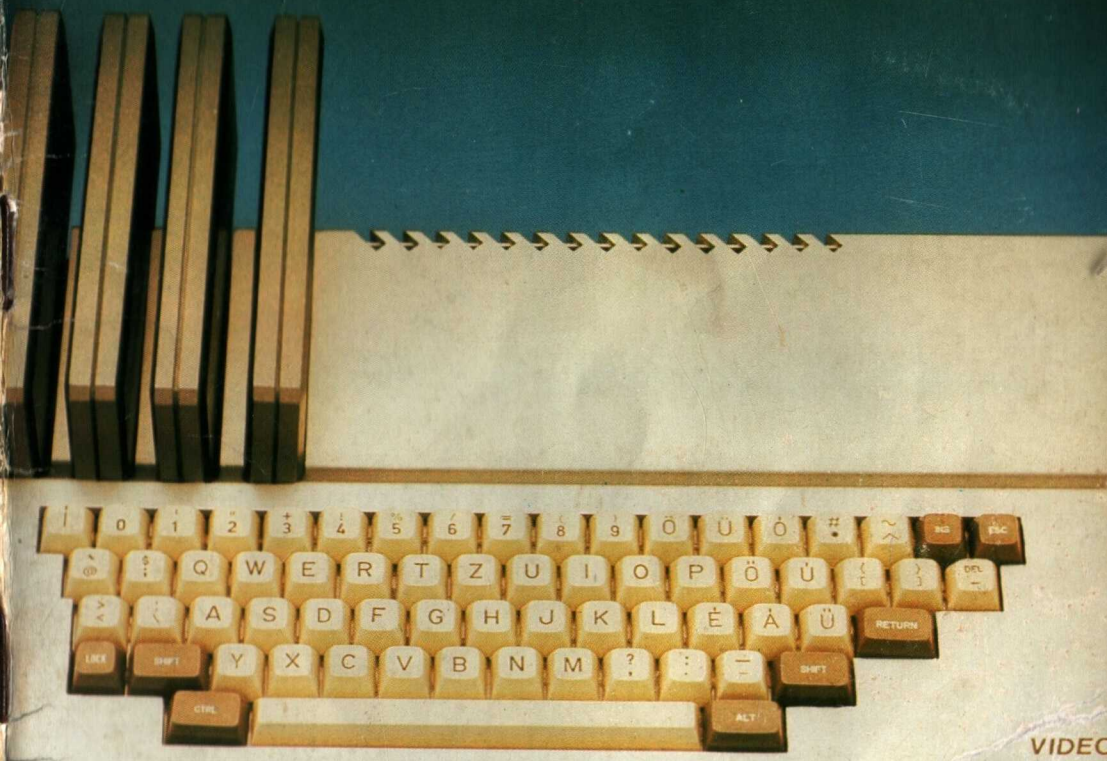
DEBUGGER



VIDEOTON

TV-Computer

TV–COMPUTER

DEBUGGER

felhasználói kézikönyv

Minden jog fenntartva!

TARTALOMJEGYZÉK

1. BEVEZETÉS

A DEBUGGER program célja, hogy segítséget nyújtson azoknak az amatőr és hivatásos programozóknak, akik a VIDEOTON TV–Computerre kisebb-nagyobb gépi kódú és/vagy assembly nyelvű programokat írnak. A debugger lehetővé teszi a gépi szintű programok utasításonkénti végrehajtását, töréspontok elhelyezését, futó programok operátori megszakítását, és még számos szolgáltatatást nyújt, amellyel a programbelövést gyorsítani lehet.

Szolgáltatásait és konvencióit tekintve a debugger megegyezik, vagy hasonlít néhány más VIDEOTON készüléknél alkalmazott debuggerhez: VDT, VDDS, VT20, VT20A, stb.

Néhány lényegtelennek tűnő konvenciót azért tartottunk be, hogy könnyebbé tegyük azon felhasználók dolgát, akik az említett készüléknél már megszokták a debugger használatát. Ilyenek pl. a hívható rutinok belépési pontjai, a parancsok paraméterezése, stb.

A debugger lehetővé teszi (de nem követeli meg) egy ún. cross-gép létezését. A cross-gép feladata, hogy biztosítsa az assembly nyelvű programok forrásszintű rögzítését, javítását, tárolását, fordítását és a tárgyprogramok átküldését a TV–Computerre. A belövés alatt álló program futtatása már a TV–Computeren történik. A cross-gép lehet pl. egy VT20A, VT20/IV, de lehet pl. bármely CP/M kompatibilis kisgép, amely rendelkezik megfelelő méretű és sebességű háttértárral. Megfelelő cross-gép céljára pl. egy floppy diszkes TV–Computer UPM operációs rendszerrel.

A cross-gépnek és a TV–Computernek rendelkeznie kell RS232 aszinkron vonali interface-szel. A TV–Computer vonali interface-ére köthető bármilyen aszinkron display. Ez lehet maga a cross-gép is echo üzemmódban (vonal a konzol display képernyőjére, konzol billentyűzet a vonalra).

A debugger kezelhető a TV–Computer billentyűzetéről és a TV–Computer képernyő segítségével, de ha ez zavarná a belövés alatt álló programot, akkor a kezelési funkciók áthelyezhetők az aszinkron vonalra kötött display-hez.

