Gépi kódú programozás kezdőknek

TV-Computer

Ludányi László

Gépi kódú programozás kezdőknek

NOVOTRADE, 1989

Lektorálta: Beszeda Tamás
Szerkesztette: Stankovics János

A kiadásért felel RÉNYI GÁBOR
a Novotrade Rt. vezérigazgatója

Budapest, 1989

ISBN 963 585 019 0

Készült a Somogy Megyei Nyomdaipari Vállalat Kaposvári Üzemében

Felelős vezető: Mike Ferenc igazgató

TARTALOM

BEVEZETÉS

Miért nehéz a gépi kódú programozás? *És valóban az?*

A kérdés -- de méginkább a rá adott válaszban a nehézségek magyarázata -- azokhoz szól, akik már régóta foglalkoznak valamilyen személyi számítógéppel, írogattak már rövidebb-hosszabb programokat is ***BASIC***-ben, és pl. a gyors és sziporkázó játékprogramjaikról tudják, hogy azok gépi kódban készültek. A játékprogramok sokszor valóban briliáns kidolgozottsága és gyorsasága szokta elöször felkelteni a gépi kódú programozás iránti érdeklödést. És nem utolsósorban az, hogy jól hangzik, izgalmasan más, mint a BASIC. A személyi számítógép tulajdonosok körében a *"felnötté válást"* jelenti, ha valaki gépi kódú programokat ír.

A személyi számítógépek gépi kódú programozásának elönyei vitathatatlanok. A gyors müködés és a tömörség a legismertebb vonások (a legtöbb játékprogram BASIC-ben nem is férne el a gép memóriájában). Ha ehhez még hozzátesszük, hogy a személyi számítógépek igazi képességei, fizikai adottságai maradéktalanul csak gépi kódban érhetök el -- teljesen egyértelmü, hogy mindenkinek, akiben már felcsillant az érdeklödés szikrája, csak javasolni tudom a gépi kódú programozás tanulását.

Miért nehéz?

Kezdetben, a sok motiváció ellenére, nagyon sok a hosszadalmas kínlódás, keserü sikertelenség a jellemzö. Éppen ezért tapasztalható részleges, de jogos idegenkedés is. Sokan, néhány sikertelen próbálkozás után visszatérnek a jól ismert BASIC-hez. A problémák eredete két nagyon is egyszerü körülményben kereshetö!

A személyi számítógépek gyártói a géppel együtt szállítanak egy ún. BASIC *operációs rendszert,* amely rendkívül hatékonyan és kulcsszavaival nagyon emberközeli módon támogatja a gép szinte azonnali használatát. Ez a BASIC rendszer -- mint felhasználói környezet -- egyrészt nagyon sokoldalú, másrészt biztonságos is. Hibakezelö rutinjai nemcsak kivédik a felhasználó tévedéseit, hanem differenciált hibaüzenetekkel e tévedések jellegére és helyére is utalnak. Így megkönnyítik a javítást, a további tanulást.

Hatékony képernyőkezelő, kazetta-, nyomtató- és hanggenerátor-kezelő -- beépített gépi kódú -- programok segítik a felhasználó munkáját.

Elhagyva a BASIC rendszert, mint egy nagy, kényelmes és biztonságos hajót, rideg közvetlenséggel találjuk magunkkal szemben a ***mikroprocesszort*** és a ***memóriát***. Megszűnik a védettség, hibakezelés nincs, hiba pedig természetesen lehet, sőt van. Programunk, sőt egész gépünk is lépten-nyomon eltérül, elszáll, esetleg több órás munkát kezdhetünk elölről! A mikroprocesszor és a memória természetesen nem sérült meg, a gépben minden úgy történt, ahogy kellett, csak a programunk nem volt egészen az, amit mi akartunk. Így aztán az eredmény is siralmas.

De még ha jól is ment a dolog, nagy kín, hogyan lehet felvenni kazettára vagy mágneslemezre. A BASIC ui. nem képes közvetlenül a mi gépi kódú munkáink kezelésére. Így állunk tehát a BASIC-kel és ez csak az egyik dolog.

Mi tehát a teendő?

A hajó analógiával élve, semmi esetre sem ugorjunk fejest az óceánba! A nagy biztonságot nyújtó BASIC helyett biztosítani kell egy olyan felhasználói rendszerkörnyezetet, amely kifejezetten a gépi kódú munkát támogatja. Ez maga is egy gépi kódú program, amelynek megírására az első időszakban még nem vállalkozhatunk. A másik teendő pedig a tanulás. Szorgalmas munkával meg kell ismernünk a mikroprocesszort, s annak utasításkészletét. A BASIC-ben megszokottól némileg eltérő, megfontoltabb logikára is szükség van -- erről azonban majd később.

Könyvünk igyekszik végigvinni az Olvasót azon az úton, amelyet végigjárva a VIDEOTON TV-Computer önálló gépi kódú programozásába kezdhet. A munkát támogató rendszerkörnyezetet készen kapja az Olvasó, egy ***MONITOR*** nevű gépi kódú program formájában. Erről is később lesz szó. A tanulást a könyv első részében rendszerezett ismeretekkel, utána részletesen kidolgozott mintaprogramokkal segítjük.

A könyv a lehetőségeken belül igyekszik minden lényeges kérdésre kitérni. A tanulás mellett kézikönyvként is szolgál, ami a haladó programozók munkáját is segíti.

1. ÁLTALÁNOS ISMERETEK

1.1 A számítógép működése - testközelből

A ***BASIC*** egy magas szintű programozási nyelv. Segítségével az emberi nyelvhez közelálló kulcsszavakkal fogalmazhatjuk meg a számítógéppel megoldandó feladatokat. A végrehajtás persze ilyenkor is gépi kódban történik, de a tényleges működés és a BASIC utasításssorozatok között olyan nagy a logikai szintkülönbség, hogy az előbbi a felhasználó számára szinte teljesen rejtve marad. A BASIC utasításssorozat végrehajtása egy nagy terjedelmű, igen összetett értelmező- és fordítóprogram közreműködésével zajlik, de ezzel a munkával a felhasználónak nem kell törődnie.

Ezzel szemben a gépi kódú programozás során kiiktatjuk a beépített közvetítő programok munkáját, nem használjuk a BASIC kulcsszavait, figyelmünket tisztán a ***gépi szintű működésre*** fordítjuk!

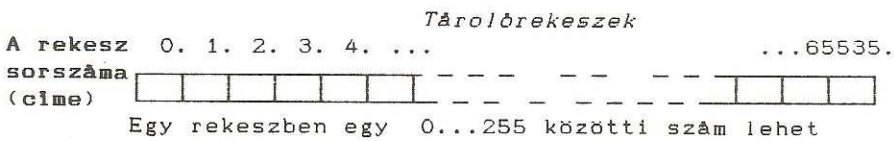
Ismerkedjünk meg -- alapfokon -- ennek lényegével!

Kezdjük a ***memóriával*** (l. 1. ábra).

Képzeljük úgy, hogy a számítógép igen sok kis tárolórekeszt tartalmaz, amelyeket sorszámukkal különböztetünk meg. Egy-egy ilyen rekeszben *egyetlen számot* helyezhetünk el. Csak számot, és nem is akármilyet: egy rekeszben csak a *0...255* közötti számok egyike lehet.

A tárolórekeszek számítástechnikában használt neve: ***byte*** (ejtsd: bájt), a rekeszek sorszámát pedig azok *címének* mondjuk.

Egy számítógépben a rendelkezésre álló rekeszek száma a típustól függően nagyon különböző lehet. Ez az adat a gép egyik lényeges mennyiségi és minőségi jellemzője.



1. ábra ***A memória alapfokú logikai modellje***

Ezeknek a rekeszeknek az összessége alkotja a számítógép memóriáját.

A tárolóelemek számát *kbyte*-okban (ejtsd: kilobájt) fejezzük ki, ugyanúgy, ahogy a nagyobb távolságokat km-ben. Csak itt 1 kbyte 1024 db memóriarekeszt jelent. A szám azért nem 1000, mert a számítástechnikában kulcsfontosságú a 2-es számrendszer és a nagyobb számolási egységet a 2 alapszám tizedik hatványaként állapítjuk meg. Ez egyszerű 2-vel való szorzásokkal adódik:

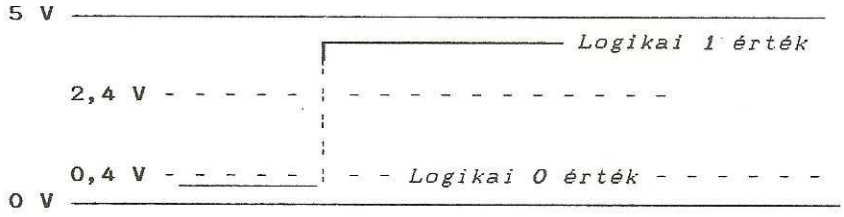
2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1024	512	256	128	64	32	16	8	4	2	1

Tehát: *1024 db memóriarekesz 1 kbyte.*

Ha egy számítógép pl. 64 kbyte-os, ez azt jelenti, hogy benne 64 * 1024 = 65536 db tárolórekesz áll a rendelkezésünkre. Ezt a nem kevés helyet néhány integrált áramkör valósítja meg. Ennek meglehetősen bonyolult részleteivel azonban nekünk, felhasználóknak nem kell törődnünk.

Nyilvánvalóan felmerülhet azonban a kérdés, hogy az egy memóriarekeszben tárolható szám miért éppen 0...255 között lehet? Ennek tisztázása azért is fontos, mert közelebb visz minket az előbb már említett 2-es számrendszerbeli számoláshoz.

A tárolást olyan miniatűr áramköri egységek valósítják meg, amelyeknek kimeneti vezetékén vagy 0...0,4 V között, vagy pedig 2,4...5,0 V között van a feszültség (l. 2. ábra).



2. ábra ***1 bites tárolóelem logikai modellje***

A *digitális áramkörökben*, helyes működés esetén, *közbülső értékek nem fordulhatnak elő*, változás esetén az átmenet nagyon gyorsan következik be. Kijelenthető, hogy

egy-egy ilyen vezeték bármely időpontban vagy az *alacsony*, vagy pedig a *magas* feszültségű állapotban van. Ezeknek a feszültségszinteknek számítástechnikai szempontból csak annyi a jelentőségük, hogy egyértelműen megkülönböztethető kétféle állapotot képviselnek. Rendeljük hozzá ezekhez a *0* és az *1* számokat úgy, hogy az *alacsonyabb feszültség logikai értéke 0, a magasabb feszültségé pedig 1 legyen.*

Az *1 byte*-os tárolóelem *nyolc* ilyen áramkörből áll, ezeket ***biteknek*** nevezzük. A bit szó az angol *binary digit* kifejezés rövidítése, jelentése bináris számjegy. 1 bit értéke -- az elektromos feszültségszinttől függően, de most már attól teljesen elvonatkoztatva -- 0 vagy 1 lehet. A tárolórekeszeket alkotó nyolc bit együttesen már sokféle különböző állapotban lehet, s ezek képviselik a rekeszben található tényleges számot. Ehhez a következő egyszerű gondolatmenettel jutunk.

A nyolc bit értékét egymás mellé írjuk és balról jobbra haladva a ***b7...b0*** jelekkel különböztetjük meg. A hozzájuk tartozó 0 vagy 1 értéket pedig egy 8 jegyű szám számjegyeinek tekintjük (1. 3. ábra).

	8 db 1 bites tárolóelem								
	bit	bit	bit	bit	bit	bit	bit	bit	
	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
Byte	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	
Helyi érték:	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1	*A byte-hoz rendelt számérték:*
A bitek lehetséges állapotai:	0	0	0	0	0	0	0	0	= 0
	0	0	0	0	0	0	0	1	= 1
	0	0	0	0	0	0	1	0	= 2
	0	0	0	0	0	0	1	1	= 2+1 = 3
	0	0	0	0	0	1	0	0	= 4
	.	.	.	.	.	.	.	.	.
	:	:	:	:	:	:	:	:	.
pl.:	0	1	1	0	1	0	1	0	= 64+32+8+2= = 106
	.	.	.	.	.	.	.	.	.
	:	:	:	:	:	:	:	:	.
	1	1	1	1	1	1	1	1	= 128+64+32+ +16+8+4+2+ +1 = 255

3. ábra ***A nyolc bit által alkotott 1 byte-os adat***

Az egymás után írt számjegyek helyi értékei most jobbról balra haladva nem 1-10-100-1000..., ahogy ezt a 10-es *(decimális)* számrendszerben megszoktuk, hanem a 2-es alapszám hatványai: 1-2-4-8-16-32-64-128. Lényeges továbbá, hogy az ilyen nyolcjegyű szám mindegyik jegye 0 vagy 1. Ezért az 1 byte-on ábrázolható szám egyszerűen úgy adódik, hogy vesszük sorra a biteket, s ahol 0 áll, azt kihagyjuk, ahol pedig 1, ott a helyi értékének megfelelő számot vesszük, s ezeket összeadjuk.

Nézzünk egy példát!

Írjuk le a nyolc bit egy tetszőlegesen kiválasztott állapotát! Legyen ez mondjuk: 1011 0110. Állapítsuk meg a hozzá tartozó számot az általunk megszokott 10-es számrendszerben!

Először jobbról balra haladva írjuk fel a 8 bithez tartozó helyi értékeket, majd alá az előbbi biteket:

b7	b6	b5	b4	b3	b2	b1	b0
128	64	32	16	8	4	2	1
1	0	1	1	0	1	1	0

Ezután adjuk össze az azokhoz a helyi értékekhez tartozó számokat, ahol a bináris számban 1 áll, a 0-hoz tartozókat hagyjuk ki:

128 + 32 + 16 + 4 + 2 = 182

és ezzel kész!

Ezt az ábrázolási formát ***bináris számnak*** nevezzük. Próbáljuk gyorsan megjegyezni a helyi értékek felírásában szereplő számokat!

Nyilvánvaló, hogy az egy byte-on megadható legkisebb értéket az adja, ha minden bináris számjegy 0 -- így összesen is 0-t kapunk. A legnagyobb szám pedig úgy adódik, hogy ha mindenhová 1-et írunk, ekkor az összegezés után éppen 255 lesz az eredmény. Minden más érték csakis ezek között lehet. Ugye egyszerű?

Megtehetjük, hogy két byte-ot egymás mellé teszünk, s az ezekben levő 2 * 8 = 16 bitet együttesen vesszük egy 16 jegyű bináris számnak. Ilyenkor a jobb oldali nyolc bitet tartalmazó byte az ***"alacsonyabb helyi értékű"***, a bal oldali pedig a ***"magasabb helyi értékű byte"*** nevet kapja. Ennek bitjei -- helyiértéküknek szerint -- most nem a fenti

1, 2, 4,... 128 számokat képviselik, hanem további 2-vel való szorzásokkal értéküket így alakul (a megkülönböztetés végett magukat a biteket is tovább számozzuk):

b15	b14	b13	b12	b11	b10	b9	b8
32768	16384	8192	4096	2048	1024	512	256

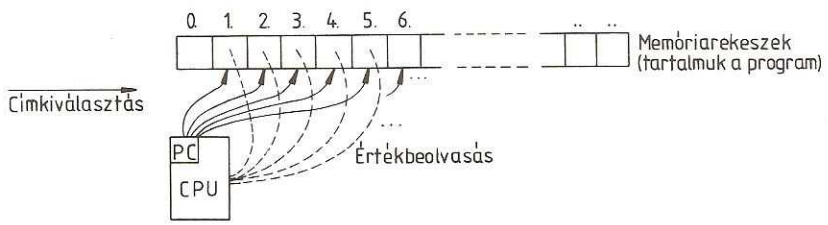
A 16 bit 0000 0000 0000 0000.....1111 1111 1111 1111 közötti összes állapota már 65536 féle lehet, így a korábban leírt módszerrel 2 byte-on a 0...65535 közötti számokat ábrázolhajuk. Az eljárást továbbfolytatva a 24 vagy akár a gyakoribb 32 bites rendszerekhez is eljuthatunk.

A gyakorlati munkában a számok ilyen kezelése ugyan logikus, de nagyon nehézkes lenne. Ezért ezzel a megoldással csak akkor élünk, ha kifejezetten a bitenkénti vizsgálat a célunk. A több bitből alkotott értékek kifejezésére általában a sokkal rövidebb és nagyon kifejező ún. ***hexadecimális*** számábrázolást használjuk! Ezzel a következő fejezetben ismerkedünk meg.

Az alapkérdéshez visszakanyarodva a számítógép memóriájáról eddig tehát azt tudjuk, hogy az címeikkel megkülönböztetett tárolórekeszek sokaságából áll és egy rekeszben a 0...255 közötti számok egyike lehet. Tudjuk azt is, hogy miért.

Menjünk tovább!

A memória mellett a számítógép legfontosabb alkatrésze, a gép ún. központi műveletvégző egysége, a ***mikroprocesszor***. Angol nevének *(**Central Process Unit**)* kezdőbetűi szerint sokszor írjuk (és mondjuk) röviden így: **CPU**. Ez a számítógép fáradhatatlan rabszolgája, amely többé-kevésbé közvetlen kapcsolatban áll a gép összes többi részegységével, így a memóriával is. Működésének lényege a következő (l. 4. ábra):



4. ábra ***A CPU és a memória kapcsolatának alapfokú logikai modellje***

A mikroprocesszor *PC* jelű belső számlálója minden pillanatban egy sorszámot (memóriacímet) tartalmaz. Ezt az értéket mi határozhatjuk meg. A **CPU** "megnézi" ("beolvassa") a memória kiválasztott rekeszének tartalmát, amely számára valamilyen elvégzendő műveletet jelent, mégpedig minden szám mást és mást. A mikroprocesszor azonnal elvégzi a beolvasott szám által előírt műveletet, azután címszámlálóját *(PC)* automatikusan 1-gyel megnöveli. Így a következő lépésben már az előbbi memóriarekesszel szomszédos, 1-gyel nagyobb című tárolóelem tartalmát olvashatja be. A **PC** megjelölés a *Programming Counter* angol kifejezés rövidítése, magyar jelentése: programszámláló.

*A memória egymás utáni tárolórekeszeiben levő számok a **gépi kódú utasítások**, ezek sorozatát nevezzük **programnak**.*

A számokat beolvasva, a mikroprocesszor a velük meghatározott tevékenységeket végzi el, sorban egymás után. Ez a *program végrehajtása.*

Az 1 byte - egy művelet megfeleltetés azonban csak nagyon szűkös lehetőségeket engedne meg. Éppen ezért sokszor nem egyetlen szám határozza meg önmagában az elvégzendő feladatot, hanem több, egymás után beolvasott tárrekesz tartalma. Ilyenkor a **CPU** az új művelethez tartozó, elsőként beolvasott számmal azt is azonosítja, hogy a pontos értelmezéshez még hány további byte tartalmát kell beolvasnia.

A számok között több olyan is van, amely arra utasítja a mikroprocesszort, hogy a következő adatot ne az 1-gyel nagyobb című tárrekeszből vegye, hanem valamely másik memóriacímtől folytatódjék a beolvasás. Így meghatározott *ugrások* hajthatók végre, s ezek segítségével a memória meghatározott részét betöltő programot a **CPU** többször is végrehajthatja.

*Azt nevezzük **programozásnak**, amikor a memória eredetileg üres tárolórekeszeibe, valamilyen kitűzött feladat megoldásához szükséges számok sorozatát tervezzük meg és írjuk be.*

Ehhez már csak azt kell tudni, hogy a mikroprocesszor összesen hányféle és milyen műveleteket képes elvégezni, és ezekhez milyen számok tartoznak!

A mikroprocesszor a memórián kívül a számítógép összes többi részegységével is kapcsolatban van. A billentyűzethez, a tv-kép megjelenítés vagy a hangképzés eszkö-

zeihez, de még a géphez kívülről kapcsolt magnetofonhoz, nyomtatóhoz is egy univerzális kapcsolatteremtő rendszeren: a *portokon* keresztül fér hozzá. A port ugyancsak angol szó, eredetileg portát, kaput jelent, s kissé átvitt értelemben a számítógép portjainak funkciója meg is felel ennek a jelentésnek. A portok is nyolcbites áramkörök, amelyek egyik oldalról a mikroprocesszorral, másik oldalról viszont egy-egy eszközzel állnak kapcsolatban. Mindegyik eszközhöz tartozik egy-két ilyen port, s ezek a **CPU** felügyelete alatt állnak. Adott esetben a mikroprocesszor a kiválasztott eszközhöz tartozó portokat mozgósítja, s rajtuk keresztül küld adatokat az eszköz felé, vagy fordítva: a **CPU** kérésére az eszköz által küldött adatot a port közvetíti a **CPU**-hoz. Ez a zseniális gondolat lehetővé teszi, hogy *a legkülönfélébb eszközök speciális működése a CPU oldaláról már egységesen 8 bites adatokkal legyen kézbentartható.*

Ezzel kész is. Az elvek, a számítógép működésének lényege -- ez. A **CPU** minden érvényes parancsot végrehajt, a tennivalók sorozatát a memóriából, az oda írt számok formájában kapja, s ennek végső eredménye és értelme az ember által kitűzött feladat elvégzése. Nincs ebben semmi csoda, nem kell mögötte valami különösen bonyolult dolgot látni, az egész rendszer logikailag ennyire egyszerű.

A hatékony programozási munkához persze ismerni kell az adott számítógépben megvalósítható fizikai lehetőségeket, ezek kihasználhatóságának módszereit. Tudni kell, hogy hogyan lehet a memória egyes tárolórekeszeibe számokat írni, az egyszer beírt számokat szükség esetén kijavítani, a számsorozat működőképességét kipróbálni, a portokat kezelni. Ehhez szükség van egy kifejezetten ezt a munkát támogató segédeszközre, amely a munka természetéből adódóan alapvetően különbözik a *BASIC*-től. Ennek egy egyszerű változata lesz az a *MONITOR* nevű program, amelyről később részletesen beszélünk.

1.2 Hexadecimális számok

A címtől ne ijedjünk meg! A szakkifejezés egészen egyszerű dolgokat takar.

Az első fejezetben addig jutottunk, hogy a *bináris számok* jól illeszkednek a számítógép elektronikus alapműködéséhez, de használatuk a gyakorlati munkában nehézkes lenne. Most megnézzük, hogyan lehet a byte-ok tartalmát egyszerűbben, kezelhetőbben, mégis kifejezőbben jellemezni.

Induljunk ki ismét a nyolc bitből. Ez könnyen kettéosztható, s az így nyert *négy-négy bit* által képviselt értéket ezután *egyetlen jellel* adjuk meg. Mint már tudjuk, a négy biten ábrázolható szám legkisebb értéke *0*, ez a *0000* bitállapotnak felel meg, a legnagyobb pedig az *1111* állapothoz tartozó szám, a helyi értékes módszer alapján adódó *15*.

Bit:	**b3**	**b2**	**b1**	**b0**	
Helyi érték:	8	4	2	1	
	0	0	0	0	= 0 + 0 + 0 + 0 = 0
	:	:	:	:	:
	:	:	:	:	:
	1	1	1	1	= 8 + 4 + 2 + 1 = 15

Ezek szerint a *0...15* értéket képviselő bitállapotok megkülönböztetéséhez *16 féle jelre* van szükség.

A 0...9 számok jelölésére célszerű továbbra is magukat a számjegyeket használni, a fennmaradó másik 6 érték kifejezéséhez viszont új szimbólumokra van szükség! A 10, a 11, a 12, a 13, a 14 és a 15 értékeket jelöljük a továbbiakban -- ugyanilyen sorrendben -- *A*, *B*, *C*, *D*, *E* és *F* betűvel! Lehetne más jeleket is használni, de ez is igen egyszerű és az egész világon ez terjedt el.

Az egybyte-os adat felét alkotó négy bit összes lehetséges állapotát illetően tehát a következő -- egy-egy jellel való -- egyszerű megkülönböztetéshez jutunk:

Helyi érték:	Bitek b3 8	b2 4	b1 2	b0 1	Érték a 10-es számrendszerben	Hexadecimális jegy
	0	0	0	0	0	0
	0	0	0	1	1	1
	0	0	1	0	2	2
	0	0	1	1	3	3
	0	1	0	0	4	4
	0	1	0	1	5	5
	0	1	1	0	6	6
	0	1	1	1	7	7
	1	0	0	0	8	8
	1	0	0	1	9	9
	1	0	1	0	10	A
	1	0	1	1	11	B
	1	1	0	0	12	C
	1	1	0	1	13	D
	1	1	1	0	14	E
	1	1	1	1	15	F

Az *egybyte-os adat nyolc bitjét* ezután nyilvánvalóan *két ilyen jellel* tudjuk kifejezni. Például az *11010110* bitállapotot így:

D6

Nézzük meg, hogy melyik számot jelenti ez a tízes számrendszerben. A *D* jel (hexadecimális számjegy) *13*-nak felel meg, azonban itt nyilvánvalóan nem 13-at jelent! Ha ui. megnézzük ismét a nyolc bit helyi értékét:

b7	b6	b5	b4	b3	b2	b1	b0
128	64	32	16	8	4	2	1

látjuk, hogy a *felső négy bithez (b7...b4)* tartozó mindegyik érték 16-szor nagyobb, mint ha az *alsó négy bit (b3...b0)* helyén lenne. Így aztán az általuk képviselt érték is 16-szorosa a hexadecimális számjegy saját értékének.

Pontosan megfelel ez annak, amit a tízes számrendszerben mindenki tud: egy kétjegyű szám második jegyét 10-zel kell szorozni. Ha pl. ezt írjuk: 23, akkor ebben a 2-es nem kettőt, hanem 20-at jelent. Ugyanez a helyzet a 16-os számrendszerben is, csak ***itt a második számjegyet 16-tal kell szorozni.***

Ezt figyelembe véve már könnyű megmondani a *D6* szám értékét: a *D*-nek megfelelő *13-at megszorozzuk 16-tal*, így megkapjuk a nagyobb helyi értékű jegy értékét, amihez

még hozzá kell adni az első jegyet. A *D6* hexadecimális szám 10-es számrendszerbeli megfelelője tehát: *13 * 16 + 6* és ez összesen *214*.

Az elmondottak nagyobb számokra is igazak: a tizes számrendszerben a 3., 4. számjegy értékét 100-zal, ill. 1000-rel kell szorozni, tehát mindig 10-szer akkora számmal -- a 16-os számrendszerben pedig minden újabb, nagyobb helyi értékű jegynél további 16-tal!

Így egy négyjegyű hexadecimális szám jegyeinek helyi értéke:

4. jegy	3. jegy	2. jegy	1. jegy
4096	256	16	1

Egyelőre a nyolcbites számoknál maradva -- játszunk tovább!

Hogy néz ki az 1 byte-on tárolt legnagyobb szám a hexadecimális számrendszerben? Tudjuk, hogy ez a szám a *255*, amelynek a bináris *1111 1111* felel meg. Mivel most mindkét 4-es bitcsoport csupa 1-ből áll, az értéküknek megfelelő *15*-öt az *F* szimbólummal fejezzük ki. A szám hexadecimális alakja tehát: *FF*. Az előbbi helyi értékek és átszámítási mód szerint ez 15 * 16 + 15, ami 255.

Próbáljunk ilyen számokat összeadni és kivonni! Végezzük el a következő műveletet:

```
  A2
+ 26
```

Az első helyi értéken nincs semmi gondunk: 6 + 2 = 8. A második -- *16-os* -- helyi értéken pedig az *A 10-et* jelent, amihez a 2-t hozzáadva 12-t kapunk. Ezt viszont *C*-vel jelöljük, tehát:

```
  A2      (10 * 16 + 2 = 162)
+ 26      ( 2 * 16 + 6 =  38)
----      -------------------
  C8      (12 * 16 + 8 = 200)
```

(Zárójelben odaírtuk a számok 10-es számrendszerbeli megfelelőjét.)

Most nézzünk egy nehezebbet: 89 + 59 = ? (Vigyázzunk: ezek is hexadecimális számok!)

9 + 9 az 18 lenne, azonban most *nem a 8-ast* kell leírnunk. 15-ig külön jeleink vannak, tehát megnézzük, hogy ***a kapott összeg mennyivel nagyobb a 16-nál*** *(ha éppen 16, akkor 0-val)*. Ezt írjuk le, és jegyezzük meg, hogy *átvitel képződött a nagyobb helyi értékre!*

```
   89
 + 59
 ----
    2
Átvitel +1 ┘
```

Példánkban 9 + 9 = ***18, ami 2-vel nagyobb a 16-nál***. A *2*-t leírjuk, a maradék *16* pedig átkerül a ***második helyre***, ahol a neki megfelelő értéket már természetesen csak egy *1*-essel kell figyelembe vennünk. Ugyanez történik a 10-es számrendszerben is, csak ott az előbbi eljárást a 10-es szám elérése után végezzük el.

A második helyen eredetileg *5 + 8 = 13*, amit a *D*-vel jelölnénk. Ehhez azonban még hozzá kell adnunk az első helyi értéken keletkezett *+1 átvitelt*. Mivel a *14*-nek az *E* felel meg, a végeredmény:

```
   89
 + 59
 ----
   E2
```

Vizsgáljuk meg és ellenőrizzük a következő összeadásokat:

```
  24      4C      79      A2      1C
+ 31    + 62    + 23    + 39    + 7A
----    ----    ----    ----    ----
  55      AE      9C      DB      96
```

Végül pedig ezt:

```
   93
 + 8A
 ----
?  1D   ???
Átvitel +1 ┘
```

Ezt meg kell beszélnünk! Most *8 + 9 = 17 miatt* a második helyre 1-et írtunk, így azonban *maradt egy +1 átvitel*. Csakhogy *1 byte-on belül* ***nincs már hová*** *átvinni*. Egy harmadik hexadecimális jegyhez további 4-es bitcsoport kellene, ám 1 byte csak 8 bitből áll. Papírra leírva persze egyszerű a helyzet, a végeredmény:

```
   93
 + 8A
 ----
  11D
```

Ha azonban ez az összeadás a memóriában, vagy általában *byte-os adatokkal* zajlik, akkor eredményként is csak a szám 1 byte-on elférő része jelenhet meg, ami nyilvánvalóan *helytelen (példánkban: 1D)*. A hiba nem nagy baj, ha tudunk róla! Most mindenesetre jól jegyezzük meg!

Nézzünk ezután egy-két kivonást:

```
  63          B9
- 21        - 26
  42          93
```

A bal oldali művelethez nem kell semmit hozzáfűznünk, sőt talán a másikhoz sem sokat (a *B* 11-et jelent).

Vizsgáljuk meg viszont a következő kivonást:

```
  71
- 25
```

Hogy 1-ből az 5-öt kivonhassuk, az 1-et most *nem 11-re* kell kiegészíteni, mint a 10-es számrendszerben tennénk, hanem a hexadecimális logikának megfelelően. *Az 1-hez most 16-ot kell hozzáadni, tehát az 5-öt a 17-ből kell kivonni.* Ettől kezdve már nincs különbség. Az első helyi értéken pluszként felhasznált 16-ot a második helyi értékről vettük kölcsön, így azt csökkenteni kell. Ez egy *negatív átvitelt* jelent, amit pl. a kivonandó jegyét 1-gyel növelve vehetünk figyelembe. Példánkban tehát a 7-ből nem 2-t, hanem 3-at vonunk ki.

```
          71
        - 25
          4C        (17 - 5 = 12, aminek a C
átvitel -1 ┘         felel meg)
```

Összeadással ellenőrizzük a kivonás helyességét:

```
  25
+ 4C
  71
```

Még érdekesebb (és tanulságosabb) a következő:

```
  38
- 43
  ?5
```

18

A 3-ból a 4-et nem lehet kivonni, ezért helyettesítjük *16 + 3 = 19*-cel. Így *19 - 4 = 15*, aminek az *F* felel meg. Azonban keletkezett egy negatív átvitel is, aminek hatása a *harmadik helyi értéken* jelenne meg, csakhogy nincs harmadik jegy -- legalábbis egybyte-os adatok esetén. El kell tehát fogadnunk, hogy *ilyen esetben a kivonás is rossz eredményre vezet.*

Most:

38	(3 * 16 + 8 = 56)
- 43	- (4 * 16 + 3 = 67)
F5	(15 * 16 + 5 = 245)

Itt is igaz, hogy a hiba többféleképpen kivédhető, csak tudnunk kell róla.

Az előbbi példának még egy fontos tanulságát kell megbeszélni. A kivonást 10-es számrendszerben elvégezve, a negatív számok használatával azt mondhatjuk, hogy *56 - 67= -11*. Vegyük észre, hogy a hexadecimális byte-os kivonásnál kapott -- helytelen -- *F5 = 245* eredmény éppen *11-gyel kevesebb a 256-nál*. Célszerű ezért ilyen esetben *az F5 számot negatívnak* venni, mégpedig a 10-es számrendszerbeli *(-11)*-nek megfeleltetni.

Ez annál is inkább indokolt, mert ha az *F5*-höz hozzáadjuk a 11-nek megfelelő hexadecimális számot (*0B*-t), akkor a *(-11) + 11 = 0* eredménnyel egybehangzóan:

F5
+ 0B
00

(Három hexadecimális helyi értéken persze *100* lenne a tényleges eredmény, de 1 byte-on nincs harmadik jegy, így valóban 00-t kapunk.)

Az F5 tehát jelenthet 245-öt, de (-11)-et is!
Vajon 1 byte-on belül minden számmal ez a helyzet?

Nézzünk újabb példát!

Az *A3* szám pozitív alapértéke: 163. Ez 93-mal kevesebb a 256-nál, vagyis (-93)-nak is megfelel. Ellenőrizzük, hogy így van-e! Adjunk hozzá egy kisebb és egy nagyobb számot a 16-os számrendszerben, pl. *39*-et, és *B7*-et:

A3	(-93)
+ 39	(+57)
DC	(-36)

Ez helyes, hiszen a *DC = 13 * 16 + 12 = 220*, ami valóban *(-36)*-nak tekinthető. A számolás természetesen pozitív ér-

tékeknek tekintve is helyes (163 + 57 = 220). Nézzük a másikat:

```
   A3      (- 93)
 + B7      (+183)
 ----      ------
   5A      (+ 90)
```

Ez is jó, hiszen a hexadecimális *5A* szám 10-es számrendszerbeli értéke valóban 90. Úgy tünik tehát, hogy a kétféle értelmezés nem vezet ellentmondásra.

A byte-os müveletekben megmutatkozó, elözökben tárgyalt kettöséget a számitástechnikában nagyon sokszor, igen elönyösen kihasználhatjuk. A TV-Computerben alkalmazott *Z80-as típusú mikroprocesszor* több utasitásában is érvényesiti ezt a lehetöséget -- azzal a szokásos leegyszerüsitéssel, hogy a 0...255 közötti számtartományt kettö osztja, s az ilyen utasitásokban a *0...127* értékeket *pozitívnak*, a *128...255* számokat pedig eleve *negatívnak* tekinti. Az eddigiek alapján látható, hogy az igy negativnak vett számok esetén a legnagyobb helyi értékü b7 bit értéke 1, pozitív számok esetén pedig 0. Ebben az értelmezésben tehát *a 7. bit a szám elöjelét képviseli.*

Ilyenkor egyszerü lehetöség van arra is, hogy egy számot a (-1)-szeresére változtassunk. Minden bit értékét meg kell forditani, s az igy kapott számhoz még hozzá kell adni 1-et.

Például:

```
          56 = 0101 0110                    ( 86)
megforditva:   1010 1001  =  A9
                          +  01
                          ----
                             AA             (-86)
```

Tehát az *56* hexadecimális szám *(-1)-szerese: AA.*
Ez az ún. *2-es komplemens* számábrázolási mód lényege.

A +01 járulék hozzáadásának szükségessége jól látható a 00 érték esetében. A 00 bitjeit megforditva az FF számot kapjuk, ami az eddigiek alapján is nyilvánvalóban a (-1)-nek felel meg, pedig a 0 szám (-1)-szerese is 0. Az FF-hez még a 01-et hozzáadva kapjuk a helyes 00 eredményt.

A 2 byte-on ábrázolt, 16 bites számokat természetesen 4 hexadecimális jeggyel adhatjuk meg. A legnagyobb, 65535-nek megfelelö, 16 db 1 bittel ábrázolt szám hexadecimális alakja: *FFFF.* A kettös értelmezhetöség itt is igaz, és pl. az FFFF szám (-1)-nek felel meg, hiszen hozzá 16 biten +1-et adva, 0-t kapunk:

```
  FFFF
+ 0001
------
  0000
```

Ezek a tulajdonságok nagyszerüen kiaknázható lehetőségeket hordoznak, ezért arra biztatom az Olvasót, hogy bennük ne csak bonyodalmakat lásson. A 8 vagy 16 bites számok furcsán önmagába záruló világa matematikai élmény.

Nyilván a kedves Olvasó is észrevette, hogy abban az esetben, ha a hexadecimális szám minden jegye *szám* (pl. 76), akkor formailag nem egyértelmű, hogy hexadecimális vagy tizes számrendszerbeli számról van-e szó. Folyamatos szövegben zavaró lehet még az is, ha a szám első jegye nagybetű. Elterjedt, hogy az egyértelműség végett a hexadecimális számokat egy utánuk írt H betüvel különböztetjük meg, a számot kezdő nagybetük esetén pedig elé még egy ún. *vezető 0-t* is írunk, pl.: *76H, 0B9H.*

Végül megjegyezzük, hogy az eddigiekben tárgyalt 2-es *(bináris)*, 10-es *(decimális)* és 16-os *(hexadecimális)* alapszámú számábrázoláson kívül a számítástechnikában használják még a *8-as számrendszert* is *(oktális számok)*. Ebben a rendszerben három bitből álló bitcsoport állapotát adják meg egyetlen jellel:

	Bitek			**Decimális**	**Jele a 8-as**
	b2	**b1**	**b0**	**érték**	**számrendszerben**
Helyi érték	**4**	**2**	**1**		
	0	0	0	0	0
	0	0	1	1	1
	0	1	0	2	2
	0	1	1	3	3
	1	0	0	4	4
	1	0	1	5	5
	1	1	0	6	6
	1	1	1	7	7

Ilyenkor tehát csak a 0...7 számjegyeket használjuk. Egy byte tartalmát három oktális helyi értéken adhatjuk meg. A legkisebb érték a *00 000 000* bitállapotnak megfelelő *000*, a legnagyobb pedig az *11 111 111* állapothoz tartozó *377*!

A négyféle számrendszer vegyes használata bizonyos esetekben indokolt lehet. Feltétlenül gondoskodni kell viszont ezek megfelelő jelekkel való megkülönböztetéséről! A leggyakrabban ez úgy történik, hogy a számjegyek után a

számrendszer kezdőbetűjét írjuk: a *bináris* számok után B-t, 8-as számrendszerbeli szám után O-t írunk (*oktális*), a *decimális* számokat az utánuk írt D betűvel, a *hexadecimális* számokat pedig H-val különböztetjük meg.

Ebben a könyvben kizárólag a decimális és a hexadecimális számokat használjuk, ezért megegyezünk abban, hogy csak a hexadecimális számokra hívjuk fel külön a figyelmet. Tízes számrendszerbeli számok esetén a D-t elhagyjuk. Tehát ezentúl pl. ha csak ennyit írunk: 63 -- ez mindig a megszokott decimális értéket jelenti. A 63 és a 63H jelöléssel az eddig oly zavaró félreérthetőség megszűnik.

Ezzel fejezetünk végére értünk. Sok és nagyon fontos dologról esett szó ebben a részben.

Ha valaki ezekkel a fogalmakkal korábban még nem találkozott, elegendő, ha először csak a hexadecimális számjegyeket tanulja meg, s hogy hogyan kell azokat a bitállapotoknak, ill. a bináris számoknak megfeleltetni, valamint a 10-es számrendszerbe átszámítani. Később, a programozási munka során, a fogalmak szinte önmaguktól is letisztulnak, megszokottá és nyilvánvalóvá válnak.

A következő oldalakon számítógépünk és a programozás legizgalmasabb és legfontosabb alkatrészével -- a *mikroprocesszorral* ismerkedünk meg.

1.3 A mikroprocesszor

A VIDEOTON TV-Computerekbe beépített ***Z80-as típusú CPU*** ebben a számítógép kategóriában az egyik leggyakrabban használt mikroprocesszor (Spectrum, HT, Enterprise stb). Gépi kódú programozásának ismerete tehát többféle számítógépben is lehetővé teszi a munkát -- hozzá csak az adott géptipus sajátosságait kell megismerni.

A következőkben -- kizárólag a programozás oldaláról -- részletesen megismerkedünk a Z80-as mikroprocesszorral.

1.3.1 Cím- és adatvonalak

A memóriához és a beépített vagy külső eszközökhöz való hozzáférést egy *16 bites, **címbusznak*** nevezett vezetékrendszer teszi lehetővé. Vezetéken természetesen nem okvetlenül huzalokat, de mégcsak nem is a nyomtatott áramköri lap fóliáit kell értenünk. Az elnevezés az elektromos összeköttetésen kívül speciális funkciójú áramköröket is magában foglal, amelyek a CPU és a többi eszköz számára lehetővé teszik ugyanahhoz a fizikai vezetékrendszerhez való konfliktusmentes hozzáférést is. A *bus* (ejtsd: busz) angol szó, a számítástechnikai szóhasználatban vonalat, csatornát jelent. A címvezetékek megfelelő hálózata valamennyi memóriát és portot közvetlenül összeköt a CPU-val. A korábbi fejezetekből tudjuk, hogy a 16-bites címbusz 65536 féle különböző bitállapotot vehet fel, így ennyi memóriarekesz közvetlen kiválasztását teszi lehetővé. Hexadecimálisan: 0000...0FFFFH közötti címeket adhat ki a CPU.

A korrekt működéshez jónéhány, ún. vezérlőjelre is szükség van. Ezek ismerete azonban programozási szempontból kevésbé fontos.

A címbusz a memóriarekeszek vagy portok kiválasztására szolgál. Ezek tartalmához viszont a CPU egy másik, *8 bites vezetékrendszer, **az adatbusz*** segítségével fér hozzá. A címvonalakon beállított sorszámmal (címmel), valamint néhány vezérlőjel segítségével a mikroprocesszor kiválaszt és aktívvá tesz egy memóriarekeszt vagy portot,

s ennek eredményeként a 8 bites adatvonalon a CPU közvetlenül hozzáfér a kiválasztott elem tartalmához. Az aktiválást kiváltó vezérlőjelek nélkül az egyes elemek tartalma el van zárva az adatvonalaktól.

Alapállapotban mindkét busz információtartalma nulla. Amikor a mikroprocesszor pl. a 8A7DH című memóriarekesz tartalmát akarja beolvasni, akkor néhány vezérlőjel hatásának érvényesítése után megszünteti a címbusz közömbös állapotát, s a 16 címvonalon a 8AD7H-nak megfelelő *1000 1010 1101 0111* bitértékeket állítja be. Ezzel a címkiadással és egy újabb vezérlőjellel eléri, hogy a következő pillanatban a teljes memóriából csak a megadott című rekesz kapcsolódjék az adatvonalakra. Ekkor az *adatbusz* korábbi közömbös állapota megszűnik, 8 bitjén a kiválasztott byte tartalma jelenik meg -- a CPU elvégezheti a beolvasást.

Fordított esetben -- amikor pl. a CPU adatot küld egy portra -- az eljárás teljesen hasonló. Sok tekintetben olyan ez, mint egy többfázisú útkeresztezödés forgalma és annak jelzölámpákkal való irányítása. Megfelelő időzítések révén mindenki eljut ahová akar, ugyanazon az úttesten többirányú, rendezett forgalom zajlik. Érezhetően kulcskérdés ebben a *vezérlés*, de az is, hogy mindenki kész ezek fogadására és a helyes reagálásra.

A biztonságosan kézbentartott vezérlés mellet nem közömbös annak tempója sem. Bizonyára a kedves Olvasó is dühöngött már egy-egy értelmetlenül hosszú ideig piros fényt adó jelzölámpa előtt. Az irányítás ettől persze még biztonságos, ám az a célszerű, ha a váltások ritmusa az adott útszakasz forgalmával dinamikus összhangban van. Így van ez a számítógépekben is. A cél itt a leggyorsabb működés biztosítása. A CPU egy ún. *órajel* bemeneti vonalon keresztül a cím- és adatbusz vezérlését, emellett azonban az összes többi funkciója ellátását is meghatározó *elektromos négyszögjel impulzusokat* kap. Az órajelek szaporasága (frekvenciája) döntö jelentöségü az egész számítógép működésére nézve. A CPU mindent az órajelek tempójában, ezekkel időzítve végez, a mikroprocesszor belsö világában ez jelenti az idö múlását. Sajnos az alkalmazott órajel-frekvencia nem lehet tetszölegesen nagy. Értékét erösen korlátozzák az egy-egy számítógéptípusban egyedileg alkalmazott áramkörök érzékelési, kapcsolási tulajdonságai, de maga a CPU is. Túl gyors órajelek esetén téves vezérlési állapotok állnak elö, az egész rendszer működése bizonytalanná válik. A kritikus alkatrészek cseréjével a helyzet némiképp javítható, ez azonban a legtöbb esetben a gép árának növekedésével jár. Egy-egy számítógép kialakítása tehát ebben az értelemben is kompromisszumok kérdése.

A TV-Computerben alkalmazott órajel frekvencia 3,125 MHz. Ez másodpercenként 3 millió 125 ezer négyszögimpulzust jelent, ami nem mondható nagyon korszerűnek, de nem is rossz. A működés lassúságáért sok esetben inkább az ügyetlen programozás a felelös.

1.3.2 Regiszterek

Az adatokkal való gyors müveletvégzés érdekében a Z80-as CPU-nak több belső, *saját tárolóeleme* van, ezeket nevezzük ***regisztereknek.*** A belső tárrekeszek egy része ugyancsak 8 bites. A megkülönböztetés végett ezeket egy-egy betüvel azonositjuk. A CPU teljes 8 bites regiszterkészlete a használatos betüjelekkel a következö:

		I	R		
A	**F**			**A'**	**F'**
B	**C**			**B'**	**C'**
D	**E**			**D'**	**E'**
H	**L**			**H'**	**L'**

Az ***I*** és az ***R*** speciális funkciójú regiszterek. Az *I* jelü az ún. *megszakitásregiszter* (*Interrupt*, ejtsd: interrapt = megszakitás). Erröl késöbb szó esik még, ám magával az *I* regiszterrel a programozási munka során nemigen kell foglalkozni. Az *R* pedig ún. *adatfrissitö* (*Refresh*, ejtsd: rifres = frissités) regiszter, amelynek az egész számitógép fizikai müködésében óriási jelentösége van, azonban programozási szempontból erröl is csak annyit kell tudni, hogy maga a CPU használja különleges -- *memóriafrissitési* -- célokra. Értéke folyton és gyorsan változik, igy pl. egyszerü *véletlenszám-generátornak* használható.

A megadott többi 16 db nyolcbites regiszter általános felhasználású. Az egyszerü és a felsö vesszövel megkülönböztetett két készlet közül a CPU mindig csak az egyiket használja, ám ezek egyszerüen felcserélhetök. Az éppen nem használt készletet *másodlagos* vagy *háttérregiszterek*nek nevezzük.

Megkülönböztetett jelentösége van az ***A*** és ***F*** jelü regisztereknek.

Az ***A*** teljes neve: *akkumulátor*. A CPU az adatokkal végzett müveletek túlnyomó részét ezzel a regiszterrel végzi. Az utasitáskészlet megismerésekor látni fogjuk,

hogy a CPU-műveletek jelentős része eleve az *A* regiszter használatát támogatja.

Az *F* regiszter jelölése az angol *flag* (ejtsd: fleg) szóra utal, amely jeladást, zászlót jelent, s a számítástechnikában meghonosodott tartalma is ez. Az *F bitjei* fontos jelzőfunkciókat látnak el. Ezeket a CPU állítja be, s egy-egy művelet elvégzése után az *F* regiszterből kiolvashatók a műveletvégzés körülményei. Az F bitjeinek beállításával a CPU az éppen végrehajtott művelet eredményének sajátosságait jegyzi meg. Ez programozási szempontból azért nagyon fontos, mert a program végrehajtása közben szükséges elágazásokat éppen ezekre a *jelzőbitekre* építhetjük.

Az F bitjeinek elnevezése a következő:

b7	b6	b5	b4	b3	b2	b1	b0
S	Z	-	H	-	P/V	N	CY

Az **S bit** (*Sign*, ejtsd: szájn = előjel) a számokkal végzett műveletek után az eredmény előjelét mutatja. Az 1.2 alfejezetben elmondottak szerint ez nem más, mint az eredmény *7. bitjének* értéke.

A **Z bit** (*Zero*) az eredmény 0 értéke esetén 1 (*akkor jelez, ha 0-t kapunk*), más esetben 0-ra állítja a CPU.

A **H jelzőbit** (*Half-carry*, ejtsd: halfkeri = félátvitel) azt jelzi, ha egy nyolcbites műveletben az alsó négy biten keletkezett átvitel, tehát a művelet *egy hexadecimális helyi értéken* belül nem volt elvégezhető.

A **P/V bitet** (*Parity/Overflow*, ejtsd: pariti/overflő = paritás/túlcsordulás) számolási műveleteknélakkor állítja 1-re a mikroprocesszor, ha az *előjelesnek* tekintett számok közötti művelet a 7. bitet is megváltoztatta, tehát ilyen értelemben *hibás eredmény* született. Más esetekben ugyanez a jelzőbit az eredmény ún. *paritását* mutatja. Értéke 1, ha az eredményben az 1 értékűbitek száma páros, 0, ha páratlan. A kettős funkcióra utal a jelölés is: *P/V*.

Az **N** jelű **bit** összeadás után 0, kivonás esetén 1.

A **CY jelzőbit** (*Carry*, ejtsd: keri = átvitel) a műveleteknél keletkező átvitel, túlcsordulás általános jelzőbitje.

A jelzőbitekről a mikroprocesszor utasításkészlete kapcsán, de főleg könyvünk programozásról szóló részében esik még szó, használatukat több példán is bemutatjuk.

A többi nyolcbites regiszter teljesen általánosan és egyenrangúan használható.

Mégis megemlítjük, hogy a ***B regiszter*** az ismétlődő, ciklikusan futtatott programrészek szervezésekor automatikus számlálóként használható egy speciális utasítás révén (DJNZ, l. később). A ***C regiszter*** pedig a portok kezelésénél játszik kiemelkedő szerepet.

A CPU 8 bites regiszterei az előbbi táblázatban egy sorban álló két-két regiszter összekapcsolásával ***16 bites regiszterpárokként*** is használhatók. Számos CPU-művelet szolgál e lehetőség kiaknázására. Ilyenkor a betűjeleiket szorosan egymás mellé írjuk, így ***AF, BC, DE*** és ***HL regiszterpárról*** beszélünk.

A ***HL*** és ***DE regiszterpár*** sok CPU utasításban memóriacímzési funkciót lát el, különösen előnyös erre a célra a ***HL*** használata. Ugyanez a regiszterpár néhány műveletben *16 bites akkumulátor* szerepet is betölt.

A ***BC regiszterpár*** igen gyakran *16 bites számlálóként* szolgál.

Az eddig tárgyaltakon kívül a Z80-as CPU még 4 db -- kifejezetten 16 bites -- regisztert tartalmaz.

Az ***IX*** és ***IY*** ún. *indexregiszterek*. Tulajdonképpen sokféle célra használhatók. Legfontosabb alkalmazási területük a *memóriarekeszek indexelt címzése*. Ennek lényege az, hogy IX-be vagy IY-ba egy memóriarekesz címét töltve, meg lehet adni még egy ún. eltolást, indexet. A tényleges memóriaművelet nem a betöltött című rekesszel, hanem az ehhez viszonyított eltolás szerinti hely tartalmával hajtódik végre. Ez a lehetőség a programozási munkában sokszor és rendkívül előnyösen használható (táblázatok kezelése).

Lehetőség van a CPU általános regisztereiben éppen bent lévő adatok ideiglenes, gyors tárolására. Ehhez a memóriában egy csakis erre a célra használt, ún. *veremterületet* kell a CPU rendelkezésére bocsátani. Ennek angol neve *stack* (ejtsd: sztek), amely zsákot, vermet jelent. A számítástechnikában való szóhasználat e memóriaterület használatának módjára utal. A szűk zsákba, keskeny verembe csak *egymás után* rakhatunk *be* dolgokat és *kivenni* belőle csak a berakással *fordított sorrendben* lehet. Az utoljára betett dolgot kell először kivenni, aztán az alatta levőt stb. Ezt az ún. *LIFO* technikát (*Last In -- First Out*, ejtsd: leszt in -- förszt aut = utoljára be -- elsőként ki), a CPU egy különleges, csakis erre a célra hasz-

nált 16 bites regiszterrel valósítja meg. Ennek neve ***SP*** (*Stack Pointer* = veremmutató). Tároláskor a mikroprocesszor egy 16 bites regiszter vagy pedig két 8 bites regiszterpár tartalmát tölti a *verembe* -- mégpedig mindig *az SP mutatta címre és az alatta levőre*. Közben az *SP* tartalma is *2-vel csökken*, így az mindig a verem utoljára betöltött memóriarekeszére mutat. Az így tárolt adatok természetesen vissza is tölthetők a CPU regisztereibe -- mégpedig mindig az *SP mutatta címről és a következő* (1-gyel nagyobb című) *memóriarekeszből* --, így éppen az utoljára tárolt adat visszatöltése valósul meg. Az *SP értéke ilyenkor 2-vel nő*, tehát ezután is a következő sorra kerülő memóriarekeszre mutat.

Igen szellemes mechanizmus.

A negyedik 16 bites speciális regiszter neve: ***PC*** (*Programming Counter*, ejtsd programing kaunter = programszámlálő). Szerepéről már volt szó, a memóriába írt programok végrehajtása közben mindig ez a regiszter mutatja a következő végrehajtandó utasítás memóriacímét. Értéke minden adatbeolvasás után *automatikusan 1-gyel nő*, így biztosítja a memóriába írt utasításssorozat *folyamatos végrehajtását*. Az egyszer már említett ugróutasítások is úgy érik el azt, hogy a program ne a következő memóriarekesz beolvasásával folytatódjék, hogy a PC regisztert írják át a kívánt címértékre. Ez természetesen azzal jár, hogy a következő beolvasási művelet már az új memóriacímről történik, utána tehát már az új programrész hajtódik végre.

A regiszterek használatának módját -- és bizonyos értelemben a célját is -- a CPU teljes utasításkészletének megismerése után látjuk világosan.

Mielőtt azonban ezt elkezdenénk, foglalkoznunk kell még a Z80-as CPU egy másik nagyon lényeges működési sajátosságával.

1.3.3 Megszakítások

A memóriába írt program végrehajtása csak a legritkább esetben zajlik folyamatosan, időbeli megszakítások nélkül. Általános esetben ui. a mikroprocesszor nem csak az általunk beírt program végrehajtásával foglalkozik. Vannak a számítógép kezelésének olyan sűrűn ismétlődő feladatai -- pl. a billentyűzet figyelemmel kísérése -- amelyet nem lenne szerencsés az éppen végrehajtás alatt álló, általunk írt programra hárítani. Célszerűbb azt egy

automatikus folyamattá szervezni, s csak az eredményt bocsátani a felhasználó rendelkezésére. Ilyen feladatok ellátására alkalmas a CPU megszakításkezelő rendszere.

Ez alapvetően kétféle lehet: ***maszkolható*** *vagy* ***nem maszkolható*** *megszakítás*. Az első esetben a kiváltó impulzusok hatását érvénytelenítheti egy direkt erre a célra szolgáló CPU utasítással (DI, l. 1.3.4.9 szakasz). Nem maszkolható utasítás esetén erre nincs lehetőség, hatása feltétlenül érvényesül.

A Z80-as mikroprocesszor háromféle *maszkolható megszakítási módban* dolgozhat. Az ***INT*** nevű (*Interrupts*) vezérlővonalra érkező impulzus hatására a CPU felfüggeszti az éppen futó program végrehajtását. A következő utasítás memóriacímét a verembe tölti, majd a beállított megszakítási módtól függően egy külön erre a célra szolgáló *megszakításkezelő programra* ugrik. Természetesen ez is egy közönséges program a memóriában, funkciója azonban kifejezetten a megszakítások idején végrehajtandó feladatok ellátása. Ennek befejezése után a CPU a veremből visszatölti a megszakított program soron következő utasításának memóriacímét, s a munka így ott *folytatódhat, ahol félbeszakadt*.

Előzetesként csak annyit, hogy a TV-Computerben a CPU általános esetben *minden másodpercben 50-szer szakítja meg* programjaink végrehajtását. A leírt folyamat *teljesen automatikus*. Tudomásul kell vennünk azonban, hogy a mikroprocesszor a programjaink végrehajtása idején elég sokszor nem azzal (hanem a megszakításkezelő programmal) foglalkozik.

Az ***INT*** vezetékre többféle forrásból juthatnak megszakítást kiváltó impulzusok. Ennek módszerei az adott számítógéptípustól függőek.

Röviden megbeszéljük a háromféle megszakítási mód működését. Ezeket ***0-s, 1-es*** és ***2-es*** *módoknak* nevezzük. Bármelyikük egy egyszerű utasítással, programból beállítható.

0-s megszakítási módban a végrehajtandó alprogram egybyte-os címét közvetlenül a nyolcbites *adatbusz* vezetékeire kell kapcsolni. A szám 0...7 lehet. Ez a mechanizmus feltételezi valamilyen intelligens külső eszköz összekapcsolását a számítógéppel.

1-es megszakítási mód esetén a végrehajtandó programnak a memória *0038H* címén kell kezdődnie. A CPU az *INT* vezérlőjel hatására ui. erre a címre ugrik, a PC értéke automatikusan 0038H lesz.

