Intel 8086 ile Mikroişlemci Programlamaya Giriş

Şadi Çağatay Öztürk
Dr. Oğuz Ergin

İÇİNDEKİLER TABLOSU

İÇİNDEKİLER TABLOSU 2

1. BİLGİSAYAR SİSTEMLERİNE GİRİŞ 5

1.1. BİLGİSAYAR NEDİR? 5
1.2. ÇAĞDAŞ ANLAMDA BİLGİSAYARIN TARİHÇESİ 5
1.3. BİLGİSAYARIN BİLEŞENLERİ 6
1.3.1. Donanım 6
1.3.2. Yazılım 11

2. SAYI SİSTEMLERİ 16

2.1. RAKAMLAR 16
2.2. RAKAMLAR VE BİLGİSAYARLAR 17
2.2.1. İkilik Taban 17
2.2.2. Onaltılık Taban 19
2.2.3. İkilik Tabanda Tümleyenler 21
2.2.4. İkilik Tabanda Aritmetik İşlemler 22
2.2.5. Onaltılık Tabanda Aritmetik İşlemler 24
2.3. İŞARETLİ SAYILAR 26
2.4. KAYAN NOKTA VERİ TÜRÜ 27

3. 8086 MİMARİSİ 29

3.1. 8086 ÖNCESİ 29
3.2. 8086 29
3.2.1. Bellek 35
3.2.2. Kesmeler 38
3.2.3. Girdi/Çıktı Kapıları 38
3.2.4. Yazmaçlar 38
3.2.5. Komutlar 40
3.2.6. Adresleme Kipleri 41

4. ÇEVİRİCİ DİLİNE GİRİŞ 48

4.1. VERİ TÜRLERİ 48
4.2. BÖLÜTLER 49
4.2.1. Kod Bölütü 49
4.2.2. Veri Bölütü 51
4.2.3. Yığıt Bölütü 53
4.2.4. Ek Bölüt 54
4.3. BÖLÜTLERİN ÖRTÜŞMESİ 54
4.4. ÇEVİRİCİ, BAĞLAYICI, KONUMLANDIRICI VE HATA AYIKLAYICI 54
4.4.1. Çevirici 55
4.4.2. Bağlayıcı 55
4.4.3. Konumlandırıcı 55
4.4.4. Intel Hex Format Çeviricisi 55
4.4.5. Hata Ayıklayıcı 56
4.4.6. FASM 57
4.5. 8086 KOMUT KÜMESİ 57
4.5.1. Makine Diline Çevrim 57
4.5.2. Veri Taşıma Komutları 61
4.5.3. Aritmetik Komutları 66
4.5.4. Mantık Komutları 74
4.5.5. Dizgi İşleme Komutları 80
4.5.6. Denetim Aktarma Komutları 83
4.5.7. İşlemci Denetimi Komutları 93

5. ÇEVİRİCİ DİL İLE MATEMATİK VE MANTIK İŞLEMLERİ 96

5.1. İŞARETSİZ SAYILAR İLE MATEMATİKSEL İŞLEMLER 96

5.1.1. Toplama *96*
5.1.2. Çıkarma *99*
5.1.3. Çarpma *100*
5.1.4. Bölme *103*
5.2. İŞARETLİ SAYILAR İLE MATEMATİK İŞLEMLERİ 104
5.2.1. Toplama *105*
5.2.2. Çıkarma *108*
5.2.3. Çarpma *109*
5.2.4. Bölme *109*
5.3. MANTIK İŞLEMLERİ 110

6. ASSEMBLY İLE DİZGİ VE BCD-ASCII İŞLEMLERİ 115

6.1. DİZGİ İŞLEMLERİ 115
6.1.1. DF Bayrak Biti *115*
6.1.2. REP MOVSB *115*
6.1.3. REP MOVSW *116*
6.1.4. REPE CMPSB *116*
6.1.5. REPNE SCASB *117*
6.1.6. LODSB *117*
6.1.7. REP STOSB *118*
6.2. BCD-ASCII İŞLEMLERİ 119
6.2.1. BCD Sayı Sistemi *119*
6.2.2. ASCII Sayılar *120*
6.2.3. Paketlenmemiş BCD – ASCII Çevirimi *120*
6.2.4. Paketlenmiş BCD – ASCII Çevirimi *121*
6.2.5. ASCII – Paketlenmemiş BCD Çevirimi *121*
6.2.6. ASCII – Paketlenmiş BCD Çevirimi *122*

7. KESMELER VE ÇEVRE BİRİMLERİNİN KULLANIMI 124

7.1. KESMELER 124
7.1.1. Donanım Kesmeleri *124*
7.1.2. Yazılım Kesmeleri *124*
7.1.3. Kural Dışı Kesmeler *124*
7.1.4. BIOS Kesmeleri *124*

8. GİRDİ/ÇIKTI İŞLEMLERİ 149

8.1.1. Seri İletişim *149*
8.1.2. 8250 ..... *149*
8.1.3. Örnek *153*
8.1.4. Paralel İletişim *155*
8.1.5. Tek Yönlü İletişim *157*
8.1.6. Vidyo Belleği Girdi/Çıktı İşlemleri *157*
8.1.7. BIOS Vidyo Kipleri *158*
8.1.8. Vidyo Kartı Yazmaçları *162*

9. BASİT BİR İŞLETİM SİSTEMİ 170

9.1. İŞLETİM SİSTEMİ NEDİR? 170
9.2. BASİT İŞLETİM SİSTEMİMİZ 170
9.2.1. Disket Yükleyici *170*
9.2.2. Önyükleme Yazılımı *174*
9.2.3. Genel Menü Yazılımı *179*
9.2.4. Hesap Makinası Yazılımı *183*
9.2.5. Metin Düzenleyici Yazılımı *186*
9.2.6. Resim Çizdirme Yazılımı *192*
9.2.7. Seri İletişim Alıcı Yazılımı *195*
9.2.8. Seri İletişim Verici Yazılımı *196*

10. EK A – X86 KOMUTLARININ LİSTESİ 199

10.1. 8086 KOMUTLARI 199
10.2. 80186/80188 İLE EKLENEN KOMUTLAR 205

10.3. 80286 İLE EKLENEN KOMUTLAR 205
10.4. 80386 İLE EKLENEN KOMUTLAR 206
10.5. 80486 İLE EKLENEN KOMUTLAR 208
10.6. PENTİUM İLE EKLENEN KOMUTLAR 208
10.7. PENTİUM PRO (P6) İLE EKLENEN KOMUTLAR 209
10.8. PENTİUM III (SSE) İLE EKLENEN KOMUTLAR 210

11. EK B 214

11.1. 1. FASM KULLANIMI 214
11.2. 2. LİNUX'TA NASM KULLANIMI 214

12. EK C 217

12.1. DOS KESMELERİ 217
12.1.1. INT 21H – Girdi/Çıktı Kesmesi 217
12.1.2. INT 33H – Fare Kesmesi 222
12.2. LİNUX KESMELERİ (SİSTEM ÇAĞRILARI) 224
12.2.1. INT 80H 224

13. EK D 228

13.1. BIOS KESMELERİ 228

14. EK E 242

14.1. KLAVYE ASCII KODLARI 242
14.2. KONTROL KARAKTERLERİ VE ANLAMLARI 242
14.3. KLAVYE TARAMA KODLARI 243

1. BİLGİSAYAR SİSTEMLERİNE GİRİŞ

1.1. Bilgisayar Nedir?

Bilgisayar, günümüzde hemen hemen tüm alanlarda kullanılan ve şu anda dünyanın en önemli aygıtı durumuna gelmiş bir araçtır. Tarihsel gelişim süreci göz önünde bulundurulduğunda, mevcut kaynakların ve bu konuda uzman kişilerin çoğu bilgisayar tarihinin antik çağlarda abaküsün kullanımı ile başladığını iddia ederler. Abaküs, bilindiği üzere boncuk dizilerinden oluşan ve basit toplama işlemlerini yapmaya yarayan bir araçtır. Çok basit bir hesap makinesi olarak da düşünülebilecek olan abaküs, genel olarak insanoğlunun ürettiği ilk bilgisayar olarak tasvir edilir. Bunun nedeni, insanların sayısal işlemleri yaparken hayatlarını kolaylaştırmada abaküsü kullanmalarıdır.

Bilgisayar, ilk bulunduğundan beri birçok kereler tanımlanmaya çalışılmıştır. Günümüzde kullanılan en doğru tanım ise "aritmetik ya da mantık işlemlerini önceden yüklenmiş programlar sayesinde çözebilen araç" tır. Bilgisayar sistemleri sürekli kendini geliştirmekte ve yenilemekte olduğundan bu tanım kalıcı bir tanım değildir. Yakın gelecekte bilgisayarların kendi kendilerini programlaması ve yapay usları sayesinde kendi kararlarını -kısıtlı da olsa- verebilmeleri mümkün olacaktır. Geçmişten abaküs olarak gelen bilgisayar, gelecekte yapay sinir ağlarıyla donatılmış "zeki" bir alet haline dönüşecektir.

1.2. Çağdaş Anlamda Bilgisayarın Tarihçesi

Çağdaş anlamda üretilen ilk bilgisayar olarak 1642'de Blaise Pascal tarafından üretilen hesap makinesidir. Bu makine, üzerinde bulunan tuşlar sayesinde girilen sayılar üzerinde toplama ve çıkarma yapabilmekteydi. Yaygın olarak kullanılmayan, kolay taşınamayan, hantal ve hayli ağır olan bu araç pek rağbet görmemiştir. Daha sonra 1671'de Gottfried Wilhelm von Leibniz bir bilgisayar tasarlamış ancak bu tasarladığı bilgisayarı 1694 yılına kadar üretememiştir. Günümüzde kullanılan bilgisayarlardan oldukça farklı olarak ağır mekanik aksam ve dişliler kullanılarak üretilmiş bu bilgisayar dört işlem yapabilmekteydi. Ancak bu bilgisayar da Pascal'ın hesap makinesinin akıbetine uğramış, kullanımının zor olması nedeniyle pek ilgi görmemiştir.

1820'de Xavier Thomas ticari amaçlı ilk hesap makinesini üretene kadar bilgisayar tasarımı konusunda bir durgunluk yaşanmıştır. Dört işlem yapabilen Xavier Thomas'ın hesap makinesi, gelişen sanayinin karmaşık hesaplamaların yapılmasına olan gereksinimi nedeniyle önceki bilgisayarlara göre daha fazla ilgi görmüştür. Bu noktadan sonra bilgisayarın gelişimi gözle görülür bir şekilde hızlanmış, 1823'te buharla çalışan ilk tam otomatik hesap makinesi yapılmıştır. Bu aracın en büyük özelliği olan sabit talimat programıyla kumanda edilmesi çağdaş anlamda ilk programlama örneği olarak görülebilir.

1890 yılı bilgisayar tarihi için bir dönüm noktasıdır. Bu tarihte Herman Hollerich, yaklaşık 80 yıl kadar dünyanın çeşitli ülkelerinde kullanılacak olan delikli kart teknolojisini bulmuş ve bu mantığı kullanan bir bilgisayar tasarlamıştır. Delikli kart teknolojisi ile delikli kartlar hem bilgisayara kod girişinin yapılması hem de bellek olarak kullanılmıştır. Böylece bilgisayara kod girişi kolaylaşmış, bilgisayarlara bellekten alınan bir veriyi kullanabilme özelliği eklenmiştir. Delikli kart kullanan ilk analitik araç nüfus sayımını yaparken hesaplamalarında kolaylık sağlamak amacıyla üretilmiştir. Bu tarihten sonra bilgisayarın gelişimi savaşa giren ülkelerin teknolojide otomasyon kullanmak istemesi ve bilime verdikleri önemin artması sonucu daha da hızlanmıştır. Örneğin Amerikalı bilim adamı Howard Hathaway Aiken'in yönettiği bir araştırma takımı, 2. Dünya Savaşı'nın öncesinde ülkelerin arasındaki rekabetin en üst düzeyde olduğu 1937'de, Mark–1 adındaki ilk tam otomatik sayısal bilgisayarı yapmayı başarmıştır. Bu bilgisayar 1890'da üretilen atası gibi delikli kartları kullanmaktaydı ve sonradan üretilecek olan ENIAC'a göre epey yavaş çalışmaktaydı. 1945'de, 2. Dünya Savaşı'nın bitmek üzere olduğu bir zamanda, askeri amaçla üretilen ENIAC bir futbol sahası büyüklüğündeydi ve teknisyenler ENIAC'ın patlayan radyo lambalarını tamir edebilmek ya da bozulan rölelerini değiştirebilmek için golf arabaları kullanmak zorundaydılar. ENIAC'ın muazzam büyüklüğünden dolayı böcekler içine yuva yapabilmekte ve elektrik kablolarını kemirmekteydiler. Bu yüzden ENIAC sık sık hata yapmakta ya da bozulmaktaydı. Günümüzde bilgisayarlarda oluşan hataları tanımlaman için kullanılan "böcek" (bug) kavramı adını ENIAC'ta böcekler yüzünden oluşan hatalardan almıştır. ENIAC ile mekanik donanımın yerine elektronik devreler kullanılmaya başlanarak bilgisayar teknolojisinde yeni bir devrim yapılmıştır.

Ticari amaçlarla kullanılabilen ve seri halde üretimi yapılan ilk bilgisayar UNIVAC I olmuştur. Bilgisayarın giriş-çıkış birimleri manyetik bant olarak tasarlanmış ve bir yazıcı bilgisayara eklenmiştir. Aynı yıllarda IBM 701 bilgisayarı piyasaya çıkmış, bu bilgisayar vakum tüplü ve kolayca programlanabilen yapısı sayesinde çok büyük ilgi görmüştür. IBM firması 1958'den itibaren bilgisayarda vakum tüpleri yerine diyot ve

transistörleri kullanmaya başlamış, buna bağlı olarak daha küçük, daha hafif ve daha az ısınan bilgisayarlar pazarlanmaya başlamıştır. Ayrıca bilgi depolama ortamları olarak delikli kartlar yerine teker (disk) ve tamburlar kullanılmaya başlanmıştır.

1964 yılından itibaren transistörler ve diğer devre elemanları ile oluşturulan devrelerin yerini bu devrelere göre daha küçük kalan bütünleşik devrelerin (IC – **I**ntegrated **C**ircuit) alması bilgisayar alanındaki gelişmelere ivme kazandırmış; daha hızlı, güvenilir ve daha ucuz bilgisayarlar üretilmeye başlanmıştır. 1970 yılından itibaren geniş çapta bütünleşik devrelerin kullanılmaya başlanmasıyla bilgisayar tasarımını yeni boyutlar kazanmıştır. Özellikle 1993 yılından itibaren geniş bellekli ve hızlı bilgisayarların yanı sıra güçlü programlama dilleri ve işletim sistemleri ortaya çıkmıştır. Artık eski bilgisayarlarda kullanılan çekirdek bellek yerine daha ucuz manyetik iç bellekler kullanılmakta ve bilgisayar maliyetleri gün geçtikçe düşmektedir. 1993'de Internet teknolojisinin HTML dilinin icadı ile daha kullanılabilir hale gelmesiyle bilgisayarlar tam olarak günlük hayatımızda yerlerini almaya başlamışlardır. Bilgisayar alanında bu tarihten sonra oluşan her gelişim günlük hayatı büyük ölçüde etkilemiştir.

1.3. Bilgisayarın Bileşenleri

Bilgisayarların bileşenleri donanım ve yazılım olmak üzere ikiye ayrılabilir. Bileşenler arasında böyle bir ayrım yapılsa da bilgisayarların düzgün olarak çalışabilmeleri için donanım ve yazılımın birbirleriyle uyum içinde olmalıdır.

1.3.1. Donanım

Donanım, bilgisayarın sahip olduğu tüm elektronik ve mekanik aksama verilen genel addır. Bilgisayarın gereksinim duyduğu gücü sağlamak, bilgi aktarımını düzenlemek, ekrana görüntü yansıtmak gibi işler donanımın denetimindedir. Donanım, birçok farklı görevdeki parçanın birleşmesi ile oluşur. Donanımın düzgün çalışması için donanımı oluşturan tüm bileşenlerin yazılımla uyum içinde çalışması gerekir.

Donanım, Temel Birimler ve Çevre Birimleri olmak üzere ikiye ayrılıp incelenebilir.

1.3.1.1. Temel Birimler

Temel birimler, bir bilgisayarın sorunsuz çalışması için bulunması şart olan birimlerdir. Bu birimlerin herhangi birisi herhangi bir şekilde bir sorun yaratırsa bilgisayarın çalışması aksar, bilgisayar kendi kendine zarar verebilir.

1.3.1.1.1. Merkezi İşlem Birimi

Merkezi İşlem Birimi - MİB (**C**entral **P**rocessing **U**nit – CPU) bilgisayarın beynidir. Bir bilgisayarın işlemcisinin başlıca görevi kullanıcı programının komutlarını alarak işlemek ve bu komutların işlenmesi sonucunda ortaya çıkan geçici verileri saklamaktır. İşlemci aynı zamanda belleğin tutarlı biçimde güncellenmesinden sorumludur.

Günümüzde bir MİB, çok küçük ölçekli devreler sınıfında tasarlanıp üretilmektedir. İşlemcilerin içerisinde milyonlarca transistör bulundurur. Örneğin Intel Core Duo işlemcisi yaklaşık olarak 151 milyon transistör içerir. Bir MİB, 1850'de Michael Boole'nin bulmuş olduğu Boole cebiri mantığı ile çalışır. Boole cebiri, 1 ve 0'lardan oluşan değerlerle işlem yapma mantığı üzerine kurulmuştur.

MİB, bilgisayarda gerekli olan tüm düzenlemeleri yapan donanım aracı olarak da tanımlanabilir. İşlemci yapılan İşlemleri düzene sokar ve tüm diğer elemanların birbirleriyle eşgüdüm içinde çalışmasını sağlar. Merkezi işlem birimlerinde yarı iletken malzemeler kullanılmaktadır. Örneğin kullanılan transistörler silisyum tetiklemeli ya da germanyum tetiklemeli olabilir. Daha çok germanyum kullanıldığı görülmektedir. Bir MİB'in iç mimarisi oldukça karmaşıktır.

MİB'in en önemli kısımlarından birisi ön bellektir (cache). Ön bellek, bilgisayara herhangi bir biçimde girilen bir verinin kullanıldıktan sonra gelecekteki kullanımlarında MİB'in veriye ulaşımının zaman ve işlem bakımından uzun süreceği düşünüldüğünden üretilmiş bir bilgisayar donanımıdır. Ön bellek, bir tür RAM olarak da düşünülebilir, ancak RAM'dan çok daha hızlıdır. Ön bellek, temel olarak kullandığı verilerin kaynaklarını –bilgisayarın sabit disklerinde saklanmış olan bir veri için- kendisine kaydeder, bu özelliği sayesinde gelecekte bilgisayarın veriye ulaşımı çok daha kolay olur. Böylece MİB'in veriye ulaşım süresi azaltılır ve bu sayede işlem süresi bakımından büyük bir kazanç sağlanır. Bilgisayarın kullanımında ön belleğin çok gerekli olduğu yapılan araştırmalarla da kanıtlanmıştır. Ön bellek, sabit diskte aynı fiziksel adrese sahip olmayan ve birlikte gruplanmış olarak bulunmayan verilerin ulaşımında bir nevi kestirme yol kullanmaktadır.

