

BİLGİSAYARLAR İÇİN BİR ARA DÜZEY DİLİ: ASSEMBLY DİLİ

Toygun S. BAŞOL

ÖZET

Bu yazıda bilgisayar dilleri arasında ayrıcalıklı bir yeri olan (Assembly) dili incelenmektedir.

SUMMARY

In this article, an Assembly Language which takes a privileged place amongs the computer programming language is discussed.

1 - GENEL

insanlar arası karşılıklı anlaşmayı sağlayan bir dil olduğu gibi insanlar ile bilgisayarlar arasında da karşılıklı anlaşmayı sağlayacak olan gene bir dil olacaktır, insanların aynı nesneyi anlatmak amacı için kullandıkları çok çeşitli diller olduğu gibi bilgisayarların da aynı nesneyi işlemek amacı için kullandıkları değişik diller vardır, insanların kullandığı dilin ana öğeleri abece (alfabe) ise bilgisayarların da kullandıkları dilin iki temel öğesi vardır. Bunlar 'T' ve '0' dir. Nasıl insanlar aynı abece ile değişik dilleri anlayabilmektedir, insanların konuştukları dilin bir kural takımı olduğu gibi bilgisayarların anladığı dilin de bir kural takımı vardır, işte bu 'T' ve '0' ların uygunca bir araya gelmesinden kaynaklanan kurallar başka bir deyimle "ikili amaç düğüm" (Binary Object Code) lar yardımı ile bilgisayara istenen bir iş yaptırılabilir. Uygun "ikili amaç düğümlerinin" arka arkaya gelmesiyle bir "Amaç program" elde edilir. Böylece belirli bir olaya özgün sorunlar belirli bir "Amaç program" ile çözümlenir.

Anlaşıldığı gibi bir "Amaç program" "1" ve "0" lardan oluşacaktır. Bu "1" ve "0" lar kullanarak belirli bir programı yapmak programcı açısından çok zor olduğu gibi bu programın hata yüzdesi de çok fazla olacaktır. Bu yüzden makine dilinde yazılmış bir program elverişli değildir. Ayrıca böyle yazılmış olan bir programın başka bir kimse tarafından göz-

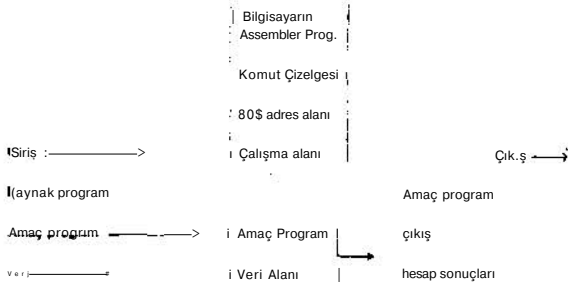
den geçirilmesinde de pratik zorluklar vardır. Bu noktada daha kolaylıkla yazabileceğimiz ve okuyabileceğimiz bir programın kuralları ile inşa edilmesi söz konusudur. Buna "Source Program" veya "Kaynak Program" denilir. İstediğimiz amaca uygun birçok kaynak program türü vardır. Bunlardan birkaçı FORTRAN, BASIC, COBOL, PL, PASCAL ve ASSEMBLY gibidir. Assembly hariç saydıklarımızın hepsi yüksek düzeyli dillerdir. İnsanların anladığı dil ile bilgisayarın anladığı dile çevirmek, yani yazılmış olan bir Kaynak Programı bir Amaç Programa dönüştürmek için özel bir program kullanılır. Buna Dil Dönüştürücü denilir. Söz gelimi FORTRAN ve COBOL için yazılmış olan dil dönüştürücü programa COMPILER, BASIC için yazılmış olan dil dönüştürücü programa INTERPRETER, ASSEMBLY için yazılmış dil dönüştürücü programa da ASSEMBLER denir. Doğal olarak Assembler programının da kendine özgün özellikleri vardır.

Yüksek düzeyli dil, doğası gereği düşük düzeyli dilin gereksiniminden çok daha fazla makine kodu adımlarına gerek duyar. Bu ise daha fazla bellek kullanılmasını gerektirir. Bilgisayar tarafından edimlenen (performed) programın adımlarının çokluğu edimlenen işin gerçek zamanını arttırır. Oysa Assembly dili düzeyi Makine Diline çok yakındır. Aynı bir iş için Assembly Dilinin kullanılması gerekli bellek miktarını azaltacaktır. Dolayısıyla Yüksek düzeyli bir dil ile karşılaştırıldığında uygulanan programın edimi daha hızlı olacaktır. Bu ise zaman ve para tutumluluğunu beraberinde getirecektir.

Assembly dili bilgisayara bağımlıdır. Bilgisayardan bilgi sayarak değişik özellikler gösterir. Sözelimi aynı bir işin değişik bilgisayarlardan edimlenmesi için değişik kaynak Programlara gereksinim duyulur. Bu derece seçenekleri bol olan ve makineden makineye değişen bir dil aşağıda genel özellikleri ile anlatılacaktır. Örneklemelerden ise kullanılan bilgisayar cinsleri belirtilecektir.

2. ASSEMBLY SÜRECİ

Assembly dilinde yazılmış bir programın edimi, makine dilinde yazılmış olan ve peş peşe gelen KOMUTLARIN işlenmesi ile olanaklıdır. Komutlar, ikili düğüm biçiminde bir KOMUT ÇİZELGE'sinde bellegin içinde belirli yerlerde saklı tutulurlar. Her komutun insan belleğini çağrıştıran bir simgesi vardır. Dolayısıyla kaynak program veya Assembly bir simgeler dizisidir. Bu simgeler dizini Assembler tarafından okunur, komut çizelgesi ile karşılaştırılır ve bellekteki belirli bir simgesel adresten itibaren Amaç program haline dönüştürülmesini yükleyici (Loader), veya Bağlayıcı (Linker) diye çağrılan bir program üstlenir. Assembly sürecini Çizim 2-1'deki gibi düzenleyebiliriz.



Çizim 2-1 Assembly Süreci Şeması

Assemblerin bu dönüştürmesi sırasında, kaynak programdaki kimi biçimsel hataları görmesi, ortaya çıkartması ve hatalar ile bulundukları satırları işaretlemesi ve bir hata listesi üretmesi, ayrıca programı listeleri sağlanır. Ek olarak, bazı assembler programında kullanılan tüm simgeler için bir çapraz ilgi (cross reference) listesi sağlarlar. Bütün bu işlerin becerilebilmesi için Assembler'in kaynak programı bir veya birden fazla okuması gerekir. Tek geçişli (one-pass assembler) 'de, akıllı bir programlama tekniği aracılığı ile program tek bir tarama ile elde edilir. İki geçişli (Two-pass assembler)'de ise olgu iki aşamalı bir tarama ile tamamlanır.

