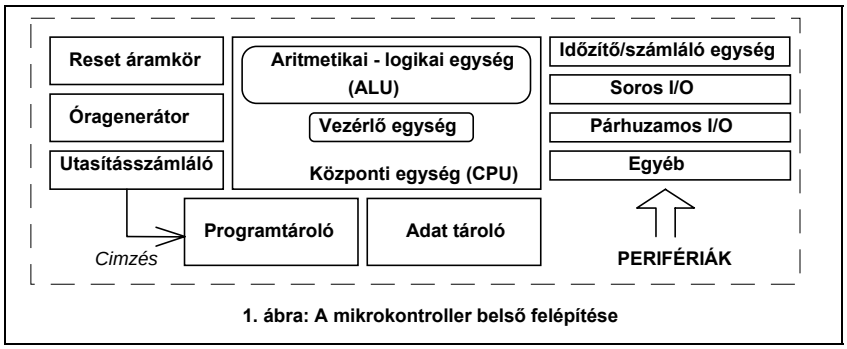
PIC mikrokontrollerek alkalmazástechnikája

Bevezetés - a sorozat elé ...

A következőkben a Rádiótechnika hasábjain egy több részes sorozatot indítunk, amelynek célja a korszerű elektronikai ismeretek megismerte-ése, azon területé, amely a mikrovezérlőkkel vagy más néven mikrokontrollerekkel foglakozik.
A történetet jól ismerjük: az elektroncsövek után jöttek a félvezetők majd az ezekből felépülő integrált áramkörök. Az integráltsági fok növekedésével lehetővé vált egyre bonyolultabb áramkörök kialakítása, megjelent az egy lapkán kialakított számítógép központi egység: a mikroperocesszor. Ahhoz hogy egy teljes működő egységet, egy számítógépet kialakítsunk, az információfeldolgozás általános modellje szerint memóriára és be-kimeneti egységekre is szükség volt. Ezeket az egységeket külön lapkákon helyezték el, és ezek kapcsolatát nyomtatott áramköri huzalozásokkal oldották meg. Igy megjelent az egy NYÁK-lemezen felépített számítógép: a Single Board Computer. Az integráltsági fok további növekedése - ami még több tranzisztor egy lapkán való elhelyezését jelentette - két irányban való fejlődést eredményezett: egyrészt biztosította hogy a mikroprocesszor még összetetebb, többet tudó eszközzé alakuljon (Intel: 8080 -> 8085 ->8086 ->80286 ->80386 ->80486 ->Pentium ->P6), másrészt lehetővé vált hogy a központi egység (a processzor) mellé az eddig külön lapkán kialakított elemeket (memória, perifériák) is elhelyezzék. A gyártástechnológia lehetővé tette, hogy az áramkörökkel szembeni kívánságok: pl. kis fogyasztás, nagy sebesség is megvalósulhassanak.
Az ilyen módon, egy tokban elhelyezett központi egységet, memóriát, ki-bemeneti egységeket és járulékos áramköröket tartalmazó rendszert hívjuk egytokos mikroszámítógépnek vagy más néven mikrokontrollernek (1. ábra).
A **Központi egység (CPU)** feladata a **Programtároló**-ban tárolt utasítások végrehajtása, az utasítások végrehajtásához az **Aritmetikai logikai egység (ALU)** - t használja fel a számításokhoz. A **Vezérlő egység** felelős az egyes részegységek jeleinek ütemezéséért. Az **Óragenerátor** szolgáltatja a rendszert működtető órajelet. A **Reset** áramkör az egész rendszer alaphelyzetbe hozását végzi el - például tápfeszültségre kapcsoláskor. A mikrokontroller a **perifériáin** keresztül kapcsolódik a környezetéhez, azaz a külvilághoz. Az **Utasításszámláló** tartalma határozza meg, hogy a programtároló melyik regiszteréből olvassuk ki az utasítást tartalmazó kódot.
A mikrokontrollerek felhasználása két tevékenységet igényel:

- meg kell tervezni a feladatnak eleget tevő áramkört,
- meg kell írni a kontroller által végrehajtandó programot, amivel az adott feladat megoldható.

A lapot olvasók közül sokan talán most kezdik keresni azt a sarkot, ahova az újságot vághatják. Ugyanis az áramkörépítésen nevelkedettek -"a hardveresek" - sokszor nehezen képesek egy módszerében más, "szoftveres" tevékenységet alkalmazni, pedig a mikrokontrollerek esetében ez a két tevékenység szorosan összefonódik. Természetesen a programozás "művészet" is lehet, de bizonyos alapismeretek megtanulásával, előre megadott minták alkalmazásával, bárki képes mesterember szintjén a programozást megtanulni, és hatékonyan használni. Ezért kérem, hogy a sarkok méregetése helyett bízzanak magukban (meg a cikk írójában...) hogy túl fognak jutni ezeken a nehézségeken.



1. ábra: A mikrokontroller belső felépítése

A világon számos mikrokontroller típus található. A sorozat megmaradhatna egy általános megközelítés szintjén is, de ezen típusok között van egy olyan - az eladási adatok szerint dinamikusan fejlődő - típus, amely egyre népszerűbb. Ezt alacsony ára, nagy teljesítménye, rugalmassága, könnyű fejleszthetősége és a hozzá adott vagy könnyen hozzáférhető dokumentált alkalmazások nagy száma biztosítja. Nem elhanyagolható az tény sem, hogy hazánkban ez a típus könnyen hozzáférhető a **ChipCAD Kft**-nél.

Ez a típus a Microchip cég által gyártott PIC család. (Itt a PIC a Programmable IC = programozható IC rövidítése.) Család azért, mert több ága és tagja van, amelyek használatával az alkalmazások ár/teljesítmény viszonya a legkedvezőbbé tehető.
A sorozat ezen típus alkalmazás technikáját kívánja bemutatni, sok - sok konkrét alkalmazáson keresztül. Az alkalmazástechnikai jellegből adódóan felhasználunk minden rendelkezésre álló információt az ismeretek megfelelő szintű közlésére.
Mivel a sorozat szerzője egy főiskolán dolgozó oktató, ezért a sorozat irásánál igyekezett a didaktikai követelményeket is figyelembe venni, azaz az új ismereteket mindig a már megmagyarázottakra alapozni. A terjedelem szabta korlátok miatt ez természetesen csak korlátozott mértékben volt lehetséges, ezért folyamatosan utalunk azon irodalmakra amelyekben az ismertetett

információk részletesen megtalálhatók.
Milyen ismereteket tételeztünk fel a sorozat megírása során? Egy átlagos technikai műveltséget, a logikus gondolkodás használatát, alapvető áramköri ismereteket és a bináris számok ismeretét. Természetesen szükséges néhány angol szó és kifejezés jelentésének a megjegyzése is, de ezeket a zárójelbe tett magyar megfelelőjükkel megmagyarázzuk. Az sorozatot úgy kívánjuk közreadni hogy a részeknek legyen egy olyan íve, fokozatosan nehezülő foka, ami az olvasókat is inspirálja a következő szám keresésére, elolvasására.
Ez után a kissé hosszúra sikeredett, de fontos bevezető után lássunk munkához.

A PIC16/17 mikrokontroller család

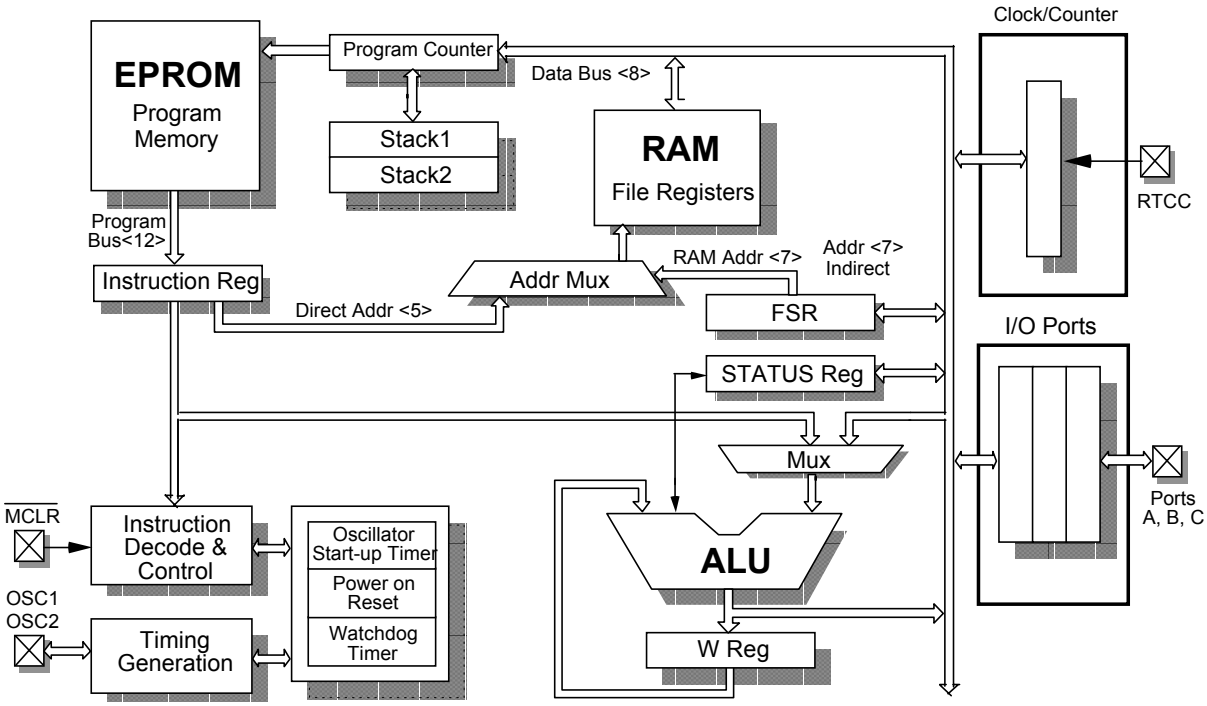
A PIC mikrokontroller család gyártója, a Microchip Technology Inc. cég központja az USA Arizona államában Chandler-ben található, de a világ számos pontján vannak részlegei. A Mikrochip történet 1990-ben a General Instruments félvezető gyárainak privatizálásával, a PIC16C5X eszközcsalád piacra dobásával indult. Az előretörés soha nem látott mértékű ebben az iparágban. Eladása szinte hihetetlen 60-70%-kos mértékben növekszik folyamatosan minden évben messze az ipari átlag felett, aminek a következtében 1993-ra a 8. helyre lépett elő. 1994-ről még nincs részletes adat, de az előzetes hirek szerint a Philips-et is sikerült tavaly megelőznie. Az idei évben az első 3 hely valamelyikének a várományosa.
Hogy megértsük ezen sikerességnek az okát, vizsgáljuk meg, hogy a felhasználók mit várnak el egy mikrovezérlőtől:

- kis fogyasztása legyen,
- gyors legyen: időegység alatt sok utasítást tudjon végrehajtani,
- rugalmas, könnyen változtatható perifériakészlete legyen,
- különböző alkalmazásokhoz az optimalizálás miatt más és más családelemet lehessen választani.
- egyszerűen és gyorsan lehessen az alkalmazásokat fejleszteni, és ehhez szükséges fejlesztőeszközök ne legyenek túl drágák,
- a mikrokontrollerek ára is alacsony legyen, nagy szériáknál ez még kisebb legyen.

A Microchip PIC16/17 kontrollerei a piacon elérhető nyolcbites eszközök között az egyetlen, amely RISC felépítésű: azaz kevés számú utasítást tartamazó utasításkészlettel rendelezik. Az utasításkészlet kódolását úgy tervezték meg, hogy az utasítások mindegyike egyetlen program memória címen eltárolható legyen. A következmény:

- rendkívül gyors működés (5 millió utasítás végrehajtása másodpercenként 20MHz-es órajel esetén)
- a program memória rendkívül hatékony kihasználása

A 2. ábrán az alaptípus, a PIC16C5X sorozat felépítése látható, amely alapja a család többi, nagyobb teljesítményű tagjának is.
Összehasonlítva az 1. ábra vázlatával az ott blokként szereplő elemek már kissé részletesebben is láthatók és röviden összefoglajuk a részegységek szerepét.



2. ábra

Az első fontos megjegyzés, hogy a kontroller ún. Harvard architektúrájú: ez azt jelenti, hogy a programmemória (Program Memory) és az adatmemória (RAM File Registers) elkülönül egymástól. (A PC-ben például ez nem így van: ott az operatív memória mind a végrehajtandó program utasításait, és az adatokat is tartalmazza. Ezt hívják kigondolója, Neumann János után von Neumann architektúrának.)
Milyen előnnyel jár ez a felépítés? Ez azért kedvező, mert az adatszavak hossza (8 bit) nem kell hogy megegyezzen az utasításszavak hosszával (itt ez 12 bit széles).
Ez a döntés teszi lehetővé, hogy a legtöbb utasítás csak 1 szóból álljon. Ugyanis sok utasítás végrehajtásához operandusokra is szükség van. Az egyszerűség kedvéért az egyik operandus mindig egy kitüntett, az akkumulátornak nevezett regiszter (az ábrán ez a W regiszter), míg a másik egy, a RAM-ban szereplő regiszter.

Az utasítás végrehajtásához ennek a címét is ismerni kell. Az egyik megoldás, hogy az utasítás nem egy szóból áll, és ekkor a második szó tartalmazza ezt a címet. Ez nem előnyös, mert kevesebb utasítást tud a CPU végrehajtani időegység alatt. A másik megoldás az, hogy az utasítást tartalmazó szó két részből áll:

UTASÍTÁS KÓD	2. OPERANDUS CÍME

UTASÍTÁS SZÓ

az egyik része határozza meg (kódolja) a végrehajtandó tevékenységet, míg a másik rész tartalmazza a második operandus címét:
Ezzel a megoldással a legtöbb utasítás mindössze egy tárolóhelyet foglal el a programmemóriában. Természetesen minél hosszabb az utasításszó, annál hosszabb lehet az utasításkód rész (= több fajta utasítás) és hosszabb lehet az operanduscím rész is (= nagyobb közvetlenül címezhető memóriaterület.) Ezért a növekvő teljesítményű PIC vezérlőkhöz növekvő utasításszó hossz is tartozik (12, 14, illetve 16 bit).
Mivel a címet tartalmazó rész rögzített, ezért a megadható címtartomány is rögzített. Pl. 8 bites címrésszel csupán 256 féle cím képezhető. Ha ennél nagyobb terjedelmű memóriát kívánunk címezni, akkor az ún. "lapozásos" technikát alkalmazzuk: a tényleges cím további bitjeit egy külön tárolóban a lapregiszterben tároljuk. A lapregiszter tartalma határozza meg hogy ténylegesen a memória melyik lapját használjuk. Ezzel a megoldással elvileg tetszőleges nagyságú memóriarész címezhető, de van egy kis szépséghibája: lap átlépésekor a lapregiszter tartalmát is módosítani kell. A PIC kontrollerek mind az adat- mind a program memória címzésénél alkalmazzák ezt a lapozási technikát.

Program-memória

A központi egység a végrehajtandó utasításkódokat a program memóriából olvassa ki. Ez a tár a különféle igények miatt különböző típusú: ROM, EPROM, EEPROM és OTP lehet. Mit is jelentenek ezek a rövidítések ?

- **ROM memória**: ennek tartalmát a gyártás során írják bele a memóriába. Ez - mivel nagy darabszám esetén a legolcsóbb memóriamegoldás - akkor előnyös, ha nagy mennyiségben akarjuk az adott felprogramozott PIC vezérlőt felhasználni. Természetesen ilyenkor a gyártóhoz el kell juttatni a ROM-ba írandó pontos tartalmat, és ez írják be a gyártástechnológia adott fázisában a ROM-ba.

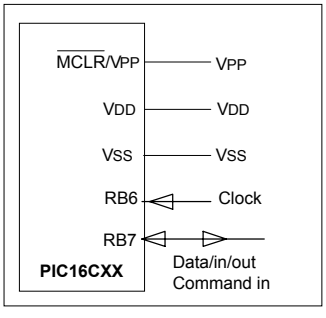
- **EPROM memória**: ez a memóriatípus olyan, hogy a felhasználó képes ennek tartalmát beírni, azaz programozni, majd új tartalom beírása előtt ultraibolya fénnyel (pl. kvarclámpa) törölni („kinapoztatni"). Ennek a típusnak az az előnye, hogy a programfejlesztő egyszerűen tudja programját a PIC EPROM-jába írva kipróbálni, módosítani.

- **EEPROM memória**: hasonló az EPROM memóriához, de itt a törlés a „kinapozás" helyett elektromos jelekkel történik, így programfejlesztéshez az ilyen típusú memória még előnyösebb.

- **OTP memória**: az előbbiekben már megállapítottuk, hogy a legolcsóbb memóriatípus a ROM. Ennek használata azonban igényli a chipgyártóval való szoros kontaktust. Jó lenne olyan memória, amely olcsó, a felhasználó által egyszer programozható, de nem kell ezt a programozást a gyártónál elvégeztetni. A megoldás : az OTP (nem takarékpénztár!) memória, (OTP = One Time Programming (egyszer progra-mozható) amely az előbb leírt követelményeknek eleget tesz.

A PIC kontrollerek esetében mindegyik kialakítás létezik, amely nagyon költségkímélő megoldást biztosít.
Az EPROM, EEPROM, és OTP típusok programozása nagyon egyszerű: a Microchip a programozásra két megoldást ajánl: a gyorsabb párhuzamos, és az egyszerűbb soros módszert. Soros programozásnál a program memóriába

Soros programozás



- Csak 2 láb kell a programozáshoz
- RB6 az órajel bemenet
- RB7 az adat be/ki- vagy a parancs bemenet
- Parancsok:
  - Load data (adatírás)
  - Read data (adatolvasás)
  - Begin programming (programozás indul)
  - End programming (programozás vége)
  - Increment address (memóriacím növelése)

3. ábra

kerülő szavakat bitenként írjuk két vezeték felhasználásával (adatvonal és az órajel). A programozáshoz a Vdd tápfeszültégnél nagyobb feszültség (Vpp) szükséges. A programozás idődiagrammját a gyártó a mikrokontrollerek adatlapján közli, és a rendszerfejlesztésekhez használt programozók általában ezt használ-ják.

Adatmemória

A programok végrehajtása során adatokkal műveleteket végzünk. Az adatokat tartalmazó regisztereket - amelyek a RAM memóriát alkotják - a gyors elérés érdekében célszerű a központi egység számára elérhető módon kialakítani. A PIC mikrovezérlők esetében ez a **File Registers** néven szereplő RAM terület, amelynek néhány fontos tulajdonsága van:

- Bármelyik regiszter lehet az ALU által végzett műveletek egyik operandusa.
- Az itt lévő regisztereknek két fajtájuk van: vannak speciális adatokat tartalmazó, rögzített című regiszterek, amelyek fontos szerepet játszanak a kontroller működésében és programozásában, valamint vannak általános célú regiszterek, amelyek tetszőleges adatok tárolására használhatók.
- Ezen utóbb említett regiszterek-ben tárolódnak a kontrollerben lévő perifériák programozásához és adatkezeléséhez szükséges regiszterek.