Ön bellek, veri alımı için yapılan girişleri kendi hafızasında bir havuzda toplar. Her giriş, bir dizine ve etikete sahiptir. Etikette gerçek adres bulunur. Dizin ise önbelleğin göreli bir adresidir. MİB bir veri okumak istediğinde öncelikle önbellek kontrol edilir. Eğer istenen veri önbellekte etiket aracılığıyla bulunabiliyorsa önbellekten veri okunur. Bu duruma bilgisayar literatüründe "cache hit" yani "önbellek vurdu" denir. Aksi halde "cache miss" yani "önbellek ıskaladı" durumu oluşur. Bu durum oluştuğunda veri sabit diskette veya başka bir depolama biriminde aranır.

Eğer ön bellek kayıt limitine ulaşırsa az kullanılan veriler önbellekten silinir ve yani kullandıkları dizinler boşaltılır ve yeni verilere yer açılır. Bu işleme önbelleğin yer değiştirme işlemi denir. En popüler yer değiştirme işlemi teknolojilerinden olan LRU (**L**east **R**ecently **U**sed- En az sıklıkta kullanılan) en az sıklıkta kullanılan girişi silerek yerine yeni gelen girişi kaydeder. Daha yeni teknolojilere sahip olan yer değiştirmelerde ise, verinin girişi yerine sahip olduğu frekans kullanılır. Bu teknolojilerin eskilere göre daha etkili olduğu kesinlikle söylenebilir.

1.3.1.1.2. Ana kart

Ana kart, kasanın içinde bulunan donanım parçalarının tümünün üstüne monte edildiği dev bir veri yolu ağı olarak düşünülebilir. Bilgisayarın ayarlama merkezi olan BIOS da ana kart üzerinde bulunmaktadır.

Peki BIOS nedir? BIOS, bilgisayarcılıkta basit giriş/çıkış sistemi olarak bilinir. Bilgisayar çalıştırıldığında saklama birimlerinde yüklü bulunan diğer programların çalışabilmesi için BIOS aktive edilir ve bilgisayar, BIOS'da bulunan yönetim kodları sayesinde belli bir düzende çalışmaya başlar. BIOS'un bir çipin üzerine yazılmış olan ve bilgisayarın tüm çalışma sistemlerini denetimi altında tutan küçük bir kod dizilim sistemi olduğu da söylenebilir. BIOS, bilgisayar ilk açıldığında ROM, EPROM veya bir flash diskten başlatılır. Bu işlemden sonra BIOS'un içinde barındırdığı kodlar belleğe yüklenir ve işlemci bu kodları işlemeye başlar. BIOS bilgisayar açıldığında otomatik bir kontrol işlemine başlayıp bilgisayara bağlı olan tüm aletleri tarar ve kontrol eder. Bilgisayarın BIOS ayarlarının rastgele değişmesi tehlikeli sonuçlar doğurabileceğinden, bu ayarlarda bir hata oluştuğunda fabrika çıkış ayarlarına geri dönülmelidir.

Ana kart üzerindeki parçalar genellikle kart yuvalarına (slot) yerleşirken, bazen bazı parçalar Ana kart üzerine entegre edilmiş olarak gelebilir. Bu durumdaki parçalara bütünleşik (on board) parçalar denmektedir. Özellikle Ethernet kartları bütünleşik üretilirken, bazen ekran ve ses kartlarının ve hatta işlemcinin bile bütünleşik olarak üretildiği görülebilir. Bütünleşik üretime genellikle dizüstü bilgisayarlarda rastlanmaktadır.

Ana kart için basit bir örnek vermek gerekirse, bir ülke düşünelim, MİB bu ülkenin başkenti, diğer parçalar ülkenin şehirleri olsun. O ülkede lojistik anlamda ve haberleşme anlamında gereken tüm uygulamaları Ana kart yapar, yani Ana kart bir nevi ülkenin yol, telefon ve haberleşme ağı olarak görülebilir. Ana kart, bilgisayarın performansını en çok etkileyen parçalardandır. Hızlı bir işlemci eğer yeterli hızda veri yollarına sahip değilse tam hızda çalışamaz. Aynı durum diğer parçalar için de geçerlidir. Eğer veri yollarında bir aksama veya bir yavaşlama olursa, bilgisayarın performansı bir anda düşer ve bilgisayarda olası bozukluklara neden olur.

Ana kart, aynı zamanda bilgisayarda güç dağıtımını yapan parçadır. Güç girişi de diğer tüm dış girişler gibi ana kartın üzerindedir. Düzgün bir sistem performansı elde edebilmek için sistemin güç beslemesinin düzenli ve yeterli olması gereklidir. Günümüzde ana kartlar olası voltaj dalgalanmalarına karşı bilgisayarın diğer birimlerini korumak amacıyla özel donanımsal parçalarla donatılmıştır.

1.3.1.1.3. Ekran Kartı

Ekran kartı tanım olarak, hafızada sayısal durumda saklanmış olan bir imgenin mantıksal gösterimini elektriksel sinyallere dönüştürüp herhangi görüntü ortamına (genellikle bir monitöre) çıktı olarak gönderilmesini sağlayan araçtır. Ekran kartları, hafızalarına ve işlem frekanslarına göre performans gösterip ona göre görüntünün kalitesini yükseltip alçaltabilirler. Yeni nesil ekran kartları içlerinde kendi işlemcilerini barındırmakta olup özellikle oyun oynama amaçlı kullanılan kişisel bilgisayarlarda bu kartlar yüksek performans göstermektedirler.

Ekran kartı, her görüntüyü çıkış ünitesine piksellerine ayırarak gönderir. Tabii ki burada performans açısından çıkış ünitesinin desteklediği maksimum çözünürlük de önemlidir.

Bir görüntü alınması gerektiğinde basit olarak şu işlem gerçekleşir:

Kullanıcının herhangi bir giriş ünitesi yardımıyla girdiği ham veri, ana kart yardımıyla işlemciye gönderilir ve burada işlemci, sabit disk veya RAM'dan gerekli bilgiyi çağırır. İşlemci gerekli uygulamaları ve karşılaştırmaları yaptıktan sonra veriyi işlemeyi tamamlar. Veri işlemesi tamamlanınca ana kart, işlemciden aldığı komutlarla işlenmiş veriyi ekran kartına transfer eder. Bu andan itibaren devreye giren ekran kartı,

işlemciden gelen sayısal bilgileri çözümleyerek pikselleri ayırmayı kendi mantığına göre tanımlar ve elde ettiği çıktıyı çıkış birimine gönderir. Bir piksel, tüm sayısal görüntülerin en küçük parçasına eşittir. Yani bir veri ile tanımlanabilecek en küçük görüntü parçacığıdır. Çözünürlük ise, bir aygıtın desteklediği en büyük piksel dizilimi miktarıdır. Çözünürlük büyüdükçe görüntü kalitesi yükselir ama gösterilen verinin bellek veya sabit diskette kapladığı alan ve gerektirdiği işlem sayısı piksel sayısı arttığından artar.

Ekran kartının yani modern ismiyle grafik işleme biriminin tarihsel gelişim basamaklarına değinelim. Yeni nesil ekran kartları 1970 ve 1980'li yılların tek parça üretilen (monolithic) ekran kartlarının geliştirilmiş modelleridir. Eski kartlar, kısıtlı BitBLT (Bit Block Transfer) desteğine sahiptirler. Biraz daha açmak gerekirse, iki boyutlu grafik ve animasyon destekleri çok azdır ve genellikle şekil çizim desteğine sahip değildirler. Bu nesildeki bazı grafik işlemciler, bir ekrana yazdırma işlem tablosunda birkaç işlemi aynı anda yapıp DMA (**D**irect **M**emory **A**ccess- Direk Hafıza Erişimi) kullanarak görüntü yollayan sunucunun üstündeki yükü azaltabilirler. Bu tip ekran kartlarına önceki nesillerden bir örnek vermek istersek, ANTIC yardımcı işletim sistemi, Atari 800 ve Atari 5200 modellerinde kullanılmıştır. 1980'lerin sonunda ve 1990'ların başında yüksek hızlı ve genel amaçlı mikroişlemciler yüksek çıkışlı grafik işlemcilere entegre edilmeye ve popülerliği iyice artmaya başlamıştır.

Çip işlemci teknolojisi geliştikçe, BitBLT fonksiyonlarını ve çizimleri aynı tabana hatta aynı çipe taşımak mümkün olmuştur. Bu teknolojinin asıl kaynağı VGA'nın (**V**ideo **G**raphics **A**rray- Video Grafik Dizisi) kullanılmasıyla başlar. Bu 2D hızlandırıcılar mikroişlemci tabanlı ekran kartları kadar esnek olmasalar da, yapımları kolay olduğu için ve dünya çapında çok satıldığı için popüler olmuşlardır.

1980'lerde ise durum biraz daha değişmiştir. Commodore Amiga ile yani video donanımında bir belleğe ve bellekten veri transferi için özel olarak üretilen ek işlemcisi ve IBM'in yeni teknolojisi olan 8514 grafik sistemini içeren 2D tabanlı bilgisayarını piyasaya sürmüştür. 1985 de ortaya çıkan "oyun konsolları krizinin" (The Console Crashdown) nedenlerinden biri Amiga'nın oyun konsollarından daha ucuza satılması ve oyun dışında da kullanıma müsait olmasıdır. Amiga, ilk tam video hızlandırıcıya sahip olması dolayısıyla tamamen özgündü. Tüm video işlemlerini pratik olarak donanıma yüklemesi de zamanında onu tercih edilen bir sistem yapan özelliklerindendir. Nintendo firmasının ürettiği NES oyun konsolu ise oyun konsolları krizini aşan ilk konsol olarak görülmektedir. İçinde grafiksel işlemler için özel bir PPU (**P**icture **P**rocessing **U**nit – Resim İşleme Birimi) bulunmakta idi. Günümüzde üretilen gelişmiş konsollarda kutu tasarımına kadar NES etkisini görebiliriz.

Günümüzde simülasyon teknolojilerine gereksinimlerin artması, oyun sektörünün büyük ölçüde gelişmesi ve sürekli daha gerçekçi grafiklerin oyunlarda kullanılmak istenmesi ve sanal gerçeklik teknolojisinin geliştirilmeye çalışılması nedeniyle ekran kartları bir hayli gelişmiştir. Günümüzde ekran kartları başlı başına "ana kartı" olan, işlemci, ram ve giriş/çıkış birimlerine sahip olan bir bilgisayar şeklini almıştır. Bazı ekran kartları paralel bağlanarak daha fazla performans elde edilebilmektedir.

1.3.1.1.4. Sabit Disk

Sabit disk (HDD – **H**ard **D**isk **D**rive) bir bilgisayarda bulunan tüm verilerin depolandığı manyetik bir kayıt ortamıdır. Günümüzde dakikada 10000 devir/dakika (rpm) hıza kadar çıkan sabit diskler üretilmiş olsa bile, yapıları mekanik olduğu için diğer bellek elemanlarından daha yavaştırlar. Sabit diskler, bilgisayarlarda yardımcı ve kalıcı depo alanı olarak kullanılırlar. Bir program işletilmeye başlandığında, programın çalışması için gerekli olan bilgiler işlemci tarafından sabit diskten okunarak çok daha hızlı ve sabit disketin aksine elektronik yapıda olan RAM belleğe aktarılır. Gereksinim duyulan kısım RAM'a sığmayacak kadar büyükse, bilgisayar sabit diskin bir bölümünü RAM bellek gibi kullanabilme opsiyonuna sahiptir.

Sabit disklerde veri yazımı; metal, cam veya plastikten yapılmış, yüzeyi FeO ya da başka manyetik özellikteki malzeme ile kaplı diskler üzerine yapılır. Bu diskler katmanlar şeklinde birbirlerinin üzerine binmişlerdir. Bu kayıt ortamlarında veriler mıknatıslanma yolu ile kaydedildiğinden kullanıcı tarafından silinene kadar sabit kalırlar, elektrik kesintileri gibi durumlarda bilgisayar bellek parçalarındaki gibi kaybolmazlar, bu nedenle sabit disk şeklinde adlandırılmışlardır.

Sabit disk içerisinde bulunan verileri okumak için gerekli olan sistem, hareketli bir kafa olarak tasarlanmıştır. Bu kafanın sabit disk yüzeyine uzaklığı bir saç telin kalınlığından daha az olduğundan (yeni nesil sabit disklerde bu uzaklık genel olarak birkaç nanometre boyutundan daha fazla değildir) bilgisayar çalışırken asla hareket ettirilmemelidir. Eğer sabit diski okuyan kafa manyetik yüzeye temas ederse sabit disk üzerinde bozuk alan (bad sector) oluşturabilir veya daha kötü ihtimalle sabit diski parçalayabilir. Masaüstü bilgisayarlarda 3.5 inç, dizüstü bilgisayarlarda ise 2.5 inç çaplarındaki sabit diskler kullanılır. Daha küçük sabit diskler ise taşınabilir mp3 çalar, taşınabilir diskler ve hafızalı telefon gibi aletlerde kullanılırlar.

Genel olarak sabit diske manyetik bir cisim yaklaştırıldığında verilerin kaybolduğundan bahsedilir. Gerçekte veriler büyük ölçüde kaybolmaz. Sadece küçük değişiklikler yüzünden dosyalar okunamaz. Bazı özel

yazılımlar sayesinde bu okunamayan dosyaların bozuk verileri tespit edilebilmekte ve düzeltilebilmektedir. Bu tür yazılımlar genellikle alt seviye işlemler gerektirdiğinden çevirici dili kullanılarak yazılır.

Sabit diskler, genel olarak birkaç yol tipine sahiptirler. Bunlar; ATA (IDE, EIDE), Serial ATA (SATA), SCSI, SAS, IEEE1394, USB ve Fiber Kanallarıdır. ST-506 ara yüzünün kullanıldığı günlerde yani ilk ST-506'ların üretiminde eski 8 inçlik diskler yerine 5.25 inçlik diskler üretilmeye başlanmıştır. İlk ST-506'nın kapasitesi 5 MB dir. Veri transfer hızı saniyede 5 Mbit gibi şu andaki hıza göre çok küçük bir değer taşımaktadır. Daha sonra ortaya çıkan Genişletilmiş Küçük Disk Ara yüzü (ESDI) ise veri seviyelerini katlayabilme özelliğini desteklemiş, ancak bu özellik genel olarak kullanıcı ve sistem arasında otomatik olarak atanmıştır. Ancak bu sistemde şöyle bir sorun bulunmaktadır, ESDI arayüzler, daha aşağı nitelikteki ara yüzlerle uyumlu değildirler. Örneğin 20 veya 25 megabitlik bir sürücü 15 megabitlik bir işlemcinin üzerinde çalışamamaktadır. ESDI sürücüler, sektör dağılımlarını artırıp azaltmak için kullanılabilecek atlama kablolarına sahiptirler. Bu ayar sayesinde bilgisayarın sahip olduğu performans direk kullanıcı tarafından ayarlanabilir. Eğer atlama kablo ayarları gereğinden fazla artırılırsa aşırı yükleme dolayısıyla güç dengesizliği oluşur ve disk yanabilir.

Veri transferinde SCSI (**S**mall **C**omputer **S**ystem **I**nterface - Küçük Bilgisayar Sistem Arayüzü) teknolojisi öne çıkmaya başladığında, öncelikle SCSI için sabit bir hız belirlenmişken (saniyede 5 MB), sonraları teknoloji geliştikçe veri taşıma hızı katlanarak artmıştır. Daha sonra ortaya çıkan IEEE 1394 ve USB (**U**niversal **S**erial **B**us) teknolojileri sayesinde günümüzde çok yüksek hızlarla sabit diskten veri transferi yapılabilmektedir. SCSI, ESDI ile geliştirilmiş olan bir sistemdir. SCSI sürücüler; sunucular, çalışma istasyonları ve Apple bilgisayarlarda standart olarak bulunmaktaydı. 1990'lardan sonra çoğu sistemin IDE (Yükseltgenmiş Sürücü Ara yüzü) teknolojisine geçmeye başlamasıyla SCSI'ların popülaritesi azalmıştır. 2005 yılında ise SCSI ailesinden sürücülerin kapasiteleri IDE'ler tarafından tamamen geçilmiş olmasına rağmen en hızlı sürücüler hala SCSI teknolojisiyle üretilmektedir.

SATA (**S**erial **ATA**) veri kablosu aygıtla disk arasındaki diferansiyel veri transferinde veri göndermek ve veri almak için birer veri yolu çiftine sahiptir. Bu sebeple SATA'da veri seri olarak gönderilip alınır. Günümüzde kullanılan SATA ara yüzünün ortalama hızı 3.0 Gbit/sn dir. SCSI ara yüzü, SATA ile aynı diferansiyel veri transfer yöntemini kullanmıştır. 2006 sezonunun başlarında geliştirilen 40 iğnelik yeni teknolojiye ise PATA (Paralel ATA) adı verilmiştir. Bu teknolojide birim sürede veri kablosundan paralel olarak 16 bitlik veri akışı sağlanabilmektedir.

1.3.1.1.5. RAM

RAM (**R**andom **A**ccess **M**emory - Rastgele Erişimli Bellek), bilgisayarda kayıtlı bir veriye herhangi bir emirle rastgele olarak ulaşılmasını sağlayan bellek tipidir. Diğer hafıza araçlarında (örneğin bir manyetik sabit diskte) o diskin sahip olduğu mekanik tasarıma göre şekillendirilmiş bir yolu izleyerek veriye ulaşılabilirken, RAM'da veriye hızlı ve kısa yoldan ulaşmak mümkündür.

Genel olarak bir bilgisayarda RAM ana hafıza veya ana depo birimi olarak adlandırılmaktadır. Bunun sebebi, RAM'ın bilgisayarda o anda yapılan tüm programları hafızasına alıp çalıştırmasıdır. Yani çalıştırdığınız tüm programlar RAM'ın üzerine yüklenir. RAM'de fazla yüklenme olduğu zaman bilgisayar performansı düşer. Buradan RAM'ın bilgisayarın performansını en çok etkileyen parçalardan biri olduğu görülebilir. İşlemciniz istediği kadar hızlı olsun, sabit diskiniz istediği kadar yüksek devir sayısına sahip olsun, eğer RAM'ınızın kapasitesi düşükse, çalıştırabileceğiniz program sayısı ve seviyesi her zaman sınırlı kalacaktır. Ana kart üzerindeki RAM veri yolunun hızı da çok önemlidir.

Bilgisayarların sistemi çalıştırırken RAM'a gerekli programları yüklemesi nedeniyle RAM'ın sabit diske göre çok daha hızlı olması gerekmektedir. Bu nedenle sabit disk gibi bir bellek tipi olan RAM, sabit diskten farklı olarak manyetik/mekanik karışımı bir sistem yerine tamamen elektronik bir sistem üzerine tasarlanmıştır. Bu özellik sayesinde RAM üzerinde bulunan tüm program kodlarına, hangi bellek noktalarında bulunurlarsa bulunsunlar, hemen hemen aynı hızda ulaşılabilinmektedir.

