Gazi Üniv. Müh. Mim. Fak. Der.
Cilt 30, No 1, 19-28, 2015

Journal of the Faculty of Engineering and Architecture of Gazi University
Vol 30, No 1, 19-28, 2015

INTEL İŞLEMCİLERE YÖNELİK BİR UNASSEMBLER TASARIMI

Bülent DÖKME, Nurettin TOPALOĞLU
Gazi Üniversitesi, Bilişim Enstitüsü, Teknikokullar, 06500, Ankara
bulet.dokme2011@os.gazi.edu.tr, nurettin@gazi.edu.tr

(Geliş/Received: 17.12.2013; Kabul/Accepted: 11.02.2015)

ÖZET

Yazılım tersine mühendisliği; yazılım kodlarının donanım bileşenleri üzerindeki etkilerini, işlevlerini, davranış ve belgelerini analiz ederek sistem soyutlarını ve tasarım bilgilerini oluşturan bir tersine mühendislik disiplinidir. Çevirici (assembler), tersine çevirici (unassembler), sanal simülatörler ve hata ayıklayıcılar (debugger) birer yazılım tersine mühendislik araçlarıdır. Bu çalışmada, Intel mikroişlemci komut kümesine ait farklı uzunluklardaki makine kodlarını sembolik dile dönüştüren bir tersine çevirici uygulaması geliştirilmiştir. Bu uygulama, 32-bitlik Intel işlemci mimarisinde kullanılan, aynı zamanda derleyicilere yönelik, sembolik dilden makine kodlarına çevrimde, güncellenebilen ve belli bir hiyerarşi sunan açık kaynak kodlu, tablo tabanlı satır-içi kod çevrimi yapabilen bir yazılımdır. Hem assembler hem de unassembler olarak çalışan bu yazılım, sadece belgelendirme ve açıklama mahiyetindeki mevcut tablo tabanlı unassembler uygulamalarına farklı bir bakış açısı ve derinlik getirmiştir.

Anahtar Kelimeler: Assembler, unassembler, derleyici, tersine mühendislik, assembly dili

AN UNASSEMBLER DESIGN FOR THE INTEL PROCESSORS

ABSTRACT

Software reverse engineering is a discipline of generating the system of abstracts and design information by analyzing the effects, functions, behaviors and documents of the software codes on hardware components. Assembler, unassembler, virtual simulators and debuggers can be listed as software reverse engineering tools. In this study, an unassembler application is developed for translating various-length machine codes, which belong to Intel microprocessor instruction set, into the symbolic form of the machine language. This application is designed both for 32-bit Intel microprocessor architectures while it can also be used for compilers. It is an open source software with the capabilities of translating table-based in-line codes and offering a certain hierarchy for the procedure. In addition, updating the software is also possible for future considerations. With its nature of working both as an assembler and an unassembler, this software brings a different perspective and depth to the existing table-based unassembler applications which are only used for certification and description.

Keywords: Assembler, unassembler, compiler, reverse engineering, assembly language

1. GİRİŞ (INTRODUCTION)

Yazılım tersine mühendisliği (Reverse Engineering-RE), yazılım bileşenlerinin gerçek zamanlı etkileri ve düşük düzeyli yapılarının ayrıntılı analiziyle ilgili bir disiplindir. Aynı zamanda RE, bir programın mikroişlemci mimarisi üzerinde gerçekleştirdiği işlemlerin incelenmesidir [1-3]. RE, yazılımın birlikte çalışabilirliği, güvenlik yamalarının etkilerinin değerlendirilmesi, güvenlik denetimi (eğer varsa, istenmeyen ve kötü niyetli uygulamaların etkisinin belirlenmesi) ve yazılım fonksiyonlarının ve performansının artırılması gibi birçok kullanım alanına sahiptir [4]. Tersine mühendislik uygulamalarında kullanılan assembler ve dissassembler gibi yazılım araçları, mikroişlemcili sistemlere yönelik geliştirilen yazılımların analizi ve testinde kullanılır [5-7]. Bu araçlar, assembly dilinde yazılmış ham programlar ile makine kodları arasındaki karşılıklı dönüşümü sağlar. Bir mikroişlemci ya da mikrodenetleyici tasarlarken ilk yapılan iş, söz konusu işlemciye ait komut kümesinin

ortaya konulmasıdır. Komut kümesi, donanımsal özellikler ve işlemci hakkında da önemli bilgiler içerir. Assembler; assembly dilinde (sembolik dil) yazılmış komutları iş-kodlarına (opcodes) çevirirken aynı zamanda bellek alanları ve diğer varlıklar için sembolik etiketleri de çözerek amaç (makine) kodlarını oluşturur [8]. Unassembler ise, bu amaç kodlarını okuyarak sembolik dile dönüştürür [9].

Yüksek seviyeli diller, yazılım geliştiricilerin, komutları daha kısa tanımlamasına ve mikroişlemci komut kümesi yerine ilgili programlama dilinin komutlarını kullanmasına izin verir. Yazılan bir programın makine kodlarına çevrim işlemini derleyiciler yapar. Bazen yazılan bir program çeşitli sebeplerle istenilen sonucu vermeyebilir. Gerek makine kodlarının üretimi, gerekse makine kodlarının sembolik dile dönüştürülmesi işlemlerine;

- Derleyici tasarımlarında,
- Kaynak kodlarının kaybolması durumunda veya kaynağı bilinmeyen çalışan bir programın kodlarının yeniden elde edilmesi amacıyla çevirici tasarımlarında,
- Çalışan programların makine kodlarının incelenmesi ve hata ayıklamasında,
- Programın makine kodlarının sembolik karşılığının bulunmasında,
- Mikroişlemci komut kümesindeki komutları kullanarak program geliştirilmesinin öğrenilmesinde ihtiyaç duyulmaktadır.

Makine kodları donanım mimarisine ve kullanılan mikroişlemciye göre farklılıklar gösterir. Her mikroişlemci üreticisinin kendisine özgü bir mimarisi ve komut kümesi (ISA) vardır. Örneğin, x86 ailesinin komut kümesini kullanan işlemciler (AMD ve INTEL) için MASM veya TASM gibi ticari ve popüler programlar assembler derleyicilerine örnek gösterilebilir. Ancak, bu tür programlar açık kaynak kodlu değildir ve üzerinde güncelleştirme yapılmasına izin verilmez.

Intel işlemcilere yönelik, tablo tabanlı çevrim işlemleriyle ilgili yapılmış çalışmalar genelde belgelendirme ve açıklama odaklı olduğundan, bu çalışmada, belgelendirme dışında farklı bir yaklaşımla satır-içi kod tersine çevrim uygulaması gerçekleştirilmiştir. Diğerlerinden farklı olarak, tersine çevrim işlemleri gerçekleştirilirken aynı zamanda hem assembler hem de makine kodlarını yüksek düzeyli dillere çeviren decompiler işlemlerine yardımcı olabilecek bir altyapı oluşturulmuştur. Gerçekleştirilen bu işlemlerde hız ve bellek kullanımları dikkate alınmamıştır.

