A red text on a white background

Description automatically generated

**İSTANBUL SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ**

**MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**YAZ302 – SİSTEM PROGRAMLAMA**

**FİNAL PROJESİ**

***Design and Implement A 2 Pass Assembler Software***

**HAZIRLAYANLAR**

|  |
| --- |
| MERT TOSUN – 210601027 |
| BEGÜM AKYOL – 210601013 |
| ***TESLİM TARİHİ***  ***MAYIS-2024*** |

***İÇİNDEKİLER***

***1. GİRİŞ***

* ***Assembler Nedir?***
* ***Tek Geçişli ve Çok Geçişli Assembler Nedir?***
* ***Assembler Çalışma Aşamaları Nelerdir?***
* ***Bu Aşamaların Önemi Nelerdir?***
* ***İki Geçişli Assembler’da Örnek Kod İncelemesi***

***2. PROJENİN GEREKLİLİKLERİ***

***3. KODUN GENEL GÖRÜNÜMÜ***

***3. KODUN AÇIKLANMASI***

***4. KAYNAKÇA***

1. ***GİRİŞ***

* **Assembler Nedir?**

Assembler, düşük seviyeli montaj koduyla yazılmış talimatları yeniden konumlandırılabilir makine koduna dönüştüren ve yükleyici için bilgi üreten bir programdır.

Kullanıcı tarafından yazılan programları makine koduna dönüştürmek gerekir. Buna üst düzey dilin makine dili olan alt düzeye çevrilmesi denir. Bu çeviri türü sistem yazılımı yardımıyla gerçekleştirilir. Assembler, bir montaj dili programını makine dili programına çeviren bir program olarak tanımlanabilir.

ekran görüntüsü, diyagram, çizgi, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **Tek Geçişli ve Çok Geçişli Assembler Nedir?**

Assembler operasyon alanındaki anımsatıcıları (sembolleri) değerlendirerek talimatlar üretir ve sembol ve değişmez değerlerin değerini bularak makine kodu üretir. Eğer assembler tüm bu işi tek bir taramada yaparsa buna tek geçişli assembler denir, aksi halde birden fazla taramada yaparsa o zaman çoklu geçişli assembler olarak adlandırılır. Burada assembler bu görevleri iki geçişe ayırır:

**Geçiş-1(ANALİZ-SENTEZ)**

1. ***Tarama ve Etiket Çözümleme:***

* Kaynak kod satır satır okunur.
* Tüm etiketler (semboller) ve karşılık gelen adresler sembol tablosuna kaydedilir.
* Her bir talimat için adresler hesaplanır ve konum sayacı güncellenir.

**Geçiş-2:**

1. ***Makine Kodu Oluşturma:***

* Sembol tablosunu kullanarak, assembly talimatları makine koduna çevrilir.
* Etiketler gerçek adreslerle değiştirilir.
* Opcode ve operand çevirimi yapılır.
* Tanımsız semboller veya geçersiz talimatlar gibi hatalar belirlenir ve raporlanır.

diyagram, çizgi, dikdörtgen, plan içeren bir resim

Açıklama otomatik olarak oluşturuldu

* ***Assembler Çalışma Aşamaları Nelerdir?***

**(1. GEÇİŞTE GERÇEKLEŞİR)**

Assembler iki ana aşamada çalışır: Analiz Aşaması ve Sentez Aşaması. Analiz Aşaması, kodun sözdizimini doğrular, hataları kontrol eder ve bir sembol tablosu oluşturur. Sentez Aşaması, Analiz Aşamasındaki bilgileri kullanarak montaj dili talimatlarını makine koduna dönüştürür. Bu iki aşama, bilgisayar tarafından yürütülebilecek son makine kodunu üretmek için birlikte çalışır. Bu iki aşamanın birleşimi, Assembler'ı montaj dilini makine koduna dönüştürmek, yüksek kaliteli ve hatasız yazılım sağlamak için önemli bir araç haline getirir.