RAM'ın en önemli özelliği ise geçici bir bellek olmasıdır. Yani bilgisayar çalışmaya başladığı andan itibaren RAM'a gerekli veriler yüklenir, güç kapatıldığı anda ise RAM üzerinde bulunan tüm veriler silinir. Burada RAM, diğer bellek tiplerinden dezavantaj sayılabilecek bir farkla ayrılmaktadır. Sizin üzerinde çalıştığınız herhangi bir dosya sistem açıkken anlık olarak RAM'a kaydedildiğinden, eğer dosya üzerinde yaptığınız değişiklikleri kaydetmediyseniz güç kesintisi anında sizin yaptığınız tüm kaydedilmemiş çalışmalar kaybolacaktır. Günümüzde RAM'ların bu dezavantajını ortadan kaldırmak için çalışmalar yürütülmektedir. Bazı RAM üreticisi şirketler, gelecekte üretilebilecek bir teknoloji olarak geçici olmayan RAM üretmeye çalışmaktadırlar. Klasik elektrik devreleriyle böyle bir sistem çok karmaşık ve maliyetli olacağından, güç kesildiğinde bile üzerinde bulunan verileri koruyacak şekilde bir RAM üretebilmek için bilim adamları değişik

teknolojilere başvurmayı denemektedirler. Şu anda bu iş için kullanılması denenen teknolojiler ise karbon nanotüp teknolojisi ve manyetik tünel etkisidir.

RAM'ların tek özelliği veri saklamak değildir. Bazı yeni nesil RAM'lar sistemin çalışması esnasında ortaya çıkabilecek olan kayıtlı verideki rastgele hataları RAM paritesi kullanarak algılayabilmekte ve algıladıkları bu hataları düzeltebilmektedir. Bu sistem sayesinde bilgisayar çalışırken ortaya çıkabilecek olan olası sistem hataları engellenmektedir.

2003 yazında, 128 KB'lık ve 0.18 µm teknolojisiyle üretilmiş ilk manyetik RAM bilgisayar piyasasına tanıtılmıştır. Bu tasarımdaki manyetik RAM'ın çekirdek dizaynı tamamen manyetik tünel etkisi teknolojisi kullanılarak yapılmıştır. 128 KB elbette günümüzde kullanılan yüksek kapasiteli RAM'lara karşı komik bir rakam olarak kalmaktadır, ancak sabit disk gibi kalıcı belleğe sahip olan RAM teknolojisinin henüz geliştirme aşamasının bitmediği de göz önünde bulundurulmalıdır. Zaten 2004 yılına gelindiğinde Infineon Technologies şirketinin tamamen aynı tasarımda ve aynı boyuttaki 18 MB hafızaya sahip bir manyetik RAM geliştirdiği görülebilir. Manyetik tünel etkili RAM teknolojisi, karbon nanotüplü kalıcı RAM teknolojisine göre biraz daha az tercih edilmektedir. Geleceğin nanoteknolojide olacağı düşüncesi göz önünde bulundurularak kalıcı RAM üretimin Ar-Ge'sinde öncelik karbon nanotüplerle yapılan araştırmalara verilmiştir. Örnek vermek gerekirse, 10 GB'lik karbon nanotüp serisine sahip bir kalıcı RAM tasarımı için Ar-Ge çalışmaları 2004 yılının sonlarında başlatılmıştır.

Kalıcı RAM teknolojisi, bilgisayar için yapılan Ar-Ge çalışmalarından belki de en önemlisidir. Bu teknoloji üretildiği an bilgisayarda yapılan hiçbir işlemin siz istemedikçe kaybolmayacağı, bilgisayarın elektrik verildiği anda yüklemeye gerek kalmadan çalışacağı aşikardır. Nanoteknoloji yardımıyla üretilecek olduğu için, günümüzde kullanılan RAM'lara göre oldukça hızlı bir teknolojiye sahip olacaktır. Kullanılacak kurtarma sistemleri ile sık sık rastlanan sistem hataları hemen hemen sıfıra inecek, hata meydana gelse de RAM kalıcı olduğu için gerekli dosyalar zarar görmeyecektir.

1.3.1.2. Çevre Birimleri

Bilgisayarın temel birimlerine bir kullanıcı ara yüzü sağlamak amacıyla çevre birimleri kullanılır. Çevre birimlerinden bazılarını tanıyalım.

1.3.1.2.1. Klavye

Delikli kartların ardından ilk klavye tipleri ortaya çıkmaya başladı ve 1980 den sonra neredeyse tüm bilgisayarlar klavye teknolojisi ile donatılmaya başlandı. Klavyeden sonra birçok kullanıcı girdisi almaya yarayan çevre birimi piyasaya sürülmesine rağmen klavye günümüzde bile hala bir numaralı girdi birimidir.

Klavyenin çalışma mantığı çok basittir. Klavyenin bir tuşuna basıldığında tuşun altında bulunan bir tel bu tuş için ayrılmış ana iletim teline değer ve iletim sağlanır. İletim sağlandığında ana iletim telinin bağlı olduğu deneyletici çipe bir sinyal gider ve bu çip gelen sinyalin hangi veri yolundan geldiğine bakarak bir kod oluşturur ve bu kodu bilgisayara gönderir. Bilgisayardaki bir çip ise bu kodu çözer ve basılan tuşun görevi yerine getirilir.

1.3.1.2.2. Fare

Fare Douglas Engelbart tarafından 1963 de icat edilmiştir. Fare aslında fare benzeri ama vücudun farklı bölgeleri ile kontrol edilen bir dizi icat içinden seçilmiştir. Fare en kolay ve alışılmış kullanıma sahip olduğundan tercih edilmiştir.

Fare ile ekrandaki bir imlecin x ve y koordinatları değiştirilebilmektedir. Yani fare ileri yönde hareket ettirildiğinde y koordinatı azaltılır, geri hareket ettirildiğinde y koordinatı artar. Sağa hareket ettirildiğinde x koordinatı artar ve sola doğru hareket ettirildiğinde x koordinatı azalır. Burada dikkat edilmesi gereken şey koordinat sisteminin sol üst köşeden başlamasıdır. Aslında bu bir uzlaşımdır. Tasarladığınız sistemde fare ile çok farklı işlemler yapabilirsiniz.

1.3.1.2.3. Yazıcı

Bir yazıcı ile bilgisayar ortamındaki bir verinin basılı kopyası çıkarılabilir. Farklı yazıcı çeşitleri bulunmaktadır.

"Karakter yazıcılar" ilk otomatik yazıcı türüdür. Tasarım olarak daktilolara çok benzemektedirler. Baskı şeridi üzerine vuruşlar yapılarak karakter yazılmaktadır. İlerleyen zamanlarda daha farklı türleri de piyasaya sürülmüştür. Örnek vermek gerekirse aynı anda bir satırı dolduran karakter yazıcılar da üretilmiştir. Karmaşık ve yavaş olduklarından artık üretilmemektedirler.

"Nokta vuruşlu yazıcılar" ise bilgisayardan gönderilen verileri baskı iğnelerini uygun sırada kağıdın üstünde bulunan şeride vurarak baskı yapan yazıcılardır. Hızlarının düşük ve çok sesli çalışmalarına rağmen harf ve rakam basmak için uygun olduklarından günümüzde halen kullanılmaktadırlar.

"Mürekkep püskürtmeli yazıcılar" mantık olarak nokta vuruşlu yazıcılar ile aynı şekilde çalışmaktadırlar. İğne ile şeride vurup kağıda iz bırakmak yerine kağıdın üzerindeki küçük noktalara yüksek hızda boya püskürtmektedirler. Bu yüzden grafik baskısında da kullanılabilmektedirler. Çözünürlükleri çok yüksektir. Daha eski yazıcılara göre sessiz ve hızlıdırlar. Farklı boya püskürtme yöntemleri kullanılmaktadır. Isıl püskürtmeli yazıcılar mürekkebi yüksek sıcaklıklara ısıtıp ortaya çıkan patlamanın verdiği hız ile kağıda püskürtmektedirler. Piezoelektrik püskürtmeli yazıcılarda ise piezoelektrik kristallerin titreşimi ile mürekkep püskürtülmektedir. Kullanılacak olan mürekkepte herhangi bir ısıl özellik aranmadığından ısıl püskürtmeli yazıcılara göre daha yaygın olarak kullanılmaktadırlar.

"Lazer yazıcılar" sessiz, yüksek baskı kalitesine sahip ve hızlıdır. Bilgisayardan gelen veriler lazer yardımıyla ışığa duyarlı baskı davuluna çizilir. Bu sayede çizilen bölgeler statik elektrik ile yüklenir ve baskı tozları bu bölgelere yapışır. Davul dönerken ortam ısıtılarak kağıdın üzerine bu tozların yapışması sağlanır ve ardından da soğutulur.

"Isıl yazıcılar" ise ısıya duyarlı kağıtlar üzerine baskı yapabilen yazıcılardandır. Isıl yazıcılarda ince çizgi olarak dizilmiş sabit ısıtıcı birimler bulunmaktadırlar. Kağıt geçerken bu ısıtıcı birimler çalıştırılarak kağıdın birim altında bulunan bölgesine baskı yapılabilmektedir. Hızlı ve sessiz çalışmaktadırlar. Yüksek baskı kalitesine sahip değildirler bu yüzden de bilet, ATM çıktıları ve fişlerde kullanılırlar.

1.3.2. Yazılım

Yazılım bir bilgisayarın en önemli parçalarından biridir. Donanımın düzgün, koordineli ve istenen şekilde çalışmasını sağlar. Günümüzde bilgisayarlardaki en temel yazılım işletim sistemidir. İşletim sisteminin denetimi altında ise diğer yazılımlar çalışır. Yazılım geliştirmek için programlama dilleri icat edilmiştir.

Günümüzde birçok programlama dili mevcuttur. Farklı alanlar için programlama dilleri geliştirilmiştir. Programlama dilleri sınıflandırılırken ise aslında genel geçer bir kural yoktur. Bazı kitaplarda işleve göre, bazılarında derleyicilerine göre, bazılarında ise tarihe göre sınıflandırılır. Biz ise burada programlama dillerini tarihsel açıdan getirdikleri yeniliklere göre gruplara ayırarak listeleyeceğiz.

1.3.2.1. 1. Grup – Makine Dilleri

Makine dilleri 0 ve 1'lerden oluşan en alt seviyeli dillerdir. Bu diller tamamen işlemci ve donanıma bağlı olarak değişmektedir. Bu yüzden de günümüzde direk olarak kullanılmamaktadır.

Makine diline verilebilecek ilk örnek ilk tabii ki ilk bilgisayardaki makine dilidir. 1943 senesinde ENIAC makinesi için geliştirilen makine dili buna bir örnektir.

Günümüzde üretilen her bilgisayar için özel bir makine dili vardır.

Programlamada oluşturduğu zorluklardan dolayı makine dili kullanılmamaktadır. Makine dili yerine günümüzde kullanılan en alt seviyeli diller çevirici dilleridir.

1.3.2.2. 2. Grup – Çevirici Dilleri

Her işlemci platformu için farklı çevirici dilleri mevcuttur. Bir Motorola işlemci için farklı bir çevirici dili varken bir Intel işlemci için farklı bir çevirici dili vardır. PIC gibi mikrodenetleyiciler için de çevirici dilleri mevcutur.

Bu kitapta x86 mimarisi için geliştirilmiş olan çevirici dili incelenmektedir. Aşağıda Merhaba Dunya! İletisini size veren birkaç farklı çevirici dili örneği bulabilirsiniz.

x86 ASM

İşletim Sistemsiz:

```
MOV     AH,03H
INT     10H
MOV     AL,01H
MOV     BH,00H
MOV     BL,01001111B
MOV     CX,MESAJ_SON-MESAJ_BAS
PUSH    CS
```

```
POP     ES
MOV     BP, MESAJ_BAS
MOV     AH,13H
INT     10H
JMP     SON
MESAJ_BAS       DB      'Merhaba Dunya!'
MESAJ_SON       DB      00H
SON:    RET
```

Yukarıdaki kod herhangi bir işletim sistemi gerektirmeden çalışabilmektedir. Aşağıdaki örnekler DOS ve Windows'da çalışan örnekler olduğundan DOS, Windows ve Linux için de örnekler verelim.

DOS:

```
MOV     AX,CS
MOV     DS,AX
MOV     AH,09H
MOV     DX,OFFSET MESAJ
INT     21H
XOR     AX,AX
INT     21H
MESAJ:
DB      "Merhaba Dunya!",13,10,"$"
```

WINDOWS:

```
TITLE Merhaba Dunya ASM. Tasm
VERSION T310
Model use32 Flat,StdCall
start_code segment byte public 'code' use32
Begin:
        Call MessageBox,0,ofset sMesaj, ofset sBaslik,0
        Call ExitProcess,0
start_code Ends
start_data segment byte public 'data' use 32
sMesaj DB 'Merhaba Dunya',0
sBaslik DB "Merhaba!",0
start_data Ends
End begin
```

LINUX:

```
SECTION .data
mesaj           db "Merhaba Dunya!",0xa;
uzunluk equ $ - mesaj
SECTION .text
global main
main:
        mov     eax,4
        mov     ebx,1
        mov     ecx,mesaj
        mov     edx,uzunluk
        int     0x80
        mov      eax,1
        mov     ebx,0
        int     0x80
```

68000 Amiga ASM

```
move.l          #DOS
move.l          4.w,a6
jsr             -$0198(a6)
move.l          d0,a6
beq.s           .Out
```

```
move.l          #MerhabaDunya,d1
jsr             -$03B4
move.l          a6,a1
move.l          4.w,a6
jsr             -$019E(a6)
.Out    rts
DOS                     dc.b 'dos.library',0
MerhabaDunya  dc.b 'Merhaba Dunya!',$A,0
```

1.3.2.3. 3. Grup – Orta Seviyeli Diller

Orta seviyeli dillere örnek olarak C, C++, D, Java, C# ve ADA gibi dilleri verebiliriz. Java aslında orta seviyenin biraz daha üstünde bulunmaktadır. Bunun nedeni Java'nın derlendikten sonra farklı bir makine diline (Java Byte Code) çevrilmesidir.

Bu dillere örnek olarak aşağıdaki kodları verebiliriz.

C

```
#include <stdio.h>
int main(void)
{
        printf("Merhaba Dunya!\n");
        return 0;
}
```

C++

```
#include <iostream>
#include <ostream>

int main()
{
        std::cout << "Merhaba Dunya!" << std::endl;
        return 0;
}
```

D

```
import std.stdio;
void main(char[][] argumanlar)
{
        writef("Merhaba Dunya!\n");
}
```

Java

```
import java.io.*;
class MerhabaDunya{
        static public void main( String args[] )
        {
                System.out.println("Merhaba Dunya");
        }
}
```

C#

```
class MerhabaDunya
{
        static void Main()
        {
                System.Console.WriteLine("Merhaba Dunya!");
        }
}
```

ADA

```
with Ada.Text_IO;
procedure Merhaba is
begin
        Ada.Text_IO.Put_Line("Merhaba Dunya!");
end Merhaba
```

1.3.2.4. 4. Grup – Yüksek Seviyeli Diller

Bu diller algoritmik dillerdir. Fortran, Pascal, Cobol ve Basic'i gibi diller dördüncü guruba giren dillerdendir.

Bu dillere örnek olarak aşağıdaki kodları verebiliriz.

Fortran

```
WRITE (6,7)
FORMAT(14H MERHABA DUNYA)
STOP
END
```

Pascal

```
program MerhabaDunya(output);
        begin
                writeln('Merhaba Dunya!')
        end
```

Cobol

```
IDENTIFICATION DIVISION.
PROGRAM-ID MERHABA-DUNYA.
PROCEDURE DIVISION.
PARAGRAPH-1.
        DISPLAY "Merhaba Dunya!".
        STOP RUN.
```

Basic

```
10 REM Merhaba Dunya BASIC
20 PRINT "Merhaba Dunya!"
```

1.3.2.5. 5. Grup – Çok Yüksek Seviyeli Diller

Bu diller insan ile etkileşim açısından insana en yakın olan dillerdir. Görsel programlama gibi nitelikleri vardır. Bu dillere örnek olarak Visual Basic, Access, Paradox, XBase ve Foxpro verilebilir.

Visual Basic

```
Private Sub Command1_Click()
        Print "Merhaba Dunya!"
End Sub
```

1.3.2.6. 6. Grup – İnternet Dilleri

İnternet dilleri aslında "Çok Yüksek Seviyeli Diller" gurubu ile "Yüksek Seviyeli Diller" gurubu arasındaki dillerdir. HTML, Javascript, PHP bunlara örnektir.

Bu dillere örnek olarak aşağıdaki kodları verebiliriz.

HTML

```
<html>
<body>
<p>Merhaba Dunya!</p>
</body>
</html>
```

Javascript

```
document.write('Hello World');
```

PHP

```
<?php
echo 'Merhaba Dunya!';
?>
```

1.3.2.7. 7. Grup – Betik Dilleri

Betik dilleri Yüksek Seviyeli Diller seviyesinde ama betik amaçlı kullanılan dillerdir. Bu dillerin en büyük özelliği diğer diller ile beraber kullanılabilmeleridir. Bu dillere örnek olarak Python, Ruby ve Perl verilebilir.

Bu dillere örnek olarak aşağıdaki kodları verebiliriz.

Python

```
print "Merhaba Dunya!"
```

Ruby

```
puts "Merhaba Dunya!"
```

Perl

```
#!/usr/bin/perl
print "Merhaba Dunya!\n";
```

2. SAYI SİSTEMLERİ

2.1. Rakamlar

Sayılar ilk çağlardan beri insanlık için önemlidir. Sayılar sayesinde insanlar ellerinde hangi varlıktan ne kadar bulunduğunu bilebilmişler ve sahip oldukları şeylerin denetimini sağlayabilmişlerdir.

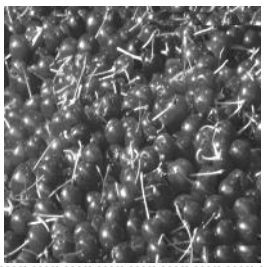
İlk insanlar büyüklükleri gösterebilmek için çizgileri kullanmışlardır. Örneğin bir elma için bir çizgi çizen insan, on elma için on çizgi çizmiştir. Bu düzen içerdiği tek rakam çizgi olan bir sayı sistemidir. Tek rakamlı bu sayı sistemine birli sayı sistemi denir. Aşağıda bu sayı sistemi kullanılarak bazı büyüklükler gösterilmiştir.

I I I I I Elma

I Şeftali

Birli sistemde, gösterilecek sayının büyüklüğü arttığında yazılan ifadenin okunması zorlaşmış ve bazı karışıklıklar ortaya çıkmıştır. Bu sorunu aşmak için I ile gösterilen çubuklardan öbekler oluşturulmaya başlanmıştır. 𝍸 gruplamasıyla beş tane I ifade edilmektedir. Böylece sayıların gösterimi daha düzenli hale getirilmiştir.

Çizgilerin art arda eklenmesinin ya da basitçe gruplandırılması bazı durumlarda pek de uygulanabilir değildir. Aşağıda bu durumlardan biri gösterilmiştir:



𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸... Kiraz

Gruplamaların büyük sayıları göstermede yetersiz kalması üzerine bir adım daha ileriye gidilmiş ve yaratılan gruplandırmalar da aralarında gruplandırılmaya başlanmıştır. Ayrıca gruplar kolaylık açısından artık simgelerle gösterilmeye başlanmıştır. “Bir” büyüklüğü “I” ile ifade edilirken, “Beş” büyüklüğü beş tane “I” yerine tek bir “V” ile “On” büyüklüğü ise on tane “I” ya da iki tane “V” yerine bir tane “X” ile gösterilmeye başlanmıştır.