Düzgün ikili formattaki dosyalar; giriş kodu ve başlık bölümü bulunmayan, sisteme özgü bölüm içermeyen, yalnızca kod ve statik verilere sahip dosyalardır. Çalışma konusu, daha çok satır-içi kod çevrimi uygulamasına yöneliktir. Dinamik veya statik çevrim çalışmalarında bu konu ile ilgili kaynak kodları içeren uygulamalar incelenmiştir. Gerçekleştirilen tersine çevrim çalışmalarında Intel işlemcilerin dokümanları [10,11] göz önüne alınmıştır.

Unassembler ve assembler ile ilgili yapılmış çalışmalar tarandığında; düz ikili dosya üretebilen, açık kaynak kodlu, en yaygın sembolik makine dili derleyicisinin Nasm (Netwide Assembler) olduğu görülmüştür. Bu assemblerin aynı zamanda Ndisasm adlı bir unassembleri bulunmaktadır [12]. Bu programın kaynak kodları incelendiğinde, dönüştürme işleminin IA-32 komut kümesi formatına göre bit seviyesinde çözümü yapıldığı, çok sayıda "if else" ve "switch case" yapıları kullanıldığı görülmüştür. Kaynak kodları çok uzundur, hata takibi ve güncelleştirme yapılması zordur. Ayrıca, bu yazılım sadece gerçek mod (real mode) için tasarlanmıştır. Ma [13] çalışmasında, Intel dokümanlarındaki komut kümesi formatlarına uygun olarak her bir komut için çok sayıda makro veya "struct" yapısı içerisinde tanımlama yapılması öngörülmüştür. Tanımlamaların yapılması kadar, bunların kullanılmasını içeren prosedürlerin yazılması da ayrı bir çalışmayı gerektirmektedir [14]. Değişken uzunluktaki kodların çözümünde, komut uzunluğunun kaç bayt olduğunun bulunması önemli bir sorundur. Paleari [15] bu soruna değişik bir yol önermiştir. Bir bayt okunmakta ve mikroişlemcide komut olarak yürütülmeye çalışılmaktadır. Mikroişlemci hata verirse, sıradaki ikinci bayt, ilk okunan bayt ile birlikte mikroişlemciye komut olarak verilmekte, bu adımlar mikroişlemci hatasız işlem yapıncaya kadar devam etmektedir. Sonuçta, okunan bayt miktarı komut uzunluğu kadar olmaktadır. Tanımsız kodlarda ise, mikroişlemci sürekli hata vermekte ve komut uzunluğu asla bulunamamaktadır. Fadıl [16], Assembly dilinde yazılmış bir programın makine kodlarına dönüştürülmesi işleminin, ağaç yapısı içerisinde yapılmasını öngören çalışmasıyla tersine yapılabileceğini göstermiştir. Ancak, Intel 32-bit komut kümesinin tamamının bu yöntemle yapılması durumunda binlerce satır kod yazılması gerekebilecektir. Payer [17] çalışmasında, tablo tabanlı düşük seviyeli çevirme yapmış ve tabloyu bütün makine komutları için çok seviyeli bir arayüz olarak tasarlamıştır. Tablolarda her bir komut için *action* bulunmakta, kod üretimi esnasında seçilen komut hakkında kaydedici kullanımı, bellek erişimi, değişik ifade ediliş şekilleri, kontrol akışı, fonksiyon çağırmaları gibi detaylı bilgiler içermektedir. Bu tablo, yapılan programla beraber derlenmekte ve bir kütüphane olarak kullanılmaktadır. Tablolar çok seviyeli hazırlandığı için komut okuması bitene kadar tablolar arası geçişler, işaretçiler (pointers) vasıtası ile sağlanmaktadır. Bu yöntem, aynı zamanda analiz yapılmasına imkân vermekte ve dinamik çevirmede kullanılabilmektedir. Aynı mantıkla gerçekleştirilmiş tablo bazlı düşük seviyede çeviri yapabilen bazı çalışmalar bulunmaktadır [18-20].

İncelenen açık kaynak kodlu çalışmalar göz önüne alınarak, farklı uzunluklardaki makine kodlarının doğrudan sembolik gösteriminin sonlu durum olacak şekilde yapılmasında üç teknik kullanılmaktadır.

- Bit bazında işlem yapılarak düşük seviyede çevrim yapılabilir,
- Struct yapısı içerisinde makrolar tanımlanabilir,
- Tablo tabanlı düşük seviyeli çözümle çevrim yapılabilir.

Bu çalışmada, kodların çözülmesi ve sembolik dile dönüştürülmesinde tablo tabanlı satır-içi çevrim yöntemi öngörülmüştür. Tablo tabanlı çevrim işlemi yapan çalışmalar özellikle dinamik çevrim için tasarlanmışlardır. Uygulanan yöntemin mantığı; söz konusu dinamik çevrim için tasarlanan tablo tabanlı çözümlemelere benzemekle beraber uygulanmasında farklılıklar içermektedir. Şöyle ki;

- Çevrim tabloları farklı derleyiciler ve platformlar içinde kullanılabilmektedir
- Dinamik çevrim için ilave tanımlamalar yapılarak kullanım amacı genişletilmektedir
- Kullanılması için çok az komut satırı gerekmektedir
- Assembly dilinden makine kodu üretimine de imkân verilmektedir
- Uygulaması basit ve güncellenmesi halinde hata giderimi oldukça kolaydır. Yeni bir mikroişlemci piyasaya çıktığında sadece komut kümesi ilave edilmektedir
- IA-32-64 mimarisine ait komut kümesi farklı her mikroişlemci için ayrı tanımlama yapılabilmektedir
- Çevrim tabloları bir defa doğru olarak hazırlandıktan sonra, farklı kullanıcılar bu tabloları kullanabilmektedir.

Bu çalışmada Intel'in komut kümesi mimarisine uygun olarak, değişik uzunluklardaki makine kodlarının sembolik dile çevrilmesini sağlayan bir yazılım tasarlanmış ve çalışma sonucunda, derleyiciler için sembolik dilden makine kodlarına çevriminde güncellenebilen ve bir hiyerarşi sunan, tablo tabanlı bir çevrim dosyası oluşturulmuş ve unassemblere kaynak sağlanmıştır. Bu çevrim dosyaları bir kez doğru olarak hazırlandıktan sonra kullanıcıların bu dosyaların nasıl hazırlandığını bilmelerine gerek yoktur. Önemli olan bu dosyaların farklı platformlar ve derleyiciler dâhil olmak üzere yapılacak çevrim çalışmalarında kolaylıkla kullanılabilir olmasıdır.

