedig utasításoknak nevezzük. A parancsokat legtöbbször közvetlen üzemmódban, az utasításokat pedig programozott üzemmódban használjuk. Lehetséges azonban szinte mindegyik utasítást közvetlen üzemmódban és szinte minden parancsot programozott üzemmódban használni. (Kivétel az INPUT utasítás és a CONT parancs.)

Közvetlen üzemmódban a beírt sort a NEW LINE billentyű lenyomására a számítógép azonnal végrehajtja. Program írásakor minden sornak egy sorszámot kell adni, amely 1 és 65529 közötti egész szám lehet. A programot ezen sorszámok szerint növekvő sorrendben hajtja végre a számítógép, hacsak nem mondunk mást (ld. például Vezérlésátadás). Természetesen az egyes programsorok végét is a NEW LINE billentyű lenyomása jelzi. Ha egy sorba több utasítást kell írni, akkor azokat kettőspontal (:) kell egymástól elválasztani.

Fontos tudnivaló, hogy az alapszavak belsejében nem szabad szóközt hagyni, egyéb helyeken viszont tetszőleges darabszámút elhelyezhetünk, s ezzel formázhatjuk a programot.Ugyanígy a sorszám sem tartalmazhat szóközt.

2. A HT-1080Z/64 BASIC nyelv

2.1 ADATTÍPUSOK-MŰVELETEK-KIFEJEZÉSEK

A BASIC program írásakor lépten-nyomon adatokkal kapcsolatos ismereteket kell felhasználnunk, ezért különösen fontosak lesznek számunkra az e fejezetben elmondottak, meghatározóak lesznek az értékadó utasítás esetében, amelyről a 2.3. Fejezetben szólunk.

A BASIC nyelv alapvetően két adattípust ismer:

- a szám típusú (numerikus) illetve,
- a szöveg típusú (string) adatokat.

A szám típusú adat fogalma nem igényel különösebb magyarázatot, olyan adatok ezek, amelyekkel aritmetikai műveleteket (pl: összeadást, kivonást, szorzást, stb.) lehet végezni. Számon általában (valamilyen véges alakban tárolt) valós számot értünk.

A szöveg típusú adatok nem szorosan vett "matematikai" információkat tartalmaznak, hanem - mint nevük is utal rá - szöveget, például szavakat, mondatokat, általánosabban megfogalmazva: a BASIC karakterkészletébe tartozó tetszőleges jelből összeállítható jelsorozatot. A jelekhez (karakterekhez) hozzárendeltek egy 1 byte-ban tárolható számot,amelyet a karakter kódjának neveznek. E kód segítségével tárolja a gép a szövegeket. (Ld. a 3.6. Fejezetet!)

Példák:

az ALMAATA szót a következő 7 byte "testesíti meg" a gépben: 65 76 77 65 65 84 65,

Kossuth Lajos nevét a gép tárában az alábbi 13 byte jelenti: 75 111 115 115 117 116 104 32 76 97 106 111 115

A számokat leírhatjuk a szokásos fixpontos alakban pl. 135.79, vagy lebegőpontosan 1.3579E+2. Ez utóbbi a matematikában szokásos féllogaritmikus ábrázoláson alapszik (normálalak). A fenti példának az 1.3579 10↑2 (↑ hatványozás jele) felel meg.

Tehát

a valós szám fixpontos alakja:
előjel
egészrész
törtrész

lebegőpontos alak:
előjel
egészrész
törtrész
"E"
exponens előjele
exponens értéke

A törtrészbe beleértjük a tizedespontot is, amellyel kezdődik. Ezzel kapcsolatban hívjuk föl a figyelmet arra, hogy a számítástechnikában a magyar helyesírási szabályoktól eltérően nem tizedesvesszőt, hanem tizedes-

pontot használnak! Majd látni fogjuk, hogy e nyelvben több helyen (PRINT, SET stb.) is felbukkan a vessző, mint elhatároló jel, így komoly félreértések támadhatnak abból, ha nem vigyázunk a tizedespont használatára!

Az előjel elhagyható akár az egészrész, akár az exponens elejéről, ha az pozitív. Az egészrész vagy törtrész egyike elhagyható.

A bonyolultabb lebegőpontos alak alkalmazásának az az értelme - mint ahogy utaltunk már rá -, hogy a számítógépek csak bizonyos véges hosszúságú számokkal képesek számolni pl. 6 jegyre, így fixpontosan a 999999-nél nagyobb ill. a 0.000001-nél kisebb (nem 0) abszolut értékű számmal nem lehetne dolgozni, ez pedig a feladatok nagy többségénél megengedhetetlen.

Megjegyezzük, hogy akár fixpontosan, akár lebegőpontosan írjuk le a számot a szám az alábbi un. normalizált lebegőpontos alakban ábrázolódik a számítógép tárában.

Példák: leírás szerint	ábrázolás szerint
-9876	-.987600E+4
0	+.000000E+0
0.0	+.000000E+0
6.63E-34	+.663000E-33
3.14159	+.314159E+1
6.02E23	+.602000E+24

A példákat annyiban pontosítanunk kell, hogy a belső ábrázolás nem decimálisan (azaz tizes alapú számrendszerben) történik (ahogy gondolni lehetne a példa alapján), hanem binárisan (azaz kettes szamrendszerben). (A számok belső ábrázolásáról egészen pontosan a 3.5.Fejezetben lehet olvasni.) A belső számábrázolásból adódóan a legnagyobb abszolut értékű valós szám: 1.70141E+38,a legkisebb (ami nem 0) 5.87748E-39.

Általános törekvés a számítógépes nyelveknél a "normálisnál" nagyobb pontosságú számítás elősegítése, amely egyik lehetősége egy új típus bevezetése. Foglaljuk össze mit is kell tudni erről a "többlet" típusról! Megengedett, hogy az ábrázolási pontosságnál több jegyű számot is leírjunk. Ekkor a HT-1080Z/64 BASIC a következőképpen intézkedik: ha csak egy "többlet" jegye van a számnak, akkor ez alapján 6-jegyűre kerekíti. Ha további számjegyeket is tartalmaz a szám, akkor un. duplapontos számábrázolásra tér át automatikusan.

Peldák:

az 1.234567 szám helyett az egyszeres pontosságú 1.23457 számmal,
az 123.456789 helyett a duplapontos 123.4567890000000 számmal,
vagy a
3.141592653 helyett a 3.141592653000000 számmal fog számolni.

A HT-1080Z/64 BASIC duplapontos számtípusa maximum 16 számjegyet tartalmazhat. Abrázolása lényegét tekintve megegyezik az egyszeres pontosságú valósokéval, csak a mantissza tárolása nem három, hanem hét byte-on történik.

A fenti duplapontos ábrázolásra való automatikus áttérés mellett lehetőség van a pontosság közvetlen kijelölésére is. Erre a számjegy mögé

tett speciális jel hívja föl a BASIC értelmező figyelmét. Ha a számot a !-jel zárja le, akkor egyszeres pontosságot írtunk elő (esetleg annak ellenére, hogy maga a szám értékes jegyeinek száma már a nagyobb pontosságot kívánná meg). A # a duplapontosság jele. A szám duplapontosra való áttéréskor - ha szükséges - kiegészül megfelelő "hosszúságú" duplapontos számmá. A lebegőpontosan leírt számoknál e megkülönböztetést az exponens elé írt E - egyszeres pontosság esetén -, illetve D - duplapontos esetben - jelöli ki.

Példák:

987 , 987! , 9.87E3 - egyszeres pontosságúak;
654# , .3333333333333333 ,
.5# , 0.12345D-19 - duplapontosak,
de természetesen a .3333333333! egyenértékű az egyszeres .333333-mal.

A HT-1080Z/64 BASIC lehetővé tesz kifejezetten egész számokkal való számolást is, ekkor az alapvető eltérés a fentiektől ennek belső ábrázolásában van (hiányzik az exponens és a törtrész) Az új típus korlátja a kisebb ábrázolási tartomány. Ez a tartomány: -32768 és +32767 közé esik, szemben a lebegőpontos már megismert tartományával.Az egész számokat is kettes számrendszerben tároljuk, 2 byte-on helyezzük el, s ebből adódik a fenti korlátozás. Az ábrázolásból származó előny a valósokénál kisebb helyfoglalás és gyorsabb műveletvégzés. (Az adatok értéktartományára vonatkozó korlátozások okairól bővebb magyarázat a belső ábrázolásukat taglaló 3.5. Fejezetben található.)

Az egész számokat természetesen a leíráskor is meg kell tudni különböztetni a fixpontos törtrész nélküli számoktól. Ezt is egy speciális kiegészitő jellel, a % jellel tehetjük meg, amelyet az egész szám mögé kell írnunk. Ilyen értelemben az alábbi fixpontos és egész típusú számokat - bár értékük megegyezik - mégis meg kell különböztetnünk.

Példák:

valós "ábrázolású"	egész
-1	-1%
0	0%
1	1%
-32478	-32478%

Szintaktikailag hibásak a következő számleírások:
3.14% -.5% 3456.%

Megjegyzendő: a % jel a szám szerves "tartozéka",s nem tulajdonítható neki semmifajta függvényszerű funkció, így nem használható a szám egész értékének képzésére. Ezt a feladatot töltik be a később ismertetésre kerülő INT, vagy FIX függvények (ld. 2.2. Fejezetben).

Az alábbi táblázat segítségével összefoglaljuk a legfontosabb tudnivalókat a három számtípusról.

	megkülönböztető kiegészítés: fix-pontos	megkülönböztető kiegészítés: lebegő-pontos	érték-tartomány	legkisebb abszolut érték	értékes jegyek száma
egész	"%"		-32768 - +32767	---	---
egyszeres valós	"!" elmaradhat	E	+EMAX	EMIN	6
dupla-	"#"	D	+DMAX	DMIN	16

DMAX = 1.701411834604692D+38 , EMAX = 1.70141E+38
DMIN = 5.877480161902224D-39 , EMIN = 5.87748E-39

Néhány korlátozásra kell felhívni a figyelmet:
- A számolás során a valósak táblázatban szereplő "elvi" maximumát kihasználni nem lehet a t ú l c s o r d u l á s veszélye nélkül!
- A típusjelző karakter után szóközt ne írjunk,mert hibát okozhat! Egyéb helyeken tartalmazhatnak szóközt a számok.
- Sokszor jól használható a számok egy másik értelmezése, amely szerint tekinthetjük őket logikai értékként is. A -1-hez az IGAZ, és a 0-hoz a HAMIS logikai értéket rendeli a HT-1080Z/64 BASIC. Így megengedett az IF utasításban önálló használatuk is.

Ahogy a különböző típusú számokat megkülönböztettük egymástól, ugyanúgy világosan el kell különülnie a szöveg típusú (karakteres) adatoknak is. Az ilyen típusú adatokat szokták idegen szóval 'string'-eknek nevezni. E megkülönböztetés érdekében a szöveg-adatot idézőjelpárok közé ("....") zárjuk. A szöveg tartalmazhatja a 3.6. Fejezetben felsorolt karakterek bármelyikét a "-jel kivételével. Pontosabban, ha szükség van az idézőjelre a szövegen belül is, mintegy annak részeként, akkor használjuk helyette az aposztróf jelet (')!

Példák:

"a BASIC szöveg ilyen" szöveg a következő karaktersorozatot jelenti:

a BASIC szöveg ilyen

"a 'HT-1080Z/64 BASIC' ezt is szövegnek tekinti"
ennek megfelel:

a 'HT-1080Z/64 BASIC' ezt is szövegnek tekinti

"Sőt idézet 'szöveg' is lehet benne!"
ez az alábbi mondat:

Sőt idézet 'szöveg' is lehet benne!

A helytelenül leírt szövegadatok a programban igen furcsa - néha csak nehezen észrevehető - hibákat eredményezhetnek, ezért különös gondossággal járjunk velük el!

Példák hibás használatukra:

"nincs bezáró idézőjel

"rossz helyen van az " idézőjel

" rossz az "idézett" idézőjel "

Az eddig tárgyalt adatokat szokás összefoglalóan konstansoknak nevezni, utalva arra, hogy ezek a memória egy területén tárolódva a program futása során nem változnak, ellentétben az úgynevezett változókkal, amelyekre épp a változtathatóságuk miatt van szükség.

A változók

A matematikában szokásos változók mintájára, a BASIC is használ szimbolikus neveket adatok azonosítására. Ezeket a neveket hívjuk azonosítóknak. A BASIC "szabvány" szerint

az azonosító : egy betű, vagy
egy betű és egy számjegy lehet.

Az azonosítóban az ABC betűi A-tól Z-ig használhatók, akár kis, akár nagy betűk alakjában, ékezetesek azonban nem.

A fenti "definícióhoz" néhány megjegyzést kell fűznünk, ugyanis a HT-1080Z/64 BASIC e szabályt némiképpen kiterjeszti. E BASIC lehetővé teszi akármilyen hosszúságú un. alfanumerikus karaktereke ből (=betűkből és számokból) álló azonosítók használatát.Az alábbi 5 megkötésre kell ügyelnünk:

1. az első karaktere csak betű lehet.
2. csak az első kettő jelet használja fel az azonosításra!
3. az azonosítók nem tartalmazhatják a BASIC egyetlen utasításának un. kulcsszavát, (ld. a 3.2. fejezetben), a RE (RENUMBER=újra sorszámozz!) parancs kulcsszava kivételével, amelyet csak értékadások baloldalán álló azonosító elején tiltja!
4. a hossza elvileg nem haladhatja meg a 255 karaktert, gyakorlatilag a BASIC sor hosszára rótt korlátozás miatt el sem érheti.
5. az azonosítóban a kis és nagy betűk között nincs különbség, a szóközöket figyelmen kívül hagyja az értelmező.

Példák:

I , AO , Z9 - szabványos változó nevek;
BB , DALMA , DALAILAMA - a HT-1080Z/64 BASIC által megengedett azonosítók, a DALMA és DALAILAMA ugyanazt a változót azonosítják!
1K, ÖTLET, LOTTO, OTLET, IFJU hibásak, ugyanis az első nem betűvel, a második egy tiltott ékezetessel kezdődik, a továbbiak pedig tartalmaznak BASIC kulcsszót (TO, LET, IF).

Mivel a változók adatok tárolására szolgálnak, és egy változóban csak egyféle típusú adat tárolható, így a változók típusok szerinti felbontása megegyezik a fentiekben tárgyalt adatokéval. Valóban beszélhetünk egyszeres, és duplapontos valós, illetve egész típusú, valamint szöveg típusú változókról. Leírásban is tükröződnie kell e típus szerinti megkülönböztetésnek:

egész típus = azonosító %-jellel,
valós típus = azonosító !-jellel
vagy azonosító (megkülönböztető jel nélkül)
duplapontos valós típus =
azonosító #-jellel
szöveg típus = azonosító $-jellel lezárva.

A változók azonosítói általában tartalmazhatnak szóközöket is, és nem befolyásolja értelmezésüket, de nem célszerű ezzel a lehetőséggel élni.

Példák:

valós változók : I, J1, AA!, Y B L
duplapontosak : AA#, SZUMMA#
egész típusúak : K%, L1%, X9%, LEVIS%
szöveg típusúak : SZTRING$,X9$, B77$

Fontos tudni, hogy ugyanolyan azonosítójú, de más típusmegjelölést kapott változók más-más változókat jelölnek.

A típus deklaráció

A típusokkal való kényelmes munka érdekében a HT-1080Z/64 BASIC módot ad arra is, hogy kiegészítő jelek nélkül is leírhatók legyenek a változók. A FORTRAN nyelvnél szokásos módszert veszi át, azaz bevezet un. típus deklarációs utasításokat, amelyekkel bizonyos betűcsoportokhoz típusokat rendel. A megadott betűvel kezdődő változók automatikusan a deklarált típusba fognak tartozni.A típus deklarációs utasítások:

DEFINT betűtartomány(ok) = egész típusúak
DEFSNG betűtartomány(ok) = egyszeres pontosságú valósak
DEFDBL betűtartomány(ok) = duplapontosságú valósak
DEFSTR betűtartomány(ok) = szöveg típusúak

A betűtartomány megadása:
egy betű = ekkor az adott betűvel kezdődő változók a meghatározott típusba fognak tartozni.
betű-intervallum = ekkor minden a betű-intervallumba eső betűvel kezdődő változó a specifikált típusba fog tartozni.

Példák:

DEFINT A,E,I-N
DEFSTR C,F
DEFDBL T-Z,D

Megjegyzés:
1. A deklarációs utasítások használata esetén nem kell kitenni a típus jelző karaktert. Tehát, ha érvényes a fenti példában szereplő típusdeklaráció, akkor az ICIKE, JO, KARMAN, L, MICIKE, NAGYI, AA, EL változók egészek, a CHAR, CC, FUZER karakter típusú, a TATAMI, Z77, DUP, YBL duplapontos változók.
2. Ha a deklaracióban szereplő betűvel az adott típustól eltérő típusú változót akarunk használni, akkor az azonosító után tett típus kijelölő jellel felül bírálhatjuk a típus deklarációt. Például az érvényben lévő DEFINT A után az A és az A! változók két különböző (típusú) változót azonosítanak.
3. A típusdeklarációk hatását a CLEAR és a NEW utasítások megszüntetik (ld. a 2.10. és a 2.22. Fejezeteket).

Aritmetikai műveletek és kifejezések

Mivel a konstans és a változó a program futásának minden pillanatában jól meghatározott értékkel rendelkezik (az értéke típustól függően szám vagy szöveg), van értelme rajtuk műveleteket (operációkat) végezni.
A szám típusúak között a matematikában megszokott aritmetikai műveletek végezhetők: összeadás (jele:+), kivonás (-), szorzás (*), osztás (/), hatványozás (↑).
Aritmetikai kifejezések alatt a BASIC-ben is az adatok (változók, konstansok) és műveletek matematikában megszokott összekapcsolását értjük. Ennek kiértékelését teszi egyértelművé az alábbi két szabály:

- először a nagyobb prioritású (elsőbbségű) művelet hajtódik végre,
- az azonos prioritásúak között a balról elsőként előforduló értékelődik ki először. (Balról-jobbra szabály)

A prioritások sorrendje a legmagasabbal kezdve:
- hatványozás
- szorzás, osztás
- összeadás, kivonás

Megjegyzés:
A hatványozást a kitevő értéke szerint megkülönböztetve egész esetén (tehát, amelynek törtrésze 0) ismételt szorzással, törtrésszel is rendelkező kitevő esetén a e^{B*lnA} formulával számol (az A^B helyett).

Példák:

matematikai kifejezés	BASICbeli kifejezés
$\frac{1}{2} M V^2$	0.5*M*V↑2 másként 0.5*M*V*V
$\frac{M R T}{V}$	M*R*T/V
$\frac{\frac{1}{N^2}}{\frac{1}{M^2}}$	1/N 2-1/M↑2 vagy 1/N/N-1/M/M

A fenti kiértékelési sorrendtől el lehet térni zárójelezéssel,ekkor ugyanis a zárójelek közé írt kifejezés önállóan kiértékelődik a környezeteben lévő műveletek prioritásától függetlenül. A zárójelezés természetesen nagyobb mélységig egymásba skatulyázható. Ilyenkor a "legbelső" zárójelek közé tett kifejezést értékeli ki a gép leghamarabb.

$\frac{2 A}{(A+B)}$	2*A/(A+B)
$\frac{2+B}{(A-B)^2}$	(2+B)/(A-B)↑2
$\frac{B}{A+\frac{}{C}+D}$	A+B/C+D
$\frac{1}{\frac{1}{A}+\frac{1}{B}}$	1/(1/A+1/B)

Ügyelnünk kell arra, hogy több műveleti jel nem kerülhet egymás mellé, legfeljebb, ha az egyik a jobboldali a számhoz tartozó előjel.

Példák:

A+-3.14 , B*+2.7# , A↑-E , --8 megengedettek, de az általánosan elfogadott párjaik világosabbak, érthetőbbek,egyszerűbbek:
A-3.14 , B*2.7# , A↑(-E) , 8 .

A+*3.14 , B-*2.7# , A-↑E , 8↑↑2 szintaktikusan hibásak,

(-10)↑3.14 kiszámításakor következik be hiba, mert nem egész értékű exponens esetén az alap (-10) logaritmusával kellene számolni.

Az aritmetikai kifejezésekben helyet kaphatnak függvények is. Beillesztésüket ugyanazok a szabályok írják le, amelyek az eddig megismert adatokra vonatkoztak. Részletezésükre, felsorolásukra a 2.2. Fejezetben térünk ki.

Szöveg művelet és szövegkifejezés

A **HT-1080Z/64** BASIC a szövegek között is definiál egy műveletet az un. **konkatenáció** (összeláncolás, másként mondva egymásután írás) műveletét. Jele a "+". Nyilván nem keverhető össze az aritmetikai összeadással, hiszen ennek operandusai szöveg tipusúak. Ez a művelet azt a szöveget eredményezi, amely az operandusként megadott szövegek (legyen ez konstans formájában leírva, vagy változóban) egymásután írásával áll elő. Itt magától értetődően az operandusok sorrendje lényeges!

Példák:

"gömb"+"hullám" = "gömbhullám"
"hullám"+"gömb" = "hullámgömb"
tartalmazza a V$ az "Apor" vezetéknevet, a K$ a
"Péter" keresztnevet, akkor
V$+" "+K$ = "Apor Péter"

A szövegen értelmezett, illetve szöveget eredményező függvények is vannak, ezekről és a szövegmanipulációról a következő illetve a 2.10. Fejezetben lesz szó.

2.2 FÜGGVÉNYEK

A feladatok igen nagy hányadában szerepelnek matematikai függvények. Hogy ezek programunkba kényelmesen beépíthetők legyenek, jónéhányat a gyakran használtak közül a BASIC-ben meg is valósítottak. A függvényeket nevükkel lehet azonosítani. Ez többnyire a matematikából származik. A név mellett meg kell adnunk azt az értéket (argumentumot) is, amely felhasználásával a függvény értékét meg kell határoznunk.
A HT-1080Z/64 BASIC a következő standard (beépített) függvényekkel rendelkezik:

Trigonometrikus és arkusz függvények:

SIN	színusz	argumentumukat radiánban értelmezik
COS	koszinusz	
TAN	tangens	
ATN	arkusz tangens	értéke radiánban

Gyökfüggvény (=SQuare Root):

SQR	négyzetgyök	nem negatív argumentummal

Hatvány- és logaritmus függvények:

EXP	e-alapú hatvány	e=2.71828...
LOG	e-alapú logaritmus	pozitív argumentummal

Más matematikai függvények:

ABS	abszolut érték	
INT	egészrész	legnagyobb egész, amely nem nagyobb az argumentumnál
FIX	egész érték	törtrésztől megfosztott szám
SGN	előjel	ha a szám <0, akkor -1, ha =0, akkor 0, ha >0, akkor +1
RND	véletlenszám	ld. 2.12. Fejezetben

Konverziós függvények:

CINT	konverzió egész típusba
CSNG	konverzió valós típusba
CDBL	konverzió duplapontosba: értékes jegyeinek száma megegyezik az argumentumbeliével.

A programozási gyakorlatban előforduló nem matematikai jellegű függvények:

POINT	egy képernyő pont világít-e (logikai értékű) ld. 2.14. Fejezetet
INP	gépi beolvasó függvény ld. 2.15. Fejezetet
PEEK	a memória adott cimű byte-jának tartalmát eredményezi ld. 2.17. Fejezetet
FRE	az argumentum tipusától függően: - a memóriában a BASIC értelmező által szabadon hagyott byte-ok száma - szám tipusú argumentum - a szövegek feldolgozásához még rendelkezésre álló tárméret - szöveg tipus ld. 2.17. Fejezetet

MEM = FRE (szám) (nincs argumentuma) ld. 2.17. Fejezetet

POS	a kurzor pozíciója az aktuális sorban (álargumentum) ld. 2.4. Fejezetet
VARPTR	az adott azonosítójú változó címe a memóriában ld. 2.17. Fejezetet

A hibakezelés függvényei (ld. még a 2.21. Fejezetet):

ERR	az utoljára bekövetkezett hibára utaló kód
ERL	a hibát okozó utasítás sorszáma

Nagyon sok programban dolgozunk szövegekkel. Az alábbi függvények teszik lehetővé, hogy a velük való munka viszonylag kényelmes legyen. E fejezetben csak a teljesség kedvéért vesszük sorra az un. szövegfüggvényeket;ezek részletes ismertetése a 2.10. fejezetben található meg.

ASC	megadja a szöveg első jelének ASCII kódját
CHR$	adott kódú karaktert eredményez
INKEY$	a lenyomott billentyű karaktere ld. 2.4. Fejezetet
STRING$	adott kódú karakterből álló karakter-sorozatot eredményez
LEN	megadja a szöveg hosszát
LEFT$	az adott szöveg baloldali részszövege
MID$	az adott szöveg középső részszövege
RIGHT$	az adott szöveg jobboldali részszövege
STR$	a számot szöveg típusú alakban állítja elő
VAL	a karakteres formában levő szám numerikus értékét adja

A vegyes nyelvű programok írását teszi lehetővé az alábbi függvény, amely mintegy a BASIC nyelvű programból való kijáratatként,ill. a gépi kódú program bejáratatként tekinthető:

USR	a felhasználó gépi kódú programjának meghívása

A függvények leírását az alábbi formai megkötés szabályozza:

függvény-név (függvény argumentum)

Ez alól többek között az INKEY$ kivétel, amelynek nincs argumentuma (így a ( ) -k elhagyhatók). A függvény-név ne tartalmazzon szóköz karaktert, mert az megzavarja az értelmezőt! Viszont szerepelhetnek benne kisbetűk is. A szám jellegű argumentum (ahol egyáltalán szerepe van) tetszőleges szám típusú lehet, minden esetben automatikusan a megfelelő típusúra konvertálódik. Az INT az argumentum típusát megőrizve adja annak egész értékét, amíg a CINT eredménye típusa is egész lesz. Így bár matematikai "definíciójuk" ekvivalens, működésük mégis eltér:

INT(35000.66)=35000.0 ,
CINT(35000.66) túlcsordulási hibát (?OV Error) jelez.

Példák:

SQR(2*E/M)	SIN(FI)+COS(FI)
H*SQR(1-(V/C)↑2)/MO/V	SQR(6.2831*N)*(N)↑N/EXP(N)
A-INT(A/M)*M	LAMBDA/SIN(2*FI+1)

az alábbiak hibás függvényhivatkozások:

SZIN(3.14)	a függvény neve nem korrekt
LOG(0)	az argumentumbeli érték nincs benne az értelmezési tartományban
TAN(3.141592653/2)	
C S N G(2#/3#)	a szóközök miatt nem ismeri fel benne a konverziós függvényt, külön probléma, hogy az előző hibákkal ellentétben a hibajelzés is elmarad!

2.3 ÉRTÉKADÓ UTASÍTÁS

A BASIC programok leggyakrabban előforduló utasítása az é r t é k a d ó utasítás. Ezzel az utasítással rendelhetünk egy változóhoz valamely értéket. Az értékadó utasítás alakja:

```
        sorszám          LET  változó=kifejezés
vagy
        sorszám               változó=kifejezés
```

Az utasítás szemantikája (végrehajtása):
Először kiértékelődik az "=" jobb oldalán álló kifejezés, azaz a kifejezést alkotó változók pillanatnyi értékükkel helyettesitődnek, a függvények kiszámitódnak és értékükkel helyettesitődnek, a kijelölt műveletek a zárójelek és már megismert kiértékelési szabályok figyelembe vételével elvégződnek. Majd ezt az értéket a bal oldalon levő változó felveszi (korábbi értéke elvész). Ha egy olyan változóra történik hivatkozás, amely eddig még nem kapott értéket, akkor szám típusú esetben 0 értékűként, szöveg típusú esetban ü r e s s z ö v e g ("") értékűként veszi figyelembe az értelmező. (A továbbiakban már ezek a változók is létezni fognak.)
A LET alapszó utal arra, hogy itt nem a szorosan vett matematikai egyenlőségről van szó, hanem egy tevékenységről (tudniillik), hogy "LEGYEN EGYENLŐ"). Így már érthető a matematikailag értelmetlen X=X+1 formájához hasonló BASIC értékadás: LET X=X+1.

Természetes megkötés, hogy a kifejezés és a változó típusa megegyezzen. Tehát

aritmetikai kifejezés értékét csak szám típusú változó kaphatja, ill.
szöveg típusú kifejezés eredményét csak szöveg típusú változó veheti föl!

A HT-1080Z/64 BASIC-re (ahol többféle szám típus is megtalálható) is vonatkozik ez a megkötés azzal a megjegyzéssel, hogy a kifejezésben szerepelhetnek együtt egész és valós (egyszeres valamint duplapontosságú) konstansok vagy változók ugyanazon művelet operandusaiként. Ilyenkor kiértékelés közben a kifejezésben szereplő legnagyobb pontosságú típusúvá konvertálva használja a BASIC az ettől eltérő típusú tagokat, tényezőket (vegyes aritmetika). Ez a CSNG, CDBL függvények automatikus beépülését jelenti. Ha az aritmetikai értékadásban a baloldalon levő változó típusa illetve a kifejezés típusa nem egyezik meg,akkor automatikus konverzió következik be, a megfelelő típuskonverziós függvény használatával, azaz a baloldalon szereplő változó típusára kisérli meg a konverziót.

Szöveg típusú értékadásnál egyetlen szemantikai hibalehetőség adódhat: a szövegkifejezés hossza nagyobb, mint 255 karakter. Ez az a maximális hossz, amilyen hosszban egy változó képes tárolni a szöveget: sőt a szövegkifejezés részeredményére is vonatkozik ez a korlátozás!

