



www.chipcad.hu

Lamár Krisztián

A világ leggyorsabb mikrovezérlője

Az "SX mikrokontroller és alkalmazástechnikája" című tanfolyam
segédanyaga

Lektorálta: Dr. Kónya László

Budapest, 1999.

ChipCAD Elektronikai Disztribúció Kft.
1046 Budapest, Kiss Ernő utca 3.
Tel: 399-42-90, Fax: 399-42-99
e-mail: info@chipcad.hu
<http://www.chipcad.hu>

Tartalomjegyzék

TARTALOMJEGYZÉK.....	1
ÁBRAJEGYZÉK	4
BEVEZETÉS.....	5
1. A VILÁG LEGGYORSABB MIKROVEZÉRLŐJE	6
1.1 Az SX mikrokontroller általános felépítése	6
1.2 Memóriakialakítás.....	9
1.2.1 A programtár felépítése.....	10
1.2.2 A programszámláló	10
1.2.3 A programszámlálót módosító utasítások	10
1.2.4 A verem.....	15
1.2.5 Az adattár felépítése.....	15
1.2.6 A bankok	16
1.2.7 Közvetlen címzés	19
1.2.8 Közvetett címzés	19
1.2.9 Adat kiolvasása a programtárból.....	20
1.2.10 Adattáblázatok készítése	21
1.2.11 A speciális funkciójú regiszterek	22
1.3 I/O portok.....	24
1.3.1 A portok írása és olvasása	25
1.3.2 A portok üzemmódbeállítása	25
1.3.3 A portvezérlő regiszterek működése	26
1.3.4 A portok áramköri felépítése.....	28
1.3.5 A portok két speciális tulajdonsága.....	30
1.3.6 Multi-Input Wakeup/Interrupt (MIWU).....	31
1.3.7 Az analóg komparátor	33
1.4 Időzítő/számláló modulok	34
1.4.1 Az RTCC áramkör	34
1.4.2 Eseményszámláló üzemmód	35
1.4.3 Időzítő üzemmód.....	35
1.4.4 Az RTCC regiszter írása	36
1.4.5 A Watchdog áramkör	36
1.4.6 A Watchdog-áramkör használata	37
1.4.7 Az előosztó.....	37

1.5 Megszakítási rendszer	38
1.5.1 Az RTCC megszakítása	39
1.5.2 A "B port" megszakítása	39
1.5.3 A megszakításkiszolgálás hardver része	40
1.5.4 A megszakításkezelő rutinok lezárása	40
1.5.5 Példa egy megszakításkezelő rutinra.....	42
1.6 Egyéb fontos tulajdonságok.....	43
1.6.1 Konfigurációs biztosítók	43
1.6.2 Oszcillátor típusok	44
1.6.3 A RESET folyamat	46
1.6.4 Energiatakarékos üzemmód	50
1.7 Az SX mikrovezérlő utasításkészlete	50
2. PROGRAMFEJLESZTÉS ÉS A VIRTUÁLIS PERIFÉRIÁK.....	57
2.1 Egy egyszerű mintapélda	57
2.2 Az SX-KEY fejlesztőrendszer.....	59
2.3 Virtuális perifériák.....	63
2.3.1 A virtuális perifériák típusai.....	63
2.3.2 Egy egyszerű virtuális periféria-minta	65
3. BEÁGYAZOTT RENDSZEREK SOROS KOMMUNIKÁCIÓJA	67
3.1 A soros kommunikáció jelentősége	67
3.2 RS-232 aszinkron soros adatátvitel.....	68
3.2.1 RS-232C szabvány	68
3.2.2 RS-232 kommunikációt támogató rutinkönyvtár az SX-hez	74
3.3 Az I²C busz.....	76
3.3.1 Bit átvitel.....	78
3.3.2 Bájtt átvitel	79
3.3.3 Adatforgalom a buszon	80
3.3.4 I ² C kommunikációt támogató rutinkönyvtár az SX-hez	81
3.4 SX mikrokontrolleren futó monitor program.....	82
3.5 Alternatív soros csatlakozó lehetőségek	83
3.6 RS-232 —TTL szintillesztő megoldások.....	84
3.6.1 Egyszerű ellenállásos szintillesztő	84
3.6.2 Tranzisztoros szintillesztő.....	85
3.6.3 FET-es szintillesztő.....	86
3.6.4 Optocsatolt szintillesztő	87

4. A "UNICOMM" PROGRAM	88
4.1 A programmal kapcsolatos tudnivalók:	88
4.2 A program parancssorból való hívása:.....	88
4.3 Parancsértelmező mód	90
4.3.1 A V3.x verziókban értelmezett billentyű-parancsok.....	90
4.4 Terminál emulációs mód.....	93
4.5 A V4.x verziók változásai a V3.x verziókhöz képest	94
4.6 További tervek.....	95
IRODALOMJEGYZÉK	96

Ábrajegyzék

1.1 ábra: Az SX mikrokontroller felépítése	8
1.2 ábra: A "többszintű pipe-line" folyamata.....	9
1.3 ábra: Az ugró utasítások működési mechanizmusa	12
1.4 ábra: A szubrutinhívó utasítások működési mechanizmusa	13
1.5 ábra: Az adattár felépítése.....	17
1.6 ábra: A TTL és CMOS bemenetek transzfer karakterisztikái.....	27
1.7 ábra: A Schmitt Triggeres bemenetek transzfer karakterisztikája	28
1.8 ábra: A "B port" (RB) felépítése	29
1.9 ábra: A MIWU áramkör felépítése	32
1.10 ábra: Az analóg komparátor áramkör felépítése	33
1.11 ábra: Az időzítő/számláló áramkör felépítése.....	38
1.12 ábra: A megszakítási áramkör.....	43
1.13 ábra: Kvarckristály vagy kerámia rezonátor bekötése	44
1.14 ábra: Független külső oszcillátor bekötése	45
1.15 ábra: Külső RC oszcillátor bekötése.....	46
1.16 ábra: Természetes RESET folyamat	47
1.17 ábra: A Vdd és az MCLR lábak összekötve. A tápfeszültség jelváltozási sebessége túl alacsony.....	47
1.18 ábra: Az MCLR lábra jutó jel késleltetése külső RC hálózattal	48
2.1 ábra: Az SX-KEY fejlesztőrendszer hibakresó/nyomkövető üzemmódjának képe.....	60
3.1 ábra: DTE és DCE egységek kapcsolata.....	68
3.2 ábra: Szabványos 25 pontos soros csatlakozó	69
3.3 ábra: RS232 jelszintek	69
3.4 ábra: DTE-DCE összekötő vezetékek.....	70
3.5 ábra: RTS-CTS kézfogásos kapcsolat.....	70
3.6 ábra: Az aszinkron soros adatátvitel elve	71
3.7 ábra: Null-modem: Két DTE összekötése	73
3.8 ábra: Egy soros jelalak	73
3.9 ábra: Az I ² C kommunikáció elve.....	77
3.10 ábra: Bit átvitel, START és STOP feltétel az I ² C buszon	78
3.11 ábra: Bájtt átvitel az I ² C buszon.....	79
3.12 ábra: Különféle típusú adatforgalom az I ² C buszon	81
3.13 ábra: Telefoncsatlakozó-bekötési megoldások	84
3.14 ábra: Ellenállásos RS-232 — TTL szintillesztő	85
3.15 ábra: Tranzisztoros RS-232 — TTL szintillesztő.....	86
3.16 ábra: FET-es RS-232 — TTL szintillesztő.....	86
3.17 ábra: Optocsatolt RS-232 — TTL szintillesztő	87

Bevezetés

Jelenleg a harmadik ipari forradalom korát éljük. Ez az informatika forradalma. Három sokkoló adat ezzel kapcsolatban:

- 1971:** Az első mikroprocesszor megjelenése — **Csak 28 év!**
- 1981:** Az első IBM PC megjelenése — **Csak 18 év!**
- 1999:** A számítógépek részt vesznek az újabb számítógép generációk létrehozásában — **Evolúció?!**

Gondoljuk végig! El tudnánk képzelni ezek nélkül ma az életünket? Természetesen nem! A számítógép úgyszólván minden korábbi technológiát elsöpört és elterjedése egyúttal új értékrendet is teremtett. Soha nem álltak generációk olyan távol egymástól, mint korunkban. A XVII. századi és a XV. századi ember gondolkodásmódja között sokkal kevesebb különbség volt, mint a mai és az ötven évvel ezelőtti ember gondolkodásmódja között.

Ez a szédületes fejlődés egyeseket lenyűgöz, másokat megrémít, de egy biztos, az emberiség többsége tisztában sincs a dolog jelentőségével és veszélyével. Nem is tudják, hogy milyen mértékben beépült életükbe a technika. Ma már a kenyérpirító is processzoros áramkört tartalmaz. A naphosszat számítógép előtt dolgozó milliók többsége sem gondolta még végig, képes lenne-e a munkáját mechanikus írógéppel írni, akár több példányban újra és újra, vagy papír-ceruza módszerrel fejben számolni. Nagy valószínűséggel nem.

Kik tehetnek minderről? Kik azok az emberek, akik ezeket lenyűgöző, de sokszor félelmetes démonokat életre hívták? Természetesen a villamosmérnökök. Nekik kell a féktelenül lelkendezőket lehiggasztani, és a kételkedőket meggyőzni.

Mindezt csak azért írtam le, hogy akik a soron következő fejezetek hatására kapnak kedvet a mikroprocesszoros áramkörök fejlesztéséhez, egy nagyon fontos gondolatot mindig tartsanak szem előtt: **az a célunk, hogy egy jobb világot hagyjunk hátra az utókornak és nem pedig az, hogy egy önműködő, emberektől mentes világot.**

1. A világ leggyorsabb mikrovezérlője

1.1 Az SX mikrokontroller általános felépítése

A mikrokontrollerek, vagy más néven mikrovezérlők — amelyek egy tokban hordozzák egy komplett számítógép összes részét: a központi egységet, az adat és programmemóriát és a perifériákat — kétségtelenül a technikai fejlődés motorjai. Megtaláljuk ezeket számos berendezésben: az autókban, a számítástechnikai eszközökben, háztartási gépekben, a szórakoztató és egyéb elektronikai eszközökben. Számos gyártó és típus létezik, a legjobban elterjedt a Microchip cég PIC családjá, és az Intel MCS-51 családjá. 1996-ban azonban egy új gyártó — és ezzel együtt egy új mikrokontroller — indult hódító útjára. Az amerikai **SCENIX** cég **SX** típusú mikrokontrollere új dimenziókat nyithat meg a beágyazott rendszerek fejlesztésével foglalkozó mérnökök számára.

A CPU kialakítása a MICROCHIP cég PIC kontrolleréhez hasonló, a kompatibilitás érdekében azonos utasításkészlettel, amit kiegészít még 11 új, a hatékonyabb működést biztosító utasítás, és a tok kivezetés-elrendezése is azonos. Például lehetséges a program memóriában lévő programutasítások szavaiból egy ellenőrző összeget készíteni, és ilyen módon bekapcsoláskor az ismételt kiszámított összeget az eredetivel összevetve a programunk épségét ellenőrizni.

A maximális 50 MHz-es órajelel frekvenciával egészen álló állapotig lemehetünk, a tok ilyenkor is helyesen működik. Előnyös a tokban lévő 4 MHz-es órajelet előállító belső R-C oszcillátor felhasználása: ezt az órajelet egy belső, 1:1-től 1:128-ig hét lépésben változtatható belső osztóval leoszthatjuk csökkentve a tok órajelfrekvenciáját (és a fogyasztását.) Ez **a belső oszcillátor csökkenti a kialakított rendszer költségeit** és a kibocsátott elektromágneses sugárzást.

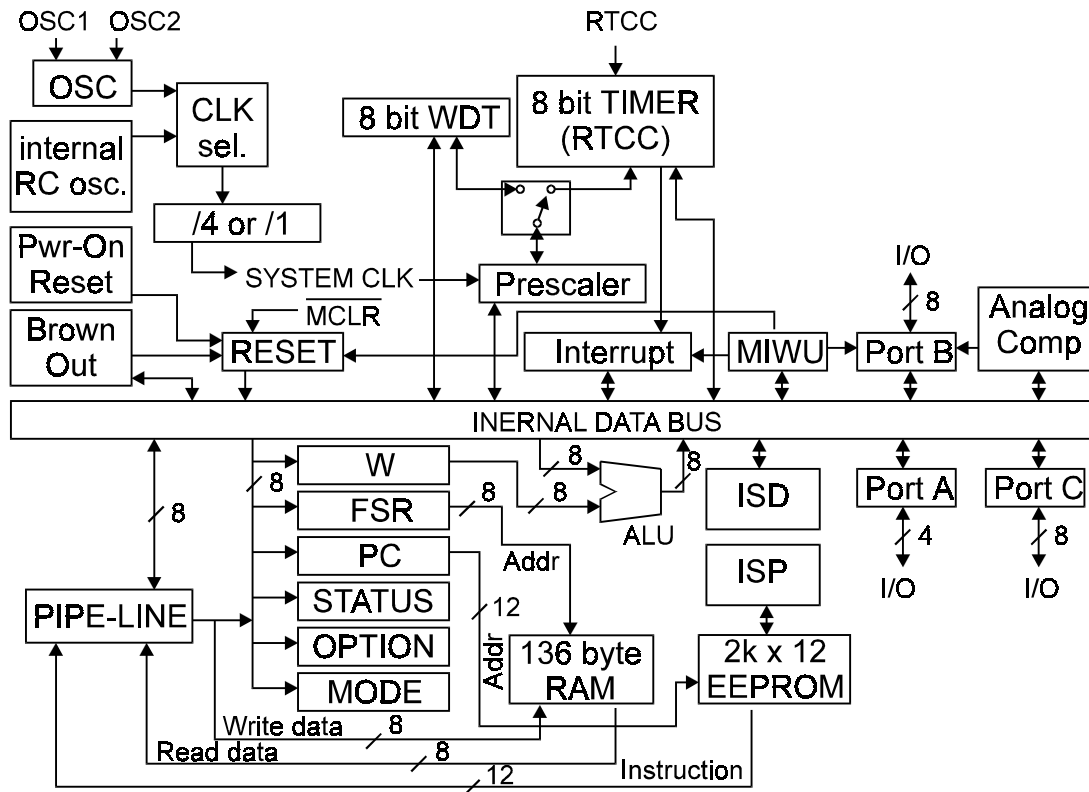
A mikrokontrollerek korlátozott sebessége miatt a perifériák kezelése sok esetben csak önálló periféria egységek felhasználásával lehetséges. Ez sajnos két hátránnyal jár: a lapka mérete a perifériák által elfoglalt további lapkaterület miatt megnövekszik, és feladatspecifikus mikrokontroller családok kialakítását igényli. Ezekben a központi egység, a CPU közös (ez biztosítja a programok kompatibilitását) és csupán a hozzátett perifériaegységekben és az adat- és programmemóriák méretében különböznek. Emiatt a nagymennyiségű gyártás okozta költségcsökkenés elmarad, hiszen a felhasználás több külön és kisebb tételben gyártott mikrokontroller között oszlik meg. Még egy gondolat: míg a processzorok sebessége és integráltsága folyamatosan növekszik, a

mikrokontrollerek nem követték ezt a fejlődést: egy átlagos kontroller sebessége mindössze 1-5 MIPS (millió utasítás másodpercenként).

A SCENIX cég egy új és jobb megoldást választott. Egy olyan nagysebességű mikrovezérlőt fejlesztett ki, amely mérföldkő lehet a mikrokontrollerek történetében. **A mikrokontroller sebessége 50 MIPS!** Ez a sebesség lehetővé teszi, hogy a realizált vezérlési feladatok mellett a processzornak elég ideje maradjon perifériaprogramok futtatására. Áramkörileg megvalósított perifériák helyett ún. **virtuális perifériákat** használhatunk. Ezek lényegében kis program-modulok az adott perifériaműködés megvalósítására. Ezeket a modulokat akkor futtatjuk, amikor szükséges, és mindegyik modul futtatása a processzorteljesítmény egy részének a felhasználását igényli. A legcélszerűbb ezeket a modulokat időosztásos módon futtatni, ilyen módon látszólag egy időben futnak, egyszerre léteznek. Mivel a perifériákat ilyen módon programmal valósítjuk meg, a kontrollerben kevés perifériát támogató áramkört kell beépíteni.

A nagy processzorteljesítmény a megszakításokra adott gyors választ is lehetővé tesz. **Két esemény okozhat megszakítást:** egyik a belső időzítő/számláló túlsordulása, a másik külső: az RB port kivezetésein megjelenő állapotváltozás (pl. egy odakötött billentyű megnyomása). A megszakítás érvényre jutása (a válasz) a belső megszakítás esetén 3 ciklus (60 nsec), míg külső megszakítás esetén 5 ciklus (100 nsec.). A megszakított program legfontosabb regisztereit a program automatikusan egy speciális egyszintű verembe elmenti, majd a megszakítási rutin befejezésekor visszatölti. Ez a fajta kialakítás gyors váltást tesz lehetővé az egyes végrehajtandó feladatok (taszkok) között.

PIC kontrollerekkel való kompatibilitás érdekében a RESET megoldás is hasonló, amely külső kapcsolódó elemek számát tovább csökkenti. A tápfeszültség megjelenésekor először egy belső számláló kezd el számolni, és ennek túlsordulása után egy, az oszcillátor feléledését figyelembe vevő másik számláló is számol. Ez — a két számláló okozta — késleltetés után jutunk ki a RESET állapotból, ami igen megbízható indulást tesz lehetővé. Ha valamilyen hiba miatt a program futása megszakad, akkor a beépített és aktivizált ún. watchdog időzítő (timer) léphet működésbe. Ez lényegében egy folyamatosan növekvő számláló, amely túlsordulásakor RESET-et okoz. Ha a periodikusan végrehajtott programhurokban elhelyezünk egy Watchdog-számlálót törlő utasítást, a túlsordulás a helyes programvégrehajtás során soha nem következik be. Ha azonban a program "eltéved", akkor a számláló törlése elmarad, és a számláló túlsordulásakor a RESET folyamat játszódik le.



1.A ábra: Az SX mikrokontroller felépítése

A mikrokontrolleres rendszerfejlesztés megfelelő fejlesztőeszközök használatát igényli. Az SX chipben az egyik legkorszerűbb ilyen megoldást alkalmazták: a minden tokban bent lévő, a programfutást vezérlő áramkör segítségével a fejlesztett program azonnal a tényleges környezetében tesztelhető, miután az EEPROM programmemóriába betöltöttük a programot. (In circuit emulátor). Azért, hogy ez az áramkör minél egyszerűbb legyen, csak a legegyszerűbb műveletek elvégzésére alkalmas: a program memória írása és olvasása, az utasítás-végrehajtás elindítása, illetve megállítása. A hibakeresésnél használt felhasználói felületet és utasításokat a fejlesztő számítógépen futó program valósítja meg. Mivel a CPU futás közben bármikor leállítható, ezért igazi hardver lépésenkénti üzemmód is megvalósítható. Az emuláláshoz és a programozáshoz a tok lábait kell felhasználni: az SX-nél ez a két oszcillátor bemeneten (OSC1, OSC2) keresztül történik, ezért az a ki-bemeneti portlábakat teljes egészében az alkalmazások használhatják.

Az SX programfejlesztő környezetét a Parallax cég által kifejlesztett **SX-KEY** eszköz biztosítja. Ez egy kis irányítóáramkör, mely az SX tokhoz kapcsolható, és a PC-vel soros porton kommunikál.

1.2 Memóriakialakítás

Az SX mikrokontroller memóriája Harvard architektúra szerint van szervezve. Ez azt jelenti, hogy **az adat- és a programmemória fizikailag külön van választva**, külön buszt használnak, és különböző szóhosszúságúak. (A klasszikus Neumann-elvű számítógépnél — Nemann János matematikus nyomán — az adatok és az utasítások azonos memóriában helyezkednek el, és azonos buszt használnak.) A Harvard típusú memóriaszervezés nagy előnye, hogy az utasítás beolvasás és az adatforgalom a külön busz miatt azonos időben, átlapolva megy végbe. Ezt a folyamatot "**többszintű pipe-line**"-nak nevezik, és azt jelenti, hogy a soron következő utasítás beolvasása már megkezdődhet, mikor az aktuális utasítás végrehajtása még folyik, függetlenül attól, hogy az aktuális utasítás használja-e az adatmemóriát. A klasszikus Harvard architektúrával ellentétben **az SX még arra is lehetőséget ad, hogy adatokat vigyünk át a programtárból az adattárba.**

A többszintű pipe-line az SX mikrokontroller esetében négy szintet jelent. Ezek:

1. *Utasítás beolvasás*
2. *Utasítás értelmezés*
3. *Utasítás végrehajtás*
4. *Adat kiírás a célregiszterbe*

	1. ciklus	2. ciklus	3. ciklus	4. ciklus	5. ciklus	6. ciklus	...
1. utasítás	beolvasás	értelmezés	végrehajtás	kiírás			
2. utasítás		beolvasás	értelmezés	végrehajtás	kiírás		
3. utasítás			beolvasás	értelmezés	végrehajtás	kiírás	
4. utasítás				beolvasás	értelmezés	végrehajtás	...
5. utasítás					beolvasás	értelmezés	...

1.B ábra: A "többszintű pipe-line" folyamata

Minden szint feldolgozása négy órajel periódust igényel, de az átlapolás miatt a soron következő utasítás beolvasása már megkezdődik az előtte lévő utasítás értelmezése alatt, így a "látszólagos" utasításvégrehajtási idő egy órajelciklus. Az utasításvégrehajtás idejét szokás **gépi ciklusnak** is nevezni.

A mikrogépek teljesítményének meghatározására elterjedt egy nem teljesen objektív, de ma már elfogadott jellemző, **a MIPS**. Ez **azt mutatja meg, hogy hány utasítást képes az adott eszköz egy másodperc alatt végrehajtani**. 50 MHz-es órajelet feltételezve az SX esetében ez 50 MIPS-re adódik, az ún. turbó módban. Az

SX mikrokontroller konfigurálható kompatibilis (PIC kompatibilis) módba is, ilyenkor az utasításvégrehajtási idő négyszeresére nő, egy gépi ciklus négy órajelciklus idejű.

1.2.1 A programtár felépítése

A programtár hosszúsága 2k-szó (2048 szó), egy szó 12 bites. A teljes 2k programtár eléréséhez 11 bites címre van szükség. A programtár tartalmazhat utasításokat (a felhasználói programot), illetve a módosított Harvard architektúra következtében 12 bites nem-változtatható adatokat is. **A programtár fontos szervezeti egysége a lap**, ez 512 szóból áll, így a 2048 szavas programtár 4 lapra osztható (Page0..Page3).

1.2.2 A programszámláló

A programszámláló (**PC - Program Counter**) egy 11 bites speciális regiszter, mely a 2k hosszúságú **programtárat címzi utasításvégrehajtáskor**. A programszámláló alsó 8 bitje közvetlenül is elérhető az adatregiszterek közt, a felső 3 bit a felhasználó számára nem hozzáférhető.

A programszámláló értéke eggyel növekszik minden utasításvégrehajtási ciklusban. Természetesen a programszámláló értékét egy ugró-jellegű utasítás vagy egy megszakítás-kiszolgálás felülírhatja.

Bekapcsolási **RESET** (POWER-ON-RESET) esetén a programszámláló a programtár tetejére mutat, ennek címe 2k esetén 7FFH. Ide NOP utasítást írva a valós reset vektorcím 00H lesz, de ez nem célszerű, mivel 000H a fix megszakítási vektorcím is egyben. A gyakorlatban a reset vektorcímre egy ugró utasítást helyeznek el, ami a tényleges programkezdetre ugrik.

1.2.3 A programszámlálót módosító utasítások

TEST-SKIP utasítások

Ezek az utasítások egy **feltételt tartalmaznak, és ha a feltétel igaznak bizonyul, átugorják a soron következő utasítást**, így a programszámláló egy helyett kettővel növekszik. Ha a feltétel nem teljesül, akkor a programvégrehajtás a soron következő utasítással folytatódik. Az esetleges átugrás természetesen a pipeline folyamatot is megzavarja, mivel az átugrandó utasítás feldolgozása már folyamatban van, tehát azt el kell dobni. Ezért **a TEST-SKIP utasítások két gépi ciklus idejűek, ha átugrást valósítanak meg.**

Ugrás abszolút címre

Ezek az utasítások **feltétel nélküli ugrást hajtanak végre a programtár területén** úgy, hogy az ugrási címet beírják a programszámlálóba. Az új cím alsó 9 bitjét a JMP utasítás operandusa tartalmazza, a felső két bitet a **STATUS** regiszter **PA1** és **PA0** bitjei adják, oda kell őket beírni. Korábban már volt szó az 512 szavas lapokról, az ezeken belüli címzéshez pont 9 bit kell, így a **PA1** és **PA0** bitekkel tulajdonképpen azt jelöljük ki, hogy melyik lapra ugrunk. Például, ha az 5E0H címre (2. lap) szeretnénk ugrani:

```
clrb PA2
setb PA1
clrb PA0
jmp $1E0
```

Megjegyzés: A **PA2** bitnek itt még nincs szerepe, csak a későbbi — 2k-nál hosszabb programtárral rendelkező — SX mikrokontroller típusokkal való kompatibilitás miatt építették be.

A **PAGE** utasítás használatával időt és programhelyet takaríthatunk meg, és ugyanazt a célt érjük el:

```
page $400
jmp $1E0
```

A **PAGE** utasítás úgy állítja át a **STATUS** regiszter **PA2...PA0** bitjeit, hogy a többi bit nem változik. Operandusként egy 12 bites számot kell megadni, amiből az assembler csak a felső három bitet veszi figyelembe, és ezek alapján állítja be a **PA2..PA0** biteket. Ez a látszólag körülményes megoldás valójában a mi kényelmünket szolgálja, ugyanis ha a programunkban valahol van egy címke, amihez egy abszolút programtár-cím rendelhető, elég ennyit tennünk:

```
page cimke
jmp cimke
```

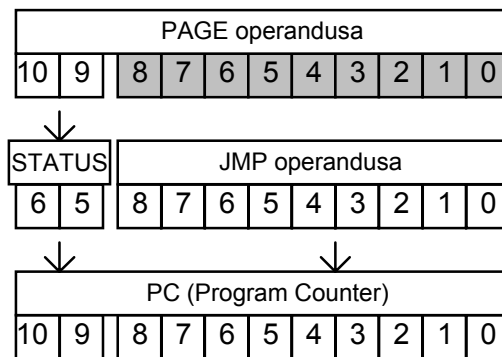
A fenti utasításpár esetén a fordító felkínál egy hasznos makró lehetőséget (a makró utasításokról részletek az **1.7 szakaszban**):

```
jmp @cimke
```

Ez a változat csak írásmódjában különbözik az előzőtől, a végeredmény ugyanaz az utasításpár lesz: `JMP @cimke = PAGE cimke + JMP cimke`.

Természetesen, ha egymás után többször is ugrunk ugyanarra a lapra, elegendő egyszer beállítani a lap-kijelölő biteket. **Megjegyzés:** A `JMP` utasítás két gépi ciklus idejű a kompatibilis, és három gépi ciklus idejű a turbó módban.

Nagyon fontos: A **PA2..PA0** lapkiválasztó biteket a processzor magától soha nem változtatja. Amit egyszer oda beírunk, az addig marad ott, amíg át nem írjuk. Tehát amikor a program fut, és a programszámláló átlép az egyik lapról a másikra, a lapkiválasztó bitek nem frissítődnek.



1.C ábra: Az ugró utasítások működési mechanizmusa

Kiszámított (indirekt és relatív) ugrások

Mivel a programszámláló alsó nyolc bitje a felhasználó számára is hozzáférhető, ezért ugrást úgy is előidézhetünk, hogy közvetlenül módosítjuk a **PC** regisztert (indirekt ugrás).

```
mov    W, $0B
mov    PC, W
```

A kiszámított ugrások második fajtájával azt érhetjük el, hogy **az adott pozíciónktól előre vagy visszafelé lépünk** egy meghatározott hosszúságú lépést azáltal, hogy a **W** regisztert hozzáadjuk a **PC**-hez, vagy kivonjuk belőle (relatív ugrás).

```
mov    W, #4
add    PC, W ; növeljük PC-t négygel (ugrunk előre az 5. utasításra)
```

Az `ADD PC,W` utasítás végrehajtásakor a programszámláló már a soron következő utasításra mutat, tehát ha $W=0$, akkor ezzel az utasítással folytatódik a programvégrehajtás (PC értéke nem változik a művelet után).