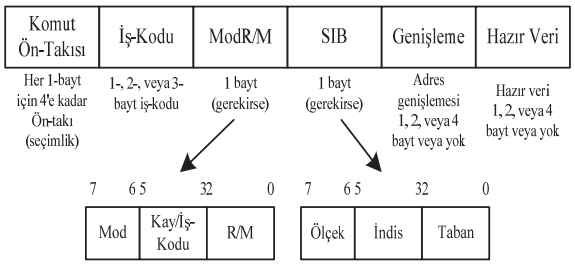
2. MATERYAL VE METOT (MATERIAL AND METHOD)

Intel işlemcilerin donanımsal yapıları ve özellikleri, komut kümesini meydana getiren komutlara yansımıştır. Intel komut kümesi temel alınarak; işlemci mimarisinin davranışlarının gözlenmesi, mimarinin ve yazılımın karakter özelliklerinin analizi, program işlevlerinin test edilmesi ve hatalardan arındırılması maksadıyla, yazılım tersine mühendislik araçlarından birisi sayılan unassembler geliştirilmiştir [21].

2.1. Intel Komut Yapısı (Intel Instruction Format)

Mikroişlemci kendi içerisinde her bir işlemi bir veya birkaç makine koduyla gerçekleştirir. Makine kodları, işlemciye hangi işlemin yapılacağını anlatan baytlar topluluğudur. Assembly dilinde tüm program bu komut dizileriyle ifade edilmek zorundadır. Bu çalışmada Intel işlemcisinin örnek alınmasının sebebi, bu işlemciye dayalı sistemlerin dünyada çok yaygın olarak kullanılmasıdır. AMD işlemcilerin de Intel komut kümesini temel alması dolayısıyla bu çalışmada gerçekleştirilen uygulamlar aynı zamanda AMD platformlarında da çalışabilir. Şekil 1.'de, 32 ve 64 bitlik Intel komut kodlarının genel formatı görülmektedir.

Komutlar; seçimlik ön-takılar (herhangi bir sırada), bir veya iki birincil iş-kodu baytları, gerekirse adres tanımlama belirteci olarak ModR/M (Register/Memory) baytı ve bazen SIB (Scale-Index-Base/Ölçekli Taban Indeksi) baytı, gerekirse bir genişleme (displacement/disp) baytı ve gerekirse hazır veri alanından meydana gelmektedir.



Şekil 1. Intel 32 ve 64-bit komut formatı (Instruction format of Intel 32 and 64-bit)

Bir komut kodunun uzunluğu 1, 2 veya 3 bayt olabilir. Bu kodun ardından Şekil 1.'de bayt olarak gösterilen diğer kodlar gelebilir veya gelmeyebilir. Bazı komutlarda iş-kodunun kendisi ModR/M adresleme tablosu içerisinde tanımlanmıştır [11]. Intel işlemci komut kümesi ile ilgili dökümanlar [10,11] incelendiğinde, bir bayt ve iki baytlık komut tablosunun yanısıra üç baytlık komut tablosunun varlığı görülmektedir.

Bir iş-kodunun okunması, gerek makine, gerekse tersine çeviriciler (unassembler) tarafından, bir bayt komut tablosundaki karşılığının bulunması ile başlamaktadır. Yani, değişken uzunluktaki komutların ilk baytlarının karşılığına bir bayt komut tablosundan bakılmaktadır. İki bayt komut tablosuna geçmek için komutun başına 0FH değeri gelmektedir.

2.2. Karşılaşılan Zorluklar (The Difficulties Encountered)

Karmaşık komut kümesine (CISC) sahip bilgisayarlar çok sayıda komut ve adres modundan oluşan bir yapıya sahiptir. Değişken uzunluktaki makine kodlarının sembolik dile çevrimi karmaşık komut yapısından dolayı zordur. Örneğin, "07" makine kodunun sembolik gösterimi "POP ES" iken, "692801D1D2D3" makine kodlarının sembolik gösterimi, "IMUL EBP, [EAX], 0xD3D2D101" şeklindedir. Birinci örnek bir bayttır ve bir operanda sahipken, ikinci örnek altı bayttır ve üç operanda sahiptir. Intel IA-32 komut kümesinde bir mikrokod en az bir bayt en fazla 14 bayt olabilir [22]. Şekil 1.'deki komut formatına göre bir komutta; adresleme ile indisleme baytları kullanıldığı takdirde 16 bit, yani 65536 adet farklı anlam ortaya çıkabilmektedir. Buna ilave olarak, komutlar aynı anda bir veya daha fazla ön-takı alabilir. Sadece 66H, 67H ile altı tane segment değiştirme (segment override prefix) ön takılarının bir komutta kullanılabildiği farz edilirse, adresleme verileri hariç yüz milyonlarca farklı komut yazmak zorunda kalınmaktadır.

Gerçekleştirilen unassembler tasarımında; komut uzunlukları, operand ve adresleme verilerinin sayıları ile uzunlukları tespit edilmiş, geçersiz komutlarda hata verilmiş ve hatanın nereden kaynaklandığı gösterilmiştir.

2.3. Çevrim Tabloları (Conversion Tables)

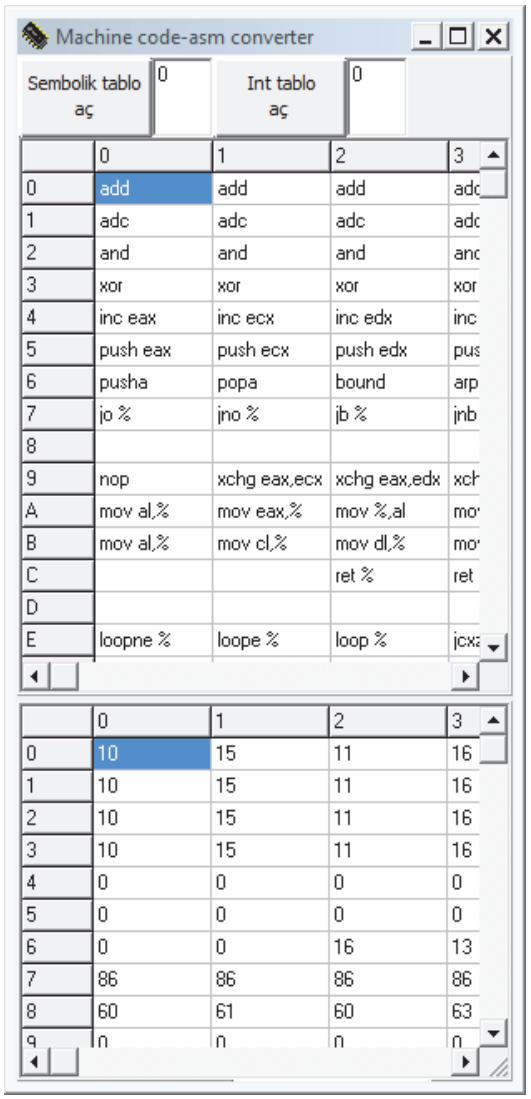
Kod çözme ve sembolik dile dönüştürme için içeriği string ve tamsayı olan 16x16'lık (256 elemanlı dizi) matris tablosu oluşturulmuştur. Yapılan işlem, Intel'in komut kümesi mimarisinin bire bir uygulanması olarak da düşünülebilir. Şekil 1.'deki komut formatındaki her bir baytın 16x16'lık matris tablolarının genel formatları Intel'in ilgili dokümanlarında verilmiştir [10,11]. Ancak, komut kümesindeki tüm komutlara ait alt tabloların, Şekil 2.'de örnek gösterilen formatlara göre yeniden düzenlemesi gereklidir.

