5.BÖLÜM

5.ASSEMBLER DİREKTİFLERİ

5.1.Data Direktifleri

5.1.1. Sembol Tanımlama

EQU : Bir yazı veya matematiksel ifadeyi bir isme kalıcı olarak atar.

Biçim : isim EQU yazı

İsim EQU matematiksel ifade

= **:** matematiksel bir ifadeyi bir isme atar,fakat isme yeni bir değer atanabilir.

Biçim : isim=matematiksel_ifade

Data direktiflerinde atanan ifade,16 bitlik bir sabit,adres,başka bir sembolik isim veya saklayıcı ismi olabilir.

Örnek.5.1.: K EQU 247

```
TABLO  EQU  DS : [BP] [SI]
HIZ    EQU  ORAN
SAYI   EQU  CX
MAX.HIZ  EQU  2*HIZ
SABİT  = 56
SABİT  = SABİT + 1
```

5.1.2. Data Tanımlama

DB : Değişken tanımlar veya değişkene ilk değer atar. DB, hafızada 1byte yer ayırır.

Biçim : [isim] DB ifade [,…]

DW : DB ile aynı fakat hafızada 2 byte'lık yer ayırır.

DD : DB ile aynı fakat hafızada 4 byte'lık yer ayırır.

Pek çok program, değişkenleri saklamak için hafızada yer ayırır. DB (Define Byte) Byte tanımla,DW (Define Word) word tanımla, DD (Define Double Word), iki word tanımla direktifleri değişkenler için hafızada yer ayırır.

Data tanımlama direktiflerinin genel biçimi: [isim] DB ifade [,….]

[isim] DW ifade [,….]

[isim] DD ifade [,….]

şeklindedir. Burada kullanılan ifade değişkenleri nasıl tanımlamak istediğinize bağlı olarak farklı şekillerde olabilir.

Örnek.5.2: (sabit olabilir)

```
 MAX_SAYI    DB    255

MAX_SAYI    DW    65535

MAX_ISARETLI_SAYI    DB    -128

MAX_ISARETLI_SAYI_1    DW    -32768
```

Örnek.5.3 : (tablo olabilir)

```
B_TABLO    DB     0, 0, 0, 0, 1, 123, 46, -56

W_TABLO   DW    1025, 567, -3100, 300

B1_TABLO  DB  4  DUP(0), 1, 123, 46, -56
```

-Eğer değişkene bir başlangıç değeri vermek istemiyorsak (?) kullanabiliriz.

Örnek.5.4 :

```
MAX_SICAKLIK  DB  ?

MAX_HIZ            DW ?

AYLIK_SATIS      DB  30 DUP (?)
```

-DB direktifi karakter katarını (string) bir ifade olarak kabul eder.

Örnek.5.5 :

```
  MESAJ  DB  'BIR SAYI GIRIN'
```

5.1.3. Segment - Prosedür Tanımlama :

5.1.3.1.Segment: Segmentin sınırlarını tamınlar.Bütün segment tanımlamaları ENDS ile bitmelidir.SEGMENT ve ENDS direktifleri kaynak programları segmentlere ayırır. Bir program dört çeşit segmente (data, kod, ekstra , yığın) ayrılabilir. SEGMENT saklayıcısının 3 adet operantı vardır. Bunlar; Yerleştirme Tipi (Align Type), Birleştirme Tipi (Combine Type) ve sınıf (Class) dır.

Biçim : segment_ismi SEGMENT [Yerleştirme_Tipi] (align_type)

[Birleştirme Tipi] (combine_type)

[Sınıf] (class)

….

……

segment_ismi ENDS

Yerleştirme tipi : Segment hafızaya yüklendiğinde başlangıç sınır değerini belirtir.

BYTE : segmentin hafızada herhangi bir yerden başlayabileceğini belirtir.

WORD : segmentin hafızada çift adres değerinden başlayabileceğini belirtir.

PARA: segmentin hafızada 16 ya bölünebilen adres değerinden başlayabileceğini belirtir.

PAGE : segmentin hafızada 256 ya bölünebilen adres değerinden başlayabileceğini belirtir.

Birleşme Tipi: Segmentin diğer egmentlere nasıl birleştirileeğini gösterir. Data ve extra segmentler PUBLIC veya COMMON olabilir.

PUBLIC : segmentlerin üst üste birleşebileceğini söyler.

COMMON : segmentlerin içiçe yerleştirileceğini söyler.

STACK : Yığın Segmenti (SS) mutlaka STACK tipinde olmalıdır.

Sınıf Tipi : Hafızaya yüklenecek segmentin sırasını beliritir. Aynı sınıf ismine sahip olan segmentler hafızaya birbiri ardına yüklenirken, farklı sınıf ismine sahip olan segmentlerin hafızaya üklenme biçimini linker düzenler.

Örnek.5.6: Data Segment : SEGMENT PARA PUBLIC 'DATA'

Kod Segment : SEGMENT PARA PUBLIC 'CODE'

Ekstra Segment : SEGMENT PARA PUBLIC 'EXTRA'

Yığın Segment : SEGMENT PARA PUBLIC 'STACK'

Örnek.5.7: Bir data segmentin yapısı

```
DSEG  SEGMENT  PARA  PUBLIC  'DATA'
A     DB  ?
B     DB
```

KARE DB 1,2,4,,8,16,32

DSEG ENDS

Örnek.5.8: Bir kod segmentin yapısı

CSEG SEGMENT PARA PUBLIC 'CODE'

CSEG ENDS

ASSUME

SEGMENT ve ENDS direktifleri bir segmentin başlangıç ve sonunu belirtir. Fakat segmentin ne tür bir segment olduğunu belirtmez. Segmentin ne tür bir segment olduğunu belirtmek için ASSUME direktifi kullanılır. ASSUME , Assembler'a hangi segmentin hangi hangi segment saklayıcısına (CS,DS,ES) ait olduğunu söyler. NOTHING direktifi daha önce tanımlanmış ASSUME direktifini iptal eder.

Biçim :

ASSUME seg_reg :segment_ismi [,….]

Seg_reg ; CS,DS,ES,SS olabilir

Segment_ismi ; SEGMENT direktifi ile tanımlanan isimdir.

ASSUME direktifi, Assembler'in tanımlanan etiketleri adres değerine dönüştürmesini sağlar.ASSUME direktifi, kod segmetle SEGMENT tanımlanmasından hemen sonra tanımlanmalıdır.

Örnek.5.9: CSEG SEGMENT PARA PUBLIC 'CODE'

ASSUME CS:CSEG, DS:DSEG

MOV AX, DESG

MOV DS, AX

NOT: Data ve Extra segment adreslerini saklayıcıları ayrıca atamak gerekir. ASSUME komutu bu işlemi yapmaz.

PROC : Assembler'a bir prosedür tanımlandığını belirtir. Bütün PROC tanımlamaları mutlaka ENDP direktifi ile bitmelidir.