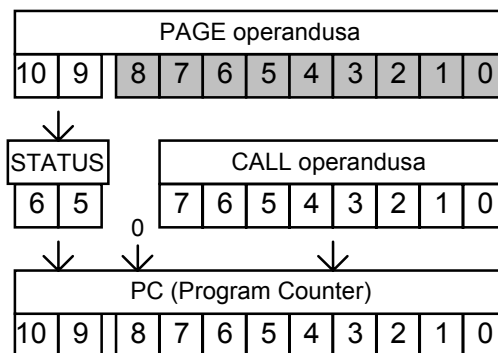
Fontos odafigyelni arra, hogy ezeknek az utasításoknak az operandusa csak 8 bites. Műveletet végezni a programszámlálónak csak az alsó nyolc bitjén tudunk, ezért az ugrási cím kilencedik (8-as számú) bitje törlődik. Emiatt az **ugrási célnak mindig az adott lap első felén (256 szó) kell lennie**. A konkrét lapot itt is a STATUS regiszter **PA1..PA0** bitjei jelölik ki.

A kiszámított ugrások segítségével **eredményfüggő elágazásokat tudunk megvalósítani**, vagyis egy adott művelet eredményétől függően más és más címre ugorhatunk. **Megjegyzés:** A kiszámított ugrási utasítások két gépi ciklus idejűek a kompatibilis, és három gépi ciklus idejűek a turbó módban.

A fordítóprogram további szolgáltatása, hogy a `MOV PC,W` helyett használhatjuk a `JMP W`, illetve az `ADD PC,W` helyett a `JMP PC+W` utasításformulát. Ez csak ízlés kérdése.

Szubrutinhívás

A `CALL` utasítás szintén feltétel nélküli ugrást valósít meg, de különbözik a `JMP`-tól, mivel a **soron következő utasítás címét elmenti a verembe** (stack-be), ezáltal lehetővé válik a szubrutinból való visszatérés, és így a programfutás folytatása az elmentett címtől. Továbbá a `CALL` utasítás operandusa csak 8 bites, így az ugrási cím kilencedik (8-as számú) bitje törlődik. Ez azt eredményezi, hogy a **szubrutinoknak mindig az adott lap első felén (256 szó) kell kezdődniük**. A konkrét lapot itt is a STATUS regiszter **PA1..PA0** bitjei jelölik ki, tehát itt is használható a `PAGE` utasítás illetve a `CALL @cimke` formula. **Megjegyzés:** A `CALL` utasítás két gépi ciklus idejű a kompatibilis, és három gépi ciklus idejű a turbó módban.



1.D ábra: A szubrutinhívó utasítások működési mechanizmusa

Sokszor elég körülményes az összes szubrutint a lap első felére bezsúfolni, ráadásul, az ilyen megoldások a program áttekinthetőségét is rontják. Mivel az egyes szubrutinok kezdetének kell csak a lapok első felén lenni, megoldást jelenthet a szubrutin-ugró tábla alkalmazása. A JMP utasításoknál ugyanis már nincs ez a korlát.

A következő példában azon a címen, amit a CALL meghív, csak egy JMP utasítás található. Ez ugrik el a tényleges szubrutinra, ami így már bárhol elhelyezkedhet a programtárban. Most már szubrutinonként csak egy — az adott szubrutinhoz rendelt JMP — utasítás foglalja a helyet a lapok elején. A bemutatott példában a szubrutinokból RETP utasítással térünk vissza. Ennek magyarázata következik a következő szakaszban.

```
                ORG    $00          ; ugrótábla a 0-ás lap első felén
Sub1            JMP    @_Sub1
Sub2            JMP    @_Sub2

Start          CALL   @Sub1        ; Fő rutin
                JMP    @Folyt

                ORG    $200        ; 1-es lap
Folyt          CALL   @Sub2
                JMP    @Start

_Sub1          .....             ; első szubrutin tényleges része,
                RETP                ; ami már bárhol elhelyezkedhet
_Sub2          .....             ; második szubrutin tényleges része,
                RETP                ; ami már bárhol elhelyezkedhet
```

Visszatérés egy szubrutinból

Ezek a RETURN típusú utasítások, **minden szubrutint ilyennel kell lezárni**. A RETURN utasítás visszatölti a programszámlálóba azt a címet, amit a CALL utasítás mentett el a verembe, így a programfutás a CALL utáni utasítással folytatódik. Mivel a CALL utasítás a teljes 11 bites címet elmenti, a RETURN előtt nem szükséges a PA1..PA0 bitek (vissza)állítása, a programvégrehajtás mindig a megfelelő címen fog folytatódni.

Az SX mikrokontroller utasításkészlete öt RETURN típusú utasítást tartalmaz. Ezek a szubrutinból való visszatérés mellett egyéb műveleteket is végrehajthatnak.

Megjegyzés: A RETURN utasítások két gépi ciklus idejűek a kompatibilis, és három gépi ciklus idejűek a turbó módban.

RET	Egyszerű visszatérés egy szubrutinból. Semmilyen regisztert vagy jelzőbitet nem változtat meg.
RETP	Visszatérés egy szubrutinból egy másik lapra. Ugyanúgy működik, mint a RET utasítás, de a visszatérési cím felső két bitjét beírja a STATUS regiszter PA1..PA0 bitjeibe. Így automatikusan az a lap lesz kijelölve, ahol a visszatérési cím elhelyezkedik.
RETW literal	Visszatérés egy szubrutinból úgy, hogy az operandusként megadott konstanst (literal) beírja a W regiszterbe. Táblázatok készítésére is használhatjuk, lásd az 1.2.10 szakaszban.
RETI	Visszatérés egy megszakításból. A programszámlálón kívül visszaállítja a megszakításkor elmentett W, STATUS és FSR regiszterek értékét. A részleteket lásd az 1.5.4 szakaszban.
RETIW	Visszatérés egy megszakításból és RTCC korrigálása W-vel. Az utasítás hozzáadja a W regiszter tartalmát az RTCC-hez. Megjegyzés: A [3] szakirodalom kivételével eredeti adatlapok itt következetesen kivonást emlegetnek, aminek nem sok értelme lenne, mivel nem biztosítja, hogy az RTCC megszakítás konstans időközönként következzen be. (ld. az 1.5.4 szakaszt).

1.2.4 A verem

A verem (stack) egy 11 bit széles, 8 szint mélységű LIFO (last-in-first-out) tár. Ebből az következik, hogy **nyolc szubrutint ágyazhatunk egymásba a veremtár túltöltése nélkül**. Megjegyzés: A SX mikrokontroller konfigurálható úgy is, hogy a verem csak két szintes legyen.

Mivel az SX-nél a verem nem elérhető a felhasználó számára, így hiányzik a más rendszereknél igen közkedvelt és hasznos utasításpár: a PUSH és a POP.

Ha a verem túltöltődésére nem figyelünk, akkor az a programunk rendszeres összeomlását fogja előidézni. Szintén beláthatatlan következményei lehetnek egy CALL nélküli RETURN utasításnak. Jelentős újítás, hogy a megszakításrutin nem töltheti túl a vermet, mivel a megszakítások egy külön elszeparált helyre (tükörregiszterbe) mentik a programszámláló tartalmát.

1.2.5 Az adattár felépítése

Az adattár statikus RAM típusú memória, programból tetszőlegesen felülírható, de tartalmát a tápfeszültség kikapcsolásakor elveszti. Az adattár 256

regisztert tartalmaz, viszont az átfedések miatt ez ténylegesen csak 136 általános és nyolc speciális funkciójú regisztert jelent. A regiszterek nyolcbites (bájtos) szervezésűek. **Az adattárat nyolc ún. bankra osztották fel (Bank0..Bank7)**, mivel a regiszterkezelő utasítások csak 5 bites operandussal dolgoznak. Egy bank 32 bájtot tartalmaz, amik részben átfedik egymást. Mivel a bank-szervezést fel lehet fogni mappákként (File) is, ezért **szokás ezeket fájl-regisztereknek is nevezni.**

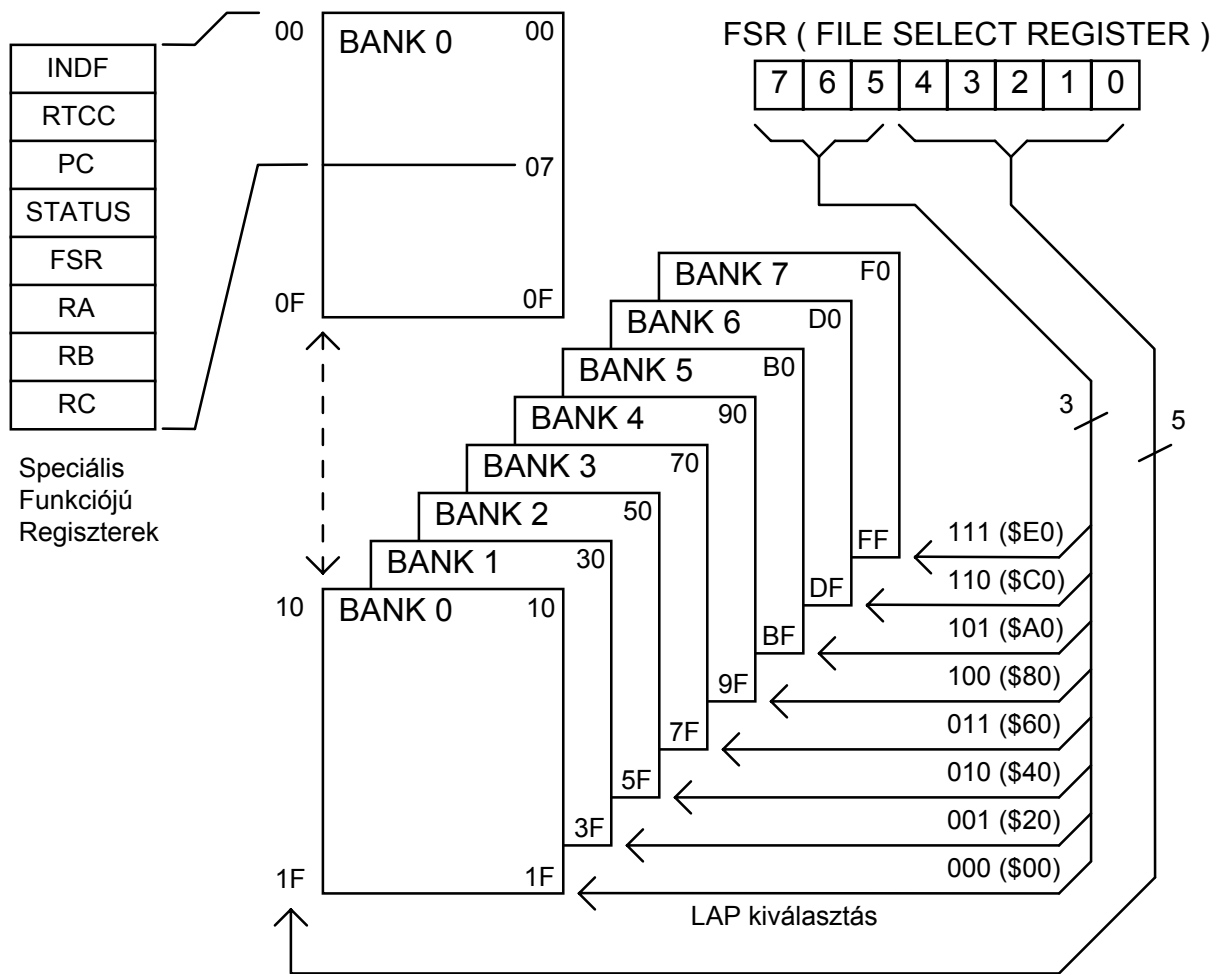
1.2.6 A bankok

Programfutás közben egyszerre csak egy bankot érhetünk el a nyolc közül. Hogy melyik ez az egy, azt a **FSR** regiszter felső három bitje jelöli ki (nem tévesztendő össze a **STATUS** regiszter **PA2..PA0** bitjeivel, melyek a programtár lapkijelölését végzik). Bekapcsoláskor a Bank0 van kiválasztva. Bankot váltani az **FSR** regiszter vonatkozó bitjeinek közvetlen állításával, vagy pedig a **BANK** utasítás segítségével tudunk. Programozáskor az **SX** konfigurálható úgy is, hogy nyolcnál kevesebb (4, 2, vagy 1) bankot érhessünk csak el. Ilyenkor — mivel az elérhető regiszterek száma is kevesebb — nincs szükség az **FSR** regiszter minden bitjére, a nem használt bitek olvasáskor 1 értékűnek látszanak, még akkor is, ha előzőleg 0-ba állítottuk őket. Tehát pl. négy bank esetén az **FSR** legfelső (7-es) bitje mindig 1 értékű, két bank esetén pedig a 6-os is.

Egy bank 32 regisztert tartalmaz (00H..1FH), így egy regiszter bankon belüli megcímzéséhez ötbites cím szükséges. Ezt az ötbites címet kell megadni egy regiszter kezelő utasítás operandusában. Egy regiszter tényleges 8 bites címe úgy áll össze, hogy a felső három bitet az **FSR** regiszter felső három bitje adja, az alsó öt bitet pedig az adott regiszter bankbéli öt bites (relatív) címe.

Minden bank első 16 regisztere a 0-ás bank első 16 regiszterét címzi. Tehát ha például a 4EH regisztert (Bank2, 13-as reg.) írjuk, akkor ténylegesen a 0EH regiszterbe (Bank0, 13-as reg.) írunk. Ebből a 16 átlapolt regiszterből az első nyolc ún. speciális funkciójú regiszter, a második nyolc pedig egy bankfüggetlen 8 bájtos általános felhasználású regiszterblokk.

Az **1.5 ábra** és az azt követő táblázat segít a Bank-szervezés megértésében. A táblázatban az átlapolt regiszterterületek szürke tónussal vannak jelölve. Az első oszlopban az egyes regiszterek bankbéli (relatív) címe található. A belső mezők az átlapolt regiszterek esetén azt mutatják, hogy melyik az a regiszter, ami az átlapolás miatt ténylegesen megcímződik, míg a nem átlapolt regiszterek esetén az adott regiszter abszolút címét láthatjuk.



1.E ábra: Az adattár felépítése

Relatív cím	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
\$00	INDF	INDF	INDF	INDF	INDF	INDF	INDF	INDF
\$01	RTCC	RTCC	RTCC	RTCC	RTCC	RTCC	RTCC	RTCC
\$02	PC	PC	PC	PC	PC	PC	PC	PC
\$03	STATUS	STATUS	STATUS	STATUS	STATUS	STATUS	STATUS	STATUS
\$04	FSR	FSR	FSR	FSR	FSR	FSR	FSR	FSR
\$05	RA	RA	RA	RA	RA	RA	RA	RA
\$06	RB	RB	RB	RB	RB	RB	RB	RB
\$07	RC	RC	RC	RC	RC	RC	RC	RC
\$08	08H	08H	08H	08H	08H	08H	08H	08H
\$09	09H	09H	09H	09H	09H	09H	09H	09H
\$0A	0AH	0AH	0AH	0AH	0AH	0AH	0AH	0AH
\$0B	0BH	0BH	0BH	0BH	0BH	0BH	0BH	0BH
\$0C	0CH	0CH	0CH	0CH	0CH	0CH	0CH	0CH
\$0D	0DH	0DH	0DH	0DH	0DH	0DH	0DH	0DH
\$0E	0EH	0EH	0EH	0EH	0EH	0EH	0EH	0EH
\$0F	0FH	0FH	0FH	0FH	0FH	0FH	0FH	0FH
\$10	10H	30H	50H	70H	90H	B0H	D0H	F0H
\$12	11H	31H	51H	71H	91H	B1H	D1H	F1H
\$12	12H	32H	52H	72H	92H	B2H	D2H	F2H
\$13	13H	33H	53H	73H	93H	B3H	D3H	F3H
\$14	14H	34H	54H	74H	94H	B4H	D4H	F4H
\$15	15H	35H	55H	75H	95H	B5H	D5H	F5H
\$16	16H	36H	56H	76H	96H	B6H	D6H	F6H
\$17	17H	37H	57H	77H	97H	B7H	D7H	F7H
\$18	18H	38H	58H	78H	98H	B8H	D8H	F8H
\$19	19H	39H	59H	79H	99H	B9H	D9H	F9H
\$1A	1AH	3AH	5AH	7AH	9AH	BAH	DAH	FAH
\$1B	1BH	3BH	5BH	7BH	9BH	BBH	DBH	FBH
\$1C	1CH	3CH	5CH	7CH	9CH	BCH	DCH	FCH
\$1D	1DH	3DH	5DH	7DH	9DH	BDH	DH	FDH
\$1E	1EH	3EH	5EH	7EH	9EH	BEH	DEH	FEH
\$1F	1FH	3FH	5FH	7FH	9FH	BFH	DFH	FFH

1.2.7 Közvetlen címzés

A megcímezhető regiszter bankbéli címét az utasítás operandusa adja. Például a 0FH című regiszter inkrementálása:

```
inc    $0F
```

Ha egy olyan regiszterrel akarunk dolgozni, ami valamelyik bank 10H..1FH tartományában van, akkor először ki kell választanunk az adott bankot. Például, ha az F2H abszolút című regisztert (Bank7, 18-es (12H) reg.) akarjuk inkrementálni:

```
mov    W, # $E0
mov    FSR, W
inc    $12
```

Mivel ilyenkor csak az **FSR** regiszter felső 3 bitjét veszi figyelembe az processzor, sokkal kevesebb gondolkodással (és hibalehetőséggel) jár, ha egyszerűen beírjuk az abszolút címet az **FSR**-be.

```
mov    W, # $F2
mov    FSR, W
inc    $12
```

A **BANK** utasítás a az **FSR** regiszter felső három bitjét módosítja úgy, hogy a többi bitet nem változtatja meg. A **BANK** utasítás operandusa nyolc bites, de az assembler csak a felső három bitet veszi figyelembe, ez másolódik át az **FSR** felső három bitjébe.

```
bank   $FF
inc    $1F
```

Természetesen, ha egymás után többször is ugyanazt a bankot kívánjuk elérni, akkor elegendő egyszer beállítani az **FSR** regiszter bitjeit.

1.2.8 Közvetett címzés

Közvetett címzésnél az FSR regisztert használjuk fel mutatóként (pointer), ilyenkor az FSR-be a megcímezhető regiszter nyolc bites abszolút címét kell betölteni. A processzort úgy utasítjuk az FSR-ben lévő cím figyelembevételére,

hogy az utasításban operandusként az **INDF** regiszter címét (00H)adjuk meg. Megjegyzés: az **INDF** regiszter nincs fizikailag kialakítva, csak a processzort informálja az indirekcióról.

Például, ha egy konstans értéket (C3H) szeretnénk betölteni közvetett módon az F5H abszolút című regiszterbe:

```
mov    W, #F5
mov    FSR, W
mov    W, #C3
mov    INDF, W
```

Egy gyakorlati példa: a program minden bankban törli a felső 16 regisztert (10...1F).

```
clr    FSR
loop  setb FSR.4
      clr  INDF
      incsz FSR
      jmp  loop
```

Mivel a hurok (loop) minden ciklusában 1-be állítjuk az **FSR** 4-es bitjét, így az alsó 16 regiszter nem sérül meg.

Speciális eset, ha az **FSR** regiszter 00H-t tartalmaz (vagyis magát a fizikailag nem létező **INDF** regisztert címzi). Ha így hajtunk végre indirekt olvasást, akkor az eredmény 00H lesz, ha indirekt írást végzünk, akkor ténylegesen egy NOP utasítást hajt végre a processzor.

1.2.9 Adat kiolvasása a programtárból

A módosított Harvard architektúrának köszönhetően lehetőségünk van adatokat átvenni a programtárból az adattárba, és ezeket az adatokat a felhasználói programban feldolgozni.

Az adatbeolvasás indirekt módon történik. A 11 bites programtár-cím (alsó 8 bitjét a **W** regiszternek, a felső 3 bitjét pedig a **MODE** regiszter alsó 3 bitjének kell tartalmaznia. A cím megfelelő betöltése után az **IREAD** utasítást kell kiadni. Ez beolvassa a megcímezett programtár-rekeszt, a 12 bites eredmény felső 4 bitje a **MODE** regiszter alsó 4 bitjébe kerül, az alsó 8 bitje pedig a **W** regiszterbe. A **MODE** regiszter felső 4 bitje törlődik. Látható, hogy a **W** és a **MODE** regiszterek

korábbi tartalma elvész, így szekvenciális olvasásnál minden egyes IREAD utasítás előtt be kell tölteni az olvasni kívánt programtár-rekesz címét.

Megjegyzés: Az IREAD utasítás egy gépi ciklus idejű a kompatibilis, és négy gépi ciklus idejű a turbó módban.

Példa: Beolvassuk a 380H abszolút című programtár-rekeszt és letároljuk a 0FH:0EH regiszterpárba (adattár-rekeszpárba) :

```
mov  W,$03      ; a cím felső 3 bitjének a betöltése
mov  M,W        ;
mov  W,$80      ; a cím alsó 8 bitjének a betöltése.
iread          ; adat beolvasás
mov  $0E,W      ; az adat alsó 8 bitjének letárolása
mov  W,M        ; az adat felső 4 bitjének a letárolása
mov  $0F,W      ;
```

1.2.10 Adattáblázatok készítése

Olykor (pontosabban igen gyakran) szükség lehet konstanstömbök használatára. Ezeket **a konstanstömböket rendszerint a programmemóriában tároljuk el, és adattáblázatnak vagy röviden táblázatnak hívjuk őket.** Táblázatok készítésére két mód kínálkozik. Az első az imént bemutatott IREAD utasítás, ilyenkor a letárolt adatok 12 bitesek. A másik a már korábbról (**1.2.3 szakasz**) ismert kiszámított ugrások és a RETW utasítás együttes alkalmazása, ebben esetben az adatok 8 bitesek. Erre példa:

```
ORG  $200      ; 1-es lap első fele, a teljes
TABL ADD  PC,W  ; táblázatnak itt kell elhelyezkednie.
      RETW 'ABCDEF' ; szöveges adatok
      RETW 10,100,255,0 ; numerikus adatok
      ....
      MOV  W,#3    ; táblázat feldolgozás
      CALL TABL    ; visszatérés után W a „D” betű ASCII
                  ; kódját tartalmazza
      MOV  $0E,W   ; letároljuk a 0EH című regiszterbe
```

1.2.11 A speciális funkciójú regiszterek

Az SX mikrokontroller 26 speciális funkciójú regisztert tartalmaz, melyből nyolc az átfedések miatt bármely bankból elérhető a 00H..07H címen. A további 18 (**W**, a 15 portvezérlő, **MODE** és **OPTION**) nem memóriába ágyazott módon van kialakítva, ezek speciális utasításokkal érhetőek el és szintén nyolc bit szélesek.

W - Munkaregiszter

A **W** regiszter a fő munkaregiszter (working register), sok utasítás használja cél vagy forrásregiszterként. Például, ha két általános regiszter között adatot akarunk mozgatni, azt csak a **W**-n keresztül tudjuk elvégezni. Először bevisszük a forrás regiszter tartalmát a **W**-be, majd a **W** tartalmát átmásoljuk a célregiszterbe.

Alapbeállításban a **W** regiszter nem memóriába ágyazott regiszter. Ez a probléma feloldható, ha programozáskor úgy állítjuk be az SX-et, hogy az **OPTION** regiszter kiterjesztett módban legyen használható. Ilyenkor az **OPTION** regiszter **RTW** bitjével választhatjuk ki, hogy a 01H címen az **RTCC** vagy a **W** regiszter legyen elérhető. Ha **W**-t akarjuk a 01H címen elérni, akkor a forráskódban is írunk **RTCC** helyett **WREG** szimbólumot a jobb áttekinthetőség miatt. Pl.: DEC WREG.

Az INDF és az FSR regiszterek

INDF = Indirect through FSR

FSR = File Select Register

A részleteket lásd az **1.2.8 szakaszban**.

RTCC (Real Timer Clock & Counter)

Valós idejű időzítő és számláló. Időzítő üzemmódban az **RTCC** regiszter tartalma az órajel hatására, eseményszámláló üzemmódban **RTCC** lábán történő pozitív vagy negatív él hatására inkrementálódik. Az üzemmódot az **OPTION** regiszter határozza meg. Az **RTCC**-hez rendelhető előosztó is, az osztásviszonyt szintén az **OPTION** regiszter határozza meg. Az **RTCC** regiszter túlsordulása megszakítást válthat ki, ha ez engedélyezve van. A részleteket lásd az **1.4 szakaszban**.

PC - Programszámláló

A részleteket lásd az **1.2.2 szakaszban**.

STATUS - Állapotregiszter

A **STATUS** regiszter bitjei a mikrokontroller állapotáról adnak tájékoztatást. Ezeket bizonyos események automatikusan törlik vagy állítják

	STATUS bit	Leírás
7..5	PA2..PA0	Programtár lapkiválasztó bitek. 000: 0-ás lap 001: 1-es lap 010: 2-es lap 011: 3-as lap A részleteket lásd az 1.2.2 szakaszban.
4	TO	Watchdog túlfutás jelző. Bekapcsoláskor 1-be íródik, és a watchdog időzítő túlfutásakor törlődik. a részleteket lásd az 1.4.5 szakaszban.
3	PD	Az energiatakarékos üzemmódot jelző bit. Bekapcsoláskor 1-be íródik, a SLEEP utasítás hatására törlődik. A részleteket lásd az 1.6.4 szakaszban.
2	Z	Zéró (null-)bit. Bizonyos műveletek az eredményük függvényében törlik vagy állítják ezt a bitet. Ha az eredmény nulla, 1-be írják, ha nem, akkor törlik.
1	DC	Digit-Carry. 1-be íródik, ha egy összeadási művelet után átvitel (carry-out) volt a 3-as bitről a 4-es bitre, illetve törlődik, ha egy kivonási művelet után nem volt áthozat (borrow-out) a 4-es bitről a 3-as bitre.
0	C	Carry. 1-be íródik, ha egy összeadási művelet után átvitel (carry-out) volt a 7-as bitről a 0-ás bitre (a regiszter túlcordult), illetve törlődik, ha egy kivonási művelet után nem volt áthozat (borrow-out) a 0-ás bitről a 7-as bitre (a regiszter nem csordult alul). Megjegyzés: az SX programozáskor konfigurálható úgy is, hogy a C bit is részt vegyen az összeadó és kivonó műveletekben, megkönnyítve ezzel a több-bájtos összeadási és kivonási műveleteket. Továbbá rotáló utasítások (RL, RR) is ezen a biten keresztül végzik el forgatást.

RA, RB és RC (Port adatregiszterek)

Ha ezeket a regisztereket írjuk, a beírt adat közvetlenül megjelenik az egyes portokon. Ha ezeket a regisztereket olvassuk, közvetlenül a portlábakat olvassuk, ami nem szükségszerűen azonos a port adatregiszterek beírt tartalmával. a részleteket lásd az **1.3 szakaszban**. A 18 lábú SX típusoknál — nem lévén C port — az RC regiszter egy általánosan felhasználható regiszter.

A portvezérlő regiszterek és a MODE regiszter

Egy porthoz minimum három maximum nyolc portvezérlő regiszter tartozik. Egy port minden port vezérlő regisztere ugyanazzal az utasítással érhető el, például az **RA port** esetén: `MOV !RA, W`. Azt, hogy ezzel az utasítással melyik portvezérlő regisztert érjük el, a **MODE** regiszter értéke határozza meg. A **MODE** regiszter a `MOV M, W` vagy a `MOV M, #lit` utasítással írható, és a `MOV W, M` utasítással olvasható. Bekapcsoláskor a **MODE** regiszter értéke 0FH.

Az egyes portvezérlő regiszterek funkcióit részletesen az **1.3.2 szakasz** írja le.

Az **OPTION** regiszter

Ezzel a regiszterrel az SX különféle üzemmódjait állíthatjuk be. Az **OPTION** regiszter egy nem memóriába ágyazott, csak olvasható regiszter. Bekapcsoláskor minden bitje 1 értékű.

	OPTION bit	leírás
7	RTW	RTCC or W. Ez a bit csak akkor elérhető, ha a programozáskor az SX-et ún. kiterjesztett OPTION módba állítottuk. Ha a bit 0 értékű, akkor a 01H címen a W regisztert érhetjük el, ha 1 értékű, akkor az RTCC regisztert.
6	RTE_IE	RTCC Interrupt Enable. Ez a bit csak akkor elérhető, ha a programozáskor az SX-et ún. kiterjesztett OPTION módba állítottuk. Ha a bit 0 értékű, akkor engedélyezett az RTCC megszakítás túlsordulásakor, ha 1 értékű, akkor tiltott.
5	RTS	RTCC Trigger Source: Ha a bit 0 értékű, akkor az órajel hatására (időzítő üzemmód), ha 1 értékű, akkor az RTCC lábán megjelenő él hatására (eseményszámláló üzemmód) inkrementálódik az RTCC regiszter.
4	RTE_ES	RTCC Edge Select. Csak akkor van hatása, ha az RTCC eseményszámlálóként működik. Ha a bit 0 értékű, akkor felfutó, ha 1 értékű, akkor lefutó él hatására inkrementálódik az RTCC regiszter tartalma.
3	PSA	Prescaler Assignment. Ha ez a bit 0 értékű, akkor az előosztó az RTCC-hez, ha 1 értékű, akkor a Watchdog időzítőhöz van hozzárendelve.
2..0	PS2..PS0	Prescaler Value. Ezek a bitek az előosztó osztásviszonyát határozzák meg. Attól függően, hogy az előosztó az RTCC-hez vagy a Watchdog-hoz van e rendelve, az egyes értékek más osztásviszonyt adnak! A beállítási lehetőségeket lásd az 1.4.7 szakaszban.