Intel'in, komut kümesi mimarisini (Şekil 1.) bayt-bayt tablo mantığı içerisinde yapmış olmasına rağmen, komut kullanım açıklamalarını bu doğrultuda yapmaması, alt tabloların düzenlenmesini ve komutların anlaşılmasını oldukça zorlaştırmıştır. Bir komutun geçerli kullanımı kadar, tanımsız kullanımları da çevrim için önemli bir konudur ve bu konuda bir bilgiye rastlanmamıştır (undocumented) Tabloların olabilecek bütün ihtimalleri karşılayabilecek şekilde doldurulmaları gerekmektedir.

Bazı komutlar tanımsızdır veya belge haline getirilmemiştir. Bu durumda komut kullanımı biliniyorsa tabloya geçirilebilir. Bu çalışmada bazı alt tablolar debugger programı kullanılarak deneme yanılma ile oluşturulmuştur.

Burada önemli olan tabloların bir kere ve doğru bir şekilde hazırlanmasıdır.

Şekil 3.'te "one byte opcode map" tablosunun her bir gözünde komuta ait, komut ön tanımı ile içeriğine bakılacak tablo isimleri gösterilmektedir



Şekil 2. "00h" tanımlı "add" komutu ve alt tablosu ("00h" defined "add" instruction and its subtable)

	0	1	2	3	4	5	6	7
0	ADD Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH ES^{i64}	POP ES^{i64}
1	ADC Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH SS^{i64}	POP SS^{i64}
2	AND Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=ES (Prefix)	DAA^{i64}
3	XOR Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=SS (Prefix)	AAA^{i64}
4	INC^{i64} general register / REX^{o64} Prefixes eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	PUSH^{d64} general register rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA^{i64}/ PUSHAD^{i64}	POPA^{i64}/ POPAD^{i64}	BOUND^{i64} Gv, Ma	ARPL^{i64} Ew, Gw MOVSXD^{o64} Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)

Şekil 3. Tek baytlık iş-kodu haritası (One-byte opcode map) [10, 11]

Bu tablo isimleri başka bir opcode tablosu olabileceği gibi, komut ön-takı tablo isimleri, ModR/M tabloları isimleri ile genişleme (disp) tabloları da olabilir. Ayrıca, her alt tablo başka alt tablolar da içerebilmektedir. Tablonun satır ve sütunları onaltılık tabanda gösterilmiştir. Herhangi bir ızgara gözünün sayısal onaltılık ifadesi önce sütun, sonra satır değerinin birleştirilmiş halidir. Örneğin, POP ES komutu için alt tablo referans verisi 70h dir.

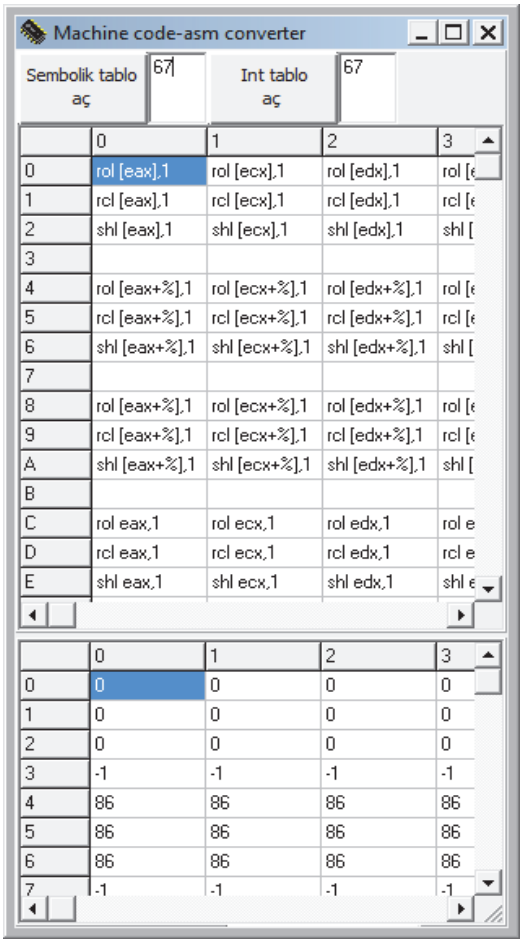
```
#define  hata  -1

#define  op_1   0
#define  op_2   1

#define  seg_es    2
#define  seg_cs    3
#define  seg_ds    4
#define  seg_ss    5
#define  seg_fs    6
#define  seg_gs    7

#define  opd   8
#define  addr  9
```

Şekil 4. Tanım dosyası (Definition file)

Yapılan her string tablosu için bir de sayısal tablolar yapılmıştır. Bu tabloların numaraları program içerisinde ayrı bir başlık dosyasında tanımlanmıştır (Şekil 4.). String tablolar, sembolik gösterimi içerirken, sayısal tablolar içeriğine bakılacak tablo numaralarını göstermektedir. Şekil 5.'te, 67 numaralı sembolik ve sayısal tablonun içeriği gösterilmiştir. Sembolik tablosunda boş olan ızgara gözlerinin sayısal değeri "-1" dir, yani tanımsızdır ve hata durumu söz konusudur. Tablodaki "0" değerleri okuma bitimini, "0" dan büyük değerler ise başka alt tablolardan okumanın devam edeceğini göstermektedir. Program arayüzünde tablo içeriklerinin gösterimi eğitim amaçlı olup, tablo içeriğindeki hataların bulunmasını kolaylaştırmak içindir. Şekil 5'te, "d1h" ile başlayan makine koduna ait sembolik ve sayısal alt tablolar gösterilmiştir.

Örnek:

D100 rol [eax],1 *tablo okuması 86 ncı*
D140 rol [eax, %],1 *tablodan devam edecek*



