Mikrovezérlők szoftverfejlesztése, az assembly alapjai

A mikrovezérlők programozásához először tekintsük át általánosan a szoftverfejlesztés alapjait.

Mikrovezérlők szoftverfejlesztése

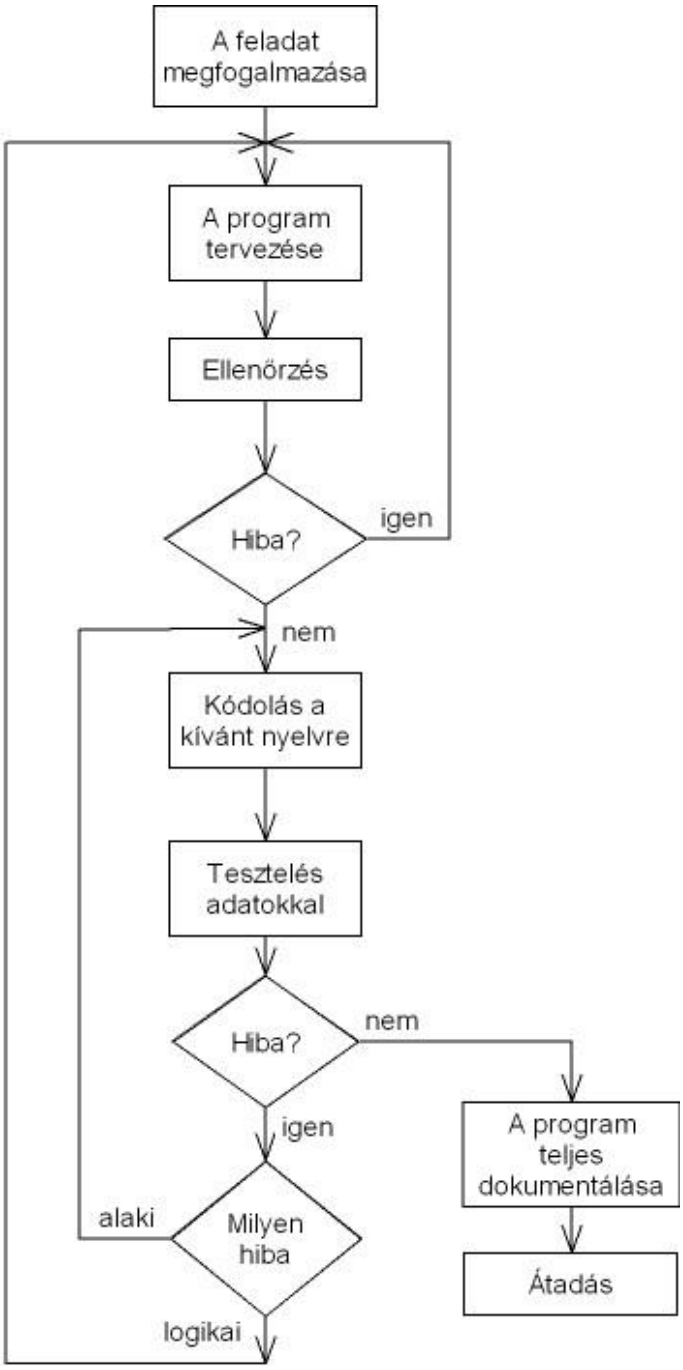
A mikrovezérlős fejlesztésekben a programozásnak a legtöbb esetben nagyobb a jelentősége van, mint a hardver áramköri kialakításának. Ezért a programozás alapjainak elsajátítása nagy fontossággal bír (*Dr. Kónya*, 2003)!

A programozás nem csak a mikrovezérlőknél kerül előtérbe, hanem ez általános fogalom az informatikában, amelynek általánosan elterjedt módszerei, szabályai vannak. A programnak nagy jelentősége van, hiszen a mikrovezérlőben futó programmal valósítjuk meg a megoldandó feladatot!

