Eötvös Loránd Tudományegyetem

Informatika Kar, Média- és Oktatásinformatika Tanszék

Tanári képzés szakdolgozata

Integrált áramkörök programozása

Témavezető:

Heizlerné Bakonyi Viktória

mestertanár

Készítette:

Németh Zsolt Balázs

informatika-matematika

Budapest, 2008

Előszó a szakdolgozathoz

Az informatika rohamos fejlődésével egyre nagyobb szerepet kap a programozás, kulcskérdés a jövő programozó generációjának kinevelése. Középiskolában minden tantárgy esetében, így az informatikánál is jelentős probléma az érdeklődés felkeltése és megtartása. Szakdolgozatom célja, hogy kedvet hozzon a középiskolás programozási tudás elmélyítéséhez. Az elektronika segítségével és programozási ismeretekkel viszonylag gyorsan igen látványos eredményeket érhetünk el, amit még a digitális szakadék túloldalán állók is értékelni fognak: hiszen valamilyen kézzelfoghatót alkottunk. Véleményem szerint ez felkeltheti az érdeklődést és a tanulók szívesebben foglalkoznak mélyebben a programozással, valamint gyorsabban fognak haladni annak elsajátításában is.

Jelen tanári segédtankönyv az integrált áramkörök programozásáról szól. A könyv által ismertetett mikrovezérlőket sok egyéb mellett Pascal nyelven is lehet programozni, én ezt választottam, több okból: egyrészt egyszerű nyelv ahhoz, hogy beletanuljunk a programozásba, gyorsan eljuthatunk az első működő programhoz, így hamar sikerélménye lesz a tanulóknak, illetve igen sok középiskolában még mindig Turbo Pascalt tanítanak. Másrészt teljesen hasonló C és Basic fordító is elérhető, így az anyag könnyen átültethető más nyelvekre.

Szakdolgozatom írásakor Microchip PIC Pascal nyelven történő programozásáról készült könyv nincs magyar nyelven, ez a könyv ezt az űrt hivatott betölteni. A magyar nyelven kifejezetten kezdők számára írt PIC-ekkel foglalkozó szakirodalom is elég csekély.

A könyv felépítése olyan, hogy a megfelelő előismeretekkel rendelkező olvasó a minimális és egyben elengedhetetlen elméleti tudás megszerzése után máris építhet és programozhat egy önállóan működő áramkört. A későbbiekben fokozatosan mélyítheti el a tudását az egyre nehezebb feladatok megoldása során. Az elméleti és gyakorlati ismeretanyag nem határolódik el élesen egymástól.

A könyv tanári segédkönyv, így a tanár felkészülését hivatott segíteni. A könyv alapján a megszerzett tudás a középiskolai tanulók képességeinek figyelembe vételével átadható, a könyvben leírt feladatokkal pedig elmélyíthető és gyakorolható a programozás.

Bevezető

A könyv az alapoktól indulva ismerteti az integrált áramkörök programozásának rejtelmeit. Használható otthoni tanulásra, valamint egy szakkör tananyagának alapjaként is.

Az új fogalmak bevezetésükkor dőlten vannak szedve, valamint minden fejezet végén található egy összefoglalás és feladatok segítik elő a tananyag megértését és elmélyítését.

A könyvben szereplő programok forráskódja, a felhasznált programok telepítőkészlete, a használt IC-k adatlapja megtalálható a könyvhöz tartozó CD mellékleten. A programok forráskódja a Példaprogramok könyvtár program nevének megfelelő könyvtárában található meg.

Az alábbi ismereteket feltételezi a könyv:

- Magabiztos programozási ismeretek, egyszerűbb programozási tételek ismerete (pl. összegzés, megszámolás, eldöntés, kiválasztás, keresés).
- Valamilyen Pascal nyelv alapszintű ismerete (pl. Turbo Pascal), legalább az elágazás, ciklusok, függvények és eljárások, érték és hivatkozás szerint átadott paraméterek, globális változók, konstansok, elemi típusok, valamint tömbök ismerete ajánlott. A könyvben minden a szokásos Pascal nyelvjárástól eltérő nyelvi elemet ismertetek.
- Alapvető elméleti elektronikai ismeretek: feszültség, áramerősség, ellenállás és teljesítmény fogalma, Ohm-törvény, soros és párhuzamos kapcsolás, elektronikus zaj fogalma.
- Alapvető gyakorlati elektronikai ismeretek: a könyvben tárgyalt alkatrészek működési elvének ismerete, valamint gyakorlati használata, pl. feszültségforrás, ellenállás, LED, nyomógomb.
- Erős (emelt szintű) matematikai és informatikai ismeretek, így pl. számrendszerek, Boole-algebra (logikai műveletek).
- Angol nyelv alapfokú ismerete (ugyanis a használt programok (pl. a fejlesztői környezet) valamint az adatlapok angol nyelvűek)

A könyvben Microchip cég által forgalmazott ún. PIC-ekkel fogunk foglalkozni. Sok fajta programozható integrált áramkör van, azért ezekre esett a választás, mert igen elterjedtek, olcsók, meglehetősen strapabíróak, valamint elég széleskörű a támogatottságuk.

Ezeket az PIC-eket többféle nyelven lehet programozni, mi a Pascal nyelvű programozással fogunk megismerkedni, ehhez pedig a mikroPascal fejlesztői környezetet fogjuk használni (ami ingyenesen elérhető).

Amire szükségünk lesz

Sajnos az elektronika nem olcsó hobbi, de mégis próbáltam a lehető legolcsóbb lehetőségeket összegyűjteni, ha valaki meg szeretne ismerkedni a mikrovezérlők programozásával. Az alábbiakban a szükséges dolgokról összeállított lista olvasható.

- Windows XP vagy Vista operációs rendszer. A fejlesztői és szimulációs környezetek telepítéséhez és futtatásához nem feltétlenül szükséges rendszergazdai jogosultság, viszont a PIC programozók telepítéséhez szinte már biztosan (bővebb információ és vásárlás: http://www.microsoft.com/).
- MikroPascal fejlesztői környezet: ingyenesen hozzájuthatunk a bemutató változathoz (bővebb információ: http://www.mikroe.com/).
- MiSim szimulációs környezet: ingyenes program (letöltés: http://www.feertech.com/misim/). Ehhez szükséges egy Java virtuális gép is (letöltés: http://java.sun.com/).
- Microchip PIC, típusa szerint PIC16F877-20/P vagy a vele kompatibilis PIC16F877A-20/P. Beszerezhető pl. a ChipCAD Kft.-nél (bővebb információ: http://www.chipcad.hu/).
- A PIC-eket programozni tudó készülék (PIC programozó vagy PIC bootloader programozó), ezzel fogjuk az általunk megírt programokat a PIC-be égetni. A könyvben ismertetem a Microchip PICSTART PLUS-t, a GTP-USB PLUS-t és a házilag összerakott PIC programozókat, bemutatom ezek előnyeit, hátrányait és hogy melyiket hol lehet beszerezni. PIC programozó esetében sajnos választanunk kell, hogy drága vagy körülményes (esetleg nem működő) eszközt szeretnénk. A bootloadereknek szintén egy külön fejezetet szenteltem. Ezekkel az eszközökkel jelentős megtakarítást érhetünk el, de ettől még minden PIC-be be kell programozni egyszer az ún. bootloadert, viszont ezek után használhatjuk a PIC bootloader programozókat a programozáshoz. Minden programozó esetében figyeljünk arra, hogy van-e megfelelő csatlakozónk a számítógépen, illetve, hogy a programozó szoftvere támogatja-e a számítógépünk architektúráját (pl. 64 bites és többprocesszoros gépek). Sajnos ezek az eszközök sokszor elavultak. Bővebb információ és linkek a későbbi fejezetekben.
- Elektronikai alkatrészek:

- forrasztás-mentes próbapanel (protoboard),
- vezetékek a protoboardon való összekötéshez,
- 20 Mhz-es kvarckristály,
- 2 db 22pF-os monolitikus kondenzátor,
- 8 db LED,
- 8 db 220 Ohmos ellenállás,
- 3 db 10 Ohmos ellenállás,
- 2 db nyomógomb,
- 1 db piezoelektromos hangszóró,
- 4 db közös anódos 7 szegmenses LED kijelző,
- 4 db 1 kOhmos ellenállás,
- 4 db NPN tranzisztor (pl. 2N3904, de lényegében bármilyen jó),
- egy 8 bites HD44780-kompatibilis szöveges LCD (2x16-os),
- egy tetszőleges potenciométer,
- egy PS/2-es anya csatlakozó,
- egy PS/2-es billentyűzet.

- Lehetőleg 5 V-os megbízható, zajmentes feszültségforrás.

A szükséges alkatrészek jelentős részét (ellenállások, tranzisztorok, LED-ek, nyomógombok stb.) beszerezhetjük kidobásra ítélt, működésképtelen elektronikai berendezésekből is, egyszerűen kiforraszthatjuk őket onnan, majd egy gyors kipróbálás után ugyanúgy használhatóak, akár az újak.

Egyes önállóan megoldható feladatok további alkatrészeket igényelhetnek (pl. LED, ellenállás, 7 szegmenses kijelző, tranzisztor).

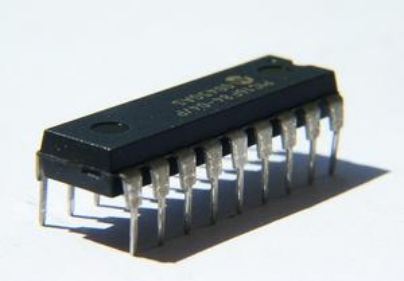
Különösen vigyázzunk a drága és könnyen sérülő alkatrészekre, így a PIC-re, az LCD-re és a PIC programozóra.

Rengeteg időt és vesződést takaríthatunk meg, ha veszünk egy normális PIC programozót, ami évek múltán is megbízhatóan fog működni – ezért gondoljuk át a beszerzését és olvassuk el figyelmesen az idevonatkozó fejezeteket.

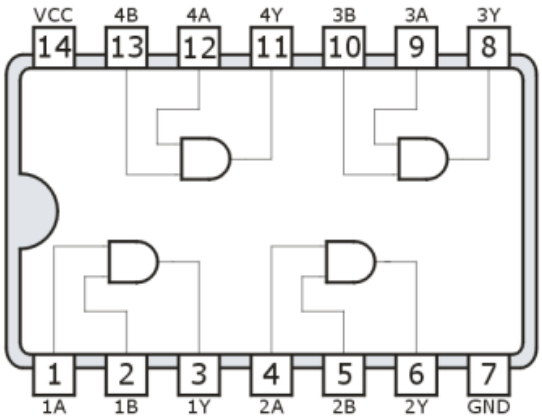
Ismerkedés a PIC-ekkel

Integrált áramkör: az IC

Idézzük fel egy kicsit a számítógép generációkat! A második generációról azt tudjuk, hogy megjelent a tranzisztor, és felváltotta az elektroncsövet (vákuumcsövet). De mi is az a tranzisztor? A *tranzisztor* egy igen sokrétű félvezető eszköz, amely többek között kapcsolásra is használható, azaz akkor engedi átfolyni két lába között az áramot, ha a harmadikon „ezt engedélyezik”.



Továbbhaladva, a harmadik generációban jelentek meg az *integrált áramkörök* (integrated circuit vagy IC), ami egy félvezető lapkán kialakított sok egymáshoz kapcsolódó tranzisztorból és egyéb áramköri elemből álló mikroelektronikai eszköz, amit valamilyen (többnyire fekete) tokba zárnak be és lábakat (kivezetéseket) helyeznek el rajta, hogy legyen lehetősége kapcsolódni a külvilághoz és az áramforráshoz. Az integrált



áramköröket úgy kell elképzelni, hogy ami korábban csak hosszadalmasan, több ezernyi alkatrész megtervezésével és összekötésével volt csak kivitelezhető (beleértve a tranzisztort is), belekerült egy kicsi tokba. Rengeteg cég gyárt különféle IC-ket, mindegyikük egy-egy speciális funkciót valósít meg. Például kaphatunk olyan IC-t, amiben a logikai ÉS műveletet valósították meg. Ezt úgy kell elképzelni, hogy ilyenkor ki van osztva, hogy melyik láb milyen célt szolgál, pl. van 4 db ÉS kapu, minden ÉS kapuhoz

tartozik két bemeneti láb és egy kimeneti láb. Ezen kívül még további lábak szolgálnak az IC táplálására. Az ábrán a 7408-as IC-t láthatjuk lábkiosztásával együtt, felülnézetből.

Ha időben még tovább megyünk, elérünk a negyedik generációig, ahol megjelentek a *mikroprocesszorok*. Ezek még tovább miniatürizált és sűrített integrált áramkörök, amiket pl. megtalálhatunk a számítógépekben, mobiltelefonokban. Itt az egyes mikroprocesszorok már többmilliónyi tranzisztort tartalmaznak, így igen összetettek és elég hatékonyak, valamint sok célra használhatóak.

A mikroprocesszoroknak van egy speciális változata, a *mikrovezérlők* családja (microcontroller, MCU vagy µC). Mi ilyen IC-kel fogunk foglalkozni. Számítógép esetében ugyanis az egész működés el van osztva: külön egység szolgál a végrehajtásra (CPU), külön az adatok és programok ideiglenes tárolására (memória), külön pedig tartós tárolásra (háttértárak), valamint külön a külvilággal való kapcsolattartásra (alaplapi kivezetések, portok, pl. USB). Mindez azért van így, hogy számítógépünk könnyen bővíthető, javítható legyen, valamint a megnövekedett igényeket csak specializálódott alkatrészekkel lehet kiszolgálni.

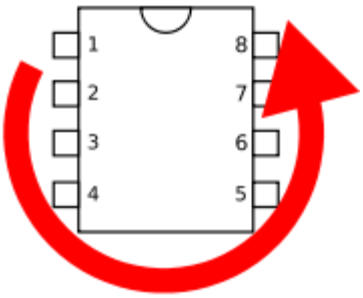
A mikrovezérlők ezzel ellentétben viszont integráltan, beépítve tartalmaznak mindent, azaz mind a feldolgozó egységet, mind az átmeneti memóriát, mind pedig az elmentett programot. A külvilággal való kapcsolattartásra pedig lényegében csak a lábakat fogjuk használni. Mivel egy ilyen eszköz képességei sokkal szerényebbek, ezért szükségünk lehet arra, hogy különböző alkatrészek segítségével bővítsük az eszköz funkcionalitását, pl. hiába változtatjuk a feszültséget az egyik kimeneti lábon, ha ezt nem lehet látni, ekkor jól jöhet egy világító dióda, egy LED azon az adott lábon. Hasonlóan köthetünk rá nyomógombokat és egyéb összetettebb alkatrészeket is. Ezek vezérlése pedig a mikrovezérlő feladata lesz, a benne található *program* fogja ezt vezérelni.

Az IC-k igen összetetten működnek. Az előbb leírtak alapján láthatjuk, hogy a bennük található kapcsolás fix, azaz nem változtatható, egy előre megadott kapcsolás alapján működnek. Ezt a működést gyártáskor úgymond rányomják az IC-re (azaz az alkatrészeket). Egy ilyen működés lehet például a már említett ÉS-kapu. Egy másik ilyen működés a mikrovezérlő, ahol egy processzor, memória és pl. Flash ROM is elhelyezésre került. *Flash ROM*-ot használnak az adatok tartós tárolására, pl. a pen-drive-okban is ilyen

található. Ebben az esetben a Flash ROM fogja tárolni a programot, ezt pedig akárhányszor újraírhatjuk (azaz magát a programot). A processzor az itt található utasításokat fogja végrehajtani.

Az említetteken kívül még rengetegféle IC és IC-programozási lehetőség van, pl. FPGA-k esetében a tranzisztorok összekapcsolását is mi határozhatjuk meg, programozáskor, ebben az esetben nincs processzor, hanem az egész áramkör eleve azt csinálja, amit mi akarunk, olyan mintha eleve így nyomták volna az IC-t. Mi ilyesmivel nem fogunk foglalkozni, csupán programozni fogunk.

Szerencsére manapság már az IC-k hobbisták számára is elérhetőek (mind árban, mind pedig kiskereskedelmi forgalomban), így mi is elkezdhetjük a velük való ismerkedést. Ha nem tudnánk programozni, valószínűleg játszanánk az említett ÉS kapukkal és társaival, de mivel már tudunk programozni, beléphetünk a profik világába: csinálja azt az IC, amit mi akarunk! Vezérelhetünk LED-eket, kijelzőket, akár LCD-t is, kelthetünk hangot, reagálhatunk nyomógombokra, de akár fogadhatjuk a billentyűzetről érkező jeleket is!



Érdemes megismerkedni a tokozással, ugyanis többféle van használatban. Kezdőknek javasolt tokozás a *DIP* (dual in-line package), mint a neve is jelzi, itt két párhuzamos sorban helyezkednek el a lábak. Ez a toktípus azért nagyon jó, mert könnyen kezelhető, viszont vigyázni kell vele, mert könnyen eltörhetjük a lábakat, ha nem vagyunk elég óvatosak vele. Ha lehet, használjunk külön IC-aljzatot, ez védi az eredeti IC-t, és mindig vékonyfejű csavarhúzóval szedjük ki az IC-t, mindkét végénél egyszerre, óvatosan emelve, de még jobb, egy jó IC csipesz használata. A lábakat számukkal szoktuk azonosítani, a számozás az óramutató járásával ellentétes irányban történik, és az első és az utolsó lábak a sorok végén találhatóak, köztük pedig egy kis bemélyedés jelzi a számozás kezdetét; néha az 1-es lábhoz szoktak rakni egy kis pontot.

Programozható IC: a PIC

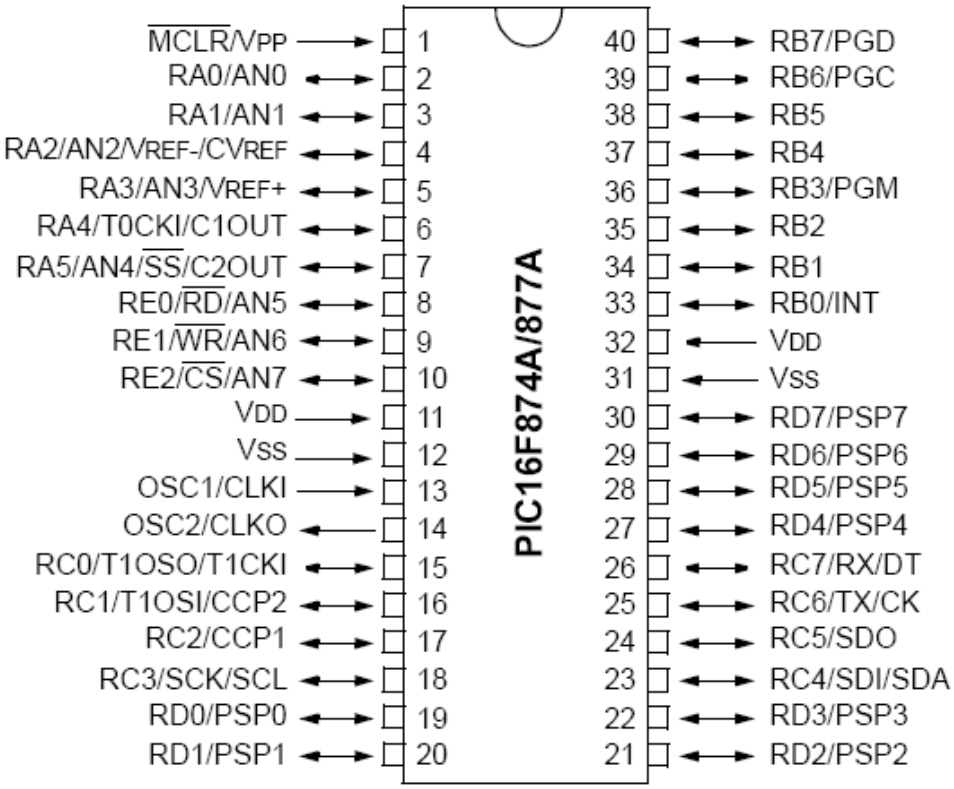
Mikrovezérlőket sok cég gyárt a világon, ezek közül mi a Microchip által gyártott PIC mikrovezérlőkkel (PICmicro) fogunk foglalkozni. A Microchipnek jó pár versenytársa van a világon, pl. Atmel, Infineon, Freescale, STMicroelectronics, Texas Instruments, Analog

Devices és Maxim Integrated Products – tehát egyáltalán nincs monopolhelyzetben a Microchip.

A *PIC* a Programmable Intelligent Computer rövidítése, lényege, hogy a Microchip elsősorban üres, program nélküli mikrovezérlőket gyárt, amikbe mi később belerakhatjuk a programunkat. Mint korábban volt róla szó, a program a Flash ROM-ban, helyezkedik el (habár a Microchip gyárt egyszer programozható és egyéb mikrovezérlőket is).

A több száz különböző PIC közül mi a 16F877 típusjelűvel fogunk foglalkozni. Ennek összesen 40 lába van, amiből 33-at tudunk a saját céljainkra használni. Használható a mi szempontunkból vele teljesen kompatibilis 16F877A is.

Ha megnézünk egy PIC típusjelét, ehhez hasonlót láthatunk: PIC16F877-20/P. A 16F877 a leglényegesebb, ez azonosítja be a PIC típusát. A kötőjel után olvasható a maximális



órajel, ebben az esetben ez 20 Mhz (ez a maximum ennél a típusnál), erről később még részletesebben lesz szó. A végén a /P pedig a tokozásra utal.

A PIC adatlapját töltsük le a Microchip honlapjáról, a http://www.microchip.com/ címről, ezt nagyon sokat fogjuk használni. Ha később belemerülünk a fejlesztésbe, ne feledkezzünk meg a hibajegyzékről sem (errata), amit időnként közzétesznek minden PIC-hez.

Ez a PIC egy közepes képességű (mid-range) PIC, azért fogjuk ezt használni, mert az ingyenesen elérhető eszközök közül ezt támogatják a legtöbben, valamint olcsó és a céljainknak tökéletesen megfelel. A PIC-ek különböző perifériákat tartalmaznak, többek között ezért is van ennyiféle eszköz. Ez a PIC perifériaként pl. többféle időzítőt, analóg-digitál átalakítót tartalmaz. Drágább PIC-ekben lehet akár USB vagy Ethernet interfész is.

Az IC lábkiosztását láthatjuk az ábrán. Első ránézésre ijesztőnek tűnhet, viszont a könyv végére érve mindegyik lábról tudni fogjuk, hogy melyiket mire lehet használni! Amit már most érdemes észrevenni, az az, hogy a legtöbb láb több funkciót lát el, ezek rövidítései megtalálhatóak a lábak mellett, és egy per jellel vannak elválasztva; erre tekinthetünk úgy is, hogy a lábaknak az éppen ellátott funkciótól függően különböző nevük is lehet. Például ha az RC6 lábról akkor beszélünk, ha pl. egy LED-et kötünk rá, viszont a TX-ről akkor, ha soros kommunikációra használjuk (akár a számítógépünk soros portjával). A lábak mellett levő nyíl pedig az áram folyásának irányát jelenti, ezt a digitális világban felfoghatjuk a jel irányának is, azaz kimeneti vagy bemeneti célt szolgál-e a láb.

A PIC fogyasztása rendkívül alacsony, kb. 1 mA, sőt, ezt még tovább csökkenthetjük, ha alvó állapotba rakjuk a PIC-et, erre egy külön utasítás szolgál.