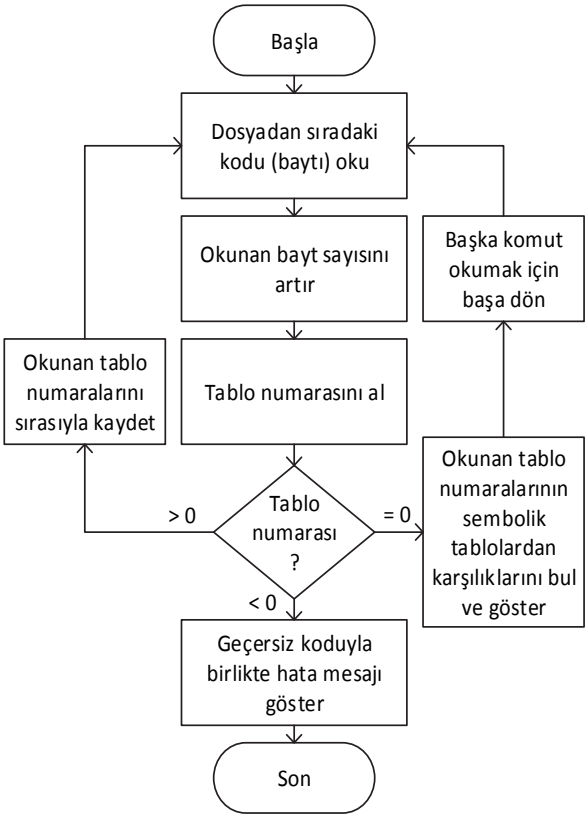
	0	1	2	3
0	rol [eax],1	rol [ecx],1	rol [edx],1	rol [e
1	rcl [eax],1	rcl [ecx],1	rcl [edx],1	rcl [e
2	shl [eax],1	shl [ecx],1	shl [edx],1	shl [
3				
4	rol [eax+%],1	rol [ecx+%],1	rol [edx+%],1	rol [e
5	rcl [eax+%],1	rcl [ecx+%],1	rcl [edx+%],1	rcl [e
6	shl [eax+%],1	shl [ecx+%],1	shl [edx+%],1	shl [
7				
8	rol [eax+%],1	rol [ecx+%],1	rol [edx+%],1	rol [e
9	rcl [eax+%],1	rcl [ecx+%],1	rcl [edx+%],1	rcl [e
A	shl [eax+%],1	shl [ecx+%],1	shl [edx+%],1	shl [
B				
C	rol eax,1	rol ecx,1	rol edx,1	rol e
D	rcl eax,1	rcl ecx,1	rcl edx,1	rcl e
E	shl eax,1	shl ecx,1	shl edx,1	shl e

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	-1	-1	-1	-1
4	86	86	86	86
5	86	86	86	86
6	86	86	86	86
7	-1	-1	-1	-1

Şekil 5. Sembolik ve sayısal tablolar (Symbolic and numerical tables)

2.4. Tersine Çevirici Program (Unassembler Program)

Tablolama mantığının akış şeması Şekil 6.'da gösterilmiştir. Bu şemada; değişik uzunluktaki bir komutun okunması yapılmakta ve her tablo içi okumada karşılaşılan değerlerin neticesine göre hareket edilmektedir. Değer ifadesi tablonun içerik değeridir.



Şekil 6. Tersine çevirici programı akış şeması (Flowchart of unassembler program)

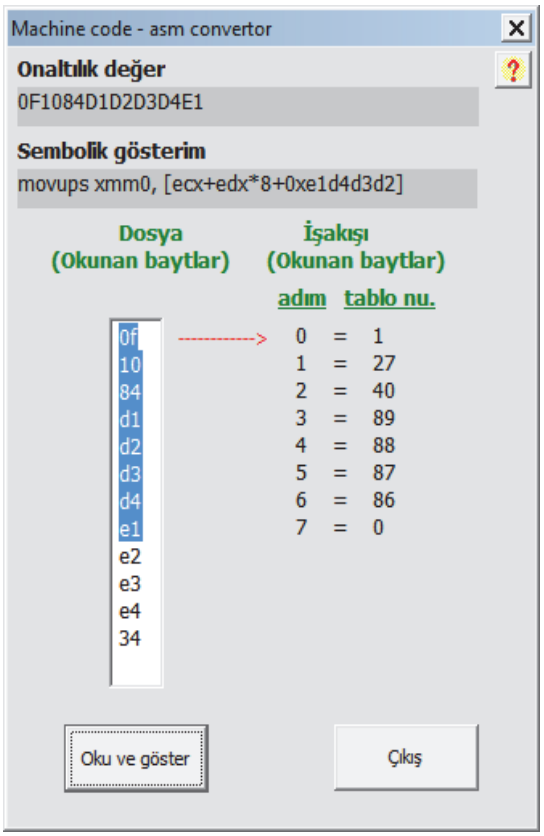
Program, dosyadan herhangi bir kodun (baytın) okunması (Şekil 7.'de Dosya (okunan baytlar)) aşamasında;

- "0" dan büyük bir değerle karşılaşırsa; komut okuması devam etmektedir, sıradaki bayt okunur,
- "0" değeriyle karşılaşırsa; komut okuması doğru bir şekilde bitmiştir, string olarak gösterimini yapar ve yeni bir komut okumak için başa döner,
- "0" dan küçük bir değerle karşılaşırsa; komut okuması yanlıştır ve hata durumu vardır. Hata mesajı verilir ve programdan çıkılır.

Akış şemasındaki iş akışı ve program arayüzü Şekil 7.'de görülmektedir.

Şekil 6.'daki çevirici programın akış şemasına göre yazılan program kodları, değişken uzunluklardaki makine kodlarını okumak için yeterlidir. Ancak, kodların anlaşılabilmesi için okunan bilgilerin formatlı bir şekilde gösterimi gerekmektedir.

Formatlı gösterim, sayısal tabloların bire bir karşılığı olan sembolik tabloların içerik ifadelerinin string olarak birleştirilmesinden ibarettir. Bu birleştirmenin nasıl yapıldığı formatlı gösterim bölümünde açıklanmıştır. Programda tabloların girilen veriye göre okunması yapılmaktadır. Komutu oluşturan veriler bayt bayt okunmaktadır



Şekil 7. Program arayüzü (Program interface)

Zira komut tasarımında bir bayt'tan daha az bir komut yoktur. Diğer önemli bir nokta ise, bir sonraki komut verisi önce gelen veriye bağımlıdır ya da bir komut verisi sonraki verinin ne olabileceğini tayin etmektedir.

2.4.1 Adresleme Verileri (Addressing Data)

Intel komut kümesinde tanımlanan komutlar, adresleme verileri ile sabit veya değişken veriler alabilirler.

add al,12H ; AL 8-bitlik bir kaydedicidir

add ax,12H ; AX 16-bitlik bir kaydedicidir

mov eax, 12H ; EAX 32-bitlik bir kaydedicidir

mov [eax+disp8], 12H ; komutunda bir adet 8 bitlik ve bir adet 32 bitlik veri vardır

mov [eax+disp32], 12H ; komutunda iki adet 32 bitlik veri vardır.

Makine kodunu sembolik dile çevirirken bu veriler, aşağıda ifade edildiği şekilde, daima komut dizisinin sonunda yer alır ve ters sırada yazılır.

C7803412000012000000
mov [eax+00001234H],00000012H

C744141200001200
mov [esp+edx+12H], 00120000H

Altı çizgili olan sayılar, sembolik dile çevrilirken ters sırada yazılmıştır. Bu verilerin boyutları farklı olabilmektedir. Her komut değişik uzunlukta bir veya iki adet veri alabilir. Gerçekleştirilen programda bu verilerin boyutları ile kendileri ayrı ayrı tespit edilmiştir. Bu maksatla her ihtimal için ayrı ayrı tablolama yapılmıştır.

Veri okuma tabloları içeriği onaltılık tabandaki ifadelerdir. Bu sayede, ayrıca onaltılık tabanda dönüşüm yapmaya gerek kalmamaktadır.

