5.

- **LOW kifejezés:** egy szó alsó (alacsonyabb helyértékű) byte-ja;
- **HIGH kifejezés:** egy szó felső (magasabb helyértékű) byte-ja;

Pl.:

```
ADATW   dw      1234H
        mov     al,LOW ADATW   ; al ⇐ 34H
        mov     ah,HIGH ADATW  ; ah ⇐ 12H
```

6. Előjelek:

- **+ :** pozitív előjel;
- **– :** negatív előjel;

7. Multiplikatív műveletek:

- *** :** szorzás;
- **/ :** osztás;
- **MOD:** (modulo) a legkisebb nem negatív maradék, pl.:

```
mov     al,20 MOD 16   ; al ⇐ 4
```

- **kifejezés SHL lépés:**
  kifejezés léptetése balra **lépés** bittel;
- **kifejezés SHR lépés:**
  kifejezés léptetése jobbra **lépés** bittel;

  lépés is lehet kifejezés!

A kifejezésben előforduló műveleti jelek (**SHL**, **SHR**, és a később előforduló **NOT**, **AND**, **OR**, és **XOR**) nem tévesztendők össze a velük azonos alakú műveleti kódokkal: az előbbiket a fordító program, az utóbbikat a futó program hajtja végre!

8. Additív műveletek:

- **+ :** összeadás;
- **– :** kivonás;

9. Relációs operátorok (igaz=-1, hamis=0): általában feltételes fordítással kapcsolatban fordulnak elő

- **EQ : =**
- **NE : ≠**
- **LT : <**
- **LE : ≤**
- **GT : >**
- **GE : ≥**

```
// -1 EQ 0FFFFFFFFH   igaz
// -1 NE 0FFFFFFFFH   hamis
33 bites argumentumok!
1 GT -1            igaz
1 GT 0FFFFFFFFH    hamis
```


10. **NOT :** bitenkénti negálás;

11. **AND :** bitenkénti és művelet;

12. Bitenkénti vagy és kizáró vagy művelet:

- **OR :** bitenkénti vagy művelet;
- **XOR :** bitenkénti kizáró vagy művelet;

Feltételes fordítás

A fordító programok általában – így az assembler is – feltételes fordítási lehetőséget biztosít. Ez azt jelenti, hogy a program bizonyos részeit csak abban az esetben fordítja le, ha – a fordítóprogram számára ellenőrizhető – feltétel igaz illetve hamis.

```
IFxx        feltétel
...         ; lefordul, ha a feltétel igaz
ELSE        ; el is maradhat
...         ; lefordul, ha a feltétel hamis
ENDIF
```



```
IF      kifejezés  ; igaz, ha
                   ; kifejezés≠0
IFE     kifejezés  ; igaz, ha
                   ; kifejezés=0
```

Pl.:

```
IF      debug GT 20
        call   debug1
ELSE
        call   debug2
ENDIF
```

```
IF1                 ; igaz a fordítás
                    ; első menetében
IF2                 ; igaz a fordítás
                    ; második menetében

IFDEF  Szimbólum    ; igaz, ha Szimbólum
                    ; definiált
IFNDEF Szimbólum    ; igaz, ha Szimbólum
                    ; nem definiált
```

Pl. Csak akkor definiáljuk **buff**-t, ha a hossza ismert:

```
IFDEF  buff_len
buff   db  buff_len dup (?)
ENDIF
```



```
IFB    <arg>          ; igaz, ha
                      ; arg üres (blank)
IFNB   <arg>          ; igaz, ha
                      ; arg nem üres

IFIDN  <arg1>,<arg2>  ; igaz, ha
                      ; arg1≡arg2 teljesül
IFDIF  <arg1>,<arg2>  ; igaz, ha
              ; arg1≡arg2 nem teljesül
```


Makró és blokk ismétlés

Makró definíció:

```
M_név  MACRO  [fpar1[,fpar2...]]
                      ; makró fej (kezdet)
...                   ; makró törzs
       ENDM           ; makró vége
```

fpar1,fpar2... formális paraméterek vagy egyszerűen paraméterek.

A makró definíció nem lesz része a lefordított programnak, csupán azt határozza meg, hogy később mit kell a makró hívás helyére beírni (makró kifejtés, helyettesítés).

A makró törzsön belül előfordulhat makró hívás és másik makró definíció is.

Makró hívás:

M_név [apar1[,apar2...]]

apar1,apar2... aktuális paraméterek/argumentumok.

