

3. fejezet

Mikrogépek assembly nyelvű programozása

A 3. fejezet tartalomjegyzéke:

3. Mikrogépek programozása	42
3.1. A programozásról általában	42
3.1.1. A programozás célja	42
3.1.2. A programfejlesztés lépései	43
3.2. Assembly nyelvű programozás	44
3.2.1. A forrásnyelvi program	44
⇒ A címke (label)	47
⇒ Műveleti utasítás és operandusok	48
⇒ Szimbólumok használata	50
⇒ A számok, szövegek ábrázolása	51
⇒ Magyarázatok (comment)	52
⇒ Fordítást vezérlő utasítások	52
3.2.2. Assembler DIREKTIVÁK	52
⇒ Szimbólum definiáló direktívák:	52
⇒ Helyfoglaló, és inicializáló direktívák	55
⇒ Modulok közötti kapcsolatok direktívái	55
⇒ Cím beállító és szegmens választó direktívák	56
3.2.3. Fordítási parancsok	59
⇒ Elsődleges parancsok:	59
⇒ Másodlagos parancsok	63
⇒ Feltételes fordítási parancsok	64

3. Mikrogepek programozása

A fejezetben a 8031/51 típusú mikrokontroller család valamelyik típusával megépített mikrogep assembly nyelvű programozásával foglalkozunk.

3.1. A programozásról általában

A különböző automatizálási, adatfeldolgozási, vagy egyéb alkalmazói feladatok céljára ma már csak programozható „intelligens” berendezéseket, mikroprocesszoros készülékeket használnak. Az ilyen készülékek működését a programmemóriában tárolt kód-sorozat – a program – határozza meg.

A mikrogepek két csoportba sorolhatók aszerint, hogy a felhasználó módosíthatja-e a működtető programot, vagy nem.

A **háztartási** gépekben, vagy a korszerű vezérlőelektronikával ellátott **gépjárművekben**, vagy különböző **gyártó** berendezésekben lévő mikrogepek működtető programját gyárilag írják be a csak olvasható típusú memóriákba (ROM, PROM, EPROM stb.), s így az a felhasználó által nem módosítható. Ezeket **fix programú** készülékeknek nevezzük.

A különböző feladatokra is **felhasználható** készülékeknél – számítógépeknél (PC), programozható logikai vezérlőknél (PLC) stb. - gyárilag csak a **konfigurációnak** megfelelő memória, és periféria-kezeléseket (BIOS, monitor program stb.) készíti el a készülékgyártó, és tárolja a **fix memóriában**. A készülék programmemóriájának egy jelentős területe írható-olvasható (RAM), amelybe felhasználói programok tölthetők, és innen futtathatók. Az ilyen berendezéseket nevezik **szabadon programozható** típusúaknak.

A letölthető felhasználói programokat akkor kell írunk, ha ugyanazon készüléket más-más feladat ellátására is fel kívánunk használni.

A következőekben tömören foglalkozunk a programfejlesztés céljával, eszközeivel, módszereivel, és ellenőrzésének fajtáival.

3.1.1. A programozás célja

A mikrogep működését meghatározó programot az un. **code-**, vagy **programmemóriába** kell tárolni az alkalmazott mikroprocesszor, vagy mikrokontroller - a továbbiakban **processzor** - típusától függő gépi kódok sorozatával. A program végrehajtásakor a processzor innen olvassa ki az utasításokat, és paramétereket.

A programozás *célja* tehát.

- egy olyan *kódsorozat* létrehozása, amelyet
- egy *mikrogep* (mikroprocesszor bázisú számítógép) *programmemóriájába* töltve,
- meghatározza a mikrogep *feladat* szerinti *működését*.

3.1.2. A programfejlesztés lépései

A programfejlesztés az alábbiakban tömören ismertetett lépésekből áll.

A program *felépítésének* megtervezése. Ezt segíti a *folyamatábra* megrajzolása.

- A programban használt *változók*, *állandók*, *periféria-címek* meghatározása.
- szöveges formában - valamilyen szövegszerkesztő (editor) segítségével - kell megírni a programozási-nyelv "helyesírási" (szintaktikai), és tartalmi (szemantikai) szabályai szerint. Az így megírt szöveg a *program forrásnyelvi* formája. A számítógépek, mikrogepek alkalmazásának széleskörű elterjedése a különböző *programnyelv* választékot is bővítette. A programnyelveket *gépközel*- (assembly), és a *magas szintű* nyelvek csoportjába sorolhatjuk.
 - *Gépközel* az a programozás, amelynél minden *programlépés* – programsor - az alkalmazott processzor egy-egy *utasítása*. Mivel minden processzornak saját utasításkészlete van, ezért a programírás is processzorfüggő. Az ilyen programírást nevezzük *assembly* vagy más közelítésben *processzorutasítás szintű* programírásnak.
 - A *magas szintű* programozási nyelvek (Pascal, C, stb) rendszerint a nemzetközileg elfogadott *matematikai*, *logikai* műveletek, és különböző *általános*, a programszerkezetet meghatározó *utasítások* segítségével írják le a feladatot. Ezért e nyelveken a programozás független attól a géptől, amelyen a programot futatni akarjuk. A magas szintű nyelven történő programírást *művelet - központú* programozásnak is nevezhetjük.
- A forrásnyelvi programból a különböző *fordító* programok (assembler, compiller) állítják elő a *program tárgykódú* (object) alakját, illetve a programlistát, amelyek már a processzor utasításkódjait is tartalmazzák. (A leírtakból következik, hogy a magas szintű nyelveknél is a fordítóprogram már processzortól függő.)