Órajel megoldások

Egy digitális rendszer a legtöbb esetben működéséhez órajelet igényel, amit egy oszcillátor áramkör állít elő. PIC kontrollerek esetén különféle órajelmegoldások közül választhatunk: a megoldás frekvenciatabilitása és felhasznált alkatrész ára alapján:

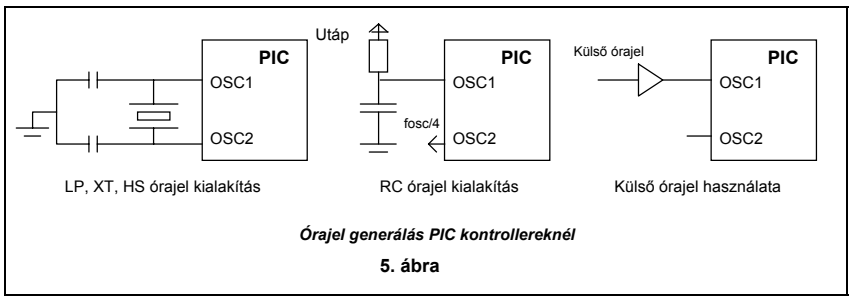
LP- Low Power Crystal (kisfogyasztású kvarc)
XT- Kvarc/kerámia rezonátor
HS- Nagyfrekvenciájú kvarc/kerámia rezonátor
RC- ellenállás - kondenzátor

Minden PIC mikrovezérlőn van két kivezetés: az OSC1 és OSC2 elnevezésű, amelyekre kell kötni az alkalmazni kívánt órajel előállító alkatrészeket:
A tokban egy belső konfigurációs bitpárral kell megadni, hogy milyen típusú órajel kialakítást használunk.

RESET áramkör

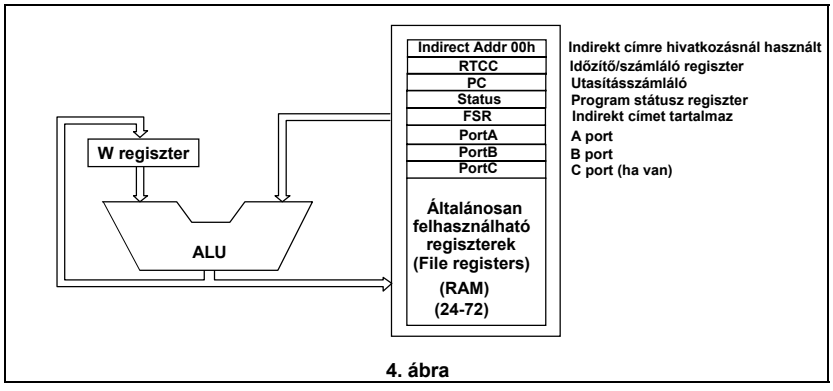
Digitális rendszerek bekapcsolásakor szükséges egy jól definiált alaphelyzet beállítására, mert csak ez biztosítja a rendszer helyes működését. Ez a RESET folyamat. A tápfeszültség bekapcsoláskor emelkedik a tápfeszültség és az oszcillátor is működni kezd. Nyilvánvaló, ha túl lassan növekszik a tápfeszültség, vagy az oszcillátor frekvenciája lassan stabilizálódik, a RESET folyamat helytelenül játszódhat le. Ez egy kontrolleres rendszer megbízhatóságát nagy mértékben csökkenti, mert esetleg a rendszert többször ki be kell kapcsolgatni. Ezt elkerülendő a PIC-ekben táp-eszültség bekapcsolása után egy speciális folyamat játszódik le:
Bekapcsoláskor a tápfeszültség emelkedik. Amikor egy adott szintet elér, egy belső, tokon belüli oszcillátorról működő számláló áramkör elindul (a neve DEVICE RESET TIMER vagy POWER UP TIMER) és tipikusan T_{PWRT}=18 msec ideig még fenntartja a tok RESET állapotát. Ez idő alatt képes a tokot működtető külső oszcillátoros órajelgenrátor elindulni és frekvenciáját stbilzálni. És ha ez idő alatt mégsem stabilizálódik a külső oszcillátor? Akkor baj van. Ezért az újabb PIC típusoknál (16CXX, 17CXX) a RESET folyamatot még egy egy újabb késleltetést biztosító időzítő megnyújta: ez az OSC START UP TIMER .
A PIC kontrollereknél egy külön láb (a neve: MCLR) alacsony, majd magas szintre állításával is előidézhető bármikor egy RESET.

Ha erre nincs igény (ami az alkalmazások többségénél igaz), akkor a fentiek miatt ez a láb egyszerűen a tápfeszültségre köthető, és a tápfeszültség tokra kapcsolása RESET-eli a mikrokontrollert. Mindezeket a 6. ábrán foglaltuk össze.



Órajel generálás PIC kontrollereknél

5. ábra

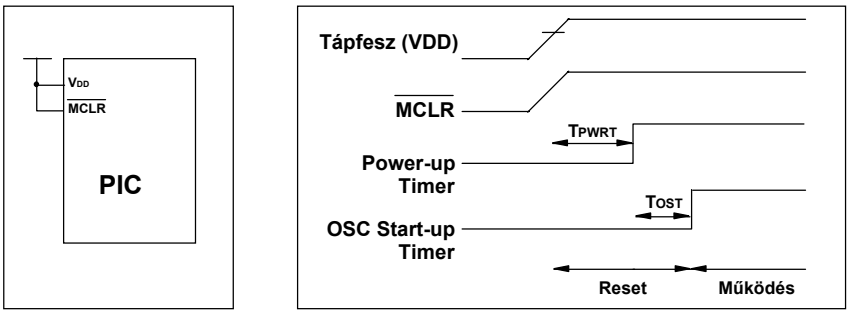


4. ábra

A Watch Dog Timer (WDT)

A mikrokontroller folyamatosan működése során egy ciklikusan ismétlődő utasítás sorozatot hajt végre. Egyik utasítás jön a másik után, és abban az esetben ha valamilyen külső ok miatt (áramköri zavar) az utasítás tévesen kerül beolvasásra, ez a ciklikusság megszűnik, és a futó program "eltérül", és nem tér vissza a

ciklusba. Mi a megoldás az ilyen problémák megoldására? **A watch dog**, szó szerint fordítva: **órakutya**. Maga a kifejezés onnan származik, hogy amerikában az éjjeliőröknek a nyakukban lógó kulccsal az őrzött objektum különböző pontjain pl. óránként lejáró órákat kell újra és újra felhúzni. Ha nem húzza fel időben, az óra lejár, és jelzi a mulasztást. A watch dog áramkör is így működik: egy önállóan működő számlálót a ciklikus programban elhelyezett utasítással (CLRWDT) törölni kell. Ha ez nem történik meg akkor a számláló lejár és ez RESET folyamatot indít el, ami a kontrollert újra elindítja, vagyis a program csak a számláló (neve: watch-dog timer) lejártáig "kószálhat".
Mivel nem minden esetben van szükség a WDT használatára, ezért egy belső konfigurációs bit programozásával engedélyezhetjük, vagy tilthatjuk a működését.

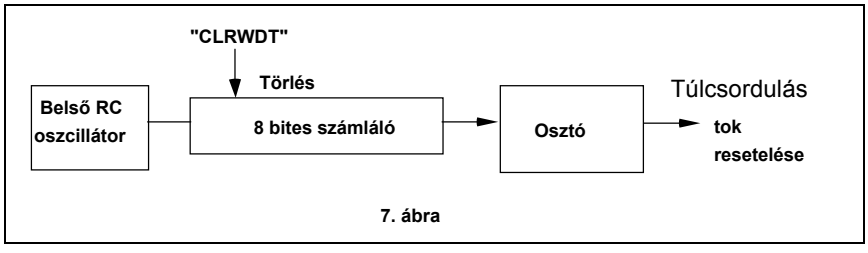


6. ábra

A kontroller periféria egységei

A következőkben röviden összefoglaljuk a PIC kontrollerekben alkalmazott legfontosabb periféria egységeket, néhányról majd a konkrét alkalmazások ismertetésekor ejtünk néhány szót.

Kétállapotú ki-bemeneti egységek - más néven digitális I/O portok: Mivel a kontroller bináris adatokkal dolgozik, a legkézenfekvőbb, ha a külvilággal való kapcsolat is ilyen kétállpotú jelekkel valósul meg. Mivel csak egy konkrét alkalmazás esetén tudjuk már pontosan, hogy hány be- illetve kimenetre van szükség, ezért a PIC-ek esetén az I/O vonalak mindegyike akár bemenet, akár kimenet lehet, és ezt a programfutáskor állítjuk be. Ezen I/O vonalaknak a RAM - ban lévő regiszterek felelnek meg: az irányregiszterek bitjeibe írt 0-ák jelzik, hogy a hozzátartozó vonal kimenet, 1-esek pedig az adott I/O vonalat bemenetre programozzák. Az adatregiszterek hordozzák az adatokat. A PIC kontrollerek esetén a kimenetek terhelhetősége nagy: képesek közvetlenül LED-eket meghajtani.