Példák:

```
A=(SIN(FI)+COS(FI))/2
FI=ATN(S/C)*360/3.14159
CIM$="'PRO"+" és "+"KONTRA'"
```

Később válik pontosan érthetővé az alábbi példa, de a teljesség kedvéért és az értékadással való szoros kapcsolata miatt már itt említjük a l o g i k a i é r t e k a d á s lehetőségét:

```
IGAZ=(1<3)        (ld. 2.5. Fejezetet)
```

h i b á s a k:

```
A+B=C
A="ALFA+GAMMA"                de  DEFSTR A  után jó
LAP="1"+"2"
F77$='cim cim cimborák'
ha A$-ban egy - mondjuk - 100 karakterből álló szöveg van,
akkor
B$=A$+A$+A$  értékadás közben hibát jelez.
```

2.4 ÍRÓ-OLVASÓ UTASÍTÁSOK

Az író-olvasó (output-input)utasítások segítségével adatokat jeleníthetünk meg a képernyőn, illetve olvashatunk be a billentyűzetről.

Kiíró utasítás

sorszám **PRINT** felsorolás

A felsorolásban megadott értékek megjelennek a képernyőn. A kiírás az aktuális karakterpoziciótól kezdve - kurzor helye - történik.
A **PRINT** szó helyettesíthető a kérdőjel -?- karakterrel.

Példa:

```
>10 ? A/B
>LIST
10 PRINT A/B
READY
>
```

Megjegyzés: a program írásakor a **PRINT** utasítás kérdőjellel való helyettesítése után, a program listázásakor már a **PRINT** szót fogjuk találni a kérdőjel helyén!

Az üres **PRINT** utasítás - amikor a **PRINT** szó után nem írunk semmit - a kurzort a következő sor elejére állítja.

A felsorolás elemei a következők lehetnek:

- szám, vagy szöveg típusú kifejezés
- **TAB** függvény
- @mutató, felsorolás
- **USING** formátum; felsorolás.

A felsorolás elemei közé elválasztó jelet teszünk.

- Szám-, szöveg típusú kifejezés

A kifejezés a legegyszerűbb esetben egy **konstans**. Szám típusú konstans esetén a szám elé és mögé szóköz íródik ki a képernyőre (elé az előjel miatt - pozitiv értéknél szóköz, negativnál pedig a negativ előjel -). Természetesen ez a szám típusú változóra is igaz! A valós típusú konstansok ábrázolási pontossága miatt, a nagyon sok számjegyű konstansok lebegőpontos alakban, bizonyos számú jegyre csonkítva jelennek meg. A számokat írhatjuk lebegőpontos alakban is (lásd részletesen a 2.1. Fejezetben).

Példák:

	Hatása a képernyőn
PRINT 2/3	.66666667
PRINT -1;2;3	-1 2 3
PRINT 99999*9999999	9.9999E11
PRINT 2E5	200000
PRINT "HT-1080Z/64"	HT-1080Z/64
PRINT : PRINT	kettő üres sor

A fenti példák mutatják, hogy a **PRINT** utasítás parancsként is kiadható. A **PRINT** alapszó után írt kifejezés eredményét rögtön kiszámolja, és megjeleníti a képernyőn. Így a **HT-1080Z/64** személyi számítógép remekül használható akár egyszerű kalkulátorként (számológépként) is!

A kifejezés lehet egyetlen **változó** is. A **PRINT** utasítás hatására a változó értéke jelenik meg a képernyőn.

Példák:

```
>10 A=3.14 : B%=-5
>20 T#=12345.678901234 : A$="SZOMBAT"
>30 PRINT A
>40 PRINT T#
>50 PRINT B%
>60 PRINT A$
>RUN
 3.14
 12345.678901234
-5
SZOMBAT
READY
>
```

Általánosabb esetben a megjelenítést megelőzi a kifejezés kiértékelése (a 2.1. Fejezetben leírt módon).

Példák: Legyen B értéke 23.754, D% értéke 15 és B$ értéke "HIRADÁS".

Utasítás	Hatása
10 PRINT -B*B	-564.253
20 PRINT D%/4	3.75
30 PRINT B+D%/2¤3.14	47.304
40 PRINT B$+"TECHNIKA"	HIRADÁSTECHNIKA

Elválasztó jelek

A **PRINT** felsorolásbeli elemei közé **vesszőt** vagy **pontosvesszőt** tegyünk. A pontosvessző közvetlen egymásutánírást jelent, a vesszõ hatására pedig a kiírást a következő tabulátor pozión folytatja (0,16,32,48). A negyedik tabulátor pozició utáni kiírást a következő sor 0. pozicióján kezdi. Ezek a tabulátor pozicók csak akkor érvényesek, ha soronként 64 karakter a kijelzési formátum! (lásd. az 1.2. Fejezetben)

Amikor egy **PRINT** utasítást pontosvessző karakterrel zárunk le (fejezünk be), a következő **PRINT** utasítás ott kezdi a kiírást, ahol az előző abbahagyta. Ha vessző karakterrel fejezzük be, akkor a következő tabulátor pozión, egyébként a sor elejétől kezdi a megjelenítést.

Példák:

```
10 PRINT "A";"B";2;4
RUN
AB 2  4
READY
>10 PRINT "1.POZ","2.POZ","3.POZ","4.POZ","5.POZ"
RUN
1.POZ           2.POZ           3.POZ           4.POZ
5.POZ
READY
>10 PRINT "1.POZ",,,"4.POZ
RUN
1.POZ                                           4.POZ
READY
>
```


- TAB(P) függvény

A **TAB**(P) függvény alkalmazásával megadhatjuk, hogy az adott soron belül hányadik oszloptól (pozíciótól) kezdjen írni a képernyőre. A P értéke **0 - 255** között kell legyen!

A **TAB** csak egy sorban pozícionál, 63-nál nagyobb szám esetén ujra a sor elejétől kezdi pozícionálást.

Példa:

```
>PRINT TAB(22);"22. OSZLOP"
 21 szóköz    22. OSZLOP
READY
>PRINT TAB(15);"FA";TAB(2);"ALMA
               FAALMA
READY
>PRINT TAB(64);"2"
2
```

Megjegyzés:

A FA szó a sor 15. oszlopában jelenik meg, az ALMA szó azonban nem a 2.-ban, hanem a FA szó mögött közvetlenül 16. pozíciótól kezdve, ugyanis előbb kell megadni a kisebb pozíciójú **TAB**-ot.

Ha így használjuk, akkor a második **TAB** már hatástalan.

- PRINT@mutató,felsorolás

A **PRINT@**mutató,felsorolás hatására a képernyőn a mutató által megadott helytől írja ki a felsorolás elemeit. A mutató értéke 0 és 1023 között lehet. (A képernyő 16 soros és 64 oszlopos.)

A sor és oszlop sorszám a következő lehet:

0 <= sor <= 15 és 0 <= oszlop <= 63

Pozicionálás a képernyőn:

0. sorban	0-tól	63-ig,
1. sorban	64-től	127-ig,
. . .		
15. sorban	960-tól	1023-ig (ld. 3.4. Fejezetben)

Példák:

```
50 PRINT@10*64+11,"10. SOR ÉS 11. OSZLOP"
```

A **PRINT** utasítás az idézőjelek között megadott szöveget a képernyő 10. sorának 11. oszlopától kezdve írja ki.

```
10 PRINT@961,"EZ A 15. SOR 1. OSZLOPA LESZ!"
```

- USING formátum;felsorolás

A **PRINT** szó után írt **USING formátum;felsorolás** lehetővé teszi a felsorolás elemeinek megadott formátumban való kiírását. A formátum csak szöveg típusú kifejezés lehet.

A formátum elemei a következő karakterek:

Szám típusú kifejezések formátumai:

KARAKTER	HATÁSA
#	A szám típusú kifejezés számjegyeinek a számát határozza meg. Ha a számoknak több számjegye van, mint a megadott formátumnak (a tizedesponttól jobbra levő számjegyekről van szó), akkor kerekítve jelenik meg az eredmény, ha kevesebb, akkor a számtól balra szóközök, a tizedesponttól jobbra pedig nullák jelennek meg. A tizedespontot a # jelek között bárhová tehetjük. Ha a tizedespont és az első # jel közé tetszőleges helyre vesszőt teszünk, akkor kiíráskor a tizedespont előtti minden harmadik számjegy elé egy vessző kerül. Ha a megadott formátum a szám megjelenítésére nem alkalmas, akkor egy % jellel kezdődik a kiírás.

Példák:

Utasítás	Hatása
10 PRINT USING"###.##";-85	-85.00
20 PRINT USING"#.#";215.5	%215.5
30 PRINT USING"####.#";256.4	256.4
40 PRINT USING"##,###.#";51434.1	51,434.1
50 PRINT USING"##.###";15.4555	15.456

KARAKTER	HATÁSA
**	Ha a formátum elejére két csillagot teszünk, akkor a tizedesponttól balra a számjegyekkel ki nem töltött helyekre csillagokat ír.
$	Ha a formátum elejére írjuk, akkor a legnagyobb helyiértékű számjegy elé egy dollárjel ($) kerül a kiíráskor.
**$	Ha a formátum elejére írjuk, akkor az előző két tulajdonságot tudjuk vele elérni. A legnagyobb helyiérték elé egy $ jel kerül, a többi üres helyre csillag kerül.

Példák:

Utasítás	Hatása
10 E$="**#.#" : N$="$##.##"	
20 Z$="**$##.##"" : A=58.3	
30 PRINT USING E$;A	*58.3
40 PRINT USING N$;A	$58.30
50 PRINT USING Z$;27.8	**$27.80

KARAKTER	HATÁSA
—	Ha kötőjelet írunk a formátum végére, akkor kiíráskor negativ szám esetén - jelet ír a szám után, pozitiv számnál pedig egy szóközt tesz oda.
+	Ha plusz jelet teszünk a formátumunk elejére vagy végére, akkor a szám elé vagy után plusz jel, illetve minusz jel kerül, a szám előjelétől függően.
↑↑↑↑	Ha az eddig emlitett formátumok után négy felfelé nyilat írunk, a megadott formátumú szám lebegőpontos alakban jelenik meg.

Példák:

Utasítás	Hatása
10 PRINT USING "+##.###";25.678	+25.678
20 PRINT USING "#.##+";5.71	5.71+
30 PRINT USING "#.##-";-1.255	1.26-
40 PRINT USING "##.###↑↑↑↑";1#/3#	3.334E-01
50 PRINT USING "##.#↑↑↑↑";10/5	2.00E+00

Szöveg típusú kifejezések formátumai:

KARAKTER	HATÁSA
!	A felkiáltójel segítségével a szöveg típusú kifejezések első karakterét tudjuk kiírni. Ha több felkiáltójelet adunk meg a formátumban, akkor szöveg típusú kifejezések első karakterét vonhatjuk össze velük (egymásután írást jelent). A szövegek száma megegyezik a felkiáltójelek számával. Ha a felkiáltójelek között szóköz van, akkor az összevont karakterek között is ugyanannyi szóköz lesz.
% %	Százalékjel,szóköz,százalékjel. A felsorolásban megadott szöveg típusú kifejezésből balról a szóközök száma plusz kettő darab karaktert leválaszt,és csak ezt írja ki a képernyőre.

Példák:

Utasítás	Hatása
10 PRINT USING "!";"DIÁK"	D
20 PRINT USING "!!!";"FA";"TURA";"CECO"	FTC
30 PRINT USING "% %";"KORSO"	KOR

Megjegyzés:

Ha a formátumban olyan jel (karakter) szerepel, amit nem ismertettünk, vagy nem a megfelelő helyen használtuk,akkor ezek egyszerűen csak megjelennek a képernyőn!
Ha egy formátumhoz több számot adunk meg, akkor a számok a formátum szerint jelennek meg egymásután!

Példák:

Utasítás	Hatása
10 PRINT USING"B(#)=";9	B(9)=
20 PRINT USING"##.#+";33.3;44.4	33.3+44.4+
30 PRINT USING"↑↑↑↑##.#";66.6	↑↑↑↑ 66.6

Nyomtatóra írás

sorszám **LPRINT** felsorolás

A felsorolás elemei megegyeznek a **PRINT** utasításnál felsoroltakkal, kivéve a @ pozícionálást.

Képernyő törlő utasítás

sorszám **CLS**

Az utasítás végrehajtásakor a képernyő tartalma törlődik - **a program nem** -, és a kurzor a képernyő 0. sorának 0. oszlopába áll (a képernyő bal felső sarkába).

Olvasó utasítások

sorszám **INPUT** felsorolás

Az utasítás hatására egy kérdőjel - ? - jelenik meg a képernyőn, (ahol a kurzor éppen áll) és a program adatok beadását várja a billentyűzetről. A bevitelt a **NEW LINE** billentyű lenyomásával fejezzük be!

Példa:

```
>10 INPUT A
>20 PRINT A
>RUN
? 12.4
 12.4
READY
>
```

Megjegyzés:

A kérdőjel után mindig van egy szóköz is!

A felsorolásban szereplő azonosítók lehetnek:

(skaláris) változó,
tömbelem (indexes változó).

Az indexes változók leírását lásd a 2.8. Fejezetben.

A kérdőjel elé az adatokhoz tartozó, magyarázó mondatot is írhatunk. Ez a mondat (**megjegyzés**) a kérdőjel előtt fog megjelenni a képernyőn.

sorszám **INPUT** "szöveg"; felsorolás

Az **INPUT** szó után tetszőleges számú azonosítót írhatunk vesszővel elválasztva. A billentyűzetről beadott adatokat vesszővel lehet elválasztani egymástól. Ha kevesebb adatot adtunk meg, mint amennyi változót felsoroltunk, akkor a gép **két** kérdőjelet ír és várja a többi adat megadását.

Példák:

```
10 INPUT "EMBEREK SZÁMA";N
20 INPUT G,V%,K,A#
30 INPUT K
40 INPUT A,A%
```

Ha több adatot írunk, mint amennyi változót az **INPUT** utasításnál felsoroltunk, akkor a képernyőn a következő üzenet jelenik meg:

```
?EXTRA IGNORED        (túl sok adat)
```

Majd a kiírás után a gép a felesleget figyelmen kívül hagyja, és folytatja az utasítások végrehajtását.

A beírt adatoknak és azonosítók típusának meg kell egyeznie. Ha például szöveget adunk meg szám típusú változónak, akkor az alábbi két sor fog megjelenni a képernyőn:

```
?REDO     (ismételd meg elölről)
?
```

A hibajelzés után az **INPUT** utasításban felsorolt összes változó adatait újra kéri.

Az **INPUT** utasítás a szöveg típusú változóba történő beolvasásnál figyelmen kívül hagyja a bevezető szóközöket és a vessző, vagy kettőspont karakternél befejezi az adatbevitelt. Tehát, ha szóközöket és egyéb írásjeleket is be akarunk olvasni pl. egy változóba, akkor a kérdőjel után az adatokat idézőjelek közé kell tennünk.

Példa:

```
10 INPUT A$
20 PRINT A$
```

A program végrehajtása:

```
READY
>RUN
? "SZOVEG : EGY,KETTO"
SZOVEG : EGY,KETTO
READY
>
```

Egy karakter beolvasása a billentyűzetről

INKEY$

Példa:

```
10 CLS
20 A$=INKEY$  : IF A$="" THEN 20
30 PRINT A$
```

(Az **IF** utasítás működését lásd a 2.5. Fejezetben.)

Az **INKEY$** egy **függvény** ami végrehajtásakor figyeli, hogy az adott pillanatban lenyomtak-e billentyűt. Ha nem ütünk le semmilyen billentyűt, akkor az üres szöveget adja ("" - Ez nem a szóközt jelenti!).

A leütött billentyű képe a képernyőn nem jelenik meg.

Feladat: Várjunk addig, amíg leütik az I billentyűt!

Megoldás:

```
110 PRINT@960,"FOLYTATHATJUK (I)?",
120 A$=INKEY$ : IF A$<>"I" THEN 120 ELSE PRINT A$
```

Ezt az utasítást használhatjuk táblázatok készítésénél: a képernyő teleírása után egy billentyű leütésére töröljük a képernyőt, majd folytatjuk a táblázat megjelenítését. De ugyanígy jól használható például játékprogramoknál is.

A következő függvénnyel a kurzor aktuális pozícióját kapjuk meg egy soron belül.

POS (argumentum)

Az argumentum értéke tetszőleges lehet, mivel jelenléte formális, nem befolyásolja az eredményt. A POS függvény eredménye pedig 0 és 63 közötti szám lesz.

Példa:

```
1100 PRINT TAB(61)"A"; : C=POS(0)
```

Az utasítás végrehajtása után C értéke 62 lesz!

2.5 FELTÉTELES UTASÍTÁS

Reláció

Két azonos típusú mennyiséget relációjelekkel hasonlíthatunk össze. A relációk eredményül logikai értéket adnak. Az IGAZ logikai érték a -1, a HAMIS pedig a 0. Minden más számérték is IGAZ értéket jelent.

kifejezés relációjel kifejezés

A használható relációjelek:

<	kisebb-e?
>	nagyobb-e?
=	egyenlő-e?
<=	kisebb vagy egyenlő-e?
>=	nagyobb vagy egyenlő-e?
<>	különbözőek-e?

Számok összehasonlításakor a relációk a természetesen elvárt értéket adják, szövegek összehasonlításakor pedig karakterenként hasonlítanak. Az a karakter a kisebb, amelynek a kódja kisebb (ld. Karakterkészlet). Két szöveg közül az a kisebb, amelyben előbb fordul elő kisebb karakter. Ha a rövidebb hosszáig megegyeznek a karakterek, akkor a rövidebb a kisebb. A két kifejezés nem lehet különböző típusú: mindkettő vagy szám- vagy pedig szövegkifejezés lehet.

Példák:

"BAKCSO"<"BIBIC"	IGAZ
2*2=4	IGAZ
-1*-1>0	IGAZ
"SZALONKA">"PINTY"	IGAZ
A<=B*B	HAMIS, ha A=10 és B=3
"SZARKA"<>"HOLLO"	IGAZ
B*B-4*A*C>0	IGAZ, ha A=1, B=2 és C=-2 HAMIS, ha A=1, B=2 és C=2
SIN(X)>COS(X)	IGAZ, ha X=1.5

Feltétel

A feltételek is logikai értékkel rendelkező kifejezések, így minden reláció feltétel.

Megjegyzés: Mivel a logikai értékeket is számmal ábrázoljuk, ezért a szám típusú kifejezések is lehetnek feltételek.

Feltételekből logikai műveletekkel állíthatunk elő ujabb,bonyolultabb feltételeket. A feltételek a következő formájúak lehetnek:

```
        NOT   feltétel
feltétel  AND   feltétel
feltétel  OR    feltétel
```

TAGADÁS esetén (NOT) az eredmény IGAZ értékű lesz, ha a feltétel HAMIS volt. Az ÉS művelet (AND) esetén az eredmény akkor IGAZ, ha mindkét feltétel IGAZ volt, míg VAGY műveletnél (OR) akkor IGAZ,ha a két feltétel közül legalább az egyik IGAZ volt.

Példák: A<B AND B<C	IGAZ, ha A=1, B=2 és C=3, HAMIS, ha A=1,B=2 és C=1,
"CSIZ"<"CSOKA" OR "REZNEK"<"TUZOK"	IGAZ,
2*3*5*7<>15*15 OR NOT 7*7<100	IGAZ.

Egy feltétel kiértékelése a relációk logikai értékének meghatározásával kezdődik. Ezután következik a logikai műveletek elvégzése, amelyek sorrendjét az alábbi elsőbbségi szabályok definiálják:

NOT	a legfontosabb,
AND	a következő,
OR	a legutolsó.

Megjegyzés: Szokták az AND műveletet logikai szorzásnak is nevezni,az OR-t pedig logikai összeadásnak.

Az azonos prioritású műveletek kiértékelése - az aritmetikai műveletekhez hasonlóan - balról jobbra történik.

Példák:

(NOT X>Y OR X<0) AND Y<0	A feltétel kiértékelése a zárójelben lévő résszel kezdődik, az pedig a NOT művelettel, így ha X=-1 és Y=2, akkor értéke HAMIS lesz.
NOT X>Y OR X<0 AND Y<0	A feltétel most IGAZ értékű lesz, mert az AND műveletet az OR művelet előtt kell végrehajtani.
A=B=C	Itt két azonos prioritású műveletet kell elvégezni, ezeket balról jobbra végezzük, tehát a feltétel akkor IGAZ értékű, ha a C változóban az A=B reláció igazságértéke van.
"GUVAT"<"GYURGYALAG" AND ("BUBOS BANKA">="CSUSZKA" OR "KUVIK"="UHU")	A feltétel HAMIS, mert a zárójeles kifejezés értéke HAMIS.

Feltételes utasítás - elágazás

A feltételes utasítás arra szolgál, hogy bizonyos utasítások végrehajtását valamely feltételtől tegyük függővé.

```
sorszám  IF feltétel THEN sorszám1
sorszám  IF feltétel THEN sorszám1 ELSE sorszám2
```

Ha a feltétel értéke IGAZ, akkor a program végrehajtása a sorszám1 soron folytatódik, különben a következő soron. A második esetben HAMIS érték esetén a sorszám2 soron.

Példa: Olvassunk be egy 0 és 1 közötti számot (0,1 még lehet)!

```
10 INPUT "SZÁM(0-1)";S
20 IF S<0 OR S>1 THEN 10
```

Az IF utasítás ágaira nem csak sorszámot, hanem tetszőleges utasítást, sőt több utasítást is írhatunk.

```
sorszám IF feltétel THEN utasítás(ok)
sorszám IF feltétel THEN utasítás(ok) ELSE utasítás(ok)
```

A végrehajtás menete megegyezik az előző esettel: ha a feltétel IGAZ értékű, akkor a THEN utáni rész, különben a következő sor, illetve az ELSE utáni rész következik. Ha valamelyik ágra több utasítást kell írni, akkor azokat kettősponttal lehet egymástól elválasztani.

Példa: Három szám nagyság szerinti kiírása.

```
10 INPUT A,B,C
20 IF A<B THEN X=A : A=B : B=X
30 IF B<C THEN X=B : B=C : C=X
40 IF A<B THEN X=A : A=B : B=X
50 PRINT A,B,C
```

Természetesen a fenti két alakot vegyesen is használhatjuk.

Példa:

Az **IF** utasítás mindkét ágán szerepelhet egy ujabb **IF** utasítás is, ilyenkor az **ELSE** utáni rész mindig a közelebbi **IF**-hez tartozik.

Példa: Egy adott pont az egységkörön belül,kívül vagy a körvonalon van-e?

```
100 INPUT X,Y
110 T=X*X+Y*Y
120 IF T<1 THEN PRINT "BELUL" ELSE IF T=1 THEN
        PRINT "RAJTA" ELSE PRINT "KIVUL"
130 STOP
```

A feltételes utasításból a **THEN** ág is elhagyható:

```
sorszám IF feltétel ELSE sorszám1
sorszám IF feltétel ELSE utasítás(ok)
```

Megjegyzés:

A feltételes utasításból a **THEN** szó elhagyható,ha a feltétel az utána következő utasítástól egyértelműen elkülönül. A **THEN** szó továbbá a ","-vel vagy a ":"-tal helyettesíthető.

2.6 VEZÉRLÉSÁTADÁS - MAGYARÁZÓ SZÖVEGEK

Feltétlen vezérlésátadás

A legegyszerűbb vezérlésátadás az, amikor mindentől függetlenül kijelöljük a következő végrehajtandó utasítássort, ezt a **GOTO** utasítással tehetjük meg.

sorszám **GOTO** sorszám.

Ha a **GOTO** utasításban egy nem létező sorra hivatkozunk, akkor ?UL ERROR hibajelzést kapunk az utasítás végrehajtásakor. Természetesen, ha az ilyen utasítást nem hajtjuk végre a programban, akkor hibajelzést sem fog okozni.

Példa: Olvassuk be egy évszak sorszámát (TÉL,TAVASZ,NYÁR,ŐSZ sorrendet feltételezve), majd írjuk ki az évszak nevét!

```
10 INPUT "ÉVSZAK SORSZAM";E
20 IF E=1 THEN PRINT "TÉL" : GOTO 60
30 IF E=2 THEN PRINT "TAVASZ" : GOTO 60
40 IF E=3 THEN PRINT "NYÁR" : GOTO 60
50 PRINT "ŐSZ"
60 ...
```

A következő utasitás hatására a végrehajtás a 10-es soron folytatódik, mert a sorszám végét egy nem számjegy karakter jelzi:

```
100 GOTO 10*10
```

Több irányú elágazás

A feltételes vezérlésátadásnak van egy speciális esete: a sokirányú elágazás. Ha egy kifejezés 1,2,3, ... értéke szerinti sokfelé ágazást kell készíteni, akkor a BASIC nyelv **ON GOTO** utasitását használhatjuk.

sorszám **ON** számkifejezés **GOTO** sorszám1,...,sorszámN

Amennyiben a kifejezés értéke

1	akkor	sorszám1 ,
2	akkor	sorszám2 ,
...		...
N	akkor	sorszámN

soron folytatódik a program végrehajtása. Ha a kifejezés értéke nem egész, akkor a BASIC az egészrészével számol. Az **ON GOTO** utasításban maximum 255 sorszám adható meg. Ha a kifejezés értéke negativ, akkor hibajelzést kapunk, ha több mint 255, akkor is. Ha pedig a felsorolt sorszámok számánál több, de 255-nél nem, akkor a végrehajtás a következő soron folytatódik. Ez történik 0 esetén is. Azok a sorszámok, amelyekre vezérlésátadás nem történik, az utasításból elhagyhatók.

Példák: Oldjuk meg az előbbi feladatot **ON GOTO** utasítással!

```
10 INPUT "ÉVSZAK SORSZAM";E
15 ON E GOTO 20,30,40,50
20 PRINT "TÉL" : GOTO 60
30 PRINT "TAVASZ" : GOTO 60
40 PRINT "NYÁR" : GOTO 60
50 PRINT "ŐSZ"
60 ...
```

Egy áru vásárlásakor a nyugdíjasok 20% kedvezményt kapnak, a nagycsaládosok 30%-ot, míg mások nem kapnak kedvezményt. Egy %-összeg alapján a program írja ki a kedvezmény okát!

```
10 INPUT "KEDVEZMÉNY(0,20,30 %)";K
20 IF K<>0 AND K<>20 AND K<>30  THEN 10
30 ON K/10 GOTO ,50,60
40 PRINT "NINCS KEDVEZMÉNY" : GOTO 70
50 PRINT "NYUGDÍJAS" : GOTO 70
60 PRINT "NAGYCSALÁDOS"
70 ...
```

Ha egy változó csak 2, 3 vagy 5 lehet, akkor helyes a következő utasítás:

```
100 ON X GOTO ,200,300,,500
```


Megállító utasítások

A program végrehajtásának befejezésére két utasítás is szolgál, amelyek hatása a HT-1080Z/64-en azonos:

sorszám **STOP**
sorszám **END**

Hatásukra pontosan a következők történnek:

- A program végrehajtása befejeződik, a gép parancsra vár, kiírja a

```
Break in sorszám
READY
>
```

  üzenetet. Egyetlen különbség: az első sort csak a **STOP** hatására írja, a végrehajtás ebben a sorban fejeződött be.
- A program változóinak az értéke megmarad, s így leállás után kiírható.
- A program futtatása a **CONT** paranccsal folytatható.

Magyarázó szövegek a programban

Az elkészült program érthetőségét, "olvashatóságát" nagyon megkönnyíthetik a programban elhelyezett magyarázó szövegek:

sorszám **REM** szöveg

Az utasítás "hatástalan", azaz nem változtatja meg a program változóinak értékét és a végrehajtás a következő soron folytatódik. A **REM** után tetszőleges szöveg állhat, így a kettőspont nem jelenti egy uj utasítás kezdetét.

Megjegyzések:

- Egy sort ideiglenesen hatástalaníthatunk a programban, ha a sorszám után beszúrjuk a **REM** kulcsszót.
- A **REM** helyettesíthető az ' karakterrel.

2.7 CIKLUS

A ciklus fogalmának megközelítése céljából kiindulunk a következő feladatból: döntsük el programmal azt, hogy egy évszám szökőév-e, vagy sem! A szökőévnek azt az évet nevezik, amely 4-gyel osztható, illetve az évszázadok közül csak azokat, amelyek 400-zal is. E definíciót pontosan követve igen egyszerű "programot" kapunk:

```
10 INPUT "Az évszám=";E
20 E4=E/4 : EE=E/100 : ES=E/400 : REM hányadosok
30 IF E4=INT(E4) AND EE<>INT(EE) OR ES=INT(ES) THEN 40 ELSE 50
40 PRINT "szökőév" : GOTO 60
50 PRINT "nem szökőév"
60 STOP
```

Módositsuk e feladatot úgy, hogy ne csak egyetlen évszámról tudja megmondani a program, hogy szökőév-e vagy sem, mondjuk például 100-ról. Arra nyílván senki sem gondol, hogy megoldásként a programot az eredeti megoldás megsokszorozásával állítsa elő. Olyan megoldást kell keresnünk, amely magjaként tartalmazza a fenti feladat egyszeri megoldását, s mintegy kereteként azt számlálja (adminisztrálja), hogy hányadik végrehajtásnál tart (ill. kell-e még ismételten végrehajtani). Amint szükséges számban elvégezte a feladatot, kilép e programhurokból, vagy - ahogy gyakrabban fordul elő a számítástechnikai terminológiában - **ciklus**ból. Tehát ciklusnak nevezzük utasítások ismételt végrehajtását, ciklusszervezésnek azt az utasításkörnyezetet, amellyel gondoskodunk arról, hogy az ismétlendő utasítások megfelelő sokszor hajtódjanak végre. Eddigi ismereteinkkel is már megbírkóznánk e feladattal:

```
 5 I=1
10 INPUT "Az évszám=";E
20 E4=E/4 : EE=E/100 : ES=E/400 : REM hányadosok
30 IF E4=INT(E4) AND EE<>INT(EE) OR ES=INT(ES) THEN 40 ELSE 50
40 PRINT "szökőév" : GOTO 60
50 PRINT "nem szökőév"
60 I=I+1 : IF I<=100 THEN 10
```

Jól felismerhető, hogy három résztevékenységből áll elő a ciklus:

1. a ciklusváltozó (=számláló) kezdeti értékének beállitása.
2. a ciklusmag utasításai,
3. a ciklus lefutását (un. terminálását) ellenőrzi, az ismétlést elökészitő utasitás, valamint a programhurkot bezáró utasitások együttese.

