Bilgisayar Mühendisliğine Giriş – BLM 101

Hafta 12: Assembly Dili

Fenerbahçe Üniversitesi

12. Hafta İçeriği

- Assembly Dili ile Programlama
- Komutlar
- Assembly İşlemleri
- Makine Kodu Üretme
- Çalıştırılabilir Dosya Yaratma
- Birden Fazla Obje Dosyası ile Tasarım

Okunabilir Makine Dili

- Bilgisayarlar / elektronik devreler 1 ve 0'lar ile çalışırlar

```
0001110010000110
```

- Okunabilir kodlar…

```
ADD R6,R2,R6  ; R6 = R2 + R6
```

- Assembler yazılımı, okunabilir Assembly dilinde yazılmış olan yazılımı, 1 ve 0'lara makine diline dönüştürür.
  - Bu işlemi, makine diline dönüştürülecek olan işlemciye göre yapmaktadır.

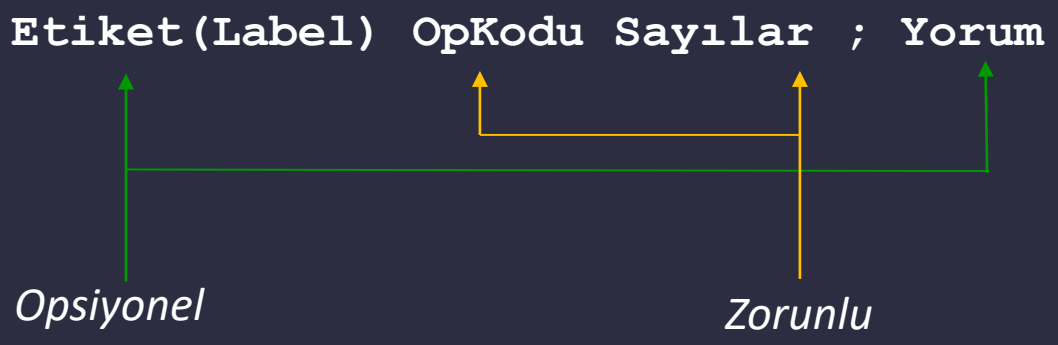
Assembly Dili Örneği

```
; 6 ile sayıyı çarpan uygulama

.ORIG   x3050
LD      R1, SIX
LD      R2, NUMBER
AND     R3, R3, #0          ; R3 = 0


XYZ     ADD     R3, R3, R2
        ADD     R1, R1, #-1   ; R1 döngüyü ayarlamaktadır.
        BRp     XYZ

        HALT

NUMBER  .BLKW   1
SIX     .FILL   x0006

.END
```

LC-3 Assembly Dili Sözdizimi (Syntax)

- Programdaki satırlar aşağıdaki gibi olabilir:
  - Komut
  - Assembler yazılımı için talimatlar
  - Yorum satırları
- Semboller arasındaki boşlukların önemi yoktur.
- Yorum satırlar (“;” ile başlayanlar) programa dahil olmazlar.

- Komutların formatı aşağıdaki gibidir



Operasyon Kodları ve İşleme Giren Sayılar

- Operasyon Kodları (OpCode)
  - Örn. `ADD, AND, LD, LDR`, …

- İşleme Giren Sayılar (Operands)
  - Saklayıcılar – Rn ile isimlendirilirler, n 0-7 arasındadır.
  - Sayılar-- #(onluk tabanda) veya x(16'lık tabanda) ile ifade edilmektedirler.
  - Etiket (Label) – bir bellekteki yerin sembolik ismi

  - Virgül işareti ile ayrılmışlardır
  - Örnek format

```
ADD R1,R1,R3
ADD R1,R1,#3
LD  R6,NUMBER
BRz LOOP
```

Etiketler (Labels) ve Yorumlar (Comments)

- Etiket
  - Komutun başında bulunur.
  - Bulunduğu satırda, adresi tutan sembolik bir isim atanır.
    - Örnek:

```
LOOP  ADD  R1,R1,#-1
      BRp  LOOP
```

- Yorumlar
  - Noktalı virgül işaretinden sonra gelen her şey yorumdur.
  - Assembler tarafından atılırlar (Kullanılmazlar).
  - Programın anlaşılması için kullanılırlar.

