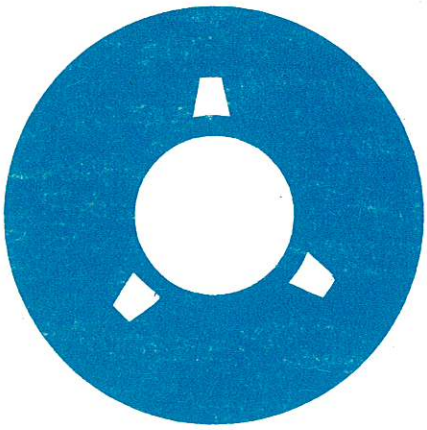


TÜRKİYE BİLİŞİM DERNEĞİ

2nci ULUSAL BİLİŞİM KURULTAYI

18-20 ARALIK, 1978

ANKARA

BİLİŞİM'78

BİLDİRİLER

Türkiye Bilişim Derneği Yayınları, Sayı: 4

ANKARA, 1978

BİR XPL DERLEYİCİSİ

Fazlı Can

Araştırma Asistanı, ODTÜ Elek.Müh.Bölümü

Özet

Bu bildiride ODTÜ Elektronik Hesap Bilimleri Bölümünde yüksek lisans tezi olarak geliştirilen XPL derleyicisinin yapımı anlatılmaktadır. Geliştiricileri tarafından, bir çevirici yazma dizgesi çerçevesinde kullanılan XPL, PL/I benzeri bir dil olup dizge programlamasına yöneliktir. Bildiride XPL dilinin özellikleri tanıtılmış, çevirici (translator) yazılımında biçimsel bir yaklaşım olan çevirici yazma dizgelerine değinilmiş, McKeeman, Horning, Wortman çevirici yazma dizgesi anlatılmıştır. Bir XPL derleyicisini, ODTÜ'de geliştirirken izlenen yol ayrıca konu edilmektedir.

1. Giriş

Yazılım dizgeleri, özellikle derleyici yapımına yönelik olan XPL bir PL/I benzeridir. McKeeman'ın oluşturduğu bir grubun üç yıllık çalışmasının ürünüdür[3, viii]. XPL dilinin BNF gösteriminde sözdizimi ve IBM/360 için yazılmış öz-derleyicisi [3] de görülebilir. Aynı yapıtda, yazarların geliştirdiği özgün bir çevirici yazma dizgesi de (translator writing system) verilmiştir. Çeşitli araştırma özeklerinde geliştirilmiş öteki dokuz XPL derleyicisiyle ilgili bilgi [4] de görülebilir. XPL diline gösterilen ilginin ana nedeni çevirici yazma dizgesi (ÇYD) üretmeye yatkın bir dil olmasıdır.

Bu çalışmadaki XPL derleyicisi, ODTÜ Elektrik Mühendisliği Bölümü bilgi işlem laboratuvarlarında araştırma amacıyla kullanılan Interdata 7/32 bilgisayarında geliştirilmektedir.

2. XPL Dili

İlk bakışta XPL,PL/I'a benziyorsa da önemli ayrımları şöyle sıralanabilir:

1. Bir XPL programı bir ana program ve onun içine yerleştirilmiş iç yordamlardan oluşur, dolayısıyla bütün programlar bir arada derlenir.
2. Tamsayı, simge ve bit dizileri (string) değişken türleridir.
3. Kümesel bellek (heap storage) yöntemiyle işletim zamanında yer atanan simgesel diziler dışındaki değişkenlere derleme sırasında yer atanır.
4. Yordamlar özyineli değildir.
5. Ayrılmış sözcükler (procedure, do, if gibi) için önceden belirtilmiş kısaltmalar yoktur.
6. Değişkenler kullanımlarından önce tanımlanmalıdır, tanımlamalar zorunludur.
7. Bütün değişken türlerinde tek boyutlu dizi (array) tanımlamak olanaklıdır, alt sınır sıfırdır.
8. Yap (do) döngülerinde adımlar yalnız sıfırdan büyük olabilir.