Neumann- és Harvard-architektúra

Az *architektúrák* a számítógép működésének, felépítésének fajtáit jelentik. Két alapvető architektúra van, az egyik az általunk már megismert és a számítógépekben manapság is használt *Neumann-architektúra*, ami teljesen egészében a Neumann-elveket követi. Ennek a lényege, hogy a program és az adatmemória megegyezik, akár közvetlenül egymás mellett is elhelyezkedhetnek, ugyanabban a RAM-modulban (számítógépben gondolkodva). Ez persze korábban problémákat is okozott: rengeteg olyan Internetes

támadás volt régen, amikor a kiszolgáló program várt 100 bájtnyi adatot, helyette viszont kapott 5 kB-nyit és mivel nem ellenőrizte a bejövő adat hosszát, elkezdte fogadni a 100 bájtos tömbben, aminek az lett az eredménye, hogy az adat „kifolyt”, akár egy olyan memóriaterületre, ahol a következőleg meghívandó függvény lett volna. Így a támadó bármilyen kód lefuttatására kényszeríteni tudta a kiszolgálót. Ezt a végletekig leegyszerűsített példát ma már nem tudnánk eljátszani, ugyanis a mai operációs rendszerek és processzorok a kiosztott memórialapoknál figyelik azt is, hogy az kódot tartalmaz-e vagy adatot.

A másik megközelítés, amit a PIC-ekben is használnak, *Harvard-architektúra*, annyiban különbözik a Neumann-architektúrától, hogy a program és az adatmemória különbözik. A biztonság nem játszik szerepet ilyen alacsony szinten, csupán gazdaságossági megfontolásból döntöttek emellett az architektúra mellett.

Miért is? Neumann-architektúra esetében a memóriát bájtokra osztották. Emiatt minden utasítás egy vagy több bájtot foglal el. Kezdetben volt néhány egy-két bájtos utasítás a számítógépeinkben, manapság viszont már nem ritka a 4 bájtos hosszúságú elemi utasítás sem, azaz jellemzően változó hosszúak az utasítások. Harvard-architektúra esetében az adatmemória és a programmemória állhat különböző bitszélességű elemekből is. Így pl. az általunk használt PIC-ek esetében az adatmemória a szokásos bájt-alapú, 8 bites, a programmemória (a Flash ROM) viszont 14 bites, azaz minden elemi utasítás 14 bites, kivétel nélkül. Lehetne lehetőség itt is változó hosszúságú utasításokra, de a Microchip mérnökei céltudatosan választották meg ezt a 14 bites értéket.

A *szó* a különböző architektúrák egyik fontos jellemzője, minden a mikroprocesszor által kezelt egész érték általában szó méretű. A 32 bites számítógépeknél a szó mérete 32 bit, míg a 64 biteseknél 64 bit. PIC esetében kétféle szóméret van a különböző memóriatípusok miatt, itt a programmemóriabeli szóméret 14 bit, az adatmemóriabeli szóméret 8 bit.

Szinte minden mikroprocesszor tartalmaz beépített *regisztereket*, amelyek kis átmeneti tároló rekeszek, amin tipikusan a processzor a műveleteket végzi. Ezek mérete a legtöbb esetben megegyezik az adatmemóriabeli szóméretettel.

Egy *gépi utasítás* egy programmemóriabeli szót jelent, azaz egy konkrét alacsonyszintű utasítást, pl. adjon hozzá egy számot valamelyik regiszterhez. A *gépi kód* gépi utasítások sorozata, vagyis maga a program.

Fejlesztési lehetőségek a PIC-re

Láttuk, hogy a PIC önmagában nem képes semmire, valahogy kell rá írnunk egy programot. Ezt hasonlóan fogjuk megírni, mintha a számítógépre írnánk valamilyen programot. Az eddig használt környezetünk `.exe` kiterjesztésű futtatható fájlokat készített, ami mint láttuk, teljesen más architektúra, így az eddigi eszközöket nem használhatjuk. Bizonyára tapasztaltuk, hogy a számítógépre történő fejlesztést is sok nyelven végezhetjük, hasonló a helyzet a PIC-ek esetében is, vannak nyelvek, amivel gyorsabban haladhatunk és vannak, amivel lassabban, viszont jobban a mélyére járhatunk a dolgoknak.

A Microchip által javasolt eszköz az ingyenesen elérhető MPLAB (letölthető a http://www.microchip.com/ címről), ami beépítve támogatja az assembly nyelvet. Az *assembly* a gépi kódhoz álló legközelebbi nyelv, ami azt jelenti, hogy minden egyes gépi utasításnak egy assembly kulcsszó felel meg, amit további paraméterek követhetnek (például: add – szám hozzáadása egy memóriarekeszhez). Ebben igen körülményes és lassú programozni és igen könnyen véthetünk hibákat. A PIC igen szűk utasításkészlettel rendelkezik, összesen 35 utasítást támogat az általunk használt PIC, ezt igen könnyű megtanulni, viszont utána igen nehéz nagyobb és összetettebb programokat megírni ezek segítségével. A 14 bit természetesen arra van kihasználva, hogy az egyes utasításokhoz egyéb paramétereket is tároljon a program.

Sokkal kényelmesebb valamilyen magasabb szintű nyelv használata, pl. Basic, C vagy Pascal. Ezekre is lehetőségünk van, rengeteg cég gyárt különböző fordítókat és fejlesztői környezeteket a PIC-ekhez.

Érdekesség, hogy az általunk használt PIC például nem támogatja beépítve a szorzás műveletet, azaz assemblyből ezt nem érhetjük el, ezt nekünk kellene megírnunk (vagy felhasználnunk egy meglévő kódrészletet). Ha viszont a magas szintű nyelvek fordítóit használjuk, a környezet legenerálja nekünk a szorzás kódját, és mi ugyanúgy használhatjuk a szorzás műveletet a programunkban, mint bárhol máshol.

Fontos, hogy mivel mid-range PIC-ről van szó, a programmemória meglehetősen korlátozott, a 16F877 esetében ez 8k, ami itt nem 8 kilobájt, hanem 8 kiloszó, azaz 8×1024 utasítás (ugyanis egy utasítás felel meg egy programmemóriabeli szónak). Ez igen kevés, és assembly kóddal rendkívül hatékony kódot tudunk írni, viszont a magas szintű nyelvek pazarlóan bánnak a hellyel, így könnyen elérhetjük ezt a határt egy összetettebb projekt esetében.

Az adatmemóriája 368 bájt, aminek egy része konkrét célokra van fenntartva (ezek a *speciális célú regiszterek*), így nem mindet használhatjuk fel teljesen szabadon. A PIC-ek specialitása, hogy az adatmemória minden bájtját regiszternek hívják. Az általunk szabadon használható regiszterek az *általános célú regiszterek* (general purpose register, GPR), melynek mérete összesen 336 bájt.

További korlátozás, hogy a könnyű elérés érdekében több részre, ún. *bankokra* oszlik, és jellemzően a „nagy" adatszerkezetek nem nyúlhatnak át több bankon keresztül, így könnyen kaphatunk ilyen hibaüzeneteket a későbbiekben.

Fejlesztői környezet: mikroPascal

Az elérhető fejlesztői környezetek közül mi a mikroElektronika által készített mikroPascalt fogjuk használni, ami letölthető a http://www.mikroe.com/ címről. Ez a program shareware, ami jelen esetben azt jelenti, hogy a demó verziót ingyenesen letölthetjük, és időkorlát nélkül használhatjuk. A demó verzió egyetlen korlátozása, hogy a lefordított program mérete nem lépheti túl a 2k-t (a könyvben szereplő programok nem lépik túl ezt a határt).

A program 7.0-s verziója magyar nyelven egyelőre nem érhető el, de a programozási nyelv amúgy is angol és a környezetet nem fogjuk olyan mélyen használni, hogy ez problémát jelentsen. Egyedül a hibaüzenetek megértése jelenthet problémát az angolul még nem tudók számára. A program telepítése különösképpen nem bonyolult, egyedüli probléma, hogy a program nincs kellően felkészítve a több felhasználós környezetre. Egyrészt csak a telepítő programot futtató felhasználó számára hoz létre parancsikonokat, valamint indítás után folyamatosan ír az alkalmazás könyvtárába, így ez problémát jelenthet később (pl. Windows XP esetében, ha nem rendszergazdai fiókkal, vagy Windows Vista esetében kikapcsolt fájlrendszer-virtualizációval szeretnénk később futtatni a programot).

Megoldás, hogy telepítsük egy mindenki által elérhető könyvtárba vagy adjunk az alkalmazás könyvtárára megfelelő jogosultságokat. A telepítő az alkalmazás telepítése után felajánlja különböző PIC programozók és driverek telepítését, ezekre nincs szükségünk a program működéséhez.

A fejlesztői környezetből elérhető a Basic és C nyelvű változat is, amelyek nagyon hasonlítanak a Pascalos változatra és a licenszelésük is megegyezik, így akár ezeket is használhatjuk.

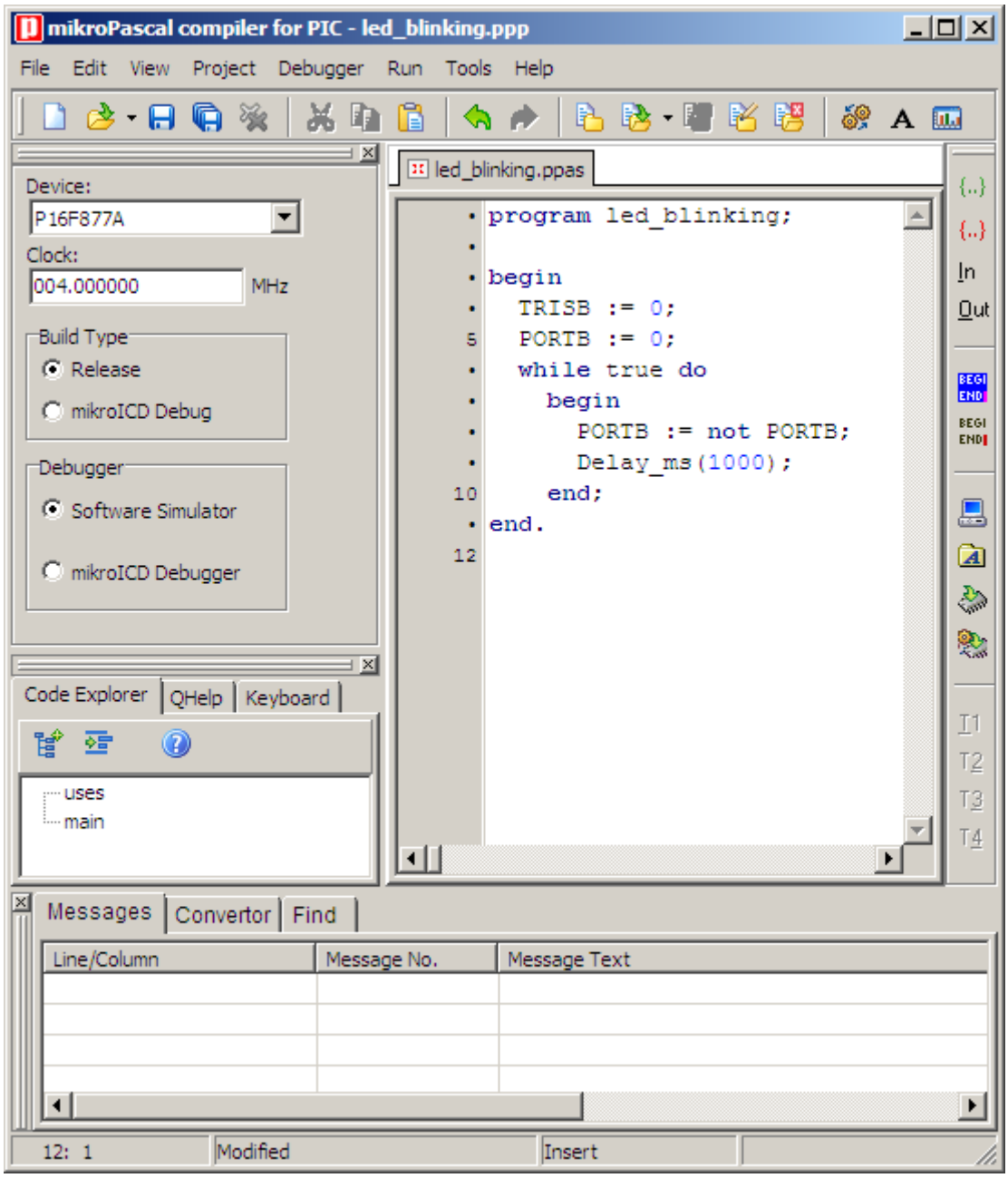
Fontos, hogy ezek a termékek nem hibabiztosak, így ne használjuk őket olyan célokra, ahol a fordítók hibás működése bármilyen kárhoz vezethet.

A programot elindítva láthatjuk a fejlesztői környezet fő ablakait. Ezekből számunkra a leglényegesebb az *editor*, ahol a programunk forráskódját fogjuk szerkeszteni. Itt ugyanúgy működnek a vágólap-műveletek, a kurzormozgató billentyűk, mint bárhol máshol, sőt a környezet az ún. szintaxis-kiemelést is támogatja, különböző színekkel segíti a programkód olvasását. Néhány hasznos funkciójára szeretném felhívni a figyelmet: az egyik a *code assistant*, amit bármikor előhívhatunk a CTRL-SPACE segítségével. Ez az eszköz felsorol minden függvényt, eljárást, változót és konstanst, amit az adott környezetben használhatunk. Amint beírjuk az adott függvény vagy eljárás nevét, majd leütjük a nyitó zárójelet, máris segítségünkre igyekszik a *parameter assistant*, ami segít a paraméterlista kitöltésében. Ezt bármikor később is előhívhatjuk a CTRL-SHIFT-SPACE segítségével. Ha bármelyik eljárásról vagy függvényről további segítségre van szükségünk, álljunk rá a nevére a kurzorral az editorban, majd nyomjuk meg az F1 billentyűt, ekkor megjelenik a mikroPascal súgója az eljárásra vagy függvényre vonatkozóan. A mikroPascal Libraries témakör alatt megtaláljuk a mikroPascal összes beépített eljárását és függvényét, csoportosítva: LCD kezelés, matematikai, string kezelés. Az editornak még sok további hasznos funkciója van, pl. ha egy eljárást szeretnénk írni, elég csak annyit beírnunk, hogy `proc`, majd nyomjuk le a CTRL-J-t.

Egy másik fontos ablak a *Messages*, itt találhatjuk majd a fordító által küldött üzeneteket, többek között a hibákat is. Szintén itt lesz a kód által felhasznált memória mérete is, ami érdekes adat, mivel a memória mérete korlátozott.

Bal oldalt, a *Code Explorer*ben az alkalmazásban deklarált függvények között tallózhatunk egyszerűen.

A program *projekteken* dolgozik, aminek a célja, hogy összefogja az általunk írt program forráskódjait és a hozzá tartozó beállításokat. Minden új programhoz új projektet fogunk létrehozni, nem írhatunk programokat anélkül, hogy azok ne tartoznának egy projektbe. A



projektfájlok, amelyek a projekt információit tartalmazzák, a `.ppp` kiterjesztést kapják. Egy projektbe egy vagy több forrásfájl is tartozhat, a Pascal forrásfájlok kiterjesztése `.ppas`. Különösen figyeljünk erre oda, mert ha a `.ppas` fájlt csak magában nyitjuk meg, nem fogjuk tudni lefordítani, ugyanis a fordítónak különböző egyéb információkra is szüksége van, pl. hogy milyen PIC-re szeretnénk fordítani, és ezeket a projektfájl tárolja. Mielőtt egy új projektet nyitunk meg, az előzőt zárjuk be, ugyanis a projekt fájljait nyitva hagyva a környezet, és így később gondot okozhat, hogy miért nem a kiválasztott forrásfájlhoz tartozó projekt fordul.

A fejlesztés során használhatjuk a beépített eljárásokat és függvényeket, ezek közül igen sok van, viszont a fordító mindig csak azokat fogja befordítani a programunkba, amelyeket éppen használtuk, így törekedve a legkisebb helyfoglalásra. Használhatunk saját változókat, de használhatjuk a PIC regisztereit is, amelyeket felfoghatunk úgy, mint beépített változókat. A regiszterek leírását megtaláljuk a PIC adatlapjában.

Programfejlesztés menete

Turbo Pascalban már megismertük a programfejlesztés menetét: megírtuk a programot Pascal nyelven, majd a fordítóprogram segítségével lefordítottuk gépi kóddá, ez lett az `.exe` kiterjesztésű fájl, majd ezt futtattuk.

PIC programozás esetében is hasonló a helyzet, itt is megírjuk a forráskódot Pascal nyelven, lefordítjuk a mikroPascal megfelelő parancsával, ezután pedig elkészül a gépi kód. Alapvető különbség, hogy itt nem `.exe` fájl keletkezik, hanem `.hex`. Ezt a fájlt természetesen csak a cél PIC-en tudjuk futtatni. A PIC-et szimulálhatjuk a számítógép segítségével, vagy futtathatjuk a programot egy igazi PIC-en. Ehhez viszont fel kell programozni azt (meg kell mondanunk neki valahogy, hogy mit csináljon) – ez esetben kicsit bonyolultabb a helyzet, mint a számítógép programozásánál. Mindkét futtatási módszerről később részletesen lesz szó.

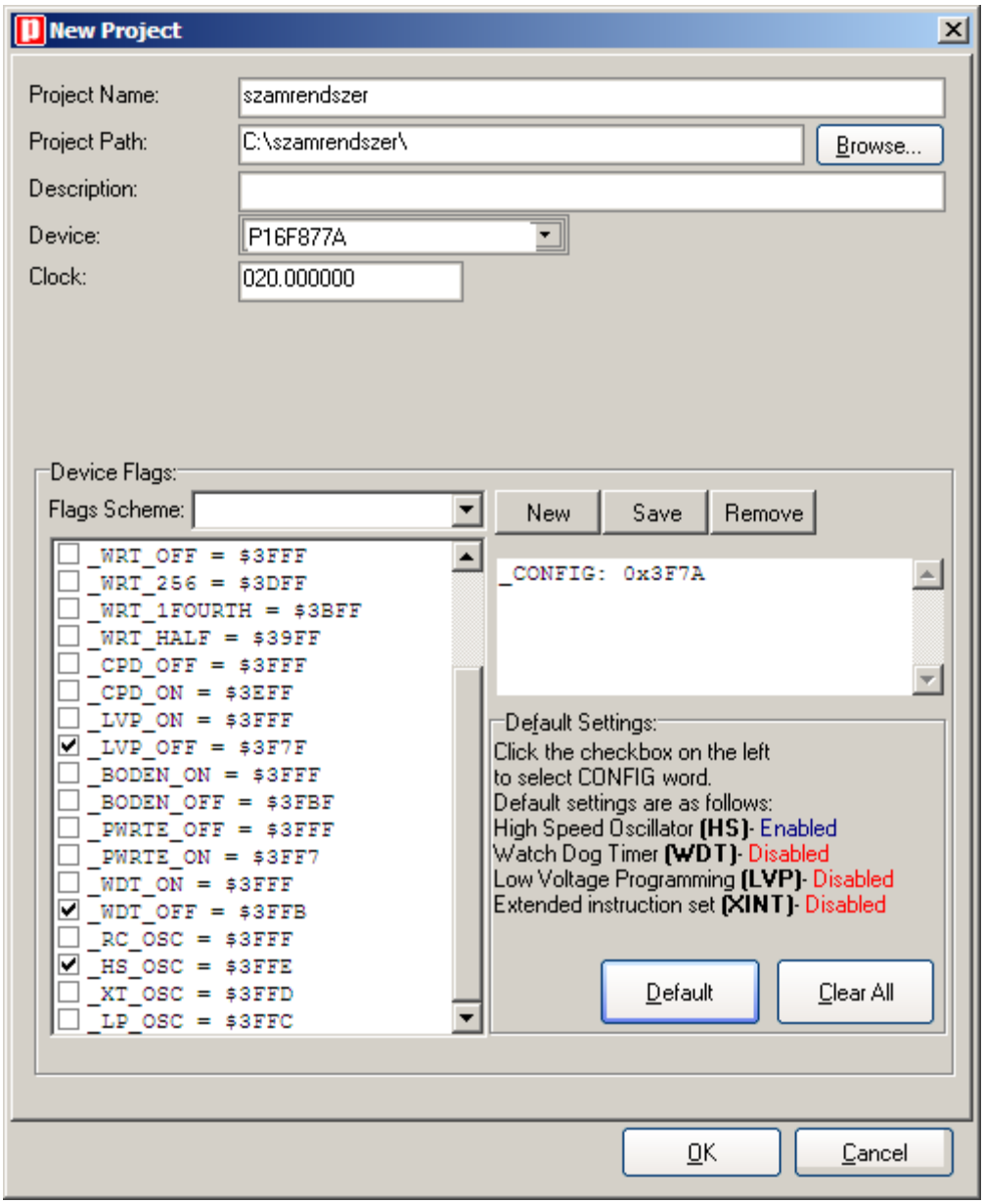
Amikor futtatjuk a programot, esetleg hibákat vehetünk észre benne, viszont ahhoz, hogy ezeket a hibákat kijavítsuk, először be kell határolnunk azokat, ami hagyományos programozásnál sem egyszerű feladat, pláne, ha sötétben tapogatódzunk. Szerencsére bizonyos mértékig PIC-ek esetében is elérhető a debuggolás, erről is részletesen lesz még szó.

Első programunk: számrendszer átváltó

Elérkezett az ideje, hogy megírjuk első programunkat, ami most nem a „Helló világ!”-ot kiíró program lesz, hiszen kijelzőt csatolni egy IC-re már bonyolultabb feladat (de később ezt is meg fogjuk valósítani). Ehelyett első programunk egyszerűen váltson át egy tízes számrendszerbeli számot binárissá! Az eredményt a PIC lábaihoz kapcsolt LED-eken fogjuk megjeleníteni. A program egyáltalán nem lesz olyan nehéz, mint amilyennek tűnik.

Indítsuk el a mikroPascalt, majd hozzunk létre egy új projektet (Project / New project)! Legyen a project neve `szamrendszer`, ezt a nevet fogja adni a `.ppp` fájlnak és a hozzá tartozó egyetlen `.ppas` fájlnak is. Adjuk meg projekt könyvtárát, ide fogja menteni ezeket a fájlokat. Válasszuk ki a Device legördülő listában a használt PIC-et (attól függően, hogy éppen mit használunk). Majd állítsuk be a Clocknál az órajelet, más néven működési frekvenciát, ez legyen 20 Mhz. Ez lesz gyakorlatilag a PIC sebessége (hasonlítsuk össze egy átlagos számítógép által használt 2-3 Ghz-cel). Ez a másodpercenkénti órajelek számát adja meg. Az általunk használt PIC egy elemi utasítást (egy assembly vagy ennek megfelelő gépi utasítást) 4 órajel alatt hajt végre, azaz egy másodperc alatt 20 Mhz-en üzemeltetve 5 millió elemi utasítást fog tudni végrehajtani.

A párbeszédpanel alján az ún. flageket tudjuk beállítani, ezek hatással vannak a PIC működésére, egyelőre hagyjuk őket az alapértelmezett beállításon, azaz nyomjunk rá a Default gombra, hogy ezeket betöltse, ekkor a jobb ablakban található `_CONFIG` felveszi a biteknek megfelelő értéket.



Az OK gombra kattintva a Pascal programok megszokott szerkezete fogad minket, a mi egyszerű programunk a következő lesz, gépeljük ezt be:

```
program szamrendszer;

begin
  TRISB:= 0;
  PORTB:= 170; // bináris 10101010
end.
```

A programunk egyszerűen a következőt csinálja: a decimális 170-et binárisként (ami 10101010) megjeleníteni a lábain. A PORTB egy-egy bitje egy-egy lábnak felel meg. A PORTB bitjei felváltva lesznek 0-k és 1-ek, így a PIC megfelelő lábai (konkrétan RB7-től RB0-ig) is ennek megfelelően lesznek alacsony vagy magas szintűek. Így ha ezekre a lábakra egy-egy LED-et kötünk a megfelelő módon, akkor a 170 bináris megfelelőjét fogjuk látni. A TRISB szerepéről később fogunk beszélni.