Assembler Talimatları (Directives)

- İşlemci tarafından çalıştırılmazlar. Assembler yazılımı bu talimatlara bakarak, assembly diline dönüştürürken çeşitli işlemler yapar
  - Komut gibi gözükmektedirler ancak değildirler.
  - Başlarında . işareti vardır.

Operasyon Kodu	*İşleme Girecek Sayılar*	*Açıklama*
`.ORIG`	**adres**	**Programın başlangıç adresini belirtir**
`.END`		**Programın sonu**
`.BLKW`	**n**	**N adet adresi rezerve eder**
`.FILL`	**n**	**1 adresi rezerve edip, içine n sayısını yazar**
`.STRINGZ`	**N karakterli metin**	**N+1 alan rezerve edip, n karakterli metni yerleştirir, son n+1. alana ise '\0' sonlandırma karakterini yerleştirir**

Trap Kodları

- LC-3 assembler, kullanılan HALT, IN, OUT gibi… komutlar için arka planda TRAP kodu yerleştirmektedir.

Kod	*Karşılığı*	*Açıklama*
`HALT`	`TRAP x25`	**Uygulamayı bitirir.**
`IN`	`TRAP x23`	**Klavyeden alınan bir karakteri R0[7:0] saklayıcısına kaydeder ve ekrana bastırır.**
`OUT`	`TRAP x21`	**Ekrana R0 saklayıcısındaki değeri yazdırır.**
`GETC`	`TRAP x20`	**Klavyeden alınan bir karakteri R0 saklayışına kaydeder.**
`PUTS`	`TRAP x22`	**Konsola başlangıç adresi R0 olan metni yazdırır. Metnin sonunda '\0' karakteri bulunmalıdır.**

Kodlama Prensipleri

- Kodunuzun okunabilirliğinin yüksek olması için aşağıdaki maddelere uygun yazılmalıdır.

1. Programın başına yorum satırı olarak
    1. Programı yazan kişi
    2. Yazılma tarihi ve güncellemeler
    3. Programın amacı

    yazılmalıdır.

2. Kullanılan saklayıcıların görevlerini yorum satırı olarak yanına yazınız.

Kodlama Prensipleri

3. Kullanılan komutların yanına gerekliyse yorum satırı ekleyiniz.

4. Kullanılan etiket bilgileri anlaşılır olmalıdır.
   - Büyük ve küçük harflerin bir arada kullanımı okumayı kolaylaştırır.
   - ASCIIdenBinaryDonusum, KaydetR1, DonguBasi gib…

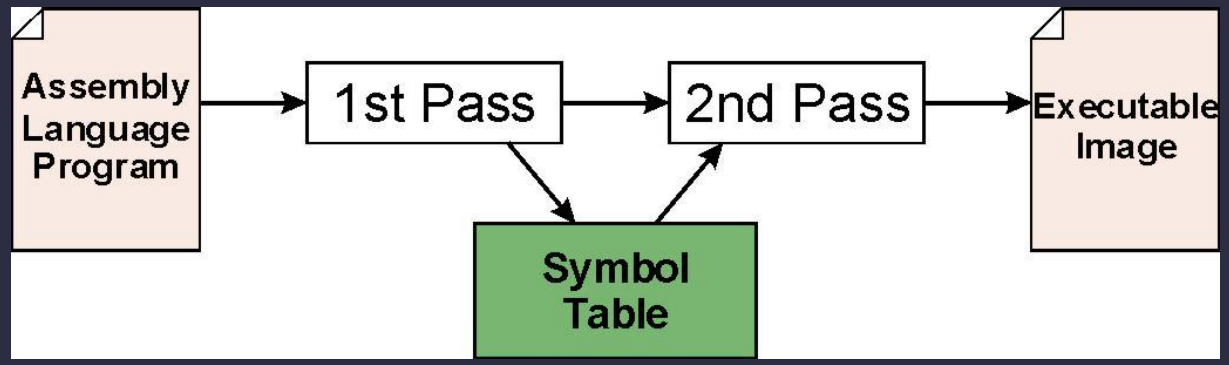
5. Uzun ifadelerde (çok fazla koşulun kontrolü gibi) bölünerek ifade edilmelidir.