XPL'de yapıca düzenli programlama için gerekli özellikler vardır. Ardıştırma için gerekli yapı, do end ayraçlarıyla oluşturulur:

```
do; T1;T2;...Tn; end
```

Koşullu tümceler:

```
if B then T1; else T2;
```

ya da

```
if B then T; biçiminde olabilir.
```

Yineleme tümcesi:

```
do while B; T; end;
```

seçici tümce:

```
do case A; T1;T2;...Tn; end
```

biçimindedir. Yukardakilerin yanı sıra yineleme için

```
do i=A1 to A2; T; end;
do i=A1 to A2 by A3; T; end;
```

biçiminde döngü olanakları vardır. Sözdizimi verilen tümcelerde, T_i tümce, B Boole deyimi, A_i aritmetiksel deyim anlamına gelmektedir.

Program yazarken tamsayılara 0 la $2^{31}-1$ arasındaki değerler atanabilir. Tamsayılar işletim zamanında sıfırdan küçük olabilir. İki, dört sekiz ve onaltı tabanına göre sayı tanımlamak olanaklıdır. On tabanı dışındaki sayılar "imi içinde tanımlanır. Bu tür tanımlamalarda taban belirtilmezse 16 alınır, belirtilirse "(" ve ")" içinde verilir.

Simge dizileri 0'la 256 byte uzunluğunda olabilir. Örneğin, "(boş simge dizisi), 'ODTÜ', 'EUGENE O"NEIL' geçerli simge dizileridir.

İstenilen uzunlukta olan değişken adları abecesel bir simgeyle başlar. Özgün derleyicide en fazla 256 simgeden oluşabilen adlar bu uyarlamada sınırsız uzunlukta olabilirler, ancak ilk 8 simge ayrımlı olmalıdır.

XPL'nin yapısı öbek yapılı dillere benzer.

```
declare a fixed;                    } 1
a = 3;                              }
P:procedure;                        }
  declare a fixed;                  }
  a = 4;                            }
  /* a'nın değerini yaz. */         } 2
  output = a;                       }
  end P;                            }
  /* P'yi çağır */                  }
  P; output = a;                    } 1
```

1 Genel a'nın geçerlilik alanı
2 Yerel a'nın geçerlilik alanı

Yukardaki programın çıktısı 4 ve 3 biçimindedir. "Procedure" içindeki yerel a'ya yapılan atamalar genel a'yı değiştirmez.

Değişkenleri tanımlarken derleme zamanında yapılacak ilk değer ataması "initial" sözcüğüyle sağlanabilir.

XPL'de yalınç makro olanağı vardır, değiştirgen (parameter) kullanılamaz.

```
declare a literally '20';
declare b(a) fixed;
        c(a) fixed;
b(a)=c(a)+a;
```

Yukardaki programın eşdeğeri şöyledir:

```
declare b(20) fixed;
        c(20) fixed;
b(20)=c(20)+20;
```

Dizilerde (array)taşma denetimi yapılmaz. Böylece belleğin rasgele bir yerine erişilebilir. Atamalar birden çok olabilir: A,B,C=0; gibi. Atamalarda sağ yandaki deyim (değişken) sol yandaki değişken(ler)le türdeş olmalıdır; tersi durumda uygunsa tür değişikliği yapılır, ya da programcı uyarılır.

XPL'de simge dizileri için hazır dizge yordamları vardır, altdizi (substr) ve bitiştirme (concatanetion) gibi. Bir simge dizisinin uzunluğu "length", uzunluk ve bellekteki yeri "descriptor" adlı dizge yordamıyla öğrenilebilir. Ayrıca belleğin istenilen byte'ına "corebyte", istenilen sözcüğüneyse "coreword" adlı dizge yordamıyla ulaşılabilir. Bir değişkenin bellekteki yeri "addr" adlı dizge yordamıyla öğrenilebilir.

Yordamlar arası iletişim "değerle çağırma" biçiminde yapılır, dolayısıyla yordam dönüşü bilgi iletimi işlev yordamıyla, ya da genel bir değişkene değer atamayla yapılır.