A program végét az end. jelzi, viszont felvetődik a kérdés, hogy mi fog ezután történni? A kérdés nem teljesen magától értetődő, hiszen a PIC működése nem áll le, mikroPascal esetében az end. nem csupán a program végét jelzi, hanem valójában ez egy üres végtelen ciklus, itt egy helyben fog tovább futni a programunk.

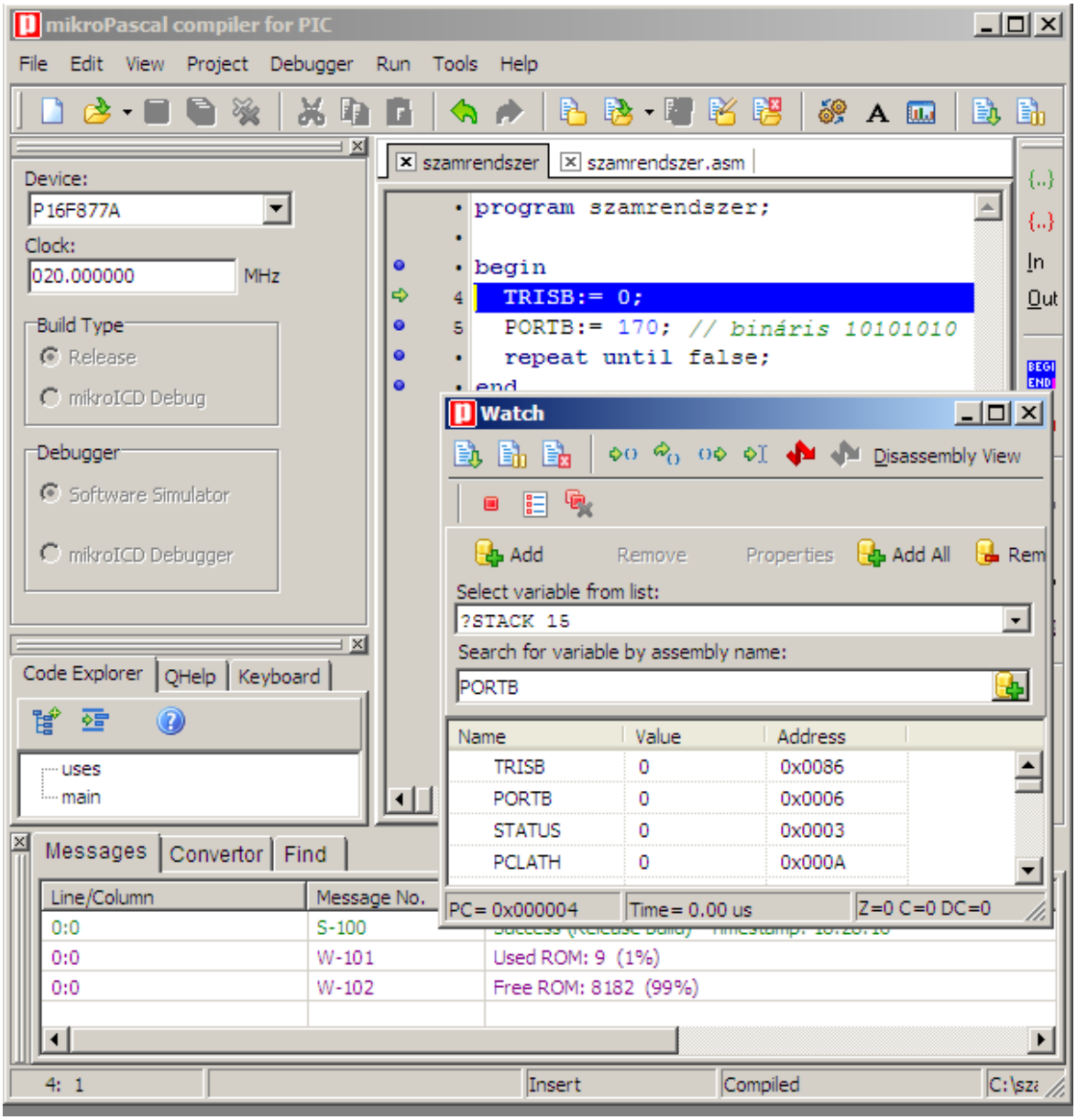
A mikroPascal támogatja az egysoros megjegyzéseket is, amik // jellel kezdődnek és a sor végéig tartanak.

Miután megírtuk a programot, fordítsuk le (Project / Build)! Ekkor többek között elkészül a PIC-be írható .hex fájl is. Ha sikeres volt a fordítás, a Messages ablakban megjelenik a Success üzenet. Azt is láthatjuk, hogy ez egyszerű programunk 9 szót foglal el a Flash ROM-ban.

Debuggolási lehetőségek a mikroPascalban

Ezek után szeretnénk kipróbálni a programunkat. Először nézzük meg a mikroPascal beépített szimulátorát, ami nyomkövetési (debuggolási) lehetőségekkel is rendelkezik. Indításához válasszuk a Run / Start debugger opciót. Ekkor megjelenik egy Watch ablak, ahol a változók és regiszterek értékeit tudjuk nyomon követni. Ha minden változót hozzá szeretnénk adni a Watch ablakhoz, kattintsunk az Add all gombra. A mi szempontunkból a PORTB lesz az érdekes, hiszen erre lesznek kötve a LED-ek. Ha továbblépünk a következő utasításra (Step into, F7), majd végrehajtjuk a PORTB-nek értéket adó utasítást is, láthatjuk a Watch ablakban, hogy a PORTB felveszi a megfelelő értéket (itt a Watch ablakban beállíthatunk akár bináris megjelenítést is). Azaz a program itt fogja csak bekapcsolni a megfelelő LED-eket.

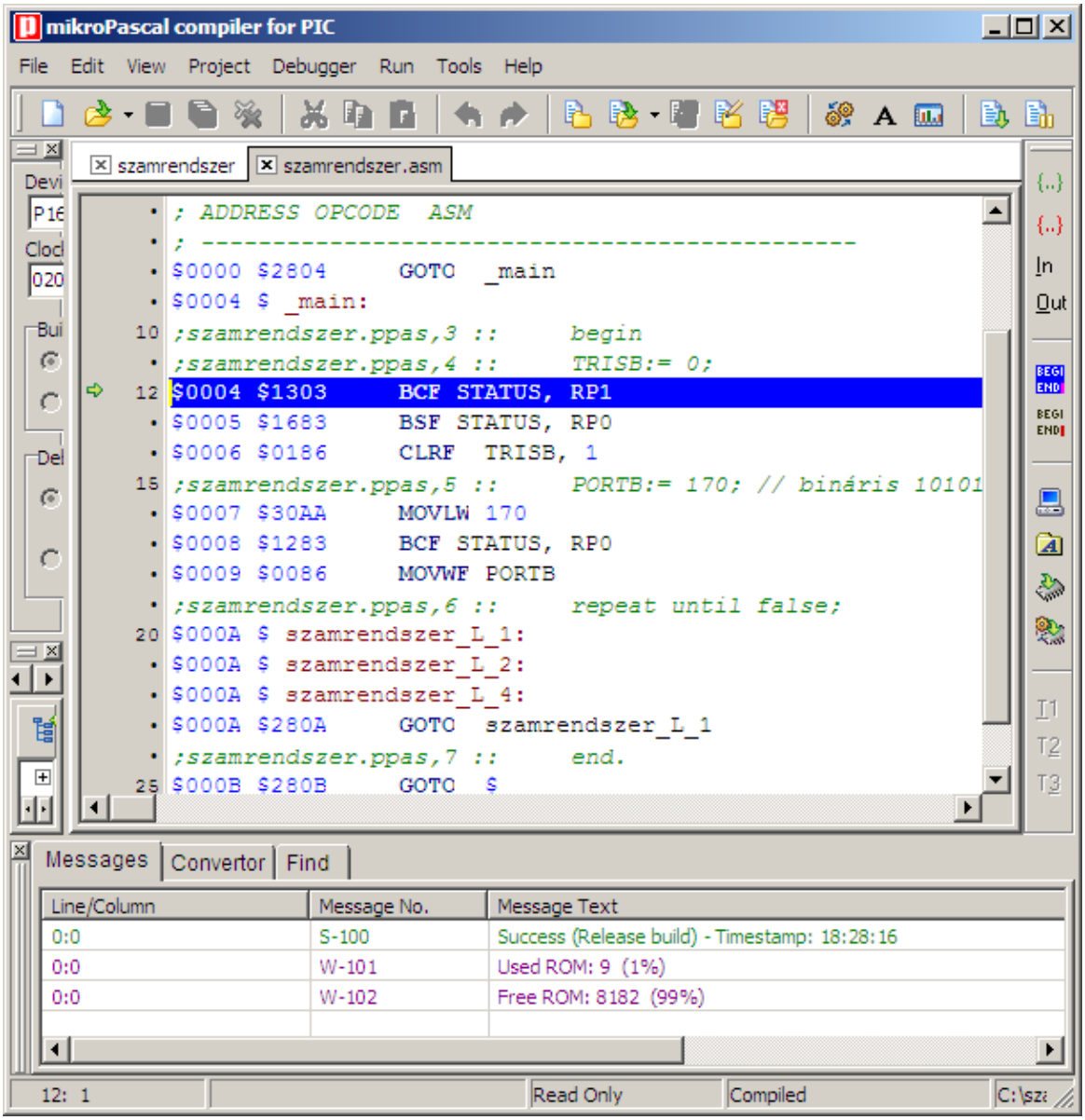
A beépített debuggerben lehetőségünk van mindenre, amit megszokhattunk a Turbo Pascalban, így elhelyezhetünk akár töréspontokat is (breakpoint), vagy pl. futtathatjuk a programunkat addig, amíg el nem ér egy töréspontot, és a program ekkor megáll.



Érdemes kipróbálni a Disassembly View opciót is, ekkor az editor átvált az assembly nyelvű kódra, ami a `.asm` fájlban található. Ezt a fordító a `.hex` fájl létrehozása közben generálta. Itt ugyanúgy nyomon követhetjük a program működését, mint az előbb, csak az assembly nyelvű megfelelőjét látjuk: azaz azt a 9 utasítást, amiből ez a program ténylegesen áll. Azt is láthatjuk, hogy melyik Pascal-beli sornak mi az assembly megfelelője. Érdemes egy későbbi, nagyobb projekt esetében is megnézni az assembly kódot, ugyanis idővel ez rettenetesen megnő (főleg akkor, amikor bonyolult

megvalósítású függvényeket fogunk használni pl. az LCD vezérléséhez). Aki már látott számítógépre írt assembly kódot, észreveheti, hogy itt kissé más utasítások vannak.

A program futását a Stop debugger gombbal állíthatjuk le.

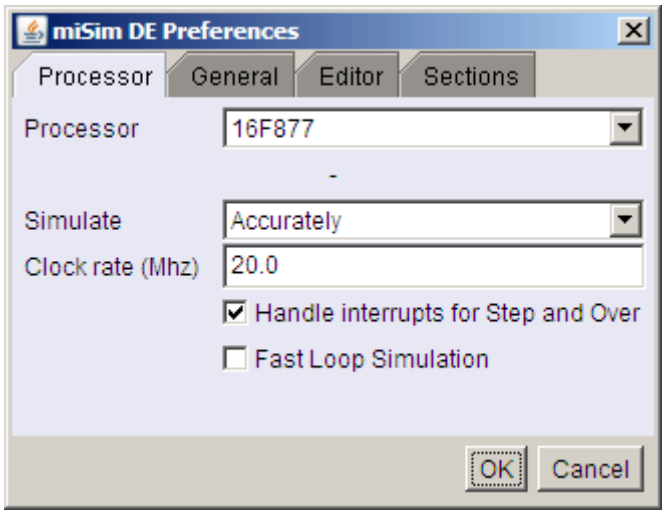


Egy másik szimulátor, a miSim

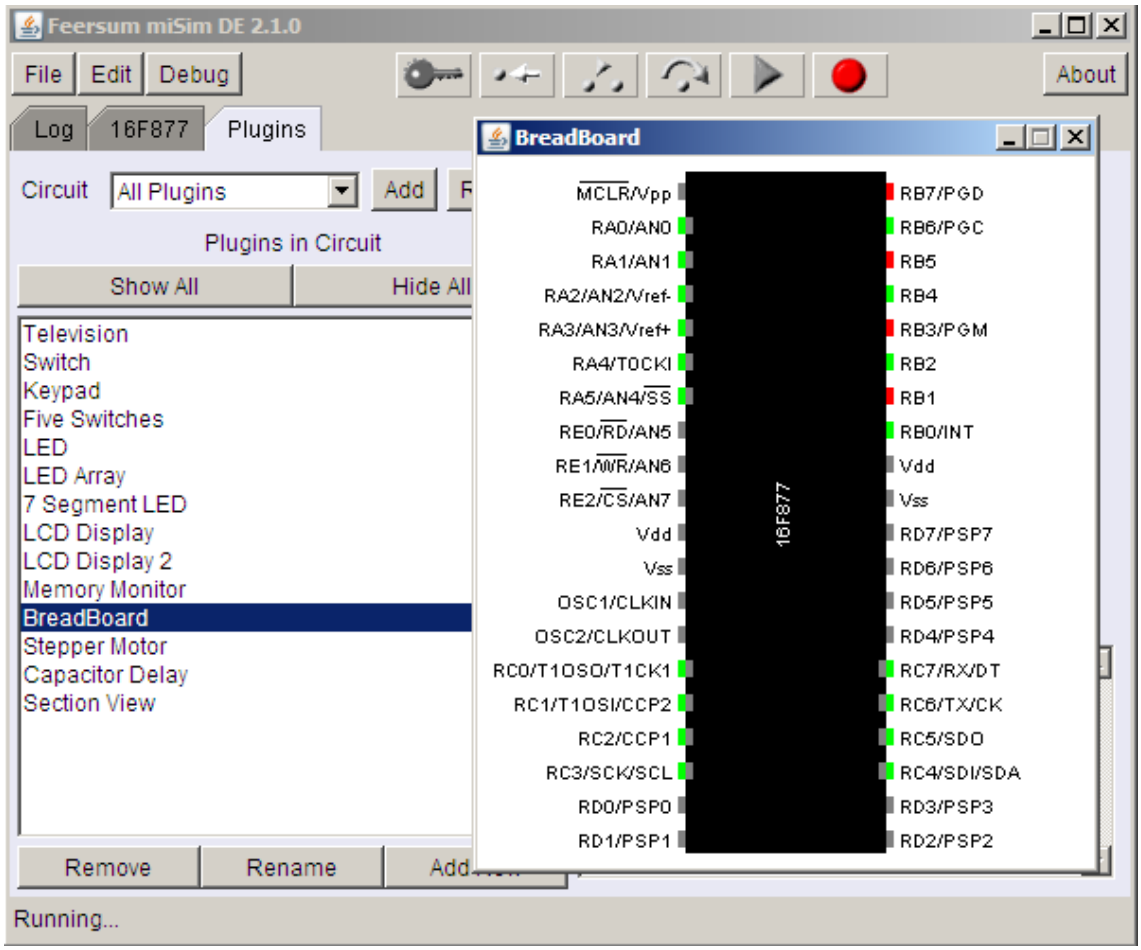
Mint láttuk a beépített szimulátor is elég jó, de nézzünk meg egy másik ingyenes eszközt, amely szintén sokat tud, sőt egyes esetekben még hasznosabb is lehet: ez a miSim, amely ingyenesen letölthető a http://www.feertech.com/misim/ címről. Futtatásához szükség van egy Java virtuális gépre, a Sun által fejlesztett futtatómotor letölthető a http://java.sun.com/ címről.

Ehhez az eszközhöz nem készítettek telepítőt, ugyanis platformfüggetlen. Az egyetlen feladatunk, hogy kitömörítsük, majd futtassuk a `misim.bat` fájlt.

Következő lépés a miSim szimulációs környezet bekonfigurálása, ehhez válasszuk az Edit / Preferences opciót! Válasszuk ki az általunk használt PIC-et (pl. 16F877) és az órajelet (20 Mhz)!



Miután ezt jóváhagytuk, betölthetjük az előbb elkészített `.hex` fájlt, a File / Load binary opció használatával. Miután betöltöttük, láthatjuk a `.hex`-ből visszafordított assembly forrást. Sajnos az eszköz nem támogatja a Pascal nyelvet, így debuggolásra elég körülményesen tudjuk csak használni (de erre is van lehetőség). Futtassuk le a programunkat, kattintsunk a zöld háromszögre! A programunk már fut, viszont nem látjuk az eredményét. A Plugins fület kiválasztva különböző segédeszközök közül választhatunk, válasszuk ki a BreadBoardot és kattintsunk a Show gombra! Ekkor megjelenik az általunk használt PIC, melynek lábai pirosak akkor, ha logikai egy van az adott lábon, és zöldek akkor, ha logikai nulla van az adott lábon.



Éles teszt

Az alábbiakban kipróbáljuk a megírt programunkat éles környezetben, azaz igazi PIC-en. Két feladatunk lesz: az egyik a PIC beprogramozása, majd az áramkör megépítése.

Beprogramozás

Ez a PIC fejlesztés egyik legkényesebb pontja, ugyanis nincs tökéletes PIC programozó. Az alábbiakban a legelterjedtebb programozókról olvashatunk.

Minden PIC programozó esetében alapszabály, hogy amíg nem vagyunk teljesen tisztában a PIC programozás menetével (ez némi utánaolvasást igényel), addig válasszuk el teljesen az IC-t az eredeti áramkörtől, szedjük ki onnan. Másik fontos dolog, hogy mivel igen drágák ezek a programozók, nagyon figyeljünk az IC behelyezésére.

Ha van rá lehetőség, programozás után mindig ellenőriztessük vissza a beírt adatokat! Ez főleg otthon barkácsolt PIC programozók esetében fontos, ugyanis előfordulhatnak hibák.

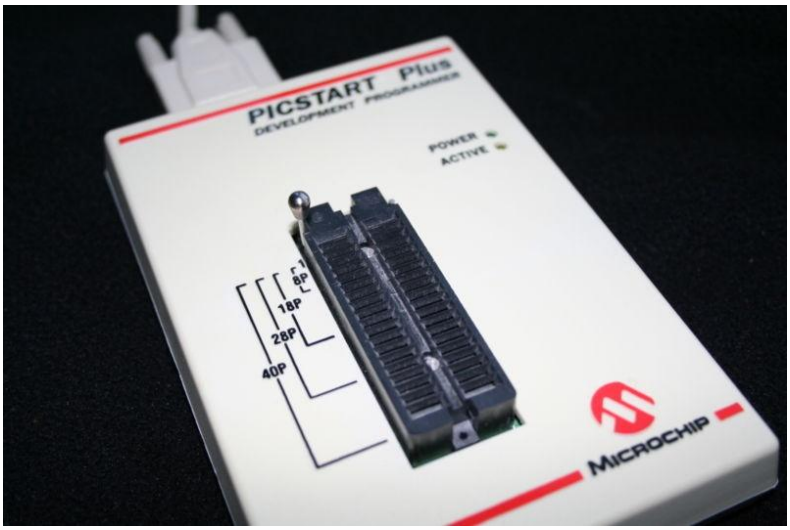
A bootloaderekről csak később lesz szó, itt csak a teljesen tiszta (frissen vásárolt) PIC-ek teljes programozásával foglalkozunk.

PICSTART PLUS

A PICSTART Plus a Microchip PIC programozója, ez a hivatalosan ajánlott programozó. Minden bizonnyal ugyanott beszerezhetjük, ahol a PIC-hez hozzájutottunk. Az ára elég borsos, 47000 Ft körül van. Előnye, hogy minden PIC-et tud programozni és a Microchip fejlesztői környezetéből, az MPLAB-ból vezérelhető a programozás, valamint a programozó eszköz firmware-jét is frissíthetjük, így támogatni fogja az újabb PIC-eket is.

Hátránya az árán kívül, hogy soros portos és rettenetesen lassú (8 kiloszó programozása 5-10 perc).

Bővebb információ a http://www.microchip.com/ és http://www.chipcad.hu/ címeken.



Otthon barkácsolt PIC programozók: párhuzamos és soros port

Aki nem szeretne sok pénzt költeni drága PIC programozókra, az otthon is készíthet egyet, erre is van lehetőség, rengeteg kapcsolási rajz lelhető fel az Interneten ebben a témában. Léteznek párhuzamos és soros portos programozók is. Közös jellemzőjük, hogy az eszközöknek csak egy nagyon szűk részhalmazát támogatják, és nem igazán megbízhatóak, valamint a velük kompatibilis szoftverek is elavultak sokszor.

Legfőbb problémájuk, hogy sokszor az egyik számítógépen tökéletesen működnek, a másikon viszont használhatatlanok. Ezt a problémát a soros vagy párhuzamos port vezérlője okozza, ugyanis az előállított vezérlő jel nagyon zajos, ami megzavarja a PIC

programozó áramkört, legtöbbször a beépített zajszűrőn is keresztülhalad, és a PIC rosszul értelmezi a jelet.

Áruk néhány ezer Ft, viszont nagyon sok idő megépíteni, és kiválasztani a megfelelő kapcsolást.

GTP-USB PLUS

Rengeteg harmadik fél által gyártott eszköz van, a GTP-USB PLUS egy olyan ezek közül, ami megbízhatóan működik és az árához képest igen sokat tud. Sajnos Magyarországon nem kapható, ára 60€, ami postaköltséggel együtt kb. 17000 Ft-nak felel meg.

Előnye, hogy a hozzá tartozó ingyenesen elérhető program automatikusan frissíti a firmware-ét, így a legújabb PIC-eket, valamint még Atmel gyártmányú mikrovezérlőket is és EEPROM-ot is tudunk vele programozni. A csatlakoztatása USB 2.0-n történik, egy 8 kiloszavas PIC beprogramozása csupán néhány másodperc. Automatikusan érzékeli a behelyezett PIC típusát és tud magyarul is a hozzá tartozó program.

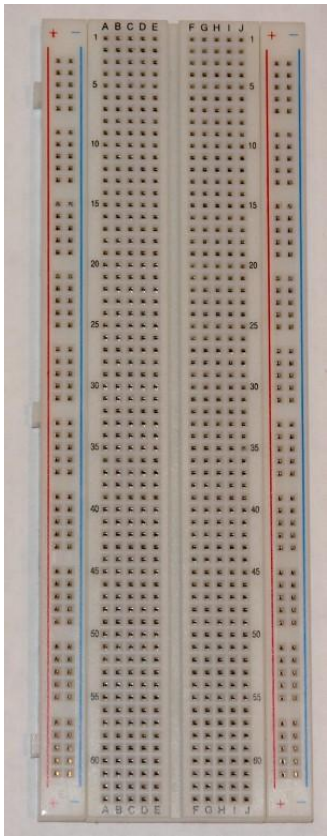
Hátránya, hogy IC aljzat nem jár hozzá: vagy nekünk kell az IC illesztéséről gondoskodni vagy külön kell megvásárolnunk az adaptert.

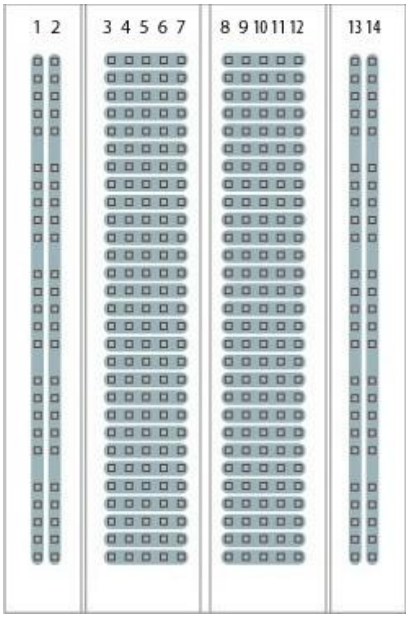
A PIC programozó érdekessége, hogy a számítógéppel való kapcsolattartást és az üres PIC programozását is egy PIC végzi, ennek a firmware-jét lehet valójában frissíteni.