3- BİR ASSEMBLY 'DEYİMİNİN ÖĞELERİ VE ÖZELLİKLERİ

Programcı her istenen komut için, Kaynak Programın her satırına, bir DEYİM (STATEMENT) ile onu izleyen bir satır sonlandırıcısını yazar. Komutun tam isminin yazılması yerine bellekte yer edebilecek bir kısaltılmış biçimi yazılır. Buna MNEMONIC denilir. Bu kısaltılmış biçimler komutun ismine bağlı olarak 3 veya 4 anahtar harften oluşurlar. Bu anımsamayı kolaylaştırır, yazmayı basitleştirir. Her deyim 4 alandan oluşur. Bunlar sırasıyla, Etiket Alanı (Label),

İşlem Kodu Alanı (operation Code), işlenen Alanı (Operand) ve Yorum Alanı (comment) dır.

Bunlar yukarıda yazıldığı sıra dizilişinde DEYİM içinde yer alır, DEYİM'in alanları birbirinden bir veya daha fazla boşluk ile ayrılır. Deyim sonu işareti ile DEYİM sonlandırılmış olur. Böyle bir DEYİM Çizim 3-1'de görülmektedir. Bir deyim de görünen

Etiket alanı/İşlem kodu alanı/islenen alanı/Yorum alanı..
Çizim 3-1 Bir Assembly ifadesi

Damgalar (Characters) şunlardır. A'dan Z'ye abecesel damgalar, 0 'dan 9'a sayısal damgalar, + = .- ve boşluk gibi işlemler (operatör) bir deyim değeri programın yerleştirilmesinden etkilenmiyorsa o deyim SALTİK (abseute) tır. Bir saltık programda bütün deyimler saltıktır. Bir deyim değeri yerele göre değişiyorsa, o deyim yerdeğişir (RELOCATABLE) dir.

3-1 ETİKET ALANI

Etiket komutun kimliğini saptar ve programdaki diğer komutlar tarafından bir dayanak noktası kullanılır. Etiket Alanı - Deyimin ilk konumundan hemen sonra başlar. İlk konum bir önceki deyim sonu işaretini izler. Etiket Alanı boş ise deyim etiketlenmemiştir. Etiket simgesel olmalıdır. Etiket Alanında alfa-sayısal damgalar bulunabilir. Etiket değeri program sayacının değerine bağlı olarak saltık veya yerdeğişir olabilir. Her etiket program içinde tek olmalıdır. İki veya daha fazla komutta aynı simgesel isim gözükmemelidir. Çizim 3-2'de Etiket Alanı ile ilgili örnek bulunmaktadır.

AEDTU	Geçerli Etiket
ABC45	Geçerli Etiket

Çizim 3-2 Etiket Alanı örneği

3-2 İŞLEM KODU ALANI

İşlem Kodu, Assembler tarafından edimlenecek olan işlemi kısa ad (mnemonic) olarak tanımlar. İşlem Kodu Alanı, Etiket Alanını izler. İki alan arasında hiç olmazsa bir boşluk bulunur. Etiket yoksa, işlem kodu ikinci konumdan hemen sonra başlayabilir. İşlem Kodu bir boşluk ile sonlanır. İşlem Kodları birçok sınıflamada düzenlenebilir. Bunları genellikle aşağıda anladığımız gibi MAKİNE İŞLEM KODLARI ve YAPAY İŞLEM KODLARI olmak üzere iki ana dalda inceleyebiliriz.

3-2-1 MAKİNE İŞLEM KODLARI

Bilindiği gibi kaynak programındaki her Deyim Assembler Tarafından Amaç Programında bir makine işlemine karşılık getirilir. Makinenin yaptığı işlemlere göre bu komutları aşağıdaki gibi sınıflayabiliriz:

a) **BİLGİ AKTARMA KOMUTLARI** : Bilgiyi yazmaçlar (registers) arasında veya yazmaçlar ile bellek arasında veya bellek ile yazmaçlar arasında aktaran işlem kodlarının komutları kalkış ve varış noktaları komut içerisinde belirtilir. Z-80 mikro-işlemcisinde, örnek olarak LOAD ve EXCHANGE işlem kodlarını gösterebiliriz. Bunların mnemonic yazılışları LD ve EX dir. Komut içinde kullanılışları ise LD r, s ve EX, DE, HL biçimindedir. PUSH ve POP işlem kodları ile Yığıt Göstergesi (Stack Pointer) isimli yazmaca girilip çıkılabilmektedir.

b) **ÖBEK (block) AKTARMA KOMUTLARI** : Belleğin herhangi bir büyüklükteki bir öbeği gene belleğin diğer bir yerine tek bir komut ile taşınabilir, örneğin Z-80 mikro-işlemcisinde LDIR ve LDDR işlem kodları ile bu iş yapılabilir. İşlem kodlarının açık yazılışları LOADINCREMENT and REPEAT ile LOAD DECREMENT and REPEAT dir. İşlemin yapılması şöyledir. HL yazmaç çiftinin belirttiği adresteki bilgi DE yazmaç çiftinin belirttiği adrese aktarılır. Her iki yazmaç çiftinin değerleri 1 arttırılır. Bu işin kaç kere yapılacağını ise BC yazmaç çiftinin değeri belirler. BC yazmaç çiftinin içeriği sıfır olana kadar işlem devam eder. LDDR komutunda ise, her üç yazmaç çiftinin değeri birer eksiltilir. BC sıfır olana kadar olaya devam edilir.

c) **ARİTMETİK İŞLEM KOMUTLARI** : Yazmaçlardaki veya bellekteki bilgiyi Birikeç (Accumulator) bilgiyle toplayan, çıkartan arttıran veya eksiltten işlem kodlarının kümesidir, örnek olarak Z-80 mikro-işlemcisindeki, INC, DEC, ADD, ve SUB işlem kodlarını gösterebiliriz. Bu işlemler 16 bit üzerine yapıldığı taktirde birikeç kullanılmamaktadır. Kısaad (Mnemonic) olarak 8 bitlik bir toplama işlemi ADD A, r: 16 bitlik bir toplama işlemi ise ADD HL, ss şeklinde gösterilebilir. Bu gösteriş biçimindeki bir yazmacı, ss ise bir yazmaç çiftini belirtmektedir. Doğal olarak A birikeçi HL ise Yazmaç çiftini göstermektedir.

d) **MANTIKSAL İŞLEM KOMUTLARI** : Yazmaçlardaki veya bellekteki bilgiye çeşitli mantıksal işlemleri yaptırtan işlem kodlarının kümesidir, örnek olarak Z-80 mikro-işlemcisinde AND, OR, EX-OR, NEG, CPL ve CP işlem kodlarını gösterebiliriz. Bütün mantıksal işlemler birikeçte yapılmaktadır. Kısaad olarak CP s komutunun anlamı, s yazmacındaki bilgiyi birikeç ile KARŞILAŞTIR 'dır

e) **DÖN (ROTATE) VE KAY (SHIFT) KOMUTLARI**: Yazmaçlardaki veya bellekteki veya birikeçteki bilginin sağa veya sola döndürülmesi veya kaydırılması bu grubdaki işlem kodları ile olanaklıdır, örnek olarak Z-80 mikro-işlemcisinde RLA-RRA, SLAs ve SRAs işlem kodlarını gösterebiliriz. Buradan verilen ve verilmeyen bu işlem kodlarının geniş bir uygulama alanı tam sayılı çarpma ve bölme işlemleridir.