***1. Analiz Aşaması:***

1. Analiz aşamasının gerçekleştirdiği birincil işlev, sembol tablosunun oluşturulmasıdır. Bir programda kullanılan her sembolik ismin, N'nin adresinin montaj programında ilişkilendirildiği hafıza adresini belirlemelidir; ancak kendisinden önce gelen tüm program elemanlarının (ister talimatlar ister hafıza alanları) adresleri sabitlendikten sonra bilinecektir. Bu fonksiyona hafıza tahsisi denir.
2. Bellek tahsisi, konum sayacı (LC) adı verilen bir veri yapısı kullanılarak gerçekleştirilir.
3. Analiz aşaması, konum sayacının her zaman hedef programdaki bir sonraki hafıza sözcüğünün sahip olması gereken adresi içermesini sağlar.
4. İşlemenin başlangıcında, konum sayacını START deyiminde belirtilen sabite başlatır.
5. Bir deyimi işlerken deyimin etiketi olup olmadığını kontrol eder.
6. Eğer öyleyse, konum sayacında bulunan etiketi ve adresi sembol tablosunun yeni bir girişine girer. Daha sonra, montaj ifadesi tarafından temsil edilen talimat veya veriler için kaç tane hafıza kelimesine ihtiyaç duyulduğunu bulur ve konum sayacındaki adresi bu sayıya göre günceller.
7. Her bir derleme ifadesi için gereken bellek miktarı, bir derleme ifadesinin anımsatıcısına bağlıdır. Bu bilgiyi anımsatıcı tablosundaki uzunluk alanından alır.
8. Bu bilgiyi anımsatıcı tablosundaki uzunluk alanından alır. DC ve DS ifadeleri için bellek gereksinimi ayrıca işlenen alanında görünen sabite bağlıdır, dolayısıyla analiz aşaması bunu uygun şekilde belirlemelidir.
9. Konum sayacında yer alan adres için **<LC>** gösterimini kullanırız .
10. Sembol tablosu analiz sırasında oluşturulur ve sentez sırasında kullanılır.

***2.Sentez Aşaması:***

1. Anımsatıcılara karşılık gelen makine işlem kodunu Anımsatıcılar tablosundan edinin.
2. Sembol tablosundan her hafıza işleneninin adresini alın.
3. Duruma göre bir makine talimatını veya bir sabitin doğru temsilini sentezleyin.

* ***Bu Aşamaların Önemi Nelerdir?***

Analiz Aşaması ve Sentez Aşaması, Assembler'ın temel bileşenleridir. Analiz Aşaması, montaj dili kodunu semboller, etiketler ve talimatlar gibi kurucu parçalarına ayırır. Bu aşama, kodun sözdizimini doğrulamaktan, hataları kontrol etmekten ve sembolleri bellek adresleriyle eşleştiren bir sembol tablosu oluşturmaktan sorumludur. Bu aşama, kodun hatasız ve iyi biçimlendirilmiş olmasını sağlamak için çok önemlidir; bu, programın yürütülmesi sırasında sorunların önlenmesine yardımcı olur.

Sentez Aşaması, montaj dili talimatlarının bilgisayar tarafından yürütülebilecek makine koduna dönüştürülmesinden sorumludur. Bu aşama, makine kodunu oluşturmak ve her talimat için makine kodunu ve bellek adreslerini gösteren bir liste dosyası oluşturmak için, sembol tablosu gibi Analiz Aşamasından gelen bilgileri kullanır. Sentez Aşaması, bilgisayar tarafından yürütülebilecek son makine kodunun üretilmesi açısından kritik öneme sahiptir ve bu, onu montaj dili kodunu makine koduna dönüştürme sürecinin son adımı haline getirir.

Sonuç olarak, hem Analiz Aşaması hem de Sentez Aşaması Assembler'da hayati bir rol oynamaktadır. Analiz Aşaması kodun hatasız ve iyi biçimlendirilmiş olmasını sağlarken, Sentez Aşaması bilgisayar tarafından yürütülebilecek son makine kodunu üretir. Bu iki aşamanın birleşimi, Assembler'ı, montaj dili kodunu makine koduna dönüştürmek için önemli bir araç haline getirir; bu, yüksek kaliteli ve hatasız yazılım üretmek için kritik öneme sahiptir.

metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Analiz ve sentez aşamalarının gerçekleştirdiği görevler

yukarıdaki şemada özetlenmiştir.

* ***İki Geçişli Assembler’da Örnek Kod İncelemesi***
* **Bir assembly kodunun iki geçişli assemblerda işlenmesi;**

START: MVI A, 05H

MOV B, A

JMP NEXT

NEXT: HLT

**İlk Geçiş:**

* START etiketi sembol tablosuna eklenir ve adresi 0000H olarak belirlenir.
* NEXT etiketi sembol tablosuna eklenir ve adresi 0006H olarak belirlenir.

**İkinci Geçiş:**

* MVI A, 05H komutu 0000H adresine 3 byte olarak yazılır.
* MOV B, A komutu 0003H adresine 1 byte olarak yazılır.
* JMP NEXT komutu 0004H adresine 3 byte olarak yazılır ve NEXT etiketinin adresi olan 0006H'ya atlanır.
* HLT komutu 0006H adresine 1 byte olarak yazılır.

Bu süreç sonunda, makine kodu üretilir ve çıktı dosyasına yazılır.

