Globális szimbólumok

A több modulból is elérhető szimbólumok.

A globális szimbólumok teszik lehetővé, hogy a programjainkat modulokra bontva készítsük el. Az egyes modulok közötti kapcsolatot a globális szimbólumok jelentik.

Globális szimbólumok

Ha egy szimbólumot globálissá kívánunk tenni, akkor **PUBLIC**-ká kell nyilvánítanunk annak a modulnak az elején, amelyben a szimbólumot definiáljuk:

```
PUBLIC    sz1[, sz2...]
```

Azokban a modulokban, amelyekben más modulban definiált szimbólumokat is használni szeretnénk, az ilyen szimbólumokat **EXTRN**-né kell nyilvánítanunk:

```
EXTRN sz1:típus1[, sz2:típus2...]
```


INCLUDE utasítás

```
INCLUDE     File_Specifikáció
```

hatására az assembler az **INCLUDE** utasítás helyére bemásolja az utasítás paraméterében specifikált file szövegét.

Az **INCLUDE**-olt file-ok is tartalmazhatnak **INCLUDE** utasítást.

Ha makró definícióinkat a **MyMacros.i** file-ba kívánjuk összegyűjteni, akkor célszerű ezt a file-t így elkészítenünk:

```
IFNDEF   MyMacros_i
MyMacros_i   = 1
...      ;; makró, struktúra definíciók
...      ;; EXTRN szimbólumok
ENDIF
```

Ekkor a **MyMacros.i** file legfeljebb egyszer kerül bemásolásra, mert az összes további esetben a feltételes fordítás feltétele már nem teljesül.

A **.**-ot **_**-sal helyettesítettük!

A legtöbb include file-ban ezt a konvenciót alkalmazzák.

Lista vezérlési utasítások

```
TITLE       cím
```

A fordítás során keletkező lista minden oldalán megjelenik ez a **cím**. Egy modulon belül csak egyszer alkalmazható.

```
SUBTITLE    alcím
```

Többször is előfordulhat egy modulon belül. A program lista minden oldalán – a cím alatt – megjelenik az utolsó **SUBTITLE** utasításban megadott **alcím**.


```
PAGE        [op1][,op2]
```

Paraméter nélkül lapdobást jelent.

Ha egyetlen paramétere van és az egy **+** jel, akkor a fejezet sorszámát növeli eggyel, és a lapszámot **1**-re állítja.

Ettől eltérő esetekben **op1** az egy lapra írható sorok (**10≤op1≤255**), **op2** az egy sorba írható karakterek számát jelenti (**60≤op2≤132**).

Ha valamelyik paramétert nem adjuk meg, akkor természetesen nem változik a korábban beállított értéke.

A sorok száma kezdetben **66**, a karaktereké **80**.

A **TITLE**, a **SUBTITLE** és a **PAGE** egy elkészült programcsoport végső papír-dokumentációjának jól olvashatóságát segíti.

A programfejlesztés során ritkán készítünk program listákat.

NAME név

A modul nevét definiálhatjuk vele. A szerkesztő ezt a nevet fogja használni. Ha nem szerepel **NAME** utasítás a modulban, akkor a **TITLE** utasítással megadott cím a modul neve. Ha ez sincs, akkor a file nevének első **6** karaktere lesz a modul neve.

COMMENT határoló_jel szöveg határoló_jel

Segítségével több soros kommentárokat írhatunk. Az assembler a **COMMENT** utáni első látható karaktert tekinti **határoló_jel**-nek, és ennek a jelnek az újabb előfordulásáig minden kommentár. Nyilvánvaló, hogy a kommentár belsejében nem szerepelhet **határoló_jel**.

%OUT szöveg

Amikor ehhez az utasításhoz ér a fordítóprogram, akkor a paraméterként megadott **szöveg**-et kiírja a képernyőre.

.RADIX számrendszer_alapja

Ha programban egy szám nem tartalmaz számrendszer jelölést, akkor az illető számot ebben a számrendszerben kell érteni (alapértelmezésben decimális).

.LIST

Engedélyezi a forrás- és tárgykódú sorok bekerülését a lista file-ba (alapértelmezés).

.XLIST

Tiltja a forrás- és tárgykódú sorok bekerülését a lista file-ba. Jól használható arra, hogy **INCLUDE** előtt tiltsuk a listázást, utána **.LIST** -el újra engedélyezzük, és ezzel az **INCLUDE** file-ok többszöri listázását elkerüljük.

.LFCOND

Minden feltételes blokk kerüljön listára.

.SFCOND

Csak a teljesülő feltételes blokkok kerüljenek listára (alapértelmezés).

.TFCOND

Vált a két előző listázási mód között.

.CREF

Készüljön kereszthivatkozási (cross-reference) tábla (alapértelmezés). Ez a tábla azt a célt szolgálja, hogy könnyen megtaláljuk az egyes változókra történő hivatkozásokat a programban.

.XCREF

Ne készüljön kereszthivatkozási tábla.

.LALL

Kerüljön listára a makró hívások kifejtése.