Bővebb információ a http://www.winpic800.com/ címen érhető el.

Próbapanel

A *próbapanel* (breadboard, protoboard) célja, hogy gyorsan, forrasztás nélkül össze tudjunk rakni egyszerűbb áramköröket, ezzel elkerülve a hosszadalmas forrasztgatást. A tápfeszültséget a panel szélén szokás vezetni, az IC-ket pedig a középső oszlopokban helyezzük el. A panel megfelelő lukai között galvanikus összeköttetés van, ennek logikáját a jobb oldali ábrán láthatjuk. A próbapanelen való összeépítéshez természetesen szükségünk lesz egyéb vezetékekre is, célszerű olyan vezetékeket használni, amiket könnyen és határozottan be tudunk dugni az egyes lukakba, így merev, nem sodort vezetékek használata javasolt.





Feszültségforrás

Az általunk használt PIC-eket 2 V-tól 5,5 V-os feszültségen üzemeltethetjük. A digitális világban az elterjedt az 5 V-os működési feszültség, ha lehet, ezen üzemeltessük az elkészült áramköröket, például azért is, mert sok olyan IC és egyéb periféria van, ami csak 5 V-ról üzemel (pl. LCD). Az áramkör tervezésekor az üzemeltetési feszültséget is figyelembe kell venni, hiszen ez alapján lesz meghatározva az egyes ellenállások értéke, így később nem ajánlatos más feszültségről üzemeltetni az áramkört.

Két fő lehetőségünk van a tápellátásra: akkumulátorról (elemről) vagy transzformátorról. Az alábbiakban ezeket a lehetőségeket vesszük sorra.

Ha előbbit választjuk, tiszta, egyenletes, zajmentes és (az elemek lemerüléséig) megbízható áramforráshoz jutunk, viszont jobban hozzájárulunk a Föld szennyezéséhez.

Ha hálózati feszültséget transzformátorral alakítjuk 5 V-ra, különböző problémákkal kell szembenéznünk. Az első, hogy 5 V-os tápegységhez kicsit nehezebb hozzájutni. Viszont egy kis hozzáértéssel könnyen előállíthatjuk az 5 V-ot egy állítható feszültségű transzformátor segítségével: beállítjuk legalább 9 V-ra, majd egy stabilizátor IC (regulátor) segítségével előállítjuk a kívánt feszültséget. Ilyen IC pl. a 7805, 7905 jelű, amelyek különböző teljesítményű kiszerelésben kaphatóak. Ha ilyennel szeretnénk stabilizálni a feszültséget, nézzük meg az IC adatlapjában, hogy milyen összeállítást javasol a gyártó (kondenzátorok segítségével szokás zajmentesíteni az előállított feszültséget). Mivel ez egy kis utánajárást igényel, legjobb, ha beszerzünk egy 5 V-os tápegységet. A zajmentesítésre mindenképpen figyeljünk, mert az olcsó adapterek áteresztik a hálózatról érkező zajokat, és a PIC-ek igen érzékenyek a zajokra, ez lefagyásban vagy helytelen működésben jelentkezhet. Az ilyen hibák okát gyakorlatilag lehetetlen kideríteni, mert egyáltalán nem, vagy csak nehezen reprodukálhatóak.

Akármelyiket is választjuk, nagyon figyeljünk oda a polaritásra valamint a megfelelő feszültségre, ugyanis már 8-9 V tönkreteheti ezeket az IC-ket. Ezt leszámítva viszont nagyon strapabíróak a PIC-ek.

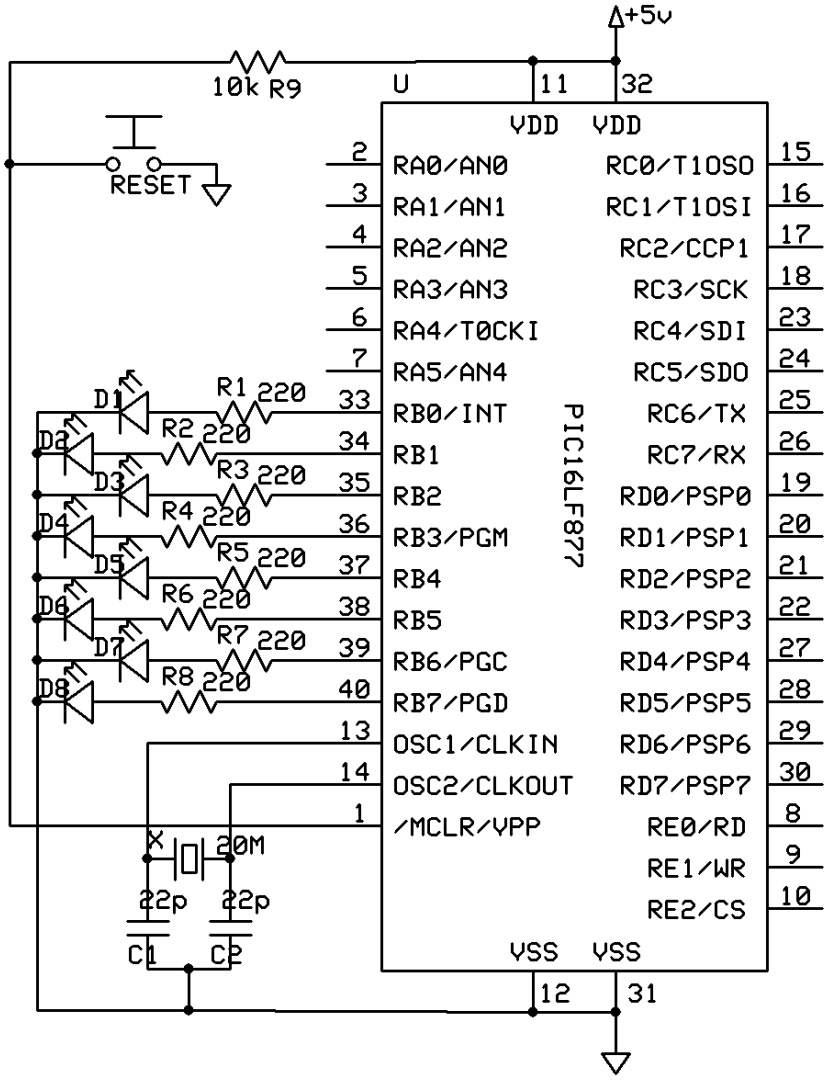
Minden kapcsoláson és adatlapon jelölik az áramforrást. A pozitív feszültséget többnyire *Vdd*-vel (Vcc-vel) jelölik és egy felfele mutató háromszöggel szokás ábrázolni. A negatív feszültséget vagy földelést *Vss*-sel (Vee-vel) jelölik és valamilyen lefele mutató háromszöggel szokás ábrázolni (a zárójelben levő jelölések más típusú tranzisztoroknál használatosak, de lényegében a jelentésük ugyanaz, hasznos, ha ezeket is ismerjük).

Megépítés

Építsük meg az áramkört egy próbapanelen! Helyezzük el a PIC-et a panel közepén, majd építsük köré a megfelelő alkatrészeket. Az RB0-RB7 lábakra kössünk egy-egy LED-et. Annak érdekében, hogy nehogy túl nagy áram folyjon a LED-en (és így tönkremenjen),

rakjunk vele sorosan egy-egy 220 Ohmos ellenállást. A LED egyik lába (katódja) a földön, a másik lába (anódja) pedig a PIC megfelelő lábán lesz, tehát ha ezen a lábon pozitív feszültség, pl. 5 V lesz, a LED világítani fog (a LED egy dióda, az áramot jellemzően normális körülmények között csak az egyik irányba vezeti: az anódtól (+) a katód (-) felé).

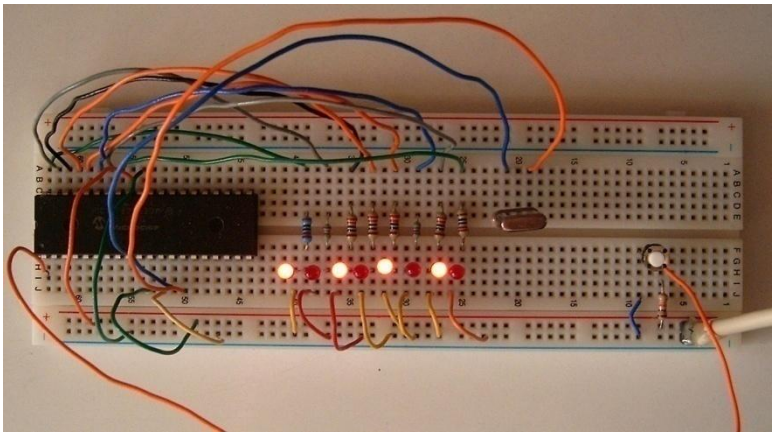
A PIC működéséhez viszont még kell néhány dolog, az egyik a Vss (-) és Vdd (+) lábak



megfelelő bekötése. Gyakorlatban azt tapasztalhatjuk, hogy elég csak egyet-egyet bekötni, mert úgy is működik, viszont szokjuk meg, hogy mindig mindegyiket rákötjük a táp megfelelő pólusára.

A következő fontos dolog a PIC órajelének biztosítása valamilyen oszcillátor segítségével. Mi ehhez egy 20 Mhz-es kvarckristályt fogunk használni. Használhatunk ettől eltérő rezgésszámú kristályt is, viszont ha azt szeretnénk, hogy a programunk ne fusson gyorsabban vagy lassabban a kelleténél, akkor fordításnál is ügyeljünk a helyes órajel

megadására (ez most még nem fontos, de később lényeges lesz). A kristály egyik lábát az OSC1, a másikat az OSC2 lábra kell kötni. Ha megnézzük a PIC adatlapját, láthatjuk, hogy ajánlott egy-egy kondenzátort kötni a kristály lábai és a föld közé (az adatlap szerint 20 Mhz-hez az ajánlott érték egy-egy 15-33 pF-os kondenzátor). Ez a kondenzátor lehetőleg ún. monolitikus kondenzátor legyen. Fontos tudni, hogy mivel minden kristálynak egyedi karakterisztikája van, itt nincs univerzális javaslat, az is lehetséges, hogy csak kondenzátorok nélkül fog használható órajelet adni és csak így fog működni az áramkörünk. Egyes oszcillátorok beépítve tartalmazzák ezeket a kondenzátorokat, ezeket onnan lehet megismerni, hogy nem két, hanem három kivezetésük van, és a középsőt a földre kell kötni (többnyire ezek kerámia tokban vannak).



Még egy fontos dolog maradt hátra, ez pedig az MCLR láb felkötése egy 10 kOhmos ellenálláson keresztül a pozitív tápfeszültségre. Ennek a lábnak reset funkciója van, ugyanis ha lehúzzuk a földre, akkor újraindítja a PIC-et (erre szolgál a RESET nyomógomb). Pontosabban csak akkor fog az futni, ha a pozitív tápfeszültségen van a láb. Így az itt lévő nyomógombot használhatjuk arra, hogy újraindítsuk a PIC-et.

Ha mindent jól csináltunk és már beprogramoztuk a PIC-et, ellenőrizzük le még egyszer a kapcsolást, majd kapcsoljuk be az áramkört, ha mindent jól csináltunk, minden második LED fog csak világítani.

Sajnos nem ritka, hogy első nekifutásra nem fog működni az áramkörünk. Próbáljuk meg elemről üzemeltetni, három sorba kapcsolt 1,5 V-os elem vagy négy 1,2 V-os akkumulátor tökéletesen megteszi (persze mint láttuk, akár 2 V-ról is megy). Másik igen kényes pont az oszcillátor és környéke: próbáljuk ki más kristállyal, más méretű kondenzátorokkal (akár nélkülük), hogy megy-e. Ellenőrizzük, hogy a PIC támogatja-e az általunk használni kívánt órajelet és hogy az MCLR lábon található ellenállás helyesen van-e bekötve.

Lábak elérése: a portok

Most nézzük meg részletesen, hogy hogyan működik ez az egyszerű program, hogyan érjük el a PIC lábait!

A külvilággal a kapcsolatot *portokon* keresztül tarthatjuk. Az általunk használt PIC-ben öt ilyen port van: A, B, C, D és E. Mindegyik porthoz tartozik a mikroPascalban egy-egy regiszter, amin keresztül a portot olvasni és írni tudjuk: `PORTA`, `PORTB`, `PORTC`, `PORTD` és `PORTE`. Minden porthoz tartozik a PIC néhány lába, mégpedig úgy, hogy a port egyes bitjei vannak kölcsönösen hozzárendelve bizonyos lábakhoz. Mindegyik port 8 bites, így maximum 8 láb tartozhat hozzá. A hozzárendelés teljesen egyértelmű, a következőképpen történik az általunk használt PIC esetében:

	7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
PORTA			RA5	RA4	RA3	RA2	RA1	RA0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
PORTE						RE2	RE1	RE0

Az egyes lábak irányát meg kell határoznunk: vajon LED-et fogunk meghajtani vele (kimeneti láb) vagy nyomógomb állapotát fogjuk lekérdezni vele (bemeneti láb). Ezek meghatározására tartozik minden porthoz egy-egy újabb regiszter, név szerint: `TRISA`, `TRISB`, `TRISC`, `TRISD` és `TRISE`. Ezek is 8 bites regiszterek, de itt a lábaknak megfelelő bitek azt határozzák meg, hogy az adott láb kimeneti (0) vagy bementi láb (1) legyen-e. Mi a programunkban a `TRISB`-nek 0 értéket adtunk, tehát a B port mind a 8 lába kimeneti lábként funkcionál. Ezt hívják a port konfigurálásának, ez után már használhatjuk a portot. Majd a programunkban valamilyen értéket adtunk a `PORTB`-nek, így megfelelő bitek beálltak 0-ra és 1-re.

A kimenetként funkcionáló lábaknál figyeljünk arra, hogy ne vegyünk le 25 mA-nél nagyobb áramot, mert az tönkreteheti a PIC-et (azaz mindig használjunk megfelelő méretű ellenállást, vagy ha szükséges tranzisztoros erősítést). Ha egy adott lábat

bemenetként konfigurálunk, akkor ezután szabadon olvashatjuk az értékét, később erre is fogunk látni példát.

Ha a regiszterek egyes bitjeire, mint boolean típusú változóra szeretnénk hivatkozni, a regiszter neve után pontot rakva majd a bit számát írva tehetjük meg, pl.: `PORTB.0`.

Jótanács: egyelőre ne használjuk az A portot semmilyen célra, mert azt nehezebb bekonfigurálni és használni.

Hogyan fog a logikai 0 és 1 megfelelni valamilyen feszültségnek? Ha 5 V-ról üzemeltetjük a PIC-et, akkor a szabvány TTL szerinti feszültségszinteket használja a PIC. A logikai 0-nak a föld, a logikai 1-nek pedig a pozitív tápfeszültség fog megfelelni. Természetesen a kettő között egy bemenet esetén kell lennie átmenetnek, ezt határozza meg a *TTL szabvány*, amelyet a PIC követ.

Összefoglaló kérdések, feladatok

Megismerkedtünk a PIC-ek alapjaival, valamint elkészítettük első működő mikrovezérlőt tartalmazó áramkörünket. Engedjük szabadra a fantáziánkat és játsszunk a PIC-cel! Kössük a LED-eket más lábakra, próbáljunk meg azon valamit megjeleníteni.

Az anyag elmélyítése érdekében válaszoljunk röviden az alábbi kérdésekre:

1. Mondjuk példát integrált áramkörökre!
2. Mit nevezünk mikrovezérlőnek? Miben különbözik egy számítógéptől?
3. Hogyan, milyen rendszer szerint számozzák az IC-k lábait?
4. Mi az a PIC?
5. Miben különbözik a Harvard- és a Neumann-architektúra?
6. Milyen különbségeket látunk a fejlesztői környezet kapcsán eddig az általunk használt környezetekkel (pl. Turbo Pascallal)?
7. Hány V-ról üzemeltethető a PIC? Mire kell figyelni a feszültségforrás biztosításakor?
8. Mire szolgál a `TRISB` regiszter?
9. Mit nevezünk egy PIC (be)programozásának?
10. Töltsük le és tanulmányozzuk a PIC adatlapját!

Órajel és késleltetés

Az órajel

A PIC-eket bármilyen órajelről üzemeltethetjük (akár 1,8432 Mhz-ről is). Az ehhez hasonló órajelek akkor fontosak, ha valamilyen speciális feladatra használjuk a PIC-et és nagyon fontos az időzítés. Ha másodperc alapú órát készítünk, valószínűleg a mi szemünknek is „szép” órajelet választunk, pl. 1 Mhz. Az általunk használt kvarckristályok igen pontosak, gyakorlatilag minden elemmel működő órában is ilyet használnak.

Kristály helyett használhatnánk RC oszcillátort is, amit egy ellenállás és egy kondenzátor segítségével összeépített rezgőkör alkot. Sőt, vannak PIC-ek, amik beépített oszcillátort is tartalmaznak (az általunk használtban nincs beépítve ilyen).

Flagek

A PIC programozásánál láttuk a `_CONFIG` értéket, ami a flageket tárolta. Nézzük meg, melyek a fontosabb beállítások:

- Itt kell megadnunk a használt oszcillátor típusát is: LP-t 200 kHz-ig, XT-t ha kvarckristályt használunk 100 Hz-től kezdődően és 4 Mhz-el bezárólag, illetve HS-t, ha 4 Mhz feletti kristályunk van.
- Meg tudjuk adni, hogy a PIC védje-e a beprogramozott kódot (code protection, CP). Ha ez be van kapcsolva, akkor semmilyen PIC programozóval sem tudjuk kiolvasni a kódot, csak igen drága laboratóriumi eszközökkel.
- Brown-out detect (BOR) áramkörnek a feladata, hogy ha a feszültség egy bizonyos szint alá csökken, automatikusan reseteli a PIC-et.
- Power-up timer (POR) áramkörnek az a feladata, hogy amíg az oszcillátor el nem éri a kívánt rezgésszámot (azaz be nem indul), a PIC ne induljon be.
- Watchdog timer áramkörnek az a célja, hogy automatikusan újraindítsa a PIC-et, ha az lefagyott. Ha bekapcsoljuk ezt a funkciót, folyamatosan meg kell hívnunk egy speciális eljárást, mégpedig a `ClrWdt`-t: ez jelzi a watchdog timernek, hogy még normálisan fut a program. Működését tekintve lenullázza a watchdog timert, ami a program futásával párhuzamosan a háttérben automatikusan számol. Amint elér egy meghatározott értéket, reseteli a PIC-et.

Karácsonyfa villogó készítése

Módosítsuk előző programunkat úgy, hogy villogjon! A program kódja a következő:

```
program villogo;

begin
  TRISB:= 0;
  PORTB:= 170; // bináris 10101010
  repeat
    PORTB:= not PORTB;
    Delay_ms(1000);
  until false;
end.
```

Ha megnézzük a programunkat, láthatjuk, hogy bekerült egy végtelen ciklus, abba pedig két utasítás. Az első bitenként tagadja a `PORTB` értékét, azaz ha az értéke előzőleg bináris `10101010` volt, akkor `01010101` lesz, ha pedig `01010101` volt, akkor újra `10101010` lesz. Láthatjuk, hogy nem probléma, hogy a port kimeneti portként van konfigurálva, ugyanúgy tudjuk olvasni az értékét (azaz azt, amit utoljára beírtunk neki, azt vissza tudjuk olvasni).

A második utasítás pedig egy várakozás, ahol a PIC a paraméterben megadott számú ezredmásodpercet fog várakozni. Jelen esetben ez 1 másodperc lesz. Ezután elölről kezdődik a ciklus.

Nézzük meg a debuggert fordítás után, majd váltsunk át az assembly kódra. Nézzük meg, hogy a `Delay_ms` eljárás egy igen hosszú kódra fordult le:

```
...
;villogo.ppas,8 ::          Delay_ms(1000);
$000B   $301A               MOVLW     26
$000C   $00FC               MOVWF     STACK_12
$000D   $30FF               MOVLW     255
$000E   $00FB               MOVWF     STACK_11
```

```
$000F   $30FF              MOVLW     255
$0010   $00FA              MOVWF     STACK_10
$0011   $0BFC              DECFSZ    STACK_12, F
$0012   $2814              GOTO $+2
$0013   $281B              GOTO $+8
$0014   $0BFB              DECFSZ    STACK_11, F
$0015   $2817              GOTO $+2
$0016   $281A              GOTO $+4
$0017   $0BFA              DECFSZ    STACK_10, F
$0018   $2817              GOTO $-1
$0019   $2814              GOTO $-5
$001A   $2811              GOTO $-9
$001B   $307F              MOVLW     127
$001C   $00FB              MOVWF     STACK_11
$001D   $30FF              MOVLW     255
$001E   $00FA              MOVWF     STACK_10
$001F   $0BFB              DECFSZ    STACK_11, F
$0020   $2822              GOTO $+2
$0021   $2825              GOTO $+4
$0022   $0BFA              DECFSZ    STACK_10, F
$0023   $2822              GOTO $-1
$0024   $281F              GOTO $-5
$0025   $3059              MOVLW     89
$0026   $00FA              MOVWF     STACK_10
$0027   $0BFA              DECFSZ    STACK_10, F
$0028   $2827              GOTO $-1
$0029   $0000              NOP
$002A   $0000              NOP
;villogo.ppas,9 ::         until false;
...
```

Szerencsére ezt is készen nyújtotta nekünk a mikroPascal, nem nekünk kellett leprogramozni a várakozást. Ez a várakozás egyébként függ az órajeltől, és ezért fontos,

hogy pontosan megadjuk a projekt beállításainál. Az itt megadott órajel ugyanis közvetlenül a `.hex` fájlba sem kerül bele, csak a `Delay_ms`-hez hasonló eljárásoknak van szükségük erre az információra.

Most próbáljuk ki a programunkat a miSim szimulátorban! Azt fogjuk tapasztalni, hogy a lábak felváltva villognak, a karácsonyfa-izzókhoz hasonló rend szerint.

Ezek után próbáljuk ki az új programunkat éles környezetben! A kapcsolás ugyanaz, mint az előző, csak a programot kell újra beégetni a PIC-be. A PIC programozáshoz mindig távolítsuk el a PIC-et az eredeti áramkörből!

Miután sikeresen kipróbáltuk, cseréljük ki a próbapanelen a 20 Mhz-es kristályt kisebbre, pl. 10 Mhz-esre! Azt fogjuk tapasztalni, hogy a program ugyanúgy működik, csak kétszer olyan lassú (mivel felére csökkentettük az órajelet).