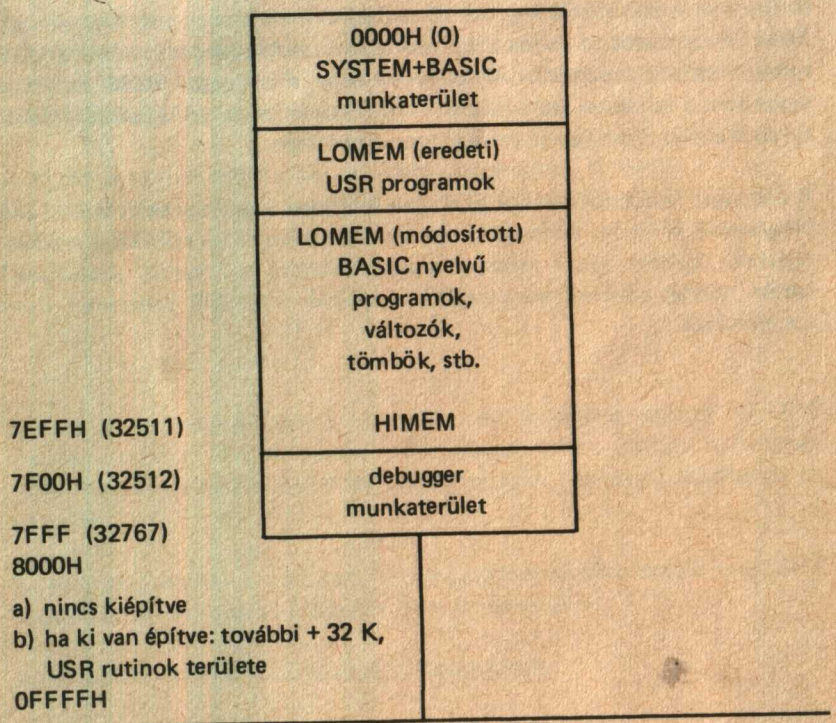
A debugger ROM-kazettában kerül forgalomba, amelynek bentléte nem

zavarja a normál BASIC működését, hatása csak a debugger indítása után van. A debugger által végzett I/O műveletek a rendszer (SYSTEM–ROM) szolgáltatásaira épülnek. Ez a leírás részletesen csak a debugger ROM-kazetta szolgáltatásait ismerteti. Feltételezi, hogy a felhasználó ismeri a TV–Computer rendszertechnikai felépítését, az operációs rendszer szolgáltatásait és azok elérési módját, a mikroprocesszor utasításkészletét, stb., egyszóval azokat az információkat, amelyek az assembly és gépi kód szintű programozáshoz szükségesek, és egyéb dokumentációkból elérhetőek.

2. MEMÓRIAFELHASZNÁLÁS

A debugger *programterülete* ROM-kazettában van, így a BASIC által használható területet *nem csökkenti.* Szüksége van viszont kb. 256 byte *munkaterületre* is, amelynek feltétlenül írható-olvasható RAM-memóriában kell lennie. Ezt a területet mindenképpen a BASIC szabad területéből kell elszakítani. Erre két lehetőség kínálkozik: megnövelni a LOMEM (értéke 6639=19EFH), vagy lecsökkenteni a BASIC HIMEM értékét (32767=7FFFH a 32 K-s, illetve 49151=0BFFFH a 64 K-s változatnál).

A BASIC alatt indított assembly nyelvű programok általában az első módszert választják, mert így lehetnek futtathatók változatlan formában a bővített és az alapgépen. Éppen ezért a debugger esetében a másik lehetőséget választottuk. A memóriafelhasználást az alábbi ábra szemlélteti:



A memóriafelhasználás ilyen szervezésének van egy hátrányos következménye: abban az esetben, ha a bővített memóriához használjuk a debuggert, akkor a HIMEM értéke a szükségesnél 16 Kbyte-tal nagyobb mértékben csökken. Ez azonban csak akkor lehet baj, ha nagyméretű BASIC programból akarunk assembly programokat hívni és kevés a BASIC számára maradó terület.

Feltehető azonban, hogy az alkalmazások jelentős része (pl. bonyolult játékok) a BASIC-et csak indítás vagy keretprogram céljára használja. Ilyen esetben a HIMEM értéke a 7EFFH értékről tovább csökkenthető (ld. HIMEM SET rendszerszolgáltatás), és ezzel további terület szabadítható fel USR rutinok céljára.

Ugyanez a helyzet a 32 K-s gép esetén: nemcsak a LOMEM növelésével, hanem a HIMEM csökkentésével is nyerhető szabad terület.

A RAM memória PAGE3-ra eső része BASIC alatt amúgy sem használható. Ezért ha a memória teljesen ki van építve, elsősorban ezt célszerű USR rutinok céljára használni. Az a tény, hogy a debugger ROM kazettája ugyanezen a területen helyezkedik el, *nem akadályozza* a programkövetést és egyéb szolgáltatások igénybevételét ezen a területen.

A debugger munkaterületének akaratlan felülírása — ami egy belövés alatt álló programnál könnyen előfordulhat — természetesen hibás működést eredményezhet. Ugyanez igaz a rendszer munkaterületére is. Gyanús „elszállások" esetén mindig célszerű megvizsgálni a belövés alatt álló programot ilyen szempontból.

3. MEMÓRIA LAPOZÁS (PAGING)

Mint ismeretes, a TV–Computer 64 kbyte memóriacímtartományt tud címezni, de ennél lényegesen több memóriát tartalmaz. A több memória felhasználása a memória lapozás (paging) segítségével történik.

Alaphelyzetben (azaz a BASIC futás alatt) a memória 4 lapja az alábbi állapotban van kiválasztva:

PAGE0	PAGE1	PAGE2	PAGE3
USER RAM	USER RAM	USER RAM	SYSTEM ROM

Röviden: UUUS.

Ezt a kiosztást a felhasználó által írt programok módosíthatják, sőt bizonyos esetekben módosítaniuk is kell. Ilyen pl. az az eset, ha a P3 alatti USER RAM területen USR rutinokat akarunk futtatni, vagy a P2 területen levő VIDEO RAM területét akarjuk írni vagy olvasni.

