Tóth Ferenc

A HT–1080Z iskolaszámítógép néhány szoftvertulajdonsága

A HT–1080Z iskolaszámítógépek első példányai már csaknem két éve működnek a magyar középiskolákban. Ez idő alatt kialakult a felhasználók egy igényes csoportja, akik nem elégednek meg a BASIC nyújtotta lehetőségekkel, és túl akarnak lépni annak korlátain. Ez a legegyszerűbb eszközökkel a gépi kódú programozás révén valósítható meg.

Bevezetés

Ebben a cikkben a gépi kódú programok készítéséhez nem feltétlenül szükséges, de igen hasznos lehetőségekkel foglalkozunk. Gépi kódú programok megfogalmazásakor gyakran fordulnak elő olyan részfeladatok, amelyeknek korrekt megoldását a HT–1080Z gép BASIC értelmező fixtárolója szubrutin formájában tartalmazza. Az ilyen szubrutinok beépítése a felhasználó gépi kódú programjába olyan bonyolult problémák megoldására is reális esélyt ad, amire egyébként kezdő programozók nem lennének képesek.

Itt kívánjuk megjegyezni, hogy cikkünk egy sor olyan ismeretet is feltételez, amely nem kizárólag a HT–1080Z gép sajátossága. A gépi kódú programozás elképzelhetetlen a gépet vezérlő Z80 típusú mikroprocesszor logikai felépítésének és gépi utasításkészletének ismerete nélkül. Ezeket részletesen tárgyalja például az Ipari Informatikai Központ gondozásában megjelent könyvsorozat, melynek témánk szempontjából fontosabb köteteit az irodalom ([1], [2]) tartalmazza. A Z80-as mikroprocesszor programozására vonatkozó módszerekkel, alapismeretekkel például az Ötlet c. folyóiratban megjelent cikksorozat foglalkozik [3]. Szükséges továbbá a HT–1080Z-re vonatkozó néhány hardver információ (memóriatérkép, perifériacímek, a képernyő és a billentyűzet kezelési módja stb.), amelyet a Magyar Elektronikában korábban publikáltunk [4]. Természetesen a BASIC értelmező használatát is alaposan kell ismerni [5]. Ezeket az információkat a terjedelem korlátai miatt itt nem ismételjük meg.

A gépi kódú programozók által felhasználható, általános érdeklődésre számot tartó szubrutinok három fő típusba sorolhatók:

1. aritmetikai és matematikai szubrutinok
2. adatkonverziós és adatmozgató eljárások
3. bemeneti- kimeneti eszközök kezelőprogramjai

Aritmetikai és matematikai szubrutinok

Ezek a programrészletek lehetővé teszik, hogy alapműveleteket (összeadás, kivonás, szorzás, osztás, összehasonlítás) végezzünk két azonos típusú operandus között (a "típus" itt a BASIC értelmező egyik adatábrázolási módja a következők közül: egész, egyszeres pontosságú lebegőpontos és kétszeres pontosságú lebegőpontos), továbbá lehetőséget adnak arra, hogy a BASIC-ből ismert matematikai függvényeket gépi kódú programunkba is beépíthessük. Ez a rész feltételezi a BASIC számábrázolási módjainak részletes ismeretét (lásd irodalom [4]).

Az alábbiakban közöljük az aritmetikai és függvényszubrutinok munkaterületeire vonatkozó általános tudnivalókat. A BASIC értelmező két, egyenként 8 byte terjedelmű munkaváltozót ("regisztert") használ. Nevezzük ezeket REG1-nek és REG2-nek. A REG1 tartalmának értelmezése attól függ, hogy milyen típusú számértéket tárolunk benne (1. ábra). A REG1 a 411DH-tól 4124H-ig tartó RAM területet foglalja el. (A memóriacímeket hexadecimális formában közöljük, mivel a gépi kódú programokban ez a kézenfekvő.)

Cím	Egész típusú szám	Lebegőpontos szám	
		Egyszeres pontosságú	Kétszeres pontosságú
411D			mantissza alsó byte
411E			mantissza
411F			mantissza
4120			mantissza
4121	alsó byte	mantissza alsó byte	mantissza
4122	felső byte	mantissza	mantissza
4123		mantissza felső byte	mantissza felső byte
4124		exponens	exponens

1. ábra

Az 1. ábrán üresen hagyott rovatoknak az adott számábrázolási módban nincs jelentésük. A matematikai szubrutinok részére meg kell adni a 40AFH címen található típusbyte segítségével (2. ábra), hogy a REG1 milyen típusú számot tartalmaz.