Bu gruplama sistemini ilk olarak Romalılar kullandığından bu simgelere “Romen Rakamları” denir. Aşağıda Romen rakamlarının büyüklük tablosu verilmiştir:

Romen Rakamları	Büyüklük
I	1
V	5
X	10
L	50
C	100
D	500

M	1000

Bazı büyüklükler Romen rakamları kullanılarak aşağıdaki gibi gösterilir:

MCMLXXXVI → M – CM – LXXX – VI → 1000 + 900 + 80 + 6 = 1986

MMVI → M – M – VII → 1000 + 1000 + 7 = 2007

Romen sayı sistemi toplama ve çıkarma üzerine kuruludur. Örneklerde görüldüğü gibi herhangi bir sıfır kavramı bulunmamaktadır. Basamak gibi bir kavram da yoktur. Bu yüzden Romen rakamları çarpma, bölme gibi aritmetik işlemlerin yapılması için pek uygun değildir.

Bütün bu sorunlar yüzünden bugün kullanılan rakamlar ve sayı sistemi geliştirilmiştir. Günümüzde kullanılan 0, 1, 2, 3, 4, 5, 6, 7, 8 ve 9 rakamları Hint-Arap Rakamları'dır. Daha önceki sayı sistemlerinde bulunmayan sıfır ilk olarak Hintliler tarafından bulunmuştur. Hint-Arap sayı sisteminin temelleri büyük ölçüde Arap matematikçiler tarafından geliştirilmiş ve ardından tüm dünyaya yayılmıştır.

2.2. Rakamlar ve Bilgisayarlar

İçinde bulunduğumuz teknoloji çağının "Sayısal Çağ" adını almasında şüphesiz en büyük etken artık sayısal teknolojinin hayatın her evresine girmiş olmasıdır. Sayısal bilgisayarların en önemli özelliği yazılım adı verilen bir dizi komutu izleyebilmesi ve verilen girdi üzerinde işlem yapabilmesidir. Kullanıcı ise girdileri ya da yazılımı gereksinimlerine göre değiştirebilmektedir.

Sayısal bir sistemdeki bu "yazılımlar" aslında bir rakam topluluğudur ve bu rakamlar "veri" veya "komut" öbeklerini oluşturur. Yazılımları oluşturan verilerin ve komutların gerçek elektrik devrelerinde gerilim ve akım denetimi kullanılarak yaratılır. Günümüzde bütün sayısal bilgisayarlarda yüksek ve düşük olmak üzere iki çeşit sinyal kullanılmaktadır. Bu yüzden de bu sinyallere ikili sinyaller adı verilir. İkili düzendeki sayılar, 1 (yüksek) ve 0 (düşük) değerlerinden oluşur ve bu düzendeki sayıların 0 ve 1'den oluşan her bir basamağına "bit" denir.

Bitler, yani 0 ve 1'lerden oluşan diziler, kullanılarak her türlü veri ikili düzende gösterilebilir. İkili düzende oluşturulan bu dizilerin uzunlukları farklı olabilir. Oluşturulan bu dizilere aynı zamanda "kod" da denilebilir. Örneğin bilgisayarlarda sıfırdan dokuza kadar olan sayılar dört bit ile kodlanmıştır. Onluk düzende gösterilen sayıların ve diğer türde verilerin bu kodlanma biçimine ikili kodlama denir.

Rakam	İkilik Tabandaki Kod
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Sayısal sistemlerde veri olarak tanımlanabilen her şey bu bir ve sıfırlardan oluşmaktadır. Harfler, resimler, karakterler, sayılar, girdi ve çıktı gibi her türlü bilgi yalnızca iki tane rakamdan oluşan bir sayı sistemi kullanılarak gösterilmektedir.

Günlük hayatta gereksinim duyulan her türlü büyüklük 0, 1, 2, 3, 4, 5, 6, 7, 8 ve 9 rakamları kullanılarak gösterilir. Toplam on rakamdan oluştuğu için günlük hayatta kullanılan bu tabana onluk taban denir. Bilgisayarlar ise yalnızca 0 ve 1 rakamlarını kullandıklarından ikili tabana göre çalışırlar.

Özellikle çevirici dili (assembly) ile program yazarken sayı tabanlarının ve birbirleri arasındaki çevrimlerin bilinmesi çok önemlidir. Yazılımlar oluşturulurken ikili taban dışında onaltılık sayı tabanı da sıklıkla kullanılır. Onaltılık sayı tabanı uzun 0 ve 1 dizilerini kısaltmak amacıyla kullanılır. Onaltılık tabandaki bir sayıyı ikili tabana çevirmek ya da ikili tabandaki bir sayıyı onaltılık tabana çevirmek diğer tabanlar arasında çevrim yapmaktan daha kolaydır.

2.2.1. İkilik Taban

Onluk tabandaki her türlü sayı büyüklüğünün gösterilmesi için 0 dan 9 a kadar rakamlar kullanılır. İkili tabanda ise her sayı 0 ve 1 rakamları kullanılarak oluşturulur. Aşağıda ikilik tabanda gösterilen bazı sayı örnekleri verilmiştir:

$(10011)_2$
$(111)_2$
$(101111)_2$
$(101001)_2$
$(11111011011)_2$
$(01110011)_2$
$(10011111)_2$

Değişik sayı tabanındaki sayılar arasında dönüşüm yapmak mümkündür. Onluk tabandaki sayılar ikili tabandaki sayılara, ikilik tabandaki sayılar onaltılık tabandaki sayılara kolayca dönüştürülebilir.

2.2.1.1. İkilik - Onluk Taban Çevrimleri

Aşağıda onluk tabanda bir sayı gösterilmiştir:

12345

İnsanlar onluk tabanda işlem yaptıklarından sayıların onluk tabanda olduğunu belirtmek için herhangi bir işaretleme yapılmaz. Eğer bir işaretleme yapılması gerekirse aşağıdaki gibi yapılabilir:

$(12345)_{10}$

Yukarıda gösterilen sayı onluk tabandaki 12345 değerine eşittir. Bu sayının gerçek değeri diğer tüm tabanlardaki gibi taban dönüşümü kuralları kullanılarak bulunur. Aşağıda onluk düzendeki bu sayının değerinin basamaklarının ayrıştırılması yoluyla nasıl bulunduğu gösterilmiştir.

$(12345)_{10} = 5 \times 10^0 + 4 \times 10^1 + 3 \times 10^2 + 2 \times 10^3 + 1 \times 10^4$

$(12345)_{10} = 5 \times 1 + 4 \times 10 + 3 \times 100 + 2 \times 1000 + 1 \times 10000$

$(12345)_{10} = 5 + 40 + 300 + 2000 + 10000$

$(12345)_{10} = 12345$

Görüleceği gibi sayının en sağındaki haneden başlanarak basamaktaki rakam taban değerinin ilgili basamağın kuvveti ile çarpılarak sayıyı oluşturan rakamların sayıya kattıkları değer bulunmaktadır. Sayının toplam değeri her bir basamağın sağladığı değerin toplamıdır. Aynı mantığı kullanarak aşağıdaki ikili tabandaki sayının ondalık tabandaki değeri aşağıdaki gibi bulunur:

$(11000000111001)_2 = 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 0 \times 2^7$

$+ 0 \times 2^8 + 0 \times 2^9 + 0 \times 2^{10} + 0 \times 2^{11} + 1 \times 2^{12} + 1 \times 2^{13}$

$(11000000111001)_2 = 1 + 0 + 0 + 8 + 16 + 32 + 0 + 0 + 0 + 0 + 0 + 0 + 4096 + 8192$

$(11000000111001)_2 = 12345$

Diğer tüm tabanlardaki sayıların onluk tabandaki karşılığının bulunması için yukarıda gösterilen yol aynı biçimde kullanılır.

2.2.1.2. Onluk - İkili Taban Çevrimleri

Onluk tabandan ikili tabana geçerken, ikili tabandan onluk tabana geçme yolundan farklı bir yol izlenerek art arda bölme işlemi uygulanır ve bu bölmenin sonucunda ortaya çıkan kalana bakılır. Aşağıda verilen onluk tabandaki sayı üzerinde onluk tabandan ikilik tabana geçme yöntemi gösterilecektir.

$(123)_{10}$

Onluk tabandaki bu sayının ikili tabana çevrilmesi için aşağıdaki yöntem izlenmelidir.

1.	Sayı çevrilmek istenen tabana bölünür. Kalan belirlenir. 123 / 2 = 61 kalan = 1
2.	Bulunan kalan sayının çevrilmek istenen tabandaki değerinin ilk basamağını gösterir. Bulunan kalan 1 olduğundan sayının ikili tabandaki son basamağı 1 olmalıdır ($s_0 = 1$).
3.	Bölüm yeniden ikiye bölünür. Kalan belirlenir. 61 / 2 = 30

	kalan = 1
4.	s_1 = 1. Bölüm ikiye bölünür ve kalan belirlenir. 30 / 2 = 15 kalan = 0
5.	s_2 = 0. Bölüm ikiye bölünür ve kalan belirlenir. 15 / 2 = 7 kalan = 1
6.	s_3 = 1. Bölüm ikiye bölünür ve kalan belirlenir. 7 / 2 = 3 kalan = 1
7.	s_4 = 1. Bölüm ikiye bölünür ve kalan belirlenir. 3 / 2 = 1 kalan = 1
8.	s_5 = 1. Bölüm daha fazla ikiye bölünemediği için çevrilme işlem sona erer ve bölümün değeri ikili düzendeki sayının en anlamlı basamağı olan s_6'nın değerini oluşturur. s_6 = 1.

Bu çevirme işlemi sonucunda aşağıdaki sayı bulunur:

$$(s_6s_5s_4s_3s_2s_1s_0)_2 = (1111011)_2$$

Sonucun doğrulanması için $(1111011)_2$ sayısı yeniden onluk tabana çevrilirse:

$$(1111011)_2 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6$$

$$(1111011)_2 = 1 + 2 + 0 + 8 + 16 + 32 + 64$$

$$(1111011)_2 = 123$$

Onluk tabandaki sayılar diğer sayı tabanlarına çevrilirken de aynı yöntem kullanılır. Sayı bu durumda ikiye değil dönüştürülmek istenen sayı tabanına, bölüm bu sayı tabanından büyük olduğu sürece, bölünür ve kalanlar ile bölüm yeni sayı tabanındaki sayıyı oluşturur.

2.2.2. Onaltılık Taban

Onluk tabanda sayıların gösterilmesi için 0 dan 9 a kadar olan 10 adet rakam kullanılır. Onaltılık tabanda ise sayıların gösterilmesi için 16 ayrı rakam gerektiğinden 9 dan sonraki rakamları göstermek için harflerden yararlanılır. Aşağıdaki tabloda onaltılık tabandaki sayılar ve bu sayıların onluk tabandaki değerleri karşılıklı olarak gösterilmiştir.

Onaltılık Rakam	Onluk Değer
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

2.2.2.1. Onaltılık – Onluk Taban Çevrimleri

Onaltılık tabandaki sayılar onluk tabana çevrilirken ikili taban ile onluk taban arasındaki çevrim yöntemi aynı biçimde uygulanır. Örneğin onaltılık tabandaki A12F sayısı onluk tabana aşağıdaki gibi çevrilir:

$$(A12F)_{16} = F \times 16^0 + 2 \times 16^1 + 1 \times 16^2 + A \times 16^3$$

Onaltılık tabanda kullanılan harfler onluk tabandaki değerleri ile değiştirilirse:

$$(A12F)_{16} = 15 \times 16^0 + 2 \times 16^1 + 1 \times 16^2 + 10 \times 16^3$$

$$(A12F)_{16} = 15 \times 1 + 2 \times 16 + 1 \times 256 + 10 \times 4096$$

$$(A12F)_{16} = 15 + 32 + 256 + 40960$$

$$(A12F)_{16} = 41263$$

Görüldüğü gibi sayılar ikili tabandan onluk tabana çevrilirken kullanılan çevrim mantığı burada da geçerlidir.

2.2.2.2. Onluk – Onaltılık Taban Çevrimleri

Onluk tabandaki sayılar onaltılık tabana çevrilirken onluk düzendeki sayı bölüm 16'dan küçük oluncaya dek onaltıya bölünür. Bölme işlemi tamamlandıktan sonra bölüm onaltılık düzendeki sayının en anlamlı basamağını, kalanlar ise diğer basamaklarını gösterirler. Onluk tabandaki değeri 41263 olan sayı onaltılık tabana aşağıdaki biçimde çevrilir:

1.	41263 / 16 = 2578 kalan = 15 $s_0 = 15 = F$
2.	2578 / 16 = 161 kalan = 2 $s_1 = 2$
3.	161 / 16 = 10 kalan = 1 $s_2 = 1$
4.	İşlemin sonucunda ortaya çıkan 10 sayısı 16'ya daha fazla tam olarak bölünemeyeceğinden $s_3 = 10 =$ A olarak belirlenir.

Basamakları oluşturan rakamlar bir araya getirildiğinde $(41263)_{10} = (A12F)_{16}$ sonucu bulunur.

2.2.2.3. İkili – Onaltılık Taban Çevrimleri

İkili tabandaki sayıların boyutu çık hızlı büyüdüğünden gösterilen sayılardaki basamak sayısı hızlı bir biçimde artar. Bu durum ikili tabandaki sayıların okunurluğunu azalttığından bilgisayar işlemlerinde yüksek basamaklı sayılar onaltılık tabanda gösterilir.

İkili tabandaki sayıların onaltılık tabana dönüştürülmesi için sayı önce onluk tabana daha sonra da onluk tabandan onaltılık tabana dönüştürülebilir. Bu işlemi daha kısa zamanda yapmayı sağlayan ve arada sayıyı onluk düzene çevirmeyi engelleyen kolay bir yöntem vardır. İkili düzendeki sayının basamakları sol yandan başlayarak dörderli öbeklere ayılır. Daha sonra bu öbeklerin onaltılık tabandaki karşılıkları bulunur ve öbekler bulunan rakamlarla değiştirilir. Böylece ortaya çıkan sayı ikili tabandaki asıl sayının onaltılık tabana dönüştürülmüş biçimidir.

Örneğin $(1111101010101010110)_2$ sayısının 16lık tabandaki karşılığının bulunması için sayının basamakları önce aşağıdaki gibi dörderli öbeklere ayrılır:

$$(1)_2\ (1111)_2\ (0101)_2\ (0101)_2\ (0110)_2$$

İkili düzendeki sayı dörderli öbeklere ayrıldıktan sonra her öbeğin onluk düzendeki değeri aşağıdaki gibi yazılır:

$$(0001)_2 = 1$$

$$(1111)_2 = 15$$

$$(0101)_2 = 5$$

$(0101)_2 = 5$

$(0110)_2 = 6$

Ondalık tabandaki bu değerler onaltılık tabana çevrilirse:

$1 = (1)_{16}$

$15 = (F)_{16}$

$5 = (5)_{16}$

$5 = (5)_{16}$

$6 = (6)_{16}$

Öbeklerin onaltılık tabandaki karşılıkları yerine yazılırsa:

$(1F556)_{16}$

sayısı elde edilir.

Çevirici diliyle program yazılırken çokça kullanıldığından bu iki taban arasında geçiş yapma kurallarının bilinmesi oldukça önemlidir.

2.2.2.4. Onaltılık – İkili Taban Çevrimleri

Onaltılık tabandaki bir sayının ikili tabana çevrilmesi için yıkarıda açıklanan yöntemin tersi kullanılabilir. Bu durumda onaltılık sayının her bir basamağının ikili tabandaki dört bitlik karşılığı yazılır. Örneğin bir önceki bölümde sonuç olarak bulunan onaltılık tabandaki 1F556 sayısının yeniden ikili tabana çevrilmesi için her basamak ayrı ayrı ikili tabana çevrilir. Eğer rakamın ikili tabana çevrildiği durumda dört basamaktan daha az basamak bulunuyorsa, sayı dört basamaklı olana kadar sayının soluna sıfır eklenir.

$(6)_{16} = (0110)_2$

$(5)_{16} = (0101)_2$

$(5)_{16} = (0101)_2$

$(F)_{16} = (1111)_2$

$(1)_{16} = (0001)_2$

Rakamlar ikili tabana çevrildikten sonra dört basamaklı öbekler bir araya getirilerek ikili tabandaki sayı oluşturulur:

$(00011111010101010110)_2$

Çıkan sayının sol tarafındaki 0lar anlamsız olduğu için sayıdan atılır:

$(11111010101010110)_2$

Bulunan ikili tabandaki bu değer $(1F556)_{16}$ sayısının karşılığıdır.

2.2.3. İkilik Tabanda Tümleyenler

Tümleyenler bilgisayarlarda çıkarma işlemini ve mantık işlemlerini yalın bir biçimde gerçekleştirmek amacıyla kullanılır. Bilgisayarlarda eksi sayılar, tümleyen yöntemleri kullanılarak yaratılır ve saklanır. Bu nedenle sayıların tümleyenlerinin nasıl alındığını ve bu düzende gösterilen sayılarla nasıl işlem yapıldığını bilmek önemlidir.

2.2.3.1. 1'e Tümleyen Yöntemi

İkili tabanda gösterilen bir sayının 1'e tümleyenini bulmak için her bitin (yani ikili sayının her bir basamağının) değeri terslenir. Eğer tümleyeni alınan bitin değeri sıfır ise sonuç bir, bir ise sonuç sıfır olur. Aşağıda ikili tabandaki bazı sayılar ve bu sayıların 1'e tümleyenleri örnek olarak verilmiştir:

Sayı	1'e tümleyeni
1	0
0	1
101	010
1111	0000

11011010101001111	00100101010110000

Devre düzeyinde bir sayının 1'e tümleyeni alınırken her bir basamağın değerinin tersi *DEĞİL* (*Evirici*) kapısı kullanılarak bulunur.

2.2.3.2. Tümleyen Yöntemi

İkili tabanda gösterilen bir sayının 2'ye tümleyenini bulmak için sayının önce 1'e tümleyeni alınır ve bulunan sonuç bir ile toplanır. Toplama sonucunda bulunan sayının basamak sayısı tümleyeni alınan sayının basamak sayısından fazla olursa, sonuçta bulunan sayının en soldaki basamağı atılır. Aşağıda ikili tabandaki bazı sayılar ve bu sayıların 2'ye tümleyenleri örnek olarak verilmiştir:

Sayı	2'ye tümleyeni
1	1
0	0
101	011
1111	0001
1101101	0010011

İkiye tümleyen almanın bir başka yolu da aşağıda adımları verilen yöntemi kullanmaktır:

1.	Sayının sağından başlayıp soluna doğru ilerleyerek değeri 1 olan ilk basamağa kadar bulunan tüm 0'ları kopyala. (Eğer sayının en sağdaki biti bir ise bu adım atlanır)
2.	0'ların ardından gelen ilk 1'i kopyala.
3.	Geri kalan basamakların tamamının tersini al.