Hangkeltés

Most LED helyett kapcsolgassunk egy piezoelektromos hangszórót, csak kicsit gyorsabban, úgy, hogy az hangot adjon ki. Programunk a következő lesz:

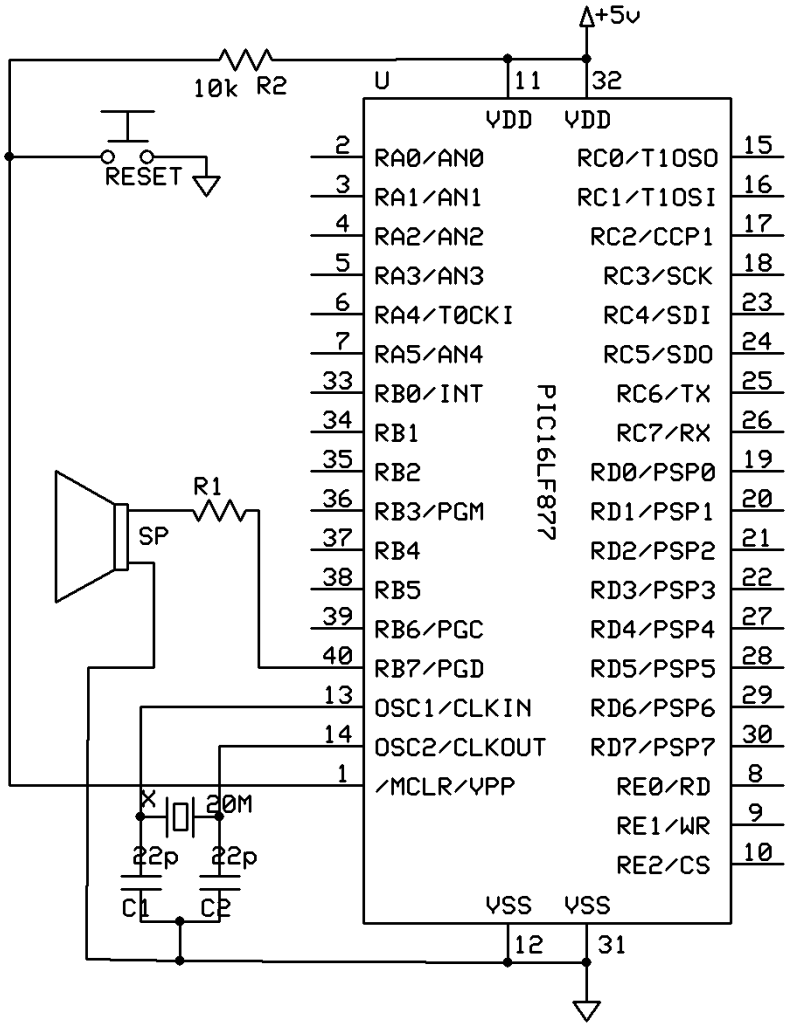
```
program hang;
var
  i: byte;

begin
  TRISB.7:= 0;
  repeat
    // kb. 500 Hz 500 ms-ig
    for i:= 1 to 250 do begin
      PORTB.7:= 1;
      Delay_ms(1);
      PORTB.7:= 0;
      Delay_ms(1);
    end;
    // 500 ms várakozás
    Delay_ms(500);
```

```
  until false;
end.
```

Az RB7 lábra fogjuk kötni a hangszórót. Az alkalmazott hangszórótól függően válasszuk meg a vele sorosan kapcsolt ellenállást, mégpedig úgy, hogy ne folyjon 25 mA-nél nagyobb áram, mert az tönkreteheti a PIC-et.

Nézzük meg, hogyan működik a programunk. Először is, bekonfiguráljuk az RB7 lábat kimeneti lábként. Majd a program magját képező végtelen ciklusban felváltva adunk egy kb. 500 Hz-es hangot fél másodpercig, majd várakozunk fél másodpercig a `Delay_ms` eljárással.

A hangkeltést egy ciklussal végezzük, mégpedig úgy, hogy gyorsan változtatjuk az RB7 láb értékét 0 és 1 között, ezáltal gyors rezgésre és hangkeltésre kényszerítve a hangszórót.



Az eredmény egy másodpercenként egyet pittyegő áramkör.

Nyomógomb kezelése

Következő programunkban egy nyomógombot fogunk kezelni. A nyomógomb egy LED-et fog vezérelni, mégpedig azt szeretnénk, hogy a következő módon: kezdetben a LED nem világít. Ha megnyomjuk a gombot, folyamatosan ég. Ha megint megnyomjuk, elalszik, majd újra bekapcsolhatjuk a gomb újbóli megnyomásával és így tovább. Erre a feladatra a programunk a következő:

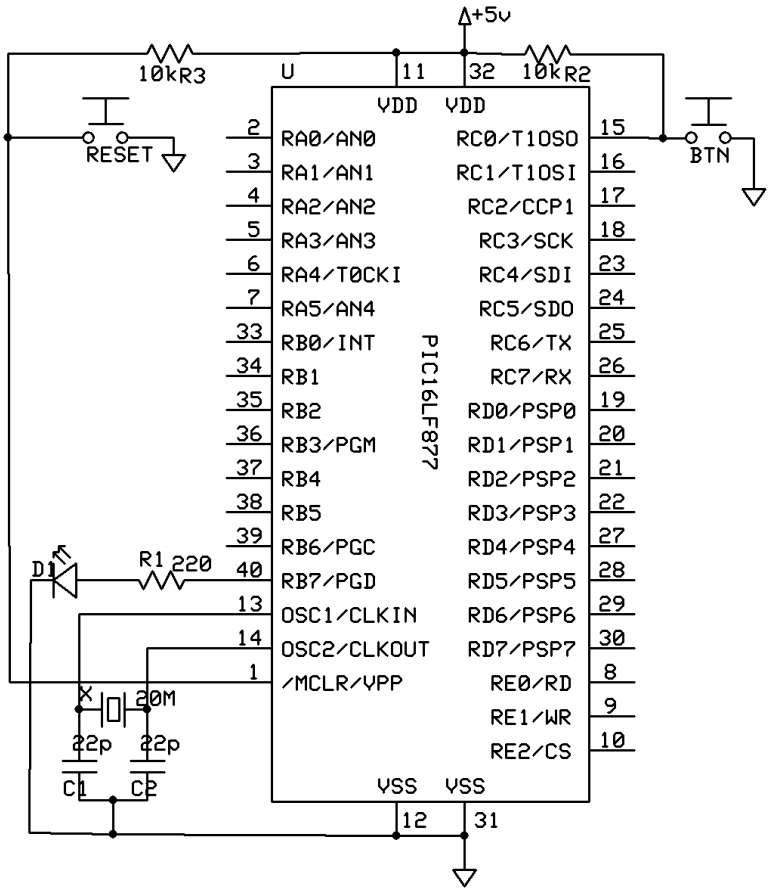
```
program nyomogomb;
var
  lenyomva: boolean;

begin
  TRISB.7:= 0; // a LED
  PORTB.7:= 0;
  TRISC.0:= 1; // a nyomógomb
  repeat
    // lenyomást vizsgálunk
    if not lenyomva and PORTC.0 = 0 then
      lenyomva:= true;
    // felengedést vizsgálunk
    if lenyomva and PORTC.0 = 1 then begin
      lenyomva:= false;
      // átkapcsoljuk a LED-et
      if PORTB.7 then
        PORTB.7:= 0
      else
        PORTB.7:= 1;
    end;
  until false;
end.
```

A portok bekonfigurálása után, a program magjában folyamatosan vizsgáljuk a nyomógomb állapotát, a port állapotának olvasásával. Az áramkörbe épített nyomógombunk nyugvó állapotban pozitív tápfeszültségen van, így nyugvó állapotban az RC0 láb logikai 1-et olvas, ellenben ha megnyomjuk a gombot, lehúzza a földre és a lábon logikai 0-t olvashatunk.

Az első feltétel a lenyomást vizsgálja, ha a gombot lenyomták, akkor ezt megjegyezzük. A második feltételben a felengedést vizsgáljuk, ha ez megtörtént, megint megjegyezzük, majd negáljuk az RB7 láb értékét, ezzel átkapcsolva a LED-et. Nem véletlen, hogy nem a `not` operátorral negáltunk, erről még később lesz szó.

Ahogy teszteljük a programot, szépen működik, viszont azt vesszük észre, hogy néha mintha nem venné figyelembe a gomb megnyomását. A hiba nem a programunkban, hanem a nyomógomb működésében keresendő.

Egy nyomógomb lenyomásakor két fémes felület érintkezik, ami legtöbbször nem egyenletes, így a legtöbb nyomógomb benyomáskor és elengedéskor berezeg, azaz nem



egyszerűen vált állapotot, hanem egy rövid idő erejéig oda-vissza pattog a két állapota között. Ha a nyomógombról közvetlenül hajtanánk meg egy LED-et, úgy, hogy ha lenyomjuk a nyomógombot világít a LED, ha felengedjük, nem, akkor a szemünk ezt nem venné észre, mivel annyira rövid idő alatt zajlik le mindez. Ellenben a 20 Mhz-en futó mikrovezérlő ezt észreveszi. Ebből kifolyólag az egy lenyomást esetleg két lenyomásnak érzékeli, emiatt a LED-et kétszer kapcsolja át, viszont ez olyan gyorsan történik, hogy mi csak azt vesszük észre, hogy nem változott a LED állapota. Természetesen az is lehet, hogy a lenyomást nem kettő, hanem akár még több lenyomásnak érzékeli.

Használhatunk jobb nyomógombokat, de ez a probléma mindig előfordulhat, így jobb, ha szoftverből oldjuk meg ezt a problémát, ezt hívják pergésmentesítésnek.

Nyomógomb kezelése jobban: pergésmentesítés

Az alábbi program már megoldja a problémát:

```
program nyomogomb2;
var
  lenyomva: boolean;

begin
  TRISB.7:= 0; // a LED
  PORTB.7:= 0;
  TRISC.0:= 1; // a nyomógomb
  repeat
    // lenyomást vizsgálunk
    if not lenyomva and Button(PORTC, 0, 1, 0) then
      lenyomva:= true;
    // felengedést vizsgálunk
    if lenyomva and Button(PORTC, 0, 1, 1) then begin
      lenyomva:= false;
      // átkapcsoljuk a LED-et
      if PORTB.7 then
        PORTB.7:= 0
      else
```

```
        PORTB.7:= 1;
    end;
  until false;
end.
```

Programunk maradt ugyanaz, egyedül a feltétel második része változott, nem egyenlőséget vizsgálunk, hanem egy beépített `Button` függvényt használunk. A függvény az első és második paraméterében meghatározott lábra kötött nyomógomb állapotát vizsgálja, ez jelen esetben a C port 0. lába, azaz az RC0. A negyedik paraméter határozza meg azt, hogy a nyomógomb milyen logikai érték esetén lesz aktív, a `Button` függvény ekkor fog logikai igazzal visszatérni. Így amikor a lenyomást vizsgáljunk az első feltételben, az utolsó paraméterben 0-t adunk meg (hiszen megnyomáskor a földre húz le a nyomógomb). Eltároljuk a lenyomást, majd várakozunk felengedésig, ezt egy újabb feltételben vizsgáljuk. Itt az utolsó paraméterben már 1-et adtunk meg, hiszen a felengedést szeretnénk vizsgálni.

A lényeg pedig a harmadik paraméterben található, itt adjuk meg ezredmásodpercben, hogy mennyi ideig várakozzon a függvény (a `Delay_ms`-hez hasonlóan), ezáltal kiküszöbölve a pergést. Figyeljünk arra, hogy ekkor ez a függvény itt 1 ms-ig várakozni fog! Ha sok gombot kezelünk, erre különösen kell ügyelni. Ha a gombok állapotát egymás után vizsgáljuk, megoldás lehet, hogy csak az utolsó `Button` hívásnál adunk át egy `>0` értéket.

Ha kipróbáljuk a programot, már tényleg tökéletesen működik. Pergésmentesítéssel csak akkor bonyolítsuk a programunkat, ha valóban problémát okoz az effektus!

Összefoglaló kérdések, feladatok

Ellenőrző kérdések:

1. Mi az órajel, milyen szerepe van, milyen órajelű kvarckrisztály oszcillátort használunk a PIC-ben?
2. Mit nevezünk flagnek, melyek a fontosabb flagek az általunk használt PIC-ben?

A feladatok során jobban megismerkedtünk a PIC felépítésével és az általunk használt fejlesztői környezettel. A gyakorló feladatok célja a megszerzett tudás elmélyítése.

1. Nézzünk utána a súgóban, hogy milyen `Delay_` kezdetű függvények érhetőek még el! Próbáljuk meg a nevük alapján kitalálni, hogy ezek milyen célt szolgálnak! Hogyan tudjuk közvetlenül a fejlesztői környezetből hozzájutni ehhez az információhoz?
2. Készítsünk olyan 8 LED-ből álló villogót, amelyen minden másodpercben a következő LED világít. Hogyan tudjuk ezt a feladatot a legkevesebb kóddal, matematikai ismereteink bevonásával megoldani?
3. Készítsünk az előző feladathoz hasonló, de 16 LED-ből álló villogót. Ha az előző feladathoz hasonló rövid, frappáns kódot szeretnénk írni, miért nem tudunk? Miért van baj a bájt-méretű regiszterekkel?
4. Írjunk olyan programot, amely eljátszik egy általunk kedvelt dallamot. Ha megvan a dallam kottája, utána kell néznünk, hogy az egyes hangok pontosan mekkora frekvenciát jelentenek, és ez alapján megfelelő időzítéssel már meg tudjuk oldani a feladatot.
5. Készítsünk olyan villogót, ami néhány rákötött nyomógomb állapotát is vizsgálja. A gombok ugyan nyomógombok, de rádiógombként viselkednek, azaz mindig a legutoljára megnyomott gomb szerinti program fusson. Emiatt nem kell pergésmentesítéssel foglalkoznunk. Néhány villogási mód pl.: karácsonyfa, középről kifele, folyamatjelző.
6. Készítsünk több dallamot ismerő programot, és a dallamok közül a nekünk megfelelőt az előbbi feladathoz hasonlóan nyomógombok segítségével válasszuk ki. Pergésmentesítés itt sem szükséges, mert csak zavarná a dallam lejátszását a felesleges `delay`-ekkel.

Kijelzők illesztése

Lényegesebb különbségek, jellegzetességek a mikroPascalban

Miközben tapasztalatra tettünk szert a mikroPascalban használt Pascal nyelvben, láthattunk néhány szokatlan dolgot. Az alábbiakban a főbb nyelvbeli és fejlesztői környezetre vonatkozó eltéréseket olvashatjuk, amire érdemes odafigyelni a fejlesztés során.

- Fordításkor automatikusan elmenti a projekthez tartozó fájlokat a környezet.
- Mindig csak egy megnyitott projekt lehet, viszont lehetséges, hogy más projekthez tartozó fájlt szerkesztünk éppen. Ha ekkor fordítunk, az eredetileg megnyitott projektet fordítja a környezet.
- Debuggolás indításakor nem fordítja automatikusan újra a projektet, amennyiben az változott, ezt nekünk kell megtenni.
- A `byte` típusnak karakter (`char`) is értékül adható és fordítva, a két típus között automatikus típusátalakítás van.
- A `string` típusú változóknak kötelező megadni a méretet. A `string` egy-egy karaktere egy-egy bájtot foglal el a memóriában. A `string` és a karakterekből álló tömb egy és ugyanaz. A `string`ek nem a szokványos Pascal-stílusú karakterláncok, hanem C-stílusúak, ami azt jelenti, hogy az első karakter a 0. pozíción kezdődik és a `string` hossza-1 pozíción van az utolsó. A `string` utolsó karakterén a `null` (`0`) karakter található, ez kötelező. Így például egy `string[10]` típusú változóban az „`alma`” a következőképpen tárolódik (az 5-9 pozíciókon bármilyen érték lehet):

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.
a	l	m	a	null					

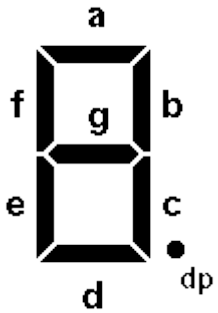
- Sok figyelmeztető hibaüzenet és figyelmeztetés nincs implementálva a fordítóban, ilyen pl. a tömb alul- vagy túlindexelése, ha hosszabb `string` típusú változót adunk értékül egy rövidebbnek. Ez azért veszélyes, mert a változó után található memóriaterület tartalma íródik felül (ami esetleg egy másik változó). Ha típuscsonkítás történik, pl. kétbájtos értéket adunk egy egybájtos változónak, azt is szó nélkül hagyja a fordító.

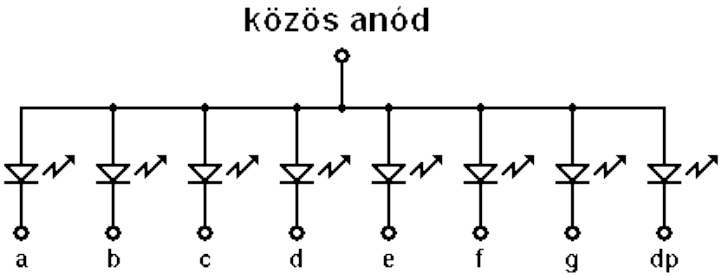
- Hasonlóan, a túlcsordulásról sem értesülünk, pl. a következő kód egy végtelen ciklust fog eredményezni, ha a `b byte` típusú: `for b:= 0 to 300 do ...`
- A nullával való osztás végtelent ad eredményül, ami a típusban tárolható legnagyobb számot jelenti. A PIC-ekben nincs futási idejű hibajelzés, így ez sem számít hibának.
- Az `end.` kulcsszónak megfelelő végtelen ciklus nem törli a watchdog timert, így ha erre hagyatkozunk és be van kapcsolva a watchdog timer, ki fog futni az időből a watchdog timer és a program végére érve egy idő múlva újra fog indulni a PIC.
- A többágú `case` elágazásban lehet `string` szerint is elágazni.
- Hexadecimális (16-os számrendszerbeli) számot megadhatunk a `$` jel segítségével, pl. `$FFFF`, valamint bináris (kettes számrendszerbeli) számot is megadhatunk a `%` jel segítségével, pl. `%10101010`.
- A logikai `false` (hamis) értéke 0, míg a logikai `true` (igaz) értéke 255. Minden 255-től különböző szám `false`-ként értékelődik ki, így a következő ciklus egyszer sem fog lefutni, ahelyett, hogy végtelen ciklust eredményezne: `while 1 do ...`; ehelyett használjuk a `true`-t: `while true do ...`
- Hasonló okokból nem működik a `PORTB.2:= not PORTB.2` sem, habár a súgó szerint a `not` operátor használható logikai operátorként is, de ez nincs így.
- Ha lehet, használjuk egy változó eggyel való növelésére az `inc` függvény és eggyel való csökkentésére a `dec` függvényt, ugyanis futási időt és helyet takarítunk meg vele.
- Az operátorok precedenciája Pascal nyelvi sajátosság és nagyon oda kell rá figyelni. A logikai és bitenkénti `and`, `or` stb. operátorok előbb kerülnek kiértékelésre, mint az összehasonlító (=) operátorok, így pl. a `szam = 0 and logikai_feltetel` úgy értékelődik ki, mintha `szam = (0 and logikai_feltetel)` lenne, nem pedig úgy, hogy `(szam = 0) and logikai_feltetel`. Ügyeljünk a helyes zárójelezésre!
- Sajnos nem mindig derül ki egyértelműen a súgóból, hogy pontosan mit csinál az adott függvény, és mivel a beépített könyvtárak forráskódja sem áll rendelkezésre, marad a próbálkozás és a lefordított kód assembly hívásainak elemezgetése.

Sajnos érezhető, hogy a típusok témaköre, főleg a `boolean` típus nincs eléggé jól átgondolva, és sok hiba van még a fejlesztői környezet 7.0-s változatában is. Fejlesztés közben ügyeljünk a fentebb felsorolt jellegzetességekre, és csak apránként haladjunk, közben sokat tesztelve.

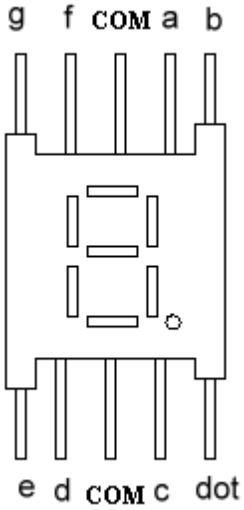
7 szegmenses kijelző kezelése

Következő feladatunk 4 db 7 szegmenses kijelzőn a „`hELO`” felirat megjelenítése. A 7 szegmenses kijelzőt alapvetően számok megjelenítésére találták ki, de a legtöbb betűt is meg lehet rajta jeleníteni. Nevét onnan kapta, hogy egy digit (számjegy) megjelenítéséhez 7+1 LED-et használ, olyan elrendezésben, hogy az egyes pálcikák egy 8-as számjegyet adjanak ki. A bónusz LED a pontot vezérli a számjegy után. Ha megnézünk egy ilyen alkatrészt, azt látjuk, hogy tipikusan 10 kivezetése van a 8 LED-hez.

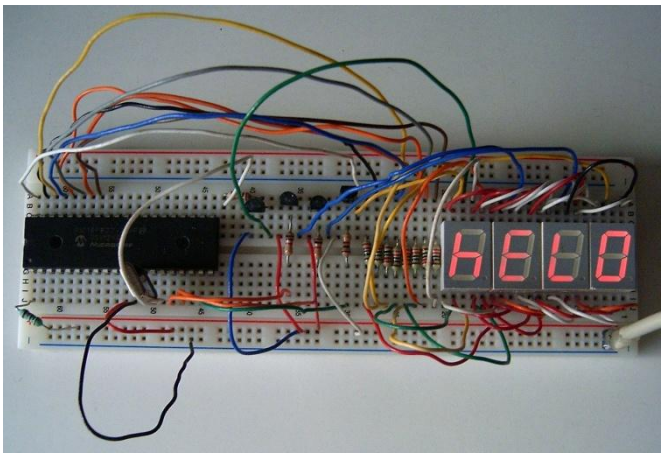




A 8 LED-nek van egy közös kivezetése, attól függően, hogy ez a LED-ek anódjához vagy katódjához kapcsolódik, ennek megfelelően ezt közös anódnak vagy közös katódnak hívják. Ez a láb jellemzően (DIP tokozás esetében) az alsó és a felső sor középső lába szokott lenni. A maradék nyolc kivezetés a LED-ek másik lába. Az alkatrész nem tartalmaz belső ellenállásokat, így azt nekünk kell külön sorosan kötnünk az egyes LED-ekhez (a közös anódra ne kössük, hiszen akkor ugyanakkora feszültség esetén nem fog ugyanakkora áram folyni).



A feladatunk 4 ilyen kijelző vezérlése – egyszerre. Mint tudjuk, az általunk használt PIC-nél 33 lábat használhatunk saját céljainkra, ezek valamilyen porthoz vannak hozzárendelve. A 4 db kijelző meghajtására összesen 4×8=32 lábra lenne szükségünk, amivel még pont rendelkezünk. Viszont jó, ha már most megszokjuk, hogy a PIC-ek lábaival spórolni kell, így ennél az áramkörnél multiplexelni fogjuk az egyes számjegyeket, ez azt jelenti, hogy a négy közös anódokhoz, amiket egyébként a pozitív tápfeszültségre kötnénk, beiktatunk egy-egy engedélyező tranzisztort, aminek az lesz a szerepe, hogy az egész kijelzőt ki vagy bekapcsolhatjuk. A kijelzőkön a megfelelő szegmens LED-ek lábait pedig összekötjük, és ezeket egyszerre fogjuk kezelni. A módszer lényege az, hogy egyszerre mindig csak egy kijelzőn jelenítünk meg valamit: először csak az elsőn megjelenítjük a „h” betűt, majd a másodikon az „E”-t, majd a harmadikon az „L”-t és végül a negyediken az „O”-t, majd kezdjük az egészet elölről. Mindezt olyan gyorsan tesszük, hogy a villogás nem tűnik fel az emberi szemnek. Cserébe viszont igen hatékonyan használtuk a PIC lábait: 8 db-ot az egyes szegmenseknek és 4-et az egyes kijelző-engedélyező tranzisztoroknak, azaz összesen 12-t.



A közös anódos kijelzőknél az anódot a pozitív tápfeszültségre, az egyes szegmensek kivezetéseit pedig (egy ellenálláson keresztül) a földre kell kötni, ahhoz, hogy világítson az adott LED. Ha ezek közül bármelyik nem teljesül, nem fog folyni az áram, így nem fog világítani az adott LED. Itt spórolhatunk egy kicsit az ellenállásokkal, ugyanis a megfelelő LED-ek közül mindig csak egy fog égni (így fogjuk vezérelni a tranzisztorokat).