Tablo içerikleri bir sonraki tablonun numarasını içerir, son tablonun içeriği ise okumanın sonlanması için "0" yapılır. Şekil 8.'de okunan tablo numaraları görülmektedir. Tanımsız komutlar için eksi değerli hata kodları verilir. Her komut için okuma farklı olmakla beraber örnek bir okuma sırası aşağıda verilmiştir.

Bir adet 8 bit + bir adet 32 bit okuma;

disp8_32 tablosu sadece 8 bit okuma içindir.

disp32 tablosu 32 biti 4 aşamada okur.

Opcode→regR/M→SIB→disp8_32 → disp32→ disp24→ disp16→ disp8→0

Eğer iki adet 32 bit okunacaksa;

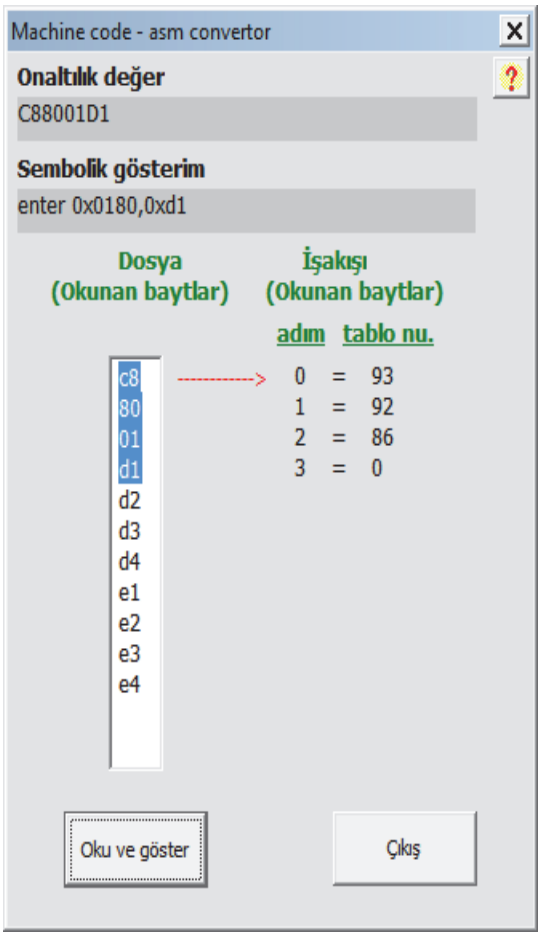
Opcode→regR/M→SIB→disp32 →disp24→ disp16→ disp8→disp32 →disp24→ disp16→ disp8→ 0

Toplamda iki adet 32 bit okunmuştur. Burada "disp32" tablosu iki defa kullanılmıştır.

Yukarıdaki örneklerde gösterilen "disp" okumaları için hazırlanan tablolar ayrı ayrı olmalı ve yeteri kadar yol tanımı yapılmalıdır.

2.4.2. Formatlı Çıkış (Formatted Output)

Bir komutun okunmasının bitmesi ile beraber, okunan tabloların içeriğinden sembolik karşılıkları elde edilir. Formatlı gösterim için bu ifadelerin birleştirilmesi gerekmektedir. Sembolik gösterim tablolarında bir ifadenin başka bir ifadedeki yeri "%" karakteri ile sembolize edilmiştir. Bu veri sayı olabileceği gibi, string ifade de olabilir.



Şekil 8. Tablo okuma sırası (Table reading order)

"mov eax, %" şeklindeki ifadede "%" karakteri yerine bir veri gelme durumu söz konusudur.

"*mov [%],%"* ifadesinde "%" karakterleri yerine birer adet veri gelmesi söz konusudur.

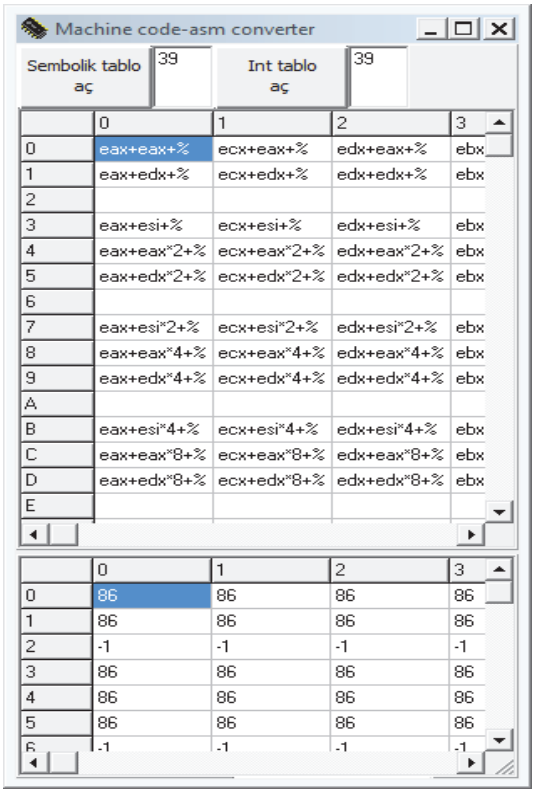
add + [eax+%], ecx + 78H
↓
add [eax+78H], ecx

Yukarıdaki ifadenin formatlı bir şekilde gösterilmesi için "%" karakteri yerine 78H disp verisi gelir ve "%" karakteri silinir.

Programın kod bölümündeki bir tablonun;

- Disp almış mı, almışsa kaç tane,
- SIB taban indisleme almış mı,
- SIB taban indisleme almışsa, disp almış mı,
- Komut ön takı baytı almış mı, almışsa, hangi grup ya da gruplar

durumları kontrol edilmelidir. Örnek olarak, bir komut Şekil 9.'daki gibi SIB baytı kullanmışsa; Disp almayabilir veya Disp8 ya da Disp32 alabilir.



	0	1	2	3
0	eax+eax+%	ecx+eax+%	edx+eax+%	ebx
1	eax+edx+%	ecx+edx+%	edx+edx+%	ebx
2				
3	eax+esi+%	ecx+esi+%	edx+esi+%	ebx
4	eax+eax*2+%	ecx+eax*2+%	edx+eax*2+%	ebx
5	eax+edx*2+%	ecx+edx*2+%	edx+edx*2+%	ebx
6				
7	eax+esi*2+%	ecx+esi*2+%	edx+esi*2+%	ebx
8	eax+eax*4+%	ecx+eax*4+%	edx+eax*4+%	ebx
9	eax+edx*4+%	ecx+edx*4+%	edx+edx*4+%	ebx
A				
B	eax+esi*4+%	ecx+esi*4+%	edx+esi*4+%	ebx
C	eax+eax*8+%	ecx+eax*8+%	edx+eax*8+%	ebx
D	eax+edx*8+%	ecx+edx*8+%	edx+edx*8+%	ebx
E				

	0	1	2	3
0	86	86	86	86
1	86	86	86	86
2	-1	-1	-1	-1
3	86	86	86	86
4	86	86	86	86
5	86	86	86	86
6	-1	-1	-1	-1

Şekil 9. SIB baytı tablosu (SIB byte table)

Aşağıda kod bölümüne yazılan komutta, SIB tablosuna girildi ise, "int sib_var" değerine, okunan bayt sayısı değeri verilerek ikinci bir defa bu gruptaki tabloların okunması ve "sib_var" içeriğinin değişmesi engellenmekte, ayrıca kaçıncı okuma baytında hangi SIB tablosuna girildiği tespit edilmektedir.

```
if(sib_varmi>=0) goto sib_var;

if(tablo==sib_0)   {sib_varmi=adim;}

if(tablo==sib_8)   {sib_varmi=adim;}

if(tablo==sib_32) {sib_varmi=adim;}

sib_var:
```

Program çalışması esnasında komut ancak doğru bir şekilde okunmuş ise formatlı çıkış yapılmaktadır. Diğer gruplar için de benzer işlemler yapılır. Bu şekilde tek bir karakter kullanılarak hem komutlar sembolik olarak tanımlanır, hem de istenilen şekilde formatlı çıkış elde edilir.