Végül a ***2-es módban*** a megszakításkezelő program kezdő memóriacímének felső byte-ját az *I regiszter* tartalmazza, alsó byte-ját pedig az *adatbusz* pillanatnyi értéke adja.

A TV-Computerben memóriaszervezési okokból, valamint a géphez kapcsolt külső eszközök kezelésének egységesen kialakított technikája miatt elsősorban az *1-es megszakítási mód* használható.

Az eddigi -- nagyon jelentős -- CPU sajátosságok áttekintése után rátérhetünk a programozás szempontjából legfontosabb tudnivalókra. Fejezetünk további részében a CPU-nak kiadható gépi kódú utasításokat adjuk közre.

1.3.4 A Z80-as CPU utasításkészlete

1.3.4.1 Előkészítő megjegyzések

E nagy jelentőségű témakör részletes tárgyalását néhány idetartozó alapvető kérdés tisztázásával kezdjük.

Az első fejezetben láttuk, hogy a memória egymás utáni tárolórekeszeiben elhelyezett *számok alkotják a mikrogép szintű programot*. Ezeket a CPU egymás után beolvassa, s mivel minden szám meghatározott tevékenységet ír elő, a végrehajtott elemi utasítások sorozata eredményezi a magasabb rendű emberi célok, a kívánt programozási feladat megoldását.

Gondolatmenetünket az ottani első megközelítésben azzal zártuk, hogy a programozáshoz már csak azt kell tudni, hogy milyen szám milyen műveletet jelent a CPU számára, s a megfelelő számsorozatot (programot) el kell helyezni a memóriában.

Valójában azonban nagyon sok számról van szó. Egy meghatározott feladat megoldására alkalmas számsorozatot önmagában csak rendkívül koncentrált figyelem, megfeszített munka árán lehetne megtervezni, utólagos javítása pedig szinte lehetetlen lenne. A 0...255 számok sorozata ui. semmiféle konkrét formai kapcsolatban nincs azzal a feladattal, aminek megoldását a CPU-műveletek révén szolgálja. És vegyük még figyelembe, hogy komolyabb feladatok esetén a program esetleg több ezer számból áll!

Célszerűbb tehát eleve lemondanunk arról, hogy a programot ezekkel a számokkal tervezzük meg. Ha jól meggondoljuk, *a valódi programozói munka szempontjából nem is az a fontos, hogy egy-egy műveletnek milyen szám felel meg a memóriában.* Sokkal fontosabb az, hogy a mikroprocesszor egyáltalán milyen típusú műveleteket képes elvégezni, s ezekből az elemi műveletekből hogyan lehet összeállítani azt a sorozatot, amely végül a kívánt programozói célnak megfelelő működést eredményezi.

A hangsúlyt tehát inkább a lehetséges CPU-műveletek ismeretére kell helyezni -- a tényleges programozás szempontjából a konkrét számértékek (az ún. műveleti kódok) ismerete másodrendű kérdés.

A műveleteknek ezért egy-egy szimbolikus, emlékeztető nevet is adunk, s amikor majd valóban elkezdjük a programozást, akkor *először ezekkel az emlékeztető műveleti jelekkel -- idegen szóval: **mnemonikokkal** -- tervezzük meg a feladatokat.*

A mnemonikok (tehát a műveletekre emlékeztető rövidítések, szimbólumok) együttesét ***assembly** nyelvnek* nevezzük. Ez már alkalmas a tényleges programozásra, mivel rövid betanulás után képesek felidézni bennünk a megfelelő CPU-műveletet.

Am az így megírt program önmagában csakis a tervezési, javítási munkát szolgálja. Ha a feladatot assembly nyelven megterveztük, utána a benne szereplő műveleteket megvalósító számokat természetesen be kell vinni a memóriába -- a CPU csak a számokat képes kezelni.

Mivel a mnemonikok és a nekik megfelelő számok kapcsolata teljesen egyértelmű, így aránylag könnyen készíthetők olyan segédprogramok, amelyek a szimbólumok alapján beírják a memóriába a megfelelő számokat. Ezeket nevezzük *assembler fordítóprogramoknak.*

Ilyenek készen kaphatók a TV-Computerhez is, pl.: **TVC Assembler, Profimon, Debugger.** Az első kettőt magnetofonkazettáról kell betölteni, a harmadik pedig *programmodulként*, kívülről csatlakoztatható a számítógéphez.

Ha már jól megértettük és megtanultuk, megszoktuk a gépi kódú programozás alapelveit, akkor a hosszú és bonyolult programok fejlesztéséhez feltétlenül indokolt ezek használata. Az ajánlott három program a gépi kódok előállításán kívül számos egyéb különszolgáltatással is segíti a munkát.

Könyvünkben azonban még nem megyünk el ilyen messzire. Célunk a gépi kódú programozásba való bevezetés, *a legalapvetőbb elvek bemutatása és magyarázata.* Az ajánlott -- és más, hasonló funkciójú -- programok használata ezek *alkalmazását* segíti. Az általunk tárgyalt programokat mi

is a *mnemonikokkal* tervezzük meg, de a memóriába mi magunk írjuk be a hozzájuk tartozó számokat -- közvetlenül gépi kódban dolgozunk.

A memóriához, a mikroprocesszorhoz, az utasításokhoz, sőt ezek kódjaihoz való ilyen testközeliség nagyon hasznos, a gépi működés mély megértését, sajátos *átélését* segíti elő, s az így megszerzett tapasztalatok akkor is jó szolgálatot tesznek, ha az Olvasó később *assembler fordítóprogramokat* használ.

A rövidesen ismertetésre kerülő ***MONITOR*** nevű program ugyancsak sokoldalúan segíti a gépi kódú munkát, csak itt az utasításokat megvalósító számokat -- az előbbiekkel összhangban -- magunknak kell a gépbe írni.

A következőkben sorra vesszük a Z80-as mikroprocesszornak adható utasításokat. Megadjuk a műveletek szimbólumait, röviden leírjuk ezek tartalmát, s természetesen a hozzájuk tartozó, memóriába beírandó számokat: *az utasítások **hexadecimális** kódját* is.

1.3.4.2 Adatbetöltő utasítások

Ez az egyik legegyszerűbb, bár igen terjedelmes utasításcsoport. Azért vettük előre, mert közben sok fontos fogalom tisztázható lesz.

Az adatbetöltés elnevezés azt jelenti, hogy a CPU valamelyik memóriarekesz tartalmát bemásolja az egyik regiszterébe vagy fordítva, de lehet egy regiszter tartalmát a CPU-n belül egy másik regiszterbe is áttölteni. Az *eredeti adat nem változik, mindkét helyen ugyanaz lesz.*

Az utasítás általános alakja:

LD cél,forrás

Az *LD* szimbólum (*mnemonik*) határozza meg az utasítás tulajdonképpeni tartalmát. A ***load*** (ejtsd: loud) angol szó első és utolsó betűjét írjuk le, amely betöltést jelent. Ezzel a két betűvel emlékeztetjük magunkat a műveletre.

Az *LD* után leírjuk, hogy ***hová*** történik az adatbetöltés (*cél*), majd vesszővel elválasztva megadjuk, hogy ***honnan*** (*forrás*).

Egy lehetséges utasítás pl. a következő:

LD H,E

Azt jelenti, hogy az *E regiszterben* levő 8 bites adatot a CPU másolja át a *H regiszterbe* (közben az adat az E-ben is változatlanul megmarad).

Lehet konkrét számot is betölteni. Az

LD B,3A

utasítás hatására a *B regiszter* tartalma a *3A* hexadecimális szám lesz.

Sokszor a memória meghatározott című helyéről kell az adatot valamelyik regiszterbe másolni. Ezt így jelöljük:

LD A,(4C53)

Fontos megállapodás, hogy ilyenkor a zárójel fejezi ki azt, hogy nem a leírt adatról van szó, hanem *annak a memóriarekesznek a tartalmáról, amelynek címe a megadott érték*. A példában tehát a mikroprocesszor a *4C53H* című memóriarekesz tartalmát tölti az *A regiszterbe*.

Az előbbi utasítás fordítottja:

LD (4C53),A

-- az *A regiszter* tartalma a *4C53H* című tárrekeszbe kerül.

Vannak természetesen 16 bites adatmozgató utasítások is. Pl. az

LD DE,15AF

hatására a CPU a *DE regiszterpárba* tölti a *15AFH* értéket úgy, hogy a D tartalma 15H, az E regiszterben pedig az AFH szám lesz.

Érdekes és nagyon fontos utasítástipusra példa a következő:

LD (4C53),HL

Ebben az esetben a CPU a *HL regiszterpár* 16 bites tartalmát másolja át a memória megadott címére. Nyilvánvaló azonban, hogy ehhez két egymás utáni byte betöltésére van szükség. A 16 bites adat egyik 8 bites része a *4C53H* című memóriarekeszbe, a másik fele a *4C54H* címre kerül.

Nagyon fontos megjegyezni, hogy ilyenkor minden esetben ***a 16 bites adat alacsonyabb helyi értékű byte-ja másolódik át először a megadott címre, s a következő memóriarekeszbe kerül az adat magasabb helyiértékű byte-ja!***

Példánkban tehát az *L regiszter* tartalma kerül a *4C53H* címre, a *H-beli* érték pedig *4C54H*-ra.

Az

LD HL,(4C53)

utasítással megadható fordított betöltés is hasonlóan történik: az *L regiszterbe* kerül a *4C53H* memóriacímen levő 8 bites adat, a *H* pedig a *4C54H tartalmával töltődik be.*

A memóriarekeszekre való hivatkozásnak egy másik módja, hogy a zárójelek között nem a konkrét címet adjuk meg, hanem azt előzőleg valamelyik regiszterpárba, ill. az IX vagy IY indexregiszterbe töltjük, s a zárójelek közé is ezt írjuk. Néhány példa:

LD B,(HL)
LD A,(IX+17)
LD (DE),A

Ezekben az esetekben a zárójelek között éppen úgy egy-egy *memóriarekesz címére hivatkozunk*, mint ha konkrét számokat szerepeltettünk volna. Az utasítások végrehajtásakor a CPU a *HL regiszterpár*, az *IX regiszter* és a *DE regiszterpár* tartalmát memóriacímként kezeli. Így az első esetben a *HL mutatta címről* tölti be az adatot a *B regiszterbe*, a második példában az *IX*-hez először még hozzáadja az utasításban megadott *0017H eltolást*, s az így kapott címen található *memóriarekesz tartalmát* tölti az *akkumulátorba.* Végül a harmadik példában a *DE* választja ki azt a *memóriarekeszt, ahová* a mikroprocesszor az *A regiszter* tartalmát másolja át.

A következőkben felsoroljuk az adatbetöltéssel kapcsolatos lehetséges utasításokat és ezek hexadecimális kódját. A *dd* jelölés konkrét egybyte-os adatot, az *nnnn* konkrét 16 bites címet, *ee* pedig egybyte-os, *előjeles* számként értelmezett *eltolást* jelent.

02	LD	(BC),A			
12	LD	(DE),A			
77	LD	(HL),A	DD77ee	LD	(IX+ee),A
70	LD	(HL),B	DD70ee	LD	(IX+ee),B
71	LD	(HL),C	DD71ee	LD	(IX+ee),C
72	LD	(HL),D	DD72ee	LD	(IX+ee),D
73	LD	(HL),E	DD73ee	LD	(IX+ee),E
74	LD	(HL),H	DD74ee	LD	(IX+ee),H
75	LD	(HL),L	DD75ee	LD	(IX+ee),L
36dd	LD	(HL),dd	DD36eedd	LD	(IX+ee),dd

```
FD77ee    LD   (IY+ee),A
FD70ee    LD   (IY+ee),B          32nnnn     LD   (nnnn),A
FD71ee    LD   (IY+ee),C          ED43nnnn   LD   (nnnn),BC
FD72ee    LD   (IY+ee),D          ED53nnnn   LD   (nnnn),DE
FD73ee    LD   (IY+ee),E          22nnnn     LD   (nnnn),HL
FD74ee    LD   (IY+ee),H          DD22nnnn   LD   (nnnn),IX
FD75ee    LD   (IY+ee),L          FD22nnnn   LD   (nnnn),IY
FD36eedd  LD   (IY+ee),dd         ED73nnnn   LD   (nnnn),SP


0A        LD   A,(BC)
1A        LD   A,(DE)

7E        LD   A,(HL)             46         LD   B,(HL)
DD7Eee    LD   A,(IX+ee)          DD46ee     LD   B,(IX+ee)
FD7Eee    LD   A,(IY+ee)          FD46ee     LD   B,(IY+ee)
3Annnn    LD   A,(nnnn)           47         LD   B,A
7F        LD   A,A                40         LD   B,B
78        LD   A,B                41         LD   B,C
79        LD   A,C                42         LD   B,D
7A        LD   A,D                43         LD   B,E
7B        LD   A,E                44         LD   B,H
7C        LD   A,H                45         LD   B,L
ED57      LD   A,I
7D        LD   A,L
3Edd      LD   A,dd               06dd       LD   B,dd
ED5F      LD   A,R

ED4Bnnnn  LD   BC,(nnnn)
01nnnn    LD   BC,nnnn


4E        LD   C,(HL)             56         LD   D,(HL)
DD4Eee    LD   C,(IX+ee)          DD56ee     LD   D,(IX+ee)
FD4Eee    LD   C,(IY+ee)          FD56ee     LD   D,(IY+ee)
4F        LD   C,A                57         LD   D,A
48        LD   C,B                50         LD   D,B
49        LD   C,C                51         LD   D,C
4A        LD   C,D                52         LD   D,D
4B        LD   C,E                53         LD   D,E
4C        LD   C,L                54         LD   D,H
0Edd      LD   C,dd               16dd       LD   D,dd


ED5Bnnnn  LD   DE,(nnnn)
11nnnn    LD   DE,nnnn
```

5E	LD	E,(HL)	66	LD	H,(HL)
DD5Eee	LD	E,(IX+ee)	DD66ee	LD	H,(IX+ee)
FD5Eee	LD	E,(IY+ee)	FD66ee	LD	H,(IY+ee)
5F	LD	E,A	67	LD	H,A
58	LD	E,B	60	LD	H,B
59	LD	E,C	61	LD	H,C
5A	LD	E,D	62	LD	H,D
5B	LD	E,E	63	LD	H,E
5C	LD	E,H	64	LD	H,H
5D	LD	E,L	65	LD	H,L
1Edd	LD	E,dd	26dd	LD	H,dd
2Annnn	LD	HL,(nnnn)	6E	LD	L,(HL)
21nnnn	LD	HL,nnnn	DD6Eee	LD	L,(IX+ee)
			FD6Eee	LD	L,(IY+ee)
			6F	LD	L,A
ED47	LD	I,A	68	LD	L,B
			69	LD	L,C
			6A	LD	L,D
DD2Annnn	LD	IX,(nnnn)	6B	LD	L,E
DD21nnnn	LD	IX,nnnn	6C	LD	L,H
FD2Annnn	LD	IY,(nnnn)	6D	LD	L,L
FD21nnnn	LD	IY,nnnn	2Edd	LD	L,dd
ED4F	LD	R,A	ED7Bnnnn	LD	SP,(nnnn)
			F9	LD	SP,HL
			DDF9	LD	SP,IX
			FDF9	LD	SP,IY
			31nnnn	LD	SP,nnnn

Az utasításokat nézegetve látható, hogy bármelyik regiszter tartalma átmásolható bármelyikbe, sőt önmagába is, bár ez valójában üres utasítás. A regiszterek közötti adatmozgatás kódja egy-egy szám, kivéve az I és R regiszterrel kapcsolatos müveleteket.

Amikor valamelyik regiszterbe konkrét adatot akarunk tölteni, ehhez már a memória két egymás utáni byte-ját kell használni. Az első szám azonosítja magát a müveletet. A CPU ebből kideríti, hogy egyáltalán mit is akarunk, de ezzel együtt azt is, hogy a művelet végrehajtásához még egy további adatra is szükség van. Ezért beolvassa a következő memóriarekeszben található számot, így *a két byte tartalma most együtt határoz meg egy utasítást.*

Figyelemre méltó különbség van e számok funkciója között. Az *első* azt mondja meg, hogy mit kell csinálni -- ez a ***műveleti kód***. A *másik szám* a tevékenység tárgya, ezzel kell a műveletet elvégezni -- ez az ***utasítás operandusa***.

Ha egy műveletben valamelyik memóriarekeszre annak konkrét címével hivatkozunk, akkor a művelet jellegét azonosító egy- vagy kétbyte-os műveleti kód után még további két byte-on kell megadni az utasításban szereplő címet. Ezek tehát már *három- vagy négybyte-os utasítások*.

Nagyon fontos, hogy a konkrét címet tartalmazó összes utasításban elöször a cím alacsonyabb helyi értékű byteját kell beírnunk a memóriába, s a felső byte-ot adjuk meg másodikként! Például az

LD A,(48F3)

utasítást -- amelynek hatására a CPU a ***48F3H*** *című memóriarekesz tartalmát az* ***A*** *regiszterbe* másolja -- a memóriában a következőképpen kell elhelyezni:

... 3A F3 48 ...

A program végrehajtásakor a CPU *először beolvassa a műveleti kódot: a* ***3AH***-*t*. Ez meghatározza egyrészt azt, hogy az A regisztert kell betölteni, másrészt, hogy egy memóriarekeszből kell az adatot átmásolni, de azt is hogy ennek a tárrekesznek a címét a *műveleti kód után álló két byte-ból* kell venni. Beolvasásra kerül tehát *elsőként a* ***0F3H*** érték, majd a *PC* ismételt növelésével a *következő memóriarekeszből a* ***48H***. Ezekből a CPU összeállítja a ***48F3H*** *címet*, amelyet a címbuszra téve megtörténhet a beolvasás, amelynek végén az adat az *akkumulátorba* kerül.

A leírt mechanizmus ismerete és következetes tisztelетben tartása az eredményes programozás elemi feltétele.

Azokban az utasításokban, ahol *(IX+ee)* vagy *(IY+ee)* szerepel, ***IX*** és ***IY*** egy memóriacímet tartalmaz, amihez még *hozzá kell adni* az ***ee***-vel jelölt nyolcbites konkrét adatot. Az ee számot a CPU *előjeles értékként* kezeli (-128...+127, l. 1.2 alfejezet).

A *(HL)* jelölést tartalmazó utasításokban a ***HL*** *regiszterpár* értéke adja meg az adatmozgatásban érintett memóriarekeszt.

Konkrét példaként elemezzük a következő utasításokat:

```
7E        LD   A,(HL)
36F0      LD   (HL),F0
DD3605A9  LD   (IX+05),A9
FD4EFE    LD   C,(IY+FE)
```

Az első esetben a *HL*-ben levő 16 bites adatot a CPU egy memóriacímnek tekinti, s e cím tartalmát az *A regiszterbe* másolja.

A második példa utasítása a *HL* értékével megadott memóriarekeszbe a *0F0H* adatot tölti.

A harmadik példában a CPU az *IX* tartalmát tekinti memóriacímnek, ehhez *hozzáad 5-öt*, s az így kapott című memóriarekeszbe tölti a *0A9H* értéket.

Az utolsó példában pedig az *IY* értéke szerinti címhez a *0FEH -- negatív számnak értelmezett -- eltolást adja hozzá* a CPU (0FEH = -2, l. 1.2 alfejezet), tehát az *IY értékénél 2-vel kisebb című memóriarekesz tartalmát* másolja át a *C regiszterbe.*

Az *IX* és *IY indexregiszterekkel* kapcsolatos műveleteket azonosító műveleti kód mindig kétbyte-os. Vegyük észre, hogy ezek a *HL*-lel kapcsolatos műveletekből képezhetők úgy, hogy elé írunk *IX* esetén egy *0DDH* kódot, míg *IY* esetén *0FDH*-t.

A harmadik példában szereplő utasítást a memóriában négy egymás utáni byte-on tudjuk megadni. A *0DDH* jelenti azt, hogy az *IX regiszterrel* kapcsolatos műveletről van szó. A *36H* azonosítja a végrehajtandó műveletet: konkrét adatot kell betölteni a meghatározott című memóriarekeszbe. A *05* adja meg az *IX* értékéhez viszonyított *eltolást* a memóriában. Végül a *0A9H* a betöltendő adat. A program végrehajtásakor a mikroprocesszor a *PC regiszter* memóriacímző mechanizmusával sorra beolvassa a négy byte-ot. Miután ezekből teljesen pontossá válik az elvégzendő művelet, végrehajtja a betöltést. Ha pl. az *IX* tartalma *0BC43H* volt, akkor az utasítás végrehajtása után a *0BC48H című memóriarekesz tartalma 0A9H lesz.*

A betöltési utasításokat tartalmazó táblázatból látható, hogy a *regiszterpárok* tartalma egymásba közvetlenül nem másolható át! Feltölthetők viszont konkrét 16 bites adattal (a memóriába *a két byte-ot fordítva kell beírni!*), vagy a memória megadott két egymás utáni tárrekeszének tartalmával (ilyenkor *a két byte fordított sorrendben másolódik be*), vagy a regiszterpárok tartalma a memória két egymás utáni rekeszébe tölthető (*fordított byte-sorrendben*). Itt is figyeljük meg a *HL*-re és az *IX*-re vagy *IY*-ra vonatkozó műveletek kódjai közötti kapcsolatot!

A Z80-as CPU programozási szempontból rendkívül előnyös tulajdonsága, hogy többféle, ún. ***blokkos adatkezelési utasítást*** képes végrehajtani. Az adatok átmásolására is négy ilyen utasítás létezik:

LDD	LDI	LDDR	LDIR
ED A8	ED A0	ED B8	ED B0

LDD: *LoaD with Decrement* = áttöltés a címek csökkentésével

LDI: *LoaD with Increment* = áttöltés a címek növelésével

LDDR: *LoaD with Decrement and Repeat* = áttöltés a címek csökkentésével és ismétlés

LDIR: *LoaD with Increment and Repeat* = áttöltés a címek növelésével és ismétlés

A négy utasítás *mnemonikja* alá írtuk a megfelelő kódokat. Ezek tehát kétbyte-os utasítások.

Bármelyik parancs kiadása előtt a *HL regiszterpárba* kell betölteni annak a memóriaterületnek a kezdőcímét, *ahonnan* az adatokat mozgatni akarjuk, *DE-be* annak a memóriaterületnek a kezdőcímét, *ahová* az előbbi adatokat át akarjuk másolni, a *BC regiszterpárba* pedig azt kell beírni, hogy *hány byte* tartalmát kell áttölteni.

Az ***LDD*** *utasításban* a *HL* által címzett hely tartalma átmásolódik a *DE* mutatta helyre, ezután a CPU a *HL*, a *DE* és a *BC* tartalmát *1-gyel csökkenti*. Így egy újra kiadott *LDD* végrehajtása esetén már az eggyel kisebb című memóriarekeszek átmásolására kerül sor. A *BC regiszterpár* értéke a még átmásolandó byte-ok számát mutatja, s ha 0-ra csökken, ez az előírt átvitel végét jelzi. Ez a körülmény az *F regiszterből* kiolvasható.

Az ***LDI mnemonikkal*** megadott utasítás lényegileg ugyanúgy működik, mint az *LDD*, csak itt a *HL* és a *DE* értéke a végrehajtás után *1-gyel nő*, tehát az adatáttöltés a nagyobb memóriacímek felé halad. A *BC* most is 1-gyel csökken, szerepe ugyanaz, mint az *LDD* utasításban.

Az ***LDDR és LDIR*** utasítások záró *R* betűje a *repeat* (ejtsd: ripit) szó rövidítése, amely ismétlést jelent. E két utasítás az *LDD* és *LDI* szerinti műveleteket hajtja végre, ám *automatikusan ismétli* egészen addig, *amíg BC értéke 0 nem lesz*. Így ezekkel az utasításokkal -- a programban mindössze két byte-on -- akár sokezer memóriarekesz átmásolását, áttöltését írhatjuk elő.

1.3.4.3 Számolási és logikai műveletek, összehasonlítások

A Z80-as CPU a számolási műveletek közül közvetlenül csak *összeadásra* és *kivonásra* vonatkozó utasításokkal programozható.

Négyféle mnemonikot használhatunk:

ADC ADD SBC SUB

Az első kettő összeadást, az utóbbiak kivonást jelentenek.

ADC: *ADdition with Carry* = összeadás és a carry bit hozzáadása
ADD: *ADDition* = összeadás

SBC: *SuBtraction with Carry* = kivonás a carry bittel

SUB: *SUBtraction* = kivonás

Az adatok 8 vagy 16 bitesek lehetnek. *A 8 bites műveleteknél az egyik adat mindig az akkumulátor (az A regiszter) tartalma, és az eredményt is itt állítja elő a CPU. A 16 bites műveletekben pedig a HL, IX és IY szerepe kitüntetett: az egyik adat ezek értéke, s az eredményt is itt kapjuk.*

Ha a 8 vagy 16 bites összeadás vagy kivonás közben az eredmény túllép az 1 vagy 2 byte-on ábrázolható értéken -- tehát az 1.2 alfejezetben megbeszélt módon *hamis eredmény* képződik --, akkor a CPU az *F regiszterben CY = 1-et* állít be. Ha pedig az eredmény *0*, akkor a *zero jelzőbit* lesz *Z = 1*. Az *N bit* összeadás után 0, kivonásnál 1; az *S* a művelet előjelét mutatja (*7. bit*), a *P/V bit* pedig azt jelzi, ha az *előjeles* összeadás értelmezése szerint *hamis eredmény* született.

Az *ADC és SBC* utasításokban a *C* betű a *CY jelzőbitre* utal: a *carry* korábbi értéke is hozzáadásra vagy kivonásra kerül. Az *ADD* és *SUB* utasítások ezzel nem foglalkoznak.

A lehetséges műveletek:

8 bites összeadás: **ADC A,r** vagy **ADD A,r**
kivonás: **SBC A,r** vagy **SUB A,r**

Az *r helyére a következőket írhatjuk (a táblázatban természetesen a műveletek kódja szerepel):*

	(HL)	(IX+ee)	(IY+ee)	A	B	C	D	E	H	L	dd
ADC	8E	DD8Eee	FD8Eee	8F	88	89	8A	8B	8C	8D	CEdd
ADD	86	DD86ee	FD86ee	87	80	81	82	83	84	85	C6dd
SBC	9E	DD9Eee	FD9Eee	9F	98	99	9A	9B	9C	9D	DEdd
SUB	96	DD96ee	FD96ee	97	90	91	92	93	94	95	D6dd

A 16 bites műveletek:

ADC HL,rr	ADD HL,rr	SBC HL,rr
	ADD IX,rr	
	ADD IY,rr	

Az *rr* helyére konkrét regiszterpárokat írhatunk a következő táblázat szerint:

		BC	DE	HL	SP	IX	IY
ADC	HL,	ED4A	ED5A	ED6A	ED7A	----	----
ADD	HL,	09	19	29	39	----	----
ADD	IX,	DD09	DD19	----	DD39	DD29	----
ADD	IY,	FD09	FD19	----	FD39	----	FD29
SBC	HL,	ED42	ED52	ED62	ED72	----	----

A számolási műveletekhez soroljuk a CPU regisztereinek vagy a memória rekeszeinek tartalmát *1-gyel növelő és csökkentő utasításokat* is.

Az **INC r** *mnemonikkal* megadott műveletekben *nő*, **DEC r** esetén csökken az *r* értéke. (*INCrement*, ejtsd: inkrement = növekedés; *DECrement*, ejtsd: dikrement = csökkenés.) A következő táblázat mutatja az ezzel kapcsolatos lehetőségeket és az utasítások hexadecimális kódját.

8 bites műveletek:

	(HL)	(IX+ee)	(IY+ee)	A	B	C	D	E	H	L
INC	34	DD34ee	FD34ee	3C	04	0C	14	1C	24	2C
DEC	35	DD35ee	FD35ee	3D	05	0D	15	1D	25	2D

16 bites műveletek:

	BC	DE	HL	IX	IY	SP
INC	03	13	23	DD23	FD23	33
DEC	0B	1B	2B	DD2B	FD2B	3B

A Z80-as CPU-val végrehajtható, ún. *logikai műveletek nyolcbites adatokkal* végezhetők el, az egyik adatot és az eredményt most is az *A regiszter* tartalmazza. A logikai műveleteket a CPU az adatok nyolc bitjével külön-külön végzi el.

A mikroprocesszor a következő, általános mnemonikokkal meghatározott logikai műveletek elvégzésére képes:

AND r **OR r** **XOR r**

(*AND*, ejtsd: end = és; *OR*, ejtsd: or = vagy; *eXclusive OR*, ejtsd: exkluziv or = kizárò vagy).

A három logikai művelet értelmezése:

Mnemonik	Magyar	Egyik bit	Másik bit	Eredmény bit
AND	*logikai ÉS művelet*	0	0	0
		0	1	0
		1	0	0
		1	1	1
OR	*logikai VAGY művelet*	0	0	0
		0	1	1
		1	0	1
		1	1	1
XOR	*logikai KIZÁRÓ VAGY művelet*	0	0	0
		0	1	1
		1	0	1
		1	1	0

A konkrét utasításokat ismét egy táblázatban foglaltuk össze:

Logikai műveletek

	(HL)	(IX+ee)	(IY+ee)	A	B	C	D	E	H	L	dd
AND	A6	DDA6ee	FDA6ee	A7	A0	A1	A2	A3	A4	A5	E6dd
OR	B6	DDB6ee	FDB6ee	B7	B0	B1	B2	B3	B4	B5	F6dd
XOR	AE	DDAEee	FDAEee	AF	A8	A9	AA	AB	AC	AD	EEdd

Ehhez az utasításcsoporthoz sorolhatjuk még a

NEG és **CPL**
ED 44 2F

mnemonikokkal meghatározott műveleteket (*NEGation*, ejtsd: negéjsn = tagadás; *ComPLement*, ejtsd: komplement = kiegészítò rész). Mindkettò csak az *akkumulátorra* vonatkozik. A *NEG* az *A regiszterbeli* számot (-1)-szeresére változtatja (l. 1.2 alfejezet), a *CPL* pedig egyszerűen megfordítja az *A* mindegyik bitjét.

A programozási munkában rendkivül fontosak az összehasonlitó utasitások. Általános mnemonikjuk:

CP r

Az összehasonlitás mindig az *A regiszter* tartalmával történik. Az eredmény a *carry* és *zero jelzőbitekből* olvasható le, s úgy képezhetjük, hogy az A-ból gondolatban kivonjuk *r*-t. A CPU is tényleges kivonást végez, de ennek eredménye nem kerül az *A regiszterbe -- minden adat változatlan marad*. Ezek szerint:

ha	*r = A*	*Z = 1*	*és*	*CY = 0*
ha	*r kisebb, mint A*	*Z = 0*	*és*	*CY = 0*
ha	*r nagyobb, mint A*	*Z = 0*	*ás*	*CY = 1*

(*CY* a *carry jelzőbit* rövidítése -- az egyszerű *C* jelölés a CPU ugyanilyen nevü *regisztere* miatt nem alkalmazható.)

A lehetséges összehasonlitó müveletek:

	(HL)	(IX+ee)	(IY+ee)	A	B	C	D	E	H	L	dd
CP	BE	DDBEee	FBBEee	BF	B8	B9	BA	BB	BC	BD	FEdd

A következő utasitások meghatározott memóriaterületek vizsgálatát teszik lehetővé:

CPD	CPI	CPDR	CPIR
ED A9	ED A1	ED B9	ED B1

CPD *ComPare with Decrement* = összehasonlitás a címeket csökkentve

CPI *ComPare with Increment* = összehasonlitás a címek növelésével

CPDR *ComPare with Decrement and Repeat* = összehasonlitás, a címek csökkentése és ismétlés

CPIR *ComPare with Increment and Repeat* = összehasonlitás, a címek növelése és ismétlés

Az összehasonlitás ezekben az esetekben is az *A regiszterrel* történik, de az eredményt most az *S* és *Z jelzőbitek* mutatják. A *CY carry bit* nem változik. A *HL* tartalmazza a vizsgált memóriarekesz címét, a *BC* pedig számlálő szerepet tölt be -- éppúgy, mint az *LDD*, *LDI*, *LDDR* és *LDIR* utasitásokban. A *CPD* és a *CPDR* után *HL* értéke 1-gyel *csökken*, igy az összehasonlitás a kisebb memóriacímek felé halad, a *CPI* és *CPIR* után pedig *HL* értéke *nő*. A *BC* számlálő mindig 1-gyel *csökken*, s pillanatnyi állapotát a *P/V jelzőbit* mutatja: értéke *1*, amig *BC* nem éri el a *0*-t, a *BC = 0000* elérésekor pedig *P/V = 0*-t állit be a CPU.

A *CPD és CPI* müveletekben az eredmény a *Z, S és P/V jelzőbitek* alapján minden végrehajtás után külön megvizsgálható. A *CPDR és CPIR* esetén az összehasonlítás automatikusan *ismétlődik* addig, amíg a vizsgált memóriarekesz tartalma *A*-val egyenlő nem lesz, vagy amíg a *BC* számláló nem éri el a *0*-t.

Érdekes müveletet jelent a

DAA
27

(*Decimal Adjust Accumulator* = az akkumulátor decimális kiigazítása). Ez az utasítás lényegileg az *A regiszterben* lévő, két hexadecimális jeggyel megadható számot alakítja át *kétjegyű, tízes számrendszerbeli* számmá. A művelet közben a mikroprocesszor használja a *H, N és CY jelzőbiteket*, a művelet végén pedig megfelelően beállítja a *Z, S* és *P/V* értékét. A *P/V bit* most a *paritást* mutatja. Csak közvetlenül egy összeadás vagy kivonás művelet után használható.

1.3.4.4 Bitmüveletek

A következő, ugyancsak nagyszámú utasítást tartalmazó csoportot olyan müveletek alkotják, amelyekkel a kiválasztott adatok bitjeit *egyenként* tudjuk kezelni. Ezek az utasítások négy nagy csoportba sorolhatók:

-- a bitek értékének *vizsgálata és beállítása*;
-- a bitek *eltolása, léptetése* a byte-on belül;
-- az adatok bitjeinek *forgatása, ciklikus léptetése*;
-- 4 bitből álló *hexadecimális számjegyek forgatása.*

Az első csoportot a következő *mnemonikokkal* megadott utasítások alkotják:

BIT b,r SET b,r RES b,r

BIT *test BIT* = bitvizsgálat
SET *SET bit* = bitbeállítás
RES *RESet bit* = a bit alaphelyzetbe állítása, törlés

Ezek az utasítások konkrét esetben az *r* helyére írt CPU *regiszter* vagy valamelyik címzési móddal meghatározott memóriarekesz tartalmának arra a bitjére vonatkoznak, amit a *b* számmal jelölünk ki. A *b* értéke így *0...7* lehet.

A ***BIT*** utasítás megvizsgálja a kijelölt bitet, s a Z = 1 jelzőbitbeállítással jelzi, ha értéke 0. (Figyelem: *a zero bit* tehát *azt jelzi, ha a vizsgált bit 0!*)

A ***SET*** utasítás a kijelölt bitet *1-re állítja*, a ***RES*** pedig *törli*, a bit értéke *0 lesz.*

BIT

b	0	1	2	3	4	5	6	7
r								
(HL)	CB46	CB4E	CB56	CB5E	CB66	CB6E	CB76	CB7E
(IX+ee)	DDCB ee46	DDCB ee4E	DDCB ee56	DDCB ee5E	DDCB ee66	DDCB ee6E	DDCB ee76	DDCB ee7E
(IY+ee)	FDCB ee46	FDCB ee4E	FDCB ee56	FDCB ee5E	FDCB ee66	FDCB ee6E	FDCB ee76	FDCB ee7E
A	CB47	CB4F	CB57	CB5F	CB67	CB6F	CB77	CB7F
B	CB40	CB48	CB50	CB58	CB60	CB68	CB70	CB78
C	CB41	CB49	CB51	CB59	CB61	CB69	CB71	CB79
D	CB42	CB4A	CB52	CB5A	CB62	CB6A	CB72	CB7A
E	CB43	CB4B	CB53	CB5B	CB63	CB6B	CB73	CB7B
H	CB44	CB4C	CB54	CB5C	CB64	CB6C	CB74	CB7C
L	CB46	CB4D	CB55	CB5D	CB65	CB6D	CB75	CB7D

SET

b	0	1	2	3	4	5	6	7
r								
(HL)	CBC6	CBCE	CBD6	CBDE	CBE6	CBEE	CBF6	CBFE
(IX+ee)	DDCB eeC6	DDCB eeCE	DDCB eeD6	DDCB eeDE	DDCB eeE6	DDCB eeEE	DDCB eeF6	DDCB eeFE
(IY+ee)	FDCB eeC6	FDCB eeCE	FDCB eeD6	FDCB eeDE	FDCB eeE6	FDCB eeEE	FDCB eeF6	FDCB eeFE
A	CBC7	CBCF	CBD7	CBDF	CBE7	CBEF	CBF7	CBFF
B	CBC0	CBC8	CBD0	CBD8	CBE0	CBE8	CBF0	CBF8
C	CBC1	CBC9	CBD1	CBD9	CBE1	CBE9	CBF1	CBF9
D	CBC2	CBCA	CBD2	CBDA	CBE2	CBEA	CBF2	CBFA
E	CBC3	CBCB	CBD3	CBDB	CBE3	CBEB	CBF3	CBFB
H	CBC4	CBCC	CBD4	CBDC	CBE4	CBEC	CBF4	CBFC
L	CBC6	CBCD	CBD5	CBDD	CBE5	CBED	CBF5	CBFD

RES

b / r	0	1	2	3	4	5	6	7
(HL)	CB86	CB8E	CB96	CB9E	CBA6	CBAE	CBB6	CBBE
(IX+ee)	DDCB ee86	DDCB ee8E	DDCB ee96	DDCB ee9E	DDCB eeA6	DDCB eeAE	DDCB eeB6	DDCB eeBE
(IY+ee)	FDCB ee86	FDCB ee8E	FDCB ee96	FDCB ee9E	FDCB eeA6	FDCB eeAE	FDCB eeB6	FDCB ee8E
A	CB87	CB8F	CB97	CB9F	CBA7	CBAF	CBB7	CBBF
B	CB80	CB88	CB90	CB98	CBA0	CBA8	CBB0	CBB8
C	CB81	CB89	CB91	CB99	CBA1	CBA9	CBB1	CBB9
D	CB82	CB8A	CB92	CB9A	CBA2	CBAA	CBB2	CBBA
E	CB83	CB8B	CB93	CB9B	CBA3	CBAB	CBB3	CBBB
H	CB84	CB8C	CB94	CB9C	CBA4	CBAC	CB84	CBBC
L	CB85	CBBD	CB95	CB9D	CBA5	CBAD	CBB5	CBBD

A második csoportba tartozó utasítások:

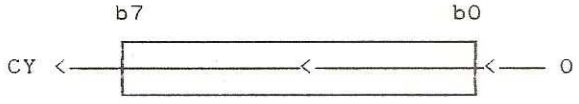
SLA r **SRA r** **SRL r**

SLA *Shift Left Arithmetic* = aritmetikai jellegű biteltolás balra
SRA *Shift Right Arithmetic* = aritmetikai jellegű biteltolás jobbra
SRL *Shift Right Logical* = logikai biteltolás jobbra

Az *r* konkrét esetben most is valamelyik *regiszter* vagy memóriarekesz tartalma.

Az SLA művelet a nyolcbites adat bitjeit *balra* lépteti úgy, hogy a bal szélső *b7 bit a carry jelzőbitbe kerül*, a többi 1-gyel balra lép, a jobb szélső *b0 bit pedig 0 lesz:*

SLA



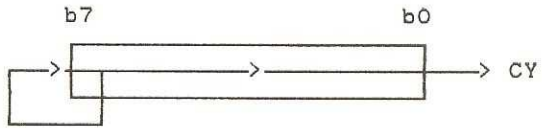
Ha pl. a *C regiszter* tartalma *0A7H* (bitenként ez 1010 0111), akkor az *SLA C* művelet hatása a következő:

		b7	b6	b5	b4	b3	b2	b1	b0
Eredeti:		1	0	1	0	0	1	1	1
Eltolás:	*CY* <—	*1*<—	*0*<—	*1*<—	*0*<—	*0*<—	*1*<—	*1*<—	*1*<— *0*
Eredmény:	*CY=1*	0	1	0	0	1	1	1	0

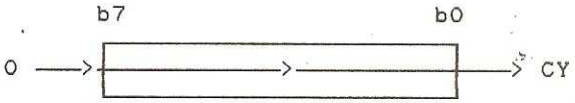
A *C regiszter* új értéke tehát *4EH* és *CY = 1* lesz.

Az *SRA és SRL* utasítások végrehajtásakor a mikroprocesszor a nyolc bitet 1-gyel *jobbra* lépteti. Most a jobb szélső *b0 bit kerül CY-ba*, a bal szélső *b7* pedig az *SRA* esetén nem változik, az *SRL* után pedig *0* lesz:

SRA



SRL



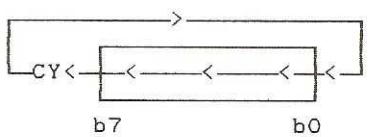
Az előbbi *C = 0A7H* példában az *SLA C* művelet eredménye *C = 4EH* és *CY=1* lett, most az *SRA C* után *C = 0D3H*, az *SRL C* után pedig *C = 53H*, a jobbra kilépő *b0=1* miatt pedig mindkét esetben *CY = 1* lesz az eredmény.

A következő táblázatban megadjuk a lehetséges *SLA, SRA* és *SRL* utasításokat, valamint ezek hexadecimális kódját.

	SLA	**SRA**	**SRL**
(HL)	CB26	CB2E	CB3E
(IX+ee)	DDCBee26	DDCBee2E	DDCBee3E
(IY+ee)	FDCBee26	FDCBee2E	FDCBee3E
A	CB27	CB2F	CB3F
B	CB20	CB28	CB38
C	CB21	CB29	CB39
D	CB22	CB2A	CB3A
E	CB23	CB2B	CB3B
H	CB24	CB2C	CB3C
L	CB25	CB2D	CB3D

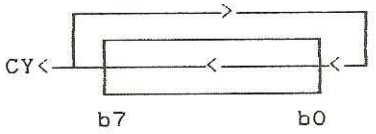
A bitműveletek harmadik csoportja ún. *forgatást* hajt végre az adat nyolc bitjén, felhasználva közben a *carry jelzőbitet* is, a következő sémák szerint:

RL r



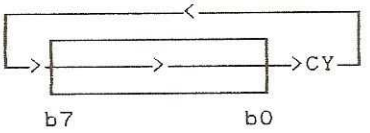
Forgatás a *CY jelzőbiten* át balra

RLC r



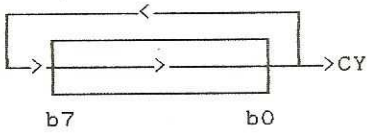
Forgatás balra, a 7. bit a *CY*-ba is beíródik

RR r



Forgatás a *CY jelzőbiten* át jobbra

RRC r



Forgatás jobbra, a 0. bit a *CY*-ba is beíródik

A *mnemonikok* a következő elnevezéseket rövidítik:

RL *Rotate Left* = forgatás balra
RLC *Rotate Left Circular* = közvetlen körbeforgatás balra
RR *Rotate Right* = forgatás jobbra
RRC *Rotate Right Circular* = közvetlen körbeforgatás jobbra

A konkrét utasításokat a következő táblázat foglalja össze:

	(HL)	(IX/IY+ee)	A	B	C	D	E	H	L
RL	CB16	DD/FD ee16	CB17	CB10	CB11	CB12	CB13	CB14	CB15
RLC	CB06	DD/FD ee06	CB07	CB00	CB01	CB02	CB03	CB04	CB05
RR	CB1E	DD/FD ee1E	CB1F	CB18	CB19	CB1A	CB1B	CB1C	CB1D
RRC	CB0E	DD/FD ee0E	CB0F	CB08	CB09	CB0A	CB0B	CB0C	CB0D

Kiemelt szerepe van ezekben a müveletekben is az *A* használatának. Az előbbi táblázatban megadott lehetőségek helyett, azokkal egyenértékűen használhatók az

RLA	**RLCA**	**RRA**	**RRCA**
17	07	1F	0F

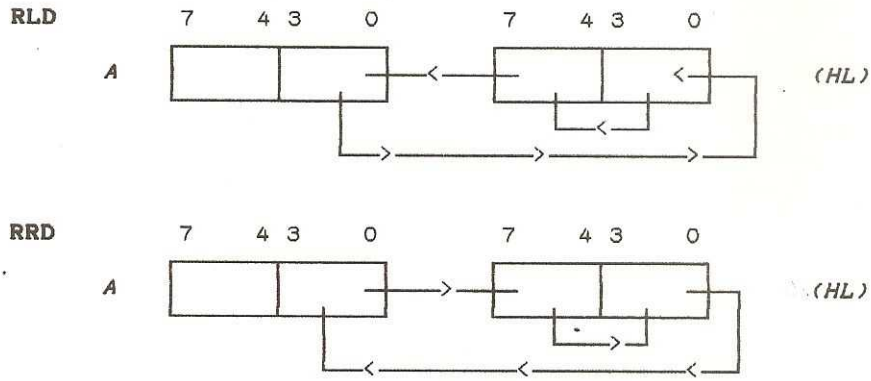
egybyte-os utasítások. Utolsó betűjük az *akkumulátor* használatára utal.

A bitműveletek negyedik csoportját alkotó utasítások:

RLD	és	RRD
ED 6F		ED 67

RLD *Rotate Left Digits* = számjegyértéket hordozó bitcsoport forgatása balra
RRD *Rotate Right Digits* = számjegyértéket hordozó bitcsoport forgatása jobbra

E műveletekben a CPU a *HL regiszterpárban* levő 16 bites adatot egy memóriarekesz címeként értelmezi, s az így meghatározott byte alsó és felsö helyi értékü négy bitjét, valamint az *A regiszter* alsó négy bitjét (tehát tulajdonképpen *egy-egy hexadecimális jegyhez tartozó bitcsoportot*) *cseréli fel egymás között*. A müveletek müködési sémáját a következő ábrák mutatják.



Végül a bitműveletek között említjük meg a

CCF	és	SCF
3F		37

utasításokat.

CCF Complement Carry Flag = az átvitelt jelző bit komplementálása
SCF Set Carry Flag = az átvitelt jelző bit beállítása

Az első egyszerüen ellenkezőjére változtatja a *CY jelzőbitet*, az utóbbi pedig *CY* = *1*-et állit be.

1.3.4.5 A portok kezelése

Az 1.1 alfejezetben vázoltuk a számítógépek portjainak szerepét: egyszerű, nyolcbites közvetítőkapocsnak tekinthetjük a mikroprocesszor és a számítógép különböző funkciójú eszközei között. Ott elmondtuk azt is, hogy ez a kapcsolat a CPU irányítása alatt áll. Most megismerjük azokat az utasításokat, amelyek révén e kapcsolat ténylegesen megvalósul.

A portokat *egybyte-os sorszámaikkal* azonosítjuk.

Az alapvető két lehetőséget az

IN és **OUT**

mnemonikokkal adjuk meg. Jelentésük: be, ill. ki.

Az

IN r,(p)

utasítás azt jelenti, hogy a mikroprocesszor a *p* sorszámú portról (valójában a hozzákapcsolt eszközről) beolvasható értéket töltse be az *r* jelű *regiszterbe*. Az *r* helyén a CPU valamelyik általános felhasználású nyolcbites regisztere állhat, a *p* helyén pedig a *C regiszter* (ilyenkor *C* értéke a port sorszáma), vagy pedig a konkrét *portsorszám*.

Az

OUT (p),r

utasítással az előbbi művelet fordítottját írjuk elő: a CPU az *r regiszterben* levő adatot a *p* sorszámú portra másolja (küldi).

A következő táblázat a konkrét lehetőségeket és az utasításkódokat foglalja össze:

	r:	A	B	C	D	E	H	L
IN	**r,(C)**	ED78	ED40	ED48	ED50	ED58	ED60	ED68
OUT	**(C),r**	ED79	ED41	ED49	ED51	ED59	ED61	ED69

IN A,(dd)	**OUT (dd),A**
DB dd	D3 dd

Itt is használhatók a már ismert működésmódú *blokkos* utasítások:

IND	**INDR**	**INI**	**INIR**	**OUTD**	**OTDR**	**OUTI**	**OTIR**
EDAA	EDBA	EDA2	EDB2	EDAB	ED8B	EDA3	EDB3

IND *INput with Decrement* = beolvasás a címek csökkentésével

INDR *INput with Decrement and Repeat* = beolvasás a címek csökkentésével és ismétlés

INI *INput with Increment* = beolvasás a címek növelésével

INIR *INput with Increment and Repeat* = beolvasás a címek növelésével és ismétlés

OUTD *OUTput with Decrement* = adatkiküldés a címek csökkentésével

OTDR *OuTput with Decrement and Repeat* = adatkiküldés a címek csökkentésével és ismétlés

OUTI *OUTput with Increment* = adatkiküldés a címek növelésével

OTIR *OuTput with Increment and Repeat* = adatkiküldés a címek növelésével és ismétlés

Ezekben a müveletekben a port sorszámát (címét) a *C regiszternek* kell tartalmaznia. A *HL*-ben most is a memóriarekeszek kezdőcímét kell elhelyezni és értéke a *D* jelü müveletekben *csökken*, az *I* jelzésüekben *nő*. Fontos különbség viszont az eddigi blokkos utasitásokhoz képest, hogy most a *C regiszter fontos funkciója* miatt számlálóként csak a *B regisztert* használja a CPU. Az *IN* müveletekben a *C*-vel megadott portról beolvasott adat a *HL* által címzett memóriarekeszekbe másolódik, *OUT*-nál pedig a *HL*-lel meghatározott memóriahelyek tartalmát küldi a mikroprocesszor a *C* szerinti portra.

B értéke egyesével *csökken*, s az *IND, INI, OUTD és OUTI* müveletek esetén *HL* és *B* csak beáll a soron következő értékre, mig az *INDR, INIR, OTDR és OTIR* esetén automatikus *müveletismétlés* történik, amig *B = 00* nem lesz.

1.3.4.6 Szubrutinok kezelése

Szubrutinnak, alprogramnak nevezzük egy utasitássorozatnak azt a részét, amelyet a CPU direkt erre szolgáló utasitások hatására hajt végre, s egy speciális *szubrutin vége* utasitás zár le.

Hangsülyozzuk, hogy ez a meghatározás a fogalomnak csak *programszerkezeti, szervezési szempontú* megközelitése. A szubrutinok programjainkban betöltött *funkcionális, tartalmi szerepéről* a későbbiekben sok szó esik még.

Egy *szubrutin* végrehajtására adott utasítás általános alakja:

CALL f,nnnn vagy **RST dd**

A *CALL* (ejtsd: koll) szó a müveletet azonosító *mnemonik*. Ez hivást, szólitást jelent, de az angol alapszó pl. kiáltást is jelenthet.

Az *f* meghatározott *feltétel* lehet, amely az *F regiszter jelzőbitjeinek* -- a korábbi müveletek eredményeit tükrözö -- pillanatnyi állapotára vonatkozik. A CPU csak akkor fogja végrehajtani a szubrutint, ha a megadott feltétel igaz.

A *CALL* utasitás feltétel nélkül is megadható, ekkor a CPU a megadott szubrutint mindenképpen végrehajtja.

A feltétel után, tőle vesszővel elválasztva azt a *címet* adjuk meg (*nnnn*), ahol a kivánt szubrutin utasitássorozata kezdődik a memóriában. A lehetséges feltételek:

Feltétel	**Jelentés**	**Igaz, ha...**
C	*CY=1*	van carry
NC	*CY=0*	nincs carry
Z	*Z=1*	nulla vagy egyenlő
NZ	*Z=0*	nem nulla vagy nem egyenlő
P	*S=0*	pozitiv
M	*S=1*	negativ
PO	*P/V=0*	páratlan paritás vagy túlcsordulás
PE	*P/V=1*	páros paritás vagy sima, probléma mentes művelet

Az *RST* utasitásokban semmiféle feltétel nem lehet és az alprogram memóriacimét is egyetlen byte-tal adjuk meg, amelynek értéke még ezen belül sem lehet tetszőleges. A CPU ilyenkor a *00dd* cimen kezdődő szubrutint fogja végrehajtani.

A következő táblázat az alkalmazható szubrutinhivásokat foglalja össze:

CALL f,nnnn

f:	**C**	**NC**	**Z**	**NZ**	**P**	**M**	**PO**	**PE**	-
	DC	D4	CC	C4	F4	FC	E4	EC	CD
	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn

RST dd

dd:	**00**	**08**	**10**	**18**	**20**	**28**	**30**	**38**
	C7	CF	D7	DF	E7	EF	F7	FF

Ezeket az utasításokat a CPU a következőképpen hajtja végre. A kellő számú utasításkód (3 vagy 2) beolvasása után a *PC programszámlálló regiszter* most is a program következő utasításának memóriacímére mutat, azonban *PC* értékét most -- feltételes utasítás esetén: ha a feltétel igaz -- a mikroprocesszor a *verembe* tölti és a *PC-t* a szubrutinhívó utasításban *megadott címre* állítja. Emiatt a következő beolvasott kód már a hívott szubrutin első utasítása lesz, a CPU ettől kezdve a megadott alprogramot hajtja végre. Egészen addig, amíg egy *visszatérés a szubrutinból, szubrutin vége* utasításnak megfelelő kódot nem olvas be.

A szubrutinok végét programjainkban a

RET f

utasítással adjuk meg (*RETurn*, ejtsd: ritörn = visszatérés). Az *f* ugyanolyan feltétel lehet, mintaz előbb. Ha nem adunk meg feltételt, akkor az utasítás a szubrutin végrehajtásának feltétlen befejezését jelenti. A programba beírandó kódok:

RET f

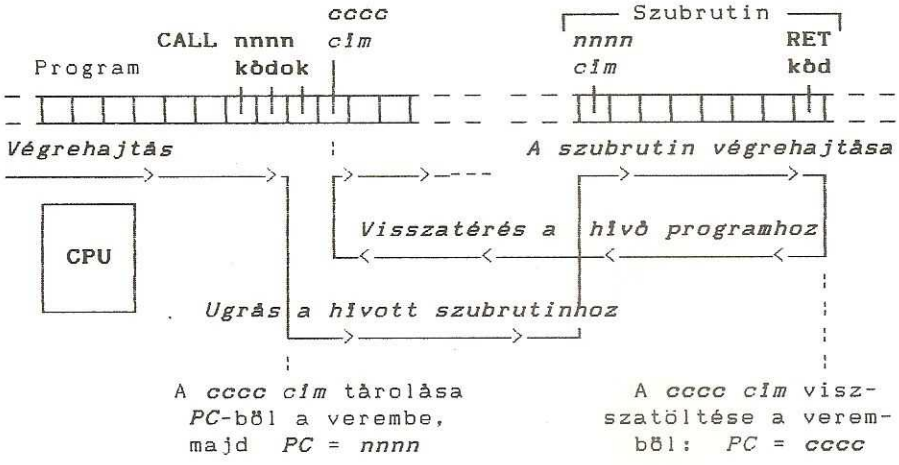
f:	**C**	**NC**	**Z**	**NZ**	**P**	**M**	**PO**	**PE**	**-**
	D8	D0	C8	C0	F0	F8	E0	E8	C9

A *megszakításokat* kiszolgáló szubrutinok végét a

RETI	**RETN**
ED4D	ED45

utasításokkal is lezárhatjuk. Az első a *maszkolható megszakításokból* való visszatérésre szolgál, a második pedig a CPU *NMI* (*Non Maskable Interrupt*, ejtsd: non maszkéjbl interrapt = nem maszkolható megszakítás) vezérlő bemenetére adott és feltétlen (tehát *nem letiltható) megszakítást* okozó jelek hatására végrehajtott szubrutin befejezését jelenti.

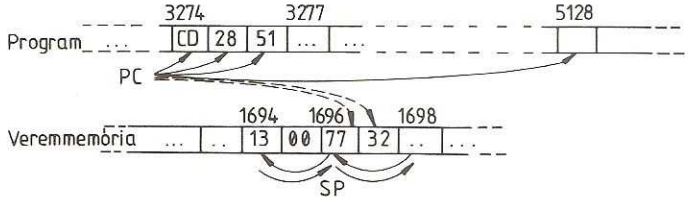
E kódok beolvasása után -- feltételes utasításnál: ha a feltétel igaz -- *a CPU visszatölti a veremből a korábban ott elhelyezett memóriacímet (az eredeti programban a szubrutin hívását követő utasítás címét), így a végrehajtás ismét az ún. főprogramban megy tovább (l. 5. ábra).* A címek veremben való tárolásának és visszatöltésének SP vezérelte mechanizmusát már ismerjük (l. 1.3.2 pont).



5. ábra Szubrutinok

A működés ilyen egyszerű eseteiben természetesen gondoskodni kell arról, hogy a szubrutinban használt *veremműveletek* -- amelyek az *SP regisztert* óhatatlanul elállítják a CPU által itt tárolt visszatérési címről -- a szubrutin végéig ugyanannyi adatot vegyenek ki a *veremből*, amennyit bepakoltak. Hiszen a *RET* hatására a CPU mindenkor az *SP* által éppen mutatott memóriarekeszből tölti vissza a *PC*-be a címet. A hibás *programozói veremkezelés* így rendkívül súlyos következményekkel járhat.