2.2.4. İkilik Tabanda Aritmetik İşlemler

2.2.4.1. İkili Tabanda Toplama İşlemi

İkili tabandaki toplama işlemi günlük hayatta onluk tabandaki toplama işleminin yapıldığı gibi iki sayının her bir basamağının ayrı ayrı toplanması yoluyla yapılır. Tabana göre elde var ise (yani iki basamağın toplamı birden büyük çıktıysa) elde bir soldaki basamağa eklenir. Aşağıda ikili tabandaki toplama işlemine iki örnek verilmiştir:

İkili tabandaki 0110101 ile 011 sayıları aşağıdaki gibi toplanır:

```
  0 1 1 0 1 0 1
+         0 1 1
-------------
  0 1 1 1 0 0 0
```

İkili tabandaki 1101101 ile 1101101 sayıları aşağıdaki gibi toplanır:

```
    1 1 0 1 1 0 1
+   1 1 0 1 1 0 1
-------------
  1 1 0 1 1 0 1 0
```

2.2.4.2. İkili Tabanda Çıkarma İşlemi

Bilgisayarlarda çıkarma işlemi yapılırken tümleyenler kullanılır. Bir sayı diğerinden çıkarılırken çıkarılan sayının tümleyeni (bir diğer deyişle eksilisi) alınır ve bulunan bu sayı diğer sayı ile toplanır. Böylece çıkarma işlemi gerçekleşmiş olur.

Örnek: 2'ye tümleyen yöntemi kullanılarak ikili tabandaki 110001101 sayısından 011110101 sayısının çıkarılması işlemi aşağıdaki biçimde gösterilebilir:

110001101 - 011110101

Bu çıkarma işlemi ikinci sayının ikiye tümleyeni alınıp ilk sayı ile toplanması yoluyla toplama işlemine çevrilir. Böylece yukarıdaki çıkarma işlemi aşağıdaki toplama işlemine dönüşür:

```
    1 1 0 0 0 1 1 0 1
+   1 0 0 0 0 1 0 1 1
-----------------
  1 0 1 0 0 1 1 0 0 0
```

Toplama işleminin sonucu görüldüğü gibi ikili tabanda 1010011000 bulunmuştur. Çıkarma işleminin sonucunu ise bulunan bu sonuçla aynı değildir. Çıkarma işlemi yapılırken sonuç, işlenenlerin basamak sayısı en çok olanından daha geniş olamaz. Bu nedenle yukarıdaki sonuçta ortaya çıkan en soldaki bit fazladır ve bu fazlalık silinerek çıkarma işleminin doğru sonucu bulunur. Çıkarma işlemi için yapılan toplamanın sonucunda basamak sayısının bir artması bulunan sonucun sıfırdan büyük olduğunu gösterir. Fazladan bulunan basamak silindiğinde çıkarma işleminin sonucu aşağıdaki gibi bulunur:

010011000

Bu sonuç çıkarma işleminin sonucudur ve en soldaki biti sıfır olduğu için bulunan değer sıfırdan büyük bir değerdir.

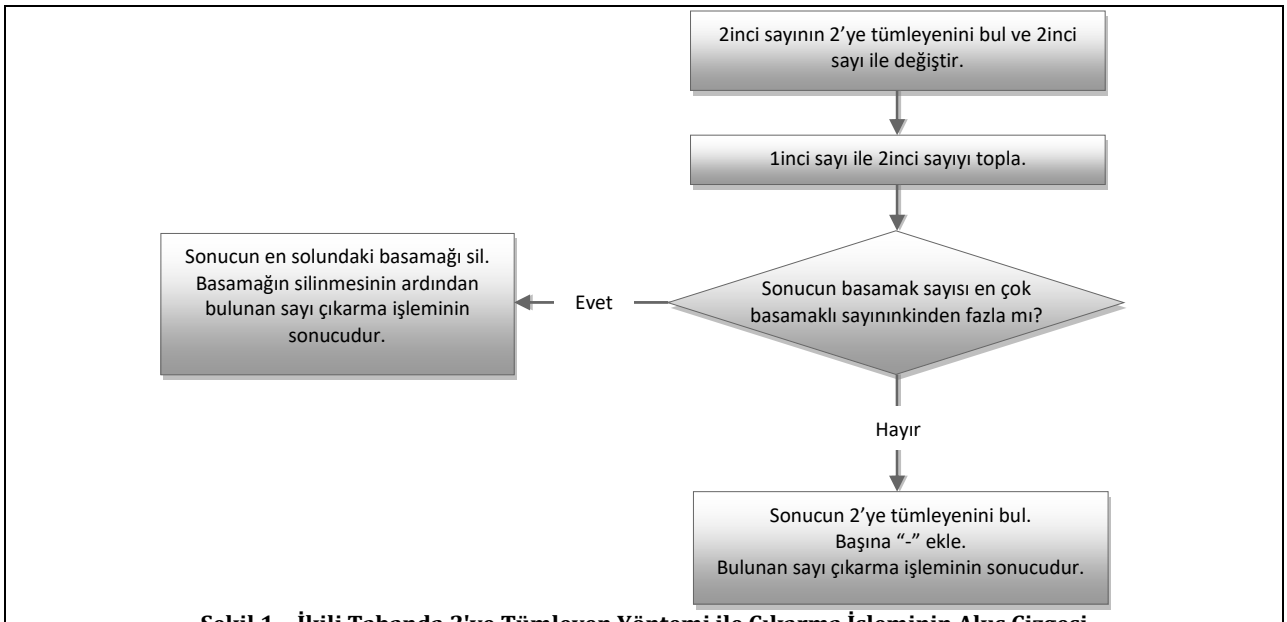
Örnek: 011110101 sayısından 110001101 sayısının ikiye tümleyen yöntemi kullanılarak çıkarılması işlemi aşağıda gösterilmiştir.

Çıkarılacak olan 110001101 sayısının ikiye tümleyeni 001110011 sayısıdır. Elde edilen bu sayı ile 011110101 sayısı aşağıdaki biçimde toplanır:

$$\begin{array}{rrrrrrrrrr} & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ + & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

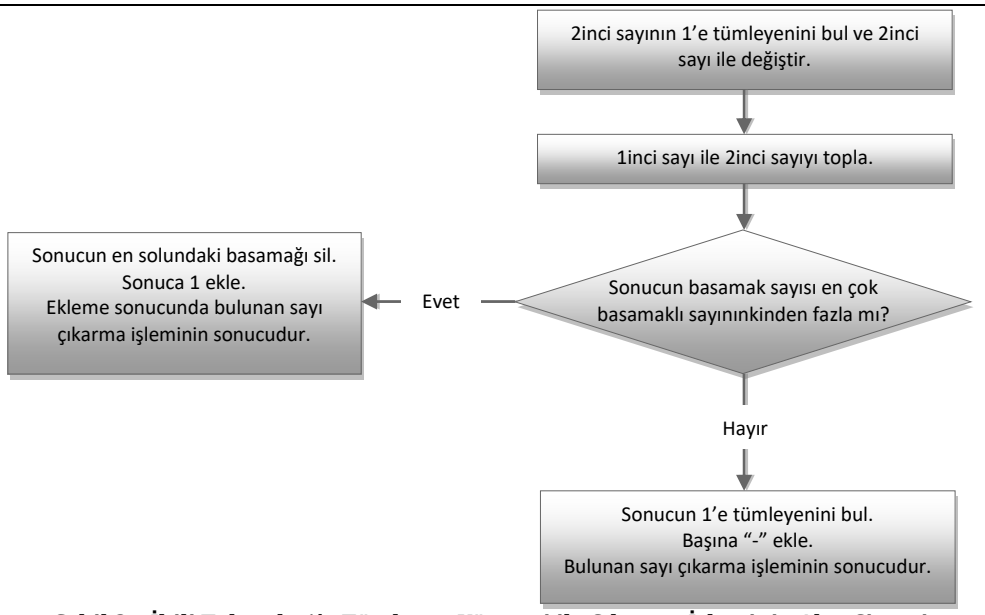
Görüldüğü gibi toplama işleminin sonucu 101101000 sayısıdır. Bir önceki örneğin aksine basamak sayısında bir artış olmadığından bu konuda bir işlem yapılmasına gerek yoktur. İşlemde küçük bir sayıdan büyük bir sayı çıkarıldığı için sonucun sıfırdan küçük olması beklenir. Çıkarma işlemi için yapılan toplamanın sonucunda basamak sayısında artış olmaması bulunan sayının eksi olduğunu gösterir. Bu yüzden çıkarma işleminin sonucunu bulabilmek için bulunan 101101000 sayısının ikiye tümleyeninin alınması gerekir. 101101000 sayısının ikiye tümleyeni 010011000 sayısı olduğundan çıkarma işleminin sonucu – 010011000 sayısıdır.

İkili tabanda 2'ye tümleyen yöntemi kullanılarak yapılan çıkarma işlemi aşağıdaki akış çizgesi ile özetlenebilir:



Şekil 1 – İkili Tabanda 2'ye Tümleyen Yöntemi ile Çıkarma İşleminin Akış Çizgesi

Örneklerde verilen çıkarma işlemlerinin aynıları çıkarılan sayının 1'e tümleyeni kullanılarak da yapılabilir. Bu yapılırken 2'ye tümleyen yöntemine benzer biçimde çıkarılan sayının 1'e tümleyeni alınır ve bu tümleyen birinci sayı ile toplanır. İkili tabanda 1'in tümleyeni kullanarak yaptılan çıkarma işlemini aşağıdaki akış çizgesi özetlemektedir:



Şekil 2 – İkili Tabanda 1'e Tümleyen Yöntemi ile Çıkarma İşleminin Akış Çizgesi

Örnek: İkili tabandaki 110001101 sayısından 011110101 sayısının 1'e tümleyen yöntemi kullanılarak çıkarılması aşağıdaki biçimde yapılır:

Bu çıkarma işleminin yapılmasının ilk adımı ikinci sayının 1'e tümleyeninin bulunmasıdır. İkinci sayının tüm basamaklarının tersi alındığında 1'e tümleyeni 100001010 sayısı olarak bulunur. Çıkarma işleminin sonucunun bulunması için bulunan bu tümleyen birinci sayı ile toplanır:

$$\begin{array}{rrrrrrrrrr} & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ + & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Toplama işlemi sonunda bulunan sonuç 1010010111 sayısıdır. Ortaya çıkan sonucun basamak sayısı işlenenlerin basamak sayısından büyük olduğu için, sonucun en soldaki "artan" basamağı silinir ve kalan sayı bir ile toplanır. Yapılan silme ve toplama işlemi sonucunda ortaya çıkan sonuç 10011000 sayısıdır.

Bire tümleyen yöntemiyle yapılan çıkarma işlemlerinde bulunan sonuçların yanlış olmaması için artan basamak silinip kalan sayıya eklenerek çıkan sonuç düzeltilir.

Örnek: 2'ye tümleyen yöntemi kullanılarak çözülen ikinci çıkarma örneğinde yapılan 011110101 – 110001101 işleminin ikinci sayının bire tümleyeni alınarak yapılışı aşağıda gösterilmiştir.

110001101 sayısının 1'e tümleyeni 001110010 sayısıdır. Bu sayı ile birinci sayı aşağıdaki gibi toplanır:

$$\begin{array}{rrrrrrrrrr} & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ + & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Görüldüğü gibi çıkan 101100111 sonucunda herhangi bir basamak artışı olmamıştır. Bu nedenle çıkarma işleminin doğru sonucunun bulunması için, bulunan toplama sonucunun 1'e tümleyeni alınır ve başına sayının sıfırdan küçük olduğunu belirten "-" işareti konur. Böylece çıkarma işleminin sonucu -010011000 sayısı olarak bulunur.

2.2.5. Onaltılık Tabanda Aritmetik İşlemler

2.2.5.1. Onaltılık Tabanda Toplama İşlemi

Onaltılık tabandaki toplama işlemleri ile ikili tabandaki toplama işlemlerinin yapılma yöntemleri aynıdır. Diğer tabanlarda da olduğu gibi basamaklar ayrı ayrı toplanır. Basamakların toplamı 15'ten büyük olduğunda, oluşan artan kısım bir sonraki basamağa eklenir.

Örnek: Onaltılık tabandaki AA1234F0 sayısı ile 1234ABCD sayısı aşağıdaki biçimde toplanır:

$$\begin{array}{rrrrrrrrr} & A & A & 1 & 2 & 3 & 4 & F & 0 \\ + & 1 & 2 & 3 & 4 & A & B & C & D \\ \hline B & C & 4 & 6 & E & 0 & B & D \end{array}$$

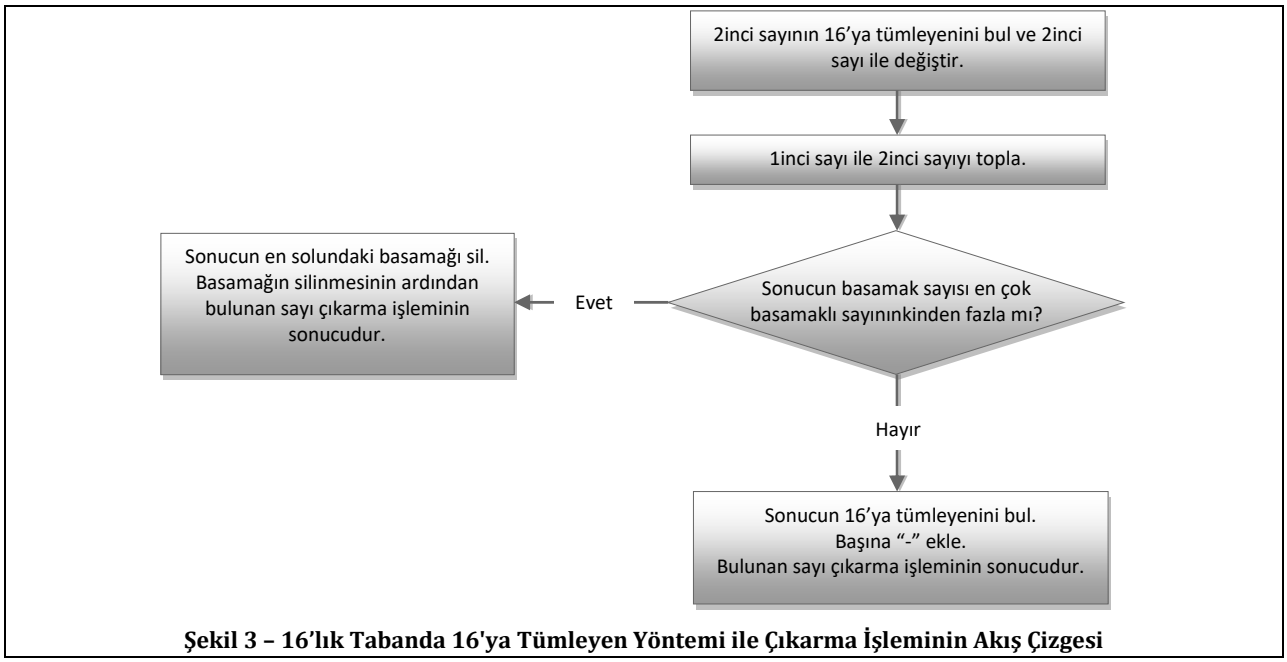
2.2.5.2. Onaltılık Tabanda Çıkarma İşlemi

Çıkarma işlemi de ikili tabandaki çıkarma işlemleri ile aynı mantık kullanılarak yapılır. İkili tabandan farklı olarak yalnızca 2'ye tümleyen yerine 16'ya tümleyen ve 1'e tümleyen yerine 15'e tümleyen kullanılır. 16'nın tümleyeni yöntemi kullanılarak yapılan çıkarma işleminin akış çizgesi Şekil 3'te gösterilmiştir.

Örnek: Onaltılık tabandaki 1234ABCD sayısı AA1234F0 sayısından 16'ya tümleyen yöntemi kullanılarak aşağıdaki biçimde çıkarılır:

```
      A A 1 2 3 4 F 0
+     E D C B 5 4 3 3
  1 9 7 D D 8 9 2 3
```

Bu işlem yapılırken, akış çizgesine göre öncelikle ikinci sayının 16'ya tümleyeni alınır. 1234ABCD sayısının 16'ya göre tümleyeni EDCB5433 sayısıdır. Bu sayı ile birinci sayıyı toplandığında 197DD8923 sayısı bulunur. Basamak sayısı arttığından en soldaki basamak silinir ve çıkarma işleminin sonucu $(97DD8923)_{16}$ olarak bulunur.



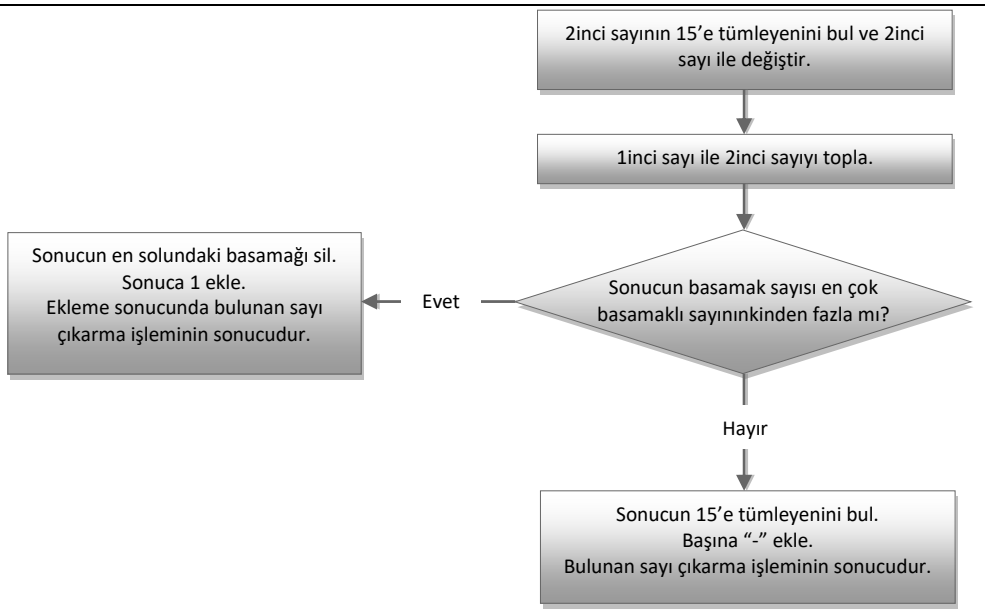
Şekil 3 – 16'lık Tabanda 16'ya Tümleyen Yöntemi ile Çıkarma İşleminin Akış Çizgesi

Örnek: Onaltılık tabandaki AA1234F0 sayısı 1234ABCD sayısından 16'ya tümleyen yöntemi kullanılarak aşağıdaki biçimde çıkarılır:

```
      1 2 3 4 A B C D
+     5 5 E D C B 1 0
      6 8 2 2 7 6 D D
```

AA1234F0 sayısının 16'ya tümleyeni 55EDCB10 sayısıdır. Bulunan bu sayı ile 1234ABCD sayısı toplanarak çıkarma işlemi gerçekleştirilir. Bulunan sonuçta basamak artışı olmadığından, 682276DD sayısının 16'ya göre tümleyeni alınıp, başına sayının sıfırdan küçük olduğunu belirten "-" işareti konularak çıkarma işleminin sonucuna ulaşılır. Çıkarma işleminin sonucu - 97DD8923 sayısıdır.

16'lık tabanda çıkarma işlemini gerçekleştirmenin ikinci yolu olan 15'e tümleyen kullanma yönteminin akış çizgesi Şekil 4'te gösterilmiştir.