A számítógépes problémamegoldás során a következőkből indulunk ki:

- Ismert bemenő adatok
- az ismeretlen kimeneti adatok
- összefüggések az ismert és az ismeretlen között

A problémamegoldás során megfelelően egymás után kapcsolt műveletek sorozatával eljutunk az ismerttől az ismeretlenig. A programfejlesztés főbb lépéseit a 1. ábra foglalja össze:


A feladat megfogalmazása
A program tervezése
Ellenőrzés
Hiba?
igen
nem
Kódolás a kívánt nyelvre
Tesztelés adatokkal
Hiba?
nem
igen
A program teljes dokumentálása
Átadás
alaki
Milyen hiba
logikai

1. ábra

- A feladat matematikai és logikai alakban való megfogalmazása.
- Tervezés, melynek során megfelelő **algoritmust** választunk. Algoritmus: aritmetikai és logikai műveletek sorozata, amely lehetővé teszi a feladat megoldását. Az algoritmikus lépések mindig műveleteket valósítanak meg. A programozás során alapvetően négy algoritmuslépést használunk:
    - Számítás: numerikus, logikai, karakterműveletek
    - Döntés: összehasonlítás alapján alternatív lépések kiválasztása
    - Bemenet: adatok bevitele a műveletvégzéshez
    - Kimenet: a kiszámított eredmények kiírása

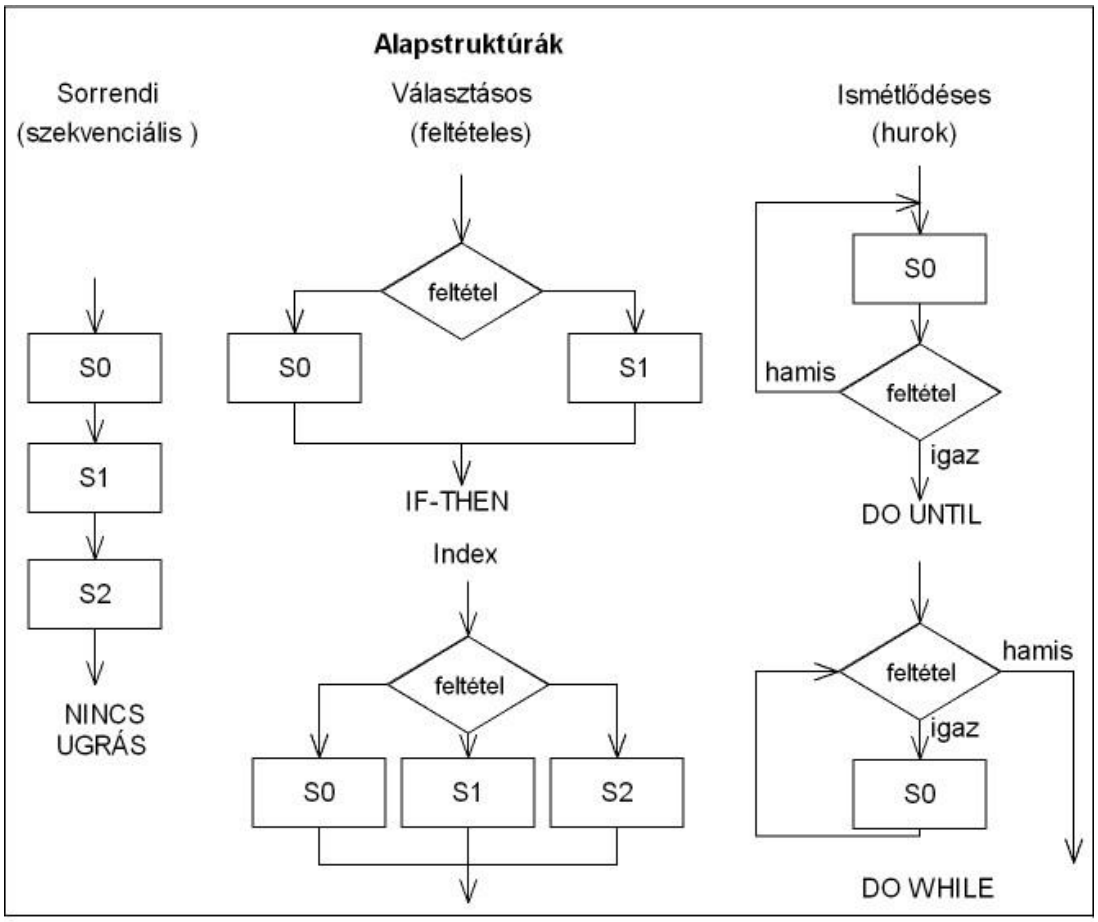
A programok tervezésénél a következő módszerek használatosak (*Dr. Kónya*, 2003):