(Az 1. és 3. tevékenységek alkotják az un. megszámlálásos ciklusok szervezési környezetét. Vannak másfajta ciklusok is, de mivel a **HT-1080Z/64** BASIC ez utóbbiakhoz nem ad utasitástámogatást, ezekről nem ejtünk szót.)

A fenti módosított program egyszerűbb megvalósítását teszi lehetővé az alábbi BASIC utasításpár:

```
sorszám1 FOR ciklusváltozó=kezdet TO vég STEP lépés
  ...          a ciklusmag utasításai
sorszám2 NEXT ciklusváltozó
```

ahol a kezdet, a vég és a lépés szám értékű kifejezés lehet, a ciklusváltozó pedig egész vagy valós típusú változó.

Bár az 1. megoldással pontosan megadtuk a ciklus utasítás szemantikáját, szavakkal mégis összefoglaljuk:

A ciklusváltozó először felveszi a kezdőértéket, majd a ciklusmag végrehajtódik. Ezután a ciklusváltozó értékéhez a lépésközt hozzáadva vizsgálja a ciklus lejártát. Ha a ciklusváltozó értéke már tullépte a végértéket (figyelembe véve a "változás" irányát, azaz a lépésköz előjelét), akkor a ciklusmag utáni utasításon folytatódik a végrehajtás, ha még nem lépte túl, akkor ujra végrehajtódik a ciklusmag. Pozitiv lépésköz mellett a ciklus akkor fejeződik be, ha a ciklusváltozó nagyobb,mint a végérték, negativ lépésköz esetén pedig, ha a ciklusváltozó kisebb,mint a végérték. Igy, ha a ciklusváltozó értéke "átlépte" (a fenti értelemben) a végértéket, a ciklus befejeződik: a program végrehajtása a **NEXT**-et követő utasításon folytatódik.

A fejezet elején módosított program most már igy irható:

```
 5 FOR I=1 TO 100 STEP 1
10 INPUT "Az évszám=";E
20 E4=E/4 : EE=E/100 : ES=E/400 : REM hányadosok
30 IF E4=INT(E4) AND EE<>INT(EE) OR ES=INT(ES) THEN 40 ELSE 50
40 PRINT "szökőév" : GOTO 60
50 PRINT "nem szökőév"
60 NEXT I
```

Mivel gyakran van szükség 1 lépésközü ciklusra, lehetővé teszi a BASIC azt is, hogy a **STEP**-pel kezdődő szintaktikai egységet elhagyjuk. Ilyenkor a lépésköz 1 lesz. Tehát a

```
FOR I=1 TO 100     egyenértékű     FOR I=1 TO 100 STEP 1
```

Feladat:

Egy 'm' tömegű testet 'h' magasságból leejtünk. A test helyzeti energiája folyamatosan átalakul mozgási energiává. Ezeket az értékeket foglaljuk táblázatba, a magasság függvényében!

Megoldás:

Egy test helyzeti energiája 's' magasságban:

$$E_{hely.}(s) = m * g * s \qquad (g=9.81)$$

A csúcspontban ('h') a mozgási energia nulla, így a test mozgási energiája 's' magasságban:

$$E_{mozg.}(s) = E_{hely.}(h) - E_{hely.}(s) = m*g*h-m*g*s$$

A program bemenő adatai legyenek: a test tömege,az ejtés magassága és a táblázat "lépésköze".

```
10 PRINT "Szabadon eső test helyzeti "
15 PRINT "és mozgási energiája"
20 INPUT "A test tömege";M
30 INPUT "Az ejtés magassága";H
40 INPUT "A táblázat lépésköze (>0)";L
60 REM Az input adatokat beolvastuk
70 W=9.81*M : REM Az m*g szorzat
80 E=W*H : REM Az energia a csúcspontban
90 REM A táblázat fejléce
100 PRINT "Magasság","helyzeti energia","mozgási energia"
120 REM A táblázat elkészítése
130 FOR S=H TO 0 STEP -L
140 PRINT S,W*S,,E-W*S
150 NEXT S
160 STOP
```

Egy programban természetesen több ciklus is használható. Sőt megengedett a ciklusok skatulyázása is azzal a magától értetődő megszoritással, hogy az egymást tartalmazó ciklusok ciklusváltozóinak különbözőeknek kell lenniük, valamint a **FOR** és a **NEXT** utasítások helyesen legyenek párositva.

Példa:

Írassuk ki az 1 és 2 számokból összeállítható összes számpárt!

```
110  FOR I%=1 TO 2
120  FOR J%=1 TO 2
130  PRINT I%;",";J%
140  NEXT J%
150  NEXT I%
```

A kiírt számpárok:

```
1 , 1
1 , 2
2 , 1
2 , 2
```


Megjegyzések:

- A ciklus lefutása után a ciklusváltozó értéke az lesz, amivel a ciklusmag már nem hajtódott végre.

- A ciklusmagban ne változtassuk meg a ciklusváltozót!

- A ciklusmagon belül a kezdőérték, végérték és lépésköz megváltoztatása nincs hatással a ciklus lefutására.

- Egy ciklusba csak a **FOR** - ciklusnyitó - utasításon keresztül lépjünk be!

- A **NEXT** utasításban szereplő ciklusváltozót nem kötelező kiírni. Ez azonban a program hordozhatóságát és olvashatóságát nagy mértékben csökkenti, ezért alkalmazását nem javasoljuk.

- Az egymás után álló két vagy több **NEXT** utasítás összevonható, például:

```
90 NEXT J
95 NEXT I
```

helyett

```
90 NEXT J,I
```

is irható. Mivel ez is erősen HT-1080Z/64 sajátosság, ezért az összevonást sem javasoljuk.

- Amennyiben a ciklus lépésközének nullát adunk meg, a ciklus magja vég nélkül ismétlődni fog.

- Az alábbi furcsaság a számábrázolással magyarázható:

```
10 FOR X=1 TO 1.3 STEP .1
20 PRINT X;
30 NEXT X
```

A kiírt eredmény :

```
1 1.1 1.2
```

A ciklus magja - legnagyobb meglepetésünkre - X=1.3-ra nem hajtódott végre. Ennek az az oka, hogy a kiírt szám az ábrázolt szám kerekítésével keletkezik. Így lehetséges az, hogy az 1-hez 3-szor adva (körülbelül!) 0.1-et, X tartalma (a gépben) nagyobb lesz mint 1.3 (ld. még 3.5. Fejezetet).

- Figyelni kell a **FOR** ciklus azon sajátosságára, hogy legalább egyszer végrehajtódik (hiszen a **NEXT** utasításnál vizsgál). Ezért, ha nem garantálható az, hogy a ciklus kezdő és végérték mindig a kívánt, értelmes relációban állnak, akkor az esetleges nem kívánatos ciklusba lépést egy előzetes vizsgálattal ki kell kerülni.
Például így:

```
100 IF I+1>2*N THEN 210
110 FOR J=I+1 TO 2*N
...    ...
200 NEXT J
210 ...
```


2.8 INDEXES VÁLTOZÓK-TÖMBDEKLARÁCIÓK

Indexes változók

A matematikában megszokott módszer, hogy bizonyos változókat **indexelünk**. Erre a **HT-1080Z/64** BASIC is lehetőséget nyújt.
Alakjuk:

```
azonosító (index)
azonosító (sorindex,oszlopindex)
azonosító (index1,index2 ...,indexN)
```

Az első esetben **vektorról**, a másodikban **mátrixról**, a harmadikban pedig **általános tömbről** beszélünk. Az index minden esetben olyan kifejezés, amely pozitív számot ad eredményül. Ha ez nem egész, akkor a **HT-1080Z/64** egészre csonkítja. A tömbelemek típusa minden eddig megismert típus lehet. Az ilyen változókat indexes változónak nevezzük. Műveleteket csak a tömbök egyes elemeivel végezhetünk, magával a tömbbel nem. A V(5) indexes változónak semmi köze nincs a V nevű változóhoz.

Példák:

A(11)	A valós típusú A vektor 11. eleme.
AA%(I,J)	Az egész típusú AA% mátrix I. sorának J. eleme.
V$(0)	A szöveg típusú V$ vektor 0. eleme.
M#(X,Y,Z)	A dupla pontosságú M# tömb X, Y, Z "koordinátákkal" megadott eleme.

Egy szöveg típusú tömb minden egyes elemében természetesen nem csak egy karaktert, hanem egy-egy tetszőleges szöveget helyezhetünk el.

Példák:

```
110  S$(1)="Arany János"
120  T$(1,1)="Toldi" : T$(1,2):="Tetemrehívás"
```

Mint a példákból is kiderült, a tömbök indexelése 0-tól kezdődik és pozitív egész számokkal történik. Az index felső határa, ha nem mondunk mást, akkor 10 lesz. Tehát ekkor a vektorok 11 elemet tartalmaznak, a mátrixok pedig 11 sort és 11 oszlopot, azaz 121 elemet.

Példa: Adjuk meg az A(N,M) mátrix sorösszegeit!

```
... ...
100 FOR I=1 TO N
110   S=0
120   FOR J=1 TO M
130     S=S+A(I,J)
140   NEXT J
150   PRINT I;". SORÖSSZEG=";S
160 NEXT I
... ...
```

Megjegyzés: A **FOR** utasítás ciklusváltozója nem lehet indexes változó!

Tömbdeklarációk

Ha nem felel meg az alapértelmezés szerinti tömbméret, mert például 10-nél nagyobb index kellene, vagy pedig takarékoskodni akarunk, akkor meg kell adni a tömb indexeinek felső határát. Erre szolgál a **DIM** utasítás:

sorszám **DIM** azonosító (indexhatár1,...indexhatárN)

ahol az indexhatárok olyan kifejezések,amelyek szám értékkel rendelkeznek.

Példák:

```
10  DIM A(5)
```

Az A vektor valós számokat tartalmaz és 0-tól 5-ig lehet indexelni.

```
20  DIM S$(1,20)
```

Az S$ mátrix minden egyes eleme egy tetszőleges szöveg, első indexe 0 és 1 lehet, a második pedig 0 és 20 közötti egész szám, azaz a mátrixnak 2*21=42 eleme van.

Egy **DIM** utasításban több tömb méretét is megadhatjuk. Ezeket ilyenkor vesszővel kell elválasztani egymástól.

Példa:

```
30  DIM A(10,20),B(20,10),C(10,10)
```

Egy tömb helyfoglalása, azaz méretének meghatározása futás közben történik, így lehetőség van arra, hogy az index felső határát futás közben kiszámított adattal és ne konstanssal adjuk meg.

Példa: N név közül a legkisebb kiválasztása

```
100 INPUT "NEVEK SZÁMA(>1)";N
110 IF N<2 THEN 100
120 DIM A$(N)
130 FOR I=1 TO N
140   PRINT I;". NÉV"; : INPUT A$(I)
150 NEXT I
160 L=1
170 FOR I=2 TO N
180   IF A$(I)<A$(L) THEN L=I
190 NEXT I
200 PRINT "LEGELSŐ NÉV:";A$(L)
210 STOP
```

TILOS egy tömb méretét újradeklarálni, még akkor is, ha a méret valójában nem változik. Ha mégis megkísérelnjük, akkor hibajelzést kapunk.

Vigyázni kell arra is, hogy a tömb deklarálása előzze meg a tömbre való első hivatkozást, ugyanis ilyenkor beállítja a feltételezett értéket,s a későbbi **DIM** utasítást már újradeklarálásnak fogja fel a BASIC, s hibát jelez rá.

A parancsok között ismertetett **CLEAR** parancs viszont törli a deklarációkat is. Ez ad lehetőséget arra, hogy a programban mégis használjuk az újradeklarálást.

Példa: Újradeklarálás megvalósítása

```
10 CLEAR
20 INPUT "TÖMBMÉRET";N
30 DIM A(N)
.. ...
99 GOTO 10
```

A 10-es sorban megszüntetjük a deklarációkat, a 20-asban beolvassuk az index felső határát, s a 30-asban deklaráljuk a tömböt. A 99-es sorból a 10-re visszatérve újra "tiszta lappal" indulhatunk.

2.9 ADATOK ELHELYEZÉSE A PROGRAMBAN

Az ebbe az utasításcsoportba tartozó utasítások feladata,hogy változókhoz megadott értékeket rendeljenek hozzá. Ez megoldható értékadó utasításokkal is (2.3. Fejezet). Ezek az utasítások így tulajdonképpen nélkülözhetők, de bizonyos feladatoknál áttekinthetőbbé,rövidebbé teszik a programot.

Az adatokat tároló utasítás (DATA)

Alakja:

sorszám **DATA** konstansl,konstans2,...,konstansN

Konstansl, konstans2, ... szám vagy szöveg-konstans lehet (2.1. Fejezet). A szöveg-konstansokat - eltérően az értékadó utasítástól és általában a szöveg kifejezésektől - nem kötelező idézőjelek (") közé tenni.

Idézőjelek közé kell tenni a szöveg-konstanst, ha:

- a szöveg-konstans vesszőt tartalmaz, ugyanis a vessző a konstans végét jelezné,
- a szöveg-konstans kettőspontot tartalmaz,
- a szöveg-konstans szóközt tartalmaz,különben a szóközöket a konstans elejéről elhagyja.

A **DATA** utasításokat úgy képzelhetjük el, mintha konstansaik egymás után fel lennének fűzve egy láncra.

A konstansokat olvasó utasítás (READ)

Az utasítás alakja:

sorszám **READ** változól,változó2,...,változóN

Változól, változó2,..., változóN tetszőleges típusú, indexes és index nélküli változó lehet.
Az utasítás eredménye, hogy a változók felveszik a **DATA** utasítások soronkövetkező konstansainak értékét.

A **READ** és **DATA** utasítások elemeinek megfeleltetése: a program indulásakor egy mutató rááll a program első **DATA** utasításának első konstansára. Ez a konstans lesz az első **READ** utasítás első változójának értéke, majd a mutató a következő konstansra áll rá.

Példa:

```
10 READ A,B$,C(1),D%
20 DATA 1,"KONSTANS",3.14,5%
```

Eredménye: A=1, B$="KONSTANS", C(1)=3.14, D%=5%

Emlékezzünk rá, hogy a **DATA** utasítások konstansai egy képzeletbeli láncra vannak felfűzve. Ha egy **DATA** utasításból a konstansok elfogynak, akkor a **DATA** konstansok mutatója a sorszám szerint következő **DATA** utasítás első konstansára mutat. (Ha van következő **DATA** utasítás.)
A **READ** és a **DATA** utasítások megfelelő elemeinek típusaiban is illeniük kell egymáshoz az alábbi táblázatnak megfelelően:

READ-ben változó típusa	DATA elem típusa		
	Valós	Egész	Dupla pontos
Valós	+	+	Konverzió és kerekítés
Egész	INT függv.	+	INT függv.
Dupla pontos	Konverzió ld. példa	HIBA	+ ld. 1. példa

Ha a **READ** utasításban szöveg típusú változó következik, ennek a **DATA** listán tetszőleges típusú konstans megfelelhet.

Ha nincs ! vagy E betű a konstansban, akkor a valós számot duplapontosnak tekinti, ha az értékes számjegyek száma legalább 7.

```
10 DATA 123456789,1.23456789,1.23456789E0
20 READ A#,B#,D#
30 PRINT A#,B#,D#
```

Eredménye:

```
A#=123456789 B#=1.23456789 D#=1.234567880630493
és kiirja
123456789        1.23456789     1.234567880630493
```

```
10 DATA A BC,  DEFG,HIJKL   ,MNO
20 READ W$,X$,Y$,Z$
30 PRINT W$;X$;Y$;Z$
```

Eredménye: W$="A BC"X$="DEFG"Y$="HIJKL "Z$="MNO" és kiírja
A BCDEFGHIJKL MNO
Látható, hogy az idézőjel nélkül leírt szöveg-konstansok elejéről a szóközöket elhagyja.

A DATA utasítások mutatóját mozgató utasítás (RESTORE)

Az utasítás alakja:

sorszám **RESTORE**

RESTORE hatására a program legelső **DATA** utasítására áll a mutató. **READ** utasítással csak annyi változó kaphat értéket, ahány konstans a **DATA** utasításokban van. Ezt dinamikusan kell érteni, mert egy **READ** utasításra többször is ráadódhat a vezérlés, egy **DATA** utasításra pedig **RESTORE** utasítással többször visszatérhetünk.

Néhány példa a fenti utasításokra.

Feladat: döntsük el egy karakterről, hogy magánhangzó-e!

Megoldás:

```
10 DATA a,e,i,o,u
20 N%=5%
30 A$=INKEY$:IF A$="" THEN 30
40 RESTORE
50 FOR I%=1%TO N%
60 READ B$:IF B$=A$  THEN PRINT "Magánhangzó":GOTO 30
70 NEXT I%
80 PRINT "Nem magánhangzó":GOTO 30
```

Feladat: állapítsuk meg, hogy egy adott évben mely hónapokban esett péntek 13-ára.

Megoldás:

```
10 DATA 31,28,31,30,31,30,31,31,30,31,30,31
20 RESTORE
30 INPUT "Melyik évre kíváncsi";EV:INPUT "Milyen napra esett január
1";NA
40 S=0
50 FOR I=1 TO 12
60 H=S+13-6+NA:IF H/7=INT(H/7) THEN ?I;". hónapban esik 13-ára"
70 READ H:S=S+H:IF I=2 AND EV/4=INT(EV/4) THEN S=S+1
80 NEXT I
90 GOTO 20
```

Megjegyzések: a 30-as sorban a nap sorszámát kell beírni;
hétfő=1, kedd=2,....,vasárnap=7
a 70-es sor a szökőévet figyeli, de a szökőévet itt nem ellenőrizzük olyan szigorúan, mint 2.7-ben.

2.10 SZÖVEGKEZELÉS

A szöveg típusú változókban vagy a szöveg típusú indexes változók elemeiben tetszőleges, a BASIC karakterkészletébe tartozó karakterekből álló szövegeket lehet elhelyezni. Az így megadott karaktersorozatokkal műveletet is lehet végezni, pl. két ilyen sorozatot egymás mögé lehet írni (ld. értékadás). Szövegekkel azonban sokkal több műveletet is végezhetünk. Az aritmetikai függvényekhez hasonlóan léteznek **szöveg függvények** is. Így lehet egy-egy szövegrészletet megvizsgálni, kiemelni, szövegrészletekből új szöveget összeállítani stb. A szövegkezelő függvények ismertetése előtt még néhány tudnivaló a szöveg típusú változókról:

Az egyes szöveg típusú változókban, illetve a szöveg típusú vektor egyes elemeiben nem lehet 255-nél több karaktert elhelyezni. A változó minimális "hossza" 0, ekkor nincs benne egyetlen karakter sem. A számítógép a szöveg típusú változókat egy külön memóriaterületen tárolja (ld. A memória felosztása) amelynek méretét bekapcsoláskor 50 byte hosszúságúra választja. Ez a terület bonyolultabb szövegkezelő programok esetén nagyon kevés, ha több kellene, akkor a programnak kell intézkedni több hely lefoglalásáról. A szövegek kezelésére rendelkezésre álló területet a következő utasítással lehet megnövelni:

sorszám **CLEAR** byte-szám

A szükséges érték kiszámításához a következőket kell figyelembe venni:

- a program szöveg típusú változói, indexes változói helyfoglalása,
- a szövegkezelő függvények használatához szükséges munkaterület,
- a szövegkifejezések kiértékelésekor az összes részkifejezés tárolásához szükséges hely.

Mivel a fentiekből az utolsó kettőt viszonylag nehéz meghatározni, ezért általában célszerű felülbecsülni a szükséges memóriaméretet. FIGYELEM! A **CLEAR** parancs egyéb hatásaira ügylni kell: a változók értékét, típusát, a tömbök méretdefinícióját törli! Ha szükség van rá, célszerű a program első utasításaként elhelyezni. Segítséget jelent a szükséges méret meghatározásához a következő függvény:

FRE (szöveg típusú változó vagy konstans)

A függvény értéke egy szám, amely megadja, hogy hány byte áll még rendelkezésre szövegek kezeléséhez.

Példák: 30 PRINT FRE("A")
60 PRINT FRE(X$)
80 PRINT FRE(X$+"MI")

Gyakori használata ellen szól viszont, hogy jelentősen lelassítja a program futását. A függvény értékének meghatározásához ugyanis a BASIC értelmezőnek végig kell vizsgálnia az egész, szövegkezelésre kijelölt területet,meg kell állapítania, hogy mely részei szabadok, ami esetenként hosszabb időbe telhet.
Megjegyzés: A függvény argumentumának megadhatunk tetszőleges szöveg típusú változót vagy szövegkonstanst, szerepe ilyen szempontból nincs, csupán formai szempontok miatt kötelező. (Minden BASIC függvénynek van argumentuma.)

Szövegkezelő függvények:

Értékük szempontjából két alapvető csoportra oszthatjuk:

- a függvény értéke egy új szöveg,
- függvény értéke szám jellegű.

A szöveg értékű függvények neve után a "$" - szöveg jelet kell írni, így formailag is megkülönböztetjük a szám értékű függvényektől. Szöveg kifejezés lehet szöveg konstans, szöveg típusú változó, vagy szöveg függvényekkel és a "+" művelettel előállított karaktersorozat.

Szöveg hosszának megállapítására használható a következő függvény:

LEN (szöveg kifejezés)

Példák:

```
10 A$="ALMA"                    40 A$="ROZSA"
25 H=LEN(A$)                    50 PRINT LEN(A$+"LEVEL")
```

után H értéke 4. után a kiírt szám 10.

de vigyázni kell az ékezetes betűkkel, mert
70 PRINT LEN("RÓZSALEVÉL")
eredménye 12 lesz, mivel az ékezetes betűket a HT-1080Z/64 2 karakteren tárolja.

Szöveg valamely részének kiválasztására három függvény szolgál. Az első a **szöveg baloldalát** választja le, a szöveg első adott darabszámú karakterét adja meg:

LEFT$ (szöveg kifejezés,hossz)

Példák:

```
10 A$="MEDVESZOLO"
20 B$=LEFT$(A$,5)
```

utána B$ értéke a "MEDVE" szöveg lesz.

```
10 A$="FUZFABOKOR"
20 PRINT LEFT$(A$,5),LEFT$(A$,3)
```

hatására a FUZFA és a FUZ szavakat írja ki a képernyőre

de itt is vigyázni kell, mert a HT-1080Z/64 a
60 PRINT LEFT$("FÜZFA",3)
hatására a FÜ szöveget, a
70 PRINT LEFT$("FÜZFA",2)
hatására pedig az F szöveget adja eredményül, a második karakter nem látszik.

Megjegyzés: az ékezetes betűk nagyon nehézzé teszik a szövegkezelést, használatukat inkább csak kiíró utasításokban javasoljuk, s esetleg nagyon bonyolult és fontos szövegkezelő programokban forduljanak elő.

Szöveg jobboldali részszövegét az alábbi függvény szolgáltatja. Itt is karakterszámot kell megmondani.

RIGHT$ (szöveg kifejezés,hossz)

Példák:

```
30 A$="CSOMBOR"
40 C$=RIGHT$(A$,3)
```

után C$ értéke a "BOR" szöveg lesz.

```
60 AA$="REBARBARA"
70 PRINT RIGHT$(AA$,7)
```

hatására a "BARBARA" szöveg kerül a képernyőre.

Megjegyzés: Mindkét függvény a teljes szöveget adja eredményül, ha a megadott hossz nagyobb, mint a szöveg hossza.

Nem csak bal- vagy jobboldali részét választhatjuk ki egy szövegnek, hanem **tetszőleges középső részét** is, adott sorszámú karaktertől adott darabszámú karaktert, esetleg adott sorszámútól a szöveg végéig:

MID$ (szöveg kifejezés,sorszám,hossz)
MID$ (szöveg kifejezés,sorszám)

Példák: 80 A$="SZEKFUVIRAG"
90 B$=MID$(A$,5,2)
után B$ értéke a "FU" szöveg lesz.

40 A$="FARKASALMA"
50 AA$=MID$(A$,7,6)
után AA$ értéke az "ALMA" szöveg lesz, ugyanis ha az adott karaktertől kezdődő rész-szöveg hossza kisebb, mint a megadott hossz, akkor az eredmény a teljes maradék rész lesz.

70 A$="NADRAGULYA"
80 PRINT MID$(A$,6)
hatására az eredmény a "GULYA" szöveg lesz.

A **HT-1080Z/64** a szövegeket karakterenként tárolja, a karaktereket ASCII kódjukkal ábrázolja. A karakterek kódját részletesen a Karakterkészlet fejezet tartalmazza. Ez alapján történik a szöveg típusú adatok, változók összehasonlítása is, tehát az a karakter a "kisebb",amelynek a kódja kisebb. Egy karakter **kódját**, illetve egy szöveg első karakterének kódját a következő függvénnyel kaphatjuk meg:

ASC (szöveg kifejezés)

Példák: 100 V=ASC("A")
után V=65, az A betű kódja lesz

100 V$="LIBATOP"
110 V=ASC(V$)
utasítás végrehajtása után V=76, azaz az "L" betű kódja.

Létezik a fenti függvény "ellentettje" is, amely egy ASCII **kódhoz** megadja a megfelelő **karaktert.**

CHR$ (kód)

Példák: 10 A$=CHR$(65)
után A$ az "A" betűt tartalmazza.

10 A$=CHR$(65)
20 A$=CHR$(72)+A$+CHR$(78)+CHR$(71)+A$
hatására pedig a "HANGA" szöveget.

Megjegyzés: A grafikus üzemmód tárgyalásánál fogunk találkozni az un. grafikus jelekkel: bizonyos kódokhoz nem a mindennapi életben megszokott jeleket rendeltek, hanem grafikus jeleket. Ezek megjelenítéséhez is a CHR$ függvény szükséges.

Példa: PRINT CHR$(144)
utasítás olyan grafikus karaktert ír ki, amelynek bal alsó sarkában van kigyújtva egy fénypont.

Sok olyan karakterkód is van, amelyhez nem tartozik billentyű, így csak a **CHR$** függvény segítségével adható ki.

Példa: 100 PRINT CHR$(28)
az aktuális kurzorpozíciót a képernyő bal felső sarkába állítja.

Ha olyan karaktersorozatot kell kiírni, amely sok azonos karakterből áll, akkor a **STRING$** függvényt használhatjuk, amelynél a darabszámon kívül vagy az ismétlendő karaktert vagy pedig az ASCII kódját kell megadni.

STRING$ (darabszám,kód)
STRING$ (darabszám,"karakter")

A karaktersorozat hossza maximum 255 lehet (gondoljunk a szöveg típusú változók maximális hosszára), még akkor is, ha csak kiírni akarjuk, s nem elhelyezni egy változóban.

Példa: szöveg kiírása keretbe

10 PRINT STRING$(31,"-")
20 PRINT "!";STRING$(10," ");"CICKAFARK";STRING$(10,32);"!"
30 PRINT STRING$(31,45)

A **HT-1080Z/64** lehetőséget biztosít arra is,hogy számokat ne a belső ábrázolási formában, hanem decimálisan tároljunk. A decimális tárolást szöveg típusú változókban oldhatjuk meg, de a **HT-1080Z/64** nem ad műveleteket, függvényeket az ilyen módszerrel tárolt számokhoz. Ezért szükség van olyan függvényekre, amelyek a művelet elvégzése előtt binárisra alakítják a számokat, utána pedig újra decimálisra. A **decimálisra átalakító függvény:**

STR$ (aritmetikai kifejezés)

Példa: óra, perc, másodperc kiírás két-két jegyre (OO:PP:MP formában)

10 PRINT RIGHT$(STR$(100+0),2);":";
20 PRINT RIGHT$(STR$(100+P),2);":";
30 PRINT RIGHT$(STR$(100+MP),2)

Egy másik függvény szolgál arra, hogy a **decimális számokat átalakítsuk binárisra**, ez a VAL.

VAL (szöveg kifejezés)

Példa:
```
10 A$="3" : B$="14159"
20 A=VAL(A$+"."+B$)
```
után A értéke 3.14159 lesz

```
20 X$="1495" : Y$="8"
30 PRINT VAL(X$+"E"+Y$)
```
hatására a képernyőn megjelenik a 1495E8-nak megfelelő 1.495E+11

Ha a szöveg típusú kifejezés nemcsak számjegyeket tartalmaz, akkor csak az első nem számjegy karakterig fogja átalakítani (de a szóközöket természetesen nem veszi figyelembe).

Példa:
```
10 A$="3/6"
20 A=VAL(A$)
```
után A értéke: 3 lesz.

Feladat: egy adott szövegben számoljuk meg, hogy az ABC betűi hányszor fordulnak elő!

(Legyen például a vizsgált szöveg:
"Este van, este van: kiki nyugalomba",
ebben az A betűk száma: 4, a B betűk száma: 1, stb.)

