MİKROİŞLEMCİLER İÇİN GENELLEŞTİRİLMİŞ ASSEMBLY DERLEYİCİ

Salih FADIL[1] **Selçuk İYİKALENDER**[2]

[1,2]Elektrik-Elektronik Mühendisliği Bölümü
Mühendislik ve Mimarlık Fakültesi
Osmangazi Üniversitesi, Batı Meşelik, 26480, ESKİŞEHİR
[1]e-posta:sfadil@ogu.edu.tr [2] e-posta: siyikalender@hotmail.com

Anahtar sözcükler: Mikroişlemciler, Assembler, Derleyiciler, Assembly Dili, XML

ÖZET

Bu bildiride, değişik mikroişlemcilere ait assembly dillerinin ifade edilebileceği ortak bir yapı anlatılmaktadır. Farklı mikroişlemcilerin, farklı Assembly dilleri de olsa, hepsinin genelde standart bir yazım biçimi vardır. Yazım biçiminin standart olması, ortak bir yapı tanımlayabilmeyi de mümkün kılmaktadır. Böyle bir yapının tanımlanması ile hedeflenen ise, assemblerin farklı mikroişlemciler ve mikrokontrolörler için yazılmış programları derleyebilmesini sağlamaktır.

1.GİRİŞ

Assembler ve dissassembler mikroişlemci projeleri için en temel araçlardır. Bu kısımlar metin ile makine kodları arasındaki dönüşümü sağlar. Dolayısı ile bunlar platforma bağlı olan kısımları teşkil ederler.

Günümüzde bir çok üretici, ürettiği çeşitli marka mikroişlemcilerin ve mikrokontrolörlerin yanında, bunlarla uygulama geliştirmek için assembler ve dissassembler yazılımlarını da üretmektedir. Bunlardan 8085 için ASM85, 8051 için A51, 80x86 türevi işlemciler için TASM veya MASM, assembler derleyicilerine örnek olarak gösterilebilir [1]. Bu programlar komut satırından veya tümleşik geliştirme ortamları (IDE) üzerinden çalıştırılarak kullanılır. Ancak bu derleyiciler genelde tek bir platform için yazıldıklarından, mikroişlemci komutlarının tanımları yazılımın kendisi içinde tanımlanmıştır ve bunu değiştirmek mümkün değildir.

Bu çalışmanın ağırlık noktası, değişik assembly dillerini derleyebilecek bir assembler çözümüdür. Varolan bütün assembly dillerinin tanımlamalarını, yazılımın bünyesine dahil etmek tutarsız bir çözüm olduğundan, tanımlamalar için ayrı bir dosya yapısı üzerinde çalışılmıştır. Assembler çalıştırılırken ilgili işlemci parametre olarak belirtilir ve bu işlemcinin tanımları dosyasından yüklenir. Bu tanımlamalara göre de istenilen assembly kaynak programı derlenir.

2. ASSEMBLY DİLİ

Her türlü programlama dili BNF yapısında tanımlanabilir [2]. Bir assembly programının BNF tanımı kabaca aşağıdaki şekilde gösterilebilir.

```
PROGRAM::={SATIR}1+

SATIR::={ETİKET}opt İŞLEM {AÇIKLAMA}opt
| {ETİKET}opt {AÇIKLAMA}opt

ETİKET::=ALFA_NÜMERİK :
AÇIKLAMA::=;{ALFA_NÜMERİK}opt

İŞLEM::=KOMUT {PARAMETRE}opt
```

Bu biçime göre bir program bir veya daha fazla satırdan oluşur. Her satır ETİKET , İŞLEM, AÇIKLAMA'nın iki varyasyonu şeklinde ifade edilebilir. İşlem ise KOMUT ve PARAMETRE olarak iki kısımdan oluşur. Örneğin,

```
START:mov A,B ;8085'te mov komutu   (1)
```

assembly programı satırı, yukarıdaki BNF tanımına göre,

```
ETİKET : START
İŞLEM  : mov A,B
       KOMUT   : mov
       PARAMETRE: A,B
AÇIKLAMA: ;8085'te mov komutu
```

şeklinde kısımlara ayrılmış olur. Farklı işlemciler için aynı BNF tanımı aşağıdaki şekilde uygulanabilir.

```
mov EAX,DS:[EBX+8] ;80386'da mov (2)
ETİKET:
İŞLEM:mov EAX,DS:[EBX+8]
       KOMUT:mov
       PARAMETRE: EAX,DS:[EBX+8]
AÇIKLAMA:; 80386'da mov

START: ld A,10 ; Bir Z-80 Programı
ETİKET: START
İŞLEM: ld A,10
       KOMUT: ld
       PARAMETRE A,10
AÇIKLAMA:; Bir Z-80 Programı
```

Yukarıda verilen örneklerden açıkça görüldüğü gibi programların yazım düzenlerinde farklılık yoktur. Farklılık KOMUT ve PARAMETRE kısımlarında olmaktadır. Parametrelerin kullanım biçimi bile genelde aynıdır. Ancak 80386 örneğinde olduğu gibi,

daha gelişmiş işlemcilerde parametrelerin kullanımı daha karmaşıktır.