Tegyük fel, hogy a programunkat tartalmazó memóriaterület *3274H* címén a *CALL 5128* utasítás található. Amikor a CPU a program végrehajtásakor beolvassa a szubrutinhívás *0CDH, 28H és 51H* kódjait, akkor a *PC* értéke *3277H* (tehát már a következő utasítás memóriacímére mutat), s a mikroprocesszor ezt a címet teszi a verembe (l. 6. ábra).



6. ábra

Ha a *veremmutató* korábbi értéke mondjuk *1698H* volt, akkor a *1697H* című memóriarekeszbe a *32H*, *1696H*-ra a *77H* kerül és *SP* = *1696H* lesz.

Ha most az *5128H* címen kezdődő szubrutinunk végrehajtása közben újabb adatot teszünk a *verembe*, pl. a *0013H*-t, ez egyben *SP* = *1694H* beállítást okoz. Ez önmagában nem baj. De ha a megfelelő inverz utasítással az adatot nem vesszük ki a veremből, emiatt a szubrutin végét jelentő *RET* utasítás beolvasásakor *SP* = *1694H* marad, akkor a mikroprocesszor az ezen a címen tárolt *0013H* értéket, *hamis címet* tölti a *PC programszámláló regiszterbe*, és a végrehajtás is ott, a *0013H* című memóriarekesz beolvasásával folytatódik, tehát nem a helyes *3277H* címen. Az ilyen eltévelyedések pedig kiszámíthatatlan következményekkel járnak.

Ha a veremben -- egyébként korrekt módon -- tárolt *0013H* adatot a szubrutin még a lezáró *RET* előtt kiveszi a veremből, akkor *SP* = *1696H*-ra áll vissza. Így a *RET* utasítás végrehajtásakor a CPU a *1696H* címtől tárolt *3277H*-t tölti vissza a *PC*-be. A program végrehajtása tehát a helyes címen folytatódik.

Megjegyezzük, hogy az előbbi szigorú veremkezelési sémától nyugodtan el lehet térni, az sem fontos, hogy a szubrutin végrehajtását egy *CALL* vagy *RST* utasítással írjuk elő. A lényeg csak az, hogy *a szubrutint lezáró RET végrehajtása pillanatában a veremben értelmes, a további működés szempontjából érvényes cím legyen.*

A Z80-as CPU utasításkészletének megismerését a verem kezelésével kapcsolatos utasításokkal folytatjuk.

1.3.4.7 Veremkezelés

Ötféle művelet lehet. A 16 bites *regiszterek* és *regiszterpárok* közvetlen tárolására és visszatöltésére szolgáló utasítások:

PUSH rr és **POP rr**

PUSH = betol -- képletesen egy szük zsákba, verembe
POP = hirtelen kiszabadít, kipattint -- képletesen egy szük zsákba szorított tárgyat

A *PUSH* művelet az *rr* helyére írt *regiszterpár* vagy *16 bites regiszter* tartalmát másolja a verem *SP* által címzett két, egymásutáni memóriarekeszébe. *SP értéke 2-vel csökken, s mindig az utoljára használt címre mutat.*

A *POP* műveletnél az *SP* című és az azt követő memóriarekesz tartalmát másolja a CPU az utasításban megadott *regiszterpárba* vagy *16 bites regiszterbe*. Ekkor *SP értéke 2-vel nő*. A veremben tárolt értékek mindig 16 bitesek, s a byte-ok elhelyezése itt is mindig *fordított*: a kisebb című memóriarekeszben az alacsonyabb, a következőben a nagyobb helyi értékű byte van.

A lehetséges utasítások:

	AF	BC	DE	HL	IX	IY
PUSH	F5	C5	D5	E5	DDE5	FDE5
POP	F1	C1	D1	E1	DDE1	FDE1

A másik három utasítás az *SP* értékén nem változtat. Közös mnemonikjuk: **EX** (*EXchange*, ejtsd: ixcséjndzs = felcserél).

EX (SP),HL	**EX (SP),IX**	**EX (SP),IY**
E3	DDE3	FDE3

Ezek az utasítások az *SP* által mutatott című, egymás utáni *két memóriarekesz tartalmát felcserélik a HL, IX, vagy IY tartalmával.*

1.3.4.8 Ugrások a programban

Ha a CPU egy utasítás végrehajtása után -- éppen annak hatására -- nem az 1-gyel nagyobb című memóriarekeszből olvassa be a következő művelet kódját, akkor *ugrásról* beszélünk. Tudjuk már, hogy a számok sorozatával előírt program folyamatos végrehajtását a mikroprocesszor *PC* jelű *programszámláló regisztere* biztosítja azáltal, hogy mindig a következő utasításkód memóriacímére mutat, s hogy minden kiolvasás után automatikusan 1-gyel nő. *A program végrehajtása egyszerű esetben a növekvő memóriacímek felé halad.*

Ezt a folyamatot bonthatjuk meg az ugró utasításokkal. Ezek a *PC regiszterbe* tetszőleges adatot tölthetnek. Így a program előbb leírt egyszerű végrehajtását annak bármely pontján megszakíthatjuk, s egy *másik pontra helyezhetjük át.*

Ezek az utasítások két csoportra oszthatók: vannak ún. *direkt* és *relatív ugrást* előíró utasítások.

Az első esetben közvetlenül azt a memóriacímet adhatjuk meg, ahol a program végrehajtását folytatni akarjuk. A direkt ugrást jelentő utasítások általános alakja:

JP f,nnnn

A *JP* jelölés az angol *jump* (ejtsd: dzsamp) szóra utal, jelentése: ugrás. Az *f* helyére itt is a már korábban megismert *feltételeket* írhatjuk, de el is maradhat, ekkor a CPU feltétlenül végrehajtja a kijelölt ugrást. Az utasításban *nnnn* helyére közvetlenül a kívánt memóriacímet kell írni. A lehetséges utasítások:

JP f,nnnn

f:	**C**	**NC**	**Z**	**NZ**	**P**	**M**	**PO**	**PE**	**-**
	DA	D2	CA	C2	F2	FA	E2	EA	C3
	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn	nnnn

Speciális direkt ugrást írnak elő a

JP (HL)	**JP (IX)**	**JP (IY)**
E9	DDE9	FDE9

utasítások. Ezekben a cél-memóriarekesz címét a *HL*, *IX* vagy *IY* tartalmazza, ahová azt az utasítás végrehajtása előtt természetesen be kell tölteni.

A másik utasításcsoportot alkotó ***relatív ugrások*** esetén nem a memória tényleges címét adjuk meg, hanem csak egy ún. *eltolást*, amely a soron következő utasítás memóriacíméhez viszonyítva azt adja meg, hogy *mennyivel kell előre vagy hátra ugrani* a memóriában. Az eltolás egybyte-os érték, amelyet a CPU *előjeles* számként kezel (l. 1.2 alfejezet). Így a relatív ugrás a pillanatnyi helytől előre csak *127*, vissza (a kisebb címek felé) pedig *128* címtávolságra lehet.

A relatív ugrások általános *mnemonikja*:

JR f,ee

Itt *JR* a *jump relatív* kifejezés rövidítése, az *f* ismét meghatározott feltételt kifejező szimbólum lehet, *ee* pedig az egybyte-os, *előjeles eltolás*. A következő táblázatból is kiderül, hogy most szűkösebbek a lehetőségeink:

JR f,ee

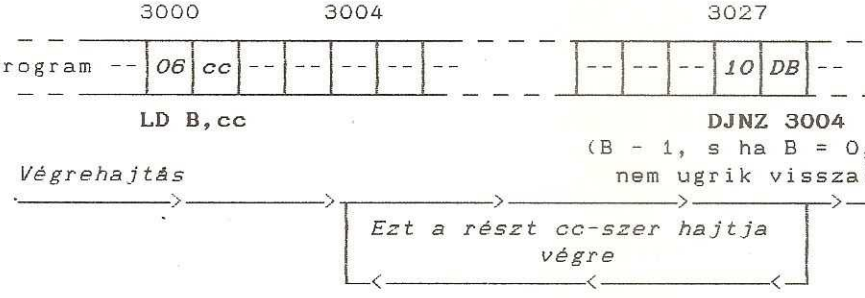
f:	**C**	**NC**	**Z**	**NZ**	**-**
	38ee	30ee	28ee	20ee	18ee

Speciális feltételes relatív ugró utasítás rövidítése a következő:

DJNZ ee

(*Decrement and Jump relative if Not Zero* = csökkentés, majd relatív ugrás, ha még nem lett 0).

Ebben az esetben *a feltételt a B regiszter pillanatnyi értéke jelenti.* Végrehajtásakor a CPU *B* értékét csökkenti *1*-gyel, s ha még *B* nem lett *0*, akkor végrehajtja az *ee*-vel megadott *relatív ugrást*, míg *B = 00* esetén nem -- ebben az esetben ugrás helyett a soron következő utasítással *folytatódik a program.* Látható, hogy a *DJNZ* utasítás meghatározott programrészek *ciklikus ismétlését* teszi lehetővé (1. 7. ábra).



*7. ábra A **DJNZ** utasítás*

Figyeljük meg, hogy a *DJNZ DB cikluszáró utasításban* a *ODBH eltolás* hatása hogyan érvényesül. A *10H*, *ODBH* kódok beolvasása után *PC* a következő utasítás memóriacímére mutat, tehát *PC = 3029H.* Ehhez adódik hozzá az *ODBH* eltolás. Mivel ez *negatív* értéket jelent -- a decimális *(-25)*-nek felel meg -- a CPU az ennek megfelelő 16 biten végzi el az összeadást:

```
  3029
+ FFDB
------
  3004
```

A program végrehajtásakor tehát a CPU a *3029H* helyett mindaddig visszatér a *3004H* címre, amíg a korábban *B*-be töltött *cc* érték a *DJNZ* utasítás ismételt, egymás utáni végrehajtásakor mindig *1-gyel csökkenve*, végül B = 00-t eredményez. (Feltéve, hogy a *3004...3026H* címeken elhelyezett programrész a *B* tartalmát nem változtatja meg.) A 3004H-n kezdődő programrész az újbóli visszatérések miatt tehát *annyiszor hajtódik végre, amennyi a B kezdő értéke volt.* Ha a csökkentés után végül *B = 00* lett, akkor a

program végrehajtása a *DJNZ-t követő utasítással folytatódik* (példánkban a *3029H* címen).

Programjainkban a ciklusok vezérlését legtöbbször erre az utasításra bízzuk.

1.3.4.9 Egyéb utasítások

Még kilenc olyan utasítás van, amelyeket eddig nem soroltunk be egyik csoportba sem. Az

IM 0	IM 1	IM 2
ED46	ED56	ED5E

parancsokkal a CPU háromféle *megszakításkezelési módja* állítható be (*Interrupt Mode* = megszakítási mód).

A

DI	és	EI
F3		FB

utasításokkal *tilthatjuk* (*Disable Interrupts* = a megszakítások tiltása) és újra *engedélyezhetjük* (*Enable Interrupts* = a megszakítások engedélyezése) a *maszkolható* megszakításokat.

A

HALT
76

utasítás megállítja a CPU programvégrehajtó munkáját a következő megszakításkérő impulzus pillanatáig, vagy egy ún. *reset* jel beérkezéséig. Ez utóbbi a mikroprocesszort alaphelyzetbe állítja, a maszkolható megszakításkérések letiltódnak, s a program végrehajtását a CPU a *0000* című memóriarekesznél kezdi újra. *Reset* hatású a TV-Computer jobb oldalán, alul található nyomógomb használata.

Végül, az

EX AF,AF'	EX DE,HL	és	EXX
08	EB		D9

utasítások *felcserélik* (*EXchange* = csere) az *AF és A'F'*, a *DE és HL* tartalmát, az utolsó pedig a CPU aktuálisan használt *B, C, D, E, H és L regisztereit a másodlagos készlet ugyanilyen jelű regisztereivel cseréli fel*, azok

válnak aktuálissá. Az ***EXX*** művelet közben az érintett ***regiszterek tartalma nem változik.*** Sok paraméterrel dolgozó programok esetén tehát egyszerű lehetőségünk van arra, hogy *8* helyett a CPU mind a *16 regiszterét* használjuk. Könyvünk programozással foglalkozó részében erre is látunk példát.

1.3.5 Összefoglalás

Ezzel a mikroprocesszorról szólő fejezet végére értünk. Belátom, aki most találkozott először az itt leírtakkal, számára rendkívül töménynek, nehéznek, sokszor bonyolultnak tűnhetett a sok utasítás, a számtalan új fogalom. Emiatt azonban nem kell visszariadni a gépi kódú munkától!

Csak a cél legyen előttünk, és hogy a gépi kód, amit most tanulunk, a számítógépünk egyetlen igazi nyelve. Bármilyen programnyelven is írjuk programjainkat, a tényleges végrehajtás mindig gépi kódban, és csakis az ebben a fejezetben megismert utasításokkal zajlik. *Sok ez -- **de csak ennyi.***

Adjon hitet és erőt a továbbiakra nézve is az a meggyőződés, hogy számítógépünk gépi kódban való programozása -- kezdő vagy akár profi szinten -- egy olyan izgalmas világ, amely messze felülmúlja pl. a *BASIC*-ből elérhető lehetőségeket, hogy a gép igazi képességeit, látványos és gyors grafikákat, hangeffektusokat, különleges billentyűzetkezelési megoldásokat éppen a ***gépi kódú programjainkkal*** tudjuk megvalósítani.

Végül az sem nagy baj, ha valaki első olvasásra nem érti a fejezet minden részletét. Először az is elég, ha csak tudomásul vesszük, hogy milyen utasítások vannak egyáltalán. Elég, ha megértjük az egyszerűbb műveletek lényegét, s megszokjuk azt a nagyszerű gondolatot, hogy ezek az ismeretek *a számítógép működésének már a legvégső, leglényegibb részét jelentik.* Ezek a működés alapjai, amelyekre később nyugodtan és megbízhatóan építeni lehet.

Már az is komoly eredmény, ha most -- a CPU-ról szólő fejezet tanulmányozása után -- átérezzük: tűnjék akár könnyűnek, akár nehéznek amiről eddig szó volt -- csak ennyi és nem több az az ismeret, amit a hatékony gépi kódú programozáshoz gépünk legfontosabb és legbonyolultabb alkatrészéről, a mikroprocesszorról tudnunk kell.

Később -- a gyakorlati programozás során -- fokozatosan mélyíthetjük ismereteinket. A legtöbb -- most még esetleg zavarosnak tűnő -- kérdés önmagától is tisztul, érthetőbbé és természetessé válik.

2. SZAMITOGEPÜNK SAJATOSSAGAI

Könyvünk eddigi részében megismerkedtünk azokkal az alapvető elvekkel, amelyek minden számítógép működésében közösek. Lényegi szempontból tisztáztuk a *memóriával*, a *portokkal* és a *mikroprocesszorral* kapcsolatos legalapvetőbb kérdéseket, az előző fejezetben pedig már részletesen is megismerkedtünk a *Z80-as CPU* programozási szempontból lényeges tulajdonságaival, *regisztereivel*, *utasításkészletével*.

Ezzel el is érkeztünk ahhoz a ponthoz, hogy figyelmünket most már a *TV-Computer* használatához szükséges, csakis rá jellemző sajátosságokra fordíthassuk. Ez a gépi kódu programozáshoz szükséges ismeretek másik alapvető oldala. Bármilyen alapossággal tanulmányozta ui. valaki az előző fejezetet, hiába lehet esetleg már gyakorlott programozó -- tudomásul kell vennünk, hogy minden program egy *meghatározott típusú számítógépre* készül. Bár a nyelv lehet közös, mégis: ugyanazt a feladatot az eltérő memóriakiosztás, portcímek, fizikai jellemzők és az egész szerkezet kialakításának sajátosságai miatt egy *kicsit másképpen* kell megoldani mindegyik számítógépen. Hasonlóan ahhoz, amikor valaki egy másik lakásba költözik: meg kell ismerni az egyes helyiségeket, megtanulni, mit hol találunk, minek hol lehet majd az új helye, hol vannak az ajtók stb. Meg kell szokni -- sőt tanulni -- az új lakás használatát, a régivel szemben tapasztalható előnyeit és hátrányait.

Nézzük, mit kínál nekünk a TV-Computer!

Ebben a fejezetben számítógépünk következő specialitásairól lesz szó:

-- *memóriafelépítés, a memóriák használata;*
-- *a tv-kép megjelenítése;*
-- *a billentyűzet;*
-- *hangképzés;*
-- *röviden a többi eszközről;*
-- *összefoglalás az általunk is használt portokról;*
-- *a rendszerváltozók és a munkaterületek;*
-- *funkcióhívások.*

Ezeket a témaköröket különböző részletességgel feldolgozva találja az Olvasó. Most csak azokkal a részekkel ismerkedünk meg alaposabban, amelyek a bevezető szintű

gépi kódú programozáshoz, könyvünk további részének megértéséhez elengedhetetlenül szükségesek.

2.1 A TV-Computer memóriája

Könyvünkben a *64k jelű gépek* programozásával foglalkozunk. A 32 kbyte-os, a külön bővített, vagy a VIDEOTON Gyár újabb fejlesztésű gépeiben ehhez képest kisebb eltérések lehetnek, amelyek sem a könyv használatát, sem pedig a programozási munkát komolyan nem gátolják. Ennek ellenére célszerű óvatosnak lenni, s az eltérő változatú gépek esetén a számítógép saját operációs rendszere vagy BASIC-je által használt memóriacímekre való hivatkozásokat minden esetben ellenőrizni!

A 64k jelű gépek *104 kbyte beépített memóriát* tartalmaznak. Lehetőség van arra, hogy külső eszközök, a kiépített csatlakozókon keresztül (a felül kialakított *négy bővítőkártya* és a bal oldalon egy *programmodul csatlakozó* segítségével) további *48 kbyte* memóriát kapcsoljanak a Z80-as CPU látókörébe. Ezeket a számokat összeadva láthatjuk, hogy ezen lehetőségek teljes kihasználásával 152 kbyte terjedelmű memóriát tudunk közvetlenül dolgoztatni a TV-Computerben. Ez a számadat csodálatosan nagy, ám valójában nem rendelkezünk szabadon ennyi tárolórekesszel.

Először is tudnunk kell, hogy a számítógépekben használt memóriák egyik típusába már a gyártás során beírnak valamilyen programot. A TV-Computerben a ***BASIC értelmező és végrehajtó program***, a billentyűzetet, a tv-képernyőt és más eszközöket kezelő alprogramok (***az operációs rendszer***) ilyen memóriákban vannak. Ezek a gép kikapcsolásakor sem vesznek el és tartalmuk nem is változtatható meg. A gép bekapcsolásakor ez a program indul el, ez biztosítja a gép azonnali, BASIC nyelvű használatát. (A gépi kódú programozás előtt éppen ebből a programból kell kiszabadítanunk a mikroprocesszort.)

Az ilyen, fix programokat tartalmazó memóriákat idegen mozaikszóval **ROM**-oknak nevezzük (*Read Only Memory* = csak olvasható memória).

A másik memóriatípus rekeszeinek tartalmával szabadon

rendelkezünk, ezek a **RAM**-ok (*Random Access Memory* = véletlen hozzáférésű tároló, az elnevezés ma már nem a fogalom tényleges tartalmát fejezi ki).

Az írható-olvasható memória egy részét ugyancsak nem használhatjuk szabadon.

Tekintsük át részletesen a 64k típusjelű TV-Computer memóriakiépítését.

A teljes memória *16 kbyte méretű szegmensekre van felosztva*. Ezek jelölése és funkciója a következő:

SYS	a *rendszer-ROM* fő része;
EXT/IOMEM	a *rendszer-ROM* folytatása (*felső 8k*) és a négy *bővítőkártya egyikén* elhelyezett memória (*alsó 8k*);
CART	ez nem a gépben van, hozzá a baloldali csatlakozóba *kívülről illeszthető programmodul*;
VID	*videomemória* (ebből általános esetben csak *15k*-t használunk);
U0,U1,U2,U3	felhasználói *RAM-szegmensek*.

Tekintettel arra, hogy a CPU *csak 64 kbyte memóriaterület közvetlen címzésére alkalmas*, a konkrét felhasználás előtt mindig ki kell választani a megadott szegmensek közül azt a *négyet*, amellyel dolgozni akarunk. Ezt a műveletet *memórialapozásnak* nevezzük.

Az egyértelműség végett a 64 kbyte-nak megfelelő 0000...0FFFFH címtartományt is négy egyenlő részre osztjuk, ezeket *lapoknak* nevezzük és 0...3 sorszámokkal látjuk el:

0. lap: *0000... 3FFFH címek;*
1. lap: *4000... 7FFFH címek;*
2. lap: *8000... BFFFH címek;*
3. lap: *C000... FFFFH címek.*

Elvileg bármelyik *memóriaszegmens* kerülhetne bármelyik *lapra*, azonban -- mint ez a következő táblázatból kiderül -- a TV-Computer csak meghatározott lapozási összetételek (*lapozási konfigurációk*) használatát engedi meg.

A lapozást a számítógép meghatározott *hardver* egységei a *02-es portra* küldött egybyte-os adat alapján állítják be. A következő táblázat a lehetséges konfigurációkhoz tartozó értékeket tünteti fel.

Memórialapozás:

3. lap	0 - 1 - 2 lap					
	SYS U1 VID	**SYS U1 U2**	**CART U1 VID**	**CART U1 U2**	**U0 U1 VID**	**U0 U1 U2**
CART	00	20	08	28	10	30
SYS	40	60	48	68	50	**70**
U3	80	A0	88	A8	90	B0
EXT	C0	E0	C8	E8	D0	F0

A bekapcsolást követő *inicializáló program* lefutása utáni alapbeállítás a *70H*-nak megfelelő *U0-U1-U2-SYS*, tehát a *0000...3FFFH* címeken az *U0 RAM*, *4000...7FFFH* között az *U1*, a *8000...0BFFFH* címeken az *U2* felhasználói *RAM-szegmensek*, a *0C000...0FFFFH* címeken pedig a *rendszer-ROM* érhető el.

Egy meghatározott memórialapozási összetételhez a táblázat szerinti adatot a *02-es portra* kell írni. Ez egy megfelelő *OUT* utasítással elvégezhető. Előtte azonban ugyanazt az értéket a *0003* című memóriarekeszbe is be kell tölteni -- emlékeztető feljegyzésül saját magunk és a *rendszer-ROM*-ban futó programok számára!

2.2 A képmegjelenítés

A számítógépek használata során általában is, de a gépi kódú programozás szempontjából különösen alapvető kérdés a tv-kép előállításának ismerete. Ugyanakkor ez az a terület, ahol a konkrét megoldás számítógépenként nagyon eltérő lehet. A képmegjelenítés mechanizmusa az adott géptípusban alkalmazott *hardvertől* függ és a vele való közvetlen munka során éppen ezeket a specialitásokat kell jól ismerni.

A TV-Computer esetében a *videomemória* egymás utáni memóriarekeszei és a képernyő tv-sorai között lineáris kapcsolat van. Egy sorban 512, 256 vagy 128 db pont

jelenik meg a képernyön aszerint, hogy a gép az ún. *2-es, 4-es vagy 16-os grafikus üzemmód* közül melyikre van állítva.

Az egy tv-sorhoz tartozó pontokat a *videomemóriában* mindig 64 egymást követő memóriarekesz tartalma határozza meg, mégpedig az első sort az első 64 db, a második sort a következő 64 db és így tovább, egészen a *240.* tv-sorig -- *ennyi sorból áll ui. a teljes tv-kép.*

Számoljunk! Az egy sorhoz tartozó 64 db memóriarekeszre a három üzemmódban 512, 256, ill. 128 pont jut. Ez byte-onként 8, 4 vagy 2 pontot jelent. Mivel egy memóriarekesz 8 bitből áll, így a *2-es üzemmódban minden pontot 1-1 bit, 4-es üzemmódban 2 bit, 16-os üzemmódban pedig 4 bit határoz meg.* A videomemória 1 byte-jához tehát a háromféle üzemmódban 8, 4 vagy 2 pont tartozik a képernyön. Ebből persze az is következik, hogy a megjelenő pontok *szélessége* a három esetben különböző. 2-es üzemmódban kapjuk a legsoványabb pontokat, 16-os üzemmód-ban pedig már inkább rövid vonaldarabokról beszélhetünk -- a felbontás igen durva.

A pontok a képernyön *meghatározott színűek* lehetnek. Ezt határozzák meg a videomemóriába írt adatok.

2-es üzemmódban 1 pontra csak 1 bit jut a vieomemóriában, ezért a megjelenő pontok színe is csak kétféle lehet. A bitek 1 és 0 értékéhez egy-egy színt rendelhetünk, s ez az egész képernyön érvényes lesz (kétszínű üzemmód). A két színt a *0.* és az *1.* ún. *palettaregiszterbe írt színkódokkal* jelölhetjük ki.

4-es üzemmódban 1 pontra már 2 bit jut, ami négyféle állapot kijelölését teszi lehetővé. A hozzájuk tartozó négyféle színt a *0...3* sorszámú *palettaregiszterekbe* írt kódok határozzák meg. A négy palettaregiszter *írási portcíme: 60H, 61H, 62H és 63H.*

Nagyon fontos tudni, hogy a videomemória byte-jait a TV-Computer *képmegjelenítő hardvere* ***négyszínű üzemmódban*** a következő bitkiosztással értelmezi:

b7	b6	b5	b4	b3	b2	b1	b0
1. pont	2. pont	3. pont	4. pont	1. pont	2. pont	3. pont	4. pont
kisebb helyi értékű bitje				nagyobb helyi értékű bitje			

A négy pont színét meghatározó bitpárok *alacsonyabb* helyi értékű tagját tehát a memóriarekeszek *felső* bitjei képviselik!

A ***16 színű üzemmódban*** pedig 1 pontot már 4 bit határoz meg. Ezek 16 különböző állapotához egy-egy szint rendelünk a következő bitkiosztás szerint:

b7	b6	b5	b4	b3	b2	b1	b0
1. pont	2. pont	1. pont	2. pont	1. pont	2. pont	1. pont	2. pont
Fényesség		Zöld szín		Piros szín		Kék szín	

Ha egy képernyőponton *keverékszínt* akarunk beállítani, akkor a megfelelő alapszíneket képviselő bitek értékét 1-re kell állítani. A színek kialakításához a következő táblázat használható, amelyet a fényességbittel is kiegészítettünk:

Fényesség bit	Zöld bit	Piros bit	Kék bit	A képernyőn megjelenő pont színe
0	0	0	0	fekete
0	0	0	1	sötétkék
0	0	1	0	sötétpiros
0	0	1	1	sötétlila
0	1	0	0	sötétzöld
0	1	0	1	sötét kékeszöld
0	1	1	0	sötétsárga
0	1	1	1	szürke
1	0	0	0	fekete (világos fekete!)
1	0	0	1	világoskék
1	0	1	0	világospiros
1	0	1	1	világoslila
1	1	0	0	világoszöld
1	1	0	1	világos kékeszöld
1	1	1	0	világossárga
1	1	1	1	fehér

A háromféle képmegjelenítési üzemmódról elöljáróban elég ennyit tudnunk.

A képernyő *keretének* színét az ún. *borderregiszterbe* írt adattal határozhatjuk meg, amelynek *portcíme: 00.*

A különböző színek eléréséhez a *border-* és a *paletta-regiszterekbe* töltendő adatok a következő táblázatból olvashatók ki (hexadecimális értékek):

Szín	Border	Paletta
fekete	00	00
sötétkék	02	01
sötétpiros	08	04
sötétlila	0A	05
sötétzöld	20	10
sötét kékeszöld	22	11
sötétsárga	28	14
szürke	2A	15
fekete	80	40
világoskék	82	41
világospiros	88	44
világoslila	8A	45
világoszöld	A0	50
világos kékeszöld	A2	51
világossárga	A8	54
fehér	AA	55

A kép megjelenítéséhez a TV-Computerben még egyéb eszközök (és ennek megfelelő portcímek) is tartoznak, a tv-kép kezeléséhez az alapfokú gépi kódú programozás szintjén ennyi ismeret elegendő.

2.3 A TV-Computer billentyűzete

A billentyűzet számítógépünk kiemelkedő jelentőségű belső eszköze. Programjainkban -- a *ROM-ba* írt belső *operációs rendszer* munkájától függetlenül -- sokszor lehet szükség a billentyűk állapotának közvetlen vizsgálatára. Nagyon fontos ezért tudnunk, hogy a CPU segítségével, a memóriába írt programmal ez a feladat hogyan végezhető el.

Az összes nyomógomb *10 sorba* van szervezve és 1 sorban *8 nyomógomb* helyezkedik el. A *03-as port alsó 4 bitjébe* írt adattal választhatjuk ki a 10 sor közül azt, amelyikhez tartozó 8 billentyűvel foglalkozni akarunk. Ezt követően az *58H-s port* beolvasásával állapíthatjuk meg a kiválasztott sorhoz tartozó billentyűk állapotát. A beolvasott nyolcbites adatban minden bit 1-1 nyomógombot azonosít úgy, hogy *a lenyomott állapothoz a bit 0 értéke tartozik.*

A billentyűk elhelyezése a 10 sorban, a beolvasásnál érvényes bitsorrend szerint a következő:

Sor	b7	b6	b5	b4	b3	b2	b1	b0
0.	! 4	' 1	Ł	/ 6	& 0	" 2	+ 3	% 5
1. /	= 7	ö	Ő	# *	Ü	) 9	(8	~ ^
2.	R	Q	` @	Z	$;	W	E	T
3.	U	P	U	{ [	Ű	O	I	}]
4.	F	A	> <	H	¦ \	S	D	G
5.	J	É	Ú	RET	Á	L	K	DEL
6.	V	Y	LOCK	N	SHIFT	X	C	B
7.	M	_ -	SPACE	CTRL	ESC	: .	? ,	ALT
8.		RJL	RJR	RJA	RJF	RJD	RJU	INS
9.		LJL	LJR	LJA	LJF	LJD	LJU	

A 8. sor a beépített és az elülső (jobb oldali), a 9. sor pedig a hátsó (bal oldai) botkormányhoz tartozik.

SPACE szóköz
RJL *Right Joystick Left* = jobb botkormány balra
RJR *Right Joystick Right* = jobb botkormány jobbra
RJA *Right Joystick Accelerator* = jobb botkormány gyorsítás
RJF *Right Joystick Fire* = jobb botkormány tűz
RJD *Right Joystick Down* = jobb botkormány le
RJU *Right Joystick Up* = jobb botkormány fel

A 9. sor ugyanezeket jelenti a bal oldali botkormány használata esetén, a kezdő *L* betű a *left* (bal) szóra utal.

2.4 Hangképzés

Már a bevezető szintű gépi kódú programozás során is szükség lehet a TV-Computer hangképzési lehetőségeinek ismeretére.

A hanggeneátor *közvetlen* programozásánál háromféle feladatot kell megoldani.

1. ***Hangmagasság beállítása.*** A hallható hangtartomány meglehetősen nagy, ezért erre a célra két byte-ot használunk: 0000...0FFFH a beállítható, ún. *PITCH* értéktartomány. A 0000 érték a legmélyebb, 47.68 Hz frekvenciájú hangot eredményezi, a 0FFFH-hoz pedig már nem is hallhatóan magas, 195312.5 Hz-es érték tartozik. Beállításkor, programjainkban *a tervezett PITCH érték alsó byte-ját a 04-es portra kell küldeni, a felsö byte négy bitjét pedig a 05-ös port alsó négy bitjébe kell írni.*

2. Fontos funkció a megszólaló hang ***hangerejének beállítása.*** Sajnos, ez csak eléggé durva fokozatokkal lehetséges: 0-tól 15-ig terjedő szimbolikus értekekkel 16 féle hangerőszint állítható be. *A kiválasztott számot a 06-os port b5...b2 bitjeibe kell írni.*

3. A számítógép bekapcsolt állapotában a hanggenerátor kimenetén állandóan és folyamatosan jelen vannak a ***hangképzéshez használt*** elektromos négyszögjelimpulzusok. A hangerő szabályozása mellett szükség van egy olyan kapcsolóra, amely ezeket az ***impulzusokat*** a hangképzésben résztvevő többi eszköz felé ***engedélyezi***, vagy fordítva: továbbjutásukat teljesen ***letiltja.*** Ezt a szerepet *a 03-as port b4 bitje* tölti be. Ha ide 0-t írunk, akkor a generátor jelei nem jutnak tovább -- a hang megszólaltatásához viszont ezt a bitet 1-re kell állítani.

2.5 A többi eszköz

Az TV-Computer müködésében az eddig megismert eszközökön kívül egyéb *hardver egységek* is komoly szerepet töltenek be. Csak utalunk arra, hogy a számítógép teljes eszközkészletéhez hozzátartozik még a géphez kívülről csatlakoztatható *négy bővitőkártya* vezérlése, a számítógép és

más intelligens eszköz -- pl. további számítógépek -- együttmüködését megvalósító *soros vonal*, vagy pl. a *nyomtatóval* és a *magnetofonnal* való kapcsolattartást biztosító eszközök.

Ezek müködésének vezérlése további, általunk itt nem ismertetett *portcímekkel* lehetséges. Csak megemlítjük, hogy pl. a CPU megszakításkezelő funkciójának nagyon rugalmas kihasználását a TV-Computerben hét további port segíti. Vagy az általunk is tanulmányozott képmegjelenítés bizonyos jellemzői a vezérlő integrált áramkör (6845-ös CRT) közvetlen programozásával megváltoztathatók -- ehhez is további portok tartoznak.

Mindezek ismerete azonban a TV-Computer kezdő és közepes szintü programozásához nem szükséges. A téma iránt teljes mélységében érdeklődő Olvasó számára ajánljuk a TV-Computerről szóló kiadványsorozatban eddig megjelent **Operációs rendszer** és **A TV-Computer ROM programja** cimü könyveket.

2.6 Összefoglaló a tárgyalt portokról

Tekintettel arra, hogy az általunk is ismertetett portcímek egy része többféle funkció ellátására szolgál, a későbbiek szempontjából is hasznos lesz ezeket portonként összefoglalni.

00 port OUT BORDERREGISZTER

Csak a képernyő *keretszínének* beállítására szolgál és csakis négy bitjének van szerepe.

b7	b6	b5	b4	b3	b2	b1	b0
I	-	G	-	R	-	B	-
Intenzitás (fényesség)		Zöld		Piros		Kék	

02 port OUT PAGE REGISZTER (Másolat: 0003H)

A *memórialapozási konfiguráció* beállítására szolgál.

03 port OUT **(Másolat: 0B11H)**

Felső két bitje a csatolókártyákon elhelyezett memóriák egyikének kiválasztására szolgál (*IOMEM*).

Alsó négy bitjének írásával a billentyűzet egy sorát választjuk ki, beolvasás előtt.

b7	b6	b5	b4	b3	b2	b1	b0
I O M E M kiválasztás		-	-	Billentyűzetsor- -kiválasztás			

04 port, OUT **PITCH, alsó byte**

Nyolc bitjén a *hangmagasság* beállítására szolgáló két-byte-os érték *kisebb helyi értékű részét* helyezzük el.

05 port, OUT **(Másolat: 0B12H)**

Négyféle funkcióhoz tartozó biteket tartalmaz.

Felső két bitjével a TV-Computerhez kapcsolt *két magnetofon motorját* tudjuk ki- vagy bekapcsolni, ha azok erre alkalmasak.

Az 5. bit 1 értékre állítása esetén a *hangjelek a CPU megszakítástkérő bemenetére* jutnak.

A 4. bit hangképzéskor a *hangjel engedélyezésére és tiltására* szolgál.

Az alsó négy bitbe a *hangmagasságot* meghatározó PITCH érték *felső byte-jának alsó négy bitjét* kell írni (csak ezek a hatásos bitek).

b7 -- b6		b5	b4	b3 - b2 - b1 - b0
Magnetofon-motor-vezérlés jobb	bal	Hang IT engedélye-zés/tiltás	Hangjel engedélye-zés/tiltás	P I T C H felső 4 bit

06 port, OUT **(Másolat: 0B13H)**

Bitjei háromféle funkciót látnak el.
A legfelső, 7. bit a *nyomtató* vezérlésére használt.
A középső négy bitjén a megszólaló *hang amplitudója* (hangereje) állítható be.
Az alsó két bitjébe írt adattal a *grafikus üzemmódot* határozzuk meg.

b7	b6	b5	b4	b3	b2	b1	b0
PRINTER vezérlés	-	A hangerő beállítása				üzemmód: 00: 01: 10: 11:	2-es 4-es 16-os 16-os

58H port, IN **BILLENTYŰ adatbeolvasás**

Nyolc bitjén annak a nyolc nyomógombnak a felengedett (1) vagy lenyomott (0) állapota olvasható be, amelyek a 03-as port alsó négy bitjébe írt számmal kiválasztott sorban vannak.

60H port, OUT **0. PALETTAREGISZTER, színbeállítás**
61H port, OUT **1. PALETTAREGISZTER, színbeállítás**
62H port, OUT **2. PALETTAREGISZTER, színbeállítás**
63H port, OUT **3. PALETTAREGISZTER, színbeállítás**

A *többfunkciós portokra* küldött adatokat a későbbi helyes felhasználás érdekében egy-egy kitüntetett című memóriarekeszbe is nyilván kell tartani. Ezek címét a megfelelő portok fejléc sorában feltüntettük.

Példa: a *06-os portra a hangerő beállításakor* nem küldhető ki minden további nélkül a kívánt adat (pl. a 9-es hangerőnek megfelelő 0010 0100 bitképű 24H érték), mert ezzel a b7 és a b1-b0 biteket is 0-ra írnánk át, miáltal *vezérlőjelet küldenénk a nyomtatóhoz és egyben **kétszínű üzemmódot állítanánk be.*** Ilyenkor természetesen csak az adott funkcióhoz -- jelen esetben a hangerőhöz -- tartozó biteket szabad átállítani, a többit változatlanul kell hagyni. Ehhez viszont tudni kell *a megmaradó bitek korábbi értékét* -- erre valók a megadott címen tárolt *portmásolatok*.

2.7 Rendszerváltozók, munkaterületek

Az *UO* ***RAM*** igen jelentős része, a *0000...19EEH* címek közötti memóriaterület -- képletesen szólva -- a TV-Computer operációs rendszerének és BASIC-jének irodája.

Bekapcsoláskor néhány rövid programrészt és nagyszámú táblázatot, működési paramétert másol ide a *ROM-ból* az ún. *inicializáló program.*

A *ROM-ba* írt program végrehajtása közben ezen a memóriaterületen nagyszabású adminisztrációs munka folyik.

Hogy csak néhányat említsünk:

-- aktuális *memórialapozási érték* tárolása (0003H);
-- hibakezelő rutinok (0008...0017H);
-- számolórutinok belépési pontja (0018H);
-- a *BASIC-ből* hívható felhasználói *gépi kódú programok címtáblázata* (0021...002FH);
-- a *funkcióhívások* belépési pontja (0030H);
-- a *megszakításkezelő rutin* belépési pontja (0038H);
-- kommunikáció a *csatolókártyák* révén géphez kapcsolt külső eszközökkel (0040...00FFH);
-- *ASCII* kódú képernyőmásolat (0100...073FH);
-- az ún. *definiálható karakterek* byte-jainak tárolása (0740...0AFFH);
-- *a funkcióhívások hozzárendelési táblázata* (0B00...0B0FH);
-- fontos működési jellemzőket meghatározó aktuális értékek tárolása (0B10...0B22H);
-- a funkcióhívásokhoz és a megszakításkezelő alprogramhoz tartozó néhány utasítás (0B23...0B48H);
-- az egyes eszközök további működési paraméterei (0B49...0B71H);
-- munkaterületek (0B72...0EABH);
-- a CPU *veremterülete* (0EAC...16ABH);
-- *BASIC* paraméterek és munkaterületek (16AC...19EEH).

Ezeket jobban nem részletezzük. Egy részüket megtalálja az Olvasó a géppel együtt vásárolt **Programozási segédletben,** még részletesebben az **Operációs rendszer** és **A TV-Computer ROM programja** című könyvekben, de a gépi kódú programozás során ezek közül csak igen keveset kell használnunk. Ilyenek pl. a *portmásolatok* (0B11H: 03. port, 0B12H: 05. port, 0B13H: 06. port), amelyekről már beszéltünk. Hasznos továbbá, ha ismerjük a *funkcióhívások hozzárendelési táblázatát,* erről a továbbiakban lesz szó.

2.8 Funkciôhîvâsok

A TV-Computer *operâciôs rendszere* a felhasznâlô szâmâra egyszerûen elérhetôvé teszi a szâmîtôgépbe beépîtett és a hozzâ kîvülrôl csatlakoztathatô *hardver eszközöket*. Az operâciôs rendszer feladata éppen ezeknek az eszközöknek az összerendezett mûködtetése, amelyet a ROM-ban târolt program egy része valôsît meg.

Az eszközök mûködtetésével kapcsolatos funkciôk jelentôs része a gépi kôdban programozôk szâmâra is egyszerûen hozzâférhetô azâltal, hogy az eszközökkel végrehajtandô, ismétlôdô és jôl körülhatârolhatô feladatokat egységesen a CPU ***RST 30*** egybyte-os szubrutinhîvô utasîtâsâra épîtették.

Funkciôkôd

A funkciôhîvâsok a következô eszközökre irânyulhatnak:

0. *képmegjelenîtés (VIDEO);*
1. ***billentyûzet;***
2. ***szövegszerkesztô*** (**BASIC** *programszerkesztô, EDITOR*);
3. ***hang;***
4. ***nyomtatô;***
5. ***magnetofon;***
6. ***csatolôkârtyâk;***
7. ***belsô felügyelôrendszer*** (KERNEL).

Minden külsô eszköz esetén létezik néhâny jôl meghatârozott mûvelet, eljârâs, feladat, amelyet *mindig ugyanûgy kell végrehajtani* (pl. egy betû kirâsa a képernyôre vagy a nyomtatôra, egy lenyomott billentyû azonosîtâsa stb). Az ezeket a feladatokat megvalôsîtô programok a ***ROM-ban*** vannak és kezdô cîmeik ismeretében normâl szubrutinhîvâssal is elérhetôk. Az egyes eszközök feladatait ellâtô ún. *alacsony szintû* rutinok azonban a ***funkciôhîvâsokkal*** egyszerûbben aktivizâlhatôk.

Ezeket a fontos szubrutinokat egyszerû *sorszâmmal* lâtjuk el, s az ezekre valô hivatkozâs automatikusan elvégzi a vele kapcsolatos feladatokat. Az *operâciôs rendszer* a sorszâm -- és néhâny késôbb leîrt jellegzetesség -- alapjân *kikeresi a szubrutin ROM-beli cîmét*, bizonyos

ellenőrzéseket is végez a téves hívások kivédésére, majd végrehajtatja a kívánt feladatot.

Az *RST 30 utasítás után* a memóriában el kell helyezni egy egybyte-os értéket: ez a ***funkciókód.*** Alsó négy bitjébe kell írni a feladat sorszámát, következő három bitjébe az eszköz sorszámát, a legfelső, 7. bit pedig a művelet irá-nyát jelöli ki:

b7	**b6 - b5 - b4**	**b3 - b2 - b1 - b0**
Irány	A funkcióosztály	A funkció sorszáma
1: INPUT	sorszáma	
0: OUTPUT	(eszközkijelölés)	

A funkció sorszáma 0...15 lehet. Az 1-es és a 2-es funkció minden eszköznél (amelynél ez egyáltalán értelmes lehet) ugyanazt a feladatot jelenti:

1-es funkció: karakter kiküldése a meghatározott eszközhöz vagy onnan egy karakter beolvasása;

2-es funkció: karakterblokk (egymás után több karakter) kiküldése a memóriából a meghatározott eszközre, vagy onnan egymás után több karakter beolvasása, majd tárolása a memóriában.

A többi funkció eszközönként eltérő tartalmú lehet.

A művelet iránya lehet adatkiküldés az eszköz felé *(OUTPUT)* vagy adatbeolvasás az eszközről *(INPUT).*

A *funkciókód* ***b6-b5-b4*** *bitjein* a végrehajtásra kiszemelt eszközt választjuk ki (a korábban már megadott sorszámokkal azonosítva). Megjegyezzük, hogy ez a sorszám nem okvetlenül jelenti azt, hogy a feladat éppen azon az eszközön hajtódik végre. A memória 0B00...0B0FH címein található hozzárendelési táblázatban ui. *előírhatjuk, hogy az adott számú eszközhöz intézett funkcióhívást valójában melyik eszköz hajtsa végre.* A táblázat szerkezete:

0B00 a ***VIDEO*** osztályhoz intézett funkcióhívást melyik eszköz hajtsa végre, ***INPUT*** művelet esetén;
0B01 a ***BILLENTYŰZET*** osztályhoz rendelt eszköz;
0B02 az ***EDITOR*** osztályhoz rendelt eszköz;
: :
: :
0B06 a ***CSATOLÓKARTYA*** osztályhoz rendelt eszköz sorszáma;
0B07 a ***CSATOLÓKARTYA*** sorszáma (0...3) vagy 0FFH, ha nincs kijelölés.

A **OB08 ...OBOFH** cimü memóriarekeszekbe ugyanigy, az *OUTPUT* müveletek esetén hozzárendelt eszközök sorszámát irhatjuk.

Alapbeállitás esetén az *operációs rendszer minden osztályhoz a saját eszközét rendeli*, ezekkel tehát csak akkor kell nekünk foglalkozni, ha ezt az alaphozzárendelést meg akarjuk változtatni. Ez alól egyik tipikus kivétel az az eset, amikor a számitógéphez hajlékony mágneslemezes (*floppy disc*) meghajtó egységet csatlakoztatunk. Ekkor az eredetileg a magnetofonra irányuló **LOAD, SAVE, VERIFY, OPEN,** stb. müveletek automatikusan a diszkegységen hajtódnak végre. Ilyenkor az előbbi táblázat OB05H és OBODH cimü rekeszeiben a 80...83H értékeket találjuk aszerint, hogy a meghajtót a négy csatolókártyahely közül melyikbe illesztettük. Ez a csatolókártyák valamelyik funkcióosztályhoz való hozzárendelésének egyik lehetséges formája. Ha mégis a magnetofont akarjuk használni, akkor elegendő e cimek tartalmát visszairni 05-re.

A hozzárendelések teljes elméletével könyvünkben nem foglalkozunk, erről kimerítően részletes ismertetést talál az Olvasó az **Operációs rendszer** c. kézikönyvben.

Számunkra sokáig elegendő lesz a leggyakrabban használt funkciókódok ismerete. Ezeket soroljuk fel a következőkben.

A gyakrabban használt funkciókódok

Kód	**Leirás** (*alaphozzárendelés esetén*)
VIDEO	
01	A CPU C regiszterében megadott *ASCII kódú karakter* kiirása a képernyő korábban meghatározott helyére.
02	A memória DE regiszterpárban megadott kezdőcimü területén, az egymás utáni memóriarekeszekben *ASCII kódban* tárolt karakterek kiirása a képernyőre, a korábban meghatározott pozicitól kezdve. A szöveg hosszát a BC regiszterpárnak kell tartalmaznia.
03	Karakterpozició kijelölése: C = sorszám (1...24), B = soron belüli pozició.

VIDEO

04 Grafikus üzemmód beállítása. C = 0: 2-es, C = 1: 4-es, C=2: 16 színű üzemmód.

05 Képernyőtörlés.

06 Az elektronsugár elmozgatása új pontpozícióba.
BC = X koordináta (0...1023 = 0000...03FFH),
DE = Y koordináta (0...959 = 0000...03BFH).
Ha a korábbi ponthoz tartozó helyzetében az elektronsugár be volt kapcsolva, akkor az elmozgatás közben a képernyőn egyenes vonalat húz.

07 Relatív elmozdulás a korábbi pontpozícióból.
BC = X irányú elmozdulás, DE = Y irányú elmozdulás (előjeles, 2 byte-os értékek).

08 Az elektronsugár bekapcsolása. Hatására a korábban meghatározott pontpozícióban megjelenik egy pont.

09 Az elektronsugár kikapcsolása.

0A Zárt vonalakkal körülhatárolt alakzat kifestése.

0C Színbeállítás a palettaregiszterekben. A négy színkódot a memóriában négy egymás utáni rekeszbe kell tölteni, ennek kezdőcímére mutasson a DE regiszterpár.

BILLENTYŰZET

91 Megvárja, amíg valamelyik gombot lenyomjuk a billentyűzeten, s annak kódját a C regiszterben adja vissza.

93 A C regiszterben jelzi, hogy van-e lenyomott gomb ebben a pillanatban (C = 0FFH, ha igen).

HANG

33 Hívása előtt PITCH értékét a DE regiszterpárba, a hangerőt a C-be (0...15), míg a hang tervezett időtartamát a B regiszterbe kell tölteni. A hang annyiszor *20 ms* ideig szól, amilyen számot a funkció hívásakor a B regiszter tartalmaz.

2.9 ASCII

Ebben a fejezetben többször szó esett a karakterek ún. *ASCII kódjáról*. A számértékek kezeléséről már sok mindent tudunk, azonban gyakran szükség van a szöveges információk tárolására, feldolgozására is.

A számítástechnikában *a betűket, számjegyeket, írásjeleket úgyancsak egybyte-os számértékek formájában tárolják a memóriában*, és feldolgozásuk is ezekkel az azonosító kódokkal történik. Az egy betűhelynyi területen megjeleníthető, kinyomtatható karakterekhez egy-egy számot rendelünk, ezenkívül bevezetünk ún. *vezérlőkaraktereket* is -- ugyanilyen azonosító számokkal. A vezérlőkarakterek a tv-képernyőn megjelenő vagy nyomtatásra kerülő szöveg kiírási formátumát befolyásolják (pl. soremelés, új sor kezdés).

Az írásjelek és a kiírást vezérlő funkciók megfeleltetésére többféle szabvány is elterjedt, köztük az egyik leggyakoribb az *ASCII (American national Standard Code for Information Interchange*, magyarul: Amerikai Nemzeti Szabványos Információcsere Kódrendszer). Az eredeti kódrendszer 7 bites, nincs azonban akadálya annak, hogy az 1 byte-on ábrázolható 0...255 számok mindegyikéhez rendeljünk valamilyen funkciót. Tekintettel arra, hogy a számok, írásjelek, nagy- és kisbetűk, valamint a legfontosabb vezérlési funkciók együttes száma is jóval kevesebb 256-nál, lehetőség van a szabványban le nem kötött értékek és tetszőleges új funkciók egymáshoz rendelésére. Az így *kibővített kódrendszer* számítógéptípusonként nagy eltéréseket mutat -- nyilvánvalóan összefüggésben a gép jellegzetes plusz szolgáltatásaival és a konstruktőrök leleményességével is.

A TV-Computer esetében érvényes teljes kódrendszert a **Függelékben** adjuk meg.

3. A MONITOR

5.1 A MONITOR elkészítése

Mostantól célszerű, ha kézközelben van a számítógép. Eléggé nagyszabású munkába kezdünk: két lépcsőben elkészítjük magunknak azt a programot, amely a gépi kódú munkáinkhoz szükséges *felhasználói környezetet* biztosítja, így ezután egyik legfontosabb támaszunk lesz.

A ***MONITOR*** maga is egy gépi kódú program. Elkészítését a biztonságos ***BASIC*** rendszerre támaszkodva végezzük el. A rövidesen közlésre kerülő ***BASIC*** program a ***MONITOR-t*** megvalósító számokat a memória megfelelő területére tölti, majd -- önálló és alkalmazásra kész gépi kódú programként -- magnetofonkazettára írja. A későbbiekben már csak magát a ***MONITOR-t*** kell a kazettáról a számítógépbe tölteni.

Az első lépcső a ***MONITOR-t*** alkotó, mintegy 5000 db szám beírása, ill. az ezeket feldolgozó ***BASIC*** programsorok elkészítése. Ez az előttünk álló munka legfáradságosabb, leginkább időt rablő része. Itt sokféle hibát is elkövethetünk. Hogy ezen mielőbb túllegyünk, érdemes lesz a lehető legfigyelmesebben dolgozni. Ha majd készen leszünk, akkor magát a ***BASIC*** programot is azonnal kazettára mentjük -- így egy esetleges áramszünet, vagy a futtatáskor jelentkező végzetes hiba elkövetése esetén sem kell az egészet újra írni.

Egyébként amit csinálni fogunk, alapjában véve egyáltalán nem bonyolult. Még a ***BASIC*** nyelvű programozásban teljesen járatlan Olvasó is könnyűszerrel és biztonságosan el tudja végezni.

Gépelési hibák sajnos adódhatnak, szerencsére azonban a kezünk munkája nyomán megszületó ***MONITOR*** működőképességére nézve veszélyes elírások ellen a ***BASIC*** adatbeviteli program eleve tűrhető védelmet nyújt. Ha ez hibajelzések nélkül fut le, akkor nagy valószínűséggel egy kazettára írt végleges és jól működő gépi kódú program lesz az eredmény. A futtatás és a közben elvégzendő magnetofonkezelési tennivalók azonban már a második lépcsőt jelentik.

Kapcsoljuk be a számítógépet, csatlakoztassunk hozzá magnetofont, s tegyünk bele olyan kazettát, amelyen majd a beírt ***BASIC*** programot tárolni lehet! E műveleteket a **Kezelési Útmutató** szerint végezzük.

Ha a tv képernyőjén megjelent a számítógép ismert bejelentkező nyitóképe, nyomjunk le egy billentyűt. Helyezkedjünk el kényelmesen és kezdjük a munkát!

A *BASIC* programot pontosan úgy kell begépelni, ahogy a könyvben van.

Jól tagolt bekezdéseket látunk. Mindegyik egy sorszámmal kezdődik, utána *BASIC* utasítások állnak. Egy-egy bekezdés általában több sorból áll -- így lesz ez a képernyőn is. A sorok végével azonban nem kell törődni, a beírást folyamatosan végezzük. A sorok végén a gép magától a következő sor elejére ugrik.

Fontos viszont, hogy amikor egy bekezdés végére értünk, akkor az utolsó karakter után mindig nyomjuk le a **'RETURN'** billentyűt! Ezt a könyvben külön nem jeleztük.