f) **BİT AYARLAMA (MANIPULATION) KOMUTLARI**: Bir yazmaç yada bir bellek yerelindeki herhangi bir 'bit' i sınamayı sağlayan, o biti "1" veya "0" yapabilen işlem kodlarının kümesidir, örnek olarak Z-80 mikro-işlemcisinde BİT, SET ve RES işlem kodlarını gösterebiliriz, örneğin, Bit B, r komutu r yazmacından b ninci biti sına anlamına gelir. Aynı biçimde SET b, r ise, r yazmacının b ninci bitini 1 yap anlamına gelir.

g) **DALLANDIRMA, SAPMA KOMUTLARI**: Assembly programındaki değişik yereller arasından bilginin koşullu veya koşulsuz olarak aktarılması bu kümedeki işlem kodları ile olanaklıdır. Ana program içinde değişik koşullara bağlı olarak program sayacının içeriğini değiştiren JUMP İşlem kodudur. Ana program ile alt programlar arasındaki gidiş dönüşü CALL ve RETURN işlem kodları sağlamaktadır, örneğin Z-80 mikro-işlemcisinde koşulsuz dallanma için var olan komutlardan biri JP nn dir. Bu komutla program sayıcısının yeni adresinin nn olması sağlanır. Koşullu dallanma için var olan komutlardan biri JR NZ, e dir. Koşul sağlandığında program sayacı o andaki adresi üzerine e kadar bir değer alır ve program, program sayacını gösterdiği yeni adresten devam eder. Koşulsuz, alt programa geçişi CALL nn komutu sağlar. Bu komut, program sayıcısının olması gereken değerini SP yazmacına götürür ve ona yeni değeri olan nn'yi verir. RET komutu, koşulsuz olarak 1 alt programdan ana programa geçişi sağlar. Komutun yaptığı iş, SP yazmacındaki bilgiyi program yazmacına aktarmaktır.

h) **GİRİŞ / ÇIKIŞ KOMUTLARI** : Bir dış aygıt ile yazmaçlar veya bellek yerelleri arasında karşılıklı bilgi aktarılmasına izin veren işlem kodlarıdır, örnek olarak Z-80 mikro-işlemcisinde İN ve OUT işlem kodlarını gösterebiliriz

i) **MAKİNEYİ DENETLEYEN KOMUTLAR**: Bilgisayarların Merkezi İşlem Birimini denetleyen işlem kodlarının toplandığı grubdur. örnek olarak Z-80 mikro-işlemcisinde NOP, HALT, Dİ ve EI işlem kodlarını gösterebiliriz. NOP komutu hiç bir işlem yapmaz. HALT komutu makinenin çalışmasını bir kesme işareti gelene değin erteler. Dİ ve EI komutları ise kesme işaretini engellemekte ve yol vermekte kullanılır.

3.2.2 SÖZDE (PSEUDO) İŞLEM KODLARI

Sözde komutları assembly sürecini denetleyen ve programın yeniden yerleştirilmesini sağlayan komutlardır. Bunun yanında programda kullanılan etiketleri, bellek öbeklerini çeşitli değişmez türlerini belirttiği kadar programın yönlendirilmesini tanımlar ve çıkış listesini de denetler. Bu komutları aşağıdaki gibi sınıflandırırız.

a) **ASSEMBLERİ DENETLEYEN KÜME:** Bu küme sözde işlem kodları, program sayıcısının içeriğini değiştirir, yenisini oluşturur ve Assembly sürecinin sonlandırılmasına neden olurlar. Bu işlem kodlarının bulundukları deyimler de Etiket kullanılırsa da bu assembler tarafından bilmezlikten gelir, örnek olarak HP2118 bilgisayarında NAM, ORG ve END yapay işlem kodlarını gösterebiliriz. NAM, bir yerdeğişir (RELOCATABLE) programın ismini tanımlar. Bir yerdeğişir program bir NAM komutu ile başlamalıdır. Aynı biçimde program başlangıç yereli olarak bir (zahiri) sıfır ile assembly edilir, işlenen alanında bulunması gereken İSİM en fazla 5 karakterli bir alfasayısal simge olabilir. Simgenin ilk karakteri abecesel olmalıdır. ORG, bir saltık programın başlangıç noktasını tanımlar. Bir saltık program ORG, sözde işlem kodu ile başlamalıdır. Deyimin işlenen alanında ondalık veya sekizli bir tam sayı bulunur. Bu program sayıcısının başlangıç değerini belirtir. Bir ORG komutunu izleyen bütün komutlar, işlenenin değeri ile başlayan ardası I adreslerde assembly edilirler. END sözde işlem kodu ise assembly programını sonlandırır. Aynı zamanda Kaynak dili deyimlerinin fiziksel sonunu da işaret eder. Deyimin operand alanında bir isim bulunabildiği gibi boş da olabilir.

b) **AMAÇ PROGRAMI YÖNLENDİRİCİ KOMUTLAR:** Ana program ile alt program arasında veya birçok alt program arasında iletişimi sağlayan sözde komutlar bir kümede toplanır. Bu komutlar sadece yerdeğişir programlarında kullanılabilir. Komutların etiket alanı genelde gözönünde bulundurulmaz, işlenen alanı ise birçok alt alana bir virgül ile bölünür, örnek olarak HP 2118 Bilgisayarında COM sözde işlem kodunu gösterebiliriz. Bu komutun görevi, birçok alt Programın ortak olarak kullanacağı bir öbek bellek yerelini ayırt etmektir. Komutun işlenen Alanında görülen her isim alt program için öbeğin bir parçasını tanımlar, isme bağlı olan boyut ise, ilgili parçalara adanan kelimelerin sayısıdır. Boyut gözardı edilmiş ise 1 olarak yorumlanır.

c) **ADRES VE SİMGE TANIMLAYAN KOMUTLAR :** Programın başka bir yerinde bir işlenen olarak kullanılan bir simgeye, bir sayısal değer veya bir kelimelik yerel atayan sözde işlem kodlarıdır, örnek ol-

mak üzere HP 2118 bilgisayarından EOU sözde işlem kodunu gösterebiliriz. Deyimin etiket alanındaki simge, işlenen alanında verilen değerle belirlenir. Bu değer bir hesap sonucu ortaya çıkabilir. Ama eksi bir sayı olamaz, işlenen alanında görülen simgeler daha önce kaynak programında tanımlanmalıdır.