A megoldáshoz a betűk kódjait fogjuk felhasználni, a legkisebb kódú az A betű (65), utána jön a B (66),.. végül a Z (90). A kisbetűk kódjai az a betűvel kezdődnek (97), utána a b betűé következik (98),.. A megoldásban minden egyes betűhöz, pontosabban a betű kódjához az eredmény vektor (DB) egy elemét rendeljük:

```
A kódja: 65     DB(1)
B kódja: 66     DB(2)
C kódja: 67     DB(3)
          :
          .
          .
Z kódja: 90     DB(26)
```

A megfelelő elem indexét tehát úgy kapjuk meg, hogy a betű kódjából levonunk 64-et. A vektor 0-ás indexű elemében pedig számoljuk a szövegben előforduló egyéb jelek számát. A kisbetűket is számolni kell, mégpedig a nagybetűkkel együtt, így a kisbetűk kódjából 96-ot kell levonni.

Megoldás:

```
10 CLEAR 1000 : DIM DB(26)
20 CLS : PRINT "BETUK GYAKORISAGAT SZAMLALO PROGRAM"
30 PRINT "KEREM A VIZSGALANDO SZOVEGET"
40 INPUT S$
50 REM SZOVEG KARAKTERENKENTI VIZSGALATA
60 FOR I=1 TO LEN(S$)
70 J=ASC(MID$(S$,I,1))
80 REM INDEX MEGALLAPITASA
90 IF J>=65 AND J<=90 THEN J=J-64 : GOTO 120
100 IF J >=97 AND J<=122 THEN J=J-96 : GOTO 120
110 J=0
120 DB(J)=DB(J)+1
130 NEXT I
140 REM KIIRAS
150 PRINT "AZ EGYES BETUK GYAKORISAGA:"
160 FOR I=1 TO 26
170 IF DB(I)>0 THEN PRINT CHR$(I+64);" BETU:";DB(I),
180 NEXT I
190 PRINT : PRINT "EGYEB JELEK:";DB(0)
200 END
```

2.11 SZUBRUTINOK

A programozási gyakorlatban sokszor találkozunk azzal a problémával, hogy a program több helyén is ugyanazt a néhány utasítást kell végrehajtani. Tekintsük a következő igen egyszerű példát! Tegyük föl, hogy több táblázatot szeretnénk készíteni programunkkal. Mindegyik táblázat készítésénél meg kell oldanunk a képernyő barátságos kezelését, ami a következőket jelenti: képernyő törlése, fejléc kiírás, a táblázat képernyő nagyságú lapokra tagolása, stb.

```
... ...
100 IF INKEY$="" THEN 100 : REM várakozás
110 CLS : PRINT FEJ$ : REM fejléc
120 SOS=2
... ...
```

Készíthetjük a programot úgy, hogy minden szükséges helyre ugyanazt a kis programrészletet leírjuk.

```
... ...
100 IF INKEY$="" THEN 100 : REM várakozás
110 CLS : PRINT FEJ$ : REM fejléc
120 SOS=2
... ...
300 IF INKEY$="" THEN 300 : REM várakozás
310 CLS : PRINT FEJ$ : REM fejléc
320 SOS=2
... ...
```

Ez igen kényelmetlen dolog lenne különösen, ha nem 2-3 utasításról van szó! Ügyesebb megoldásra is van lehetőségünk: a többször használandó programrészt kiemeljük, s csak egyszer, a program egy elkülönített helyén írjuk le. Azokon a pontokon, ahol szükség van rá, valahogy jelezzük,hogy itt most azt a kis különálló programrészletet kell végrehajtani. Ezt a különálló programrészt nevezzük **szubrutin**nak. A szubrutin szó jelentése: "alprogram", ez az az utasítás-sorozat, melyet egyszer, a program egy különálló részén adunk meg, majd több helyen is használhatunk.Egy speciális vezérlésátadással jelezzük, hogy az alprogramot kell végrehajtani, ezt nevezzük **szubrutinhívás**nak:

sorszám **GOSUB** sorszám1

Ennek hatására a vezérlés átadódik a sorszám1-en kezdődő alprogramnak és végrehajtódik a kívánt programrészlet. Jogosan merülhet föl a kérdés: ezt egy egyszerű feltétlen vezérlésátadással (**GOTO** utasítással) is megtehetnénk, mi akkor a különbség? A szubrutin végrehajtása után vissza tudunk térni a hívás utáni első utasításra, ami szükséges is, ha a szubrutint a program több helyéről is szeretnénk aktivizálni, meghívni, majd annál a pontnál folytatni, ahonnan a szubrutint hívtuk. Ezt már általában nem (vagy csak nagy ügyeskedés árán) tudnánk **GOTO** utasításokkal megoldani, hisz a szubrutin végéről **GOTO**-val csak egy konkrét sorszámra adhatjuk a vezérlést, és nem a mindenkori hívás utáni első utasításra! Az alprogram végének jelzésére egy külön utasítás szolgál, nevezetesen a szubrutinból való **visszatérés** utasítása:

sorszám **RETURN**

Példa:

```
... ...
100 GOSUB 500 : REM ide fog visszatérni először
... ...
300 GOSUB 500 : REM ide fog visszatérni másodszor
... ...
490 STOP
500 REM itt kezdődik a szubrutin
...     a szubrutin utasításai
580 RETURN
```

Amikor a program a 100-as sorszámú utasítást hajtja végre, a vezérlés átadódik az 500-as utasításra, s ott folytatódik a végrehajtás mindaddig, amíg a **RETURN** utasításhoz nem ér, amelynek hatására a vezérlés visszatér a hívás utáni első utasításra, példánkban a 100-as 2. utasítására. Természetesen amikor a 300-as utasítás hívja meg ugyanazt a szubrutint, az a 300-as sor 2.utasítására tér vissza. Ha a szubrutin tartalmaz elágazó utasításokat, a szubrutin végét jelző **RETURN** utasítást több helyre is elhelyezhetjük. (Bár megjegyezzük, hogy számos programozási indok támasztja alá azt, hogy célszerű egyetlen pontra helyezni a szubrutin logikai végét jelentő **RETURN** utasítást.) Éppúgy, ahogy a ciklusok egymásba skatulyázhatók, a szubrutin belsejéből hívhatunk újabb szubrutin(oka)t, azokban ismét lehetnek szubrutinhívások stb. Ilyenkor a **RETURN** mindig az utolsó szubrutinhívás mögé tér vissza.Megjegyezzük, hogy a **RETURN** utasítás végrehajtása egy megelőző **GOSUB** nélkül értelmetlen, (?RG ERROR in sorszám) hibajelzést okoz, ezért a fenti programnak valahol a 300-as és 500-as utasítás között **STOP** vagy **END** utasítást kell tartalmaznia (esetleg vezérlésátadással **GOTO**) ki kell "kerülnie" a szubrutint. Éppen emiatt célszerű a szubrutinokat a program végére tenni, elkülönítve így a főprogramtól. A program áttekinthetőségét segítik a **REM** utasítások. Szubrutinokat tartalmazó program esetén érdemes elhelyezni mind a hívásnál egy magyarázó szöveget, hogy milyen szubrutint hívunk meg, mind pedig a szubrutin első utasítása legyen egy **REM** utasítás, ami a szubrutin feladatát írja le.

Példa:

```
... ...
100 GOSUB 500 : REM ujlap
... ...
300 GOSUB 500 : REM ujlap
... ...
490 END
500 REM a táblázat következő lapjának előkészítése
510 PRINT@15*64,"Ha elolvasta,üssön le egy billentyűt!";
520 IF INKEY$="" THEN 520 : REM várakozás
530 CLS : PRINT FEJ$ : REM fejléckiírás
540 SOS=2
550 RETURN
```

A szubrutinok alkalmazhatóságának köre többrétű. Első és legkézenfekvőbb alkalmazása, amiről a fejezet elején már szó volt: ha ugyanazt a programrészletet többször is tartalmazza a programunk különböző helyeken. Ilyenkor egyszer, egy szubrutinban megírjuk, és ahol szükséges, szubrutinhívással aktivizáljuk. A másik példa a használatára: hosszú, bonyolult (néhány száz soros) programok esetén a program áttekinthetősége és jól tagoltsága érdekében célszerű az elkülöníthető, önállóbb részleteket (pl. adatok beolvasása, eredmények kiírása, algoritmus egyes részletei stb.) szubrutinnak megírni, így a program "fő" vezérlési struktúrája könnyen áttekinthetővé válik, szinte csak szubrutinhívásokat tartalmaz.

Annak eldöntésekor, hogy a programban helyezzünk-e el szubrutint (és ha igen, mi legyen az), érdemes mind a két szempontot figyelembe venni.

Ahogy a GOTO utasításnak volt un. kiszámított változata (ON GOTO), éppúgy a szubrutinhívásnak is van kiszámított szubrutinhívás alakja:

sorszám ON kifejezés GOSUB sorsz1,sorsz2,...,sorszN

Végrehajtása az ON GOTO utasításhoz hasonlóan: a vezérlés a kifejezés értékének egész része szerint a megfelelő, sorsz1-en, sorsz2-n, ...sorszN-n kezdődő szubrutinra adódik, majd a szubrutinból való visszatérés után az ON GOSUB utasítás utáni első utasítás hajtódik végre. Ha a szubrutint kiválasztó kifejezés értéke kívül esik az értelmes 1-N tartományon, akkor vagy a következő utasításon folytatódik a végrehajtás(mintegy "üres" szubrutinhívást hajtva végre), vagy hibajelzést kapunk. Az utóbbira akkor kerül a sor, ha a 0-255 tartományon kívülre mutat az érték. (Ld. még a 2.6. Fejezetben.)

Példák:

K=1 esetén az első szubrutint hajtjuk végre, K=2 esetén a másodikat, stb.

```
... ...
100 ON K GOSUB 500,600,700 : REM ide fog visszatérni, a meghívott szubrutinból
... ...
490 GOTO 10
500 REM első szubrutin
... ...
550 RETURN
600 REM második szubrutin
... ...
680 RETURN
700 REM harmadik szubrutin
... ...
850 RETURN
```

Az alábbi példában hibajelzés és tényleges szubrutinhívás nélkül következik a végrehajtás az ON GOSUB utasítást követőn:

```
... ...
100 KAPCS=4
110 ON KAPCS GOSUB 1000,2000,3000 : REM szubrutinhívás nélkül itt folytatódik a végrehajtás
... ...
990  STOP
1000 REM 1. szubrutin
.... ...
1990 RETURN
2000 REM 2. szubrutin
.... ...
2990 RETURN
3000 REM 3. szubrutin
.... ...
3990 RETURN
```

Az ON GOSUB utasítás végrehajtása az ?FC Error in 210 üzenettel szakad meg:

```
... ...
200 ROSSZ=-1
210 ON ROSSZ GOSUB 500,505
... ...
490 END
500 REM 1. szubrutin
... ...
504 RETURN
505 REM 2. szubrutin
... ...
509 RETURN
```

Feladat: Gyakorlásként írjunk programot, amely egy háromszög oldalainak ismeretében kívánságra kiszámítja a területét vagy kerületét!

Megoldás: Használjuk fel az új ismereteinket! (Bár ezek nélkül is megoldható lehet a feladat.) Két szubrutint készítünk, az egyik a kerületet, a másik a területet határozza meg. A felhasználó "menü" alapján választhat, a választ felhasználva a megfelelő szubrutint számított szubrutinhívással aktivizáljuk.

```
10  CLS
20  INPUT "Kérem a háromszög oldalainak hosszát!";A,B,C
30  IF A+B<=C OR A+C<=B OR B+C<=A THEN PRINT  "Ebből nem lesz  há-
romszög!" : GOTO 20
40  PRINT "Mit mondjak meg?"
50  PRINT TAB(10);"1. Kerület"
60  PRINT TAB(10);"2. Terület"
70  INPUT "Válaszod (1/2)";V%
80  IF (V%<1) OR (V%>2) THEN 70
110 ON V%GOSUB 200,300
120 INPUT "Van még kérdésed (I/N)";V$
130 V%=ASC(LEFT$(V$,1))AND 95 : REM kis betűből nagy
140 IF V%=ASC("I") THEN 10
150 IF V%<>ASC("N") THEN 120
160 PRINT "Viszontlátásra"
190 STOP
200 REM Kerület számitása
210 PRINT "A háromszög kerülete:";A+B+C
220 RETURN
300 REM Terület számitása
310 S=(A+B+C)/2
320 T=SQR(S*(S-A)*(S-B)*(S-C))
330 PRINT "A háromszög területe:";T
340 RETURN
```

Figyeljük meg a fenti programbeli programozási fogásokat:

- a lehetséges ellenőrzéseket az **INPUT** utasítás után rögtön elvégezzük.
- A program a szubrutinok alkalmazása miatt könnyen kibővíthető, hogy a háromszög egyéb adatait is ki tudja számolni: be kell venni a "menü"-be az új lehetőséget, megírni a tevékenységet elvégző szubrutint,és az **ON GOSUB** utasitást kiegészíteni az uj szubrutin sorszámával.
- Az IGEN/NEM kérdésre adott választ akkor tekintjük "IGEN"-nek,ha az I-vel kezdődik, tehát elfogadja az "IGEN" minden rész-szavát, de például az "ISKOLA" választ is. Hasonlóan kezeljük a "NEM" választ. A kisbetűket a 130-as sorban "nagy párjukra" transzformáltuk (ld. még 2.16. Fejezetben).

A következő feladatban a BASIC-ben gyakorta fellépő egyik komoly nehézség (ti. a szubrutinok nem paraméterezhetőségéből adódó gond) megoldására mutatunk példát. Mit jelent a paraméterezhetőség? A szubrutinok általában valamilyen változókon, tömbökön végeznek el valamilyen transzformációt. Nevezzük ezeket a változókat, tömböket paramétereknek! Sokszor a hívó programrészlet nem csak egy paraméter-együttessel szeretné használni a szubrutint. Ilyenkor a következőt tehetjük: a szubrutin megírásakor lerögzítünk, és csak ebben a szubrutinban használhatónak tekintünk néhány változót, tömböt, amelyen keresztül bonyolítjuk le a kommunikációt a szubrutin és az őt hivó programrész között.

Feladat: Írjunk szubrutint a következő - szövegekkel kapcsolatos feladatokat megoldó programokban gyakran felbukkanó - részproblémára: a szubrutin döntse el, hogy az A9$ szöveg tartalmazza-e a B9$ szöveget az I9-edik poziciótól kezdve. Ha igen, akkor az I9 változóban azt a karaktersorszámot adja meg,amelynél az A9$ szövegben a B9$ szöveg kezdődik. Ha nem, akkor értéke legyen 0. Tehát a szubrutin paraméter-változói: A9$, B9$, I9; az első kettő bemenő paramétere, míg a harmadik két feladatot is betölt. Egyrészt bemeneti célokat szolgál, másrészt az eredményt is tartalmazni fogja. Szükségünk lesz még egy segédváltozóra,amit a ciklusszervezéshez használunk. Jó tanácsként mondhatjuk, hogy a szubrutinban használt segédváltozóknak érdemes minél szokatlanabb, ritkán használt nevet adni, így elkerülhetjük, hogy a szubrutin elrontsa valamely fontos, a főprogramban is használt változó értékét. Szubrutinunkban ciklusváltozónak ezért egy IW nevű változót használunk. (Használunk még egy AHOSSZ, ill. egy BHOSSZ nevű segédváltozót is.)

Megoldás:

```
1000 REM I9:=INSTR(A9$,B9$,I9)
1010 AHOSSZ=LEN(A9$) : BHOSSZ=LEN(B9$)
1020 IF AHOSSZ-BHOSSZ+1<I9 THEN GOTO 1060
1030 FOR IW=I9 TO AHOSSZ-BHOSSZ+1
1040 IF B9$=MID$(A9$,IW,BHOSSZ) THEN I9=IW : GOTO 1070
1050 NEXT IW
1060 I9=0
1070 RETURN
```

Megjegyezzük, hogy a BASIC implementációk többsége tartalmaz egy a fenti funkciót betöltő függvényt, amelyet **INSTRING** néven szoktak beépíteni a standard függvényeik közé.

2.12 VÉLETLENSZÁMOK

A számítógépek **véletlenszám generátora** segítségével nagyon sok játékot, természeti jelenséget szimuláló programot készíthetünk. A **HT-1080Z/64** gépen kétféle fajtája van, az egyik a [0,1) intervallumban képes egyenletes eloszlású véletlen valós számokat adni (tehát olyan valós számokról van szó, amelyek közé a 0 beletartozik, az 1 pedig már nem), a másik pedig 1 és N közötti egészeket. Számítógépekkel mindig ál-véletlenszámokat állitunk elő, amelyek nagyon hasonlítanak a valódi véletlenhez. Véletlenszám generátoraink az előző véletlenszám értékéből állítják elő a következőt.

Ez felvet egy problémát. Ha a véletlenszámokat determinisztikusan állítjuk elő, akkor ezek elvileg megismételhetők lennének, ha ugyanabból a kezdőszámból indulnánk ki. A **HT-1080Z/64** erre is szolgáltat megoldást. Az **induló véletlenszám** érték megválasztására szolgál a **RANDOM** utasítás, alakja:

sorszám **RANDOM**

A **véletlenszámok készítésére** egy függvény szolgál, amely a kétféle véletlenszámhoz igazodva kétféle alakban használható. Első alakja:

RND (0)

Ez a függvény olyan számokat szolgáltat, melyekre teljesül mindaz, amit a véletlentől várunk: azaz előre nem jósolható meg az értéke, a lehetséges értékei egyenlő eséllyel fordulnak elő.

Másképp fogalmazva: annak a valószínűsége, hogy egy véletlenszám a [0,1) intervallum egy adott részintervallumába esik, csak annak a hosszától függjön. Valós véletlenszámok esetén annak a vizsgálata értelmes, hogy a véletlenszám beleesik-e egy adott intervallumba. Ez az a tulajdonság, melyet felhasználva valóban hasznos programokat készíthetünk.

Példa: Modellezzünk 10 darab pénzfeldobást, s írjuk ki az eredményt:

```
10 FOR I=1 TO 10
20 IF RND(0) < .5 THEN PRINT "FEJ" ELSE PRINT "IRAS"
30 NEXT I
40 STOP
```


Sok esetben nem a [0,1) intervallumban kell véletlenszámokat előállítani, hanem például a [0,A) intervallumban. Ekkor a véletlenszám értékét transzformálni kell:

```
10 X=A*RND(0)
```

Az [A,B) intervallumban pedig igy állíthatunk elő véletlenszámokat:

```
10 X=(B-A)*RND(0)+A
```

Más a helyzet, ha egész számokra van szükségünk, ekkor használhatjuk az **RND** függvény másik alakját:

RND (N)

Ez 1 és a megadott N természetes szám (N>1) között fog adni véletlenszerűen választott egész számot. N értéke 1 és 32767 közötti egész szám lehet.

Példa: lottószámok készítése (1 és 90 közötti egész számok).

Ezt a következőképpen lehet megtenni (egyelőre nem törődünk azzal, hogy igy egy számot kétszer is kiválaszthatunk):

```
.. ...
20 DIM A(5)
30 FOR I=1 TO 5
40 A(I)=RND(90)
50 NEXT I
.. ...
```

Hasonlóképpen 10 kockadobást a következőképpen végezhetünk el:

```
10 FOR I=1 TO 10
20 PRINT RND(6)
30 NEXT I
```

Véletlenszámok kezelésekor gyakran elkövethetjük az alábbi hibákat:

Példa: pénzfeldobás szimulálása

```
10 IF RND(0)<.5 THEN PRINT "FEJ"
20 IF RND(0)>=.5 THEN PRINT "IRAS"
```

A programot kipróbálva gyakran előfordul, hogy semmit sem ír ki, máskor pedig az, hogy a FEJ és az IRAS szöveget is kiírja. Ennek magyarázata az, hogy az RND függvény minden végrehajtásakor egy újabb véletlenszámot készít, s így a fenti program 10-es és 20-as sorában két különböző számot vizsgálunk. Ezért lehet egyszerre mindkét feltétel igaz értékű is és hamis értékű is, s természetesen előfordul az is, hogy egyszerre csak az egyik teljesül. Hasonló eset áll fenn a következő példában is:

Példa: Írjuk ki, hogy egy kockadobás eredménye 2 és 3 közé esik-e!

```
10 IF 2<=RND(6) AND RND(6)<=3 THEN PRINT "IGEN"
```

Ebben az esetben is kettő különböző véletlenszámot vizsgálunk,s így nem a várt gyakorisággal kapjuk az eredményt.

Feladat: Készítsünk 5 lottószámot és írjuk ki őket nagyság szerint növekvő sorrendben!

A megoldás első lépésében elkészítjük az 5 számot, ügyelve arra, hogy ne legyen közöttük két egyforma. Ezután sorbarendezzük őket úgy, hogy a még rendezetlen sorozatból mindig a legkisebbet választjuk következő számnak.

```
10 DIM A(5) : RANDOM
20 REM LOTTOSZAM GENERALAS
30 FOR I=1 TO 5
40   A(I)=RND(90)
50   IF I=1 THEN 90
60   FOR J=1 TO I-1
70     IF A(J)=A(I) THEN 40 : REM NEM UJ SZAM
80   NEXT J
90 NEXT I
100 REM RENDEZES
110 FOR I=1 TO 4
120  MI=I
130  FOR J=I+1 TO 5
140    IF A(MI)>A(J) THEN MI=J
150  NEXT J
160  X=A(I) : A(I)=A(MI) : A(MI)=X
170 NEXT I
180 REM EREDMENYEK KIIRASA
190 PRINT "LOTTOSZAMOK:";
200 FOR I=1 TO 5
210   PRINT A(I);
220 NEXT I
230 STOP
```


2.13 ADATTÁROLÁS KAZETTÁN

A kazettát programok tárolásán kívül adatok megőrzésére is használhatjuk. Egy program eredményeit kiírhatjuk a kazettára, illetve adatait beolvashatjuk a kazettáról.

Erre szükség lehet például akkor, ha:

- az adatok gyűjtése és feldolgozása több lépésben, esetleg felváltva történik,
- az adatokat módosítani, javítani kell,
- nagyon sok adatot kell többször feldolgozni egy programmal,
- nagy tömegű adatot több program is használ.

Az adatkezelés a kazettára kiíró- és kazettáról beolvasó utasításokkal történik. Ezekhez hasonló utasításokkal a 2.4. Fejezetben már találkoztunk, most kisebb módosítással használjuk őket.

Kazettára kiíró utasítás

sorszám **PRINT#**-kazettaszám,felsorolás

A kazettaszám lehet:

1 (ez a számítógép beépített magnója),
2 (kimenetre csatlakoztatható magnó).

Az utasítás hatására a kazettaszámmal kijelölt magnetofon motorja elindul (természetesen a magnónak felvétel állapotban kell lennie) és a felsorolásban megadott elemeket kiírja a magnóra - a képernyőre való kiíráshoz hasonlóan, karakteresen. Egy **PRINT** utasítással maximum 247 db karaktert írhatunk ki a kazettára.
A felvitel után a magnetofon motorja megáll.

Az **adatrekord** egy rekord bevezető részből (257 byte), valamint a program által kiírt valódi adatokból áll. A felsorolás elemei a **PRINT** utasításban megengedettek lehetnek.

Megjegyzések:

Nem használhatjuk a @mutató, felsorolás lehetőséget (lásd 2.4. Fejezet)!
Az idézőjelek között beírt egyéb írásjelek közül a kettőspontot nem tudjuk visszaolvasni!
Lásd részletesen a kazetta olvasó utasításnál!
A vessző adat elválasztó jel egy rekordon belül, ezért kell kiírnunk (mint az INPUT utasításnál)!

Példák:

```
5  N=12*34.5/3 : K(5)=SQR(151.2) : D=-625.25
10 PRINT#-1,"SZAMOK";",";5.1;",";-23.67;",";0.678342;",";
"SZAMOK VEGE"
20 PRINT#-1,N
30 PRINT#-2,K(5) : REM KÜLSŐ MAGNÓRA!
40 PRINT#-1,D
```

Feladat:

Írjuk ki a következő vektorok értékeit a további feldolgozás céljából kazettára! A vektorok N eleműek legyenek! (NE$(N),R(N),K$(N))

Megoldás:

A kiírás elkezdése előtt jelezzük, hogy a magnót felvételre kell állítani! Az adatok elé írjunk egy címet, amivel azonosítani tudjuk majd beolvasáskor! Írjuk ki a vektorok méretét is! Erre szükség lesz majd a beolvasáskor! A képernyőre is írjuk ki az adatokat!

```
100 PRINT "ÁLLITSA A MAGNÓT FELVÉTELRE !" : PRINT
110 PRINT "HA KÉSZ, NYOMJON MEG EGY BILLENTYÜT !";
120 IF INKEY$="" THEN 120
130 PRINT "ADAT KIÍRÁS KEZDETE:"
140 PRINT#-1, "ADATOK.11.20." : REM A CÍM
150 PRINT "ADATOK.11.20."
160 PRINT#-1,N : REM VEKTOROK MÉRETE
170 PRINT "N=";N
180 FOR I=1 TO N
190 PRINT#-1,NE$(I);",";R(I);",";K$(I)
200 PRINT I;". ADAT:"; NE$(I);R(I);K$(I)
210 NEXT I
220 PRINT "ADAT KIÍRÁS VÉGE!"
230 END
```


Megjegyzés:

Érdemes egy PRINT utasítással több adatot kivinni a kazettára!
Mint már említettük egy adatrekord írása a valódi adatokon kívül bevezető rész kiírását is jelenti. Emiatt egy rekord kazettára kiírása (egy PRINT utasítás) hosszú lesz.

Ha egy PRINT utasítással több adatot akarunk kiírni, akkor kiírt adatok közé írjunk egy vessző karaktert, ugyanis beolvasáskor az INPUT utasításnál megismert módon az adatokat a vessző választja el egymástól (lásd 2.4. Fejezet olvasó utasítás).

Kazettáról olvasó utasítás

sorszám INPUT#-kazettaszám,felsorolás

Az utasítás végrehajtásakor a magnó motorja elindul (a magnónak lejátszás állapotban kell lennie! - ld. 1.3. Fejezet -).
A felsorolásban megadott változók a kazettán tárolt adatok beolvasásával sorban megkapják értéküket. A soron következő változóhoz tartozó adatok végét egy vessző karakter jelzi - mint a billentyűzetről való beolvasásnál -, vagy az adatok vége.
Miután a változók megkapták értéküket, a magnó motorja leáll és az utasítás végrehajtása befejeződik.
Ha a kiírt adatok között a kettőspont karakter előfordul vagy több adat esetén, a következő üzenet jelenik meg a képernyőn:

?EXTRA IGNORED

A kettőspont utáni karakterek, illetve a többi adatok elvesznek. A figyelmeztető üzenet kiírása után a program folytatja az utasítások végrehajtását.

Példák:

```
10 INPUT#-1,B(2,1),B%,B#
20 INPUT#-1,K$
30 INPUT#-2,D$,Y(3,2,1) : REM KÜLSŐ MAGNÓRÓL!
```

Megjegyzések:

Egy INPUT utasítással max. 247 karakter olvasható be.

V i g y á z z u n k ! !

Ha kevesebb adatot olvastunk be egy rekordból,mint amennyit előzőleg oda kiírtunk, akkor a következő INPUT utasítás már a következő rekordot olvassa!

Feladat:

Olvassuk be a korábbi feladatunkban kiírt adatokat, az ott ismertetett tömbökbe! Az első rekord itt is a cím legyen! Írjuk ki a képernyőre az olvasás kezdetét és végét, valamint a beolvasott adatokat!

Megoldás:

```
500 PRINT "ÁLLÍTSA A MAGNÓT LEJÁTSZÁSRA !" : PRINT
510 PRINT "HA KÉSZ, NYOMJON MEG EGY BILLENTYŰT !";
520 IF INKEY$="" THEN 520
530 REM OLVASÁS KEZDETE
540 INPUT#-1,A$
550 PRINT "CÍM :";A$
560 IF A$<>"ADATOK.11.20." THEN 540
570 PRINT "OLVASÁS KEZDETE:"
580 INPUT#-1,N : REM VEKTOROK MÉRETE
590 PRINT "N=";N
600 DIM NE$(N),R(N),K$(N)
610 FOR I=1 TO N
620 INPUT#-1,NE$(I),R(I),K$(I)
630 PRINT I;".ADAT:"NE$(I);R(I);K$(I)
640 NEXT I
650 PRINT "BEOLVASÁS VÉGE!"
700 END
```


2.14 GRAFIKUS UTASÍTÁSOK

A HT-1080Z/64 számítógép rendelkezik úgynevezett grafikus utasításokkal. Ezekkel a képernyőre ábrákat, alakzatokat, grafikonokat tudunk rajzolni. Ezeken kívül a számítógép karakterkészletében találhatók olyan karakter kódok is, melyeknek grafikus szimbólumok (ábrák) felelnek meg. Ezt szintén használhatjuk a képernyőn rajzolásra, ábrák készítésére, megjelenítésére.

A grafikus utasítások ismertetése előtt a képernyő felépítéséről ejtsünk néhány szót.

A képernyő 16 soros és soronként 64 oszlopos. Egy karakter pozíció hat pontból áll (3 sor és 2 oszlop). Lásd a 3.4. Fejezetben.A grafikus utasításokkal a karakter pozíció pontjai közül bármelyiket elérhetjük. Így a képernyő ponthatárai a következők lesznek:

O <= SOR <= 47 és O <= OSZLOP <= 127

Az értelmezési tartomány túllépésekor ?FC Error hibaüzenettel leállítja a programot!

A képernyő 48 grafikus sorból és 128 grafikus oszlopból áll. A sorok illetve az oszlopok számozása a képernyő bal felső sarkában kezdődik.

Grafikus utasítás

A grafikus utasításokkal a képernyő tetszőleges pontja (az oszlop és sor koordinátáival megadott pont) kigyújtható, illetve törölhető.