3.TANIM DOSYASI

Daha önce belirtildiği gibi bu çalışmadaki assembler derleyicisi assembly dili tanımlarını bir dosyadan almaktadır. Bu tanım dosyası, ilgili mikroişlemcinin yazmaçlarının adlarını, boyutlarını, komutlarının adlarını, yazım kurallarını ve makine kodu çevrim bilgilerini içermektedir.

Bu dosyanın yapısı oluşturulurken dikkat edilen önemli bir konu uyumluluktur. Diğer derleyiciler için yazılmış programlarda mümkün olduğu kadar (hatta hiç) değişiklik yapılmaması gerekir. Bunun için de bazı işlemcilerin assembly dillerinin yazım biçimleri arasında olabilecek ufak faklılıkların da tanımlanabilmesi gerekir. Örneğin, 8051'de bir nümerik değer atama, dolaylı adresleme ve doğrudan adresleme için bazı ön ekler kullanılmaktadır. Bu tip ön ekler ve son ekler de tanımlanabilmelidir.

Bir başka önemli konu, ilgili tanım dosyasının hazırlanması veya üzerinde değişiklik yapılmasının kolay olabilmesi için metin biçimli olmasıdır. Metin biçimli dosyalar arasında şu an oldukça popüler olan XML göz önüne alınan iş için oldukça uygundur. XML bir kullanıcıya kendi isteklerine göre herhangi bir veriyi hiyearşik ilişkileri ile tanımlamasına imkan verir. Ayrıca bir standart olması ve kullanımının gittikçe yaygınlaşması nedeniyle de önemli bir tercih sebebidir.

XML dosyasının 8085 mikroişlemcisi için biçimi kabaca aşağıdaki şekildedir.

```
<ASM platform="8085">
      <RULES>
            <RULE name="COMMENT" begins=";" ends="CRLF"/>
            <RULE name="IMMEDIATE" begins="" ends=""/>
            <RULE name="DIRECT" begins="" ends=""/>
            <RULE name="INDIRECT" begins="" ends=""/>
            <RULE name="ORDER" begins="" ends="LE"/>
            .
            .
      </RULES>
      <REGS>
            <REG name="A"/ size="8"/>
            <REG name="B"/ size="8"/>
            .
            .
      </REGS>
      <GROUPS>
            <GRP name="G1">
                  <ITEM name="A" value="7"/>
                  <ITEM name="B" value="0"/>
                  .
                  .
            </GRP>
            .
            .
      </GROUPS>
      <INSTS>
            <INS name="mov" param="A,B"  code="78"/>
            <INS name="mov" param="B,G1" code="40"/>
            <INS name="add" param="G1"   code="80"/>
            <INS name="mvi" param="A,#8" code="3E#1"/>
            <INS name="jmp" param="@16"  code="C3@1@1"/>
            .
            .
      </INSTS>
</ASM>
```