- d. A tárgykódú programból, vagy programokból (több modul esetében) a **szerkesztő** (linker) program állítja elő a **futtatható** programváltozatot.

Az így előállított programot kell **beírni** (letölteni) a programmemóriába. A programmemória lehet fix (ROM, EPROM), illetve írható-olvasható (RAM) kialakítású. A beíratás, vagy letöltés történhet teljes kódú (bináris), vagy tömörített (pl. HEX) alakú fájlból. Ennek megfelelően a programozás utolsó lépése a kívánt **kódkonverzió** elvégzése.

3.2. Assembly nyelvű programozás

A különböző assembly nyelvek nagyon sok azonos elemet tartalmaznak. Rendszerint a felépítésük, szintaktikai szabályaik közel azonosak. Lényeges **eltérés** a processzorok utasítás-készleteiben van.

Az assembly nyelvek formai követelményei viszonylag szigorúak. A legerősebb kötöttség a processzor utasítások írása, amelyeket az adott processzor dokumentációjának deklarált formájában szabad használni. Szabadságfok, hogy nincs különbség a kis-, és nagybetűs írásmód között.

Mindegyik assembly - ben használhatók:

- **szimbólumok** a változók, állandók, és címek megadásához,
- **címkék** (label) a programrészek azonosításához,
- **helyfoglaló-értékadó** utasítások (direktívák) a program- és az adatmemória kezeléséhez,
- **műveleti jelek** (**operátorok**) a szimbólumok, címek közötti aritmetikai, logikai műveletekhez,
- **fordítási** (**command**) utasítások, és
- **magyarázatok** (**comment**) a program szöveges leírásához.

3.2.1. A forrásnyelvi program

A **forrásnyelvi** program írásakor törekedni kell az **áttekinthető felépítésre**, a megfelelő **magyarázatokkal** támogatni az érthetőséget. Ugyanakkor szigorúan be kell tartani a **szintaktikai** szabályokat.

A program általános szerkezete a következő:

- **fej** a program jellemzőinek tömör összefoglalása,
- **fordítási** parancsok,

- a használt **változók**, **konstansok**, memória **szegmensek deklarálása**,
- a **program** – törzs (inicializálás, főprogram),
- a **szubrutinok** (alprogramok).

A mikrogép által elvégzendő feladatokat a **főprogramban**, és a **szubrutinokban** kell leírni a processzor **utasításainak** segítségével.

A forrásnyelvi programban minden utasítást külön sorba kell írni. Egy sor általános felépítése a következő:

címke: műveleti utasítás és operandusok ; magyarázat

A sorokból - felsoroltak közül - egy, kettő, vagy mindhárom hiányozhat, vagyis egy sor lehet **üres**, tartalmazhat csak **címkét**, vagy csak **magyarázatot**, vagy csak **utasítást**, illetve ezek lehetséges variációit.

A forrásnyelvi program **ajánlott szerkezete** látható az alábbi példán.