LC3 Örnek Assembly Program

```
.ORIG x3000
;---------------------------------------
; Örnek: Döngü
;---------------------------------------
LD R1, LOOPMAX; R1 = LOOPMAX
LD R2, NEG_ONE; R2 = NEG_ONE
LEA R0, LOOP_MSG; LOOP_MSG dizisinin başlangıç adresini yükler

DONGU_BASLAT
PUTS; Ekrana R0'daki adresten başlayarak metni yazdırır.
ADD R1, R2, R1; R1 = R1 + R2, R1--
BRp DONGU_BASLAT; R1 halen pozitif ise dongu devam edecek
LOOP_END; Değilse programın sonuna gidecek, bu etiket bir iş yapmamaktadır

LEA R0, DONE_MSG; Tamamlandi mesaji bastıracak, R0'a yüklenir
PUTS; Ekrana R0'ın adresinden başlayarak metni yazdirir

HALT

LOOPMAX .FILL x000A ; Hex olarak 10 sayısı
NEG_ONE .FILL #-1 ; -1 sayısı
LOOP_MSG .STRINGZ "Dongu...\n"
DONE_MSG .STRINGZ "Tamamlandi!\n"
.END
```

Assembly Süreci

- Assembly (.asm) kaynak kodunu, LC-3 simülatörünün çalıştırabileceği obje (.obj) koduna dönüştürür.



- İlk Aşama:
  - Programı baştan sona tarar
  - Tüm etiketleri (label)'leri bulur, karşılık gelecek adresleri atar. Bu adreslerin belirlendiği alana sembol tablosu (Symbol Table) denir.
- İkinci Aşama:
  - Sembol tablosundaki bilgileri kullanarak Assembly dilindeki yazılmış olan komutları, makine diline çevirir.

İlk Aşama: Sembol Tablosunu Oluşturma

1. `.ORIG` ifadesini bul,
   ilk çalıştırılacak komutun adresini ifade etmektedir.
   - Lokasyon Sayacı (Location Counter), kodun içindeki komutların hangi adrese atanacağını saklar.

2. Boş olmayan her satır için:
   a) Eğer etiket (label) var ise, sembol tablosuna etiket ve lokasyonu eklenir.
   b) Lokasyon sayacını arttır.

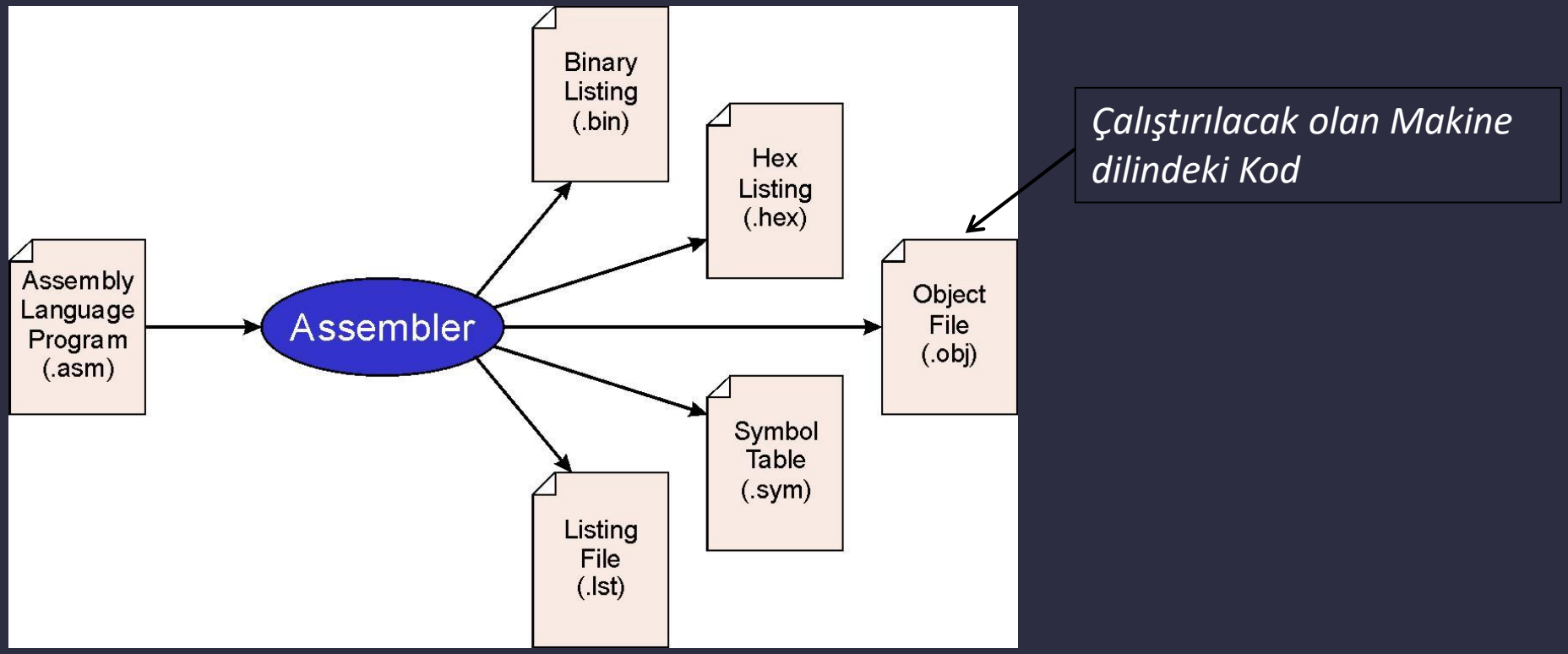
3. `.END` ifadesini gördüğünde işlemi durdur.