Yukarıda verilen dosya yapısı aşağıda açıklanmıştır.
<ASM>: Kök blok 'tur. Tanımların hangi platforma ait olduğu bilgisi burada belirtilir.
<RULES>: Derleyici kurallarının belirtildiği kısımdır. Bir işlemcinin assembly diline ait özel yazım kuralları varsa, her bir kural **<RULE>** ifadesi şeklinde tanımlanabilir. **<RULE>** ifadesindeki *name* özelliği kuralın adını verir. Kural adı keyfi olmayıp derleyici tarafından tanınabilen ifadelerdir. Buna göre kural adları;
"COMMENT": Açıklama satırının tanımdır. *begins* özelliğinde ne ile başladığı, *ends* özelliğinde ne ile bittiği belirtilir.
"IMMEDIATE", "DIRECT", "INDIRECT": sırası ile sayısal değer atama, doğrudan adresleme ve dolaylı adresleme kullanımlarındaki ön ekler ve son eklerin tanımlamaları için kullanılır.
"ORDER": makine kodu üretimi sırasında gereken bilgidir. Little Endian ve Big Endian şeklinde İki farklı bayt sıralaması kullanılmaktadır. Little Endian veriyi en küçük baytından başlayarak saklar. Big Endian ise bunun tersidir. Kural tanımındaki *value* değeri "LE" olduğundan Little Endian bayt sıralaması, "BE" olduğunda ise Big Endian bayt sıralaması kullanılır.
<REGS>: işlemcinin yazmaçlarının tanımlandığı bloktur. Yazmaçların kural dışı kullanımını belirlemek için buradaki tanımlardan yararlanılır. Ancak bu tanımlamaların mutlaka yazmaç olması şartı yoktur. Örneğin 8085'te 'M', 'PSW' ifadeleri gerçekte yazmaç olmamalarına karşılık, yazmaçmış gibi işlem görürler. Bu tip tanımlamalar da bu blok altında yapılır. **<REG>** tanımı iki özellik içerir. Bunlardan *name* özelliği yazmaç ismini içerir; *size* özelliği yazmacın bit cinsinden büyüklüğünü belirtir.
<GROUPS>: Grup tanımlamalarının bulunduğu bloktur. Birden fazla grup içerebilir. Her bir grup, **<GRP>** alt bloğu olarak tanımlanır. Grup üyeleri ise, **<ITEM>** ifadesi ile tanımlanır. **<ITEM>**'in *name* özelliği üyenin ismini belirtir; *value* özelliği ise üyenin makine kodu katkısını belirtir. Grup tanımlamalarının tam olarak işlevi ileride anlatılacaktır.
<INSTS>: Komut tanımları bu blokta bulunur. Her bir tanımlama **<INS>** ifadesiyle yapılır. **<INS>** ifadesi üç özellikten oluşur. *name* özelliği komutun ismini içerir ve arama için referanstır. *param* özelliği komutun parametrelerinin yazım biçimini belirtir. Kaç parametre olduğu nasıl ayrıldığı ve parametrelerin neler olduğu tanımlıdır. *code* ise onaltılık sistemde bir sayı olarak komutun makine kodu çevrim bilgisini içerir. *code* içindeki her karakter onaltılık basamak değerini temsil eder.

Her bir komuta ait tüm varyasyonların ayrı **<INS>** ifadeleriyle tanımlanabileceği gibi, işlemciye özel bazı kurallardan yararlanılarak daha sade tanımlama mümkündür. Daha önce bahsedilen grup tanımlamaları burada işe yaramaktadır. Mikroişlemcilerde bir çok komutun makine kodu değerlerinin, (bu komutların aldığı parametrelere göre) bir sırası vardır. Örneğin 8085'teki ADD komutuna ait tüm varyasyonlar ve onaltılık sistemde makine kodları aşağıdaki gibidir.

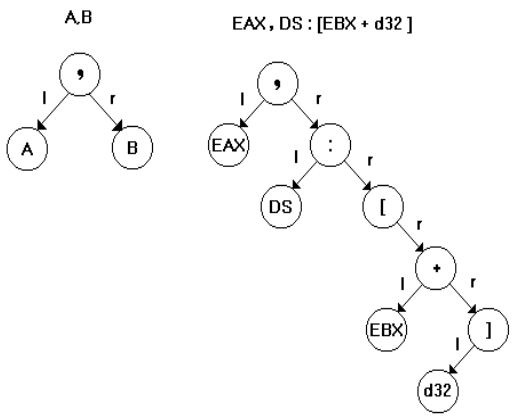
```
ADD B = 80
ADD C = 81
ADD D = 82
ADD E = 83
ADD H = 84
ADD L = 85
ADD M = 86
ADD A = 87
```

Tüm bu varyasyonları ayrı ayrı tanımlamak yerine, tüm kullanım şekillerindeki parametreler (A,B,C,D,E,H,L,M) bir grupta toplanabilir. Bunun sayesinde ADD komutu için sadece aşağıdaki tanımı yapmak yeterli olmaktadır.

```
ADD G1 = 80
```

4. ANALİZ

Derleyici ilk önce tanım dosyasını yükler. Sonra assembly programı bu tanımlar üzerinden satır satır analiz edilir. Her satır, daha önce bahsedilen ETİKET, İŞLEM, AÇIKLAMA kısımlarına ayrıştırılır. İŞLEM kısmında herhangi bir ifade var ise ilk boşluk karakterinden KOMUT ve PARAMETRE kısımlarına ayrılır. PARAMETRE opsiyonel bir kullanım olduğundan boş olabilir. KOMUT olarak ayrılan kısım tanım dosyasındaki **<INS>** ifadelerinin *name* özelliği içersinde aranır. Bir karşılık bulunamaz ise, hata üretilir. Eğer bir karşılık bulunursa tanım dosyasından komutun bir parametre kullanımının olup olmadığına bakılır. Yok ise işlenen komut için, **<INS>** ifadesindeki *code* özelliğinde tanımlanan makine kodu karşılığı üretilir. Eğer komutta parametre kullanılmış ise, kaynak programındaki parametre ve tanım dosyasındaki parametre, ayrıştırma karakterleri üzerinden ikili ağaç (binary trees) şekline dönüştülür. Örneğin, Satır (1) ve (2) 'deki kullanımlar için , : [+] işaretleri ayıraç olarak kullanılırsa, ağacın sol ve sağ dalları aşağıda gösterildiği gibi olur.