3. SONUÇLAR VE TARTIŞMA (RESULTS AND DISCUSSION)

Gerçekleştirilen unassembler yazılımıyla, değişken uzunluktaki makine kodlarının sembolik dile çevrimi başarı ile yapılmıştır. Intel 32-bit korumalı mod komut kümesindeki, 66H, 67H, F2H, F3H tanımlı komut ön takıları haricindeki bütün komutlar tamamıyla uygulamaya dâhil edilmiştir.

Okunan her baytın mutlaka bir tabloda karşılığı vardır. Bir bayt 256 adet olasılık içerdiğinden, 256 adet elemandan oluşan sembolik tablolardan uygun tabloyu bulmak için sayısal tablolar oluşturulmuştur. Uygulama, makine komutlarını bayt bayt okumakta ve okunan her baytın tabloda içerdiği sayısal değer bir sonraki okunacak tabloyu göstermektedir. Bu sayede sembolik gösterim ifadeleri birleştirilerek anlamlı hâle getirilmiştir.

Çalışmada 105 adet tablo ve bu tabloların okunması için bir adet prosedür kullanılmıştır (Bu sayılar komut kodlarının çözümü için yeterli olmuştur). Eksik kalan kısımların eklenmesi veya uygulamaya üç bayt komut tablosunun geçirilmesi kolay, ancak dikkatli bir çalışmayı gerektirmektedir. Bu sebeple tablolar doğru bir şekilde hazırlanmalıdır.

Bu çalışmada gerçekleştirilen unassembler yazılım uygulamasının yanısıra; sembolik dilden makine kodlarının üretilmesi (assembler), assemblerin eğitim amaçlı kullanılması, sanal simülatör ve debugger (hata ayıklayıcı) gibi yazılım tersine mühendislik aracı olarak kullanılabilir olması elde edilen diğer çıktılardır.

3.1. Sembolik Dilden Makine Kodlarının veya Tersinin Üretilmesi (Generation of Codes From Symbolic Language to Machine Codes or Vice Versa)

Daha önce ifade edildiği gibi, tabloların okunmasında temelde iki adet dizi tanımlanmıştı.

AntiString *sembolik [tablo_sayısı][256];*

Int *tablolar_int [tablo_sayısı][256];*

Makine kodları bayt bayt okunmakta, her okunuştaki bayt değeri *string* tablolarının içeriğini yani makine kodunun sembolik karşılığını vermektedir. String olarak yazılmış bir assembly dili (asm) ifadesinin makine kodlarına dönüşümü ise, aynı yöntemle yapılabilir.

Yalnız buradaki "asm" ifadesi, sembolik tablonun içeriğine bakılarak bulunurken, tablonun kaçıncı elemanı olduğu makine kodu değerini göstermektedir.

Genelde bir çevirme işlemi; sözel analiz (lexical analysis), yazım hatası analizi (syntax analysis) ve kod üretimi (code generation) adımlarından oluşmaktadır. Uygulanan yöntemle, sözel analizin, yazılım hatası analizinin ve kod üretiminin basit ve hatasız bir şekilde yapılabilir olduğu görülmüştür. Gerçekleştirilen assembler ve unassembler, programcıya, yazılım analizi ve yazılımın arkasında yatan mantıkla birlikte, işlemci mimarisinin derin işlevlerinin tersine mühendislik açısından sınamasına imkân sağlamıştır.

3.2. Uygulamanın Assembler Eğitiminde Kullanılması (Using the Application in Assembler Training)

Projede kullanılan tablo numaraları ile tablo içi değerleri referans olarak kullanılarak yardım dosyası oluşturulabileceği gibi, programın çalışması esnasında okunan tablolar ve bu tablolarının içeriği bilindiğine göre, her durumda farklı bir yorum yaparak bilgi veren bir program olarak da kullanılabilir.

```
string yardim[tablo_sayısı][256];
```

şeklinde bir dizi oluşturulurken derleyici, her bir eleman için 256 bayt yer ayırır. Tablonun her elemanı için 256 bayt'a kadar açıklama içeren mesajlar yazılır. Bu sayede bir komutun ne iş yaptığı, kullanımı, geçerli bir komut olup olmadığı kolaylıkla ifade edilebileceğinden, assembly dili eğitiminde yardımcı program olarak kullanılabilecektir.

3.3. Uygulamanın Sanal Simülatör veya Debugger Olarak Kullanılması (Using the Application as a Virtual Simulator or Debugger)

Bu çalışmada, makine kodlarının uzunlukları ve bayt bayt ne işe yaradıklarının tespiti yapılabilmekteydi. Okunan komut verilerinin yorumlanması, ara dil oluşturulması ve yapılan hesaplamaların gösterilmesi sanal olarak yapılabilir. Ancak, okunan her tablo ve tablo içeriği için yorumlayıcı bir fonksiyonun yazılması gerekir. Her tablo 256 elemandan oluştuğu için yazılacak fonksiyonların sayısının normal olarak her tablo için 256 adet olması gerekir. Ancak, ortak kullanımlar birleştirilerek fonksiyon sayısı en aza indirilir. Okunan makine komutlarının mikroişlemciye tek tek yaptırılması da mümkün olabilir. Her komut icrasından sonra kaydedici değerleri okunarak monitörde gösterilebilir. Her komut icrasında trap bayrağı etkin hâle getirilerek kaydedicilerin durumu görülebilir. Çalışmada kullanılan modelin, yukarıda açıklanan yazılım geliştirme araçlarından birisi kullanılarak, özellikle işlemci mimarisinin fonksiyonlarının test edilmesinde ve anlaşılmasında önemli bir rol oynadığı, yapılan uygulama sonuçlarından gözlenmiştir.