1.3 I/O portok

A 18 lábú SX típusok két, a 28 lábúak három valódi kétirányú bemeneti/kimeneti (I/O - Input/Output) portot tartalmaznak.

- 1. Az **A port** (RA) 4 bites, szimmetrikusan terhelhető mindkét irányba, ami azt jelenti, hogy aktív állapotban a terhelésre jutó feszültség független a porton átfolyó áram irányától. Magyarul mindegy, hogy a terhelést a portláb és a föld, vagy a portláb és a tápfeszültség közé kötjük be.*

Megjegyzés: az aszimmetrikus kimeneti terhelhetőség a TTL rendszereknek volt a sajátossága.

- 2. A **B port** (RB) 8 bites, használható általános I/O-ként is, de más funkciókat is rendeltek hozzá, melyek közül szoftverből választhatunk.*
- 3. A **C port** (RC) 8 bites és csak a 28 lábú típusokban található meg.*

1.3.1 A portok írása és olvasása

Az SX mikrokontroller I/O portjai a speciális funkciójú regiszterek között memóriába ágyazottan érhetők el, emiatt nincs szükség külön portkezelő utasításokra, **az egyes portok a hagyományos regiszterkezelő utasításokkal írhatók és olvashatók.** Megjegyzés: ezt a módszert először a Motorola cég alkalmazta 6800-as típusjelű mikroprocesszoraiban a 70-es évek elején. Ezt az ötletet adaptálta az Intel a második generációs mikrokontroller családjába, az MCS-51-be azzal a kiegészítéssel, hogy létrehozott egy külön memóriablokkot a speciális funkciójú regiszterek számára, így az összes beépített perifériát memóriába ágyazottan lehetett elérni. Ezáltal vált lehetővé, hogy az MCS-51 család minden fejlettebb tagja felülről kompatibilis az eredeti első családtaggal. Mára már minden mikrokontroller gyártó elfogadta, és átvette ezt a szemléletmódot.

Az egyes portokhoz rendelt memóriacímek RA - 05H, RB - 06H, RC - 07H, ezek az ún. port-adatregiszterek. **Az egyes regiszterek olvasása esetén az adott portlábakon lévő feszültségek logikai értékeit kapjuk eredményül,** függetlenül attól, hogy az adott port kimenetként vagy bemenetként van-e konfigurálva. Ha egy portlábba szeretnénk írni, akkor azt kimenetként kell konfigurálni. Az SX felépítése lehetőséget ad arra, hogy a felhasználó döntse el, mely portlábakat milyen irányúként szeretné használni (a részleteket lásd hamarosan), ezután a vonatkozó port-adatregiszterbe írt logikai érték az adott port — kimenetként konfigurált — lábain is megjelenik logikai feszültségérték formájában. **Bekapcsoláskor minden egyes portláb bemenetként működik.**

1.3.2 A portok üzemmódbeállítása

Minden egyes portláb külön-külön beállítható a következő üzemmódok szerint:

- 1. adatirány (bemenet vagy kimenet)*
- 2. bemeneti feszültség szint (TTL vagy CMOS)*
- 3. felhúzó ellenállás lehetősége*
- 4. Schmitt triggeres bemenet lehetősége (csak a **B** és a **C portok** esetében)*

5. A B port ezeken kívül további alternatív funkciókkal is rendelkezik, ezeket lásd hamarosan.

A portok üzemmódját a portvezérlő regiszterek segítségével állíthatjuk be. Egy porthoz minimum három maximum nyolc port vezérlő regiszter tartozik. Egy port minden port vezérlő regisztere ugyanazzal az utasítással érhető el, például az **RA port** esetén: MOV !RA,W. Azt, hogy ezzel az utasítással melyik portvezérlő regisztert érjük el, a **MODE** regiszter értéke határozza meg. A **MODE** regiszter a MOV M,W vagy a MOV M,#lit utasítással írható, és a MOV W,M utasítással olvasható. Bekapcsoláskor a **MODE** regiszter értéke 0FH.

	MOV !RA,W	MOV !RB,W	MOV !RC,W
08H	—	CMP_B	—
09H	—	WKPND_B	—
0AH	—	WKED_B	—
0BH	—	WKEN_B	—
0CH	—	ST_B	ST_C
0DH	LVL_A	LVL_B	LVL_C
0EH	PLP_A	PLP_B	PLP_C
0FH	RA irány	RB irány	RC irány

A **MODE** regiszternek mindig egy 08H..0FH közötti számértéket kell tartalmaznia. A **MODE** regiszter tartalmát a processzor magától soha nem változtatja. Amit egyszer oda beírunk, az addig marad ott, amíg át nem írjuk.

1.3.3 A portvezérlő regiszterek működése

Adatirány kijelölő regiszterek: (MODE = 0FH)

Minden egyes regiszterbit a hozzá rendelt portláb bemeneti vagy kimeneti üzemmódját határozza meg. Ha az adott regiszter-bit 1 értékű, akkor a portláb bemenet, ha 0 értékű, akkor kimenet. (Amikor egy portláb bemenetként van konfigurálva, akkor nagyimpedanciás állapotban van, tehát **a portok valódi kétirányú portok**, ellentétben az például az MCS-51 család kvázi-kétirányú portjaival.)

Ha azt akarjuk, hogy egy portláb a kimenetté történő konfiguráláskor azonnal egy bizonyos logikai szintet vegyen fel, akkor a kívánt logikai értéket a kimenetté konfigurálás előtt be kell írni a vonatkozó port-adatregiszterbe.

Fontos külön kiemelni, hogy **bekapcsoláskor minden portláb bemenet**, mivel a portvezérlő regiszterek értéke ilyenkor FFH.

PLP_A, PLP_B és PLP_C: Pullup Enable register (MODE = 0EH)

Minden portlábhoz van egy opcionális felhúzó ellenállás rendelve, melynek tipikus értéke 20k Ω . Ha az adott regiszter-bit 1 értékű, akkor a felhúzás hatástalan, ha 0 értékű, akkor engedélyezett.

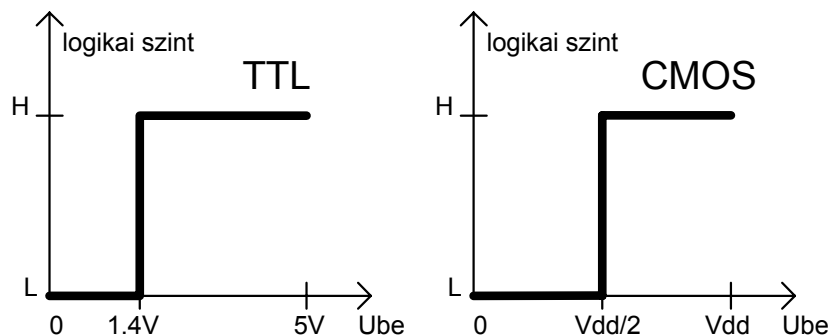
LVL_A, LVL_N és LVL_C: Input Level register (MODE = 0DH)

Minden portlábánál beállítható, hogy bemenetként konfigurálva milyen logikai szintekre legyen érzékeny (TTL vagy CMOS). Ha az adott regiszter-bit 1 értékű, akkor a bemenet TTL típusú, ha 0 értékű, akkor CMOS típusú. Csak emlékeztetőül:

1. TTL: $V_{IL} = 0.8 V$, $V_{IH} = 2.0 V$
2. CMOS: $V_{IL} = 1.5 V$, $V_{IH} = 3.5 V$
3. V_{IL} : Az a maximális bemeneti feszültség, amit még 0 logikai szintűnek érzékel.
4. V_{IH} : Az a minimális bemeneti feszültség, amit már 1 logikai szintűnek érzékel.

Ez az elméleti rész. A szabványos logikai rendszerek azért adnak meg ilyen tág határokat, mert régen a gyártási szórások miatt a tényleges billenési feszültség bárhova eshetett ezeken a tartományokon belül. Az SX-nél már más a helyzet, a billenési feszültségek rendkívül pontosan definiáltak.

1. TTL: $V_{TH} = 1.4V$ (TTL rendszer lévén feltételezzük, hogy $V_{DD} = 5V$)
2. CMOS: $V_{TH} = 0.5 V_{DD}$
3. V_{TH} : Billenési feszültség.



1.F ábra: A TTL és CMOS bemenetek transzfer karakterisztikái

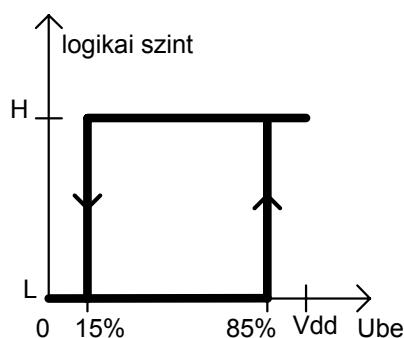
ST_B és ST_C: Schmitt Trigger Enable register (MODE = 0CH)

A **B** és a **C** portok minden lába beállítható, hogy — bemenetként konfigurálva — a logikai szintváltási feszültségértékeknél bizonyos hiszterézist mutasson (Schmitt Triggeres bemenet). Ha az adott regiszter-bit 1 értékű, akkor a Schmitt Trigger lehetőség hatástalan. Ha 0 értékű, akkor engedélyezett, ebben az esetben nincs jelentősége, hogy a bemeneti szinteket TTL vagy CMOS értékűre állítottuk.

Schmitt Triggeres bemenetek alkalmazásával az áramkör zavarérzékenysége csökkenthető. A billenési feszültségek:

1. $V_{L \rightarrow H} = 0.85 V_{DD}$

2. $V_{H \rightarrow L} = 0.15 V_{DD}$



1.G ábra: A Schmitt Triggeres bemenetek transzfer karakterisztikája

1.3.4 A portok áramköri felépítése

Az előzőek szemléltetésére az **1.8 ábrán** a **B** port felépítését tanulmányozhatjuk. A másik két port felépítése ezzel teljesen analóg.

Mintaprogram a portvezérlő regiszterek beállítására

Az **A** port minden lába kimenet, ezek közül induláskor az alsó kettő 1 értékű, a másik kettő 0 értékű. A **B** port felső négy lába kimenet, ezek közül induláskor a 6-os és a 4-es 1 értékű, a többi 0. A **B** port alsó négy lába bemenet, ezek közül a 0-ás TTL szintű, az 1-es és a 2-es CMOS szintű, a 3-as Schmitt Triggeres. A 2-es és 3-as lábon felhúzó ellenállás van.

1.3.5 A portok két speciális tulajdonsága

Bizonyos regiszterkezelő utasítások belső működési mechanizmusa olyan, hogy először beolvassák az adott regiszter tartalmát, elvégzik vele a műveletet, majd visszaírják az eredményt. Ezeket **Read-Modify-Write** utasításoknak nevezi a szakirodalom. Ilyen pl. a DEC utasítás, de olyan utasítások is ide tartoznak, amikről ezt elsőre nem is gondolnánk, mint pl. a SETB.

A Read-Modify-Write utasításokkal **akkor jelentkezik a probléma, ha a regiszter, amin a műveletet végzik, történetesen egy port adatregiszter.** Tudvalevő, hogy a port adatregiszterek olvasásakor nem a regiszterben lévő tényleges adatot kapjuk, hanem az I/O portok logikai állapotát olvassuk be. Ez többek között akkor jelenthet problémát, ha egy port bemenetként konfigurált állapotánál akarjuk az egyes bitjeit beállítani az adatregiszterben, majd ezután ezeket az előre beállított értékeket a portok kimenetként konfigurálásával ténylegesen is megjeleníteni a portlábakon. A most következő példában a **B port** 6-os és 7-es lábain felhúzó ellenállás van például azért, mert más, hasonló eszközzel huzalozott-ÉS kapcsolatba van kötve. Az ilyen hálózatoknál csak 0 kimeneti szint van engedélyezve, az 1 szintet a felhúzó ellenállás állítja elő. Huzalozott-ÉS kapcsolatot alkalmaznak többek között I²C busznál (lásd a **3.3 szakaszban**).

	; Port adatregiszter	Portláb logikai
	; tartalma	szintjei
	; -----	-----
clr RB.7	; 01xx.xxxx	11xx.xxxx
clr RB.6	; 10xx.xxxx	11xx.xxxx
mov M, #0F	;	
mov W, %00111111	;	
mov !RB, W	; 10xx.xxxx	10xx.xxxx

Azt vártuk volna, hogy mindkét láb 0 szintű legyen, ennek ellenére ez csak a 6-osnál valósult meg, a 7-es lábra a tápfeszültség szint került ki. Ha most történetesen egy másik, a 6-os lábbal egy huzalozott-hálón lévő eszköz 0 szintbe kapcsol, akkor a föld és a tápfeszültség között akkora rövidzárási áram alakul ki, amely mindkét eszköz garantált tönkremenetelét fogja okozni.

A másik probléma akkor állhat elő, ha egy port kimenetként van konfigurálva. Tétélezzük fel, hogy először kiírunk egy adatot a portra, majd rögtön vissza is olvassuk azt (a Read-Modify-Write utasítások is ide tartoznak!). A többszintű pipeline folyamatot szemléltető **1.2 ábrán** látható, hogy az előző utasítás írási fázisa pont egybeesik a soron következő utasítás végrehajtási — esetünkben olvasási — fázisával. Mivel portok esetén az írás a regiszterbe történik, az olvasás viszont közvetlenül a port lábáról, a hardver késleltetések miatt az írás okozta változás még

nagy valószínűséggel nem jelent meg a portlábakon az olvasás pillanatában, ezért ilyenkor célszerű egy NOP utasítást beiktatni az írási és az olvasási művelet közé.

1.3.6 Multi-Input Wakeup/Interrupt (MIWU)

A **B port** lábai fel vannak készítve arra, hogy bemenetként konfigurálva őket a rajtuk végbemenő pozitív, avagy negatív irányú bemeneti jelszintváltozás (él) hatására akár az energiatakarékos üzemmódból (lásd az **1.6.4 szakaszt**) történő felébredést, akár megszakítást (lásd az **1.5 szakaszt**) idézzenek elő.

A **MIWU** funkcióhoz három portvezérlő regiszter tartozik, melyek a már megismert módon, a **MODE** regiszter segítségével írhatók.

WKEN_B: Port B Wakeup Enable register (MODE = 0BH)

Ha a regiszter-bit 1 értékű, akkor a **MIWU** funkció hatástalan az adott portlábban, ha 0 értékű, akkor engedélyezett.

WKED_B: Port B Wakeup Edge Select register (MODE = 0AH)

Ha a regiszter-bit 1 értékű, akkor a **MIWU** funkció az adott porton végbemenő lefutó élre, ha 0 értékű, akkor felfutó élre aktivizálódik.

WKPND_B: Port B Wakeup Pending Flag register (MODE = 09H)

Ebben a regiszterben vannak a jelzőbitek, melyek megmutatják, hogy melyik portláb aktivizálta a **MIWU** funkciót. Ha valamelyik — **MIWU** engedélyezett — portlábban érvényes jelszintváltozás (él) megy végbe, akkor az adott portlábhoz tartozó bit 1-be íródik, és ébredési reset-et vagy megszakítást generál. Ezután a **WKPND_B** regiszter tartalmát kiolvasva megállapíthatjuk, hogy melyik portláb váltotta ki az ébredést vagy a megszakítást.

A **WKPND_B** regiszter tartalma bekapcsoláskor véletlenszerű értéket vesz fel, ezért a **MIWU** funkció engedélyezése előtt azt mindenképpen törölni kell, hogy pontosan meg tudjuk állapítani, melyik portláb aktiválta a **MIWU** funkciót.

Mivel a portvezérlő regisztereket csak írni lehet, a **WKPND_B** regiszter tartalmát a törlésével egybekötve tudjuk csak kiolvasni. Pontosabban ez azt jelenti, hogy a `MOV !RB, W` utasítás a **MODE=09H** esetén a **W** és a **WKPND_B** regiszter tartalmának felcserélését végzi el.

```
mov    M, #09          ; WKPND_B kijelölése
clr    W               ; W = 00H
mov    !RB, W         ; W = WKPND_B, WKPND_B = 00H
```

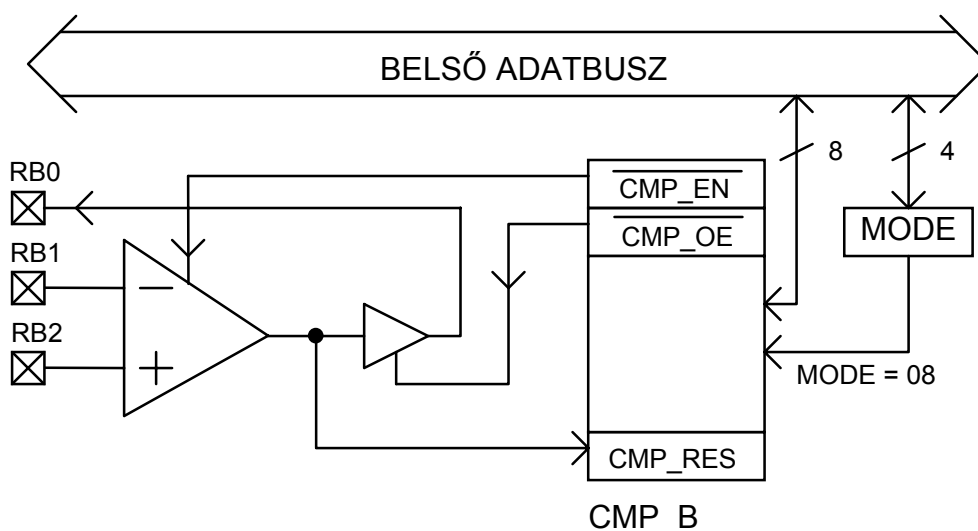

1.3.7 Az analóg komparátor

A **B port** másik kiegészítő funkciója az analóg komparátor. A komparátor bemenetei az **RB1** és **RB2** portlábak, kimenete opcionálisan megjeleníthető az **RB0** lábon is. **RB1** a komparátor invertáló bemenete, tehát ha az RB2-re adott feszültség nagyobb, mint az **RB1**-en lévő, akkor a komparátor 1 szintet szolgáltat a kimenetén.

A komparátor üzemmódját a **CMP_B** (Comparator register on Port B) portvezérlő regiszteren lehet beállítani (**MODE=08H**). A **CMP_B** regiszter tartalmát a **WKPND_B** regiszternél bemutatott módon olvashatjuk ki. Ez azt jelenti, hogy a **MOV !RB,W** utasítás a **MODE=08H** esetén a **W** és a **CMP_B** regiszter tartalmának felcserélését végzi el.

	CMP_B bit	leírás
7	CMP_EN	Comparator enable: Ha ez a bit 1 értékű, a komparátor funkció hatástalan, ha 0 értékű, akkor engedélyezett. További feltétel, hogy előzőleg a RB2 és RB1 portlábakat bemenetként konfiguráljuk.
6	CMP_OE	Comparator Output Enable: Ha ez a bit 1 értékű, az RB0 általános I/O lábként használható, ha 0 értékű, akkor a komparátor kimenete megjelenik az RB0 lábon, További feltétel, hogy előzőleg a RB0 lábat kimenetként konfiguráljuk
5..1	—	—
0	CMP_RES	Comparator result: a komparátor eredményét ennek a bitnek a kiolvasásával tudjuk a programunkban felhasználni.

Az előzőek szemléltetésére az **1.10 ábrán** a komparátor áramkör felépítését tanulmányozhatjuk.



1.J ábra: Az analóg komparátor áramkör felépítése

Mintaprogram a komparátor funkció használatára:

```
mov  M,#$08      ; CMP_B kijelölése
clr  W           ; W = 00H
mov  !RB,W       ; a komparátor és a komparátor kimenet
                        ; engedélyezése
....
....           ; az eredmény kiértékelése:
mov  M,#$08      ; CMP_B kijelölése
clr  W           ; W = 00H
mov  !RB,W       ; W = CMP_B, CMP_B = 00H
and  W,#%01      ; maszkolás
snz                    ; Ha W=0 ( $V_{RB2} < V_{RB1}$ ), akkor STATUS Z bitje = 0
jmp  rutin       ; akkor ugrunk, ha  $V_{RB1}$  a nagyobb.
```

1.4 Időzítő/számláló modulok

Az SX mikrokontroller két időzítőt (TIMER-t) tartalmaz, amelyek a közös előosztó miatt szorosan összekapcsolódnak. Az egyik az **RTCC** (Real Time Clock & Counter - Valós idejű időzítő és számláló), a másik az ún. Watchdog-timer (**WDT**).

1.4.1 Az RTCC áramkör

Az **RTCC** áramkört **időzítő** vagy **eseményszámláló üzemmódban használhatjuk**. A számlált jellemző értéke az **RTCC** nevű (címe: 01H) 8 bites speciális funkciójú regiszterben érhető el. Időzítő üzemmódban az **RTCC** regiszter tartalma az órajel hatására, eseményszámláló üzemmódban **RTCC** lábon történő pozitív vagy negatív él hatására inkrementálódik. A két üzemmód között az **OPTION** regiszter **RTS** bitjével választhatunk. Ha ez a bit 1 logikai értékű, akkor az eseményszámláló, ha 0 logikai értékű, akkor az időzítő üzemmódot jelöli ki. Az **RTCC**-hez mindkét módban rendelhető előosztó is.

Az **RTCC** regiszter túlcserélésére (FFH-ról 00H-ra) **megszakítást válthat ki**, ha ez engedélyezve van. Ez azonban csak akkor lehetséges, ha a programozáskor az SX-et az ún. **kiterjesztett OPTION** módba állítottuk. Ha ez megtörtént, akkor az **RTCC** megszakítást az **OPTION** regiszter **RTE_IE** bitjének törlésével engedélyezhetjük, 1-be állításával pedig bármikor letilthatjuk a felhasználói

programunkban. Mivel az **RTCC** túlsordulás okozta megszakításhoz nem tartozik külön jelzőbit, ezért a megszakításokat kiszolgáló rutinban meg kell vizsgálnunk az **RTCC** regiszter tartalmát (megszakításkor értéke nulla, vagy egy alacsony érték), és ennek alapján dönthetjük el, hogy a megszakítást az időzítő/számláló váltotta-e ki.

1.4.2 Eseményszámláló üzemmód

Ebben az üzemmódban az **RTCC** regiszter számtartalma az **RTCC** jelű portlábon bekövetkező jelszintváltozás (él) hatására inkrementálódik. Az él irányát az **OPTION** regiszter **RTE_ES** bitje határozza meg. Ha ez a bit 0 értékű, akkor lefutó, ha 1 értékű, akkor felfutó él hatására inkrementálódik az **RTCC** regiszter tartalma.

Ha az előosztót engedélyezzük, akkor elérhetjük, hogy a regiszter inkrementálódása csak a **STATUS** regiszter **PS2..PS0** bitjeivel beállított értékű impulzusonként (eseményenként) történjen meg.

Az **RTCC** lábon beérkező jel frekvenciája nem lehet nagyobb, mint a gépi ciklus által meghatározott frekvencia. Ha egy gépi ciklus alatt a két él érkezik, azt belső élfigyelő áramkör csak egy élnek fogja értelmezni.

1.4.3 Időzítő üzemmód

Ebben az üzemmódban precíz — szélsőséges esetben akár 20 ns-os (elméleti alsó határ) — időzítéseket valósíthatunk meg. **Az időzítő gyakorlatilag az eseményszámláló egy speciális esete, amikor a számlálást kiváltó esemény maga a mikrokontroller órajele.** Előosztó nélkül minden egyes gépi ciklusban inkrementálódik az **RTCC** regiszter tartalma. Ha a programozáskor az **SX**-et turbó módba állítottuk, akkor a gépi ciklus ideje azonos az órajelciklus idejével, ha ún. kompatibilis módba állítottuk, akkor a gépi ciklus ideje az órajelciklus idejének négyszerese. Például, ha 4 Mhz-es órajelet használunk és az **SX** turbó módban üzemel, akkor az **RTCC** regiszter 250 ns-onként inkrementálódik.

Ha az előosztót engedélyezzük, akkor elérhetjük, hogy a regiszter inkrementálása csak a **STATUS** regiszter **PS2..PS0** bitjeivel beállított értékű gépi ciklusonként történjen meg.

Ha a mikrokontrollert energiatakarékos üzemmódba állítjuk (**SLEEP** utasítás), akkor az oszcillátorral együtt az **RTCC** áramkör is leáll, és a regiszter eredeti tartalma ébredés után sem áll vissza.

1.4.4 Az RTCC regiszter írása

Ha az előosztó az **RTCC**-hez van rendelve, akkor a regiszter írása az előosztót is törli. Pontosabban az előosztó számtartalmát, és nem a **PS2..PS0** biteket!

A MICROCHIP cég PIC kontrollereinél az **RTCC** írása esetén a regiszter inkrementálása két gépi ciklus idejére lehetetlenné vált. Ilyen jelenségről az eredeti SX dokumentumok ezidáig nem tettek említést, viszont bizonyos, hogy létezik. Az SX-KEY fejlesztőeszközzel szerzett tapasztalatok szerint írás után az **RTCC** inkrementálása turbó módban két, kompatibilis módban egy gépi ciklus idejére tiltott.

1.4.5 A Watchdog áramkör

Előfordulhat, hogy valamilyen külső villamos zavar hatására a program futása megszakad, vagy egy olyan programtár területre téved, aminek az utasításait végrehajtva rendellenes működés és akár katasztrófa is felléphet .

A Watchdog-timer (**WDT**) egy szabadon futó RC oszcillátor alapú időzítő, amely nem igényel külső áramköri elemeket. Ez azt jelenti, hogy a **WDT** akkor is fut, ha az órajel szünetel, pl. azért, mert energiatakarékos állapotba állítottuk a rendszert. A **WDT** időtúlfutása egy **RESET** feltételt generál, és törli a **STATUS** regiszter **TO** bitjét. Ha a **WDT** szolgáltatást igénybe vesszük, akkor az időtúlfutás elérése előtt rendszeresen törölnünk kell a **CLR !WDT** utasítással. Normál programfutás esetén ezek a törölő parancsok meggátolják a **RESET** kialakulását. Viszont ha valamilyen zavar hatására program normális futása megszakad, a **WDT** nem kap töröljelet, kiváltja a **RESET**-et, aminek hatására a mikrovezérlő a teljes programrendszerét újra elölről felépíti, és ismét működőképessé lesz.

A **WDT** engedélyezését vagy tiltását programozáskor kell beállítani, amit működés közben a felhasználói programból már nem bírálhatunk felül.

A **WDT** névleges futási ideje **18 ms**, ha nincs hozzárendelve az előosztó. Az előosztóval a futási idő megnövelhető, akár **2.34 másodperc is lehet**. A **CLR !WDT** utasítás ebben az esetben az előosztót is törli. Pontosabban az előosztó számtartalmát, és nem a **PS2..PS0** biteket! A **CLR !WDT** utasítás járulékosan még a **STATUS** regiszter **Z**, **TO** és **PD** bitjeit is 1-be állítja.

A **WDT** energiatakarékos állapotban is tovább működik, így lehetséges a **WDT** segítségével történő ébresztés. A **CLR !WDT** utasításon kívül a **SLEEP** utasítás is törli a **WDT** regisztert. Az energiatakarékos üzemmód részleteit lásd az **1.6.4 szakaszban**.