Ezek után lássuk a programot:

```
program ledkijelzo;
const
  // megfelelő szegmensek
  __a = 1;
```

```
  __b = 2;
  __c = 4;
  __d = 8;
  __e = 16;
  __f = 32;
  __g = 64;
  __dp = 128;

  // számjegyek
  _0 = 255 - __a - __f - __e - __d - __c - __b;
  _1 = 255 - __b - __c;
  _2 = 255 - __a - __b - __g - __e - __d;
  _3 = 255 - __a - __b - __g - __c - __d;
  _4 = 255 - __f - __g - __b - __c;
  _5 = 255 - __a - __f - __g - __c - __d;
  _6 = 255 - __a - __f - __g - __e - __d - __c;
  _7 = 255 - __a - __b - __c;
  _8 = 255 - __a - __b - __c - __d - __e - __f - __g;
  _9 = 255 - __a - __b - __f - __g - __c - __d;
  // betűk (k, m, w és x megjelenítése nem lenne
egyértelmű, ezért azok itt nem szerepelnek)
  _a = 255 - __e - __f - __a - __b - __c - __g;
  _b = 255 - __f - __e - __d - __c - __g;
  _c = 255 - __g - __e - __d;
  _d = 255 - __b - __g - __e - __d - __c;
  _e = 255 - __a - __f - __e - __d - __g;
  _f = 255 - __a - __f - __e - __g;
  _g = 255 - __a - __f - __e - __c - __d;
  _h = 255 - __f - __g - __e - __c;
  _i = 255 - __b - __c;
  _j = 255 - __a - __b - __c - __d;
  _l = 255 - __d - __e - __f;
```

```
  _n = 255 - __e - __f - __a - __b - __c;
  _o = 255 - __a - __f - __e - __d - __c - __b;
  _p = 255 - __e - __f - __a - __b - __g;
  _q = 255 - __f - __g - __a - __b - __c;
  _r = 255 - __e - __g;
  _s = 255 - __a - __f - __g - __c - __d;
  _t = 255 - __f - __g - __e - __d;
  _u = 255 - __f - __e - __d - __c - __b;
  _v = 255 - __e - __d - __c;
  _y = 255 - __b - __f - __g - __c - __d;
  _z = 255 - __a - __b - __g - __e - __d;

begin
  TRISB:= 0;
  TRISC:= 0;
  repeat
    // első digit
    PORTC:= %00000001;
    PORTB:= _h;
    Delay_ms(5);
    // második digit
    PORTC:= %00000010;
    PORTB:= _e;
    Delay_ms(5);
    // harmadik digit
    PORTC:= %00000100;
    PORTB:= _l;
    Delay_ms(5);
    // negyedik digit
    PORTC:= %00001000;
    PORTB:= _o;
    Delay_ms(5);
  until false;
```

```
end.
```

A program elején definiáltuk a 7 szegmenses kijelzőhöz tartozó lábakat, konstansok formájában. Mivel az RB0-ra kötöttük az „a” lábat, az RB1-re a „b”-t és így tovább, ennek megfelelően adtunk értéket az `__a`, `__b`, `__c` stb. konstansoknak. Mivel az egyes LED-ek akkor fognak világítani, ha az adott láb a földön van, ezért ha a B portra `0`-t írunk ki, minden szegmens világítani fog az éppen engedélyezett kijelzőn, ha pedig `255`-öt, akkor egyik sem (fordított logika). Ezért az egyes betűknek megfelelő konstansokat úgy adjuk meg, hogy a `255`-ből levonjuk azokat a szegmenseket jelző konstansokat, amelyek világítaniuk kell (ekkor az ezeknek a szegmenseknek megfelelő bitek 0-k lesznek, a többi pedig marad 1). Ha ezt az értéket írjuk ki a B portra, akkor a kívánt betű fog megjelenni a kijelzőn. A programban az összes számjegy, valamint szinte az összes betű definiálva van, így ezeket használhatjuk akár most, akár későbbi fejlesztéseink során.

Fontos, hogy ezek a konstansok nem foglalnak helyet sem az adat-, sem a programmemóriában, használatukkor a fordító befordítja a megfelelő kiszámolt értéket.

A program első lépése a portok bekonfigurálása: a B portról fogjuk meghajtani az egyes szegmenseket (0-val engedélyezünk), a C port alsó 4 bitjével pedig az egyes digiteket fogjuk engedélyezni (1-el engedélyezünk), a 0. bit fogja vezérelni az első kijelzőt, az 1. bit a másodikat és így tovább. Az egyes bitekre egy-egy tranzisztor van kötve, ez fogja vezérelni az egyes kijelzőket.

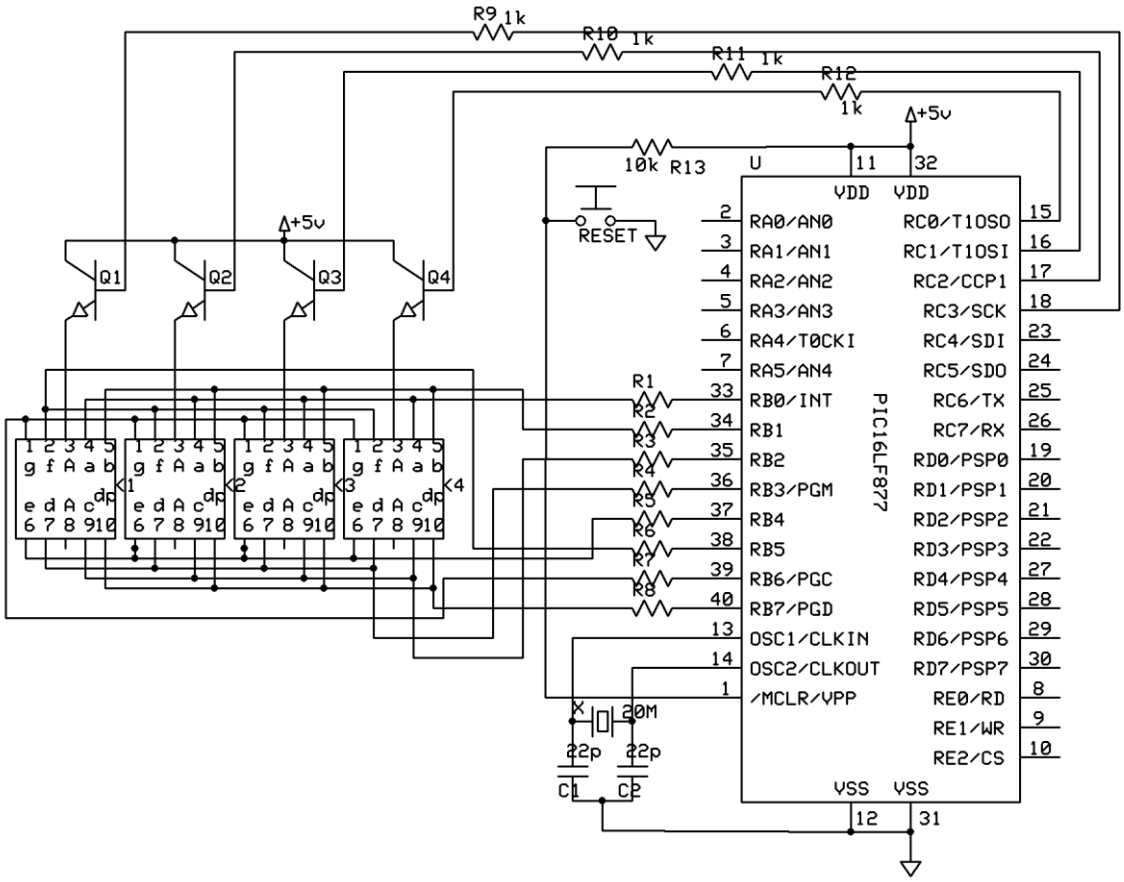
Mi itt ún. NPN típusú tranzisztort használunk az egyes kijelzők engedélyezésére. Rákötheteknénk közvetlenül is a PIC lábaira a kijelzők anódját, viszont ha engedélyezzük 1-el az adott kijelzőt, akkor a megengedettnél nagyobb áram folyna a PIC lábain. Ahhoz, hogy ezt kivédjük, tranzisztort fogunk használni az anód és a pozitív tápfeszültség között, ezen fog folyni a „nagy” áram, mi csak engedélyezzük a tranzisztort, vagy letiltjuk, azaz egy egyszerű kapcsolóként fogjuk használni. A tranzisztornak három kivezetése van: kollektor (collector, C), emitter vagy kibocsátó (E) és bázis (basis, B). Az NPN tranzisztor leegyszerűsítve két áramkörből áll: egyrészt folyik az áram a bázistól az emitter felé, másrészt a kollektortól az emitter felé. Amennyiben a bázis és az emitter között megfelelő nagyságú áram folyik, a tranzisztor

engedélyezi az áram folyását kollektortól és emitter felé. Ez a mi esetünkben azt jelenti, hogy ha a kollektort rákötjük a pozitív tápfeszültségre, a kijelző anódját pedig az emitterre, akkor, ha bázison 5 V van, a tranzisztor bekapcsol és folyik az áram a kollektortól az emitter felé. Ha a bázison 0 V van, a tranzisztor kikapcsol, és nem folyik az áram a kollektortól az emitter felé.

A bázison csak kicsi áramnak szabad folynia, ehhez 5 V esetén használhatunk pl. egy 1 kOhmos ellenállást. Az ellenállás másik lábát a PIC C portjának megfelelő lábára kötjük. Ha a láb értéke 1, akkor a közös anód és a pozitív tápfeszültség között folyhat az áram, így az adott kijelző szegmensei világíthatnak.

A tranzisztor igen sokrétű és bonyolult eszköz, rengeteg célra használják. A mi esetünkben csupán engedélyezésre használtuk, ezért gyakorlatilag bármilyen NPN tranzisztor megfelel, ilyen például a 2N3904 is, de az általam megépített kapcsolásban pl. BC546B-ket használtam. A tranzisztorok lábkiosztása igen eltérő, nézzünk utána az adatlapjában a kivezetések megfeleltetésének.

A program egyetlen ciklust tartalmaz, amelynek első részében kijelzünk egy „h” betűt az



első kijelzőn: engedélyezzük az első kijelzőt vezérlő tranzisztort, a többit letiltjuk a C port segítségével, majd a B porton beállítjuk a „h” betűnek megfelelő biteket. Ezek után várunk egy kicsit, majd ugyanezt elvégezzük a többi kijelzőn is. Mindez olyan gyorsan zajlik le, hogy szemünk észre sem veszi, hogy egyszerre mindig csak egy kijelző világít – hasonlóan a mozivásznon használt technikához. Egyetlen hátránya ennek a megoldásnak, hogy a LED igen jól reagál a ki/bekapcsolásokra, így mivel egy-egy kijelző egy másodperc alatt gyakorlatilag csak negyed másodpercnyit világít, ezért lényegében negyed annyira fog világítani, mintha folyamatosan csak ő égne (ezt egyébként a szemünk úgy érzékeli, mintha kb. feleannyira lenne erős a világítás).

Visszaszámláló a 7 szegmenses kijelzőn

Készítsünk egy egyszerű számlálót, ami, ha letelt az idő, jelzi nullák villogtatásával! Az első két digiten jelenjen meg a hátralévő percek száma, a második két digiten pedig a hátralévő másodpercek száma!

Kapcsolásunk marad változatlan, programunk a következő lesz:

```
program ledkijelzoido;
const
  // megfelelő szegmensek
  __a = 1;
  __b = 2;
  __c = 4;
  __d = 8;
  __e = 16;
  __f = 32;
  __g = 64;
  __dp = 128;

  // számjegyek
  _0 = 255 - __a - __f - __e - __d - __c - __b;
  _1 = 255 - __b - __c;
  _2 = 255 - __a - __b - __g - __e - __d;
  _3 = 255 - __a - __b - __g - __c - __d;
```

```
  _4 = 255 - __f - __g - __b - __c;
  _5 = 255 - __a - __f - __g - __c - __d;
  _6 = 255 - __a - __f - __g - __e - __d - __c;
  _7 = 255 - __a - __b - __c;
  _8 = 255 - __a - __b - __c - __d - __e - __f - __g;
  _9 = 255 - __a - __b - __f - __g - __c - __d;

  // tömb a számjegyeknek
  szamjegyek: array[0..9] of byte= (_0, _1, _2, _3, _4,
_5, _6, _7, _8, _9);
var
  perc: byte;
  masodperc: byte;
  i: byte;

begin
  TRISB:= 0;
  TRISC:= 0;

  perc:= 2;
  masodperc:= 0;
  repeat
    for i:= 1 to 50 do begin
      PORTC:= %00000001;
      PORTB:= szamjegyek[perc div 10];
      Delay_ms(5);
      PORTC:= %00000010;
      PORTB:= szamjegyek[perc mod 10];
      Delay_ms(5);
      PORTC:= %00000100;
      PORTB:= szamjegyek[masodperc div 10];
      Delay_ms(5);
```

```
      PORTC:= %00001000;
      PORTB:= szamjegyek[masodperc mod 10];
      Delay_ms(5);
    end;

    if (perc = 0) and (masodperc = 0) then begin
      // ha lejárt az idő, várunk 1 mp-t, ezzel villogást
idézve elő
      PORTC:= %00000000;
      Delay_ms(1000);
    end else begin
      // ellenkező esetben levonunk egy másodpercet
      if masodperc = 0 then begin
        dec(perc);
        masodperc:= 59;
      end else
        dec(masodperc);
    end;
  until false;
end.
```

Programunk nagyban hasonlít az előzőre. Az elején ugyanúgy deklaráltunk a szegmenseknek konstansokat, majd azok segítségével az egyes számjegyeknek konstansokat. Most, mivel csak számjegyeket fogunk kijelezni, készítettünk egy 0-tól 9-ig indexelhető bájtokból álló tömböt is, aminek az elemei a számjegy-konstansok, később ez még hasznos lesz. Majd definiáltunk két változót is, a perc és másodperc tárolására. A portok szokásos konfigurálása után ezeket a változókat is inicializáltuk, a visszaszámlálást 2 perc 0 másodpercről indítjuk. A végtelen ciklus első felében egy újabb ciklus található, ami összesen 1 másodpercig fog futni, eddig az előző programban ismertetett módon kijelezzük az aktuális perc-másodperc értéket. Az egyes számjegyeket az egész osztás `div` operátorával és a maradékképzés `mod` operátorával állítjuk elő: egy kétszámjegyű `x` szám esetében az `x div 10` az első számjegyet, az `x mod 10` pedig az utolsó számjegyet jelenti.

Ezek után egy feltétellel megvizsgáljuk, hogy lejárt-e már az időzítő. Ha igen, üresen hagyjuk a kijelzőt egy másodpercig. Mivel a 0000 kijelzése is egy másodpercig tartott és ezt fogjuk ismételni ezután, ezért ez egy lassú villogás lesz.

Végül, ha a feltétel hamis volt, eggyel csökkentjük a hátralevő másodpercek számát, valamint ha az elérte a nullát, akkor csökkentjük a hátralévő percek számát is.

LED-es kijelzők

Jó, ha tudunk róla, hogy ha kifejezetten szöveg megjelenítésére szeretnénk LED-es kijelzőt használni, rengeteg különböző kijelző közül választhatunk. Pl. létezik kifejezetten szöveg megjelenítésére tervezett 14 szegmenses kijelző is, amivel szép, olvasható szövegeket tudunk megjeleníteni. Továbbá léteznek ún. LED-mátrixos kijelzők is, amivel bármilyen alakzatot meg tudunk jeleníteni.

Ezek a kijelzők többféle színben kaphatóak, az új fehér és kék LED-ek viszont meglehetősen drágák, áruk többszöröse a hagyományos piros, zöld és sárga LED-ekének. A LED-eket nagy fényerejük miatt sok helyen használják nagy kijelzőkben (pl. reklám célokra), ahol sok ezer LED van összekapcsolva és az RGB-hez hasonló színkeveréssel akár színes képeket is elő tudnak állítani. Ezen kívül fényszórókban is használatos, mivel már különösen nagy fényerejű LED-eket is képesek gyártani: felérnek egy izzóval, nem égnek ki, viszont fogyasztásuk a töredéke.

Elektronikai projektjeink során a fogyasztást mindig tartsuk szem előtt, ugyanis ha 5 V-on üzemeltetjük az áramkört és egy 220 Ohmos ellenállással van sorba kapcsolva minden LED, akkor a rajtuk átfolyó áram egyenként 5/220=23 mA. Jelenleg egyidejűleg akár 8 LED is világíthat, ez már 8×23=184 mA, ami azért nem is olyan kevés, ha elemről szeretnénk üzemeltetni az áramkört. Ez még csak egy digit fogyasztása, de a többi csak a következő időszeletben ég, viszont ezt nem csinálhatjuk a végtelenségig. Használhatunk LCD-s kijelzőket, aminek ugyan egyáltalán nincs természetes világítása, viszont sokkal kevesebb a fogyasztása (ezért használják ezt a digitális kijelzésű karórákban).

LCD kezelése

A következő feladatunk legyen a „`Helló világ!`”, pontosabban ennek a szövegnek a kiírása egy LCD-re. Természetesen itt csak általános célú LCD-ket fogunk meghajtani, egy

HD44780-kompatibilis szöveges LCD-t fogunk használni. Ez az LCD ASCII karakterek megjelenítésére hivatott, az általunk használt LCD 2 sorból áll, az egyes sorokban pedig 16 karakter található. Az egyes LCD-k lábkiosztása eltérő lehet, bekötés előtt nézzünk utána az adatlapjában. Az egyes elektronikai alkatrészek adatlapjának megtalálása az Internet segítségével igen könnyű, egyszerűen rákeresünk egy internetes kereső segítségével az alkatrész típusára, az általam használt LCD pl. Displaytech 162B (a típusa mindig rá van írva). Az adatlapokat legrosszabb esetben szkennelt PDF formájában találhatjuk meg az Interneten.

Lehetséges, hogy ahhoz, hogy az LCD-t a próbapanelhez illeszteni tudjuk, kivezetéseket kell forrasztanunk. Alapvetően a következő lábak találhatóak meg egy LCD-n:

Láb	Irány	Funkció
Vss	-	Föld
Vcc	-	+5 V
Vee / V0	-	Kontranszt beállítás
RS	I	0: vezérlés 1: adat
R/W	I	0: írás az LCD modulba 1: olvasás az LCD modulból
E	I	Engedélyezés
D0	I/O	Adatbusz 0. bit
D1	I/O	Adatbusz 1. bit
D2	I/O	Adatbusz 2. bit
D3	I/O	Adatbusz 3. bit
D4	I/O	Adatbusz 4. bit
D5	I/O	Adatbusz 5. bit
D6	I/O	Adatbusz 6. bit
D7	I/O	Adatbusz 7. bit
A	-	Háttérvilágítás anódja
K	-	Háttérvilágítás katódja

Ez összesen 16 láb. Az általunk használt LCD modulhoz tartozik egy kis vezérlő IC is, ugyanis az LCD közvetlen meghajtása nem egyszerű feladat, sokkal könnyebb, ha egy másik IC ezt elvégzi helyettünk. Ez az IC 5 V-ról üzemel, így figyeljünk a helyes

tápfeszültség biztosítására, erre szolgálnak a Vss és Vdd lábak. Maga a kijelző más feszültségről működik, erre szolgál a Vee láb. Ha nagyobb feszültséget kötünk az LCD-re, erősebb kontrasztot kapunk, ha kisebbet, gyengébb lesz a kontraszt. A megfelelő kontraszt az LCD típusától erősen függ, jellemzően ez az érték 4,5 V körül mozog, ezt egy potenciométer (változtatható értékű ellenállás) segítségével állítjuk be: egyik lábát a földre kötjük, másikat az 5 V-ra, a csúszkát pedig a Vee-hez kötjük.

Ha csak ennyit kötünk be, az LCD máris működőképes. A legtöbb LCD-n megjelenik valamilyen karaktersorozat bekapcsolás után, akkor is, ha nem írtunk rá semmit, például kitöltött négyzet karakterek az első sorban. Ha ilyet látunk, valószínűleg tökéletesen beállítottuk a kontrasztot. Ha a kontrasztot beállítjuk az optimális kontraszt fölé, akkor minden karakter helyén kitöltött négyzeteket fogunk látni, ez a beállítás nem jó, mert majd a kiírt szöveget sem fogjuk látni.

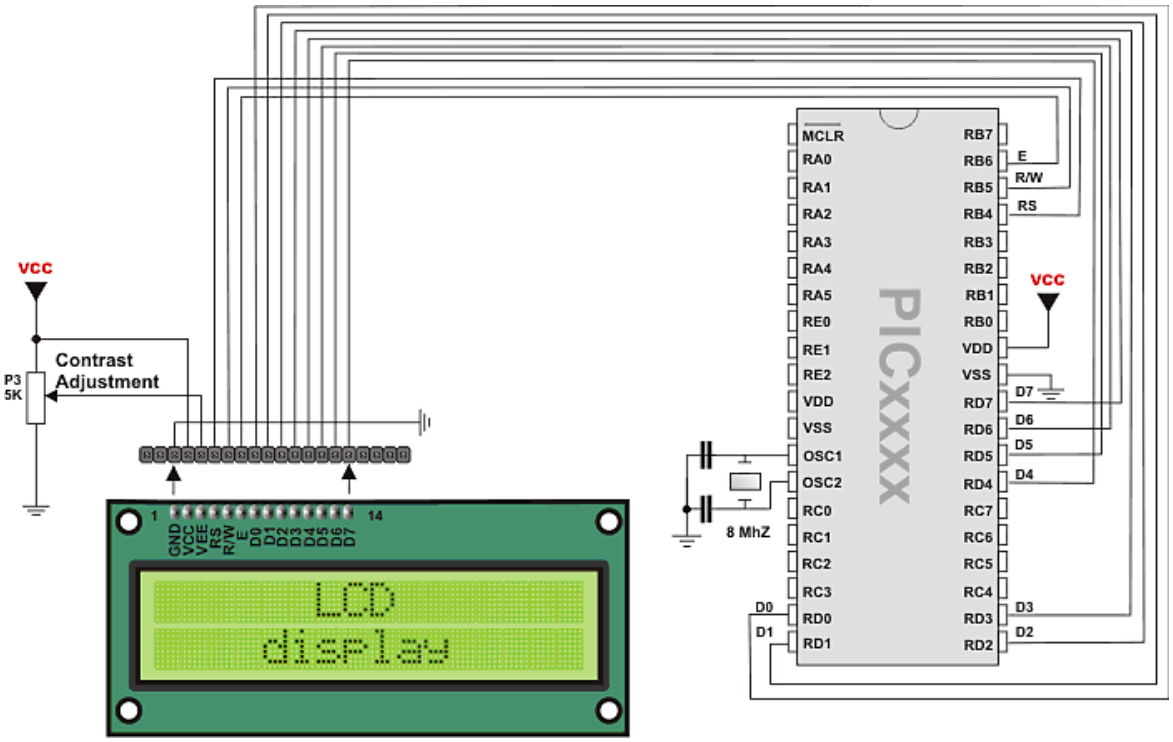
Az LCD opcionálisan LED-es háttérvilágítással is rendelkezik, ha van „A” és „K” jelű lába, ez a LED anódja és katódja. Itt is figyeljünk egy megfelelő méretű ellenállás soros kapcsolására! Ezt a háttérvilágítást akár a PIC-ről is vezérelhetjük, mintha ez is csak egy közönséges LED lenne.