7. ábra

A kontroller a külvilággal kapcsolatban álló adatok alapján működik, és az eredmény is oda kell hogy kerüljön. Általánosan egy perifériális egység a működése során négy típusú jelet használ:

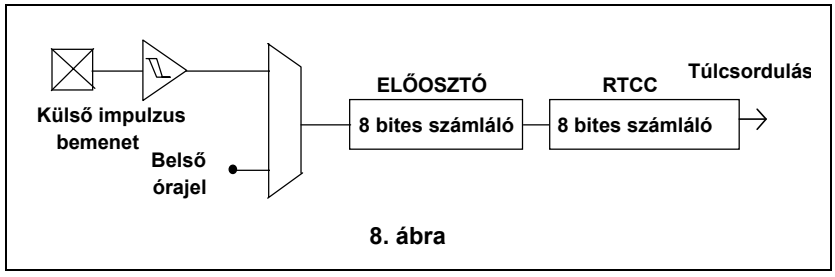
- bemenő adatok - ezek jönnek a külvilág felől a periféria egységbe,
- kimenő adatok - ezek mennek a külvilág feléla periféria egységből,
- vezérlő jelek - ezekkel történik a periféria működésmódjának és tényleges működésének vezérlése,
- státusz jelek - ezeket a jeleket a periféria adja az aktuális állapotáról.

Természetesen adott periféria esetén ezen jelek bármelyike hiányozhat.

Időzítő/számláló egység (RTCC): (8. ábra) Nagyon sok feladatnál van szükség időzítésre illetve impulzusok számlálására. Ezt egy közös - egyszerre csak egyik módon használható egység valósítja meg: egy számláló áramkör: a léptető jel forrása vagy külső impulzus (számlálás) vagy a belső órajel (időzítés) lehet. Egy előosztó alkalmazásával bővíthetjük az egység "mérési tartományát". A számláló értéke mindig a RAM RTCC regiszterében található, ami írható vagy olvasható. Az ELŐOSZTÓ - feladata a rugalmasabb alkalmazhatóság megvalósítása, amelynek osztásviszonya szintén programozható.
A legegyszerűbb kialakítású PIC16C5X vezérlők esetén más perifériát a kontroller nem tartalmaz.

A PIC vezérlők fejlettebb változatai egyéb perifériákat is tartalmaznak: röviden felsorolva:

A/D átalakító: a bemeneti analóg jelet alakítja át digitális adattá,
PWM átalakító: egy regiszterben lévő bináris értéktől függ a kimenetén megjelenő négyszögjel 1 ill. 0 szintjének aránya,
Capture regiszter: egy külső jel megjelenésekor a belső számláló értéke ebbe a regiszterbe íródik,



8. ábra

tárolódik, és onnan kiolvaható,
Compare regiszter: ebbe a regiszterbe írt értékkel történik a belső számláló értékének összehasonlítása. Jelzést ad a két érték egyezése esetén.
Soros periféria: soros adatátvitelt megvalósító egység
Egyéb perifériák: ezekről majd az alkalmazásoknál esik szó.

Megszakítások:

Ha egy kontrollerrel valamilyen esemény létrejöttét kívánjuk érzékelni, ezt szokásos módon kétféleképpen tehetjük meg. Az első módszernél a külső események létrejőttét egy bemenet állapotának figyelésével érzékelhetjük.
Például ilyen megoldás alkalmazható, mikor egy billentyűzetről akarunk beolvasni.
Bármelyik billentyű megnyomásakor a billentyűzet kimenetén lévő "adat érvenyes" jel szintet vált. Ha ezt egy bemeneti portra kötjük, akkor az állapotának a programból való figyelése lehetővé teszi a billentyű megnyomásának az érzékelését, majd a kód beolvasását. Ezt a módszert általánosan elterjedt kifejezéssel "polling"-nak hívják. Alkalmazása azonban lelassítja a rendszer tényleges működési sebességét, hiszen a mikrokontroller idejének egy részét azzal tölti, hogy ciklikusan megvizsgálja a kijelölt bemeneti bit állapotát.

Sokkal szerencsésebb, ha az esemény maga jelzi a processzor számára állapotának megváltozását. Ez a megoldás a megszakítás vagy ismert angol kifejezéssel az interrupt (e.: interrapt). (Szokták IT-nek rövidíteni).
A megszakítás megszakítja az utasítások sorozatának (a program) végrehajtását, és a processzor egy úgynevezett megszakítási alprogramot hajt végre, ami az esemény kezelését elvégzi, majd ennek befejeztével a processzor visszatér a megszakított program végrehajtására.

Az előbbi példánál maradva, a billentyű megnyomását jelző "adat érvényes" jel megszakítást okoz, a megszakítási alprogram elvégzi a lenyomott billentyűhöz tartozó kód beolvasását, majd utána folytatódik a megszakított program.

A processzor oldaláról a megszakítási lehetőség kialakítása azt kivánja meg, hogy legyen olyan bemenete, ami állapota megváltozásakor képes a processzor működését felfüggeszteni, a megszakított program programszámlálójának az értékét elmenteni, és helyébe a megszakítási alprogram kezdőcímét betölteni, majd az alprogramot elindítani. A végrehajtás befejeztével (amit általában az utolsónak elhelyezett, speciális utasítás jelez) a program számlálóba a megszakított program program számlálójának elmentett értéke töltődik vissza, és a megszakított program folytatódik. Az a hely, ahová a programszámláló értékét elmentjük, az a verem. Ha a processzor több megszakítási vonallal rendelkezik, ezek mindegyikéhez egy-egy eseményt rendelhetünk hozzá. Ilyen felfogásban a processzort alaphelyzetbe állító RESET jel is egy megszakítás: megszakítja a futó programot, a programszámlálóba egy kezdeti értéket (általában nullát) tölt, és elindítja a program futását.

A megszakítások prioritásai Olyan rendszerekben, ahol több esemény okozhat megszakítást, megtörténhet, hogy egyszerre egy időben két megszakítás is fellép. Ilyen esetben a megszakítások kiszolgálásának fontossági sorrendje - a prioritása - dönti el a kiszolgálási sorrendet.
A program futása nem minden esetben szakítható meg káros következmények nélkül. Ezért a legtöbb rendszer biztosítja, hogy a megszakítások programból tilthatók, illetve engedélyezhetők legyenek. Ha egy rendszerben csak egy megszakítás van, akkor a tiltás és engedélyezés egy-egy utasítással lehetséges. Több megszakítás esetén ún. megszakítás maszkot használnak. Ez nyolc megszakítás esetén azt jelenti, hogy egy bájt nyolc bitjéhez az egyes megszakításokat rendeljük hozzá, és ha a bit értéke nulla, akkor a hozzátartozó megszakítás tiltott, különben engedélyezett. Igy a bájttal "maszkoljuk" a megszakításokat.

Megjegyzés: a tokok programozása során egy normál módon el nem elérhető memóriarekeszbe írjuk be néhány beállítható hardver tulajdonságot definiáló biteket. Ezek:

- az oszcillátor típusa,
- WDT engedélyezése,
- Power-up timer engedélyezése,
- kiolvashatóság tiltása.

"A családi fotó"

Azt hiszem az eddigiek alapján végül bemutathatjuk a PIC családot: a következő táblázat ezt foglalja össze, és reményeink szerint a táblázat döntő része már érthető. Jól látható, hogy egy adott feladathoz mindig kiválasztható a megfelelő típus. A következő, 9. ábrán a család néhány elemének tokbekötését mutatjuk be. A lábak három csoportra oszthatók:

Tápfeszültség: VSS, VDD,
Működtetés: OSC1,OSC2, MCLR, -
I/O kivezetések: RA, RB, RC portok.

16C5X család: az elsőnek forgalomba hozott legkisebb teljesítményű modell. Csak I/O vonalakat és számláló/időzítő áramkört tartalmaz perifériaként, megszakítási lehetősége nincs. A programutasításszó 12 bit hosszúságú. Az utasításszámláló 9 bites, amivel 512 szavas memória címezhető, de további különálló két címbit felhasználával maximum 2K-s memória használható.

16CXX család: fejlettebb típus. A perifériaválasztéka bő, megszakítási lehetőségei vannak. A programutasításszó 14 bit hosszúságú. Az utasításszámláló 12 bites, amivel 4k-s memória címezhető.

17CXX család: a legnagyobb teljesítményű modell. A programutasításszó 16 bit hosszúságú. Az utasításszámlálója is 16 bites, amivel 64k szavas memória címezhető. Néhány megjegyzés:

- A CMOS gyártástechnológia és a gondos tervezés következtében kimagaslóan jók az eszközcsaládok fogyasztási adatai.

- Az eszközöket a felhasználók programozzák, az olcsó, egyszer programozható tipusok kis és közepes szériák esetében is gazdaságos megoldást jelentenek.
- Sokféle tipus közül a felhasználók ki tudják kiválasztani a feladatuk megoldására legelőnyösebb, leggazdaságosabb megoldást.

2. A PIC mikrovezérlők utasításkészlete

A család jellemzőit összefoglaló táblázatból is látható a RISC felépítés fontos jellemzője, a viszonylag kevés utasítás (33, 35, illetve 55). A következő táblázat a 16C5X, 16CXX típusok utasításkészletét foglalja össze, csoportosítva. Az utasításoknál meghagytuk az eredeti angol szöveget, mivel az utasítások rövid neve (más néven: mnemonikja) ezekből származik. Az utasítások részletes ismertetése a [1,2,3] irodalmakban megtalálható.