Şekil 4 – 16'lık Tabanda 15'e Tümleyen Yöntemi ile Çıkarma İşleminin Akış Çizgesi

Örnek: Onaltılık tabandaki 1234ABCD sayısı AA1234F0 sayısından 15'e tümleyen yöntemi kullanılarak aşağıdaki biçimde çıkarılır:

```
    A A 1 2 3 4 F 0
+   E D C B 5 4 3 2
  1 9 7 D D 8 9 2 2
```

Bu işlem yapılırken, Şekil 4'teki akış çizgesine göre öncelikle ikinci sayının 15'e tümleyeni alınır. 1234ABCD sayısının 15'e tümleyeni EDCB5432 sayısıdır. Bulunan bu sayı ile birinci sayıyı toplanarak çıkarma işlemi gerçekleştirilir. Toplama işleminin sonucunda basamak sayısı arttığından en soldaki basamak silinir ve toplamanın sonucuna bir eklenerek çıkarma işleminin sonucu $(97DD8923)_{16}$ olarak bulunur.

Örnek: Onaltılık tabandaki AA1234F0 sayısı 1234ABCD sayısından 15'e tümleyen yöntemi kullanılarak aşağıdaki biçimde çıkarılır:

```
    1 2 3 4 A B C D
+   5 5 E D C B 0 F
    6 8 2 2 7 6 D C
```

AA1234F0 sayısının 15'e tümleyeni 55EDCB0F sayısıdır. Bu sayı ile 1234ABCD sayısı toplanarak çıkarma işleminin sonucu bulunur. Toplama işleminin sonucunda basamak artışı olmadığından bulunan 682276DC sayısının 15'e tümleyeni alınıp başına sayının sıfırdan küçük olduğunu belirten "-" işareti konularak çıkarma işleminin sonucuna ulaşılır. 15'e tümleyen yöntemiyle yapılan bu çıkarma işleminin sonucu -97DD8923 sayısıdır.

2.3. İşaretli Sayılar

Bu bölüme kadar anlatılan aritmetik işlemler ve sayı gösterimleri sıfırdan büyük sayılar için geçerlidir. Örneklerde bulunan tüm sayılar ya sıfırdan büyüktür ya da sıfırdan küçük oldukları başlarına konulan "-" işareti ile gösterilmiştir. Ancak bilgisayar donanımları yalnızca 0 ve 1 değerleriyle çalışabildiği için sayının eksi olduğunu gösteren bu "-" işaretlerinin donanımın anlayacağı biçime sokulmaları gerekir.

İşaretli sayılar belirtilen bu soruna çözüm olarak ortaya atılmıştır. Bilgisayarlarda gösterilen ikilik düzendeki işaretli sayılarda en soldaki bit sayının işaretini tutar. Sayının en başındaki bu "işaret bitinin" içeriğine göre sayının sıfırdan büyük ya da küçük olduğu anlaşılır.

Genel kural olarak işaret biti 1 olan sayı sıfırdan küçük, işaret biti 0 olan sayı sıfırdan büyük kabul edilir. Kullanılan işaretleme yöntemine göre en soldaki işaret biti sayının değerine dahil olabilir ya da olmayabilir. 1'e tümleyen ve 2'ye tümleyen gösterimlerinde işaret biti sayının değerine dahildir.

Aşağıdaki tabloda dört bit kullanılarak değişik yöntemlerle gösterilebilecek tüm işaretli sayılar sıralanmıştır.

İkilik	İşaret biti sayıdan	1'e tümleyen	2'ye tümleyen

Tabanda Sayı	ayrıysa değeri	düzeninde değeri	düzeninde değeri
0000	+0	+0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

İşaret bitinin ayrı olduğu durumda ilk bit her zaman işaret için ayrılmıştır ve sayıya dahil değildir. İşaret biti sayıdan ayrıldıktan sonra elde kalan 3 bit sayının değerini belirler. 1'e tümleyen düzeninde işaret biti 1 olduğunda sayının sıfırdan küçük olduğu anlaşılır ve değerinin anlaşılması için sayının işaret biti dahil 1'e tümleyeni alınarak değeri bulunur ve başına "-" işareti konulur. 2'ye tümleyen düzeninde de benzer biçimde, işaret biti 1 olduğunda sayının 2'ye tümleyeni alınarak değeri bulunup başına eksi işareti konulur.

İşaret bitinin değerden ayrı olarak kullanıldığı durumda ve 1'e tümleyen yönteminde +0 ve -0 olmak üzere iki ayrı sıfır değeri olduğundan bu gösterim yöntemlerinde gösterilebilecek sayı sayısı 2'ye tümleyen yöntemiyle gösterilebilecek sayı sayısından bir eksiktir. Bu nedenle aynı sayıda bitle işaretli sayıları göstermek için 2'ye tümleyen yöntemi daha verimlidir.

2.4. Kayan Nokta Veri Türü

Bu kitabın kapsamında anlatılan konular içinde verilen örneklerde genellikle 16 bitlik (2 bayt) sayılardan büyük sayılar ile işlemler yapılmayacaktır. Eğer bu 16 bitin en solundaki bit işaret biti olarak kullanılırsa -2^{15} (-32768) ile $2^{15}-1$ (32767) arasındaki sayılar gösterilebilir. Bir başka deyişle 15 bit duyarlılığında sayı gösterme aralığı 2^{15}'tir.

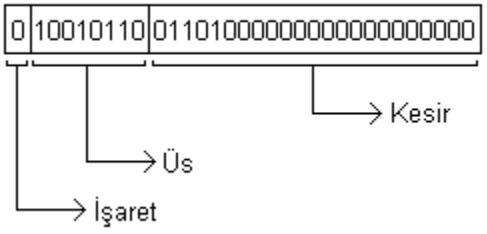
Bilgisayarlarda bazı durumlarda ondalıklı sayılarla ya da çok büyük sayılarla işlem yapmak gerekebilir. Örneğin $1{,}01101 \times 2^{23}$ gibi bir sayı ile yapılması gerekebilir. Böyle durumlarda verilerin gösterilmesi için kayan nokta veri türü kullanılır. Değişik çeşitleri olan kayan nokta veri türünün günümüz bilgisayarlarında genellikle 32 bitlik bir çeşidi kullanılmaktadır. Bu 32 bitin ilk biti işaret, ilk bitten sonraki sekiz bit üs ve bu dokuz bitten sonra gelen yirmi üç bit ise anlamlı kısım için kullanılmaktadır. Anlamlı kısım, gösterilmek istenen sayı $1{,}aaa \times 2^{ü}$ biçimine getirildiğinde virgülden sonra gelen aaa kısmı, üs ise ikinin kuvvetini belirten ü'dür. Bu gösterim biçimine IEEE 754 biçimi denir. Aşağıda 32bitlik IEEE 754 kayan nokta saklama biçimi özetlenmiştir:

İşaret (1 bit) = A	Üs (8 bit) = B	Anlamlı Kısım (23 bit) = C

IEEE 754 biçiminde gösterilen 32 bitlik bir verinin değeri aşağıdaki gibi hesaplanabilir:

$$\text{Sayının Değeri} = (-1)^{A} \times 1{,}C \times 2^{B-127}$$

Örnek: $1{,}01101 \times 2^{23}$ sayısı IEEE 754 kayan nokta veri türü biçiminde 32 bitle aşağıdaki gibi gösterilir:



Örnek: IEEE 754 kayan nokta veri türü biçiminde gösterilen 10101010101111110111000000000000 sayısının değeri aşağıdaki gibi bulunur:

Sayının işaretini belirten otuz birinci bitin (en soldaki bitin), değeri 1'dir. Bu bitin değerinin 1 olması sayının eksi (sıfırdan küçük) olduğunu gösterir. Ardından gelen 8 bitin değeri 01010101 olduğundan B = 85 alınır ve buradan B - 127 nin değeri -42 olarak bulunur.

Üs bitlerinden sonra gelen bitler sayının virgülden sonra gelen anlamlı kısmını gösterdiği için 01111110111000000000000 sayısı virgülden sonra gelen kısımdır.

Böylece sayının değeri

$$(-1)^1 \times 1,01111110111000000000000 \times 2^{-42} = -1,01111110111 \times 2^{-42}$$

olarak bulunur.

3. 8086 MİMARİSİ

3.1. 8086 Öncesi

Günümüzdeki mikroişlemci tanımına uyan ciddi anlamda ilk mikroişlemci projeleri 1968 yılında *Intel* firmasının yürüttüğü 4004, *Texas Instruments* firmasının yürüttüğü TMS 1000 ve *Garrett AiResearch* firmasının yürüttüğü Central Air Data Computer projeleridir. Bu projeler hemen hemen aynı zamanlarda tamamlandığından tam olarak çalışır halde bir mikroişlemcinin ilk olarak hangi firma tarafından üretildiği bilinmemektedir.

1971'de bitirilen ve Intel firmasının ilk mikroişlemcisi olan 4004 mikroişlemcisi aynı anda yalnızca 4 bitlik veriler ile çalışabilen 4 bitlik bir işlemci idi. Saat sıklığı 740 kHz olan bu işlemcinin 16 iğnesi bulunmakta ve sistem ile bağlantısı bu iğneler aracılığıyla yapılmaktaydı. 4004 işlemcisinde iki saat vuruşu arasında geçen süre (çevrim süresi) 1/740000 = 1,35 mikro saniye tutmaktaydı. Günümüzde kullanılan ve saat sıklığı 3GHz olan işlemciler, yalnızca saat sıklığı düşünüldüğünde, 4004 mikroişlemcisinden yaklaşık 4000 kat daha hızlıdır. Çağdaş işlemcilerde, başarımın artırılması için pek çok değişik yöntem kullanıldığından, aradaki hız farkı gerçekte çok daha fazladır. 4004 işlemcisinin 4 bitlik veri yoluna karşın, günümüzdeki işlemciler 32 veya 64 bitlik veri yoluna sahip olduğundan aynı anda daha çok veri alabilmekte ve işleyebilmektedir. Bugünkü işlemcilerde daha sonra da değineceğimiz gibi kullanılan bazı yöntemler sayesinde aynı anda birden fazla işlem yapılabilmektedir. Örneğin bazı öngörü birimleri sayesinde işlemciler programın ne yönde akacağını tahmin edip ona göre hareket edebilmektedir.

4004 işlemcisinde komutlar 8 bitlik idi ve tam olarak 46 tane komut bulunmaktaydı. 4004 her biri 4 bitlik genişlikte veri tutabilen 16 tane yazmaç içeriyordu. Bellek adreslemesi için 12 bitlik adresler kullanılabildiği için 4 kilobaytlık belleklere erişmek mümkündü. Tasarımında 2300 transistor kullanılan 4004'ün altyordam yığını 3 düzeyli olduğundan iç içe 3 tane alt yordam girebiliyordu.

4004 işlemcisinin ardından Intel firması 8008 mikroişlemcisini piyasaya sürdü. Bu işlemci aynı anda 8 bitlik verilerle çalışabilmekteydi ve 0,8 MHz'lik bir saati vardır. 14 bitlik adresleme olanağı bulunduğundan 16 kilobayt bellek erişimi yapılabiliyordu. 8008 işlemcisinin bir saniyede işleyebildiği komut sayısı 4004 mikroişlemcisinden daha az olmasına rağmen, 8 bitlik 8008, aynı anda işleyebildiği veri sayısının yüksek olması dolayısıyla 4 bitlik 4004'e göre daha hızlıydı.

8008'den sonra Intel firması kullanılabilirliği yüksek olan 8080 mikroişlemcisini piyasaya sürdü. 8080 8 bitlik bir mikroişlemci idi ve 2 MHz'lik bir saati vardı. 8080 işlemcisinin 8 bitlik 7 adet yazmacı vardı ve bu 7 yazmaçtan 6 tanesi birleştirilip 16 bitlik yazmaç olarak da kullanılabilmekteydi. 16 bitlik bellek adreslemesi ve 8 bitlik veri yolu sayesinde 8080 mikroişlemcisiyle 64 kilobaytlık belleklere erişilebilmekteydi. Tasarımında 6000 transistör kullanılan 8080'de 8 bitlik birikeç (akümülatör), 16 bitlik yığıt imleci ve 16 bitlik program sayacı bulunmaktaydı. Bu terimlerin anlamları bölümün sonlarına doğru açıklanacaktır.

8080 den sonra üretilen 8085'in kullanımı, beraberinde kullanılması gereken devre elemanı sayısı daha az olduğu için 8080'den daha kolaydı. Örneğin 8085 besleme olarak yalnızca 5V'luk bir güç kaynağı gerektirirken, 8080 işlemcisine +5V, -5V ve +12V luk gerilimlerin sağlanması gerekmekteydi. Kullanım kolaylığından ötürü 8085 işlemcisi mikro denetleyici olarak uzun süre kullanıldı ve halen bazı uygulamalarda kullanımı sürmektedir.

Bu işlemcilerin ardından Intel günümüzde yaygın olarak kullanılan x86 mikroişlemci mimarisinin ilk örneğini üretti: 8086.


Intel® 8086 Processor

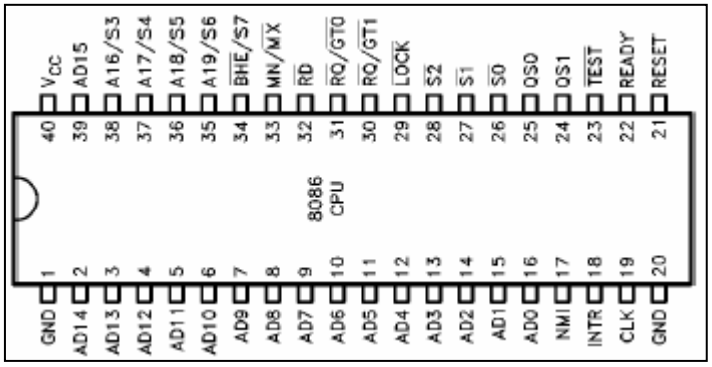
3.2. 8086

1978 yılında Intel firması 16 bitlik 8086 mikroişlemcisini piyasaya sürdü. 8086 işlemcisinin bazı temel teknik özellikleri aşağıdaki gibidir:

- 16 bit veri yolu; tek seferde 16 bit veri işleyebilme özelliği (16 bitlik mikroişlemci)
- 4,7 MHz – 10 MHz arası saat sıklığı
- 20 bitlik adresleme özelliği; 2^{20} = 1MBlık belleğe erişebilme
- 8 adet 16 bitlik yazmaç

x86 komut kümesini kullanan bütün mikroişlemciler (örneğin Intel Pentium ve AMD Athlon serisi) 8086 ile geriye dönük olarak uyumludur. 8086 işlemcisi için yazılan tüm komutlar bugünkü x86 tabanlı bilgisayarlarda çalışabilmektedir. Ancak günümüz bilgisayarları için yazılmış olan ve genişletilmiş x86 komut kümesi kullanan yazılımlar 8086 mikroişlemcisi üzerinde çalışamamaktadırlar.

8086 mikroişlemcisi aşağıdaki iğne diyagramından da görüldüğü gibi 40 iğnelik bir tümleşik devre şeklindedir. Şeklin altındaki tabloda iğnelerin işlevlerini açıklanmıştır.



İğne	Kısaltma	Tür	İşlev
1	GND	-	Topraklama iğnesi
2	AD14	G/Ç	On beşinci adres ve veri yolu iğnesi
3	AD13	G/Ç	On dördüncü adres ve veri yolu iğnesi
4	AD12	G/Ç	On üçüncü adres ve veri yolu iğnesi
5	AD11	G/Ç	On ikinci adres ve veri yolu iğnesi
6	AD10	G/Ç	On birinci adres ve veri yolu iğnesi
7	AD9	G/Ç	Onuncu adres ve veri yolu iğnesi
8	AD8	G/Ç	Dokuzuncu adres ve veri yolu iğnesi
9	AD7	G/Ç	Sekizinci adres ve veri yolu iğnesi
10	AD6	G/Ç	Yedinci adres ve veri yolu iğnesi
11	AD5	G/Ç	Altıncı adres ve veri yolu iğnesi
12	AD4	G/Ç	Beşinci adres ve veri yolu iğnesi
13	AD3	G/Ç	Dördüncü adres ve veri yolu iğnesi
14	AD2	G/Ç	Üçüncü adres ve veri yolu iğnesi
15	AD1	G/Ç	İkinci adres ve veri yolu iğnesi
16	AD0	G/Ç	Birinci adres ve veri yolu iğnesi
17	NMI	G	Maskelenemeyen kesme iğnesi
18	INTR	G	Kesme istemi iğnesi
19	CLK	G	Saat darbesi iğnesi
20	GND	-	Topraklama iğnesi
21	RESET	G	Yeniden başlatma iğnesi
22	READY	G	Hazır belirteci iğnesi
23	TEST’	G	Deneme iğnesi
24	QS1	Ç	İkinci sıra durum iğnesi
25	QS0	Ç	Birinci sıra durum iğnesi
26	S0’	Ç	Birinci durum iğnesi
27	S1’	Ç	İkinci durum iğnesi
28	S2’	Ç	Üçüncü durum iğnesi
29	LOCK’	Ç	Kilitleme iğnesi
30	RQ’/GT1’	G/Ç	İkinci istem ve atama iğnesi
31	RQ’/GT0’	G/Ç	Birinci istem ve atama iğnesi
32	RD’	Ç	Okuma iğnesi
33	MN/MX’	G	Minimum/maksimum kipi seçici iğnesi
34	BHE’/S7	Ç	Veri ve adres yolu etkinleştirme ve sekizinci durum iğnesi
35	A19/S6	Ç	Adres yolu ve yedinci durum iğnesi
36	A18/S5	Ç	Adres yolu ve altıncı durum iğnesi
37	A17/S4	Ç	Adres yolu ve beşinci durum iğnesi
38	A16/S3	Ç	Adres yolu ve dördüncü durum iğnesi
39	AD15	G/Ç	On altıncı adres ve veri yolu iğnesi

40	VCC	-	+5V Besleme iğnesi

2 – 16. ve 32. iğneler (ADDRESS/DATA BUS – ADRES/VERI YOLU):

2-16 ve 32. iğneler adres ve veri yolu iğneleridir. Bu iğneler zamanda çoğullanmış T1 adres yolu ile T2, T3, TW ve T4 veri yollarını oluşturmaktadır.

AD0, D7 – D0 veri yolunu seçtirdiğinden BHE' iğnesine benzer bir işlev görmektedir. AD0 T1 süresince mantıksal 0 iken D7 – D0 veri yolundan veri aktarımı yapılabilir. 8 bitlik aygıtlar D7 – D0 veri yoluna bağlı ise AD0 kullanılarak seçme işlemi yapılabilir.

Bu iğneler mantıksal 1 iken etkindirler ve ayrıca kesme alındığında ya da bekletme istemi alındığında ise üç durumlu "kapalı" durumuna geçerler.

T1, T2, T3, T4, TW ve TI

İşlemcinin her veri yolu çevrimi en az dört tane CLK çevriminden oluşur. Bu çevrimlere T1, T2, T3 ve T4 denir. T1 süresince işlemci adresi verir. T3 ve T4 süresince ise veri alış-verişi gerçekleştirilir. T2 süresi ise okuma işlemleri gerçekleştirilirken veri yolunun yönünün değiştirilmesinde kullanılır.

T1 süresince işlemci tarafından adreslenen birim "Hazır Değilim" sinyali yollarsa T3 ve T4 sürelerinin arasına bir CLK çevrimi uzunluğunda bir TW (Wait – Bekle) süresi eklenir.

Veri yolu çevrimleri arasında bazı boş çevrimler oluşabilir. Bu çevrimler TI (Idle – Boş) ile belirtilir. Bu "boş" çevrimler işlemci tarafından iç işlemlerin yapılmasında kullanılır.