d) **DEĞİŞMEZLER TANIMLAYAN KOMUTLAR :** Amaç programının ardaşıl kelimelerine, bir dizinin (string) bir veya daha fazla değişmez değerini sokan sözde işlem kodlarıdır, örnek olmak üzere ASCII damgaları için ASC, ondalık değişmeler için DEC, sekizli değişmezler için de OCT işlem kodlarını HP 2118 bilgisayarında kullanırız.

e) **BELLEK AYIRMA KOMUTLARI:** Veri çalışma alanı için belleğin bir bölümünün ayrılmasını sağlayan sözde işlem kodlarıdır. Bu işi HP 2118 bilgisayarında BSS sözde işlem kodu yapar.

f) **ASSEMBLY LİSTESİNİ DENETLEYEN KOMUTLAR :** Assembly sürecinin 2. veya 3. geçişi sırasında Assembly çıkış listesini kullanıcının denetimine izin veren sözde işlem kodlarıdır, örnek olmak üzere HP 2118 bilgisayarında. LIST sözde işlem kodu kaynak programının listelenmesine neden olur. Bir UNL sözde işlem kodu ile başlayan program, kaynak programı listesini üretmez.

3-2-3. SÖZDE İŞLEM KODLARI İLE PROGRAMLAMA ÖRNEĞİ

Anlatılacak örnekte, bir assembly dili programındaki ORG, EOU ve END sözde işlem kodlarının kullanılışı, HP 2116 bilgisayarında gösterilmektedir. Programda kullanılacak X, Y ve Z gibi abecesel damgaların bellek adresleri sırasıyla 00300₈, 05126₈ ve 05127₈ olsun. Aynı biçimde LDA, STA, ADA, CPA ve JMP komutları için kodlar sırasıyla 1100₂, 1110₂, 1010₂, 0101₂ ve 1000₂ olsun. HLT ise 16 bitlik bir komut olduğundan işlem kodu 102000₈ dir. örnekteki ORG komutu, assembly işlemini 5000₈ yerinde başlamaya zorlamaktadır, öte yandan HALT komutu 5006₈ yerinde assembly edildiği andan itibaren komutun simgesi 5006₈ adresi ile gösterilir.

Assembler'in yaptığı temel işlerden biri, simgesel isimleri bellek adreslerine tekrar yerleştirmektir. Assembler, bu işi becermek için, simge çizelgesi diye çağrılan simgesel isimlerin bir çizelgesini kurar. Simge çizelgesinde her simgesel isim, uygun damga temsili içinde bellek adresleri ile birlikte saklanır. Böylece geliştirilen programdaki simge çizelgesi aşağıdaki gibidir.

SİMGE	ADRES
HALT	5006 ₈
MOVE	5007 ₈
X	0300 ₈
Y	5126 ₈
Z	5127 ₈

Şimdi assemblerin simge çizelgesini nasıl kurduğunu görelim, örneğimizdeki assembler, 5000₈ adresinden başlayarak ve her komut için adres sayacını 1 arttırarak HALT ve MOVE için adresleri hesaplamaktır. Assembler, bu amaç için program sayacını kullanır.ORG sahte komutu program sayacını başlangıç değerine koyar. Bu komutun işlenen alanındaki B eki assembler'e 8 esasını temel alan sayı dizgesinin kullandığını anlatmaktadır. Sıradan komutlar görevlerini tamamladığı sırada, program sayacının değerinin artmasına neden olurlar.

Bu nedenle, Assembler süreci sırasında HALT görünüşünde program sayacının değeri 5006₈ olacaktır. Etiket Alanında simge olan bir DEYİM'in işlenmesinde assemblerin davranışı aşağıda anlatıldığı gibi olacaktır.

Önce, simge çizelgesindeki simge ile program sayacının değeri yer alır, daha sonra komut işlenir ve program sayacının değeri 1 arttırılır. Son olarak örnek programdaki X, Y ve Z'nin simgesel adresleri komutların adresi yerine bilginin bellek adreslerini temsil eder ve program fiziksel deyimlerin sonunu belirleyen END sahte komutu ile sonlanır

SIR*	ETİKET	İŞLEM KODU	İŞLENEN	YORUMLAR
1		ORG	5000 ₈	5000 ₈ adresinde başla
2		LDA	X	X'i akümülatöre al
3		CPA	Z	Z'yi akümülatöre karşılaştırmak
4		IMP	MOVE	X = Z + 1
5		LDA	Y	Y'yi akümülatöre al
6		ADA	Z	Z ile topla
7		STA	X	*Akümülatörün X'e yükle
8	HALT	HLT	0	Dur
9	MOVE	STA	Y	X'i Y'ye yükle
10		JMP	HALT	Halı etiketine atla
11	X	EOU	3008	X'in adresi 3008'dir.
12	Y	EOU	5126	Y'nin adresi 5126'dir.
13	Z	EOU	5127	Z'nin adresi 5127'dir.
14		END		Programın fonudur.

3.3 İŞLENEN ALANI

İşlenen alanı genelde adresleri ve simgesel adresleri içerir, işlenen alanının anlamı ve (formate) işlem Kodunun türüne bağlıdır. İşlenen alanı işlem kodu-

nun alanını izler. İki alan arasında hiç olmazsa bir boşluk bulunur, işlenen alanı bir boşluk ile sonlanır işlenen alanında aşağıdaki deyimlerden biri bulunabilir. Bunlar, tek simgesel terim, tek sayısal terim, asterisk, sayısal simgesel ve asteriskin birleşimleri ile aritmetik işleç (operator) lerdir. Aynı biçimde Alanında bir (literal) de bulunabilir.

Bir simgesel terimde 1'den 5'e kadar alfa-sayısal denge bulunabilir. HP 2116 bilgisayarında, ilk damga abeceseli olmalıdır. Bir sayısal terimde 1'den 5'e kadar sayısal damga bulunabilir. Sayısal damgaların alabileceği ondalık sayının en büyük değeri 32767'dir. Sayısal terimde sekizli sayıda temsil edilebilir. İşlenen alanındaki en fazla 6 hanelik sekizli sayıyı bir B izler.

Buna karşılık sayısal terimde Hexa-decimal karakterlerde temsil edilir. Bunlar yazıldıktan sonra H ile sonlanır. İşlenen Alanındaki bir Asterisk program sayacının değerine başvuru anlamı taşır. Mikro - bilgisayarlarda, varış noktası veya kalkış noktası bir yazmaç olduğunda "A, B, C, D, E, H veya L" veya bir bellek işlemi olduğunda "M" işleminin alanında görünür. Benzer biçimde " " içindeki ASCII damgaları, etiket olarak tanımlanan adres simgeleri, etiketleri içeren aritmetik deyimler ve o aşamadaki yereli içeren aritmetik deyimler işlenen Alanı içinde yer alabilir.