A memória lapozás módosítása az alábbi utasításokkal történik:

```
DI
LD A, X
LD (3), A
OUT (2), A
EI
```

ahol X az ún. paging regiszter új értéke. Az új paging regiszter az OUT utasítás hatására válik érvényessé. Vigyázzunk tehát, nehogy a program „kilapozza maga alól a memóriát", vagy ha igen, akkor az kiszámítottan történjen.

X lehetséges értékei a hardware dokumentációkból állapíthatók meg, de a leggyakrabban előforduló értékeket itt is ismertetjük.

A táblázatban az alábbi rövidítéseket alkalmazzuk:

P0, . . . P3 : PAGE0, . . . , PAGE3
U : USER RAM
S : SYSTEM ROM
C : CARTRIDGE ROM
V : VIDEO RAM
E : Expanziós ROM (SYST bővítés)

	Memória			
Kód	P0	P1	P2	P3
B0	U	U	U	U
30	U	U	U	C
70	U	U	U	S
F0	U	U	U	E
10	U	U	V	C
50	U	U	V	S
90	U	U	V	U
D0	U	U	V	E
28	C	U	U	C
68	C	U	U	S
A8	C	U	U	U
E8	C	U	U	E
08	C	U	V	C
48	C	U	V	S
88	C	U	V	U
C8	C	U	V	E

Az LD (3), A utasítás a memória 3 című rekeszébe elteszi a page regiszter aktuális értékét. Ennek célja, hogy operációs rendszer hívás (RST 30H) vagy megszakítás után a hívás, illetve megszakítás előtti érték visszaállítható legyen.

Ugyancsak a 3 című memóriarekeszbe beírt érték közli a debugger programmal a felhasználó által alkalmazott page regiszter értéket akkor, amikor töréspont, megállási cím vagy operátori megszakítás után a vezérlés

a debugger programhoz kerül. A debugger számára azonban a felhasználói page regiszter ismeretének kettős szerepe van:

- továbbindítás előtt vissza kell állítania a felhasználó page regisztereit, hogy a megállított program korrekt módon futhasson tovább; maga a debugger ugyanis UUUC page regiszter állással (X= 30H) fut, ez teszi lehetővé, hogy a munkaterületen kívül ne foglaljon helyet a memóriából;
- debugger funkcióinak jelentős része a felhasználói page regiszter által meghatározott memóriára vonatkozik.

Pl. Töréspontot akarunk elhelyezni a 0C000H–0FFFFH címtartományban vagy be akarunk pillantani a VIDEO RAM tartalmába, miután megállítottunk egy VIDEO RAM-ot manipuláló programot (amelynek page regiszter állása pl. UUVS.)

A fentiek miatt a debugger elmenti magának a 3 című memóriarekesz tartalmát és a megfelelő időpillanatban visszaállítja. A debugger a funkciók végrehajtása közben időről időre felhasználja ezt az értéket, azaz minden funkciót ezzel az értékkel hajt végre.

A felhasználói page regisztert a debugger a

PGSAV = 7F03H

memóriacímen tartja. Ennek a rekesznek a tartalmát a debugger M parancsával tetszőlegesen módosíthatja a felhasználó, így a memória más részeibe is bepillanthat, akár magába a debuggerbe is.

Fontos figyelmeztetés!

A PGSAV rekesz módosítása után a felhasználói programot *nem szabad továbbindítani, míg a PGSAV eredeti értékét vissza nem állítottuk!* Egyetlen utasítás végrehajtása (I parancs) is lehetetlen, ha kilapoztuk alóla a memóriát ezzel a módosítással.

A következmények mindig az adott helyzettől függnek!

Egyszerűbb esetekben, mikor a felhasználó megelégszik a BASIC által nyújtott UUUS értékű page regiszterrel, akkor a lapozás kérdése nem okoz problémát, gyakorlatilag odafigyelést sem igényel.

A memória lapozás technikájából következik, hogy a stack pointer beállításával nagyon óvatosan kell eljárni. Ha a felhasználó nem fogadja el a BASIC által állított stack pointert (ez olyan programok esetén fordul elő, amelyek nem akarnak visszatérni a BASIC-hez), akkor olyan stack pointer értéket kell állítani, amely mindig RAM és soha nem lesz „ellapozva"! Célszerű a P1 használata, mert azt nem lehet ellapozni. A P0 és P2 használható, de megfelelő odafigyeléssel. A P3 használata *stack céljára* a debuggerrel egyidőben tilos!

4. A DEBUGGER INDÍTÁSA

a) A készülék kikapcsolt állapotában helyezzük el a ROM kazettát a bal oldalon található csatlakozó nyílásban!

b) Ezután kapcsoljuk be a készüléket!

Bekapcsolás után a BASIC a szokásos módon kijelentkezik és használható. A debuggernek mindaddig nincs hatása a működésre, míg az inicializáló szubrutinját le nem futtatjuk. Az inicializáló szubrutin indítható a 0C000H címen, miután a ROM kazettát a P3-ra lapoztuk. Ennek két módja van.

4.1. Általános eset

BASIC–USR függvényként le kell futtatni az alábbi utasításokat:

```
F3                DI
3E  30            LD A,30H
D3  02            OUT (2), A (49933)
C3  00  C0        JP 0C000H
```

Ennek érdekében javasoljuk az alábbi BASIC programot:

```
10 LOMEM 6700
20 DATA 243, 62, 48, 211, 2, 195, 0, 192
30 FORI=0TO7:READJ:POKE 6750+I, J:NEXT
40 X=USR (6750)
RUN
```

E program hatására a P3-ra a ROM kazetta kapcsolódik, a debugger inicializáló rutinja lefut és kijelentkezik egy * karakterrel. Ebben a helyzetben használhatók a következő fejezetben leírt parancsok. Közülük a J parancs segítségével visszatérhetünk a BASIC-hez.