Kezdjük el!

```
1 DATA 15,1,0,221,32,85,83,82,150,57,57,57,57,149,255,
0,0,0,1,0,32,1,0,0,0,0,0,0,0,0,0,55!
2 DATA 0,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,
32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,224!
3 DATA 32,32,32,32,32,32,32,32,32,32,32,32,32,13,10,74
,32,52,52,52,52,32,32,32,32,32,32,32,32,32,32,32,81!
4 DATA 32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,0,
13,10,184,84,112,84,0,60,0,85,0,10,255,19,240,253,97!

5 DATA 25,2,19,19,34,47,2,153,69,0,23,255,191,152,22,
241,70,82,76,67,32,82,82,67,32,82,76,32,32,82,82,32,214!
6 DATA 32,83,76,65,32,83,82,65,32,83,76,76,32,83,82,76
,32,66,73,84,32,82,69,83,32,83,69,84,32,73,78,32,14!
7 DATA 32,79,85,84,32,83,66,67,32,76,68,32,32,78,69,71
,32,82,69,84,78,73,77,32,32,76,68,32,32,65,68,68,162!
8 DATA 32,65,78,68,32,65,68,67,32,67,65,76,76,67,67,70
,32,82,69,84,73,82,82,68,32,82,76,68,32,67,80,32,244!

9 DATA 32,67,80,76,32,68,65,65,32,68,69,67,32,68,73,32
,32,68,74,78,90,69,73,32,32,69,88,32,32,69,88,88,148!
10 DATA 32,72,65,76,84,73,78,67,32,74,80,32,32,74,82,32
,32,78,79,80,32,79,82,32,32,80,79,80,32,80,85,83,238!
11 DATA 72,82,69,84,32,82,76,65,32,82,76,67,65,82,82,65
,32,82,82,67,65,82,83,84,32,83,67,70,32,83,85,66,160!
12 DATA 32,88,79,82,32,76,68,73,32,76,68,68,32,76,68,73
,82,76,68,68,82,67,80,73,32,67,80,68,32,67,80,73,70!

13 DATA 82,67,80,68,82,73,78,73,32,73,78,68,32,73,78,73
,82,73,78,68,82,79,85,84,73,79,85,84,68,79,84,73,62!
14 DATA 82,79,84,68,82,32,32,32,32,66,32,32,32,67,32,32
,32,68,32,32,32,69,32,32,32,72,32,32,32,76,32,32,205!
```

```
  15 DATA  32,40,72,76,41,65,31,32,32,40,83,80,41,88,76,32
,32,89,72,32,32,89,76,32,32,40,73,88,43,40,73,89,1!
  16 DATA  43,66,67,32,32,68,69,32,32,72,76,32,32,83,80,32
,32,73,88,32,32,73,89,32,32,65,70,32,32,78,90,32,194!


  17 DATA  32,90,32,32,32,78,67,32,32,67,32,32,32,80,79,32
,32,80,69,32,32,80,32,32,32,77,32,32,32,40,67,41,243!
  18 DATA  32,73,32,32,32,82,32,32,32,40,66,67,41,40,68,69
,41,48,32,32,32,56,32,32,32,49,48,32,32,49,56,32,125!
  19 DATA  32,50,48,32,32,50,56,32,32,51,48,32,32,51,56,32
,32,49,32,32,32,50,32,32,32,110,32,32,32,40,110,41,106!
  20 DATA  32,110,110,32,32,40,110,110,41,41,0,0,15,15,47,
15,33,8,38,15,0,38,1,0,31,1,0,15,1,45,47,0,255!

  21 DATA  0,35,21,21,20,17,15,15,8,33,31,15,0,38,2,0,31,2
,0,15,2,45,49,0,0,33,25,0,15,16,47,15,54!
  22 DATA  34,8,38,16,0,38,3,0,31,3,0,15,3,45,46,0,0,40,25
,0,20,17,16,15,8,34,31,16,0,38,4,0,32!
  23 DATA  31,4,0,15,4,45,48,0,0,40,22,25,15,17,47,15,48,1
7,38,17,0,38,5,0,31,5,0,15,5,45,30,0,110!
  24 DATA  0,40,23,25,20,17,17,15,17,48,31,17,0,38,6,0,31,
6,0,15,6,45,29,0,0,40,24,25,15,18,47,15,118!


  25 DATA  48,8,38,18,0,38,7,0,31,7,0,15,7,45,51,0,0,40,2,
25,20,17,18,15,8,48,31,18,0,38,8,0,89!
  26 DATA  31,8,0,15,8,45,24,0,0,15,1,1,15,1,2,15,1,3,15,1
,4,15,1,5,15,1,6,15,1,7,15,1,31!
  27 DATA  8,15,2,1,15,2,2,15,2,3,15,2,4,15,2,5,15,2,6,15,
2,7,15,2,8,15,3,1,15,3,2,15,234!
  28 DATA  3,3,15,3,4,15,3,5,15,3,6,15,3,7,15,3,8,15,4,1,
15,4,2,15,4,3,15,4,4,15,4,5,236!

  29 DATA  15,4,6,15,4,7,15,4,8,15,5,1,15,5,2,15,5,3,15,5,
4,15,5,5,15,5,6,15,5,7,15,5,10!
  30 DATA  8,15,6,1,15,6,2,15,6,3,15,6,4,15,6,5,15,6,6,15,
6,7,15,6,8,15,7,1,15,7,2,15,18!
  31 DATA 7,3,15,7,4,15,7,5,15,7,6,37,0,0,15,7,8,15,8,1,15
,8,2,15,8,3,15,8,4,15,8,5,32!
  32 DATA  15,8,6,15,8,7,15,8,8,20,8,1,20,8,2,20,8,3,20,8,
4,20,8,5,20,8,6,20,8,7,20,8,86!


  33 DATA  8,22,8,1,22,8,2,22,8,3,22,8,4,22,8,5,22,8,6,22,
8,7,22,8,8,52,1,0,52,2,0,52,187!
  34 DATA  3,0,52,4,0,52,5,0,52,6,0,52,7,0,52,8,0,14,8,1,
14,8,2,14,8,3,14,8,4,14,8,5,162!
  35 DATA  14,8,6,14,8,7,14,8,8,21,1,0,21,2,0,21,3,0,21,4,
0,21,5,0,21,6,0,21,7,0,21,8,35!
  36 DATA  0,53,1,0,53,2,0,53,3,0,53,4,0,53,5,0,53,6,0,53,
7,0,53,8,0,42,1,0,42,2,0,42,77!
```

```
  37 DATA  3,0,42,4,0,42,5,0,42,6,0,42,7,0,42,8,0,28,1,0,
28,2,0,28,3,0,28,4,0,28,5,0,142!
  38 DATA  28,6,0,28,7,0,28,8,0,45,22,0,43,15,0,39,22,47,
39,47,0,23,22,47,44,15,0,20,8,45,50,35,221!
  39 DATA  0,45,23,0,45,0,0,39,23,47,0,0,0,23,23,47,23,47,
0,22,8,45,50,36,0,45,24,0,43,16,0,39,201!
  40 DATA  24,47,13,46,8,23,24,47,44,16,0,52,45,0,50,37,0,
45,2,0,36,0,0,39,2,47,12,8,46,23,2,47,17!

  41 DATA  0,0,0,14,8,45,50,38,0,45,26,0,43,17,0,39,26,47,
35,9,17,23,26,47,44,17,0,21,45,0,50,39,3!
  42 DATA  0,45,27,0,39,17,0,39,27,47,35,16,17,23,27,47,14
, 17,16,53,45,0,50,40,0,45,28,0,43,21,0,39,49!
  43 DATA  28,47,32,0,0,23,28,47,44,21,0,42,45,0,50,41,0,
45,29,0,15,18,17,39,29,47,34,0,0,23,29,47,52!
  44 DATA 0,0,0,28,45,0,50,42,0,0,1,0,10,0,100,3,232,39,16
,80,82,73,78,84,69,82,63,32,40,89,41,80,179!

  45 DATA  114,101,115,115,32,82,69,84,85,82,78,32,102,111
,114,32,109,111,114,101,44,88,32,102,111,114,32,101,110,10
0,33,73,163!
  46 DATA  82,32,32,65,39,70,39,66,39,67,39,68,39,69,39,72
,39,76,39,65,70,32,32,66,67,32,32,68,69,32,32,72,144!
  47 DATA  76,32,32,73,88,32,32,73,89,32,32,83,80,32,32,80
,67,32,32,66,65,83,73,67,63,32,40,89,41,77,79,78,90!
  48 DATA  73,84,79,82,78,85,77,66,69,82,32,40,72,32,111,1
14,32,68,41,83,65,86,69,76,79,65,68,66,101,103,105,110,89!

  49 DATA  58,76,97,115,116,32,58,78,97,109,101,32,58,83,
116,97,114,116,32,116,97,112,101,44,32,116,104,101,110,32,
112,114,216!
  50 DATA  101,115,115,32,82,69,84,85,82,78,33,83,65,86,69
,32,69,82,82,79,82,33,76,79,65,68,32,69,82,82,79,82,48!
  51 DATA  33,70,73,76,69,32,78,79,84,32,70,79,85,78,68,33
,64,46,247,35,78,255,79,0,48,0,32,0,0,0,0,0,131!
  52 DATA  0,245,62,80,24,3,245,62,112,211,2,50,3,0,241,
201,17,0,40,58,229,11,60,200,27,122,179,32,246,60,201,14,
221!

  53 DATA  23,213,229,17,61,26,183,237,82,69,225,209,201,
205,110,32,205,80,32,197,62,120,205,32,212,205,95,32,193,
40,12,197,201!
  54 DATA  62,32,205,32,212,205,95,32,193,32,232,62,32,205
,32,212,175,50,229,11,205,85,32,58,233,11,201,213,197,205,
80,32,52!
  55 DATA  14,1,205,224,212,205,85,32,33,62,26,229,62,32,6
,32,119,35,16,252,225,193,209,201,229,213,197,1,23,1,247,
35,72!
  56 DATA  1,32,0,17,62,26,247,34,193,209,225,201,197,213,
229,205,199,32,205,124,32,1,10,2,17,0,15,245,247,51,241,
225,153!
```

```
  57 DATA  209,193,201,254,91,56,2,230,95,254,58,48,5,254,
48,216,24,6,254,65,216,254,71,63,201,245,230,240,31,31,31,
31,111!
  58 DATA  198,48,254,58,56,2,198,7,119,35,241,245,230,15,
198,48,254,58,56,2,198,7,119,35,241,201,122,205,8,33,123,
205,235!
  59 DATA  8,33,201,26,205,8,33,201,43,126,230,79,254,10,
56,2,214,55,79,43,126,230,79,254,10,56,2,214,55,7,7,7,137!
  60 DATA  7,129,201,6,4,205,219,32,205,242,32,48,5,254,88
,200,24,243,119,35,16,239,229,205,55,33,95,205,55,33,87,
225,191!

  61 DATA  4,201,35,35,205,82,33,200,237,83,70,32,35,205,
82,33,200,237,83,72,32,35,201,205,219,32,254,13,200,254,
120, 32,177!
  62 DATA  246,225,195,197,44,205,219,32,254,13,200,254,
121,40,8,254,120,32,242,225,195,197,44,183,201,33,0,26,17,
1,26,1,210!
  63 DATA  59,0,54,32,237,176,201,54,77,35,35,205,82,33,
202,197,44,35,205,50,33,14,251,35,6,2,205,219,32,205,242,
32,217!
  64 DATA  48,26,254,88,202,180,44,245,205,170,32,241,254,
13,40,6,254,77,32,3,24,209,19,205,41,33,24,213,12,202,180,
44,36!


  65 DATA  119,35,5,32,213,229,43,126,230,79,254,10,56,2,
214,55,235,237,103,27,26,230,79,254,10,56,2,214,55,237,103
,235,221!
  66 DATA  225,24,180,1,10,0,17,62,26,33,194,31,237,176,
205,199,32,235,205,148,33,202,197,44,33,0,112,34,32,23,34,
34,202!
  67 DATA  23,221,33,0,23,195,211,218,54,36,35,35,205,82,
33,40,20,35,26,254,224,48,4,254,32,48,2,62,63,119,35,35,
145!
  68 DATA  205,219,32,254,27,202,180,44,254,224,48,9,254,
32,56,5,119,18,205,199,32,205,170,32,19,205,41,33,24,211,
35,35,44!

  69 DATA  205,50,33,27,205,50,33,19,19,19,205,199,32,205,
170,32,201,197,205,170,32,193,229,217,209,1,4,0,237,176,19
,213,222!
  70 DATA  217,225,197,205,109,34,193,16,237,201,217,33,
142,31,217,17,103,26,6,1,205,128,34,6,3,197,6,4,205,128,34
,193,242!
  71 DATA  16,247,195,180,44,54,66,35,35,205,82,33,40,29,
205,199,32,33,63,32,114,35,115,213,35,235,1,3,0,237,176,
209,126!
  72 DATA  62,205,18,19,62,15,18,19,62,39,18,195,197,44,54
,75,35,35,229,33,63,32,86,35,94,213,35,1,3,0,237,176,105!
```

```
 73 DATA  209,225,205,41,33,205,199,32,195,197,44,54,74,
35,35,205,82,33,202,197,44,205,134,33,235,247,5,237,123,
124, 26,193,12!
 74 DATA  237,115,124,26,229,49,102,26,241,237,71,8,241,
8, 217,193,209,225,217,241,193,209,225,221,225,253,225,237
,123,124,26,59,16!
 75 DATA  59,201,54,73,62,1,24,9,54,68,175,24,4,54,70,62,
2,50,69,32,205,113,33,202,197,44,1,0,2,17,68,32,13!
 76 DATA  205,219,32,205,242,32,48,5,254,88,202,197,44,
119,230,79,254,10,56,2,214,55,235,237,111,235,35,16,227,
205,134,33,164!

 77 DATA  58,69,32,254,0,40,46,254,2,40,85,217,42,72,32,
229,237,91,70,32,167,237,82,229,193,3,209,213,58,68,32,131
,196!
 78 DATA  95,48,1,20,225,237,184,35,58,68,32,71,175,119,
35,16,252,217,195,180,44,217,42,70,32,58,68,32,79,6,0,9,
104!
 79 DATA  229,229,193,42,72,32,167,237,66,229,193,3,225,
237,91,70,32,237,176,58,68,32,71,175,18,19,16,252,217,195,
180,44,9!
 80 DATA  42,72,32,237,75,70,32,167,237,66,218,197,44,35,
229,193,237,91,70,32,58,68,32,18,19,11,121,176,32,246,195,
180,204!


 81 DATA  44,54,65,205,113,33,202,197,44,205,82,33,202,
197,44,237,83,68,32,205,134,33,42,72,32,237,75,70,32,167,
237, 66,214!
 82 DATA  218,197,44,35,229,193,42,70,32,237,91,68,32,237
,82,56,12,42,70,32,237,91,68,32,237,176,195,180,44,42,68,
32,93!
 83 DATA  9,43,229,209,42,72,32,237,184,195,180,44,54,80,
205,113,33,202,197,44,205,199,32,205,170,32,33,98,31,17,62
,26,186!
 84 DATA  1,12,0,237,176,235,205,148,33,253,54,0,0,40,4,
253,203,0,198,205,168,33,237,91,70,32,6,20,197,6,8,205,2!

 85 DATA  170,32,42,72,32,183,237,82,56,117,33,62,26,205,
41,33,35,35,205,50,33,19,229,42,72,32,183,237,82,225,56,2,
144!
 86 DATA  16,239,205,199,32,253,203,0,70,40,53,213,17,0,
26,33,62,26,1,29,0,253,203,0,78,253,203,0,206,40,13,253,
147!
 87 DATA  203,0,142,17,31,26,33,68,26,1,23,0,237,176,253,
203,0,78,32,11,17,0,26,1,62,0,247,66,205,168,33,209,34!
 88 DATA 193,16,153,213,205,170,32,17,62,26,33,110,31,1,
32,0,237,176,205,199,32,205,170,32,209,205,134,33,195,105,
36,205,88!
```

```
  89 DATA  199,32,253,203,0,70,202,180,44,253,203,0,78,40,
247,17,0,26,1,62,0,247,66,195,180,44,54,90,205,113,33,202,
211!
  90 DATA  197,44,205,199,32,205,170,32,33,98,31,17,62,26,
1,12,0,237,176,235,205,148,33,253,54,0,0,253,54,1,0,40,237
!
  91 DATA  4,253,203,1,198,6,20,167,237,91,70,32,42,72,32,
237,82,218,180,44,197,205,170,32,33,62,26,205,41,33,35,26,
182!
  92 DATA  245,205,50,33,19,237,83,70,32,34,68,32,254,221,
32,6,253,54,0,8,24,235,254,253,32,6,253,54,0,12,24,225,236
!

  93 DATA  254,203,202,100,39,254,237,202,32,40,205,140,38
,253,203,0,150,253,203,0,158,241,254,247,32,26,237,91,70,
32,33,69,146!
  94 DATA  26,205,50,33,62,58,33,85,26,119,35,35,205,50,33
,19,237,83,70,32,205,199,32,253,203,1,70,40,17,38,1,14,9!
  95 DATA  32,247,65,37,32,249,17,62,26,1,34,0,247,66,193,
5,194,54,37,205,170,32,17,62,26,33,110,31,1,32,0,237,250!
  96 DATA  176,205,199,32,205,170,32,205,134,33,195,52,37,
42,70,32,126,35,34,70,32,217,235,205,8,33,229,42,68,32,205
,8,70!


  97 DATA  33,225,235,201,42,70,32,94,35,86,35,34,70,32,
213,217,225,235,205,41,33,229,42,68,32,123,205,8,33,122,
205,8,140!
  98 DATA  33,225,235,201,33,148,27,253,203,0,94,200,254,
17,32,10,198,2,253,203,0,86,40,1,60,201,254,7,192,198,6,
253, 79!
  99 DATA  203,0,86,40,1,60,135,135,133,48,1,36,111,1,4,0,
 237,176,217,42,70,32,126,35,34,70,32,42,68,32,205,8,116!
 100 DATA  33,34,68,32,217,235,205,8,33,62,41,119,35,235,
217,225,195,244,38,253,203,0,94,200,198,6,253,203,0,86,40,
1,229!

 101 DATA  60,201,253,203,0,94,200,42,70,32,43,43,126,235,
205,8,33,62,41,119,253,203,0,150,253,203,0,158,201,33,88,
28,56!
 102 DATA  71,135,48,1,36,128,48,1,36,133,48,1,36,111,126,
35,229,17,124,26,135,135,48,1,20,131,48,1,20,95,1,4,237!
 103 DATA  0,33,78,26,235,237,176,6,2,217,17,83,26,217,225
,126,35,254,0,200,229,79,120,254,1,32,6,217,62,44,18,19,
202!
 104 DATA  217,121,254,45,48,87,254,25,40,31,217,205,19,
38,135,135,133,48,1,36,111,0,0,62,32,1,4,0,237,160,226,243
,93!
```

```
105 DATA 38,190,32,248,217,16,199,225,201,42,70,32,126,
35,34,70,32,229,42,68,32,205,8,33,225,254,128,48,21,133,24
,3,188!
106 DATA 195,134,44,48,1,36,111,229,217,225,235,205,41,
33,235,217,24,211,237,68,79,125,145,48,1,37,111,24,234,254
,47,56,67!
107 DATA 26,254,48,56,16,62,40,217,18,19,217,205,243,37,
62,41,18,19,217,24,219,205,243,37,217,24,213,254,46,56,16,
62,103!
108 DATA 40,217,18,19,217,205,220,37,62,41,18,19,217,24,
234,205,220,37,217,24,228,26,205,8,33,253,203,0,94, 40,5,
19,77!

109 DATA 26,205,8,33,19,237,83,70,32,34,68,32,245,230,
192,32,67,241,245,203,63,203,63,203,63,71,33,128,26,135,
135,133,230!
110 DATA 48,1,36,111,17,78,26,1,4,0,237,176,241,230,7,
245,205,98,38,33,152,27,135,135,133,48,1,36,111,17,83,26,
176!
111 DATA 241,62,32,1,4,0,237,160,226,189,39,190,32,248,
205,113,38,195,124,37,167,23,23,23,61,71,33,160,26,135,135
,133,35!
112 DATA 48,1,36,111,17,78,26,1,4,0,237,176,241,245,230,
192,71,241,245,144,203,63,203,63,203,63,245,198,48,17,83,
26,175!


113 DATA 18,19,62,44,18,19,241,203,39,203,39,203,39,128,
71,241,144,205,98,38,33,152,27,135,135,133,48,1,36,111,62,
32,161!
114 DATA 1,4,0,237,160,226,26,40,190,32,248,205,113,38,
195,124,37,26,205,8,33,19,237,83,70,32,34,68,32,33,31,31,
2!
115 DATA 54,0,35,54,0,35,54,0,33,31,31,245,254,128,210,
71,41,230,7,254,3,40,94,210,205,40,254,2,40,52,254,1,146!
116 DATA 32,24,33,31,31,54,13,35,54,30,35,241,203,63,203,
63,203,63,214,7,119,62,237,195,121,37,33,31,31,54,12,35,39
!

117 DATA 241,203,63,203,63,203,63,214,7,119,35,54,30,62,
237,195,121,37,241,230,63,203,95,33,31,31,40,5,54,22,35,24
,185!
118 DATA 3,54,14,35,54,17,35,203,63,203,63,203,63,203,63
,198,15,119,195,92,41,54,15,35,241,6,0,203,95,32,1,4,62!
119 DATA 5,4,40,3,54,48,35,230,63,203,63,203,63,203,63,
203,63,198,15,119,35,5,4,32,2,54,48,195,92,41,33,31,146!
120 DATA 31,241,245,230,15,254,4,32,4,54,16,24,127,254,5
,32,4,54,17,24,119,254,13,32,4,54,25,24,111,230,7,254,234!
```

```
121 DATA 6,32,23,241,54,18,35,203,63,203,63,203,63,214,8
,32,4,54,35,24,88,198,41,119,24,83,241,203,111,32,43,54,
255!
122 DATA 15,35,203,95,40,18,203,103,40,7,54,8,35,54,32,
24,60,54,32,35,54,8,24,53,203,103,40,7,54,8,35,54,254!
123 DATA 31,24,42,54,31,35,54,8,24,35,230,15,254,8,56,4,
54,27,24,25,54,26,24,21,230,7,23,23,71,241,230,31,224!
124 DATA 203,63,203,63,203,63,128,198,54,119,24,1,241,62,
237,195,121,37,1,15,0,235,33,211,31,237,176,205,199,32,205
,170,125!

125 DATA 32,205,219,32,254,120,202,197,44,254,100,40,72,
254,104,32,240,54,72,35,205,82,33,40,237,205,134,33,54,61,
35,62,159!
126 DATA 1,167,1,0,0,229,24,2,203,127,245,229,235,22,32,
175,41,203,17,203,16,143,254,10,56,3,214,10,35,21,32,240,
118!
127 DATA 198,48,84,93,237,74,225,32,223,119,35,241,40,
251,209,205,199,32,195,180,44,54,68,35,6,6,205,219,32,254,
13,40,56!
128 DATA 20,254,120,202,197,44,254,48,56,240,254,58,48,
236,119,35,16,232,195,197,44,235,213,217,33,0,0,217,33,88,
31,6,102!



129 DATA 6,197,70,35,78,35,27,26,254,68,40,20,214,48,60,
197,217,209,71,5,40,3,25,16,253,217,193,16,228,195,197,44,
232!
130 DATA 217,229,217,209,193,225,54,61,35,205,41,33,205,
199,32,195,180,44,197,213,229,205,127,32,1,10,2,17,0,15,
245,247,18!
131 DATA 51,241,225,209,193,201,33,62,26,62,4,245,205,33
,42,254,8,32,12,241,254,4,40,243,60,5,43,54,32,24,236,254,
44!
132 DATA 27,32,6,241,225,225,195,180,44,254,13,32,5,241,
183,32,218,201,205,242,32,56,213,87,241,183,40,207,114,205
,113,42,238!

133 DATA 24,201,245,229,197,98,247,35,76,247,33,193,225,
241,4,35,61,201,1,9,6,247,35,17,234,31,1,6,0,247,34,1,133!
134 DATA 9,12,205,53,42,205,55,33,50,70,32,205,55,33,50,
71,32,201,1,11,6,247,35,17,240,31,1,6,0,247,34,1,242!
135 DATA 11,12,205,53,42,205,55,33,50,68,32,205,55,33,
50,69,32,201,33,63,26,62,8,245,205,33,42,254,27,32,6,241,
128!
136 DATA 225,225,195,180,44,254,8,32,12,241,254,8,40,233
,60,43,54,32,5,24,226,254,13,32,5,193,62,8,144,201,254,48,
25!
```

```
 137 DATA  56,214,254,58,56,8,254,65,56,206,254,91,48,202,
87,241,183,40,196,114,205,113,42,24,190,1,13,6,247,35,17,
246,238!
 138 DATA  31,1,6,0,247,34,1,13,12,205,193,42,201,183,192,
247,84,247,212,1,16,1,213,247,35,209,1,11,0,247,34,1,95!
 139 DATA  16,13,205,33,42,254,120,32,4,225,195,180,44,254
,13,32,238,201,1,16,1,197,247,35,1,32,32,197,247,33,193,16
,21!
 140 DATA  250,193,247,35,1,4,0,247,34,201,213,1,16,1,247,
35,17,252,31,1,30,0,247,34,1,16,32,205,33,42,254,120,224!

 141 DATA  32,4,225,195,180,44,254,13,32,238,209,205,65,43
,201,17,62,26,247,83,183,192,33,0,26,6,16,119,35,16,252,60
,241!
 142 DATA  50,1,26,50,4,26,42,72,32,34,2,26,33,0,26,6,16,
197,78,247,81,35,193,16,248,237,75,72,32,237,91,70,51!
 143 DATA  32,247,82,183,192,247,84,183,201,247,5,1,6,1,
247,35,17,226,31,1,4,0,247,34,205,129,42,205,161,42,205,8,
222!
 144 DATA  43,17,26,32,205,28,43,40,224,50,62,26,42,68,32,
237,91,70,32,175,237,82,56,233,35,34,72,32,17,226,31,205,
243!



 145 DATA  89,43,205,126,43,183,202,180,44,175,24,213,17,
62,26,247,211,183,192,33,0,26,6,16,119,35,16,252,33,0,26,
6,217!
 146 DATA  16,197,247,209,113,35,193,16,248,183,192,237,91
,70,32,237,75,2,26,247,210,183,192,247,212,183,201,183,200
,254,128,245,240!
 147 DATA  1,16,1,247,35,241,32,8,1,15,0,17,48,32,24,6,1,
11,0,17,37,32,247,34,247,212,1,16,16,205,33,42,83!
 148 DATA  254,120,32,7,225,195,180,44,254,13,32,238,201,
247,5,1,6,1,247,35,17,230,31,1,4,0,247,34,205,129,42,205,
154!

 149 DATA  8,43,50,62,26,17,230,31,205,89,43,205,251,43,
205,42,44,183,32,217,195,180,44,237,115,124,26,49,124,26,
253,  229,44!
 150 DATA  221,229,229,213,197,245,217,229,213,197,217,8,
245,8,237,87,103,237,95,111,229,237,123,124,26,225,229,43,
43,43,34,126,156!
 151 DATA  26,195,31,45,0,205,170,32,17,62,26,33,204,31,1,
7,0,237,176,205,199,32,205,170,32,205,219,32,254,109,202,
182,216!
 152 DATA 33,254,114,202,153,34,254,98,202,180,34,254,107,
202,221,34,254,106,202,250,34,254,105,202,49,35,254,100,
202,55,35,254,159!
```

```
 153 DATA  102,202,60,35,254,97,202,240,35,254,59,202,55,
34,254,112,202,59,36,254,104,202,18,34,254,122,202,9,37,
254,110,202,200!
 154 DATA  97,41,254,115,202,184,43,254,108,202,92,44,24,
171,0,0,253,33,74,32,1,22,1,247,35,17,204,31,1,7,0,247,
220!
 155 DATA  34,33,62,26,229,62,32,6,32,119,35,16,252,225,
195,200,44,245,14,14,0,0,241,225,32,217,201,0,0,0,0,0,231!
```

Ha eddig eljutott a kedves Olvasó, akkor már a munka nehezén túl van! Annál is inkább, mivel az eddigi meglehetősen monoton és érdektelen sorok után a *feldolgozó rész* következik -- ez pedig már beírás közben is nyújt információkat a működésről.

Folytassuk a programot!

```
 200  LOMEM 12288: GRAPHICS 2
 205  B=6607: RESTORE 1: S=0
 210  FOR J=1 TO 155
 215  B=B+32: S=S+1: C=0: PRINT S;
 220  FOR I=0 TO 31: READ E: C=(C+E) AND 255: POKE (B+1),E
: NEXT I
 225  READ CE: IF CE<>C THEN CLS: PRINT S;". sor hibás!":
PRINT: STOP
 230  NEXT J

 300  CLS: PRINT "Hibátlan!": PRINT: PRINT: PRINT "Állítsa
 felvételre és indítsa el a magnetofont!": PRINT: PRINT"Ha
 mindez kész, nyomja le a 'RETURN' billentyűt!"
 305  GET B$: IF ORD(B$)<>13 THEN 305
 310  CLS: PRINT "Felvétel!"
 315  A$="MONITOR": B=6718: POKE B,8:  FOR C=1 TO 8:  POKE
(B+C),ORD(A$(C)): NEXT C: POKE 8262,239: POKE 8263,25:POKE
8264,78: POKE 8265,45: A=USR(11134)
 320  CLS: PRINT "Kész! Állítsa meg a magnetofont!"
```

Ezzel pedig valóban készen is vagyunk!

Most mielőtt bármi mást csinálnánk, nyomjuk le a RETURN-t, állítsuk felvételre és indítsuk el a magnetofont, majd gépeljük be:

```
SAVE "BASICMON"
```

és ismét: RETURN!

Ha a felvétel kész, akkor egy kicsit már nyugodtabbak lehetünk.

Állítsuk le a magnetofont, a kazettát tekercseljük vissza a felvétel elejére és ellenőrizzük, hogy hibátlan-e. Gépeljük be:

VERIFY

és nyomjuk le a RETURN-t, majd indítsuk el a magnetofont! Erre a parancsra számítógépünk összehasonlítja, hogy pontosan az van-e a szalagon, amit a gépbe írtunk.

Ha igen, akkor az ellenőrzés végén egyetlen "*ok*" üzenetet kapunk a képernyőn.

Ha nem, akkor vagy a magnetofon, vagy az összekötő vezeték, annak csatlakoztatása, vagy pedig a kazetta nem megfelelő! Ha rögtön az elején hibajelzést kapunk, a kábellel vagy a magnetofonnal van valami baj. Ha azonban a képernyő keretének csíkozódása jelzi a beolvasás kezdetét, s ezután észlel hibát a gép, akkor *feltétlenül cseréljük ki a kazettát!*

Bármilyen hiba esetén ismételjük meg a felvételt és csak ha teljesen biztonságosan a kazettán van a programunk, akkor menjünk tovább!

Ezután már egyszerű a dolgunk. Helyezzük a magnetofonba azt a kazettát, amelyen a kész *MONITOR*-t tárolni fogjuk és kezdjük el a beírt *BASIC* program futtatását! Írjuk be:

RUN

és nyomjuk le a RETURN billentyűt! Figyeljük a képernyőt! Helyes működés esetén egymás után megjelennek a feldolgozott **DATA** sorok sorszámai. Ha a program valahol hibás adatot észlel, akkor a feldolgozás megszakad, s a

" n . sor hibás!"

üzenetet kapjuk -- az *n* helyére a képernyőn természetesen a hibás **DATA** sor száma kerül. Ilyenkor a hibás sort javítsuk ki (**LIST** *n* parancs, majd javítás), vagy pedig írjuk be újra! Nagyon fontos, hogy *a kijavított programot **mindig vegyük fel újra az első kazettára is!***

A gép által észlelt egyéb hibák esetén a **200**-as sortól kell ellenőrizni és kijavítani a programot.

Programunk előbb-utóbb helyesen működik, s a képernyőn megjelenik a következő felirat:

"Hibátlan!

Állítsa felvételre és indítsa el a magnetofont!
Ha mindez kész, nyomja le a RETURN billentyűt!"

Tegyük ezt!

Ha pedig a felvétel is kész, akkor *ettől a pillanattól rendelkezésünkre áll a gépi kódú* ***MONITOR*** *program!*

Kapcsoljuk ki a számítógépet, hogy a memória teljesen törlődjön, majd kis várakozás után kapcsoljuk be újra.

A következő fejezetrész tanulmányozása nélkül nincs lehetőség arra, hogy programunkat közvetlenül ellenőrizzük. Ha az eddig leírt műveleteket hibátlanul végeztük és a gép sem észlelt semmi rendellenességet, akkor gyakorlatilag biztosak lehetünk benne, hogy a kazettára került gépi kódú program is helyes.

Ha a ***BASIC*** program pontosan megegyezett a könyvvel, akkor ez a bizonyosság természetesen 100 %-os.

Sajnos lehetnek azonban az adatokat tartalmazó sorokban olyan -- egymást kompenzáló -- hibák, amelyek a beépített ellenőrzések során nem derültek ki. Ezek csak a ***MONITOR*** használata közben lépnek majd fel és esetleg teljes működésképtelenséget is okozhatnak.

A fejezet következő részében a ***MONITOR*** használatával, parancsaival ismerkedünk meg. A tanulás mellett azonban *figyeljünk arra is, hogy az általunk készített, saját* ***MONITOR*** *programnak pontosan úgy kell viselkednie, ahogy azt leírjuk!*

Bármilyen eltérést tapasztalunk, annak oka a ***BASIC*** program begépelése közben elkövetett -- egymást kompenzáló, tehát *legalább kétszeres* -- hiba. Sajnos ilyenkor csak egy dolgot tehetünk. A számítógép ki-, majd bekapcsolása után ***töltsük be újra az első kazettáról a*** ***BASIC*** *programot, és soronként ellenőrizzük a könyvvel való egyezését!* Az összeolvasáshoz célszerű egy másik személy közreműködését is kérni. A felfedezett hibákat javítsuk ki, *vegyük fel újra kazettára*, majd a programot lefuttatva készítsük el az új ***MONITOR*** programot!

Beolvasási hiba esetén (rossz kazetta) ugyanezt lehet tenni. A ***BASIC*** program betöltésével és ismételt futtatásával bármikor *friss* ***MONITOR*** *programot* kapunk. Természetesen *a bizonytalannak minősülő kazettát érdemes* ***egyszer s mindenkorra kiiktatni,*** *mert csak újabb bosszúságokat okoz.*

3.2 A MONITOR parancsai

A számítógépet hozzuk használatra kész állapotba, a *MONITOR-t* tartalmazó kazettát tekercseljük az elejére, majd a billentyűzeten a szokásos módon gépeljük be:

LOAD

és lenyomva a RETURN billentyűt, indítsuk el a magnetofont. A betöltés kezdetén a képernyőn megjelenik a

READING: MONITOR

felirat, s a memóriába töltődik az a rendszerprogram, amely további munkánkhoz már nem nélkülözhető.

A betöltés után kapcsoljuk ki a magnetofont, s nézzük a képernyőt!

A *MONITOR* program máris elindult. Működését azzal kezdte, hogy *a CPU-t kiléptette a ROM-beli BASIC-ből*, s szolgáltatásaival mostantól a rendelkezésünkre áll.

Egyben tudomásul kell vennünk azt is, hogy ezzel a *BASIC* minden szolgáltatását is elveszítettük, csakis magunkra, eddig megszerzett tudásunkra vagyunk utalva. Nincsenek szövegszerkesztő *BASIC* funkciók, nem írhatunk és nem törölhetünk ismert kulcsszavakat, nincs hibakezelés. Ha valamit elrontunk, annak következményeit magunknak kell vállalni és elviselni.

Mit is akarunk?

Célunk az, hogy közvetlenül a memóriába írt számokkal, gépi kódú utasításokkal vezéreljük a CPU-t. Ehhez:

-- meghatározott című memóriarekeszekbe -- meghatározott számokat kell *írni*;
-- szükség lehet a korábban beírt adatok *módosítására*, egy-egy memóriaterület *vizsgálatára*;
-- ha a gépi kódú programunkban valamit kifelejtünk, akkor a beszúráshoz helyet kell csinálni, ami a beszúrás helye utáni hosszabb *memóriaterület elmozgatásával* járhat;
-- más esetben egyszerűen *törölni kell* a programból néhány byte-ot;

-- alapvetően fontos az is, hogy a megírt gépi kódú programot *azonnal és közvetlenül tárolhassuk, kazettán vagy mágneslemezen*, onnan pedig *bármikor visszatölthessük.*

Ez utóbbi lehetőség most különösen fontos. A programokat még az első kipróbálás előtt ajánlatos *háttértárolóra* vinni, hiszen elég egyetlen byte rossz beírása ahhoz, hogy a CPU-t eltérítse a tervezett működéstől. A mikroprocesszor ui. hű rabszolgaként végrehajt minden utasítást, amit a memóriába beírt számokkal mondunk, s ha véletlenül nem a tervezett parancsot adjuk ki, azt is. Ilyen esetben pedig a CPU esetleg *elhagyhatja a programunkat*, s mint egy elszabadult kis szellem, száguldozhat a memóriában -- ennek következményei pedig kiszámíthatatlanok. Rossz esetben, a memória legkülönbözőbb címeiről beolvasott parancsokra *a programunk egy-egy byte-ja vagy akár az egész is átíródhat, a MONITOR is megsérülhet.* Ezért jó, ha a program beírása után rögtön készítünk arról a szalagon egy másolatot. Ha a baj megtörtént, akkor a kazettáról újra betölthető, akár a *MONITOR-ral* együtt. A hibát persze meg kell keresni és ki kell javítani!

A *MONITOR* lehetővé teszi programjaink beírását a memóriába, ezek ellenőrzését, javítását, módosítását, elindítását, sőt azt is, hogy hibakeresés végett a program végrehajtását annak bármelyik pontján megszakítsuk. Így akár utasításonként, lépésről lépésre követhetjük programjaink működését, ellenőrizhetjük a memóriarekeszek tartalmát, a CPU regisztereinek állapotát, s hogy egyáltalán az történik-e, amit terveztünk.

Ha a CPU véletlenül -- programozási hiba folytán -- elhagyja a programunkat, sokszor segít a gép alján levő *RESET* gomb egyszeri megnyomása. Ilyenkor a rendszer automatikusan visszatér a *BASIC-be*, de egyszerű lehetőség van arra, hogy ismét belépjünk a *MONITOR-ba*, s ha nincs nagy baj, folytatni lehet a gépi kódú munkát. Célszerűbb azonban, ha ilyen eltérülés esetén inkább betöltjük újra a kazettára mentett programot. Kellemetlen lehet ui., ha az eltérülés miatt mégis átíródott egy-két általunk is használt memóriarekesz, ezt nem vettük észre, s egy ilyen *"vírust"* tartalmazó programot fejlesztünk tovább. Később nagyon nehéz az ilyen hibákat megtalálni.

Ilyen munkára készüljünk tehát fel, s mindehhez megfelelő segítséget nyújt *segédprogramunk: a MONITOR.* Mivel ezután alapvető *munkaeszközként* fogjuk használni, célszerű nagyon jól megismerni a használatát, parancsait, az általa nyújtott szolgáltatásokat.

Amikor a ***MONITOR*** *elindul*, a képernyő aljára kiírja:

```
MONITOR
*
```

és az *"x"* villog: *a* ***MONITOR*** *munkára kész, dolgozhatunk.*

Vegyük sorra a ***MONITOR***-*ban* használható parancsokat. Mindegyik egy betűből áll: ez azonosítja a kért funkciót. A betűhöz tartozó billentyű lenyomása után még több-kevesebb adatot gépelhetünk be. A parancsok ismertetésekor a zárójelek közé tett paramétereket nem kötelező begépelni, ha azonban több ilyen szerepel egymástól vesszővel elválasztva, akkor *közülük egyet feltétlenül* használni kell.

M parancs: írás a memóriába

A parancs formátuma:

M CCcc vv (aa) (jj) (RETURN, X, M, egyéb)

Ez a ***MONITOR*** egyik legösszetettebb parancsa.

Nyomjuk le az **M** billentyűt (**SHIFT** nélkül, a ***MONITOR*** parancsaihoz ezt soha nem kell használni). Az **M** a képernyőn is megjelenik (az *x* helyén), a villogó *x* pedig kettővel jobbra lép, automatikusan kihagyva tehát az **M** után egy betűhelyet.

Most be kell írnunk azt a *memóriacímet*, amellyel foglalkozni akarunk. Ezt jelöli a formátum megadásában szereplő: **CCcc**. A címet *mindig négyjegyű hexadecimális* értékként kell megadni. Legyen ez most pl. *0F3AH*, tehát nyomjuk le egymás után ezt a négy gombot a billentyűzeten. Figyeljük meg, hogy a beírt számjegyek a képernyőn is azonnal megjelennek, s az *x* is mindig eggyel jobbra lép.

A 4. jegy (az **A**) beírása után pedig egy betűhelyet kihagyva, *a* ***MONITOR*** *azonnal kiírja a 0F3AH című memóriarekesz jelenlegi tartalmát*, s a villogó *x* ismét két helylyel tovább lép:

```
M 0F3A 00 x
```

Ebből azt tudjuk meg, hogy a kiválasztott memóriarekeszben -- ebben a pillanatban -- a *00 hexadecimális érték* van. Ezt jelöltük a formátum sorban **vv**-vel.

A villogó *x* azonban mutatja, hogy ezzel még nincs vége.

Most *két hexadecimális jegy* beírásával megadhatjuk, hogy a *00 helyett mi legyen ezután* a megadott memóriarekeszben. Ezt jelöltük aa-val. Gépeljünk be pl. *A7-et!*

Ez is kiíródik a képernyőn, és a 7-es lenyomása pillanatában *a OA7H adatot a MONITOR máris beírja a OF3AH címre.*

Így tölthetjük be a memóriába gépi kódú programjainkat, ill. az utasításokat jelentő számokat.

Az *x* azonban ismét két hellyel jobbra lépett, tehát még mindig *van lehetőségünk változtatásra.* Valóban, begépelhetünk még *két hexadecimális jegyet,* s akkor ezt írja be a memóriarekeszbe a *MONITOR* (**jj**, javítás).

Több lehetőségünk azonban már ebben a parancssorban nincs, bár az *x* az utolsóként beírt jegy után villog.

Most többféle lehetőség közül választhatunk. Ha a **RETURN** billentyűt nyomjuk le, akkor *a MONITOR a következő memóriacímre áll*: az előbbi sor alatt a képernyőn megjelenik a cím és azonnal annak tartalma is kiíródik:

```
M OF3A 00 A7                          <RETURN>
OF3B 00 x
```

és folytathatjuk a beírást, most már ezzel a memóriarekesszel. (A parancsok soraiban hegyes zárójelek közé a ténylegesen lenyomandó billentyű jelét írjuk.)

Azt is megtehetjük, hogy az elsőként begépelt, vagy akár a *MONITOR* által kiírt memóriatartalom után *azonnal a* **RETURN**-*t nyomjuk le.* Ekkor a memóriarekesz tartalma nem változik, a soron következő cím és annak tartalma íródik ki. Így -- változtatás nélkül -- *a memória egy rövid területét vizsgálhatjuk meg.*

Ha az **X** billentyűt nyomjuk le az adat beírása után, vagy előtte is bármikor, akkor *a MONITOR kilép ebből a parancsból,* amit a bejelentkezésekor megismert szöveg kiírásával jelez:

```
M OF3A 00 A7                          <RETURN>
OF3B 00                               <X>
MONITOR
x
```

Így lehet *abbahagyni* a megkezdett memóriaírási folyamatot.

Ha az **M** billentyűt nyomjuk le az adat beírása után, akkor *a MONITOR kilép ebből a parancsból, de mindjárt el is kezd egy másik memóriaírási folyamatot.* Az **M** után egy új, tetszőleges címet gépelhetünk be, s ettől kezdve minden ugyanúgy zajlik, ahogy azt a parancs ismertetése elején leírtuk. Távoli memóriacímek tartalmának megváltoztatása esetén használjuk ezt a lehetőséget.

Más billentyűk lenyomására a *MONITOR* az új sor elején megismétli a memóriacímet, s az adatbeírás ugyanúgy folytatódhat, ahogy korábban megbeszéltük. Ha pl. az **A7** begépelése után a **T** billentyűt nyomjuk le, akkor a következő történik:

```
M 0F3A 00 A7                    <T>
0F3A A7 x
```

Tehát ismét átírhatjuk (mégpedig újra kétszer is) az adott cím tartalmát.

Ha azonban a cím vagy adat beírása közben nyomunk le (*véletlenül*) olyan billentyűt, amely nem értelmezhető *hexadecimális* jegyként, azt a *MONITOR* egyszerűen *nem veszi figyelembe.*

A *MONITOR* itt leírt viselkedése a többi parancsnál is hasonló, így a továbbiakban csak az eltérésekre hívjuk fel a figyelmet.

F parancs: memóriaterület feltöltése azonos adattal

A parancs formátuma:

F KKkk VVvv aa (RETURN, X)

Akkor használjuk, ha meghatározott kezdő- és végcím közötti területet *ugyanazzal az adattal* akarunk feltölteni. Az **F** billentyű lenyomása után -- értelemszerűen -- be kell írnunk a *kezdő memóriacímet négy hexadecimális jegygyel* (**KKkk**), majd a *végcímet* (**VVvv**), végül pedig azt a *kétjegyű számot, amivel az így meghatározott memóriarekeszeket fel kell tölteni* (**aa**).

Az **X** billentyű lenyomásával bármikor megszakíthatjuk a parancsot, az **aa** beírása után pedig -- ha közben nem gondoltuk meg magunkat -- a **RETURN** lenyomásával *érvényesíthetjük.* Ekkor a *MONITOR* elvégzi a memóriarekeszek feltöltését.

Ha pl. ezt írjuk be:

```
F 0F3A 0F42 A7                  <RETURN>
```

akkor a *MONITOR a 0F3AH címtől kezdve 0F42H-ig 0A7H-t ír a memóriarekeszekbe.* Ellenőrzésül az *M parancsal* vizsgáljuk meg ezeket! Írjuk be **M** után a kezdő memóriacímet, majd soronként nyomkodjuk egymás után a **RETURN**-t. Ezt látjuk:

```
F 0F3A 0F42 A7                    <RETURN>
MONITOR
M 0F3A A7                         <RETURN>
0F3B A7                           <RETURN>
0F3C A7                           <RETURN>
.    .                               .
:    :                               :
0F42 A7                           <RETURN>
0F43 00 x
```

Ezután az **X** lenyomásával lépjünk ki a parancsból.

P parancs: memóriaterület kiírása

A parancs formátuma:

P KKkk VVvv

Itt is meg kell adni annak a memóriaterületnek a *kezdő- és zárócímét*, amelyek tartalmát látni akarjuk.

A zárócím utolsó jegyének begépelése után *a **MONITOR** megkérdezi:*

```
PRINTER? (Y)
```

vagyis, hogy a kiírás *nyomtatóra* (is) történjék-e? Ha van a számítógépünkhöz nyomtató csatlakoztatva, s ha valóban azt akarjuk, hogy a megadott memóriarekeszek tartalma papíron is megjelenjen, akkor *nyomjuk le az* **Y** *billentyűt.* Ha csak a képernyőn akarjuk megjeleníteni az adatokat, **RETURN** választ kell adni. Az **X** lenyomásával most is bármikor megszakíthatjuk a parancsot.

Ezután a ***MONITOR*** kiírja a megadott memóriarekeszek tartalmát. A képernyő egy sorába nyolc egymást követő byte kerül, a sor elejére pedig az *első adat memóriacíme.* Ha nyomtatást is készítünk, ott egy sorban *16 db számot ír a **MONITOR***, a sorok elejére pedig most is az első adat címe kerül. Pl.:

```
P 0F3A 0F43
PRINTER? (Y)                      <RETURN>
0F3A A7 A7 A7 A7 A7 A7 A7 A7
0F42 A7 00
MONITOR
x
```

Nagy memóriaterületek kiíratása közben, *ha a képernyő megtelik*, akkor a ***MONITOR*** félbeszakítja a parancs további végrehajtását és kiírja:

```
Press RETURN for more,X for end!
x
```

azaz *nyomjuk le a* **RETURN** *billentyűt, ha mehet tovább, vagy az* **X** *billentyűt, ha itt be akarjuk fejezni.*

A parancs végrehajtása után (ha a meghatározott teljes memóriaterület kiírása megtörtént), a villogó *x* már az új sor elején áll, jelezve, hogy a *MONITOR* kész az új parancs fogadására.

Másik példa (a nyomtatóra vonatkozó kérdésre válaszként most is a **RETURN** billentyűt nyomjuk le):

```
P DADA DAEA
PRINTER? (Y)                          <RETURN>
DADA 31 AC 16 21 03 17 7E B7
DAE2 36 00 C4 10 DE DD 36 05
DAEA 20
MONITOR
x
```

Ez a *rendszer-ROM-ba írt BASIC parancselemző program* egy darabja.

Z parancs: disassemblált lista készítése

A parancs formátuma:

Z KKkk VVvv

A *disassemblálás* azt jelenti, hogy a memória kiválasztott területén számkódokkal tárolt gépi kódú programot *mnemonikokkal megfogalmazott utasításszimbólumok formájában kapjuk vissza*. Hasonlóan ahhoz, ahogyan a tervezés során a programjainkat magunk is írjuk.

A parancsban szereplő **KKkk** és **VVvv** most is a memória kiválasztott területének *kezdő-* és *végcíme*.

A *MONITOR* itt is megkérdezi:

```
PRINTER? (Y)
```

és ugyanúgy kell válaszolnunk, mint a *P parancsnál*. A végrehajtás folyamata is hasonló.

Példa (ha nyomtatónk nincs, akkor a PRINTER? (Y) kérdésre a **RETURN** billentyűt nyomjuk le):

```
Z DADA DAEA
PRINTER? (Y)                          <RETURN>
DADA 31AC16     LD    SP,16AC
DADD 210317     LD    HL,1703
DAE0 7E         LD    A,(HL)
DAE1 B7         OR    A
DAE2 3600       LD    (HL),00
DAE4 C410DE     CALL  NZ,DE10
DAE7 DD360520   LD    (IX+05),20
MONITOR
x
```

Íme, így néz ki az előbbi *ROM-terület disassemblált listája.* Talán nem is kell tovább nagyon hangsúlyoznunk, hogy a *Z parancsot* milyen sokszor használjuk majd. Segítségével elővarázsolhatjuk a memóriában tárolt programot, mégpedig úgy, hogy azonnal látjuk, miről van szó! A ***MONITOR*** feltünteti az utasítások kezdő memóriacímét, a hexadecimális számkódokat, s elénk tárja azok szimbolikus jelentését is.

A parancs: memóriaterület áthelyezése

A parancs formátuma:

A KKkk VVvv CCcc (RETURN, X)

Ezzel a paranccsal *az* **A** *után írt* **KKkk** *kezdő-* és **VVvv** *végcímű memóriaterület tartalmát* ***másolhatjuk át a*** **CCcc** *kezdőcímű másik memóriaterületre.*

Ha pl. ezt írjuk be:

```
A DADA DAEA 0F3A
```

és lenyomjuk a **RETURN** billentyűt, akkor a *MONITOR* az előbb vizsgált *ROM-beli* programrészt átmásolja a *0F3AH* kezdőcímű memóriaterületre. Erről az *M, P vagy Z parancsal* egyszerűen meggyőződhetünk. Pl.:

```
Z 0F3A 0F4A
PRINTER? (Y)                          <RETURN>
0F3A 31AC16     LD    SP,16AC
0F3D 210317     LD    HL,1703
0F40 7E         LD    A,(HL)
0F41 B7         OR    A
0F42 3600       LD    (HL),00
0f44 C410DE     CALL  NZ,DE10
0F47 DD360520   LD    (IX+05),20
MONITOR
x
```

A parancs érdekessége, hogy akkor is helyesen hajtódik végre, ha a másolandó és a cél-memóriacímek *egymást részben átfedik.*

Az eddig leírt parancsokkal már tudunk a memóriába gépi kódú programokat írni, módosítani, a beírt memóriaterületek tartalmát számok sorozata formájában vagy disassemblált listán ellenőrizni.

A most következő parancsok *programjaink fejlesztése során* tesznek jó szolgálatot. Segítségükkel a memóriába már beírt programokon hajthatók végre jelentősebb változtatások és azok kipróbálását, tesztelését teszik lehetővé.

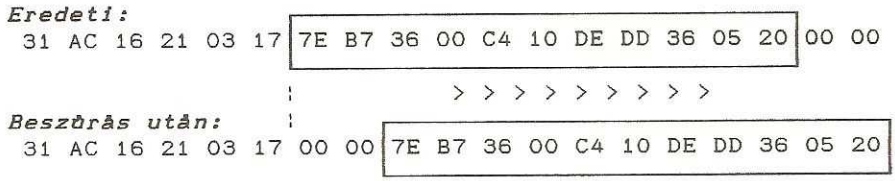
I parancs: beszúrás

A parancs formátuma:

I KKkk VVvv nn (RETURN, X)

Akkor használjuk, ha a programunk *korábban beírt utasításai közé utólag szeretnénk további utasításokat írni. Ezt a MONITOR az adott hely utáni byte-ok elmozgatásával teszi lehetővé.*

Ha *pl.* az előbbi programrészbe a *OF3DH és OF4OH* kezdő memóriacímű utasítások *közé be akarunk szúrni még egy 2 byte-os utasítást* -- természetesen anélkül, hogy a program utána következő része megváltozna --, akkor nyilvánvaló, hogy *a OF4OH és OF4AH címek közötti programrészt 2 byte-tal előre kell tolni a nagyobb memóriacímek felé.* Erre természetesen csak akkor van lehetőség, ha a memória *OF4AH* címét követő rekeszekben már számunkra érdektelen adatok állnak, tehát nyugodtan átírhatók. Az elmozgatás mechanizmusát a 8. ábra mutatja.



8. ábra Az I parancs

A *MONITOR-nak* meg kell adni a beszúrás helyének *kezdőcímét* (**KKkk**), a programrész *végét* (**VVvv**, *az elmozgatás*

ezzel fog kezdődni), s azt, hogy *hány byte-ot akarunk beszúrni (***nn***).* A felszabadított memóriarekeszekbe *a MONITOR OO-t ír.*

Az előbbi művelet elvégzéséhez tehát a következő parancsot kell megadni:

```
I OF40 OF4A 02                          <RETURN>
MONITOR
×
```

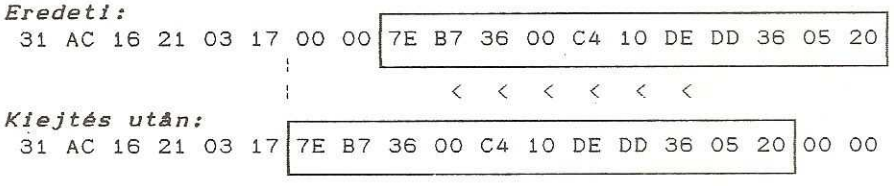
Ha a parancs beírása után az **X**-et nyomjuk le, akkor a beszúrási műveletet a *MONITOR* nem hajtja végre.

D parancs: kiejtés (törlés)

A parancs formátuma:

D KKkk VVvv nn (RETURN, X)

Ez éppen az *I parancs* fordítottja. *A MONITOR* most a **KKkk** *és* **VVvv** *címek közötti memóriaterületet visszafelé mozgatja*, ezáltal a megadott kezdőcímtől **nn** db byte tartalma elveszik, törlődik. Helyükre az utána álló programrész utasításai kerülnek (l. 9. ábra).



9. ábra A D parancs

Az előző parancsnál leírt beszúrás hatását (amelyet az ábra is mutat) a következő parancs állítja helyre:

```
D OF40 OF4C 02                          <RETURN>
MONITOR
×
```

J parancs: gépi kódú program indítása

A parancs formátuma:

J CCcc (RETURN, X)

Ezzel a paranccsal indíthatjuk el a korábban beírt, ellenőrzött és célszerűen már kazettára is mentett programunkat. A *MONITOR* a parancsot úgy hajtja végre, hogy a megadott CCcc címet a mikroprocesszor *PC programszámláló regiszterébe* tölti, ezáltal CPU a következő utasítást már a megadott memóriarekeszből olvassa be. Ez pedig éppen azt jelenti, hogy a mikroprocesszor ettől a pillanattól kezdve *az általunk írt programot hajtja végre.*

Hogy ezután mi történik, az már kizárólag rajtunk, pontosabban az általunk írt program helyességén múlik.

B parancs: töréspont elhelyezése a programban

A parancs formátuma:

B CCcc

Ez a *MONITOR* egyik legegyszerűbb és egyben talán legpraktikusabb parancsa. A CCcc cím 4. jegyének beírása után *a MONITOR a megadott címtől kezdődően 3 byte tartalmát átmásolja a saját munkaterületére, s ezek helyére a CALL 270F utasításnak megfelelő 0CDH, 0FH és 27H hexadecimális számokat írja.* Ennek eredményeként, amikor a CPU a programunk végrehajtása közben ehhez a memóriacímhez ér, a szubrutinhívást jelentő utasításbyte-ok hatására *visszatér a MONITOR-ba* (*270FH* a belépési cím). A program végrehajtása tehát a parancsban megadott CCcc címnél megáll. A programfutás megszakításának ezt a formáját nevezik (találóan) *töréspontnak. A MONITOR egyidejűleg csak egy töréspontot képes kezelni.*

Alkalmazásának célja programjainkban nyílvánvaló. Újonnan beírt programok első futtatásai alkalmával a legfigyelmesebb munka ellenére is adódhatnak olyan hibák, amelyekre a program írása közben nem gondoltunk. Ezek miatt az első futtatások eredménye kisebb-nagyobb *eltérülés* -- ilyen esetekben a hibakeresés alapvető módszere az, hogy a program különböző pontjain, nagyon módszeresen *töréspontokat* helyezünk el. Így lehetőségünk van arra, hogy a kiválasztott utasításssorozat működését *akár soronként ellenőrizzük.* Ezzel a módszerrel -- és megfelelő türelemmel -- a legrejtettebb hibák is kideríthetők.

A hibák javítása után -- ami bizony elég sokszor a program bizonyos részeinek módosításával, átírásával jár -- az elhelyezett töréspontot meg kell szüntetni. Erre szolgál a *MONITOR* következő parancsa.

K parancs: töréspont megszüntetése

A parancs formátuma:

K

Ezzel a paranccsal a programunkban korábban *B parancsal* elhelyezett töréspontot szüntetjük meg. Kiadása után -- a **K** billentyű lenyomásakor azonnal -- *a MONITOR a munkaterületéről visszamásolja az ott elhelyezett utasításbyte-okat a CCcc címre.* Így helyreállítja a töréspontal korábban elrontott eredeti utasításssorozatot. Ezután -- programunk ismételt futtatása esetén -- a végrehajtás már úgy történik, mint ha a töréspontot korábban nem is helyeztük volna el.

Mégegyszer felhívjuk a figyelmet arra, hogy a *MONITOR* egyidejűleg csakis egy töréspontot képes kezelni, mindig azt, amit legutoljára beírtunk. Ha tehát a programunkban máshol akarunk újabb töréspontot elhelyezni, vagy ha a régire már nincs szükség, akkor a *K paranccsal* feltétlenül szüntessük azt meg, s csak ezután helyezzünk el új töréspontot!

R parancs: regisztertartalom kiírása

A parancs formátuma:

R

Ez is a *MONITOR* legtöbbet használt és leghasznosabb parancsai közé tartozik. Az **R** billentyű lenyomása után a képernyőn *kiíródik a CPU valamennyi regiszterének tartalma a következő formában:*

```
IR      0066

A'F'    B9A8
B'C'    00FF
D'E'    0265
H'L'    9C39

AF      1302
BC      2213
DE      0265
HL      4599

IX      1700
IY      BFFF
SP      1698
PC      46F1
```

Ilyenkor *azok az értékek láthatók, amelyek a MONITOR-ba való belépés pillanatában a regiszterekben voltak.* A képernyőn tehát *egy korábbi állapotot látunk*, s az *R parancs* ismételt használata esetén mindaddig ugyanazokat az értékeket kapjuk, amíg egy *J paranccsal* nem futtatunk ismét valamilyen programot.

Ha az Olvasó először esetleg furcsának is tartaná ezt a megoldást, könnyen belátható, hogy a működés valójában *csakis így értelmes.* Gondoljuk meg ui., hogy amikor a CPU egy program végrehajtásából visszatér a *MONITOR-ba*, akkor regiszterei rövid működés után már *a MONITOR programjához tartozó értékekkel töltődnek fel.* Ha az *R parancs* úgy működne, hogy mindig az adott pillanatban érvényes regisztertartalmakat írná a képernyőre, akkor -- az előbbiek miatt -- a programunk működéséről nem tudnánk meg semmit.