Biçim :isim PROC [NEAR]

veya

```
isim  PROC  [FAR]
      …..
.     RET
isim  ENDP
```

Bir prosedürün iki operandı vardır. Bunlar NEAR ve FAR 'dır. Eğer bu operand yazılmazsa, macroassembler bunu NEAR sayar. NEAR olarak tanımlanmış prosedürler sadece tanımlandıkları kod segment saklayıcısı içine çağırılabilir.

Örnek.5.10 :

```
CSEG  SEGMENT  PARA  PUBLIC  'CODE'
          ASSUME  CS:CSEG
ANA     PROC   FAR
          ....
          CALL  ALT_1

RET
   ANA     ENDP
      ALT_1   PROC NEAR
          .....
  RET
 ALT_1      ENDP
CSEG        ENDS
```

FAR olarak tanımlanmış prosedürler başka bir kod segment içinden çağırılabilir.

Örnek.5.11:

```
CSEG   SEGMENT  PARA  PUBLIC  'CODE'
          ASSUME   CS:CSEG1
ALT1   RROC     FAR
```

.....

.... .

RET

ALT_1 ENDP

C SEG_1 ENDS

Mikroişlemci bir prosedürü çağırdığında, geri dönüş adresini yığına atar. Eğer prosedür FAR olarak tanımlanmışsa yığına segment ve offset (CS:IP) adres değerleri atanır. Eğer prosedür NEAR olarak tanımlanmışsa yığına sadece offset (IP) adres değeri atanır. RET komutunda ise yığına atılan bu adres değerleri (prosedürün NEAR veya FAR olarak tanımlandığına bağlı olarak) CS=IP ye geri yüklenir.

Bir prosedürün operandları tanımlanırken aşağıdaki kurallar hatırlanmalıdır.

1-) .EXE türü programlar için ana prosedür FAR olmalıdır.

.COM türü programlar için NEAR olmalıdır.

2-) Eğer bütün kod segmentlerin ismi aynı ise bütün prosedürler , (ana prosedür haricinde) NEAR olmalıdır.

5.1.4. Assembler Kontrol :

END **:** Kaynak programın sonunu gösterir.

Biçim : END [Program_başlangıç_noktası]

EVEN **:** Programın başlangıcının adresinin çift değerde olmasını sağlar.

Biçim : EVEN

ORG **:** İfadenin başlangıç değerinin gösterir. ORG direktifi, komut veya ifadenin hafızada belli bir yerde saklanmasını sağlar. Çoğunlukla .COM tipi programlarda kullanılır.

Biçim : ORG ifade

Örnek.5.12: ORG 100H

END direktifi programın sonunu gösterir. Bundan dolayı her programın sonunda mutlaka END direktifi bulunmalıdır. Buradaki başlangıç noktası DOS'un programı çalıştırmaya başladığı adrestir.

Biçim: END Başlangıç

5.2. Mode Direktifleri

Macro Assembler, Intel 80x86 ailesi işlemcilerinde kullanılmak amacıyla tasarlanmıştır. Intel 80x86 işelmcileri birbirleri ile geriye doğru uyumludurlar fakat 8086 dan sonra her yeni işlemcide yeni komutlar eklenmiştir.Eğer Macro Assembler programında 8086komut kümesinden farklı istemcinin (Örnek 80286, 80386…) komut kümesi kullanılmışsa, bu programın ilk başında tanımlanmalıdır.

Örnek.5.13: .8086 : Tanımlamaya gerek yok

.80286 : 80286 gerçek mod

.80286 P : 80286 korunmuş mod.

.8087 : 8087 matematik işlemci

5.3. Operatörler

Operatörler, operand alanında ve operantı değiştirmek amacıyla kullanılır. 5 çeşit operatör vardır. Bunlar matematiksel,mantıksal,ilişkisel, değer geri döndüren ve öznitelik operatörleridir. Kısaca tablo halinde aşağıda verilmiştir.

5.3.1. Matematiksel :

+ **Biçim** Deger 1 + Deger 2 (Deger 1 ve Deger 2'yi toplar.)

- **Biçim** Deger 1 - Deger 2 (Deger 1 den Deger 2'yi çıkarır.)

***** **Biçim** Deger 1 * Deger 2 (Deger 1 ile Deger 2 çarpılır.)

/ **Biçim** Deger 1 / Deger 2 (Deger 1'i Deger 2'ye böler; bölüm değerini verir.)

MOD **Biçim** Deger 1 MOD Deger 2 (Deger 1'i Deger 2'ye böler. Kalan değerini verir.)

SHL **Biçim** Deger 1 SHL Deger 2 (Değeri, belirtilen ifade kadar bit pozisyonunda sola kaydırır.)

SHR **Biçim** Deger 1 SHR Deger 2 (Değeri, belirtilen ifade kadar bit pozisyonunda sağa kaydırır.)

5.3.2. Mantıksal :

AND **Biçim** Deger 1 AND Deger 2 (Deger 1 ve Deger 2 arasında mantıksal AND işlemi yapar.)

OR **Biçim** Deger 1 OR Deger 2 (Deger 1 ve Deger 2 arasında mantıksal OR işlemi yapar.)

XOR **Biçim** Deger 1 XOR Deger 2 (Deger 1 ve Deger 2 arasında mantıksal XOR işlemi yapar.)

NOT **Biçim** NOT Deger (Deger'in 1'e göre değilini alır.)

5.3.3. İlişkisel :

EQ **Biçim** operand 1 EQ operand 2 (Her iki operand birbirine eşit ise doğru.)

NE **Biçim** operand 1 NE operand 2 (Her iki operand birbirine eşit ise doğru.)

LT **Biçim** operand 1 LT operand 2 (Operand 1 operand 2'den küçük ise doğru.)

GT **Biçim** operand 1 GT operand 2 (Operan 1 operand 2'den büyük ise doğru.)

GE **Biçim** operand 1 GE operand 2 (Operand 1,operand 2'den büyük veya eşit ise doğru.)

LE **Biçim** operand 1 LE operand 2 (operand 1, operand 2 'den küçük veya eşitse doğru.)

5.3.4. Değer Geri Döndüren :

$ **Biçim** **$** **(**O anki adresin değerini geri döndürür.**)**

SEG **Biçim** SEG degisken veya SEG etiket (Degişken veya etiketin segment adres değerini geri döndürür.)

OFFSET **Biçim** OFFSET degisken veya OFFSET etiket **(**Degisken veya etiketin offset adres değerini geri döndürür.)

5.3.5. Öznitelik :

PTR **Biçim** Tip PTR ifade (İfadenin, daha önce tanımlanan özniteliği tip ile tanımlanan yeni özniteliğe dönüştürür. Bu öznitelikler (BYTE, WORD) gibi tip gösteren veya (NEAR, FAR) gibi mesafegösteren operandlar olabilir)

SHORT **Biçim** JMP SHORT etiket (Jmp koomutunun +/- 127 sınırları içinde olması sağlanır.)

Örnekler

Matematiksel Operatörler : Bu operatörler sayısal operandları birleştirir ve sayısal yeni bir sonuç üretir.

Örnek.5.14:

```
PI    EQU    22/7
MASKE   EQU   110010  B
MASKE_SOL_2   EQU   MASK  SHL  2
MASKE_SAG_2   EQU   MASK  SHR  2
```

Mantıksal Operatörler_ : Bu operatörler iki operand arasında mantıksal işlemler yapar.