1.4.6 A Watchdog-áramkör használata

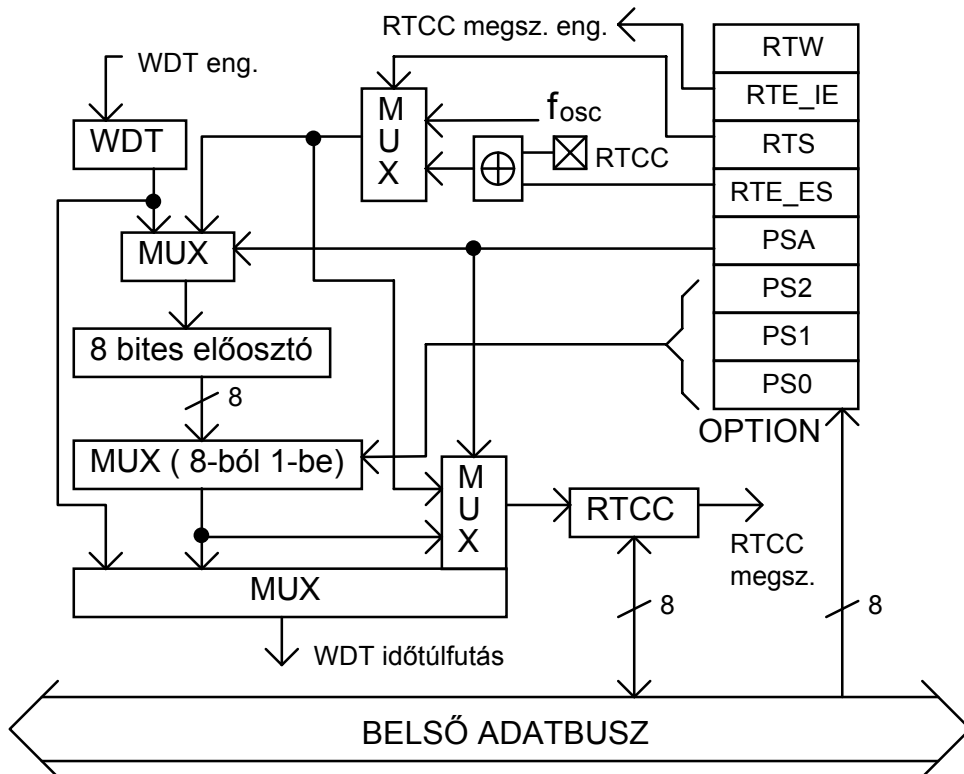
1. Watchdog hatékonysága függ attól, milyen jól van a felhasználói program megírva.
2. Az egész programban csak egy CLR !WDT utasítást használjunk, és a főhurokba tegyük. Semmiképpen se tegyük szubrutinba vagy megszakítási rutinba.
3. Olyan minimális WDT túlfutási időt válasszunk, amit a fő hurok végrehajtási ideje megenged.
4. A WDT-t csak akkor engedélyezzük, ha a programunk "laboratóriumi körülmények közt" már hibátlanul működik.

1.4.7 Az előosztó

Az előosztó (prescaler) egy 8 bites, nem memóriába ágyazott regiszter, tartalma a programfutás során nem érhető el. Hozzárendelhetjük akár az RTCC-hez, akár a Watchdog-hoz, de egyszerre csak az egyikhez. A választást az OPTION regiszter PSA bitjével végezhetjük el. Ha ez a bit 0 értékű, akkor az előosztó az RTCC-hez, ha 1 értékű, akkor a WDT-hez van hozzárendelve.

A watchdog esetében valójában utóosztóról (postscaler) kell beszélnünk. Az osztásviszonyt az OPTION regiszter PS2..PS0 bitjeivel állíthatjuk be.

PS2..PS0	RTCC osztásviszony (PSA=0)	Watchdog osztásviszony (PSA=1)
000	2	1 (timeout = 0.018 sec)
001	4	2 (timeout = 0.037 sec)
010	8	4 (timeout = 0.073 sec)
011	16	8 (timeout = 0.15 sec)
100	32	16 (timeout = 0.29 sec)
101	64	32 (timeout = 0.59 sec)
110	128	64 (timeout = 1.17 sec)
111	256	128 (timeout = 2.34 sec)



1.K ábra: Az időzítő/számláló áramkör felépítése

1.5 Megszakítási rendszer

Ha egy számítógépes rendszerben valamilyen esemény(ek) létrejöttét kívánjuk érzékelni, azt a szokásos módon kétféleképpen tehetjük meg. Például egy külső esemény létrejöttét egy bemeneti port-bit megváltozásának a figyelésével érzékelhetjük. Ezt a módszert a szakirodalom "polling"-nak, vagy csúnyán magyarul pollozásnak, pollingolásnak nevezi. Alkalmazása azonban lelassítja a rendszer tényleges működési sebességét, hiszen a processzor idejének jelentős részét azzal tölti, hogy ciklikusan megvizsgálja a kijelölt bemeneti bit állapotát. Sokkal szerencsésebb, ha **az esemény maga jelzi a processzor számára állapotának a megváltozását**. Ez a megoldás a **megszakítás** vagy ismert angol kifejezéssel az **interrupt**.

A megszakítás félbeszakítja az utasítások sorozatának (a felhasználói programnak) a végrehajtását, és a processzor egy ún. megszakításkiszolgáló szubrutint hajt végre, ami az események kezelését elvégzi. Ennek befejeztével a processzor visszatér a megszakított felhasználói program végrehajtására.

Az **SX** mikrokontrollernél két esemény válthat ki megszakítást, az **RTCC** túlsordulása, illetve a **B port** valamely lábán bekövetkezett jelszintváltás (**MIWU**).

Az **SX** mikrovezérlő megszakítási hardvere még az átlagos mikrokontrollerekénél is szerényebb. Így a megszakítás feldolgozás nagy részéről a programozónak kell szoftverből gondoskodni.

Egy megszakítási vektorcím van (000H), így a megszakításrutin elején nekünk kell megállapítani, hogy mely periféria kért megszakítást. **Nincs prioritási struktúra**, így az egyik megszakítás nem szakíthatja meg a másik kiszolgálását. Az ezidő alatt esetlegesen bekövetkező megszakítás "várakozó listára kerül", és az aktuális megszakításkérés kiszolgálása után azonnal kiszolgálásra kerül. Ha ezidő alatt több megszakítás érkezik, ezeket egyszerre kell feldolgoznunk. Például ha fut egy korábbi **MIWU** megszakítás kiszolgálása, és közben jön egy **RTCC**, aztán egy **MIWU**, majd még egy **MIWU** megszakításkérés, akkor az éppen futó megszakításkérés kiszolgálásának befejeztével csak egy további megszakítás következik be, és nem három. Mindhárom megszakításkérést ebben az egy rutinban kell kiszolgáltatnunk. Megjegyzés: Korábbi **SX** csipek gyártási hiba miatt a megszakításkiszolgálás közben bejövő újabb megszakításokat teljesen figyelmen kívül hagyták, de ez a probléma már megoldottnak tekinthető.

1.5.1 Az **RTCC** megszakítása

Az **RTCC** regiszter túlsordulása (**FFH**-ről **00H**-ra) megszakítást válthat ki, ha ez engedélyezve van. Ez azonban csak akkor lehetséges, ha a programozáskor az **SX**-et az ún. **kiterjesztett OPTION** módba állítottuk. Ha ez megtörtént, akkor az **RTCC** túlsordulást az **OPTION** regiszter **RTE_IE** bitjének törlésével engedélyezhetjük, 1-be állításával pedig tilthatjuk bármikor a felhasználói programunkban. Mivel az **RTCC** túlsordulás okozta megszakításhoz nem tartozik külön jelzőbit, ezért a megszakításokat kiszolgáló rutin elején meg kell vizsgálnunk az **RTCC** regiszter tartalmát (megszakításkor értéke nulla, vagy egy alacsony érték), és ennek alapján dönthetjük el, hogy a megszakítást az időzítő/számláló váltotta-e ki.

1.5.2 A "**B port**" megszakítása

A **B port** **MIWU** áramköre lehetővé teszi, hogy a port valamely lábán (lábain) végbemenő jelszintváltozás (él) megszakítást generáljon. A **B port** minden egyes lábán engedélyezhető/tiltatható a **MIWU** funkció a **WKEN_B** regiszter, a megszakítást kiváltó él típusa (felfutó/lefutó) a **WKED_B** regiszter segítségével állítható be. A megszakítás kiszolgáló rutin elején el kell döntenit, hogy a **B porton** van-e megszakításkérés folyamatban, ezt a **WKPND_B** regiszter kiértékelésével

tehetjük meg. **Nagyon fontos, hogy a megszakításkiszolgáló rutin folyamán — ha a MIWU-megszakítás engedélyezve van — a WKPND_B regisztert legalább egyszer kiolvassuk**, különben az újabb MIWU-megszakítások nem fognak érvényre jutni. A MIWU beállítását részletesen lásd az **1.3.6 szakaszban**.

1.5.3 A megszakításkiszolgálás hardver része

A mikrokontroller a következő folyamatot hajtja végre, ha megszakításkérést észlel:

1. *Megvizsgálja, hogy az adott megszakítás engedélyezve van-e (RTE_IE, WKEN_B).*
2. *Automatikusan elmenti a **programszámlálót**, a **W**, a **STATUS** és az **FSR** regisztereket az erre a célra fenntartott tükörregiszterekbe (tehát nem a verembe).*
3. *Végrehajt egy feltétel nélküli ugrást a 000H címre. **Turbó módban a megszakításra adott válaszidő az RTCC esetében három, a B port esetében öt gépi ciklus.** Ez az idő, amely a többszintű pipe-line (ld az 1.2 ábrát) kiürítése és újratöltése miatt eltelik a megszakításkérés észlelése és a megszakításkezelő rutin első utasításának a végrehajtása közt. **Kompatibilis módban a válaszidő kettő, illetve négy gépi ciklus.** Bizonyos könyvek itt 8 és 10 ciklust emlegetnek, de nem teszik hozzá, hogy gépi- vagy órajel ciklusról van-e szó.*
4. *Végrehajtja a megszakításkezelő szubrutint. **Figyelem!** Ennek a rutinnak mindig rövidebbnek kell lennie, mint két egymás utáni megszakítás bekövetkezése közötti legrövidebb lehetséges idő.*
5. *Visszaállítja az előzőleg elmentett **programszámláló**, **W**, **STATUS** és **FSR** regiszterek megszakítás előtti értékét, majd folytatja a felhasználási program végrehajtását ott, ahol a megszakítás előtt abbahagyta.*

1.5.4 A megszakításkezelő rutinok lezárása

A megszakítási rutint a szubrutinokhoz hasonlóan RETURN típusú utasításokkal zárhatjuk le, melyeket az **1.2.3 szakaszban** ismerhetünk meg. A két ide vonatkozó utasítás a RETI és a RETIW. Mindkét változat visszaállítja az előzőleg elmentett **programszámláló**, **W**, **STATUS** és **FSR** regiszterek megszakítás előtti értékét, majd folytatja a felhasználói program végrehajtását ott, ahol a megszakítás előtt abbahagyta.

A RETIW utasításnak van egy további funkciója az egyszerű RETI-hez képest: Mielőtt visszaállítaná a W korábbi értékét, hozzáadja **W** aktuális értékét az **RTCC**-hez.

Alapesetben az **RTCC** túlszordulása előosztó nélkül 256 gépi ciklusonként következik be. Ha engedélyezzük az **RTCC** megszakítást, akkor ilyenkor megszakítás is generálódik. Ha ennél sűrűbb időközönként szeretnénk **RTCC** megszakítást, akkor használhatjuk a **RETIW** utasítást:

```
MOV W, #-50
```

```
RETIW
```

Ebben az esetben az **RTCC** megszakítások 50 (ezt a számot **RETIW** értéknek hívjuk) gépi ciklusonként fognak bekövetkezni. Hogy is jön ez ki? Feltételezzük, hogy az éppen futó megszakítás **RTCC** = 0 értéknél következett be. Elértünk a megszakítási rutinunk végére, **RTCC** értéke most legyen mondjuk 29. Először is bevisszük **W**-be az 50 kettes komplementjét, ez: $256-50 = 206$ (az újabb végrehajtott utasítás miatt most már **RTCC** = 30). Majd **W**-t hozzáadjuk **RTCC**-hez (**RETIW** utasítás), ekkor **RTCC** értéke $206+30 = 236$ lesz. A következő megszakítás $256-236 = 20$ gépi ciklus múlva fog bekövetkezni. Figyelembe véve, a **RETIW** végrehajtásakor **RTCC** értéke 30 volt, kimondhatjuk, $30+20 = 50$ gépi ciklusonként fog **RTCC** megszakítás bekövetkezni.

A fenti "levezetést" tanulmányozva belátható, hogy — némely szakirodalom állításával ellentétben — a **RETIW** utasítás hozzáadja **W** tartalmát az **RTCC**-hez, nem pedig kivonja belőle.

Nagyon fontos, hogy a megszakítási rutin végrehajtási idejének mindig rövidebbnek kell lennie, mint a két egymás utáni megszakítás bekövetkezte közötti legrövidebb lehetséges idő. A megszakítási rutin végrehajtási idejébe természetesen a **RETI** (vagy **RETIW**) utasítás is beleszámít! A következő képlettel számítható a minimális **RTCC megszakítási periódus** (turbó módot feltételezve):

Minimum RTCC periódus = (megszakítási rutin max. végrehajtási ideje) + 6

A **RETIW** utasítás előtt ennek a kettes komplementjét kell **W**-be beírni. Természetesen az **RTCC periódust** célszerű ennél hosszabbra választani, hogy legyen kellő idő a főrutin üzemszerű végrehajtásához is, ha túl gyakran jön megszakítás, az túl sok processzor teljesítményt von el a főrutintól, szélsőséges esetben — ha az **RTCC periódust** a minimumra állítjuk — a főprogram végrehajtására már semmi ideje sem jut a processzornak.

Már csak egy kérdést kell tisztázni: hogyan jött ki a fenti képletbe a +6. Tudjuk, hogy turbó módban az **RTCC** megszakításra adott válaszütem három gépi ciklus. Ha a megszakítás egy olyan utasítás végrehajtása alatt következik be, melynek végrehajtási ideje több ciklus, akkor ehhez még hozzáadódik az idő, amíg

az adott utasítás végrehajtása befejeződik. Az SX-nél ilyen szempontból a "leghosszabb" utasítás az IREAD, amely 4 gépi ciklus idejű. tehát legrosszabb esetben emiatt még további három gépi ciklust várni kell a tényleges megszakítás kiszolgálás megkezdéséig.

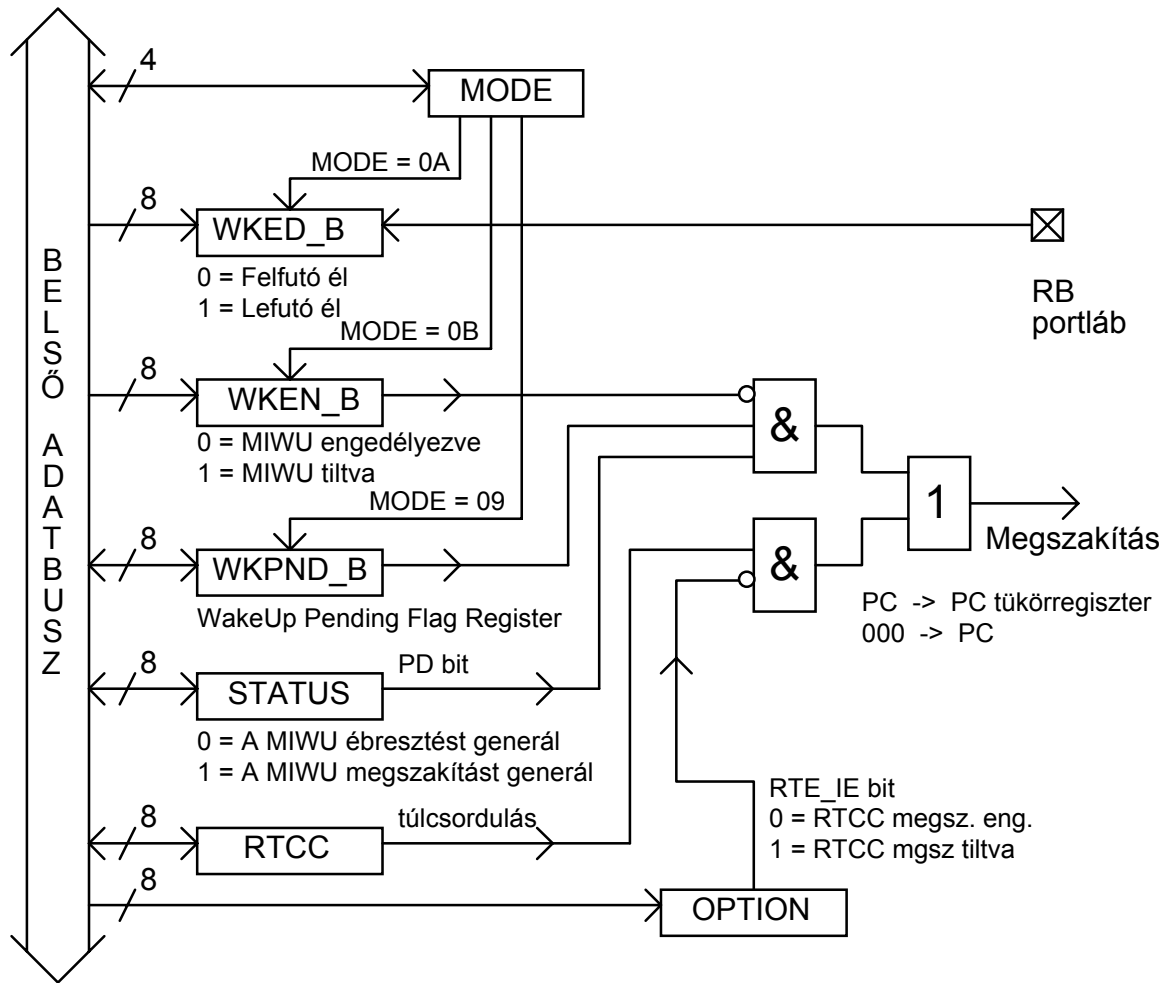
1.5.5 Példa egy megszakításkezelő rutinra

```
                                ; Engedélyezett megszakítások:
                                ; RTCC, RB0 és RB1 MIWU
                                ; -----
org    0                        ; a megszakításkezelő rutin kezdőcíme: 000H
mov    M,#$09                   ; WKPND_B kijelölése
clr    W                        ; W = 0
mov    !RB,W                    ; W = WKPND_B, WKPND_B = 0
and    W,#$03                   ; alsó két bit maszkolása
                                ; W most megmutatja, ki kért megszakítást:
                                ; 00H: RTCC, 01H: RB0, 02H: RB1
add    PC,W                     ; kiszámított ugrás W függvényében
jmp    rtcc_i                   ; ha W = 00H
jmp    rb0_i                    ; ha W = 01H
jmp    rbl_i                    ; ha W = 02H

rtcc_i                          ; RTCC megszakítását kiszolgáló rutin
...
reti

rb0_i                            ; RB0 megszakítását kiszolgáló rutin
...
reti

rbl_i                            ; RB1 megszakítását kiszolgáló rutin
...
reti
```



1.L ábra: A megszakítási áramkör

1.6 Egyéb fontos tulajdonságok

1.6.1 Konfigurációs biztosítók

A konfigurációs biztosítókat (biztosítékokat) a mikrovezérlő felprogramozásakor állítjuk be, és **ezek nem változtathatók a felhasználás során**. Az SX-KEY fejlesztőrendszer segítségével ezek megadása rendkívül egyszerű. Lássuk, miket lehet (kell) beállítani:

1. A mikrokontroller lábainak száma (18/28/40)
2. Az engedélyezett programtár lapok száma (1/2/4)
3. Az engedélyezett bankok száma az adattárban (1/2/4/8)

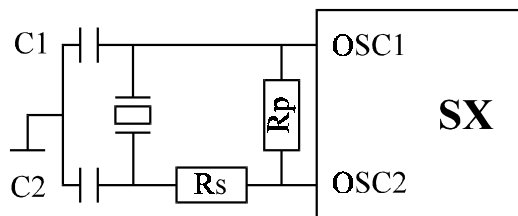
4. *Turbó vagy kompatibilis mód*
5. *Kiterjesztett verem üzemmód (8 vagy 2 szintes stack)*
6. *Kiterjesztett OPTION mód (RTW és RTE_IE bitek engedélyezése)*
7. *Kiterjesztett CARRY mód (C bit felhasználásra kerül összeadáskor ill. kivonáskor)*
8. *Szinkronozott bemenetek engedélyezése/tiltása (Ezt a funkciót mindig hagyjuk az alapértelmezett állapotban!)*
9. *Watchdog engedélyezése/tiltása*
10. *Kódvédelem bekapcsolása (Ekkor a programtár tartalma nem olvasható ki)*
11. *Brown-Out Reset engedélyezése/tiltása (ld. az 1.6.3 szakaszt)*
12. *Oscillátor típusa (ld. a következő szakaszt)*

1.6.2 Oszcillátor típusok

- LP:** kis teljesítményű üzem (alacsony frekvenciás kristállyal)
- XT:** kvarckristály vagy kerámia rezonátor
- HS:** nagy sebességű kvarckristály vagy kerámia rezonátor
- RC:** külső RC oszcillátor
- IRC:** belső RC oszcillátor.

LP, XT, HS oszcillátor típusok

A kvarckristály vagy a kerámia rezonátor bekötését az **1.13 ábrán** láthatjuk. A szükséges ellenállás és kapacitás értékek a táblázatban találhatóak.

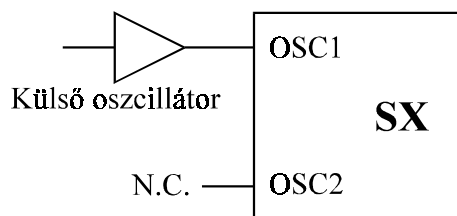


1.M ábra: Kvarckristály vagy kerámia rezonátor bekötése

Oscillátor típusa	Frekvencia	C ₁ [pF]	C ₂ [pF]	R _p	R _s
XT (rezonátor)	455 kHz	220	220	1M	6.8k
	1 MHz	100	100	1M	6.8k
	2 MHz	100	100	100k	680Ω
HS (rezonátor)	4 Mhz	100	100	100k	0
	4 MHz	belső (47p)	belső (47p)	100k	470Ω
	8 MHz	30	30	1M	0
	8 MHz	belső (47p)	belső (47p)	1M	470Ω
	12 MHz	30	30	1M	0
	20 MHz	10	10	1M	0
	50 MHz	belső (5p)	belső (5p)	33k	0
XT (kristály)	4 MHz	20	47	1M	0
HS (kristály)	8 MHz	20	47	1M	0
	12 MHz	20	47	1M	0
	16 MHz	15	30	1M	0
	20 MHz	15	30	1M	0
	25 Mhz	5	20	10k	0
	50 MHz	5	15	220k	0

Független külső oszcillátor áramkör alkalmazása

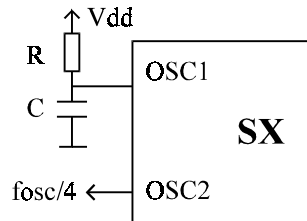
Független külső oszcillátor áramkör alkalmazása esetén az oszcillátor típusát a sebességtől függően **LP**, **XT** vagy **HS** módba állítjuk. A külső oszcillátor kimenetét az **SX OSC1** lábára csatlakoztatjuk, az **OSC2** lábat nem kötjük be sehova.



1.N ábra: Független külső oszcillátor bekötése

Külső RC oszcillátor

A külső RC oszcillátor bekötési módja az **1.15 ábrán** látható. Ebben az esetben az **OSC2** lábón az oszcillátor frekvencia negyede mérhető.



1.O ábra: Külső RC oszcillátor bekötése

Az ellenállás és a kondenzátor értéke szabadon választható, de célszerű betartani a következőket:

$$3\text{k}\Omega < R_{\text{EXT}} < 100\text{k}\Omega$$

$$20\text{pF} < C_{\text{EXT}}$$

Belső RC oszcillátor

A belső RC oszcillátor alkalmazása esetén az **OSC1**, **OSC2** lábakra nem csatlakoztatunk semmit. Az **SX** nyolc különböző értéket kínál fel a belső RC oszcillátor frekvenciájára: **31.25 kHz, 62.5 kHz, 125 kHz, 250kHz, 500 kHz, 1 MHz, 2 MHz, 4 MHz**. A belső oszcillátor frekvencia-pontossága +/- 8%.

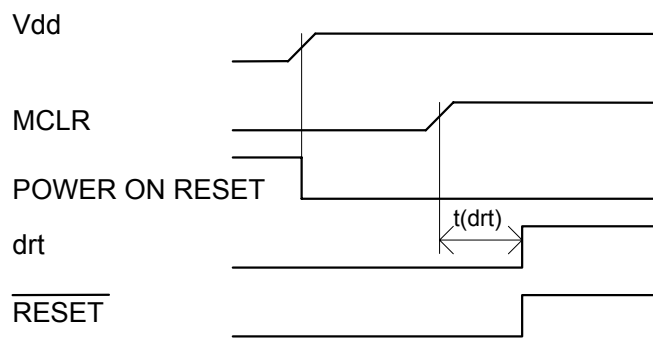
1.6.3 A RESET folyamat

RESET után a programszámláló a programtár tetejére mutat, ennek címe 2k esetén 7FFH. Ide NOP utasítást írva a valós reset vektorcím 00H lesz, de ez nem célszerű, mivel 000H a fix megszakítási vektorcím is egyben. **A gyakorlatban a reset vektorcímre egy ugró utasítást helyeznek el, ami a tényleges programkezdetre ugrik.** Reset folyamatot a következő események válthatnak ki:

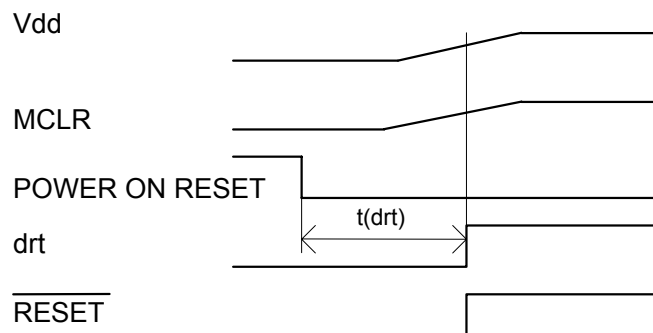
1. *Bekapcsolás (Power-On-Reset: POR)*
2. *Alacsony szintű impulzus az MCLR (Master Clear Reset) lábón*
3. *WDT túlfutása*
4. *Ébredés energiatakarékos üzemmódból (MIWU funkcióval)*
5. *Visszatérés Brown-Out állapotból*

Bekapcsolás (Power-On-Reset: POR)

Ideális esetben a bekapcsolási folyamat úgy zajlana le, hogy az SX V_{DD} lábára ugrásszerű feszültség kerül. Mint tudjuk a valóságban ez soha nincs így, a tápfeszültség mindig egy meghatározott (közelítőleg ismert) időtartam alatt éri el az állandósult állapotbeli értékét. Az SX-ben beépített **POR-áramkör** gondoskodik arról, hogy a processzor működése csak ezen tranzienst követően induljon meg. Ez az áramkör egy aszinkron számláló (Delay Reset Timer - **DRT**), mely hozzávetőlegesen 72 ms alatt csordul túl. A számláló akkor indul el, amikor érvényes logikai magas szint jelenik meg az **MCLR** lábon. Ha nem akarunk külső **RESET**-et kezdeményezni, és a tápfeszültség jelváltozási sebessége kellően magas, az **MCLR** lábat összeköthetjük a V_{DD} lábbal. Viszont ha ez a jelváltozási sebesség alacsony, és a tranzienst ideje meghaladja a **DRT** által megszabott 72 ms-ot, akkor a processzor működése megindul, mielőtt még a tápfeszültség elérte volna a kívánt értéket (**1.17 ábra**).



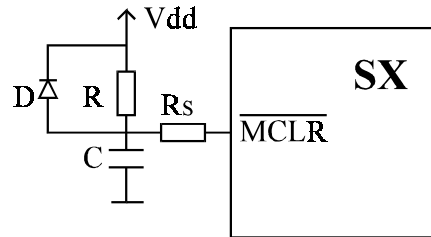
1.P ábra: Természetes RESET folyamat



1.Q ábra: A V_{DD} és az **MCLR** lábak összekötve. A tápfeszültség jelváltozási sebessége túl alacsony.

Ez az állapot a helyes programvégrehajtás szempontjából beláthatatlan következményekkel járhat. Ilyenkor az **1.18 ábra** szerinti RC-hálózat segít, az **MCLR** lábra késleltetett jel kerül. A késleltetés idejét mindig hosszabbra kell

választani, mint az az idő, ami alatt a tápfeszültség eléri a kívánt értéket. A késleltetés idejét az RC-hálózat időállandója határozza meg. R értékét célszerű 40k Ω -nál kisebbre választani. A dióda szerepe mindössze annyi, hogy felgyorsítja a kondenzátor kisülését kikapcsolás esetén, a gyakorlatban akár el is hagyható. R_s áramkorlátozó ellenállás a mikrovezérlő **MCLR** lábát védi a külső zavaroktól, sokszor ez is elhagyható, de ha használjuk, értéke ne legyen nagyobb 1 k Ω -nál.



1.R ábra: Az MCLR lábára jutó jel késleltetése külső RC hálózattal

Alacsony szintű impulzus az MCLR (Master Clear Reset) lábon

Igény esetén bármikor **RESET** folyamat kezdeményezhető kívülről, egy az **MCLR** lábára adott alacsony szintű impulzussal. Az SX mindaddig **RESET** állapotban marad, amíg az **MCLR** lábon lévő jel alacsony szintű. Ha nem akarunk külső **RESET**-et kezdeményezni, és a tápfeszültség jelváltozási sebessége kellően magas, az **MCLR** lábat összeköthetjük a V_{DD} lábbal.

WDT túlfutása

Hatása teljesen ugyanaz, mintha alacsony szintű impulzust adnánk az **MCLR** lábára.