35 – 38. iğneler (ADDRESS/STATUS – ADRES/DURUM):

T1 süresince en anlamlı dört adres iğnesi bu iğnelerdir. Girdi/Çıktı işlemleri yapılırken bu iğneler mantıksal 0 durumundadır. T2, T3, TW ve T4 süresince bellek ve girdi/çıktı işlemleri yapılırken bu iğnelerden durum bilgisi okunabilir. Kesme etkinleştirilme bayrak biti S5 her CLK çevrim başlangıcında güncellenmektedir.

A17/S4 ve A16/S3 iğnelerinin farklı durumları için yüklendikleri farklı anlamlar aşağıdaki tabloda verilmiştir. A17/S4 ve A16/S3 iğneleri hangi yer değiştirme yazmacının veri erişiminde kullanılacağını belirtir. S6 iğnesi aşağıdaki her durum için mantıksal 0 durumunda olması gerekmektedir.

A17/S4	A16/S3	Tip
0	0	Dönüşümlü Veri
0	1	Yığıt
1	0	Kod ya da boş
1	1	Veri

Bu iğneler mantıksal 1 iken etkindirler ve ayrıca kesme alındığında ya da bekletme istemi alındığında ise üç durumlu kapalı durumuna geçerler.

34. İğne (BHE/STATUS – BHE/DURUM):

T1 süresince D15 – D8 veri yolunun etkinleştirilmesi için **BHE' (B**us **H**igh **E**nable**'** – Üst Veri yolu Etkinleştirme') iğnesi kullanılmalıdır. 8 bitlik aygıtlar D15 – D8 veri yoluna bağlı ise BHE' kullanılarak seçme işlemi yapılabilir. BHE' iğnesi T1 süresince okuma, yazma ve kesme alma çevrimleri için mantıksal 0 durumunda olmalıdır. S7 durum bilgisi T2, T3 ve T4 süresince alınabilir.

BHE' iğnesi mantıksal 0 konumunda iken etkindir ve üç durumlu kapalı durumuna "bekleme" sinyali alındığında geçer. T1 süresince ilk "kesme alındı" çevrimi için mantıksal 0 durumundadır.

BHE'	A0	Tip
0	0	Tam sözcük
0	1	Üst bayt ↔ Tek Adres
1	0	Alt bayt ↔ Çift Adres
1	1	-

32. İğne (READ – OKU):

RD' iğnesine gelen tetikleme darbesi, işlemcinin bellek ya da girdi/çıktı okuma çevriminde olduğunu belirtir. Bellek ya da girdi/çıktı okuma çevrimi olması S2 iğnesi tarafından belirlenir.

RD' iğnesi T2, T3 ve TW süresince mantıksal 0 durumunda etkindir ve T2 süresince işlemcinin yerel hattı kayana kadar mantıksal 1 durumunu korumaktadır. Üç durumlu kapalı durumuna "bekleme" sinyali alındığında geçer.

22. İğne (READY – HAZIR):

Adreslenen bellek ya da girdi/çıktı işlemi yapan aygıttan işlemin yapılacağına dair alındılama geldiğinde mantıksal 1 durumuna geçer ve bu durumda etkindir. Alınan sinyalin anuyumlu olması 8284A Saat Darbesi Üreteci tarafından sağlanmaktadır.

Bu iğne anuyumlu değildir ve düzgün işlem yapabilmek için kurulum ve bekleme zamanlarının denetlenmesi gerekmektedir.

18. İğne (INTERRUPT REQUEST – KESME İSTEMİ):

Her komutun son saat çevriminde işlemcinin kesme alındı işlemi yapmasına veya yapmamasına karar verebilmek için kullandığı bir sinyaldir. Yazılım tarafından kesme etkinleştirme biti sıfırlanarak maskelenebilir. INTR tamamen anuyumludur. Mantıksal 1 konumunda iken etkindir.

23. İğne (TEST – DENEME):

"WAIT" komutu işlenirken bu girişe bakılır. Eğer TEST' mantıksal 0 ise işlemci yaptığı işi sürdürür; asi halde "bekleme alındı" kipine geçer. Bu giriş her saat çevriminin artan ucunda anuyumlu hale getirilir.

17. İğne (NMI):

Bu giriş sayesinde ikinci tip bir kesme yaratılabilir. **NMI** (**N**on **M**askable **I**nterrupt – Maskelenemeyen Kesme) yazılım tarafından maskelenemez. Mantıksal 1 ve 0 geçişi işlenen komutun ardından bir kesme oluşmasına neden olur. NMI girişi anuyumludur.

21. İğne (RESET – BAŞTAN BAŞLAT):

İşlemcinin yaptığı işlemi hemen sonlandırmasına neden olur. Bunun için iğnenin en az dört saat çevrimi için mantıksal 1 konumda aktif olması gerekmektedir. İğne mantıksal 0 konumuna getirildiğinde ise işlemci baştan başlatılır. RESET girişi anuyumludur.

19. İğne (CLOCK – SAAT):

İşlemci ve veri yolu denetleyicisi için gerekli saat darbesini sağlar. Asimetriktir ve %33 doluluk boşluk oranına sahiptir. Bu sayede en iyi iç zamanlama oluşturulabilmektir.

33. İğne (MN/MX'):

Mi**n**imum/**M**a**x**imum' (**MN/MX'**) iğnesi ile işlemcinin çalıştığı kip seçilebilir. Minimum kipi için mantıksal 1, maksimum kipi için ise mantıksal 0 uygulanmalıdır.

Aşağıda önce maksimum kip için geçerli iğne bilgileri verilecektir. Ardından minimum kip için geçerli iğneler işlenecektir.

26. – 28. İğneler (STATUS – DURUM):

T1, T2 ve T4 süresince aktiftir. T3 süresince pasif veya TW süresince READY mantıksal bir ise pasif. Pasif durumda S2',S1' ve S0' mantıksal bir durumundadır.

Bu durum bitleri 8288 Veri Yolu Denetleyicisi tarafından bellek ve girdi/çıktı erişimleri için gerekli sinyalleri üretmede kullanılır.

T4 süresince S2',S1' veya S0' iğnelerinde meydana gelen bir değişiklik bir veri yolu çevriminin başladığının göstergesidir. T3 veya TW süresince pasif duruma geçiş ise veri yolu çevriminin sonlandığını gösterir.

Bu iğneler "bekleme alındılandı" durumu oluştuğunda üç durumlu kapalı durumunu alır.

S2',S1' ve S0' nün aldığı değerler ve bu değerlerin anlamları aşağıdaki tabloda verilmiştir.

S2'	S1'	S0'	Tip
0	0	0	Kesme İstemi Alındı
0	0	1	Girdi/Çıktı Kapısı Oku
0	1	0	Girdi/Çıktı Kapısına Yaz
0	1	1	Dur
1	0	0	Kod Erişimi

1	0	1	Bellek Oku
1	1	0	Bellek Yaz
1	1	1	Edilgin

30. ve 31. İğneler (REQUEST/GRANT – İSTEM/ATAMA):

Bu iğneler yerel ana veri yolları tarafından işlemcinin devam eden veri yolu çevriminin bitiminde kullanılan veri yolunu salmasını sağlamak için kullanılır. RQ'/GT0' iğnesinin RQ'/GT1' iğnesine göre önceliği vardır.

RQ'/GT' iğneleri kaynak gerilimine bağlı dirençlere (pull-up) sahiptirler ve gerektiğinde boş bırakılabilirler.

İstem ve atama dizisi aşağıdaki biçimde gerçekleşir:

1. Yerel ana veri yolundan gelen bir CLK genişliğindeki sinyal 8086 mikroişlemcisine bir "yerel veri yolu istemi" olduğunu belirtir.

2. T1 veya T4 süresince 8086 mikroişlemcisi bu istemciye bir CLK genişliğinde bir sinyal yollar ise istem kabul edilmiştir ve bir sonraki CLK darbesinde "bekleme alındılandı" durumuna geçecektir. İşlemcinin veri yolu ara yüzü "bekleme" durumuna geçildiğinde mantıksal olarak veri yolundan ayrılır.

3. İstemci tarafından 8086 mikroişlemcisine gönderilen bir CLK genişliğindeki sinyal "bekleme" durumunun biteceğini ve 8086 mikroişlemcisinin bir sonraki CLK darbesinde yerel veri yolunu bağlayabileceğini belirtir.

Her veri yolu değişimi yukarıdaki biçimde üç tane darbe ile yapılmaktadır. Her değişim arasında ise bir CLK genişliğinde bir darbe gönderilmelidir. Mantıksal 0 durumunda darbeler etkindir.

Eğer istem mikroişlemci bir bellek çevriminde iken yapılırsa, mikroişlemci T4 süresince aşağıdaki durumların hepsi gerçeklendiğinde yerel veri yolunu salar:

1. İstem T2 süresince ya da T2'den önce oluşursa,

2. İstem anındaki çevrim bir kelimenin alt baytı değil ise ya da tek adres değil ise,

3. İstem anındaki çevrim bir kesme alındı dizisinin ilk alındılaması değil ise,

4. Kilitlenmiş bir komut işlemci tarafından çalıştırılmamaktaysa.

Eğer yerel veri yolu istem yapıldığında boşta ise iki farklı durum oluşabilir:

1. Bir sonraki saat darbesinde yerel veri yolu salınabilir.

2. Bir bellek çevrimi üç saat darbesi içinde başlayabilir. Yukarıdaki dört kural geçerlidir.

29. İğne (LOCK – KİLİTLE):

LOCK' etkin (mantıksal 0) iken yerel ana veri yolları arasında değişim yapılamaz. LOCK' sinyali LOCK komutu tarafından etkinleştirilir ve bir sonraki komut işlenene kadar etkin olarak kalır.

Sinyal etkin iken mantıksal 0 durumundadır ve "bekleme alındılandı" durumuna geçildiğinde üç durumlu kapalı durumunu alır.

24. ve 25. İğneler (QUEUE STATUS – SIRA DURUMU):

Sıra işlemi yapıldıktan sonraki CLK çevrimi süresince bu iğne geçerlidir. QS1 ve QS0 iğneleri dışarıdan 8086 mikroişlemcisinin komut sırasını takip etmek için kullanılmaktadır.

Aşağıdaki tabloda QS1 ve QS0 değerleri ve anlamları bulunmaktadır:

QS1	QS0	Tip
0	0	İşlem yok

0	1	İşlem kodunun ilk baytı
1	0	Sırayı boşalt
1	1	Sıradaki bir sonraki bayt

Minimum kip için geçerli iğneler aşağıda tanıtılmıştır.

28. İğne (STATUS LINE – DURUM HATTI):

Mantıksal olarak maksimum kipteki S2 iğnesine eşdeğerdir. Bellek erişimi mi yoksa girdi/çıktı erişimi mi yapıldığını anlamak için kullanılır. M/IO' veri yolu çevriminden sonra gelen T4 süresince geçerlidir ve çevrim sonuna kadar T4 süresince geçerlidir.

"bekleme alındılandı" durumuna geçildiğinde M/IO' üç durumlu kapalı durumunu alır.

29. İğne (WRITE' – YAZ):

Bu iğne sayesinde işlemcinin belleğe yazma işlemi yaptığını veya girdi/çıktı sisteminde bir yazma işlemi yaptığı anlaşılır. Bir yazma işlemi yapılırken WR' iğnesi T2, T3 ve TW süresince etkindir.

WR' iğnesi mantıksal 0 durumunda iken etkindir ve "bekleme alındılandı" durumunda üç durumlu kapalı durumunu alır.

24. İğne (INTERRUPT ACKNOWLEDGE – KESME ALINDILANDI):

INTA' iğnesi kesme alındı çevrimlerinin belirlenmesinde kullanılır. T2, T3 ve TW süresince her kesme alındılandı çevriminde etkindir.

Mantıksal 0 durumunda iken etkindir.

25. İğne (ADDRESS LATCH ENABLE – ADRES MANDALI ETKİNLEŞTİRME):

8282/8283 adres mandalına işlemci tarafından adreslerin mandallanabilmesi için kullanılır. Mantıksal 1 durumunda iken etkindir.

T1 süresince veri yolu çevrimlerinde etkindir.

27. İğne (DATA TRANSMIT/RECEIVE – VERİ İLETİM/ALIŞ):

8286/8287 alıcı-verici yongaları kullanılmak istenen ve minimum kipte çalışan sistemlerde kullanılan iğnedir. Alıcı-verici yongasından geçen verilerin yönünün ayarlanmasında kullanılır.

Mantıksal olarak DT/R' iğnesi maksimum kipteki S1' iğnesine eşdeğerdir ve zamanlaması ise M/IO' iğnesine eşdeğerdir. İğne yongayı mantıksal 1 durumunda iken verici, mantıksal 0 durumunda ise alıcı olarak ayarlar.

"bekleme alındılandı" durumunda üç durumlu kapalı durumunu alır.

26. İğne (DATA ENABLE – VERİ ETKİNLEŞTİR):

8286/8287 alıcı-vericisi kullanan minimum kipteki sistemler tarafından kullanılır. Alıcı-verici tümleşik devresinin çıkışını etkinleştirmek için kullanılır. Bellek ve girdi/çıktı işlemleri süresince ayrıca INTA çevrimlerinde mantıksal sıfır durumundadır ve etkindir.

Okuma ve INTA' çevrimlerinde T2'nin ortasından T4'ün ortasına kadar etkindir. Yazma çevrimlerinde ise T2'nin başından T4'ün ortasına kadar etkindir.

"bekleme alındılandı" durumunda üç durumlu kapalı durumunu alır.

30. ve 31. İğneler (HOLD – BEKLETME / HLDA – BEKLETME ALINDILANDI):

Başka bir ana veri yolunun bir yerel veri yolu "beklemesi" istemi gönderdiğini belirtir. Alındılanma yapılabilmesi için HOLD'un mantıksal 1 durumunda olması gerekmektedir. HOLD mantıksal 1 durumunda iken etkindir. "bekletme" istemini alan mikroişlemci T4'ün ya da T1'in ortasında HLDA iğnesini etkinleştirir. HLDA iğnesi mantıksal 1 durumunda iken etkindir. HLDA etkinleştirildikten sonra mikroişlemci yerel veri yolunu ve denetim hatlarını boşa alır.

HOLD mantıksal 0 durumuna getirildiğinde ise mikroişlemci HLDA iğnesini mantıksal 0 durumuna getirir. İşlemci bundan sonra bir çevrime girerse yerel veri yolunu ve kontrol hattını yeniden bağlar.

HOLD ve HLDA iğneleri kaynak gerilimine bağlanan dirençlerine sahiptirler.

RQ'/GT' için yerel veri yolunun boşa alınması için açıklanan kuralların hepsi HOLD ve HLDA iğneleri için de geçerlidir.

HOLD eşzamansız bir giriş değildir. Eğer sistem belirli bir kurulum süresini garantileyemiyorsa dışarıdan bir eşzamanla uygulanması gerekebilmektedir.

3.2.1. Bellek

8086'da komutlar (kod), veri ve yığıt aynı bellek alanlarında tutulur. 8086'nın 20 bitlik adres yolu sayesinde 2^{20} = 1MBlık bellek erişimi sağlanabilir. Ancak işlemcide işlenebilen komutlarının pek çoğunun adresleme alanı 16 bitlik olduğundan, çoğu komutla 2^{16} = 65536 bayt = 64kB bellek erişimi yapılabilmektedir. 64 kB bellek erişiminin üzerine çıkabilmek için özel bölüt yazmaçları kullanılmaktadır.

16 bitlik veriler ve imleçler aşağıdaki gibi tutulur:

ADRES + 0000H	ADRES + 0001H
Alt Bayt	Üst Bayt

Dikkat edilirse adresin alt baytı ilk, üst baytı bir sonraki adreste tutulmaktadır. Böyle bir durumun oluşmasının nedeni "Küçüğü Başta" ya da "Sağdan Anlamlı" (Little endian) Kuralı'dır. Bu kurala ileriki bölümlerde değinilecektir.

32 bitlik adresler BÖLÜT ADRESİ : GÖRECELİ KONUM ADRESİ biçiminde gösterilir. Bu ikili adresleme şekli ile gerçek bir adrese işaret edilir. 8086'da 32 bitlik adres aşağıdaki şekilde tutulur:

ADRES + 0000H	ADRES + 0001H	ADRES + 0002H	ADRES + 0003H
Alt Bölüt Baytı	Üst Bölüt Baytı	Alt Göreceli Konum Baytı	Üst Göreceli Konum Baytı

Bölüt ve göreceli konum kavramları çevirici dilinin komutları incelenirken daha ayrıntılı işlenecektir.

3.2.1.1. Komut (Kod) Belleği

Yazılan kodlar belleğin herhangi bir yerinde tutulabilir. Zorunlu olarak yazılması gereken bir yer yoktur. Aslında kod mikroişlemcinin anlayabildiği komutlar ve bu komutlar ile birlikte gelen verilerdir.

Örneğin aşağıdaki çevirici kodu incelenirse:

```
1  MOV AL, 'S'
2  MOV BL, AL
3  MOV AL, 'C'
4  MOV BX, 1234H
```

Birinci satırda AL yazmacına 'S' ASCII karakterinin değeri olan 53H sayısı kaydedilir.

Ardından ikinci satırda BL yazmacına AL yazmacının değeri kaydedilir. AL yazmacının o andaki değeri birinci satırdaki işlemden dolayı 'S' ASCII karakterinin değeri olduğundan BL yazmacına 'S' karakterinin ASCII değeri yazılır.

Üçüncü satırda AL yazmacına 'C' ASCII karakterinin değeri olan 43H sayısı kaydedilir. Dördüncü satırda ise BH yazmacına AL yazmacının değeri ('C' ASCII karakteri) yazılır.

Yukarıdaki kodun işlevinin anlaşılması bu aşamada önemli değildir. Asıl önemli nokta yukarıdaki çevirici kodunun nasıl bir koda çevrileceği ve mikroişlemcinin bu kodu nasıl işleyeceğidir.

Yukarıda yazdığımız çevirici kodu herhangi bir çevirici ile çevrilip belleğin 0000:0000 adresine kaydedilirse aşağıdaki tablodaki koda dönüşür. Birinci sütuna bellek adresi yazılmıştır. İkinci sütuna ise belleğin birinci sütunda belirtilen adresinde tutulmakta olan veri yazılmıştır.