Örnek.5.15 :

```
DEGER   EQ   10111010 B
DEGER  EQ    00001111 B
MASKE_VE_DEGER   EQU   DEGER   AND   MASKE
```

İlişkisel Operatörler : Bu operatörler iki sayısal değeri veya aynı segmentteki hafıza değerini karşılaştırır. Sonuç eğer ilişki yanlışsa 0000H, doğru ise FFFFH olur.İlişkisel operatörler bazen diğer operatörlerle beraber kullanılır.

Örnek.5.16 :

```
SECIM  EQ  18                    |    SECIM   EQ   25
        MOV  AX,  SECIM  LT  20  |          MOV    AX,  SECIM  LT  20
           ise assembler         |    ise assembler
MOV  AX,  0FFFFH                 |    MOV    AX,  0
```

komutunu işler

Örnek.5.17 : MOV AX, ((SECIM LT 20) AND 5) OR ((SECIM GE 20) AND 6)

Değer Geri Döndüren Operatorler : Bu operatorler değişkenler veya etiketler hakkındaprograma biilgi verir.

Örnek.5.18 :

```
MESAJ          DB   'Herhangi Bir Tusa Basın. '
MESAJ_DEGER   EQU   $ MESAJ
```

Örnek.5.19 :

```
TABLO  DB  100 DUP (0)
MOV  AX,  SEG  TABLO
MOV  BX,  OFFSET  TABLO
```

Öznitelikse Operatörler :

Örnek.5.20 : WORD_TABLO DW 100 DUP (?)

```
ILK_BYTE        EQU  BYTE  PTR  WORD_TABLO
BESİNCİ_BYTE    EQU   ILK_BYTE   + 4
```

5.4.İleri Direktifler

Bu bölüm daha az genel veya ileri programcılar tarafından kullanılan ilave talimatları tanımlar. Tablo.5.1'de bu ileri talimatlar Veri, koşul, listeleme olmak üzere üç grupta listelenmiştir.

Tablo.5.1. İleri Talimatlar:

Tip	Talimatlar		
Veri	Grup	Etiket	
Koşul	ELSE	IFNDEF	IF1
	ENDIF	IFDEF	IF2
	IF	IFE	
	IFDEF	IFIDN	
Listeleme	CREF	%OUT	%XLIST
	LFCOND	SFCOND	
	LIST	XREF	

5.4.1.Data Direktifleri:

Tablo.5.2, assembler'ın ileri data direktiflerini tanımlar.

Tablo.5.2. İleri Data Direktifleri:

Direktif	İşlev
Grup	Biçimi: isim GROUP seg_ismi [.....] Belirtilmiş segmetleri bir isim altında toplar. Böylece bunlar 64K-byte'la fiziksel segmentte konumlanır.
Etiket	Biçimi: isim ETİKET tip İsmin özelliklerini tanımlar

GROUP direktifi segmentlerin topluluğuna bir isim veir. Bu segmentler içindeki segment saklayıcısıyla bir ayarlamayla adreslenebilir yapar. Macro Assembler manual içinde özetlenmiş bu kuralları izlerseniz.Zaten bir 64K fiziksel segment ile saklanmış gruplanmış segmentler elde edersiniz.

Örnek olması için ayrı kod ve data segmentlerde bir COM dosyası yaratarak GROUP kullanabilirsiniz. Bütün datada JMP ile bir segment'ten daha iyidir.

Bir diğer uygulamada, dışsal disk dosyalarında iki prosedür kullanmak istediğiniz varsayılıyor. Sorun şu prosedürlerin her ikiside NEAR bloğu, ancak bunlar farklı isimlerdeki kod segment'ler içeriyorlar.

Kendine özgü olarak PROC1 CODESEG içinde ve data DATASEG içinde, iken PROC2 CODE_SEG içinde; data DATA_SEG içinde NEAR'dan itibaren prosefürler yalnızca aynı isimle çağrılabilirler.Bu size göre şanssızlık gibi görünüyor. Kurtulmak için GROUP.

Sizin programı çağırma; bu yabancı tehlikeye Örnek.5.21'deki gibi bir örnekle ulaşabilirsiniz.

Örnek.5.21 : GROUP'ların örneği:

```
; dışsal kullanılan segmetleri tanımlar.
CODESEG   SEGMET   PARA  PUBLIC 'CODE'
                EXTRN      PROC1  NEAR
CODESEG   ENDS
CODE_SEG  SEGMENT   PARA PUBLIC 'CODE'
                EXTRN  PROC2  NEAR
```

```
CODE_SEG  ENDS
DATASEG  SEGMENT PARA PUBLIC ‘DATA’
DATASEG   ENDS
DATA_SEG  SEGMENT PARA PUBLIC ‘DATA’
DATA_SEG  ENDS
; Kod ve data gruplarını tanımlar.
CGROUP      GROUP  CSEG, SODESEG, CODE_SEG
DGROUP      GROUP DATASEG, DATA_SEG
; Ana program buradan başlar.
CSEG   SEGMENT PARA PUBLIC ‘CODE’
          ASSUME  CS:GROUP, DS:DGROUP, SS:STACK
ENTRY   PROC  NEAR
              ---------
              ---------

MOV  AX, DGROUP ;   DS noktasını GROUP yapar
MOV  DS, AX
--------
--------
CALL   PROC1
CALL   PROC2
--------
--------
RET
ENTRY  ENDP
CSEG  ENDS
          END ENTRY
```

Burada, biz aptal tanımlamaları gruplanmış segmentler olması için ayarlıyoruz. Ve EXTRN'leri görünen kod segmentlerle birlikte koyuyoruz. (Bu yardımlar onları bulan bağlayıcılardır.) sonra iki grup tanımlıyoruz. GCROUP kod segmentler için, DGROUP data segmentler içindir. En sonunda çağırma programı ayarlıyoruz.

Not: Bu ASSUME grup noktaları ayrı ayrı bireysel segmentlerden daha iyidir, ve biz bunu DGROUP'a DS ile yapıyoruz.

MOV DX, OFFSET MESSAGE'daki gibi bu direktiflerde grup kullanmanın bir riski var. Offset kullanıldıktan beri segmentin içindeki offset, segment tutucusundaki tabandan gelmez. Çalışır yapmak için

MOV DX, OFFSET CGROUP : MSG gibi blok girmelisiniz.

LABEL direktifi segmenti, offset'i ve tip özelliğinin ismini tanımlar. LABEL'ı nir FAR özelliğine bir direktif vermek için kullanabilirsiniz. Böylece bu atlmama direktifi başka bir segment'i buna aktarabilir.

Örneğin:

```
HERE    LABEL   FAR
        MOV     DX, 0
```

Etiketler MOV direktifidir. 'HERE'

Zaten sözcük değerlerinin tablosundaki byte'lara ve aktaramazsınız.