Ébredés energiatakarékos üzemmódból (MIWU funkcióval)

Ebben az esetben feltételezzük, hogy a V_{DD} lábon az állandósult tápfeszültség van jelen, ezért a Delay Reset Timer (**DRT**) nem lép üzembe, így elmarad a 72 ms-os **RESET**-késleltetés

Brown-Out állapot

Azt az esetet, amikor a tápfeszültség üzem közben valamilyen ok miatt egy meghatározott érték alá csökken, Brown-Out állapotnak hívjuk. Ez az állapot a helyes programvégrehajtás szempontjából beláthatatlan következményekkel járhat, így célszerű ellene védekezni. Természetesen az SX-ben erről is beépített áramkör gondoskodik, melynek engedélyezését vagy tiltását programozáskor kell beállítani, ezt működés közben a felhasználói programból már nem bírálhatjuk felül. Ha engedélyeztük, a **Brown-Out áramkör folyamatosan figyeli a tápfeszültség**

értékét, ha az 4.2V alá csökken RESET állapotba hozza a processzort, és mindaddig ott is tartja, amíg a tápfeszültség újra 4.2V fölé nem kerül.

A regiszterek állapota különböző RESET-ek esetén

A következő táblázat megmutatja az egyes speciális funkciójú regiszterek értékét különböző **RESET** folyamatok lezajlása után. Azokat a regisztereket, melyek ismeretlen értéket vesznek fel, célszerű **RESET** után kezdőértékkel ellátni (inicializálni).

Regiszter	Power-On	Ébredés	Brown-Out	WDT	MCLR
W	ismeretlen	változatlan	ismeretlen	változatlan	változatlan
OPTION	FFH	FFH	FFH	FFH	FFH
MODE	0FH	0FH	0FH	0FH	0FH
RTCC	ismeretlen	változatlan	ismeretlen	változatlan	változatlan
PC	FFH	FFH	FFH	FFH	FFH
STATUS	0..2 bit: ismeretlen 3..4 bit: 1 5..7 bit: 0	0..2 bit: változatlan 3..4 bit: változatlan 5..7 bit: 0	0..2 bit: ismeretlen 3..4 bit: ismeretlen 5..7 bit: 0	0..2 bit: változatlan 3..4 bit: lásd a köv. pontot 5..7 bit: 0	0..2 bit: változatlan 3..4 bit: lásd a köv. pontot 5..7 bit: 0
FSR	ismeretlen	0..6 bit: változatlan 7 bit: 1	0..6 bit: ismeretlen 7 bit: 1	0..6 bit: változatlan 7 bit: 1	0..6 bit: változatlan 7 bit: 1
Port irány	FFH	FFH	FFH	FFH	FFH
Port adat	ismeretlen	változatlan	ismeretlen	változatlan	változatlan
Egyéb reg.	Ismeretlen	változatlan	ismeretlen	változatlan	változatlan
CMP_B	0, 6..7 bit: 1 1..5 bit: ismeretlen	0, 6..7 bit: 1 1..5 bit: ismeretlen	0, 6..7 bit: 1 1..5 bit: ismeretlen	0, 6..7 bit: 1 1..5 bit: ismeretlen	0, 6..7 bit: 1 1..5 bit: ismeretlen
WKPND_B	ismeretlen	változatlan	ismeretlen	változatlan	változatlan
WKED_B	FFH	FFH	FFH	FFH	FFH
WKEN_B	FFH	FFH	FFH	FFH	FFH
ST_x	FFH	FFH	FFH	FFH	FFH
LVL_x	FFH	FFH	FFH	FFH	FFH
PLP_x	FFH	FFH	FFH	FFH	FFH
WDT	ismeretlen	változatlan	ismeretlen	változatlan	változatlan

A TO, PD bitek kiértékelése RESET után

a RESET folyamat lezajlása után a STATUS regiszter TO és PD bitjeinek állapota alapján következtethetünk a RESET-et megelőző eseményekre.

TO	PD	RESET oka
0	0	WDT időtúlfutás energiatakarékos üzemben
0	1	WDT időtúlfutás normál üzemben
1	0	külső RESET (MCLR) energiatakarékos üzemben
1	1	Power-On, Bekapcsolás
X	X	külső RESET (MCLR) normál üzemben

1.6.4 Energiatakarékos üzemmód

Energiatakarékos üzemmódban az SX órajele és ezzel együtt minden szolgáltatása leáll, kivéve a **WDT** (ld. az **1.4.5 szakaszt**), ha engedélyezve van. Az energiatakarékos állapotba a **SLEEP** utasítás hatására lép be a mikrovezérlő. A **STATUS** regiszter **PD** bitje törlődik, a **TO** bit 1-re vált, és ha engedélyezve van, akkor a **WDT** is törlődik, de ezután tovább számlál. Mivel az energiatakarékos üzemmódba a **SLEEP** (alvás) utasítással lépünk be, értelemszerűen az onnan való kilépést "ébredésnek" hívjuk. Három esemény válthat ki ébredést:

1. *A WDT túlfutása*
2. *Érvényes jelszintváltás (él) egy olyan portlábon, melyre a MIWU funkció (ld. az 1.3.6 szakaszt) engedélyezett*
3. *Alacsony szintű impulzus az MCLR (Master Clear Reset) lábon*

Bármelyik eset is áll elő, a processzor egy RESET folyamatot hajt végre (ld. az **1.3.6 szakaszt**).

1.7 Az SX mikrovezérlő utasításkészlete

Egy kis történelem: A MICROCHIP cég a kilencvenes évek elején jelent meg a mikrovezérlő piacon, de PIC kontrollereinek szokatlan utasításkészlete sokáig távol tartotta a tapasztalt, Intel jellegű kódokhoz szokott felhasználókat. Ez kezdetben a PIC mikrovezérlők elterjedésének is komoly gátja is volt. Ezt a helyzetet jókor ismerte fel egy másik amerikai cég, a Parallax, és egy megszokott felépítésű és működésű utasításkészletet definiált. Ez az utasításkészlet a népszerű

MCS-51 utasításkészletéhez hasonlít. Egy Parallax-utasítás egy vagy több eredeti PIC utasításból épül fel, így a Parallax utasításkészlet tulajdonképpen egy előre definiált makrórendszer. Szokták az ilyen utasításokat makró utasításoknak is hívni.

Közben azonban telt az idő és felnőtt a programozók egy újabb generációja, akik mögött már nem volt több éves MCS-51-es gyakorlat, így ők már az eredeti MICROCHIP utasításkészletet kezdték használni. **Itt a mese vége.**

Az eredeti '5x-es PIC család utasításkészlete 33 utasítást tartalmazott. Lényegében ezt vette át a SCENIX az SX mikrokontrollerhez, de kibővítette a készletet további néhány utasítással. Minden utasítás egyszavas, ami azt jelenti, hogy egy rekeszt foglal el a 12-bites szóhosszúságú programtárban. Az utasításokhoz rendelt mnemonikokat viszont már nem vették át a PIC-ektől, ezeket újradefiniálták. (Megjegyzés: **Mivel a gépi kódú utasítások csak számok, mnemonikoknak hívjuk azokat az ember által könnyebben megjegyezhető szöveges szimbólumokat, amiket az assembler program fordít majd le az adott gépi kódú utasítást reprezentáló számra.**) A `MOVLW 55H` PIC-utasítás gépi kódja teljesen azonos a `MOV W, #55` SX-utasítás gépi kódjával. Ennek az új mnemonikának az alapja a Parallax-féle programnyelv volt. Így viszont sajnos ugyanaz a probléma állt elő, mint a PIC-ek megjelenésénél: azok a felhasználók, akik az eredeti MICROCHIP kódban programoztak PIC-et, szerették volna az SX-et is így programozni, de ezt sem a SCENIX, sem a Parallax nem támogatja. (Ráadásul a Parallax korábbi fordítói még elfogadták az eredeti MICROCHIP kódot is.)

Tévedések elkerülése végett: A Scenix által definiált utasításkészlet nem azonos a Parallax utasításkészlettel, Scenix csak az egyszavas Parallax utasítások mnemonikjait vette át, a makró utasításokat nem. A makró utasítások lehetősége csak a Parallax fordító egyik extra-szolgáltatása, ezeket nem kötelező használni. A [3] szakirodalom részletesen leírja, hogy az egyes makró utasítások mely egyszavas utasításokból épülnek fel. Viszont **ha makró utasításokat is használunk, néhány dologra nem árt odafigyelni:**

- 1. Néhány makró utasítás — melyek látszólag nem érintik a W regisztert — végrehajtása után W eredeti tartalma megváltozhat. Pl.: `MOV fr1, fr2`
Ezt a műveletet csak úgy tudja végrehajtani, hogy először fr2 tartalmát beviszi W-be, majd W tartalmát átmásolja az fr1-be. Tehát az végrehajtás után W az fr2 regiszter értékével lesz azonos, eredeti tartalma pedig elvész.*
- 2. A TEST-SKIP utasítások csak egyszavas utasításokat tudnak átugrani. Ha egy TEST-SKIP utasítást egy makró utasítás követ, akkor átugrás esetén szépen beleugrik a makró közepébe.*
- 3. Az SX programozáskor konfigurálható úgy is, hogy a C (carry) bit is részt vegyen az összeadó és kivonó műveletekben, megkönnyítve ezzel a többbájtos összeadási és kivonási műveleteket. Bizonyos összehasonlító makró*

utasítások (pl.: CSB fr1,fr2) tartalmaznak kivonást, ilyenkor ezek C=1 esetén hibás eredményt adnak.

Az utasítások részletes ismertetése megtalálható az [1], [2], [3] szakirodalmakban, a következő táblázat csak összefoglalást nyújt az összes Parallax által definiált utasításról, függetlenül attól, hogy egyszavasak, vagy makró utasítások azok. A látszólagos utasításbőséget tovább növeli, hogy bizonyos egyszavas utasításokhoz több mnemonikot is rendeltek. Ki-ki azt használhatja, amelyik neki szimpatikus. Így az ADD PC,W utasítás teljesen azonos a JMP PC+W utasítással, a SETB C az STC -vel, stb.

Az első oszlop tartalmazza az utasítás mnemonikáját. A második oszlop megmutatja, hogy hány szóból áll, ha ez kettő vagy több, akkor makró utasításról van szó. A harmadik oszlop megmutatja, hogy az adott utasítás mely jelzőbitekkel állítja, vagy módosítja. Ha ugró vagy szubrutinhívó utasítások esetén ebben az oszlopban a szám mellett egy csillag is áll, az arra utal, hogy @cím használata esetén a @ jel miatt beiktat egy további PAGE utasítást is (ld. az 1.2.3 szakaszt). A negyedik oszlop azt mutatja meg, hogy az utasítás használja-e a W regisztert, vagyis végrehajtása után W értéke megváltozik-e. A negyedik oszlopban az utasítás precíz angol nyelvű elnevezése látható, ez megkönnyíti a mnemonikok megjegyzését.

INSTRUCTION SET

Instruction		Words	Flags	W used	Description
byte-oriented operations					
MOV	W,#lit	1	-	-	Move literal into W
MOV	W,fr	1	Z	-	Move fr into W
MOV	fr,W	1	-	-	Move W into fr
MOV	fr,#lit	2	-	W	Move literal into fr
MOV	fr1,fr2	2	Z	W	Move fr2 into fr1
ADD	W,fr	1	C,DC,Z	-	Add fr into W
ADD	fr,W	1	C,DC,Z	-	Add W into fr
ADD	fr,#lit	2	C,DC,Z	W	Add literal into fr
ADD	fr1,fr2	2	C,DC,Z	W	Add fr2 into fr1
MOV	W,fr-W	1	C,DC,Z	-	Move fr-W into W
SUB	fr,W	1	C,DC,Z	-	Subtract W from fr
SUB	fr,#lit	2	C,DC,Z	W	Subtract literal from fr
SUB	fr1,fr2	2	C,DC,Z	W	Subtract fr2 from fr1
AND	W,#lit	1	Z	-	AND literal into W
AND	W,fr	1	Z	-	AND fr into W
AND	fr,W	1	Z	-	AND W into fr
AND	fr,#lit	2	Z	W	AND literal into fr
AND	fr1,fr2	2	Z	W	AND fr2 into fr1
OR	W,#lit	1	Z	-	OR literal into W
OR	W,fr	1	Z	-	OR fr into W
OR	fr,W	1	Z	-	OR W into fr

OR	fr,#lit	2	Z	W	OR literal into fr
OR	fr1,fr2	2	Z	W	OR fr2 into fr1
XOR	W,#lit	1	Z	-	XOR literal into W
XOR	W,fr	1	Z	-	XOR fr into W
XOR	fr,W	1	Z	-	XOR W into fr
XOR	fr,#lit	2	Z	W	XOR literal into fr
XOR	fr1,fr2	2	Z	W	XOR fr2 into fr1
CLR	W	1	Z	-	Clear W
CLR	fr	1	Z	-	Clear fr
MOV	W,++fr	1	Z	-	Move fr+1 into W
INC	fr	1	Z	-	Increment fr
MOV	W,--fr	1	Z	-	Move fr-1 into W
DEC	fr	1	Z	-	Decrement fr
MOV	W,<<fr	1	C	-	Move left-rotated fr into W
RL	fr	1	C	-	Rotate left fr
MOV	W,>>fr	1	C	-	Move right-rotated fr into W
RR	fr	1	C	-	Rotate right fr
MOV	W,<>fr	1	-	-	Move nibble-swapped fr into W
SWAP	fr	1	-	-	Swap nibbles in fr
NOT	W	1	Z	-	Perform not on W
MOV	W,/fr	1	Z	-	Move not'd fr into W
NOT	fr	1	Z	-	Perform not on fr
TEST	W	1	Z	-	Test W for zero
TEST	fr	1	Z	-	Test fr for zero

bit-oriented operations

CLRB	bit	1	-	-	Clear bit
CLC		1	C	-	Clear carry
CLZ		1	Z	-	Clear zero
SETB	bit	1	-	-	Set bit
STC		1	C	-	Set carry
STZ		1	Z	-	Set zero
ADDB	fr,bit	2	Z	-	Add bit into fr
ADDB	fr,/bit	2	Z	-	Add not bit into fr
SUBB	fr,bit	2	Z	-	Subtract bit from fr
SUBB	fr,/bit	2	Z	-	Subtract not bit from fr
MOVB	bit1,bit2	4	-	-	Move bit2 into bit1
MOVB	bit1,/bit2	4	-	-	Move not bit2 into bit1

inc/dec-conditional branches

MOVSZ	W,++fr	1	-	-	Move fr+1 into W, skip if zero
INCSZ	fr	1	-	-	Increment fr,

IJNZ	fr,addr	2 *	-	-	skip if zero Increment fr, jump if not zero
MOVSZ	W,--fr	1	-	-	Move fr-1 into W, skip if zero
DECSZ	fr	1	-	-	Decrement fr, skip if zero
DJNZ	fr,addr	2 *	-	-	Decrement fr, jump if not zero

compare-conditional branches

CSE	fr,#lit	3	C,DC,Z	W	Compare, skip if equal
CSE	fr1,fr2	3	C,DC,Z	W	Compare, skip if equal
CSNE	fr,#lit	3	C,DC,Z	W	Compare, skip if not equal
CSNE	fr1,fr2	3	C,DC,Z	W	Compare, skip if not equal
CSA	fr,#lit	3	C,DC,Z	W	Compare, skip if above
CSA	fr1,fr2	3	C,DC,Z	W	Compare, skip if above
CSAE	fr,#lit	3	C,DC,Z	W	Compare, skip if above or equal
CSAE	fr1,fr2	3	C,DC,Z	W	Compare, skip if above or equal
CSB	fr,#lit	3	C,DC,Z	W	Compare, skip if below
CSB	fr1,fr2	3	C,DC,Z	W	Compare, skip if below
CSBE	fr,#lit	3	C,DC,Z	W	Compare, skip if below or equal
CSBE	fr1,fr2	3	C,DC,Z	W	Compare, skip if below or equal
CJE	fr,#lit,addr	4 *	C,DC,Z	W	Compare, jump if equal
CJE	fr1,fr2,addr	4 *	C,DC,Z	W	Compare, jump if equal
CJNE	fr,#lit,addr	4 *	C,DC,Z	W	Compare, jump if not equal
CJNE	fr1,fr2,addr	4 *	C,DC,Z	W	Compare, jump if not equal
CJA	fr,#lit,addr	4 *	C,DC,Z	W	Compare, jump if above
CJA	fr1,fr2,addr	4 *	C,DC,Z	W	Compare, jump if above
CJAE	fr,#lit,addr	4 *	C,DC,Z	W	Compare, jump if above or equal
CJAE	fr1,fr2,addr	4 *	C,DC,Z	W	Compare, jump if above or equal
CJB	fr,#lit,addr	4 *	C,DC,Z	W	Compare, jump if below
CJB	fr1,fr2,addr	4 *	C,DC,Z	W	Compare, jump if below
CJBE	fr,#lit,addr	4 *	C,DC,Z	W	Compare, jump if below or equal
CJBE	fr1,fr2,addr	4 *	C,DC,Z	W	Compare, jump if below or equal

bit-conditional branches

SB	bit	1	-	-	Skip if bit
SC		1	-	-	Skip if carry
SZ		1	-	-	Skip if zero
SNB	bit	1	-	-	Skip if not bit
SNC		1	-	-	Skip if not carry
SNZ		1	-	-	Skip if not zero

JB	bit,addr	2 *	-	-	Jump to address if bit
JC	addr	2 *	-	-	Jump to address if carry
JZ	addr	2 *	-	-	Jump to address if zero
JNB	bit,addr	2 *	-	-	Jump to address if not bit
JNC	addr	2 *	-	-	Jump to address if not carry
JNZ	addr	2 *	-	-	Jump to address if not zero

unconditional branches

SKIP		1	-	-	Skip next instruction
JMP	addr	1 *	-	-	Jump to address
JMP	PC+W	1	C,DC,Z	-	Add W into PC(L)
JMP	W	1	-	-	Move W into PC(L)
CALL	addr	1 *	-	-	Call to address
RETW	lit,lit...	1	-	-	Return from call, literal in W
RET		1	-	-	Return from call
RETP		1	-	-	Return from call, affect PA2:PA0
RETI		1	-	-	Return from interrupt
RETIW		1	-	-	Return from interrupt, and compensate RTCC

i/o and control operations

PAGE into	addr	1	-	-	Transfer addr.11:addr.9 PA2:PA0
BANK	fr	1	-	-	Transfer fr.7:fr.5 into FSR.7:FSR.5
MOV	M,#lit	1	-	-	Move literal into M
MOV	M,W	1	-	-	Move W into M
MOV	M,fr	2	Z	W	Move fr into M
MOV	W,M	1	-	-	Move M into W
MOV	fr,M	2	-	W	Move M into fr
MOV	!port,W	1	-	-	Move W into port's TRIS
MOV TRIS	!port,#lit	2	-	W	Move literal into port's
MOV	!port,fr	2	Z	W	Move fr into port's TRIS
MOV	!OPTION,W	1	-	-	Move W into OPTION
MOV	!OPTION,#lit	2	-	W	Move literal into OPTION
MOV	!OPTION,fr	2	Z	W	Move fr into OPTION
CLR	!WDT	1	TO,PD	-	Clear WDT and prescaler
SLEEP		1	TO,PD	-	Clear WDT and enter sleep mode
IREAD		1	-	-	Read instruction at M:W into M:W
NOP		1	-	-	No operation

2. Programfejlesztés és a Virtuális Perifériák

2.1 Egy egyszerű mintapélda

Nagyon fontos, hogy forrásnyelvű programunk megjelenése, formátuma valamilyen módon ésszerű és szabványos legyen. Ezenkívül számos dolgot kell definiálnunk ahhoz, hogy egyáltalán lehetséges legyen a program fordítása (pl. tárterület-deklarációk, a program végének szabványos jelzése stb.). Ezért a következő mintapélda egy egyszerű programstruktúra vázat mutat be, amely azonban minden fontos és elengedhetetlen elemet tartalmaz, így a nagyobb lélegzetű programok készítésénél is iránymutató lehet.

```
*****
;*                               K K M F  Automatika Intézet          *
;*                               Elektronika Szakcsoport            *
;*                               *****                          *
;* Program:                       FIRST.SRC                        *
;* Változat:                       V1.0                            *
;* Dátum:                          1999. április 02.             *
;* Készítette:                      Lamár Krisztián                *
;*                               *****                          *
;
        DEVICE PINS28, PAGES1, BANKS4, OSCXT, OPTIONX
        ID      'FIRST'
        FREQ    500_000          ; 500 kHz
        RESET   START

;*****
;                               változók
;*****
        ORG     $08

BYTE1   DS     1
BYTE2   DS     1
WORD    DS     2          ; 16 bites helyfoglalás
;                               ; LSB van előbb (a kisebb címen)
;                               ; A WATCH diraktíva miatt van így,
;                               ; általában fordítva szoktuk.

LED      EQU    RB.6
BUZZER   EQU    RB.7

        WATCH  BYTE1,8,UHEX
        WATCH  BYTE2,8,UHEX
        WATCH  WORD,16,UHEX
        WATCH  LED,1,UBIN
        WATCH  RTCC,8,UDEC
```

```
;*****
; megszakítási vektorcím
;*****
ORG      $000

SETB     LED                ; LED & zümi bekapcs, ha
SETB     BUZZER            ; RTCC = 0, RTCC túlcsor-
RETI     ; dulásakor megszakítás

;*****
; felhasználói program
;*****

;===== [ inicializálás ] =====

START    MOV     BYTE1,#$00          ; kezdőértékek
          MOV     BYTE2,#$00
          MOV     WORD,#$00
          MOV     WORD+1,#$00
          MOV     FSR,#$10

          MOV     !OPTION,#%10000111 ; OPTION beállítás:
          ; RTCC az órajel hatására inkrementálódik, előosztó = 1/256
          ; 01H címen az RTCC érhető el, RTCC interrupt engedélyezve

          MOV     !RB,#%00111111    ; port irány beáll.
          ; RB.6, RB.7 kimenet

;===== [ fő programhurok ] =====

          BREAK          ; töréspont ide
LOOP      MOV     BYTE1,#3
LOOP2     MOV     W,BYTE1           ; három ciklusos DJNZ
          DJNZ    BYTE1,LOOP2

          ADD     BYTE2,#5          ; BYTE2 = BYTE2 + 5

          MOV     INDF,BYTE2        ; indirekt címzés
          INC     FSR                ; az alsó 16 bájtot nem
          SETB    FSR.4             ; bántjuk

          INC     WORD                ; 16 bites inkrementálás
          SNZ     ; először LSB, ha túlcsordult:
          INC     WORD+1            ; MSB inkrementálása

          CJNE    RTCC,#$80,LOOP
          CLRB    LED                ; LED & zümi kikapcs,
          CLRB    BUZZER            ; ha RTCC = 80H (128 decimal)
          JMP     LOOP

END
```

Minden sornál, ami pontosvesszőt tartalmaz, a pontosvessző utáni információt a fordító figyelmen kívül hagyja. Ezek az ún. **kommentek**, és azt **célt szolgálgják**, hogy a programozó könnyebben eligazodjon a saját munkáján.

A direktívák olyan speciális utasítások, amelyek a fordítónak szólnak, nem az SX mikrovezérlőnek, a végleges gépi kódú programban ezek már nem játszanak szerepet. A fenti példában a következő parancsok direktívák: DEVICE, ID, FREQ, RESET, ORG, DS, EQU, WATCH, BREAK, END. Az egyes

direktívák használata majd hogyanem kézenfekvő a mintapélda alapján, ez alól talán csak a WATCH a kivétel. A pontos specifikációk megtalálhatók a [3] szakirodalomban. Ha a WATCH direktíva megértése elsősre gondot okozna, lépünk tovább, szerepe nem kifejezetten jelentős, kihagyása nem végzetes.

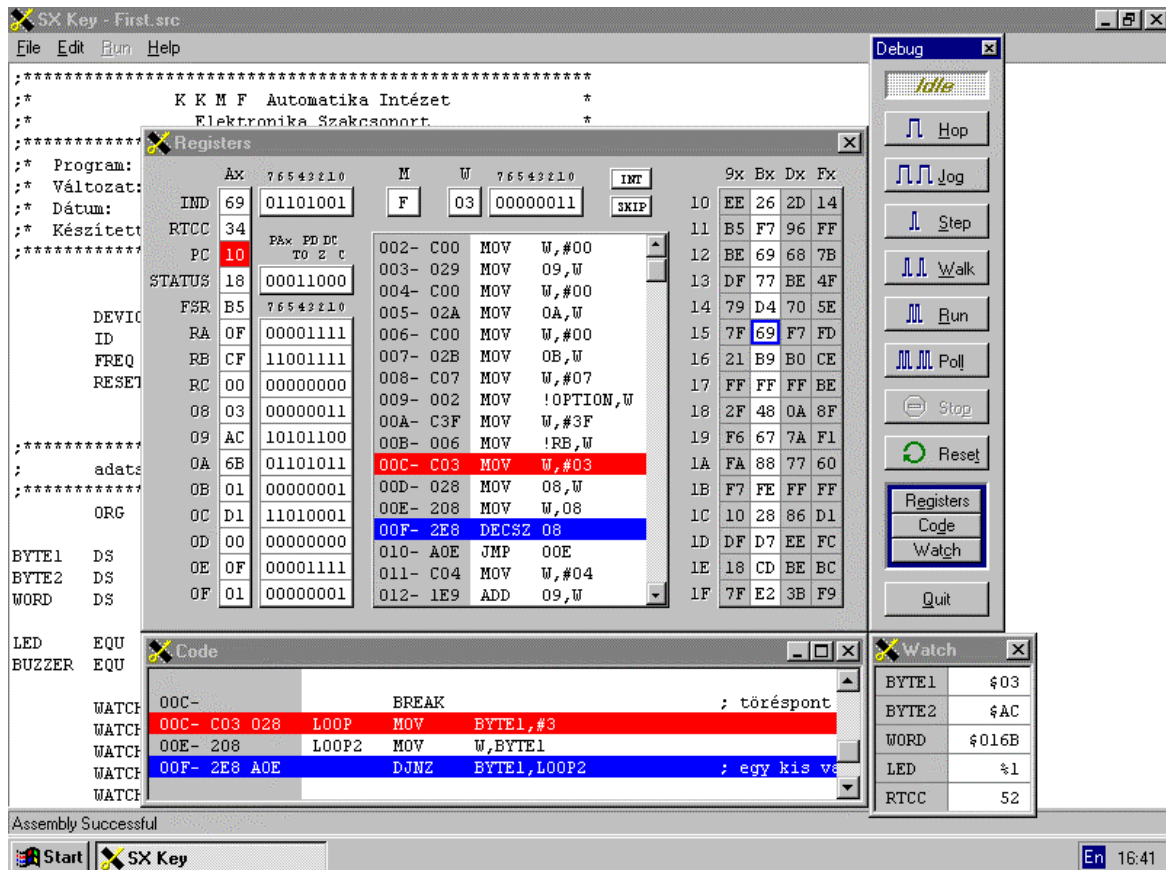
Néhány fontos megjegyzés:

- 1. A DEVICE direktíva azokat a paraméterekeket tartalmazza, melyeket az SX programozásakor lehet csak beállítani, és a programfutás közben már nem módosíthatók. Itt lehet például a kiterjesztett OPTION módot is beállítani.*
- 2. RESET után a programszámláló a programtár tetejére mutat, ennek címe 2k esetén 7FFH. Ide NOP utasítást írva, a valós reset vektorcím 00H lesz, de ez nem célszerű, mivel 000H a fix megszakítási vektorcím is egyben. A gyakorlatban a reset vektorcímre egy ugró utasítást helyeznek el, ami a tényleges programkezdetre ugrik. Ennek egyszerű megvalósítására szolgál a RESET direktíva.*
- 3. A BREAK direktíva automatikusan elhelyez egy töréspontot az utána következő sorra.*
- 4. A programot az END direktívával kell lezárni.*

2.2 Az SX-KEY fejlesztőrendszer

Miután elindítottuk az **SX-KEY.EXE** programot, olvassuk be a **FIRST.SRC** mintapéldát (File menü | Open). Amennyiben az SX-KEY fejlesztőeszköz (a továbbiakban hívjuk egyszerűen kulcsnak) megfelelően csatlakoztatva van a PC-hez, illetve a — szintén Parallax fejlesztésű — SX-KEY Demo Board-hoz, töltsük le és indítsuk el a programot (Run menü | Run). Rövid művelet után a Demo Board-ra épített hangszóróból lassú kattogás hallatszik, illetve a sárga LED ugyanilyen ütemben villog. A program ezen kívül még egyéb egyszerű műveleteket is elvégez, de ezt már csak a hibakereső/nyomkövető módban figyelhetjük meg.

A hibakereső/nyomkövető módba a Run menü | Debug parancsával léphetünk be. A fejlesztőrendszer ismételten letölti a programot, és megjelenik a **2.1 ábrán** láthatóhoz hasonló kép.