Az engedélyező (enable, E) láb azt jelzi az LCD-nek, hogy most vele szeretnénk beszélgetni. Ez a láb alapállapotban magasan (high, logikai 1) van, majd miután minden bitet beállítottunk annak megfelelően, hogy mit szeretnénk az LCD-től, ezt az engedélyező bitet alacsony szintre (low, logikai 0) állítjuk, ez a lehúzás jelzi az LCD-nek, hogy most olvassa el az utasítást. Ugyanis, ha csak találomra sorba állítgatjuk be az egyes lábakat, lehet, hogy az LCD pont rosszkor olvassa ki az adott utasítást. Arról nem is beszélve, hogy az egyes utasítások között is eltelik valamennyire idő és ez bezavarhatja az LCD-t. Az RS láb azt mondja meg, hogy mit szeretnénk az LCD-től: adatot vagy utasítást forgalmazni (pl. kurzormozgatás). Az R/W azt adja meg, hogy írni szeretnénk az LCD-be, vagy olvasni onnan. Ha írni szeretnénk, a PIC által beállított D0..D7 láb értékeit kiolvassa az LCD, ha

pedig olvasni szeretnénk, akkor a D0..D7 lábak értékeit az LCD állítja be, a PIC pedig csak beolvassa. Az LCD adatlapjában pontosan olvashatók az utasítások és az egyes lépések közötti minimális és maximális időzítés. Szerencsére a mikroPascal az LCD kezeléséhez is tartalmaz beépített függvénygyűjteményt, ami nagyban leegyszerűsíti a dolgunkat. Mi 8 biten fogunk kommunikálni az LCD-vel, ami azt jelenti, hogy a D0..D7 lábak mindegyikét kihasználjuk. Léteznek 4 bites LCD-k is, azok csak a D0..D3 lábakat használják. Nézzük az alábbi programot:

```
program lcd;

begin
  TRISB := 0;
  TRISD := 0;
  // LCD inicializálása
  Lcd8_Init(PORTB, PORTD);
  // Szöveg kiírása
  Lcd8_Out(1, 1, 'Hello vilag!');
end.
```

Nézzük meg a mikroPascal súgójában az `Lcd8_` kezdetű függvényekhez tartozó oldalt. Ennek az alján szerepel egy kapcsolási rajz, ami azt mutatja, hogy mi az LCD alapértelmezett összekötési módja a PIC-cel. Ez az ábra ugyan nem tartalmazza a 10 kOhmos ellenállást az MCLR láb és a pozitív tápfeszültség között, ettől ez még szükséges a mi PIC-ünk esetében, kvarckristályként pedig használhatjuk a 20 Mhz-est ugyanúgy, mint eddig. Látjuk, hogy az adatbusz bitjei a D portra kerültek, a többi vezérlő bit pedig a B portra. Az ábra nem mutatja a háttérvilágítás kivezetéseit, ha szeretnénk, ezt is beköthetjük a megfelelő módon. A portok irányának beállítása után az `Lcd8_Init` utasítással megmondjuk, hogy melyik portokra kötöttük a vezérlő és adatbusz lábakat és a függvények a későbbiekben ezeket használják. Mivel az alapértelmezett összekötést használtuk, nem kell külön-külön minden lábhoz megmondanunk, hogy melyik bitre került. Ez az utasítás nem csak arra szolgál, hogy a későbbiekben tudják a függvények, hogy hova került az LCD, hanem inicializálja is az LCD-t egy speciális vezérlő utasítás



segítségével. A következő utasítás pedig kiírja a „`Helló világ!`” szöveget az első sor első oszlopával kezdődően.

Ha esetleg nem működik, annak egyik lehetséges oka lehet, hogy egyes LCD-k esetében szükség lehet több inicializáló utasítás kiadására, így módosíthatjuk a programot ennek megfelelően vagy az LCD resetelése nélkül reseteljük meg a PIC-et, hogy újra kiadja az utasítást.

Az LCD-k többféle méretben kaphatóak, pl. 2-sorosak vagy 4-sorosak, míg a bennük található logika (a vezérlést végző IC) teljesen ugyanaz. Így, ha csak 2x16 karakteres LCD-t használunk szó nélkül engedelmeskedni fog az IC, ha mi a 3. sorra szeretnénk ugrani vagy a 20. oszlopba szeretnénk írni valamit.

Az LCD kurzort is támogat, ennek a megjelenését is vezérelhetjük. A beépített adatmemória nagyobb, mint a kijelző mérete, olyan helyre is írhatunk, ami nem látszik. Az adatmemória mérete típusoktól függően változhat. Ahhoz, hogy a nem látható részt láthatóvá tegyük, eltolhatjuk (shiftelhetjük) a kijelzőt az adatmemória felett. Ha jobbra shiftelünk úgy fog tűnni, mintha minden a kijelzőn található karakter eggyel jobbra lépett volna.

Nézzük sorra az LCD-vel kapcsolatos utasításokat! Az első sor és oszlop száma 1 minden utasítás esetében.

- `Lcd8_Init(var ctrlport, dataport : byte)`: inicializálja az LCD-t a kapcsolási rajzon látható lábkiosztással (a `ctrlport` 5-ös lába lesz az R/W, 4-es lába az RS, 6-os lába az E, a `dataport` egyes lábai pedig a D0..D9-nek fognak megfelelni).
- `Lcd8_Config(var ctrl_port : byte; var data_port : byte; rs, ctrl_rw, enable : byte; db7, db6, db5, db4, db3, db2, db1, db0 : byte)`: inicializálja az LCD-t ugyanúgy, mint az `Lcd8_Init`, viszont részletesebben testre szabhatjuk, hogy melyik láb hova van az LCD-n kötve.
- `Lcd8_Out(row, col : byte; var text : array[255] of char)`: kiír egy tetszőleges ASCII szöveget az LCD-re. A szöveg (`text`) első karaktere adott sornak (`row`) adott oszlopához (`col`) fog kerülni.
- `Lcd8_Out_Cp(var text : array[255] of char)`: kiírja az adott szöveget (`text`) a kurzor pozíciójától kezdődően majd a kurzort ennek megfelelően lépteti jobbra.
- `Lcd8_Chr(row, col, character : byte)`: kiír egy tetszőleges karaktert (`character`) az LCD adott sorának (`row`) adott oszlopába (`col`).
- `Lcd8_Chr_Cp(character : byte)`: kiír egy tetszőleges karaktert (`character`) a kurzor helyére, majd a kurzort eggyel jobbra lépteti.
- `Lcd8_Cmd(command : byte)`: kiadja a `command` utasítást az LCD-nek. Az utasítás a következők valamelyike lehet:
    - `LCD_FIRST_ROW`: kurzor mozgatása az első sorba
    - `LCD_SECOND_ROW`: kurzor mozgatása a második sorba
    - `LCD_THIRD_ROW`: kurzor mozgatása a harmadik sorba
    - `LCD_FOURTH_ROW`: kurzor mozgatása a negyedik sorba
    - `LCD_CLEAR`: képernyő törlése
    - `LCD_RETURN_HOME`: kurzor mozgatása a kiinduló állapotba, valamint az eltolt képernyőt is visszahelyezi a kiindulási helyére
    - `LCD_CURSOR_OFF`: kurzor kikapcsolása

- `LCD_UNDERLINE_ON`: kurzor megjelenítése aláhúzásként
- `LCD_BLINK_CURSOR_ON`: villogó kurzor bekapcsolása
- `LCD_MOVE_CURSOR_LEFT`: kurzor balra mozgatása
- `LCD_MOVE_CURSOR_RIGHT`: kurzor jobbra mozgatása
- `LCD_TURN_ON`: kijelző bekapcsolása
- `LCD_TURN_OFF`: kijelző kikapcsolása
- `LCD_SHIFT_LEFT`: egész képernyő balra léptetése
- `LCD_SHIFT_RIGHT`: egész képernyő jobbra léptetése

Léteznek grafikus LCD-k is, amelyekre tetszőleges grafikát tudunk rajzolni, az ehhez tartozó függvényeket megtaláljuk a mikroPascal súgójának Graphic LCD Library témakörében.

Összefoglaló kérdések, feladatok

A fejezet elején összegyűjtve olvashattunk néhány olyan dologról, amire figyelnünk kell vagy eltér a szokásos Pascal nyelvjárástól. Ha az ellenőrző kérdésekre tudunk válaszolni, akkor a legfontosabb ismereteket elsajátítottuk:

1. Hogyan adhatunk meg bináris és hexadecimális számokat?
2. Mit jelent a `PORTB.2`?
3. Hogyan ábrázolja a mikroPascal a `string` típust, mire kell figyelni a használatakor?
4. Mire kell ügyelni a `for` ciklus használatakor?
5. Mit jelent a túlcsordulás, mikor fordul elő?
6. Hogyan működik a `boolean` típus, mire kell figyelni a használatakor? Milyen `integer` értéknek felel meg a `false` és a `true`?
7. Mire kell figyelni olyan kifejezésekben, ami logikai operátorokat (mint az `and` vagy az `or`) is tartalmaznak, és egyenlőség (=) vagy nem egyenlőség (<>) vizsgálatot is?

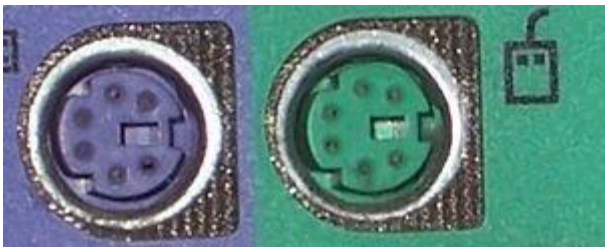
Ebben a fejezetben megismerkedhettünk a leggyakrabban használt kijelzők működésével és PIC-ből való használatukkal. Láttuk, hogy néhány számjegy kijelzésére remekül használhatóak a 7 szegmenses kijelzők, viszont az LCD-knek kisebb a fogyasztásuk, ellenben jóval drágábbak, sőt vannak grafikus LCD-k is. Valószínűleg magunk is rengeteg

érdekes feladatot találunk ki, az alább olvasható feladatok csak kedvcsináló jelleggel szerepelnek itt:

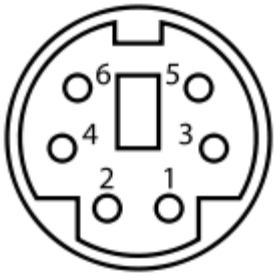
1. Módosítsuk a visszaszámláló programunkat úgy, hogy legyen egy gomb, amivel leállíthatjuk vagy folytathatjuk a visszaszámlálást! Ügyeljünk a pergésmentesítésre, illetve arra, hogy pergésmentesítés idejét figyelembe véve is pontosan számoljon vissza az óra!
2. Futószöveg 7 szegmenses kijelzőn. Autórádiókhoz hasonlóan hosszabb szövegeket is ki tudunk jelezni, ha olvasható sebességgel mozgatjuk a kijelzőn egy irányba a szöveget az elejétől a végéig.
3. Az előző feladatot LCD-n is megvalósíthatjuk, itt több karakter fér el, és olvashatóbb a szöveg is.
4. Valósítsunk meg egy pontos órát lehetőleg 6 darab 7 szegmenses kijelző segítségével úgy, hogy az időt nyomógombokkal állíthassuk be! Ha nyomva tartjuk a percet beállító gombot, gyorsan pörgeti a percet, ha az órát beállító gombot, az órát növeli gyorsan. Rakhatunk az óra és a perc csökkentésére is egy-egy gombot.

Haladóknak

Billentyűzet kezelése



Mivel már tudunk szöveget kiírni az LCD-re, jó lenne, ha be is tudnánk gépelni azt, méghozzá egy közönséges billentyűzeten! Ehhez egy PS/2 csatlakozójú billentyűzetre lesz szükségünk, valamint egy PS/2 anya csatlakozóra, amit akár egy régi alaplapból is kiforraszthatunk.



A PS/2-es billentyűzet PIC-re való illesztéséhez legalább 6Mhz-es kvarckristályra lesz szükségünk. Az ábrán egy anya csatlakozó lábkiosztása látható, szemből nézve. Az egyes lábak funkciója (általános esetben):

Láb száma	Jelölés
1	DATA
2	-
3	GND
4	Vcc
5	CLK
6	-

Azaz a 6 lábból összesen 4 van használatban. Ebből is kettő a tápellátásért felelős, a GND-t kell a földre kötni, a Vcc-t pedig az 5 V-ra. Egy billentyűzet maximális fogyasztása 100 mA lehet, figyeljünk, hogy az áramforrásunk biztosítani tudja-e ezt a szükséges teljesítményt.

A kommunikáció a DATA és CLK lábakon történik. Valójában a CLK egy órajel, amit a PIC fog biztosítani, az adat pedig a DATA lábon fog folyni. Ajánlott mindkét lábat összekötni a pozitív tápfeszültséggel egy 10 kOhmos ellenálláson keresztül (ezt hívják felhúzó ellenállásnak).

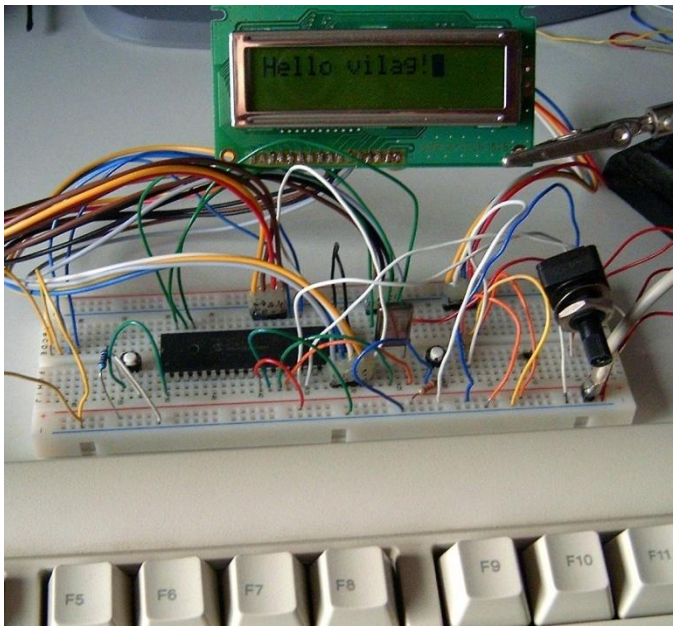
Szerencsére programunkban nem kell „rángatnunk a drótokat”, azaz adni az órajelet és kezelni a DATA lábat, hanem használhatjuk a beépített mikroPascal függvényeket.

Programunk minden hagyományos leütött karaktert kiír, a BACKSPACE billentyű hatására pedig visszafele töröl:

```
program billentyuzet;
var
  karakter, specialis, lenyomva: byte;

begin
  // LCD inicializálása
  TRISB := 0;
  TRISD := 0;
  Lcd8_Init(PORTB, PORTD);
  Lcd8_Cmd(LCD_BLINK_CURSOR_ON);

  // PS/2 billentyűzet inicializálása
  INTCON:= 0;
  Ps2_Init(PORTC);

  while true do begin
    // megnézzük, történt-e változás a billentyűzeten
    if Ps2_Key_Read(karakter, specialis, lenyomva) =
1 then begin
      if lenyomva then begin
        if (specialis = 1) and (karakter = 16) then begin
          // backspace
          Lcd8_Cmd(LCD_MOVE_CURSOR_LEFT);
          Lcd8_Chr_CP(' ');
          Lcd8_Cmd(LCD_MOVE_CURSOR_LEFT);
        end else if specialis = 0 then begin
          // hagyományos
          Lcd8_Chr_CP(karakter);
        end;
      end;
```

```
    end;
    // pattogás ellen
    Delay_ms(10);
  end;
end.
```

Először a szokásos módon inicializáljuk az LCD-t, valamint bekapcsoljuk a villogó kurzort. Ezek után jön a billentyűzet inicializálása. A C port 0-s lába a DATA, az 1-es lába a CLK. Az első, amit észrevehetünk, hogy ezeket a lábakat nem inicializáltuk külön – ezt a `Ps2_Init` eljárás elvégzi helyettünk, ezért nem szükséges. Lényeges az előtte található utasítás, amely az `INTCON` regiszternek 0 értéket ad: ennek részletes értelmezése túlmutat e könyv keretein, a lényeg az, hogy ezzel az utasítással minden ún. megszakítást letiltunk, ami menet közben megzavarhatná programunk működését.



A programban található ciklus feladata, hogy figyelje a billentyűleütéseket, és megjelenítse azokat az LCD-n. Ehhez mindig lekérdezzük a billentyűzet állapotát a `Ps2_Key_Read` függvénnyel, ami 1-et ad vissza akkor, ha történt valami, azaz ha le lett nyomva vagy fel lett engedve egy billentyű. Három `byte` típusú paramétere van, amik hivatkozás szerint adódnak át (`var`-ként vannak deklarálva), azaz a függvény fog értéket adni nekik. Az első azt mondja meg, hogy milyen karakter lett lenyomva. A második 0, hogy ha egy közönséges billentyűről van szó (ekkor az első paraméter ennek ASCII kódját tartalmazza), és 1, ha valamilyen speciális billentyűről van szó, pl. BACKSPACE, F1, ENTER, ESC stb. Ezek kódjait megnézhetjük a súgóban. Az utolsó paraméter 1, ha az adott billentyűt éppen lenyomták és 0, ha az adott billentyűt éppen felengedték.

Ha épp lenyomtak egy gombot, akkor megvizsgáljuk, hogy vajon ez a BACKSPACE (ezt onnan tudjuk, hogy speciális billentyű és 16-os a kódja). Ha igen, akkor kezdetlegesen megvalósítjuk: balra mozgatjuk a kurzort, kiírunk egy üres (SPACE) karaktert, majd mivel ez jobbra mozgatta a kurzort, ezért megint balra mozgatjuk azt. Egyéb esetben, ha közönséges billentyűről van szó, kiírjuk az LCD-re. A ciklus végén található várakozás pergésmentesítés céljából van – a nyomógombkezelésnél megismertekhez hasonlóan.

Ha ettől a lábkiosztástól eltérőt szeretnénk használni itt is használhatjuk a `Ps2_Config` eljárást, aminek első paramétere a port, a második a CLK láb sorszáma, a harmadik pedig a DATA láb sorszáma az adott porton belül (akárcsak az LCD-nél).

A függvények használata során ne felejtsük el a billentyű speciális voltát vizsgálni, ugyanis pl. a 32-es kódú karakter jelentheti a közönséges szóközt is (ha nem speciális) vagy a felfele nyilat is (ha speciális).

A billentyűzet valójában nem ASCII kódokat küld, hanem ún. scan kódokat, ami gyakorlatilag a billentyű sorszáma. Így a Q-é 1, W-é 2, E-é 3 és így tovább. Az általunk használt függvények ezeket értelmezik, és ezeket alakítják ASCII kóddá. Sőt, megjegyzik azt is, hogy a SHIFT le van-e nyomva, és ha igen, nagybetűssé alakítják a szöveget, valamint a CAPS LOCK állapotát is figyelembe veszik.

EEPROM kezelése

Mindennapos probléma az áramszünet és az a jelenség, hogy az elektronikai berendezések elvesztik állapotukat. Számunkra már teljesen természetes, hogy pl. egy televízió hosszabb áramszünet esetén is megjegyezze a beprogramozott adókat, viszont sok (olcsó) berendezés, pl. hordozható MP3 lejátszók nem jegyzik meg, hogy éppen hol tartottak kikapcsolás előtt, pedig ez is igen hasznos lenne.

Most azt nézzük meg, hogy a PIC milyen beépített lehetőséggel rendelkezik valamilyen állapot megjegyzésére áramszünet vagy kikapcsolás esetére. Mi a beépített EEPROM-ot fogjuk használni, ami egy programból többször írható-olvasható memória. Ez a Flash ROM-hoz hasonló memória, viszont szerényebb képességű. A mi szempontunkból az fontos, hogy ebből a memóriából 256 bájt áll rendelkezésünkre és bájtonként tudjuk olvasni és írni (tehát az független az adatmemóriától és a programmemóriától).

Az alábbi program a számrendszeres feladaton alapszik, a különbség annyi, hogy most egy számláló programot fogunk írni, ami 0-tól 255-ig számol (másodpercenként egyet lép), és bináris formában megjeleníti a B portra kapcsolt LED-eken. A kapcsolás rajz változatlan. A különlegesség az benne, hogy ha kikapcsoljuk, majd újra be, akkor ugyanonnan folytatja, ahol előzőleg abbahagyta. Programunk meglepően rövid:

```
program szamlalo;

begin
  TRISB:= 0;
  // előző állapot kiolvasása
  PORTB:= EEprom_read(0);
  repeat
    // növelés
    inc(PORTB);
    // állapot mentése
    EEprom_write(0, PORTB);
    // várakozás
    Delay_ms(1000);
  until false;
end.
```

A program működésében két új utasítást fedezhetünk fel, az egyik az `EEprom_read` függvény, ami az EEPROM-ból olvassa be és adja vissza a paraméterként átadott pozíción található bájtot (jelen esetben a 0. pozíción levő bájtot fogja visszaadni). Ez fogja tárolni azt, hogy honnan kell folytatni a számolást. A program első beprogramozáskor nem kiszámítható ez az érték, viszont ha új a PIC, valószínűleg 0 szerepel ott. Később ide fogjuk menteni azt, hogy éppen hol tartunk. A `PORTB` értékét beállítjuk a függvény által visszaadott értékre. A soron következő ciklus egyszerűen növeli eggyel az `inc` eljárás segítségével a `PORTB` értékét (ha az már előzőleg 255 volt, akkor túlcsordul a `PORTB` és újra 0 lesz). Az `inc`-et (és az eggyel való csökkentést végző `dec`-et) érdemes használni, mert sokkal kevesebb helyet foglal a végső kódban, mint a `PORTB:= PORTB+1` utasítás

és emiatt gyorsabb is. Ezek után elmentjük a PORTB aktuális értékét az EEPROM 0. pozíciójába az `EEprom_write` eljárás segítségével, majd várakozunk egy másodpercig.

Ha bármikor kikapcsoljuk, majd később újra áramot adunk a PIC-nek, az EEPROM-ból visszaolvassa az utoljára elmentett értéket, és onnan folytatja a program futását. Mi csak a 0. pozicióra mentettünk adatot, az EEPROM maradék 255 bájtját nem használtuk.

Ha időnként mentünk az EEPROMba, tartsuk szem előtt, hogy az EEPROM élettartalma véges. Az adatlap szerint 100.000 újraírást visel el (minden egyes bájt), szemben a Flash ROM 1.000.000-szoros újraírhatóságával. Ez azért fontos, mert ha másodpercenként mentünk ugyanarra a bájtra, akkor elvileg csak 100.000 másodpercig garantált az EEPROM megbízhatóságra, ami csak 27 óra. Ezért a legtöbb eszköz csak akkor menti el az állapotát, ha az megváltozott, vagy pedig ha állandóan változik, akkor csak ritkább időközökkel, pl. néhány óránként (ez már több tízéves EEPROM működőképességet biztosít).

Bootloaderek

Állandó probléma a PIC programozók drágasága, nagyon kell rájuk vigyázni, nem körültekintő használat mellett könnyen tönkremennek. További probléma, hogy nehéz őket beszerezni, és egy szakkör esetében nem lehet minden tanuló számára biztosítani az eszközt a magas ára miatt.

Ezeket a problémákat hivatottak a bootloaderek kiküszöbölni. A módszer lényege, hogy ugyan ebben az esetben is programozunk, viszont egy néhány száz forintos, olcsóbb eszközzel. A dolgot az korlátozza, hogy a PIC-ben már egy speciális programnak benne kell lennie, ami levezérli a programozást, ez tulajdonképpen maga a bootloader.

A hagyományos módon történő programozás többnyire magasabb feszültségű égetéssel történik (pl. 14 V). Ha egy PIC programozóba helyezzük a PIC-et, nincs szükség órajelre, ugyanis az órajelet és minden egyebet a programozó eszköz ad a PIC-nek. A PIC-ben egy egyszerű logika van a programozás levezérlésére. Ezzel a módszerrel lényegében minden PIC-et lehet programozni.