Örneğin, 80286 birbirini izleyen B_TABLE isminde byte tablosu ya da W_TABLE isminde sözcük tablosunu tanır.

```
B_TABLE    LABEL  BYTE
W_TABLE    DW     2F24H, 36AH, 0817H,3
```

5.4.2.Şartlı Direktifler:

Şartlı direktifler assembler'a assembler zamanında "true" ya da "false" belirtilmediği tabanda kaynak ifadelerin ayarının derlenmesine ya da atlanmasına neden olur. Bu seçici derleme/derlememe yeteneği deneme çalışmalarında taşhis edilmiş ya da ya da özel durumların yerleştirilmesine ya da çokamaçlı programıların özelleştirilmiş sürümlerinin yaratılmasında size izin verir.

"Şartlı derleme" programın bir parçasıdır. IF direktifinden önce gelir. (bkz Tablo.5.3.) ve ENDIF direktifiyle izler. Her durumda, deneme durumu "true" olarak değerlendirirse kapsanmış kod derlenmiştir.; deneme durumu "false" olarak değerlendirilmişse assembler kodu atlar. Ve ENDIF'ten sonraki ifadeyle devam eder.

Devamında, grupta 4 çift içinde 8 tane IF direktifi kullanabilirsiniz.

IFE – "true" ise açıklaması "0'dır"; IF "true" ise açıklaması "0 değildir".

IF1 assembler 1'i çalıştırmaya geçerken "true" dur. IF2 2'yi çalıştırmaya geçerken "true" dur

IFDEF simge tanımlanmış ya da EXTRN direktifiyle dışsal olrak bildirilmişse "true" dur; aksi halde IFNDEF "true" dur.

IFIDN stringlerde parça-1 ve parça-2 özdeşse "true"dur; farkılı ise IFIDF "true" dur.

Tablo.5.3. Şartlı Direktifler

Direktif	Fonksiyon
IFE	**Biçimi:** IFE ifade İfade "0" sa "true"
IF	**Biçimi:** IF ifade İfade "0" değilse "true"
IF1	**Biçimi:** IF1 Assembler 1 .çalışma geçişinde "true"
IF2	**Biçimi:** IF2 Assembler 2. çalışma geçişinde "true"
IFDEF	**Biçimi:** IFDEF simge Simge tanımlanmış ya da EXTRN direktifiyle dışsal olarak bildirilmişse "true"
IFNDEF	**Biçimi:** IFNDEF simge Simge tanımlanmamış ya da EXTRN direktifiyle dışsal olarak bildirilmemişse "true"

IFIDN	**Biçimi:** IFIDN <string>, <string> string1 ve string2 özdeşse "true". Köşeli parantezler gerekli
IFIDF	**Biçimi:** IFIDNF <string>, <string> string1 ve string2 farklıysa "true". Köşeli parantezler gerekli

Örneğin, Deneme çalışmasında teşhisli yordamlar içermesi için, bunları IFE ve ENIF ile kapsar ve sabit çağrılmış FOR_TEST_ONLY ayarlanır.

Aseembly zamanında, assembler FOR_TEST_ONLY'nin değerini denetler. Eğer 0'sa : teşhisliler programın içinde bulunmaktadır. (kapsar). Eğer FOR_TEST_ONLY başka bir değerdeyse, teşhisliler atlanmıştır.

Program bu formda olacaktır.

```
        IFE     FOR_TEST_ONLY
DIAG1 : ------          (teşhis deneme direktifi)
        -----
        ENDIF
```

İfade yalnızca

FOR_TEST_ONLY = 0 gibiyse direktif DIAG1 ve ENDIF arasında derlenir.

İfade

FOR_TEST_ONLY = 1

Programda daha erken görünür. Assembler direktifleri DIAG1 ve ENDIF arasında atlar.

5.4.2.1.ELSE Seçeneği:

Zaten siz ELSE terimi içererek "false" durumu için değişik direktif ayarı üretebilirsiniz.

Genel Biçimi:

```
IFXX  [argument]  durumu
  ----      (ifade "true" durumu için)
  ----
```

[ELSE] (ifade “false” durumu için)

ENDIF

ELSE seçeneği size izin verir.

Örneğin,bir programın iki sürümünü yapıyorsunuz: Biri ekrana İngilizce mesaj yazıyor diğeri ise Türkçe mesaj yazıyor.Bunu yapmak için sabir LANGUAGE çağrısını ayarlamak, seçip talimatları her iki dil için paylaştırmak gerekiyor. Eğer LANGUAGE değeri 0 ise assembler İngilizce sürüm üretecek; eğer LANGUAGE değeri 1 ise, Türkçe sürüm üretecek.

Program mesaj tabanlı bölümü bu formdaki gibi olur.

```
IFE     LANGUAGE
    ----      (İngilizce-üretme ifadesi)
    ----
ELSE
    ----      (Türkçe-üretme ifadesi)
    ----
ENDIF
```

5.4.2.2.Yuvalama Şartları:

Yuvalama şartı tümcecikleriyle assembler’a ikiden fazla seçenek verebilirsiniz. Örneğin; varsayalım programınızın Türkçe sürümünü gerçekten yakaladınız, ve diğer sürümleri üretmeye karar verdiniz. Ekrana Fransızca ve Almanca mesajlar yazan sürümler. Bunu yapmak için programı değiştirmeliyiz. Böylece Assembler dil tabanlı LANGUAGE 1,2 ya da 3 değerlerini (İngilizce, Türkçe, Fransızca, Almanca) seçer. Şimdi mesaj bölümü bu yapıya sahip olur.

```
IFE     LANGUAGE
    -----         (İngilizce-üretim ifadeleri)
    -----
ELSE
```

```
IFE     LANGUAGE_1
  -----       (Türkçe-üretim ifadeleri)
  -----
ELSE
  IFE   LANGUAGE_2
    -----           (Fansızca-üretim ifadeleri)
    -----
  ELSE
    -----     (Almanca-üretim ifadeleri)
    -----
  ENDIF
ENDIF
ENDIF
```

Not: IFE blokları dengelemek için 3 ayrı ENDIF'e ihtiyacımız var.

Not: Zaten programı kolya okunur yapmak içn IFE – ENDIF çiftlerini girintili hale getirdik.

5.5. Makrolar :

5.5.1.Makrolarla Tanışma:

Bir makro program içinde birkaç kere görünen assembler ifadesi (talimatlar ve direktifler) dizisidir. Prosedürlerde olduğu gibi, makronun da ismi vardır. Bir defa makro. Kaynak progeamınızın talimat dizisinde makronun ismini normal tipte girebilirsiniz.

5.5.2.Makrolar Prosedürlere Karşı:

Her ne kadar makrolar ve prosedürlerin her ikiside kısa, kullanımı kolay talimat dizisini desteklersede, ancak bunlar aynı değildir. Kod prosedür için bir kere meydana gelir. Ve işlemci kodu zorunlu olarak transfer eder. (CALL "code")

Aksine, makro için program içinde birkaç defa meydana gelebilir. Her meydana gelemsinde isim sunumlu talimatlarla makro ile assembler yerini değiştirir. (yani assembler makroyu "genişletir"). Böylelikle; programı çalıştırdığınmızda, prosedürmüş gibi işlemci makro talimatını

“hatta” belleğin başka bir yerine aktarmadan çalıştırır. Bir makro ismi kullanıcı-tanımlı bir assembler direktifidir. Makro assembler komutlarını mikroişlemciden daha iyi sağlar.