.SALL

Ne kerüljön listára a makró hívások kifejtése.

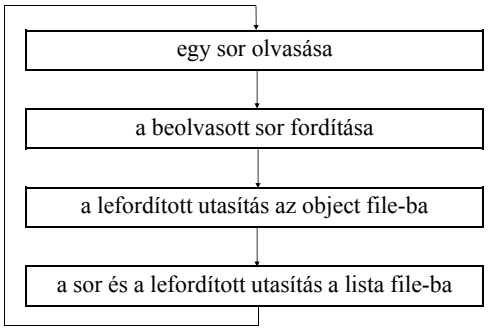
.XALL

A makró hívások kifejtéséből csak a kódot generáló rész kerüljön listára (alapértelmezés).

END kifejezés

A modul végét jelzi, **kifejezés** a program indítási címe. Ha a programunk több modulból áll, akkor természetesen csak egy modul végén adhatunk meg kezdő címet.

Assembler



előre hivatkozási probléma

Megoldási lehetőség:

Az assembler kétszer olvassa a program szövegét (két menet). Az első menet célja összegyűjteni, táblázatba foglalni a szimbólum definíciókat, így a második menet idején már minden (a programban definiált) szimbólum ismert, tehát a második menetben már nem jelentkezik az előre hivatkozási probléma.

Valahogy megpróbálni a fordítást egy menetben. Késleltetni a fordítást ott, ahol előre hivatkozás van, pl. táblázatba tenni a még le nem fordított részeket. A menet végén már minden szimbólum ismert, ekkor feldolgozni a táblázatot. Esetleg minden szimbólum definíciót követően azonnal feldolgozni a szimbólumra vonatkozó korábbi hivatkozásokat.

Mindkét esetben szükség van szimbólum tábla készítésére, de az utóbbi megoldásban a még le nem fordított utasítások miatt is szükség van táblázatra. További nehézséget jelent, hogy nem sorban készülnek el a tárgy kód (object code) utasításai, ezért ezeket pl. listába kell helyezni, majd rendezni a listát, és csak ezután történhet meg az object és a lista file elkészítése.

Manapság a legtöbb assembler két menetben működik.

Két menetes assembler, első menet

Legfontosabb feladata a szimbólum tábla felépítése.

A szimbólum tábla:

A szimbólum neve	értéke	egyéb információk
. . .	. . .	. . .

érték: – címke címe,
– változó címe,
– szimbolikus konstans értéke.

A szimbólum neve	értéke	egyéb információk
. . .	. . .	. . .

egyéb információk:
- típus,
- méret,
- szegmens neve, amelyben a szimbólum definiálva van,
- relokációs flag,
- . . .

Literál:

pl. az **IBM 370**-es gépcsaládon:

```
L       14,=F'5'  ; Load register 14 az 5-ös
                  ; Full Word konstanssal
```

Többek között a literálok gyakori használata vezetett a közvetlen operandus megadás kialakulásához és elterjedéséhez.

Egy lehetséges operációs kód tábla részlete:

mnemonic	op1	op2	kód	hossz	osztály
AAA	-	-	37	1	6
ADD	reg8	reg8	02	2	10
ADD	reg16	reg16	03	2	11
. . .	. . .	. . .	. . .	. . .	. . .
AND	reg8	reg8	22	2	10
AND	reg16	reg16	23	2	11
. . .	. . .	. . .	. . .	. . .	. . .

```
procedure ElsőMenet;  {1. menet, vázlat}
const méret = 8; EndUtasítás = 99;
var
  HelySzámláló, osztály, hossz, kód:
    integer;
  VanInput: boolean;
  szimbólum, literál, mnemo:
     array[1..méret] of char;
  sor: array[1..80] of char;
begin
  Előkészítés;
  TáblákIinicializálása;
  HelySzámláló := 0;
  VanInput = true;
```



```
while VanInput do begin     {sorok feldolgozása}
  SorOlvasás(sor);
  Megőrzés(sor);
  if NemKomment(sor) then begin  {nem kommentár}
    SzimbólumDef(sor, szimbólum);
    if szimbólum[1] <> ' ' then
                          {szimbólum definíció}
      ÚjSzimbólum(sor,szimbólum, HelySzámláló);
    LiterálKeresés(sor, literál);
    if literál[1] <> ' ' then
      ÚjLiterál(literál);
    hossz := 0;
    OpKódKeresés(sor, mnemo);
    OpKódTáblában(sor, mnemo, osztály, kód);
```



```
      if osztály < 0 then   {nem létező utasítás}
        PszeudoTáblában(sor,mnemo, osztály, kód);
      if osztály < 0 then HibásOpKód;
      hossz := típus(osztály);  {utasítás hossza}
      HelySzámláló := HelySzámláló + hossz;
      if osztály = EndUtasítás then begin
        VanInput := false;
        LiterálTáblaRendezés;
        DuplikátumokKiszűrése;
        Lezárások;
      end;   {if osztály = }
    end;     {nem kommentár}
  end;       { while VanInput }
end;         {1. menet}
```