Bilginin ve komutların bellekte veya yazmaçlarda yerleştirilmesi işlemine adresleme denilir. Bir bellek yerelinin veya bir yazmacın kapladığı yerin içeriğine SÖZCÜK (WORD) denir. Her bir SÖZCÜK de 1 ve 0'lardan oluşan bir ikili amaç kod bulunur. 1 ve 0 lar BİT diye anılır. Bir SÖZCÜĞÜN uzunluğu 4, 8, 16, 32 veya 64 bit olabilir. Bu, bilgisayarın türüne bağlıdır. Bir SÖZCÜKLÜK uzunluğa bir komut yerleştirilebildiği gibi 2 veya 3 sözcüklük bir uzunluğa da 1 KOMUT konabilir. Bu komutun ve makinenin türüne bağlıdır. Bellek ve yazmaç alanlarındaki her sözcüğün bir adresi vardır. Dolayısıyla adresler hem bilgi ve hem de komutlar için belirlenir. Çeşitli adresleme teknikleri vardır. Bunların makine tarafından anlaşılan söz dizinleri bilgisayardan bilgisayara değişir. Genelde, adresleme tekniklerini şöyle sıralayabiliriz.

3.3.1 DOLAYSIZ ADRESLEME KİPLERİ (Mods)

Bu adresleme kipinde işlenen olarak seçilmiş yazmacın içeriği işlem görür. Bu içerik 1 arttırılarak veya

azaltılarak ya da dizin yazmacının (index register) ile toplanarak yeni işlenen sayacı veya yığıt (Stack) Göstericisi de alınabilir. Şimdi bunları teker teker inceleyelim.

a) YAZMAÇ KİPİ: Seçilmiş kayıtlayıcının içeriği işlenendir. Bu kipin simgesel gösterimi aşağıdaki şekildedir.

KOMUT → İŞLENEN

örnek olarak LSI 11 bilgisayarında INC R3 komutunu inceleyelim. Burada seçilmiş yazmaç R3 'dür. Komutun işlem kodunda seçilmiş yazmacın içeriğinin 1 artırılması komut edilmektedir. Komutun işlenen Alanında ise bu işin yapılacağı yer sergilenmektedir.

Böylece R3 yazmacının içeriği 1 arttırılarak komut işlenmiş olur.

b) KENDİLİĞİNDEN ARTIRMA KİPİ (AUTO INCREMENT MODE): Seçilmiş yazmacın içeriğine her byte için 1 ve her sözcük (Word) için 2 eklenir. Eski içerik işlenenin kendisidir. Ekleme işleminin kaç kez yapılacağı bir yazmaçta verilmiştir. Bu teknik dizilerde ve yığıt işlemlerinde yararlıdır. Bu kipin simgesel gösterilimi aşağıdaki çizimdeki gibidir, örnek olarak LSI 11 bilgisayarında

KOMUT → ADRES → İŞLENEN
↑
↑
2VEYA 1

CLR (R5) + komutunu inceleyelim. Burada seçilmiş yazmaç R5'dir. Komutun işlem kodunda seçilmiş yazmacının içeriğinin kendiliğinden 2 arttırılması ve eski içeriğin gösterdiği adresteki sözcüğün sıfırlanması istenmektedir. Komutun işlenen alanında ise bu işin yaptırıldığı yer sergilenmektedir.

c) KENDİLİĞİNDEN EKSİLTME KİPİ (AUTO - DECREMENT MODU): Seçilmiş yazmacın içeriği her byte için 1 ve her sözcük için 2 eksiltir. Yeni içerik işlenenin kendisidir. Eksiltme işleminin kaç kere yapılacağı bir yazmaçta verilmiştir. Bu teknik, bir listedeki bilginin ters yönde işlenmesinde yararlıdır. Bu kipin simgesel gösterilimi aşağıdaki çizimdeki gibidir.

KOMUT -jADRES ->-2 VEYA -1 -İŞLENEN

örnek olarak LSI. 11 bilgisayarında INC-{RO} komutunu inceleyelim. Burada seçilmiş kayıtlayıcı RO'

dur. Komutun işlem kodunda seçilmiş yazmaç içeriğinin kendiliğinden 2 eksiltmesi ve yeni içeriğin gösterdiği adresteki kelimeye 1 eklenmesi istenmektedir. Komutun işlenen alanında ise bu işin yaptırıldığı yer sergilenmektedir.

d) DİZİN KİPİ (INDEX MODU): Seçilmiş genel yazmaç içeriği ile dizin yazmacının içeriği toplanır. Yeni içerik işlenen alanının adresidir. Bu kipin simgesel gösterilimi aşağıdaki gibidir, örnek olarak LSI 11 bilgisayarında CLR 200 (R 4) komutunu inceleyelim.

KOMUT → ADRES → + → İŞLENEN
X

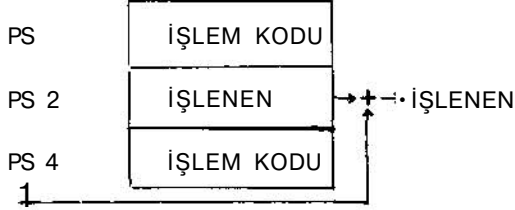
Burada seçilmiş yazmacının R4, dizin yazmacının içeriği ise 200'dür. Komutun işlem kodu seçilmiş yazmaç içeriği ile dizin yazmacının içeriğini toplamaktadır. Toplamanın sonucunu yeni adres olarak almakta ve bu adresteki kelimeyi sıfırlamaktadır.

e) IMMEDIATE KİPİ: Burada seçilmiş yazmaç özel olarak komutun işlenen alanının kendisidir, işlenenin değeri komutun içindedir. Bir değişmezin anında, komut içinde verilmesi özelliğini taşır. Bu kipin simgesel gösterilimi aşağıdaki gibidir, örnek olarak LSI TT bilgisayarındaki ADD f 10,RO komutunu inceleyelim. Burada seçilmiş yazmacın içeriği komutun işlenen alanında görünen 10 değişmez değeridir. Komutun işlem kodunda, RO yazmacının içeriği ile 10 değişmez değerinin toplanması ve sonucun RO yazmacına konması istenmektedir.

İŞLEM KODU
İŞLENEN

f) GÖRELİ KİP (RELATIVE MOD): Komutun işlenen alanının içeriği ile program sayacının bir sonraki komutu belirleyen değeri toplanarak istenen işlenenin adresi bulunur. Bu teknik dallanma işlemlerinin editminde kolaylık sağlar. Ayrıca yerdeğişir kodlara izin verme açısından çekicidir. Bu kipin simgesel gösterimi çizimdeki gibidir. PS, program sayıcısının kısaltılmış yazılışdır, örnek olarak, LSI 11 bilgisayarında INC A komutunu gözden geçirelim. Hemen şunu belirtmekte yarar vardır. Makine, böyle bir simgesel komut ile karşılaştığı zaman otomatik olarak görelî adresleme

kipinde olduğunu anlar. Bu andan itibaren komutun işlenen alanındaki sayısal değer ile program sayacının bir sonraki komutu belirleyen sayısal değerini toplar. Elde ettiği sayısal değer komutun INC işlemini yapacağı yerin adresidir. Bu adresteki sözcüğe 1 eklenecek komut işlenmiş olur.