A REG2-t csak a kétszeres pontosságú aritmetikai funkciók használják (csak ilyen típusú adatot tartalmazhat), ezért külön típusbyte nem tartozik hozzá. A REG2 a memóriában a 4127H címtől a 412EH címig helyezkedik el. Elrendezése azonos az 1. ábra kétszeres pontosságú adatábrázolást mutató oszlopával.

Az alábbiakban megadjuk a négy alapművelet különböző adatábrázolású változatainál a műveletek operandusainak és az eredmény keletkezésének helyét (3. ábra). (A táblázatban a DE és a HL betűjelzés a Z80 mikroprocesszor megfelelő regiszterpárjaira utal, a (BCDE) jelölés a BC és DE regiszterpárból formált 4 byte méretű regisztert jelenti.)

Kód	REG1 adattípus
02H	egész
03H	karakterlánc
04H	egyszeres pontosságú lebegőpontos
08H	kétszeres pontosságú lebegőpontos

2. ábra

Egész operandusú műveletek

Művelet	Operandusok	Eredmény
Összeadás	HL+ DE	HL
Kivonás	HL– DE	HL
Szorzás	HL•DE	HL
Osztás	DE/HL	REG1

Egyszeres pontosságú műveletek

Művelet	Operandusok	Eredmény
Összeadás	REG1 + (BCDE)	REG1
Kivonás	REG1 – (BCDE)	REG1
Szorzás	REG1 • (BCDE)	REG1
Osztás	REG1 / (BCDE)	REG1

Kétszeres pontosságú műveletek

Művelet	Operandusok	Eredmény
Összeadás	REG1 + REG2	REG1
Kivonás	REG1 – REG2	REG1
Szorzás	REG1 • REG2	REG1
Osztás	REG1 / REG2	REG1

3. ábra

A példákban szereplő Z, P betűk a Z80 jelzőbitjeit jelentik.

Aritmetikai szubrutinok

Egész összeadás **CALL 0BD2H**

DE tartalmát hozzáadjuk HL tartalmához. Az eredmény HL tartalmát írja felül, DE változatlan marad. Ha az egész típusú számábrázolás határait meghaladó eredmény adódik, DE és HL értéke egyszeres pontosságú lebegőpontos típusúra konvertálódik, majd az összeadást így végzi el. Ilyenkor az eredmény REG1-ben keletkezik, a típusbyte értéke 04H lesz. A szubrutin hívása előtt a típusbyte 02H (egész) értékű legyen. Ha az egész típusból túlcsordulás nincs, a típusbyte a művelet után 02H lesz.

Példa: Tegyük fel, hogy N1 és N2 tartalma a két operandus.

```
.
.
.
LD      A,2             a típusbyte feltöltése
LD      (40AFH), A      02H (egész) értékkel
LD      HL, (N1)        az operanduson
LD      DE, (N2)        elhelyezése
CALL    0BD2H           összeadás HL: = DE+HL
LD      A,(40AFH)       típusbyte megvizsgálása
CP      2               Z = 0 : egész; Z = 1 : lebegőpontos
```

Egész kivonás **CALL 0BC7H**

Kivonja DE tartalmát HL tartalmából. A különbség HL-ben keletkezik, DE változatlan marad. Ha a kivonás nem végezhető el az egész típusú számábrázolás keretében, az operandusok egyszeres pontosságú lebegőpontos típusra konvertálódnak, majd a kivonást így végzi el. A különbség ilyenkor a REG1-ben keletkezik. A típusbyte értéke a művelet előtt 02H legyen, a művelet után az eredmény típusától függő értéket kap.

Példa: Tegyük fel, hogy N1 tárolja a kisebbítendőt, N2 a kivonandót.

```
.
.
.
LD      A,2             típusbyte beállítása
LD      (40AFH),A       egész típusra
LD      HL, (N1)        az operandusok
LD      DE, (N2)        elhelyezése
CALL    0BC7H           kivonás HL: = HL – DE
LD      A,(40AFH)       típusbyte megvizsgálása
CP      2               Z = 0 : egész, Z = 1:lebegőpontos
.
.
.
```

Egész szorzás **CALL 0BF2H**

Megszorozza HL-t DE-vel. A szorzat HL-ben keletkezik, DE változatlan marad. Ha a szorzat nem ábrázolható egész típusban, az operandusok egyszeres pontosságú lebegőpontosra konvertálódnak, majd a szorzást így végzi el. A szorzat ilyenkor a REG1-ben keletkezik. A típusbyte értékét a művelet előtt egész típusúra (02H) kell beállítani, a művelet után értéke az eredmény típusának megfelelő.