Makrolar prosedürlere göre 3 avantaja sahiptir.

Makrolar “dinamiktir”. Giriş parametrelerini değiştirerek kolayca makronun işletimini (yalnızca o an işlettiğini) her zaman değiştirebilirsiniz. Aksine prosedürde bunun için yalnızca verileri değiştirebilirsiniz. Prosedür yapmak daha az esnektir.

Makrolar programları hızlı çalışır yapar. Çünkü işlemci prosedürlerdeki gibi, çağırma ve döngü talimatlarıyla durdurulmuştur.Makrolar programcıların diğer programlarda yaratmış olduklarından çekmek için “makro kütüphanesi”ne girebilirsiniz. (*.mlb, *.lib)

5.5.3.Makrolar Programlamayı Hızlandırır:

Makrolar programlamayı hızlandırabilirlerve gelecekte programı güncelleyeceğinz zaman çalışmayı ayıklar. Makrolar bir kere yarattığınızda programın istedğiniz yerinde kullandığınız zaman programlamayı hızlandırır. Uzun talimat dizileri girmek yerine, yalnızca makro ismini girin, bu isimi makroyu ifade eder. (Makrolar alt prosedürlerle ortak olarak bunlara sahiptirler)

Makrolar ayıklama işini de hızlandırır. Çünkü ayrı olarak her makroyu yaratır ve ayıklarız. Birincisi makro uygun bir şeklide çalışır. Programınızın bu parçası doğru diye endişe etmezsiniz. Herhangi bir hata bulmaya konsantre olabilirsiniz.

Makro içeren programlar genellikle daha kolay okunur ve anlaşılırlar. Bu aynı zamanda daha kolay güncelleneceği anlamına da gelir.

Makroların nasıl çalışma yükünüzü azalttığını görün. Talimatların ekran üzerine karakter göstermek için aldığını düşünün. Bu 21AH=2 seçeneği tipini kapsar. Bir D göstermek için; Örneğin,

```
MOV   AH, 2    ;  karakter gösterme seçeneğini seçer.
MOV   DL, ‘D’  ;  karekteri belirtir.
INT   21H      ;  DOS’un 21h kesmesini çağırır.
```

Benzer şekilde bir E göstermek için gereken

```
MOV   AH, 2    ;  karakter gösterme seçeneğini seçer.
MOV   DL, ‘E’  ;  karakteri belirtir.
INT   21H      ;  DOS’un 21h kesmesini çağırır.
```

Varsayalım ki zaman zaman değişik harfler gösteren bir programınız var. Bölümünüzde ne içerir. Her zaman aynı talimat dizisinde girmeyi (hatırlatmayı) içerir. Ya da alt prosedürün içinde dizi koymayı içerir. Düzgün bir alt prosedürle, her gösterme noktasında iki talimat girebilirsiniz.

Bunlar:

```
MOV    DL, 'E'    ;  Bir E gösterir.
CALL   SHOW_CHAR
```

Bazı durumlarda, zaten bu talimat dizileri kısadır. Bu ihtiyacınız olduğu zaman bunları yeniden girmek hala sıkıntı verir. Bununla birlikte, makro gibi basit dizi tanımlamışsanız. Yerine aşağıdakilerden birini girebilirsiniz.

```
SHOW    D   ;   D gösterir
SHOW    E   ;   E gösterir
SHOW    Y   ;   Y gösterir
```

Hangi teknolojinin kullanılması ve anlaşılması daha kolaydır. Üç talimatlı dizi ya da bir çizgi makro ? En son olarak, zaten makrolar programı değiştirmeyi kolaylaştırıyor. Makro tanımını değiştirir ve assembler kendiliğinden bu yeni sürümü her yerde öneki eskisinin yerine kullanır.

5.5.4.Makroların İçeriği:

Her makro tanımı 3 bölümden oluşur.

Başlık: MAKRO direktifi etiket alanında makronun ismiyledir ve seçimlik olarak operand alanında bir durgun-listedir. Durgun-liste sizin her zaman makroyla çağıracağınız giridi parametrelerini ve değişkenleri belirtir.

Gövde: Assembler talimatları dizisi (talimatlar ve direktifler) makronun ne yapacağını tanımlar.

Sonlandırıcı: ENDM direktifi, makro tanımını bitiren işaret. ENDM'yi atlarsanız, assembler hata mesajı gösterir. "End of file encountered on input file"

Örneğin, Aşağıdaki basit makroylasözcüğün boyu değerini ekleyebilirsiniz.

```
ADD_WORKS MAKRO  TERM1, TERM2, SUM
        MOV   AX, TERM1
        ADD   AX, TERM2
        MOV   SUM, AX
              ENDM
```

Assembler operandlar için saklayıcı isimlerini, bellek bellek bölgesini ve yakın değeri belirtmediğiniz zaman dikkat etmez. (Şüphesiz SUM için, kesin kullanamazsınız.). Uzunca son için geçerlidir.. Assembler yerine koymaları sorunsuz bir şekilde yapar. Örneğin: Programın bir yerinde, aşağıdaki kodu girerek iki bellek bölgesi ekleyebilirsiniz.

```
ADD_WORKS      PRICE,  TAX,  COST
```

Bu kodla assembler programda aşağıdaki talimatları ekler.

```
MOV   AX, PRICE
ADD   AX,  TAX
MOV   COST, AX
```

Bazı yerlerdeyse iki saklayıcı ekleyebilirsniz. Böyle:

```
ADD_WORKS   BX, CX, DX
```

Şimdi de assembler bunları girer.

```
MOV  AX, BX
ADD  AX, CX
MOV  DX, AX
```

NOT: Geçiş parametreleri makrolarda prosedürlerden ne kadar daha kolay. Bir makroyla, yalnız parametreyi girersiniz, bir prosedürle, bunu ya bir saklayıcının içine yada bellek bölgesine koymalısınız.

5.6.Makro Direktifleri:

Tablo.5.4, Microsoft Macro Assembler desteklediği makro direktfilerini özetlemektedir. Bunları biz 4 gruba böldük – Genel amaçlı, döngü, şatrlı ve listeleme olarak...