- Folyamatábra készítés
- Strukturált programozás
- Felülről lefelé tervezés
- Objektumorientált programozás

2. ábra

Folyamatábra módszer: A program tervezése jól látható, grafikus módon történik. Szabványos szimbólumokkal (2. ábra) dolgozik. A folyamatábrák megérthetőek programozási ismeretek nélkül is, de bonyolult megtervezni, megrajzolni, változtatni. Nincs egyszerű módszer a folyamatábrán történő hibakeresésre, tesztelésre. Annak eldöntése, hogy milyen részletes, vagy tömör legyen a folyamatábra, nem egyszerű. Abban az estben, ha túl tömör nem adja vissza a program minden fontos részletét, ha túl részletes, akkor pedig nehezen áttekinthető. A folyamatábrában könnyű a nyilakat ide-oda húzni, a kódolás során azonban ezt már nehéz megvalósítani a sok ugrás miatt.

Moduláris programozás: A teljes programot részekre, más néven modulokra osztjuk. Az alapvető probléma az, hogy hogyan osszuk fel modulokra a programot, és hogyan egyesítsük ezeket komplett működő programmá? A modulokat könnyű megírni, tesztelni, a bennük lévő hibát megkeresni. Egy-egy modul valószínűleg sok



3. ábra

má

s feladatban is felhasználható. Több programozó is könnyen együtt tud működni. A hibák általában csak egy modulhoz kapcsolódnak, így könnyebb kijavítani. A modulok nagyon pontos, részletes dokumentációt igényelnek, mivel a program más részeire is kihatással vannak. A végső, a programba beépülő modulok tesztelése nehézkes lehet, mert egyes modulok olyan bemenő adatot kaphatnak, amelyet másik modul állít elő.

Strukturált programozás: ennél a módszernél – nevéből következően – a programokat a programozó alapstruktúrákból építi fel. Nevezzük el az egy utasítást, vagy utasítássorozatot S-nek. A jelölés felhasználásával a következő alapstruktúrákat kapjuk (3. ábra):

Szekvenciális (sorrendi) struktúra: S0, S1, S2 … struktúrák a programban sorban, egymás után kerülnek végrehajtásra.
Feltételes struktúra: abban az esetben, ha C igaz, akkor S0, különben S1. A C itt egy feltételt jelent (IF-THEN-ELSE).
Hurok struktúra: amíg C igaz, csináld S-t. A C itt is egy feltételt jelent (DO-WHILE, DO-UNTIL).
Index struktúra: I esetén S0, S1, S2 …, Sn, ahol I az index, és 0, 1, 2, …, n értékű lehet. Abban az esetben, ha I=0, akkor S0; ha I=1, akkor S1; … ha I=n, akkor Sn hajtódik végre (DO-CASE). Matematikailag bizonyítható, hogy ezen négy struktúra felhasználásával bármilyen program elkészíthető. Előnye, hogy a műveletek sorrendjét könnyű követni, ezért könnyű hibát keresni és tesztelni. Az így felépített program már önmagát dokumentálja, könnyen átlátható. A gyakorlat azt mutatja, hogy ezt a módszert használva nő a programozói teljesítmény. Hátránya, hogy a strukturált program általában hosszabb futásidejű, és több memóriát foglal el, mint a nem strukturált változat. A korlátozott számú struktúrával néhány esetben nagyon nehéz bonyolult feladatot megvalósítani. Ugyan bármely program megírható ezeknek, a struktúráknak a felhasználásával, de nem biztos, hogy az így kapott program hatékony, és kis tárigényű lesz. Az egymásba ágyazott „ha…akkor…különben” struktúrákat nehéz követni. A programozó gondolkodásmódját leszűkíti.
Felülről lefelé programozás: a módszernek az a lényege, hogy először a teljes átfogó programot írjuk meg, a benne szereplő egyes feladatokat, eljárásokat csak kijelöljük – nevet adunk neki – és csupán definiáljuk a név mögött álló feladatot. A következő lépésben ezeket a feladatokat ismét részfeladatokra bontjuk – ha szükséges – és ismét definiáljuk azokat, és így tovább. Amikor a lebontásban eljutottunk az elemi szinthez, akkor kezdjük a program megírását. Innen kezdődik a felfelé történő programírás, és a minden szinten elvégezhető tesztelés. Hátrány egyrészt az, hogy a meglévő programokat, rutinokat nem feltétlenül tudjuk könnyen felhasználni, nem eredményez általánosan felhasználható modulokat. Másrészt nem biztos, hogy hatékony programot nyerünk a módszer alkalmazásával.