A programozás szempontjából *a CPU MONITOR-beli munkájának részletei teljesen érdektelenek.* Sokkal inkább azt akarjuk tudni, hogy a programunknak azon a pontján, ahol a töréspontot elhelyeztük, a végrehajtásai folyamatnak ebben a pillanatában mit tartalmaznak a CPU regiszterei. Ezek ismeretében tudunk a programunk eddig végrehajtott részének helyes vagy hibás működéséről messzemenő következtetéseket levonni. Végsősoron ui. valamennyi utasítás a CPU regisztereiben, de legalábbis azok közreműködésével zajlik, azokban valamilyen nyomot hagy. Így *programunk munkájának megítéléséhez ezek a legelemibb és legalapvetőbb információk. A MONITOR-ba való belépés pillanatában tárolásra kerül a mikroprocesszor valamennyi regiszterének aktuális értéke, s az R parancs kiadásakor ezeket láthatjuk a képernyőn.*

A regiszterek tárolása *a MONITOR munkaterületén* voltaképpen még a *töréspont* működéséhez tartozik. A *J parancs* kiadásakor, tehát amikor egy programot elindítunk vagy folytatunk, akkor *a MONITOR ezeket, a munkaterületén tárolt értékeket tölti vissza a CPU regisztereibe.* Ez a programvégrehajtás folyamatosságának alapvető feltétele. A törésponttal megszakított programot ezáltal a CPU úgy folytathatja tovább, mint ha a töréspontot el sem helyeztük volna. Programjainkban nem marad nyoma annak, hogy valamelyik ponton megszakítottuk és a *MONITOR*-ral közben különböző vizsgálatokat végeztünk.

A *B parancs megállítja* a kipróbálás alatt álló program végrehajtását, *a J pedig továbbindítja.* Ezek -- és közben az összes többi *MONITOR* parancs alkalmazása -- jelentik a *MONITOR* segédprogram tulajdonképpeni komplex, céltudatos, programfejlesztő munkaeszközként való, a funkciónak megfelelő működtetését.

A programok beírását, fejlesztését, a hibakeresést és javítást támogató parancsok után a *MONITOR* további, nagyon is praktikus segédparancsait ismertetjük.

S parancs: memóriatartalom tárolása kazettán

A parancs formátuma:

S

Az **S** billentyű lenyomása után a *MONITOR* az ún. ***SAVE*** (ejtsd: szejv = mentés) *almenüt* írja a képernyőre:

```
SAVE
     Begin: x
```

A ***begin*** szó után be kell írni a tárolandó program *első memóriarekeszének címét, négy hexadecimális jeggyel*. Ha elrontjuk, a **DEL** billentyűvel visszaléphetünk, s az elrontott jegyeket kijavíthatjuk. A villogó *x* minden érvényes jegy beírása után eggyel jobbra lép, azonban összesen csak négy jegy megadását teszi lehetővé. A hexadecimális jegyként nem értelmezhető billentyűk lenyomása hatástalan marad. Az utolsó karakter begépelése után pedig már csak a **RETURN** vagy -- további javítás céljából -- a **DEL** használatát fogadja el.

Ha a **RETURN** billentyűt nyomjuk le, akkor a parancs feldolgozása folytatódik:

```
SAVE
     Begin: <kezdőcím>
     Last:  x
```

A *last* szó után be kell írni *programunk végcímét*, az általunk használt legnagyobb memóriacímet. A beírás folyamatának vezérlése ugyanaz, mint a kezdőcím esetén volt.

Utána a parancs végrehajtása így folytatódik:

```
SAVE
     Begin: <kezdőcím>
     Last:  <végcím>
     Name:  x
```

Meg kell még adnunk a tárolásra kerülő memóriaterület, program, összefoglaló idegen szóval: *file* (ejtsd: fájl) nevét. Ennek hossza legfeljebb nyolc karakter lehet, s *csak nagybetűkből és számokból állhat. Hosszú magánhangzók használata sem megengedett*. A név beírását a *MONITOR* a címeknél leírtak szerint vezérli.

Ha a **RETURN** lenyomásával jelezzük, hogy a tervezett nevet beírtuk, a képernyőn a következő figyelmeztető felirat jelenik meg:

```
SAVE
     Begin: <kezdőcím>
     Last:  <végcím>
     Name:  <a file neve>

Start tape, then press RETURN!x
```

Azaz: *indítsd el a magnetofont, majd nyomd le a* **RETURN** *billentyűt!* Ekkor tekercseljük a kazettát arra a helyre, ahová a felvételt készíteni akarjuk, kapcsoljuk a magnetofont felvétel üzemmódba, indítsuk el, majd nyomjuk le a **RETURN** gombot!

Az előbbi figyelmeztető szöveg helyén ismét megjelenik a

```
SAVE
```

felirat, s *a MONITOR a megadott névvel, a megadott memóriaterület pontos másolatát írja a kazettára.*

Megjegyezzük, hogy a parancs használata közben bármikor lenyomva az **ESC** billentyűt (kivéve a felvétel ideje alatt) a parancs végrehajtása félbeszakad, s a

```
MONITOR
x
```

felirat jelzi, hogy a *MONITOR* kész az újabb parancsok fogadására. Ugyanez történik a felvétel végén is.

Ha a számítógépünkhöz *lemezegység* van csatlakoztatva, akkor a tárolási művelet eleve ezen hajtódik végre.

L parancs: programbetöltés kazettáról

A parancs formátuma:

```
L
```

Ezt a parancsot ugyanúgy vezérli a *MONITOR*, ahogy az *S parancsnál* leírtuk. Két különbség mégis van.

Betöltéskor *csak a kezdőcímet* kell begépelni, azaz a file a kazettáról a memóriába bármilyen címre betölthető.

A név beírása helyett -- ha kazettáról olvasunk be -- elegendő lenyomni csak a **RETURN** billentyűt. *Ekkor a MONITOR a kazettán éppen következő programot tölti be,* bármi is annak a neve. Ha azonban a *"Name:"* után *file-nevet* is írunk, akkor a *MONITOR* csak ezt fogja beolvasni.

Ha a kazettán más nevű programokat talál, azokat a *BASIC-ből* ismert

FOUND: <név>

kiírással jelzi.

Némileg más a helyzet *lemezről* való betöltés esetén. Ha nevet nem adunk meg, akkor most is a lemezen található első file töltődik be. Ha azonban beírjuk a program nevét, akkor -- amennyiben van egyáltalán ilyen file a lemezen -- *azonnal ezt tölti be a* ***MONITOR***, ha pedig nincs, akkor a

FILE NOT FOUND!

szöveg jelenik meg a képernyőn. Értelme: *ilyen nevű file nem található ezen a lemezen!* A ***MONITOR*** ezután újrakezdi az *L parancsot*, s a betöltést -- most már megfelelő *file-névvel* -- megismételhetjük.

$ parancs: szöveg beírása a memóriába

A parancs formátuma:

$ KKkk (l) **(karakter, RETURN, vagy ESC)**

Ezzel a paranccsal tetszőleges *szövegek ASCII kódjait tölthetjük be a memóriába, a* **KKkk** *címtől* kezdődő területre.

A **$** jel után meg kell adni négy hexadecimális jeggyel annak a memóriarekesznek a címét, *ahová az első karakter kódját akarjuk írni*. Válaszként a ***MONITOR*** -- az *M parancshoz* hasonlóan -- a cím mellé most *azt a karaktert írja, amelynek ASCII kódja az adott című memóriarekeszben van.* Ha a szám nem egy kiírható karaktert jelent (hanem ún. vezérlőkarakter kódja), akkor a cím után egy *?* jelenik meg. Ezután a kívánt karakterhez tartozó gombot lenyomva, *a* ***MONITOR*** *előállítja ennek ASCII kódját, s ezt írja be* a memória adott című rekeszébe. A parancs több karakterből állló szövegek beírását támogatja, ezért a továbblépéshez most nem kell a **RETURN** billentyűt lenyomni. A ***MONITOR*** egy-egy karakter beírása után azonnal rááll a következő memóriacímre. Ha mégis a **RETURN**-t nyomjuk le, akkor az adott című memóriarekesz tartalma nem változik, de a ***MONITOR*** most is továbblép a memóriában.

A parancsból az **X** billentyű lenyomásával most nem lehet kilépni, hiszen az is egy betű, *kódját éppen úgy tárolja a* ***MONITOR***, *mint bármelyik más betűjét*. Ha a szöveg

beírását abba akarjuk hagyni, akkor az **ESC** billentyűt nyomjuk le.

A szövegek beírásakor a TV-Computerben megjeleníthető bármilyen karakter használható.

Példaként az *"Ablak"* szó beírása a *4000H memóriacímtől a következőképpen történik:*

```
$ 4000 ? A                          <A>
4001 ? b                            <b>
4002 ? l                            <l>
4003 ? a                            <a>
4004 ? k                            <k>
4005 ? x                            <ESC>
MONITOR
x
```

A *4005H* cím kiírása után nyomjuk le az **ESC**-t. Erre a következő sor elején megjelenik a *MONITOR* ismert bejelentkező felirata, s a munka új parancsok beírásával folytatható.

N parancs: számrendszer konverzió

A parancs formátuma:

N

A hexadecimális és 10-es számrendszerbeli számok átalakítására szolgál. Az **N** billentyűt lenyomva megjelenik a következő felirat:

```
NUMBER (H OR D)                     (H, D, X)
```

Ha az **X** billentyűt nyomjuk le, akkor a parancs végrehajtása a szokásos módon félbeszakad.

A **H** billentyű lenyomása után egy *négyjegyű hexadecimális számot* gépelhetünk be. Ezután a **RETURN** billentyűt lenyomva *a MONITOR kiírja a begépelt szám 10-es számrendszerbeli alakját.* Pl.:

```
NUMBER (H OR D)                     <H>
H4000=16384
MONITOR
x
```

Fordítva, ha 10-es számrendszerbeli számot kell átváltanunk hexadecimális alakra, akkor a **D** billentyűt kell

lenyomni. Utána *legfeljebb ötjegyű decimális számot* írhatunk be *0 és 65535 között, s a* **RETURN** *lenyomását követően az adott szám hexadecimális alakja jelenik meg.*

Az elrontott beírás csak a parancs ismételt kiadásával javítható. Ha számjegyként nem értelmezhető karaktereket gépelünk be, azokat a *MONITOR* nem veszi figyelembe.

Ha a 10-es számrendszerben begépelt szám *65535-nél nagyobb, akkor belőle a MONITOR kivon 65536-ot, s csak a maradékot alakítja át* (tehát a szám alsó négy hexadecimális helyi értékre eső részét). Nézzünk erre is egy példát!

```
NUMBER (H OR D)                    <D>
D65546=000A
MONITOR
*
```

Ezt a kis segédparancsot jól használhatjuk a programfejlesztés közben adódó számolásokhoz.

H parancs: visszatérés a BASIC-re

A parancs formátuma:

H

A **H** billentyű lenyomása után, a téves begépelések elkerülése végett *a MONITOR rákérdez:*

```
BASIC ? (Y)
```

Ha valóban ezt akarjuk, akkor nyomjuk le az **Y** billentyűt. Ha azonban véletlenül került sor a **H** billentyű lenyomására, akkor a **RETURN** választ adjuk. Erre a

```
MONITOR
*
```

kiírást kapjuk, tehát *a MONITOR-ban dolgozhatunk tovább.*

Időnként előfordul, hogy a gépi kódú programozás közben bizonyos számolásokat, vagy akár egy-egy részfeladat kipróbálását *BASIC-ben* akarjuk elvégezni. Ilyenkor a *H paranccsal* átmenetileg elhagyhatjuk a *MONITOR*-t.

Ha ismét gépi kódban akarunk dolgozni, akkor a

PRINT USR(9999)

parancsot írjuk be!

A *H parancs* a *BASIC közjátékok* számára a *7000... 7FFFH* területet állítja be. Ügyeljünk arra, hogy ezen a részen ne legyen hasznos gépi kódú programunk, vagy ha mégis van, akkor a *BASIC-be* való belépés előtt ezeket másoljuk át valamilyen szabad memóriaterületre.

Ezzel, kedves Olvasó, a *MONITOR-ról* szóló fejezet végére értünk. Mint láttuk, programunk hozzáférhetővé teszi számunkra a számítógép gépi kódú programozásához szükséges eszközeit, hatékonyan támogatja a mikroprocesszor és a memória követlen kezelését, segíti a programozói munkát. Mindezt pedig imponálóan kicsi memóriaterület lefoglalásával teszi: *a MONITOR a 19EFH...2D3FH című memóriarekeszekben található* számokban testesül meg. Nyilvánvaló, hogy programozáskor ezt a területet nem szabad átírni, *megsérülése egész addigi munkánkra végzetes lehet!*

Emlékeztetem az Olvasót a CPU utasításkészletének tárgyalása elején tett megjegyzésekre (1.3.4.1 szakasz). A *MONITOR* nem teszi lehetővé, hogy programjainkat a gépbe is *assembly mnemonikokkal* írjuk be -- a megtervezett utasításokat közvetlenül a megfelelő számok formájában kell a gépbe vinni. Ismét hangsúlyozni kell, hogy a hatékony, időben és munka ráfordításban gazdaságos gépi kódú programozásnak *nem ez* a gyakorlati módja. Mi azonban most elsősorban tanulunk, mégpedig a legalapvetőbb dolgokban kell jártasságot szereznünk. Munkánkat a számkódok testközelisége kicsit lassítja, ugyanakkor azonban *különlegesen alapossá* is teszi. A számítógép működésének megértésében, a hozzá való egész viszonyulásunkban *meghatározó jelentőségű ismeretekre tehetünk szert, s csakis ezek birtokában képzelhető el a MONITOR-nál fejlettebb rendszerprogramok igazán hatékony alkalmazása is.*

E programok általában kisebb-nagyobb változtatásokkal *tartalmazák a MONITOR megismert funkcióit*, azonban lényegesen túlléphetik azzal, hogy lehetővé teszik az utasítások *mnemonikokkal* való beírását is. Ha az Olvasó majd komolyabb és bonyolultabb gépi kódú programozásba kezd, feltétlenül ajánlom, hogy szerezzen be egy ilyen, a kereskedelmi forgalomban (sajnos eléggé borsos áron) kapható, a *professzionális munkát* támogató felhasználói programot. Kimerítőnek sajnos nem mondható *kezelési leírásaik* alapján további tanulással és némi türelemmel elsajátítható ezek használata.

Könyvünk további részében *programozunk, s segédeszközként a MONITOR-t használjuk.* Az Olvasó igyekezzék az ebben a fejezetben megismert parancsokat jól megtanulni, fontos, hogy azokat mielőbb, mindig a célnak legmegfelelőbben és önállóan használni tudja! Ehhez természetesen könyvünk további részében is minden segítséget megadunk.

4. PROGRAMOZUNK

4.1 Memóriakezelési gyakorlatok

A gép bekapcsolását követően -- az ún. *inicializáló program* lefutása után -- a memórialapozási alapállapot: *U0-U1-U2-SYS*, tehát a 0000...3FFFH címeken (0.lap) az U0 RAM-ot, a 4000...7FFFH és 8000...0BFFFH (1. és 2. lap) címeken az U1 és U2 RAM-okat, a legfelsö: 0C000...0FFFFH területnek megfelelö 3. lapon pedig a SYS-t, a rendszer-ROM fö részét tudja közvetlenül elérni a CPU.

Mint tudjuk, *az U0 RAM 0000...19EEH címtartományba esö része az operációs rendszer és a BASIC számára foglalt.* Itt találhatók a rendszervältozók, a munkaterületek, a definiálható karakterek számára fenntartott terület, a funkcióhívások hozzárendelési táblája stb.

A 19EFH címtöl felfelé a memória elvileg a miénk. Tudnunk kell azonban, hogy az ún. *BASIC verem a 0BFFFH legnagyobb RAM-címtöl lefelé terjed* (64k típusú gépek). Bár jöllehet gépi kódban dolgozunk, esetenként sor kerülhet olyan ROM-rutinok hívására, amelyek ezen a memóriaterületen dolgoznak.

A 0C000...0FFFFH címeken elhelyezkedö *ROM-ot* gépi kódú munkáink során elöre kidolgozott szubrutinok gazdag gyüjteményének tekintjük.

Munkaeszközünk a MONITOR az U0 RAM-ban a 19EF...2D3FH memóriaterületet tölti be.

Így tehát ebben az alapkonfigurációban 36 kbyte-nyi terjedelmü memória áll programjaink rendelkezésére. Ha pedig ez is kevésnek bizonyulna (!), a háttérben ott van még az *U3 RAM, amely további 16 kbyte méretü.*

Nagyon sokszor célszerü a videomemória byte-jainak közvetlen elérése, ahol azonban -- egy-két kivételtöl eltekintve -- csakis a képernyön megjelenö információkat tároljuk.

Igen nagy memóriaterület ez. Egy közepes bonyolultságú és kidolgozottságú játékprogram is csak 10--30 kbyte-ot foglal le a memóriában (pl.: MRALEX: 17k, RING: 24k vagy OTHELLO: 20k) -- emiatt tehát valószínüleg nem kell aggódnunk. Vannak persze ennél jóval nagyobb méretü programok is, pl. a "NYUSZI OLVASNI TANÍT" cimü kisgyermekeknek szóló játékos sorozat első részének TV-Computerre irt változata csaknem 60 kbyte.

Az alapkonfigurációt használva a memória 2D40...0BFFF cimü tartományába irhatjuk programjainkat. Ez a leggyakoribb állapot. Az *U3 RAM*, a *videomemória* vagy az *EXT/IOMEM szegmens* használatához (itt található a *rendszer-ROM* folytatása) a memórialapozást meg kell változtatnunk. Mint tudjuk, ez a *02-es portra* irt megfelelő számmal lehetséges. A *megszakításkezelő program* és a *funkcióhivások* azonban meghatározott memórialapozást állitanak be, emiatt a helyes müködéshez szükséges, hogy *a 0003H cimü memóriarekeszbe is beirjuk a 02-es portra küldött adatot.* Az aktuális lapozási érték megőrzésére egyébként nekünk, magunknak is szükségünk lehet.

A memórialapozás tehát a következő két lépésben változtatható meg (a sorrend lényeges!):

1. az új lapozási érték bemásolása a *0003* memóriacimre;
2. az érték kiküldésre a *02-es portra.*

A Z80-as CPU utasitáskészletének tárgyalásakor láttuk, hogy bármelyik portra csak az A regiszterből lehet adatot küldeni, ezért azt először ebbe a regiszterbe kell irni!

Példaként az *U0-U1-U2-SYS* lapozás helyett hozzuk be a második lapra a *videomemóriát*! Ehhez az 50H lapozási érték tartozik. Teendőink tehát:

Beirjuk az A regiszterbe az új lapozáshoz szükséges értéket, 50H-t:

LD A,50

Amikor a CPU ezt végrehajtja, az akkumulátor tartalma 50H lesz. (A leirt utasitást szokás igy ejteni: "löud á-ba ötven", azaz: betöltés A-ba, 50).

Ezután az új lapozási értéket beirjuk a 0003 cimre. Azért volt célszerü az A-ba (és nem a C-be) irni a kivánt adatot, mert konkrét memóriacimre csak az A-t lehet másolni. Ez a programlépés a következő *mnemonikkal* irható le:

LD (0003),A

Amikor a CPU ezt végrehajtja, hatására a memória 0003H című rekeszének tartalma is 50H lesz.

Hátra van még a memórialapozás tényleges érvényesítése. Az A regiszter tartalmát kell a 02-es portra küldeni:

OUT (02),A

E művelet után a mikroprocesszor a *8000...0BFFFH* címeken máris a *VID szegmenst* éri el. Ez azt jelenti, hogy a képernyőn megjelenő pontokhoz tartozó számok a 8000H... 0BFFFH címekről közvetlenül kiolvashatók vagy fordítva: az erre *a memóriaterületre írt számok azonnal megfelelő pontokat jelenítenek meg a képernyőn.*

Próbáljuk ki!

Töltsük be a kazettáról a *MONITOR*-t! Megírjuk első, rövidke gépi kódú programunkat! A kezdőcím legyen pl. 3000H. A beírandó utasításokat soronként egymás alá írjuk, majd a megfelelő táblázatból (l. a Függelékben) kikeressük és a mnemonikok mellé odaírjuk a hozzájuk tartozó számokat. Könyvünkben a hivatkozások megkönnyítésére a sorok elején a javasolt beírási memóriaterületnek megfelelő kezdőcímeket is odaírjuk.

PR1. Írás a videomemóriába

```
3000  LD    A,50        3E,50
3002  LD    (0003),A    32,03,00
3005  OUT   (02),A      D3,02
3007  LD    A,6B        3E,6B
3009  LD    (942A),A    32,2A,94
300C  LD    A,70        3E,70
300E  LD    (0003),A    32,03,00
3011  OUT   (02),A      D3,02
3013  CALL  MONITOR     CD,0F,27
```

A számokkal majd később foglalkozunk, most nézzük csakis az utasításokat.

A 3000--3005 címeken lévő utasításokat korábban már megbeszéltük. Ezzel teljesen megegyező utasításokat látunk a 300C--3011 címeken, csak itt új lapozási értékül 70H-t írtunk, ami visszaállítja az eredeti memórialapozási konfigurációt, az *U0, U1, U2 és SYS szegmenst* tesszük ismét a CPU elé.

A 3007--3009 címen álló utasításokkal a *942AH* címre *6BH*-t töltünk. Ez most a *videomemória* egy rekesze, tehát hatása a képernyőn azonnal látható lesz: négy pont jelenik meg egymás mellett, sorrendben a 2., az 1., a 3., majd ismét a 2. *palettaregiszter* által meghatározott színnel.

Igen fontos az utolsó utasítás. Vele azt írjuk elő a mikroprocesszornak, hogy térjen vissza a ***MONITOR-hoz.*** Ez minden programunk fontos befejező eleme, hiszen a tulajdonképpeni programhoz tartozó utolsó utasítás után a CPU mindenképpen folytatná a végrehajtást a következő memóriarekesz beolvasásával, majd az azután következővel és így tovább. Mindezeknek azonban a mi programunkhoz már semmi köze nincs. Sőt előbb vagy utóbb a CPU olyan műveletekkel találkozna, amelyek a további munkáinkat is veszélyeztetnék. *Jegyezzük jól meg: a CPU-t nekünk kell állandóan kézben tartani! Programjaink végén is meg kell mondanunk, hogy a mikroprocesszor mit csináljon azután!*

A ***MONITOR*** ezt a feladatot biztonságosan megoldja úgy, hogy amikor látszólag nem történik semmi, az alatt az idő alatt a CPU-t csakis a gép billentyűzetének figyelésével foglalkoztatja -- készen állva bármelyik pillanatban a hozzá intézett parancsaink fogadására.

Gyakoroljuk az előbbi programban mnemonikokkal leírt utasítások olvasását, értelmezését! Figyeljük meg, hogy ezekkel a rövid szimbólumokkal megadott utasításssorozat a műveleteket tömören és egyértelműen jellemzi, kis gyakorlás után a program jól olvasható, a működés követhető, áttekinthető, érthető! Ne menjünk tovább, amíg nem látjuk világosan a program működését!

A mnemonikok mellé írt *számokat* a megfelelő táblázatokból vettük. A 3EH hexadecimális szám az adatot az A regiszterbe töltő utasítás kódja. Ugyanígy a 0D3H, 32H és 0CDH is *egy-egy műveletet azonosít*. Utánuk áll az utasítások *operandusa*, azok a konkrét számok, amelyekkel a megadott műveleteket el kell végezni: melyik számot kell A-ba tölteni (a 3000H-s címen pl. *50H*), melyik *portra* kell az adatot küldeni (*02*), melyik memóriarekeszbe kell az A-t másolni (a 3002H és 300EH címeken *0003H*, a 3009H címen *942AH*), mi a hívandó szubrutin pontos memóriacíme (*270FH*).

Fontos, *hogy a kétbyte-os adatokat fordított sorrendben kell a memóriába írni!* Ez a CPU belső működési mechanizmusából adódik.

Ha mindezeket jól megértettük, írjuk be programunkat a számítógépbe! Az első utasítást a 3000H című memóriarekeszbe írjuk (lehetne máshová is, a rendelkezésünkre álló terület: 2D40...0BFFFH). A ***MONITOR***-t már korábban betöltöttük a gépbe, így a programot megvalósító számokat az *M parancsal* írhatjuk be. Soronként, egymás után gépeljük be az előbb kigyűjtött számokat a 3000H-tól kezdődő memóriaterületre. Ha valamit elrontunk, az *M parancsnál* ismertetett lehetőségek valamelyikével kijavíthatjuk. Hibátlan munka esetén a képernyő így fest:

```
M 3000 00 3E                    <RETURN>
3001 00 50                      <RETURN>
3002 00 32                      <RETURN>
3003 00 03                      <RETURN>
3004 00 00                      <RETURN>
3005 00 D3                      <RETURN>
3006 00 02                      <RETURN>
3007 00 3E                      <RETURN>
.    .  .                         .
:    :  :                         :
3013 00 CD                      <RETURN>
3014 00 0F                      <RETURN>
3015 00 27                        <X>
MONITOR
x
```

Minden szám beírása után **RETURN**-nel megyünk tovább és a végén az **X**-szel fejezzük be.

Ha ezzel készen vagyunk, ellenőrizzük a munkánkat. Célszerű azonnal *disassemblált listát* készíteni a programunkról, ezért gépeljük be:

Z 3000 3013

majd a PRINTER? (Y) kérdésre adjunk **RETURN** választ. A ***MONITOR*** elénk tárja a beírt program listáját, ami -- ha jól dolgoztunk -- lényegileg teljesen megegyezik az eredeti, papírra írt listával:

```
Z 3000 3013
PRINTER? (Y)                    <RETURN>
3000 3E50      LD   A,50
3002 320300    LD   (0003),A
3005 D302      OUT  (02),A
3007 3E6B      LD   A,6B
3009 322A94    LD   (942A),A
300C 3E70      LD   A,70
300E 320300    LD   (0003),A
3011 D302      OUT  (02),A
3013 CD0F27    CALL 270F
MONITOR
x
```

Minden sor elején a memóriacím áll, utána az utasításhoz tartozó hexadecimális kódok, majd pedig a műveletek mnemonikjai.

Ha bármelyik sorban eltérést tapasztalunk, akkor ellenőrizzük a listában is szereplő számokat azzal, amit be kellett volna írni, s a sorok elején kiírt memóriacímre ráállva az *M parancsal*, javítsuk ki a hibás értéket. Egyébként elég egyetlen számot eltévesztenünk, utána az

egész lista teljes összezavarodottságot mutathat, hiszen a rossz érték miatt az utána álló utasításssorozat értelme más lehet.

Az esetleges hibák kijavítása után munkánkat ismét ellenőrizzük a *Z parancsal*!

Ha eddig minden jónak látszik, akkor már csak el kell indítanunk a programot. Ehhez a *MONITOR J parancsát* használhatjuk. Írjuk be:

J 3000

és nyomjuk le a **RETURN** billentyűt! Ha közben a szám beírását is elrontottuk, akkor a **RETURN** helyett az **X**-et lenyomva kiléphetünk a parancsból, s ezután újra kezdhetjük.

A **RETURN** lenyomása után a képernyő felső harmadának alján, a sor közepétől kissé jobbra megjelenik a négy színes pont, alul pedig a

MONITOR
*

felirat -- jelezvén, hogy a CPU végrehajtotta programunkat és már vissza is tért a *MONITOR-ba*, amely várja további parancsainkat.

Ha nem így történt, akkor valamit mégis elrontottunk, mitöbb ezt észre sem vettük. Ez is tanulságos! Ha gépünk az "ok" felirattal visszatért a *BASIC-be*, akkor a

PRINT USR(9999)

parancs beírásával még megkísérelhetjük újra elérni a *MONITOR-t*. Rosszabb esetben a **RESET** gomb egyszeri megnyomása után próbáljuk meg ismét az előbbieket, s ha ezután is furcsa viselkedést tapasztalunk, akkor sajnos nem marad más, mint *újra betölteni a kazettáról a MONITOR-t, majd kezdeni elölről az egész munkát* -- de most már, ha lehet, figyelmesebben!

Amennyiben programunk már helyesen működik, írjuk át a 3008H című memóriarekeszben a *6BH*-t, a 300AH-ban és 300BH-ban pedig a *videomemória-címet* és kísérletezzünk kicsit!

Ha bármilyen hibát észlelünk, próbáljuk meg azt azonnal kideríteni és kijavítani!

Ha ezt megunta az Olvasó, máris menjünk tovább!

Mentsük át előbbi programunkat az *U3 RAM* elejére!

Az *U3 szegmens* csak a 3. lapra helyezhető, *0B0H lapozási értékkel*, tehát ott a *0C000...0FFFFH* címeken lesz elérhető. A lapozási eljárást már ismerjük:

```
LD    A,B0
LD    (0003),A
OUT   (02),A
```

Tetszőleges memóriaterületek másolására a CPU *LDIR és LDDR utasítása* szolgál. Az *LDIR* megadott kezdőcímtől a növekvő, az *LDDR* pedig a csökkenő memóriacímek felé haladva mozgatja át az adatokat -- a HL regiszterpár mutatta címről a DE regiszterpár mutatta címre. A mozgatás hosszát (a byte-ok számát) az utasítás végrehajtása előtt a BC regiszterpárba kell tölteni.

Az átmásolást elvégezve most is visszaállítjuk az eredeti *U0-U1-U2-SYS memórialapozást*.

Programunk most a következő:

PR2. Átmásolás az U3 RAM-ba

```
3020   LD   A,B0          3E,B0
3022   LD   (0003),A      32,03,00
3025   OUT  (02),A        D3,02
3027   LD   HL,3000       21,00,30
302A   LD   DE,C000       11,00,C0
302D   LD   BC,0014       01,14,00
3030   LDIR               ED,B0
3032   LD   A,70          3E,70
3034   LD   (0003),A      32,03,00
3037   OUT  (02),A        D3,02
3039   CALL MONITOR       CD,0F,27
```

A 3020--3025, a 3032--3037 és 3039 utasítások szerepét már ismerjük. A 3027--302D utasításokkal pedig az *LDIR működését készítik elő*. Először a HL regiszterpárba betöltjük az átmásolandó memóriaterület kezdőcímét (3027), majd a DE regiszterpárba a kezdő célcímet (ez 0C000H, hiszen *a ROM helyére most az U3 RAM-ot tettük*). Végül a BC regiszterpárba a másolandó byte-ok száma (14H) kerül (302D sor). Az *LDIR* utasítás hatására a CPU így a kívánt átmozgatást végzi el.

Írjuk be a programot a 3020H memóriacímtől kezdődően, majd ellenőrizzük úgy, ahogy azt az előző programnál tettük! Ha munkánk a disassemblált lista alapján is jónak bizonyul, akkor indítsuk el a programot:

J 3020

és **RETURN**. Ha jól dolgoztunk, szinte azonnal bejelentkezik a *MONITOR* és első programunk az *U3 RAM* elejére került.

Ezt közvetlenül ellenőrizni nem tudjuk, hiszen ezt a *szegmenst* már nem látja a mikroprocesszor.

Hogy mégis ellenőrizhessük, töröljük a 3000...3013H címről az első programot! Ezt legegyszerűbben az

```
F 3000 3013 00                          <RETURN>
```

paranccsal tehetjük meg. Ha ezután kiíratjuk ezt a memóriaterületet:

```
P 3000 3016
PRINTER? (Y)                            <RETURN>

3000  00 00 00 00 00 00 00 00
3008  00 00 00 00 00 00 00 00
3010  00 00 00 00 00 00 00
MONITOR
*
```

láthatjuk, hogy első programunk valóban törlődött.

Második programunk ellenőrzéseként másoljuk most az *U3 RAM*-ból vissza -- ugyanide -- az előbb átmásolt programot!

PR3. Visszatöltés az U3 RAM-ból

```
3050   LD   A,B0          3E,B0
3052   LD   (0003),A      32,03,00
3055   OUT  (02),A        D3,02
3057   LD   HL,C000       21,00,C0
305A   LD   DE,3000       11,00,30
305D   LD   BC,0014       01,14,00
3060   LDIR               ED,B0
3062   LD   A,70          3E,70
3064   LD   (0003),A      32,03,00
3067   OUT  (02),A        D3,02
3069   CALL MONITOR       CD,0F,27
```

A 3050--3055 címeken behozzuk a 3. lapra az *U3 RAM-ot*, majd a 3057--305D címeken előkészítjük a másolást az *U3* elejéről (C000H) az *U0*-ba (3000H) 14H hosszon, elvégezzük a másolást (3060), végül visszaállítjuk az eredeti memórialapozást és visszatérünk a *MONITOR-ba*.

Írjuk be ezt a programot a 3050H címtől, ellenőrizzük, a hibákat javítsuk ki, majd futassuk le a **J 3050** *paranccsal*!

Ha a **PR2.** és a **PR3.** beírása és futtatása valóban hibátlan volt, akkor a CPU az első programot visszamásolta a 3000H kezdőcímű területre.

Ezt ellenőrizzük ismét a **PR1.** kilistázásával:

```
Z 3000 3013
PRINTER? (Y)                    <RETURN>
```

PR4. Átmásolás az EXT-IOMEM szegmensből

```
3080    LD    A,F0          3E,F0
3082    LD    (0003),A      32,03,00
3085    OUT   (02),A        D3,02
3087    LD    HL,F772       21,72,F7
308A    LD    DE,3100       11,00,31
308D    LD    BC,0009       01,09,00
3080    LDIR                ED,B0
3082    LD    A,70          3E,70
3084    LD    (0003),A      32,03,00
3087    OUT   (02),A        D3,02
3089    CALL  MONITOR       CD,0F,27
```

Ez a program lényegileg nem tér el az előbbiektől, csak a paraméterek mások. A *OFOH lapozási érték* az *EXT-IOMEM szegmenst* teszi a 3. lapra, így a *SYS-beli ROM folytatása* a OF000...OFFFFH címekre kerül. Innen másoljuk át a OF772H kezdőcímű, 09 hosszúságú szubrutint az *UO RAM* 3100H címére. Erről a **PR4** beírása, ellenőrzése és lefuttatása után a 3100...3108H terület disassemblálásával győződhetünk meg:

```
Z 3100 3108
PRINTER? (Y)                    <RETURN>

3100 210B0D     LD   HL,0D0B
3103 7E         LD   A,(HL)
3104 B7         OR   A
3105 C8         RET  Z
3106 36EC       LD   (HL),EC
3108 C9         RET
```

Ez a szubrutin a *rendszer-ROM* kibővítésében a magnetofonkezelő programcsomag része.

4.2 Számolási gyakorlatok

A fejezet következő részében néhány egyszerű, ám nagyon is gyakori számolási példát beszélünk meg. Egyrészt azért, hogy folytassuk a programozási munkát, másrészt, hogy némi mintát adjunk az ilyen természetű feladatok megoldására, de nem utolsó sorban: *fejlesztjük a MONITOR kezelésében eddig megszerzett ismereinket* is.

Nagyon fontos, hogy végül *a MONITOR összes parancsát* rutinosan, szinte gondolkodás nélkül és *mindig a legcélszerűbben tudjuk használni*. Ez a feltétele annak, hogy a figyelmünket a későbbiekben már ne a *MONITOR, hanem a programozási munka* kösse le!

A *MONITOR* legyen betöltve a számítógépbe és kezdjük az alapműveletekkel.

PR5. Nyolcbites összeadás: A = H + L

```
3000    LD    HL,0203       21,03,02
3003    LD    A,H           7C
3004    ADD   A,L           85
3005    CALL  MONITOR       CD,0F,27
```

Írjuk be ezt a programot (a 3000H kezdőcímtől), ehhez most is a *MONITOR M parancsát* használjuk:

```
M 3000 00 21                    <RETURN>
3001 00 03                      <RETURN>
3002 00 02                      <RETURN>
3003 00 7C                      <RETURN>
3004 00 85                      <RETURN>
3005 00 CD                      <RETURN>
3006 00 0F                      <RETURN>
3007 00 27                        <X>
MONITOR
X
```

és ellenőrizzük a disassemblált listán (ezt soha ne mulasszuk el, a legtöbb gépelési hiba, egy-egy kód kihagyása stb azonnal kiderül):

```
Z 3000 3005
PRINTER? (Y)                          <RETURN>
3000 210302    LD    HL,0203
3003 7C        LD    A,H
3004 85        ADD   A,L
3005 CD0F27    CALL  270F
MONITOR
X
```

Mint látjuk, az első sorban töltjük be a *H és az L regiszterekbe* a két számot. A 03 és 02 helyett természetesen bármi mást is írhatunk. Ha pl. 15H-hoz akarunk 8BH-t hozzáadni, akkor ehhez az első sort kell csak módosítani, így:

```
M 3001 03 15                          <RETURN>
3002 02 7C                            <X>
MONITOR
X
```

Mivel a nyolcbites összeadás csak az *A akkumulátorban végezhető el*, ezért először a H-t átmásoljuk A-ba (2. sor). Az összeadást a CPU a 3. sorban (a 3004H címen) található ***85H kód*** végrehajtásakor intézi el: az L regiszter tartalmát hozzáadja A-hoz.

Az utolsó sor: visszatérés a *MONITOR*-ba.

Gyakorlásul kövessük nyomon lépésenként a program végrehajtását!

Ha azt akarjuk, hogy a CPU csak az első sort hajtsa végre, akkor utána -- tehát a 3003H memóriacímre -- *töréspontot* kell elhelyezni. Írjuk be:

B 3003

Ezután indítsuk el a programot:

J 3000 <RETURN>

Amikor a *MONITOR* ismét bejelentkezik, nézzük meg a CPU *regisztereit*:

R

Figyeljük meg, hogy az első utasítás végrehajtása eredményeként HL = 0203, az összeadandó számok tehát a H és az L regiszterekben vannak! A *PC értéke 3003H*, vagyis a mikroprocesszor valóban *csak az első utasítást* hajtotta végre.

Továbblépni úgy tudunk, hogy először is a programunkból kivesszük az előbbi töréspontot. Ehhez elég lenyomni a **K** billentyűt:

Mint látjuk, tájékoztatásul -- ha közben elfelejtettük volna -- a *MONITOR kiírja a megszüntetett töréspont helyét.*

Egyetlen utasítással akarunk csak továbbmenni, ezért ismét *töréspontot* helyezünk el a 2. sor után, tehát a 3004H memóriacímre:

B 3004

Mivel az első utasítást már végrehajtotta a CPU, most *a 2. sorra adjuk a vezérlést:*

J 3003 <RETURN>

Ennek végrehajtása után nézzük meg ismét a regisztereket (*R parancs*)! Láthatjuk, hogy az A-ba valóban bemásolódott a H regiszter tartalma: A = 03 (a mellette levő F regiszterrel egyelőre nem kell foglalkoznunk).

Ismét továbblépünk. Kivesszük az előbbi töréspontot (*K parancs*), majd a **B 3005** *paranccsal* ismét elhelyezzük a következő végrehajtandó utasítást követő címre. Végül a **J 3004** (és **RETURN**) *paranccsal* végrehajtjuk. Írassuk ki ismét a regisztereket (*R parancs*) és ellenőrizzük, hogy az A regiszterben megtörtént az összeadás: A = 05. Végül szüntessük meg a *töréspontot* (K).

A mikroprocesszorról szóló alfejezetben már megbeszéltük az *F regiszter* szerepét. Tanulmányozzuk, hogy az előbbi művelet után (02 + 03 = 05) mit mutatnak a jelzőbitek. Nézzük meg: most F = 00.

Nézzük bitenként:

F	b7	b6	b5	b4	b3	b2	b1	b0
Jelzőbit	S	Z	-	H	-	P/V	N	C
Érték	0	0	0	0	0	0	0	0

Jelentés (balról jobbra):

S az eredmény pozitív (előjel + 7 bites értékként felfogva);
Z az eredmény nem 0;
H az A alsó négy bitjén nem keletkezett átvitel;
P/V előjeles túlcsordulás nincs, az A-beli eredmény előjeles számként is helyes;
N a művelet összeadás volt;
C nincs átvitel.

A CPU minden művelet után szorgalmasan beállítgatja ezeket a biteket.

A későbbiekben a *feltételes vezérlésátadást* jelentő utasításokat éppen erre alapozva tudjuk megszervezni!

Az első sorban a számokat változtatva gyakoroljuk az előbbieket!

Legyen az összeadandó két szám pl. 15H és 7CH! Futtassuk le egyszerre az egész programot (**J 3000**) és nézzük a regisztereket (**R**)!

Most az eredmény: A = 91H, a jelzőbitek pedig: *F = 94H*. Az összeadás: 7CH + 15H = 91H természetesen helyes, vizsgáljuk meg viszont ismét az *F bitjeit*:

F	b7	b6	b5	b4	b3	b2	b1	b0	
	S	Z	-	H	-	P/V	N	C	
	1	0	0	1	0	1	0	0	*(= 94H)*

S = 1 jelzi, hogy az eredmény negatív számot jelent, ill. jelentene, ha *"előjel + 7 bites érték"* alakú számként értelmeznénk;

P/V = 1 túlcsordulást jelez: az eredmény nem fért el 7 biten (b6...b0);

H = 1 azt mutatja, hogy az alsó két hexadecimális jegy (5 és C) összeadása nem volt lehetséges tisztán ezen a helyi értéken. Vagy másképpen: az alsó 4 biten (b3...b0) *átvitel* keletkezett a következő, b4 bitre:

```
          15
      +   7C
      ------
          91
átvitel: +1 ┘
```

Javaslom, hogy az Olvasó végezzen önállóan különböző összeadásokat és minden alkalommal végezze el -- hasznos gyakorlásul -- az előbbi elemzést!

Változtassuk meg ezután a programot úgy, hogy a CPU összeadás helyett *kivonást* végezzen! Ehhez mindössze a 3. sorban kell *ADD A,L* helyett *SUB L* utasítást írni. Ennek *kódja 95H*, tehát:

```
M 3004 85 95                          <X>
MONITOR
X
```

Disassemblálással ellenőrizzük a módosítást (a *Z parancsal*), majd a H és L regiszterekbe az első sorban beírható számokat változtatva, önállóan gyakoroljuk tovább a ***MONITOR*** kezelését, a hexadecimális számolást és az *F regiszter bitjeinek* értelmezését!

A következő program mintát mutat arra, hogyan lehet az A regiszter tartalmát pl. 7-tel megszorozni.

PR6. Szorzás: A = A * 7

```
3000   LD   A,1C     3E,1C      Egy szorzandó szám
3002   LD   B,A      47         B-be is
3003   ADD  A,A      87         *2    (A+A=2*A)
3004   ADD  A,A      87         *4  (az előbbi 2-szerese)
3005   ADD  A,A      87         *8  (az előbbi 2-szerese)
3006   SUB  B        90         *7
3008   CALL MON      CD,0F,27
```

A listában az egyes sorokhoz odaírtuk *a működés lényegét kifejező megjegyzéseket (**komment**)*, a sorok elején álló sorszám pedig egyben a beíráshoz javasolt memóriacím.

A program működésének alapja az, hogy ha egy szám 8-szorosából kivonjuk az eredeti számot, akkor annak 7-szeresét kapjuk. Írjuk be ezt a programot és ellenőrizzük a működését! Ennek mintájára elvileg tetszőleges szorzások programját is megírhatjuk. Ügyeljünk azonban arra, hogy *az A regiszterben kapott eredmény csak addig lesz helyes, amíg 8 biten elfér*, efölött éppen a legértékesebb bitek vesznek el!

Ezután jóval komolyabb munkába kezdünk.

PR7. Szorzás: HL = B * C

```
3000 012C33     LD   BC,332C   A két szám, pl 33H és 2CH
3003 AF         XOR  A         A=00
3004 67         LD   H,A       H=00
3005 B0         OR   B         Ha C-t 0-val kell szoroz-
3006 2807       JR   Z,300F    ni, ugrás a végére: az
                               eredmény: HL=0000 lesz!
3008 AF         XOR  A         A=00, kezdőérték
3009 81         ADD  A,C       + 1*C
300A 3001       JR   NC,300D   Ha még elfért A-ban: ug-
                               rás előre, egyébként
300C 24         INC  H         H-t is növeljük 1-gyel
300D 10FA       DJNZ 3009      Számláljuk az összeadáso-
300F 6F         LD   L,A       kat, ha kész, az eredmény
                               alsó byte-ját L-be írjuk
3010 CD0F27     CALL MONITOR   és kész.
```

A tényleges szorzó rutin a 3003--300F címekhez tartozó 13 byte-on valósul meg. Itt már mindkét összeszorzandó szám tetszőlegesen megadható nyolcbites érték lehet, s az eredmény is -- amit a HL regiszterpárban kapunk, tehát 16 biten -- mindig helyes.

A szorzást most is *ismételt összeadásokkal* valósítjuk meg. Közben azonban figyeljük az A-beli nyolcbites *túlcsordulásokat, s ezeket mint átviteleket* H-ban számláljuk (egyszerű átvitelek a második hexadecimális helyi értékről a harmadikra).

A listában szerepeltetett 33H és 2CH esetén pl. az A regiszterhez mindig 2CH-t adunk. A hatodik hozzáadás után az A-ban ODCH lesz, ezért a következő műveletben

```
      DC
+     2C
  ------
      08
átvitel +1 ┘
```

Az A-ban keletkezett 08 eredmény önmagában természetesen nem helyes, de a második helyi értéken bekövetkezett túlcsordulást a CPU carry jelzőbitje egyértelműen jelzi. Minden átvitel a harmadik helyi értékre 256-ot jelent, s ha ilyenkor a H regisztert 1-gyel megnöveljük, akkor ezzel végülis azt számláljuk, hogy az A-beli értékhez *hányszor kell 256-ot hozzáadni.*

Példánkban a H korábbi értéke 00 volt, így ha az átvitel után 1-gyel növeljük, H értéke 01 lesz, ami az A-ban levő 08-cal együtt már a helyes részeredményt mutatja:

```
      DC
+     2C
  ------
    0108
```

Az eljárás pontos összhangban van azzal, amit a nyolcbites összeadásról és a többjegyű hexadecimális számok helyi értékéről az 1.2 alfejezetben tanultunk.

A 3003 és 3008 címen ***XOR A*** *nullázza az akkumulátort.* Gondoljuk meg ui., hogy az XOR műveletet a CPU bitenként végzi, s *egyforma értékű bitek esetén az* ***XOR*** *művelet eredménye:* ***0.*** Márpedig -- az A önmagával való művelete miatt -- minden bitnél ez a helyzet.

A 3004 címen így a H regiszterben is 00 lesz, amely az összeadások közben keletkező átvitelek során egyesével nőhet (kis számok szorzásánál nem biztos, hogy lesz akár egyetlen átvitel is).

A 3005 címen szereplő ***OR B*** hatására -- mivel A = 00 -- a B regiszter tartalma bitenként átmásolódik A-ba! Ugyanis amelyik bit a B-ben 0, az OR művelet után ugyanez a bit az A-ban is 0 marad, míg ahol a B-ben 1 értékű bit van, az OR után az A megfelelő bitje is 1 lesz.

Ezt a műveletet azért kellett beírni a programunkba, hogy hatására *beálljanak a jelzőbitek*! Érdemes ui. külön megvizsgálnunk, hogy a B-ben megadott szám 00-e. Mert ha igen, akkor az eredmény is 00 lesz, tehát nem is kell

szorozni. De azért is fontos ezzel foglalkozni, mert B = 00 esetén a 300D cím *DJNZ utasításban* a CPU először csökkenti B-t és csak ezután vizsgálja meg, hogy 00 lett-e már -- így az összeadás 256-szor futna le, pedig B = 00 esetén egyszer sem kellett volna.

Ha B = 00, akkor az OR B hatására az A továbbra is 00 marad, ám ezt -- mint müveleti eredményt -- az F regiszter Z bitje jelzi, aminek alapján külön intézkedni lehet.

A 3006 cím *JR Z,300F* utasítása éppen ezt teszi. Jelentése: *ha Z = 1, akkor ugrás a 300F címre!* Azaz: a CPU kihagyja a 3008--300D című utasításokat, s csak a program végét hajtja végre. Az A-t átmásolja az L-be (tehát most 00-t) és visszatér a *MONITOR*-ba. Mivel B = 00 volt az összeszeszorzandó egyik szám, ilyenkor a HL = 0000 eredmény a helyes!

Ha B értéke nem 00, akkor az OR B után az A sem lesz az, így a Z jelzőbit sem lesz 1. Ekkor -- mivel *a feltétel nem teljesül -- a JR Z hatástalan*, a CPU a következő, 3008 utasítással *folytatja* a működést.

A 3008 címen az A regisztert ismét nullázzuk, előkészítjük az összeadáshoz.

A 3009 címen az A-hoz hozzáadjuk C-t. Ezt az összeadást annyiszor fogjuk elvégezni, amennyi a B értéke. Az A-ban tehát egymás után 1 * C, 2 * C, 3 * C,..., végül pedig B * C lesz. Ha bármikor 255 fölé kerülünk, akkor az A-ban természetszerüleg csak az eredmény alsó nyolcbites része marad, az átvitelt az F regiszter CY bitje jelzi.

Ekkor kell megnövelni H-t, ami így éppen a 3. és 4. hexadecimális jegyet tárolja.

Ehhez a döntéshez szolgál a 300A cím *JR NC,300D* utasítása. Ha *nincs carry, tehát CY = 0* -- vagyis az eredmény még elfért az A-ban --, *akkor a CPU a 300DH címre ugrik*, átlépve az INC H utasítást. Ha viszont az utolsó összeadásnál átvitel keletkezett, akkor *CY = 1 lesz, az NC feltétel nem teljesül: az ugrási utasítás hatástalan.* Ebben az esetben a CPU folytatja tovább a következő utasítással: H-t növeli 1-gyel.

A 300D cím *DJNZ 3009* utasítását az 1.3.4.8 szakaszban már megmagyaráztuk. A CPU B-t 1-gyelcsökkenti, s ha még nem lett 00, visszaugrik a 3009H cím-re: a program újabb összeadással folytatódik. Ezáltal annyi összeadást hajt végre a mikroprocesszor -- ciklikusan ismételve a 3009--300D címeken szereplő utasításokat -- amennyi a B regiszter eredeti értéke volt.

Befejezésül az A utolsó értékét átmásoljuk az L regiszterbe, beállítva így HL-ben a helyes végeredményt.

Figyeljük meg a feltételes relatív ugróutasításokban az *ugrás távolságát* megadó értékeket!

A 3006 címen *a JR Z,300F kódjai: 28, 07.* A *28H* kód a CPU számára a *JR Z* feltételes utasítást *azonosítja*, a *07* pedig az ugrás *távolsága*.

Ennek helyes kiszámítása programjainkban rendkívül fontos. A CPU eredetileg a *JR Z* utasítás 2. byte-ja utáni memóriarekesszel folytatná az utasítások beolvasását. Azt kell megállapítanunk, hogy *ehhez a címhez mennyit kell adnunk ahhoz, hogy a tervezett utasítás kezdő memóriacímét kapjuk.* Egyszerűbb esetekben ez akár közönséges számlálással is történhet. Ehhez persze tudnunk kell, hogy melyik utasítás -- hány byte-os. Ennek alapján a *JR Z* után álló utasítástól indulva, egyesével megszámoljuk, hogy hány byte-tal kell előremenni a kívánt cím eléréséhez. Ez a szám jelen esetben: 7.

A másik ilyen utasítás a *JR NC,300D.* Most a CPU *PC* regisztere az *INC H*-ra mutat, s mivel ez egybyte-os utasítás, innen csak egyet kell előrelépni ahhoz, hogy a kívánt *DJNZ*-hez érjünk. Ezért a helyes *kódok: 30,01.*

A *DJNZ 3009* utasítás kicsit bonyolultabb, mert itt visszafelé kell lépni. Végrehajtáskor a CPU programszámláló regisztere a *DJNZ* utáni utasításra: *LD L,A*-ra mutat. Innen kell haladnunk visszafelé az *ADD A,C*-ig. Számoljuk meg, hogy hány byte-tal kell visszalépni: 06. Az ellentétes irányt *a megfelelő egybyte-os negatív (!) számmal kell megadnunk*, amit legegyszerűbben úgy kapunk meg, hogy a szóbanforgó távolságot *256-ból kivonjuk.* Példánkban így 250-et kapunk, amely hexadecimálisan *0FAH*, tehát ezt kell a *DJNZ kódja (10H)* után írni.

Nagyobb távolságok esetén jobb, ha a programba beírandó értéket *kivonással számoljuk.* A program begépelésekor a beírandó érték helyét hagyjuk ki (helyére pl. írjunk 00-t), s ha a teljes program kész, akkor már az ismert memóriacímek alapján a helyes érték egyszerűen kiszámítható:

300F	300D	3009
- 3008	- 300C	- 300F
0007	0001	FFFA

(a kezdő FF érdektelen)

Remélem ez a program meggyőzte az Olvasót arról, hogy mennyire fontos a programozói munkában az *F regiszter* vizsgálata és a feltételes utasítások használata. Az előbbi elemzést érdemes többször is átgondolni, sőt mindaddig

ne menjünk tovább, amíg e program működése minden részletében tisztává nem válik! Ezt az utasításssorozatot itt és most feltétlenül meg kell érteni: ez az értelmes továbbhaladás feltétele!

A *MONITOR M parancsával* írjuk be a programot (pl. a javasolt 3000H címtől), a *Z paranccsal* kilistázva ellenőrizzük, majd az első sorban a B és a C regiszterekbe különböző értékeket töltve próbáljuk ki!

Befejezésül két osztást végző programot mutatunk be. Mindkettő ún. *maradékos osztást* végez, az első nyolcbites adatokkal, a második pedig 16 biten.

PR8. Osztás: B = B : C és A-ban a maradék

```
3040 010415     LD   BC,1504   B-be az osztandót, C-be az
                               osztót töltjük (15H és 04)
3043 78         LD   A,B       Az osztandó A-ban
3044 0600       LD   B,00      B-ben lesz az eredmény

3046 91         SUB  C         Ismételt kivonásokat vég-
3047 04         INC  B         zünk, s a kivonások számát
3048 30FC       JR   NC,3046   számláljuk (B-ben), amíg az
                               eredeti osztandó elfogy
304A 05         DEC  B         Ha már túl sok kivonás
304B 81         ADD  A,C       volt: korrekció
304C CD0F27     CALL MONITOR
```

PR9. Osztás: BC = HL : DE és HL-ben a maradék

```
3050 212873     LD   HL,7328   HL=osztandó
3053 116D00     LD   DE,006D   DE=osztó
3056 010000     LD   BC,0000   BC-ben lesz az eredmény
3059 B7         OR   A         CY=0
305A ED52       SBC  HL,DE     Ettől kezdve a rutin elvi-
305C 03         INC  BC        leg ugyanúgy működik, mint
305D 30FB       JR   NC,305A   a PR8, csak 16 biten
305F 0B         DEC  BC
3060 19         ADD  HL,DE
3061 CD0F27     CALL MONITOR
```

Írjuk be a programokat a 3040H és a 3050H címtől, majd az első sorokba különböző számokat írva, próbáljuk ki!

A második programban csak az *OR A* szerepéről kell külön beszélnünk. Hatására *CY = 0* lesz, s itt ez a lényeg. A kivonásokat végző *SBC HL,DE* utasítás ui. a HL értékéből

DE mellett még a *CY jelzőbitet* is kivonja. Ezért a helyes eredményhez szükséges, hogy a kivonás előtt *CY = 0* legyen!

Az előző programokhoz írt bőséges megjegyzések után ezekhez már nem kell semmi egyebet hozzáfűzni.

A fejezet ezen részének lezárásaként még egyszer megjegyezzük, hogy a közölt néhány példával csak ötleteket akartunk adni az ilyen típusu feladatok egyszerű megoldásához. Ha az Olvasó a programok futtatása során a kapott eredményeket *10-es számrendszerben* is ellenőrizni akarja, ehhez gyors segítségül hívhatja a *MONITOR N parancsát.*

4.3 Videomemória rutinok

Könyvünk programozási szempontból talán legérdekesebb és egyben legösszetettebb része előtt állunk.

Túljutva a fejezet eddigi részein, nagyobb tempóban is haladhatunk tovább.

Fontos azonban, hogy ha jelen pillanatban az Olvasó még bizonyos technikai nehézségekkel küzd -- akár a *MONITOR parancsainak* használatát, akár az eddigi példaprogramok értelmezését illetően -- akkor mielőtt továbbmenne, feltétlenül nézzen utána a szükséges részleteknek! Bár ezt később is megteheti, mégis szerencsésebb, ha ezután *minden figyelmünket csakis a programozásra tudjuk fordítani.*

A fejezet itt következő részében a *videomemóriával* foglalkozunk.

A TV-Computer három különböző tv-képmegjelenítési üzemmódban dolgozhat (2, 4 és 16 színű mód). Mivel minden esetben *ugyanazt a videomemóriát* használjuk, a három üzemmódban ugyanazt a memóriatartalmat *különbözőképpen kell értelmezni.*

Ennek illusztrálására nézzünk először egy egyszerű programot!

PR10. Változatlan videomemória-tartalom megjelenítése a háromféle üzemmódban

Először a képernyő középső sorát írjuk tele pl. a nevünk betűivel, a többi sor maradhat üres. Legegyszerűbb a *02-es funkcióhívás* segítségével ezt megtenni, amelyhez az

RST 30 : 02

utasítást kell használni. Ehhez előbb a betűk *ASCII kódjait* el kell helyezni valahol a memóriában. A *MONITOR parancsa* lehetővé teszi, hogy a kódok keresgélése és beírása helyett közvetlenül a karakterek nyomógombjait használjuk. Helyezzük el nevünk betűit a memória 3100H kezdőcímű területén!