Példa: Tegyük fel, hogy N1 tárolja a számlálót, N2 a nevezőt.

```
.
.
.
LD     A,2          típusbyte beállítása
LD     (40AFH),A    egész típusra
LD     HL,(N1)      az operandusok
LD     DE,(N2)      elhelyezése
CALL 0BF2H          szorzás HL: = HL * DE
LD     A,(40AFH)    típusbyte megvizsgálása
CP     2            Z = 0 : egész, Z = 1:lebegő-
                    pontos
.
.
.
```

Egész osztás **CALL 2490H**

Elosztja DE-t HL-lel. Mindkét érték egyszeres pontosságú lebegőpontosra konvertálódik az osztás elvégzése előtt. A hányados a REG1-ben keletkezik, a típusbyte értéke az egész típusnak megfelelő lesz. DE és HL tartalma elvész.

Példa: Tegyük fel, hogy N1 és N2 a két operandust tartalmazza.

Egész összehasonlítás **CALL 0A39H**

Összehasonlítja DE és HL értékét. DE és HL értéke nem változik. Az összehasonlítás eredménye az A regiszterben keletkezik ("A" a Z80 CPU akkumulátora) az alábbiak szerint:

```
DE > HL   A = −1 (FFH)
DE < HL   A = +1
DE = HL   A = 0
```

Példa: Tegyük fel, hogy N1 és N2 tartalma a két operandust tartalmazza.

```
LD     DE,(N1)      az operandusok
LD     HL,(N2)      elhelyezése
CALL 0A39H          A értéke beáll DE és HL tar-
                    talma szerint
.
.
.
```

Egyszeres pontosságú lebegőpontos összeadás **CALL 0716H**

Összeadja a BC-ben és DE-ben tárolt — a továbbiakban (BCDE)-vel jelölt — egyszeres pontosságú lebegőpontos értéket a REG1-ben tárolt egyszeres pontosságú lebegőpontos számmal. Az összeg a REG1-ben keletkezik.

Példa: Tegyük fel, hogy P1 és P2 egyszeres pontosságú lebegőpontos számokat tároló 4 byte méretű tárolóhelyek kezdetére mutat.

```
.
.
.
LD     HL,P1        a P1 helyen levő számot
CALL   09B1H        REG1-be tölti
LD     HL,P2        a P2 helyen levő számot
CALL   09C2H        BCDE-be tölti
CALL   0716H        REG1: = REG1+(BCDE)
```

(A 09B1H és 09C2H címen levő szubrutinokat lásd az "Adatmozgató eljárások"-nál.)

Egyszeres pontosságú lebegőpontos kivonás **CALL 0713H**

Kivonja REG1-ből a (BCDE)-t, és az eredményt REG1-be teszi.

Példa: Mint az egyszeres pontosságú lebegőpontos összeadásnál, csak a CALL 0716H helyett CALL 0713H áll.

Egyszeres pontosságú lebegőpontos szorzás **CALL 0847H**

Összeszorozza (BCDE)-t REG1 tartalmával. Az eredményt REG1-be teszi.

Példa: Mint az egyszeres pontosságú összeadásnál, csak a CALL 0716H helyett CALL 0847H áll.

Egyszeres pontosságú lebegőpontos osztás **CALL 08A2H**

Elosztja (BCDE)-t a REG1-ben levő egyszeres pontosságú számmal. A hányadost REG1-be teszi.

Példa: Mint az egyszeres pontosságú összeadásnál, csak a CALL 0716H helyett CALL 08A2H áll.

Egyszeres pontosságú lebegőpontos összehasonlítás **CALL 0A0CH**

Összehasonlítja (BCDE)-t a REG1-ben levő egyszeres pontosságú számmal. Az összehasonlítás eredménye az A akkumulátorban keletkezik az alábbiak szerint:

```
(BCDE) > REG1   A = −1 (FFH)
(BCDE) < REG1   A = +1
(BCDE) = REG1   A = 0
```

Példa: Mint az egyszeres pontosságú összeadásnál, csak a CALL 0716H helyett CALL 0A0CH áll.

Kétszeres pontosságú lebegőpontos összeadás **CALL 0C77H**

Összeadja a REG1-ben és a REG2-ben tárolt kétszeres pontosságú lebegőpontos számokat. Az összeg REG1-ben keletkezik.

Példa: Tegyük fel, hogy P1 és P2 kétszeres pontosságú lebegőpontos számokat tároló 8 byte méretű tárolóhelyek kezdetére mutat.

```
LD      A,8             típusbyte beállítása
LD      (40AFH),A       kétszeres pontosságú (08H)
                        értékre
LD      DE,P1           1. operandus címe DE-be
LD      HL,411DH        REG1 címe HL-be
CALL    09D3H           áthelyezi az 1. operandust
                        REG1-be
LD      DE,P2           2. operandus címe DE-be
LD      HL,4127H        REG2 címe HL-be
CALL    09D3H           áthelyezi a 2. operandust
                        REG2-be
CALL    0C77H           REG1: = REG1+REG2
```

(A 09D3H címen levő szubrutint lásd az "Adatmozgató eljárások"-nál.)