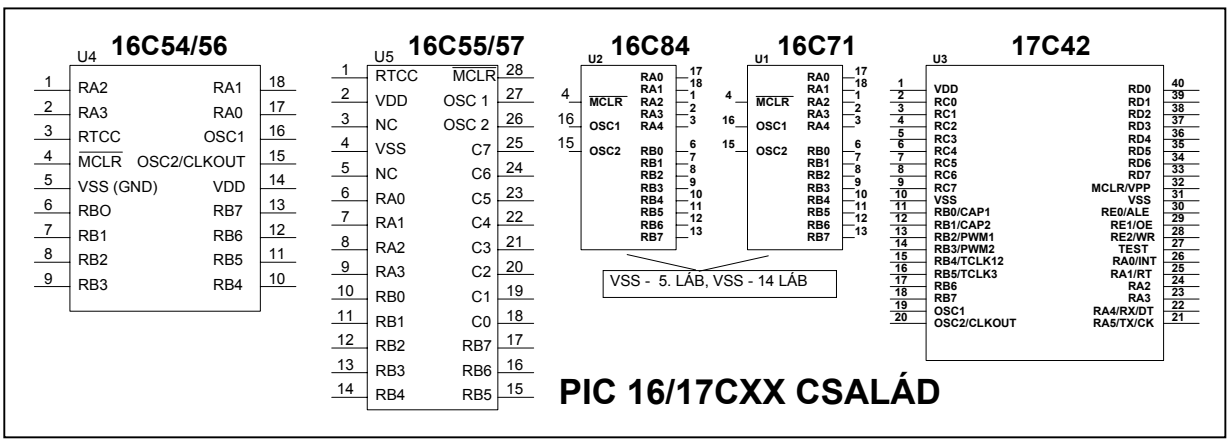
Összefoglalásul három megjegyzés:

1. Látható hogy az alapvető logikai és aritmetikai utasításokat a készlet tartalmazza.
2. Két operandust igénylő műveleteknél az egyik a W regiszter, a másik a RAM egy tetszőleges regisztere, és az eredmény helye is meghatározható az utasítás kódjában szereplő "d" bittel.
3. A bitműveletek vezérlési feladatok könnyű elvégzésére teszik a vezérlőket alkalmassá.

Aki ismeri más mikrovezérlő típusok programozását, megérti, hogy ez a szokatlan, az Intel típusoktól eltérő mnemonikájú és felépítésű utasításkészlet nem talált lelkes fogadtatásra a felhasználók körében. Ez a kezdetekben a PIC mikrovezérlők elterjedésének komoly gátja is volt. Ezt felismerve, egy másik amerikai cég a Parallax egy új, megszokott mnemonikájú és működésű utasításokat tartalmazó készletet definiált. Ez az utasításkészlet a népszerű MCS-51 utasításkészletéhez hasonlít. Ezt a jelölésmódot használó program menemonikjai először lefordítódnak az utasítást megvalósító egy vagy több PIC utasításra, amelyből a futtatható kód már előállítható.

lyen módon programjainkat két - az eredeti Microsim és az MCS51-szerű Parallax - jelölésmóddal is megírhatjuk, de a Parallax megoldása

TIPUSOK	PIC16C5X								PIC16CXX					PIC17CXX
Utasítás-hossz (bit):	12								14					16
Jellemzők	54	54A	R54	55	56	57	57A	58A	61	64	71	74	84	42
SEBESSÉG	20	20	20	20	20	20	20	20	20	20	20	20	10	20
EPROM	512	512		512	1K	2K		2K	1K	2K	1K	4K		2K
ROM			512				2K							
EEPROM													1K	
RAM	25	25	25	25	25	72	72	72	36	128	36	192	36	232
EEPROM													64	
IDŐZÍTŐK	1+WDT	1+WDT	1+WDT	1+WDT	1+WDT	1+WDT	1+WDT	1+WDT	1+WDT	3+WDT	1+WDT	3+WDT	1+WDT	4+WDT
CAPTURE/COM-PARE/PWM										1		2		2
SOROS PORT										SPI/I2C				SCI
PÁRHUZAMOS SLAVE PORT										IGEN		IGEN		
8 bites A/D											4	8		
MEGSZAKÍTÁ-SOK									3	8	4	12	4	11
I/O vonalak	12	12	12	20	12	20	20	12	13	33	13	33	13	33
Tápfesz (V)	2.5-6.2	2.5-6.2	2.0-6.2	2.5-6.2	2.5-6.2	2.5-6.2	2.5-6.2	2.5-6.2	3.0-6.0	2.5-6.0	3.0-6.0	3.0-6.0	2.0-6.0	4.5-5.5
Utasításszám	33	33	33	33	33	33	33	33	35	35	35	35	35	55



9. ábra

akár "kevert" módon megírt programot is megenged. A példák ismertetése során mindkét jelölésmódot használni fogjuk.

PIC16CXX Utasításkészlet - Összefoglaló

Byte-Oriented Operation - bájtos utasítások

No Operation	NOP	-
Move W to f	MOVWF	f
Clear W	CLRW	-
Clear f	CLRF	f
Subtract W from f	SUBWF	f,d
Decrement f	DECF	f,d
Inclusive OR W and f	IORWF	f,d
AND W and f	ANDWF	f,d
Exclusive OR W and f	XORWF	f,d
Add W and f	ADDWF	f,d
Move f	MOVF	f,d
Complement f	COMF	f,d
Increment f	INCF	f,d
Decrement f, skip if zero	DECFSZ	f,d
Rotate right f	RRF	f,d
Rotate left f	RLF	f,d
Swap halves f	SWAPF	f,d
Increment f, skip if zero	INCFSZ	f,d

Bit-Oriented Operations - bit utasítások

Bit clear f	BCF	f,b
Bit set f	BSF	f,b
Bit test f, skip if clear	BTFSC	f,b
Bit test f, skip if set	BTFSS	f,b

Literal and Control Operations
Konstanskezelő és vezérlő utasítások

Go into standby mode	SLEEP	-
Clear Watchdog Timer	CLRWDT	-
Return, place Literal in W	RETLW	k
Return from interrupt	RETFIE	-
Return	RETURN	-
Call Subroutine	CALL	k
Go to address (k is 9 bit)	GOTO	k
Move Literal to W	MOVLW	k
Inclusive OR Literal and W	IORLW	k
Add Literal to W	ADDLW	k
Subtract Literal from W	SUBLW	k
AND Literal W	ANDLW	k
Exclusive OR Literal W	XORLW	k

Jelölések:: f = a RAM (file) regiszter címe
d = a művelet eredménye hova kerül; 0 = W regiszter, 1 = RAM (file) regiszter
k = egy 8 bites fix érték (konstans) vagy egy utasításra mutató cím (ez hoszabb mint 8 bit!)
Megjegyzés: A szürkével jelölt utasítások a 16CXX típusok "új" utasításai a PIC16C5X típushoz képes

Parallax-féle utasításkészlet

ADD	**fr,#lit**	**ADD**	**fr1,fr2**	**ADD**	**fr,W**
ADD	**W,fr**	**ADDB***	**fr,bit**	**AND**	**fr,#lit**
AND	**fr1,fr2**	**AND**	**fr,W**	**AND**	**W,#lit**
AND	**W,fr**	**CALL**	**addr8**	**CJA**	**fr,#lit,addr9**
CJA	**fr1,fr2,addr9**	**CJAE**	**fr,#lit,addr9**	**CJAE**	**fr1,fr2,addr9**
CJB	**fr,#lit,addr9**	**CJB**	**fr1,fr2,addr9**	**CJBE**	**fr,#lit,addr9**
CJBE	**fr1,fr2,addr9**	**\CJE**	**fr,#lit,addr9**	**CJE**	**fr1,fr2,addr9**
CJNE	**fr,#lit,addr9**	**CJNE**	**fr1,fr2,addr9**	**CLC**	
CLR	**fr**	**CLR**	**W**	**CLR**	**WDT**
CLRB	**bit**	**CLZ**		**CSA**	**fr,#lit**
CSA	**fr1,fr2**	**CSAE**	**fr,#lit**	**CSAE**	**fr1,fr2**
CSB	**fr,#lit**	**CSB**	**fr1,fr2**	**CSBE**	**fr,#lit**
CSBE	**fr1,fr2**	**CSE**	**fr,#lit**	**CSE**	**fr1,fr2**
CSNE	**fr,#lit**	**CSNE**	**fr1,fr2**	**DEC**	**fr**
DECSZ	**fr**	**DJNZ**	**fr,addr9**	**IJNZ**	**fr,addr9**
INC	**fr**	**INCSZ**	**fr**	**JB**	**bit,addr9**
JC	**addr9**	**JMP**	**addr9**	**JMP**	**PC+W**
JMP	**W**	**JNB**	**bit,addr9**	**JNC**	**addr9**
JNZ	**addr9**	**JZ**	**addr9**	**LCALL***	**addrll**
LJMP*	**addrll**	**LSET***	**addrll**	**MOV**	**fr,#lit**
MOV	**fr1,fr2**	**MOV**	**fr,W**	**MOV**	**OPTION,#lit**
MOV	**OPTION,fr**	**MOV**	**OPTION,W**	**MOV**	**!port fr,#lit**
MOV	**!port fr,fr**	**MOV**	**!port fr,W**	**MOV**	**W,#lit**
MOV	**W,fr**	**MOV**	**W,/fr**	**MOV**	**W,fr-W**
MOV	**W,++fr**	**MOV**	**W,--fr**	**MOV**	**W,<<fr**
MOV	**W,>>fr**	**MOV**	**W,<>fr**	**MOVB**	**bit1,bit2**
MOVB	**bitl,/bit2**	**MOVSZ**	**W,++fr**	**MOVSZ**	**W,--fr**
NEG*	**fr**	**NOP**		**NOT**	**fr**
NOT	**W**	**OR**	**fr,#lit**	**OR**	**fr1,fr2**
OR	**fr,W**	**OR**	**W,#lit**	**OR**	**W,fr**
RET		**RETW**	**lit,1it,...**	**RL**	**fr**
RR	**fr**	**SB**	**bit**	**SC**	
SETB	**bit**	**SKIP**		**SLEEP**	
SNB	**bit**	**SNC**		**SNZ**	
STC		**STZ**		**SUB**	**fr,#lit**
SUB	**fr1,fr2**	**SUB**	**fr,W**	**SUBB***	**fr,bit**
SWAP	**fr**	**SZ**		**TEST**	**fr**
XOR	**fr,#lit**	**XOR**	**fr1,fr2**	**XOR**	**fr,W**
XOR	**W,#lit**	**XOR**	**W,fr**		