Ha a *MONITOR* számítógépünkben parancsra készen rendelkezésre áll, akkor gépeljük be:

$ 3100

majd ugyanúgy, mint amikor a képernyőre *BASIC-ben* írunk, nyomkodjuk le egymás után a nevünkhöz tartozó billentyűket:

```
$ 3100 ? L
3101 ? u
3102 ? d
3103 ? á
3104 ? n
3105 ? y
3106 ? i
3107 ?              <szóköz>
3108 ? L
3109 ? á
310A ? s
310B ? z
310C ? l
310D ? ó
310E ?              <szóköz>
310F ?              <szóköz>
.    .    .
:    :    :
```

Ismételjük ezt a nevünk betűivel mindaddig, amíg a 3120H címig el nem érünk. Ekkor nyomjuk le az **ESC** billentyűt, amivel megszakítjuk a ***szövegbetöltő*** *parancsot.*

A képernyő középső sorának kijelöléséhez a *03 funkciókódot* használhatjuk.

Kezdjük el beírni a programot a 3000H címtől:

```
3000 F705      RST  30: 05      Képernyőtörlés
3002 010C01    LD   BC,010C     A kiírási pozíció beál-
3005 F703      RST  30: 03      lítása: 12. sor, 1. osz-
                                lop
3007 110031    LD   DE,3100     A szöveg kezdőcíme
300A 012000    LD   BC,0020     A szöveg hossza
300D F702      RST  30: 02      A szöveg kiírása
```

Rövidesen folytatjuk, egyelőre azonban -- hogy az eddigieket kipróbálhassuk -- helyezzünk el a végére egy *töréspontot*:

B 3000F

A *Z parancsal* listázzuk ki, s ha eddig hibátlan, futtassuk le: **J 3000.**

Ha jól dolgoztunk, akkor a képernyő közepére kiíródott (néhányszor) a nevünk.

Ebben most csak azt kell látnunk, hogy *a videomemóriába bekerültek bizonyos adatok*. Mégpedig pontosan azok, amelyek a nevünk betűit kirajzoló pontokhoz szükségesek voltak. Ezt az -- egyébként meglehetősen bonyolult -- munkát a programunkban csak *segítségül hívott, ROM-ban tárolt szubrutinok* végezték el, a 3005H és 300AH címre írt *funkcióhívások* hatására.

Az egész *ROM* hatalmas rutingyűjteményként bármikor és bárkinek a rendelkezésére áll -- csak azt kell tudni, hogy melyik, hol található. Felhasználásukkal temérdek plusz munkától kímélhetjük meg magunkat! Jelentős segítséget kapunk ehhez **A TV-Computer ROM programja** c. könyvből. Példánkban a két *ROM-rutin* aktiválása a *funkcióhívásokkal* megoldható volt -- ez azonban nincs mindig így.

Továbbmegyünk!

Célunk az, hogy a videomemóriába az előbbi módon bejuttatott adatokat számítógépünk felváltva a 2-es, a 4-es, ill. a 16-os grafikus üzemmódban jelenítse meg.

Az üzemmódot a *06-os port alsó két bitjébe* írt adattal tudjuk megváltoztatni: 00 bitérték 2-es, 01 bitbeállítás 4-es, az 10 vagy 11 bitértékek pedig 16-os üzemmódot eredményeznek. Az ezeknek megfelelő *00, 01 és 02 vagy 03* értékű byte-okat azonban nem írhatjuk közvetlenül a *06-os portra, mert a többi bit más funkciókat lát el, azokat nem szabad megváltoztatni*. Mit lehet tenni?

Ehhez -- és minden hasonló probléma megoldásához -- használhatók a többfunkciós portok *másolatai*. A *06-os port* esetében a *0B13H* című memóriarekeszből olvashatjuk ki azt az adatot, amit erre a *portra* utoljára küldtünk. Ennek természetesen elengedhetetlen feltétele, hogy valahányszor ezt az értéket megváltoztatjuk, a *portra* való kiküldéssel egyidőben erre a címre is írjuk be az új adatot. Ez a tisztességesség, a további korrekt munka feltétele.

A minket kizárólag érdeklő *két bit átírása a portmásolat alapján* a következőképpen lehetséges:

```
300F 3A130B     LD   A,(0B13)     A-ba beolvassuk a 06-os
                                  portra küldött régi ér-
3012 E6FC       AND  FC           téket, majd az alsó két
                                  bitet nullázzuk
3014 D306       OUT  (06),A       A 2-es mód kapcsolása
```

A 3012 címen az A és a OFCH szám közötti *AND művelettel* beállítottuk a 2-es grafikus módhoz szükséges *00 állapotot az alsó két biten*. Ugyanis OFCH az 1111 1100 bitállapotot jelenti, így a felső hat bit az *A*-ban változatlan marad (l. az AND művelet táblázatát az 1.3.4.3 szakaszban), *az alsó két bit pedig feltételenül 0 lesz*, bármi is

volt korábban. Az A-ban így kapott értéket a *veremben* tároljuk. A további utasítások ui. átírják az akkumulátort, pedig értékére nekünk még szükségünk lesz:

```
3016 F5          PUSH AF
```

A 2-es grafikus módra való átkapcsolás után célszerű, ha egy billentyű lenyomásáig megállítjuk programunkat. Ezt legegyszerűbben a

```
3017 F791        RST  30: 91        Várakozás egy billentyű
                                    lenyomására
```

utasítással tehetjük meg.

Eközben ui. megszemlélhetjük a 2-es üzemmódra való átváltás hatását. Amikor akarjuk, lenyomhatunk egy billentyűt (bármelyiket), s ekkor a *91H-s funkciókódhoz* tartozó *ROM*-rutin a CPU C regiszterében visszaadja a lenyomott billentyű *ASCII* kódját, A-ban 00-t, ami *a funkcióhívás hibátlan* végrehajtását jelzi. A program csak ezután megy tovább.

Kapcsoljunk ismét 4-es üzemmódra. Az A korábbi értékét a *veremben* tároltuk, így nem kell ismét a 0B13H-s portmásolattal bajlódnunk, elég a *veremben* tárolt értéket visszatölteni A-ba:

```
3019 F1          POP  AF
301A F601        OR   01            A 0. bit beállítása
301C D306        OUT  (06),A
301E F5          PUSH AF
```

A 301AH címre írt *OR 01 művelettel* az A regiszterben korábban 0 értékű 0. bit 1 lesz, így ezt küldve a *06-os portra*, a 4-es üzemmódot állítjuk be. Az A-t ismét a *verembe* tesszük.

Megjegyezzük, hogy az *OR 01* művelet A értékét mindössze 1-gyel növelte, amit az egyszerűbb *INC A* utasítással is elérhettünk volna -- a funkció szempontjából azonban az *OR 01* eljárás szemléletesebb, sőt általánosabb esetben csak ez használható.

A 4-es üzemmódnál ismét megállítjuk a programot:

```
301F F791        RST  30: 91        Várakozás egy billentyű
                                    lenyomására
```

Ezután beállítjuk a 16-os grafikus üzemmódot:

```
3021 F1          POP  AF            Az A korábbi értéke
```

```
3022 F602      OR   02          Az alsó két bit: 11
3024 D306      OUT  (06),A      16-os üzemmód
3026 F5        PUSH AF
3027 F791      RST  30: 91      Várakozás egy billentyű
                                lenyomására
```

A 16-os grafikus üzemmódba való átkapcsolás hatását most is tetszőleges billentyű lenyomásáig tanulmányozhatjuk.

A program további részét célszerű lesz úgy megszervezni, hogy kilépést biztosítson a *MONITOR-ba*, azonban tegye lehetővé számunkra az előbbiek ismétlését is. Kihasználhatjuk, hogy a *91H kódú funkcióhívás a billentyű lenyomása után a C regiszterben visszaadja annak ASCII kódját*. A *MONITOR*-ba való visszatérést a **RETURN** billentyű lenyomására szervezzük meg, minden más esetben a program 3012 címére visszaugorva folytatható legyen az üzemmódok vizsgálata.

```
3029 79        LD   A,C         A-ban a lenyomott bil-
                                lentyű kódja
302A FE0D      CP   0D          összehasonlítás ODH-val,
                                ez a RETURN kódja
302C C1        POP  BC          Az A régi értéke B-ben
302D 78        LD   A,B
302E 20E2      JR   NZ,3012     Ha nem RETURN volt: ug-
                                rás vissza, egyébként a
3030 E6FD      AND  FD          4-es mód beállítása
3032 D306      OUT  (06),A
3034 32130B    LD   (0B13),A    A 06-os portmásolat be-
3037 CD0F27    CALL 270F        írása és visszatérés
```

A 302CH címen rendkívül fontos a *POP BC* utasítás. Egyrészt a helyes működéshez szükséges, hogy egy program a *verembe* írt adatokat befejezés előtt onnan távolítsa is el (l. 1.3.4.6 szakasz), másrészt mind a tovább folytatott vizsgálatokhoz, mind pedig a befejezéshez szükség is van az A régi értékére. Tekintettel azonban arra, hogy a 302A cím összehasonlító utasítása az eredményt az *F regiszterben* adja, az A régi értékét nem szabad közvetlenül egy *POP AF* utasítással visszatölteni, mert ezzel az *F-et* is átírnánk!

Ezért a *POP BC, majd LD A,B* eljárással az A régi értékét először a B regiszterbe töltjük vissza, ahonnan már átmásolható az A-ba, míg a jelzőbitek régi értéke a veremből C-be kerül, ami számunkra most érdektelen. Így a beállított aktuális jelzőbitek megmaradnak.

Ha a lenyomott billentyű kódja ODH volt -- tehát a **RETURN** gombot nyomtuk le --, ezt a ODH-val való összehasonlítás után az *F regiszterben Z = 1* jelzi. Más billen-

tyű lenyomásakor a visszaadott kód nem lesz egyenlő ODH-val, így *Z = 0* marad.

Ennek alapján a 302EH címen szereplő, visszaugrást előíró utasítás akkor lesz hatásos, ha *nem* a **RETURN** billentyűt nyomjuk le! Ilyenkor a program a 3012H címtől fut ismét. **RETURN** után -- *Z = 1* miatt -- az *NZ feltétel nem teljesül*, a CPU a következő sorral folytatja a végrehajtást.

Az *AND FD* művelet az A regiszter bi bitjét 0-ra írja át, így az alsó két bit a 16-os üzemmódhoz beállított 11-ről 01-re változik, ami a *06-os portra* küldve megoldja a 4-es üzemmód visszatérés előtti beállítását.

A 3034 cím utasítása csak a forma kedvéért került a program végére, hiszen, mint láttuk, a *06-os portra* végül ugyanazt az értéket írtuk, ami a programunk futása előtt is volt.

Ha még nem történt meg, akkor most írjuk be a 300FH címtől folytatódó programrészt, s ha készen vagyunk, ellenőrizzük az egészet disassemblált listán:

```
Z 3000 3039
PRINTER? (Y)                    <RETURN>
```

Ha a munkánk hibátlannak bizonyul, akkor indítsuk el a programot a **J 3000** *MONITOR paranccsal*!

Kiíródik a nevünk, rögtön ezután a program átváltja a számítógépet 2-es üzemmódra, tehát amit látunk, az a nevünkhöz a 4-es üzemmódban tartozó videomemória-tartalom 2-es üzemmódbeli képe.

Nyomjuk meg pl. a szóközbillentyűt: visszatértünk az eredeti 4-es grafikus módba.

Ismét szóköz: amit most látunk, az ugyanannak a videomemória-tartalomnak a 16-os üzemmódbeli értelmezése.

Nyomkodjuk tovább a szóközbillentyűt, s figyeljük meg a három üzemmódban a karakterek változását! Akár lenyomva is tarthatjuk: akkor az üzemmódok váltogatása gyorsan ismétlődik.

Ha meguntuk, a 16-os üzemmódú állapotban nyomjuk le a **RETURN** billentyűt: 4-es üzemmódban visszatérünk a *MONITOR*-ba.

Programunk bizonyára értékes tapasztalatokhoz juttatta az Olvasót, amelyekből most a legfontosabb a konkrét *videomemória-tartalom* és a háromféle grafikus mód kapcsolata. Emellett maga a program, a portmásolat kezelése, a billentyűzet vizsgálata és a feltételes utasítások használata.

A képmegjelenítésről tisztázott részletek alapján

alapján (2.2 alfejezet) érthető, hogy ugyanaz a *videomemória-tartalom* miért jelenik meg másképpen az egyes üzemmódokban.

A gépi kódú programozás során gyakran kell közvetlenül is alkalmazni a *videomemória* megjelenítéséről szerzett ismereteinket. Ilyenkor a 2. memórialapra be kell hozni a *VID szegmenst*.

PR11. VID-ON Szubrutin a videomemória belapozására

```
1.  LD    A,50          3E,50
2.  LD    (0003),A      32,03,00
3.  OUT   (02),A        D3,02
4.  RET                 C9
```

Az itt szereplő műveletek értelmét már ismerjük. A végére nem a *CALL MONITOR utasítást* írtuk, ezt a programot csak *szubrutinként* fogjuk használni olyan programokban, amelyeknek közvetlenül a *videomemóriában* kell dolgozni. A lezáró *RET utasítás* hatására a CPU visszatér majd a hívó programhoz (l. 1.3.4.6 szakasz).

Írjuk be ezt a szubrutint a 4000H címtől. Ekkor a *RET kódja (C9H)* a 4007H memóriarekeszbe kerül.

Bizonyára az Olvasónak is feltűnt, hogy a sorok elején nem a beíráshoz javasolt memóriacímet, hanem egyszerű sorszámot adtunk meg. Ezzel programozási gyakorlataink folyamatában logikailag ismét továbbléptünk.

Egyrészt maga a szubrutin is teljesen *címfüggetlen:* a memóriában bárhol elhelyezve működőképes, csak az utasítások sorozata a fontos.

Másrészt -- összetettebb programok fejlesztésekor különösen -- az egyes *programrészek funkcióinak megtervezése* az elsődleges szempont, s hogy ezt milyen utasításokkal lehet megvalósítani.

A programozásnak ebben az első fázisában a meghatározott feladatokat ellátó programrészeknek -- némi fantáziával a funkcióra utaló -- nevet adunk, s ezzel hivatkozunk rájuk, amikor kell. Így jártunk el eddig is a *CALL MONITOR utasítás* esetén, ez sokkal kifejezőbb és szemléletesebb, mint a *CALL 270FH* lett volna. De még a program kiemelt jelentőségű sorait (pl. ugrási címeket) is csak egy-egy rövid elnevezéseel -- ún. *címkével* -- különböztetjük meg, lehetővé téve az ezekre való egyértelmű hivatkozást. Munkánk így áttekinthetőbbé, jobban követhetővé válik.

Programjaink memóriakiosztásának pontos részletei -- *beültetése* -- csak a munka következő fázisában válik fontossá. Bár a kettő nem egészen független -- már a programozási feladat első megfogalmazásában is helyet kaphatnak bizonyos memóriakiosztási meggondolások -- a szigorúan vett programírás időszakában ritkán kell magunkat az egyes utasítások memóriacímeivel terhelni! A fejlett fordítóprogramok használata pedig ezt a munkát is leegyszerűsíti.

PR12. VID-OFF A memórialapozási alapkonfiguráció visszaállítása

```
1.  LD    A,70          3E,70
2.  LD    (0003),A      32,03,00
3.  OUT   (02),A        D3,02
4.  RET                 C9
```

Folytassuk a szubrutin beírását közvetlenül az előbbi után, a 4008H címtől. Most a *0C9H* a 400FH címre kerül.

PR13. 15 színű vonal a képernyőn

```
 1. LD    A,(0B13)     3A,13,0B     A 06-os port másolata,
 2. PUSH  AF           F5           ezt a verembe tesszük
 3. AND   FC           E6,FC
 4. OR    02           F6,02        16-os grafikus mód
 5. OUT   (06),A       D3,02
 6. RST   30 : 05      F7,05        Képernyőtörlés
 7. CALL  VID-ON       CD,00,40     A VID belapozása
 8. LD    DE,9E1C      11,1C,9E     A vonal kezdőcíme,
 9. LD    HL,CKOD      21,38,40     a színbyte-ok címe
10. LD    BC,0008      01,08,00
11. LDIR               ED,B0        Másolás a VID-be
12. CALL  VID-OFF      CD,08,40     U0-U1-U2-SYS lapozás
13. RST   30 : 91      F7,91        Várakozás egy billentyű
14. POP   AF           F1           lenyomására
15. OUT   (06),A       D3,06        A korábbi üzemmód visz-
16. CALL  MON          CD,0F,27     szaállítása és kész

CKOD  01,0D,31,3D,C1,CD,F1,FD
```

Mint látjuk, ebben a programban valamivel többet foglalkoztunk a memóriával: meghatározott adatokat kell feldolgozni, amelyeket majd a memória meghatározott helyén el kell helyeznünk. Nyolc byte-ról van szó, s a 9. sorban ezek kezdőcímét a *CKOD névvel -- címkével* -- láttuk el. Programunk végén pedig feltüntettük, hogy *a CKOD nevű és kezdőcímű memóriaterületen milyen adatoknak kell állni.*

Az *assembly* mnemonikok mellett feltüntetett számkódok a 9. sorban *CKOD* értékét is rögzítik: 4038H lesz az adatok kezdő memóriacíme. Más választás is lehetséges, ekkor azonban a 9. sorhoz tartozó konkrét címbyte-okat át kell írni.

Elemezzük most magát a programot!

A 7. sorban a processzor a *verembe* tölti a 8. utasítás címét, *elugrik a hívott szubrutin 4000H kezdőcímére*, végrehajtja az ottani utasításokat, majd *a RET hatására a veremből visszatölti a PC regiszterpárba a korábban félretett címet, így visszaugrik programunk 8. sorára.*

Az ilyen szubrutinhívások egyszerű értelme tehát abban áll, hogy a mikroprocesszor a programunk soron következő utasítása előtt elvégzi a szubrutinban meghatározott feladatokat. Ebben az esetben belapozza a *videomemóriát.*

A szubrutinok alkalmazása programjainkban fölöttébb praktikus. Az ismétlődő részfeladatok logikus elkülönítésével tömör és áttekinthető, jól *struktúrált programfejlesztést* tesz lehetővé.

Bármelyik komplex funkciójú problémát részfeladatok sorozatára bonthatunk, benne logikai szinteket szervezhetünk, s a legalacsonyabb szintű szubrutinokból, mint dominókból, egyre összetettebb, de továbbra is elhatárolt funkciójú, magasabb szintű újabb szubrutinokat építhetünk.

Így *az egyre magasabb működési és logikai szintek felé haladva*, az összetett probléma egyre nagyobb része fogható át, míg végül olyan építményhez jutunk, amely a tervezett funkciót már mindenben teljesíti, ugyanakkor *az egész rendszer jól tagolt, áttekinthető, javítható, működése biztonságosan kézbentartható.*

Szubrutinok nélkül bonyolultabb programok megírása tulajdonképpen alig képzelhető el (vagy talán nem is)! Áttekinthető struktúra nélkül ui. a szerteágazó részfeladatok megoldása nem tartható kézben, a hibák elkövetésének lehetősége egyre nő, felderítésük egyre nehezebb lesz, végül az egész program egyetlen monumentális káosszá válik. Az ilyen stílusú programfejlesztés ellen a szubrutinok szervezése már eleve nyújt bizonyos garanciát.

A továbbiakban -- az említett okokból -- mi is egyre gyakrabban szervezünk programjainkban szubrutinokat.

Alkalmazásuk további előnye -- *de funkcionális szempontból éppen az is a célja* -- hogy programjaink bármelyik pontján kiadhatunk egy szubrutinhívó utasítást, amelynek eredményeként a CPU a soron következő feladat végrehajtása előtt elvégzi az ott előírt műveleteket, majd pedig automatikusan visszatér programunkhoz.

Sokszor szükség van arra, hogy az ilyen *alprogramok* bizonyos, előre nem konkrétan meghatározott adatokat dolgozzanak fel. Ezeket hívás előtt a CPU megfelelő regisztereibe lehet tölteni, de a memória kijelölt és közösen használt adatterületein is megtörténhet *a szükséges paraméterek átadása*. Ez utóbbi lehetőséggel bonyolultabb esetekben élünk.

Programunkat tovább elemezve, nézzük a 8. sorban elhelyezett címet! A célunk az, hogy a 8 byte-on ábrázolt, 16 pontból álló vonal a képernyő közepére kerüljön. Ez a 120. számú tv-sor, ennek címét kell tehát megkeresni a *videomemóriában.*

Mivel egy tv-sorhoz 40H byte tartozik, így 0100H címtartomány éppen 4 tv-sort fog át. A cím felső byte-jába tehát azt a számot kell írni, *ahányszor négy sor elfér az elérni kívánt tv-sorszámban.* A 120. sor esetén ez 30, aminek a hexadecimális 1EH felel meg. Mivel pedig *a videomemória kezdőcíme a 2. lapon 8000H*, így a 120. tv-sor elejének címe: 9E00H.

A 40H hosszúságú sor közepének címe: 9E20H. Mivel a tervezett 16 pont 8 byte-on fér el (16 színű üzemmód), így a kezdő memóriacím a sor közepétől 4-gyel balra, a kisebb címek felé lesz, így jön ki a programban szereplő 9E1CH érték.

A képernyőn megjelenő pontok színe balról jobbra: *fekete -- sötétkék -- sötétpiros -- sötétlila -- sötétzöld -- sötét kékeszöld -- sötétsárga -- szürke -- fekete -- kék -- piros -- lila -- zöld -- kékeszöld -- sárga --fehér.* Ezek kettesével adhatók meg egy-egy byte-on, így a *01, 0DH, 31H, 3DH, 0C1H, 0CDH, 0F1H, 0FDH* értékeket kapjuk (l. 2.2 alfejezet). Ezeket az adatokat tároljuk a programunk után, 4038...403FH címeken.

A 8--11. sorokban ezeket az értékeket másoljuk át az előbbi memóriarekeszekből a videomemória 9E1C...9E23H című byte-jaiba, ami ezek megjelenítésével egyenértékű.

A 12. sor ismét egyetlen szubrutinhívással intézi el a memórilapozási alapkonfiguráció visszaállítását, majd egy billentyű lenyomását megvárva, a program elején a verembe eltett *06-os portmásolat* alapján visszaállítjuk a korábbi üzemmódot és visszatérünk a *MONITOR*-ba.

Írjuk be a programot a 4010H címtől és munkánkat ellenőrizzük a *Z paranccsal*. Ha a program beírása jó, akkor a 4038H címtől írjuk be a pontok színét meghatározó értékeket, azután indítsuk el a programot: **J 4010**!

Ha eddig minden jól sikerült, írjuk át úgy a programot, hogy az előbbi vonal minden tv-sorban megjelenjen, így a képernyő közepén 16 pont szélességű, függőleges oszlopot alkosson!

A program itt következő listájában, tekintettel arra, hogy nagyon sok *veremműveletet* alkalmazunk, az ilyen utasítások soraiban sorszámmal megjegyeztük, hogy hányadik adat tárolásáról vagy visszatöltéséről van szó.

PR14. Színes oszlop

```
1.       LD   A,(0B13)          3A,13,0B
2.       PUSH AF        1. be   F5
3.       AND  FC                E6,FC
4.       OR   02                F6,02
5.       OUT  (06),A            D3,06
6.       CALL VID-ON            CD,00,40
7.       LD   B,F0              06,F0      240 sor lesz
8.       LD   DE,801C           11,1C,80   Az 1. sor címe
9.       LD   HL,4038           21,38,40   Az adatok címe
10. S1:  PUSH HL        2. be   E5         4038H tárolása
11.      PUSH DE        3. be   D5         Aktuális sor
12.      PUSH BC        4. be   C5         Ciklusváltozó
13.      LD   BC,0008           01,08,     Egy sor ábrá-
14.      LDIR                   ED,B0      zolása
15.      POP  BC        4. ki   C1         Ciklusváltozó
16.      POP  HL        3. ki   E1         Aktuális sor
17.      LD   DE,0040           11,40,00   Sorhossz
18.      ADD  HL,DE             19         A következő
19.      EX   DE,HL             EB         sor címe DE-be
20.      POP  HL        2. ki   E1         Adatok: 4038H
21.      DJNZ S1                10,EE      Ismétlés (240)
22.      CALL VID-0F            CD,08,40
23.      RST  30 : 91           F7,91      Várakozás
24.      POP  AF        1. ki   F1
25.      OUT  (06),A            D3,06
26.      CALL MONITOR           CD,0F,27
```

Írjuk be a programot a 4040H memóriacímtől és végezzük el a szokásos ellenőrzéseket, majd próbáljuk ki!

Értelmezzük a programban alkalmazott *ciklikus veremkezelési folyamat* értelmét! Gondoljuk át, hogy miért kell egyáltalán az egyes regiszterpárok tartalmát a *veremben* tárolni!

Figyeljük meg a *DJNZ S1 utasítás* működését, számoljunk utána a 21. sorban található 0EEH eltolási távolságnak (a 22. sor címe, mínusz a 10. sor címe)!

Feltételezhető, hogy az Olvasó az utóbbi programokat már gyorsan és könnyedén dolgozta fel. Munkája akkor tekinthető igazán eredményesnek, ha az egyre csökkenő bőségű megjegyzések alapján is jól eligazodik már az egymást követő utasításokon, ha a feladatok szóbeli megfogalmazása után már saját magában is körvonalazódnak azok megvalósítását szolgáló utasítások.

A *kódokkal* változatlanul nem kell sokat törődni. Könyvünkben eddig minden utasítás után odaírtuk a memóriába begépelendő számokat, s természetesen ezek alapján realizálódik a program végrehajtása -- de ezeket megtanulni nem kell. A leggyakrabban használt utasítások kódjai hosszabb-rövidebb programozási munka után ui. megmaradnak a fejünkben, ami pedig nem, azt kikereshetjük a megfelelő táblázatokból. Még később pedig úgyis érdemes lesz *assembler fordítóprogramot* használni, ami eleve megkímél minket a kódolás fárasztó munkájától. Akkor persze a mnemonikokkal megírt programot kell begépelni -- valamelyest szigorúbb formai szabályok szerint -- de az még mindig egyszerűbb.

Mivel a legelső gyakorlati és elvi nehézségeken már minden bizonnyal túltette magát az Olvasó, bátran kijelenthetjük, hogy ami eddig történt, az volt a bemelegítés.

Mostantól, könyvünk hátralevő részében, az eddigieknél összetettebb és programozási szempontból is értékesebb munkát végzünk. Lassan közelítünk az igazi programtervezési, programfejlesztési módszerekhez.

Formailag is érezhető változások lesznek. Először tisztázzuk a megoldandó programozási feladatot, azt önállóan megvalósítható, független részekre bontjuk, s ezt követi a programírás. Először azonban csakis a mnemonikokkal írjuk le a műveleteket.

Ha a program egy-egy *funkcionális* része összeállt, a megfelelő utasításokat az Olvasó megkísérelheti önállóan beírni a memóriába -- erre minden szükséges alkalommal talál útmutatást.

A teljes disassemblált listát -- a memóriacímek és az utasításkódok feltüntetésével -- ezután a kitűzött teljes programozási feladat elvégzése után, *egybefüggően adjuk meg.*

PR15. Két négyzet -- színváltással

Feladat: *ábrázoljunk a képernyő közepén egy sötét kékeszöld négyzetet, s bele egy kisebb, piros színűt. Érjük el, hogy a* **szóközbillentyű** *lenyomására ezek színeik cseréjével villogjanak a képernyőn, míg* **RETURN** *lenyomására a CPU térjen vissza a* **MONITOR**-*ba.*

Elemezzük a feladatot!

Legyen a nagyobbik négyzet vízszintes oldala a teljes képszélesség fele, azaz az *videomemóriában 20H byte.* Ez 4 * 20H db pontot jelent, tehát a függőleges méret -- *a tv-sorok száma* -- is legyen 80H.

Dolgozzunk 4-es üzemmódban, s a sötét kékeszöld szín kódját töltsük a *3. palettaregiszterbe.* A piros pedig legyen pl. a *2. palettaregiszterben.* Ez a kiosztás azért jó, mert a *0. és az 1. palettaregiszter* -- tehát a PAPER (a képernyő alapszíne) és az INK (az aktuális tintaszín) -- *beállítását nem befolyásolja.*

Írjuk meg a program bevezető részét!

. Képernyőtörlés után betöltjük a *palettaregiszterekbe* a megfelelő színkódokat, majd alkalmas ciklus szervezésével a *videomemória* megfelelő byte-jaiba a sötét kékeszöld színű négyzetet eredményező értékeket írjuk.

```
 1.        RST  30 : 05     Képernyőtörlés
 2.        LD   A,11        A sötét kékeszöld szín kódja
 3.        OUT  (63),A      Betöltés a 3. palettaregiszterbe
 4.        LD   A,44        A piros szín kódja
 5.        OUT  (62),A      Betöltés a 2. palettaregiszterbe
 6.        CALL 4000        A videomemória belapozása
 7.        LD   HL,8810     Próbaként az 1. tv-sorban kezdjük
 8.        LD   DE,0020     Eltolási érték a következő sorhoz
 9.        LD   A,FF        Ciánbyte a videomemóriában
10.        LD   B,80        Először 80H db sort próbálunk
11. S1:    PUSH BC          A sorszámlálő tárolása
12.        LD   B,20        Egy sorban 20H byte lesz
13. S2:    LD   (HL),A      Egy byte megjelenítése
14.        INC  HL          A következő cím a VID-ben
15.        DJNZ S2          Ismétlés 20H-szor
16.        ADD  HL,DE       A következő sor eleje
17.        POP  BC          A sorszámlálő
18.        DJNZ S1          Ismétlés 80H-szor
```

Írjuk be a program eddigi részét az 5000H címtől, ellenőrizzük listázással, s a végére *töréspontot* helyezve (**B 5022**) próbáljuk ki! (Segítségül l. a **PR15.** program teljes listáját!)

A program lefuttatásával láthatjuk, hogy eléggé feltűnően kevés a 80H db sor. Vegyünk hozzá még 10H-t, a 10. sorban írjuk át 80H-t 90H-ra:

```
M 5016 80 90                    <X>
MONITOR
*
```

Ismét próbáljuk ki! A képernyőn így már elfogadható négyzetet látunk.

Jegyezzük fel a vízszintes és függőleges méretek arányát: 20H : 90H = 1 : 4,5 (dec).

Most állítsuk be a négyzet függőleges helyzetét. A legfelső sor pl. 32 tv-sorral lejjebb kerül, ha a program első sorában 8010H helyett 8810H-t írunk. Javítsuk ki, futtassuk le, s mondjuk találjuk jónak ezt a helyzetet.

Ezután írjuk tovább a programot!

Legyen a belső, piros négyzet mérete a külsőnek mintegy kétharmad része! Az ennek megfelelő pontos érték 20,33 lenne, válasszuk ehelyett a némileg kisebb 20-at, ami 14H!

A méretarányt tartva: 4,5 * 20 = 90, tehát hexadecimálisan 5AH lesz a belső négyzet magassága, *tv-sorokban* mérve. Ez 36H-val kisebb mint az előző magasság, ezért ezt kettéosztva, 1BH db tv-sorral kell lejjebb kezdeni a piros négyzet első tv-sorát, mint a külső négyzeté volt. Mivel 1BH = 27, ami 6 * 4 + 3, így a piros négyzet első sorának kezdőcíme: 8ED6H. (*Javaslom, hogy az Olvasó ezeknek számoljon utána!*)

Írjuk meg most a piros négyzetet megjelenítő programrészt!

```
19.        LD   DE,002C      Eltolás a következő sor elejéhez
20.        LD   HL,8ED6      Az első piros sor elejének címe
21.        LD   A,0F         Piros szint adó byte
22.        LD   B,5A         A TV-sorok száma
23.  S3:   PUSH BC
24.        LD   B,14         A byte-ok száma egy tv-sorban
25.  S4:   LD   (HL),A       Egy byte megjelenítése
26.        INC  HL           A következő cím a VID-ben
27.        DJNZ S4           Ismétlés 14H-szor
28.        ADD  HL,DE        A következő sor eleje
29.        POP  BC           A sorszámlálló
30.        DJNZ S3           Ismétlés 5AH-szor
```

Gépeljük be az új programrészt -- folytatólagosan, az 5022H címtől -- és a végére helyezzünk el ismét *töréspontot* (**B 5037**). A *Z paranccsal* ellenőrizzük, s próbáljuk is ki. Ha a cián alapon megjelent a piros négyzet, akkor máris mehetünk tovább.

Most már csak az a kérdés maradt, hogy hogyan lehet a ***színváltást*** egyszerűen megoldani?

A legegyszerűbb eljárást az jelentené, ha a *2.* és *3. palettaregiszter* tartalmát felcserélnénk. Gondoljunk azonban arra, hogy a feladat csakis e két négyzet színének átváltása -- nem pedig az egész képernyőé. Most ez nem okoz-

na bajt, de ugyanilyen feladatokat sokszor kell úgy megoldani, hogy közben a képen más alakzatok is láthatók és *azok színe nyilvánvalóan nem változhat.*

Az ilyen feladatokra is felkészülve vessük el tehát a palettaregiszterek cseréjével járó, naivul egyszerű megoldást, s helyette dolgozzunk ki olyan eljárást, amelyben *a kívánt színváltás csak a kitűzött alakzatokra korlátozódik*!

Vegyük alaposabban szemügyre a *videomemória* cián és piros színt adó byte-ját -- 4-es üzemmódban:

cián	1	1	1	1	1	1	1	1
piros	0	0	0	0	1	1	1	1

Egy színváltáshoz jól láthatóan a felsö négy bitet kell csak megváltoztatni, mégpedig 1-röl 0-ra és fordítva. Ez pedig egyszerű *XOR* müvelettel elintézhetö:

	1	1	1	1	1	1	1	1 **cián**
XOR	1	1	1	1	0	0	0	0
	0	0	0	0	1	1	1	1 **piros**

és ugyanígy:

	0	0	0	0	1	1	1	1 **piros**
XOR	1	1	1	1	0	0	0	0
	1	1	1	1	1	1	1	1 **cián**

A program ennek megfelelöen a következö módon folytatható (ha az Olvasónak van kedve a megfelelö kódokat kikeresni, akkor akár azonnal be is gépelheti az 5037H címtöl).

```
31. S5:  RST  30 : 91      Várakozás egy billentyüre
32.      LD   A,C          A lenyomott billentyü kódja
33.      CP   20           Szóköz?
34.      JR   Z,S6         Ha igen: ugrás elöre!
35.      CP   0D           RETURN?
36.      JR   NZ,S5        Ha nem: ugrás vissza!
37.      CALL VID-OFF      RETURN után a memórialapozás
38.      CALL MON          visszaállítása és kész!

39. S6:  LD   DE,0020      A színváltások lebonyolítása:
40.      LD   HL,8810      ¦
41.      LD   B,90         ¦
42. S7:  PUSH BC           ¦
43.      LD   B,20         ¦ Minden byte tartalmát beol-
44. S8:  LD   A,(HL)       ¦ vassuk, az XOR F0 müvelettel
45.      XOR  F0           ¦ a felsö 4 bitet megfordítjuk,
46.      LD   (HL),A       ¦ majd ugyanoda visszatöltjük.
```

```
47.       INC  HL          ;
48.       DJNZ S8          ; Egy soron belül ezt ismétel-
49.       ADD  HL,DE       ; jük 20H-szor, majd áttérünk
50.       POP  BC          ; a következő tv-sorra és
51.       DJNZ S7          ; ezt csináljuk 90H-szor
52.       JR   S5          Ha a színváltás kész: ugrás
                           vissza!
```

Készen vagyunk! A program beírása után (l. a teljes listát) ellenőrizzük azt a szokásos módon, majd egy kazettára vegyük fel, pl. *PR15* néven. Az eljárás a következő:

1. Helyezzünk a magnetofonba egy felvétel készítésére alkalmas kazettát, tekercseljük a megfelelő helyre, állítsuka magnetofont felvételre, de még ne indítsuk el!
2. A program tárolásához használjuk a *MONITOR S parancsát*. A kezdőcím: 4000H, a végcím: 505FH, a program neve: PR15.
3. Ha ezeket beírtuk, indítsuk el a magnetofont és a **RETURN** billentyűt lenyomva készítsük el a felvételt!

Amikor kész, nyugodtan elkezdhetjük a program futtatásos vizsgálatát.

A PR15. program teljes listája

Szubrutinok:

VID-ON

```
4000 3E50      LD   A,50
4002 D302      OUT  (02),A
4004 320300    LD   (0003),A
4007 C9        RET
```

VID-OFF

```
4008 3E70      LD   A,70
400A D302      OUT  (02),A
400C 320300    LD   (0003),A
400F C9        RET
```

Főprogram:

```
                                 Előkészítés
5000 F7  05    RST  30: 05       ;
5002 3E11      LD   A,11         ; Képernyőtörlés, a pa-
5004 D363      OUT  (63),A       ; lettaregiszterek fel-
5006 3E44      LD   A,44         ; töltése és a videome-
5008 D362      OUT  (62),A       ; mória belapozása
500A CD0040    CALL 4000         ;
```

```
                                    Sötét kékeszöld négyzet
500D 211088     LD   HL,8010     ; (Próba után: 8810H!)
5010 112000     LD   DE,0020     ;
5013 3EFF       LD   A,FF        ;
5015 0680       LD   B,80        ; (Próba után: 90H!)
5017 C5         PUSH BC          ;
5018 0620       LD   B,20        ; A kijelölt videomemó-
501A 77         LD   (HL),A      ; ria területet kettös
501B 23         INC  HL          ; ciklusban a sötét ké-
501C 10FC       DJNZ 501A        ; keszöld szinnek megfe-
501E 19         ADD  HL,DE       ; lelö értékkel (OFFH)
501F C1         POP  BC          ; töltjük fel.
5020 10F5       DJNZ 5017        ;
                                    A piros szinü négyzet
5022 112C00     LD   DE,002C     ;
5025 21D68E     LD   HL,8ED6     ;
5028 3E0F       LD   A,0F        ;
502A 065A       LD   B,5A        ; Ujabb kettös ciklusban
502C C5         PUSH BC          ; -- most a piros szint
502D 0614       LD   B,14        ; adö tartalommal (0FH)
502F 77         LD   (HL),A      ; -- irjuk tele a video-
5030 23         INC  HL          ; memória belsö négyzet-
5031 10FC       DJNZ 502F        ; hez tartozö byte-jait.
5033 19         ADD  HL,DE       ;
5034 C1         POP  BC          ;
5035 10F5       DJNZ 502C        ;

5037 F7 91      RST  30: 91      A billentyüzet leolvasá-
5039 79         LD   A,C         sa
503A FE20       CP   20
503C 280A       JR   Z,5048      Csak a szököz vagy a
503E FE0D       CP   0D          RETURN lenyomását fogad-
5040 20F5       JR   NZ,5037     juk el

5042 CD0840     CALL 4008        Ha RETURN: VID-OFF és
5045 CD0F27     CALL 270F        vissza a MONITOR-ba!

5048 112000     LD   DE,0020     Szököz esetén szinváltás
504B 211088     LD   HL,8810     ;
504E 0690       LD   B,90        ;
5050 C5         PUSH BC          ;
5051 0620       LD   B,20        ; Ismét kettös ciklusban
5053 7E         LD   A,(HL)      ; változtatjuk meg a vi-
5054 EEF0       XOR  F0          ; deomemória négyzetek-
5056 77         LD   (HL),A      ; hez tartozö byte-jai-
5057 23         INC  HL          ; nak felsö 4 bitjét.
5058 10F9       DJNZ 5053        ;
505A 19         ADD  HL,DE       ;
505B C1         POP  BC          ;
505C 10F2       DJNZ 5050        ;

505E 18D7       JR   5037        Az elöbbiek ismétlése
```

Talán szolgál némi tanulságul -- és a további gépi kódú munkáinkhoz kedvcsinálónak sem rossz -- *ha programunkat BASIC-ben is megírjuk.*

Hagyjuk el a *MONITOR*-t a *H paranccsal*, a

```
BASIC ? (Y)
```

kérdésre válaszoljunk most Y-nal! Számítógépünk az *ok* szótöredék és a villogó négyszög kiírásával jelzi, hogy a *BASIC-ben* vagyunk.

Íme a megfelelő *BASIC program* (gyakorlatilag ugyanazt csinálja, amit mi az előbb gépi kódban):

```
 10 CLS
 15 SET PALETTE 0,85,68,17

 20 B1=51216: B2=52950
 25 FOR S=0 TO 143
 30 FOR P=0 TO 31
 35 POKE B1+S*64+P,255
 40 NEXT P: NEXT S

 45 FOR S=0 TO 89
 50 FOR P=0 TO 19
 55 POKE B2+S*64+P,15
 60 NEXT P: NEXT S

 65 A$=INKEY$: IF A$="" THEN 65
 70 IF A$=" " THEN 100
 75 IF ORD(A$)=13 THEN END: ELSE 65

100 FOR S=0 TO 143
105 FOR P=0 TO 31
110 POKE B1+S*64+P,PEEK(B1+S*64+P) XOR 240
115 NEXT P: NEXT S

120 GOTO 65
```

Ehhez nem fűzünk magyarázatot. Indítsuk el a **RUN** *paranccsal* és türelmesen játsszunk végig legalább egy színváltást! *Ehhez sem fűzünk magyarázatot!*

A **PRINT USR(9999)** *paranccsal* térjünk vissza máris a *MONITOR-ba!*

Könyvünk legbonyolultabb, egyben azonban legizgalmasabb programjához is érkeztünk.

PR16. Téglalap sprite (ejtsd: szprájt = szellem, manó) mozgatása

Feladat: *a fejezet videomemóriával foglalkozó részének befejezéseként varázsoljunk a képernyőre irányítható, téglalap alakú kis "szellemet" -- igen sokszor használt idegen szóval: sprite-ot --, amelyet a beépített botkormánnyal tudunk mozgatni, mégpedig az előbbi négyzeteken át is!*

Állapodjunk meg a téglalapszellem méretében. Legyen mondjuk a vízszintes szélesség -- pontokban kifejezve -- 12 pont, a téglalap magassága 8 tv-sor.

A szellem megjelenítése az eddigiek alapján egyszerűnek tűnik, vegyünk azonban figyelembe *két új, bonyolító körülményt!*

1. Az első az, hogy most a téglalapot *mozgatni akarjuk.* A függőleges irányú elmozdítás még mindig nem tűnhet túl nehéznek, hiszen az egész alakzatot csak egy-egy tv-sorral kell feljebb vagy lejjebb megjeleníteni, a feleslegessé váló sort pedig törölni.

A vízszintes mozgatás azonban problematikusabb. Nem lehet ui. 1 byte-tal jobbra, vagy balra tolni a téglalap képét a ***videomemóriában***, hiszen ez egyszerre 4 pontnyi helyváltozást jelentene. Ez nagyon durva, szakaszos, ugrálló mozgatást eredményezne -- így megengedhetetlen. *Finom mozgást kell megvalósítani.*

*Ha viszont egy ponttal visszük pl. jobbra, akkor a **videomemória** byte-jain belül a pontokhoz tartozó 4 bitpárral külön-külön kell foglalkoznunk.*

A párhuzamos függőleges oldalak miatt elég egyetlen sort elemezni. Ha a téglalap egy sorát, pl. a felsőt, már tudjuk mozgatni, akkor a többit ugyanúgy, alá kell csak másolni.

A látható pontokat egy-egy *-gal ábrázolva, legyen a felső sor alaphelyzete ez:

A byte sorszáma:	1.				2.				3.			
A pontok sorszáma												
1 byte-on belül:	1	2	3	4	1	2	3	4	1	2	3	4
	*	*	*	*	*	*	*	*	*	*	*	*

Mozgassuk el *jobbra:*

A byte sorszáma:	1.				2.				3.				4.			
A pont sorszáma:	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
		*	*	*	*	*	*	*	*	*	*	*	*			

Látszik, hogy a téglalap ábrázolása alaphelyzetben 3, jobbra elmozgatva 4 byte átírásával jár. Látható az is,

hogy 4 jobbra mozgatás után ismét az alaphelyzethez jutunk.

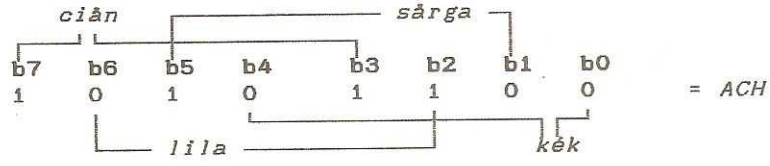
A megjelenítés lehetőségeit mérlegelve látható továbbá, hogy egyszeri elmozdításnál csak a két szélső pont állapota változik: a jobb oldalon egy új pontot kell megjeleníteni, a bal oldalon pedig a szélső pontot törölni kell!

2. Mielőtt továbbmennénk, második bonyolító körülményként vegyük figyelembe, hogy a szellemet úgy akarjuk mozgatni, hogy a képernyőn jelenlevő korábbi alakzatokon *legyen képes áthatolni*. Azaz: ahol a képernyő nem üres, ott a szellem jelenlétében megváltozhat ugyan az alakzatok színe, azonban mindegyiknek látszódnia kell, s a szellem távozása után pedig teljesen vissza kell állnia az eredeti állapotnak!

Az ábrázolás előbbi két problémáját összevetve látszik, hogy a kitűzött feladat megoldása az eddigieknél bonyolultabb elemzést, de feltehetően bonyolultabb programozási eljárását is kíván.

A két négyzet programozása során szerzett tapasztalataink alapján kijelenthetjük, hogy egyszerű esetben az *áthatolás* -- a megváltozás és a visszaállítás problémája -- *az XOR műveletre épített programmal* megoldható. Ugyanis az *XOR* nagyszerű tulajdonsága, hogy 1 értékű bitekkel kétszer egymás után végzett művelet után az eredeti állapot áll vissza.

Nézzünk konkrét példát! Legyen a *videomemória* egy byte-jában balról jobbra haladva egy cián, egy lila, egy sárga és egy kék színű pont bitképe. Dolgozzunk 4-es üzemmódban és *a palettaregiszterek kiosztása* legyen a következő: 0. szín = kék, 1. szín = sárga, 2. szín = lila, 3. szín = cián. Ezzel a byte értéke:



Hajtsunk végre most ezzel a byte-tal egy *XOR FF* műveletet! A színeket cián: C, lila: L, sárga: S és kék: K betűkkel jelölve:

```
          1 0 1 0   1 1 0 0      C-L-S-K
XOR       1 1 1 1   1 1 1 1
          0 1 0 1   0 0 1 1      K-S-L-C
```

Mind a négy pont színe érdekesen megváltozott!
Ismételjük meg az *XOR FF* átalakítást:

```
          0 1 0 1   0 0 1 1      K-S-L-C
XOR       1 1 1 1   1 1 1 1
          1 0 1 0   1 1 0 0      C-L-S-K
```

Visszakaptuk az eredeti pontokat!
Ugyanezt persze OFFH-tól eltérő értékekkel is megtehetjük. Néhány más változat:

```
     1010 1100   C-L-S-K          1010 1100   C-L-S-K
XOR  0000 1111                XOR  0110 1011
     1010 0011   S-K-C-L           1100 0111   S-C-L-L (!)

XOR  0000 1111                XOR  0110 1011
     1010 1100   C-L-S-K           1010 1100   C-L-S-K
```

A lehetőségek száma igen nagy.
Ez azt is jelenti, hogy ha az egész megjelenítést az *XOR* művelettel végezzük, ezzel az *áthatolás* kérdése már önmagában megoldódik.

Marad mindössze a ***mozgatás***. A téglalap igen egyszerű alakzat. Állandó szélességét felhasználva olyan szubrutinokat szervezünk, amelyekkel a négy vízszintes és gyakorlatilag egyetlen függőleges átmenet megoldható.

Tételezzük fel, hogy a téglalap már jelen van a képernyőn -- mégpedig alaphelyzetben -- és mozgassuk el egy ponttal jobbra! Legyen az első byte címe CCcc, s ez legyen a HL regiszterpárba betöltve, a *palettaszínek* kiosztása pedig: 0. fekete, 1. fehér. A *videomemória* vizsgált bytejaihoz tartozó fehér pontokat jelöljük *-gal, a feketéket pedig egy ponttal (.)!

```
   E r e d e t i                 Első eltolt helyzet
CCcc   CCcc+1 CCcc+2          CCcc   CCcc+1 CCcc+2 CCcc+3
****   ****   ****            .***   ****   ****   *...
```

Tervezzük meg a jobbra mozgató programot:

```
LD   A,(HL)        A CCcc című byte tartalma
XOR  80            A bal szélső pont fekete-fehér szín-
LD   (HL),A        váltása, majd visszatöltése
INC  HL            CCcc + 1
INC  HL            CCcc + 2
INC  HL            CCcc + 3, itt kell a jobb szélső pon-
LD   A,(HL)        tot felkapcsolni
XOR  80            A módszer ugyanaz
LD   (HL),A
```

A vízszintes *alaphelyzetet 0-nak*, az eltolással nyert helyzeteket *1-nek, 2-nek, ill. 3-nak nevezve* azt mondhatjuk, hogy az előbbi programrész a *0-1 átmenetet* valósítja meg.

A rutint ismételten alkalmazva természetesen vissza áll az eredeti állapot, amivel így nyilvánvalóan egy *1-0 típusú balra léptetés* valósul meg. Ez a kettős alkalmazási lehetőség mindegyik esetre érvényes.

Tervezzük meg az 1-2, a 2-3 és a 3-0 átmeneteket! Figyeljük meg a különböző pozíciókban levő pontok színváltásához szükséges számokat az *XOR* műveletekben.

1 - 2

```
.*** **** **** *...
..** **** **** **..
```

```
LD   A,(HL)
XOR  40
LD   (HL),A
INC  HL
INC  HL
INC  HL
LD   A,(HL)
XOR  40
LD   (HL),A
```

2 - 3

```
..** **** **** **..
...* **** **** ***.
```

```
LD   A,(HL)
XOR  20
LD   (HL),A
INC  HL
INC  HL
INC  HL
LD   A,(HL)
XOR  20
LD   (HL),A
```

3 - 0

```
...* **** **** ***.
.... **** **** ****
```

```
LD   A,(HL)       A CCcc című memóriarekesz tartalma
XOR  10           A bal szélső pont törlése,
LD   (HL),A       majd visszatöltése a VID-be
INC  HL           CCcc + 1
INC  HL           CCcc + 2
INC  HL           CCcc + 3, ez a cím tartalmazza a
LD   A,(HL)       jobb szélső pontot, mellé kerül
XOR  10           jobbra az új pont
LD   (HL),A
```

Ez a négy rutin annyira egyforma, hogy helyette *egyetlenegy* is használható lesz.

Eddig azt mondtuk, hogy a HL regiszterpárnak a kezdő videomemória-címet kell tartalmaznia. Ha ezenkívül pl. a C regiszterben megadjuk az XOR műveletnél használandó értéket, akkor mind a négy esetet feldolgozhatjuk a következő programmal:

```
LD    A,(HL)          A CCcc kezdőcímen levő adat
XOR   C               Művelet az aktuális értékkel
LD    (HL),A          Színváltás után megjelenítés
INC   HL
INC   HL
INC   HL              A téglalap sorának másik széle
LD    A,(HL)
XOR   C               A művelet ugyanaz
LD    (HL),A
```

Az egész téglalap megjelenítéséhez ugyanezt kell tenni nyolc soron át. Végül a következő praktikus szubrutint kapjuk:

PR16.1 Téglalap vízszintes mozgatása

in: HL = kezdőcím a videomemóriában;
C = a helyzetet és a mozgás irányát jellemző ún. "bitmaszk"

```
 1.       LD   DE,003D      Eltolás a következő sorhoz
 2.       LD   B,08         8 sor lesz

 3. S1:   LD   A,(HL)       ¦
 4.       XOR  C            ¦
 5.       LD   (HL),A       ¦
 6.       INC  HL           ¦ A sor elejének és végének át-
 7.       INC  HL           ¦ állítása a megbeszélt módon
 8.       INC  HL           ¦
 9.       LD   A,(HL)       ¦
10.       XOR  C            ¦
11.       LD   (HL),A       ¦

12.       ADD  HL,DE        A következő sor
13.       DJNZ S1           Ismétlés 8-szor

14.       RET               és kész
```

Figyeljük meg, hogy a következő sorokhoz szükséges eltolást a program elején töltjük be a DE regiszterpárba, az 1. sorban! A *DJNZ ciklus* működése során ez az érték már készen rendelkezésünkre áll a 13. sorban. A ciklus belsejében csak a működés lényegét alkotó utasítások állnak.

Korábbi programjainkban is így jártunk el, és általános szabályként is leszögezhető, hogy *egy többször ismétlődő, ciklikus utasításssorozatban a lehető legkevesebb, csakis a valóban szükséges műveleteket szerepeltessük!* A gyorsabb működés érdekében minden lehetséges beállítási, paraméterezési tennivalót *a cikluson kívül kell elintézni!*

Nézzük ezután, hogyan oldható meg a *függőleges irányú mozgatás!*

Ez esetben a téglalap alsó és felsö sorát kell teljes egészében megváltoztatni. Sajnos most sem mindegy, hogy melyik vízszintes helyzetben van a téglalap (0...3).

Az elsö sor kezdöcímét itt is megadhatjuk a HL regiszterpárban, a helyzetre jellemzö adatot pedig a mikroprocesszor C regiszterében. Elemezzük a feladatot pl. a *felfelé mozgatásnál, 1. helyzetü téglalap esetén.*

E r e d e t i	**Felmozgatás után**
....	.*** **** **** *...
.*** **** **** *...	.*** **** **** *...

A fehér-fekete, ill. a piros-ciân színváltási elvet tartva, egyszerü *átmaszkolás* válik lehetövé, ha a C-be ilyenkor a 0111 0000 bitképnek megfelelö 70H-t töltünk. Ez a sor elejét elintézi, utána két teljes byte következik. A sor végén azonban láthatóan éppen az a pont kell, amire az elején nem volt szükség. Az ehhez tartozó bitkép: 1000 0000, azaz 80H.

A sor eleje és vége között egyértelmü kapcsolat van. A megfelelö értékpárok, most már mind a 4 helyzetre:

	H e l y z e t			
	0	**1**	**2**	**3**
A sor eleje	F0	70	30	10
A sor vége	00	80	C0	E0

Vegyük észre, hogy ez a kapcsolat a C bitjei és a 0F0H-nak megfelelö 1 értékü bitek XOR müveleteivel megfogalmazható:

```
F0 XOR F0 = 00
70 XOR F0 = 80
30 XOR F0 = C0
10 XOR F0 = E0
```

Végül azt kell még megfontolnunk, hogy a téglalap felfelé mozgatásakor az elsö sor fölé -- egy korábban üres sorba -- kell az üj helyzethez tartozó felsö sort megjeleníteni, az eredeti alsó sort pedig törölni. Ezzel szemben lefelé mozgatás esetén a téglalap elsö sorát kell törölni, s az utolsó sor alá -- a korábban üres sorba -- kell egy üj sort rajzolni. A két feladat eléggé eltérö, ezért valahogy jelezni kell, hogy az aktuális függöleges mozgatás milyen irányü legyen. *Erre a funkcióra válasszuk ki pl. az F regiszter CY jelzöbitjét!*

Mindezek figyelembevételével a *függőleges mozgatás* most is megoldható *egyetlen szubrutinnal.*

PR16.2 Téglalap függőleges mozgatása

in: HL = a téglalap felső során kezdőcíme a videomemóriában;
C = a helyzetet jellemző bitmaszk;
CY = 1 lefelé kell mozgatni
0 felfelé kell mozgatni

Előkészítés a mozgatás iránya szerint:

```
 1.       JR   C,S2       ; Ha lefelé kell mozgatni: ugrás!
 2.       LD   DE,0040    ; Felfelé mozgatáskor HL-t az elő-
 3.       SBC  HL,DE      ; zö sorra állítjuk
 4. S2:   LD   DE,01FD    ; Eltolás a legalsó sorhoz
 5.       PUSH HL         ; A felső sor címének tárolása
 6.       PUSH AF         ; A CY jelzőbit tárolása
```

A mozgatás megvalósítása:

```
 7.       LD   B,02       ; 2-szer kell ugyanazt elvégezni
 8. S3:   LD   A,(HL)     ; (fent és lent)
 9.       XOR  C          ; A sor eleje
10.       LD   (HL),A     ;
11.       INC  HL         ;
12.       LD   A,(HL)     ;
13.       XOR  F0         ; Az elsö teljes byte
14.       LD   (HL),A     ;
15.       INC  HL         ;
16.       LD   A,(HL)     ;
17.       XOR  F0         ; A második teljes byte
18.       LD   (HL),A     ;
19.       INC  HL         ;
20.       LD   A,F0       ;
21.       XOR  C          ; A-ban a forditott maszk a sor
22.       XOR  (HL)       ; végéhez,
23.       LD   (HL),A     ; majd megjelenítés
24.       ADD  HL,DE      ; Az alsó sor címe
25.       DJNZ S3         ; Ismétlés az alsó sorban
```

A korrekt befejezés megszervezése:

```
26.       POP  AF         ; Vesszük az eredeti CY-t
27.       POP  HL         ; A felsö sor címe. Ha felfelé
28.       RET  NC         ; kellett mozgatni, akkor kész
29.       LD   DE,0040    ; Lefelé mozgatáskor az új tégla-
30.       ADD  HL,DE      ; lap felsö sora az eredeti alatt
                          ; van. Ezt tesszük a HL-be és igy
31.       RET             ; is kész
```

Mint látjuk, visszatéréskor a szubrutin a HL regiszterpárban az új téglalap felső sorának címét adja vissza, és az irányt jelző *carry jelzőbit* sem változik.