Derleme sırasında makine dilinde istenen kod, derlenen programın içine "inline" adlı dizge yordamı çağırılarak yerleştirilebilir (crutch coding).

Derleme sırasında derleyici denetim değişkenleriyle, üretilen kod ve simge çizelgeleriy-

le ilgili sayılama bilgileri alınabilir. Bir kartın program için geçerli alan boyu değiştirilebilir.

3. Çevirici Yazma Dizgeleri

Çevirici (translator) yazılımında karşılaşılan genel sorunlar ÇYD'lerin gelişmesine yol açmıştır. ÇYD, çevirici (derleyici, yorumlayıcı gibi) yazılımını kolaylaştıracak bir program, ya da programlar topluluğu olarak tanımlanabilir [2,s.436] . Bu tanıma göre özdevimsel ayrıştırıcı (parser) için gerekli çizelgelerin üreticisi de bir ÇYD olabilir.

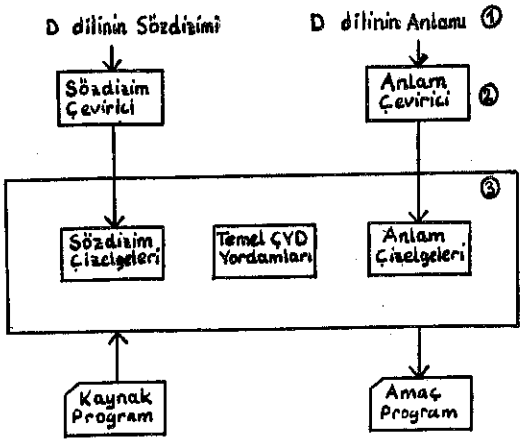
ÇYD üzerine yapılan çalışmaların çoğunluğu derleyici geliştirimine yönlendirilmiştir, bu türden ÇYD'ler derleyici-derleyicisi (compiler-compiler) olarak adlandırılır. Derleyici-derleyicisi (DD) deyiminin çıkışı bir derleyicinin yapımında başka bir derleyicinin kullanımından kaynaklanmaktadır.

ÇYD'lerde yüksek düzeyli diller kullanılır. ÇYD kullanarak geliştirilmiş bir derleyici ile, yapımında çevirici (assembler) dili kullanılmış bir derleyici karşılaştırılırsa, birincinin daha fazla bellek gerektireceği ve daha uzun sürede derleneceği gözükür. Brooker,Morris ve Rohl, DD ile yazılmış bir derleyiciyi bir yıl içinde kullandığı bellek açısından 1,6 zaman açısından 1,7 oranında kısaltmışlardır. Bu kısaltma, derleyici yazılımında çevirici dili kullanarak elde edilmiştir [2,s.436]. Buna karşın, derleyici yapımında izlenen yolun elden geldiğince çevirici dilinden kaçma biçiminde olduğunu söylemek de yanlış olmaz. Derleyici yazılımında yüksek düzeyli dil kullanarak, amaçlanan yazılım dizgesi kolayca programlanmakta, yanılgı düzeltimi süresi ve gider azalmakta, yazmaç ve bit'lerle uğraşmak ortadan kalkmaktadır. ÇYD kullanılsın kullanılmasın eğilim bu yöndedir. Örneğin, IBM'in FORTRAN-H derleyicisinde FORTRAN, Burroughs'un genişletilmiş AIGOL derleyicisinde genişletilmiş ALGOL kullanılmıştar [2,s.448]. Bu türden çalışmalarda başarı oldukça yüksektir ve bu yadsınmaz bir biçimde derleyici yapımında yüksek düzeyli dil kullanılmasından kaynaklanmaktadır. Örneğin, Lowry ve Medlock FORTRAN-H'deki kod eniyilemesinde ulaştıkları sonucun, FORTRAN kullanmalarından doğduğunu belirtmektedirler [2,s.436].