1. ***PROJENİN GEREKLİLİKLERİ***

Projenin amacı, 8085 mikroişlemcisi için iki geçişli bir assembler yazılımı tasarlamak ve uygulamaktır. Sembolik çözümleme, opcode (işlem kodu) çevirisi ve bellek tahsisi gibi temel sistem programlama kavramlarını kullanarak, bir assembly dilinden makine diline çeviri yapabilen fonksiyonel bir assembler dizaynı yapılacaktır.

* Assembly dili kodu bir dosyadan okunacak, makine kodu hexadecimal formatında başka bir dosyaya yazılacaktır.
* Birinci Geçiş: Etiketleri (labels) çözümleyecek ve sembol tablosu (symbol table) gibi gerekli tabloları oluşturacaktır.
* İkinci Geçiş: Opcode çevirisi yaparak makine kodunu üretecektir.
* Basit talimatları ve adresleme modlarını işleyebilecektir.
* Söz dizimi hataları, tanımsız semboller gibi yaygın sorunları yönetmek için sağlam hata yönetimi mekanizmaları içerecektir.
* Proje C++ programlama dili kullanılarak tasarlanacaktır.

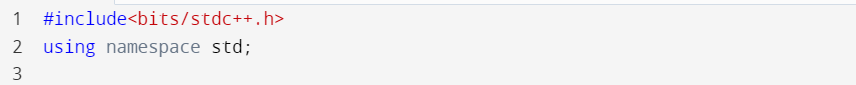
1. ***KODUN GENEL GÖRÜNÜMÜ***

* **GEÇİŞ 1:**
* Tüm sembolleri ve bölümleri çıkarır.
* Bir sembol ve bölüm tablosu oluşturulur ve bir csv dosyasında saklanır.
* **GEÇİŞ 2:**
* Her talimat makine koduna dönüştürülür.
* Program, her sembolle ilişkili adres ve boyutu almak için sembol tablosunu tekrar okur.
* **DİĞER FONKSİYONLAR:**
* **dec\_to\_bin(int) :** Verilen ondalık sayıyı ikili dizeye dönüştürür.
* **search\_MOT(string) :** MOT'ta Makine Operasyon Kodunu arar ve dizini döndürür.
* **search\_symbol\_table(string) :** Sembol tablosunda arama yapar ve sembolün adresini döndürür.
* **get\_size(string) :** Sembolün/operasyonun boyutunu döndürür.
* **get\_data(string) :** Dizeden veriyi çıkarır ve ikili eşdeğerine dönüştürür.
* **store\_symbol\_table() :** Sembol tablosunu csv formatında saklar.
* **store\_sec() :** Bölüm tablosunu csv formatında saklar.
* **STRUCTURES**
* **mnemonics :** Makine Operasyon Tablosu
* **symbol :** Sembol Tablosu
* **section :** Bölüm Tablosu
* **VERİ YAPILARI:**
* Vektör
* Dosya Akışı

|  |  |  |
| --- | --- | --- |
| * assembler.cpp | * input.txt | * section.csv |
| * assembler.exe | * output.txt | * symbol.csv |

1. ***KODUN AÇIKLANMASI***

**1. Gerekli Kütüphaneler ve Fonksiyonlar**



Bu satırlar, gerekli standart kütüphaneleri içeri aktarır. bits/stdc++.h ile tüm standart kütüphaneler dahil ediliyor.

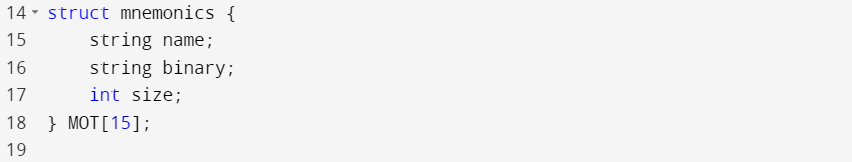
* **dec\_to\_bin Fonksiyonu:**

****

Bu fonksiyon, verilen ondalık sayıyı 8-bit ikili bir stringe dönüştürür. Sayıyı ikiliye çevirir ve eğer uzunluğu 8 bitten kısa ise başına sıfır ekleyerek 8 bit yapar.