A vízszintes és a függőleges elemi mozgató rutinok felhasználásával most már könnyen megszerkeszthetők a tartós mozgatást *vezérlő és felügyelő programok*. Két feladatot kell megoldanunk:

1. *A szubrutinok hívása előtt* a HL regiszterpárba és a C regiszterbe a megfelelő értékeket kell betölteni. Ezt legegyszerűbben úgy tudjuk megoldani, hogy a memóriacímet és a pillanatnyi helyzet sorszámát (0...3) a memóriában meghatározott címen tároljuk, s innen olvassuk be, amikor kell. Fontos persze, hogy ezekben a memóriarekeszekben mindig az aktuális (érvényes) értékek legyenek.
   A téglalap kezdő videomemória-címét tároló két byte-ra a továbbiakban hivatkozzunk *CIM* névvel, a helyzet sorszámát tartalmazó memóriarekesz neve pedig legyen *TIP*. Mint arról már korábban beszéltünk, ezek mögött *az elnevezések mögött a programban a hivatkozott memóriarekeszek címét kell érteni.*

2. Mozgatás közben *a téglalap nem mehet le a képernyőről.* Ezt legegyszerűbben úgy oldhatjuk meg, hogy a képernyő szélein nem hajtjuk végre a továbbmozgatási parancsot.

PR16.3 A vízszintes mozgatás vezérlőrutinja

in : A = 01, ha jobbra kell lépni
FF, ha balra kell lépni

```
1.      LD   HL,(CIM)     Az aktuális videomemória-cím
2.      LD   B,A          Az irányjelzést megőrizzük
3.      INC  A            A = FF? -- ha igen: Z = 1
4.      JR   NZ,S4        Ha jobbra kell mozgatni: ugrás!
```

A balra lépés lehetőségének vizsgálata:

```
5.      LD   A,L          A képernyő bal szélén a cím al-
6.      AND  3F           só 6 bitje 0! Csak ezeket vizs-
7.      JR   NZ,S5        gáljuk. Ha nem a szélső byte,
                          akkor biztosan lehet balra lép-
                          ni, ugrás előre!
8. LD   A,(TIP)           Ha a képernyő bal szélén va-
                          gyunk akkor meg kell nézni a
9.      OR   A            helyzet sorszámát. Ha ez is 0,
10.     RET  Z            innen már nem lehet balra lép-
                          ni, visszatérés munka nélkül.
11.     JR   S5           Egyébként ugrás előre
```

A jobbra lépés ellenőrzése:

```
12. S4:  LD   A,L        Itt is csak a soron belüli pozí-
13.      AND  3F         ciót jellemző biteket nézzük
14.      CP   3B         összehasonlítás: a pozició 3BH?
15.      JR   NZ,S5      Ha nem, akkor a téglalap széles-
                         ségét is figyelembe véve, bizto-
                         san lehet jobbra lépni!
16.      LD   A,(TIP)    A helyzet sorszáma. Ha a téglalap
17.      CP   03         már a 3 helyzetben van: nem le-
18.      RET  Z          het jobbra lépni -- visszatérés
                         érdemi munka nélkül!
```

Ha a program eddig eljutott, akkor megengedett a vízszintes léptetés, a szellem nem megy le a képernyöről.

Ezután *beállítjuk az új helyzetet*. Ez az adat jobbralépéskor 1-gyel nő, de 03 után ismét 00 lesz. Balra lépéskor 1-gyel csökken, de 00 után ismét 03 lesz. A lépésirányt megadó és B-ben tárolt érték (01 vagy 0FFH) kitünően alkalmas a növelés és csökkentés végrehajtására, mivel a 0FFH éppen (-1)-nek felel meg. Ezt felhasználva az új helyzet sorszáma a következőképpen kapható meg:

```
19. S5:  LD   A,(TIP)    A téglalap korábbi helyzete
20.      LD   C,A        A sorszámot C-ben örizzük meg
21.      ADD  A,B        Az új helyzet (nyers) sorszáma
```

Ezzel persze a 03 + 01 := 00, ill. 00 - 01 := 03 átalakítás még nincs megoldva, azonban vegyük jobban szemügyre az *ADD A,B műveletet:*

```
      03                  00
   + 01                + FF
  ------              ------
      04                  FF
```

bitképe: 0000 0100 1111 1111

Jól látható, hogy az eredményből *csak az alsó két bitet hagyva meg*, ezeken éppen a kívánt átalakításnak megfelelő értékek állnak elö. Igy az igen egyszerü

```
22.      AND  03
```

müvelettel, amely a felsö hat bitet törli, máris az új helyzetre jellemző, a követelményeinknek megfelelő adatot kapjuk. Tüstént be is írjuk a megfelelő memóriarekeszbe:

```
23.      LD   (TIP),A
```

Eddig tehát elintéztük a vízszintes lépés lehetöségének vizsgálatát, s a *TIP változó adminisztrációját.*

Most *beállítjuk az új címet*. A téglalap felsö sorának *kezdö videomemória-címe* akkor változik, ha a 0 helyzetböl balra lépünk, vagy pedig a 3. helyzetböl jobbra. Az elöbbi esetben a cím 1-gyel csökken, jobbra lépéskor pedig nö. Ezeket a körülményeket részletesen meg kell vizsgálni!

```
24.       DEC  HL          Elöször feltételezzük, hogy a cí-
25.       PUSH HL          met csökkenteni kell!

26.       LD   A,B         A lépés iránya
27.       INC  A           Ha balra lépés volt, most A = 00
28.       OR   C           A korábbi helyzet. Ha a 0 hely-
                           zetböl balra lépés volt, akkor
                           az A továbbra is 00 és Z = 1!
29.       JR   Z,S6        Ha igy van: a cím helyes, ugrás
                           elöre!
30.       POP  HL
31.       INC  HL          Ha nem: a címet visszaállitjuk
32.       PUSH HL

33.       LD   A,B         Ismét a lépésirány
34.       DEC  A           Ha jobbra lépés volt, most A = 00
35.       OR   C           A korábbi helyzet. Ha a 3 hely-
                           zetböl volt jobbra lépés: A = 03
36.       CP   03          Valóban az?
37.       JR   NZ,S6       Ha ez sem, akkor a cím marad a ko-
                           rábbi, ugrás elöre!
38.       INC  HL          Ha viszont igen: növelni kell!

39. S6:   LD   (CIM),HL    A beállitott új címet tároljuk
40.       POP  HL
```

Erröl az utolsö müveletröl érdemes külön is néhány szót ejteni. A programrész végén a HL regiszterpár értéke kétféle lehet. Figyeljük meg, hogy a kritikus balra lépés esetén a 29. sorról ugrunk a 39.-re, tehát a *veremben* is ugyanaz az érték van, amit a *CIM* nevü rekeszekbe töltünk, vagyis a balra lépés miatt csökkentett kezdöcím. Más esetben -- tehát 3-0 tipusú kritikus jobbra lépéskor vagy ha a címet nem is kell változtatni -- a 40. sorban azt az értéket töltjük vissza a *veremböl*, amit a 32. sorban helyeztünk oda, vagyis a korábbi kezdöcímet.

Fontos, hogy ezek szerint *a HL regiszterpárban tárolt cím kritikus balra lépés esetén változik, jobbra lépéskor viszont nem!*

Erre a nagyon jelentös járulékos funkcióra azért van szükség, mert balra lépéskor az új szélsö pont valóban az 1-gyel kisebb videomemória-címhez tartozik -- és a megjelenitést is ezzel kell kezdeni. Ezzel szemben a kritikus jobbra lépés esetén a korábbi bal szélsö pontot kell elöször eltüntetni, ami valóban a HL regiszterpárban tárolt régi kezdöcímhez tartozik. A müködés csakis igy helyes.

Jó példa ez arra, hogy mennyire érdemes a legalaposabban és a legaprólékosabban nyomon követni a működés körülményeit és *lehetőleg minden részletre odafigyelni -- a programlépések első összeállítása idején.* A programfejlesztésnek ebben a stádiumában elkövetett néhány -- akár aprónak tűnő -- hiba is többszörösen kellemetlen következménnyel jár. A program mégsem úgy működik majd, ahogy terveztük -- ez még a kisebbik baj. De a hibákat utólag kideríteni és kijavítani már sokkal gyötrelmesebb dolog. Sajnos az ilyen intelmeket könnyű megfogalmazni, ám a gyakorlatban a programok bonyolultságával nem is egyenes arányban nő a hibák elkövetésének valószínűsége.

Programjaink első elindítása -- talán éppen az eleve feltételezett hibák okozta várakozás miatt -- *a munka legizgalmasabb része.* Amikor az Olvasó majd saját programokat ír, az első futtatások sikertelensége nehogy a kedvét szegje! Ha a program már bent van a memóriában, azt azonnal vegye fel kazettára, s utána jöjjön, aminek jönnie kell! A *programbelövés*: a hibák felkutatása és javítása (annak minden gyötrelmével), a jól működő programban utólag célszerűnek látszó apróbb módosítások bevitele, de főként a kezünk és agyunk munkája nyomán fokozatosan éledező program látványa -- *csodálatos, semmihez sem hasonlítható érzés. Többszöri átélése képes az ember egész személyiségét is megváltoztatni. Ha az Olvasó teheti -- ezt ne mulassza el!*

Feladatunkhoz visszatérve: eddig megoldottuk a vízszintes lépés teljes adminisztrációját. Hátra van még a megjelenítéshez szükséges *bitmaszk* beállítása. Figyeljük meg a két lépésirányhoz tartozó, egymástól korántsem független értékeket:

	Bitmaszk	
Aktuális helyzet	**Jobbra**	**Balra**
0	80	10
1	40	80
2	20	40
3	10	20

Az aktuális helyzet sorszámából a jobbra lépéshez használható érték pl. a következő utasításokkal állítható elő:

```
41.       LD   A,B        A lépésirány A-ban tároljuk
42.       LD   B,C        Az aktuális helyzet sorszáma
43.       SCF               CY = 1 (!)
44.       LD   C,00       Kiindulási érték
45.       INC  B          B értéke minimum 1!
46. S7:   RR   C
47.       DJNZ S7         Az  RR  C  ismétlése B = 00-ig
```

Az *RR utasítás* jobbra forgatja az operandus bitjeit. Az *INC B* miatt a forgatások száma: 1, 2, 3 vagy 4 lesz aszerint, hogy a 0, 1, 2 vagy 3 helyzetről volte szó. A forgatások száma szerint -- az elökészitö *SCF* utasitás eredményeként -- pedig igy alakul a C regiszter értéke:

Helyzet	**0**	**1**	**2**	**3**
Forgatások	1	2	3	4
C bitjei	1000 0000	0100 0000	0010 0000	0001 0000
C értéke	80	40	20	10

Ha balra kell léptetni, akkor az összefüggések tanulsága szerint ezekenek az értékeknek a 2-szeresét kell venni, de a 80H helyett 10H-t! Ez igen egyszerüen megoldható. Folytassuk a programirást! Most A-ban van az irányra jellemzö érték.

```
48.       INC   A         Ha balra kell lépni, akkor ezzel az
49.       JR    NZ,S8     A = 00 lett. Ha nem ez volt, akkor
                          a C igy helyes -- mehet elöre!
50.       SLA   C         Ha pedig valóban balra kell lépni,
                          akkor C bitjeit balra léptetve, azt
                          megszorozzuk 2-vel
51.       JR    NC,S8     Ha C-ben 80H volt, akkor a 7. bit
                          kilépése miatt: CY = 1. Ha nem igy
                          van, akkor kész -- mehet elöre --,
52.       LD    C,10      ha pedig C-ben 80H volt, akkor most
                          C = 10H-t állitunk be
```

Most már a címet és a *bitmaszkot* is elökészitettük, csupán a léptetés végrehajtása maradt hátra:

```
53. S8:   CALL PR15.1     A vizszintes léptetés
54.       RET             és kész!
```

Ugyanigy meg kell irnunk a függöleges léptetések felügyeletét és végrehajtását végzö programrészt. Nézzük!

PR16.4 Függöleges vezérlörutin

```
        in:  A = 01    Lefelé kell léni
                 0FFH  Felfelé lépés

 1.     LD    HL,(CIM)   Kezdöcim a videomemóriában
 2.     LD    B,A        Az irányt B-ben örizzük meg
 3.     INC   A          Ha felfelé kell lépni: Z = 1
 4.     JR    RZ,S9      Ha nem ez van: ugrás elöre!
```

A felfelé lépés ellenörzése:

```
 5.       LD    A,L       A cím alsó byte-ja
 6.       AND   C0        A tv-sor számához csak a felsö két
                          bit kell. Az elsö sorban értékük 00
 7.       OR    H         A felsö sorban H = 80H, tehát ha
 8.       CP    80        valóban itt vagyunk, akkor most az
                          A regiszter értéke is 80H, igy Z=1
 9.       RET   Z         Ilyenkor nem lehet már felfelé lép-
                          ni -- visszatérés, munka nélkül!
10.       JR    S9        Egyébként ugrás elöre!
```

A lefelé lépés ellenörzése:

```
11. S9:   PUSH  HL        A címet tároljuk, mert a HL-t át-
                          menetileg meg fogjuk változtatni!
12.       LD    A,L       Az alacsonyabb helyi értékü byte
13.       AND   C0        alsó hat bitjét törölve az aktuális
14.       LD    L,A       tv-sor elejének címét kapjuk
15.       LD    DE,B9C0   A megengedhetö legnagyobb cím: 8
                          tv-sorral a képernyö alja felett
16.       SBC   HL,DE     Ha HL-ben is ez volt, akkor kivonás
                          után 0 lesz -- ezt Z = 1 jelzi

17.       POP   HL        Most már HL értéke visszaállitható
18.       RET   Z         Ekkor nem lehet már lefelé lépni --
                          visszatérés érdemi munka nélkül!
```

Ha viszont a program eddig eljutott, akkor a léptetés lehetséges.

Most elöállitjuk a *bitmaszkot*. Nézzük meg itt is az összetartozó értékeket:

Helyzet	Bitmaszk
0	F0
1	70
2	30
3	10

Az értékek elöállitása pl. a következö eljárással történhet:

```
19. S10:  LD    A,(TIP)   Az aktuális helyzet sorszáma
20.       LD    C,F0      Kiindulási érték
21.       OR    A         A sorszám 00? -- Ha igen: Z = 1
22.       JR    Z,S12     Ekkor C értéke jó, ugrás elöre!

23. S11:  SRL   C         Egyébként C bitjeit jobbra eltol-
24.       DEC   A         juk és A-t csökkentjük annyiszor,
25.       JR    NZ,S11    amig A = 00 nem lesz
```

Eközben C bitjei így alakulnak:

Helyzet	0	1	2	3
C	1111 0000	0111 1000	0011 1100	0001 1100

Minket csak a felsö négy bit érdekel, ezért az alsó bitek nullázásával éppen a kívánt értékeket kapjuk:

```
26.       LD   A,C
27.       AND  F0        Az alsó bitek nullázása
28.       LD   C,A
```

Ezután már csak a léptetés iránya szerint be kell állítanunk a *CY jelzöbitet: CY = 1*, ha lefelé kell mozgatni a téglalapot. Mivel az irányt megadó érték most B-ben van, és B = 01 a lefelé, B = FF pedig a felfelé léptetést jelöli ki, így az

```
29. S12:  LD   A,01
30.       SUB  B
31.       CCF
```

müveletekkel éppen a kívánt beállítást érjük el. A kivonás után ui. B = 01 esetén A = 00 és *CY = 0* lesz az eredmény, míg B = FF esetén A = 02 és *CY = 1*. A 31. sor *CCF* utasítása a *CY jelzöbitet* megfordítja, így valóban a helyes állapothoz jutunk.

Most már meghívható a függöleges mozgatást végzö szubrutin.

```
32.       CALL PR15.2    A téglalap függöleges mozgatása
33.       LD   (CIM),HL  Az új kezdö memóriacím tárolása
34.       RET            és kész!
```

A mozgatásokat vezérlö -- a müködés szempontjából nagyon lényeges -- rutinok után írjuk meg még a téglalapszellemünket *elöször megjelenítö* programrészt. Eddig csak feltételeztük, hogy már jelen van a képernyön, ám valóban csak a szelleme volt meg. Megjelenítése azonban már egészen egyszerü eljárást jelent.

Itt intézzük el az összes eddigi szubrutinban használt *CIM* és *TIP* nevü memóriarekeszek kezdöértékeinek beállítását is (*inicializálás*).

PR16.5 A téglalap elsö megjelenítése

```
1.        XOR  A         A 0 helyzet jelzése,
2.        LD   (TIP),A   beírjuk a helyére
3.        LD   HL,B81F   Legyen pl. ez a kezdöcím,
4.        LD   (CIM),HL  ezt is tároljuk
```

```
 5.          LD    B,08       Most csak 8 tv-sor lesz
 6.          LD    DE,003D    Eltolás a következő sorhoz
 7.          LD    A,F0       A fehér szint eredményező byte
 8.  S13:    PUSH  BC         A sorszámlálò tárolása
 9.          LD    B,03       A téglalap szélessége: 3 byte
10.  S14:    LD    (HL),A
11.          INC   HL
12.          DJNZ  S14        Megjelenítünk egy sort
13.          ADD   HL,DE      A következő sor kezdöcíme
14.          POP   BC         A sorszámlálò a veremből
15.          DJNZ  S13        Ismétlés 8-szor
16.          RET              és kész
```

. Ezután már csak a szellemmozgatò program *vezérlö részének megírása* maradt hátra. Ebben felhasználjuk a korábban megírt **PR15.** programunkat, amely vagy most is be van írva a gépbe, vagy pedig kazettán a rendelkezésünkre áll. Ennek csak az eleje, az 5000...5036H címek közötti rész kell, amire a továbbiakban a **PR15A** névvel hivatkozunk.

PR16.6 Föprogram

```
 1.          CALL  PR14A      Megrajzoljuk a két négyzetet, majd
 2.          CALL  PR15.5     a téglalapot

 3.  S15:    CALL  PR15.7     Ez a szubrutin intézi el a billen-
                              tyüzet vizsgálatát (a föprogram u-
                              tán megírjuk) -- az A-ban vissza-
                              adja a lenyomott billentyü ködját
 4.          CP    OD         RETURN?
 5.          CALL  Z,MON      Ha igen: visszatérés a MONITOR-ba
 6.          CP    04         A beépitett botkormányon a jobbra
 7.          JR    Z,S18      lépés ködja? Ha igen: ugrás elöre!
 8.          CP    13         Balra?
 9.          JR    Z,S18      Ha igen: ugrás elöre!
10.          CP    05         Fel?
11.          JR    Z,S16      Ha igen: ugrás elöre!
12.          CP    18         Le?
13.          JR    NZ,S15     Ha ez sem: ugrás vissza (más bil-
                              lentyükkel nem foglalkozunk)!
```

Itt a függöleges mozgatást készitjük elö:

```
14.  S16:    SUB   06         "Fel" volt? Ha igen, akkor a kivo-
15.          JR    C,S17      nás után éppen A = FF!
16.          LD    A,01       Ha pedig "le" volt, akkor A = 01
                              -- irányjelzés beállitása
17.  S17:    CALL  PR15.4     A függöleges mozgatás elvégzése,
18.          JR    S15        utána ugrás vissza!
```

A vízszintes mozgatás előkészítése:

```
19. S18:  SUB  13        "Balra" volt? Ha igen, akkor a ki-
20.       JR   Z,S19     vonás után  A = 00
21.       LD   A,02      Ha "jobbra" volt, akkor A = 02
22. S19:  DEC  A         Így végül a helyes FF vagy 01 moz-
                         gatási irányjelzést kapjuk!
23.       CALL PR.15.3   A vízszintes mozgatás elvégzése,
24.       JR   S15       utána ugrás vissza!
```

Végül, a 3. sorban hívott **PR16.7** *billentyűzetkezelő rutin* legyen egyelőre a következő:

PR16.7 A billentyűzet beolvasása

```
1.        RST  30 : 91   A már korábban is használt funk-
2.        LD   A,C       cióhívással várakozunk egy billen-
3.        RET            lentyű lenyomására, majd az ered-
                         ményt A-ba áttesszük és kész
```

Ezzel a tervezett feladatot elvileg megvalósítottuk.

Üljünk a számítógép elé, s ha kell, töltsük be a *MONITOR-t*, ill. a **PR15** programot.

Ha a **PR15**-öt is a kazettáról kell betölteni, ehhez használjuk *a MONITOR L parancsát*. A kezdőcím: 4000H, a program neve: *PR15*. Ezeket az adatokat kell beírni *a MONITOR L parancshoz tartozó almenüjében* feltett kérdésekre.

Sikeres betöltés után elkezdhetjük a szellemmozgató program összeállítását!

Most már döntenünk kell, hogy programunkat hol helyezzük el a memóriában. Állapodjunk meg abban, hogy a *CIM* nevű memóriarekeszek az 5FF0--5FF1H címen lesznek -- itt tároljuk tehát a téglalap felső sorának mindenkori kezdő *videomemória-címét*. A *TIP* nevű memóriarekesz címe pedig legyen 5FF2H -- a téglalap pillanatnyi helyzetének sorszáma! A program összeállítását a *6000H címtől* kezdjük el!

Lépésenként végezzük el a következőket:

1. Másoljuk át a *videomemóriát belapozó* ***VID-ON (PR11) rutint*** a 6000H címtől! Korábban ezt a 4000H címre írtuk, ennek megfelelően a másolási eljárás a következő:

```
A 4000 4007 6000                    <RETURN>
MONITOR
*
```

2. Másoljuk át a 6010H címre a **PR15** programnak azt a részét, amely a négyzeteket jeleníti meg:

```
A 5000 5036 6010                    <RETURN>
MONITOR
*
```

Ebben a programban -- a most 6018H címre került ***CALL VID-ON*** utasítás kezdőcímeként még 4000H szerepel, ezt írjuk át 6000H-ra:

```
M 601A 40 60                        <X>
MONITOR
*
```

Az átmásolt rutin végén, a 6047H címre pedig írjunk egy *RET utasítást:*

```
M 6047 00 C9                        <X>
MONITOR
*
```

3. Disassemblálással ellenőrizzük az eddigieket:

```
Z 6000 6047
PRINTER?(Y)                         <RETURN>
6000 3E50      LD    A,50
.    .         .     .
:    :         :     :
6047 C9        RET
MONITOR
*
```

4. Írjuk be a **PR16.1** szubrutint a 6050H címtől! Ennek lezáró *RET* utasítása 6061H-ra kerül. A beírást a **Z 6050 6061** paranccsal ellenőrizzük, az esetleges hibákat javítsuk ki!

5. Írjuk be a **PR16.2** szubrutint a 6070H címtől! Ennek vége a 609BH címre kerül. A *Z paranccsal* ezt is ellenőrizzük!

6. Írjuk be a **PR16.3** vízszintes vezérlőrutint a 60A0H címtől! Végcíme: 60F6H.

7. Gépeljük be a **PR16.4** függőleges lépésvezérlő szubrutint a 6100H címtől! Végcíme: 6137H.

8. Írjuk be a **PR16.5** téglalap megjelenítő programrészt a 6140H címtől! Végcíme: 615CH.

9. Írjuk be a **PR16.6** fő vezérlőprogramot a 6160H címtől! Végcíme: 6194H.

10. Írjuk be végül 61A0H-tól a **PR16.7** rutint! Ennek végcíme: 61A3H.

Szubrutinok:

VID-ON

```
6000 3E50      LD   A,50          ; A videomemória belapo-
6002 320300    LD   (0003),A      ; zása a 8000...0BFFFH
6005 D302      OUT  (02),A        ; területre (2. lapra)
6007 C9        RET               ;
```

NÉGYZETEK

```
6010 F7 05     RST  30: 05        Képernyőtörlés
6012 3E11      LD   A,11         ;
6014 D363      OUT  (63),A       ; A színek beállítása
6016 3E44      LD   A,44         ;
6018 D362      OUT  (62),A       ;
601A CD0060    CALL 6000          VID-ON
601D 211088    LD   HL,8810      ;----------------------
6020 112000    LD   DE,0020      ;
6023 3EFF      LD   A,FF         ;
6025 0690      LD   B,90         ;
6027 C5        PUSH BC           ;
6028 0620      LD   B,20         ;   A sötét kékeszöld
602A 77        LD   (HL),A       ;     színű négyzet
602B 23        INC  HL           ;
602C 10FC      DJNZ 602A         ;
602E 19        ADD  HL,DE        ;
602F C1        POP  BC           ;
6030 10F5      DJNZ 6027         ;
6032 112C00    LD   DE,002C      ;----------------------
6035 21D68E    LD   HL,8ED6      ;
6038 3E0F      LD   A,0F         ;
603A 065A      LD   B,5A         ;
603C C5        PUSH BC           ;
603D 0614      LD   B,14         ;
603F 77        LD   (HL),A       ;
6040 23        INC  HL           ;    A piros négyzet
6041 10FC      DJNZ 603F         ;
6043 19        ADD  HL,DE        ;
6044 C1        POP  BC           ;
6045 10F5      DJNZ 603C         ;
6047 C9        RET               ;
```

PR16.1 Vízszintes mozgatás

```
6050 113D00      LD   DE,003D      ;
6053 0608        LD   B,08         ;
6055 7E          LD   A,(HL)       ;
6056 A9          XOR  C            ; A bal és jobb szélső
6057 77          LD   (HL),A       ; videomemóriabeli byte-
6058 23          INC  HL           ; okat a mozgatás aktuá-
6059 23          INC  HL           ; lis fázisának megfele-
605A 23          INC  HL           ; lően -- a C regiszter-
605B 7E          LD   A,(HL)       ; ben hozott bitmaszk
605C A9          XOR  C            ; szerint -- változtat-
605D 77          LD   (HL),A       ; juk meg
605E 19          ADD  HL,DE        ;
605F 10F4        DJNZ 6055         ;
6061 C9          RET               ;
```

PR16.2 Függőleges mozgatás

```
6070 3805        JR   C,6077       ;
6072 114000      LD   DE,0040      ;
6075 ED52        SBC  HL,DE        ;
6077 11FD01      LD   DE,01FD      ;
607A E5          PUSH HL           ;
607B F5          PUSH AF           ;
607C 0602        LD   B,02         ;
607E 7E          LD   A,(HL)       ;
607F A9          XOR  C            ;
6080 77          LD   (HL),A       ;
6081 23          INC  HL           ;
6082 7E          LD   A,(HL)       ;
6083 EEF0        XOR  F0           ;
6085 77          LD   (HL),A       ;
6086 23          INC  HL           ;
6087 7E          LD   A,(HL)       ;
6088 EEF0        XOR  F0           ;
608A 77          LD   (HL),A       ;
608B 23          INC  HL           ;
608C 3EF0        LD   A,F0         ;
608E A9          XOR  C            ;
608F AE          XOR  (HL)         ;
6090 77          LD   (HL),A       ;
6091 19          ADD  HL,DE        ;
6092 10EA        DJNZ 607E         ;
6094 F1          POP  AF           ;
6095 E1          POP  HL           ;
6096 D0          RET  NC           ;
6097 114000      LD   DE,0040      ;
609A 19          ADD  HL,DE        ;
609B C9          RET               ;
```

PR16.3 A vízszintes mozgatás vezérlőrutinja

```
60A0 2AF05F     LD   HL,(5FF0)     ;
60A3 47         LD   B,A           ;
60A4 3C         INC  A             ;
60A5 200C       JR   NZ,60B3       ; A balra léptetés lehe-
60A7 7D         LD   A,L           ; tőségének vizsgálata
60A8 E63F       AND  3F            ;
60AA 2014       JR   NZ,60C0       ;
60AC 3AF25F     LD   A,(5FF2)      ;
60AF B7         OR   A             ;
60B0 C0         RET  NZ            ;
60B1 180D       JR   60C0          ;-----------------------
60B3 7D         LD   A,L           ;
60B4 E63F       AND  3F            ;
60B6 FE3B       CP   3B            ; A  jobbra léptetés le-
60B8 2006       JR   NZ,60C0       ; hetőségének vizsgálata
60BA 3AF25F     LD   A,(5FF2)      ;
60BD FE03       CP   03            ;
60BF C8         RET  Z             ;

60C0 3AF25F     LD   A,(5FF2)      ;
60C3 4F         LD   C,A           ; Az új helyzet sorszá-
60C4 80         ADD  A,B           ; mának beállítása (TIP)
60C5 E603       AND  03            ;
60C7 32F25F     LD   (5FF2),A      ;

60CA 2B         DEC  HL            ;
60CB E5         PUSH HL            ;
60CC 78         LD   A,B           ;
60CD 3C         INC  A             ;
60CE B1         OR   C             ;
60CF 280B       JR   Z,60DC        ;
60D1 E1         POP  HL            ;
60D2 23         INC  HL            ; Az új kezdőcím beállí-
60D3 E5         PUSH HL            ; tása  (CIM)
60D4 78         LD   A,B           ;
60D5 3D         DEC  A             ;
60D6 B1         OR   C             ;
60D7 FE03       CP   03            ;
60D9 2001       JR   NZ,60DC       ;
60DB 23         INC  HL            ;
60DC 22F05F     LD   (5FF0),HL     ;
60DF E1         POP  HL            ;

60E0 78         LD   A,B           ;
60E1 41         LD   B,C           ;
60E2 37         SCF                ;
60E3 0E00       LD   C,00          ; A helyzetnek megfelelő
60E5 04         INC  B             ; bitmaszk elkészítése
60E6 CB19       RR   C             ;
60E8 10FC       DJNZ 60E6          ;
```

```
60EA 3C          INC  A              ;
60EB 2006        JR   NZ,60F3        ; A helyzetnek megfelelő
60ED CB21        SLA  C              ; bitmaszk elkészítése
60EF 3002        JR   NC,60F3        ; (folytatás)
60F1 0E10        LD   C,10           ;

60F3 CD5060      CALL 6050           A mozgatás végrehajtása
60F6 C9          RET
```

PR16.4 A függőleges mozgatás vezérlőrutinja

```
6100 2AF05F      LD   HL,(5FF0)      ;
6103 47          LD   B,A            ;
6104 3C          INC  A              ;
6105 2009        JR   NZ,6110        ; A felfelé léptetés le-
6107 7D          LD   A,L            ; hetőségének ellenőrzé-
6108 E6C0        AND  C0             ; se
610A B4          OR   H              ;
610B FE80        CP   80             ;
610D C8          RET  Z              ;
610E 180C        JR   611C           ;

6110 E5          PUSH HL             ;
6111 7D          LD   A,L            ;
6112 E6C0        AND  C0             ;
6114 6F          LD   L,A            ; A lefelé léptetés le-
6115 11C0B9      LD   DE,B9C0        ; hetőségének ellenőrzé-
6118 ED52        SBC  HL,DE          ; se
611A E1          POP  HL             ;
611B C8          RET  Z              ;

611C 3AF25F      LD   A,(5FF2)       ;
611F 0EF0        LD   C,F0           ;
6121 B7          OR   A              ;
6122 2809        JR   Z,612D         ;
6124 CB39        SRL  C              ; A helyzetnek megfelelő
6126 3D          DEC  A              ; bitmaszk előállítása
6127 20FB        JR   NZ,6124        ;
6129 79          LD   A,C            ;
612A E6F0        AND  F0             ;
612C 4F          LD   C,A            ;

612D 3E01        LD   A,01           ;
612F 90          SUB  B              ; A lépésirány jelzése
6130 3F          CCF                 ;

6131 CD7060      CALL 6070           A mozgatás végrehajtása
6134 22F05F      LD   (5FF0),HL      Az új kezdőcím tárolása
6137 C9          RET
```

PR16.5 A téglalap első megjelenítése

```
6140 AF          XOR  A              ;
6141 32F25F      LD   (5FF2),A       ; Inicializálás
6144 211FB8      LD   HL,B81F        ;
6147 22F05F      LD   (5FF0),HL      ;

614A 0608        LD   B,08           ;
614C 113D00      LD   DE,003D        ;
614F 3EF0        LD   A,F0           ;
6151 C5          PUSH BC             ;
6152 0603        LD   B,03           ;
6154 77          LD   (HL),A         ; A téglalap megjelení-
6155 23          INC  HL             ; tése
6156 10FC        DJNZ 6154           ;
6158 19          ADD  HL,DE          ;
6159 C1          POP  BC             ;
615A 10F5        DJNZ 6151           ;
615C C9          RET                 ;
```

PR16.6 Főprogram

```
6160 CD1060      CALL 6010           Négyzetek megjelenítése
6163 CD4061      CALL 6140           A téglalap ábrázolása

6166 CDA061      CALL 61A0           A billentyűzet leolvasá-
6169 FE0D        CP   0D             sa
616B CC0F27      CALL Z,270F         RETURN-re befejezés

616E FE04        CP   04             ;
6170 2817        JR   Z,6189         ; A függőleges lépésvá-
6172 FE13        CP   13             ; lasztás vizsgálata
6174 2813        JR   Z,6189         ;

6176 FE05        CP   05             ;
6178 2804        JR   Z,617E         ; A vízszintes lépésvá-
617A FE18        CP   18             ; lasztás vizsgálata
617C 20E8        JR   NZ,6166        ;

617E D606        SUB  06             ;
6180 3802        JR   C,6184         ; A vízszintes lépés e-
6182 3E01        LD   A,01           ; lőkészítése és vizsgá-
6184 CD0061      CALL 6100           ; lata
6187 18DD        JR   6166           ;

6189 D613        SUB  13             ;
618B 2802        JR   Z,618F         ;
618D 3E02        LD   A,02           ; A függőleges lépés e-
618F 3D          DEC  A              ; lőkészítése és vizsgá-
6190 CDA060      CALL 60A0           ; lata
6193 18D1        JR   6166           ;
```

PR16.7 A billentyűzet vizsgálata

```
61A0 F7 91      RST  30: 91      Várakozás egy billentyű
61A2 79         LD   A,C         lenyomására, majd a kód
61A3 C9         RET              áttöltése A-ba
```

11. Az összes programrész ellenőrzése után *vegyük fel kazettára az egész programot a MONITOR S parancsával!* A beírandó adatok: kezdőcím 6000H, végcím 61A3H, a program neve legyen *"SPRITE"*!

Ha mindez kész, munkánk legizgalmasabb és legfelemelőbb részéhez érkeztünk! *Indítsuk el a vezérlőprogramot:*

J 6160 <RETURN>

Ha mindent jól csináltunk, akkor szinte azonnal megjelenik a két színes négyzet, s valahol a képernyő alján egy kis fehér téglalap.

Nyomjuk le a **RETURN** billentyűt! Ha programunk visszatér a *MONITOR*-ba, akkor eddig jó!

Indítsuk el újra (**J 6160, RETURN**), s most már próbáljuk meg elmozgatni a beépített botkormánnyal! Feltehetően valóban el is mozdul, s ekkor figyeljük meg, hogy mozgása pontosan követi-e a különböző irányokra vonatkozó parancsainkat. *A mozgásnak tisztának, egyértelműnek és egyszerinek kell lennie*, nem maradhatnak kísérő pontok, vonaldarabok mögötte!

Mozgassuk ezután téglalapunkat folyamatosan felfelé, s figyeljük meg a viselkedését, amikor *behalad a nagy, színes négyzetekbe!* Járjuk körbe a négyzetek határvonalát és közben *figyeljük a színváltozásokat!*

Ha közben bármilyen problémát észlelnénk, azonnal nyomjuk meg a **RETURN** billentyűt, ezzel visszatérhetünk a *MONITOR*-ba. Ha a program időközben már a **RETURN** billentyűre sem reagál, akkor még megpróbálkozhatunk a számítógép alján levő **RESET** gomb egyszeri megnyomásával. Ha ez sem segít, akkor nyomjuk meg kétszer ezt a gombot, s *töltsük be újra az előbb kimentett programot a MONITOR-ral együtt!*

Ilyen hiba esetén csak azt tehetjük, hogy disassemblálással *ismét ellenőrizzük* az összes szubrutin helyességét, nagyon figyelmes munkával keressük meg a hibás byteokat, s *a MONITOR M parancsával* javítsuk ki. Esetleg megpróbálkozhatunk *töréspontok* elhelyezésével is -- azon-

ban ha a programot valóban helyesen írtuk be, benne hibát nem találunk, akkor a működés is hibátlan lesz!

A jól működő program esetén is bizonyára azonnal feltűnik, hogy még *a folyamatos mozgás is meglehetősen, mitöbb kiábrándítóan, sőt felháborítóan lassú. Munkánk végülis jó, de csak ennyire képes?!*

Nos, a kérdés teljesen jogos -- és egyáltalán nem így van! A csigalassúság okozója az utolsó, billentyűzetet beolvasó rutin, amelyet a végére hagytunk.

Ezt kell még alaposan átdolgoznunk, s majd akkor derül ki, hogy programunk mire képes!

Ez azonban már a fejezet következő részének a feladata.

4.4 Billentyűzet

Ott folytatjuk, ahol az előbb abbahagytuk!

Nagyon negatív tapasztalatot szereztünk az

RST 30 : 91

funkcióhívás sebességéről. Sajnos, tudomásul kell vennünk, hogy a TV-Computer összes eszközéhez egyszerű és intelligens hozzáférést biztosító *funkcióhívásokkal* mindig ez a helyzet. Amikor az idő a működés döntő tényezőjévé válik, a funkcióhívások által biztosított kényelemről le kell mondani. Ezek általában nagyon sokoldalú és bonyolult programok. Ezenkívül igen sok időt rabol el a funkciókódban előírt feladat értelmezése és azonosítása is -- a *működés tehát lassú.*

Programjainkban mégis sokszor használjuk ezeket, hiszen a kényelmes kezelhetőség és *sokoldalúság,* de éppúgy a *programbeli rövidség* miatt is -- célszerű az alkalmazásuk. Am csak addig, amíg a működési sebesség a feladatban nem kritikus szempont. Ha igen, akkor ki kell váltani a funkcióhívásokat.

Éppen egy ilyen pillanathoz érkeztünk a fejezet előző részének végén.

A billentyűzet kezelésének leggyorsabb módszere a nyomógombok közvetlen vizsgálata. (Ugyanúgy, mint ahogy a grafikus programokat is közvetlenül a *videomemóriában* dolgozva tudjuk a leggyorsabbá tenni.)

A billentyűk passzív vagy lenyomott állapota egyetlen bittel megadható, így a TV-Computerben célszerűen alkalmazott megoldás az, hogy a billentyűket 8-asával csoportosítjuk, s a CPU a kiválasztott csoport mind a 8 billentyűjének állapotát egyszerre olvashatja be -- egyetlen 8 bites adatként.

Ennek részleteiről már minden fontos tudnivalót leírtunk a 2.3 alfejezetben. Lapozzunk most vissza az ott található táblázathoz!

A 8. sor a beépített és az első, a 9. pedig a hátsó botkormányhoz tartozik. Ezekkel kell most behatóbban foglalkozni.

A billentyűzet vizsgálata *két lépésben* történik. Először a *03-as port alsó négy bitjébe* be kell írni a kiválasztott sorszámot. Utána *az 58H-s portról* egyszerre olvasható be a kiválasztott sorhoz tartozó nyolc billentyű állapota. Nagyon *fontos* tudnivaló, hogy amelyik billentyű le van nyomva, közvetlen leolvasás esetén a hozzá tartozó bit értéke: *0!*

Mindezek azonnali, egyetlen és komoly alkalmazásaként térjünk vissza a **PR16** programhoz és ennek **PR16.6** vezérlőprogramjában javítsuk ki az ottani lassú működést okozó billentyűzetrutint!

A programban a 6166H címen szereplő *CALL PR16.7* utasításban a billentyűzet szubrutin címét írjuk át 61A0H-ról 61B0H-ra és kezdjük el a munkát!

PR17. A billentyűzet közvetlen beolvasása

A 2.3 alfejezetben közölt táblázatban látszik, hogy a *beépített botkormány* állapota a 8. sorban, a *RETURN* pedig az 5. sorban olvasható le.

Mivel *a 03-as port felső bitjei* más funkciókhoz tartoznak, íráskor fel kell használnunk a *0B11H* című memóriarekeszben található *portmásolatot.*

```
 1. DI                  A vizsgálat idejére tiltjuk a meg-
                        szakításokat, megakadályozva ezzel
                        a TV-Computer ROM-jában tárolt
                        program ugyanilyen tevékenységét
 2. LD   A,(0B11)       A 03-as port másolata
 3. AND  F0             A felső biteket meghagyjuk, a port-
 4. PUSH AF             másolat többi részét a verembe
 5. OR   05             tesszük, s először az 5. sort vá-
 6. OUT  (03),A         lasztjuk ki
 7. IN   A,(58)         A sor állapotát beolvassuk A-ba
 8. LD   C,A            A J, É, Ü, RETURN, Á, L, K és DEL
                        billentyűkre vonatkozó adatot át-
                        tesszük a C regiszterbe
 9. POP  AF
10. OR   08             Ezután a 8. sort választjuk ki és
11. OUT  (03),A         olvassuk be
12. IN   A,(58)
```

Így elvileg kész is -- mint látjuk, ennyire egyszerű --, de azért helyesebb, ha az eredményt nem két regiszterben adjuk vissza: végezzünk el egy kis átrendezést!

A ***botkormány*** állapotát az A regiszter bitjei mutatják -- itt azonban a legfelsö (b7) bit szabad. Ezért a *RETURN* billentyü állapotát (amelyet pillanatnyilag a C regiszter 4. bitje jelez) ide átmásolhatjuk.

```
13.        OR   80          A 7. bit elöször legyen: 1!
14.        BIT  4,C         Ha a RETURN nem volt lenyomva,
15.        JR   NZ,S1       akkor ugrás elöre,
16.        RES  7,A         egyébként a 7. bitbe 0-t irunk
16. S1:    EI               Végül ismét engedélyezzük a meg-
17.        RET              szakitásokat és kész
```

Visszatéréskor tehát az A bitjei hordozzák a minket érdeklö információkat:

b7 = 0	*RETURN*
b6 = 0	*balra*
b5 = 0	*jobbra*
b2 = 0	*le*
b1 = 0	*fel*

Irjuk be a szubrutint a 61B0H memóriacimtöl, s a szokásos módon ellenörizzük:

```
61B0 F3          DI
61B1 3A110B      LD   A,(0B11)
61B4 E6F0        AND  F0
61B6 F5          PUSH AF
61B7 F605        OR   05
61B9 D303        OUT  (03),A      A RETURN sorának beol-
61BB DB58        IN   A,(58)      vasása
61BD 4F          LD   C,A
61BE F1          POP  AF
61BF F608        OR   08
61C1 D303        OUT  (03),A      A beépitett botkormány
61C3 DB58        IN   A,(58)      sorának beolvasása
61C5 F680        OR   80          !
61C7 CB61        BIT  4,C         ! Az eredmény összege-
61C9 2002        JR   NZ,3DCB     ! zése az A regiszter-
61CB CBBF        RES  7,A         ! ben
61CD FB          EI
61CE C9          RET
```

Ezután át kell még irnunk a **PR16.6** program vezérlörutinjának végét. Korábban már kijavitottuk a billentyüzetrutin hivási címét, ezért a beirást a 6169H memóriarekesszel kezdjük.

```
6166 CDB061     CALL 61B0       A vizsgált billentyűk
6169 CB7F       BIT  7,A        RETURN?
616B CC0F27     CALL Z,270F     Ha igen: visszatérés!

616E CB77       BIT  6,A        Balra?
6170 2007       JR   NZ,6179    Ha nem: ugrás tovább,
6172 F5         PUSH AF
6173 3EFF       LD   A,FF
6175 CDA060     CALL 60A0       ha igen: balra léptetés
6178 F1         POP  AF

6179 CB6F       BIT  5,A        Jobbra?
617B 2007       JR   NZ,6184    Ha nem: ugrás tovább,
617D F5         PUSH AF
617E 3E01       LD   A,01
6180 CDA060     CALL 60A0       ha igen: jobbra léptetés
6183 F1         POP  AF

6184 CB57       BIT  2,A        Le?
6186 2007       JR   NZ,618F    Ha nem: ugrás tovább,
6188 F5         PUSH AF
6189 3E01       LD   A,01
618B CD0061     CALL 6100       ha igen: lefelé léptetés
618E F1         POP  AF

618F CB4F       BIT  1,A        Fel?
6191 20D3       JR   NZ,6166    Ha nem: ugrás vissza,
6193 3EFF       LD   A,FF
6195 CD0061     CALL 6100       ha igen: felfelé lépés

6198 18CC       JR   6166       Végül: ugrás vissza!
```

Ha kész, ellenőrizzük a *Z paranccsal*, s ha jó, mielőtt kipróbálnánk, *vegyük fel ismét kazettára, pl. SPRITE1 névvel*. (Vigyázzunk: a végcím 61CCH lett!)

Ezután vizsgáljuk meg a program működését!

Ha az utóbbi fejlesztéseket nem rontottuk el, akkor a hatás -- feltételezem -- *sokkal megdöbbentőbb* lesz, mint amit korábban a program lassúsága okozott.

Be kell látnunk: a működési sebesség az előbbinek annyira sokszorosára nőtt, hogy most a téglalap mozgását lehetetlen kézben tartani! Bármilyen finoman is próbáljuk a botkormányt egy-egy pillanatra elmozdítani, programunk annyira gyors, hogy közben a téglalapszellem egy mozdulattal végigpásztázza az egész képernyőt. Közbülsö helyzeteit a televízió elektronikája sem képes folyamatosan megjeleníteni. Programunkat feltétlenül *le kell lassítani!*

Ez a probléma mindenesetre jóval kellemesebb, mint a korábbi volt -- és igen könnyen megoldható.

Egyszerű *időhúzást* végez a következő, néhány utasításból álló szubrutin:

```
61D0 D5          PUSH DE
61D1 F5          PUSH AF
61D2 117777      LD   DE,7777      Időhúzás: kezdőérték
61D5 1B          DEC  DE           DE-t ismételten csök-
61D6 7A          LD   A,D          kentjük -- egyesével --
61D7 B3          OR   E            amíg DE = 0000-t nem ka-
61D8 20FB        JR   NZ,61D5      punk
61DA F1          POP  AF
61DB D1          POP  DE
61DC C9          RET
```

További fejlesztésként írjuk be ezt a szubrutint az itt látható 61D0H címtől.

Az időhúzást vezérlőprogramunkban a billentyűzetrutin hívása elé kell beszúrnunk -- a ***MONITOR*** parancsaival ezt a következőképpen tehetjük meg:

```
I 6166 6199 03                    <RETURN>
MONITOR
M 6166 00 CD                      <RETURN>
6167 00 D0                        <RETURN>
6168 00 61                        <X>
MONITOR
X
```

Nagyon fontos, hogy ezután még helyre kell hoznunk a 6194H címre került *JR NZ* és a 619DH-ra került *JR* utasításokhoz tartozó eltolást:

```
M 6195 D3 D0                      <M>
M 619C CC C9                      <X>
MONITOR
X
```

Ismét ellenőrizzük ezt a programrészt a **Z 6166 619C** *paranccsal*, s ha hibátlan, próbáljuk ki: **J 6160**!

Láthatjuk, hogy az időhúzás beiktatásával *sikerült programunkat a végsőkig lelassítani*. Ennek ellenére javaslom, hogy az Olvasó az időhúzást próbálja ki a leghosszabb várakozást okozó 0000 értékkel is, majd állítsa be a szerintem elfogadható: 0200H értéket (ezeket *a* ***MONITOR M*** *parancsával* a 61D3--61D4H címekre kell írni)! A beállítás olyan legyen, amelynél *a téglalap mozgása friss, dinamikus, de finom helyváltoztatások is könnyen végrehajthatók!*

4.5 Hangképzés

Az Olvasó bizonyára nem venné jó néven, ha ebben a könyvben semmi szó nem esne a TV-Computer hangképzési lehetőségeiről.

Eleget téve a nyilvánvalóan jogos igénynek, rövidesen bemutatunk néhány, ezzel kapcsolatos szubrutint.

Az igazság azonban az, hogy a TV-Computernek egycsatornás -- és nem is túlságosan fejlett -- hanggenerátora van. Így az igazán értékes hangeffektusok előállítása csak *logikailag és szerkezetileg is nagyon összetett programokkal lehetséges.* Várható, hogy a közeljövőben a témát mélyebben tárgyaló szakirodalomban az Olvasó ennek részleteit is megismerheti.

Triviális, de nem túl értékes hanghatások érhetők el az *RST 30 : 33 funkcióhívással.* Ennek alkalmazására mutat példát a következő program.

PR18. Egyhangú dallam

Az egymás után megszólaló hangok *PITCH* értékét (hangmagasságát) és hangerejét, valamint időtartamát tároljuk a memória 3000H kezdőcímű táblázatában, amelynek szerkezete:

1. byte PITCH, alsó byte;
2. byte PITCH, felső byte;
3. byte hangerő;
*4. byte időtartam (*20 ms)*

```
3800 210030     LD   HL,3000      Az adatok kezdőcíme
3803 5E         LD   E,(HL)       Betöltjük DE-be az ak-
3804 23         INC  HL           tuális hanghoz tartozó
3805 56         LD   D,(HL)       PITCH értéket
3806 23         INC  HL
3807 14         INC  D            Ha a PITCH alsó byte-ja
3808 15         DEC  D            00, akkor visszatérés a
3809 CC0F27     CALL Z,270F       MONITOR-ba!
380C 4E         LD   C,(HL)       A hangerő
380D 23         INC  HL           A címmutató továbblépése
```

```
380E 0602      LD   B,02        Szorzó az időtartamhoz
3810 AF        XOR  A           A beolvasott adat és a
3811 86        ADD  A,(HL)      szorzó alapján beállit-
3812 10FD      DJNZ 3811        juk az időtartam végsö
3814 23        INC  HL          értékét, végül az idö-
3815 47        LD   B,A         tartamot B-be töltjük

3816 F7 33     RST  30: 33      Hangképzés

3818 3A140B    LD   A,(0B14)    A folyamat vizsgálata:
381B 3C        INC  A           Még szöl?
381C 28FA      JR   Z,3818      Ha igen: várakozik, ami-
381E 18E3      JR   3803        kor pedig kész: ugrás
                                vissza, folytatja a kö-
                                vetkezö hanggal
```

A program addig dolgozik, míg *PITCH felsö byte-jaként 00-t olvas be.* Ekkor visszatér a *MONITOR*-ba. Az itt következö adatokkal ***Chr. SINDING** (1856-1941): TAVASZI ZSONGAS* címü csodálatos zongoradarabjának elsö néhány ütemét játsza le. A 00 érték beolvasására szervezett befejezés lehetövé teszi, hogy az adathalmaz tetszöleges hosszúságú darabját irjuk be -- a program müködöképes akkor is, ha akár egyetlenegy adat sincs. Ilyenkor természetesen hangot sem kapunk.

A dallamon kivül a program további érdekessége az, hogy a *lejátszás tempóját* elég tág határok között beállithatjuk a 380EH kezdöcimü utasitással.

Az adatok és a program beirása után végezzük el a szokásos ellenörzéseket és még az elsö kipröbálás elött vegyük fel egy kazettára! *A MONITOR S parancsához* begépelendö értékek: kezdöcim 3000, végcim 381E, a program neve -- pl. *ZENE1*.

```
3000  2F 0F 0E 02  2F 0F 04 02  50 0F 05 02  74 0F 06 02
3010  97 0F 07 02  74 0F 06 02  50 0F 05 02  2F 0F 04 02
3020  E8 0E 0A 02  2F 0F 04 02  50 0F 05 02  74 0F 06 02
3030  97 0F 07 02  74 0F 06 02  50 0F 05 02  2F 0F 0C 02
3040  15 0F 0B 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
3050  8A 0F 07 02  74 0F 06 02  50 0F 05 02  15 0F 04 02
3060  A0 0E 0A 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
3070  8A 0F 07 02  74 0F 06 02  50 0F 05 02  C6 0E 0A 02

3080  E8 0E 0A 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
3090  8A 0F 07 02  74 0F 06 02  50 0F 05 02  15 0F 04 02
30A0  15 0F 0C 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
30B0  8A 0F 07 02  74 0F 06 02  50 0F 05 02  15 0F 04 02
```

```
30C0   CB 04 07 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
30D0   A7 08 07 02  74 0F 06 02  7F 0A 08 02  15 0F 04 02
30E0   A1 0B 09 02  15 0F 04 02  53 0C 0A 02  74 0F 06 02
30F0   3F 0D 0A 02  74 0F 06 02  D1 0D 0A 02  15 0F 04 02

3100   2F 0F 0C 02  2F 0F 04 02  50 0F 05 02  74 0F 06 02
3110   97 0F 07 02  74 0F 06 02  50 0F 05 02  2F 0F 04 02
3120   E8 0E 0A 02  2F 0F 04 02  50 0F 05 02  74 0F 06 02
3130   97 0F 07 02  74 0F 06 02  50 0F 05 02  2F 0F 0C 02
3140   15 0F 0B 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
3150   8A 0F 07 02  74 0F 06 02  50 0F 05 02  15 0F 04 02
3160   A0 0E 0A 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
3170   8A 0F 07 02  74 0F 06 02  50 0F 05 02  C6 0E 0A 02

3180   E8 0E 0A 02  15 0F 04 02  45 0F 05 02  74 0F 06 02
3190   8A 0F 07 02  74 0F 06 02  45 0F 05 02  15 0F 04 02
31A0   15 0F 0C 02  15 0F 04 02  45 0F 05 02  74 0F 06 02
31B0   8A 0F 07 02  74 0F 06 02  45 0F 05 02  15 0F 04 02
31C0   43 07 07 02  15 0F 04 02  50 0F 05 02  74 0F 06 02
31D0   2B 0A 07 02  74 0F 06 02  A1 0B 08 02  15 0F 04 02
31E0   53 0C 09 02  45 0F 04 02  15 0D 0A 02  8A 0F 06 02
31F0   D1 0D 0A 02  8A 0F 06 02  2A 0E 0A 02  45 0F 04 02

3200   45 0F 0B 02  45 0F 04 02  74 0F 05 02  8A 0F 06 02
3210   A3 0F 07 02  8A 0F 06 02  74 0F 06 02  45 0F 05 02
3220   15 0F 0B 02  45 0F 04 02  74 0F 05 02  8A 0F 06 02
3230   A3 0F 07 02  8A 0F 06 02  74 0F 06 02  2F 0F 0B 02
3240   45 0F 0B 02  45 0F 04 02  6C 0F 05 02  8A 0F 06 02
3250   A3 0F 07 02  8A 0F 06 02  6C 0F 06 02  45 0F 05 02
3260   15 0D 05 02  67 0D 06 02  AF 0D 07 02  D1 0D 08 02
3270   0E 0E 09 02  2A 0E 0A 02  5D 0E 0B 02  8B 0E 0C 02

3280   45 0F 0B 02  45 0F 04 02  74 0F 05 02  8A 0F 06 02
3290   A3 0F 07 02  8A 0F 06 02  74 0F 06 02  45 0F 05 02
32A0   15 0F 0B 02  45 0F 04 02  74 0F 05 02  8A 0F 06 02
32B0   A3 0F 07 02  8A 0F 06 02  74 0F 06 02  2F 0F 0B 02
32C0   45 0F 0B 02  45 0F 04 02  6C 0F 05 02  8A 0F 06 02
32D0   A3 0F 07 02  8A 0F 06 02  6C 0F 06 02  45 0F 05 02
32E0   15 0D 05 02  67 0D 06 02  AF 0D 07 02  D1 0D 08 02
32F0   0E 0E 09 02  2A 0E 0A 02  5D 0E 0B 02  8B 0E 0C 02

3300   45 0F 0B 02  45 0F 04 02  74 0F 05 02  8A 0F 06 02
3310   A3 0F 07 02  8A 0F 06 02  74 0F 06 02  45 0F 05 02
3320   2F 0F 0B 02  45 0F 04 02  74 0F 05 02  8A 0F 06 02
3330   A3 0F 07 02  8A 0F 06 02  74 0E 06 02  15 0F 0B 02
3340   63 0F 0C 02  63 0F 06 02  8A 0F 07 02  A3 0F 08 02
3350   B2 0F 09 02  A3 0F 09 02  8A 0F 09 02  63 0F 09 02
3360   C6 0E 0F 02  63 0F 09 02  83 0F 0A 02  97 0F 0B 02
3370   B2 0F 0C 02  97 0F 0D 02  83 0F 0C 02  63 0F 0D 02
```

```
3380   15 0F 0D 02  63 0F 0D 02  8A 0F 0C 02  A3 0F 0C 02
3390   B2 0F 0C 02  A3 0F 0B 02  8A 0F 0A 02  C6 0E 0A 02
33A0   E8 0E 0A 02  45 0F 0A 02  74 0F 09 02  8A 0F 08 02
33B0   A3 0F 07 02  8A 0F 06 02  74 0F 05 02  8B 0E 0A 02
33C0   C6 0E 0A 02  15 0F 04 02  45 0F 04 02  63 0F 05 02
33D0   8A 0F 06 02  63 0F 05 02  45 0F 04 02  2A 0E 09 02
33E0   8B 0E 08 02  D8 0E 03 02  15 0F 04 02  45 0F 05 02
33F0   6C 0F 06 02  45 0F 05 02  15 0F 04 02  AF 0D 08 02

3400   D1 0D 07 02  8B 0E 03 02  E8 0E 04 02  15 0F 05 02
3410   45 0F 06 02  15 0F 05 02  E8 0E 04 02  8B 0E 03 02
3420   0E 0E 09 02  8B 0E 03 02  E8 0E 04 02  15 0F 05 02
3430   45 0F 06 02  15 0F 05 02  E8 0E 04 02  2A 0E 09 02
3440   8B 0E 0A 02  8B 0E 03 02  E8 0E 04 02  15 0F 05 02
3450   45 0F 06 02  15 0F 05 02  E8 0E 04 02  8B 0E 03 02
3460   D1 0D 0A 02  8B 0E 03 02  E8 0E 04 02  15 0F 05 02
3470   45 0F 06 02  0E 0E 09 02  2A 0E 09 02  5D 0E 09 02

3480   75 0E 0A 02  B3 0E 04 02  E8 0E 05 02  07 0F 06 02
3490   5A 0F 07 02  07 0F 06 02  5D 0E 0A 02  2A 0E 0A 02
34A0   E8 0E 0A 02  B3 0E 04 02  07 0F 05 02  3A 0F 06 02
34B0   5A 0F 07 02  3A 0F 06 02  07 0F 05 02  B3 0E 04 02
34C0   E8 0E 09 02  B3 0E 05 02  15 0F 06 02  3A 0F 07 02
34D0   5A 0F 08 02  3A 0F 08 02  15 0F 09 02  B3 0E 09 02
34E0   F8 0E 0D 02  B3 0E 09 02  22 0F 0A 02  45 0F 0B 02
34F0   5A 0F 0C 02  45 0F 0B 02  22 0F 0A 02  B3 0E 09 02

3500   0E 0E 0E 02  B3 0E 09 02  07 0F 0A 02  2F 0F 0B 02
3510   5A 0F 0C 02  2F 0F 0B 02  07 0F 0A 02  B3 0E 09 02
3520   44 0E 0E 02  B3 0E 09 02  07 0F 0A 02  2F 0F 0B 02
3530   5A 0F 0C 02  2F 0F 0B 02  07 0F 0A 02  5D 0E 0E 02
3540   B3 0E 0E 02  B3 0E 09 02  07 0F 0A 02  2F 0F 0B 02
3550   5A 0F 0C 02  2F 0F 0B 02  07 0F 0A 02  B3 0E 09 02
3560   0E 0E 0E 02  B3 0E 09 02  07 0F 0A 02  2F 0F 0B 02
3570   5A 0F 0C 02  44 0E 0E 02  5D 0E 0F 02  8B 0E 0F 02

3580   A0 0E 0F 02  D8 0E 09 02  07 0F 09 02  2F 0F 0A 02
3590   6C 0F 0B 02  2F 0F 0A 02  8B 0E 0F 02  5D 0E 0F 02
35A0   07 0F 0F 02  D8 0E 09 02  2F 0F 0A 02  50 0F 0B 02
35B0   6C 0F 0D 02  50 0F 0B 02  2F 0F 0A 02  D8 0E 09 02
35C0   07 0F 0F 02  D8 0E 09 02  2F 0F 0A 02  50 0F 0B 02
35D0   6C 0F 0D 02  50 0F 0B 02  2F 0F 0A 02  D8 0E 09 02
35E0   15 0F 0F 02  D8 0E 09 02  3A 0F 0A 02  5A 0F 0B 02
35F0   6C 0F 0D 02  5A 0F 0B 02  3A 0F 0A 02  D8 0E 09 02

3600   44 0E 0F 02  D8 0E 09 02  22 0F 0A 02  50 0F 0B 02
3610   6C 0F 0D 02  50 0F 0B 02  22 0F 0A 02  D8 0E 09 02
3620   75 0E 0F 02  D8 0E 09 02  22 0F 0A 02  50 0F 0B 02
3630   6C 0F 0D 02  50 0F 0B 02  22 0F 0A 02  A0 0E 0F 02
```

```
3640   D8 0E 0F 02  D8 0E 09 02  22 0F 0A 02  50 0F 0B 02
3650   6C 0F 0D 02  50 0F 0B 02  22 0F 0A 02  D8 0E 09 02
3660   44 0E 0F 02  D8 0E 09 02  22 0F 0A 02  50 0F 0B 02
3670   6C 0F 0D 02  50 0F 0B 02  22 0F 0A 02  D8 0E 09 02

3680   E8 0E 0F 02  E8 0E 09 02  15 0F 0A 02  50 0F 0B 02
3690   74 0F 0D 02  50 0F 0B 02  15 0F 0A 02  E8 0E 09 02
36A0   C6 0E 0F 02  E8 0E 09 02  15 0F 0A 02  50 0F 0B 02
36B0   74 0F 0D 02  50 0F 0B 02  15 0F 0A 02  E8 0E 09 02
36C0   A0 0E 0F 02  E8 0E 09 02  15 0F 0A 02  50 0F 0B 02
36D0   74 0F 0D 02  50 0F 0B 02  15 0F 0A 02  E8 0E 09 02
36E0   8B 0E 0F 02  E8 0E 09 02  15 0F 0A 02  50 0F 0B 02
36F0   74 0F 0D 02  50 0F 0B 02  15 0F 0A 02  5D 0E 0F 02

3700   2A 0E 0F 02  E8 0E 09 02  45 0F 0A 02  63 0F 0B 02
3710   8A 0F 0D 02  63 0F 0B 02  45 0F 0A 02  E8 0E 09 02
3720   0E 0E 0F 02  E8 0E 09 02  45 0F 0A 02  63 0F 0B 02
3730   8A 0F 0D 02  63 0F 0B 02  45 0F 0A 02  E8 0E 09 02
3740   F0 0D 0F 02  F8 0E 09 02  45 0F 0A 02  63 0F 0B 02
3750   8A 0F 0D 02  63 0F 0B 02  45 0F 0A 02  F8 0E 09 02
3760   D1 0D 0F 02  F8 0E 09 02  45 0F 0A 02  63 0F 0B 02
3770   8A 0F 0D 02  63 0F 0B 02  45 0F 0A 02  8C 0D 0F 02

3780   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
```

Szintetikusabb hangzást eredményező hangeffektus programját mutatjuk meg ezután. A gyorsabb működés érdekében, a funkcióhívás alkalmazása helyett, *közvetlenül a hanggenerátort programozzuk.*

PR19. Hangeffektus

Elsőként írjuk meg azt a szubrutint, amely *meghatározott magasságú hangot meghatározott ideig szólaltat meg.* Mint tudjuk, a hangmagasságot (PITCH) 2 byte-on kell megadni: a 0000...0FFFH tartományban. 0000 a legmélyebb hangot eredményezi, 0FFFH-hoz pedig már nem is hallhatóan magas hang tartozik.