2.A ábra: Az SX-KEY fejlesztőrendszer hibakreső/nyomkövető üzemmódjának képe

Az ablakban nyomon követhetjük, hogy hol tart a programunk végrehajtása, megfigyelhetjük regiszterek tartalmát, egy részüket még bináris alakban is. Bármelyik regiszter értéke módosítható akár két utasítás végrehajtása közt is. Az egérrel a regisztermezőre kattintunk, majd a kívánt értéket kézzel begépeljük. A változás az ENTER billentyűvel juttatható érvényre. Bármilyen téves átállítás esetén — ha az ENTER-t még nem nyomtuk meg — visszaállítható az előző állapot az ESC billentyű leütésével. Azoknak a regisztereknek, melyek tartalma binárisan is rendelkezésre áll, az értéke akár bitenként is megváltoztatható egy egérekattintással. Piros színű azoknak a regisztereknek a háttere, melyeknek értéke változott az előző ablakfrissítés óta. Kék keret jelzi azt a regisztert, amelyikre az FSR (File Select Register) mutat. Fehér háttere van annak a banknak, mely ki van jelölve az FSR felső három bitje által (lásd az 1.2.6 szakaszt).

Fontos tudni, hogy az RA, RB és RC regiszterek az egyes port-lábak logikai állapotát mutatják (mintha olvasnánk őket) és nem pedig a port adatregiszterekbe (latch-ekbe) beírt adatot.

Ha programunkban használtuk a WATCH direktívát, akkor megjelenik egy ugyanilyen nevű mező is. Ebben a felhasználó által fontosnak ítélt regiszterek tartalma külön figyelhető, sőt a kijelzés formátumát (decimális, hexadecimális, bináris, karakteres) is a felhasználó adja meg. A numerikus értékek a fent leírtak

szerint módosíthatók, hexa, bináris vagy decimális alakban, a kijelzés formátumától függetlenül. Ha például egy értéket hexadecimális alakban akarunk módosítani, a beírandó szám elé üssünk egy \$ jelet., ha bináris alakban, akkor % jelet, ha nem ütünk előtte semmit, akkor a beadott értéket decimális számnak veszi. A karakteres kijelzésű értékek újabb karakterek beadásával írhatók át. A WATCH direktíva részletes ismertetése a [3] szakirodalomban található.

Programozáskor az SX konfigurálható úgy is, hogy nyolcnál kevesebb (4, 2, vagy 1) bankot érhessünk csak el. Ilyenkor — mivel az elérhető regiszterek száma is kevesebb — nincs szükség az FSR regiszter minden bitjére, a nem használt bitek olvasáskor 1 értékűnek látszanak még akkor is, ha előzőleg 0-ba állítottuk őket. A mi példánkban most négy bank van, ezért az FSR legfelső (7-es) bitje mindig 1 értékű. A **2.1 ábrán** az FSR a BANK-1 15H relatív című regiszterére mutat, melynek abszolút címe 75H. Mivel azonban az FSR 7-es bitje állandóan 1, így a mezőben B5H látható. (Egy `MOV FSR, #75` utasítás is ezt az eredményt hozná.) Ennek megszokása nem könnyű, ezért a regiszter bankok felett megtalálható az az ofszet, amiből leolvashatjuk, hogy az adott bankhoz milyen FSR érték tartozik. (Ha a példánkban nyolc bank lenne, akkor a BANK-1 felett már "7x" állna, és a `MOV FSR, #75` utasítás hatására az FSR-ben is a 75H érték jelenne meg.)

Megírt programunk futása az alsó, CODE feliratú mezőben követhető. Azt a programsort, melynek a végrehajtása a következő, kék kiemelés jelzi. A felső mezőben középen a gépi utasításlista található. Az általunk definiált szimbólumok (cimkék, regiszternevek) itt már nem jelennek meg, és a makró utasítások is "kibontott" formában láthatók.

Lehetőségünk van a programban egy darab töréspont elhelyezésére is, egyszerűen csak arra a programsorra kell kattintanunk az egér gombjával, ahova a töréspontot szeretnénk. Az előző töréspont automatikusan törlődik. Azt az programsort, melyhez töréspont van rendelve, piros kiemelés jelzi. Ha a program végrehajtását a soron következő utasítás helyett egy másik utasítással szeretnénk folytatni, egyszerűen megtehetjük kívánt utasítás sorra történő dupla kattintással. Ilyenkor a programszámláló (PC) értéke is automatikusan átállítódik.

Az ablak felső részén található két jelzőbit is INT és SKIP felirattal. A SKIP feliratú akkor válik aktívvá, ha egy TEST-SKIP jellegű utasítás a soron következő utasítást át fogja ugrani. A jelzőbitet egy egérekattintással mi is átválthatjuk, ez azt eredményezi, a következőnek végrehajtandó utasítást a processzor átugorja. Újbóli kattintással viszont nem újabb átugrást végez, hanem visszafele lép az előzőre. Az INT feliratú jelzőbit akkor válik aktívvá, ha a processzor megszakítást generál, és mindaddig úgy marad, amíg egy RETI vagy RETIW utasítás le nem zárja a megszakítást. Ennek a jelzőbitnek a kézi álltása nem célszerű, mivel ha aktivizáltuk, akkor a megszakítások nem fognak érvényre jutni az egyszintű megszakítási rendszer miatt (a processzor azt hiszi, hogy megszakításkiszolgáló rutin fut).

Hibakereső/nyomkövető módban lehetőségünk van a programot akár teljes sebességen futtatni, de akár utasításonként lépésről-lépésre is, megfigyelve minden apró változást. Ez azért lehetséges, mert lépésről-lépésre történő programvégrehajtásnál a program az ablak tartalmát minden utasítás után frissíti.

- 1. A **HOP** gombbal a program végrehajt egy Parallax utasítást (a makró utasításokat egynek veszi), majd frissíti az ablakot, és megáll.*
- 2. A **JOG** gombbal a program folyamatosan hajtja végre a Prallax utasításokat, amíg a **STOP** gombot meg nem nyomjuk, vagy törésponthoz nem ér. Minden utasítás után frissíti az ablakot.*
- 3. A **STEP** gombbal a program végrehajt egy gépi (egyszavas) utasítást, majd frissíti az ablakot, és megáll. Ha például egy makró utasítás három egyszavas utasításból áll, akkor a **STEP** gombot háromszor kell aktiválni az adott makró teljes végrehajtásához.*
- 4. A **WALK** gombbal a program folyamatosan hajtja végre a gépi utasításokat, amíg a **STOP** gombot meg nem nyomjuk, vagy törésponthoz nem ér. Minden utasítás után frissíti az ablakot.*
- 5. A **RUN** gombbal a program teljes sebességgel hajtja végre az utasításokat, amíg a **STOP** gombot meg nem nyomjuk, vagy törésponthoz nem ér. Az ablak csak a leálláskor frissítődik.*
- 6. A **POLL** gombnak két funkciója van. Teljes sebességű programvégrehajtásnál (**RUN**), ha megnyomjuk, az ablak tartalma frissítődik. A másik, ha a programot a **POLL** gombbal indítjuk, akkor elindul a teljes sebességű programvégrehajtás, az ablak nem frissítődik eközben, csak akkor, ha egy törésponthoz ér. Ilyenkor viszont nem áll le, hanem az ablak frissítése után teljes sebességgel fut tovább.*

Megszakításkezelő rutinokban a nyomkövetés kicsit körülményesebb. Ha a programot lépésről-lépésre hajtjuk végre (**HOP, JOG, STEP, WALK**), akkor az **RTCC** megszakítás csak bizonyos esetekben, a **MIWU** megszakítás pedig egyáltalán nem fog érvényre jutni. Ha ilyen programrészben akarunk hibát keresni, akkor tegyünk egy töréspontot a megszakításkezelő rutin kívánt részére, és a **RUN** vagy a **POLL** gombbal indítsuk a programvégrehajtást.

Ahhoz, hogy a hibakereső/nyomkövető funkció helyesen működjön, a következő feltételeknek kell teljesülni:

- 1. Semmilyen oszcillátor, rezonátor, kristály vagy RC hálózat ne csatlakozzon az SX OSC1-OSC2 lábaira. Továbbá, ha 50 Mhz-től eltérő*

frekvenciát akarunk, akkor a programunknak a `FREQ` direktívát is tartalmaznia kell.

- 2. A programnak tartalmaznia kell a `RESET` direktívát.*
- 3. A watchdog áramkört ki kell kapcsolni.*
- 4. Két szó szabad helyet kell hagyni az első programtár-lapon, tetszőleges helyen, illetve 136 szó szabad helyet a legutolsó programtár-lap végén (2k-szó tárméret esetén ez 777..7FEH).*

Az `SX-KEY.EXE` program konfigurálásának és egyéb szolgáltatásainak a leírását a [3] szakirodalom tartalmazza.

2.3 Virtuális perifériák

A virtuális perifériák programmodulok, amelyek olyan feladatokat látnak el, amiket egyébként csak beintegrált perifériák, vagy külső hardverelemek segítségével lehetne megvalósítani. Az SX mikrovezérlőnek — szédületes sebessége miatt — ez az egyik fő erőssége. Már jelenleg is számos virtuális periféria minta tölthető le a Scenix honlapjáról (www.scenix.com), ezeket a modulokat szabadon adaptálhatjuk saját programjainkba. A virtuális perifériák nagy többsége semmilyen külső hardvert nem igényel.

2.3.1 A virtuális perifériák típusai

Virtuális perifériát három módon valósíthatunk meg:

- 1. Programba ágyazott módon*
- 2. Szubrutin formátumban*
- 3. Megszakításkezelő rutinba ágyazottan*

Az SX-nél leggyakrabban az utolsó két változatot alkalmazzuk a modularitás érdekében. Általános szabály azonban, hogy egy virtuális periféria egyszerű, gyors, rövid és áttekinthető, az egyes virtuális perifériák pedig egymástól függetlenek legyenek.

Programba ágyazott virtuális perifériák

Az adott perifériefunkciót megvalósító kódot egyszerűen beillesztjük a programunkba, arra a helyre, ahol a végrehajtása szükséges. Akkor célszerű az alkalmazása, ha a virtuális perifériát megvalósító kód rövid, és csak egy helyen kerül alkalmazásra a programban.

Szubrutin formájú virtuális perifériák

Az adott perifériefunkciót megvalósító kódot szubrutin formájában írjuk meg, ügyelve a szubrutinok alkalmazásának szabályaira (lásd az **1.2.3 szakaszt**). Az ilyen virtuális perifériák már bárhol és bárhányszor tetszőlegesen elérhetők a programunkból. Ügyelni kell azonban arra, hogy a túl sok szubrutin egybeágyazás könnyen a verem túltöltődését okozhatja.

Megszakításkezelő rutinba ágyazott virtuális perifériák

Ez a virtuális perifériák megvalósításának a legérdekesebb és leghatékonyabb módja. A főprogramot nem kell semilyen CALL vagy JMP művelettel megszakítani, a virtuális perifériefunkciót megvalósító kódrészek a megszakításkiszolgáló rutinba vannak beillesztve. Mivel az SX-nek két megszakításforrása van (**RTCC** és **MIWU**, lásd az **1.5 szakaszt**), a virtuális perifériákat is erre a kettőre alapozzuk. Az **RTCC** alkalmazása gyakoribb, mert jobban kézben tartható a periodikus viselkedés miatt. Viszont aszinkron külső eseményekhez rendelt virtuális perifériák csak a **MIWU**-val készíthetők.

Az **RTCC** túlsordulásából eredő megszakítás szabályos időközönként következik be, ennek intervalluma szoftveresen állítható a **RETIW** utasítás alkalmazásával. Ha erre a lehetőségre nincs szükség, akkor **RETI** használandó a megszakításból való visszatérésre. A **RETIW** alkalmazása esetén a megszakítási periódus ideje függ az oszcillátor frekvenciától, az előosztó értékétől (**OPTION** regiszter), a processzor üzemmódtól (turbó vagy kompatibilis) illetve attól a számtól, aminek a megszakítási rutin végén betöltjük a kettes komplementjét a **W**-be (**RETIW-érték**, lásd az **1.5.4 szakaszt**).

$$\text{megszakítási periódus [sec]} = \frac{\text{mód} * \text{előosztó} * \text{RETIW érték}}{\text{oszcillátor frekvencia}}$$

Az előosztó értékét az **OPTION** regiszterben állíthatjuk be, 'mód' értéke turbó üzemmód esetén egy, kompatibilis üzemmód esetén négy.

Vannak virtuális perifériák amelyek futási ideje állandó, és vannak amelyeknek esetről-esetre változó (mert ugró műveleteket is tartalmaznak).

Természetesen az az ideális, ha minél több elkészített virtuális perifériánk állandó futási idejű, de ezt nem könnyű megvalósítani.

Van néhány virtuális periféria, mely kényes arra, hogy precíz, szabályos időközönként kerüljön végrehajtásra. Ezeket általában a megszakításkiszolgáló rutin elejére tesszük, és semmiképpen sem előzheti meg őket változó futási idejű virtuális periféria-rutin.

Vannak virtuális perifériák, melyek aktiválása csak viszonylag ritkán (pl. ms-onként) szükséges. Ha mondjuk négy ilyenünk van, akkor jó megoldás lehet, hogyha a megszakításkiszolgáló rutinba időzítő programrészt építünk be, 250 μ s (1ms/4) ciklusidővel, és minden ciklusban csak egy virtuális perifériát aktiválunk a négy közül, így végeredményben mind a négy 1 ms-onként lesz végrehajtva.

Fontos adat lehet az egyes virtuális perifériák processzor-teljesítmény igénye:

$$\text{processzortelj. igény [\%]} = \frac{\text{a megszakítás teljes végrehajtási ideje} + 3}{\text{RETIW érték} * \text{előosztó érték}}$$

Ha **RTCC** és **MIWU** megszakítást együttesen akarunk alkalmazni, akkor az **1.5 szakaszban** leírtakra kell odafigyelnünk az egyszintű megszakítási rendszer miatt.

2.3.2 Egy egyszerű virtuális periféria-minta

A most következő példa minden bizonnyal a világ egyik legegyszerűbb virtuális perifériája, de bemutatásának nem ez az egyetlen oka, ugyanis segítségével egy hatalmas félreértést is tisztázhatunk. Előljáróban két alapfogalom a [9] szakirodalom felhasználásával:

Impulzusszélesség-moduláció, Pulse width Modulation (PWM): Olyan impulzusidő-moduláció, amelynél az impulzusok szélességét vezéreljük vagy szabályozzuk. Az impulzus homlokának vagy hátának (esetleg mindkettőnek) egyszerre változik az időbeni helyzete a modulálójel pillanatnyi értékétől függően. Az impulzus ismétlődési frekvenciája (ill. periódusideje), illetve az impulzusamplitúdó állandó.

Impulzusfrekvencia-moduláció, Pulse Frequency Modulation (PFM): Olyan impulzusidő-moduláció, ahol az impulzusok pillanatnyi frekvenciáját, vagyis az időegység alatti impulzusok számát változtatjuk a modulálójel függvényében. Három alapesete használatos: (1) állandó bekapcsolási / változó kikapcsolási időtartam, (2) változó bekapcsolási / állandó kikapcsolási időtartam, (3) mindkét időtartam változó. Leggyakrabban az első esetet alkalmazzák. Hátránya a PWM-hez

képest, hogy körülményesebb mellé szűrőt tervezni a változó frekvencia miatt, mivel a frekvenciával a hullámosság mértéke is változni fog.

Most következzen az — egyelőre még név nélküli — virtuális periféria forráskódja. A kódot a megszakításkiszolgáló rutinba kell beilleszteni, hogy — az RTCC túlsordulás következtében — rendszeres időközönként meghívásra kerüljön.

```
clrb rc.0
add accumulator,value
snb c
setb rc.0
```

Ennyi. Próbáljuk meg kitalálni, mit is csinálhat ez virtuális periféria! Először is törli az RC port 0-ás kimenetét. Van egy akkumlátor (accumulator) nevű regiszterünk, ehhez minden ciklusban hozzáadjuk az érték (value) nevű regiszter tartalmát. Ha az akkumlátor túlsordul, akkor 1-be állítja az RC port 0-ás kimenetét. Tehát ha az akkumlátor túlsordul, akkor egy megszakítási periódus idő erejéig 1-be állítódik az a bizonyos portláb, ami a következő ciklus elején azonnal törlődik is. Legközelebb akkor állítódik ismét 1-be, mikor az akkumlátor ismét túlsordul. Mitől is függ az akkumlátor túlsordulásának időtartama? Attól, hogy ciklusonként mennyivel növeljük az értékét, vagyis a value regiszter tartalmától.

Ezek alapján belátható, hogy ez a virtuális periféria impulzusfrekvencia-modulációt (PFM) valósít meg, és a kitöltési tényezőt a value regiszter értéke szabja meg. Tehát nem helytálló több szakirodalom azon állítása, hogy ez impulzusszélesség-moduláció (PWM) lenne.

3. Beágyazott rendszerek soros kommunikációja

3.1 A soros kommunikáció jelentősége

A mikrovezérlőt az különbözteti meg a mikroprocesszoroktól, hogy az integrált áramköri tokba a központi egységen (CPU) kívül még több-kevesebb funkcionális egységet, perifériát is beintegráltak. A mikrovezérlővel való fejlesztések során hamar kiderül, hogy a "több-kevesebb"-ből a "több" is gyakran kevesebb, mint amennyire szükség lenne, ilyenkor külső eszközök alkalmazására kényszerülünk. Tételezzük fel, hogy munkánk során eljutunk egy olyan ponthoz, amikor egy mikrokontrollerbe beépített EEPROM memória kevésnek bizonyul, külső memóriát kell alkalmaznunk. Ha ezt a külső tárat hagyományos párhuzamos adatelérésűnek választjuk, akkor a mikrokontrollernél értékes portlábakról kell lemondanunk. Például: egy 1K×8 szervezésű párhuzamos EEPROM 18 (8 adatvezeték + 10 címvezeték) portlábát fog lekötni. A lefoglalt portlábak számának csökkentése érdekében szóba jöhet, hogy a cím alsó 8 bitjét — időmultiplexelve — azonos porton kezeljük az adatvonalakkal. Egy 16 I/O lábbal rendelkező mikrovezérlő esetében még így is csak öt láb marad a tényleges feladatmegoldásra (8 láb az adatnak + a cím alsó részének, két láb a maradék címvezetéknek, és további egy lábat elvisz az cím-latch vezérlő jel).

Ugyanez a feladat megoldható úgy is, hogy a 16-ból 14 I/O láb maradjon szabadon akkor, ha egy soros EEPROM-ot alkalmazunk külső tárként.

Nem szabad elhallgatni a soros adatátvitel egyik legnagyobb hátrányát: a **bitsoros jelkezelés miatt az adatátviteli sebesség drasztikusan csökken**. Hozzá kell azonban tenni, hogy a gyakorlati irányítási feladatok jelentős részében ez még elviselhető. Sőt, a mikroelektronika fejlődése olyan irányba mutat, hogy ez a hátrány rövid időn belül el fog tűnni.

A vezérelt, irányított berendezések bonyolultságának növekedésével együtt nő a beépített elektronika bonyolultsága is. Sok esetben már nem is elegendő egyetlen mikroprocesszor vagy mikrovezérlő a kezelésükre. Az ilyenkor szóba jövő **multiprocesszoros rendszerek** kialakítására is egyszerű, gazdaságos lehetőséget kínál a soros adatkezelésű megoldás, különösen miután **néhány nagymértékben elterjedt megoldás szabványossá is vált**.

3.2 RS-232 aszinkron soros adatátvitel

A számítástechnika első szárnypróbálgatásaival egyidős az egyik legkorábbi megoldás, az RS-232 aszinkron soros jelátvitel. A személyi számítógépek elterjedésével a benne található RS-232 periféria (vagy egyszerűen csak soros port) szabványos illesztő felületté vált, ezért a soros vonalat széles körben — eredeti funkcióján túlmenően — kezdték különböző perifériális eszközök illesztésére felhasználni.

A szabványleírásban a számítógép és a terminál hivatalos neve:

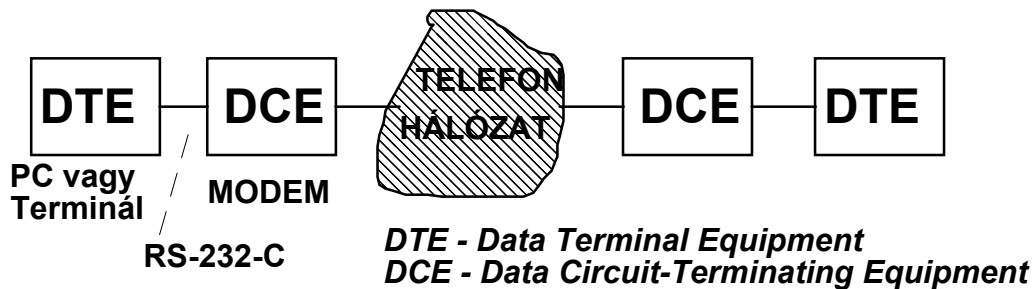
adatvég-berendezés — **DTE (Data Terminal Equipment)**

a kapcsolódó modemé:

adatáramköri-végberendezés **DCE (Data Circuit-Terminating Equipment)**,

és a köztük zajló kommunikáció az RS-232 soros vonalon folyik.

Általánosan fogalmazva egy DCE végzi a kommunikációs közeghez történő fizikai illesztést, azaz a kétállapotú bináris jeleket átalakítja a közegben átvihető fizikai jelekké. A legtöbb gyakorlati esetben a DTE egy terminál, vagy egy számítógép, míg a DCE az analóg telefonhálózathoz kapcsolódó modem.

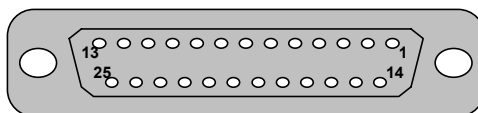


3.A ábra: DTE és DCE egységek kapcsolata

3.2.1 RS-232C szabvány

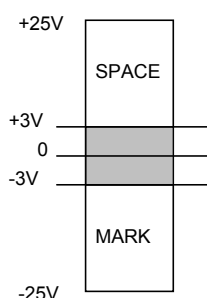
A szabvány megalkotója az Electronic Industries Association elnevezésű, elektronikai gyártókat tömörítő szakmai szervezet, így az EIA RS-232-C a pontos hivatkozás. Ennek nemzetközi változata a CCITT V.24. ajánlása, amely csak néhány ritkán használt áramkörben tér el. Az ajánlás (Recommended Standard 232 C) az eredeti ajánlás harmadik („C”) változata.

A DTE-DCE egységeket összekötő vezetékrendszer mechanikus csatlakozóját is definiálták: 25 pólusú csatlakozó (szokták DB-25-nek is nevezni). Két, egymásba dugható csatlakozó közül a dugós rész a DTE-n, a hüvelyes részt a DCE-n helyezkedik el.



3.B ábra: Szabványos 25 pontos soros csatlakozó

A villamos specifikáció szerint a $-3V$ -nál kisebb feszültség a vonalon a bináris 1-et (MARK), míg a $+3V$ -nál nagyobb feszültség bináris 0 -át (SPACE) jelent. A legfeljebb 15 méter hosszú kábeleken 20 kbit/s-os maximális adatátviteli sebesség a megengedett. A legtöbb gyakorlati esetben (pl. a számítógépek soros vonalánál) a feszültség $\pm 12V$.



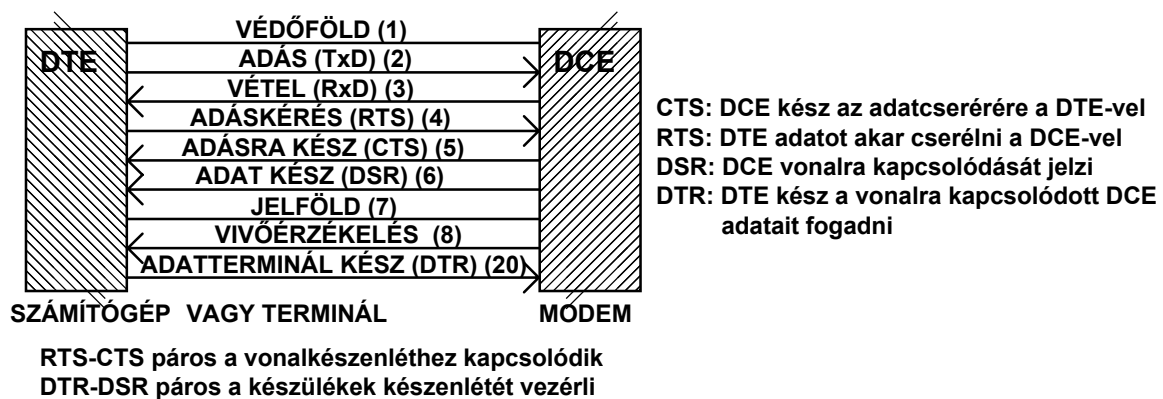
3.C ábra: RS232 jelszintek

A funkcionális előírás a 25 ponthoz tartozó vonalakat megjelöli, és leírja azok jelentését. A **3.4 ábrán** annak a 9 vonalnak a funkciója látható (a hozzá tartozó kivezetés számmal), amelyeket majdnem mindig megvalósítanak.

1. Amikor a számítógépet vagy a terminált bekapcsolják, az aktiválja (MARK-ba állítja) az Adatterminál kész (Data Terminal Ready) jelet (20).
2. Amikor a modemet kapcsolják be, akkor a modem az Adat kész jelet (Data Set Ready) (6) aktiválja.
3. Ha a modem vivőjelet érzékel a telefonvonalon, akkor a Vivőérzékelés (Carrier Detect) jelet (8) aktiválja.
4. Az Adáskérés (Request to Send) (4) jelzi, hogy a terminál adatot akar küldeni.

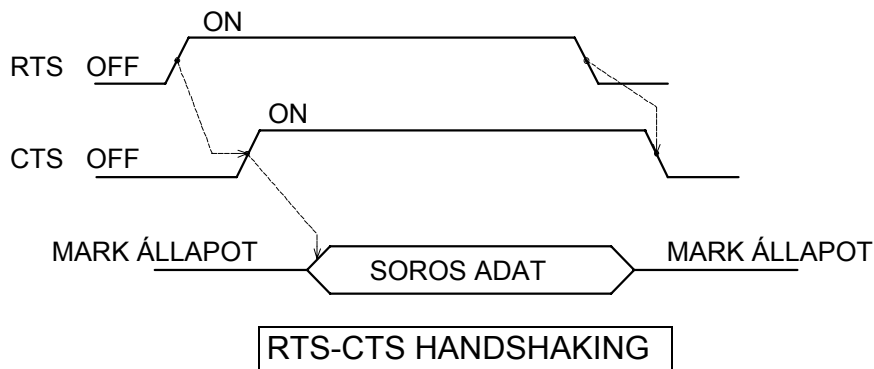
5. Az Adásra kész (Clear to Send) (5) azt jelenti, hogy a modem felkészült az adatok fogadására.
6. Az adatok adása az Adás (Transmit) vonalon (2), vétele a Vétel (Receive) vonalon (3) történik.

A többi, fel nem tüntetett áramkör a gyakorlatban alig használt funkciókkal rendelkezik: adatátviteli sebesség kiválasztása, modem tesztelése, adatok ütemezése, csengető jelek érzékelése, adatok másodlagos csatornán való fordított irányú küldése.



3.D ábra: DTE-DCE összekötő vezetékek

Az eljárásinterfész az a protokoll, amely az események érvényes sorrendjét határozza meg. A protokoll akció-reakció esemény-párokon alapszik. Amikor egy terminál kiadja pl. az Adáskérés jelet, a modem egy Adásra kész jellel válaszol, ha képes fogadni az adatokat. Ugyanilyen jellegű akció-reakció párok léteznek a többi áramkör esetén is.



3.E ábra: RTS-CTS kézfogásos kapcsolat

Ahhoz hogy a soros adatátvitel során az **ADÓ**-ról érkező biteket a **VEVŐ** egyértelműen azonosítani tudja, szükséges, hogy azonosan értelmezzék a jeleket, azaz pl. egy bájt ötödik bitjét kiküldve, azt a **VEVŐ** is annak tekintse.



3.F ábra: Az aszinkron soros adatátvitel elve

Az aszinkron soros átvitelnél a bits csoportos átviteli mód biztosítja az **ADÓ** és a **VEVŐ** szinkronizmusát. Természetesen ehhez a járulékos információhoz járulékos biteket is fel kell használni. Ezek a **START** és a **STOP** bitek. Ezen biteket szokták keretező (framing) biteknek is nevezni, mivel a tényleges információt "keretbe foglalják". A **START** bit jelzi, hogy utána következnek a tényleges információt hordozó adatbitek, míg a **STOP** bit(ek) ezek végét jelzi.

A soros protokoll szerint, ha a soros vonalon nem folyik információátvitel, a vonal állapota aktív (**MARK**) szintű. Az adatátvitel kezdetekor az **ADÓ** a vonalat egy bit átvitelének idejéig alacsony (**SPACE**) szintre állítja (*Vigyázzunk! Ez a pozitív feszültség!*) (**START** bit), majd utána történik meg az adatbitek átvitele. Az átvitt adatbitekből álló bits csoport végére az **ADÓ** **STOP** bit(ek)ből álló aktív (**MARK**) szintű jelet helyez el. A **VEVŐ** az adás kezdetéről a vonal **MARK-SPACE** állapotváltozásából szerez tudomást.