Ezután a debugger „él". Az elhelyezett töréspontok, megállási címek hatásosak lesznek, ha a BASIC-ből kiinduló USR hívások során rájuk fut a processzor. Fennáll az operátori megszakítás lehetősége is (ld. később).

4.2. Gyors indítás

A fenti leírt módszer túlságosan sok gépelést igényel, ezért egy kevesebb munkával járó indítási módszert is közlünk.

A fenti program begépelése helyett elegendő az

X=USR (49933)

függvényhívás begépelése, amelynek hatása ugyanaz mint az előbbi programé.

Ezután is kijelentkezik a debugger a * karakterrel.

Hátránya viszont, hogy abszolút memóriacím hivatkozás van benne (49933), amely az operációs rendszer változása esetén megváltozhat.

Ennek az indítási módszernek a lényege, hogy kihasználja a SYSTEM ROM jelenleg 49933 címén levő

```
3E  30      LD    A,  3UH
D3  02      OUT   (2), A
```

utasításokat, amely a P3-ra a SYSTEM ROM helyett a ROM kazettát kapcsolja. A ROM kazetta megfelelő címén olyan program van, amely várja ezt a kapcsolást, és ugrik az inicializáló szubrutin elejére.

A debugger föl van készítve az operációs rendszer kisebb változására ilyen szempontból. Ha a szóban forgó utasítás +— 48 byte-tal elcsúszik, akkor a debugger fogadni képes a gyors indítást.

Ilyen esetben a felhasználó feladata, hogy megkeresse a SYSTEM ROM-ban a szóban forgó 4 byte-nyi programrészletet:

a) Indítsa el a debuggert a 3.1. pontban leírt módon!

b) Módosítsa a PGSAV értékét UUUS-re (ha nem olyan értéken áll)!

c) A D vagy M paranccsal keresse meg a 49933 + — 48 tartományban a fenti 4 byte-ot! (49933 = 0C30DH)

d) Ha megtalálta, akkor a „3E" címét helyettesítse be a 49933 helyébe, természetesen decimálisba átszámítva!

e) Ha nem találta, akkor a SYSTEM ROM és a ROM kazetta verziójának egyeztetése céljából keresse fel a VIDEOTON legközelebbi vevőszolgálatát!

5. A DEBUGGER SZOLGÁLTATÁSAI

Az első indítás után, amikor a debugger megkapja a vezérlést, egy * karakterrel kijelentkezik, az összes szolgáltatás igénybe vehető. A kezelő különböző parancsok begépelésével kaphatja meg a kívánt információt és válthatja ki a kívánt hatást.

A parancsok általában egyetlen betűből és 0, 1, 2, 3 vagy 4 numerikus paraméterből állnak. A paraméterek mindig hexadecimálisak. Ebben a leírásban P1, P2, P3 és P4-el jelöljük őket. A paraméter 0, 1, 2, 3 vagy 4 hexadecimális számjegyből áll.

A paraméterek begépelésének szabályai:
- A szám előtti nem értékes 0-k beütése elhagyható; ha a paraméter értéke 0, akkor elegendő a zárókarakter (szóköz vagy vessző) leütése.
- Téves (de hexadecimális) számjegy beütése úgy javítható, hogy folytatólagosan újragépeljük az egész számot; mindig az utolsó 2, illetve 4 jegy számít, attól függően, hogy az adott paraméter helyén 1 vagy 2 byte-os értékre van szükség.
- A szóköz és a "," (vessző) a paraméter zárókaraktere.
- A kocsi vissza a parancs zárókaraktere.
- Bármilyen egyéb vezérlőkarakter a parancs gépelésének abbahagyását jelenti.
- Bármilyen nem hexadecimális, nem vezérlő karakter leütése a képernyőn látható, de hatástalan, érvényes hexadecimális számjeggyel fölülírható.

El nem fogadható parancs, hibás paraméter, vagy bármilyen hiba esetén a debugger "? " karakterrel jelez, és utána új parancsot vár.

Általános tudnivalóként célszerű előre bocsátani, hogy a debugger állandóan őrzi az elindított programhoz tartozó CPU állapotot (az összes regiszter és flag értékét). Ez az állapot első indításkor definiálatlan, töréspont és megszakítás után az, amit a program beállított.

A parancsokat azonosító betűk legtöbbje valamilyen mnemonikus információt hordoz. A könnyebb memorizálás érdekében ezt az egyes parancsoknál kihangsúlyozzuk. Valamennyi parancs a felhasználói page regiszter által meghatározott memóriára vonatkozik!

5.1. D–Display memory

* D P1, P2

Funkciója: memóriatartalom megjelenítése hexadecimálisan P1 kezdőcímtől P2 címig. A megjelenítés 128 byte-onként történik, azaz a kép képernyőnként megáll, lehetővé téve az olvasást. Szóköz leütésére továbbmegy, a "cursor balra" leütésére 128 byte-tal visszalép. a "." leütésével a parancs végrehajtása félbeszakad. P2>P1 nem kötelező. Ha P1<>0, akkor P2 egy szóközzel helyettesíthető, majd a parancs lezárása a "."-tal történhet. (Gyorsabb kezelhetőség érdekében.)

5.2. F–FILL memory

* F P1, P2, P3

Memória feltöltés P3 konstanssal P1 címtől P2 címig.