Aslında dizge programlamasında kullanılan diller yüksek düzeyli ancak makineye yönelik olma yolundadır. Bu yaklaşımla programcı istediği makine komutlarına programında kolaylıkla ulaşmakta,bu da programcının verimliliğini arttırmaktadır. İkinci seçenek olan genel amaçlı bir dil kullanımı, dizge programının taşınabilir olmasını sağlıyacak, ancak programcı verimliliği düşecektir.

ÇYD'ler iki parçadan oluşmaktadır;

1. Derleyicisi yazılacak olan D dilinin sözdizimini tanımlayan bir dil.
2. D dilinin anlam çözümlemesinde ve kod üretiminde kullanılabilecek bir dil (FORTRAN, ALGOL gibi, ya da özel olarak geliştirilmiş diller).



ÇYD'lerin Yapısı

① D dilinin tanımı
② ÇYD'nin yapay derleyici bölümü
③ D dili için çevirici (translator)

ÇİZİM 1

Genelde bir çevirici, kaynak programdaki sözdizim kurallarını bulan yordamlar ve anlam (semantic) yordamlarından oluşur. Çizim 1'de gösterildiği gibi bu ikisi ÇYD yardımıyla makine diline, ya da içsel bir biçime dönüştürülür. Bu yapay derleme zamanında yapılır. Çevirici, temel ÇYD yordamlarının desteğiyle artık kullanılır bir durumdadır. Derleme zamanında her sözdizim yapısı tanımlandıkça gerekli anlam yordamları işlenir. Anlam yordamları kaynak programın eşdeğeri amaç programı üretirler. Görüldüğü gibi bu yaklaşımda çevirici tek geçişli olmaktadır. Kimi ÇYD'lerde, bulunan bilgiler bir sonraki adıma iletilir (çok geçişli çeviriciler), her aşamada özel amaçlı bir dil kullanılır, örneğin ÇGS adlı ÇYD'de de bu türden bir yaklaşım vardır[1,s.97] .

Temel ÇYD yordamları kod üretimi, simge dizisi işlemleri, girdi/çıktı için gerekli yordamları, belli bir yönteme göre sözdizim denetimi yapan algoritmaları ve benzeri kolaylıkları içerir.

ÇYD'lerin sözdizimle ilgili yönü genellikle bir değişmezlik kazanmıştır. Örneğin, D dilinin sözdizimi BNF gösterimiyle sözdizim çeviriciye girdi olarak verilir, eğer D dilinin sözdizimi kullanılacak ayrıştırma yöntemine uygunsa gerekli sözdizim çizelgeleri üretilir. Gerekli ayrıştırma algoritması, temel ÇYD yordamları arasında hazırdır. ÇYD'lerin bu bölümü ayrıştırma algoritmasına uygun bütün diller için küçük değişikliklerle kullanılabilir. Başka bir ÇYD yaklaşımındaysa, temel ÇYD yordamları arasında ayrıştırma algoritması yoktur. ÇYD, D dilinin girdi olarak verilen sözdizim kurallarına göre bir ayrıştırma algoritması yaratır. Her iki seçenekte de anlam yordamlarını D diline uyarlamak gerekir.

4. <u>McKeeman-Horning-Wortman ÇYD'si</u>

McKeeman-Horning-Wortman (MHW)'ın geliştirdiği ÇYD şu bölümlerden oluşmaktadır.

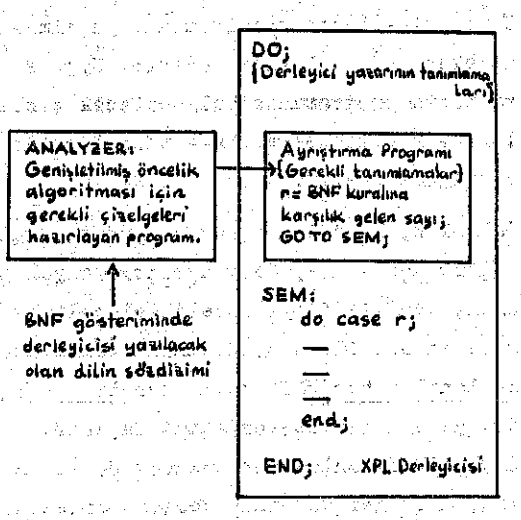
1. Sözdizim çizelgelerini üreten program: ANALYZER.
2. McKeeman'ın geliştirdiği genişletilmiş öncelik (mixed strategy precedence) algoritmasını uygulayan ayrıştırma programı: SKELETON.
3. XPL dilinde yazılmış bir öz-derleyici: XCOM.