Ezután mindig egy bit átvitelének idejéig várakozva a **VEVŐ** az adatbiteket veszi. A **STOP** bitek érkezése után már figyelheti a vonalon ismét megjelenő állapotváltozást, ami a következő bits csoport adásának kezdetét jelöli.

Fontos kérdés a vonalon időegység alatt átvitt információ mennyisége, amit bit/s-ben mérünk. Tipikus, szabványosan használt értékeit a következő táblázat tartalmazza.

bit/s	Egy bit átvitelének ideje [msec]
150	6.6666
300	3.3333
600	1.6666
1200	0.8333
4800	0.2083
9600	0.1042
19200	0.0521
38400	0.0261

Az adatátvitel során az esetleges átviteli hibák felderítését megkísérelhetjük oly módon, hogy az átviendő adatbit-csoportot egy paritás bittel egészítjük ki úgy, hogy az így kiegészített adatcsoportban lévő 1 értékű bitek száma páros (páros paritás), vagy páratlan (páratlan paritás) legyen. Ilyen módon az **ADÓ** oldalán mindig biztosítható, hogy az 1-es értékű bitek száma mindig páros/páratlan legyen, és a **VEVŐ** oldalon az egy (ill. páratlan számú) bit változása miatti hiba felderíthető.

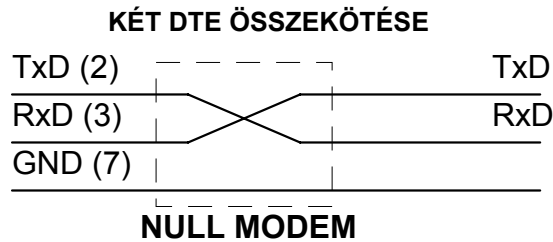
Statisztikailag igazolható, hogyha kicsi a "hibacsomósodások" valószínűsége, akkor a paritásellenőrzés az adatátvitelt sok esetben megfelelően megbízhatóvá teszi. Jó minőségű, zajmentes összeköttetésnél a paritásbitre pedig nincs is szükség.

Az előzőek alapján a soros adatátviteli protokoll konkrét kialakításánál a következőket kell rögzíteni:

- 1. **Adatbitek száma:** a gyakorlatban 5, 6, 7 vagy 8 bit.*
- 2. **Paritásbit:** használunk paritásbitet vagy nem, és ha igen, páros vagy páratlan paritást alkalmazunk.*
- 3. **Stop bitek száma:** ez a soros vonalnak a bitsorozat átvitele utáni garantált logikai 1 állapotának az idejét határozza meg az egy bit átviteléhez szükséges idővel kifejezve. Hossza 1, 1.5, vagy 2 bit lehet. A legrövidebb az egy bit, és ez biztosítja, hogy a **VEVŐ** a következő bitsorozat vételéhez szükséges szinkronizáló **START** bit indító élének érzékelésére felkészüljön. Két stop bit használata akkor előnyös, ha valamilyen okból szükséges a vett adatbitek azonnali feldolgozása és az ehhez szükséges hosszabb idő.*
- 4. **Adatátviteli sebesség (bit/s):** Igen fontos adat, mert ez határozza meg alapvetően az **ADÓ** és a **VEVŐ** szinkronizmusát.*

Mivel a soros adatátvitelt széles körben használják, ezért, megvalósítására céláramköröket fejlesztettek ki. Ezeknél az **ADÓ** oldalán csupán az adatbit-csoportot kell párhuzamosan a bemenetekre adni, az áramkör elvégzi a sorossá alakítást, a paritás, START és STOP bitekkel való kiegészítést, valamint az átvitelt. A vevőoldalon a vett soros adatokból vevőáramkör képezi a bitsorozatot. Ezek az áramkörök programozhatóak, azaz vezérlőkódokkal megadhatók az átvitel paraméterei és a soros adatátviteli protokoll. Magát a párhuzamos adatok sorossá alakítását, illetve a soros adatok visszaalakítását shiftregiszterek felhasználásával hardver úton lehet elvégezni. Mivel ezek az áramkörök TTL jelszintekkel működnek, ezért be és kimenetükön 0 és 5 V-os jeleket várnak illetve adnak. Ezért ezeket a soros periféria áramköröket mindig ki kell egészíteni egy olyan szintátalakító áramkörrel, amely a TTL szintjeiket a szabványos RS232 jelszintekké oda- és visszaalakítja a 0V → +12V; 5V → -12V szabályok szerint.

Mivel majdnem minden számítógépnek van soros vonala, gyakran előfordul, hogy két számítógépet RS-232-C soros vonalon keresztül kötnék össze. Mivel nem DTE-DCE típusú az összeköttetés, ezért a megoldás egy **null-modem** -nek nevezett „eszköz” (hiszen csak egy keresztbe kötés), amely az egyik gép adási vonalát a másik gép vételi vonalával köti össze. A null modem megoldás lényegében két DTE típusú eszközt köt össze.



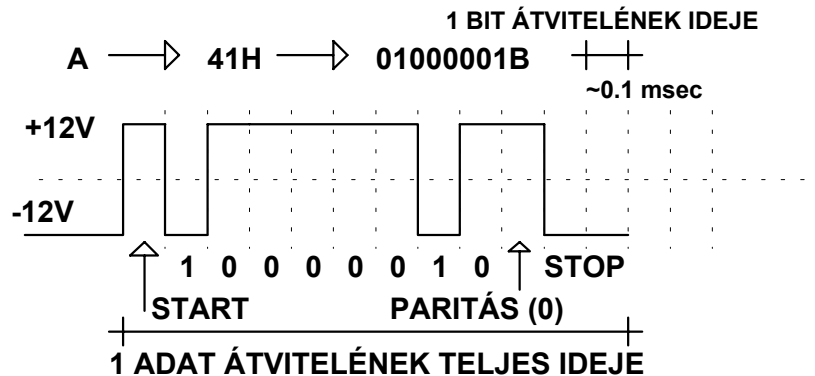
3.G ábra: Null-modem: Két DTE összekötése

A legegyszerűbb esetben ez elegendő, ha azonban a modemvezérlő vonalakat is használnunk kell, akkor hasonló módon néhány más vonal keresztbekötését is el kell végezni.

A tényleges adatátviteli sebesség számítására nézzünk egy konkrét példát: Ha például az adatátviteli sebesség 9600 bit/s, és 8 bites adatokat (bájtokat) viszünk át páros paritásbittel kiegészítve, 2 STOP bittel a végén, akkor például másodpercenként:

$9600 / (1 \text{ START} + 8 \text{ ADAT} + 1 \text{ PARITÁS} + 2 \text{ STOP bit}) = 9600 / 12 = 800$ adat (bájt) kerül átvitelre.

Ha például az „A” karaktert visszük át (ASCII kódja 41H), akkor a soros vonalon a következő jelformát láthatnánk:



3.H ábra: Egy soros jelalak

A karakter biteit fordított sorrendben visszük át (LSB az első!), a páros paritásbit ennél a karakternél 0, mert eleve páros (kettő) 1 értékű bitet tartalmazott.

3.2.2 RS-232 kommunikációt támogató rutinkönyvtár az SX-hez

A rutinkönyvtár forráskódja a mellékelt mágneslemez-mellékleten található: **SX_UART.SRC** néven. A könyvtár tartalmazza a teljes duplex RS-232 kommunikációt megvalósító virtuális periféria kódját, az ehhez szükséges port inicializáló rutinokat, és egyéb funkcionális szubrutinokat, melyek a mindennapi soros kommunikációs feladatokat valósítják meg. Ezek közül a fontosabbak:

Kiviteli rutinok

PUT_BYTE - Egy bájt kiküldése a soros portra

Bemenet: W (Nem változik!)

Kimenet: -

PUT_STRING - Szöveg kiírása a soros portra

Bemenet: DAT_HI, DAT_LO (szöveg első karakterére mutat),
a szöveg bárhol elhelyezkedhet a programtárban

Termináló karakter: 00H vagy 03H (ETX - End Of Text)

PUT_HEX - Egy bájt kiírása hexadecimális alakban

Bemenet: W (Nem változik!)

Kimenet: -

PUT_WORD - Egy 16 bites kiírása hexadecimális alakban

Bemenet: DAT_HI, DAT_LO (Nem változnak!!)

Kimenet: -

PUT_BCD - Egy bájt kiírása BCD (0..99) alakban

Bemenet: W

Kimenet: -

PUT_BIN - Egy bájt kiírása bináris alakban

Bemenet: W

Kimenet: -

Beviteli rutinok

GET_BYTE - Egy bájt vétele a soros portról

Bemenet: -

Kimenet: W

GET_CHAR - Egy bájt vétele a soros portról ECHO-val

Bemenet: -

Kimenet: W

SCAN - Nyomtak-e billentyűt?

HA IGEN: CY=1 & W=KOD , HA NEM: CY=0, W nem változik

GET_HEX - 2 digites hexa érték beolvasása a soros portról ECHO-val

A beírt szám RETPURN vagy SPACE karakterrel terminálható

SPACE esetén CY=0, RETPURN esetén CY=1

Bemenet: -

Kimenet: W, C

GET_WORD - 4 digites hexa érték beolvasása a soros portról ECHO-val

A beírt szám RETPURN vagy SPACE karakterrel terminálható

SPACE esetén CY=0, RETPURN esetén CY=1

Bemenet: -

Kimenet: DAT_HI, DAT_LO, C

GET_BCD - 2 digites BCD (0..99) érték beolvasása a portról ECHO-val

A beírt szám RETPURN vagy SPACE karakterrel terminálható

SPACE esetén CY=0, RETPURN esetén CY=1

Bemenet: -

Kimenet: W, C

Egyéb rutinok

HEXOK - W-ben HEXA karakter van ?

Bemenet: W (Nem változik!)

Kimenet: C=1 (W-ben hexa karakter van), C=0 (W = érvénytelen kód)

ASCHEX - Karaktert hexára (0..F) konvertál

Bemenet: W

Kimenet: W

HEXASC - Hexát (0..F) karakterre konvertál

Bemenet: W

Kimenet: W

BCDBIN - BCD-t (0..99)binárisra konvertál

Bemenet: W

Kimenet: W

BINBCD - Binárisat BCD-re (0..99) konvertál

Bemenet: W

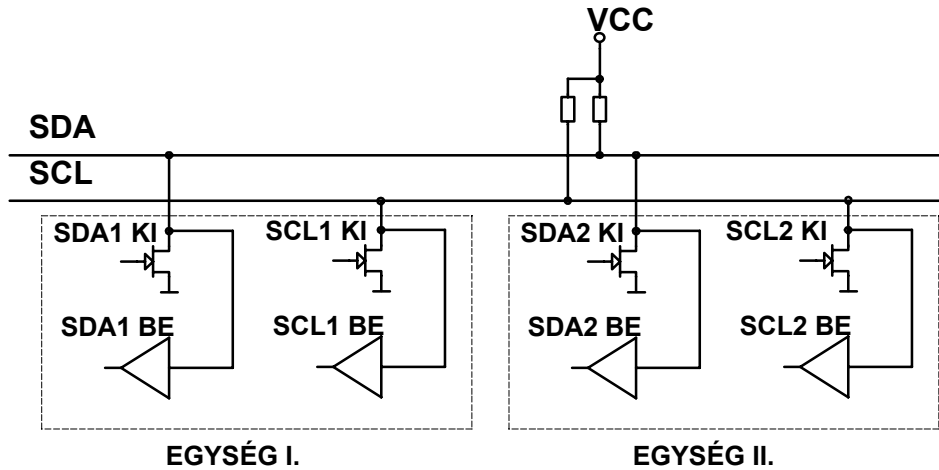
Kimenet: W

3.3 Az I²C busz

Az I²C, Inter IC azaz IC-k közötti busz. Az I²C busz nagybonyolultságú integrált áramkörök közötti soros információcserét biztosító, azt fizikailag három vezetékkel megvalósító sínrendszer. Az átviteli félduplex módon történik, sebessége kb. 100-400 kbit/s-ig növelhető. Az ilyen buszt tartalmazó nagybonyolultságú integrált áramkörök egymással könnyen, kevés vezetékkel tudnak sorosan kommunikálni. A kommunikáció kétirányú adatvonalon (SDA=Serial Data) keresztül történik, és egy külön órajel (SCL=Serial Clock) szinkronizálja az adatvezetéken az adatokat.

A busz elvi felépítése a **3.9 ábrán** látható.

A tranzistorok kikapcsolt állapotában a felhúzó ellenállás miatt a vonalak magas állapotban vannak. Ez az alaphelyzet. Ha bármelyik tranzisztort bekapcsoljuk, az a vezetéket a földre kapcsolja, így nulla állapotú. Ezt a megoldást az elektronikában huzalozott-ÉS (wired-AND) kapcsolatnak hívják.



3.I ábra: Az I²C kommunikáció elve

A vezérlési elvből következik, hogy mindig csak egy egység vezérelheti az adott vezetékét, a többi egység a tranzisztort nem kapcsolhatja be. Az eddig tárgyalt adás mellett minden egység képes a vonalon lévő adatokat is venni egy erősítőn keresztül.

Még egy érdekes megállapítás: egy egység el tudja dönteni hogy a vezeték más nem vezérli-e. Ha ugyanis az adatokat a vezetékre kapcsolja, a saját vevőerősítőjén ugyanazt az adatot kell venni, mint amit kiküldött. Ha ez nem teljesül, valamelyik másik egység is „piszkálja” a vonalat, azaz a buszfoglaltság azonosítható.

Az előzőekből következően minden egység lehet **Adó** ill **Vevő**. Ezen felül megkülönböztetünk **Master** és **Slave** eszközöket. Így összesen két funkció és két szerep különböztethető meg:

A funkciók:

TRX = Transmitter (adó): Az egység, amelyik adatot küld a buszra.

RCV = Receiver (vevő): Az egység, amelyik adatot fogad a buszról.

A szerepek:

MST = Master (mester): Az egység, amelyik kezdeményezi az átvitelt, az átvitelhez az órajelet generálja, és be is fejezi az átvitelt.

SLV = Slave (szolga): A mester által megcímzett egység.

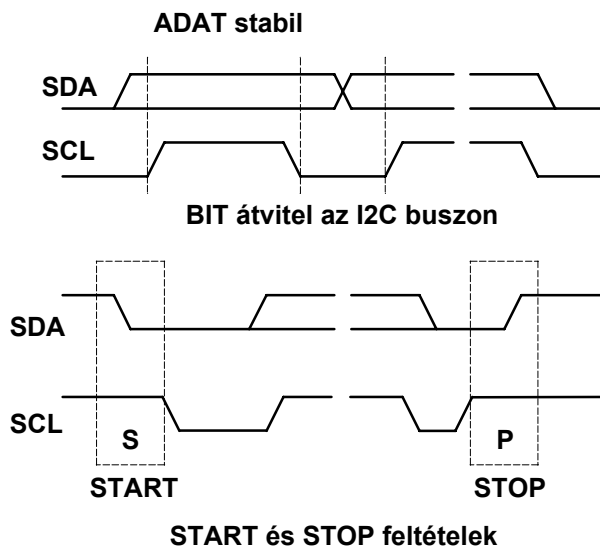
Egy mikrokontroller hardveres I²C egysége általában mindegyik szerepre és funkcióra képes. Természetesen, ha ez az egység vezérli a perifériákat (és a gyakorlatban ez a leggyakoribb eset), akkor szerepe: mester és a perifériák a szolgák.

A busz fontos jellemzője, hogy **multi-master** kialakítású. Ez azt jelenti, hogy buszra kapcsolódó eszközök közül nem csak egy, hanem több is átveheti az átvitel vezérlését.

Ezzel kapcsolatos fontos tulajdonság az **arbitration**, vagy döntés. Ez egy eljárás, ami biztosítja, ha egynél több mester akarja a buszt vezérelni, akkor ezt csak egyetlen egy tudja megtenni, így adatvesztés nem léphet fel.

3.3.2 Bit átvitel

Az átvitel bit szinten a következő: az eredetileg magas szinten lévő **SDA** vonalra kerül a 0 vagy 1 értéknek megfelelő feszültségszint. Az **SCL** vonal magas szintje alatt érvényes az adat. Az adat csak az **SCL** vonal alacsony szintje alatt változhat.



3.J ábra: Bit átvitel, START és STOP feltétel az I²C buszon

A busz aktív és inaktív állapotát a **START** és **STOP** feltételekkel tudjuk definiálni

1. **START** feltétel akkor lép fel és a busz aktív lesz, amikor **SCL** magas állapotában az **SDA** vonalon egy H-L átmenet van.
2. **STOP** feltétel akkor lép fel, amikor **SCL** magas állapotában az **SDA** vonalon egy L-H átmenet van.

A **START** és **STOP** állapotokat csak a mester generálhatja. A busz aktív a **START** és **STOP** állapot között. Ezután válik a busz szabaddá.

3.3.3 Bájtvitel

Az **SDA** vonalon sorosan átvitt adat mindegyike 1 bájt = 8 bit hosszúságú. Az átvitt bájtok száma nincs korlátozva. Az adónak a vevő minden bájt vételét egy L szintű nyugtázó (**ACK** = acknowledge) bit küldésével igazolja. Az ehhez szükséges órajelet a mester generálja, az adó az **SDA** vonalat elengedi. A vevőnek ekkor az **ACK** generálásához le kell húznia az **SDA** vonalat. Az átvitel a legmagasabb helyiértékű (**MSB**) bittel kezdődik.

Az adatbiteket az adó, az **ACK** bitet a vevő küldi. Ha egy vevő nem képes egy adatot venni — mert egyéb feladatokat lát el —, akkor ezt két módon jelezheti:

1. *Nem küld nyugtázást (**ACK** bitet) a bájt végén. Ilyenkor a mester azonnal megszakítja az adatforgalmat, és később újra próbálkozik, természetesen egy újabb **START** feltétellel kezdve.*
2. *Alacsony szinten tartja az **SCL** vonalat, ezt megteheti akár egy bájt vételének a "kellős közepén" is. Ilyenkor a mester megvárja míg a szolga "elereszti" az **SCL** vonalat, és ott folytatja, ahol az adatforgalom félbeszakadt.*

Fontos kiemelni, hogy az adat- és az órajelet vonalat az adó és a vevő felváltva használja. Ez megköveteli mind az adó, mind a vevő számára a nagyon pontos kommunikációs feltételek betartását.



3.K ábra: Bájtvitel az I²C buszon

A "minden bájt nyugtázása" szabály alól két kivétel van:

1. *Az egyik akkor lép fel, ha a mester a vevő (**MST/RCV**). Ilyenkor jelezni kell az adatbájt sorozat végét, a küldőnek nem adva **ACK**-ot. Az **ACK***

jelhez kapcsolódó órajelet a mester természetesen generálja, de az SDA vonalat nem húzza le L szintre. Ezt hívják negatív nyugtázásnak (NACK).

- 2. A másik kivétel: a szolga akkor küld NACK jelet, ha nem képes újabb adatbájtokat elfogadni. Ez akkor lép fel, ha olyan átvitelt kezdeményezünk, amit nem képes fogadni.*

3.3.4 Adatforgalom a buszon

A buszon lévő minden eszköznek saját címe van. Mielőtt adatátvitel történne a buszon, a mester **START** állapotba hozza a buszt, majd kiadja a buszra a szolga címét, amelyikkel adatot akar cserélni. Az a szolga, amelyik felismeri a saját címét, **ACK** jelet küld vissza. A címzést a mester végzi közvetlenül **START** állapot után. Ez az első küldött bájtt.

A cím hét bites. A nyolcadik bit dönti el a szolgálával történő adatcsere irányát. 0.bit jelöli az írást, ilyenkor a mester küld adatokat (W), 1 értékű bit pedig az olvasást (R).

A buszra kapcsolódó eszközök címei két kategóriába sorolhatók: Az egyik kategóriában a cím programozható, ezek általában a mikrokontrollerek. A másik kategóriát a különféle funkciókat megvalósító periféria áramkörök alkotják.

Ezeknél az eszközök címe két részből tevődik össze: egy típus címből (4 bit) és egy hardver címből (3 bit); az eszköz tokozásán a megfelelő lábak 0-ba, ill. 1-be kötésével. A típus cím az azonos (típusú) tokoknál mindig megegyezik. Ezzel a címmel jelentkezik be a slave eszköz, ill. ezzel a címmel szólítja meg a master eszköz a slave-et adatcsere előtt.

Az egy mester — több szolga struktúrájú buszon zajló adatátvitel a következő (**M** jelöli a mester, **L** a szolga által küldött adatbiteket):

MASTER WRITE

A mester **START** állapotba hozza a buszt (S) és kiküldi a szolga címét. A cím legkisebb helyiértékű bitje $W=0$. Ezt a szolga az **ACK** jel visszaküldésével igazolja (A). Ezek után a mester küldi az adatokat a szolgának (DATA), és az minden bájtt vételét (A) küldésével igazolja. Az utolsó adat küldése után a mester **STOP** állapotba hozza a buszt (P).

MASTER READ

A mester **START** állapotba hozza a buszt (S) és kiküldi a szolga címét. A cím legkisebb helyiértékű bitje $R=1$. Ezt a szolga az **ACK** jel visszaküldésével igazolja (A). Ezek után a mester fogadja az adatokat a szolgáltól (DATA), és minden bájtt vételét (A) küldésével igazolja. Az utolsó adat küldését a mester negatív nyugtázással jelzi (NA). Ezek után a mester **STOP** állapotba hozza a buszt (P).

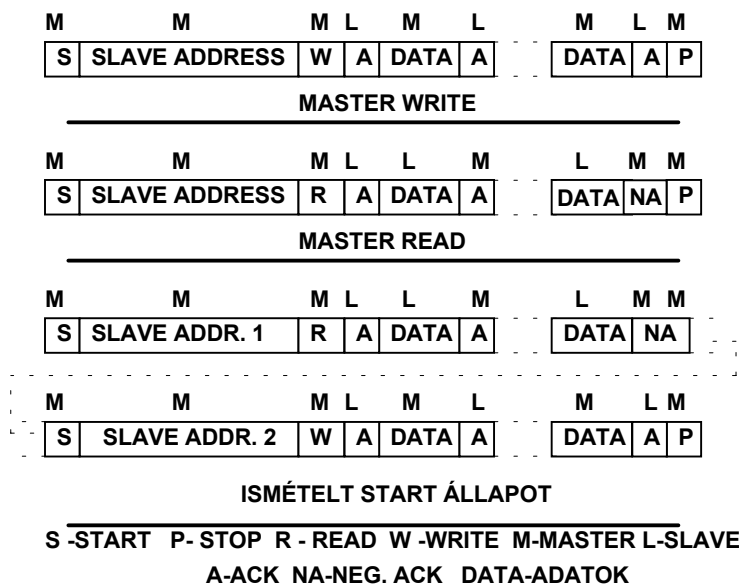
A fenti esetben a mester minden átvitelnél a buszt újból nyitja és zárja.

Amennyiben a buszon több szolgálval akar a mester adatot cserélni, a minden átvitelt lezáró **STOP**, majd az indító újabb **START** állapot sokat lassít az átvitelen. Ilyenkor használható az ismételt **START** állapot generálása. Ez az jelenti, hogy az átviteleket nem **STOP** (P), hanem a következőt indító **START** (S) állapottal fejezzük be, azaz a mester a buszt folyamatosan használja.

Ismételt **START** állapot használata

Itt egy olyan esetet látható, amikor először a mester adatokat kapott egy szolgától, majd utána adatokat küld nem szükségképpen azonos című szolgának.

A fenti adatátviteli protokollokat két módon lehet megvalósítani: vagy beépített hardver segítségével (ilyeneket tartalmaznak az erre felkészített mikrokontrollerek és az I²C buszra kifejlesztett periféria áramkörök), vagy megvalósítására szoftver "bit-billegettetés" programot írhatunk. A fent ismertetett adatátviteli változatokat a **3.12 ábrán** foglaltuk össze. Az M illetve L betűk azt jelzik, hogy az adott bájtos a mester vagy a szolga küldi.



3.L ábra: Különféle típusú adatforgalom az I²C buszon

3.3.5 I²C kommunikációt támogató rutinkönyvtár az SX-hez

A rutinkönyvtár forráskódja a mellékelt mágneslemez-mellékleten található: **SX_I2C.SRC** néven. A könyvtár tartalmazza a teljes I²C kommunikációt megvalósító szubrutin formájú virtuális periféria kódját, az ehhez szükséges port

inicializáló rutinokat, és az SX-DEMO BOARD-on lévő 24LC01B típusú soros EEPROM író és olvasó rutinjait. A fontosabb rutinok:

START_I2C - Generál egy START-ot, és elküldi a SLAVE címet

Bemenet: W

Kimenet: -

STOP_I2C - Generál egy STOP-ot

Bemenet: -

Kimenet: -

PUT_I2C - Egy bájt kiküldése az I2C buszra

Bemenet: W

Kimenet: -

GET_I2C - Egy bájt vétele az I2C buszról

Bemenet: ACKBIT=1 → ACK-ot küld, ACKBIT=0 → NACK-ot küld

Kimenet: W

PUT_EPR - Egy bájt kiírása az EEPROM megadott címére

Bemenet: W (adat), EPR_ADR (automatikusan inkrementálva művelet után!!)

Kimenet: -

GET_EPR - Egy bájt beolvasása az EEPROM megadott címéről

Bemenet: EPR_ADR (automatikusan inkrementálva művelet után!!)

Kimenet: W

3.4 SX mikrokontrolleren futó monitor program

A program forráskódja a mellékelt mágneslemez-mellékleten található: **SX_MONI.SRC** néven. A monitorprogram egyszerű kommunikációs feladatokat valósít meg, és az RS-232, I²C rutinkönyvtárak alkalmazására mutat példát. Továbbá tartalmaz néhány egyéb, az Internetről letöltött érdekes virtuális perifériát. Ahhoz, hogy a monitorprogram jól tudjon kommunikálni a PC-vel, szükség van egy terminál emulációt megvalósító programra. Erre tökéletesen megfelel a **4.0 szakaszban** ismertetett **UniCOMM program**. A terminál következő virtuális perifériákat tartalmazza:

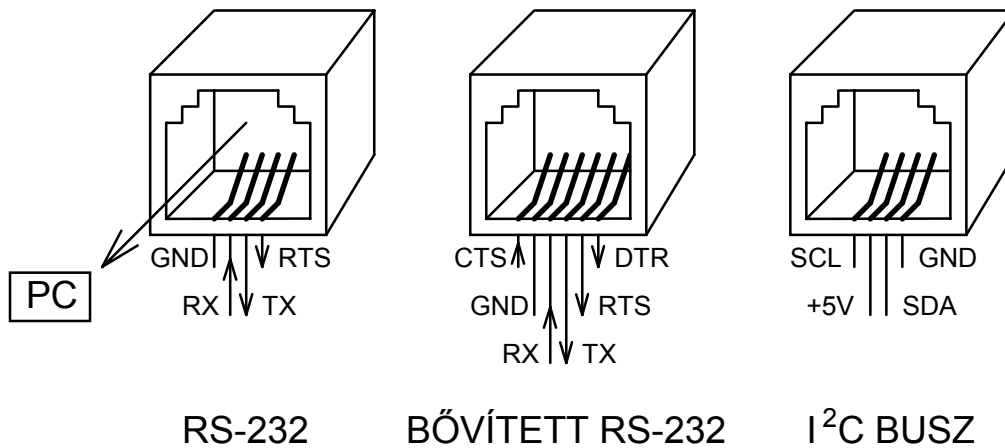
1. RS-232 kommunikáció
2. I²C kommunikáció, 24LC01B EEPROM műveletek
3. Valós idejű óra
4. Analóg bemenet (szigma-delta A/D)
5. Analóg kimenet (PFM alapú D/A)

A monitorprogram a következő parancsokat értelmezi:

- 0 - Help
- 1 - EEPROM írás, megadott címre
- 2 - EEPROM olvasás, megadott címről
- 3 - EEPROM hexdump
- 4 - EEPROM teljes törlés
- 5 - óra olvasás (írás csak a UniCOMM "t" parancsával)
- 6 - Analóg(0) kimenet beállítása
- 7 - Analóg(2) bemenet beolvasása
- 8 - Analóg(2) bemenet → Analóg(0) kimenet (átjátszó)
- t - UNICOMM V4.3 kiszolgáló, óra írás (kis "t" !)
- * - újraindítás

3.5 Alternatív soros csatlakozó lehetőségek

A telefonszolgáltatások széleskörű elterjedésével a hagyományos telefonkábel és telefoncsatlakozó más irányú felhasználási lehetősége is előtérbe került, és elsősorban a kis vezetékszám-igényű soros kommunikációs megoldásoknál terjedt el. Szakcsoportunkban (Kandó Kálmán Műszaki Főiskola - Automatika Intézet - Elektronika Szakcsoport) a **3.13 ábrán** látható házi szabványok alakultak ki.