3-3-2 DOLAYLI (DEFERRED) ADRESLEME KİPLERİ

Komuttaki işlenen alanının içeriği, GÖSTERGE (POINTER) diye isimlendirilen bir yazmacın adresini gösterir. Göstergenin içeriği arzulanan işlenenin adresidir. Bu teknik de kendi içinde alt sınıflara ayrılır.

a) YAZMAÇ DOLAYLI KİPİ: Seçilmiş yazmacın içeriği işlenen adresidir. Bu kipin simgesel gösterimi aşağıdaki gibidir, örnek olarak LS111 bilgisayarın da CLR A R5 dir. A simgesi dolaylı adresleme tekniğini belirtmektedir. İşlem kodu, R5 yazmacının içeriğinin gösterdiği adresteki Sözcüğün sıfırlanmasını istemektedir.

KOMUT—»ADRES—»İŞLENEN

KENDİLİĞİNDEN ARTAN DOLAYLI KİPİ (AUTO-INCREMENT DEFERRED MOD): Komutun işlenen alanının içeriği gösterge yazmacının adresidir. Gösterge yazmacının içeriğinin gösterdiği adresteki sözcük arzulanan işlenenin adresidir, işlem tekrarlı olduğundan gösterge yazmacının içeriği her seferinde 2 arttırılır. Arttırma işleminin kaç kere yapılacağı bir yazmaçta verilmiştir. Bu kipin simgesel gösterimi aşağıdaki gibidir, örnek olarak LSI 11 bilgisayarında INC Q(R5) + komutunu inceleyelim. Komutun işlenen alanında görülen, R5, gösterge yazmacının adresidir. Komutun işlem kodu ise, gösterge içeriğinin gösterdiği adresteki kelimeyi arzulanan işlenenin adresi olarak alır. Sonuçta bu adresteki işlenenin değerini 1 arttırır ayrıca gösterge yazmacının değerini de 2 arttırarak komutun işlenmesini bitirir.

KOMUT—» ADRES-f^ADRES—^OPERAND



c) KENDİLİĞİNDEN AZALAN DOLAYLI ADRESLEME KİPİ (AUTO DECREMENT DEFERRED MOD): Komutun işlenen alanının içeriği gösterge yazmacının adresidir. Gösterge yazmacının içeriğinden 2 eksiltildikten sonra yeni içeriğin gösterdiği adresteki sözcük arzulanan işlenenin adresidir.

İşlem tekrarlı olduğundan gösterge yazmacının içeriği her seferinde 2 eksiltir. Eksiltme işleminin kaç kere yapılacağı bir yazmaçta verilmiştir. Bu kipin simgesel gösterimi aşağıda verilmiştir.

KOMUT—»ADRES ^ - 2 _>ADRESİİŞLENEN

örnek olarak LSI 11 bilgisayarında COM O-(RO) komutunu inceleyelim. Komutun işlenen alanında görülen RO gösterge yazmacının adresidir. Komutun işlem kodu, Göstergenin içeriğini 2 eksiltir. Yeni içeriğin belirttiği adresteki Sözcüğü arzulanan işlenenin adresi olarak alır. Sonuçta işlenene deçilleme COMPLEMENT işlemini uygular.

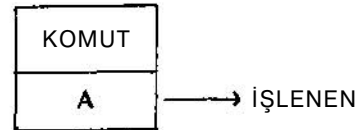
d) DİZİNLİ - DOLAYLI ADRESLEME KİPİ (INDEX DEFERRED MOD): Seçilmiş genel yazmacın içeriği ile dizin yazmacının içeriği toplanır. Bu sonra gösterge olarak alınır. Yeni içeriğin belirttiği adresteki sözcük arzulanan işlenenin adresidir. Bu kipin simgesel gösterimi aşağıdaki gibidir.

KOMUT—-ADRES++-ADRES—»İŞLENEN



örnek olarak LSI 11 bilgisayarındaki ADD O 1000 (R2), R1 komutunu inceleyelim. Komutun işlenen alanında görülen 1000 dizin yazmaç içeriği, R2 ise seçilmiş yazmaç olmaktadır. Komutun işlem kodu ise, dizin yazmacının içeriği ile R2 yazmacının içeriğini toplar. Toplamın gösterdiği adresteki sözcüğü adres olarak alır ve bu son adresin içeriğini R1 yazmacının içeriği ile toplar ve sonucu R1 'de saklar. Böylece komut işlenmiş olur.

e) IMMEDIATE DEFERRED YA DA ABSOLUTE MODU: Komutun işlenen alanındaki sayısal değer arzulanan işlenenin adresidir.



Bu kipin simgesel gösterimi yukarıdaki gibidir, örnek

olarak LSI 11 bilgisayarında ADD 0 / 2000, R3 komutunu inceleyelim. Komutun işlem kodunda, işlenen alanında görülen değişimin tanımladığı adresteki sözcük ile R3 yazmacının içeriğini toplanması ve sonucun R3 yazmacına yazılması istenmektedir.

f) RELATIVE DEFERRED MODU: Komutun işlenen alanının içeriği ile program sayacının bir sonraki komutu belirleyen değeri toplanarak gösterge yazmacının adresi bulunur. Gösterge yazmacının içeriği arzulanan işlenen adresidir. Bu modun simgesel gösterimi aşağıdaki gibidir.



örnek olarak LSI 11 bilgisayarında CLR 0 A komutunu inceleyelim. Komutun işlenen alanında görünen simge, makinenin relative deferred modunda işlem göreceğini belirtmektedir. Komutun işlem kodu işlenen alanındaki simgenin sayısal içeriği ile program sayacının bir sonraki komutu belirleyen değerini toplar. Böylece gösterge sayacının adresi bulunmuş olur. Bu içerik işlem görülecek işlenenin adresidir. Sonuçta elde edilen işlenen komut gereği sıfırlanır.

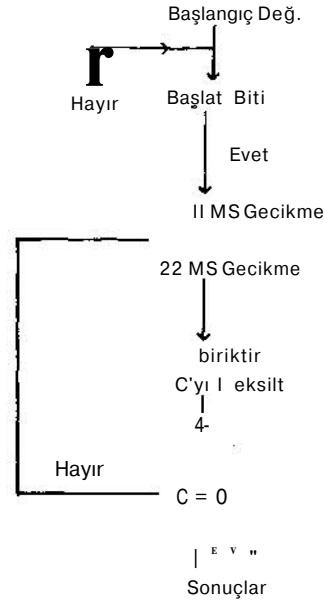
3-4YORUM (COMMENT) ALANI

Programı daha iyi okunabilir duruma getirmek için kullanılan alandır. Bu alanda tanımlayıcı açıklamalar bulunur. Bu açıklamalar assembler tarafından göz önünde bulundurulmaz. Ancak assemblerin ürettiği çıkış listesinde yer alır. Yorum Alanı işlenen alanını izler ve ondan hiç olmaz ise bir boşluk ile ayırt edilir. DEYİM'in sonu bir CR (Carriage Return) ve LF (Line Feed) satır besle ile sonlanır. İşlenen yoksa yorum alanı gözardı edilir, örneğin HP 2116 bilgisayarında HLT, NAM ve END komutlarında bu alan gözardı edilir.