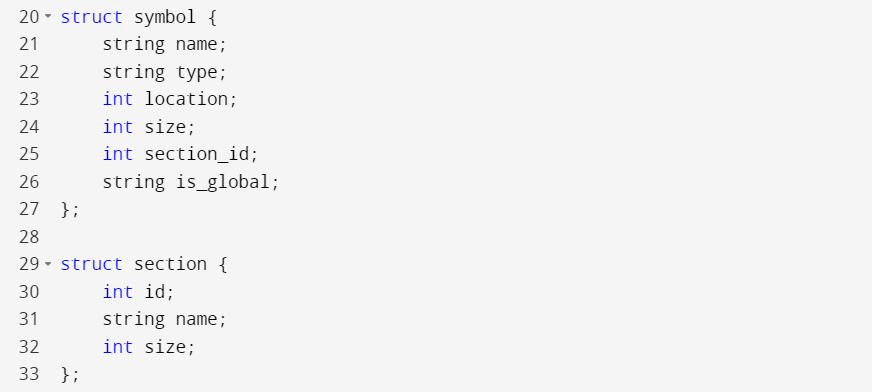
**2. Yapılar ve Global Değişkenler**

* **mnemonics Yapısı:**



mnemonics yapısı, bir mnemonic'in adını, ikili kodunu ve boyutunu tutar. MOT adlı bir dizi ile 15 mnemonics saklanır.

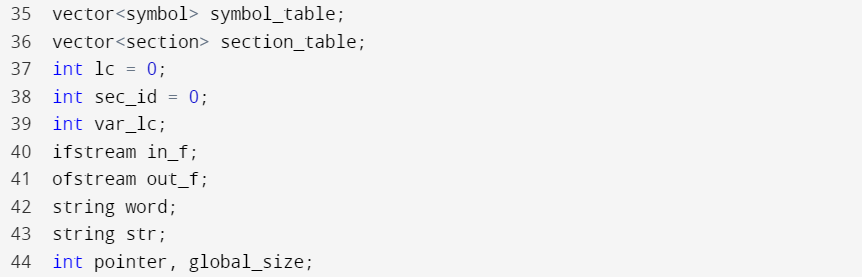
* **symbol ve section Yapıları**



symbol yapısı, sembol tablosundaki girişleri tanımlar.

section yapısı, bölüm tablosundaki girişleri tanımlar.

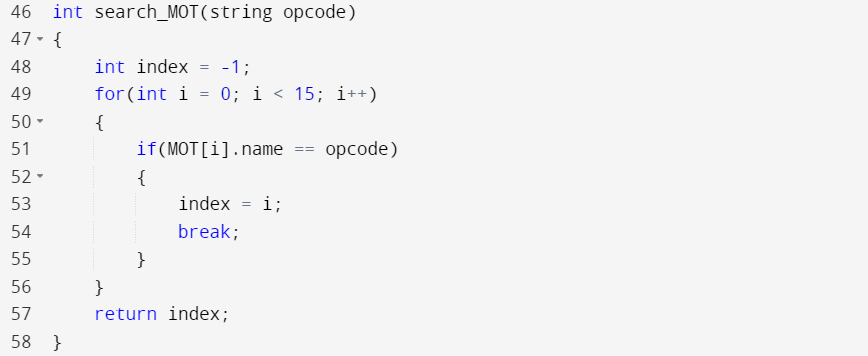
* **Global Değişkenler**

****

Çeşitli global değişkenler ve vektörler: sembol tablosu, bölüm tablosu, lokasyon sayacı (lc), bölüm kimliği (sec\_id), dosya akışları (in\_f, out\_f), geçici stringler (word, str) ve işaretçi (pointer) tanımlanır.

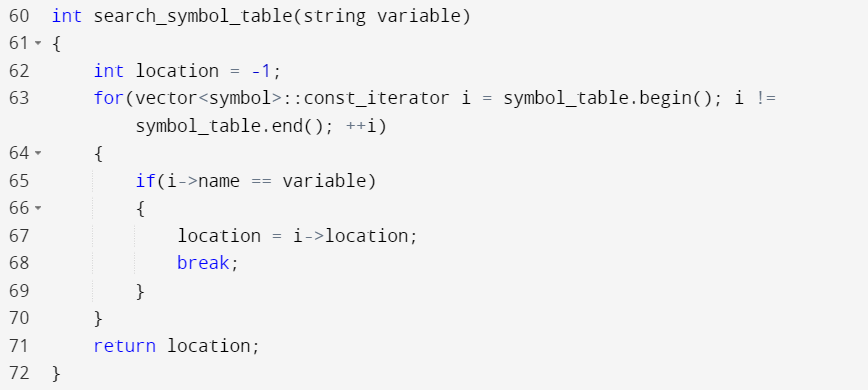
**3. Yardımcı Fonksiyonlar**

* **search\_MOT Fonksiyonu**



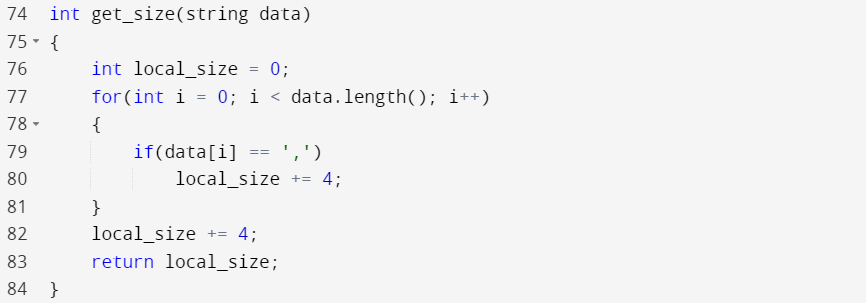
Bu fonksiyon, verilen opcode'u MOT dizisinde arar ve bulunduğu indexi döner. Bulamazsa -1 döner.