Az alkalmazott **magyarázatok** (comment-ek) segítik a szerkezet felismerését. Mint látható a magyarázó szövegek kezdhetők a sor bármelyik helyén, de mindig **pontosszöveggel** kell kezdeni. A fordító program (assembler) soronként fogja olvasni a forrásnyelvű szöveget, és amikor a sorban ; - t talál, akkor abbahagyja a sor elemzését, és a következő sort fordítja.

```
;xxxxxxx A PROGRAMAZONOSÍTÓ FEJ xxxxxxxx
;*****
;*          BMF KKVM Automatika Intézet          *
;*          Elektronika Szakcsoport              *
;*****
;* Főprogram:          GYAKORLÓ-TESZT            *
;* Változat:           V1.2                      *
;* Dátum:              1999.09.05                *
;* Készítette:         X Y                      *
;*****
;* A modul leírása:                                     *
;*          A mikrokontrolleres gyakorló belső perifériái- *
;*          nak tesztelése.                             *
;*****
```

```
;xxxxxxx FORDÍTÁSI PARANCSONK xxxxxxxx
```

```
$XREF
$DEBUG
NAME   FO_MOD
```

```
;##### DEKLARÁCIÓK #####
```

```
;***** Gyakorló port címei *****
```

```

NGS    XDATA    0C000H    ; nyomógomb-sor címe
LEDS    XDATA    0C001H    ; LED-sor címe

;***** Szegmens deklarálása *****

PROG    SEGMENT    CODE    ; program
VALT    SEGMENT    DATA    ; bájtos változók
VALTB    SEGMENT    DATA BITADDRESSABLE ; bitcímezhető változók
STACK    SEGMENT    DATA    ; stack memória

;***** ADAT szegmens bájtos változóknak *****

RSEG    VALT

DS      10H    ;helyfoglalás 16 bájt részére

;***** ADAT szegmens bitcímezhető változóknak *****

RSEG    VALTB

; xxxxx bájtos változók xxxxx

DS      2    ; nevezetlen helyfoglalás
BEM:    DS      1    ; bemeneti memória
BEPEL:   DS      1    ; pozitív él memória
JELZ:    DS      1    ; jelző bájt

; xxxxx bit változók xxxxx

NG0P    BIT    BEPEL.0    ; NG0 nyomógomb poz.él bitje
NG6P    BIT    NG0P+6    ; NG6 nyomógomb poz.él bitje
NG1N    BIT    BENEL.1    ; NG1 nyomógomb neg.él bitje
JLZB1    BIT    JELZ.0    ; 1.jelzőbit

; xxxxxx STACK kijelölés xxxxxxxx

RSEG    STACK    ;STACK kezdet

DS      10H    ;16 bájt lefoglalása STACK részére

;##### PROGRAMOK #####

;xxxxxxx Program szegmens xxxxxxxx

RSEG    PROG

;xxxxxxx Program-törzs xxxxxxxxx

START:    LJMP    INIT    ; ugrás az inicializálásra

;***** Megszakítás ugrótábla *****

DS      8    ; helyfoglalás a megszakításoknak

;***** Inicializálás *****

INIT:    MOV      SP,#STACK-1

; változóknak kezdőérté adása
; periféria üzemmódok beállítása
; megszakítások engedélyezése

;***** Fő program *****

FOPR:    CALL      BEOLV    ; beolvasás, él meghatározás

```

;xxx a feladatot megoldó program xxx

```

.
.
.

```

; a program további része

;xxx kimenetek vezérlése xxx

FOPRV:

```

CALL      KIIR      ;kimenetek vezérlése
JMP       FOPR      ;visszatérés a főprogram elejére

```

;*** Szubrutinok *******

;xxxxxxx beolvasó és él képző rutin xxxxxx

BEOLV:

```

MOV        DPTR,#NGS
MOVX  A,@DPTR      ;beolvasás a nyomógomb-sorról
.
.
.
RET        ;visszatérés a főprogramba

```

;xxxxxxx kiviteli rutin xxxxxx

KIIR:

```

MOV        DPTR,#LEDS      ;kimeneti memória kivitele
MOV        A,KIM           ;a kimeneti perifériára –LED sorra
.
.
.

```

RET

```

END

```

; a fordítóprogramnak jelzi a for-
rás
;végét

⇒ **A címke (label)**

A címkét olyan programrészek első utasítása elé kell írni, amelyekre a program egyéb helyein hivatkozunk, vagyis ahová a vezérlést át kell adni (ugrás, vagy rutin hívás). A **címke** tulajdonképpen a hivatkozási hely címét tartalmazó **szimbólum**. A programban azonos címkék nem használhatók!

A címke egy betűvel ('a' 'z', 'A'..... 'Z', '.') kezdődő karaktersor (szimbólum, jelző), amelynek további karakterei számjegyek is lehetnek, és a ':' karakterrel kell befejezni. Magyar ékezetes betűk használata tilos! Elvileg a karakterek száma nincs korlátozva, de a fordító csak az első nyolc karaktert tárolja, és nem tesz különbséget a kis és nagybetűk

között. (Ha két címke csak a kilencedik karaktertől kezdve különbözik, akkor a fordító hibát jelez, mivel azonosnak tekinti azokat.)

A címkéket rendszerint a sor elejére írjuk, kiemelve ezzel a sor jelző szerepét.

Helyes címkek:

Kezdet:

_1sor:

foprogr_2:

Helytelen címkek:

2.sor:

Ugrás.1:

Első sor:

⇒ Műveleti utasítás és operandusok

A programsorban a ***műveleti utasítás*** írja elő a CPU által végrehajtandó műveletet, és a hozzá csatlakozó ***operandus*** - ok adják meg a műveleti ***tényezőket***, vagy azok ***címét***. Az operandus lehet:

- speciális assembler szimbólum (pl. regiszter),
- program szimbólum,
- indirekt cím,
- konstans,
- belső RAM cím,
- bit-cím,
- program cím.

A használható speciális assembler szimbólumok:

A (ACC)	Akkumulátor
B	B regiszter
R0...R7	az aktuális regiszter-bank munkaregiszterei
DPTR	a külső memóriák címzéséhez használt regiszterpár (DPH-DPL)
PC	programszámláló
C	túlcsordulás bit
AB	regiszterpár szorzásnál, és osztásnál
AR0..AR7	az aktuális regiszterbank munkaregisztereinek abszolút címe

A lehetséges műveleti utasításokat, a hozzá tartozó operandusokat, és helyes írásmódjukat az alkalmazott mikroprocesszor / mikrokontroller **utasításkészlet** - e tartalmazza.