Tablo.5.4. Makro Direktifleri

Direktif	İşlev
Genel Amaçlı	
MACRO	Biçim: isim MACRO [durgun-liste] ----- -----

	ENDM Assembler tallimat dizisinin ismini belirler. Her MACRO tanımını ENDM yalancı operandıyla bitirmelisiniz.
LOCAL	Biçim: LOCAL durgun-liste Assembler durgun-listenin içine her giriş için tek simge yaratır. Her girişin genişlemesi görüldüünde bunu yerine koyar.
Döngü	
IRP	Biçim: IRP durgun, <argüman listesi> ----- ----- ENDM Assembler her argüman listesi için birkez talimatı IRP ve ENDM arasında döndürür. Her dönmede her blok içindeki durgunun gözükmesi için <argüman-listesi> yeni öğeyi içine koyar.
IRPC	Biçim: IRPC durgun, string ----- ----- ENDM Assembler string'in içindeki karakter için birkez yapıyı IRPC ve ENDM arasında döndürür. Her dönemde her blok içindeki durgun'un gözükmesi için "string"in içine yeni karakteri yerine koyar.
REPT	Biçim: REPT açıklama ----- ----- ENDM Assembler "açıklama" zamanlarıyla talimatı REPT ve ENDM

	arasınd döndürür.
Koşullu:	
EXITM	Biçim: EXITM Koşullu yalancı operandın sonucunda genişleme tabanlı makroyu sonlandırır.
IF1	Biçim: IF1 açıklama ----- ----- ENDIF True ise assembler 1.'yi çalıştırır. Genellikle kaynak programı içinde INCLUDE bir makro kütüphane dosyasıyla kullanılır.
IFB	Biçim: IFB <argüman> ---- ---- ENDIF True ise argüman boş. Köşeli parantezler gerekli.
IFNB	Biçim: IFNB <argüman> ----- ----- ENDIF True ise argüman boş değil. Köşeli parantezler gerekli
Listeleme	
.LALL	Biçimi: .LALL

	Bütün genişlemeler içi (yorumlarıyla birlikte) bütün makro metini listeler.
.SALL	Biçimi: .SALL Makro metnini listelemelerden çıkarır.
.XALL	Biçimi: .XALL Makro çizgilerini yalnız ürettiği nesne koduyla listeler. Bu varsayılan ayardır.

5.7.Genel-Amaçlı Direktifler:

Biz zaten MACRO direktiflerini tartışmıştık. Bu makronun ismini verir, makronun kullandığı operand isimlerini listeler.

5.7.1.LOCAL Direktifi:

Eğer sizing makronuz etiketli talimatlar ya da direktifler içeriyorsa assembler'a her zaman etiketleri değiştirmesini söylemelisiniz. Bu makroyu genişletir. Aksi halde, "Symbol is Multi-Defined" hatasıyla karşılaşırsınız. LOCAL direktifi assembler'a her zaman hangi etiketi değiştireceğini söyler.

Örneğin, Aşağıdaki makro'da (WAIT) makro işlemciyi COUNT değerine kadar bekletir. Taki 0'a düşene kadar.

"NEXT" LOCAL etiketi program içinde birkezden fazla kullanmamıza izin verir.

```
WAIT    MACRO   COUNT
            LOCAL   NEXT
            PUSH    CX
            MOV     CX, COUNT ;  Güncel CX'i sakla
NEXT:       LOOP    NEXT
              POP     CX
        ENDM
```

Not: LOCAL kesinlikle MACRO talimatı takip eder. Eğer LOCAL kullanılmışsa makroda ilk yapı olmalıdır.Onlar herşeyden önce gelmelidir.

Bir LOCAL etiketi bildirirken zaten diğer makroların altında gelir. Assembler etikete her zaman yeni iç ad verir. Bu makroyu genişletir; yalnız kopyalama yoktur.

5.7.2.Döngü Direktifleri:

REPT, IRP ve IRPC direktifler assembler'a makro içine assembler talimat dizilerinin dönmesini sağlar.

REPT operand alanındaki alanındaki açıklamayla dönme sayacı oluşturur. Örneğin: aşağıdaki makro LNGTH byte'ları bellekte ayırır. Ve LNGTH arasında 1'le olanları başa getirir. (tek tek)

```
ALLOCATE  MACRO  TLABEL, LNGTH
  TLABEL      EQU      THIS  BYTE
   VALUE  =  0
       REPT    LNGTH  VALUE  = VALUE  + 1
                DB   VALUE
       ENDM
            ENDM
```

Not: Burada iki ENDM'ye ihtiyaç duyduk. İlki REPT'in sonu, ikincisi MACRO tanımının sonuydu.

ALLOCATE tanımladıktan sonra, bunu bir 40-byte'lık tablo ayarlamak için kullandık. TABLE1 isminde, bu diziyle

```
 DATA    SEGMENT   PARA  'DATA'
 ALLOCATE   TABLE1, 40
 DATA    ENDS
```

İkinci döngü direktifi, IRP her dönmeyle liste argümanı durun simge için yerine koymanıza izin verir. Örneğin bu dizi:

```
IRP   VALUE,   <1,2,3,5,7,11,13,17,19,23>
      DW              VALUE   * VALUE * VALUE
ENDM
```

İlk on baş numaralarının küpünü içeren tablolardır.

IPRC, IRP gibidir, ama argümanları olan string değişkenler numaralardan daha iyidir. IPRC iki operand alır. Bir durgun simge ve bir string ve string içinde her karakter için blok içindeki talimatları birkez döndürür. Her dönmede blok içinde her durgun'un görünmesi için string içine bir sonraki karakter koyulur.

Örneğin, Bu dizi

```
IRPC    CHAR,   0 1 2 3 4 5 6 7 8 9

        DB    CHAR

 ENDM
```

0 ile 9 arasında basamak içeren ASCII kodlar içeren 10-byte text bir string ayarlar.

5.7.3.Şartlı Direktifler:

Assembler hangi şartın tatmin edip etmediğini test eder. Assembler taimat IF ve ENDIF arasındaysa derler değilse atlar.

5.7.4.IF1 Direktifi:

IF1 direktifi kaynak programdan makro kütüphanesi okunduğunda kullanılır.

5.7.5.IFB Direktifi:

Makronun istediğinden daha az parametre girdiğinizde, assembler normal olarak atlanmış veya "boş" parametreleri sıfıra ayarlar. Bununla birlikte, IFB (if blank) boş parametreler için bazı değişik eylem yolları belirtmenize izin verir. IFB genel olarak bazı gerekli parametreler kaçırıldığında makroyu erken sonlandırmada kullanılır. Biz bu bölümde EXITM 'yi erken sonlandırma hakkında daha çok şey söyleyeceğiz.

5.7.6.IFNB Direktifi:

Assembler IFNB (if not blank) ile karşlaştığında yalnızca kullanızının parametreler için bir değer verdiği talimatları derler., aksi halde atlar.

Örneğin, Bir makro bu form kullanılarak çağrılan dışlanmış olabilecek bir ismi okur.

```
    GET_NAME   FIRST_NAME,  MIDDLE_INITIAL,  LAST_NAME
```

GET_NAME için makro tanımı ilk ismi oluşturan talimatı içerir, sonra ortadaki, sonra son ismi içerir. Bununla birlikte herkesin orta ismi kullanmamaması yüzünden atlama hükmü olabilir.

Bunu IFNB kullanarak yapabilirsiniz. Orta isim talimatları yalnız kullanı girerse işler. Böylece GET_NAME'in tanımı izleyen genel form'daki gibidir.

```
GET_,NAME   MACRO  ilk_isim, orta_isim, son_,isim
                   -----       (Bu talimatlar ilk ismi okur)
                   -----
IFNB  <middle_initial>
                   -----       (Bu talimatlar orta ismi okur)
         ENDIF
                   -----       (Bu talimatlar son ismi okur)
                   -----
                 ENDM
```

Bu sayede siz bu makrodaki gibi "GET_NAME" Mehmet Tektaş girebilirsiniz ve assembler hata vermez.IFNB operand kayıplarından oluşabilecek kayıplara karşı assembler'yi korumaya yardım eder. Eğer makro PUSH reg_name talimatı içerirse reg_name parametre listesinden atlanır.Assembler PUSH 0 ile girintili değer üretir. Buna karşı korunmak için

```
          IFNB  <reg_name>
           PUSH  reg_name
       ENDIF    kullanın
```

(Bu durumda, muhtemelen benzer bir konuma POP reg_name eşlemesine de gerek duyulur.)