Jelölések: **fr - file regiszter, #lit - konstans, W - w regiszter, addr8 - 8 bites cím,**
(*Aki ismeri a 51-es utasításkészletet, annak nagyon szembetűnő a hasonlóság.)*

3. Alkalmazások fejlesztésének lépései

A cikksorozat eddigi részeiben összefoglaltuk azokat az alapvető hardver ismereteket, amelyekre alapozva hozzáfoghatunk kisebb, PIC mikrokontrolleres rendszerek fejlesztéséhez. A mikrokontrolleres rendszerek fejlesztésével kapcsolatos alapvető nehézség abban van, hogy egy ilyen feladat megoldása kettős tevékenységet igényel:

- egyrészt meg kell tervezni és megvalósítani a működő áramkört--a hardvert,
- másrészt meg kell írni a működtető és az adott feladatot akialakított hardverrel megvalósító programot--a szoftvert.

Mivel ez a két tevékenység szorosan kapcsolódik egymáshoz, ezért a fejlesztőnek mindkét témakörben alapvető ismeretekkel kell rendelkeznie.

A mikrokontrollert tartalmazó rendszerek létrehozásakor a következő lépéseket kell elvégeznünk:

- A feladat megfogalmazása.
- Áramköri tervezés.
- A működtető program megtervezése.
- Kódolás.
- Programvizsgálat (tesztelés), hibakeresés
- Dokumentálás.

Egy működő berendezés megvalósításánál mindegyik lépésnek nagy jelentősége van. A kódolás--ami a program megírását jelenti--csupán egy, és nem is mindig a legfontosabb lépés.

A feladat megfogalmazása

A mikrokrontrollerrel megoldandó feladatok számos tény és körülmény pontos rögzítését igénylik. Fontos annak az alapvető kérdésnek a tisztázása, hogy valóban szükséges-e mikroprocesszoros rendszer alkalmazása.

BEMENETEK DEFINIÁLÁSA Először a rendszer bemeneteit kell definiálni: fel kell sorolni és jellemezni, hogy a feladat megoldásához milyen bemenetek kellenek:

- Milyen a bemenő jelek formája, időbeli viselkedése?
- Mi jelzi a bemenő jel aktív állapotát?
- Milyen hosszú ideig aktív a bemenő jel?
- Milyen gyakran (gyorsan) változik a bemenő jel, képes-e a program a bemenő jel változását követni?
- A bemenőjel kapcsolatban van-e más bemenő vagy kimenő jelekkel?

KIMENETEK DEFINIÁLÁSA Ehhez hasonló kérdéseket kell megválaszolni a rendszer kimeneteinek definiálásakor is.

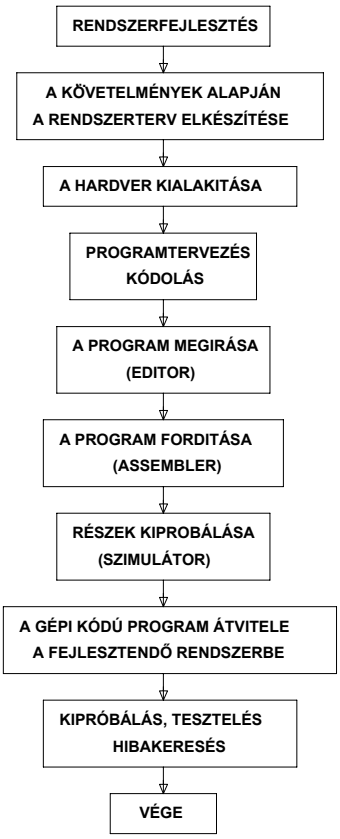
A következő lépés a **FELDOLGOZÓ RÉSZ DEFINIÁLÁSA**. Ez az a programrész, ami a bemenő jelekből és adatokból előállítja a kimenő jeleket és adatokat:

- Mi az alapvető algoritmus, ami a bemenetek alapján megadja a kimeneti jeleket,
- Milyen időkorlátok vannak? Milyen gyorsan kell a feldolgozást végrehajtani?
- Milyen memóriaterület korlátozások vannak? Van-e valami korlát a programot tartalmazó memóriarész, vagy az adatokat tartalmazó memória-részre vonatkozólag?
- Milyen szabványos (esetleg már meglévő) programrészleteket, szubrutinokat használhatunk fel?
- Milyen speciális esetek vannak, a program hogyan kezeli ezeket?
- Az eredményeknek milyen pontosnak kell lenniük?
- Hogyan tudja a feldolgozás során fellépő hibákat a program kezelni?

HIBAKEZELÉS Nagyon fontos a program működése során fellépő hibák felderítése és megfelelő módon történő kezelése.A hibakezeléssel kapcsolatban a következő szempontokat kell figyelembe venni:

- Milyen hibák lehetségesek?
- Melyik hibák a legvalószínűbbek? (Ha emberi beavatkozás is van, az emberi hibák a leggyakoribbak.)
- Milyen hibák maradhatnak rejtve egy ideig a rendszerben?
- A rendszer hogyan tudja a hibát a legkevesebb következménnyel elhárítani?

AZ EMBERI TÉNYEZŐ Sok mikroprocesszor alapú rendszer emberi közreműködéssel dolgozik. Az ilyen rendszerek tervezésekor ezt a



A RENDSZERFEJLESZTÉS BLOKKVÁZLATA

tényt fokozottan figyelembe kell venni, mivel alapvető minőségi jellemző az ember-gép kapcsolat, az ún. "humán interfész" megfelelő kialakítása. A legfontosabb szempontok:

- Milyen beviteli módszer, beavatkozás a legtermészetesebb a kezelő számára?
- Milyen megjelenítési, visszajelzési módszert a legcélszerűbb alkalmazni (vizuális, akusztikus, mechanikus jezések)?
- A kezelő hogyan kap jelzést a rendszer hibás működéséről?
- Melyek a kezelő által elkövetett

legvalószínűbb hibák?
- A kezelő beavatkozására adott rendszerválaszok egyértelműek?
- A kijelzési kép a kezelő számára könnyen olvasható és érthető?

A fentiekben leírt kérdések tisztázása után létrejön a konkrét tervezés alapjául szolgáló rendszerterv, ami alapján elkezdhető a rendszer hardverjének a megtervezése.

Áramköri tervezés

Mivel a legtöbb alkalmazásban a PIC mikrokontrollernél csupán az órajel előállításhoz kell külső alkatrész, ezért az áramköri tervezés a külvilághoz kapcsolódó perifériaáramkörökre korlátozódik. A megválaszolandó kérdések:

- Mennyi adat/program memória szükséges (típusválasztás!),
- hány be/kimeneti áramkört kell használni (típusválasztás!),
- milyen egyéb kiegészítő áramkörök szükségesek az adott feladat megoldásához. Lépései:

- A hardver megtervezése, a kapcsolási rajz elkészítése,
- a rajz alapján a nyomtatott áramköri lemez elkészítése,
- az alkatrészek beültetése a lemezre,
- a kész hardver bemérése, az egyes funkciók helyességének ellenőrzése.

Nagyon sok alkalmazásnál elegendő "csupalyuk" kártyán kézzel összehuzalozni az áramköröket.

A működtető program megtervezése.

Soroljuk fel a programtervezésnél használt legfontosabb alapelveket:
- Kis lépésekben történő haladás, és fejlesztés.
- A nagyobb részfeladatokat, kis, egymástól logikailag elkülönülő modulokból kell felépíteni, mivel ezek önállóan tesztelhetők és esetleges megváltoztatásuk nem igényli a teljes rendszer újratervezését.

- A feladatnak megfelelő programvezérlés lehetőleg egymás utáni, egyenként végrehajtható részekből álljon, és ne ide-oda ugrásokból, áttekinthetetlen program hurkokból, ciklusokból. Ez a hibakeresést is segíti.
- Minél több grafikus, vizuális leirás alkalmazása (a folyamatábra módszernek ez az előnye).
- Egyszerű és világos megfogalmazások, fogalmak használata.
- Olyan algoritmusok felhasz-nálása, amelyek ismertek és többszörösen kipróbáltak.
- A programtervezéskor figyelembe kell venni azt, hogy melyek azok a tényezők, paraméterek, amelyek megváltozhatnak.
- Programtervezés teljes befejezése után szabad csak a kódolást elkezdeni.

Kódolás.

A kódolás során a feladat és a készített programterv alapján megírjuk programot.

AZ ASSEMBLY PROGRAMOZÁS ALAPJAI

Minden számítógép, bármilyen bonyolult programot is hajtson végre, végső soron a bináris számokat utasításként értelmezve végzi el a műveleteket a szintén bináris formájú adatokon. Ez teljesen nyilvánvaló, mivel a gépben csak bináris 1 és 0 alakú információ feldolgozása lehetséges. Az ilyen 0, 1 számok formájában előálló programot nevezünk gépi kódú programnak.
A gépi kódon történő programozás azonban az ember számára rendkívül nehézkes, mivel meg kell tanulni a különböző utasításoknak megfelelő bináris, oktális vagy hexadecimális számrendszerbeli számokat (utasításkódok), és ki kell számítani a programban szereplő címek abszolút, vagy relatív értékeit.
A gépi kódu programozás ezen nehézségei az assembly (esszembli) szinten történő programozással küszöbölhetők ki.