A **műveleti utasítást** egy szóval, az ún **mnemonic** - al írják le, amely a művelet angol megnevezése, vagy annak rövidítése.

Az operandusok lehetnek **szám** - ok, vagy **szimbólum** - ok, illetve meghatározott írásjel, amely meghatározza az adott operátor címezési módját.

pl.

MOV 20 , 45

sorban a műveleti utasítás a **MOV** mnemonic a mozgatást jelenti, amely után meg kell adni először a **cél cím** - ét, (itt a 20-as című memóriahely), majd a **forrás** címét, vagy értékét (példánkban a 45 ugyancsak cím). Az utasítás tehát arra utasítja a CPU - t, hogy a 20 - as címen található adatot másolja át a 45 címen lévő helyre.

A következő utasításban

MOV 20 , #45

a második operandus előtt álló **#** jel azt jelenti, hogy az utána következő szám egy érték, vagyis a CPU feladata, hogy a 20 - as című helyre írjon 45-t.

Az operandusok **szimbólumokkal** is megadhatók.

Az operandusok, vagy új szimbólumok előállíthatók a már deklarált szimbólumok operátorokkal történő összekapcsolásával.

Operátorok:

- **aritmetikai:**

+, -	előjel	(+5, -0AH)
+, -	összeadás, kivonás	(cím + offset)
*	szorzás	(1200H*7)
/	osztás	(17/4)
MOD	maradék	(17 MOD 4)
()	csoportosítás	((cím-offset)*3)

- **logikai:**

NOT	egyes kompl.	(NOT 5)
HIGH	szó felső bájtja	(HIGH 1234)
LOW	szó alsó bájtja	(LOW 1234)
SHR,SHL	jobbra/balra lépt.	(2 SHR 8)
AND	ÉS művelet	(SIMB AND 0AH)
OR	VAGY művelet	(SIMB1 OR SIMB2)
XOR	KIZÁRÓ-VAGY m.	(12H XOR 5)

- **hasonlító:**

>= / GTE	nagyobb egyenlő	(SIMB >= 13H)
<= / LTE	kisebb egyenlő	(SIMB LTE 0A1H)
<> / NE	nem egyenlő	(SIMB NE 045H)
= / EQ	egyenlő	(SIMB = 0A2H)
< / LT	kisebb mint	(SIMB < 0B2H)
> / GT	nagyobb mint	(SIMB GT 051H)

⇒ **Szimbólumok használata**

A programozást nagyban segíti, ha számokkal megadott címek, változók helyett a jelentésre utaló szavakat, rövidítéseket, vagyis **szimbólumok** - at használunk.

Szimbólumok a már megismert **címke**, illetve utasítás **mnemonic** - ok is.

- a **címke** egy programcímet helyettesít,
- az utasítás **mnemonic** a CPU utasításkódját.

Szimbólumokkal lehet helyettesíteni

- **változók** tárolási helyének címét,
- **konstansokat**.

Szimbólumban csak betű, számjegy és a '_' karakter szerepelhet.

A processzor utasítás - mnemonicjai, illetve a belső regiszterek elnevezései, és a direktívák nem használhatók általános szimbólumként.

A szimbólum deklarálásához az **EQU** direktívát kell használni. pl.

EQU BE_GOMB 12

és felhasználása programsorban a

MOV C , BE_GOMB

azt jelenti, hogy a CPU túlsordulás bit -jébe (C) másolja át a 20 cím bit értékét.

⇒ **A számok, szövegek ábrázolása**

A számítástechnikában nem csupán a *decimális*, hanem a *bináris*, az *oktális*, és a *hexa-decimális* számábrázolást is használják, illetve az ASCII szabvány szerinti *karakter*, és *string* megadás is szükséges.

Bináris:

a bináris számjegyek sorozata után irt B betű

pl. **011010 b** vagy **011010 B**

Decimális:

a decimális számjegyek sorozata vagy utána irt D betű

pl. **345** vagy **345 D**

Oktális:

az oktális számjegyek sorozata után irt O betű

pl. **137 o** vagy **137 O**

Hexadecimális:

a hexadecimális számjegyek sorozata után irt H betű, ha szám a betű karakterek valamelyikével kezdődik akkor 0 – kell elé írni

pl. **15A H** vagy **0E34 H** .

Karakterek

' A ' formában megadott karaktert az ASCII kóddal helyettesít.

Szöveg (string)

" Ez egy szöveg " az egyes karaktereket ASCII kóddal helyettesítve helyezi le.