İkinci Geçiş: Makine Dili Üretimi

- Her bir assembly ifadesi için karşılık gelen makine dili komutu üretilmektedir.
  - İşleme giren sayılardan birisi etiket (label) olduğunda sembol tablosundan bakılarak adresi yazılmaktadır.

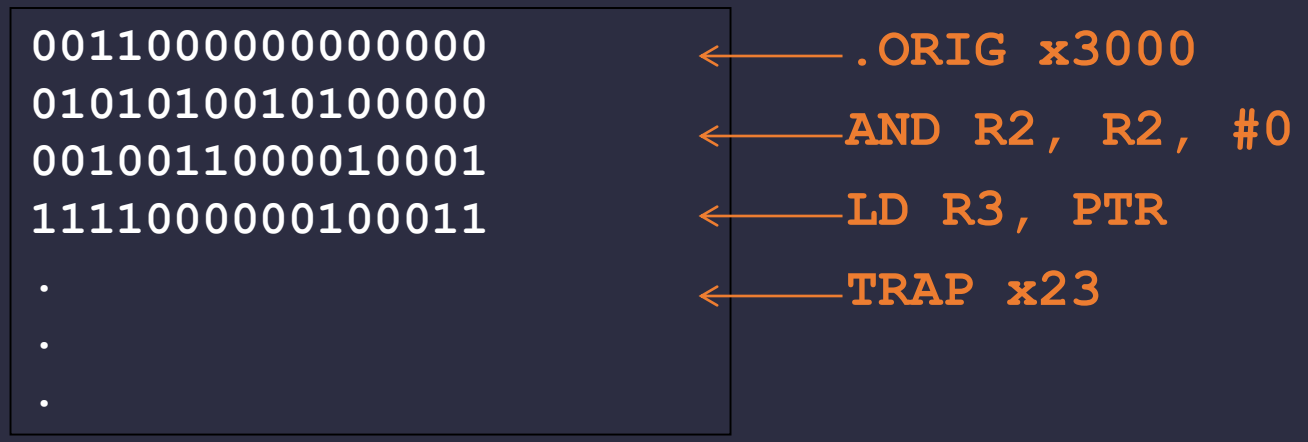
LC-3 Assembler

- Assembler, makine diline dönüşümü obje dosyasına yazdırarak yapar.



Obje Dosya Formatı

- LC-3 obje dosyası içeriği
  - Programın başlangıç adresi
  - Komutlar
- Örnek
  - "Karakter sayma" uygulamasının başlangıç kodu



Birden Çok Obje Kodu

- Bir obje dosyasında tüm uygulama olmak zorunda değildir.
  - Sistem kütüphaneleri
  - Birden çok geliştiricinin yazdığı çeşitli kod parçacıkları

Bağlama (Linking) ve Yükleme (Loading)

- *Bağlama işlemi, birbirinden bağımsız olan obje kodlarındaki sembollerin birbirleri tarafından kullanılmasını sağlar*.
  - `.EXTERNAL` notasyonu ile sembolün bir diğer obje kodunda olduğunu ifade etmektedir.

- *Yükleme işlemi, çalıştırılabilir bir kod parçacığının belleğe yüklenmesidir.*
  - Gelişmiş yükleyiciler, bellekte yeterince blok halinde yer olmadığında, uygun yerlere bölerek yerleştirebilir.
  - Atlama, yükleme, kaydetme adreslerini yeniden güncellenmesi gerekir.