Az assembly nyelv tulajdonképpen a legegyszerűbb programozási nyelv. Ez a nyelv lehetővé teszi, hogy azokat az utasításkódokat, amelyeket a gépi kódban számmal adtunk meg, könnyen megjegyezhető nevekkel írjuk le. Ezeket a kódneveket - melyek az utasítás által végrehajtott funkció megnevezésének rövidítései - mnemonikoknak nevezzük (mnemonik=emlékeztető). Az assembly nyelv lehetővé teszi azt is, hogy a program egyes belépési pontjaira, tároló rekeszeire, bájtokra, szavakra (1 szó=2 egymást követő bájt), valamint a regiszterekre ugyancsak nevekkel hivatkozzunk. Ezen neveket szimbólumoknak hívjuk. A szimbólumokat két csoportba osztjuk. Az egyik csoportba tartoznak az utasítás és direktíva mnemonikok (ld. később!), valamint a regiszterek nevei. Ezek az un. állandó szimbólumok, vagyis ezeket átnevezni, vagy új neveket adni nem lehet. A másik csoportba tartoznak a belépési pontok nevei (cimkék), és az egyes tárolórekeszek nevei. Ezek az un. felhasználói szimbólumok. Ezek nem rögzítettek, a felhasználó maga dönti el, hogy az általa használt szimbólumokat hogyan nevezi el.
Azzal, hogy a különböző belépési pontok és rekeszek nevekkel láthatók el, lehetőség nyílik arra, hogy a felhasználó (programozó) a program készítésekor megszabaduljon a gépi kódu programozásnál szükséges címszámítástól. Ezzel a gépi kódú programozás fent említett két hátrányát kiküszöböltük. Nem kell az utasításokhoz tartazó számértékeket memorizálni, és nem kell a programban címeket számolni.
Ezeket a teendőket a programozó válláról egy speciális program, egy un. "fordító program" veszi át. Ennek a programnak a feladata, hogy "megértse", és gépi kódu programra lefordítsa az általunk megírt szimbólikus programot. Az assembly nyelv esetében ezt a fordító programot assemblernek (esszemblernek) hívjuk. Az eredeti angol szó összeállítót (assembly=összeállítani) jelent, ami jól tükrözi az assembler feladatát: a gépi kódu program összeszerkesztését, össszeállítását a forrásprogram alapján. Forrásprogramnak nevezzük az általunk szöveges formában megírt, szimbólumokat tartalmazó programot. Az assembler tehát megérti és gépi kódok sorozatává -- un. tárgy

programmá -- alakítja át a forrásprogramot.
Az assembler az alapvető feladatainak ellátásán túl, különböző kényelmi szolgáltatásokat is nyújthat. Ezen szolgáltatások vezérlését végző utasításokat nevezzük direktíváknak. A direktíva tehát nem fordítható le, nincs gépi kódja, a direktíva csak a fordítás idejére aktív, csak a fordítóprogramnak "szól". Ilyen szolgáltatás például: a fordítás során keletkező lista formátumának meghatározása, előre elkészített programrészek automatikus beszerkesztése a forrásprogramba, tárolóhely foglalása, PIC tipusának, konfigurációjának megadása stb.

Az assembly nyelv szintaktikája

Mint minden nyelvnek, így az assembly nyelvnek is van egy nyelvtani formai szabályrendszere, un. szintaktikája. Ezen szintaktika elemei röviden összefoglalva a következők:
A forrásprogram utasítássorok véges sorozata. Az utasítás sor általában maximum 64 vagy 80 karaktert tartalmazó, CR (carriage-return) karakterrel befejezett sor (a PC-nél ilyenkor ütünk ENTER-t). Az utasítássor négy mezőből állhat, melyben az utasítás egyes részei kapnak helyet. A mezők az alábbi sorrendben követhetik egymást:

CM UM OM ;MM

(CM = CÍMKEMEZŐ
UM = UTASÍTÁSMEZŐ
OM = OPERANDUSMEZŐ
MM = MEGJEGYZÉSMEZŐ)

Egy utasítássorban nem kötelező mindegyik mezőt kitölteni, de egyes kombinációk (pl. operandusmező utasításmező nélkül) tiltottak. Az egyes mezők értelmezése a következő:
CIMKEMEZŐ: be írjuk azokat a neveket (címeket), melyekkel a program vezérlesi struktúrájában lényeges pontokat (pl. szubrutin belépési pont, ciklusmag kezdőpont, stb.) meg akarjuk jelölni. Az assembler a fordítás (assemblálás) során a cimkéhez a mögötte álló utasítás kezdőcímének értékét rendeli. Minden egyes, ezen cimkére való hivatkozás, az utasítás címmezejébe a cimkéhez rendelt értéket tölti be.

Az **UTASÍTÁSMEZŐ** -ben nemcsak az utasítások nevei (mnemonikok) szerepelhetnek, hanem itt találhatók a direktívák nevei is. A leggyakrabban használt direktívák:

ORG	fordítási cím megadása (a memóriába hova kerüljön a program)
DEFB	1 bájt definiálása
DEFS	tárterület definiálása
DEFW	2 bájtos szó definiálása
DEFM	szöveg definiálása
EQU	szimbólumnak értékadás
END	program vége

Az **OPERANDUSMEZŐ**-ben kell megadni az utasításmezőben szereplő utasítás vagy direktíva operandusait. Az operandusmezőben 0, 1 vagy 2 operandus állhat. Két operandus esetén, azok egymástól való elválasztására vesszőt kell használni.
Az operandusmezőben a következő elemek (melyek kifejezéseket is alkothatnak) szerepelhetnek:

- konstans (értéke lehet decimális, hexadecimális, oktális vagy bináris, melyet a számot követő H,O, vagy B- betűk jelölnek. Ha a hexadecimális szám A-F karakterrel kezdődik, akkor a bevezető nulla használata kötelező pl. 0DH).
- szövegkonstans (általában idézőjelek, vagy aposztrofok közé írt karaktersorozat).
- szimbólum (kötelezően betűvel kezdődő, max. 6 vagy 8 alfanumerikus karakterből álló név (lásd cimke)).
- regiszternév (A kontroller belső regisztereinek szimbólikus elnevezései.
- feltételkódnév (A processzor bitek szimbólikus elnevezései.
- műveleti jelek: +, -, *, /, logikai ÉS, logikai VAGY
- speciális jelek: $ a helyszámláló aktuális értékének jelölése, () az érték indirekt értelmezésének jelölése

MEGJEGYZÉS MEZŐ-be saját megjegyzésünket, magyarázó szövegünket helyezhetjük el. A megjegyzés mező kezdetét a pontosvessző ";" karakter jelzi és a sorvége "CR" karakterig tart.
Az utasítássorokban a mezők kötött vagy szabad formában követhetik egymást. Kötött mezőformátum esetén az egyes mezőknek előírt pozíciókon (tabulátor pozíciók) kell kezdődniük, míg szabad mezőformátumnál a mezők végét speciális elválasztó karakter (kettőspont, szóköz, pontosvessző) jelzi.
A PIC assemblerben használt utasítások mnemonikját, és magukat az utasítások működését majd a példáknál ismertetjük.

Feltételes assemblálás, makrók

Fejlettebb assemblerek további két olyan tulajdonsággal rendelkeznek, amelyek az assemblerben programozók munkáját segítik és teszik egyszerűbbé.
A feltételes assemblálás azt jelenti, hogy a forrásprogram bizonyos részei egy, az assemblálás során fennálló feltételtől függően a tárgyprogramban benne lesznek, vagy nem. Ez a lehetőség például akkor lehet hasznos, ha ugyanazt az assembler programot más hardver környezetben akarunk futtatni. A feltételes assemblálás szokásos formája:

```
    IF FELTÉTEL

(A FELTÉTEL IGAZ VOLTA ESETÉN BEFORDÍTANDÓ PROGRAMRÉSZ)

    ENDIF
```

Az assembler programozás során igen gyakran előfordul, hogy ugyanazt az utasítássorozatot többször használjuk a programunkban. Az un. makrók alkalmazásával elkerülhető ilyen utasítássorozatok ismételt leirása, mivel a makrók lehetővé teszik azt, hogy egy utasitássorozathoz egy nevet rendeljünk, és a továbbiakban erre a sorozatra a hozzárendelt névvel hivatkozzunk. A programunk elején definiáljuk a makrót, általában a következő módon:

```
MAC1:  MACRO
       ; MAC1 A MAKRÓ NEVE
       .....
```

IDE JÖN AZ UTASITÁS SOROZAT
.....
ENDM
; EZ A MAC1 MAKRÓDEFINICIÓ LEZÁRÓ UTASÍTÁSA

Ezek után a programban mindazokon a helyeken ahol a fenti utasitássorozatot akarjuk használni, csupán azt kell leírni, hogy: MAC1 .
A makrók a szubrutinoktól alapvetően különböznek, mert minden rájuk történő hivatkozás alkalmával a definiált utasitássorozat ténylegesen beépül a gépi kódu tárgyprogramba. Ezt a lehetőséget biztosító assemblereket makró-assemblereknek nevezzük.
PIC-ekhez a Parallax cég (PASM) és a Microchip (MPASM) is ad assemblereket, ***szabad szoftverként!***

A programfejlesztést támogató eszközök

Két nagyon fontos kérdésről az eddigiekben nem beszéltünk. Nevezetesen arról, hogy egyrészt milyen módon hozzuk létre a forrásnyelvi programot, másrészt milyen lehetőségeink vannak a megírt program helyes működésének ellenőrzésére. A forrásnyelvi program megirását egy speciális programmal, az un. szövegszerkesztő (text editor) programmal végezzük el, míg a program helyes működését a PIC-eknél szimulátorral ellenőrizhetjük.