Kétszeres pontosságú lebegőpontos kivonás **CALL 0C70H**

Kivonja a REG 2-ben levő kétszeres pontosságú lebegőpontos számot a REG1-ben levőből. A különbség REG1-ben keletkezik.

Példa: Mint a kétszeres pontosságú összeadásnál, de a CALL 0C77H helyett CALL 0C70H áll.

Kétszeres pontosságú lebegőpontos szorzás **CALL 0DA1H**

Összeszorozza a REG1-ben és a REG2-ben tárolt kétszeres pontosságú lebegőpontos számokat, és a szorzatot a REG1-be helyezi.

Példa: Mint a kétszeres pontosságú összeadásnál, de a CALL 0C77H helyett CALL 0DA1H áll.

Kétszeres pontosságú lebegőpontos osztás **CALL 0DE5H**

Elosztja a REG1-ben tárolt kétszeres pontosságú számot a REG2-ben levővel. A hányados a REG1-ben keletkezik.

Példa: Mint a kétszeres pontosságú összeadásnál, de a CALL 0C77H helyett 0DE5H áll.

Kétszeres pontosságú lebegőpontos összehasonlítás **CALL 0A78H**

Összehasonlítja a REG1-ben és a REG2-ben tárolt kétszeres pontosságú lebegőpontos számokat. REG1 és REG2 nem változik. Az összehasonlítás eredménye az A akkumulátorban keletkezik a következők szerint:

```
REG1 > REG2     A = −1 (FFH)
REG1 < REG2     A = +1
REG1 = REG2     A = 0
```

Példa: Mint a kétszeres pontosságú lebegőpontos összeadásnál, de CALL 0CC7H helyett CALL 0A78H áll.

Matematikai szubrutinok

Az alábbi szubrutin-belépési pontok megadása lehetővé teszi, hogy a felhasználó gépi kódú programjába beépítse a BASIC standard függvényeit. A szubrutinok feltételezik, hogy híváskor a típusbyte (40AFH) értéke az operandus típusának megfelel a 2. ábra szerint, továbbá hogy az operandus a REG1-ben van. A függvények értelmezési tartománya és hibaüzenetei azonosak a megfelelő BASIC függvényekével.

Abszolút érték **CALL 0977H**

A REG1-be az ott talált szám abszolút értékét teszi. Ha a REG1 tartalma egész típusú, és meghaladja a 2^{15} értéket, a REG1 tartalma előbb egyszeres pontosságú lebegőpontosra konvertálódik, és így végzi el a műveletet. A típusbyte (40AFH) értéke az eredmény típusának megfelelő.

Példa: Tegyük fel, hogy N egyszeres pontosságú lebegőpontos szám kezdetére mutat.

```
LD      A,4             típusbyte beállítása
LD      (40AFH),A       egyszeres pontosságú lebe-
                        gőpontos értékre
LD      HL,N            N érték betöltése
CALL    09B1H           REG1-be
CALL    0977H           (REG1) abszolút értéke
                        REG1-be
```

Ez a példa külön megjegyzés nélkül a további függvényeknél is használható a hívási cím értelemszerű helyettesítésével.

Arcus tangens függvény **CALL 15BDH**

A REG1-ben lebegőpontos formában elhelyezett tangens érték inverzét számítja ki. A radiánban értendő függvényérték egyszeres pontosságú lebegőpontos alakban a REG1-ben keletkezik.

Cosinus függvény **CALL 1541H**

A REG1-ben elhelyezett, radiánban értelmezett, lebegőpontos formátumú szögérték cosinusát számítja ki, és azt egyszeres pontosságú lebegőpontos számként a REG1-ben helyezi el.

Egészrész függvény **CALL 0B37H**

A REG1-ben található lebegőpontos szám egész részét állítja elő. (Az egészrész függvény definíciója szerint negatív számra is a nála kisebb – abszolút értékben nagyobb vagy egyenlő! – egész szám adódik.) Az eredmény típusát a típusbyte (40AFH) jelzi.

Egész típusra konvertálás **CALL 0B26H**
A REG1-ben levő lebegőpontos szám törtrészét csonkítja, az egész típusú eredményt a REG1-be helyezi. A típusbyte (40AFH) értékét egész típusúra állítja. (Megfelel a BASIC FIX(N) függvénynek.)