⇒ **Magyarázatok (comment)**

A programrészletekhez *magyarázatok* (comment) is írhatók a forrásnyelvi programokban. Ezzel segíteni lehet a programlépések megértését, az áttekinthetőséget. A magyarázó szöveget mindig a ; karakterrel kell kezdeni.

pl.:

; Példa a magyarázat elhelyezésére

```
KEZD:                ; itt kezdődik a program
MOV A,#5EH           ; az akkumulátor feltöltése
ANL A,KIM;           a kimeneti memória maszkolása
JZ    TOV
```

A magyarázat szerepelhet egy önálló sorban is, de ha van címke, vagy utasítás, illetve mindkettő, akkor csak ezeket követően írható.

⇒ **Fordítást vezérlő utasítások**

A forrásnyelvi fájlból a fordítóprogram (az assembler) készít tárgykódú (.obj) fájlt.

A fordító számára is lehet utasításokat adni, az ún. *direktívák* segítségével. A legfontosabb direktívák a következők:

3.2.2. Assembler DIREKTIVÁK

A direktívák tulajdonképpen a fordítást vezérlő parancsok. Ezek segítségével

- definiálhatók szimbólumok,
- foglalhatók le és inicializálhatók memória területek,
- kapcsolhatók össze modulok,
- állíthatók be program-, és szegmenscímek.

⇒ **Szimbólum definiáló direktívák:**

EQU A szimbólumhoz rendel egy értéket, vagy regiszter nevet. A szimbólum értéke később már nem változtatható.

pl.

```
HATAR EQU 1200
```

ERTEK	EQU	HAT+R - 'A'
AKKU	EQU	A
SZAML	EQU	R7

SET A szimbólumhoz úgy rendel értéket, vagy regisztert, hogy az újra definiálható.
pl.

TAR	SET	R0
VALT	SET	1AH
.		
.		
TAR	SET	R1
VALT	SET	0D2H

DATA A szimbólumhoz rendel egy direkt címezhető belső memória címet.
pl.

BEM1	DATA	20H
KIM2	DATA	22H

IDATA A szimbólumhoz rendeli a csak indirekt címezhető belső memória egy címét.
pl.

OSSZEG	IDATA	60H
MARAD	IDATA	OSSZEG - 1

XDATA A szimbólumhoz rendeli egy külső memória címét.
pl.

TABL	XDATA	100H
KIF1	XDATA	TABL + 23H

CODE A szimbólumhoz rendeli a programmemória egy címét.
pl.

START	CODE	00H
INTV_0	CODE	START + 3

SEGMENT Egy relokálható SZEGMENS deklarációját biztosítja. A következő formában használható:

Szegm_Nev **SEGMENT** **Szegm_típus** [**elh_tip**]

ahol a

Szegm_Nev a szegmens funkciójára utaló név,
Szegm_típus megadja, hogy a szegmens melyik memóriaterületre legyen letöltve,
elh_tip a letöltés kezdetét határozza meg [opcionális].

Szegmens-típusok:

CODE program memória,
XDATA külső adatmemória,
DATA a direkt címezhető belső memória,
IDATA indirekt címezhető belső memória,
BIT a bit-címezhető belső memória.

Elhelyezési típusok:

PAGE lapkezdetre igazítás (csak CODE és XDATA után),
INPAGE a szegmens csak egy lapon belül lehet (csak CODE és XDATA után),
INBLOCK a szegmens csak egy 2048 bájtos blokkot foglalhat el (csak CODE után),
BITADDRESSABLE
 a belső memória bit-címezhető 16 bájt-ja (20H - 2FH) lehet (csak DATA és IDATA után),
UNIT egy biten, vagy egy bájton kezdődő szegmens,
OVERLAYABLE
 olyan szegmens, amelyet a C-51 deklarált, és ebbe beszerkesztés engedélyezett.

⇒ **Helyfoglaló, és inicializáló direktívák**

DS adott számú összefüggő memóriaterületet foglal le (bármelyik memóriában), használata:

[Címke:] DS szám, vagy kifejezés

DBIT adott számú bitet foglal le, használata:

[Címke:] DBIT szám, vagy kifejezés

DB a programmemóriában ad kezdőértéket a felsorolt bájtoknak, használata:

[Címke:] DB szám[,kifejezés][,szimbólum]...

DW a programmemóriában ad kezdőértéket a felsorolt szavaknak, használata:

[Címke:] DW szám[,kifejezés][,szimbólum]...

⇒ **Modulok közötti kapcsolatok direktívái**

Feladatuk a különböző modulokban deklarált szimbólumok elérésének biztosítása.

PUBLIC a direktívával deklarált szimbólumok minden modulból elérhetők, használata:

PUBLIC Szimb[,Szimb[,....]]

EXTRN egy másik modulban deklarált szimbólum elérését biztosítja az aktuális modulban. Használata:

EXTRN Szegm_típus(Szimb_lista)

NAME az egyes tárgy-modulok (object) megkülönböztetését teszi lehetővé. Ha nem adjuk meg, akkor a forrásfájl neve lesz a tárgy-modul neve is. Használata:

NAME Tárgy_mod_név

⇒ **Cím beállító és szegmens választó direktívák**

A direktívák segítségével adhatók meg programrészek, szegmensek kezdő címei, illetve választhatók már létező szegmensek.