4- ASSEMBLY DİLİ İLE BİR PROGRAMLAMA ÖRNEĞİ

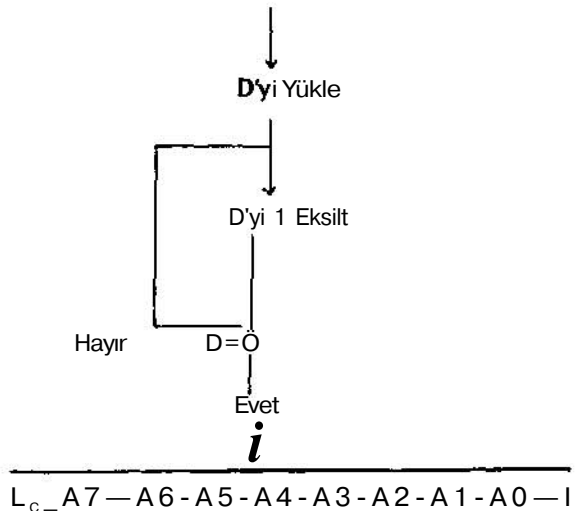
Assembly dilinin kimi temel ilkelerini kullanarak bir programlama denemesi yapalım, örneğin Telex üzerinden iletişimi donatılmış bir uzak yazıcı (Teletype) tarafından gönderilen dizi verinin alınması için bir program yazalım. Çizim 4-1-a 'da genel durumu ile işaret formatı temsil edilmektedir, örnek olarak "R" nin kodu gösterilmektedir. Her 'bit' 22 MS lik süre ile gönderilmektedir. Gönderilecek veri bulunmadığı zaman hat sürekli olarak " 1 " (STOP) konumunda kalmaktadır. Veri gönderilirken birinci bit olarak her zaman Başta (Start) Biti "0", gönderilir.

Genel	ST	BI	B2	B3	B4	B5	DUR
örnek 'R'	ST	0	1	0	1	0	DUR



Çizim 4-1 Bilginin ve Programın akış dlagramı

Çizim 4-2 Gecikmenin akış şeması ile devrede yapılışı



Zamandaslama (Senkronizasyon), 1'den 0'a başla anından itibaren her bit'in merkezine olan zamanın ölçülmesi ile de elde edilir. Şekil 4-1-b 'de ise istenen

program için ayrıntılı bir akış şeması bulunmaktadır. Burada, zaman gecikmelerini temsil eden iki öbek dikkatimizi çekmektedir. Bunlar, 11 MS ile 22 MS'lik gecikmelerdir. Çizim 4-2-a da 11 MS'lik gecikmenin ayrıntılı akış şeması gösterilmektedir. Bu yordamda (routine) D yazmacı kullanılmaktadır. D yazmaçlarındaki döngülerin sayısal adedi istenen zaman gecikmelerini sağlamaktadır. INTEL 8008 mikro-bilgisayarını kullandığımızı varsayalım.

Bunun INCREMENT komutu ile 20 A» ve JUMP komutu ile 44 **ts** lik gecikmeler sağladığını bilmekteyiz. Bu durumda döngüdeki toplam gecikmenin $20 + 44 = 64$ MS olduğu görülür. 11 MS'lik bir gecikmenin elde edilebilmesi için $11000/64 = 172$ kere döngü etrafında dolanılması gerektiği ortaya çıkar. Gecikme programının simgesel biçimi şöyledir.

ETİKET	İŞLEM KODU	OPERAND	YORUM
	MVI	D, 172	11 MS
LOOP1	DCR	D	
	JNZ	LOOP1	

Burada simgesel adreslerin yararlarını görmekteyiz. Programın bu kısmının, belleğin neresinde olacağına ilişkin bir fikrimiz yoktur. JUMP komutu ile bir önceki adrese tekrar başvurmamız. Basitçe LOOP 1 gibi keyfi bir isim yaparak, adresleri artık düşünmek sizin bu isme ilgi kurabiliriz.

Assembly programıda bu noktadan sonra ayrıntıları göz Önünde bulunduracaktır. Veyahut JUMP komutundan itibaren işlenen Alanındaki adresi (\$) olarak tanımlayabiliriz. Bu durumda LOOP 1 gibi simgesel bir adrese gereksinim duymayız. Gecikme programı aşağıdaki biçimde yazılabilir. Her iki durumda da Assembler tarafından üretilen Amaç Program birbirine eşdeğer olacaktır. İkinci yöntem "değişken sözcük uzunluk" makinelerde tümüyle tehlikeli olacaktır.

İŞLEM KODU	OPERAND	YORUM
MVI.	D, 172	11 MS GECİKME
DCR	D	
JNZ	\$ - 1	

22 MS'lik gecikme elde etmek önemli bir sorundur, zira basit bir çevrim ile üretebileceğimiz en fazla gecikme $256 \times 64 = 16700$ 16,7 S 'lik gecikme elde edilebilir Amaçlarımıza uygun olan gecikmeyi çevrim, içine basit bir komut koyarak elde edebiliriz, örneğin (CMP M) komutunu eklersek bir kaybımız olmayacak, tattı tersine çevrim zamanına bir 32 Ms daha eklemiş oluruz, böylelikle onu, $64 + 32 = 96$ [is ye çıkartmış oluruz. İstenen çevrim sayısı ise

$22000/96 = 230$ olur. Böylece 22 MS lik gecikme yordamı için Assembly Programı aşağıdaki biçimde olacaktır.

ETİKET	İŞLEM KODU	OPERAND	YORUM
	MVI	D, 230	11 MS GECİKME
LOOP2	CMP	M	
	DCR	D	
	JNZ	LOOP 2	

Her 22 MS gecikmenin sonunda, yeni bit zamanının merkezi elde edilecektir. Sonra A yazmacının 0 bitimine bir bit koyacağız. Bu biti daha sonra RAR (ROTATE A RIGHT) yapması için Çizim 4-2-b'de gösterildiği gibi Carry FFa koyacağız, önceki biriktirilmiş bitlerin B kayıtlayıcısından A yazmacına aktarılması (MOVE) daha sonra ROTATE A LEFT komutu ile yeni bitin A içine taşınması ve diğerlerinin bunu izlemesi sağlanır. Daha sonra A yazmacından B yazmacına biriktirilmiş veri aktarılır ve üzerine 1 eklenir ve sonunda bit sayısı sinanır. Yorumları ile birlikte simgesel kodlama aşağıdaki gibidir.