A képernyő egy pontjának a kigyújtása:

sorszám SET (oszlop,sor)

Példák:

>SET (90,11)

A képernyő 90. oszlopában és 11. sorában levő pontot kigyújtja, megvilágítja. Természetesen parancsként is kiadható.

```
>10 CLS : PRINT "KOCKADOBÁS EREDMÉNYE!"
>20 SET(15,15) : SET(10,10) : SET(20,20)
>30 SET(10,20) : SET(20,10)
>40 END
```

Egy ötöst dobtunk a kockával. Fél képernyős kijelzésre állítva lesz négyzet forma.

```
>SET(128,1)
?FC Error
```

Hibaüzenetet kapunk, mert a pont kívül esik a ponthatáron!

A képernyő egy pontjának a törlése:

sorszám **RESET** (oszlop,sor)

Példák:

```
65 RESET (111,27)
```

A képernyő 111. oszlopában és 27. sorában levő pontot törli.

A képernyő egy pontjának vizsgálata függvénnyel:

POINT (oszlop,sor)

A **POINT** függvény - logikai értékű - értéke igaz,ha az adott pont világít, hamis ha a pont nem világít.

Példa:

```
>10 IF POINT(25,51) THEN PRINT "A PONT VILÁGÍT" ELSE
PRINT "A PONT NEM VILÁGÍT"
```


Feladat: Gyújtsunk ki a képernyőn adott számú pontot! Ami már világít azt ne gyújtsuk ki mégegyszer! A pontok kirajzolása után csak a V billentyű lenyomására fejezzük be a program végrehajtását! (Azért, hogy ne rontsuk el az ábrát!)
A megoldásnál használjuk a véletlenszámokat generáló függvényt (**RND** - lásd 2.12. Fejezetben)!
A rajzolásnál ügyeljünk arra, hogy a pont a képernyőre kerüljön! Ezért a pont koordinátái legyenek a következők:

```
P2=RND(128)-1 : P1=RND(48)-1
```

P2 - oszlop koordináta, P1 - a sor koordináta.

Megoldás:

```
100 INPUT "A PONTOK SZÁMA";A : IF A<1 THEN 100 ELSE CLS
110 FOR I=1 TO A
120 P1=RND(48)-1 : P2=RND(128)-1
130 IF POINT(P2,P1) THEN 120
140 SET(P2,P1)
150 NEXT I
160 IF INKEY$<>"V" THEN 160
```

Grafikus szimbólumok

A képernyőre rajzolhatunk grafikus ábrákat a **PRINT** utasítással, és a **CHR$** függvénnyel.

Feladat:

Írassuk ki a képernyőre a grafikus ábrákat (grafikus szimbólumokat)!

Megoldás:

```
10 CLS : PRINT CHR$(23); : REM A KIJELZÉS.= 32 KARAKTERES
20 FOR I=129 TO 191
30 PRINT I;CHR$(I), : REM KOD;GRAF.ÁBRA
40 NEXT I
```

Megjegyzés:

A képernyő kijelzési formátumnak leírását lásd a 3.4. Fejezetben. A grafikus karakterek kódjai a 3.6. Fejezetben találhatók.

2.15 GÉPI SZINTŰ ÍRÓ-OLVASÓ UTASÍTÁSOK

Az INPUT és a PRINT utasításnál jóval egyszerűbb beolvasó utasítások is léteznek a HT-1080Z/64-en.Ezek egy byte átvitelére képesek. Ilyen utasítások alkalmazására van szükség, amikor valamilyen hardver eszközt akarunk közvetlenül vezérelni. Az így vezérelhető eszközök közé tartozik például a hanggenerátor vagy a kazettás magnó, ha nem BASIC szinten akarjuk kezelni. A byte-ok átvitelére szolgáló utasítások megfelelnek a Z80-as mikroprocesszor IN és OUT utasításainak. (A HT-1080Z/64-ben Z80-as mikroprocesszor található.) A **bemeneti függvény** alakja:

INP (berendezés cím)

A berendezést, ahonnan egy byte-ot beolvashatunk, egy 0 és 255 közötti szám azonosítja. Ezen számok közül néhány már foglalt: a 255 jelenti a magnót, a 254 a beépített és a külső magnó közüli választást, a 30,31 pedig a hanggenerátor programozására szolgál. Ha itt nem 0 és 255 közötti számot adunk meg, akkor a program végrehajtása a ?FC Error hibaüzenettel megszakad.

Példa: 10 X%=INP(N)

Ez az utasítás az N bemenetről beolvas 1 byte-ot és az X% egész változóba teszi.

A - sokkal gyakrabban használt - **kimeneti utasítás** alakja:

sorszám OUT berendezés cím, kifejezés

Példa: 10 OUT N,X%

utasítás hatására az N kimenetre kiírja az X% egész típusú változót. Nem csak a berendezés azonosító, hanem a kiírt érték is csak 0 és 255 közötti szám lehet, különben szintén ?FC Error hibaüzenetet kapunk. A kimeneten ez az érték mindaddig megmarad, amíg más értéket nem írunk oda (vagy a RESET nyomógombbal nem töröljük).

Feladat: Egy útkereszteződés két lámpáját kell állítani programmal úgy, hogy az egyes lámpakombinációk adott ideig forduljanak elő.

Megoldás:

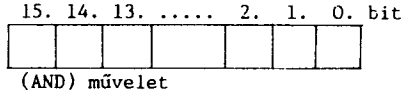
A párhuzamos kapu (ld. A HT-1080Z/64 kivezetései) R15-ös regisztere fogja vezérelni a lámpákat. Az első lámpához tartozik a 0.,1.,2. bit, a 2. lámpához pedig a 3.,4.,5. bit. A fenti sorrend megfelel a lámpák PIROS,SÁRGA,ZÖLD sorrendjének. Ha egy bit 0, akkor az adott lámpa világít, ha 1, akkor pedig nem világít. A program első két sora a megfelelő regiszter kiválasztását végzi, részletesen a "Hang és zene generálás" című fejezetben lesz róla szó.

```
10 OUT 31,7 : OUT 30,225 : OUT 31,15
20 J=255-1-32 : T=3000 : GOSUB 100 : REM PIROS, ZÖLD
30 J=255-1-16 : T=800  : GOSUB 100 : REM PIROS, SÁRGA
40 J=255-3-8  : T=800  : GOSUB 100 : REM PIROS-SÁRGA, PIROS
50 J=255-4-8  : T=3000 : GOSUB 100 : REM ZÖLD, PIROS
60 J=255-2-8  : T=800  : GOSUB 100 : REM SÁRGA, PIROS
70 J=255-1-24 : T=800  : GOSUB 100 : REM PIROS, PIROS-SÁRGA
80 GOTO 20
100 REM LÁMPA+VÁRAKOZÁS:
110 OUT 30,J
120 FOR I=0 TO T : NEXT I
130 RETURN
```

2.16 BITKEZELÉS

A **HT-1080Z/64** logikai műveleteit **(AND, OR, NOT)** fel lehet használni arra, hogy egész számok bitjei között végezzünk **bitműveleteket**. Ezek elvégezhetők nem csak egész típusú operandusok között is, de ilyenkor a műveletek az egészre konvertált értékekkel történnek meg (ha a konverzió egyáltalán lehetséges).

Egy egész szám felosztása:



Az **és** (AND) művelet

kifejezés **AND** kifejezés

hatására az eredmény azon bitje lesz 1-es, amely mindkét operandusban 1-es volt.

Példa: PRINT 7 AND 21
eredménye 5 lesz, mert 7 = 0000000000000111
21 = 0000000000010101
AND ——————————
0000000000000101 = 5

A **vagy** (OR) művelet esetén az eredmény azon bitje lesz 1-es, amely legalább az egyik operandusban 1-es volt:

kifejezés **OR** kifejezés

Példa: PRINT 8 OR 17
eredménye 25 lesz, mert 8 = 0000000000001000
17 = 0000000000010001
OR ——————————
0000000000011001 = 25

A harmadik művelet, a **tagadás** (NOT) esetén az eredmény azon bitje lesz 1-es, amely az operandusban 0 volt:

NOT kifejezés

Példa: PRINT NOT 30
eredménye -31 lesz, mert 30 = 0000000000011110
NOT ——————————
1111111111100001 = -31

Ugyanis a bitenkénti negálással a szám un. 1-es komplemens kódját kapjuk. A negativ számokat pedig 2-es komplemenssel ábrázoljuk, amelyeket úgy kapunk, hogy a negálás után a kapott számhoz még egyet hozzáadunk. Így a **NOT X** jelentése mindig **-(X+1)**.

Az egyes mérő-, vezérlő eszközök kezeléséhez általában 8 bit átvitelére van szükség, amelyeket gyakran külön-külön kell értelmezni. A bitkezeléssel megoldható ez a feladat.

Példák:

A kazettás magnó, a hanggenerátor, s egyéb eszközök vezérlésekor gyakran van szükség arra, hogy csak bizonyos biteket állítsunk be, illetve kérdezzünk le, ezt a következőképpen tehetjük meg (csak az alsó 8 bitet használjuk):

- A 3. bit kiolvasása V%=(X%AND 8)
  Megjegyzés: V%=8 vagy V%=0 lesz.

- A 2.-4. bitek kiolvasása V%=INT((X%AND 28)/4)
  Megjegyzés: 28 = 16 + 8 + 4. A néggyel osztás miatt V% értéke 0 és 7 közötti egész szám lesz.

- A 2. bit 1-re állítása X%=(X%OR 4)

- A 2. bit 0-ra állítása X%=(X%AND 251)
  Megjegyzés: 251 = 255 - 4, ahol 255 a csupa 1-est tartalmazó byte.

- A 3. bit beállítása adott Y% értékre
  X%=(247 AND X%) OR Y%*2^3

  Megjegyzés: A beállításhoz először a megfelelő bitet kell nullázni, majd csak utána lehet beállítani.

Gyakran van szükség a **HT-1080Z/64**-en nem szereplő logikai műveletekre is, ezeket a következőképpen lehet megvalósitani:

Kizáró vagy (A XOR B): (A AND NOT B) OR (NOT A AND B)

Implikáció (A IMP B): NOT A OR B

Feladat: Egy adott szöveget írjunk ki úgy, hogy a kisbetűket mindenhol a megfelelő nagybetűkkel helyettesítjük. (Az átalakítandó szövegben csak betűk szerepelnek.)

Megoldás: A kisbetű kódja az 5. bitben tér el a megfelelő nagybetűétől, pl. A = 65, a = 97. Ezt felhasználva a következő egyszerű megoldást kaphatjuk.

```
100 A$="Orion"
110 B$=""
120 FOR I=1 TO LEN(A$)
130   B$=B$+CHR$(ASC(MID$(A$,I,1))AND 95)
140 NEXT I
150 PRINT B$
```

A program eredményül az ORION szöveget fogja kiírni.

2.17 KÖZVETLEN TÁRCÍMZÉS

Bizonyos esetekben szükség lehet a számítógép memóriájának közvetlen címzésére. Ez legtöbbször mérési, vezérlési feladatoknál, a tár közvetlen használatánál merül fel. Ehhez a BASIC nyelv két alapvető lehetőséget, továbbá néhány segédfunkciót biztosít. Az első: **egy byte elhelyezése a memóriában.**

sorszám **POKE** cím,kifejezés

Címként egy egész értékű kifejezés adható meg. A **HT-1080Z/64**-en a legnagyobb cím a 65535, ezt azonban egyszerűen nem lehet elérni.

Példa: POKE 65535,0
hatására a program megáll ?OV Error hibaüzenettel, ugyanis a 65535 szám nem ábrázolható a **HT-1080Z/64**-en egész számként (a legnagyobb pozitiv egész szám a 32767).

A megoldás ilyenkor a következő: Meg kell találni azt a 2-es komplemens kódban ábrázolt értéket, amely - ha pozitiv számként értelmeznénk, akkor pontosan a 65535 lenne. Ez az érték a -1. Egy kis segítség az ilyen értékek meghatározásához:

Példa: a hibás POKE 65000,65 helyettesíthető vagy a
POKE 65000-65536,65
utasítással vagy pedig a
POKE -536,65
utasítással.

Az elhelyezett érték csak 0 és 255 közötti egész lehet,egyébként ?FC Error hibajelzést kapunk.

Példák: Egy egész típusú változó (A%) tartalmának elhelyezése adott címtől (C%) kezdve.

```
10 POKE C%,A%-INT(A%/256)*256 : REM ALACSONYABB HELYIÉRTÉKŰ BYTE
20 POKE C%+1,INT(A%/256) : REM A MAGASABB HELYIÉRTÉKŰ BYTE
```

A "VEGA" szöveg kiírása a képernyőre:

```
10 C%=15360 : REM KÉPERNYŐ TÁR KEZDŐCÍME
20 POKE C%,86 : POKE C%+1,69
30 POKE C%+2,71 : POKE C%+3,65
```

Egy byte kiolvasása a memória adott címéről a következő függvénnyel történhet:

PEEK (cím)

Példák: A C% címre elhelyezett egész szám olvasása A% változóba.

```
10 A%=PEEK(C%)+PEEK(C%+1)*256
```

Várakozás, amíg nem nyomják le az egyik szám-billentyűt.

```
40 A=PEEK(14352) OR PEEK(14368) AND 3
50 IF A=0 THEN 40
```

Megjegyzés: A billentyűzetnek megfelel egy címtartomány, ahol a számjegy billentyűk lenyomását a 14352 címen, illetve a 14368 cím 2 bitjén észlelhetjük. (Ld. 3.3. Fejezet.) Az **AND** művelet prioritása nagyobb, mint az **OR** műveleté, ezért a kifejezés kiértékelésekor ezt kell előbb elvégezni.

Gyakran szükség lehet a **változó memóriabeli helyének lekérdezésére** (főleg gépi kódú részt is tartalmazó programoknál):

VARPTR (azonosító)

A függvény kiszámítja az argumentumaként megadott változó vagy tömbelem tárbeli címét. Ettől a címtől kezdve helyezkedik el az értéke a memóriában. A különböző típusú változók más-más hosszal tárolódnak (ld. adatábrázolás), kezelésükről már a programnak kell gondoskodnia. Használatával nagyon vigyázni kell, ugyanis a változók címe nem állandó, a BASIC értelmező futás közben átrendezheti (más tárcímre helyezheti) a változókat.

Példa:

```
10 DIM A(10)
20 Z=VARPTR(A(0))
30 C=1
40 POKE Z,65
```

hatására nem az A(0) értéke lesz 65, mivel a C változó megjelenésekor a már definiált tömböket az értelmező a memóriában hátratolja (a Z változó már a **VARPTR** előtt megjelent).

A memóriában levő **szabad terület** (byte-ok) **lekérdezésére** két módszer is van (a BASIC rendszerváltozók használatán kívül):

MEM
FRE (szám kifejezés)

Mindkét függvény megadja a memóriában fel nem használt terület hosszát byte-okban.

Példa:

```
10  PRINT MEM
100 PRINT FRE(0)
```

2.18 GÉPI NYELVŰ SZUBRUTINOK

Ha egy programot a BASIC nyelvben csak bonyolultan, vagy egyáltalán nem lehetne megírni, akkor szükség lehet gépi nyelvű szubrutin írására.

A továbbiak megértése feltételezi a Z80 mikroprocesszor gépi kódú programozásának ismeretét. E fejezetben csak megemlítjük, hogy egy BASIC program milyen módon képes aktivizálni egy ilyen szubrutint,amelyet **HT-1080Z/64** esetén Z80-as gépi kódban kell megírni.
A BASIC programból egy függvény segítségével tudjuk meghívni a gépi kódú szubrutint.

USR (változó)

A szubrutin kezdőcímét a függvény hívása előtt a **POKE** utasítással a **16526**-os és a **16527**-es címre kell írnunk. A kezdőcím alacsonyabb helyiértékű részét a 16526-os címre kell beírni.

Paraméter átadása a szubrutinnak

Ha a függvény argumentumát át akarjuk adni a szubrutinnak, akkor a gépi nyelvű szubrutint egy **CALL OA7FH** (hexadecimális szám, decimálisan 2687) Z80 utasítással kell kezdenünk. Az argumentum értéke a HL regiszterpárba kerül. Az argumentum értéke **egész** kell, hogy legyen (-32768 - +32767)!

Példa:

```
10 POKE 16526,16 : POKE 16527,127 : REM 7F10H
20 A=125
30 B=USR(A)
```


Visszatérés a BASIC programba

1. érték visszaadás nélkül a **RET** Z80 utasítással,
2. a **JP OA9AH** (decimálisan 2714) Z80 utasítással, ekkor a **HL** regiszter tartalmát kapja értékül a változó.

Gépi kódú szubrutin írása

1. A **MONITOR** módban hexadecimális számokkal (lásd 2.19. Fejezet).
2. A BASIC programban elhelyezve **DATA** utasításokban, onnan beolvasva **READ** utasítással, majd **POKE** utasítással a megfelelő memória címre letéve.

Feladat:

Írjunk egy olyan gépi kódú szubrutint, ami a sorszámos **RESTORE**-t valósítja meg.

Megoldás:

```
10  POKE 16526,0 : POKE 16527,127 : REM KEZDŐCÍM=7F00H
20  DATA 17,32511        : REM DB.,KEZDŐCÍM-1
30  DATA 205,127,10      : REM CALL OA7FH
40  DATA 235             : REM EX   DE,HL
50  DATA 205,44,27       : REM CALL 1B2C
60  DATA 11              : REM DEC  BC
70  DATA 121             : REM LD   A,C
80  DATA 50,255,64       : REM LD   (40FF),A
90  DATA 120             : REM LD   A,B
100 DATA 50,0,65         : REM LD   (4100),A
110 DATA 201             : REM RET
120 READ DB,KC
130 FOR I=1 TO DB : READ A : POKE KC+I,A : NEXT I
140 STOP
```

Megjegyzés:

A gépi kódú szubrutin utasításai decimális számként a **DATA** utasításban vannak megadva.

A gépi kódú programhoz szabad területet tudunk biztosítani a bekapcsoláskor kiírt **MEMORY SIZE?** után beírt számmal.

Példa: 32512, itt kezdődik a gépi kódú szubrutin. Ekkor csak eddig a határig lehet írni BASIC programot.

Feladat:

Irjunk egy programot ami használja a sorszámos RESTORE-t! Most tulajdonképpen a RESTORE utasítás helyett a USR függvényt alkalmazzuk!

Megoldás:

```
10 DATA 10
20 DATA 20
30 DATA 30
40 X=USR(30) : READ A : PRINT A
50 X=USR(20) : READ A : PRINT A
60 X=USR(10) : READ A : PRINT A
```

Megjegyzés:

Ne felejtsük el a program futása előtt a 16526-, 16527-es címekre a gépi kódú szubrutin kezdőcímét megadni!

2.19 SYSTEM - MONITOR

A HT-1080Z/64 személyi számítógép speciális funkciókat is tartalmaz. Ezek segitségével megvizsgálhatjuk a memória tartalmát (MONITOR), gépi kódú programokat tölthetünk be kazettáról, betöltés után futtathatjuk őket, valamint a BASIC értelmezőt bővítve újabb szolgáltatásokat kapunk (újrasorszámozás, ékezetes betűk, stb.).

A speciális funkciók aktivizálása:

sorszám SYSTEM

A SYSTEM parancs hatása a következő:

Egy üres sor kiírása után a sor elején megjelenik egy csillag és egy kérdőjel (*?), és a speciális funkciókat kiválaszthatjuk.

Példa:

```
>SYSTEM
     egy üres sor
*? funkció választás
```

Speciális funkciók

- BASIC bővítés

A bővítés aktivizálása:

A *? után /-el megadott decimális számmal engedélyezzük a bővítést.

Példa:

*? /12288

A BASIC kiterjesztés (bővítés) érvényre juttatása után törli a képernyőt, majd a következő üzenettel jelzi a bővítést:

BASIC ROM EXTENSION 2.51

A BASIC bővítés a következő lehetőségeket nyújtja:

cím	tevékenység
12288	Újrasorszámozás (RE), villogó kurzor, kisbetű használata, ismétlési funkció, ékezetes betűk
12299	Újrasorszámozás, kisbetű használata, ismétlési funkció,ékezetes betűk
12294	Kisbetű használata

Megjegyzés: A 12294-es címtől indítva nem törli a képernyőt és nem írja ki a fenti üzenetet.

- MONITOR

Belépés a monitorba:

A per-jel (/) után írt 12710 decimális számmal tudjuk aktivizálni a monitort. Ennek hatására törli a képernyőt, majd kiírja a regiszterek tartalmát - ALAPÁLLAPOT - (a Z80 processzor regiszterei).

A monitor lehetővé teszi Z80 gépi kódú programok beírását (hexadecimális számjegyek formájában), végrehajtását, a memória tartalmának megjelenítését, módosítását stb.

A MONITOR

PARANCS	LEÍRÁSA
B	: visszatérés a BASIC-be.
Dnnnn	: memória tartalmának megjelenitése a képernyőn hexadecimális formában. Az nnnn egy hexadecimális címet jelent. Először törli a képernyőt, majd a megadott címtől kezdve kiírja 16 byte tartalmát. A megjelenítést vezérelhetjük lefelé mutató nyíllal, ekkor az nnnn címtől kezdve a rákövetkező magasabb című memória tartalmakat írja ki a képernyőre. Felfelé mutató nyíl esetén pedig a kisebb című tár tartalmakat listázza. Bármilyen más billentyű lenyomására befejezi a megjelenítést és visszatér az alapállapotba.
Mnnnn	: A memória tartalmát módosithatjuk az adott címtől (nnnn). A parancs törli a képernyőt, és az első sorba kiírja a módosítandó memória címét, melléje a tartalmát hexadecimális formában.Ezután tudjuk a tartalmat módosítani egy hexadecimális számpárral. A két szám beírása után rögtön megjelenik a következő cím és mellette a tartalma. A módosítás végét az X billentyű leütésével jelezzük!
R	: A paranccsal a Z80 regisztereit módosithatjuk. A képernyőt törli, majd megjelennek a regiszterpárok tartalmai (először mindig az IY regiszterpár) hexadecimális formában. A megjelenés után sorban megváltoztathatjuk a regiszterpárok tartalmait egy négyjegyű hexadecimális számmal. Ha valamelyik regiszterpárt nem akarjuk változtatni, akkor az X billentyű lenyomásával a következő regiszterpár módosítására ugorhatunk. A módosítás befejezése az utolsó regiszterpár - PC - módosítása után történik.
Gnnnn	: A parancs hatására az adott címtől (nnnn) kezdődő gépi kódú programot végrehajtja. Ha szeretnénk a gépi kódú program végén visszatérni a monitorba, akkor utolsó utasításnak írjuk a következőt: JP 3347H (C3 47 33 hex. formában).

Gnnnn,tttt : Ugyanaz mint az előző parancs. Itt azonban a tttt cím egy töréspontot jelent, aminek elérésekor visszatér a monitorba. A végrehajtás elején generál egy töréspontot a tttt címre (CALL 3347H Z80 utasítás), elmentve az ott levő 3 byteot. Ha a futás során valamilyen hiba fordult elő, akkor ez az utasítás ott marad a tttt címen. A visszaállítás az M paranccsal történhet, természetesen a RESET, valamint a SYSTEM után!

Megjegyzés:

A monitor parancsok után nem kell a NEW LINE billentyűt használni!

- Gépi kódú program betöltése kazettáról

A *? után annak a gépi kódú programnak a nevét kell megadni, amit a kazettáról akarunk betölteni. A név maximum hat karakterből állhat.

Példa:

```
SYSTEM

*? GEPI
*?      ha ez megjelent, a betöltés sikerült!
```

A betöltéshez a magnót lejátszásra kell állítani. A betöltés a BASIC programhoz hasonlóan megy végbe (lásd 1.3. Fejezet). Hibajelzés esetén (a képernyő jobb felső sarkában megjelenő két csillag közül a bal oldali helyére egy C betű kerül) meg kell ismételni a betöltést.

- Gépi kódú program végrehajtása (SYSTEM parancs esetén)

A gépi kódú program végrehajtása történhet:

1. A MONITOR aktivizálása után a G paranccsal. Azonban ehhez tudni kell a gépi kódú program indítási címét.
2. A sikeres betöltés után rögtön indíthatjuk a gépi kódú programot egy /-rel.
3. A SYSTEM parancs kiadás után, megadjuk a gépi kódú program indítási címét decimálisan egy per-jel (/) után;

Példa:

```
SYSTEM
   1 üres sor (a gép írja ki!)
*? /32005
```

2.20 HANG ÉS ZENE GENERÁLÁSA

A HT-1080Z/64 beépített hanggenerátora lehetővé teszi jóminőségű zenei hanghatások létrehozását. Ezek a hangok elsősorban figyelemfelkeltésre, a grafikai hatások fokozására alkalmasak.

A hang előállításában résztvevő egységek:

- Hanggenerátor,
- Zajgenerátor,
- Keverő,
- Amplitudó-vezérlő,
- Burkológörbe-generátor,
- Digitális-analóg átalakítók.

A hanggenerátor három csatornás. A csatornákat jelöljük A,B,C-vel! A három csatorna tiszta hangot állít elő. Mindhárom csatorna külön programozható.

A zajgenerátor zajokat állít elő.

A keverő a hanggenerátor csatornáinak és a zajgenerátornak a kimenő jeleit összegzi.

Az amplitudó-vezérlő szolgáltatja a kimenő jel burkológörbéjét. Ez lehet állandó vagy változó amplitudójú.

A burkológörbe-generátor állítja elő a változó burkológörbe mintákat.

A digitális-analóg átalakítók szolgáltatják a hanggenerátor kimenő jelét.

A hanggenerátort 14 regiszter segitségével programozzuk.

A továbbiakban a regiszterekre egységesen, mint a hanggenerátor regisztereire hivatkozunk, függetlenül attól, hogy melyik szerkezeti egységben vannak.

A hanggenerátor regiszterei 8 bitesek. Ebből következik, hogy 0 és 255 közötti számokat tartalmazhatnak, de nem minden regiszternél van értelmezve a teljes intervallum.

A hanggenerátor egyes regisztereinek funkciója:

- R0 A csatorna hangmagasság finom szabályozása,
- R1 A csatorna hangmagasság durva szabályozása,
- R2 B csatorna hangmagasság finom szabályozása,
- R3 B csatorna hangmagasság durva szabályozása,
- R4 C csatorna hangmagasság finom szabályozása,
- R5 C csatorna hangmagasság durva szabályozása,

- R6 a zajgenerátor "frekvenciájának" beállítása,

- R7 a keverő vezérlés és a csatornák engedélyezése,

- R8 A csatorna hangerő szabályozása,
- R9 B csatorna hangerő szabályozása,
- R10 C csatorna hangerő szabályozása,

- R11 burkológörbe-generátor periódusidő finom szabályozása,
- R12 burkológörbe-generátor periódusidő durva szabályozása,

- R13 burkológörbe hullámforma kiválasztása.

- R14 és R15 ebben az egységben vannak, de funkciójuk más.

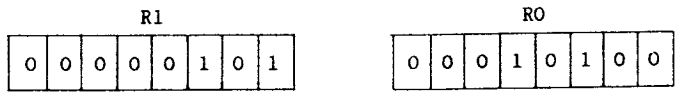
Az egyes regiszterekbe

OUT 31%,regiszter-sorszám: OUT 30%,tartalom

utasítás-párokkal írunk.

R0-R5 A hangmagasság beállítása: Egy-egy csatorna hangmagassága 2-2 regiszterrel szabályozható. Egy-egy teljes 8 bites regiszterrel (R0, R2,R4) és egy-egy 8 bites regiszter (R1,R3,R5) 4 bitjével. Ez összesen 0...4095, azaz 4096 különböző hangmagasság beállítását teszi lehetővé. Rajzon szemléltetve például az A csatorna így fest:

Az x-el jelölt bitek értéke a hangmagasság beállítása szempontjából közömbös. Ha például 1300-as hangmagasságot állítunk be az A csatornán a regiszterek így lesznek feltöltve:



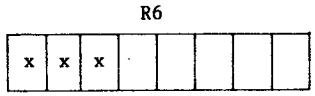
Képlettel kifejezve, ha H Hertz-es magasságú hangot akarunk kiadni valamelyik csatornán, akkor a megfelelő regiszterekbe

```
P=FIX(109166/H+.5)
```

értéket kell betölteni. Ha az érték 256-nál kisebb, akkor a durva szabályozás regiszterébe 0-t, a finomszabályozás regiszterébe P-t kell írni. Egyébként a finomszabályozás regiszterébe `(P AND 255%)`-t a durva szabályozásába `(CINT (P/256) AND 15%)`-ot.

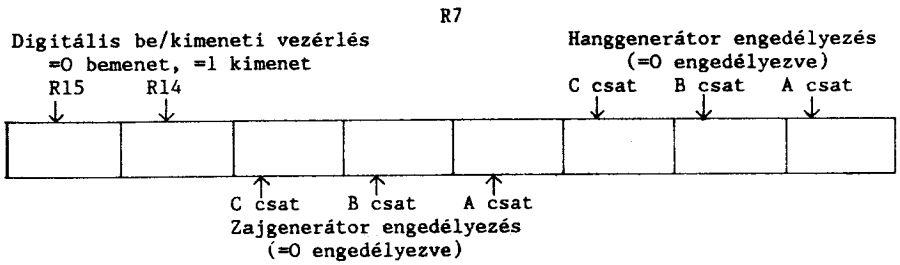
A regiszterek elnevezése - finom, ill. durva szabályozás - onnan ered,hogy a durva szabályozás értékét kicsit megváltoztatva a hangmagasság nagyot változik, ezzel ugyanis - mint az ábrákból látható - a hangmagasságot megadó számérték magasabb helyiértékeit változtatjuk.
Mivel a regiszterek tartalma a jel periódusidejét szabályozza, ezért a regiszterek nagyobb értékeihez mélyebb hangok tartoznak.