A szövegszerkesztő program feladata, hogy segítségével a felhasználó szövegfájlokat hozhasson létre, illetve a már meglévő szövegfájlokat módosíthassa. Szövegfájloknak nevezzük azokat az adathalmazokat, amelyek karaktereket tartalmaznak, ezek például megjeleníthetők a számítógép képernyőjén, vagy nyomtatón kinomtathatók. Az editoroknak igen nagy jelentősége van a számítástechnikában: bármely programnyelven megírt programot (pl. BASIC, PASCAL, FORTH, stb.) "szöveg"-ként, karakteres formában visszük be a számítógépbe, (ez a forrásnyelvi program) azaz az ember-gép kapcsolat megvalósításának a legfontosabb eszköze.

Programvizsgálat (tesztelés), hibakeresés

A programokban való hibák felderítése és kiküszöbölése a programozóknak egyik leggyakrabban végzett tevékenysége, mert a programokban hibák vannak (az igen rövid programoktól eltekintve). Mivel a program a valóságban nagy sebességgel fut, ezért meg kell találni azokat a lehetőségeket hogy a futást ellenőrizni tudjuk. Ennek a következő két gyakran használt megoldása a következő:

Lépésenkénti programvégrehajtás (single-step) Ilyenkor a mikroprocesszor az utasításokat egyenként, egymás után hajtja végre, egy jelre mindig csak egyet. Ilyenkor minden lépés után ellenőrizni tudjuk a programunk működését.

Töréspont A töréspont, vagyis a programfutás adott pontban történő felfüggesztése, igen sok feltételtől függhet. Milyen feltételek adhatók meg?

- adott cím (a címvonalak adott állapota),
- bizonyos adat (az adatvonalak adott állapota),
- adott vezérlővonalkombinációk,
- az előzőek kombinációi.

Ezeket a megoldásokat általában a program működését utánzó szoftver szimulátorban valósítják meg.
A szoftver szimulátor egy számítógépen futó, általában valamilyen magas szintű nyelven megírt program, ami az eredeti mikrokontroller minden utasítását szimulálja, figyeli a regiszterek, jelzőbitek, memóriahelyek tartalmát, azok változását. Ez természetesen papírral és ceruzával is megtehető, de nagyon fáradtságosan. Egy ilyen szimulátor legfontosabb tulajdonságai:

- Töréspont elhelyezése a legkülönfélébb feltételek teljesülésétől függően.
- Regiszterek és memória listázása.
- Nyomkövető képesség, ami egy regiszter vagy memóriahely tartalmát írja ki, ha azt a program használja.
- A regiszterekbe tetszőleges érték tölthető és onnan folytatható a program futása.

Példaként a 11. ábrán a PIC-ekhez a Parallax cég által kifejlesztett PSIM szimulátor képernyőképét mutatjuk be: A Microchip hasonló termékének neve: MPSIM.

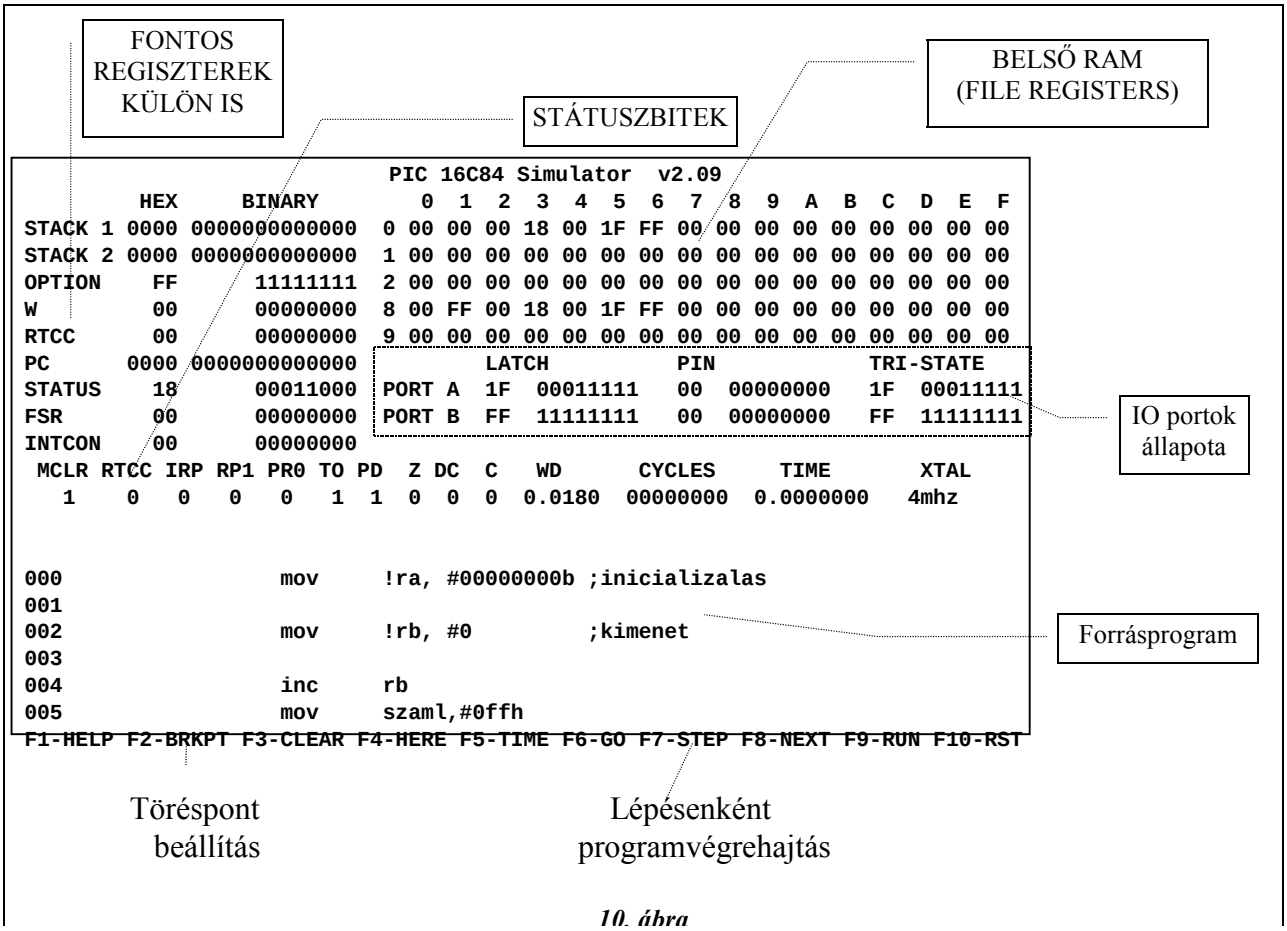
Dokumentálás

A programozók többsége igazán nem tekinti fontosnak saját munkájának dokumentálását. Az igaz hogy ez védelmet ís nyújthat az illetéktelen kíváncsiskodók ellen, de sajnos munkastílussá is válhat, és idővel ez a programozó saját és kollégáinak a munkáját is megnehezíti.
A programfejlesztés egyik legfontosabb része a pontos, a programot jól leíró dokumentáció. A dokumentációnak a programfejlesztés egész folyamatát végig kell kisérnie, a programhasználatot és a program továbbfejlesztését is támogatnia kell.

Egy rosszul dokumentált programot nagyon nehéz karbantartani, használni, vagy továbbfejleszteni. Azért, hogy egy program jól dokumentált legyen, a következő tanácsokat célszerű megfogadni:

- A program világos, egyszerű felépítésű legyen, minél kevesebb vezérlésátadást (ugrást) tartalmazzon.
- Olyan azonosítókat, cimkéket és neveket használjunk, ami jól tükrözi funkciójukat.
- Kerüljük a betűszavakat, mert nem mindenki számára nyilvánvaló a rövidítés (pl. SBR=soros bemeneti rutin).

Az előbbiekben röviden összefoglaltuk azokat a legfontosabb ismereteket ami alapján az alkalmazások mindenki számára érthetővé vállnak. Az alkalmazásokat bemutatása előtt összefoglaljuk, hogy milyen eszközökre van szükség a PIC eszközökkel történő fejlesztéshez:
A legfontosabb, és elengedhetetlen: a



10. ábra

programozó eszköz, amely egy IBM PC kompatibilis gépre kapcsolódik (286-os is jó!). A fejlesztést csak újra programozható (EPROM-os, vagy EEPROM-os) PIC tokkal érdemes végezni. Kisebb tokoknál a legelőnyösebb a PIC16C84 típus használata: ez mintegy ezerszer újraprogramozható, és a törlése igen gyorsan elektromosan, külső eszköz nélkül (pl. kvarclámpa) lehetséges. Természetesen a fejlesztés végeredményét olcsóbb tokokba lehet írni. Például, ha nem használjuk a tok belső EEPROM adatregisztereit, akkor kitünő választás az egyszer programozható 16C61-es típus.

A programfejlesztéshez szükség van egy PC-n futó szövegszerkesztőre, assemblerrre (Parallax: PASM, Microchip: MPASM, és egy szimulátor programra Parallax: PSIM, Microchip: MPLAB). ***Ezen szoftverek mindegyike külön térítés nélkül megszerezhető és használható!***