- Ellenőrzés (papíron)
- Programozás:
    - Gépi nyelven
    - Gépre orientált nyelven

- Feladatra orientált nyelven

Gépi nyelv: a programozó az utasításokat közvetlenül gépi kódban adja meg.
Az utasítások kódját általában hexadecimális formátumban viszik be a gépbe. Az utasítások, valamint az adatok címeit előre meg kell határozni. A programmódosítások az összes cím megváltozását idézik elő. Nehézkes és időigényes.

Gépre orientált nyelv: ez egy szimbolikus nyelv, melynek utasításai a hozzá tartozó berendezés gépi nyelvű utasításaival megegyezők, vagy ahhoz hasonlóak. Az utasítások neveit könnyen megjegyezhető úgynevezett *mnemonik*okkal rövidítik. Az utasítások és adatok címei szimbolikus formában is megadhatók (címkék). Ilyen nyelv például az **assembly**. Ez a programnyelv különösen alkalmas a mikrovezérlők programozására. Ezzel a nyelvvel tudjuk a legjobban kiaknázni a mikrovezérlőben rejlő adottságokat. Az assembly-vel tudjuk a legrövidebb és leggyorsabb programkódot megírni. Ezen előnyök miatt mindenképpen az assembly nyelv használatát javaslom a mikrovezérlők programozásához!

Feladatra orientált nyelv: lehetővé teszik a feladatnak a problémákhoz igazodó megoldását, nevezik magasszintű programnyelvnek is. Az idők folyamán nagyon sokféle ilyen nyelv keletkezett (ALGOL, COBOL, BASIC, PASCAL, C, C++, stb.). A PIC mikrovezérlőkhöz is létezik C fordító, így a programokat akár C nyelven is megírhatjuk. A magasszintű programnyelvek használata a fejlesztési időt lerövidíti. Tisztában kell lennünk azonban azzal, hogy az így generált tárgyprogram sokkal **redundánsabb** lesz, sok helyet fog elfoglalni a memóriában, a futási idő megnövekszik, gyors alkalmazások megvalósítására nincs lehetőség!

- A program tesztelése. Célja a hibák felderítése és kiküszöbölése. A hibák két csoportba sorolhatók:
  - Alaki (szintaktikai) hiba: olyan utasítás írtunk, amelyet a fordítóprogram nem tud értelmezni
  - Logikai hiba: elgondolásunk hibás volt, ez nem vezet el a megoldáshoz, újra kell gondolnunk a folyamatot
- A programokat a programkönytárban összefoglalva rendszerezzük és elkészítjük a teljes dokumentációját.

Látható, hogy a program írása nem egyszerű, hanem nagyon összetett feladat. A programfejlesztéshez többféle struktúrából válogathatunk. Fontos megjegyezni, hogy

ezek nem felváltják, hanem kiegészítik egymást. A folyamatábra használata mindenképp ajánlott a tanulók számára, illetve a strukturált programozásnál célszerű az alapstruktúrákat megismertetni velük. A moduláris programozást projektmunka keretében lehet gyakoroltatni.

A programfejlesztésnél a következő alapelveket célszerű betartani:

- Kis lépésekben történő haladás és fejlesztés. A nagyobb részfeladatokat egymástól logikailag elkülönülő kis 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, ne ide-oda ugrásokból, áttekinthetetlen programhurkokból, ciklusokból. Ez a hibakeresést is segíti.
- Minél több grafikus, vizuális leírás alkalmazása (folyamatábra módszer).
- Egyszerű és világos megfogalmazások, fogalmak használata.
- Olyan algoritmusok felhasználása, amelyek ismertek, és kipróbáltak.
- A programtervezéskor figyelembe kell venni azt, hogy melyek azok a tényezők, paraméterek, amelyek megváltozhatnak.
- A kódolást csak a programtervezés teljes befejezése után szabad elkezdeni!