R6 A zajgenerátor "frekvenciájának" beállítása: a regiszterbe 0 és 31 közötti értékek írhatók.



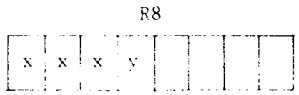
Az x-el jelölt bitek értéke közömbös.

R7 Keverő vezérlés, bemeneti/kimeneti csatorna engedélyezés: az egyes hang csatornák egyéb paramétereit hiába állítjuk be, míg a csatorna működését nem "engedélyezzük", nem hallunk hangot. Erre való az R7 regiszter.



(A hanggenerátor adhat hangot akkor is, ha egyik csatorna sincs engedélyezve, de nem állandó hangerősségű hangot állítunk be - R8,R9 vagy R10-ben az y-nal jelölt bit 1-re van állítva - és periódusidő valamint hullámforma be lett állítva.)

R8-R10 Hangerőszabályozás: ezek a regiszterek szabályozzák az egyes csatornák hangerejét. Például az A csatornánál:

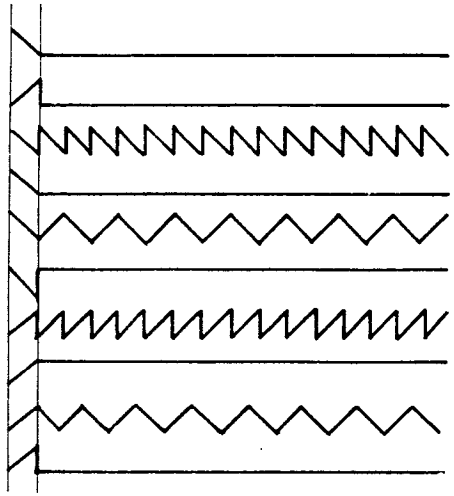


Az x-szel jelölt bitek értéke a hangerő beállítása szempontjából közömbös. Ha az y-nal jelölt bit értéke 1 akkor a hangerőt szabályozó bitek értéke közömbös. Ebben az esetben ugyanis a kimenő jel amplitudóját a burkológörbe generátor szabja meg.
A hangerő értéke 0 és 15 között lehet. A nagyobb érték nagyobb hangerőnek felel meg.
Ha a hangerőt 0-ra állítjuk be, akkor az illető csatorna nem ad ki tiszta hangot.

R11-R12 A burkológörbe-generátor frekvenciájának szabályozása: ha az R8,R9 R10 regiszterek valamelyikében az y-nal jelölt bit értéke 1, akkor van értelme a burkológörbe frekvenciáját szabályozni. A szabályozás az R11,R12 regiszterekkel történik.

R12 a frekvencia durva szabályozója, R11 a frekvencia finom szabályozója.

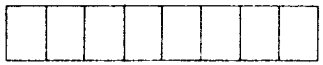
R13 A burkológörbe hullámformájának kiválasztása: ha az R8,R9,R10 regiszterek valamelyikében az y-nal jelölt bit 1, akkor van értelme a burkológörbe alakját megadni. Egyébként a hang állandó amplitudójú lesz.

R13				Decimális	Hullámforma
x	x	x	x		
0	0	x	x	0-3	
0	1	x	x	4-7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	10	
1	0	1	1	11	
1	1	0	0	12	
1	1	0	1	13	
1	1	1	0	14	
1	1	1	1	15	

Az x-szel jelölt bitek értéke a burkológörbe hullámformája szempontjából közömbös.

Most néhány példával segítséget szeretnénk nyújtani a hanggenerátor programozásához.

Fent gyakran jelöltük a hanggenerátor egy-egy regiszterét így:



Most az egyes helyiértékekhez - az egyes bitekhez - beírjuk, hogy milyen számértéket képviselnek.

128	64	32	16	8	4	2	1

Ennek segítségével például így állíthatjuk be a hanggenerátor egy-egy regiszterét: - engedélyezzük az A hangcsatorna működését, tiltsuk a többi hangcsatorna és a zajgenerátor működését! Az R7 regiszternél közölt ábra szerint ehhez a regiszter összes bitjét 1-re kell állítani, kivéve a legutolsót. A regiszterbe tehát a 2+4+8+16+32+64+128=254 értéket kell beírni,

- az A csatornát állítsuk be úgy, hogy ne állandó amplitudójú hangot adjon. Az ábra szerint ehhez az y-nal jelölt bitet kell 1-re állítani. A fent bemutatott számértékek segítségével megállapítható, hogy az y-nal jelölt bit értéke 16. Az R8 regiszterbe tehát 16-ot kell írni,

- a burkológörbe alakjául válasszuk az R13 regiszternél bemutatott jelformák közül a 4.-et. Ehhez a jobb szélső és a jobbról számított 4. bitet kell 1-re állítani. Leolvasható, hogy az R13 regiszterbe 1+8=9 értéket kell beírni.

A következő BASIC program figyelemfelhívó hangot ad.

```
10 OUT 31,0: OUT 30,120:REM A CSATORNA HANGMAGASSÁG BEÁLLITÁSA
20 OUT 31,7: OUT 30,62: REM CSAK AZ A CSATORNA MŰKÖDIK
30 OUT 31,8: OUT 30,16: REM NEM ÁLLANDÓ HANGERŐ BEÁLLÍTÁSA
40 OUT 31,12:OUT 30,16: REM BURKOLÓGÖRBE FREKVENCIA BEÁLLÍTÁSA
50 OUT 31,13:OUT 30,9:  REM BURKOLÓGÖRBE HULLÁMFORMA VÁLASZTÁSA
```

A fenti program egy hangimpulzust ad.Ha az 50-es sorban az R13 regiszterbe 9 helyett 8-at írunk, akkor periódikusan ismétlődő hangot hallunk.

Példa: Az alábbi példaprogrammal zenei hangok állíthatók elő.

```
10 CLS: CLEAR 1000
20 OUT 31%,7%:OUT 30%,254%:OUT 31%,8%:OUT 30%,15%
30 D%=8%
40 DATA C,208,D,185,E,165,F,155,G,139,A,124,H,110,c,104," ",0
50 S$=STRING$(64%,CHR$(191%)):PRINT@64%*8%,S$;:PRINT@64%*12%,S$;
60 FOR I%=0%TO 63%STEP 7%
70 FOR J%=9%TO 11%
80 PRINT@64%*J%+I%,CHR$(191%);
90 NEXT J%
100 NEXT I%
110 RESTORE
120 FOR I%=0%TO D%
130 READ T$,T%
140 PRINT@64%*10%+7%*I%+4%,T$;
150 NEXT I%
160 W$=" ":T$=W$:I%=D%
170 PRINT@64%*10%+7%*I%+4%,W$;
180 IF W$=" "THEN W$=T$ ELSE W$=" "
190 H$=INKEY$:IF H$=" "THEN 170 ELSE PRINT@64%*10%+7%*I%+4%,T$;
200 RESTORE
210 FOR I%=0TO D%
220 READ T$,T%:IF H$=T$ THEN OUT 31,0:OUT 30,T%:GOTO 170
230 NEXT I%
240 PRINT@64%*15%,CHR$(30%);"ISMERETLEN HANG.ÜSSÖN LE EGY BILLENTYÚT!";:OUT 31%,0%:OUT 30%,0%:IF INKEY$="" THEN 240 ELSE PRINT@64%
*15%,CHR$(30%);:GOTO 190
```

A 20-as sor engedélyezi az "A" hangcsatorna működését, és a hangerejét 15-re állítja be.
A 40-es sor a hangokhoz tartozó betűket és a megfelelő hangmagasságokat adja meg.
A 60...100-as sorszámú sorok kirajzolják a "billentyűket".
A 120-150-es sorszámú sorok beolvassák a hangokat jelölő betűket, és a "billentyűkre" ráírják.
A 170-es sorszámú sor írja ki a leütött hangot jelölő betűt.
A 180-as sorszámú sor váltakozva szóközt ill. a megfelelő betűt teszi a kiírandó hang nevébe. Ez okozza, hogy a leütött hangot jelölő betű villog.
A 190-es sor olvassa be a hangot jelölő betűt. Mindaddig, amíg nem adnak meg újabb hangot, a vezérlés a villogtatásra adódik vissza. Ha leütnek egy betűt, akkor az eddig kiadott hang betűjét a megfelelő helyre kiírja, nehogy a villogtatás miatt a szóköz maradjon ott.
A 210...230-as sorok megkeresik a leütött jelet az ismert hangokat jelölő betűk között. Ha megtalálja, ki is adja a hangot.
A 240-es sor hibajelzést ad, ha a leütött billentyű nem jelöl ismert hangot.

2.21 HIBAKEZELÉS

A számítógépes feladatmegoldás során előbb-utóbb eljutunk egy olyan állapotba, amikor rendelkezésünkre áll a program BASIC programozási nyelven leírt szövege. (Idáig eljutni persze nem egyszerű dolog, hiszen előtte el kell döntenünk a megoldás módját, azaz el kell készítenünk a problémamegoldó algoritmust.) Futás során azonban bekövetkezhetnek olyan események, amelyek veszélyeztethetik vagy rosszabb esetben megakadályozhatják a program további működését. Gondoljunk csak például a következő hibákra:

1. szintaktikus hibára, általában elírásból ered

2. nem megengedett indexkifejezésekre, egy tömb olyan indexű elemére hivatkoztunk, ami nem megengedett, a lehetségesnél nagyobb vagy negatív értékű

3. aritmetikai kifejezések túl-, illetve alulcsordulására, a túl kis számból nulla lesz és csak a túl nagy esetén kapunk hibajelzést

4. nullával való osztásra, negatív számból való négyzetgyökvonásra,

5. kazetta hibára, nem értelmezhető karaktert olvas,

és még folytathatnánk a sort. Ilyen esetekben a programunk futása megszakad - mivel a további végrehajtásnak nincs értelme - és a képernyőre egy **hibajelzés** íródik ki. Ezeket a műveleteket a BASIC értelmezőbe épített hibafigyelő rutinok végzik el. Hiba esetén ezek a rutinok aktivizálódnak, kiírják az észlelt hiba karakteres kódját, és megállítják a programot. Vannak bizonyos esetek, amikor célszerűbbnek látszik a hibát feldolgozni és a programunkat tovább futtatni. Ehhez az szükséges, hogy átvegyük a hibarutin szerepét és mi intézkedjünk a hibakezelésről. Ilyenkor több lehetőségünk lehet: vagy kijavítjuk a hibát - például a túl nagy tömbindexet a megfelelő jó értékre állítjuk - és tovább folytatjuk programunk végrehajtását, vagy olyan hiba következett be, amire nem készültünk fel, ekkor egy értelmes - a hiba típusát megfelelően jól mutató - kiírással befejezzük tényked esünket. Így jól használható lehetőségünk van biztonságos - a felhasználó tudatlanságából, véletlen elírásaiból fakadó, hibákat kezelni tudó - programok készítésére.

Vizsgáljuk meg, mi módon tudjuk rávenni programunkat a rendszer védekezésének hatástalanítására, illetve saját hibarutinjaink beépítésére. Erre az **ONERRORGOTO** utasítás szolgál, alakja:

sorszám **ONERRORGOTO** sorszám1

Példa: 10 ONERRORGOTO 100

Ha a program ezt az utasítást végrehajtja, a saját hibakezelésünk bekapcsolódik - azaz hiba esetén a vezérlés a sorszám1-gyel kezdődő programsorra adódik (ilyennek persze lennie kell) - és elvégezhetjük a hiba feldolgozását. A hibák elemzéséhez szükségünk van arra, hogy a bekövetkezett hibát azonosítani tudjuk:

1. Egyrészt ismernünk kell a hiba típusát, vagyis a hibakódját,ezt az **ERR** függvény értékéből, transzformációval kaphatjuk meg

A bekövetkezett hiba kódja:

ERR/2+1

Példa: 100 IF ERR/2+1 = 4 THEN GOTO 1000

Minden - a rendszer által jelzett - kétbetűs hibaüzenethez tartozik egy kódszám, melyet a 3.7. Fejezetben találhatunk meg.

2. Másrészt szükségünk lehet annak a sornak a sorszámára, melyben a hiba bekövetkezett. Ezt az **ERL** függvény értékéből tudhatjuk meg.

Annak a sornak a sorszáma, ahol a hiba bekövetkezett:

ERL

Példa: 200 IF ERL=40 THEN GOTO 2000

Ha úgy döntünk, hogy a hiba elemzése és esetleges javítása után programunk fusson tovább, lehetőséget kell adnunk a folytatási hely - milyen sorszámú sorra kerüljön a vezérlés - megadására. Itt nem használhatjuk **sem a RETURN, sem a GOTO** utasítást, ilyenkor a **RESUME** utasítást kell használnunk:

sorszám **RESUME** helymegadás

Az előbbi állításunk fordítva is igaz, tehát **RESUME** utasítás csak végrehajtott **ONERRORGOTO** utasítás után adható ki. A **RESUME** utasításnak többféle alakja is használható, a hiba jellegétől, illetve súlyosságától függően:

1. a helymegadás elmaradhat: ekkor annál az utasításnál folytatódik a vezérlés, ahol a hiba bekövetkezett, vagyis mégegyszer végrehajtódik a hibát okozó sor, emiatt ezt csak nagyon gondos hibaelemzés, illetve hibajavítás után célszerű használni,a végtelen ciklus elkerülése érdekében.

Példa: 500 RESUME

2. ha a helymegadás a **NEXT** szó - nem keverendő a ciklust záró **NEXT** utasítással -, akkor a program futása a hibát okozó utasítást követő utasítással folytatódik.

Példa: 500 RESUME NEXT

3. ha a helymegadás egy sorszám, akkor a megadott sorszámú sornál folytatódik a végrehajtás.

Példa: 500 RESUME 1500

Ezek használatával kapcsolatban FONTOS megjegyezni a következőket: az átsorszámozás (**RE** paranccsal érhető el) nem változtatja meg az **ERL**-lel relációban levő, illetve a **RESUME** utasításban szereplő utasítás-sorszámot. Ezeket kézzel kell hozzáigazítani az új sorszámokhoz.

Az általunk saját hibarutinnal lekezelt programrész végrehajtása után célszerű a hibafigyelést az értelmezőre bízni. Ha nem ezt tesszük, az esetlegesen bekövetkező későbbi hibák esetén is ez a rutin aktivizálódik, ami nem szerencsés. (Gondoljunk csak arra, amikor olyan hiba következett be, amiről a saját hibakezelő eljárásunkban nem gondoskodtunk. Ilyenkor egyrészt fölöslegesen hajtódik végre ez a rész, rosszabb esetben még félrevezető is lehet a kiírt szöveg.) Sőt, az **ONERRORGOTO** futás után is érvényben marad! A saját hibakezelés kikapcsolására az **ONERRORGOTO O** utasítás szolgál:

sorszám **ONERRORGOTO O**

Ennek az utasításnak a használatával lehetőségünk van több saját hibarutin készítésére, segítségével szabályozhatjuk, hogy mindig a megfelelő hibakezelő eljárásunk legyen érvényben.

Lehetőségünk van arra is, hogy a hibakezeléshez készített eljárásainkat egyszerű módon tesztelhessük (eldönthessük hogy helyesen működnek-e). Ehhez nyújt segítséget az **ERROR** utasítás:

sorszám **ERROR** hibakód

Egy ilyen utasítás végrehajtásakor pontosan ugyanaz történik, mint olyankor, amikor az adott kódú hiba bekövetkezett, vagyis megjelenik a képernyőn a hibakódnak megfelelő kétkarakteres hibaüzenet.

Példa: 100 ERROR 1

végrehajtásakor a képernyőn az **?NF error in 100** üzenet jelenik meg, feltéve, hogy nincs érvényben **ONERROR-GOTO** utasítás, amennyiben ezen utasítás végrehajtása előtt egy **ONERRORGOTO** utasítás végrehajtódott, akkor természetesen az ott definiált módon folytatódik a végrehajtás.

Példa: legyen a feladatunk a következő: be kell olvasnunk egy számot, és ki kell számítanunk a természetes alapú logaritmusának a négyzetgyökét. Készítsük úgy el programunkat, hogy bizonyos hibákat saját magunk kezelünk le. Ezek a következők legyenek:

1. túl nagy számot olvastunk be
2. mivel a logaritmus-függvény 0-ra és negatív számokra nincs értelmezve, ezért ezt is figyeljük
3. ha a beolvasott szám 0 és 1 közé esik, akkor annak logaritmusa negatív szám, amiből valós négyzetgyök nem vonható.

A hibakezelő-részben használjuk mind a sorszám szerinti,mind a hibakód szerinti hibafigyelést, valamint a kritikus rész végrehajtása után kapcsoljuk ki saját hibafigyelésünket.

```
200 ONERRORGOTO 1000 : REM SAJÁT HIBARUTIN BEKAPCSOLÁSA
210 CLS
220 PRINT "EGY SZÁM TERMÉSZETES ALAPÚ LOGARITMUSA"
230 PRINT "NÉGYZETGYÖKÉNEK MEGHATÁROZÁSA" : PRINT
240 INPUT "KÉREM A SZÁMOT ( >0 ) " ; A
250 B=LOG(A)
260 C=SQR(B)
270 ONERRORGOTO 0
280 IF C<0 THEN GOTO 300
290 PRINT "A KIVÁNT NÉGYZETGYÖK :";C
300 REM VÉGE A KRITIKUS RÉSZNEK
      . . . . . .
```



```
      . . . . . .
1000 REM HIBAFIGYELŐSZOLGÁLAT
1010 IF ERR/2+1 = 6 THEN GOTO 1110
1020 IF ERL = 260 THEN GOTO 1070
1030 REM 0 ÉS NEGATIV SZÁM LOGARITMUSA
1040 PRINT "0-RA ÉS NEGATIV SZÁMRA A LOGARITMUS " ::
     PRINT "NEM ÉRTELMEZETT ! "
1050 PRINT "ÜSSÖN BE EGY POZITIV SZÁMOT "
1060 RESUME 240
1070 REM A SZÁM LOGARITMUSA NEGATIV
1080 PRINT "NULLA ÉS EGY KÖZÖTTI SZÁM":
     PRINT "LOGARITMUSA NEGATIV, ÍGY BELŐLE"
1090 PRINT "NEGYZETGYÖK NEM VONHATÓ !"
1100 C=-1 : RESUME 270
1110 REM A BEÍRT SZÁM TÚL NAGY
1120 PRINT "A BEÍRT SZÁM TÚL NAGY"
1130 PRINT "ÜSSÖN BE EGY KISEBBET !"
1140 RESUME 240
```

Példa:

Második feladatunk legyen a következő: egy hosszú (beolvasandó adattól függő vagy egy számolás eredményeként adódó) vektort ciklikusan ismétlődő számsorozattal kell feltöltenünk. Sőt, az egyes alkalmazásokkor ez a hossz változhat is, így könnyen változtatható programot kell írnunk. A megoldáshoz, a ciklusban szereplő számokat **DATA** utasításban helyezzük el! Hogy ne kelljen előre leszámolni a számsorozat tagjainak számát, a feladatot **ONERRORGOTO**-val, az **OD hiba** figyelésével oldjuk meg. Ha nem ez a hiba következett be, kiíratjuk a sorszámot, a **hiba kódszámát** és megállunk, előtte hatástalanítva az **ONERRORGOTO** utasítást!

```
      . . . . . .
200 REM N : A FELTÖLTENDŐ VEKTOR HOSSZA
210 DATA 7,11,13,17,23,29
220 REM A CIKLUSBAN SZEREPLŐ ELEMEK SZÁMA MOST 6
230 ONERRORGOTO 1000
240 FOR J=1 TO N
250 READ T(J)
260 NEXT J
270 ONERRORGOTO 0

      . . . . . .
```

```
      . . . . . .

1000 IF ERR/2+1<>4 THEN GOTO 1030
1010 RESTORE
1020 RESUME
1030 REM NEM VÁRT HIBA
1040 PRINT "VÁRATLAN HIBA A ";ERL;". SORBAN!"
1050 PRINT "A HIBA KÓDJA:";ERR/2+1
1060 RESUME 1070
1070 ONERRORGOTO 0 : STOP

      . . . . . .
```


2.22 HIBAKERESÉS

Az előbbi részben felsoroltuk azokat az utasításokat, melyek lehetőséget biztosítanak a futás során előforduló hibák helyes kezelésére. Előfordulhat azonban, hogy programunk nem úgy működik, ahogy gondoltuk, azaz valamilyen - vagy algoritmikus, vagy esetleg véletlen elírásból adódó - hibát követtünk el, ezért meg kell keresnünk a programnak azt a részét, ahol elkövettük a hibát. Tekintsük tehát át a hibakeresés lehetőségeit. (Szeretnénk tudni például merre jár a programunk, lényeges változóknak mik az értékei stb...) Az első - és talán a legjobban használható lehetőségünk - a **CONT** parancs megfelelő használata. A program kritikus helyein helyezzünk el töréspontokat- a végrehajtást függesszük fel -, vagyis írjunk **STOP** utasítást. Ekkor megszakad a futás, és az általunk lényegesnek vélt változók értékeit kiírathatjuk, ha nem volt megfelelő, esetleg átírhatjuk a helyes értékre. Ezután - ha úgy gondoljuk - a **CONT** parancs hatására a **STOP** utáni utasításra adódik a vezérlés. Egy másik, ehhez kapcsolódó lehetőség a **BREAK** billentyűvel kapcsolatos. Amikor ezt a gombot lenyomjuk, a program futása megszakad. Szintén elvégezhetők az előbbi hibaelemzések, és a **CONT** paranccsal a megszakításkor végrehajtott utasítás utáni utasításra kerül a vezérlés.

Példa: Feladatunk csupán annyi, hogy határozzuk meg két szám hányadosát. Beolvasunk két számot és hibarutinnal figyeljük nagyságrendjüket, valamint a nullával való osztást.

```
100 ONERRORGOTO 1000
110 INPUT "AZ OSZTANDÓ ";A
120 INPUT "AZ OSZTÓ ";B
130 C=A/D
140 PRINT C
150 ONERRORGOTO 0

      . . . . . .

1000 REM HIBAFIGYELŐSZOLGÁLAT
1010 IF ERL<>110 AND ERL<>120 THEN GOTO 1040
1020 PRINT "ÍRJON BE KISEBB SZÁMOT"
1030 RESUME 110
1040 REM NULLÁVAL OSZTOTTUNK
1050 PRINT "AZ OSZTÓ NULLA VOLT"
1060 PRINT "ÍRJON BE MASIKAT"
1070 RESUME 120

      . . . . . .
```

Futtassuk le programunkat:

```
RUN
AZ OSZTANDÓ ?12
AZ OSZTÓ ?24
AZ OSZTÓ NULLA VOLT
ÍRJON BE MÁSIKAT
AZ OSZTÓ ?14
AZ OSZTÓ NULLA VOLT
ÍRJON BE MÁSIKAT
```

Ezek után újra kérdezné az osztót és ezt a **BREAK** billentyű lenyomásáig folytatná, annak ellenére, hogy nem nullát írtunk az osztó értékének. Helyezzünk el a 135 sorba egy **STOP** utasítást és újból próbálkozzunk.

```
RUN
AZ OSZTANDÓ ?12
AZ OSZTÓ ?24
BREAK IN 135
PRINT B
 24
READY
```

Tehát az osztás előtt még nem nulla az osztó, így csak valami elírás lehet a baj. Megnézve a 130-as sort, valóban, B helyett D betűt írtunk, mivel viszont a D változó értéke - ezekszerint - nulla volt, működésbe lépett a saját hibakezelő rutinunk. Ezt javítva, programunk már helyesen működik. Ha megtaláltuk a kritikus részt, biztosított számunkra az is, hogy csak ezt a részt futtassuk. Erre szolgál a sorszámos **RUN** parancs.

Példa: RUN 200

Ennek hatására először nullázódnak a használt változók, majd a 200-as sorra adódik a vezérlés. A másik lehetőségünk a **GOTO** utasítás parancsként való használata. Ilyenkor a változók nem nullázódnak, ami előnye is, de időnként hátránya is lehet. Természetesen azokat a változókat, melyeket ez a programrész bemenő adatként használ, a parancs kiadása előtt be kell állítani a megfelelő értékekre.

Példa: GOTO 200

Ha ezekkel az eszközökkel nem tudtuk felderíteni a hiba okát, még mindig van egy lehetőségünk a hiba megkeresésére: használhatjuk a **HT-1080Z/64** nyomkövetési rendszerét, az úgynevezett **TRACE üzemmódot**. Ebben az esetben minden végrehajtott sor sorszámát<> jelek közé kiírja. Így lényegében a program vezérlését figyelhetjük,azaz ellenőrizhetjük merre jár a programunk. A megjelenítésre a nyomkövető rendszer is a képernyőt használja, így az értékes információk keverednek a sorszámokkal. Ennek oka az, hogy a sorszámokat mindig oda kezdi el írni, ahol a villogó fénypont - k u r z o r - áll.Ha programunkon belül a képernyőt töröljük, természetesen a nyomlista is törlődik. Így - a képernyő viszonylagos áttekinthetetlensége miatt - valóban csak indokolt esetben célszerű használni.

A nyomkövetés bekapcsolása, a **TRACE üzemmód** beállítása **TRON** utasítás végrehajtásával, vagy a **TRON** parancs kiadásával érhető el:

sorszám **TRON**

A nyomkövetési üzemmód mindaddig érvényben marad, amíg nem mondjuk az ellenkezőjét. Erre az alábbi lehetőségeink vannak:

1. kikapcsoljuk a gépet
2. kiadunk egy **NEW** parancsot
3. kazettáról egy programot töltünk be
4. megszüntetjük a **TROFF** paranccsal, vagy a **TROFF** utasításként való használatával:

sorszám **TROFF**

A **TRACE üzemmód** segítségével nyomon követhetjük, csak az általunk kritikusnak vélt programrészt is: elé egy **TRON** mögé egy **TROFF** utasítást elhelyezve.

Példa: határozzuk meg, két nem negatív szám legkisebb közös többszörösét.

```
100 INPUT "AZ EGYIK SZAM ( >0 ) ";A
110 INPUT "A MÁSIK SZÁM ( >0 ) ";B
120 IF A<0 OR B<0 THEN GOTO 100
130 A1=A : B1=B
140 IF A1=B1 THEN GOTO 170
150 IF A1<B1 THEN A1=A1+A ELSE B1=B1+B
160 GOTO 140
170 PRINT "A LEGKISEBB KÖZÖS TÖBBSZÖRÖS :";A1
180 STOP
```

Először futtassuk nyomkövetés nélkül:

```
RUN
AZ EGYIK SZÁM ( >0 ) ?4
A MÁSIK SZÁM ( >0 ) ?-2
AZ EGYIK SZÁM ( >0 ) ?4
A MÁSIK SZÁM ( >0 ) ?6
A LEGKISEBB KÖZÖS TÖBBSZÖRÖS : 12
BREAK IN 180
READY
```

Bár az eredmény jó, nézzük meg, hogyan működik a program nyomkövetéssel. Most helyezzük el a **95 TRON** illetve a **175 TROFF** utasításokat, és újra futtassuk programunkat:

```
RUN
<100>AZ EGYIK SZÁM ( >0 ) ?4
<110>A MÁSIK SZÁM ( >0 ) ?-2
<120><100>AZ EGYIK SZÁM ( >0 ) ?4
<110>A MÁSIK SZÁM ( >0 ) ?6
<120><130><140><150><160><140><150><160><140><150><160>
<140><170>A LEGKISEBB KÖZÖS TÖBBSZÖRÖS : 12
<175>
BREAK IN 180
READY
```


2.23 PARANCSOK

A parancsok általában a **BASIC** programmal csinálnak valamit - listázzák, átszámozzák, futtatják - szemben az utasításokkal, amelyek a felhasználó által megadott adatokkal végeznek műveleteket. A **HT-1080Z/64** esetében a parancsok általában nemcsak közvetlen üzemmódban, de programban is használhatók. Ez alól csak a **CONT** parancs kivétel. Ha egy parancs programban hajtódik végre, a parancs végrehajtása után a program futása megszakad. Ez alól értelemszerűen kivétel a **RUN**, továbbá a **TRON** és a **TROFF**.

A **HT-1080Z/64** gép az alábbi parancsokat ismeri:

AUTO automatikusan sorszámot ír ki a beírandó sorok elejére. Lehetséges alakjai:

AUTO sorszám1,sorszám2
AUTO sorszám1
AUTO, sorszám2 Ne használjuk, 0-s sorszámot ad!
AUTO

A fenti alakokban sorszám1 jelenti a kezdő sorszámot, sorszám2 a növekményt. Az elmaradó sorszám helyére 10-et helyettesít a gép.
Az automatikus programsorszámozást a **BREAK** billentyűvel lehet megszakítani.
Ha a programban már van ilyen sorszámú sor, akkor a kiírt sorszám mögé még egy * karaktert is kiír a gép.
Sorszám1 helyett "." is használható az aktuális sor jelzésére.

Példa:	beírt parancs	kiírt sorszámok
	AUTO	10,20,30,...
	AUTO 25,12	25,37,49,...

CLEAR Ha argumentum nélkül használjuk, akkor 0 értéket tölt az összes numerikus változóba, a karakteres változók hosszát pedig 0-ra állítja. Ha numerikus argumentummal használjuk (Pl. **CLEAR** 100), akkor a változók törlésén kívül az argumentumban megadott számú byte-ot tart fenn karakterláncok tárolására (a fenti példában 100 byte-ot).
A gép bekapcsolásakor automatikusan végrehajt egy **CLEAR** 50 parancsot.
A parancs argumentum nélküli formája nem változtatja meg a szövegek tárolására fenntartott terület méretét. A parancs valójában magán a programon kívül szinte mindent töröl. Így például törlődik a szimbólumtábla, érvényét veszti az összes **DIM** utasítás - ez teszi lehetővé az újradeklarálást - és a típusdeklaráló utasítások - **DEFDBL**, **DEFSTR**, **DEFSNG** - is.