A PITCH alsó byte-ját a 04-es portra kell írni, a felső byte négy bitjét pedig a 05-os port alsó felébe. Ugyanezen port 4. bitjét 1-be írva engedélyezhető, 0-val tiltható a hang megszólalása. A felső három bit más funkciókat lát el, ezért a *port* íráshoz fel kell használni a *0B12H* című memóriarekeszben tárolt *másolatot*, amely a *portra* írt korábbi adattal azonos.

Azért, hogy ne kelljen minden hang megszólaltatása előtt ezekkel a bitekkel az időt tölteni, a program elején egyszer ezt megtesszük, s a három bitet pl. a C regiszterben tároljuk. Ez a választás azért célszerű, mert *B-t érdemes ciklusszervezési célokra fenntartani, DE és HL pedig általános 16 bites regiszterpárként, számolási és címzési célokat szolgálhat.*

A hangot megszólaltató szubrutin tehát a C regiszterben kapja a *05-ös port felső négy bitjét*, s pl. a DE regiszterpárban a *PITCH* értéket.

A hang megszólaltatása a következő utasításokkal történik (most a begépeléshez javasolt memóriacímeket és az utasításkódokat is tartalmazó, teljes disassemblált listát adjuk meg):

```
4000 7B        LD   A,E        PITCH, alsó byte
4001 D304      OUT  (04),A
4003 79        LD   A,C        A C felső négy bitjéhez
4004 E6F0      AND  F0         hozzáírjuk a PITCH bit-
4006 B2        OR   D          jeit
4007 D305      OUT  (05),A
4009 3E10      LD   A,10       Várakozás, kezdőérték
400B 3D        DEC  A          Visszaszámlálás, ismét-
400C 20FD      JR   NZ,400B    lés A = 00-ig
400E C9        RET
```

Ha az Olvasó az így elérhető várakozást *rövidnek* tartaná, utalunk arra, hogy most nem egyetlen hang kitartó megszólaltatása a célunk, ellenkezőleg: *igen gyorsan változó hangokkal, összetettebb hangeffektusokat akarunk előállítani.*

Tervezzük meg a főprogramot! Ezt írjuk meg úgy, hogy egymást követő *két fázisban* a következő funkciókra legyen képes.

Adjunk meg egy *PITCH sávszélességet*, amelyen belül a hangmagasság változhat. Ezt -- mint egy *ablakot* -- a teljes PITCH tartomány bármely pontjára helyezve, ezen belül változhat a hangmagasság. Ha pl. az ablak mérete: 0100H és az alját a 0000...0FFFH tartományon belül pl. 082BH-ra helyezzük, akkor a hangképzéshez használt PITCH értékek 0B2BH és 0C2BH között változhatnak.
Az ablak méretét tároljuk HL-ben!

Az egyszerűség kedvéért az egymás utáni PITCH értékeket mindig növekvő irányban és mindig ugyanannyival változtassuk meg. A PITCH lépésköz megadására használjuk a *másodlagos regiszterkészletben a DE' regiszterpárt.*

A hangeffektus előállításában így megadott ablakot *mozgatni* fogjuk. Véges idejű működéshez, pl. a B regiszterben adjuk meg az ablak mozgatásainak számát (*periódusszám*).

Az ablak minden egyes helyzetében futtassuk végig PITCH értékét a legalsó határtól a legfelsőig, a lépésközzel meghatározott módon -- ezután mozgassuk el az ablakot *1* értékkel lefelé!

Az így képezett hangok képezik majd a hangeffektus második fázisát.

Az *első fázis* pedig: tulajdonképpen *egy speciális bevezetés*.

(a) Helyezzük az előbbi ablakot a teljes *PITCH* értéktartomány legfelsö részére úgy, hogy teteje OFFOH legyen (még éppen hallható hang).

(b) A *PITCH* értékét állítsuk elöször ugyanide.

(c) Szólaltassuk meg a hangot, majd növeljük meg az adott lépésközzel. Ezt ismételjük addig, amíg ki nem lépünk az ablakból, vagyis a OFFOH érték fölé nem kerülünk!

(d) Ekkor *PITCH* értékét állítsuk az előbbi indulö érték alá 1-gyel és ezzel ismételjük meg az eljárást! Látható, hogy így *egyre szélesebb sávot fogunk át, s egyre mélyülö kezdöértékü, felfelé menö futamokat kapunk*. Ha *PITCH* indulö értéke eléri az ablak alját, akkor ér véget az elsö, bevezetö fázis.

Már elöre megjegyezzük, hogy a lépésközt, az ablak méretét, és az ablak mozgatásainak számát alkalmas értékre beállítva: a perceken át szirénázó mentöautótól, az ürhajók kisérteties hangjain át egy-két furcsa hatású koppanásig sok mindent elöállíthatunk.

Az igen gazdagon variálható lehetöségek közül néhányat a program végén, egy táblázatban felsorolunk.

Írjuk be a program fórészét a 4010H címtöl:

```
4010 D9          EXX
4011 11000C      LD   DE,0001      A program paramétereinek
4014 D9          EXX               elökészítése: a másodla-
                                   gos regiszterkészletben
                                   DE'= a lépésköz,
4015 21000C      LD   HL,0C00      HL-be az ablak mérete,
4018 0601        LD   B,01         B-be a periódusszám
401A 3A130B      LD   A,(0B13)     A 06-os port másolata.
                                   Ennek b5...b2 bitjein
                                   kell beállítani a hang-
                                   erőt
```

```
401D E6C3       AND  C3          Az alsó és felsö két bit
401F F63C       OR   3C          marad, a középsö négy
4021 D306       OUT  (06),A      bit 1111, ami 15-ös
                                 hangerőt jelent
4023 3A120B     LD   A,(0B12)    A 05-ös port másolata
4026 E6C0       AND  C0          A felsö három bit marad,
4028 F610       OR   10          b4-re 1-et írva engedé-
402A 4F         LD   C,A         délyezzük a hangot. Eze-
                                 ket tároljuk C-ben
```

Ezután kezdödik a bevezetö fázis:

```
402B 11F00F     LD   DE,0FF0     Az elsö PITCH érték

402E D5         PUSH DE          Az aktuális kezdöérték

402F CD0040     CALL 4000        Egy hang megszólaltatása

4032 D5         PUSH DE          Az aktuális PITCH érté-
4033 D9         EXX              ket a másodlagos kész-
4034 E1         POP  HL          letben HL'-be másoljuk
4035 19         ADD  HL,DE       és megnöveljük a lépés-
4036 E5         PUSH HL          közzel, majd elökészít-
                                 jük a visszatöltést.
4037 D5         PUSH DE          Ha túlléptünk a 0FFFH-n,
4038 110FF0     LD   DE,F00F     akkor ezt 0F00FH-hoz
403B 19         ADD  HL,DE       hozzáadva: CY = 1 lesz
403C D1         POP  DE          (DE'-t visszaállítjuk)
403D D9         EXX              A lépésközzel megnövelt
403E D1         POP  DE          értéket a veremböl át-
                                 töltjük DE-be, s ha még
403F 30EE       JR   NC,3D2F     nincs túllépés: ugrás
                                 vissza!
```

Ezzel tehát egy felfutó hangsort képeztünk az indulö *PITCH*-tól a legnagyobb értékig. Ha ezt elértük: következik az indulö érték átállítása, 1-gyel kisebbre:

```
4041 D1         POP  DE          Az indulö érték
4042 1B         DEC  DE          Ennek csökkentése
```

Ellenörizzük, hogy elértük-e már az ablak alját:

```
4043 E5         PUSH HL          Az ablak mérete
4044 D5         PUSH DE          Az új indulö érték
4045 EB         EX   DE,HL       Az ablakméret DE-ben
4046 21F00F     LD   HL,0FF0     A legnagyobb értékböl az
4049 ED52       SBC  HL,DE       ablak méretét kivonva,
                                 az aljához tartozó PITCH
                                 értéket kapjuk
404B D1         POP  DE          Ismét az új indulö érték
```

```
404C ED52      SBC  HL,DE      Ha elértük az ablak al-
                               ját: az eredmény 0 és
                               Z = 1
404E E1        POP  HL        Ismét az ablak mérete
404F 20DD      JR   NZ,3D2E   Ha még nem értük el az
                               alját, ugrás vissza: új
                               hangfutam következik
```

Írjuk meg a második fázis programját!

Elértük az ablak alját, most már magát az ablakot kell lefelé mozgatni, s minden helyzetében egy-egy hangsort futtatunk az aljától a tetejéig.

```
4051 E5        PUSH HL        Az ablak mérete
4052 D5        PUSH DE        Az utolsó induló PITCH
                               érték: itt lesz az ablak
                               alja
4053 19        ADD  HL,DE     HL: az ablak teteje

4054 CD0040    CALL 4000      Egy hang megszólaltatása

4057 D5        PUSH DE        Az aktuális PITCH érté-
4058 D9        EXX            ket a másodlagos kész-
4059 E1        POP  HL        letben HL'-be másoljuk
405A 19        ADD  HL,DE     és megnöveljük a lépés-
405B E5        PUSH HL        közzel. Ezután a másod-
405C D9        EXX            lagos regiszterkészlet-
405D D1        POP  DE        ből visszatöltjük DE-be
```

Ellenőrizni kell, hogy elértük-e már az ablak tetejét:

```
405E E5        PUSH HL        Az ablak teteje
405F B7        OR   A         CY = 0
4060 ED52      SBC  HL,DE     Ha az aktuális PITCH ér-
                               ték nagyobb, akkor kivo-
                               nás után: CY = 1
4062 E1        POP  HL        Ismét az ablak teteje
4063 30EF      JR   NC,3D54   Ha még alatta vagyunk:
                               mehet tovább -- új hang!
```

Amikor elértük az ablak tetejét, akkor egy futam kész -- elmozgatjuk 1-gyel, lefelé:

```
4065 D1        POP  DE        A régi induló érték, az
4066 1B        DEC  DE        ablak alja. Csökkentjük
4067 E1        POP  HL        Az ablak mérete
4068 10E7      DJNZ 3D51      Ha még nem volt előírt
                               számú mozgatás, ugrás
                               vissza: új futamot kez-
                               dünk
```

Végül lezárjuk a hangképzést:

```
406A 79          LD   A,C          A 05-ös port tárolt bit-
406B E6EF        AND  EF           jei: a 4. bitet törölve
406D D305        OUT  (05),A       tiltjuk a hangot és
406F CD0F27      CALL 270F         visszatérés a MONITOR-ba
```

Írjuk be a teljes programot és vegyük fel ezt is egy kazettára!

A különböző hangeffektusok beállítása a következő paraméterek megváltoztatásával történhet:

-- ***egy hang időtartama:*** a 4000H kezdőcímű szubrutinban, a 400AH címen az A-ba töltendő érték átírása;

-- ***lépésköz:*** a főprogramban DE' tartalmának átírása, a 4012--4013H címen;

-- ***az ablak mérete:*** a HL-be töltendő érték átírása, a 4016--4017H memóriacímen;

-- ***periódusszám:*** az ablak mozgatásainak számát a 4019H cím átírásával állíthatjuk be.

Futtassuk le a programot -- kedvcsinálásként -- a következő beállítási értékekkel:

Időtartam *(400A)*	**Lépésköz** *(4012,3)*	**Ablakméret** *(4016,7)*	**Periódusszám** *(4019)*
01	0001	0001	01
01	0001	0001	00
00	0001	0001	00
00	0001	0010	00
20	0001	0010	00
20	0001	0040	00
10	0001	0080	00
01	0001	0400	00
01	0010	0400	00
01	0020	0400	00
01	0080	0400	00
01	0100	0400	00
02	0100	0400	00
10	0100	0400	00
80	0100	0400	00
20	0040	0800	00
20	0080	0C00	00
20	0400	0B00	00
20	0400	0B00	01
20	0800	0B00	01

A periódusszámra és az időtartamra írt 00 érték természetesen 256-ot jelent a CPU számára -- ez a csökkentésre épített ismétlések szervezésének módjából adódik. Az első csökkentés után a 00 érték 0FFH-ra változik, s ezután még 255-ször kell lefutni a megfelelő ciklusnak ahhoz, hogy a CPU előírt regiszterében ismét 00 érték álljon elő.

Az előbbi program utasításait, de főleg a struktúráját elemezve jól érezhető az, amit e témakör bevezetőjében mondtam. *A hanképzéssel kapcsolatos problémák nem kifejezetten a programozás szintjén jelentkeznek.* Az a furcsa helyzet állt elő, hogy még a program megírása után, még működésének ismeretében is meglehetősen nehéz előre megmondani, hogy milyen hangeffektus lesz az eredmény.

A grafikus programokkal összehasonlítva a különbség figyelemreméltó. A vizuális hatások, a képernyőn megjelenő látvány elemei pontosan megtervezhetők -- *az akusztikus hatások, a hangelemekből összeálló teljes hangkép ezzel szemben előre csak nagyon hozzávetőlegesen jósolható meg.* Másképpen fogalmazva: a képernyőgrafika és az azt megvalósító *számítógépes program mozgásszintje* azonos, kapcsolatuk közvetlen -- *a hangeffektusok esetén* ugyanakkor a két szint *nem esik egybe.*

Egyelőre ennyit a hanképzésről. Mitöbb, munkánkat itt megszakítjuk, könyvünknek is a végére értünk.

A következő oldalakon olyan összefoglaló táblázatokat közlünk, amelyek bizonyára nagy segítséget jelentenek az önálló programozásban. *Ettől kezdve az Olvasó elsősorban a saját munkájára lesz utalva,* s a **Függelékben** közölt információk ehhez alig nélkülözhetők.

5. FÜGGELÉK

5.1 A Z80-as CPU

5.1.1 Regiszterek

8 bites, általános regiszterek

Aktuális		Másodlagos	
Akkumulátor A	Jelzőbitek F	Akkumulátor A'	Jelzőbitek F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

Speciális regiszterek

8 bites:	megszakításvektor	I
	memóriafrissítő	R
16 bites:	indexregiszter	IX
	indexregiszter	IY
	veremmutató	SP
	programszámláló	PC

5.1.2 Jelzőbitek

F regiszter

b7	b6	b5	b4	b3	b2	b1	b0
S	Z	-	H		P/V	N	C

A jelzőbitek működése

Utasítás	S	Z	H	P/V	N	C
8 bites ADD,ADC,SUB,SBC,CP, NEG	F	F	F	F	F	F
AND	F	F	1	F	0	0
OR, XOR	F	F	0	F	0	0
8 bites INC,DEC	F	F	F	F	F	=
16 bites ADD	=	=	x	=	0	F
16 bites ADC,SBC	F	F	x	F	F	F
RLA,RLCA,RRA,RRCA	=	=	0	=	0	F
RL,RLC,RR,RRC,SLA,SRA,SRL	F	F	0	F	0	F
RLD,RRD	F	F	0	F	0	=
DAA	F	F	F	F	=	F
CPL	=	=	1	=	1	=
SCF	=	=	0	=	0	1
CCF	=	=	x	=	0	F
IN a (C) portról	F	F	0	F	0	=
INI,IND,OUTI,OUTD,INIR, INDR,OTIR,OTDR (*1)	x	F	x	x	1	=
LDI,LDD,LDIR,LDDR (*2)	x	x	0	F	0	=
CPI,CPD,CPIR,CPDR (*3)	F	F	x	F	1	=
LD A,I; LD A,R (*4)	F	F	0	F	0	=
BIT (*5)	x	F	1	x	0	=

Jelmagyarázat:

F a jelzőbit a művelet eredménye szerint áll be
= változatlan
1 a jelzőbit értéke 1 lesz
0 a jelzőbit 0 lesz
x érdektelen

(*1) Z = 1, amikor B = 00
(*2) P/V = 0, amikor BC = 0000
(*3) Z = 1, ha A = (HL), ezenkívül P/V = 0, amikor BC =0000
(*4) P/V a megszakítást engedélyező *flip-flop* tartalmát veszi át
(*5) Z a vizsgált bit ellentettjét mutatja

5.1.3 Utasítások

Mnemonik	Kód
ADC A,(HL)	8E
ADC A,(IX+ee)	DD8Eee
ADC A,(IY+ee)	FD8Eee
ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D
ADC A,dd	CEdd
ADC HL,BC	ED4A
ADC HL,DE	ED5A
ADC HL,HL	ED6A
ADC HL,SP	ED7A
ADD A,(HL)	86
ADD A,(IX+ee)	DD86ee
ADD A,(IY+ee)	FD86ee
ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85
ADD A,dd	C6dd
ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29
ADD HL,SP	39
ADD IX,BC	DD09
ADD IX,DE	DD19
ADD IX,IX	DD29
ADD IX,SP	DD39
ADD IY,BC	FD09
ADD IY,DE	FD19
ADD IY,IY	FD29
ADD IY,SP	FD39
AND (HL)	A6
AND (IX+ee)	DDA6ee
AND (IY+ee)	FDA6ee

Mnemonik	Kód
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND dd	E6dd
BIT 0,(HL)	CB46
BIT 0,(IX+ee)	DDCBee46
BIT 0,(IY+ee)	FDCBee46
BIT 0,A	CB47
BIT 0,B	CB40
BIT 0,C	CB41
BIT 0,D	CB42
BIT 0,E	CB43
BIT 0,H	CB44
BIT 0,L	CB45
BIT 1,(HL)	CB4E
BIT 1,(IX+ee)	DDCBee4E
BIT 1,(IY+ee)	FDCBee4E
BIT 1,A	CB4F
BIT 1,B	CB48
BIT 1,C	CB49
BIT 1,D	CB4A
BIT 1,E	CB4B
BIT 1,H	CB4C
BIT 1,L	CB4D
BIT 2,(HL)	CB56
BIT 2,(IX+ee)	DDCBee56
BIT 2,(IY+ee)	FDCBee56
BIT 2,A	CB57
BIT 2,B	CB50
BIT 2,C	CB51
BIT 2,D	CB52
BIT 2,E	CB53
BIT 2,H	CB54
BIT 2,L	CB54
BIT 3,(HL)	CB5E
BIT 3,(IX+ee)	DDCBee5E
BIT 3,(IY+ee)	FDCBee5E
BIT 3,A	CB5F
BIT 3,B	CB58

Utasítások (folytatás)

Mnemonik	Kód
BIT 3,C	CB59
BIT 3,D	CB5A
BIT 3,E	CB5B
BIT 3,H	CB5C
BIT 3,L	CB5D
BIT 4,(HL)	CB66
BIT 4,(IX+ee)	DDCBee66
BIT 4,(IY+ee)	FDCBee66
BIT 4,A	CB67
BIT 4,B	CB60
BIT 4,C	CB61
BIT 4,D	CB62
BIT 4,E	CB63
BIT 4,H	CB64
BIT 4,L	CB65
BIT 5,(HL)	CB6E
BIT 5,(IX+ee)	DDCBee6E
BIT 5,(IY+ee)	FDCBee6E
BIT 5,A	CB6F
BIT 5,B	CB68
BIT 5,C	CB69
BIT 5,D	CB6A
BIT 5,E	CB6B
BIT 5,H	CB6C
BIT 5,L	CB6D
BIT 6,(HL)	CB76
BIT 6,(IX+ee)	DDCBee76
BIT 6,(IY+ee)	FDCBee76
BIT 6,A	CB77
BIT 6,B	CB70
BIT 6,C	CB71
BIT 6,D	CB72
BIT 6,E	CB73
BIT 6,H	CB74
BIT 6,L	CB75
BIT 7,(HL)	CB7E
BIT 7,(IX+ee)	DDCBee7E
BIT 7,(IY+EE)	FDCBee7E
BIT 7,A	CB7F
BIT 7,B	CB78
BIT 7,C	CB79
BIT 7,D	CB7A
BIT 7,E	CB7B
BIT 7,H	CB7C

Mnemonik	Kód
BIT 7,L	CB7D
CALL C,nnnn	DCnnnn
CALL M,nnnn	FCnnnn
CALL NC,nnnn	D4nnnn
CALL NZ,nnnn	C4nnnn
CALL P,nnnn	F4nnnn
CALL PE,nnnn	ECnnnn
CALL PO,nnnn	E4nnnn
CALL Z,nnnn	CCnnnn
CALL nnnn	CDnnnn
CCF	3F
CP (HL)	BE
CP (IX+ee)	DDBEee
CP (IY+ee)	FDBEee
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP dd	FEdd
CPD	EDA9
CPDR	EDB9
CPI	EDA1
CPIR	EDB1
CPL	2F
DAA	27
DEC (HL)	35
DEC (IX+ee)	DD35ee
DEC (IY+ee)	FD35ee
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B

Utasítások (folytatás)

Mnemonik	Kód
DEC IX	DD2B
DEC IY	FD2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ ee	10ee
EI	FB
EX (SP),HL	E3
EX (SP),IX	DDE3
EX (SP),IY	FDE3
EX AF,AF'	08
EX DE,HL	EB
EXX	D9
HALT	76
IM 0	ED46
IM 1	ED56
IM 2	ED5E
IN A,(C)	ED78
IN B,(C)	ED40
IN C,(C)	ED48
IN D,(C)	ED50
IN E,(C)	ED58
IN H,(C)	ED60
IN L,(C)	ED68
INC (HL)	34
INC (IX+ee)	DD34ee
INC (IY+ee)	FD34ee
INC A	3C
INC B	04
INC BC	03
INC C	0C
INC D	14
INC DE	13
INC E	1C
INC H	24
INC HL	23
INC IX	DD23

Mnemonik	Kód
INC IY	FD23
INC L	2C
INC SP	33
IN A,(dd)	DBdd
IND	EDAA
INDR	EDBA
INI	EDA2
INIR	EDB2
JP nnnn	C3nnnn
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9
JP C,nnnn	DAnnnn
JP M,nnnn	FAnnnn
JP NC,nnnn	D2nnnn
JP NZ,nnnn	C2nnnn
JP P,nnnn	F2nnnn
JP PE,nnnn	EAnnnn
JP PA,nnnn	E2nnnn
JP Z,nnnn	CAnnnn
JR C,ee	38ee
JR NC,ee	30ee
JR NZ,ee	20ee
JR Z,ee	28ee
JR ee	18ee
LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75
LD (HL),dd	36dd
LD (IX+ee),A	DD77ee
LD (IX+ee),B	DD70ee
LD (IX+ee),C	DD71ee
LD (IX+ee),D	DD72ee
LD (IX+ee),E	FF73ee

Utasítások (folytatás)

Mnemonik		Kód
LD	(IX+ee),H	DD74ee
LD	(IX+ee),L	DD75ee
LD	(IX+ee),dd	DD36eedd
LD	(IY+ee),A	FD77ee
LD	(IY+ee),B	FD70ee
LD	(IY+ee),C	FD71ee
LD	(IY+ee),D	FD72ee
LD	(IY+ee),E	FD73ee
LD	(IY+ee),H	FD74ee
LD	(IY+ee),L	FD75ee
LD	(IY+ee),dd	FD36eedd
LD	(nnnn),A	32nnnn
LD	(nnnn),BC	ED43nnnn
LD	(nnnn),DE	ED53nnnn
LD	(nnnn),HL	22nnnn
LD	(nnnn),IX	DD22nnnn
LD	(nnnn),IY	FD22nnnn
LD	(nnnn),SP	ED73nnnn
LD	A,(BC)	0A
LD	A,(DE)	1A
LD	A,(HL)	7E
LD	A,(IX+ee)	DD7Eee
LD	A,(IY+ee)	FD7Eee
LD	A,(nnnn)	3Annnn
LD	A,A	7F
LD	A,B	78
LD	A,C	79
LD	A,D	7A
LD	A,E	7B
LD	A,H	7C
LD	A,I	ED57
LD	A,L	7D
LD	A,dd	3Edd
LD	A,R	ED5F
LD	B,(HL)	46
LD	B,(IX+ee)	DD46ee
LD	B,(IY+ee)	FD46ee
LD	B,A	47
LD	B,B	40
LD	B,C	41
LD	B,D	42
LD	B,E	43
LD	B,H	44
LD	B,L	45

Mnemonik		Kód
LD	B,dd	06dd
LD	BC,(nnnn)	ED4Bnnnn
LD	BC,nnnn	01nnnn
LD	C,(HL)	4E
LD	C,(IX+ee)	DD4Eee
LD	C,(IY+ee)	FD4Eee
LD	C,A	4F
LD	C,B	48
LD	C,C	49
LD	C,D	4A
LD	C,E	4B
LD	C,H	4C
LD	C,L	4D
LD	C,dd	0Edd
LD	D,(HL)	56
LD	D,(IX+ee)	DD56ee
LD	D,(IY+ee)	FD56ee
LD	D,A	57
LD	D,B	50
LD	D,C	51
LD	D,D	52
LD	D,E	53
LD	D,H	54
LD	D,L	55
LD	D,dd	16dd
LD	DE,(nnnn)	ED5Bnnnn
LD	DE,nnnn	11nnnn
LD	E,(HL)	5E
LD	E,(IX+ee)	DD5Eee
LD	E,(IY+ee)	FD5Eee
LD	E,A	5F
LD	E,B	58
LD	E,C	59
LD	E,D	5A
LD	E,E	5B
LD	E,H	5C
LD	E,L	5D
LD	E,dd	1Edd
LD	H,(HL)	66
LD	H,(IX+ee)	DD66ee
LD	H,(IY+ee)	FD66ee
LD	H,A	67

Utasítások (folytatás)

Mnemonik		Kód
LD	H,B	60
LD	H,C	61
LD	H,D	62
LD	H,E	63
LD	H,H	64
LD	H,L	65
LD	H,dd	26dd
LD	HL,(nnnn)	2Annnn
LD	HL,nnnn	21nnnn
LD	I,A	ED47
LD	IX,(nnnn)	DD2Annnn
LD	IX,nnnn	DD21nnnn
LD	IY,(nnnn)	FD2Annnn
LD	IY,nnnn	FD21nnnn
LD	L,(HL)	6E
LD	L,(IX+ee)	DD6Eee
LD	L,(IY+ee)	FD6Eee
LD	L,A	6F
LD	L,B	68
LD	L,C	69
LD	L,D	6A
LD	L,E	6B
LD	L,H	6C
LD	L,L	6D
LD	L,dd	2Edd
LD	R,A	ED4F
LD	SP,(nnnn)	ED7Bnnnn
LD	SP,HL	F9
LD	SP,IX	DDF9
LD	SP,IY	FDF9
LD	SP,nnnn	31nnnn
LDD		EDA8
LDDR		EDB8
LDI		EDA0
LDIR		EDB0
NEG		ED44
NOP		00

Mnemonik		Kód
OR	(HL)	B6
OR	(IX+ee)	DDB6ee
OR	(IY+ee)	FDB6ee
OR	A	B7
OR	B	B0
OR	C	B1
OR	D	B2
OR	E	B3
OR	H	B4
OR	L	B5
OR	dd	F6dd
OTDR		ED8B
OTIR		EDB3
OUT	(C),A	ED79
OUT	(C),B	ED41
OUT	(C),C	ED49
OUT	(C),D	ED51
OUT	(C),E	ED59
OUT	(C).H	ED61
OUT	(C),L	ED69
OUT	(dd),A	D3dd
OUTD		EDAB
OUTI		EDA3
POP	AF	F1
POP	BC	C1
POP	DE	D1
POP	HL	E1
POP	IX	DDE1
POP	IY	FDE1
PUSH	AF	F5
PUSH	BC	C5
PUSH	DE	D5
PUSH	HL	E5
PUSH	IX	DDE5
PUSH	IY	FDE5
RES	0,(HL)	CB86
RES	0,(IX+ee)	DDCBee86
RES	0,(IY+ee)	FDCBee86
RES	0,A	CB87
RES	0,B	CB80
RES	0,C	CB81

Utasítások (folytatás)

Mnemonik	Kód
RES 0,D	CB82
RES 0,E	CB83
RES 0,H	CB84
RES 0,L	CB85
RES 1,(HL)	CB8E
RES 1,(IX+ee)	DDCBee8E
RES 1,(IY+ee)	FDCBee8E
RES 1,A	CB8F
RES 1,B	CB88
RES 1,C	CB89
RES 1,D	CB8A
RES 1,E	CB8B
RES 1,H	CB8C
RES 1,L	CB8D
RES 2,(HL)	CB96
RES 2,(IX+ee)	DDCBee96
RES 2,(IY+ee)	FDCBee96
RES 2,A	CB97
RES 2,B	CB90
RES 2,C	CB91
RES 2,D	CB92
RES 2,E	CB93
RES 2,H	CB94
RES 2,L	CB95
RES 3,(HL)	CB9E
RES 3,(IX+ee)	DDCBee9E
RES 3,(IY+ee)	FDCBee9E
RES 3,A	CB9F
RES 3,B	CB98
RES 3,C	CB99
RES 3,D	CB9A
RES 3,E	CB9B
RES 3,H	CB9C
RES 3,L	CB9D
RES 4,(HL)	CBA6
RES 4,(IX+ee)	DDCBeeA6
RES 4,(IY+ee)	FDCBeeA6
RES 4,A	CBA7
RES 4,B	CBA0
RES 4,C	CBA1
RES 4,D	CBA2
RES 4,E	CBA3
RES 4,H	CBA4
RES 4,L	CBA5

Mnemonik	Kód
RES 5,(HL)	CBAE
RES 5,(IX+ee)	DDCBeeAE
RES 5,(IY+ee)	FDCBeeAE
RES 5,A	CBAF
RES 5,B	CBA8
RES 5,C	CBA9
RES 5,D	CBAA
RES 5,E	CBAB
RES 5,H	CBAC
RES 5,L	CBAD
RES 6,(HL)	CBB6
RES 6,(IX+ee)	DDCBeeB6
RES 6,(IY+ee)	FDCBeeB6
RES 6,A	CBB7
RES 6,B	CBB0
RES 6,C	CBB1
RES 6,D	CBB2
RES 6,E	CBB3
RES 6,H	CBB4
RES 6,L	CBB5
RES 7,(HL)	CBBE
RES 7,(IX+ee)	DDCBeeBE
RES 7,(IY+ee)	FDCBeeBE
RES 7,A	CBBF
RES 7,B	CBB8
RES 7,C	CBB9
RES 7,D	CBBA
RES 7,E	CBBB
RES 7,H	CBBC
RES 7,L	CBBD
RET	C9
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET PO	E0
RET Z	C8
RETI	ED4D
RETN	ED45

Utasítások (folytatás)

Mnemonik		Kód
RL	(HL)	CB16
RL	(IX+ee)	DDCBee16
RL	(IY+ee)	FDCBee16
RL	A	CB17
RL	B	CB10
RL	C	CB11
RL	D	CB12
RL	E	CB13
RL	H	CB14
RL	L	CB15
RLA		17
RLC	(HL)	CB06
RLC	(IX+ee)	DDCBee06
RLC	(IY+ee)	FDCBee06
RLC	A	CB07
RLC	B	CB00
RLC	C	CB01
RLC	D	CB02
RLC	E	CB03
RLC	H	CB04
RLC	L	CB05
RLCA		07
RLD		ED6F
RR	(HL)	CB1E
RR	(IX+ee)	DDCBee1E
RR	(IY+ee)	FDCBee1E
RR	A	CB1F
RR	B	CB18
RR	C	CB19
RR	D	CB1A
RR	E	CB1B
RR	H	CB1C
RR	L	CB1D
RRA		1F
RRC	(HL)	CB0E
RRC	(IX+ee)	DDCBee0E
RRC	(IY+ee)	FDCBee0E
RRC	A	CB0F
RRC	B	CB08
RRC	C	CB09
RRC	D	CB0A

Mnemonik		Kód
RRC	E	CB0B
RRC	H	CB0C
RRC	L	CB0D
RRCA		0F
RRD		ED67
RST	00	C7
RST	08	CF
RST	10	D7
RST	18	DF
RST	20	E7
RST	28	EF
RST	30	F7
RST	38	FF
SBC	A,(HL)	9E
SBC	A,(IX+ee)	DD9Eee
SBC	A,(IY+ee)	FD9Eee
SBC	A,A	9F
SBC	A,B	98
SBC	A,C	99
SBC	A,D	9A
SBC	A,E	9B
SBC	A,H	9C
SBC	A,L	9D
SBC	A,dd	DEdd
SBC	HL,BC	ED42
SBC	HL,DE	ED52
SBC	HL,HL	ED62
SBC	HL,SP	ED72
SCF		37
SET	0,(HL)	CBC6
SET	0,(IX+ee)	DDCBeeC6
SET	0,(IY+ee)	FDCBeeC6
SET	0,A	CBC7
SET	0,B	CBC0
SET	0,C	CBC1
SET	0,D	CBC2
SET	0,E	CBC3
SET	0,H	CBC4
SET	0,L	CBC5

Utasítások (folytatás)

Mnemonik	Kód
SET 1,(HL)	CBCE
SET 1,(IX+ee)	DDCBeeCE
SET 1,(IY+ee)	FDCBeeCE
SET 1,A	CBCF
SET 1,B	CBC8
SET 1,C	CBC9
SET 1,D	CBCA
SET 1,E	CBCB
SET 1,H	CBCC
SET 1,L	CBCD
SET 2,(HL)	CBD6
SET 2,(IX+ee)	DDCBeeD6
SET 2,(IY+ee)	FDCBeeD6
SET 2,A	CBD7
SET 2,B	CBD0
SET 2,C	CBD1
SET 2,D	CBD2
SET 2,E	CBD3
SET 2,H	CBD4
SET 2,L	CBD5
SET 3,(HL)	CBDE
SET 3,(IX+ee)	DDCBeeDE
SET 3,(IY+ee)	FDCBeeDE
SET 3,A	CBDF
SET 3,B	CBD8
SET 3,C	CBD9
SET 3,D	CBDA
SET 3,E	CBDB
SET 3,H	CBDC
SET 3,L	CBDD
SET 4,(HL)	CBE6
SET 4,(IX+ee)	DDCBeeE6
SET 4,(IY+ee)	FDCBeeE6
SET 4,A	CBE7
SET 4,B	CBE0
SET 4,C	CBE1
SET 4,D	CBE2
SET 4,E	CBE3
SET 4,H	CBE4
SET 4,L	CBE5
SET 5,(HL)	CBEE
SET 5,(IX+ee)	DDCBeeEE
SET 5,(IY+ee)	FDCBeeEE
SET 5,A	CBEF

Mnemonik	Kód
SET 5,B	CBE8
SET 5,C	CBE9
SET 5,D	CBEA
SET 5,E	CBEB
SET 5,H	CBEC
SET 5,L	CBED
SET 6,(HL)	C8F6
SET 6,(IX+ee)	DDCBeeF6
SET 6,(IY+ee)	FDCBeeF6
SET 6,A	CBF7
SET 6,B	CBF0
SET 6,C	CBF1
SET 6,D	CBF2
SET 6,E	CBF3
SET 6,H	CBF4
SET 6,L	CBF5
SET 7,(HL)	CBFE
SET 7,(IX+ee)	DDCBeeFE
SET 7,(IY+ee)	FDCBeeFE
SET 7,A	CBFF
SET 7,B	CBF8
SET 7,C	CBF9
SET 7,D	CBFA
SET 7,E	CBFB
SET 7,H	CBFC
SET 7,L	CBFD
SLA (HL)	CB26
SLA (IX+ee)	DDCBee26
SLA (IY+ee)	FDCBee26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX+ee)	DDCBee2E
SRA (IY+ee)	FDCBee2E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B

Utasítások (folytatás)

Mnemonik	Kód
SRA H	CB2C
SRA L	CB2D
SRL (HL)	CB3E
SRL (IX+ee)	DDCBee3E
SRL (IY+ee)	FDCBee3E
SRL A	CB3F
SRL B	CB38
SRL C	CB39
SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX+ee)	DD96ee
SUB (IY+ee)	FD96ee
SUB A	97
SUB B	90

Mnemonik	Kód
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB dd	D6dd
XOR (HL)	AE
XOR (IX+ee)	DDAEee
XOR (IY+ee)	FDAEee
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR dd	EEee

5.2 Hexadecimális számok

Hexadecimális számjegyek							
4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4 096	1	256	1	16	1	1
2	8 192	2	512	2	32	2	2
3	12 288	3	768	3	48	3	3
4	16 384	4	1 024	4	64	4	4
5	20 480	5	1 280	5	80	5	5
6	24 576	6	1 536	6	96	6	6
7	28 672	7	1 792	7	112	7	7
8	32 768	8	2 048	8	128	8	8
9	36 864	9	2 304	9	144	9	9
A	40 960	A	2 560	A	160	A	10
B	45 056	B	2 816	B	176	B	11
C	49 152	C	3 072	C	192	C	12
D	53 248	D	3 328	D	208	D	13
E	57 344	E	3 584	E	224	E	14
F	61 440	F	3 840	F	240	F	15
b7 b6 b5 b4		*b3 b2 b1 b0*		*b7 b6 b5 b4*		*b3 b2 b1 b0*	
Magasabb helyi értékű byte				*Alacsonyabb helyi értékű byte*			

5.3 Memórialapozási táblázat

0 1	SYS U1		CART U1		UO U1	
2	VID	U2	VID	U2	VID	U2
3						
CART	00	20	08	28	10	30
SYS	40	60	48	68	50	70
U3	80	A0	88	A8	90	B0
EXT	C0	E0	C8	E8	D0	F0

5.4 Portok

PORT 00H **Output** (0B4FH)

Borderszín							
b7	b6	b5	b4	b3	b2	b1	b0
I	-	G	-	R	-	B	-

PORT 01H **Output**

Nyomtatóra küldendő adat

PORT 02H **Output** (0003H)

Memórialapozás				
b7 -- b6 3. lap	b5 2. lap	b4 -- b3 0. lap	b2 1. lap	b1 -- b0
00 CART 01 SYS 10 U3 11 EXT	0 VID 1 U2	00 SYS 01 CART 10 U0	1 U1	- -

PORT 03H **Output** (0B11H)

I/O memória-kiválasztás	———	Billentyüzet sorkiválasztás			
b7 --- b6	b5 -- b4	b3	b2	b1	b0
00 CSTL0 01 CSTL1 10 CSTL2 11 CSTL3	- -	A lehetséges 10-böl kivá-lasztott egyik sorszám			

PORT 04H **Output**

PITCH, alsó 8 bit

PORT 05H **Output** (0B12H)

Magnetofon-motor-vezérlés	Hang IT	Hang-jel	P I T C H felső 4 bit
b7 b6 jobb bal	b5 0 ki 1 be	b4 0 ki 1 be	b3 b3 b1 b0

PORT 06H **Output** (0B13H)

Nyomtató STROBE	—	Hang amplitúdó	Grafikus üzemmód
b7	b6	b5 --- b4 --- b3 --- b2	b1 --- b0
0 szint aktív	—	(0...15)*4 értéket képvisel	00 2 színű 01 4 színű 10 16 színű 11 16 színű

PORT 07H **Output**

A kurzor/hang IT nyugtázása (az írt adat közömbös)

PORT 10H,20H,30H,40H **Output/input**

Adatforgalom a soros vonalon (a 4 féle portcím a 0-1-2-3. csatlakozóba dugott soros vonali kártyát jelöli ki)

PORT 11H,21H,31H,41H **Output/input**

Üzemmódkiválasztás belső reset parancs után, ill. általános parancskiküldés a soros vonalra. Állapotbeolvasás a soros vonalról

Megjegyzés: a 10H, 11H, 20H, 21H, 30H, 31H, 40H, 41H portok ismertetése arra vonatkozik, ha mind a négy csatlakozóba soros vonali csatolókártya van bedugaszolva

PORT 50H **Output**

Kimenő jelszintváltás magnetofonfelvételhez

PORT 58H,59H,5AH,5BH **Output** (0-1-2-3. csatlakozó)

Csatoló IT							
b7 0 tilt. 1 eng.	b6 –	b5 –	b4 –	b3 –	b2 –	b1 –	b0 –

PORT 58H **Input**

A billentyűzet kiválasztott sorának beolvasása

PORT 59H **Input**

Nyomtató ACK	Szinkap-csoló	Magnetofon input	A függőben levő IT kérések kurzor, hang	csatolókártyák 3	2	1	0
b7 0 aktív	b6 0 ff 1 sz	b5	b4	b3	b2	b1	b0
			a bitek 0 értéke aktív				

PORT 5AH **Input**

A csatolókártyák azonosítói			
b7 -- b6 3.	b5 -- b4 2.	b3 -- b2 1.	b2 -- b1 0.

PORT 5BH **Input**

Hanggenerátor frekvenciaosztó alaphelyzetbe állítása

PORT 60H,61H,62H,63H **Output**

Paletta szinek A 0-1-2-3. palettaregiszter szinbeallitasa							
b7	b6	b5	b4	b3	b2	b1	b0
—	I	—	G	—	R	—	B

PORT 70H **Output**

—	—	—	Képernyővezérlő regiszterkijelölés				
b7	b6	b5	b4	b3	b2	b1	b0

PORT 71H **Output**

A képernyővezérlő kiválasztott regiszterének írása

5.5 Fontosabb rendszerváltozók, munkarekeszek táblázatok

0003	P-SAVE	Memórialapozási érték másolata
0021-002E	USR-TAB	Az EXTn címek táblája
0740-0AFF	DEFCH	A definiálható karakterek mátrixai
0B00-0B0F	I/O-TAB	Input-output hozzárendelési tábla
0B11	PORT03	A 03-as port másolata
0B12	PORT05	A 05-ös port másolata
0B13	PORT06	A 06-os port másolata
0B14	SOUND	Hang van folyamatban
0B15	SND-INT	Az új hang megszakítja a régit
0B16	STOP	A billentyűzeten CTRL+ESC lenyomása
0B19-0B1A	HI-MEM	A használható legnagyobb memóriacím
0B1D-0B1E	TIME	IT számláló, szoftver rendszeróra
0B21	WARM-FL	Meleg reset van folyamatban
0B22	COLD-FL	Hideg reset kell
0B4B	L-MODE	Vonalkeresztezési mód
0B4D	INK	Az aktuális tintaszín
0B4E	PAPER	Az aktuális alapszín
0B4F	BORDER	Az aktuális keretszín

0B51-0B5A	PICTURE	A billentyűzet aktuális állapota
0B5B-0B64	OLD-PIC	A billentyűzet előző állapota
0B65	DL-KEY	Billentyűkésleltetés
0B66	LOCK	A billentyűzet aktuális LOCK állapota
0B67	RATE	Billentyűismétlési ütem
0B68	HOLD	A CTRL+P hatás tiltása
0B73	GR-MODE	Grafikus üzemmód
0B74	PEN	A toll állapota
0BE5	KEY-ST	Van lenyomva billentyű
0BE9	CODE	A lenyomott billentyű kódja
0BE9	CODE	A lenyomott billentyű kódja
0EAC-16AB	STACK	Mikroprocesszor verem
1707	START	Az AUTORUN jelzése
1720-1721	VLOMEM	A BASIC terület kezdete
1722-1723	TEXT	Az aktuális BASIC program kezdőcíme
19EF		A standard BASIC báziscím
C5B4-C973	CHARS	Az állandó karakterek mátrixai

5.6 Színkódok

Szín	IGRB	Szín-sorszám	Paletta-kód	Border-kód
1. fekete	0000	00	00	00
2. sötétkék	0001	01	01	02
3. sötétvörös	0010	02	04	08
4. sötétlila	0011	03	05	0A
5. sötétzöld	0100	04	10	20
6. sötét kékeszöld	0101	05	11	22
7. sötétsárga	0110	06	14	28
8. szürke	0111	07	15	2A
9. fekete	1000	08	40	80
10. kék	1001	09	41	82
11. vörös	1010	0A	44	88
12. lila	1011	0B	45	8A
13. zöld	1100	0C	50	A0
14. kékeszöld	1101	0D	51	A2
15. sárga	1110	0E	54	A8
16. fehér	1111	0F	55	AA

5.7 Billentyűzetmátrix

Sor	b7	b6	b5	b4	b3	b2	b1	b0
0.	! 4	' 1		/ 6	& 0	" 2	+ 3	% 5
1.	= 7			# *		) 9	(8	~ ^
2.	R	Q	@	Z	$;	W	E	T
3.	U	P		{ [		O	I	}]
4.	F	A	> <	H	| \	S	D	G
5.	J			RET		L	K	DEL
6.	V	Y	LOCK	N	SHIFT	X	C	B
7.	M	_ -	SPACE	CTRL	ESC	: .	? ,	ALT
8.		RJL	RJR	RJA	RJF	RJD	RJU	INS
9.		LJL	LJR	LJA	LJF	LJD	LJU	

A **8. sor** a *beépített és az elülső (jobb oldali)*, a **9. sor** pedig a *hátsó (bal oldali) botkormányhoz* tartozik.

SPACE szóköz
RET RETURN

RJL *Right Joystick Left* = jobb botkormány balra
RJR *Right Joystick Right* = jobb botkormány jobbra
RJA *Right Joystick Accelerator* = jobb oldali gyorsítás
RJF *Right Joystick Fire* = jobb botkormány tűz
RJD *Right Joystick Down* = jobb botkormány le
RJU *Right Joystick Up* = jobb botkormány fel

A 9. sor ugyanezeket jelenti a *bal oldali botkormány* használata esetén, a kezdő *L* betű a *left* (bal) szóra utal.

5.8 ASCII táblázat

Kód	Funkció	Nyomógomb
00		CTRL+@
01	RJA	CTRL+A
02		CTRL+B
03		CTRL+C
04	RJR	CTRL+D
05	RJU	CTRL+E
06	RJF	CTRL+F
07	DC	SHIFT+DEL
08	DEL	DEL
09	TAB	CTRL+I
0A	NL	CTRL+J
0B	CEL	CTRL+K
0C		CTRL+L
0D	RET	RETURN
0E	IL	CTRL+N
0F		CTRL+O
10	HOLD	CTRL+P
11		CTRL+Q
12		CTRL+R
13	RJL	CTRL+S
14		CTRL+T
15		CTRL+U
16	INS	INS
17		CTRL+W
18	RJD	CTRL+X
19	DL	CTRL+Y
1A		CTRL+Z
1B	ESC	ESC
1C		CTRL+\
1D		CTRL+]
1E		CTRL+^
1F		CTRL+-
20	SPACE	SPACE
21	!	SHIFT+4
22	"	SHIFT+2
23	#	SHIFT+*
24	$	SHIFT+;
25	%	SHIFT+5
26	&	SHIFT+O
27	'	SHIFT+1
28	(	SHIFT+8
29	)	SHIFT+9
2A	*	*
2B	+	SHIFT+3

Kód	Funkció	Nyomógomb
2C	,	,
2D	-	-
2E	.	.
2F	/	SHIFT+6
30	0	0
31	1	1
32	2	2
33	3	3
34	4	4
35	5	5
36	6	6
37	7	7
38	8	8
39	9	9
3A	:	SHIFT+.
3B	;	;
3C	<	<
3D	=	SHIFT+7
3E	>	SHIFT+<
3F	?	SHIFT+,
40	@	@
41	A	SHIFT+A
42	B	SHIFT+B
43	C	SHIFT+C
44	D	SHIFT+D
45	E	SHIFT+E
46	F	SHIFT+F
47	G	SHIFT+G
48	H	SHIFT+H
49	I	SHIFT+I
4A	J	SHIFT+J
4B	K	SHIFT+K
4C	L	SHIFT+L
4D	M	SHIFT+M
4E	N	SHIFT+N
4F	O	SHIFT+O
50	P	SHIFT+P
51	Q	SHIFT+Q
52	R	SHIFT+R
53	S	SHIFT+S
54	T	SHIFT+T
55	U	SHIFT+U
56	V	SHIFT+V

ASCII kódok (folytatás)

Kód	Funkció	Nyomógomb
57	W	SHIFT+W
58	X	SHIFT+X
59	Y	SHIFT+Y
5A	Z	SHIFT+Z
5B	[	[
5C	\	\
5D	]	]
5E	^	^
5F	_	SHIFT+-
60	`	SHIFT+@
61	a	A
62	b	B
63	c	C
64	d	D
65	e	E
66	f	F
67	g	G
68	h	H
69	i	I
6A	j	J
6B	k	K
6C	l	L
6D	m	M
6E	n	N
6F	o	O
70	p	P
71	q	Q
72	r	R
73	s	S
74	t	T
75	u	U
76	v	V
77	w	W
78	x	X
79	y	Y
7A	z	Z
7B	{	SHIFT+[
7C	¦	SHIFT+\
7D	}	SHIFT+]
7E	~	SHIFT+^
7F	■	
80	Á	SHIFT+Á
81	É	SHIFT+É
82	Í	SHIFT+Í
83	Ó	SHIFT+Ó
84	ö	SHIFT+ö
85	Ő	SHIFT+Ő
86	Ú	SHIFT+Ú
87	Ü	SHIFT+Ü
88	Ű	SHIFT+Ű
89	┌	CTRL+0
8A	┐	CTRL+2
8B	└	CTRL+4
8C	┘	CTRL+6
8D	├	CTRL+8
8E	┤	CTRL+*
8F	▲	
90	á	Á
91	é	É
92	í	Í
93	ó	Ó
94	ö	Ö
95	ő	Ő
96	ú	Ú
97	ü	Ü
98	ű	Ű
99	┬	CTRL+1
9A	┴	CTRL+3
9B	─	CTRL+5
9C	┼	CTRL+7
9D	│	CTRL+9
9E	▼	
9F	►	
A0		ALT+0
A1		ALT+1
A2		ALT+2
A3		ALT+3
A4		ALT+4
A5		ALT+5
A6		ALT+6
A7		ALT+7
A8		ALT+8
A9		ALT+9
AA		ALT+*
AB		ALT+;
AC		ALT+ <
AD		ALT+-

ASCII kódok (folytatás)

Kód	Funkció	Nyomógomb
AE		ALT+.
AF		ALT+,
B0		ALT+@
B1		ALT+A
B2		ALT+B
B3		ALT+C
B4		ALT+D
B5		ALT+E
B6		ALT+F
B7		ALT+G
B8		ALT+H
B9		ALT+I
BA		ALT+J
BB		ALT+K
BC		ALT+L
BD		ALT+M
BE		ALT+N
BF		ALT+O
C0		ALT+P
C1		ALT+Q
C2		ALT+R
C3		ALT+S
C4		ALT+T
C5		ALT+U
C6		ALT+V
C7		ALT+W
C8		ALT+X
C9		ALT+Y
CA		ALT+Z
CB		ALT+[
CC		ALT+\
CD		ALT+]
CE		ALT+^
CF		CTRL+Á
D0		ALT+Á
D1		ALT+É
D2		ALT+Í
D3		ALT+Ó
D4		ALT+Ö
D5		ALT+Ő
D6		ALT+Ú
D7		ALT+Ü
D8		ALT+Ű
D9		CTRL+É
DA		CTRL+Í
DB		CTRL+Ó
DC		CTRL+Ö
DD		CTRL+Ő
DE		CTRL+Ú
DF		CTRL+Ü
E0		
E1	LJA	
E2		
E3		
E4	LJR	
E5	LJU	
E6	LJF	
E7		
E8		
E9		
EA		
EB		
EC		
ED		
EE		
EF		
F0		
F1		
F2		
F3	LJL	
F4		
F5		
F6		
F7		
F8	LJD	
F9		
FA		
FB		
FC		
FD		
FE		
FF		

190 Ft

A könyvben közölt ismeretek és a bemutatott programok alapján az Olvasó bátran kipróbálhatja alkotókészségét. Gépi kódban mindent ki tudunk hozni gépünkből, ami fizikailag egyáltalán lehetséges.

Ugyanakkor nem szabad elhallgatni, hogy a gépi kódú programozás fantáziadús gondolkodást, nagyfokú kreativitást és komoly munkát igényel. Természetesen komolyan fejleszti programozási képességeinket, ötletességünket.

A könyv Olvasója eljuthat az önálló munka örömeig.

VIDEOTON

ELEKTRONIKAI VÁLLALAT
SZÁMÍTÁSTECHNIKAI GYÁRA