* **search\_symbol\_table Fonksiyonu**

****

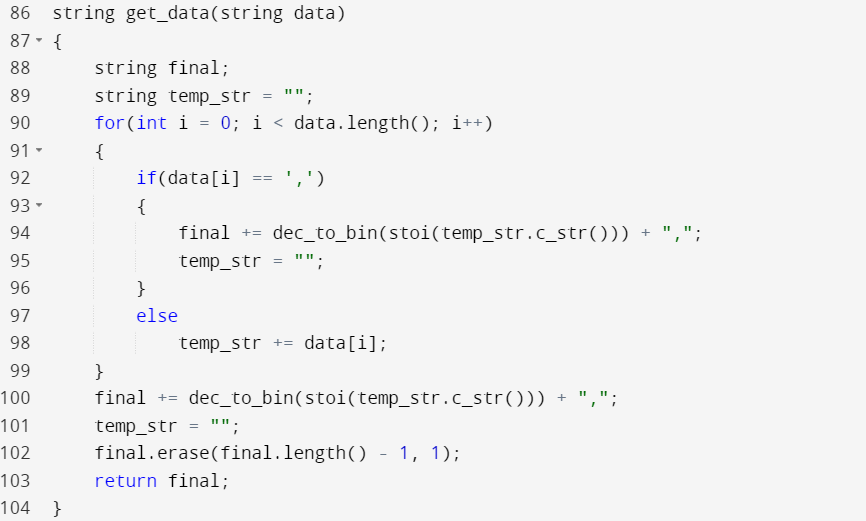
Bu fonksiyon, sembol tablosunda verilen değişkenin yerini arar ve bulursa yerini döner, bulamazsa -1 döner.

* **get\_size Fonksiyonu**



Bu fonksiyon, verilen verinin boyutunu hesaplar. Her virgül için 4 bayt ekler.

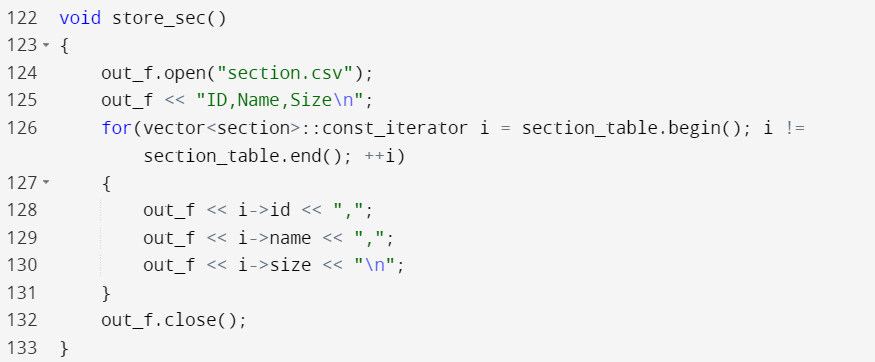
* **get\_data Fonksiyonu**

****

Bu fonksiyon, verilen veriyi işleyerek her bir elemanı ikili formata çevirir ve string olarak döner.