ANALYZER, BNF gösteriminde girdi olarak verilen dilin sözdiziminin, genişletilmiş öncelik algoritmasına uygunluğunu saptar. Uyumsuzluk varsa sözdiziminde yapılması gereken değişiklikleri önerir. Uygunsa ayrıştırma programında kullanılacak çizelgeleri XPL dilinde verilmiş tanımlamalar biçiminde üretir.

SKELETON, kaynak programın sözdizim düzgünlüğünü aşağıdan-yukarı ayrıştırma yöntemi olan, genişletilmiş öncelik algoritmasına göre sınar. Dilin sözdizimini tanımlamada kullanılan her BNF üretim kuralını bulduğunda, gerekli anlam yordamını çağırır. SKELETON'da anlam yordamları boş bırakılmış, salt gerektiğinde derleme işlemini durduracak komutlar verilmiştir. Görüldüğü gibi yaratılan derleyici tek geçişlidir. MHW ÇYD'sinin bu bölümü genişletilmiş öncelik algoritmasıyla ayrıştırılabilecek her dil için kullanılabilir. Yapılması gereken tek şey, ANALYZER tarafından üretilen tanımlamaları SKELETON'nın başına koymak ve derlenen dille ilgili imleri (token) bulan tarayıcı programı ve ilgili tanımlamaları değiştirmektir.

XCOM, XPL dilinde yazılmış bir öz-derleyicidir. Tarayıcısı XPL'e göre değiştirilmiş SKELETON'a, XPL için gerekli anlam yordamları katılarak elde edilmiştir. Ayrıştırma yordamının bulduğu her BNF kuralı için bir anlam yordamı çağırılır. Bu yordamlarda gerekiyorsa makine dilinde komutlar üretilir, belirli istiflere (stack) ve yerlere, gerekli bilgiler saklanır.

Aslında XCOM'un içinde de bulunan SKELETON'ın yaptığı, ayrıştırma sırasında bulduğu BNF kuralına karşılık gelen sayıyı, anlam yordamına iletmektir. Anlam yordamında bir "case" komutuyla gerekli tümceye ulaşılır. O tümce ya boştur, ya birkaç kısa bilgi saklama işi yapar, ya da bir alt yordam çağırır. Çizim 2'de XCOM'un çalışma biçimi gösterilmiştir.



XPL Derleyicisi XCOM'un Yapısı

ÇİZİM 2

Görüldüğü gibi bu yöntem, derleyici yazılımını önemli ölçüde mekanikleştirmektedir. Yapımcı, derleyicisini oluşturmak istediği dilin sözdizimini BNF gösteriminde ANALYZER'a girdi olarak vererek gerekli çizelgeleri üretebilir. Genişletilmiş öncelik yöntemi oldukça geniş dil gösterimlerini kabul etmektedir. ANALYZER programının gereksediği değişiklerden sonra ortaya çıkacak olan dilin BNF gösterimindeki sözdizimi, programcılara kaynak olarak verilebilir. Ayrıştırma programı SKELETON, ANALYZER'in ürettiği XPL dilindeki çizelge tanımlamaları eklendiğinde, tarayıcı için yapılan küçük değişikliklerle olduğu gibi kullanılabilir. Derleyici yapımcısının görevi, derlenecek dile göre değişecek olan, anlam yordamlarını yazmaktır. Derleyicisi yazılan dil XPL'e, kullanılan makine IBM/360'a benziyorsa, XCOM'un anlam yordamları iyi bir esin kaynağı olacaktır.