ORG meghatározza a következő utasítás, vagy adat címét.

ORG 100H

ORG START

END a forrásprogram végét jelzi. (Mindig kell használni)

RSEG egy - már korábban definiált - relokálható szegmens kiválaszt

RSEG Szegmens_név

CSEG, DSEG, XSEG, ISEG, BSEG

az egyes szegmensek kezdőcímét lehet megadni a direktívák segítségével. Használatuk:

CSEG [AT absz-cím]

DSEG [AT absz-cím]

XSEG [AT absz-cím]

ISEG [AT absz-cím]

BSEG [AT absz-cím]

Amennyiben nincs cím, akkor a fordításnál mindegyik szegmens 0 címnél kezdődik.

USING az aktuális regiszterbankot választja ki. Használata:

USING sorsszám (a sorszám 0...3 lehet)

A szegmensek kijelölésére, a változók elhelyezésére, és a különböző helyfoglalásokra mutat mintát a PELDA1.LST fájl. A bemutatott példa a forrásnyelvi fájl fordítása után létrehozott *lista-fájl*t szemlélteti. A lista végén látható *szimbólum-táblázat* megadja, a

szimbólum nevét, típusát (C,D,B,N), relokalizálható **R**, vagy abszolút címre **A** került a deklarációnál, valamint azt, hogy a lista hányadik sorában került deklarációra #, és melyik sorokban használt még az adott szimbólum. A relokalizálhatóknál még a szegmensnév is látható.

Megfigyelhető, hogy minden relatívan (relokalizálhatóan) deklarált szegmens változóinak címe 0-án kezdődik. A tényleges címet a szerkesztés (linkelés) után kapják meg.

A51 MACRO ASSEMBLER PELDA1

DATE 02/09/96 PAGE 1

MS-DOS MACRO ASSEMBLER A51 V4.4

OBJECT MODULE PLACED IN C:\XE2A51\GYAK\PELDA1.OBJ

ASSEMBLER INVOKED BY: A51 C:\XE2A51\GYAK\PELDA1.A51

LOC OBJ LINE SOURCE

```

1      ;xxxxxxx fordítási parancsok xxxxxxxx
2
3  $XREF
4  $DEBUG
5
6  ;*****
7  ;*      K K M F Automatika Intézet      *
8  ;*      Elektronika Szakcsoport      *
9  ;*****
10 ;* Fprogram: Helyfoglalás adatoknak      *
11 ;* Változat:      V1.0      *
12 ;* Dátum:      1996.08.20      *
13 ;* Készítette:      Zalotay Péter      *
14 ;*****
15 ;* A modul leírása:      *
16 ;*      Példák a különböző helyfoglalási megoldásokra      *
17 ;*****
18
19 NAME      Pelda_1
20
21 ;xxxxxxx Deklarációk xxxxxxxx
22
23 ;**** Relatív címmegadások (relokalizálható szegmensek)****
24
25 VALT1      SEGMENT      DATA ;Adatszegmens
   a direkt címezhető belső mem.-ban
26 VALT2      SEGMENT      DATA BITADDRESSABLE
   ;Adatszegm. a bit címezhető belső m.-ban
27 VALTB      SEGMENT      BIT
   ;Bit szegmens
28 STACK      SEGMENT      IDATA
   ;Stack szegmens

```

```

29  PROG    SEGMENT    CODE
      ;Program szegmens
30
31      ;***** Program szegmens *****
32
33      RSEG  PROG
34
0000 758100 F  35  MOV    SP,#STACK-1    ;A program első utasítása
36
37
      ;A program      utolsó utasítása
39
40      ;***** Adatszegmens a direkt címezhető belső memóriában      *****
41
42      RSEG  VALT1
43
0000      44  V1:    DS    1      ;Helyfoglalások a V1,V2,V3 változóknak
0001      45  V2:    DS    2
0003      46  V3:    DS    1
0004      47          DS    20      ;Nevezetlen memóriaterület lefoglalása
48
49      ;***** Adatszegmens a bit címezhető belső memóriában *****
50
51      RSEG  VALT2
52
0000      53  BEM:    DS    1      ;Helyfoglalás a BEM és KIM változóknak
0001      54  KIM:    DS    1
55
56      ;***** Bit szegmens *****
57
58      RSEG  VALTB
59
0000      60  B1:    DBIT  1      ;Helyfoglalás a B1,B2,B3 kétértékű
      ;változóknak
0001      61  B2:    DBIT  1
0002      62  B3:    DBIT  1
63
64      ;***** Stack szegmens ****
65
66      RSEG  STACK
67
0000      68          DS          10H      ;Stack memória lefoglalása
69
70
71      ;***** Abszolút címmegadások ****
72
73      CSEG  AT    1000H
74
1000 7800 F  75  FOLYT:  MOV    R0,#KIM
76
77      DSEG  AT    30H
78
0030      79  BEM2:    DS          1
0031      80  KIM2:    DS          1
0032      81  V4        DATA  32H
00AF      82  K1        EQU    0AFH
83
84      BSEG  AT    30H