A műveleti kód helyére írt **M_név** hatására a korábban megadott definíció szerint megtörténik a makró helyettesítés, más néven makró kifejtés. Ez a makró törzs bemásolását jelenti, miközben az összes paraméter összes előfordulása a megfelelő argumentummal helyettesítődik. A helyettesítés szövegesen történik, azaz minden paraméter – mint szöveg – helyére a megfelelő argumentum – mint szöveg – kerül.

A helyettesítés nem rekurzív.

Makró hívás argumentuma nem lehet makró hívás.

Az argumentumnak megfelelő formális paraméternek lehet olyan előfordulása, amely a későbbiek során makró hívást eredményez.

Dupla szavas összeadás: **(DX:AX)⇐(DX:AX)+(CX:BX)**

Eljárás deklaráció:

```
EDADD   PROC    NEAR
        ADD     AX,BX
        ADC     DX,CX
        RET
EDADD   ENDP
```

Makró definíció:

```
MDADD   MACRO
        ADD     AX,BX
        ADC     DX,CX
        ENDM
```


Ha a programban valahol dupla szavas összeadást kell végezzünk, akkor hívnunk kell az eljárást illetve a makrót:

Eljárás hívás:

```
CALL    EDADD
```

Makró hívás:

```
MDADD
```

Futás közben felhívásra kerül az **EDADD** eljárás

Fordítás közben megtörténik a makró helyettesítés:

```
ADD     AX,BX
ADC     DX,CX
```

Futás közben ez a két utasítás kerül csak végrehajtásra.

Látható, hogy eljárás esetén kettővel több utasítást kell végrehajtanunk, mint makró esetében (**CALL EDADD** és **RET**).

Még nagyobb különbséget tapasztalunk, ha **(CX:BX)** helyett paraméterként kívánjuk megadni az egyik összeadandót:

Eljárás deklaráció:

```
EDADD2 PROC   NEAR
    PUSH    BP
    MOV     BP,SP
    ADD     AX,4[BP]
    ADC     DX,6[BP]
    POP     BP
    RET     4
EDADD  ENDP
```

Eljárás hívás:

Ha **SI** az összeadandónk címét tartalmazza, akkor a felhívás:

```
    PUSH    2[SI]
    PUSH    [SI]
    CALL    EDADD2
```

Futás közben végrehajtásra kerül a paraméter átadás, az eljárás hívás, az eljárás: összesen 9 utasítás

Makró definíció:

```
MDADD2  MACRO   P
IFB     <P>
        ADD     AX,BX
        ADC     DX,CX
ELSE
        ADD     AX,P
        ADC     DX,P+2
ENDIF
        ENDM
```

Makró hívás:

```
    MDADD2  [SI]
```

Fordítás közben a hívás az

```
    ADD     AX,[SI]
    ADC     DX,[SI]+2
```

utasításokra cserélődik, futás közben csak ez a két utasítás kerül végrehajtásra.

```
    MDADD2
```

hatása:

```
        ADD     AX,BX
        ADC     DX,CX
```

Most sem része a makró definíció a lefordított programnak.

A fenti példában rövid volt az eljárás törzs, és ehhez képest viszonylag hosszú volt a paraméter átadás és átvétel. Ilyenkor célszerű a makró alkalmazása.

De ha a program sok helyéről kell meghívnunk egy hosszabb végrehajtandó programrészt, akkor általában célszerűbb eljárást alkalmazni.

Paraméter másutt is előfordulhat a makró törzsben, nemcsak az operandus részen, pl.:

```
PL      macro p1,P2
        mov   ax,p1
        P2    p1
        endm

        PL    Adat, INC
```

hatása:

```
        mov   ax,Adat
        INC   Adat
```


A **&**, **%**, **!** karakterek továbbá a **<>** és **;;** speciális szerepet töltenek be makró kifejtéskor.

& (helyettesítés operátor):

- ha a paraméter – helyettesített – értéke része egy szónak;
- idézeten belüli helyettesítés:

```
errgen      macro  y, x
err&y       db     'Error &y: &x'
            endm

            errgen 5, <Unreadable disk>
```

hatása:

```
err5        db  'Error 5: Unreadable disk'
```

<> (literál szöveg operátor): Ha aktuális paraméter szóközt vagy **,**-t is tartalmaz. Az előző példa **<>** nélkül:

```
        errgen 5, Unreadable disk
```

kifejtve:

```
err5    db      'Error 5: Unreadable'

adat    macro   p
        db      p
        endm

        adat    <'abc',13,10,0>
        adat    'abc',13,10,0
```

kifejtve:

```
        db      'abc',13,10,0
        db      'abc'
```