Az utolsó pontra hívjuk fel külön a tanulók figyelmét! Tapasztalatom szerint ugyanis a tanulók hajlamosak mindjárt a program írásával kezdeni a feladatot, anélkül, hogy megterveznék azt. Ez az első hiba jelentkezésekor nagy problémát fog okozni, ugyanis azt sem tudjuk ilyenkor, hogy hol keressük a hibát! A tanár munkáját is nagymértékben megnehezíti, mert mivel nem tudja a gondolatmenetet, segíteni sem tud hiba esetén.

Az assembly programozás alapjai

A mikrovezérlőket működtető programok megírásánál a programozó számos lehetőség közül választhat, mint ahogy az előző fejezetben olvasható.

Minden digitális számítógép, bármilyen bonyolult feladatot is hajtson végre, végül is a bináris számokat utasításként értelmezve végzi el a műveleteket a szintén bináris formátumú adatokon. Ez teljesen nyilvánvaló, mivel a gépben csak bináris 0 és 1 alakú információ feldolgozása lehetséges. Az ilyen formában előálló programot nevezzük gépi kódú programnak.

A számítógép programozásában rejlő összes lehetőség legjobb kihasználását a gépi kód teszi lehetővé. Ezen a szinten írhatók a gép működése szempontjából a leghatékonyabb programok, itt használhatók ki legjobban az egyes utasítások hatásai és mellékhatásai, itt alkalmazhatja a programozó a legszellemesebb megoldásokat és trükköket (*Dr. Kónya*, 2003).

A gépi kódban történő programozás azonban az ember számára nagyon nehézkes, kézség szintű elsajátításához, a programozói rutin megszerzéséhez igen hosszú idő szükséges. Ezt a fáradtságos munkát még az is nehezíti, hogy meg kell tanulni az utasításoknak megfelelő bináris, hexadecimális, vagy oktális kódot, és ki kell számítani a programban szereplő címek abszolút, vagy relatív értékeit.

A gépi kódú programozás ezen hátrányait – az előnyök megtartása mellett – küszöböli ki az assembly nyelven történő programozás!

Az assembly egy egyszerű programozási nyelv, amely lehetővé teszi, hogy a gépi kód helyett az utasításokat könnyen megjegyezhető nevekkel írjuk le. Ezeket, a kódneveket – melyek általában az utasítás funkciójának a rövidítései – *mnemonikoknak* nevezzük. Az utasításhoz háromféle operandus kapcsolódhat:

- Konstans (literál), vagyis egy állandó
- Adatregiszter címe, aminek tartalmával a műveletet végezzük
- Programmemória cím, amely a következő végrehajtandó utasítást tartalmazza

További egyszerűsítést jelent a gépi kódhoz képest, hogy a program egyes belépési pontjaira nevekkel hivatkozhatunk, ún. *címkéket* rendelhetünk hozzá. Ezen kívül szintén nevekkel hivatkozhatunk különböző regiszterekre, bájtokra, bitekre, azaz *szimbólumokat* rendelhetünk hozzájuk. Az elnevezésekre ügyeljünk,

mert ezek megállapodás szerint csak betűvel kezdődhetnek (ugyanis csak a számok kezdődhetnek számmal). A címkék általában kettőspontra végződnek (az MPLAB elfogadja kettőspont nélkül is). A utasítások elnevezései természetesen kötöttek, ezt a gyártó definiálja. A PIC mikrovezérlőknél az adatmemória két részre oszlik:

- **SFR** (Special Function Registers) – speciális funkciójú regiszterek
- **GPR** (General Purpose Registers) – általános célú regiszterek

A speciális funkciójú regiszterek elnevezéseit a Microchip definiálta, ezeket nem nekünk kell kitalálni (pl. a portok elnevezései: PORTA, PORTB, stb.). Az elnevezések a <típus>.inc fájlban, az MPLAB program könyvtárában találhatók meg az egyes típusokhoz. Jelen esetben mi a *„pic16f84.inc”* fájlt fogjuk használni. Az általános célú regisztereknek tetszés szerinti betűvel kezdő, akár bitenkénti elnevezést is adhatunk. A címkék, szimbólumok elnevezése tetszőleges, azonban hívjuk fel rá a tanulók figyelmét, hogy lehetőség szerint rövid, lényegretörő (beszédes) elnevezéseket válasszunk! A találó elnevezések ugyanis megkönnyítik a forrásprogram „olvasását”, ami a későbbiekben lényegesen egyszerűbbé teszi a program elemzését és a hibakeresést! A fent említett könnyítésekkel a programozó megszabadul a fáradtságos címszámítástól, ami a gépi kódú programozáshoz szükséges volt, ugyanis a címeket assembly-nél a fordítóprogram számolja ki.