3.M ábra: Telefoncsatlakozó-bekötési megoldások

3.6 RS-232 —TTL szintillesztő megoldások

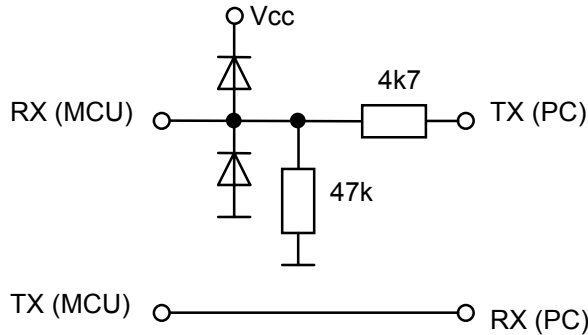
Régi probléma az RS-232 és a TTL rendszerek egymáshoz történő illesztése. TTL rendszerben a logikai 1 szintnek 5V, a logikai 0 szintnek 0V felel meg. RS-232 rendszerben az -3..-25V illetve +3..+25V (lásd a **3.2.1 szakaszt**).

Az ipari gyakorlatban a nagy megbízhatóságú (és nem kifejezetten olcsó) célintegrált áramköröket alkalmaznak, elsősorban a **MAX-232** illetve a **MAX-233** típusokat (ez utóbbi abban különbözik az előbbitől, hogy nem igényeli külső kondenzátorok bekötését). A főiskolai gyakorlatban viszont az egyes áramköri megvalósítások mélyebb megismerése érdekében — és nem utolsó sorban anyagi okok miatt — a diszkrét elemekből felépített szintillesztő megoldások is előtérbe kerülnek. Az illesztési probléma a mi esetünkben kizárólag személyi számítógépek és mikroprocesszoros/mikrokontrolleres áramkörök összekötéséből fakad. Mérések szerint a PC soros portjának a kimenő feszültsége a -15V..+15V tartományba esik.

3.6.1 **Egyszerű ellenállásos szintillesztő**

A kapcsolás a **3.14 ábrán** látható. PC→ μ C irány: A PC soros port TX pontjának a +/-15V-os feszültségét a diódákból és a 4.7k Ω -os ellenállásból álló hálózat limitálja 0..5V-ra. CMOS mikrokontrollerek port-lábai diódákkal védettek, így ebben az esetben a külső diódák akár el is hagyhatók. A másik adatirány (μ C→PC) még ennél is egyszerűbb, semmilyen külső alkatrészt nem tartalmaz. Ez azért lehetséges, mert a tapasztalat szerint a PC soros portok RX pontja a szabványostól eltérően már a 0V-os bemenő feszültséget is logikai 1-nek veszi. A 47k Ω -os ellenállás szerepe annyi, hogyha megszakad a kapcsolat, akkor stabil 0 V-os logikai 1 szintet biztosít a mikrokontroller számára. Mivel az áramkör nem

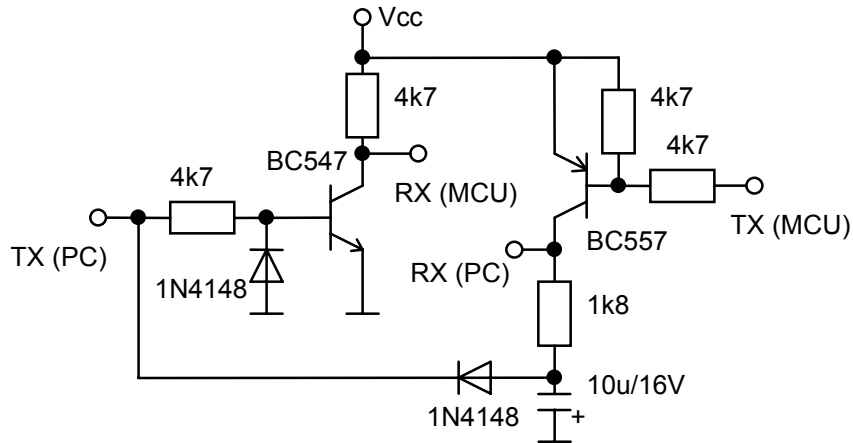
invertál, azért a mikrokontrollerben a vett és a kiküldött adatokat komplementálni kell.



3.N ábra: Ellenállásos RS-232 — TTL szintillesztő

3.6.2 Tranzisztoros szintillesztő

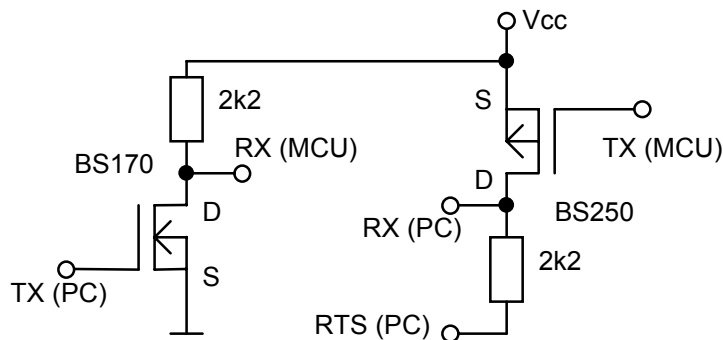
A kapcsolás a **3.15 ábrán** látható. A PC→ μ C irány egy egyszerű tranzisztoros inverter, a negatív feszültségek limitálását a tranzisztor bázisára csatlakozó dióda végzi. A μ C→PC irány szintén egy tranzisztoros inverter, de PNP tranzisztorral megvalósítva. Ha a tranzisztor kinyit, akkor +5V kerül a PC RX pontjára, ha zárva van, akkor -15V, amit a kondenzátor biztosít. A kondenzátor folyamatos töltése a diódán keresztül valósul meg, amikor a PC TX vonala inaktív (-15V-os, logikai 1 szintű) állapotban van. A kapcsolás elvileg lehetőséget ad a teljes duplex kommunikációra (mindkét irányba folyik az adatforgalom egyszerre), de a kondenzátor töltése a PC TX vonal +15V-os állapotában nem biztosított, így az PC RX vonal számára a -15V-os forrás bizonytalaná válhat a kondenzátor fokozatos kimerülésével.



3.O ábra: Tranzisztoros RS-232 — TTL szintillesztő

3.6.3 FET-es szintillesztő

A kapcsolás a **3.16 ábrán** látható. Ez a kapcsolás is két invertert tartalmaz, MOS-FET tranzisztorokkal megvalósítva (az egyik itt is p, a másik n típusú). A kapcsolás teljesen duplex, mert a PC RX vonala számára szükséges negatív feszültséget maga a PC szolgáltatja az RTS vonalán (ld. a **3.2.1 szakaszt**) keresztül, így rögtön látszik a kapcsolás hátránya is: csak olyan PC-s programmal használható, amely gondoskodik az RTS vonal -15V-on tartásáról. A **4.0 szakaszban** ismertetett **UniCOMM program** minden további nélkül képes erre. Viszont előny, hogy a FET-ek alkalmazása miatt — feszültségvezérelt elemről lévén szó — elmaradhatnak a bázisköri elemek (ellenállás, dióda), így az alkatrészigény minimális.

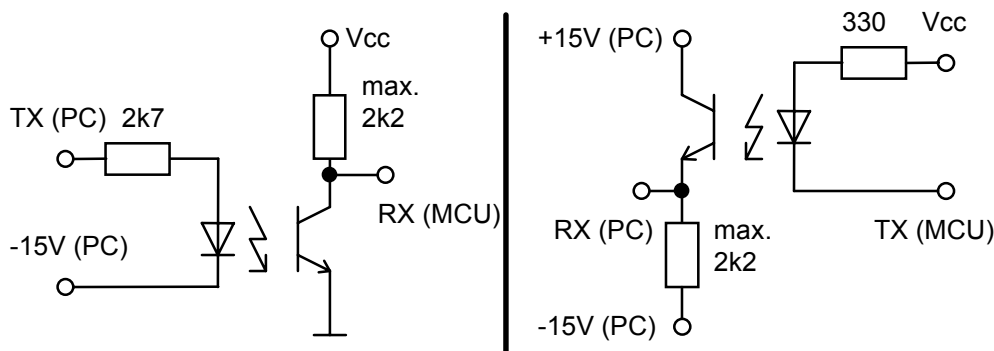


3.P ábra: FET-es RS-232 — TTL szintillesztő

3.6.4 Optocsatolt szintillesztő

A kapcsolás a **3.17 ábrán** látható. A PC→ μ C irány itt is inverteres, mely az optocsatoló kimeneti tranzisztorával van megvalósítva. A μ C→PC irány viszont emmitterkövetős, az invertálást úgy valósítjuk meg, hogy a fotodióda aktív szintje a 0V-os MCU TX feszültséghez tartozik. Erre azért volt szükség, mert a mikrokontrollerek egy jelentős részének (az SX nem ilyen) a kimenete nagyobb áramot képes elnyelni aktív 0 állapotban, mint szolgáltatni aktív 1 állapotban. A fotodiódák meghajtása pedig köztudottan jelentős áramot igényel.

Az áramkör előnye, hogy teljes galvanikus leválasztást biztosít. Viszont a PC irányából már két segédfeszültséget igényel: +/-15V-ot. A PC RTS vonala mellett erre a célra még a DTR vonal használható fel második forrásként. A **4.0 szakaszban** ismertetett **UniCOMM program** minden további nélkül képes a megfelelő feszültség szintek biztosítására.



3.Q ábra: Optocsatolt RS-232 — TTL szintillesztő

Sokan megfélekednek az optocsatolók azon tulajdonságáról, hogy a kimeneti fototranzisztort tulajdonképpen egy fotodióda és egy n-típusú tranzisztor kombinációjaként kell felfogni. A fotodióda kapacitása párhuzamos a kollektor-bázis pályával, és így Miller-kapacitásként hat. Minél nagyobb a munkaellenállás értéke, a kapcsolás feszültségerősítése és így a Miller-kapacitás értéke is egyre nagyobb. Ez a hatás magasabb frekvenciákon (10-20 kHz) már a fel- és lefutási idők jelentős növekedését idézheti elő, ezért 2.2k Ω -nál nagyobb munkaellenállások alkalmazása nem javasolt.

4. A "UniComm" program

Ezt a programot a mindennapi gyakorlati igények születték. Azokat a — PC soros portján lebonyolítandó — kommunikációs feladatokat igyekeznek kielégíteni, illetve megkönnyíteni, amelyek a szakcsoportunk (Kandó Kálmán Műszaki Főiskola - Automatika Intézet - Elektronika Szakcsoport) eddigi tevékenységében a leggyakrabban felmerültek.

A program neve (UniComm) tulajdonképpen egy rövidítés (Universal Communicator), a "universal" (univerzális) kifejezés a széleskörű alkalmazhatóságra utal, amely magában foglalja a későbbi feladatok során felmerülő problémák irányába történő bővítést is.

A most következő leírás a UniComm program legfrissebb 4.3-as verzióját ismerteti.

```
UNICOMM - Universal Communicator
Copyright (C)1998-1999 Lamár Krisztián
[ lamar@winnie.obuda.kando.hu ]
=====
```

4.1 A programmal kapcsolatos tudnivalók:

- 1. A program tetszőlegesen megszakítható az ESC, F10, ALT-X billentyűkkel. (Kilépéskor megerősítést vár!)*
- 2. Bármikor kérhető Súgó (Help) az F1 billentyű leütésével.*

4.2 A program parancssorból való hívása:

```
UNICOMM [paraméterek]
```

A paraméterek sorrendje tetszőleges lehet.

Az értelmezett paraméterek:

- COM1..COM4:** a soros kommunikációs port
- 300..57600:** a soros kommunikációs sebesség megadása. Csak a szabványos BAUD értékeket veszi figyelembe.
- S1, S2:** a soros kommunikáció során használt stopbitek száma
- PARSE:** ennek hatására az UNICOMM azonnal parancsértelmező módban indul, egyébként terminál emulációs módban.
- RTS- , RTS+:** a soros port a RTS vonalának a induló feszültség szintjét állítja be -12 vagy +12 V-ra.
- DTR- , DTR+:** a soros port a DTR vonalának a induló feszültség szintjét állítja be -12 vagy +12 V-ra.
- (Ha nem adunk meg induló RTS vagy DTR szint értéket, akkor a program ezeket nem módosítja induláskor. Továbbá a helyes működéshez a paraméterlistában a kommunikációs portot kell előbb beállítani, mert különben az RTS/DTR szintváltás a alapértelmezés szerinti COM1 porton megy végbe.)
- STIMU:** Ha a programot terminál módban indítjuk, kiküld egy szökőz karaktert, aminek a hatására a 80C552 alapú gyakorló egy '?' után promptot ad.
- [fájlnév]:** fájlnev megadása a 80C552 alapú gyakorlóba történő letöltéshez. (A fájl formátuma szabványos Intel-HEX, az alapértelmezett kiterjesztés: .HEX)
- H, /H, ?:** a parancssor-paramétereket kilistázó help megjelenítése.

Alapbeállítás: COM1 9600 S1

Megjegyzések a paraméterekhez:

A fájlnev kivételével a paraméterek megadhatók kötőjel prefix-szel is: pl. a **TERMINAL** és a **-TERMINAL** paraméter ugyanazt a hatást eredményezi.

További értelmezett paraméterek: **1 = COM1** ill. **2 = COM2**

80C552 gyakorlóval (Monitor Verzió 1.96 & 1.97) való tesztelés esetén 2 STOP bitet kell beállítani!

4.3 Parancsértelmező mód

A program karakteres billentyű-parancsokat értelmez, azokat továbbítja a beállított soros portra, illetve meghatározott karaktereket vár onnan.

A program különbséget tesz a kis és a nagybetűk közt!

Az értelmezett billentyű-parancsok köre egyszerűen bővíthető a program forráskódjának birtokában.

Nem értelmezhető billentyű-parancs esetén a program **UNKNOWN COMMAND** hibaüzenetet ad.

Ha a program bármilyen adatot vár a soros portról, és nem kapja meg 200 msec-en belül, akkor **TIMEOUT** hibaüzenetet küld.

4.3.1 A V3.x verziókban értelmezett billentyű-parancsok

e : A kommunikáció ellenőrzése. Az **e** karakter kiküldése után azonnal a nyugtázó (**o**) karaktert várja. Ha a program ezt várja, de nem ezt kapja (a soros portról), akkor az **ILLEGAL ACK** hibaüzenetet küldi.

t : A **t** karakter kiküldése után a program a rendszer dátumot és a rendszeridőt küldi ki hexa karakteres formában. Az adatbájtokat szóköz választja el, a szekvenciát CR terminálja. (20H = szóköz, 0DH = CR).

```
pl:    98.12.03.    13:52:09    >>>
39H, 38H, 20H,  31H, 32H, 20H,  30H, 33H, 20H,
31H, 33H, 20H,  35H, 32H, 20H,  30H, 39H, 0DH
```

d : HEXA formátumú karaktereket tartalmazó (nem INTEL-HEX!!) fájl letöltése a soros porton keresztül. A program csak az (a..f), (A..F) és a (0..9) karaktereket fogadja el, a többi figyelmen kívül hagyja. A parancs kiadása után a program bekéri a letöltendő fájl nevét. Ha kiterjesztést nem adunk meg, akkor a program automatikusan a **.HEX** kiterjesztést rendel a név mellé. Ha a fájl nem nyitható meg, hibaüzenetet kapunk. Ha ugyanazt a fájlt szeretnénk többször egymás után letölteni, elég egyszer beírni a nevet, a későbbiekben elég csak ENTER-t nyomni a fájlnev helyett. Ha minden rendben, a program először kiküldi a **d** betűt, majd a fájl tartalmát úgy, hogy két karakterenként (ha úgy tetszik: bájtonként) beiktat egy szóköz (SPACE) karaktert. Végül a CR karakterrel terminálja a szekvenciát. A letöltési folyamat állapotáról egy ún. hőmérő szimbólum ad tájékoztatást. Tévedések elkerülése végett: a

fájlból történő beolvasáskor csak a Hexa karaktereket veszi figyelembe, így többek között a szóközöket is 'eldobja'. Ezután teszi be két karakterenként az elválasztó-szóközöket, ami végül is újratördelésként is felfogható.

pl: Ha a fájl tartalma: aBxCds4 56 fE#8
Akkor így küldi ki: AB CD 45 6F E8

w : Bájtos formátumú adatokat tartalmazó fájl letöltése a soros porton keresztül. A parancs kiadása után a program bekéri a letöltendő fájl nevét. Ha kiterjesztést nem adunk meg, akkor a program automatikusan a **.BIN** kiterjesztést rendeli a név mellé. Ha a fájl nem nyitható meg, hibaüzenetet kapunk. Ha ugyanazt a fájlt szeretnénk többször egymás után letölteni, elég egyszer beírni a nevet, a későbbiekben elég csak ENTER-t nyomni a fájlnev helyett. Ha minden rendben, a program először kiküldi a **d** betűt. Azért a **d** betűt, mivel a program az adatokat a szolga-gép (mikrokontroller) felé hexa formátumú karakteres formába konvertálva küldi ki, tehát a szolga gép ugyanolyan formában kapja az adatokat, mint a **d** parancs esetén, a **w** betű csak az UNICOMM programot utasítja, hogy végezze el az átkonvertálást a fájl tartalmán kiküldés előtt. Ezért a **w** parancs hivatalos neve: **w(d)**. Lásd a sűgót az **F1** billentyűvel. A **d** betű után a fájl tartalmát küldi ki úgy, hogy kétkarakterenként (ha úgy tetszik: bájtonként) beiktat egy szóköz (SPACE) karaktert. Végül a CR karakterrel terminálja a szekvenciát. A leöltési folyamat állapotáról egy ún. hőmérő szimbólum ad tájékoztatást.

a : ASCII karaktereket tartalmazó fájl letöltése a soros porton keresztül. A program minden karaktert elfogad, kivéve a '03' ASCII kódút, mivel ez az ETX (End Of Text) karakter, ami a szekvenciát terminálja. A parancs kiadása után a program bekéri a letöltendő fájl nevét. Ha kiterjesztést nem adunk meg, akkor a program automatikusan a **.TXT** kiterjesztést rendeli a név mellé. Ha a fájl nem nyitható meg, hibaüzenetet kapunk. Ha ugyanazt a fájlt szeretnénk többször egymás után letölteni, elég egyszer beírni a nevet, a későbbiekben elég csak ENTER-t nyomni a fájlnev helyett. Ha minden rendben, a program először kiküldi az **a** betűt, majd a fájl tartalmát, végül a ETX karakterrel terminálja a szekvenciát. A leöltési folyamat állapotáról egy ún. hőmérő szimbólum ad tájékoztatást. Mint látható, ebben az esetben a program minden karaktert elfogad az ETX kivételével. Ez nem fog különösebb gondot jelenteni, mivel ezt a parancsot főleg formátumozott szövegek letöltésére használhatjuk.

- u :** HEXA formátumú karaktereket tartalmazó (nem INTEL-HEX!!) fájl feltöltése a soros porton keresztül. A program csak az (a..f), (A..F), (0..9) és a CR [CARRIAGE RETURN] karaktereket fogadja el, a többi figyelmen kívül hagyja. A parancs kiadása után a program bekéri a feltöltendő fájl nevét. Ha kiterjesztést nem adunk meg, akkor a program automatikusan a **.HEX** kiterjesztést rendeli a név mellé. Ha ugyanazt a fájlt szeretnénk többször egymás után felülírni, elég egyszer beírni a nevet, a későbbiekben elég csak ENTER-t nyomni a fájlnev helyett. Ha minden rendben, a program először kiküldi az **u** betűt, majd várja a soros portról a HEXA formátumú karaktereket, amíg a CR karaktert nem kapja. Ekkor a program az adatokat kiírja az előzőleg megadott fájlba. Ha az írás sikertelen volt, hibaüzenetet kapunk.
- v :** Bájtos formátumú adatokat tartalmazó fájl feltöltése a soros porton keresztül. A parancs kiadása után a program bekéri a feltöltendő fájl nevét. Ha kiterjesztést nem adunk meg, akkor a program automatikusan a **.BIN** kiterjesztést rendeli a név mellé. Ha ugyanazt a fájlt szeretnénk többször egymás után felülírni, elég egyszer beírni a nevet, a későbbiekben elég csak ENTER-t nyomni a fájlnev helyett. Ha minden rendben, a program először kiküldi az **u** betűt. Azért az **u** betűt, mivel a program a szolga-gép (mikrokontroller) felől hexa formátumú karakteres adatokat vár, majd azt konvertálja át bájtos formába, tehát a szolga gép ugyanolyan formában küldi az adatokat, mint a **u** parancs esetén, a **v** betű csak az UNICOMM programot utasítja, hogy végezze el az átkonvertálást a bejövő adatokon a fájl mentése előtt. Ezért az **v** parancs hivatalos neve: **v(u)** Lásd a súgót az F1 billentyűvel. A **u** betű kiküldése után a program várja a soros portról a HEXA formátumú karaktereket, amíg a CR karaktert nem kapja. Ekkor a program az adatokat kiírja az előzőleg megadott fájlba. Ha az írás sikertelen volt, hibaüzenetet kapunk. A program csak az (a..f), (A..F), (0..9) és a CR [CARRIAGE RETURN] karaktereket fogadja el, a többi figyelmen kívül hagyja.
- b :** ASCII karaktereket tartalmazó fájl feltöltése a soros porton keresztül. A program minden karaktert elfogad, kivéve a '03' ASCII kódút, mivel ez az ETX (End Of Text) karakter, ami a szekvenciát terminálja. A parancs kiadása után a program bekéri a feltöltendő fájl nevét. Ha kiterjesztést nem adunk meg, akkor a program automatikusan a **.TXT** kiterjesztést rendeli a név mellé. Ha ugyanazt a fájlt szeretnénk többször egymás után felülírni, elég egyszer beírni a nevet, a későbbiekben elég csak ENTER-t nyomni a fájlnev helyett. Ha minden rendben, a program először kiküldi a **b** betűt, majd várja a soros portról az adatokat, amíg az ETX karaktert

nem kapja. Ekkor a program az adatokat kiírja az előzőleg megadott fájlba. Ha az írás sikertelen volt, hibaüzenetet kapunk.

Fontos! a **d**, **w**, **a** parancsok egyazon fájlnev változót használnak, tehát, ha pl. letöltjük a BC.HEX hexa karaktereket tartalmazó fájlt, majd egy bináris fájl letöltésére adunk parancsot, de fájlnev helyett ENTER-t nyomunk, akkor a BC.HEX kerül letöltésre úgy, hogy a program a fájlt binárisként kezeli. Ugyanez vonatkozik az **u**, **v**, **b** parancs-csoportra is.

4.4 Terminál emulációs mód

Alapesetben a program terminál módban indul.

A program üzemmódja az **F3** billentyűvel váltható terminál emulációs és parancsértelmező üzemmód közt. Parancssorból is megadható, hogy a program indításkor azonnal parancsértelmező módban induljon. Terminál módban a billentyűparancsok nem működnek.

Az **ALT-F5** billentyűkombinációval az soros port **RTS** vonalának, az **ALT-F6** billentyűkombinációval pedig a **DTR** vonal feszültségszintje váltható ellenkező értékre. Az **RTS** és **DTR** vonalak indulási szintje megadható parancssorból az UNICOMM indításakor.

A képernyő aljára státusz-sor került, mely mutatja az üzemmódot (**PARSER / TERMINAL**) illetve az **RTS** és **DTR** vonalak szintjeit.

Terminál üzemmódban egy Intel-HEX formátumú MCS-51 program tölthető le a 80C552 alapú gyakorlóba az **F4** billentyűvel. A fájlnevet csak parancssorban lehet megadni. Az **F6** billentyű ezen felül még a program futást is elindítja 4000H fix címről (**GO:4000**). Az eredeti tervek között szerepelt, hogy a program az indítási címet az Intel-HEX fájl első rekordjából vegye, de tapasztalat szerint a C Compiler bizonyos esetben nem rakja cím szerint növekvő sorrendbe az Intel-HEX fájl rekordjait, így ebben az esetben rossz címen indulna a program. A letöltés leállítható az ESC billentyűvel.

Az **F5** billentyűvel lehetőség van csak a futtatást indítani (**GO:4000H**) pl. abban az esetben, ha a program már a memóriában van.

A terminál bizonyos ESCAPE szekvenciák hatására különleges műveletek azonnali és feltétel nélküli elvégzésére utasítható. Ezek a szekvenciák két karakterből állnak. Az első az ESC (ASCII-\$1B) karakter, amit egy nagybetűs karakter követ.

Az értelmezett ESCAPE szekvenciák:

ESC-H:	Cursor Home
ESC-J:	Display Clear + Cursor Home
ESC-U:	Upload

A funkció nélküli, de fenntartott ESCAPE szekvenciák:

ESC-A , ESC-B, ESC-C, ESC-D, ESC-F, ESC-O

Az ezektől különböző ESCAPE szekvenciákat a terminál megjeleníti a képernyőn. pl: ESC-G -re kiír egy 'ESC' és egy 'G' karaktert.

Annak ellenére, hogy terminál üzemmódban a billentyű parancsok nem működnek, lehetőség van a feltöltésre, de az eredmény minden esetben egy bájtos formátumú adatfájl lesz. Az automatikus feltöltés az ESC-U karakterpár vételével kezdődik meg. A feltöltési szekvencia a következő (a soros vonalon ennek kell bejönni) :

- 1. Az ESC-U karakterpár után jön a SZIGORÚAN 12 karakterből álló fájlnev (ezen a néven lesz elmentve az adathalmaz) pl.: UPLOADED.DAT*
- 2. Ezután jön a tetszőleges hosszúságú, HEXA karakteres formátumú adat sorozat (előbb a magasabb, utána az alacsonyabb helyiértékű bájt-félt [nibble] reprezentáló karakter jön be).*
- 3. A szekvenciát az EOF karakter (ASCII-\$1A) karakter zárja.*

4.5 A V4.x verziók változásai a V3.x verziókhöz képest

Megváltoztak a **t**, **d**, **w** parancsok:

A **d**, **w** és **t** parancsok az egyes adatbájtok között (két karakterenként) kiküldenek egy szóköz karaktert (20H) is. Az utolsó adat után viszont szóköz helyett CR-t küld, ezzel jelezve, hogy vége az adatoknak. A régi CR-LF terminálás megszűnt. Ezekre azért volt szükség, hogy a letöltés terminál módban a billentyűzetről kézzel emulálható legyen.

A következetesség miatt a **t** parancs is hexa karakteres formában tölti le az adatokat. (20H = szóköz, 0DH = CR).

Új ASCII letöltési és feltöltési parancs: **a** és **b**, alapértelmezett kiterjesztés: **.TXT**. Ha gondot jelent, hogy ez a módszer 03H-ra terminál, akkor használjuk a **w(d)** ill. **v(u)** parancsokat, melyek bináris fájl létrehozására szolgálnak.

Nyugtázást már csak az **e** parancs vár.

A program alapbeállítás szerint most már terminál módban indul parancsértelmező mód helyett.

4.6 További tervek

A parancsértelmező mód billentyű parancsai egy **UNICOMM.CFG** nevű konfigurációs fájlban szabadon állíthatók lesznek.

Minden paraméter (baud-rate stb.) szintén benne lesz ebben a fájlban. De továbbra is meg lehet majd adni a paramétereket parancssorból is.

Futás közben állíthatók lesznek a paraméterek pl. baud-rate.

A letöltött/feltöltött fájlok belülről hívható szövegszerkesztővel kilépés nélkül megnézhetők, módosíthatók lesznek. Mindenki a kedvenc editorát használhatja, mivel ezt is a **.CFG** fájlban kell majd megadni.

A soros portokhoz rendelt **IRQ** állítható lesz.

Irodalomjegyzék

1. *SX18AC / SX20AC / SC28AC Datasheet, Scenix Semiconductor Inc.*
2. *SX18AC / SX20AC / SX28AC User's manual, Scenix Semiconductor Inc.*
3. *SX-KEY Development System Manual Version 0.7, Parallax Inc.*
4. *Craig Webb: Designing Virtual Peripherals for the SX, Scenix Semiconductor Inc.*
5. *Dr. Kónya László: SX mikrokontroller — A világ leggyorsabb mikrovezérlője, Rádiótechnika, 1998/8-9.*
6. *Dr. Kónya László: Számítógép hálózatok, LSI oktatóközpont, 1996.*
7. *Dr. Madarász László: A PIC16C mikrovezérlők, GAMF, 1996.*
8. *Dr. Madarász László: Bitsoros jelkapcsolatok a mikroszámítógépekben, Rádiótechnika Évkönyve, 1997.*
9. *Ferenczi Ödön: Telesítményszabályozó áramkörök. Műszaki Könyvkiadó, 1981.*



ChipCAD Elektronikai Disztribúció Kft.
1046 Budapest, Kiss Ernő utca 3.
Tel: 399-42-90, Fax: 399-42-99
e-mail: info@chipcad.hu
<http://www.chipcad.hu>