Hatványozás (x^y) **JP 13F2H**
Egyszeres pontosságú lebegőpontos számként ábrázolt pozitív x értékkel és tetszőleges egyszeres pontosságú lebegőpontos y értékkel kiszámítja az x^y hatvány értékét. Az x értékét a függvény hívása előtt a veremtárban (stack), y értékét a REG1-ben kell elhelyezni. A hatvány a REG1-ben keletkezik. (Az x elrendezése a veremtárban: legmélyebben legyen a mantissza a legkisebb helyi értékű byte, legfelül az exponens.)
A szubrutin az $x^y = e^{y(\ln x)}$ azonosság alapján működik.
Példa: Az x és y operandusok legyenek az XR és YR címeken.

```
        .
        .
        .
        LD    BC,CONT     tárolja a veremben a
        PUSH  BC          hatványszubrutin le-
        LD    A,4         futása utáni folytatási
                          címet beállítja a típus-
                          byte értékét
        LD    (40AFH),A   egyszeres pontosságú
                          lebegőpontosra
CONT:   LD    HL, XR      REG1-be helyezi az
                          XR-ben levő
        CALL  09B1H       lebegőpontos számot
        CALL  09A4H       a REG1 tartalmát a ve-
                          rembe teszi
        LD    HL,YR       a REDG1-be helyezi
                          az YR-ben levő
        CALL  09B1H       lebegőpontos számot
        JP    13F2H       végrehajtjuk a hatvá-
                          nyozást folytatás
```

(Megjegyzés: A szokatlan szubrutinkezelés magyarázata: a 13F2H című szubrutin a verem tetején várja az egyik operandust jelentő lebegőpontos számot. A szubrutint így nem hívhatjuk CALL utasítással, mivel az a verem tetejére a saját visszatérési címét tenné. Így a példaprogram első két sora a későbbi visszatérési címet elhelyezi a veremben, a CALL 09A4H sor pedig föléhelyezi az x operandust. Ezáltal teljesül a hatványozó szubrutin által megkívánt veremsorrend. A JP 13F2H feltétlen ugróutasítás ezen a sorrenden nem változtat. A hatványozó szubrutin viszont elveszi a veremből az x operandust, így a szubrutinból való kilépésekor a verem tetején már az a visszatérési cím van, amelyet a mintaprogram első két sora helyezett el. A szubrutint záró RET utasítás hatására ezen a címen folytatódik a program.)

Négyzetgyök függvény **CALL 13E7H**
A REG1-ben tárolt tetszőleges típusú szám négyzetgyökét számítja ki, és azt egyszeres pontosságú lebegőpontos számként a REG1-be helyezi.

Sinus függvény **CALL 1547H**
A REG1-ben elhelyezett egyszeres pontosságú lebegőpontos, radiánban értelmezett szám sinusát számítja ki, és az egyszeres pontosságú lebegőpontos eredményt a REG1-be teszi.

Tangens függvény **CALL 15A8H**
A REG1-ben elhelyezett egyszeres pontosságú lebegőpontos, radiánban értelmezett szám tangensét számítja ki, és az egyszeres pontosságú lebegőpontos eredményt a REG1-be teszi.

Természetes alapú hatvány **CALL 1439H**
A REG1-ben elhelyezett egyszeres pontosságú lebegőpontos számot a természetes logaritmus alapszámára (e = 2,71828...) mint alapra emeli. A hatványt egyszeres pontosságú lebegőpontos számként a REG1-ben helyezi el.

Természetes logaritmus **CALL 0809H**
A REG1-ben elhelyezett egyszeres pontosságú lebegőpontos szám természetes logaritmusát számítja ki, és az egyszeres pontosságú lebegőpontos eredményt a REG1-be teszi.

Véletlenszám-generátor

Az alábbi két, véletlenszám-előállítással kapcsolatos szubrutin használatának megértéséhez ismernünk kell a véletlenszám-generálás bizonyos sajátosságait.
A véletlenszámokat valójában álvéletlen számsorozatot adó algoritmus állítja elő. Ez azt jelenti, hogy egy adott véletlenszámot mindig azonos másik véletlenszám követ. Generálásukról a "véletlenszám-generátor" függvény gondoskodik. Az álvéletlen jelsorozat megszakítható, mintegy "véletlenebb" jellegűvé tehető a "véletlen inicializálás" szubrutin alkalmazásával, amely az álvéletlen számsorozat kezdőértékét a Z80 CPU memóriafrissítő számlálójának pillanatnyi tartalmától függően módosítja.
A frissítő számláló tartalma a programmal rendszerint semmilyen determinisztikus kapcsolatban nincs, ezáltal az álvéletlen számsorozat kezdőértéke véletlennek tekinthető. Ez természetesen nem változtat az álvéletlen számsorozat két determinisztikus tulajdonságán:
— a sorozat periodikus,
— adott számérték után mindig azonos másik számérték következik.