A fordítóprogramnak a feladata, hogy értelmezze, és gépi kódra fordítsa az általunk szimbolikus formában megírt programot. Az általunk szövegszerkesztővel (ez tetszés szerinti szövegszerkesztő) megírt programot *forrásprogramnak* nevezzük, assembly-ben a kiterjesztése asm. Ezt a forrásprogramot alakítja át a fordítóprogram *tárgyprogrammá.* Az assembly programnyelvben a fordítóprogramot *assembler*-nek nevezik! Elnevezése az angol assemble (összeállítás) szóból származik (*Dr. Kónya*, 2003).

Mint minden nyelvnek, az assembly nyelvnek is van nyelvtani formai szabályrendszere, ún. szintaktikája. Az assembly program sorokból áll, melynek felépítése a következő:

címke	művelet	operandus	megjegyzés
(label)	(operation)	(operand)	(comment)
START:	MOVLW	.15	; a W-be 15-öt töltünk

Egy programsor tartalmazhat *utasítást,* amelyet a gép végrehajt, vagy *direktívát*,

amely a fordítóprogramnak szól. Az egyes mezőket minimum egy szóközzel kell elválasztani, de sokkal áttekinthetőbbé válik a forrásprogramunk, ha tabulátorokkal igazítjuk a mezőket. Az egyes mezők értelmezése a következő:

- Címkemező: ide írjuk azokat a neveket (címeket), melyekkel a program lényeges pontjait (pl. szubrutin belépési pont, ugrási cím, stb.) meg akarjuk jelölni. A fordítóprogram a fordítás során a címkéhez a memóriabeli elhelyezkedési címét rendeli. Abban az esetben tehát, ha a program során bárhol erre a címkére hivatkozunk, akkor a memóriabeli címének értéke helyettesítődik be. A címke csak betűvel kezdődhet, és általában kettőspontra végződik (a Microchip assemblerénél a kettőspont elhagyható).
- Művelet (utasítás) mező: ide írjuk a megfelelő utasításokat, valamint a direktívákat.
- Operandusmező: itt kell megadni az előző mezőben szereplő utasításhoz, vagy direktívához tartozó operandusokat. Az operandusmezőben egy, vagy két operandus állhat, illetve lehet olyan eset is, hogy nincs operandus. Két operandus esetén azok egymástól való elválasztására vesszőt kell használni. Az operandusmezőben a következő elemek – melyek kifejezéseket is alkothatnak – szerepelhetnek:

    - Konstans (literál, állandó), amely lehet decimális, hexadecimális, oktális, vagy bináris
    - Szövegkonstans: idézőjelek, vagy aposztrófok közé írt karaktersorozat, amely ASCII kódját adja vissza a kifejezés
    - Szimbólum: betűvel kezdődő elnevezés
    - Regiszternév: a processzor belső regisztereinek, bitjeinek szimbolikus elnevezései (a gyártó definiálja)
    - Feltétel (státusz) bit (Flag Bit): ezek valamilyen műveletvégzés eredményeképpen állnak be, az adott művelet után beállt állapotról adnak jelzést (pl. kinullázódott egy regiszter, átvitel történt összeadáskor, stb.), ezeket szintén a gyártó adja meg
    - Műveleti jelek:

        + összeadás

        - kivonás

* szorzás

/ osztás

& logikai ÉS

^ logikai VAGY

- Speciális jelek

$ az utasításszámláló (programszámláló) aktuális értéke

- Megjegyzés mező: ide saját megjegyzésünket, magyarázó szövegünket helyezhetjük el. A megjegyzést mindig pontosvesszővel kel kezdeni. Minden pontosvessző után írt szöveg megjegyzésnek tekinthető, ilyen módon akár egy teljes sor is lehet megjegyzés.