5.3. T–Transfer memory

* T P1, P2, P3

Memóriatartalom áthelyezése P1 címről P2 címre P3 hosszon. Az áthelyezés növekvő címekkel történik, tehát
P2 nagyobb P1 esetén
(P1+P3) nagyobb P2-nek is igaznak kell lennie a korrekt áthelyezéshez.

5.4. M–Modify memory

* M P1

Memória tartalmának kijelzése és módosítása P1 címtől. A P1 paraméter lezárása után kiírja a P1 című memóriarekesz tartalmát hexadecimálisan, utána egy "–" karaktert. Ebben a helyzetben a kezelőnek az alábbi lehetőségei vannak:

- Szóköz leütése: a memóriarekesz tartalmának változatlanul hagyása és a parancs folytatása a P1+1 memória címmel;
- Érvényes hexadecimális számjegy leütése (00 esetén is legalább 1 jegy begépelendő): P1 című rekesz tartalmának módosítása a beütött értékre és a parancs folytatása a P1+1 címen;
- Cursor balra leütése: P1–1 cím kijelzése egy új sor elejére és a parancsvégrehajtás folytatása P1–1 címen.
- Cursor jobbra leütése: P1+1 cím kijelzése egy új sor elejére és a parancs folytatása P1+1 címen;
- Kocsi vissza leütése: parancsvégrehajtás befejezése.

5.5. R–Register display and modify

* R

A CPU regisztereinek kijelzése és a módosítás lehetővé tétele az M parancsnál leírtakhoz hasonló módon. A módosítás egyenként történik "A=00–" típusú kijelzés után.

Szóköz: továbblépés módosítás nélkül
Kocsi vissza: a módosítás abbahagyása.

Visszalépésre itt nincs lehetőség, helyette a parancsot újra kell kezdeni.
F a flag byte-ot jelöli, az egyes bitek a CPU által definiált kiosztásban vannak elhelyezve:

7.	6.	5.	4.	3.	2.	1.	0.
Sign	Zero	0	Aux. Carry	0	Parity	1	Carry

5.6. G–Go to address

*G P1

Program indítása vagy továbbindítása P1 címről. Mielőtt a debugger átadná a vezérlést a megadott címre, visszaállítja a CPU állapotot. Természetesen az utasításszámláló már P1 értékét kapja.

A G parancs alkalmazásának gyakori esete, mikor a „modify" parancs segítségével gépi kódban beírunk néhány byte-os programokat, és ezt indítjuk el. Ilyen esetekre számítva ismét felhívjuk a figyelmet a felhasználói page regiszter szerepére! A G parancs csak olyan címre adható ki, amely a felhasználói page regiszter szerint is „él", mivel a végrehajtás előtt a page regiszter aktualizálódni fog.

5.7. H—Halt on address

* H P1, P2

Megállási cím definiálása P1 címen. Ha az elindított program P2-ször ráfut P1 címre, végrehajtása megszakad, visszaadja a vezérlést a debuggernak, és a debugger kijelzi az aktuális CPU állapotot. Ha P2 helyett 0-t, vagy space-t adunk meg, az egyenértékű azzal, mintha 1-et adtunk volna meg. P2 maximális értéke 0FFFFH, tehát a paraméter megadásakor az utolsó 4 jegy számít.

Megállási cím egyidejűleg csak egy lehet, új megállási cím definiálásakor a régi érvényét veszti. A „megállás címen" funkció megvalósítása a címen levő eredeti utasítás RST-re való lecserélésével történik. Ez azt jelenti, hogy megállási címnek *kizárólag valamely utasítás első byte-jának címét szabad megadni!*

A „megállási cím" fogalmát a továbbiakban megkülönböztetjük a „töréspont" fogalmától. A „megállási cím" fogalmába — szemben a „töréspont"-tal — beleértjük, hogy az adott címen levő utasítást a debugger automatikusan kezeli; tehát pl. új megállási cím definiálása vagy a megállási cím vándorlása (ld. I parancs) esetén az eredeti utasítás visszakerül a régi megállási címre az RST végrehajtása után. A megállási cím használata nagyon kényelmes, mert a programozónak nem is kell tudnia az utasításcseréről.

Ha ROM címet, vagy nem kiépített memóriacímet adunk meg, nem fogadja el a debugger, hibajelzést ad (?), viszont a korábbi megállási cím már érvényét veszti.

A debugger *nem fogadja el* a H parancsot az *aktuális címre*, azaz soron következő utasításra vonatkozóan. Ez esetben a C parancsot kell használni, vagy egy I parancs után megismételni a H parancsot.

5.8. B–Breakpoint

* B P1

Töréspont definiálása a P1 címen. Ha az elindított program ráfut a töréspontra, végrehajtása megszakad, visszaadja a vezérlést a debuggernek és a debugger kijelzi a regiszterek értékét.

Töréspont egyidejűleg tetszőleges számú lehet, ugyanis a debugger nem jegyzi meg a töréspontón lévő eredeti utasítást. Megtehetjük pl., hogy a program betöltése előtt feltöltjük a memóriát EF-el (RST 28H kódja). (Vigyázat! Csak a szabad memóriát töltsük föl!)

Ez jó védekezés olyan programhibák ellen, amikor a program kiugrik saját területéről.

Töréspontí címként csak valamely utasítás első byte-jának címét szabad megadni. Nem kiépített memória, vagy ROM cím megadása hibajelzést vált ki.

5.9. I–Instruction step

* I

Egy utasítás végrehajtása. A végrehajtás után a debugger kijelzi az aktuális CPU állapotot. A kijelzés áttekinthetően, de több sorban történik. Az I billentyű nyomkodásával kényelmesen lehet követni a regiszterek változását. Ha a végrehajtott programban vezérlésátadás történik, akkor a CPU állapotok kijelzése során egy üres sor keletkezik. Az I parancs végrehajtásakor a soron következő utasítás utáni utasítás címe megállási cím lesz.