Véletlenszám-generátor **CALL 14C9H**
Álvéletlenszámot állít elő 0 és 1 vagy 1 és N között a REG1-ben elhelyezett számértéktől függően. Az álvéletlen-számérték a REG1-ben keletkezik egész típusú szám formájában. (A típusbyte ennek megfelelően áll be.) A REG1-ben elhelyezett számérték a visszaadott álvéletlenszám tartományát határozza meg.

REG1 = 0 hatására a szám 0 és 1 közötti lesz,
REG1 = 1 hatására a szám 1 és a REG1-ben elhelyezett szám egész része közé esik.

(A 14C9H címen levő függvény a BASIC RND (X)-nek felel meg.)

Véletlen inicializálás **CALL 01D3H**
Az álvéletlenszámokat előállító "véletlenszám-generátor" függvény egymás utáni hívásainál az algoritmus kezdőértékét tároló 4 byte méretű regiszter egy közbenső byte-ját (40ABH) a memóriafrissítő számláló pillanatnyi tartalmából veszi. (Funkcionálisan megfelel a BASIC RANDOM utasításának.)

Adatkonverziós és adatmozgató eljárások

Az adatkonverziós eljárások kétirányú konverziót valósítanak meg az egész, egyszeres és kétszeres pontosságú adattípusok között, valamint az ASCII jelsorozattal megjelenített adatok és a gép belső számábrázolási módjai között. A konverziós rutinok feltételezik, hogy a konvertálandó adat a REG1-ben van, és a típusbyte (40AFH) az aktuális típusnak megfelelő. Az eredmény is a REG1-ben keletkezik, és a típusbyte is aktualizálódik.

Lebegőpontosból egész típusba konvertálás **CALL 0A7FH**
A REG1-ben tárolt egyszeres vagy kétszeres pontosságú lebegőpontos számot kerekítés nélkül egész típusúra konvertálja.

Egyszeres pontosságú lebegőpontosra való konvertálás **CALL 0AB1H**
A REG1-ben tárolt egész típusú vagy kétszeres pontosságú lebegőpontosra alakítja át, melyet a REG1-be helyez el. A típusbyte értékét is ennek megfelelően állítja.

Kétszeres pontosságú lebegőpontosra való konvertálás **CALL 0ADBH**
A REG1-ben tárolt egész vagy egyszeres pontosságú lebegőpontos számot kétszeres pontosságú lebegőpontosba alakítja át, ezt a REG1-be helyezi vissza. A típusbyte értéke a kétszeres pontosságú lebegőpontosnak megfelelően áll be.

ASCII jelsorozat egész típusúra való átalakítása **CALL 1E5AH**
Az ASCII karakterlánc alakjában adott szám egész típusú ekvivalensét állítja elő. A karakterlánc kezdőcímét hívás előtt a HL-ben kell elhelyezni. Az eredmény a CPU DE regiszterpárjában keletkezik. A konverzió az első olyan karakternél leáll, amely nem ASCII számjegy.

ASCII jelsorozat binárissá való átalakítása **CALL 0E6CH**
Az ASCII karakterlánc formájában adott számot a gép valamely belső számábrázolási módjába alakítja át az alábbiak szerint. A karakterlánc kezdetét a HL-ben kell megadni. Ha nem tartalmaz tizedespontot vagy E, vagy D exponensjelet, és értéke kisebb, mint 2^{16}, egész típusú számmá konvertálódik. Ha valamelyik feltétel nem teljesül, a karakterlánc egyszeres vagy kétszeres pontosságú lebegőpontos értékké alakul át aszerint, hogy E vagy D exponensjel fordul-e elő benne. A konverzió eredménye a REG1-be kerül, típusát a típusbyte jelzi.

ASCII jelsorozat kétszeres pontosságú lebegőpontosba való átalakítása **CALL 0E65H**
A HL-ben tárolt kezdőcímű ASCII jelsorozatot kétszeres pontosságú lebegőpontosba alakítja át. Az eredmény a REG1-ben keletkezik.

A HL tartalmának konverziója és kijelzése **CALL 0FAFH**
A CPU HL regiszterpárjában tárolt, egész típusúnak tekintett szám ASCII megfelelőjét állítja elő, és a pillanatnyi kurzorpozíciótól kezdve kiírja a képernyőre.

Egész típusú szám ASCII konverziója **CALL 132FH**
A REG1-ben tárolt egész típusú számot ASCII jelsorozattá alakítja át, és elhelyezi a memóriában a HL-ben tárolt kezdőcímtől. Hívás előtt B és C regiszterbe 5-öt kell írni. A CPU regisztereit nem változtatja meg.