Parametrelerin doğru kullanımında her iki ağaçta (işlenen program satırına ait parametreye ve tanım dosyasındaki parametreye ait ağaçlar) birbiri ile uyuşur. Ağaçlar birbiriyle uyuşmaz ise, komutun diğer varyasyonlarına bakılır. Komut için tanımlanan varyasyonların hiçbiri içinden, uyuşan bir kullanım çıkmaz ise, parametre kullanım hatası üretilir. Burada önemli olan husus, bir yanlış kullanımın söz konusu komuta ait tüm varyasyonlar için de yanlış sonuç vermesidir. Bu noktada hangi varyasyonun yanlış kullanılmış olabileceği belirlenmelidir. Bunun için izlenen yöntem puanlamadır. En yüksek puanın alındığı tanım için hata mesajı üretilir. Bu çalışmada puanlama, yukarıda bahsedilen ikili ağaçlar için yapılan karşılaştırmada ne kadar derine inildiği bilgisi üzerinden yapılır.

Eğer her iki ağaç birbiri ile tamamen uyuşuyor ise, ilgili **<INS>** tanımındaki *code* özelliğinden alınan makine kodu değeri üzerine parametre katkıları eklenir. Parametrelerin makine kodu katkıları temel olarak üç farklı kısma ayrılabilir. Birincisi, parametrenin bir katkısının olmadığı durumdur. Komut kullanımı ile tanım birebir aynı ise doğrudan makine kodu bilgisi yazılır.

İkincisi, parametre komut makine kodunun hemen ardından gelen baytlar şeklindedir. Bu tür katkı tanım dosyasında bir takım özel işaretlerle belirtilmektedir. Örneğin,

```
<INS name="jmp" param="@16"
code="C3@1@1"/>
```

ifadesindeki '@1@1' işareti veya

```
<INS name="mvi" param="A,#8"
code="3E#1"/>
```

ifadesindeki '#1' işareti parametreye ait baytların makine kodu içindeki konumlarını belirtir. İşaretlerin içindeki sayılar, aynı tür parametreden birden fazla kullanıldığı durumlarda, kaçıncı parametre olduğunu belirlemek için kullanılır.

Üçüncü tür katkı ise, grup tanımlamalarında kullanılır. Burada komut için tanımlanan makine kodu, bir taban değerdir. İlgili parametre bir grup tanımında yer alıyorsa, parametrenin değeri, taban makine kodu ile aritmetik olarak toplanır. Örneğin,

```
mov B,A
```

komutu tanım dosyasında aşağıdaki **<INS>** ifadesiyle tanımlanır.

```
<INS name="mov" param="B,G1" code="40"/>
```

Komuttaki A parametresi, tanımda 'G1' grubu olarak belirtilmiştir. 'G1' grubunun üyeleri içersinde A aşağıdaki ifade ile tanımlanmıştır.

```
<ITEM name="A" value="7"/>
```

A 'nın bu ifadedeki değeri onaltılık sistemde 7'dir. Komutun taban makine kodu değeri yine onaltılık sistemde 40'dır. Buna parametrenin katkısı aritmetik olarak eklenirse üretilen makine kodu değeri onaltılık sistemde 47 olur.

Sonuç

Yukarıda tanımlanan yapı mikroişlemcilerin assembly dili programlarını derleyebilmek için gerekli bilgileri ifade edebilmektedir. Ancak bazı eksiklikler hala mevcuttur. Bunlardan ilki programlamada önemli bir işlevi olan makrolardır. Makro tanımlamak için var olan yapının genişletilmesi gerekmektedir.

Bir diğer eksiklik derleme sonunda üretilen çıktılardır. Bu çalışmada birkaç standart çıktı biçimi, derleyicinin içinde tanımlanmıştır. Fakat bu çıktıların da biçiminin tanımlanabileceği bir yapı oluşturmak mümkün olabilir

Bu çalışmada önerilen yapıyı destekleyen geliştirme ortamları birden çok mikroişlemci üzerinde çalışabilecektir. Böylece her üreticinin kendi mikroişlemcisi için assembler derleyicisi yazmasından kaynaklanan kalabalığı azaltmak mümkündür. Ayrıca tanım dosyaları yeni bir assembler veya dissassembler yazmaktan çok daha kolay olduğu gibi, boyutları yüzünden de daha taşınabilirdir.

Bu çalışma, herhangi bir mikroişlemci için yazılmış olan programları mümkün olan başka bir mikroişlemciye çevirebilecek bir yapı için de zemin olabilir.

REFERANSLAR

[1] Kılıçlı D., ASSEMBLER UYGULAMALARIYLA PC'NİN SIRLARI, Sistem Yayıncılık, 1995.

[2] Holub A. I, COMPILER DESIGN IN C, PrenticeHall, 1991.