5. Bir XPL Derleyicisi Yapımında İzlenen Yol

Bu çalışmadaki XPL derleyicisi ODTÜ Elektrik Mühendisliği Bölümünde bulunan Interdata 7/32 bilgisayarı için geliştirilmektedir. Derleyicinin bu bilgisayar için yapılmasının nedeni, bu makinede, XPL türünden bir dilin olmayışıdır.

Derleyicinin yapımında kaynak [3] de önerilen yolun dışında uygun sayılabilecek başka yollar izlenebilirdi. XPL'ın küçük adımlarla kendi üzerinde oluşturulması gibi. Ancak böylesine bir yaklaşımda karşımızda XPL'ın hiç de kolay sayılmayacak sözdizimini ayrıştırma sorunu vardı. Ayrıca iyi sunulmuş bir deney birikiminden yararlanamıyacaktık. Bu nedenle elden geldiğince [3] de önerilen yolları uyguladık.

Elimizdeki bilgisayarda bulunan yüksek düzeyli tek dil FORTRAN'dı, bu nedenle derleyici yapımında FORTRAN kullandık. Ancak INTERDATA 7/32'de bulunan FORTRAN derleme sonunda simgesel makine komutları ürettiği için, uyarıldığında doğrudan çevirici dilini kullanma olanağını sağlar. FORTRAN programında, birinci sütuna $ASSM sözcüğü delinirse, birinci sütuna delinmiş $FORT sözcüğüne gelene dek çevirici dilinde komutlar yazılabilir. Bu da programcıya her türlü makine komutuna ulaşma olanağı sağlamaktadır ki özellikle dizge programlamasında bu oldukça büyük rahatlık getirmektedir.

Küçük sayıları içeren büyük çizelgelerin gösterimi çevirici komutlarıyla yapılmış,

bu türden çizelgelerin ulaşımında da çevirici dili kullanılmıştır. Örneğin, ayrıştırmada istifleme (stacking) kararı için kullanılan çizelgelerden birinin elemanları, değeri 0 ve 3 arasında değişen 3822 tane sayıdan oluşmaktadır. Bu türden bir çizelgenin her üyesine bir yarım sözcük ayırarak doğrudan FORTRAN'la göstermek hiç de tutumsal olmayacaktır. Sıkıştırılmış olarak FORTRAN'la göstermek bu kez sayıların anlaşırlığını bozacak, yanı sıra yanlışlıklara yol açacaktır. Bu türden çizelgeleri çevirici dilinde göstermek hem yer kazandırmakta, hem de yanlış olasılığını azaltmaktadır.

Sözdizimi denetiminde, MHW ÇYD'sinin öner diği yol kullanılmış, genişletilmiş öncelik yöntemine göre ayrıştırma işlemi yapacak olan SKELETON, XPL'ın özel durumu için FORTRAN ve çevirici dilinde yeniden yazılmıştır. Ayrıştırma için gereken çizelgeler kaynak [3] den alınmıştır.

Özgün XPL derleyicisinin anlam yordamları, doğrudan makine komutları üretmektedir. Bu çalışmada, gereken algoritmaların karmaşıklığı, derleyicinin bellek gereksinimini çoğaltacağından doğrudan makine komutu üretilmemiş, simgesel makine komutu üretme yoluna gidilmiştir. Kullanılan bilgisayarın küçük oluşu bu seçimi zorunlu kılmaktadır. Gerek simgesel makine komutu üretimi, gerekse değişik bilgisayar kullanımı, özgün derleyicinin anlam yordamlarının kullanımını önlemektedir. Bu nedenle, bu çalışmada özgün XPL derleyicisinin anlam yordamları çıkış noktası alınarak, geliştirilen derleyicinin koşullarına uygun anlam yordamları yazılmıştır.

Simgesel makine dilinde komut üretiminin sakıncalı olduğu söylenebilir, çünkü üretilen amaç programın, bu kez çevirici (assembler) tarafından işlenmesi gerekecektir. Ancak simgesel kod üretimi derleyici yazarını büyük bir yükten kurtarmaktadır.