Bináris adat ASCII konverziója **CALL 0FBDAH**
A REG1 tartalmát ASCII karakterlánccá konvertálja. A szubrutin futása után HL a karakterlánc kezdőcímére mutat.

Adatmozgató eljárások

Általános adatmozgató rutin **CALL 09D2H**
A típusbyte által specifikált adatot áthelyezi a DE által mutatott címről a HL-ben tárolt címre. Az A, DE és HL regiszterek tartalma változik.

Általános adatmozgató rutin **CALL 09D3H**
Hasonló az előzőhöz, de a HL által mutatott címről a DE által mutatott címre tölti át az adatot.

Egyszeres pontosságú lebegőpontos érték átvitele BCDE-ből REG1-be **CALL 09B4H**
A CPU B, C, D, E regisztereibe tárolt egyszeres pontosságú lebegőpontos számot a REG1-be helyezi. BC és DE nem változik. A típusbyte értékét nem állítja!

Egyszeres pontosságú lebegőpontos érték átvitele a HL által mutatott memóriahelyről REG1-be **CALL 09B1H**
A HL-ben tárolt kezdőcímű egyszeres pontosságú lebegőpontos értéket BC, DE-be, majd onnan a REG1-be helyezi. B, C, D, E, H, L értéke változik.

Egyszeres pontosságú lebegőpontos érték bevitele BC, DE-be **CALL 09C2H**
A HL-ben tárolt kezdőcímű, egyszeres pontosságú lebegőpontos értéket BC, DE-be teszi. Az összes CPU regiszter változik!

Egyszeres pontosságú lebegőpontos érték átvitele REG1-ből BC, DE-be **CALL 09BFH**
A REG1-ben tárolt egyszeres pontosságú lebegőpontos értéket BC, DE-be viszi át. Nem ellenőrzi a típusbyte értékét, így a programozónak kell biztosítania, hogy a REG1 hívásakor valóban egyszeres pontosságú lebegőpontos számot tartalmazzon.

Egyszeres pontosságú lebegőpontos érték átvitele REG1-ből a veremtárba **CALL 09A4H**
A REG1-ben tárolt egyszeres pontosságú lebegőpontos értéket a veremtárba teszi. Az összetett aritmetikai műveletsorok közben részeredmények kimentésére használható.

Karakterlánc-átvitel **CALL 29C8H**
Egy 3 byte méretű vezérlőblokkal jellemzett karakterláncot DE-ben tárolt kezdőcímre helyez át. A vezérlőblokk a HL-ben tárolt címen kezdődik, és a következőket tartalmazza:

1 byte — a karakterlánc hossza,
2. és 3. byte — a karakterlánc eredeti kezdőcíme.

Bemeneti-kimeneti eszközök kezelőprogramjai

Billentyűzetletapogatás **CALL 002BH**
Ez a szubrutin a billentyűzet állapotának figyelésére alkalmas. Ha nincs lenyomott billentyű, a szubrutin visszatérésekor a CPU A akkumulátora 0-t tartalmaz, ha a híváskor valamelyik billentyű lenyomott állapotban van, a visszatéréskor ennek a kódja található az A-ban. A CPU jelző- (F) regisztere A tartalmának megfelelő.
Példa: Az alábbi programrészlet mindaddig vár, amíg nincs lenyomott billentyű.

```
        PUSH DE     DE kimentése
        PUSH IY     IY kimentése
ISMET:  CALL 002BH  billentyűzet lekérdezése
        JR Z,ISMET  ha nincs lenyomott billentyű, ismétel
        POP IY      IY visszaállítása
        POP DE      DE visszaállítása
        .
        .
```

Várakozás billentyűlenyomásra **CALL 0049H**
Ez a szubrutin folyamatosan lekérdezi a billentyűzet állapotát, amíg lenyomott billentyűt nem talál. Ekkor az A regiszterben a lenyomott billentyű kódjával visszatér.

Sorbeolvasás a billentyűzetről **CALL 0361H**
Ez a szubrutin a billentyűzetről beolvas egy karaktersorozatot. A tárolt karaktersorozat kezdetére a HL tartalma utal a szubrutinból való visszatéréskor úgy, hogy a HL regiszterpárban található számérték az első karakter címénél 1-gyel kevesebb. A szubrutinból való visszatérést a NEW LINE vagy a BREAK billentyű lenyomása váltja ki, ez jelzi a karaktersorozat végét. A BREAK billentyűvel való kilépést a jelzőregiszter CARRY (átvitel) bitjének 1-es értéke jelzi. (Sorbeírás közben a visszalépés-billentyű (←) használata megengedett.)

Sorbeolvasás a billentyűzetről **CALL 1BB3H**
Ez a szubrutin lényegében azonos az előzővel, de a szubrutinba való belépést a képernyőn egy ? karakter megjelenése jelzi.