Az I parancs sorozatos kiadása tehát a megállási cím vándorlását okozza. Ha az aktuális utasítás valamely ok miatt RST 28H (pl. azért, mert töréspont címen állunk), akkor a debugger az I parancs kiadásakor kérni fogja az aktuális címen levő eredeti utasítás beírását I= karakterek kiírásával.

Az utasítás kódjának sikeres begépelése után a soron következő utáni cím megállási címmé változik, a korábbi megállás cím pedig töréspontté változik.

Ha a soron következő utáni cím ROM-ban vagy nem kiépített memóriában van, akkor az utasítás végrehajtására nem kerül sor és a debugger hibát jelez.

Az RST utasítást (kivéve RST 28H és RST 30H) azonban egy byte-osnak tekinti a debugger és nem kísérli meg a követést. Az RST 30H utasítás operációs rendszer hívás, melynek paramétere a kód után van elhelyezve. Ezért azt 2 byte-os utasításnak tekinti a debugger.

Lehetőség van arra is, hogy a CALL utasítások követését letiltsuk, függetlenül attól, hogy az illető szubrutin ROM-ban vagy R/W memóriában helyezkedik el. Ebből a célból a debugger FLSUB (follow subroutine) nevű flagjét W paranccsal nullázni kell.

5.10. C–Cycle

* C

Programciklus végrehajtása. Az utasításszámláló aktuális értéke „megállási cím" lesz, egyszeri ráfutással. Azaz, ha a program legközelebb ráfut erre a címre, a debugger ismét visszakapja a vezérlést és kijelzi az aktuális állapotot.

Ez a parancs olyan esetekben alkalmazható, amikor a program egy kritikus pontját akarjuk nyomon követni. Ilyenkor a C billentyű ütögetésével a kritikus ponton levő állapotra koncentrálhatunk.

Ha RST 28H utasításon áll a program akkor az I parancsnál leírt procedura szerint léphetünk tovább. Ha a soron következő utasítás ROM-ba vezet, nem hajtható végre a C parancs.

5.11. E—Execute

* E

Végrehajtás továbbindítása az aktuális címről. Ezzel indítjuk a programot

— megállási címen vagy töréspontón történt megállás után,
— operátori megszakítás után,
— program betöltés után, amikor az indulási címet a záróblokk definiálta.

Ha egy töréspontí címről akarjuk a programot továbbindítani, akkor a debugger "I=" karakterek kiírásával kérni fogja a címen levő valódi utasítás kódjának begépelését.

Ha a programot útjára engedtük egy E paranccsal, akkor kezelői beavatkozással megszakíthatjuk. Az operátori megszakítást a TVC esetében a CTRL és az ALT billentyű egyidejű lenyomása váltja ki, függetlenül attól, hogy az aktuális konzol a vonalon van vagy a TVC-n.

Az operátori megszakítás mechanizmusa a CPU megszakításon keresztül hat, tehát DI alatti programokat nem lehet megszakítani. (Nem az NMI bemenet működik!)

Az operátori IT-vel megszakított programok CPU állapotát a megállási címhez hasonló módon jelzi ki a debugger. Utána I, C, vagy E parancs kiadásával lehet továbblépni.

Figyelem! Ha ROM-ban futó programot szakítottunk meg, akkor csak az E parancs alkalmazható, lépésenként nem tudjuk követni.

5.12. P–Perform hexadecimal

* P P1, P2

Kiszámítja és kiírja a P1+P2 és a P1–P2 értékeket.

5.13. N–priNter

* N P1

P1 értékét beírja a "hard copy"jelentésű munkarekeszbe. 0 érték a hard copy kikapcsolt, 0-tól eltérő pedig a bekapcsolt állapotát jelzi.

A hard copy bekapcsolt állapotában a debugger minden egyes karaktert, amit a képernyőre ír, kiküld a nyomtatóra is. Ha a nyomtató nincs bekapcsolva, nincs csatlakoztatva, vagy nem üzemkész, akkor a hard copy bekapcsolása természetesen hatástalan. A hard copy alapbeállítása: 0.

5.14. W–followW subroutine

* W P1

Az FLSUB munkarekesz beállítása P1 értékre. A rekesz 0 értéke mellett a debugger az összes (feltételes és feltétlen) szubrutinhívást egyetlen utasításnak tekint. Ezzel lehetővé teszi, hogy a belövés alatt álló programnak, vagy programrészletnek csak a főágát kövessük. Jól struktúrált programok esetében ezzel a hibahely gyorsabb behatárolását érhetjük el.

Az FLSUB flag 0-tól különböző értéke mellett a szubrutinokat követi a debugger.

A követés természetesen az utasításonkénti végrehajtásra (I parancs) is vonatkozik.

Az FLSUB alapbeállítása: 255.

5.15. J–Jump to BASIC

* J

Az első hívás után alkalmazható, visszatérést eredményez a BASIC-hez. Mivel az inicializáló részt a BASIC-ből függvényként hívták, visszatéréskor a függvényérték a debugger munkaterületének címe lesz (7F00H).

5.16. L–Load program

* L P1 <név>

Program betöltés. P1 értéke az input perifériát határozza meg

P1 = 0 – RS 232 interface
P1 = 1 – magnókazetta v. floppy diszk.

Név megadásának P1 = 1 esetén lehet értelme.

5.16.1. Load az RS 232 interface-ről

A load parancs használata közben az RS 232 interface a cross-géppel kommunikál. Feltételezi, hogy a load parancs kiadása után a cross gépen elindítottak egy programot, amely a betöltendő programot a cross-gép könyvtárából átküldi a tárgygépre.