5.7.7.EXITM Direktifi:

EXITM (Exit Macro) direktifi assembler'ın genişlemiş makroyu erken durdurmasına neden olur. Koşullu direktifin sonuucu tabanlı böyledir.

```
 IFB  <name>
   EXITM
 ENDIF
```

Örneğin, REPT direktifiyle tanımladığımız ALLOCATE makrosunu yeniden çağıralım. Aşağıdaki yeni tanımlama assembler'ı tabloyu yalnız LNGHT parametresi 50'nin altında tahsis eder yapar.

```
ALL_LT_50     MACRO   LNGTH
          VALUE = 0
          IF  LNGTH  GE 50
              EXITM
          ENDIF
          REPT  LNGTH
              VALUE = VALUE + 1
              DB   VALUE
          ENDM
              ENDM
```

5.7.8.Listeleme Direktifleri:

.LALL, .SALL ve .XALL direktfileri.“Assembler listelemesinde (LST)” Makro text’in içerdiği makroların sayısını kontrol eder. Bu seçeneği atlarsanız assembler .XALL istediğinizi varsayar. Bu ise yalnızca üretilen nesne kodunu ve yorumu ve bellek bölgesine geri dönmeyen direktifleri listeler.

.LALL direktifi tam listeleme üretir. Yorumları da içerir. .SALL listelemeden bütün makro text’leri atlar. .LALL kullanırsanız sizin dosyanız için programın bir kopyasını üretir. .SALL kullandığınızda tamamen ayıklanmış makrolar listelenenecektir.

5.8.Makro Operatörleri:

Makro assembler kullanabileceğiniz 4 operatörü destekler. Tablo.5.5.’e bakınız...

Tablo .5.5. Makro Operatörleri:

Operatör	İşlev
&	Biçim: text & text Text ya da simgeleri birleştirir.
;;	Biçim: ;; yorum Listelemeden ve makro genişlemesinden yorumu atlar. :LALL ile aynıdır.

!	Biçim: ! karakter Argümanda kullanıldığında assembler karakteri durgun değer gibi kullanır, simgeden daha iyidir.
%	Biçim: % simge Sembolü sayıya çevirir. Assembler makroyu genişlettiğinde simge için sayı yerine konur.

5.8.1. & Operatörü:

& operatörü size özel etiketler ya da operandlar yaratmanıza izin verir.

Örneğin, Aşağıdaki makro isim ve uzunluğu tanımlanmış bir byte tablosu ayarlar.

```
DEF_TABLE   MACRO   SUFFIX, LNGTH
        TABLE & SUFFIX  DB  LNGTH  DUP (?)
            ENDM
```

Programda

DEF_TABLE ‘ a A,5 girerseniz assember buna çevirir.

TABLE DB 5 DUP (?)

5.8.2. ;; Operatörü:

;; operatörü assembler’a makro genişlemesinden yorumları atlatır. Yorumsuz sizin son programınız bellekten daha az yer alır ve böylece daha hızlı derlenir. Makro tanımlarken, düzenli; kullanın yalnızca yorumların kesinlikle gerekli olduğu durmlarda kullanın.

5.8.3.Kaynak Programın İçinde Makro Tanımlamak:

Makro kullanmanın iki yolu vardır. Ya doğrudan tanımlamaları program içine girersiniz. Ya da birkaç makro kütüphanesinden okutursunuz. Bu bölümde biz program içine nasıl makro gireceğinizi tanımlayacağız.

Özel amaçlı makronuz varsa yalnız bir programın içinde tanımlayabilirsiniz. Sonra çağırma gereklidir. Varsayalım; örneğin SHOW makrosu kullanmak istiyorsanız. SHOW tanımı böyledir.

```
SHOW  MACRO   karakter
;;   Tanımlanmış karakteri gösterir.
        PUSH  AX    ;;  etiketlenmiş tutucularını saklar
        PUSH  DX
        MOV   AH, 2
        MOV   DL, 'karakter'
        INT     21H
        POP    DX
        POP    AX
 ENDM
```

Bu materyali programın başına girin. Kesinlikle TITLE talimatından sonra ...

Programın içine doğrudan makro girmenin dexavantajı program parçasının sınırlarıdır. Diğer programlarda kullanmak içiin makro kütüphanesinin içine koyabilirsiniz.

5.9.Makro Kütüphaneleri:

Bir makro kütüphanesi birkaç programda ihtiyaç duyacağınız. Tanımlamalar içeren bir disk dosyasıdır. İlk kez bu dosyayı yarattığınızda birkaç kaynak programın içinde bunu okutabilirsiniz. Aynı isimle başka yerlerde de kullanabilirsiniz.

5.9.1.Makro Kütüphanesi Oluşturma:

Makro kütüphanenelerini EDLIN ya da herhangi bir kelime işlemciyle yaratabilirsiniz.

5.9.1.1.Makro Tanımlama İçin Rehber:

Makroları bir kütüphane içinde kullanabilmeniz için sanal olarak programın içinde genel amaçlı rutünler olabilir. Böylelikle değişik görevler için bunları tasarlayabilirsiniz. Ama programda bunları kullanırken çakışma olmamalı. Verimli makrolar tanımlamak için tüyolar:

Belge makroları aşağı yukarı mümkündür. Birçok yorum içerir. Hatırlayın; makro tanımlamalarınız onu kullananlar için anlamlı olmalıydı, yalnızca sizin için değil.

Yorum girmek için ;; operatörünü kullanın ve alanları tab tuşuyla ayırın. (boşluk tuşundan daha iyidir). Bu programınızın boyutunu en azda tutmada size yardımcı olur, daha hızlı derlemenize izin verir.

Makroları genel olarak saklamanız mümkündür. Özel-amaçlı makroya gerek duyarsanız; bir başka genel amaşçlı makro kullanırsınız. (eğer mümkünse)

Örneğin, varsayalım LOCATE isimli bir makronuz var. Bu imleci tanımlanmış sıra ve sütuna konumlandırıyor. Bir çağrıda LOCATE formunu kullanır. Sıra-sütun sonra siz HOME isminde bir başka makro tanımlamak istediğinizde imleci üzt sol köşeye taşır. HOME'un tanımı basitçe LOCATE 0,0'dır.

Makro label içeriyorsa, LOCAL talimatının içine yazın.

Her saklayıcısı saklamak için dış saklayıcıları dışlayan makrolar kullanın. Bunu yapmak yararlıdır. PUSH'larla makroyu başlatın POP'larla sonlandırın.

Makro tanımı önceden tanımlı makroyla gerçeklenmiş bir görevi gerçekleştiriyorsa, bunu yapmak için bir makroyu çağırın.