Karakterkijelzés **CALL 0033H**
Kijelzi a képernyőn a pillanatnyi kurzorpozícióban azt a karaktert, amelynek kódját a hívás előtt az A regiszterben helyeztük el.

Karakterkijelzés kurzortovábbítással **CALL 032AH**
Hasonló az előzőhöz, de a kurzorpozíciót a kiírás után megnöveli.

Képernyőtörlés **CALL 01C9H**
Ez a szubrutin törli a képernyőt, és a kurzorpozíciót alaphelyzetbe (a képernyő bal felső sarkába) állítja.

Karakterlánc nyomtatása vagy megjelenítése **CALL 28A7H**
Kiírja a képernyőre vagy kinyomtatja a nyomtatón azt a karaktersorozatot, melynek kezdőcímét híváskor a HL regiszterpárban találja. A karakterlánc végét vagy nullkarakterrel (00H), vagy kocsivissza karakterrel (0DH) kell jelezni. A kívánt kimeneti perifériát a 409CH byte-ba írt kóddal kell megjelölni:
0 – képernyő,
1 – nyomtató.

Karakterlánc nyomtatása vagy megjelenítése **CALL 2B75H**
Ez a szubrutin azonos a 28A7-en kezdődővel, de zárókarakterként csak a nullkaraktert fogadja el, a kocsivissza karaktert a többi karakterhez hasonlóan kiküldi a kimeneti perifériára.

Egy karakter nyomtatása **CALL 003BH**
Kiküldi a nyomtatóra a C regiszterben elhelyezett karakterkódot.

Kazettabemenet, - kimenet

Az alábbi szubrutinok a kazettás egység kezelésére szolgálnak. Használatukkor a 409CH byte tartalmát –1-re (FFH) kell állítani.

Kazettásegység-motor indítása **CALL 0212H**
Ez a szubrutin bekapcsolja az A regiszterben elhelyezett sorszámú kazettás egység motorját.
(A = 0 vagy 1.)

Rekordkezdet felírása a kazettára **CALL 0284H**
Beindítja a kiválasztott kazettás egységet, majd felírja az adatblokkok kezdetén szokásos periodikus jelsorozatot.

Rekordkezdet olvasása kazettáról **CALL 0296H**
Beindítja a kiválasztott kazettás egységet, majd megkeresi az első rekordkezdő jelsorozatot. Ennek a végén (az információs blokk kezdetén) visszatér.

Egy byte írása a kazettára **CALL 0264H**
Felírja a kazettára az A regiszterbe helyezett kódot.

Egy byte olvasása a kazettáról **CALL 0235H**
Leolvassa a kazettáról a soron következő byte-ot és az A regiszterbe teszi.

A fentiekben vázolt szubrutinok szemelvények a HT–1080Z BASIC értelmezőjének gépi kódú programokban felhasználható fontosabb szolgáltatásaiból. Nem törekedtünk teljességre sem a szubrutinok felsorolásában, sem pontos specifikációjuk megadásában. Ez utóbbival kapcsolatban gyakran hagytuk el a szubrutinoknak a processzor állapotára gyakorolt hatásának megadását. Ilyen esetekben a biztonság kedvéért célszerű a fontos regiszterek mentése. A futási idők megadását is mellőztük, mivel cikkünket elsősorban gondolatébresztőnek, kísérletezésre, próbálkozásra serkentő indításnak szántuk. Bízunk abban, hogy a gépi kódú programozással barátkozó érdeklődő olvasók ezt a tájékoztató jellegű információt is jól fel tudják majd használni.

(Megjegyzés: Kéziratom ellenőrzése közben számos elírást fedeztem fel és javítottam. Egy numerikus adatokkal ilyen mértékben telezsúfolt írásban azonban a gondos ellenőrzés ellenére is maradhattak hibák. Ezért az olvasó elnézését kérem. – A szerző.)

IRODALOM

[1] Z80-as sorozat I. rész: CPU. Ipari Informatikai Központ, Budapest, 1982.
[2] Z80-as sorozat VIII. rész: CPU utasításkészlet. Ipari Informatikai Központ, Budapest, 1982.
[3] Ötlet'84: Sorvezető c. rovat sorozata az 1984-es évfolyam 26. számától: Székely Jenő: Gépi kódú programozás.
[4] Tóth Ferenc: A HT–1080Z iskolai számítógép. Magyar Elektronika, 1984. 1.szám, 16–24. old.
[5] HT–1080Z Iskolaszámítógép BASIC kézikönyv.

Csak nyomja a szöveget a nyomásról, meg a hőmérsékletről, bezzeg ha számítógépes játékot játszanánk....