Az átküldött program formátuma kontroll-szummával ellenőrzött blokk-szerkezet. Kétféle formátumra van felkészülve a loader: egyik a közismert Intel-hexa formátum, ilyet produkálnak a CP/M és az UPM operációs rendszerek assemblerei (ASM, MAC, ASM80, ASMZ80), valamint a másik egy régóta használatos bináris VIDEOTON formátum, amely az Intel-hexa formátumnál tömörebb. Ilyet produkál a debugger save (S) parancsa, a VT20A Save parancsa.

A VT20/IV számítógépen külön erre a célra írott program küldi az úgynevezett procedura file-t ilyen bináris blokkformátumban az RS 232 vonalra.

A load parancs alkalmazása közben a debugger konzol funkciója nem lehet az aszinkron vonalon, ha P1 = 0. (Ld. még Y parancs.)

5.16.2. –LOAD magnókazettáról

A LOAD parancsnál megadható programnév vagy 0 hosszúságú név.

Ha nincs név megadva, akkor a debugger a szalagon az először megtalált programfile-t tölti be, egyébként pedig a BASIC-hez hasonlóan megkeresi a megadott nevű programot.

Ha a betöltött program a debugger segítségével volt kimentve, akkor a program a kimentéskor megadott címre töltődik, és a programszámláló felveszi a kimentéskor megadott indítási címet (ha az nem 0 volt). Ha az indítási cím 0, a programszámláló változatlan marad. BASIC programok vagy egyéb módon kimentett programok mindig a 19EFH (6639) címre töltődnek, a programszámláló változatlan marad.

A debugger LOAD segítségével betölthetők a BASIC vagy BASIC + gépi kódú programok is, amelyek a BASIC-be való visszatérés esetén normál módon indíthatók, de az így betöltött programok ki is menthetők változatlan formában a kazettára, felfrissítés céljából.

5.17. S–Save program

*S P1 P2 P3 P4 <név>

A P1 címtől P2 címig elhelyezkedő program kimentése a P4 által meghatározott perifériára

P4 = 0 – RS 232 aszinkron vonal.
P4 = 1 – magnókazetta v. floppy diszk.

P3 a program indulási címe, amely a load parancs után aktualizálódik.

5.17.1. Save az RS 232 interface-re

A parancs feltételezi, hogy a vonal másik végén a cross-gép helyezkedik el, és abban egy olyan program fut, amely a mentett programot saját könyvtárába fölírja.

A save parancs segítségével kiírt programot a load paranccsal tölthetjük be.

5.17.2. Save magnókazettára

A magnókazettára történő SAVE-nek 3 funkciója van:

1. Belövés alatt álló programok, programrészletek kimentése adott címtől adott címig, megadott indítási címmel.
2. BASIC + gépi kódú programok kimentése a 19EFH (6639) címtől adott címig, indítási cím nélkül.
3. A debugger LOAD-dal betöltött program kimentése változatlan formában, kazetta felfrissítés céljából.

A három funkció közül a paraméterezéssel választhatunk.

Az 1. mentéshez (amikor a program csak a debugger LOAD-dal tölthető vissza) meg kell adni a program kezdőcímét, végcímét, nevét, és megadható az indítási cím is. Ilyenkor a kezdőcím is és az indítási cím is felíródik a program fejblokkjára.

A 2. mentéshez (BASIC alatt is betölthető program) a kezdőcím 0, SPACE, vagy 19EFH, a végcímet meg kell adni, az indítási cím 0 (SPACE).

Ilyenkor megadható az autoindítás is, a 1707H címre 0FFH-t kell írni.

A 3. mentéshez (debugger LOAD után) az indítási cím, kezdőcím és a végcím közömbös, a név hossza kötelezően 0 (csak SPACE-t kell leütni).

A SAVE-vel nem menthető a debugger munkaterülete, az operációs rendszer munkaterülete és a P3 RAM.

A LOAD és a SAVE parancsoknál a magnót ugyanúgy kell kezelni, mint a BASIC SAVE és LOAD-nál.

A SAVE és LOAD parancsnál különböző operációs rendszer hibák fordulhatnak elő, ezeket a debugger

CASSETTE OR FLOPPY ERROR XX

üzenettel jelzi, ahol XX a hexadecimálisan kiírt hibakód:

F5 (245) : STOP-ot (CTRL+ESC) nyomtak
F1 (241) : hiba a CREATE alatt
F0 (240) : hiba az OPEN alatt
EF (239) : hibás file név
EE (238) : a csatorna nincs megnyitva
ED (237) : túl sok adat
EC (236) : file vég (EOF)
EB (235) : már nyitott a csatorna
EA (234) : CRC hiba olvasásnál
E9 (233) : nincs nyitott file
E8 (232) : ellenőrzési hiba
E7 (231) : belső hiba
E6 (230) : a védelem megsértése
E5 (229) : hibás blokkszám

Floppy diszkes rendszernél a LOAD és SAVE műveletek automatikusan a floppy diszkre vonatkoznak, a filenevek megadását a floppys BASIC rendszereknél megismert módon kell végezni.

5.18. Y–konzol funkció elágazás

* Y P1

A debugger konzol funkciója alaphelyzetben a TVC billentyűzetéhez és képernyőjéhez van rendelve. Ez azonban zavarhatja azokat a programokat, amelyek maguk is használják a képernyőt és/vagy a billentyűzetet. Ezért lehetőség van arra, hogy a konzol funkciót áthelyezzük az RS 232

interface-re kötött displayre. Ez a display lehet a cross-gép is, ha egy olyan program fut benne, amely displayként működteti.

P1=0 érték a konzol funkciót a TVC-hez rendeli.