```

```

85
0030      86  B4:      DBIT    1
0031      87  B5:      DBIT    1
0000      88  B6       BIT      BEM.0
89
90
91      END

```

XREF SYMBOL TABLE LISTING

NAME TYPE VALUE ATTRIBUTES / REFERENCES

```

B1 . . . . B ADDR 0000H.0    R    SEG=VALTB 60#
B2 . . . . B ADDR 0000H.1    R    SEG=VALTB 61#
B3 . . . . B ADDR 0000H.2    R    SEG=VALTB 62#
B4 . . . . B ADDR 0026H.0    A     86#
B5 . . . . B ADDR 0026H.1    A     87#
B6 . . . . B ADDR 0000H.0    R    SEG=VALT2 88#
BEM. . . . D ADDR 0000H      R    SEG=VALT2 53# 88
BEM2 . . . D ADDR 0030H      A     79#
FOLYT . . . C ADDR 1000H      A     75#
K1 . . . . N NUMB 00AFH      A     82#
KIM. . . . D ADDR 0001H      R    SEG=VALT2 54# 75
KIM2 . . . D ADDR 0031H      A     80#
PELDA_1 . . ---- ----      19
PROG . . . C SEG 0003H      REL=UNIT 29# 33
SP . . . . D ADDR 0081H      A     35
STACK.I SEG 0010H          REL=UNIT 28# 35 66
V1 . . . . D ADDR 0000H      R    SEG=VALT1 44#
V2 . . . . D ADDR 0001H      R    SEG=VALT1 45#
V3 . . . . D ADDR 0003H      R    SEG=VALT1 46#
V4 . . . . D ADDR 0032H      A     81#
VALT1. D SEG 0018H          REL=UNIT 25# 42
VALT2. D SEG 0002H          REL=BITADDRESSABLE 26# 51
VALTB.B SEG 0003H          REL=UNIT 27# 58

```

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

3.2.3. Fordítási parancsok

A forrásnyelvi fájl fordítási módozatára adhatók a fordítási parancsok. A parancsok között az *elsődleges* típusúak csak egyszer használhatók, és a forrás-program legelső - értékelhető - sorában kell megadni (megjegyzés, magyarázó szöveg megelőzheti). A *másodlagos* típusú parancsok a programban többször, és bármely helyen megadhatók. Mindkét típusú parancsot sor elejére a \$- jellel kezdve kell írni.

A fordítási parancsok egy külön csoportját alkotják a *feltételes parancsok*, melyek segítségével - megadott feltétektől függően - programrészek kitilthatók a fordításból.

Elsődleges parancsok:

DATE (rövidítve: DA)

A lista minden oldalán a fejlécbe dátum írása. A dátum maximálisan 9 betűs lehet.

Alapértelmezés: MS-DOS szerinti írásmód

pl.:

\$ DATE (12/07/96)

DEBUG/NODEBUG (rövidítve: DB/NODB)

A DEBUG parancs hatására a szimbólum információk (címkék, szimbólumok és értékük) is belekerülnek a *.obj fájlba. Ezek révén a szimulátor, illetve emulátor is megjeleníti ezeket a szimbólumokat.

Alapértelmezés: NODEBUG

pl.:

\$ DEBUG

\$ DB

\$ NODB

ERRORPRINT/NOERRORPRINT (rövidítve: EP/NOEP)

A fordítás során felismert hibákat lehet egy megadott fájlba kiíratni. Nem változtatja meg azt, hogy a hibák a lista fájlba is bekerülnek.

Alapértelmezés: NOEP

pl.:

\$ EP(PROG1:ERR)

\$ NOEP

OBJECT/NOOBJECT (rövidítve: OJ/NOOJ)

Parancs object fájl készítésére, vagy letiltására. Ha nem adunk meg nevet, akkor a forrásfájl nevén készül a fájl.

Alapértelmezés: OBJECT (fájl_név.OBJ)

pl.:

\$ OBJECT (C:\XE2A51\PELDA1.OBJ)

\$ NOOJ

PAGELENGTH (rövidítve: PL)

A listázásnál egy lapra irt sorok számát határozza meg az utasítás után zárójelbe irt - 10-nél nagyobb - szám .

Alapértelmezés: PL (68)

pl.:

\$ PLENGTH (132)

\$ PL (75)

PAGEWIDTH (rövidítve PW)

A listázásnál egy sorba irt karakterek számát határozza meg az utasítás után zárójelbe irt - 8 és 132 közötti - szám.

Alapértelmezés: PW(120)

pl.:

\$ PW (79)

\$ PAGEWIDTH(122)

PRINT/NOPRINT (rövidítve: PR/NOPR)

Készítsen, vagy ne készítsen lista fájlt. Ha a PRINT utasítás után nem adunk meg fájl nevet, akkor a forrásnéven készül a lista.