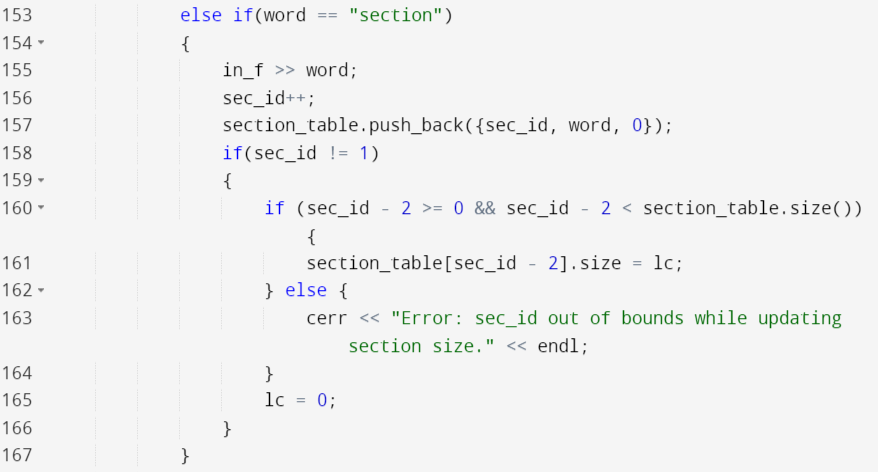
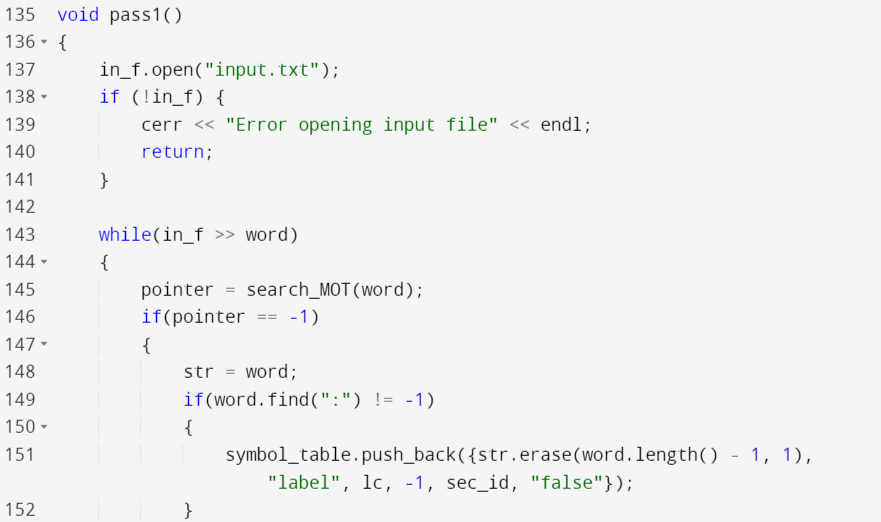
* **store\_symbol\_table ve store\_sec Fonksiyonları**