P1 <> 0 érték az RS 232 vonalat inicializálja, és P1 tartalmazza az inicializálási adatokat. Egyidejűleg a konzol funkciót az aszinkron vonalhoz rendeli.

P1 értéke:

3.–0. bit: adatátviteli sebesség beállítás lehetséges értékei

0	110	bit/sec
1	250	bit/sec
2	300	bit/sec
3	600	bit/sec
4	1200	bit/sec
5	2400	bit/sec
6	4800	bit/sec
7	9600	bit/sec
8	19200	bit/sec

4. bit 0/1 = 7/8 adatbit
5. bit 0/1 = paritásbitképzés tiltása/engedélyezése
6. bit 0/1 = páratlan/páros paritás
7. bit 0/1 = 1/2 stop bit
8. bit 0–6 = bitek érvényesítése (kötelezően 1)

Pl.: * Y 194

1200 Bps, 8 adatbit, paritásképzés tiltva, 2 stop bit. Ez a rendszer alapbeállítása is az RS 232 interface-re vonatkozólag.

5.19. O–Warm reset

Az O parancs hatására a debugger warm resetet hajt végre.

6. OPCIÓK

A parancsok végrehajtásával kapcsolatban néhány dolog a debugg munkaterületén elhelyezett flag segítségével módosítható. Ilyen módosít elvégezhető programból is, de az M parancs segítségével is. Az alábbiakb ezeket a lehetőségeket ismertetjük.

CURFL – Cursor villogjon! (7F04H)

Alapbeállítás: 255. Ha 0-t írunk ebbe a rekeszbe a cursor eltüntethető.

VTILT – vesszős regiszterek kiírásának tiltása

(VTILT=7F07H)

Alapbeállítás: 0, azaz a vesszős regiszterek töréspont, megállás, keze megszakítás esetén kiíródnak.

Célja, hogy azok a programok, amelyek nem használják a vess regisztereket, kevesebb képernyőhelyet használhassanak, kevesebb ic vegyen igénybe a képernyő ROLL vagy vonali kommunikáció, áttekinthetőbb kép legyen a programozó előtt. Gyakorlatilag a 4 állapotinformáció második 2 sorát tiltjuk ezzel a flaggel, ebben találhat stack pointer is. (Kétszínű üzemmódban az állapotinformáció 2 soros, a sort tiltjuk.)

WAITFL – állapotkiírás előtti várakozás

(WAITFL=7F08H)

Olyan gépen, ahol nincs RS 232 vonal, és képet manipuláló progran akarunk követni, a CPU állapot kiírása töréspont esetén elrontaná program által a töréspontig kialakított képet. Ilyen esetekre számít lehetőség van arra, hogy megállási címre való ráfutás, töréspont és kezel megszakítás esetén a CPU állapot azonnali kiírását letiltsuk. Letilt céljából a WAITFL rekeszbe 0-tól különböző értéket kell írni.

Letiltás esetén az állapotkiírás előtt a debugger egy kb. 1 sec. hosszú, 440 Hz frekvenciájú sípszóval jelzi, hogy töréspontra futott, illetve hogy állapotkiírás előtt áll. Egy tetszőleges billentyű leütése után ugyanúgy kiírja az állapotot, mint tiltás nélkül. A leütött billentyű „beszámít" a következő parancsba. Pl. az I billentyű egyúttal egy új utasítás végrehajtását is kiváltja. Tehát letiltott állapotban is „rá lehet támaszkodni" az I billentyűre.

A flag akkor is hatásos, ha a konzol az RS 232 vonalhoz van rendelve. A sípszó ilyenkor ott hallható, feltéve, hogy a vonalra kötött display értelmezi a 07H (BELL) kódot.

Szóközt kell ütni, ha nem akarunk új parancsot adni.

ENIT — Enable interrupt flag

(ENIT=7F0BH)

Kezelői megszakítás engedélyezése. Futó programok kezelői beavatkozással megszakíthatók: a CTRL és az ALT billentyű egyidejű megnyomása a futó program megszakadását és a CPU állapot kiírását idézi elő. Ennek a flagnak a szerepe, hogy a kezelői megszakítás hatástalan legyen, ha a program magában a debuggerben tartózkodik. Program induláskor mindig 255-re állítódik be, ezért modify paranccsal csak nem 0 értékről 0 értékre állítható, ez azonban a debuggert megszakíthatóvá teszi. A debugger megszakítása stack túlmélyülést és a CPU állapot elfelejtését okozza, ezért nem tanácsos módosítani.

Programból van lehetőség a tiltásra. Szükség lehet rá, hogy bizonyos kritikus — már belőtt — programrészek futási idejére letiltsuk a megszakítás lehetőségét.

7. EGYÉB TUDNIVALÓK

A debugger az RST 28H utasítást (kódja: 0EFH) használja fel a követés megvalósításához, ezért a USER RAM 28H, 29H, 2AH címe más célra nem használható.

A CPU 1-es IT módban dolgozik, ezért a 38H címen levő utasítások fogadják az IT-t, emiatt ez a cím másra nem használható. Ugyanebből következik, hogy a ROM kazetta a P0-ra csak DI állapotban kapcsolható.

Ugyancsak az 1-es IT módból következő tudnivaló, hogy az I regiszter szabadon felhasználható.

Felelős kiadó: Dr. Baráth Csaba

Készült a Forma-Art Kiadó gondozásában
Felelős szerkesztő: Csermák Antal
Megjelent 1000 példányban
Terjedelem: 2 ív (A/5)
86.0104 Forma-Art Nyomda
Felelős vezető: Lukácsevich Sándor

VIDEOTON

ELEKTRONIKAI VÁLLALAT
SZÁMÍTÁSTECHNIKAI GYÁRA