4. SONUÇLARIN DEĞERLENDİRİLMESİ (CONCLUSION)

Yazılım sistemlerinin tersine mühendisliği, geleneksel olarak yüksek düzeyli kodlardan veya veri tabanlarından elde edilen yüksek düzeyli bir soyutlama veya tanımlama oluşumunun tam merkezindedir. Bu çalışmada, çalışabilir makine kodlarını sembolik dile veya tersine dönüştüren, güncellemeye açık, kullanımı kolay, tablo tabanlı satır-içi kod çeviren tersine mühendislik araçlarından bir unassembler gerçekleştirilmiştir. Bu araçlar, bilgisayar sistemlerine yönelik yazılan çalıştırılabilir programların arkasında yatan mantığın analizi sürecinde ve bilgisayar donanımının işlevlerinin test edilmesi ve çalışmasının anlaşılmasında yardımcı olmaktadır.

Bu çalışmanın sonucunda; IA-32 komut kümesi mimarisi ortaya çıkartılarak, gerek makine kodundan sembolik dile, gerekse sembolik dilden makine kodları üretimine imkân veren, güncellemeye açık, kullanımı kolay, düşük düzeyli bir çevirici elde edilmiştir. Uygulama sonuçları, Borland Turbo Debugger ve DOS Debug yazılımları ile test edilmiş ve doğrulanmıştır. Bundan sonraki çalışmada, IA-64 işlemci mimarisine dâhil üç baytlık komut tablolarının oluşturulması ve bunlara dayalı assembler ve unassembler araçları, komut ön-takıları da dâhil edilerek gerçekleştirilebilir.

Uygulanan yöntem; içerisinde farklı boyutlarda komutları barındıran komut kümelerine sahip değişik mikroişlemcilere (CISC) ait assembly dillerinin ifade edilebileceği ve ortak bir yapı olarak kullanılabileceği gibi, sabit uzunlukta komut kümesi mimarisine sahip mikroişlemciler (RISC) için de kullanılabilir. İlk yöntemde tablo adedi değişken iken, diğer yöntemde sayı hep aynı kalmaktadır.

Tanımlanan tabloların sayısının, IA-64 mimarisi de dâhil edilecek olursa, 256 adede kadar olmasının yeterli olacağı varsayılmıştır. Bu durumda sayısal tabloların toplam boyutu en fazla 64KB olacaktır (her tablo 256 bayt). Mikroişlemci kod çözme biriminin ROM belleği olarak kullanılması durumunda tabloların örnek olarak matematik işlemci veya MMX işlemci üniteleri için parça parça kullanılması da mümkündür.

Çalışmada kullanılan modelin yazılım geliştirme araçlarından birisi olan makine kodlarının dönüşümü veya derleyici araçlarının tasarlanmasında ve geliştirilmesinde kullanılabilir olduğu görülmüştür.

KAYNAKLAR (REFERENCES)

1. Chikofsky, E.J. ve Cross J.H., "Reverse Engineering and Design Recovery: A Taxonomy", **IEEE Software**, Cilt 7, No 1, 13-17, 1990.
2. Younis, M.B, ve Tutunji T., "Reverse Engineering Course at Philadelphia University in Jordan", **European Journal of Engineering Education**, Cilt 37, No 1, 83-95, 2012.
3. Ali, M.R., "Why Teach Reverse Engineering", **ACM SIGSOFT Software Engineering Notes**, Cilt 30, No 4, 1-4, 2005.
4. Arbone, C., Ditu, B., Craciun, S. ve Badea, D., "Model-Driven Inline Assembler Generator for Retargetable Compilers", **Control Systems and Computer Science (CSCS), 2013 19th International Conference on Digital Object Identifier**, Bucharest, Romania, 71-76, 2013.

5. Golden, G. R., "A Highly Immersive Approach to Teaching Reverse Engineering", **Proceedings of the 2nd Conference on Cyber Security Experimentation and Test**, Berkeley, CA, USA 1-1, 2009.

6. Paleari, R., Martignoni, L.M., Roglia, G.F. ve Bruschi, D., "N-version Disassembly: Differential Testing of x86 Unassemblers", **Proceedings of the 19th International symposium on Software testing and analysis (ISSTA)**, Trento, Italy, 265-274, 2010.

7. Jamthagen, C., Lantz, P. ve Hell, M., "A New Instruction Overlapping Technique for Anti-disassembly and Obfuscation of x86 Binaries", **Proceedings of the 2013 IEEE Workshop on Anti-malware Testing Research (WATER'13)**, Montreal, Canada, 25-33, 2013.

8. Topaloğlu, N. ve Gürdal, O., "A Highly Interactive PC Based Simulator Tool for Teaching Microprocessor Architecture and Assembly Language Programming", **Journal of Elektronika ir Elektrotechnika**, Cilt 98, No 2, 53-58, 2010.

9. Cifuentes, C., "An Environment for the Reverse Engineering of Executable Programs", **2nd Asia Pasific Software Engineering Conference (APSEC'95)**, Washington, DC, USA, 410-419, 1995.

10. İnternet: Intel Corporation, "IA-32 Intel® Architecture Software Developer's Manual", **Intel Corporation**, Denver, USA, http://flint.cs.yale.edu/cs422/doc/24547112.pdf 2003.

11. İnternet: Intel Corporation, "Intel® 64 and IA-32 Architectures Software Developer's Manual", **Intel Corporation**, Denver, USA, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32 architectures-software-developer-manual-325462.pdf , Bölüm 3, 2013.

12. İnternet: The Netwide Assembler: http://www.nasm.us/, 2013.

13. Ma, W., "Design and Testing of a CPU Emulator", Texas A&M University, **Microsoft Research Technical Report**, United States of America, 1-12, 2009.

14. Hsieh, W., "Reverse-Engineering Instruction Encodings", **2002 USENIX Annual Technical Conference**, Monterey, USA, 133-145, 2002.

15. Paleari, R., "N-version Disassembly: Differential Testing of x86 Unassemblers" **International Symposium on Software Testing and Analysis (ISSTA 2010)**, Trento, Italy, 1-10, 2010.

16. Fadıl, S., "Mikroişlemciler için Genelleştirilmiş Assembly Derleyici" **Elektrik-Elektronik-Bilgisayar Mühendisliği 10. Ulusal Kongresi**, İstanbul, 523-526, 2003.

17. Payer, M., "Fast Binary Translation: Translation Efficiency and Runtime Efficiency", **2nd Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT'09)**, Austin, Texas, USA, 2009.

18. Sridhar, S., Shapiro, J.S. ve Bungale, P.P., "HDTrans: A Low-overhead Dynamic Translator", **SIGARCH Computer Architecture News,** Cilt 35, No 1, 135–140, 2007.

19. Chen, Wen-Ke, Lerner, S., Chaiken, R. ve Gillies, D.M., "Mojo: A Dynamic Optimization System", **In ACM Workshop Feedback-directed Dyn. Opt. (FDDO-3)**, 2000.

20. Olszewski, M., Cutler, J. ve Steffan, J.G. "Judostm: A Dynamic Binary-rewriting Approach to Software Transactional Memory". **In PACT '07**, Washington DC, USA, 365–375, 2007.

21. İnternet: www.pcsistem.net/unassembler,2013.

22. Brey, B.B., "The Intel Microprocessors: Architectures, Programming and Interfacing" (Eighth Edition), **Prentice Hall**, USA, June 2008.