Törlődik a szubrutinok visszatérési címeit tároló verem, stb. Alaposan gondoljuk meg tehát, hogy a CLEAR parancsot hová írjuk a programban!

CLOAD BASIC programot tölt be kazettáról. A gépben esetleg bentlévő BASIC program törlődik. A parancs lehetséges alakjai:

```
CLOAD
CLOAD "karakter"
CLOAD #-1,"karakter" vagy CLOAD #-2,"karakter"
```

A fenti példákban ha megadjuk "karakter"-t, akkor egy ilyen nevű programot keres a gép. Ha közben más nevűeket talál a a szalagon, azok nevét a jobb felső sarokba kiírja. Ha "karakter"-t nem adjuk meg, akkor a kazettán soronkövetkező programot tölti be a gép.
A CLOAD parancsnak #-1 formában a kazettás egység azonosítóját adtuk meg. Ha elhagyjuk, akkor az 1-es sorszámú kazettás egységről tölt be a gép programot.
A CLOAD parancs kiadása előtt a szalagot tekercseljük a megfelelő program elé, és a magnót állítsuk lejátszásra (F1 gomb felengedve).
A megfelelő hangerőt ki kell kisérletezni (ha külső magnót használunk).
A CLOAD parancs maga indítja a magnót.

Tanácsos a programok között "üres" helyet hagyni, és a számláló állását feljegyezni, hogy könnyebben tudjunk egy program elejére állni!

CLOAD? A parancs segítségével ellenőrizhetjük, sikerült-e a BASIC program mentése. A parancs alakjai:

```
CLOAD? vagy CLOAD? "karakter"
CLOAD #-1,? vagy CLOAD #-2,?
CLOAD #-1,?"karakter" vagy
                CLOAD #-2,?"karakter"
```

A paraméterek jelentése megegyezik CLOAD -nál leírtakkal. A parancs byte-onként összehasonlítja a kazettán levő programot a gépben tárolttal. Ha eltérést talál, akkor BAD hibajelzést ad.
Ilyenkor a mentést meg kell ismételni! (Ld. CSAVE)

CONT Ha egy program futása BREAK billentyű lenyomása miatt vagy STOP utasítás hatására megszakadt,akkor a CONT paranccsal a futás folytatható. A megszakítás alkalmas rá, hogy bizonyos változók értékeit megvizsgáljuk, esetleg megváltoztassuk.

Nem hajtható végre a CONT parancs, ha a program megállítása után a programot megváltoztattuk: pl. sorok módosítása, sorok törlése, új sorok beszúrása. Nem használható CONT parancs akkor sem, ha a program hiba miatt állt le.A CONT parancs programban nem használható!

CSAVE Ezzel a paranccsal BASIC programot menthetünk kazettára. A parancs lehetséges alakjai:

```
CSAVE "karakter"
CSAVE #-1, "karakter" vagy
          CSAVE #-2,"karakter"
```

A parancsnak kötelező megadni a kimentendő program nevét! A név 1 darab tetszőleges karakter lehet, kivéve ".

DELETE A paranccsal sorok törölhetők a BASIC programból. A parancs alakjai:

```
DELETE sorszám1-sorszám2
DELETE -sorszám2
DELETE sorszám
```

Törölhetők tehát egyes sorok, illetve megadott intervallumok. A megadott sorszám-intervallum felső határa létező sor legyen! A "." karakter akármelyik sorszám helyett használható, és az aktuális sort jelöli.
Megjegyzés: egy sort nem érdemes a DELETE paranccsal törölni, könnyebb csak a sorszámot leírni és a NEW LINE billentyűt lenyomni - a sor így is törlődik.

EDIT az EDIT parancs leírását ld. külön a 2.24.Fejezetben

LIST A paranccsal a BASIC programot a képernyőre listázhatjuk. A parancs alakjai:

```
LIST
LIST sorszám1
LIST sorszám1-
LIST sorszám1-sorszám2
LIST -sorszám2
```

Ha nem írunk sorszámot, a teljes programról kapunk listát. Ha sorszám-intervallumot adunk meg, akkor csak a program meghatározott részét listázza. Ha az 1. sorszám marad el, akkor a listázás az első sortól kezdődik, ha a 2. sorszám marad el akkor a program végéig listáz.

Az aktuális sor helyett a "." használható, pl: LIST.-200
Ha a listázást fel akarjuk függeszteni, nyomjuk le a SHIFT és a @ billentyűt. Ez a listázást leállítja. A listázás tetszőleges billentyű leütésére folytatódik.
SHIFT és BREAK egyidejü lenyomásával a listázást ugyancsak megállíthatjuk. Ebben az esetben azonban a listázás folytatódik, ha a SHIFT és a BREAK billentyűt elengedjük. A listázás BREAK billentyűvel szakítható meg.

LLIST Formáját, hatását tekintve teljesen megegyezik a LIST paranccsal, de a lista a képernyő helyett a nyomtatóra íródik. Ha a parancs végrehajtásakor nincs a géphez működőképes nyomtató kapcsolva, a gép olyan állapotba kerül, amit csak a RESET billentyűvel lehet megszüntetni.

NEW Törli a programot a memóriából, 0-t ír a numerikus változókba, 0-t ír a karakteres változók hosszába. Nem változtatja meg a karakteres változók tárolására fenntartott memória méretét.

Megjegyzés: a NEW parancs hatása hasonló a CLEAR parancséhoz, de a gépben levő BASIC program is törlődik.

RE Lehetővé teszi a program újrasorszámozását (REnumber). Lehetséges formái:

```
RE
RE sorszám1
RE sorszám1,sorszám2
```

A fenti formáknál sorszám1 lesz az átszámozott program 1. sorszáma,sorszám2 a sorszám növekménye. Ha valamelyik elmarad, 10-es értékkel pótolja a gép.
A RE parancs meggondolatlan használata nehezen kideríthető hibákhoz vezethet. (ld. példa)
A RE parancs nem használható a gép bekapcsolása után közvetlenül.
Ehhez SYSTEM módban el kell indítani a 12288 címen található programot.

FONTOS: - a RE parancs nem számozza át a RESUME (ld. 2.21 Fejezet) után álló sorszámot,
- szintén nem sorszámozza át a RE parancs azokat a sorszámokat, amelyeknél a sorszámra utaló kulcsszó hibás vagy hiányzik. (ld. példák)

Példa:
```
RE  10-től 10-esével átszámozza a programot
RE  100    100-tól 10-esével számoz
RE  20,32  20-tól 32-es növekménnyel számoz.
```

Hibát okoz a RE használata a következő esetben:

```
10 A=1
20 GOTO 100:REM NINCS ILYEN SORSZAM
RE 100
```

Az eredmény:

```
100 A=1
110 GOTO 100:REM NINCS ILYEN SORSZÁM
```

Látható, hogy a program 2. sorának értelme teljesen megváltozott.

Megjegyzés: természetesen általában a REnumber parancs helyesen számozza át a vezérlésátadó utasításokban található sorszámokat is. A fenti hiba akkor fordul elő, ha a hivatkozott sorszám hiányzik.

Példa: nem sorszámozza át a RE parancs a sorszámokat a következő utasításokban:

```
10 GODUB 100 : REM A KULCSSZÓ HIBÁS
20 ON I GUSUB 10,20,30 : REM A KULCSSZÓ HIBÁS
30 IF A=6 TEHN 10 : REM A KULCSSZÓ HIBÁS
40 IF X>Y TENH 20 ESLE 40 : REM KULCSSZÓ HIBÁS
50 IF Q<6 100 : REM A KULCSSZÓ HIÁNYZIK
```

A következő példában a sorszám át lesz számozva:

```
60 OF I GOSUB 10,20,30 : REM AZ ON KULCSSZÓ HIBÁS
```

RUN A parancs elindítja a program futását. Lehetséges formái:

```
RUN              vagy
RUN sorszám
```

A parancs után egy sorszám is írható, akkor a program a megadott sorszámú sor végrehajtásával indul. A RUN parancs egyben egy CLEAR parancsot is végrehajt. Ha azt akarjuk, hogy a CLEAR ne hajtódjon végre, GOTO-val indítsuk a programot!

SYSTEM Monitor üzemmódba vált át a gép. Részletesen ld. a 2.19. Fejezetben! (A gépi szintű programozás eszközei.)

TROFF kikapcsolja a nyomkövetést. Részletesen ld. a 2.22.Fejezetben

TRON bekapcsolja a nyomkövetést. Részletesen ld. a 2.22.Fejezetben

2.24 PROGRAMSOROK JAVÍTÁSA-SZERKESZTÉSE

A HT-1080Z/64 gépen az EDIT paranccsal lehet programsorokat javítani, szerkeszteni.
A paranccsal csak sorszámos sorok javíthatók.
A sorszám nem módosítható.
Futás közben bizonyos hibák után a képernyőn automatikusan megjelenik a hibás sor sorszáma, jelezve, hogy hol a hiba. Ilyenkor a javítás az EDIT parancs beírása nélkül azonnal megkezdhető.
A parancs megengedett formája:

EDIT sorszám

Az akutális sor sorszáma "."-al helyettesíthető. (Aktuális sornak számít például az utoljára javított sor.)
Magát a javítást a parancs leírása után un. alárendelt parancsokkal, ill. speciális billentyűkkel végezhetjük el.

Vigyázzunk arra, hogy a beszúrási módokban (X, I, H után) a szóköz közönséges karakterként viselkedik, az alárendelt parancsokat azonosító betűk pedig közönséges karaktereknek számítanak!

Az alábbi leírásban a SHIFT/x jelölést úgy kell érteni,hogy egyszerre le kell nyomni a SHIFT és az x billentyűt.
Figyelni kell arra, hogy javítás közben általában nem látható az a karakter, amelynek helyén a kurzor áll. A leírásban ezt a karaktert nevezzük a kurzor karakterének.

NEW LINE — a billentyű hatására a javított sort tárolja a BASIC és kilép a szerkesztési módból. (Beszúrási módban is hatásos.)

nSZÓKÖZ — Ha a SZÓKÖZ billentyűt egyszer leütjük, a kurzor eggyel jobbra mozdul és láthatóvá válik a következő karakter. Ha a SZÓKÖZ billentyű leütése előtt beírunk egy számot -n- akkor a kurzor n pozicióval mozdul jobbra, és n darab karakter válik láthatóvá. (Beszúrási módban a SZÓKÖZ közönséges karakternek számít.)

n← — (balra nyíl) hatására a kurzor balra mozog. Ha n-t nem adjuk meg a kurzor eggyel mozdul balra,egyébként a megadott számú pozicióval. Ha nem beszúrási módban használjuk, akkor a karakter csak a képernyőről tűnik el, beszúrási módban az eltünő karakterek a sorból is törlődnek.

SHIFT/↑ — (fölnyíl) hatására beszúrási módok befejeződnek (ld. X, I, H parancs). Bár a beszúrási mód befejeződik, változatlanul szerkesztési módban maradunk.

A — előlről lehet kezdeni a sor szerkesztését. Ha szerkesztés közben hibáztunk, az A paranccsal visszatérhetünk az eredeti sorhoz, és újra kezdhetjük a javítást anélkül, hogy az EDIT parancsot újra le kellene írni. Az A parancs kiadása előtt használjuk a SHIFT/↑ karaktert, hogy biztosan kilépjünk a beszúrási módból. (Beszúrási módban ugyanis az A nem számít parancsnak.)

nC — a kurzor karakterétől kezdve, attól jobbra haladva karaktereket cserél ki. Ha csak C-t írunk, akkor a kurzor karakterét kicseréli az első leütött karakterre. Ha megadjuk n-t, akkor egyszerre n karaktert cserél. (Beszúrási módban a C közönséges karakternek számít.)

nD — a kurzor karakterétől kezdve attól jobbra haladva n darab karaktert töröl. Ha n-t elhagyjuk, akkor a kurzor karakterét törli. A törölt karakterek !-k közé zárva jelennek meg. (Beszúrási módban a D közönséges karakternek számít.)

E — a javított sor tárolódik, kilépünk a szerkesztési módból. (NEW LINE használata célszerűbb, mert mindig hatásos, míg beszúrási módban az E közönséges karakternek számít.)

H — törli a sornak a kurzortól kezdődő részét, majd beszúrási módba vált. (Beszúrási módban a H közönséges karakternek számít.)

I — beszúrási módra vált. A beírt karakterek a kurzor helyétől kezdve be lesznek szúrva. Beszúrási módból csak a NEW LINE vagy a SHIFT/↑ speciális billentyűkkel lehet kilépni. (Beszúrási módban az I közönséges karakternek számít.)

nKc — a c karakter n. előfordulásáig töröl mindent - a karaktert már nem - és a kurzort a karakterre állítja. Ügyelni kell rá, hogy a gép itt megkülönbözteti a kis és a nagybetűket. (Beszúrási módban a K közönséges karakternek számít.)

L — kilistázza a sornak a kurzortól kezdődő részét. Célszerű a szerkesztést L paranccsal kezdeni, hogy lássuk mit is javítunk. (Beszúrási módban az L közönséges karakternek számít.)

Q — kilépünk a szerkesztési módból, a szerkesztett sor változatlan marad. Mielőtt a Q parancsot kiadnánk, használjuk a SHIFT/↑ billentyűt, hogy biztosan kilépjünk a beszúrási módból. (Beszúrási módban a Q közönséges karakternek számít.)

nSc — a kurzortól jobbra megkeresi a c karakter n-ik előfordulását és a kurzort ráállítja. Ha n-t nem adjuk meg, akkor az 1. előfordulást keresi meg. Ha a karaktert nem találja, a kurzor a sor végére áll. Ügyelni kell rá, hogy kereséskor a gép megkülönbözteti a kis és a nagybetűket. (Beszúrási módban az S közönséges karakternek számít.)

X — a képernyőre íródik a sornak a kurzortól kezdődő része, és beszúrási módba kerülünk. Ezzel az alárendelt paranccsal lehet kényelmesen beszúrni a sor végére, vagy onnan törölni. (Beszúrási módban az X közönséges karakternek számít.)

Példák: a szerkesztést a következő BASIC utasításon mutatjuk be:

```
100 FOR J = 1 TO 15 STEP .1 : PRINT J,J*J,J↑2 : NEXT
```

Most írjuk le EDIT 100 (és nyomjuk le a NEW LINE-t).
A képernyőn ez áll:

```
100 _
```

Ha néhányszor leütjük a SZÓKÖZ billentyűt, újabb és újabb karakterek válnak láthatóvá.
Ha a ← (balranyíl) billentyűt használjuk, a kurzor balra mozog, karakterek tünnek el a képernyőről, de a sorból nem törlődnek. Ha leütjük az L billentyűt, a sor hátralevő része is kiíródik a képernyőre, a kurzor új sor elejére áll és újra kiírja a sorszámot.

Irjuk be a NEXT után a ciklusváltozót. Használjuk ehhez az X parancsot. A képernyőn ezt látjuk:

```
100 FOR J = 1 TO 15 STEP .1 : PRINT J,J*J,J↑2 : NEXT_
```


Irjunk be egy SZÓKÖZ-t és a ciklusváltozót, majd üssük le a NEW LINE billentyűt!

Cseréljük ki a.1 lépésközt 3.6-ra!

Írjuk le: EDIT 100
Ezt látjuk:

```
100_
```

Nyomjuk le néhányszor a SZÓKÖZ billentyűt, míg ezt nem látjuk:

```
100 FOR J = 1 TO 15 STEP _
```

Írjuk le: 3 SHIFT/↑
Ezt látjuk:

```
100 FOR J = 1 TO 15 STEP 3
```

Írjuk le: SZÓKÖZC6
Ezt látjuk:

```
100 FOR J = 1 TO 15 STEP 3.6
```

Üssük le a NEW LINE-t.
Ezt látjuk:

```
100 FOR J = 1 TO 15 STEP 3.6 : PRINT J,J*J,J↑2 : NEXT J
```

3. Hasznos tudnivalók a HT-1080Z/64 BASIC-ról

3.1 A BASIC UTASÍTÁSAINAK ÖSSZEFOGLALÁSA

P betű jelöli a csak parancsként használható alapszavakat, U a csak utasításként használhatót, F pedig a függvényeket.

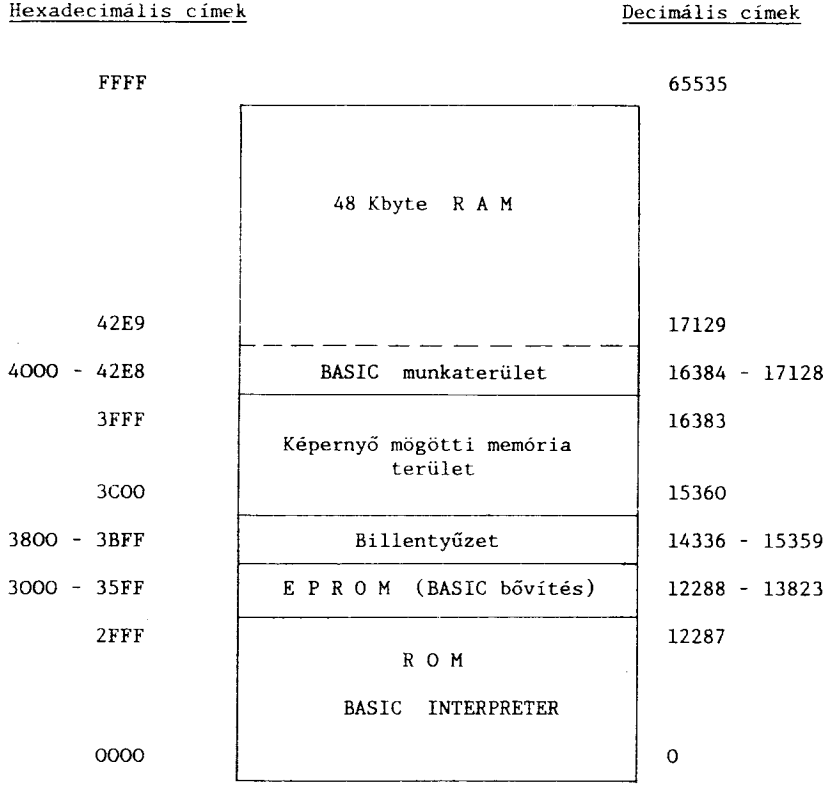
RESUME NEXT		Visszatérés hibarutinból a hibás utáni utasításra	111
RETURN		Visszatérés szubrutinból	71
RIGHT$	F	Szöveg jobboldalának leválasztása	65
RND	F	Véletlenszám előállítás, vagy 0 és 1 közötti valós, vagy 1 és - paraméterként megadott - N közötti egész	76
RUN		Program elindítása az elejéről, vagy adott sorszámról	123
SET		Egy képernyőpont kivilágítása	83
SNG	F	Előjel függvény	28
SIN	F	Szinusz függvény	28
SQR	F	Négyzetgyök függvény	28
STOP		A program futásának fejezése	51
STR$	F	Konverzió szám típusból szöveg típusba	67
STRING$	F	Azonos betűkből álló szöveg előállítása	67
SYSTEM		Monitor üzemmódba váltás	97
TAB	F	Kiírási pozició meghatározása, csak PRINT utasításban használható	37
TAN	F	Tangens függvény	28
TRON		Nyomkövetés bekapcsolása	117
TROFF		Nyomkövetés megszüntetése	117
USR	F	Gépi nyelvű szubrutin hívása, a szubrutin címét a 16526,16527 címre kell előtte elhelyezni	94
VAL	F	Karakteres változó átalakítása numerikussá	68
VARPTR	F	Változó, vagy tömbelem kezdőcíme a memóriában	92

3.2 A BASIC KULCSSZAVAK ÉS TOKEN-JEIK

A HT-1080Z/64 a BASIC kulcsszavakat belső ábrázolási kódjukkal (TOKEN-jeikkel) ábrázolja. Ezeket a kódokat tartalmazza ez a fejezet. Ezek az alapszavak nem fordulhatnak elő változónévben. Sok olyan kulcsszó is szerepel itt, amelyek a HT-1080Z/64 alapkiépítésében nem használhatók, beírásukra ?SN Error hibajelzést ad a számítógép, de ezeket sem lehet változónév belsejében használni.

Alapszó	Token	Alapszó	Token	Alapszó	Token	Alapszó	Token
ABS	D9	AND	D2	ASC	F6	ATN	E4
AUTO	B7	CDBL	F1	CHR$	F7	CINT	EF
CLEAR	B8	CLOAD	B9	CLOSE	A6	CLS	84
CMD	85	CONT	B3	COS	E1	CSAVE	BA
CSNG	FO	CVD	E8	CVI	E6	CVS	E7
DATA	88	DEF	BO	DEFDBL	9B	DEFINT	99
DEFSNG	9A	DEFSTR	98	DELETE	B6	DIM	8A
EDIT	9D	ELSE	95	END	8O	EOF	E9
ERL	C2	ERR	C3	ERROR	9E	EXP	EO
FIELD	A3	FIX	F2	FN	BE	FOR	8i
FRE	DA	GET	A4	GOTO	8D	GOSUB	91
IF	8F	INKEY$	C9	INP	DB	INPUT	89
INSTR	C5	INT	D8	KILL	AA	LEFT$	F8
LEN	F3	LET	8C	LINE	9C	LIST	B4
LLIST	B5	LOAD	A7	LOC	EA	LOF	EB
LOG	DF	LPRINT	AF	LSET	AB	MEM	C8
MERGE	A8	MID$	FA	MKD$	EE	MKI$	EC
MKS$	ED	NAME	A9	NEW	BB	NEXT	87
NOT	CB	ON	A1	OPEN	A2	OR	D3
OUT	AO	PEEK	E5	POINT	C6	POKE	B1
POS	DC	PRINT	B2	PUT	A5	RANDOM	86
READ	8B	REM	93	RESET	82	RESTORE	9O
RESUME	9F	RETURN	92	RIGHT$	F9	RND	DE
RSET	AC	RUN	8E	SAVE	AD	SET	83
SGN	D7	SIN	E2	SQR	DD	STEP	CC
STOP	94	STR$	F4	STRING$	C4	SYSTEM	AE
TAB(	BC	TAN	E3	THEN	CA	TIME$	C7
TO	BD	TROFF	97	TRON	96	USING	BF
USR	C1	VAL	F5	VARPIR	CO	+	CD
-	CE	*	CF	/	DO	↑	D1
>	D4	=	D5	<	D6	'	FB

3.3 A MEMÓRIA FELOSZTÁSA



A billentyűzet memória-mátrixának szerkezete

Minden egyes billentyűnek a billentyű-mátrxiban megfelel egy-egy bit, amelynek 1 vagy 0 értéke jelzi a hozzárendelt billentyű lenyomott ill. nem lenyomva tartott állapotát. Így a billentyűzet figyelésre a PEEK függvény használható fel. Ezzel természetesen akár több billentyű egyidejű érzékelése is megoldható. (Ld. 2.17. Fejezetben!)

Cím	0.	1.	2.	3.	4.	5.	6.	7. bit
14337	@	A a	B b	C c	D d	E e	F f	G g
14338	H h	I i	J j	K k	L l	M m	N n	O o
14340	P p	Q q	R r	S s	T t	U u	V v	W w
14344	X x	Y y	Z z					
14352	0	1 !	2 "	3 #	4 $	5 %	6 &	7 '
14368	8 (	9)	: *	; +	, <	- =	. >	/ ?
14400	NL	CL	BR	fel	le	bal	jobb	SP
14464	SH							

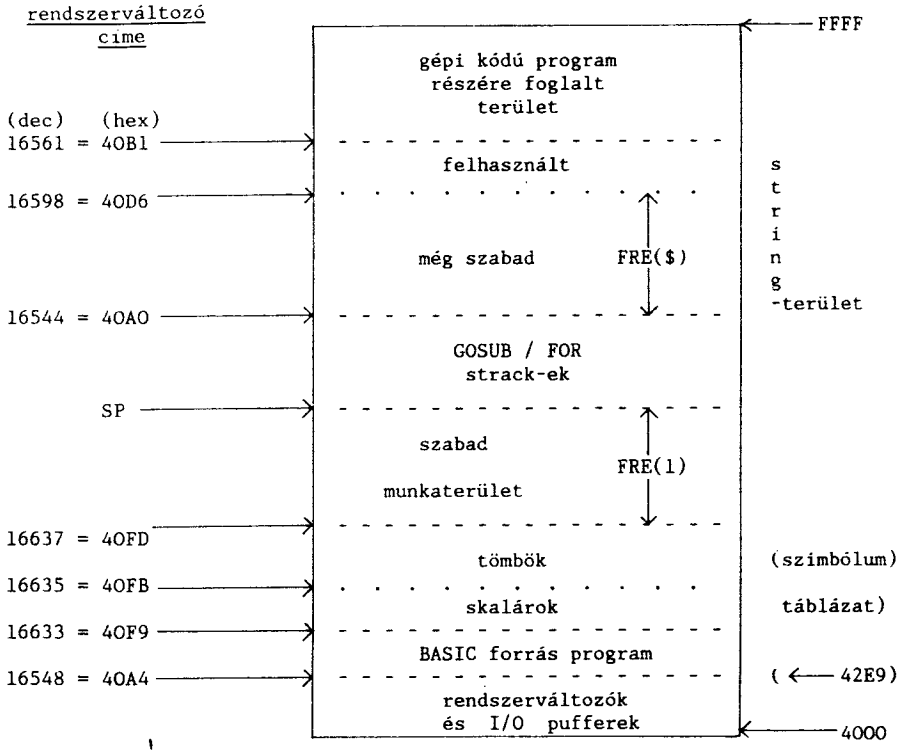
A táblázatbeli rövidítések:

NL : NEW LINE
CL : CLEAR
BR : BREAK
fel : ↑
le : ↓
bal : ←
jobb : →
SP : SZÓKÖZ
SH : SHIFT

A - HT-1080Z/64 BASIC RAM területének "finomszerkezete"

(A memória szerkezete a fontosabb rendszerváltozók tükrében)

A RAM szerkezete a program írása és futása közben állandó változásban van. Ennek a dinamizmusnak a nyomon követését, adminisztrálását végzik az un. **BASIC rendszerváltozók**. Ezek az egyes fontos tartományok kezdő (és vég) címeit tartalmazzák. (Vannak más nem ilyen jellegű információkat rögzitő rendszerváltozók is.)
Az alábbi ábrán megadjuk a legfontosabb területek helyét meghatározó rendszerváltozók kezdőcímeit. E címek mindig az alacsonyabb helyiértékű byte-jához tartoznak, a magasabb helyiértékű az őt követő byte-on van (a Z80-nál megszokott módon). Így pl. a szabad munkaterület kezdetét a **PEEK(16637)+256*PEEK(16638)** kifejezéssel határozhatjuk meg.



Megjegyzések a fentiekhez:

A BASIC forrás program kezdőcímét tartalmazó rendszerváltozó értéke eredendően 42E9. (Ezt kívántuk kifejezni azzal, hogy a táblázatban zárójelek között szerepeltettük.)

A bekapcsoláskor (vagy processzor **RESET** esetén) a **MEMORY SIZE?** kérdésre adott válasz alapján foglalja le a BASIC értelmező által el nem érhető, a gépi kódú programokhoz tartozó területet (40B1 rendszerváltozó).

A **CLEAR** állítja be a stringterület maximális méretét (4AD6, 40A0 rendszerváltozók).

Az SP az Z80 processzor SP-(Stack Pointere)t jelenti, így a BASIC-ből, közvetlenül (pl. **PEEK** segítségével) nem lekérdezhető.

A 40D6 rendszerváltozó az egybefüggően szabad területet jelöli ki, ami általában kisebb, mint a ténylegesen felhasználatlan szöveg-terület (ugyanis vannak időközben felszabadult a szövegek is, amelyek még foglalják a helyet a felső string-területből), ezért a valódi szabad méret meghatározása előtt tömörítésre van szükség. Ezt el is végzi a **FRE**(szöveg) függvény.

További fontos rendszerváltozók:

USR-cím	: 408E (=16526-27)	
akt.program sor címe	: 40A2 (=16546-47)	
utolsó **RND**szám	: 40AC (=16639-40)	
billentyű puffer kezdőcím	: 40A7 (=16551-52)	
kurzor címe a képernyőn	: 4020 (=16416-17)	
BREAK-érzékelés	: 400C (=16396)	
	kikapcsolás	POKE 16396,7
	visszaállítás	POKE 16396,201
képernyőmegjelenítés (ECHO)-letiltás	: 401D (=16413)	
	kikapcsolás	POKE 16413,0
	visszaállítás	POKE 16396,7

A gépi kódban programozóknak segítséget jelenthet, ha felhasználhatnak a **ROM** -ban eleve meglevő rutinokat. Így ezek közül néhány fontosabbat megemlítünk.

- 002B : Billentyűzet figyelés (**A**:=a lenyomott billentyű kódja, vagy 0);
- 0033 : Karakter megjelenítés a képernyőn (**A**-ban a karakter kódja);
- 0049 : Billentyűleütésre vár (**A**:=a lenyomott billentyű kódja);
- 05D9 : Sorbeolvasás (a **HL** által megjelölt címre írja a beütött sort, a sor hossza maximum annyi lehet, amennyit a **B** tartalmaz);
- 01C9 : A **CLS** utasítás;

3.4 A KÉPERNYŐ FELOSZTÁSA

A képernyőt a HT-1080Z/64 számítógépen többféle felosztásban használhatjuk. Nézzük végig ezeket a lehetőségeket.