5.9.1.2.Bir Program İçine Makro Kütüphanesi Okutmak:

Bir kaynak programını içine bir makro kütüphanesi okutmak için assembler'a INCLUE talimatıyla bir isim vermelisiniz. Bununla birlikte bunu sadece böyle yaparsınız.

```
INCLUDE   MACRO.LIB
```

Dönmeden kaçınmak için INCLUE'u bir IF1 şartlı talimatı içine koyarsınız.

```
IF1
  INCLUDE   MACRO.LIB
ENDIF
```

5.9.1.3.Makroları Temizlemek (Arındırmak):

INCLUDE'ları assembler kullanıldığında makro kütüphanesi kütüphanesi kullanımının bir dejavantajı; bütün okunan makrolar belleğin assembler'ın çalışma boşluğunda saklanır. Bu içerik makroları assembler tarafından lkullanılmayacaktır. Böylece çalışma boşluğu muhtemelen "out of memory" hata durumu yaratacaktır. Bu sorundan sakınmakiçin gereksiz makroları arındırabilirsiniz. Bunu yapmak için INCLUDE'dan sonra kesinlikle arındırma öğeleri ekleyin. Basitçe:

```
INCLUDE   MACRO.LIB
PURGE   SHOW, CLS, HOME, LOCATE olur.
```

5.10.Nesne Kütüphaneleri:

Daha önce makro kütüphanelerinin nasıl yaratılacağından bahsettik. Disk dosyaları makro tanımlamaları içindedir. Bir kütüphanede birkaç makro kullanmak basitçe programınızın içine INCLUDE ile tüm kitaplığı alabilirsiniz. (Ve makronun ismi). Böylece makro kütüphaneleri her programda gerektiği yerde sizi yeniden yazmaktan kurtarır.

Makro Assembler program modülleri için banzer olanakları destekler. Özellikle nesne kütüphaneleri yaratmanıza izin verir. Nesne kütüphanesi içinde birkaç prosedür kullnamak için prosedürü CALL ile çağırıp dışsal olarak (EXTRN direktifiyle) bildirebilirsiniz. Sonra link olayını çağırdığınızda kütüphaneyi program modülüyle link edersiniz. Link edici kendiliğinden çağrılmış prosedürü açacaktır.

Bir nesne kütüphanesinin diske bir prosedür olarak kaydedileceğini hatırlamalısınız. Ardından kaçınılmaz disk-sorunlu iş gelebilir. Şu anda nesne kütüphanesi içeren birini yalnızca bir diskin Track'ine saklamalısınız. Bir nesne kütüphanesi bir hard diske sahipseniz uygundur. Çünkü Link komutuyla bir bireysel modülü belirtmek için kaydeder. Basitçe link kütüphaneye programı çağırır ve link aracı gereken modülü açar.

5.10.1.Nesne Kütüphaneleri İnşa Etmek:

Assembler diskinde nesne kütüphanelerini tutan program LIB.EXE (Kütüphane Yöneticisi) dir. Bir nesne kütüphanesi yaratmak için LIB yazmalı ve kütüphanenin içine komak istediğiniz. İlk nesne modülünü belirtmelisiniz. Bunu yapmak için assembler diskini A sürücüsüne , nesne modülünü içeren diski B sürücüsüne koumalısınız. Sonra bilgisayarı uygun bir şekilde açın. Ekranda B> konumuna geçin. Sonra bu komutu girin.

```
a : lib   kutismi  +  nesnemodulu ;
```

LIB.EXE kendiliğinden LIB ve OBJ uzantılarını kütüphane ve nesne modülü ismi için destekler.

Örneğin: İlk girişte nesne.lib ile siralama.obj isimli kütüphane yaratmak için şunu girin;

```
a :  lib nesne  + siralama ;
```

5.10.2.Nesne Kütüphanelerini İşletmek:

İlk kez bir kütüphane yarattınız. Önceki komutu tekrarlayarak yeni modüller ekleyebilirsiniz. Örneğin NESNE adlı kütüphaneye MULU32.OBJ modülü eklemek için şunu girin

```
a :  lib   nesne  +  mulu32 ;
```

(+) kütüphane ekle anlamındadır. Kütüphaneden bir modül silmek için (+) yerine (-) kullanmalısınız. Örneğin MULU32'yi silmek için

```
a :  lib   nesne  -  mulu32 ;
```

Siz ayrıca kütüphaneden bir modülü kopyalamak isteyebilirsiniz. Belkide onu başka bir kütüphaneye eklemek için. Bu * operatörünü gerektirir. Örneğin siralama.obj 'nin bir kopyasını yapmak için.

a : lib nesne * siralama ;

bundan sonra SIRALAMA hala kütüphanenin içindedir. Ancak siralama.obj adında bir kopyası daha diskin içindedir.

LIB.EXE tamel işlemleri birleştirmenize izin verir. "-+" operatörü bir kütüphane modülünü başak bir tanesinin içeriğiyle değiştirir. Yani "-+" bir kütüphaneyi günceller.

Örneğin:

a : lib nesne -+ siralama ;

NESNE kütüphanesinden SIRALAMA modülü kaldırır. Diskten SIRALAMA.OBJ 'nin içeriğini içine koyar.

Benzer bir şekilde "-*" kütüphaneden bir nesne modülü kaldırır. Ancak diske bir kopyasını yaratır. Örneğin:

a : lib nesne -* siralama ;

NESNE kkütüphanesinden SIRALAMA modülünü SIRALAMA isimli yeni bir dosyaya tşır.

5.10.4..Bir Kitaplığın Bir Dizinini Elde Etme:

Kütüphane içindeki nesne modüllerinin ne ne olduklarını unuttuysanız LIB.EXE'yi dizin dosyası üretmek için kullanabilirsiniz. Bunu yapmak için bu bu komutu giriniz.

a : lib kutismi , kutüsmü.dir ;

sonra, dizini görüntülemek için ;

type kutismi.dir

nesne modüllünün isminin yanında dizin listeleri simgeleri herbiri içinde PUBLIC olarak bildirilir. Bu kütüphane içinde programı çağırırken hangi prosedürlere, değişkenlere ve eşitliklere başvuracağınızı söyler.

5.10.5.Nesne Kütüphanelerini Kullanma :

Bu bölümün başında ifade ettğimiz gibi. Bir nesne kütüphanesi kullanmak için nesne modülü (ya da modülleri) içeren programınızı "link" etmelisiniz. Link aracı bu amaç için bir özel ileti gösterebilir.

a : link modül , , NUL

Ekran gösterdiğinde ;

Libraries [.LIB] :

kütüphanenizin ismini girin enter'a basın

Örneğin . NESNE kütüphanesini link etnek için; bu nesne modülü MAIN_PROG tarafından çağrılıyor olsun

a : link main_prog , , NUL girin

sonra

Libraries [.ILB] : nesne

Eğer program birkaç farklı kütüphaneden nesne modülü gerektiriyorsa Libraries iletisine bunları listeleyerek yanıt vermelisiniz.

Libraries [.LIB] : lib1 + lib2 + lib3