A programok megjegyzésekkel való ellátása, „kommentezése” nagyon fontos dolog. Ugyanis az assembly-ben írt programokban „első látásra” nagyon nehéz kitalálni az egyes programrészek funkcióit. Sokszor maga a programozó is elfelejti néhány nap múlva, hogy mit is akart itt csinálni! A jó kommentezés azonban nagyban megkönnyíti a hibakeresést, illetve a programjaink továbbfejlesztését. Tapasztalatom szerint a tanulók különösen hajlamosak a megjegyzések elhagyására, ezért tudatosítsuk bennük ennek fontosságát!

A assemblyben megírt forrásprogramunkból az assembler állítja elő a tárgyprogramot. Ezt a műveletet fordításnak nevezzük. Az assembler a fordítást több menetben hajtja végre. Az első menetben az assembler végigolvassa a forrásprogramot és felépíti a szimbólumtáblázatot. Az assembler két táblázatból dolgozik, az egyikben az állandó szimbólumok találhatók, a másikban pedig a felhasználói szimbólumok. Itt rendelődnek hozzá a szimbólumokhoz a megfelelő értékek a szimbólumok memóriabeli elhelyezkedése alapján. Az olvasás befejezésekor minden szimbólumnak értéke kell, hogy legyen. Azokat a szimbólumokat, amelyek nem kaptak értéket nem definiált szimbólumnak (undefined symbol) nevezzük. A második menetben történik a tulajdonképpeni tárgyprogram létrehozása. Ilyenkor az assembler újra végigolvassa a forrásprogramot és átalakítja a programsorokban szereplő utasításokat gépi kódra. A programsorok értelmezése során az assembler felismeri a különböző *szintaktikai* (alaki), illetve *szemantikai* (értelmezésbeli) hibákat. Ezeket hibaüzenet formájában meg is jeleníti. A hibalista és a tárgyprogram mellett a második, vagy a harmadik menetben elkészül a fordítási lista is. Ez tulajdonképpen a forrásprogram másolata, amely azonban sorról sorra tartalmazza a helyszámláló aktuális értékét, valamint az adott sor fordítása révén

nyert gépi kódot, illetve hibás sor esetén a hibaüzenetet. A fordítási listában megtaláljuk még a szimbólumok névsorba rendezett táblázatát is a hozzárendelt értékekkel együtt. A lista végén pedig egy statisztikát olvashatunk a programról. A fordítási lista tehát lényegében a fordítás dokumentuma (*Dr. Kónya*, 2003).

A mikrovezérlők alkalmazásakor általában nem áll rendelkezésre az adott mikrovezérlőn futó assembler program, amivel a fordítást el tudnánk végezni. Ilyenkor az a megoldás, hogy egy másik gépen (pl. PC) végezzük el a fordítást. Ehhez megfelelő fejlesztőkörnyezet szükséges, amely rendelkezik egy ún. kereszt-fordítóval (cross-assembler). A PIC mikrovezérlőkhöz a Microchip által kifejlesztett PC-n futó MPASM kereszt-fordító a leghatékonyabb megoldás. Az MPASM az integrált MPLAB fejlesztőkörnyezet része (lásd 3.1 fejezet). A kereszt-fordítóval történő programfejlesztés során valamilyen szövegszerkesztővel kell megírni a forráskódot (jelen estben .ASM kiterjesztésű fájl). A forrásprogram megírható az MPLAB beépített szövegszerkesztőjével, vagy bármilyen más szövegszerkesztővel, a lényeg az, hogy *text* formátumú legyen. A lefordításhoz ennek a fájlnak meg kell felelnie mind a formai (szintaktikai), mind az utasítások, operátorok, direktívák helyes használatát megkívánó szemantikai követelményeknek. A fordítás során többféle fájl keletkezik (*Dr. Kónya*, 2003):

- A fordítási listát tartalmazó nyomtatható szöveges .LST kiterjesztésű fájl
- A hibaüzeneteket tartalmazó .ERR kiterjesztésű fájl
- A szimbólumokat és debug információkat tartalmazó .COD kiterjesztésű fájl
- A tárgyprogram .O kiterjesztéssel, amit az összefűző (linker) program használ fel a továbbiakban
- A gépi kódot tartalmazó, jelen esetben .HEX kiterjesztésű fájl, amit a linker hoz létre az előzőleg külön-külön lefordított modulokból