Interdata 7/32'nin kullandığı bellek erişim yönteminden ötürü, yeri bilinmeyen değişkenler kullanıldığında, bellekte çok yer tutan komutlar üretilir (6 byte). Yeri belirsiz değişkenlerin yeri belli olduktan sonra bu türden uzun komutlar yapılacak olan ek sıkıştırma geçişleriyle küçültülebilir. Bu iş için gerekli algoritmaların yazılımı derleyici üretimini oldukça zorlaştıracaktır. Derleme sonucunda simgesel makine komutları üretilerek sıkıştırma işi çevirici aktarılabilir. Çevirici, istenildiği zaman karalama kütükleri kullanarak, yapması gereken iki geçişten sonra, fazladan geçişler yaparak ürettiği makine komutlarını eniyiler, makine komutlarını elle yazılabilecekleri en kısa biçimde dönüştürür. Böylece 6 byte uzunluğundaki komutların çoğu kısaltılarak, 4 ya da 2 byte'a indirgenir. Yazılan programın 16K byte'dan daha kısa bellek yeri tutacağı kestiriliyorsa, programcı çeviriciye vereceği bir değiştirgen ile 6 byte'lık komutların üretimini önliyebilir (6 byte'lık komutlar 16K byte'dan daha uzak bellek yerlerinin ulaşımında kullanılır). Bu da, daha sonradan istenebilecek olan eniyileme geçişlerinin süresini kısaltacaktır. Gerek derleyici geliştirmedeki kolaylık, gerek derleyicinin bellek gereksiniminin küçülmesi, gerekse çeviricinin sağladığı eniyileme olanakları, çevirici tarafından yapılması gereken geçişlerin sakıncalarını önemsizleştirmektedir.

Interdata çeviricisinin sağladığı koşullu çeviri (conditional assembly) olanağından yararlanarak, Interdata'nın hem 32 bit'lik hem de 16 bit'lik makinesi için geçerli olabilecek simgesel makine komutları üretilebilir. Bu, derleyicinin taşınabilirliğini sağlıyacaktır. Ancak, yapılan çalışmada bu amaçlanmamıştır, oluşturulan derleyici, elden geçirilerek bu duruma getirilebilir.

Kullanılan bilgisayarın işletim zamanı kitaplığında bulunan girdi/çıktı yordamları

FORTRAN'a dönük olduğundan derleyiciye özgün girdi/çıktı yordamları yazma yolu seçilmiştir. Çalışmanın şimdeki aşamasında, kod üretimi için gerekli anlam yordamları üzerine çalışılmaktadır.

6. Sonuç

Bu bildiride XPL dilinin özellikleri anlatılmış, çeviriçi (translator) yazılımında biçimsel yaklaşım olan ÇYD'lere değinilmiş, bu konuda örnek olarak MHW ÇYD'sinin çalışması anlatılmıştır. ODTÜ'de geliştirilmekte olan bir XPL derleyicisinin yapımında izlenen yol ayrıca konu edilmiştir.

Bu çalışmadaki ana amaç, ODTÜ Elektrik Mühendisliği Bölümüne XPL türünden bir dil kazandırmaktı. MHW ÇYD'sinin ANALYZER ve SKELETON programları geliştirilen derleyiciye uyarlanarak, derleyici yazılımı derslerinde ve dil tasarımında yardımcı olarak kullanılabilir. Derleyici yapımında kazanılan deney çalışmanın öteki ürünlerinden biridir.

Kaynaklar

1. J.Feldman ve D.Gries; Translator Writing Systems, Comm.ACM, Cilt 11, Sayı 2, Şubat 1968, S.77-113.
2. D.Gries; Compiler Construction for Digital Computers, New York: John Wiley and Sons, 1971.
3. W.M.McKeeman, J.J.Horning, D.B. Wortman; A Compiler Generator Englewood Cliffs, N.J.: Prentice Hall, 1970.
4. D.B. Wortman; A Rooster of XPL Implementations, SIGPLAN Notices, Cilt 13, Sayı 1, Ocak 1978, S.70-74.