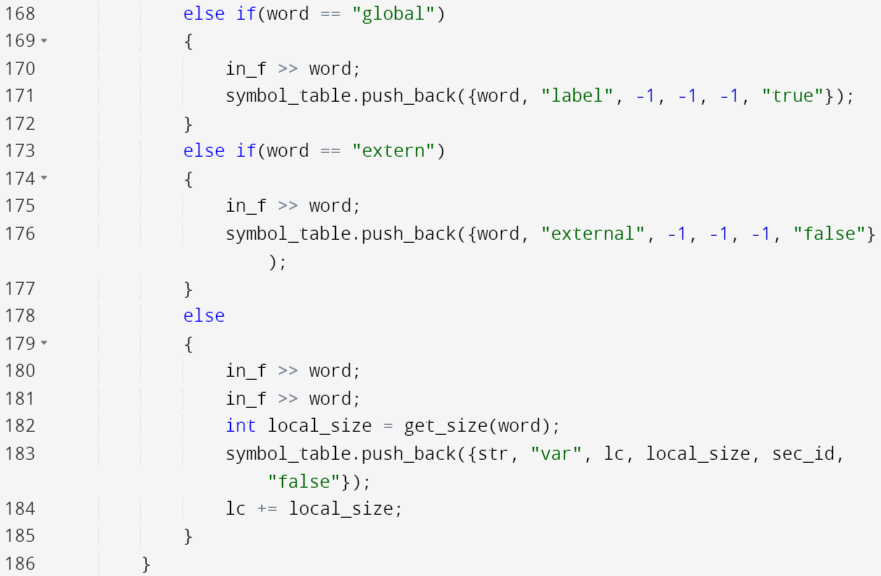
****

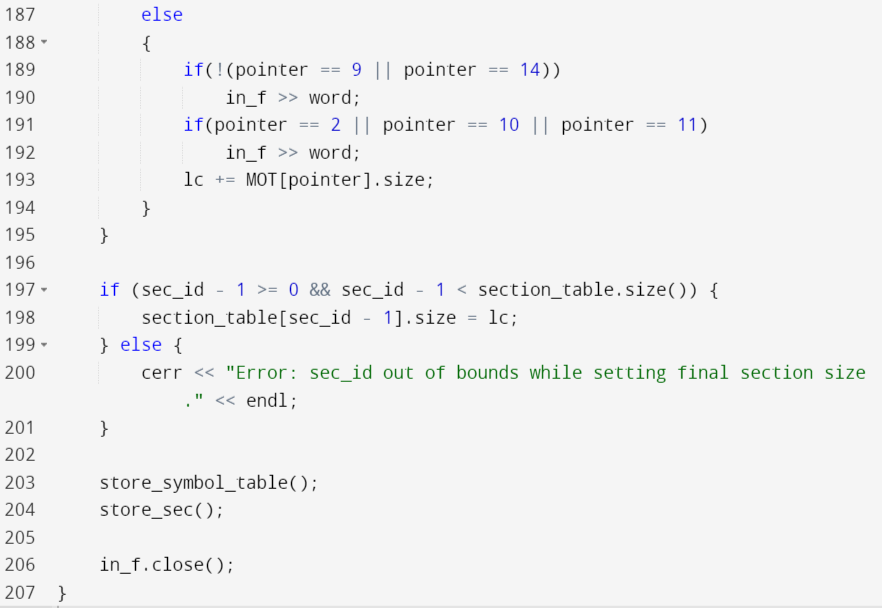
****

Bu fonksiyonlar, sembol ve bölüm tablolarını sırasıyla "symbol.csv" ve "section.csv" dosyalarına yazar.

**4. İlk Geçiş: pass1 Fonksiyonu**

****

****

****

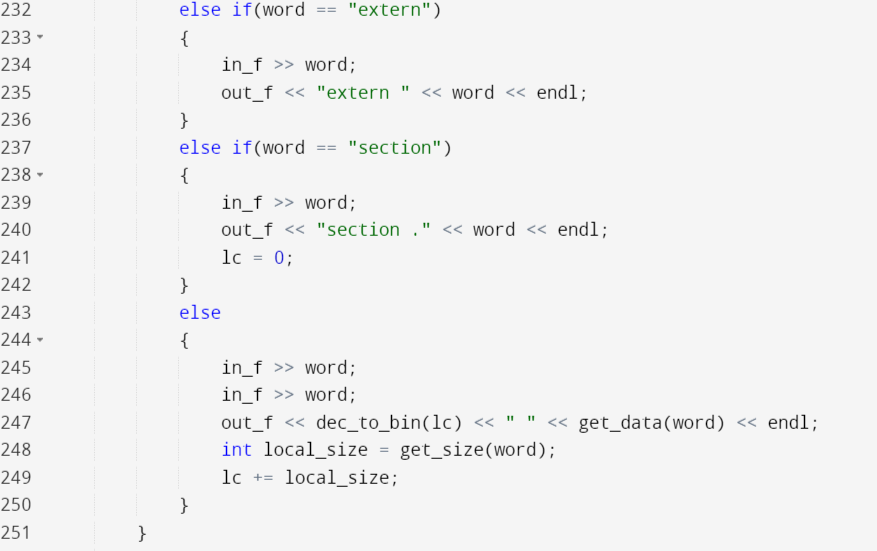
pass1 fonksiyonu, giriş dosyasını okur ve sembol ve bölüm tablolarını oluşturur. lc (lokasyon sayacı) ve sec\_id güncellenir.

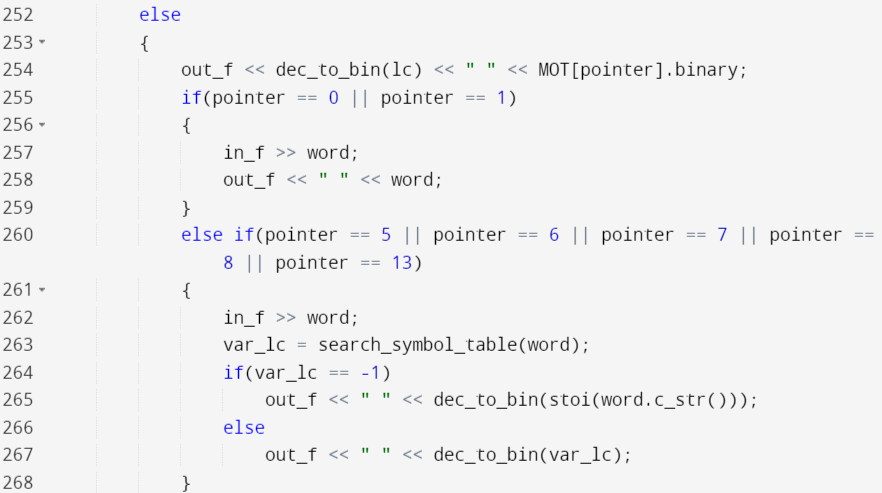
Eğer kelime bir mnemonic değilse (search\_MOT tarafından -1 döner), sembol veya bölüm olup olmadığını kontrol eder.