Alapértelmezés: PRINT(forrás_név.LST)

pl.:

\$ PRINT

\$ NOPR

\$ PR(temp.lst)

SYMBOLS/NOSYMBOLS (rövidítve: SB/NOSB)

Írjon, vagy ne írjon szimbólum azok attribútumaival táblázatot a lista végére.

Alapértelmezés: SYMBOLS

pl.:

\$ SYMBOLS

\$ NOSB**MOD51/NOMOD51** (rövidítve: MO/NOMO)

Az alapértelmezésben (MOD51) az assembler felismeri a 8051 kontroller regiszter és SFR szimbólumait. Amennyiben más típushoz készül a program, akkor a NOMOD51 parancsot kell megadni, és az alkalmazott kontroller szimbólumait tartalmazó un. include fájlt (pl.: REG552.INC) kell a program elején beolvastatni az INCLUDE másodlagos paranccsal.

Alapértelmezés: MOD51

pl.:

\$ NOMOD51

COND/NOCOND (rövidítés: nincs)

A COND parancs hatására feltételes fordítás (IF-ELSEIF-ENDIF szerkezet) érvénytelen részét is listázza, míg a NOCOND megadásnál nem.

Alapértelmezés: COND

pl.:

\$COND

\$ NOCOND

MACRO/NOMACRO (rövidítés :nincs)

Alapértelmezésben felismeri, és feldolgozza a macro - definíciókat. A NOMACRO utasítás hatására a fordító nem dolgozza fel a macro -kat.

Alapértelmezés: MACRO

pl.:

\$MACRO

\$ NOMACRO

REGISTERBANK/NOREGISTERBANK (rövidítve: RB/NORB)

Az RB parancs határozza meg, hogy a programban melyik (zárójelbe irt szám/számok) regiszterbankokat kívánjuk használni. A NORB parancs megszünteti a helyfoglalást a bank számára.

Alapértelmezés: REGISTERBANK(0)

pl.:

\$ REGISTERBANK (0,1)

\$ NORB

XREF/NOXREF (rövidítve: XR/NOXR)

Írjon, vagy ne írjon szimbólum táblázat után keresztreferencia listát.

Alapértelmezés: NOXREF

pl.:

\$ XREF

\$ NOXR

TITLE (rövidítve: TT)

Írjon a lap tetejére fejléct. A szöveget zárójelben kell a parancs után írni.

Alapértelmezés: TITLE a fájl neve kiterjesztés nélkül, vagy a 'NAME' parancssal megadott név.

pl.:

\$ TITLE (modul meghatározás)

⇒ Másodlagos parancsok

EJECT (rövidítve: EJ)

A parancs hatására a listában lapdobás következik

pl.:

\$ EJ

INCLUDE (rövidítve: IC)

A parancstól kezdve a fordító a zárójelbe irt fájlt fordítja és iktatja be az object fájlba, majd folytatja a forrásfájl fordítását. Maximálisan kilenc ilyen közbeiktatást alkalmazhatunk.

pl.:

\$ INCLUDE (REG552.INC)

LIST/NOLIST (rövidítve: LI/NOLI)

A parancsok arra adnak utasítást, hogy a következő részeket listázza, vagy ne. Ezzel meghatározott részek listázása mellőzhető.

Alapértelmezés: LIST

pl.:

\$ LIST

\$ NOLIST

GEN/NOGEN (rövidítés: nincs)

A parancsok azt határozzák meg, hogy a makró-kifejtés bekerüljön, vagy ne a listába.

Alapértelmezés: GEN

pl.:

\$ GEN

\$ NOGEN

⇒ **Feltételes fordítási parancsok****SET/RESET**

A feltételes fordítás számára deklarál, vagy szüntet meg szimbólumot és rendel a szimbólumhoz értéket. Ha csak a szimbólumot deklaráljuk, akkor értéke 0FFFFH lesz.

pl.:

\$ SET (TMP, VALT= 55)

\$ RESET (TEMP, VALT)

IF

A feltételes szerkezet kezdő parancsa. Utána kell megadni az értékelendő feltételt. Ha a feltétel teljesül, akkor folytatódik a fordítás, ellenkező esetben az ENDIF parancs utáni rész fordítása következik.

pl.: \$ IF (ALT=55)

1. programrész

\$ ENDIF

2. programrész

ELSE

Választásos feltételes szerkezet (IF-ELSE-ENDIF) parancsa. Ha az IF feltétele nem teljesül, akkor az ELSE utáni rész fordítása következik.

pl.:

```
$ IF (VALT=55)
    1. programrész
$ ELSE
    2. programrész
$ ENDIF
    3. programrész
```

ELSEIF

Egymásba ágyazott választásos feltételes szerkezet parancsa. Az ELSE ágon belül újabb IF, vagy IF-ELSE szerkezet beiktatását teszi lehetővé.

pl.:

```
$ IF (VALT=55)
    1. programrész
$ ELSEIF (VALT2)
    2. programrész
$ ELSEIF (SWITCH=2)
    3. programrész
$ ENDIF
    4. programrész
```

END pl.:IF

A feltételes fordítási szerkezet lezáró parancsa.