1. A képernyő 16 sorból áll.
2. Soronként 64 karaktert írhatunk. Ezt az üzemmódot - többfajta lehetőség miatt - nevezzük normál üzemmódnak!
3. Lehetőségünk van soronként 32 karakter írására is. Ezt nevezzük félképernyős üzemmódnak! Ilyenkor egyszerre a félképernyő látható. Ezt a VIDEO CUT gomb benyomásával tudjuk aktivizálni. (Ez a gép hátulján található.) A PAGE gomb használatával választhatjuk ki azt a képernyő részt, melyet látni szeretnénk. A VIDEO CUT gomb kiengedésével visszatérhetünk a normál üzemmódba.
4. 32 karaktert megjeleníthetünk úgynevezett ritkított üzemmódban is. Ekkor a kiírt karakterek közé szóköz kerül. Ezt a PRINT CHR$(23) paranccsal érhetjük el. A normál üzemmódba való visszatérés a NEW vagy a CLS hatására történhet meg.
5. Lehetőségünk van a teljes magyar ABC, valamint néhány, a matematikában használatos karakter megjelenítésére. Az e lehetőséget biztositó kijelzést nevezzük speciális üzemmódnak! Ehhez a BASIC rendszert ki kell terjesztenünk. Ezt a következő utasításokkal érhetjük el. Vagy rögtön bekapcsolás után, vagy parancsra várva, a következőket kell beírnunk:

```
> SYSTEM NEW LINE
?* /12288 NEW LINE
BASIC ROM EXTENSION 2.51
READY
>
```

(Az utolsó három sort a rendszer írja ki.)
A speciális karakterek a memóriában két karakteren ábrázolódnak, ahol az első karakter az aposztróf ('jel), melynek megjelenítéséhez a SHIFT @ billentyűket kell lenyomnunk. A második karakter pedig a speciális jelhez rendelt billentyű. Ezek a következőek:

A billentyűzeten található jel	jel után SHIFT -tel		jel után SHIFT nélkül	
A	nagy á	(0)	á	(16)
E	nagy é	(1)	é	(17)
I	nagy hosszú í	(2)	kis hosszú í	(18)
O	nagy hosszú ó	(3)	kis hosszú ó	(19)
U	nagy hosszú ú	(6)	kis hosszú ú	(20)
3	nagy rövid ö	(4)	ö	(20)
4	nagy hosszú ő	(5)	kis hosszú ő	(21)
5	nagy rövid ü	(7)	ü	(23)
6	nagy hosszú ű	(8)	kis hosszú ű	(24)
7	nagy omega	(15)	kis omega	(31)
8	végtelen	(11)	kis alfa	(25)
9	kis tele négyzet	(9)	négyzetgyök	(12)
B	pszi	(14)	kis béta	(26)
G	pi	(30)	kis gamma	(27)
M	lambda	(28)	mű	(29)
S	szumma	(13)	integrál	(10)

Megjegyzendő, hogy a zárójelben levő decimális számok a hozzájuk tartozó jelek karaktergenerátorbeli kódjai, íly módon a képernyőmemóriába egy POKE utasítással beírva megjeleníthetők. Lehetőségünk van arra is, hogy a speciális jel beírásakor a beütött aposztrof ne zavarja a beírt szöveget, azaz csak a speciális jel jelenjen meg a képernyőn. Ezt a

sorszám PRINT CHR$(20)

utasításként vagy parancsként beírva érhetjük el. Természetesen, ha kétszer írjuk be a 'jelet, a második már megjelenik. Ha nem adtuk ezt az utasítást, vagy a

sorszám PRINT CHR$(21)

utasítással hatástalanítottuk,az aposztróf jel is megjelenik a képernyőn. Biztosított az ASCII karakterkészlet CONTROL kódjainak a bevitele is. Ennek módja: a SHIFT és a CLEAR billentyűk egyidejű benyomása és az illető karakter leütése.

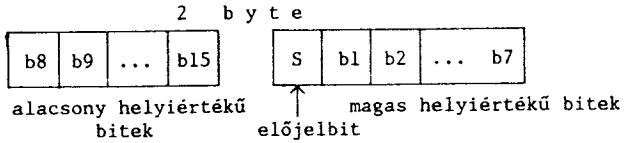
6. Grafikus üzemmódban a 16*64-es képernyőnk 48*128 grafikusan elérhető pontra bomlik, azaz egy karakter 3*2 felé osztódik, 3 grafikus sorra és 2 grafikus oszlopra, mely pontok a SET utasítással kivilágíthatók a RESET utasítással lekapcsolhatók. (Ld. 2.14. Fejezet!)

3.5 ADATÁBRÁZOLÁS

I. Szám típusúak

1. Egész adatok

Ábrázolásuk 2 byte-on, kettes komplemens kódban történik. Az első byte tartalmazza az alacsonyabb helyiértékű biteket, a második a magasabb helyiértékeket (ennek első bitje az előjelbit -> negativ=1, nem negativ=0).



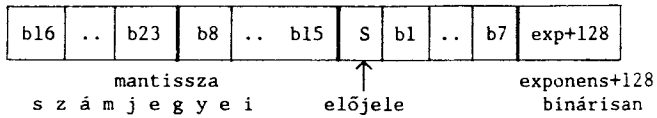
Példák az ábrázolásra:

a decimális szám	ábrázolása magas helyiértékű byte	alacsony byte
	S1234567	01234567.bitje
10=	00000000	00001010
127=	00000000	01111111
256=	00000001	00000000
32767=	01111111	11111111
-1=	11111111	11111111
-10=	11111111	11110110
-127=	11111111	10000001
-256=	11111110	00000000
-32767=	10000000	00000001
-32768=	10000000	00000000
0=	00000000	00000000

2. Valós adatok

A. Egyszeres pontosságú adatok

Ábrázolásuk 4 byte-on történik m * 2exp alakban, 6 decimális jegy pontossággal, aritmetikai kerekítéssel. A bináris ábrázolású mantissza - amely az első 3 byte-ot foglalja el - abszolut értékét tárolja a gép normalizáltan úgy, hogy az első bitet nem ábrázolja. Az exponens additiv kódú bináris ábrázolású (ha exp=0, az 128-ként van ábrázolva). A valós nullát a karakterisztika 0 volta jelzi, s a mantissza bitek értéke érdektelen.



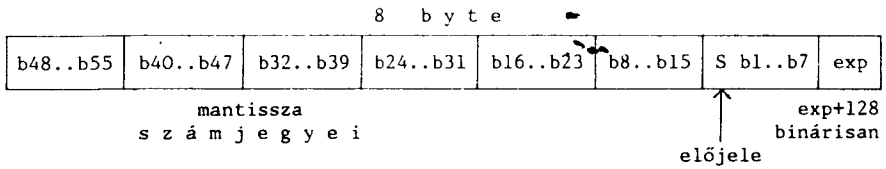
Példák az ábrázolásra:

a decimális szám		ábrázolása 1.	2.	3.	4.byte
0	=	x	x	x	0
1	=	0	0	0	129
10	=	0	0	32	132
127	=	0	0	127	135
999999	=	240	35	116	148
123456789	=	163	121	107	155
-32768	=	0	0	128	144
-0.987654321	=	234	214	252	128
-0.5	=	0	0	128	128
-0.0625	=	0	0	128	125

x jelentése = közömbös

B. Dupla pontosságú adatok

Ábrázolásuk elve azonos az egyszeres pontosságú adatokéval, csak a mantissza tárolása céljából itt 7 byte van fenntartva, ami 16 decimális jegy pontosságot garantál.

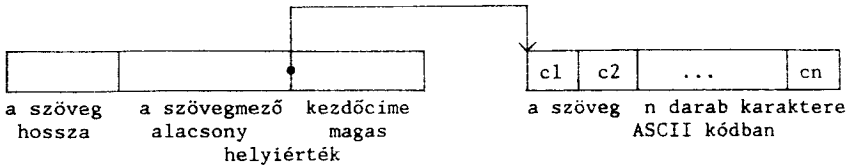


Példák az ábrázolásra:

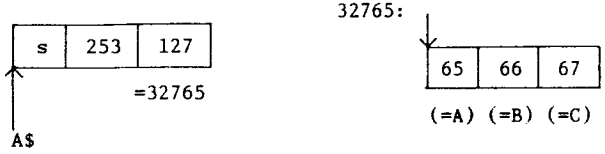
a decimális szám	ábrázolása 1.	2.	3.	4.	5.	6.	7.	8. byte
0 =	x	x	x	x	x	x	x	0
1 =	0	0	0	0	0	0	0	129
123456789 =	0	0	0	160	162	121	107	155
-0.987654321 =	197	181	114	224	233	214	252	128
123.456D-20 =	166	225	204	173	91	48	54	69
1D-38 =	35	199	83	237	220	199	89	2
1D+38 =	139	13	181	80	153	118	22	255

II. Szöveg típusú adatok

A szövegek ábrázolása két objektummal történik: fejjel és szövegmezővel. Egy szöveges adat feje mindig 3 byte-ból áll,a szövegmező pedig annyi byte-ot foglal, ahány jelből a szöveg áll. Tehát n db. karakter tárolásához n+3 byte helyet foglal a BASIC.



Példa: 10 A$="ABC" esetén A$ ábrázolása



III. A változók szerkezete (a szimbólumtáblában)

1. A skalár változók a szimbólumtábla egy a 16633-34 byte-páron levő címtől kezdődő tartományon helyezkednek el. (Ld. 3.3. Fejezetben!) E táblázatban minden egyes változóhoz tartozik egy őt leiró mező. Ez egy közös (típustól független) résszel kezdődik:
   1. byte: típusjelző
   2. byte: a nevének 2. betűje
   3. byte: a nevének 1. betűje

A további már típustól függő hosszban (értelmezésben) tartalmazza a változó értékét:

típusjelző	egész 2	szöveg 3	egyszeres 4	dupla 8
4. byte:	alsó	hossz	legalsó	1. byte
5. byte:	felső	alsócím	középső	2. byte
6. byte:	-	felsőcím	felső	3. byte
7. byte:	-	-	exponens	4. byte
8. byte:	-	-	-	5. byte
9. byte:	-	-	-	6. byte
10. byte:	-	-	-	7. byte
11. byte:	-	-	-	exponens

Az exponens 128-cal eltolt érték (a tényleges exponens érték + 128).
(A típusjelző éppen megegyezik az "értéket" leíró mező hosszával.)

2. A tömböket külön a 16635-36 rendszerváltozó által mutatott területre teszi a HT-1080Z/64 BASIC értelmezője. (Ld. 3.3. Fejezetben!) E táblázat egy eleme az alábbi adatokat tartalmazza:

   1. byte: típusjelző
   2. byte: a nevének 2. betűje
   3. byte: a nevének 1. betűje
   4.-5. byte: a tömb adatainak össz helyfoglalása
   6. byte: indexeinek száma
   7. byte-tól kezdődő byte-párok az egyes maximális indexértékeket tárolják. Ezt követik a tömb egyes elemeinek értékei típustól függő (a fejezet I., II. paragrafusában leírt) számú byte-on.

3.6 KARAKTERKÉSZLET

A karakterkészlet az ASCII (American Standard Code for Information Interchange) kódon alapul, de attól kissé eltér (pl. grafikus karakterek esetén).

kód	karakter	
00	blank	
01	CTRL/A	
02	CTRL/B	
03	CTRL/C	
04	CTRL/D	
05	CTRL/E	
06	CTRL/F	
07	CTRL/G	
08	CTRL/H	1 karakterrel visszalép és töröl
09	CTRL/I	
10	CTRL/J	linefeed (soremelés)
11	CTRL/K	új sort kezd
12	CTRL/L	új sort kezd
13	CTRL/M	RETURN (új sor)
14	CTRL/N	
15	CTRL/O	
16		nem használjuk
17		nem használjuk
18		nem használjuk
19		nem használjuk
20	speciális jelkészlet kikapcsolása	
21	speciális jelkészlet bekapcsolása	
22		nem használjuk
23	CTRL/W	ritkított kiírás
24	CTRL/X	visszalép 1 karakterrel
25	CTRL/Y	előrelép 1 karakterrel
26	CTRL/Z	1 sorral lejjebb lép
27	CTRL/[	1 sorral feljebb lép
28	CTRL/\	64 kar.=sor és HOME
29	CTRL/]	kurzor a sor elejére áll
30	CTRL/↑	a kurzortól jobbra eső sorrészt törli
31	CTRL/_	a kurzor mögötti képernyőrészt törli

kód	karakter	
32		szóköz
33	!	felkiáltójel
34	"	idézőjel
35	#	hashmark
36	$	dollárjel
37	%	százalékjel
38	&	et jel
39	'	aposztróf
40	(	nyitózárójel
41	)	csukózárójel
42	*	csillag
43	+	plusz
44	,	vessző
45	-	mínusz
46	.	pont
47	/	per-jel
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	kettőspont
59	;	pontosvessző
60	<	kisebbjel
61	=	egyenlőségjel
62	>	nagyobbjel
63	?	kérdőjel

64	@	
65	A	
66	B	
67	C	
68	D	
69	E	
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[	szögletes nyitózárójel
92	\	
93	]	szögletes csukózárójel
94	↑	felnyíl
95	_	aláhúzás

96	'	(SHIFT+7)
97	a	
98	b	
99	c	
100	d	
101	e	
102	f	
103	g	
104	h	
105	i	
106	j	
107	k	
108	l	
109	m	
110	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
120	x	
121	y	
122	z	
123	{	kapcsos nyitózárójel
124	|	
125	}	kapcsos csukózárójel
126	~	
127		

A grafikus karakterek kódjai (egy karakter 3*2 képpont) 128+B alakban állíthatóak elő, ahol B a pontoknak megfelelően a következő lehet

1	2
4	8
16	32

vagy kettős hatvány alakban

0	1
2	3
4	5

Így például a nulla és a harmadik pontot szeretném felkapcsolni, akkor B=1+8=9, azaz az utasítás: `PRINT CHR$(128+9).`

192-255 : nincs látható képük, de ezek kód-192 db szóköz kiírását eredményezik `PRINT CHR$(kód)` végrehajtásakor.

3.7 HIBAJELZÉSEK

Kód	Rövidítés	Jelentés
1	NF	A BASIC program **NEXT** utasítást akar végrehajtani a megfelelő **FOR** utasítás nélkül. Például **NEXT I**-t **FOR I=. . .** nélkül.
2	SN	Szintaktikus hiba. A BASIC értelmező parancs beírásakor vagy utasítás végrehajtásakor jelzi.
3	RG	**RETURN** utasítást akar végrehajtani megelőző **GOSUB** utasítás nélkül.
4	OD	**READ** utasítást akar végrehajtani, miután a megfelelő **DATA** utasításból már beolvasta az összes adatot, vagy **INPUT#** utasítást és a kazettán nem található több adat.
5	FC	Hibás argumentumú utasítás (pl. negativ dimenzió, 0 vagy negativ szám logaritmusa, általában valamilyen hibás függvényhívás).
6	OV	Túlcsordulás: a művelet eredménye nem ábrázolható, túl nagy.
7	OM	Nincs több tárolóhely (a tömbök túl nagyok, vagy a program túlságosan hosszú).
8	UL	Nem létező sorszámú sorra hivatkozás **GOTO, GOSUB, RUN, ON, RESUME** utasításban.
9	BS	Indextúllépés: a tömb olyan indexű elemére hivatkozunk, amely kívül esik a **DIM** utasításban megadott tartományon, túl nagy, vagy túl kicsi.
10	DD	A program már megadott dimenziójú tömb méretét **DIM** utasítással újra akarta definiálni. Vagy egy **DIM** utasítást kétszer hajtott végre, vagy pedig már a **DIM** utasítás előtt is hivatkozott a tömbre és így automatikusan elfoglalódott a megfelelő hely.
11	/O	Nullával osztás.
12	ID	Az utasítást parancs üzemmódban nem lehet használni (sorszám nélküli **INPUT** utasításra jelzi.)
13	TM	Típuskeveredés: az utasításban hibás típusú változót, függvényt használtunk.
14	OS	Nincs több hely szövegek tárolására (a BASIC alaphelyzetben 50 byte-ot használ szövegek tárolására - változók+munkaterület -, ezt a **CLEAR** utasítás segítségével lehet megnövelni).

15	LS	Szöveg típusú változóban nem lehet 255-nél több karakter.
16	ST	Ilyen bonyolult szöveg kifejezést nem tud kiértékelni.
17	CN	Nem végrehajtható a **CONT** utasítás (a program hibajelzéssel állt meg, vagy javítottunk a program valamelyik sorában vagy a programot még egyszer sem hajtottuk végre).
18	NR	Hibakezelő szubrutin végén nem a **RESUME** utasítást hajtottuk végre.
19	RW	**RESUME** utasítást akart végrehajtani megelőző hiba előfordulás nélkül.
20	UE	**ERROR** utasítás nem létező hibakóddal.
21	MO	Hiányzó operandus a kifejezésben.
22	FD	Hiányos file kazettán (beolvasási hiba).

Függelék

F.1 SZÁMÍTÁSTECHNIKAI FOGALMAK MAGYARÁZATA

Néhány, a könyvben előforduló fogalomra adunk ebben a függelékben magyarázatot. Célunk nem a fogalmak pontos - formális - definiálása, csupán a számítástechnikával ismerkedőknek szeretnénk segítséget nyújtani.

Fogalom	Magyarázat
Bit	Digitális számítógépek legkisebb információs egysége (bináris számjegy, értéke 0, vagy 1).
Byte	A memória legkisebb címezhető egysége, hossza 8 **bit**. (Többféle információt tartalmazhat: karakter, szám, utasítás.)
Hardver	Egy számítógép, mint eszköz, minden áramkörével, tartozékával együtt.
Hordozhatóság	A programok azon tulajdonsága, amely lehetővé teszi, hogy más gépen is változtatás nélkül(vagy minimális változtatással) futtatható legyen; nem érzékeny az esetleges "nyelvjárásbeli" eltérésekre.
Implementáció	Egy programnak egy adott gépre készült, annak specialitásait figyelembe vevő változata.
Interpreter (értelmező)	Olyan programozott eszköz, amely a számítógép alap utasítás-rendszerétől különböző utasításokat formai és tartalmi elemzésükkel egyidőben végrehajtja. A **HT-1080Z/64** esetében ezt a gépi kódú programot a ROM tartalmazza.
K	A számítástechnikában használatos dimenzió nélküli mennyiség, $2^{10}=1024$.
Karakter	Kódrendszerek által definiált jelhalmazok eleme. (A **HT-1080Z/64** az ASCII kódrendszert használja.)

Kettes komplemens kód	Az előjeles számok olyan bináris ábrázolása, amely szerint a legnagyobb helyiértékű **bit** az előjel (=0, ha pozitiv, 1, ha negativ a szám); negativ számoknak (n<0) megfeleltetjük a $2^{16}+n$ előjel nélküli számot. Az ábrázolás kényelmes aritmetikai műveletvégzést biztosít, mert mechanikusan végezhető az összeadás, kivonás az előjel nélküli számok mintájára.
Kurzor	A képernyőn levő speciális jel, amely az aktuális be- vagy kimeneti információ képernyőn elfoglalt pozicióját jelzi. (A **HT-1080Z/64** esetén egy aláhúzás-jel, vagy pedig egy villogó négyzet.)
Lebegőpontos	Számábrázolási forma valós számokra, amely két funkcionális részből áll: mantissza és karakterisztika. A mantissza egy rögzitett nagyságrendű valós szám (a **HT-1080Z/64** esetén ez a szám /m/: $0.5 < m < 1$); a karakterisztika ennek szorzófaktora, az alapszám (a **HT-1080Z/64** esetén 2) egész kitevőjű hatványa; a valós számot ez a szorzat reprezentálja. Előnye: ezután a valós számok nagyságrendtől függetlenül leképezhetők adott számú **byte**-ra valamilyen pontossággal.
Mikroprocesszor	A mikroszámítógép központi utasítás értelmező - és műveletvégző áramköre egy nagy integráltságú áramköri tokban realizálva.
Mikroszámítógép	Nagy integráltságú félvezető elemekből összeállított kis méretű és energiafogyasztású, olcsó, megbizható, igénytelen számítógép kategória.
Periféria	Azon berendezések gyűjtőneve, melyekkel az ember-gép dialógus megvalósul, ill. amelyek nagy tömegű adat tárolását teszik lehetővé. Ilyenek pl: billentyűzet, képernyő, nyomtató, mágneslemezes tároló, X-Y rajzoló, stb.
RAM	A számítógépek munkatárolója (Random Access Memory). Ez szolgál a felhasználására. Nevét onnan kapta, hogy az információ elérési ideje nem függ annak memóriabeli helyétől.

SZÁMÍTÁSTECHNIKAI FOGALMAK MAGYARÁZATA

RESET — A gép alapállapotba hozása a 0-ás címen kezdődő ROM-rutin közvetlen elindításával.

ROM — Csak olvasható információtároló memória (Read Only Memory). Bizonyos számítógépek esetén ROM memóriában van pl. a BASIC interpreter.

Szemantika — Az utasítások végrehajtásának mikéntje, az az algoritmus, amely pontosan leírja, hogy hogyan hajtódik végre az adott utasítás.

Személyi számítógép — Olyan mikroszámítógép, amely egyszerűsége,ára és egyéb paraméterei alapján sok ember számára elérhető és birtokolható. Jellegzetes perifériái: a billentyűzet, a képernyős megjelenítő (ami általában közönséges televízió), és a kazettás magnó.

Szintaxis — Azon szabályok összessége, amelyek definiálják egy programozási nyelven írt program formai helyességét. (Programnyelvi helyesírási szabályzat.)

Szoftver — A számítógép működtetését és felhasználását biztosító - esetleg különböző szintű - programtermékek összessége.

A HT-1080Z/64 KIVEZETÉSEI

F.2 A HT-1080Z/64 KIVEZETÉSEI

Buszcsatlakozó

50 pólusú csatlakozó, a kétoldalú nyomtatott áramköri panel legfontosabb jeleinek kivezetése.

Kivezetések — Nyomtatott áramköri panel

2	4	. . .	50
1	3	. . .	49

Az egyes kivezetések jelentése:

1 - GND (föld: 0 V)	26 - A10	
2 - GND (föld: 0 V)	27 - A13	
3 - A7	28 - A11	
4 - A6	29 - A12	
5 - A5	30 - órajel	(kb. 1.78 MHz)
6 - A4	31 - $\overline{\text{PINT}}$	(Interrupt vonal)
7 - A3	32 - Nincs bekötve	
8 - A2	33 - Nincs bekötve	
9 - A1	34 - $\overline{\text{PHLDA}}$	(Proc. Hold Acknowledge)
10 - A0	35 - $\overline{\text{PHANTOM}}$	
11 - D5	36 - $\overline{\text{HALT}}$	(Halt Acknowledge)
12 - D2	37 - $\overline{\text{PWAIT}}$	(Proc. Wait)
13 - Nincs bekötve	38 - $\overline{\text{IORQ}}$	(I/O Request)
14 - D1	39 - $\overline{\text{PHOLD}}$	(Proc. Hold)

15 - DO	40 - $\overline{WR}$	(Proc. Write)
16 - D3	41 - $\overline{RD}$	(Proc. Read)
17 - D7	42 - $\overline{CCDBS/STADBS}$	(Contr.and Status Bus Disable)
18 - D6	43 - $\overline{MREQ}$	(Memory Request)
19 - +5V	44 - $\overline{DODBS/ADDBS}$	(Data and Addr.Bus Disable)
20 - D4	45 - $\overline{M1}$	(First State of Instruction Cycle)
21 - A15	46 - $\overline{RESET}$	(CPU Reset)
22 - A8	47 - $\overline{RFSH}$	(Dynamic Memory Refresh)
23 - A14	48 - $\overline{NMI}$	(Non Maskable Interrupt)
24 - A9	49 - GND	(föld: 0 V)
25 - Nincs bekötve	50 - GND	(föld: 0 V)

Nagy vonalakban az egyes kivezetések funkciója (akiket ennél részletesebben érdekel ez, azoknak ajánljuk az Ipari Informatikai Központ által kiadott Z80 sorozat CPU-val foglalkozó könyvét):

1,2,49,50:	0V
19:	+5V
3-10,21-24,26-29:	Cím vonalak (a memória, illetve külső eszköz megcímzésekor a cím itt leolvasható).
11,12,14-18,20:	Adat vonalak (a CPU és a külső eszközök közötti adatforgalom itt jelenik meg, illetve megszakításkor a végrehajtandó utasítás kódja vagy a megszakítás vektor címének alsó 8 bitje ide kerül (IT0, illetve IT2 mód).
30:	Kivezetett órajel (kimenet).
31:	Külső eszköz megszakítás igényét ennek 0 V-ra helyezésével kezdeményezi (bemenet).
36:	A CPU státusza HALT utasítás végrehajtása után, megszakítás vagy RESET hatására tud csak továbblépni (kimenet)
37:	A CPU-t tetszőleges ideig várakozásra készteti, pl. egy lassú külső berendezés (bemenet).
38,45:	A CPU a megszakítás kérést e vonalakon nyugtázza (kimenet).
39:	Külső eszköz itt jelzi igényét a buszra (bemenet).
40:	Író utasítást hajt végre a processzor (kimenet).
41:	Olvasó utasítást hajt végre a processzor (kimenet).
43:	A tárhoz forduláskor aktiv (kimenet).
46:	Aktivizálására a CPU RESET állapotba kerül (bemenet).
48:	Nem maszkolható megszakítás kezdeményezésére szolgál, (RST 66H végrehajtásával) RESET-et hajt végre (bemenet).

A 20 pólusú csatlakozó (párhuzamos ki-bemenet)

Alkalmas 2x8 biten TTL szintű jelek fogadására, illetve kiadására, valamint 2-2 föld és hangszóró kimenetet is tartalmaz.

1	3	5	7	9	11	13	15	17	19
2	4	6	8	10	12	14	16	18	20

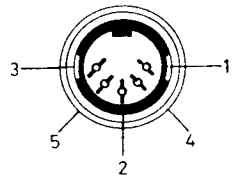
Az egyes kivezetések jelentése:

1 - R14 B0	11 - R15 B0
2 - R14 B1	12 - R15 B1
3 - R14 B2	13 - R15 B2
4 - R14 B3	14 - R15 B3
5 - R14 B4	15 - R15 B4
6 - R14 B5	16 - R15 B5
7 - R14 B6	17 - R15 B6
8 - R14 B7	18 - R15 B7
9 - A,B,C hangcsatorna	19 - GND (föld: 0 V)
10 - összegezett jel	20 - GND (föld: 0 V)

A kapu 2x8 bitje egy-egy programozható regiszteren keresztül érhető el (R14,R15).

A HT-1080Z/64 KIVEZETÉSEI

A második magnó DIN csatlakozója



Az egyes kivezetések jelentése:

1 - Távvezérlés

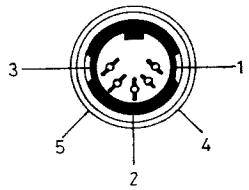
2 - Jel GND (föld: 0 V)

3 - Távirányítás

4 - Input

5 - Output

A Monitor/Video Ki DIN csatlakozója



Az egyes kivezetések jelentése:

1 - +5V

2 - Video Ki

5 - GND (föld: 0 V)

HT-1080Z HANGGENERÁTOR TÁBLÁZAT

F.3 HT-1080Z HANGGENERATOR TÁBLÁZAT

ZENEI HANG	FREKVENCIA	R1,R3,R5	RO,R2,R4
A	27.50	15	127
B	29.14	14	160
H	30.87	13	206
C	32.70	13	8
CISZ	34.65	12	76
D	36.70	11	156
DISZ	38.89	10	245
E	42.21	10	87
F	43.65	9	195
FISZ	46.24	9	55
G	49.00	8	178
GISZ	51.92	8	53
A	55.01	7	191
B	58.27	7	80
H	61.73	6	231
C	65.40	6	132
CISZ	69.30	6	38
D	73.41	5	206
DISZ	77.81	5	122
E	82.39	5	44
F	87.34	4	225
FISZ	92.52	4	155
G	98.01	4	89

HT-1080Z HANGGENERÁTOR TÁBLÁZAT

ZENEI HANG	FREKVENCIA	R1,R3,R5	RO,R2,R4
GISZ	103.79	4	27
A	109.96	3	224
B	116.54	3	168
H	123.54	3	115
C	130.80	3	66
CISZ	138.61	3	19
D	146.82	2	231
DISZ	155.61	2	189
E	164.78	2	150
F	174.54	2	113
FISZ	184.89	2	78
G	195.84	2	45
GISZ	207.78	2	13
A	439.86	0	248
B	466.18	0	234
H	493.60	0	221
C	524.45	0	208
CISZ	553.73	0	197
D	586.48	0	186
DISZ	623.34	0	175
E	661.12	0	165
F	699.26	0	156
FISZ	742.07	0	147
G	784.78	0	139
GISZ	832.71	0	131

HT-1080Z HANGGENERÁTOR TÁBLÁZAT

ZENEI HANG	FREKVENCIA	R1,R3,R5	RO,R2,R4
A	879.92	0	124
B	932.35	0	117
H	981.68	0	110
C	1048.89	0	104
CISZ	1113.11	0	98
D	1172.96	0	93
DISZ	1239.60	0	88
E	1314.28	0	83
F	1398.53	0	78
FISZ	1474.12	0	74
G	1558.36	0	70
GISZ	1652.80	0	66
A	1759.44	0	62
B	1848.90	0	59
H	1983.36	0	55
C	2097.79	0	52
CISZ	2226.22	0	49
D	2479.20	0	46
DISZ	2479.20	0	44
E	2660.61	0	41
FISZ	2948.24	0	37
G	3116.71	0	35
GISZ	3305.61	0	33

TÁRGYMUTATÓ

Kiadó: LSI Alkalmazástechnikai Tanácsadó Szolgálat
Felelős kiadó: dr. Kovács Magda
Engedélyszám: 43074

Készült: MÁTRAINVEST GT Nyomda
Felelős vezető: dr. Csák Máté

Nyomdai ív: 20,5 A/5