İŞLEM KODU	OPERAND	YORUM
IN	0	A, ya yeni bit'i gir
RAR		Yeni bit'i Carry'ye al
MOVE	A, B	önceki bit'leri A'ya koy
RAL		Bütün bit'leri sola kaydır
MOVE	B, A	Bit'leri B de tut
DCR	C	Bit sayacını 1 azalt
JNZ	LOOP 3	5. bit değilse çevrime devam

Programlanmayan tek nokta çizim 4-1-b'deki Akış şemasının başlangıç kısmıdır. Bunun için, bit sayıcısını basitçe 5 ile yüklemeliyiz.

Arkasından bir HALT komutunu işletmeliyiz. Böylece bir kesme imi gelene değin mikro-işlemci DUR konumunda kalacaktır. Bu durumda, Veri hattı o giriş kapısının 0 bitimine bağlı olduğu gibi mikro-işlemcinin kesme girişine de bağlıdır. Bir BAŞLA imi (start transition) alındığında, ilk komut olan 11 MS'lik gecikme ile program başlayacaktır. Başlama imini sinayan diğer bir döngü programı da aşağıdaki gibidir.

İŞLEM KODU	İŞLENEN
IN	0
CPI	0
JNZ	\$ - 2

Giriş imi DUR konumunda kaldığı sürece program yukarıda yazılı olan döngü yordamında kalacaktır. Başla imi geldiği an program döngü yordamından çıkar ve 11 MS'lik gecikmeyi sağlayan bölüm ile devam

eder. Yukarıdaki yöntem programa 3 fazla byte konmasına neden olur. Tamamlanmış program, tartıştığımız kısımların basitçe biraraya getirilmesiyle yazılabilir. Aşağıda Assembler tarafından listelenmiş program gösterilmektedir.

Çizelgeden görüldüğü gibi, listelenmiş assembly programının sol yanında adanmış bellek yerelleri ve onaltılı biçimde komut kodları bulunmaktadır. Soldan itibaren her sıradaki ilk 4 adanmış bellek yerelleri adresidir. Aynı sıradaki son 4'dii ise komut kodunun kendisi olup komutun her byte'ına karşılık gelen iki onaltılı hane olarak basılıdır.

Assembler tarafından basılan listedeki ilk satırdaki ilk komutun (MVI C, 5) 0000 ve 0001 no'lu bellek yerellerinde saklandığı (biriktirildiği) gözlemlenmektedir. Komut kodunun onaltılı hanesi 16'dır. Bu MCS8

assemblerinde 00 010 110 ikili işlem koduna karşılık eder.

Burada C yazmacının 010 göstermektedir. Komut kodunun ikinci onaltılı hanesi immediate işleneni temsil eden S'dir. S Buda MCS8 assemblerinde 000 001 01 ikili işlem koduna karşılık gelir. Bir sonraki komut (HLT) ikinci bellek yerelinde saklıdır. Ve içeriği sıfırdan oluşur. Görüldüğü gibi Assembler, istenen işlem kodlarını ve adanmış ardaşıl adresleri kendiliğinden üretir. Şimdi, JNZ \$- 1 komutunun ikinci byte'ını göz önüne getirelim. Burada assembler, o anki bellek yeri adres değerinden (6), 1 çıkartma işlemini (6 - 1 = 5) hesap etmektedir. Aynı biçimde assembler LOOP3 etiketini de göz önünde bulundurur. Programın son satırında işlenen alanında görünen LOOP3'ü daha önceden adanmış olan 9 sayılı bellek yereline gönderir.

ASSEMBLER TARAFINDAN ÜRETİLEN ÇIKIŞ LİSTESİ

SIRA	BELLEK YERELİ	KOMUT KODU	ETİKET	İŞLEM KODU	İŞLENEN	YORUM
1	0000	1605		MVI	C,5	bit sayıcısına 5 koy
2	0002	00		HLT		başla gelene değin bekle
3	0003	IEAC		MVI	D, 172	II MS'lik gecikme
4	0005	19		DCR	D	D'yi 1 azalt
5	0006	480500		JNZ	\$-1	D = 0 olana kadar çevrimde kal
6	0009	IEE6	LOOP3	MVI	D, 230	22 MS'lik gecikme
7	000B	BF	LOOP2	CMP	M	zaman gecikmesi ekle
8	000C	19		DCR	D	D'yi 1 azalt
9	000D	480BOO		JNZ	LOOP2	D = 0 olana kadar çevrimde kal
10	0010	41		IN	0	Yeni biti gir
11	0011	1A		RAR		biti Cary'ye al
12	0012	C1		MOV	A,B	Önceki bitleri A'ya koy
13	0013	12		RAL		Bütün bitleri sola kaydır
14	0014	C8		MOV	B,A	Bitleri B'de tut
15	0015	11		DCR	C	Bit sayıcısını 1 azalt
16	0016	480900		JNZ	LOOP3	5. bit değilse çevrimde kal

KAYNAKÇA

1. Software Design Far Microprocessors, J.G. VVester, W.D. Simpson, 1976, Texas Instrument Inc.,
2. Microcomputer Handbook, DEC, 1976
3. A Pocket Guide To Hewlett • Packard Computers, H.P.,
4. Minicomputer Systems, Organization and Programming (PDP 11), R.H. Eckhouse, Prentice Hall Inc., 1975,
5. Minicomputers in Data Processing And Simulation, B. Soucek, John Wiley and Sons, 1972,
6. Digital Design With Standart MS 1 And LS 1, T.R. Blakeslee, John Wiley and Sons, 1979,
7. Introduction To Computer Organization And Data Structures, H.S. Btome, McGraw Hill Book Company. 1972
8. Microprocessors in Instrument And Control. R.J. Bibbero John Wiley and Sons Inc, 1977,
9. 8080 Microcomputer Systems User s Manuel, Inte 1,1975,

10. Z-80 Microcomputer Data Book, Mostek, 1981,
11. Microprocessors and Logic Design, R.L. Kura, John Wiley And Sons Inc, 1980,
- 12- Microprocessors Programming And Software Development, F.G. Duncan, Prentice - Hall International Inc., 1979,
- 13- Model 70 User's Manual, Interdata Inc., 1970,
- 14- 8080/Z80 Assembly Language, Techniques For Improver Programming A.R. Miller, John Wiley and Sons Inc., 1981,
- 15- TMS 9980 Microprocessor Data Manual, Texas Instrument Inc, 1979,
- 16- Z8 Single Chip Microcomputer, Technical Manual. Syner-tek, 1979,
- 17- Am Z 8000 Microprocessors Specifications, Advanced Micro Devices, 1978,
- 18- Microprocessors User's Guide, Pro-Log Corp., 1979 1980.