Ezzel szemben viszont egy bootloaderre tekinthetünk úgy, mintha egy normális program lenne, amit ugyanúgy először valahogy be kell égetnünk a PIC-be. Ehhez szükségünk lesz

egyszer a PIC élete során egy (drága) PIC programozóra. Viszont később már nincs rá szükség. Bizonyos PIC-ek ugyanis támogatnak olyan utasításokat, amivel a programmemória (a Flash ROM) írható és olvasható – magából a programból. A bootloaderes programozás úgy működik, hogy a PIC-ben elindul a bootloader program (természetesen ehhez már szükséges oszcillátor), majd megnézi, hogy most programozni szeretnénk-e a PIC-et vagy futtatni a beágyazott programot. Ha ez az első programozásunk, nyilván nincs beágyazott program. Ezek alapján átvált programozás módba, majd valamilyen protokoll szerint beolvassa a programot, és beírja a Flash ROM megfelelő helyére. A PIC következő indításánál újra megvizsgálja, hogy most programozni szeretnénk-e vagy a beágyazott programot futtatni, és ha az utóbbi mellett dönt, akkor elindítja az előzőleg beprogramozott beágyazott programot. Ez a lépéssorozat tetszés szerint ismételhető.

A bootloader a program betöltését a PIC lábain keresztül végzi. A lábakat megfelelő módon pl. a számítógép soros portjára csatlakoztathatjuk egy nagyon olcsó hardver segítségével.

A bootloader ugyanúgy a programmemóriában helyezkedik el, mint a később beprogramozásra kerülő beágyazott program. A bootloader lesz a programmemória elején, így vele kezdődik a program futása. Pontosabban szólva az elején csak egy ugrás található a programmemória végére, ahol található maga a bootloader, de az egész bootloader mérete jellemzően nem több 256-512 bájtnál. A beágyazott program mérete ennyivel kisebb lehet csak a programmemória maximális méreténél.

A legtöbb bootloader igényli, hogy az előállított `.hex` fájl kompatibilis legyen vele (ugyanis a `.hex` fájl pontosan megmondja, hogy a programmemória milyen pozícióján milyen szónak kell szerepelnie). Ilyen `.hex` fájlokat előállíthatunk úgy, hogy a mikroPascalban kiválasztjuk a Project Setup ablakban a Build Type-nál a mikroICD Debug opciót (a Release helyett). Ekkor a program nem a legelső bájton kezdődik, hanem egy kicsit később, hogy kompatibilis legyen a bootloaderrel.

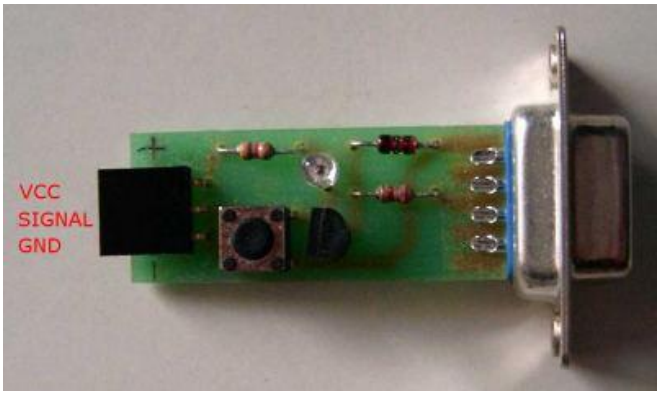
Ipari környezetben, ha használnak is bootloadert, csak a fejlesztési fázisban használják, a kész termék előállítása után már normálisan beprogramozzák a programot, ugyanis a

bootloader egyfajta megbízhatatlanságot eredményezhet (pl. olyankor is átválthat programozás módba, amikor nem is azt szeretnénk).

Az alábbiakban konkrét bootloaderekről lesz szó.

LDRKEY

Az (1) könyvhöz mellékelnek egy bootloadert ill. egy hozzá tartozó kis hardvert is, amivel a PIC-et összeköthetjük a számítógép soros portjával, valamint a hozzá tartozó számítógépes programot is letölthetjük a ChipCAD Kft. honlapjáról.

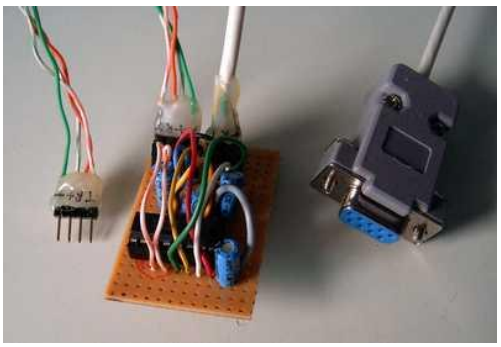


Maga a bootloader a PIC bekapcsolásakor ellenőrzi az RB7 láb állapotát. Ha ez logikai magas, azaz fel van húzva a láb, elindul a bootloader és az RB7 lábon betölti a programot. A letöltő eszközön található LED jelzi a letöltés állapotát, illetve folyamatosan világít, ha valamilyen hiba történt. Az eszközön a lábak nincsenek jelölve, de sorban: VCC, RB7 (SIGNAL), GND.

A program 2002-ben készült, nem nyílt forráskódú és sajnos meglehetősen hibás, pl. nem menti el a beállításokat, megbízhatatlan a programozás, néha megszakad, valamint tisztán egyirányú a programozás. Viszont előnye, hogy a ChipCAD Kft. kb. 100 Ft-ért beégeti helyben a PIC-be a bootloadert, így nincs szükségünk PIC programozóra.

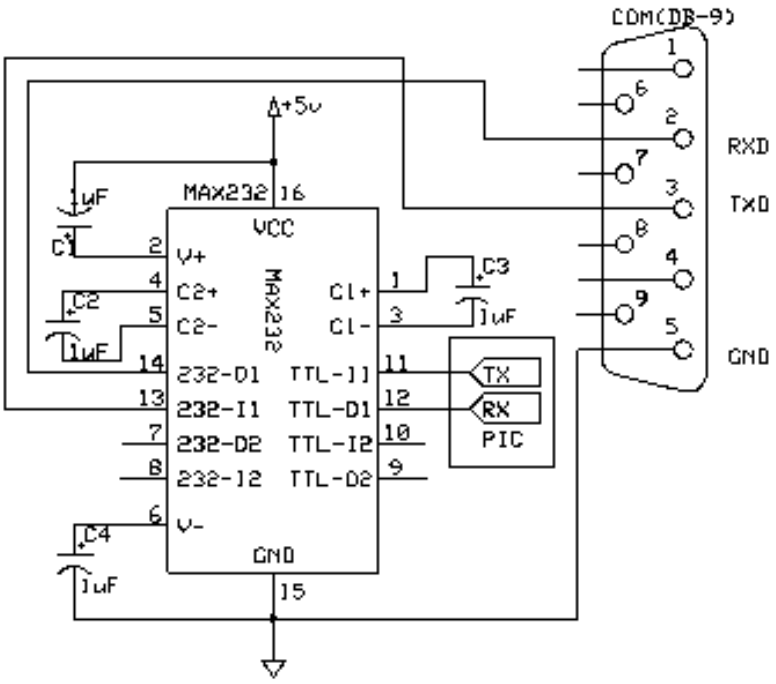
A programozó hardver csupán néhány olcsó elemből áll, így néhány száz forintból magunk és építhetünk egyet egy meglévő alapján.

Microchipc



Ez a bootloader a http://www.microchipc.com/ oldalról tölthető le, ez egy C nyelvű Microchip PIC fejlesztéssel foglalkozó oldal, ahol találhatunk egy bootloadert is.

Ez a programozó a PIC két lábát használja, a TX-et és az RX-et. Ez a kettő tartja a kapcsolatot egy speciális hardver segítségével a



számítógép soros portjával. A hardver kapcsolási rajzán a DB-9-es csatlakozó csatlakozik a számítógép soros portjához (pl. COM1, COM2 stb.), a PIC-beli TX és RX pedig a PIC megfelelő lábai. A feszültség szintillesztését egy speciális IC, egy MAX232 végzi, összességében ez a hardver is olcsó.

Az egész projekt teljesen nyílt forráskódú, bármit módosíthatunk rajta. A letöltő program elég intelligens, ha valahol megszakad a programozás, azt a letöltőprogram azonnal visszajelzi, ezen kívül elmenti az utolsó beállításokat is.

Összefoglaló feladatok és további lehetőségek

Ebben a fejezetben haladó dolgokról volt szó, pl. a billentyűzet kezeléséről és a beépített EEPROM használatáról. Ezen kívül láthattunk néhány széles körben használt bootloadert is, valamint olvashattunk ezek működéséről. Fontoljuk meg ezek használatát, néhány esetben gyorsabban programozhatunk a segítségükkel, mint egy PIC programozóval.

Billentyűzet segítségével rengeteg lehetőségünk van, néhány példa feladatokra (ezeknél a feladatoknál már lehet, hogy túllépjük a 2k-s programmemória-határt a mikroPascalban):

1. Írjunk olyan programot, amely végrehajtja a billentyűzeten begépelt és LCD-n megjelenített parancsokat, pl. LED be- vagy kikapcsolása, pittyegés, háttérvilágítás be- vagy kikapcsolása, stb.

2. Írjunk egy egyszerű játékot. A játékot a billentyűzet kurzormozgató billentyűivel lehet vezérelni, és a pálya az LCD-n jelenik meg. A játék lehet pl. egy hajózós játék, ahol a pálya (a folyó) folyamatosan mozog a hajó alatt, és ki kell kerülni bizonyos akadályokat (pl. bójákat).

Sok mindenről nem esett szó a PIC-ek lehetőségei közül, így pl. a megszakításokról, amelyeket igen előszeretettel használnak a fejlesztők. Szintén nem volt szó az analóg-digitál és digitál-analóg átalakítókról, előbbihez beépített támogatást is találunk a PIC-ben. Sok interfészt is támogat a PIC, így pl. az elterjedt I^2C buszt.

Szintén nem esett szó az RA4 (A port 4-es lába) speciális működéséről, ha kimeneti lábként van konfigurálva, ez ún. open drain lábként funkcionál. Ha lehet, kerüljük a használatát. Nem volt szó a timer (számláló) modulokról, amelyek precíz időzítéseknél lehetnek hasznosak (ugyanis a háttérben automatikusan futnak), ezeket megszakítások segítségével szokás használni, de használhatjuk lekérdezéses módszerrel is (befejeződött-e már egy adott számláló). Pl. a watchdog timer is az egyik timer modult használja, ha be van kapcsolva.

A későbbiekben érdemes még tovább ismerkedni a lehetőségekkel, mivel sok időt takaríthatunk meg vele, ha ismerjük, hogy hogyan kell ezeket használni és mivel mindezt készen kapjuk, nem kell alacsonyabb szintű eszközökhöz nyúlnunk.

A mikroPascal lehetőségeit sem merítettük ki teljesen. Ugyan ez jelenleg is fejlesztés alatt levő környezet, így kisebb-nagyobb változásokon áteshet a közeljövőben, de pl. a többdimenziós tömbök vagy a mutatók témakörét egyáltalán nem érintettük.

A továbbiakban érdemes lehet az assembly nyelvű programozásba is belekóstolni, vagy legalább megpróbálni Pascal nyelven megírni az LCD-t vezérlő függvényeket (az LCD-k adatlapjában benne van minden láb pontos funkciója használata és időzítések, ami alapján meg lehet írni a függvényeket).

A könyv alkalmazhatósága

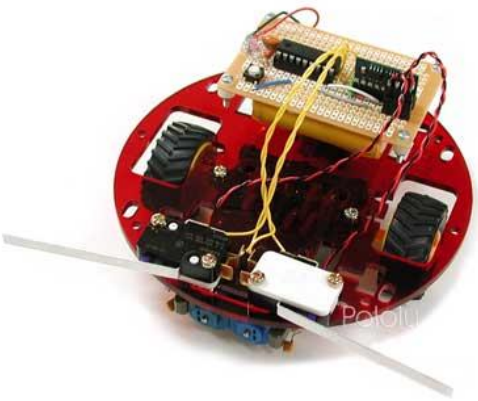
Mikrovezérlő szakkör

Középiskolában jelen könyv tartalmát leginkább egy szakkör keretein belül lehet alkalmazni. Itt azok az érdeklődő tanulók sajátíthatják el az adott témakört, akik a megfelelő előismeretekkel már rendelkeznek. Szinte biztos, hogy ezt a feltételezett tudást a szakkör keretein belül el kell elmagyarázni a tanulóknak, valamint meg is kell győződni arról, hogy a megfelelő szinten sikerült elsajátítaniuk. Előfordulhat emiatt, hogy igen hosszú idő telik el addig, amíg ténylegesen működő áramkört készítenek a tanulók. A tanár felelőssége, hogy megfelelő arányban tanítsa az elméletet és gyakorlatot, hiszen túl sok elmélet után a téma egyre kevésbé lesz érdekes, ha viszont túl sok gyakorlati feladat van, a tanulók nem tudják önállóan megcsinálni, és hiába látványos, ha nem értik kellő mértékben a működését.

A könyv teljes tartalmát egy szakköri év keretein belül leadni nem egyszerű feladat akkor sem, ha a tanulók már rendelkeznek a megfelelő ismeretekkel, kell időt hagyni arra, hogy maguk kísérletezzenek, játsszanak az eszközökkel, ugyanis csak így tudják igazán mélyen elsajátítani a részleteket. Minden bizonnyal a tanulók otthon is szívesen fognak ezzel foglalkozni, így a szakkörön csak az esetlegesen felmerülő problémákat, valamint a nehezebb témaköröket szükséges alaposabban megtárgyalni.

Ez a téma kiválóan alkalmas a projektmunkára, 2-4 tanuló közösen készíthet valamilyen kapcsolást. Tapasztalataim szerint igazán változatos ötletek születnek, amit meg is tudnak valósítani a tanár segítségével. Néhány ilyen ötlet:

- Önjáró robot építése. Ennek egyszerűbb változata az ún. gearbox, ami egy két külön vezérelhető kerék által meghajtott mini-jármű, esetleg mechanikus ütközésérzékelővel ellátva. Erre lehet pl. olyan programot írni, ami faltól falig megy, de sohasem ütközik teljesen a falnak.



- Riasztó: igen komplex riasztóberendezést is készíthetünk, amivel elláthatjuk és figyelhetjük az egész házat. Különböző mozgásérzékelők segítségével igen professzionális riasztóberendezést is megvalósíthatunk.
- Fedélzeti számítógép gépjárműbe. Egyre jellemzőbb a „kocsi-moddolás” ezen változata, amikor egy régi autóba barkácsolunk fedélzeti komputert, ami mutatja a sebességet, megtett távolságot, fogyasztást stb. Igen komoly ismereteket igényel és nem teljesen veszélytelen a feladat.
- Automatizált SMS írás mobiltelefonon: olyan szerkezet készítése, amely a mobiltelefon gombjainak nyomkodására képes, így egy előre megtanult módszer szerint pl. SMS írásra képes. Ha összekötjük a számítógéppel, akár a gyorsan megírt SMS-t is elküldhetjük a szerkezet segítségével.
- Meteorológiai központ: egyszerűbb esetben digitális hőmérő, bonyolultabb esetben komplett meteorológiai központ építése légnyomás- és páratartalom-mérővel, esett csapadékmennyiség, szélirány kijelzéssel. Az adatokat akár számítógépen is archiválhatjuk, ott pedig összetettebb kimutatásokat is készíthetünk belőle.

A tanulók nagyon kreatívak, a mechanikai megvalósítást, ha mással nem, LEGO elemekkel megoldják. A LEGO-nak egyébként van egy programozható termékcsaládja is, a LEGO Mindstorm. Ez meglehetősen drága, cserébe viszont komplett környezetet kapunk, a programokat különösebb programozási tudás nélkül, vizuális eszközökkel is megvalósíthatjuk.

A legfontosabb kérdés, hogy az iskola az eszközökből mennyit szerezzen be. Ahhoz, hogy hány felszerelést vegyünk, először azt kell eldönteni, hogy hány tanuló dolgozhat egyszerre. A PIC programozóból magas ára miatt egy iskolában elegendő egy, viszont érdemes megfontolnunk a bootloader használatát, ezzel gyorsíthatjuk a programozást (minden PIC-hez egy-egy PIC bootloader programozó építésével), így párhuzamosan is tudnak a tanulók dolgozni. A drága alkatrészeket (pl. LCD) csak a tanár felügyelete mellett engedjük használni.

Úgy gondolom, hogy a tanulók nagyon fogják élvezni a szakkört, ugyanis ilyet csak nagyon kevés középiskolában tanítanak.

Verseny

Ha a tanulók már túl vannak a kezdeti lelkesedésen, felkelthetjük bennük a versenyszellemet például egy programozói verseny meghirdetésével. Ez lehet iskolai szintű, viszont időnként megrendezésre kerül egy országos szintű verseny is, amelyet a győri Jedlik Ányos Gépipari és Informatikai Középiskola hirdet meg, „Országos Mikrovezérlő-alkalmazói komplex tanulmányi verseny" címmel. Ez a verseny assembly nyelven követeli meg a feladatok megoldását, így a feladatok is jóval egyszerűbbek, pl. 7 szegmenses kijelző meghajtása. A http://www.jagik.sulinet.hu/ honlapról bővebben tájékozódhatunk.

A versenyeredmények alapján láthatjuk, hogy több középiskolásnak sikerült elmélyednie ebben a témában, ráadásul assembly nyelven.

Fejlesztett kompetenciák

Érdemes röviden áttekinteni, hogy milyen kompetenciákat fejleszt, ha valaki mikrovezérlők programozásával foglalkozik.

Először is, fejleszti a gondolkodást, hogyan tudunk olyan programot írni, ami a legkisebb eséllyel hibázik és megbízhatóan működik. Fejleszti a rendszerező képességet is, ugyanis a tanulók megpróbálják megtalálni a kapcsolatokat az eddig általuk megismert architektúrákkal, pl. a számítógépekkel. A korábban felsorolt ötletek után már nem is kell részletezni, hogy fejleszti a kreativitást, az ötletgazdaságot a téma. Megfigyelhető, hogy akik elektronikával foglalkoznak, legtöbbször olyan szerkezeteket építenek, amelyek valamilyen szempontból mindennapi életüket könnyítik meg, segítik vagy gyorsítják, vagy egyszerűen csak új megoldást adnak egy már létező problémára. A tanulók ezzel kiterjesztik szemléletmódjukat, nyitottabbak lesznek az új ismeretekre, könnyebben befogadják majd azokat. Fontos, hogy ne féljenek az elektronikától, hiszen gyengeáramról van szó, legfeljebb anyagi kár keletkezhet, és igenis merjenek új ötleteket kipróbálni.

Fejleszti a problémamegoldás képességét: a hibakeresés ebben az esetben még nehezebb, mint számítógép esetében, sokszor külső segítséget is igénybe kell venni, viszont idővel már önállóan is képesek lesznek a hibákat feltárni és kijavítani. Néhány hónap után már rutinná válik, egyre gyorsabban megtalálják a hibákat és egyre kevesebbet vétenek, tanulnak korábbi hibáikból. A hibák elkerülése végett próbálják a

legjobb megoldást kiválasztani. Mivel a program fejlesztése sok ideig eltarthat, így egy-egy program megírásánál jobban átgondolják a tennivalókat: információgyűjtés az adott témáról, milyen meglévő elemeket lehet felhasználni, hogyan lehet (lehet-e) azokat egymáshoz illeszteni, majd a rendszer megtervezése után annak implementálása majd tesztelése.

Valamint fejleszti a munkastílus kompetenciákat: a módszeres munkavégzés, a tanuló képes meghatározott lépések szakszerű elvégzésére, de benne van az újítás vágya is. A figyelem összpontosítása és a megfelelő hozzáállás a projekthez szintén előtérbe kerül.

Oktathatóság

A mikrovezérlők programozása kétségkívül igen nehéz téma. Rengeteg mindennel kell tisztában lenni ahhoz, hogy pontosan értsük, hogy mi is történik. A PIC-eket architektúrálisan kell ismerni ahhoz, hogy ismerjük a program pontos futtatásának menetét. A programozási nyelvet nagyon magabiztosan kell használni, hiszen nehézkes a program nyomkövetése (habár léteznek hardveres ún. in-circuit debuggerek is, amelyeket a kész áramkörben használhatunk, de ezek drága eszközök). Miután megírtuk a programot, ki is szeretnénk próbálni. Ehhez alapvető elektronikai ismeretek nem elegendőek, hiszen ismernünk kell az alkatrészeket is, megfelelően kell őket csatlakoztatnunk. Tapasztalataim szerint ez az egyik legnehezebb pont, mert erről nehéz megfelelő irodalmat találni.

Egy segítő, egy tanár szerepe kulcskérdés az elindulásnál, ugyanis kezdetben „semmi sem működik”. Olyan sok a hibalehetőség, hogy nagyon valószínű, hogy valamit elrontunk. A könyvben próbáltam mindenből a legbiztosabb, legkönnyebb megoldást bemutatni. Sajnos sok helyen az ár a fő döntő tényező, ezért pl. PIC programozókból bemutattam többfélét, az előnyökkel és hátrányokkal együtt, ezek ismeretében már könnyebb a választás. Általában igaz, hogy ha több pénzt szánunk rá, később kevesebb gondunk lesz vele.

Az elektronika világában szinte már elvárás, hogy valaki azon kívül, hogy ismeri az alkatrészeket, tud forrasztani, sőt nyomtatott áramkört (nyákot) tervezni is. PIC bootloader programozóhoz pl. igen nehéz hozzájutni, így ha tudunk forrasztani, legjobb, ha megépítjük. Ha ebbe is csak most tanulunk bele, ne törjük a fejünket nyomtatott

áramkörök tervezésén és maratásán, lehet kapni forrasztható próbapanelt is, használhatjuk azt. Természetesen a protoboardon is megépíthetjük.

Sokszor előfordul, hogy valamilyen megmagyarázhatatlan hibajelenséget tapasztalunk. PC esetében sem mindig a programban van a hiba, itt is hasonló a helyzet, csak itt még nagyobb valószínűséggel lehet a hardverben, de még inkább a kapcsolásban a hiba.

Összefoglalva sok idő, energia és pénz, de olyan ismeretekhez jutunk hozzá, ami minden elektronikai berendezésben használatos jelenleg, ezáltal jobban átlátjuk az informatika alacsonyabb szintű rétegeit és olyan ismeretekre teszünk szert, ami megváltoztathatja a programozásról eddig alkotott képünket. Sok sikert kívánok a firmware-programozáshoz!

Irodalomjegyzék

1. **dr. Kónya, László.** *PIC mikrovezérlők alkalmazástechnikája.* hely nélk. : ChipCAD Kft., 2003.

2. **mikroElektronika.** *mikroPascal Help.*

3. **Madarász, László.** *A PIC16C mikrovezérlők.* Kecskemét : Gépipari és Automatizálási Műszaki Főiskola, 1996.

4. **Priesterath, Willi.** *Hobbielektronika.* Budapest : Cser Kiadó, 1999.

5. **Benkő, Tiborné, és mtsai.** *Programozzunk Turbo Pascal nyelven!* Budapest : Computerbooks, 1996.

6. Fairco oldala. [Online] 2008.03.02.. http://www.freeweb.hu/fairco/.

7. Hobbielektronika. [Online] 2008.03.02.. http://pic.hobbielektronika.hu/kapcsolasok/_pic.html.

8. Jedlik Ányos Gépipari és Informatikai Középiskola, Elektronika szakmacsoport, Győr. [Online] 2008.03.02.. http://www.jagik.sulinet.hu/.

9. Microchip. [Online] 2008.03.02.. http://www.microchip.com/.

10. Mikrovezérlő.lap.hu. [Online] 2008.03.02.. http://mikrovezerlo.lap.hu/.

11. Rádióvilág szerkesztőség honlapja. [Online] 2008.03.02.. http://www.radiovilag.hu/.

12. Tudomány és Technika. [Online] 2008.03.02.. http://www.t-es-t.hu/.

13. Wikipedia. [Online] 2008.03.02.. http://www.wikipedia.org/.

14. **Istvánfi, Béla és Braun, Gábor.** Ajánlott összefoglaló PIC kezdőknek. [Online] 2008.03.02.. http://www.elektronline.hu/konyvtar/other/pickezdo/PICkezdo.htm.

15. **Juhász, Róbert.** [Online] 2008.03.02.. http://plc.mechatronika.hu/piclei/piclei.htm.