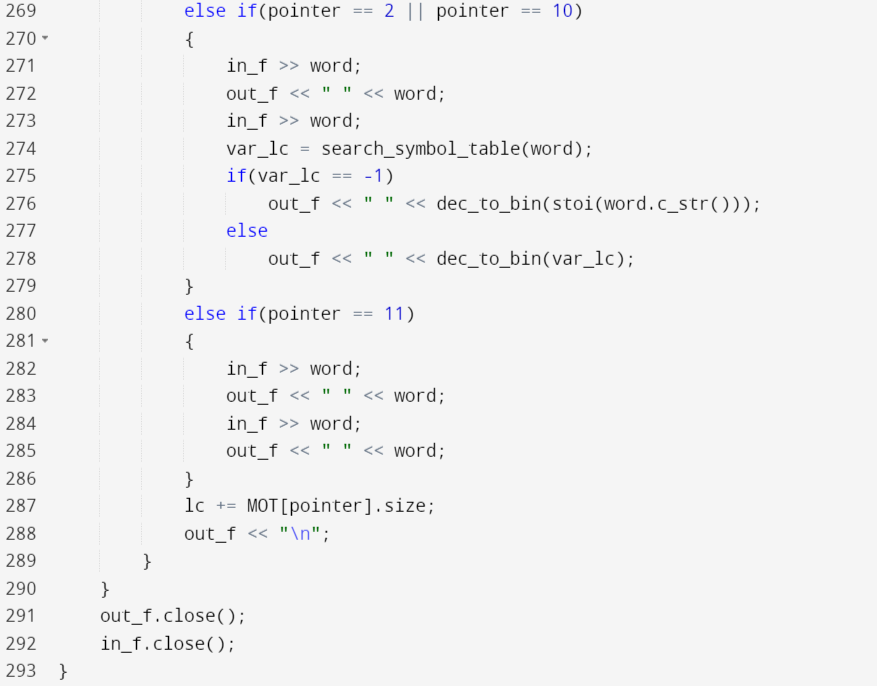
Sembol ve bölüm tablosu girişleri oluşturur ve bunları dosyalara yazar.

**5. İkinci Geçiş: pass2 Fonksiyonu**

****

****

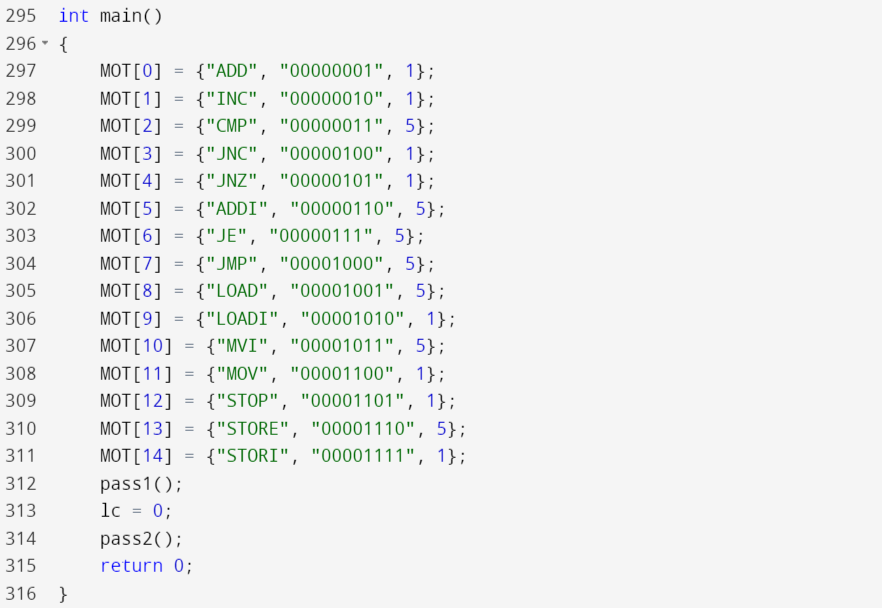
****

****

pass2 fonksiyonu, giriş dosyasını tekrar okur ve makine diline çevirir.

pass1'de oluşturulan sembol ve bölüm tablolarına göre lokasyon sayacı (lc) güncellenir ve çıktılar dosyaya yazılır.

**6. Ana Fonksiyon: main**

****

main fonksiyonu, MOT dizisini mnemonics ile doldurur.

pass1 ve pass2 fonksiyonlarını çağırarak giriş dosyasını işler ve çıktıları oluşturur.

Bu program, basit bir assembler simülasyonu yapar. İlk geçiş (pass1) sembol ve bölüm tablolarını oluşturur. İkinci geçiş (pass2) bu tabloları kullanarak makine kodunu üretir. Kod, sembol ve bölüm tablolarını dosyalara yazar ve giriş dosyasını okuyarak gerekli işlemleri gerçekleştirir.

***4. KAYNAKÇA***

<https://github.com/ayushoriginal/Two-pass-Assembler/tree/master>

<https://github.com/deveshsangwan/Two-pass-assembler/tree/master>

<https://www.geeksforgeeks.org/introduction-of-assembler/>

<https://www.scribd.com/presentation/554189027/two-pass-assembler>