Bellek Adresi	Adreste Saklanan Veri
0000:0000	B0H
0000:0001	53H
0000:0002	8AH
0000:0003	D8H
0000:0004	B0H
0000:0005	43H
0000:0006	BBH
0000:0007	34H
0000:0008	12H

Mikroişlemcinin komutları okumaya 0000:0000 adresinden başladığını varsayarak kod mikroişlemcinin işlediği biçimde aşağıdaki gibi işlenir:

1.	0000:0000 bellek adresi okunur
2.	Okunan veri MOV AL, DEĞER komutuna karşılık gelmektedir
3.	AL yazmacı 8 bit olduğundan 1 baytlık veri okunacağı anlaşılır
4.	Bir sonraki bellek adresi olan 0000:0001 adresi okunur
5.	53H verisi okunur. Bu veri, komutun özelliği gereği AL yazmacına yazılmalıdır
6.	0000:0002 bellek adresi okunur
7.	Okunan veri 8AH'dir. Bu veri MOV BL, YAZMAÇ komutuna karşılık gelmektedir
8.	BL yazmacının 8 bitlik bir yazmaç olduğunu bilindiğinden 1 baytlık veri okunur
9.	0000:0003 bellek adresindeki veri okunur
10.	D8H verisi AL yazmacını simgeler. Bu yüzden AL yazmacının değeri BL yazmacına kaydedilir
11.	0000:0004 bellek adresi okunur
12.	Okunan kod B0H'dir. Bu kod MOV AL, DEĞER komutuna karşılık gelmektedir
13.	AL yazmacı 8 bitlik olduğundan 1 bayt veri okunmalıdır
14.	0000:0005 bellek adresindeki veri okunur
15.	43H verisi AL yazmacına kaydedilir
16.	0000:0006 bellek adresi okunur
17.	Okunan veri BBH'dir. Bu veri MOV BX, DEĞER komutuna karşılık gelmektedir
18.	BX yazmacı 16 bitlik olduğundan 2 bayt okunmalıdır
19.	0000:0007 adresindeki veri okunur ve BX yazmacının alt baytına yazılır
20.	0000:0008 adresindeki veri okunur ve BX yazmacının üst baytına yazılır

Yukarıdaki adımlar izlenerek çevirici (assembly) dilinde aşağıdaki çevirici kodu ile ifade edilen işlemleri yapmıştır:

```
1  MOV AL, 'S'
2  MOV BL, AL
3  MOV AL, 'C'
4  MOV BX, 1234H
```

Mikroişlemci kodu okumaya 0000:0000 adresi yerine 0000:0001 adresinden başlarsa işlemler aşağıda belirtildiği gibi gerçekleşir:

Bellek Adresi	Adreste Saklanan Veri
0000:0001	53H
0000:0002	8AH
0000:0003	D8H
0000:0004	B0H

0000:0005	43H
0000:0006	BBH
0000:0007	34H
0000:0008	12H

Yine aynı şekilde yukarıdaki kod mikroişlemcinin işlediği biçimde incelenirse:

1.	0000:0001 adresindeki veri okunur
2.	Okunan veri 53H'dir. Bu veri PUSH BX komutuna karşılık gelmektedir.
3.	PUSH BX komutu herhangi bir işlenen almadığı için bellekten veri okunmasını gerektirmez. Komutun özelliği gereği BX yazmacının değeri okunarak okunan veri yığıt belleğine kaydedilir.
4.	0000:0002 bellek adresi okunur
5.	Okunan veri 8AH'dir. Bu veri MOV BL, YAZMAÇ komutuna karşılık gelmektedir
6.	BL yazmacının 8 bitlik bir yazmaç olduğu bilindiğinden 1 baytlık veri okunmalıdır
7.	0000:0003 bellek adresindeki veri okunur
8.	D8H verisi AL yazmacını simgeler. Bu yüzden AL yazmacının değeri BL yazmacına kaydedilir
9.	0000:0004 bellek adresi okunur
10.	Okunan kod B0H. Bu kod MOV AL, DEĞER komutuna karşılık gelmektedir
11.	AL yazmacı 8 bitlik olduğundan 1 bayt veri okunmalıdır
12.	0000:0005 bellek adresindeki veri okunur
13.	43H verisi AL yazmacına kaydedilir
14.	0000:0006 bellek adresi okunur
15.	Okunan veri BBH'dir. Bu veri MOV BX, DEĞER komutuna karşılık gelmektedir
16.	BX yazmacı 16 bitlik olduğundan 2 bayt okunmalıdır
17.	0000:0007 adresindeki veri okunur ve BX yazmacının alt baytına kaydedilir
18.	0000:0008 adresindeki veri okunur ve BX yazmacının üst baytına kaydedilir

Yukarıdaki işlemlerin çevirici dilindeki karşılığı aşağıdaki gibidir:

1	PUSH BX
2	MOV BL, AL
3	MOV AL, 'C'
4	MOV BX, 1234H

Aslında mikroişlemci için kod, veri ya da yığıt gibi kavramlar yoktur. Mikroişlemci yalnızca birler ve sıfırlarla ilgilenir. Mikroişlemci, sistem ilk açıldığında belleğin bir hücresine erişerek 1 bayt okur. Bu baytın değeri eğer mikroişlemcide bir komut olarak algılanıyorsa, bu algılanan komutun işlevine göre bir sonraki bayt okunur ya da okunmaz. Sistemin tüm işleyişi okunan ilk veriye bağlıdır.

3.2.1.2. Veri Belleği

Mikroişlemci kod bölütüne, veri bölütüne, yığıt bölütüne ve ek bölütteki verilere erişebilir. Her bölüt 64 kB lık bir bellek alanını adresleyebildiğinden toplamda 256 kB veriye erişilmesi mümkündür. Yığıt, kod, veri ve ek bölütün tam olarak ne olduğu bir sonraki bölümde açıklanacaktır.

Veri belleğine veri kaydederken ya da bu verilere erişirken dikkat edilmesi gereken bir durum vardır. Eğer 16 bitlik bir veri bellek adresi tek sayı olan bir bellek hücresinden başlanarak kaydedilecekse işlemci bunu iki bellek erişimiyle tamamlayabilir. 16 bitlik veri adresi çift sayı olan bir bellek hücresinden başlanarak kaydedilecekse tek bellek erişimiyle kayıt tamamlanır.

3.2.1.3. Yığıt Belleği

Yığıt belleği, belleğin herhangi bir yerinde bulunabilir. Yığıt bir sonraki bölümde açıklanacağı gibi geçici kayıt işlemlerinde kullanılabilmektedir. Yığıta erişim sağlayan komutlar bazı yazmaç erişim komutlarından çok daha hızlı olduğundan geçici kayıt işlemlerinde kullanılabilmektedir.

Veri belleği erişiminde olduğu gibi yığıt erişiminde sonu sıfırla biten adresten başlamak başarımı artırır.

3.2.1.4. Ayrılmış Bölgeler

Belleğin bazı bölgeleri, bir takım özel verileri tutmak için ayrılmıştır. Örneğin belleğin 0000H ile 03FFH adresleri arasında kalan bellek alanı kesme yöneyleri (vektörleri) için ayrılmıştır. FFFF0H ya da FFFFFH adresleri bilgisayar ilk açıldığında ya da yeniden başlatıldığında başlangıç adresi olarak kullanılır.

3.2.2. Kesmeler

Kesmeler, işlemcinin içinde çalışan programın akışını durdurup denetimi özel bir kesme yordamına aktaran özel komutlar ya da sinyallerdir. Çoğu kesme komutunun işlenmesinin ardından kesme yordamı işini tamamladığında program kaldığı yerden çalışmayı sürdürür. Kesmeler yazılım ya da donanım kullanılarak uygulanabilir. Kesmeler konusunu yedinci bölümde ayrıntılı olarak incelenecektir.

3.2.3. Girdi/Çıktı Kapıları

8086'da girdi/çıktı işlemleri bellek adresleri ile aynı biçimdedir ve 65536 tane bayt yazmacını ya da 32768 tane sözcük yazmacını adresleyebilir. İşlemcinin dış dünya ile olan bağlantısını sağlayan girdi/çıktı kapıları birçok işlemde kullanılır. Girdi/çıktı kapıları ve bu kapıların kullanımları sekizinci bölümde ayrıntılı olarak incelenecektir.

3.2.4. Yazmaçlar

8086'nın sahip olduğu yazmaçlar aşağıda sıralanmıştır:

AX
BX
CX
DX

DI
SI

SP
BP

IP															
-	-	-	-	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

CS
DS
ES
SS

8086'nın yukarıda sıralanan yazmaçları aşağıda tek tek açıklanmıştır:

3.2.4.1. CS Yazmacı

CS (**C**ode **S**egment – Kod Bölütü) yazmacı 16 bitlik bir yazmaçtır. CS yazmacı işlemci tarafından IP yazmacı ile birlikte işlemcinin işleyeceği bir sonraki komutun adresi olan CS:IP adresini belirtir. CS yazmacının değeri herhangi bir atama ile doğrudan değiştirilemez. Uzun mesafeli CALL (CALL FAR) komutu,

uzun mesafeli geri dönme RET komutu ve uzun mesafeli JMP (JMP FAR) komutları ile değeri dolaylı olarak değiştirilebilir.

3.2.4.2. SS Yazmacı

SS (**S**tack **S**egment – Yığıt Bölütü) yazmacı CS yazmacı gibi 16 bitliktir. İşlemci SP ve BP yazmaçları ile adreslenen verilerin yığıt bölütünde olduğunu kabul eder. Yani kodda yazılmasa bile SP SS:SP ve BP SS:BP olarak kullanılır. SS yazmacının değeri POP komutu ile değiştirilebilir.

3.2.4.3. DS Yazmacı

DS (**D**ata **S**egment – Veri Bölütü) yazmacı 16 bitliktir. İşlemci AX, BX, CX, DX, SI, DI yazmaçları ile adreslenen verilerin veri bölütünde olduğunu kabul eder. POP ya da LDS komutları ile değeri değiştirilebilir.

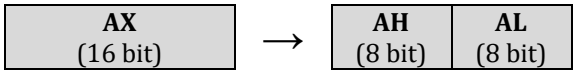
3.2.4.4. ES Yazmacı

ES (**E**xtra **S**egment – Ek Bölüt) yazmacı 16 bitliktir. Dizgi işlemlerinde işlemci DI yazmacı ile adreslenen verinin ek bölütte bulunduğunu varsayar. POP ya da LES komutları ile değeri değiştirilebilir.

> **Not:** İşlemcinin CS, ES, DS ve SS'de yapıldığını varsaydığı adreslemeler, yazılan kodun başına ES, CS, DS yada SS yerleştirilmesi yoluyla kullanılan bölüt açıkça belirtilerek ortadan kaldırılabilir.

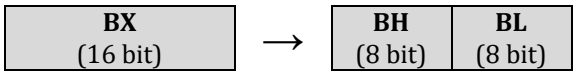
3.2.4.5. AX Yazmacı

AX (**A**ccumulator Register – Birikeç Yazmacı) 16 bitlik bir yazmaçtır. Üst ve alt 8 biti sırasıyla AH ve AL yazmaçları olarak kullanılır. Birikeç yazmacı olan AX genellikle dizgi işlemlerinde ve girdi/çıktı işlemlerinde kullanmaya uygundur.



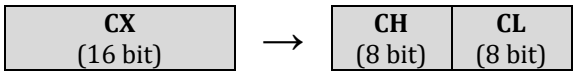
3.2.4.6. BX Yazmacı

BX (**B**ase Register – Taban Yazmacı) 16 bitlik bir yazmaçtır. Üst ve alt 8 biti sırasıyla BH ve BL yazmaçları olarak kullanılır. BX yazmacının değeri genellikle adreslemelerde taban adresi olarak kullanıldığı için taban yazmacı adını almıştır. Adreslemelere 4. bölümde ayrıntılı olarak değineceğiz.



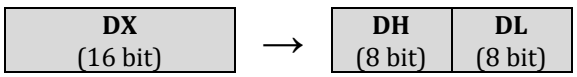
3.2.4.7. CX Yazmacı

CX (**C**ount Register – Sayaç Yazmacı) 16 bitlik bir yazmaçtır. Üst ve alt 8 biti sırasıyla CH ve CL yazmaçları olarak kullanılır. Sayaç yazmacı genellikle dizgi işlemleri gibi işlemlerde sayaç olarak kullanıldığından bu adı almıştır.



3.2.4.8. DX Yazmacı

DX (**D**ata Register – Veri Yazmacı) 16 bitlik bir yazmaçtır. Üst ve alt 8 biti sırasıyla DH ve DL yazmaçları olarak kullanılır. Girdi/çıktı komutlarında kapı numarası ve matematiksel işlemlerde ek veri tutulan yazmaçtır. Bu yüzden veri yazmacı adını almıştır.



> **Not:** AX, BX, CX, DX yazmaçlarına genel amaçlı yazmaçlar da denir. Bunun nedeni yalnızca belirtilen işler için değil, tamamen farklı işler için de kullanılabilmeleridir. Örneğin bu yazmaçlar geçici program değişkenlerinin değerlerini tutmak için de kullanılırlar. Benzer biçimde, özel yöntemler kullanılarak, bütün diğer yazmaçlar da başka işler için kullanılabilir.

3.2.4.9. SP Yazmacı

SP (**S**tack **P**ointer – Yığıt İmleci) yazmacı 16 bitlik bir yazmaçtır. İçerdiği değer yığıtı işaretlemektedir.

3.2.4.10. BP Yazmacı

BP (**B**ase **P**ointer – Taban İmleci) yazmacı 16 bitliktir. İçerdiği değer yığıttaki bir veriyi işaretlemektedir.

3.2.4.11. SI Yazmacı

SI (**S**ource **I**ndex – Kaynak Dizini) yazmacı 16 bitliktir. Adreslemelerde kaynak adresi olarak kullanılır.

3.2.4.12. DI Yazmacı

DI (**D**estination **I**ndex – Hedef Dizini) yazmacı 16 bitliktir. Adreslemelerde hedef adresi olarak kullanılır.

3.2.4.13. IP Yazmacı

IP (**I**nstruction **P**ointer – Komut İmleci) daha önce CS yazmacında da belirtildiği gibi CS yazmacı ile birlikte işlenecek olan komuta işaret eder.

3.2.4.14. Bayrak Yazmacı

Bayrak yazmacı 16 bitliktir ve içerdiği bitlerden 9'u bayrak biti olarak kullanılabilir.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	-	**AF**	-	**PF**	-	**CF**

OF (**O**verflow **F**lag – Taşma Bayrağı) biti işaretli sayılar üzerinde yapılan işlemlerin sonucunda sonuç -128 ve 127 sayıları arasında değilse 1 değerini alır. Aksi halde 0 değerindedir.

Örnek: 64H (-100) ve 3CH (-60) işaretli sayıları toplandığında sonuç FF60H (-160) olarak bulunur. -160 sayısı -128'den küçük olduğu için bir taşma oluşur ve OF bitinin 1 durumuna geçmesi gerekir.

DF (**D**irection **F**lag – Yön Bayrağı) biti bazı komutlar tarafından kullanılmaktadır. Eğer bu bayrak bitinin değeri 0 ise işlemler ileriye doğru, 1 ise işlemler geriye doğru yapılır. Dizgi işlemleri yapan komutlarda kullanılır.

IF (**I**nterrupt **F**lag – Kesme Bayrağı) biti 1 olduğu zaman işlemci çevre birimlerinden gelen kesmelere cevap verir. Aksi halde kesmeler yanıtlanmaz.

TF (**T**rap **F**lag – Yakalama Bayrağı) biti bazı özel uygulamalarda kullanılır. Örneğin bir kod adım adım çalıştırılmak isteniyorsa bu bayrak bitine 1 değeri verilir. Böylece bir sonraki komutun ardından bir yakalama kesmesi oluşur.

SF (**S**ign **F**lag – İşaret Bayrağı) bitinin değeri 1 ise yapılan son yapılan işlemin sonucu sıfıdan küçük, 0 ise yapılan işlemin sonucu sıfırdan büyüktür.

ZF (**Z**ero **F**lag – Sıfır Bayrağı) biti 1 ise yapılan işlemin sonucu 0'dır. Bitin değeri 0 ise son yapılan işlemin sonucu 0 dan farklıdır.

AF (**A**uxiliary **F**lag – Yardımcı Bayrak) biti 1 ise yapılan işlemin sonucunun alt 4 bitinde bir taşma oluşmuş demektir. Yalnızca işaretsiz sayılar için geçerlidir.

PF (**P**arity **F**lag – Eşlik Bayrağı) biti 1 ise yapılan işlemin sonucunda ortaya çıkan değerde çift sayıda 1 bulunmaktadır. PF biti 0 ise yapılan işlemin sonucunda tek sayıda 1 bulunmaktadır.

CF (**C**arry **F**lag – Elde Bayrağı) bitine işaretsiz sayılarla işlem yapılırken elde oluştuğu zaman 1 değeri kaydedilir. Aksi halde değeri 0 değerine sahiptir.

Örnek: 7FH (onluk tabanda 127) ile C8H (onluk tabanda 200) sayılarını toplarsak sonuç 013DH (317) bulunur. Görüldüğü gibi bu işlemin sonucunu tutmak için 2 basamak yetmemiştir ve üçünü basamağın kullanılması gerekmiştir. Böyle durumlarda CF biti 1 değerini alır.

3.2.5. Komutlar

8086 mikroişlemcisinin komut kümesi şu alt başlıklara ayrılabilir:

- **Veri Taşıma Komutları:** MOV, PUSH, POP, XCHG, IN, OUT, XLAT, XLATB, LEA, LDS, LES, LAHF, SAHF, PUSHF, POPF
- **Aritmetik İşlemi Komutları:** ADD, ADC, INC, AAA, DAA, SUB, SBB, DEC, NEG, CMP, AAS, DAS, MUL, IMUL, AAM, DIV, IDIV, AAD, CBW, CWD
- **Mantık İşlemi Komutları:** NOT, SHL, SAL, SHR, SAR, ROL, ROR, RCL, RCR, AND, TEST, OR, XOR
- **Dizgi İşleme Komutları:** REP / REPE / REPNE / REPNZ / REPZ, MOVSB / MOVSW, CMPSB / CMPSW, SCASB / SCASW, LODSB / LODSW, STOSB / STOSW
- **Denetimi Aktarma Komutları:** CALL, JMP, RET / RETF, JE / JZ, JL / JNGE, JLE / JNG, JB / JNAE, JBE / JNA, JP / JPE, JO, JS, JNE / JNZ, JNL / JGE, JNLE / JG, JNB / JAE, JNBE / JA, JNP / JPO, JNO, JNS, LOOP, LOOPZ / LOOPE, LOOPNZ / LOOPNE, JCXZ, INT, INTO, IRET
- **İşlemci Denetimi Komutları:** CLC, CMC, STC, CLD, STD, CLI, STI, HLT, WAIT, ESC, LOCK, NOP

8086 mikroişlemcisinin komut kümesi 4. Bölüm'de ayrıntılı olacak incelenecektir.

Yukarıda verilen komutlar makine diline çevrilerek işlemcinin iç elektronik donanımı tarafından kullanılmaya uygun hale getirilir. İşlemci için bir komutun MOV olması herhangi bir anlam taşımaz. İşlenen komutun 0 ve 1'lerden oluşan bir makine dili karşılığı olmalıdır. Makine dili 4. Bölüm'de ayrıntılı olarak açıklanacaktır.

3.2.6. Adresleme Kipleri

8086 mikroişlemcisinin kullandığı birçok adresleme kipi vardır. 8086'nın ve bu işlemcinin kullandığı çevirici dilinin kullanılabilmesi için adresleme kiplerinin çok iyi anlaşılması gerekir.

8086 mikroişlemcisinin adresleme kipleri 2 ana başlığa ayrılabilir:

- Yazmaç Adresleme Kipleri
- Bellek Adresleme Kipleri

3.2.6.1. Yazmaç Adresleme Kipleri

Yazmaçlar, bellek bölgelerine göre işlemciye daha yakın olduğundan erişim hızı belleğe göre çok yüksektir. Yazılımlar yazılırken bu yazmaçların bu özelliği göz önüne alınırsa başarım ciddi oranda artırılabilir.

Daha önce de belirtildiği gibi 8086 mikroişlemcisinde 16 bitlik yazmaçlar bulunmaktadır. 8086 mikroişlemcisinin komutları bu yazmaçlarla çalışabilmektedir. İşlenen kullanan bir komutta işlenen olarak bir yazmaç kullanılarak yazmaç adreslemesi yapılabilir.

Her komut çeviriciden (assembler) geçip makine diline çevrilir. Genelde çevirici dilindeki komutların tek bir makine dili karşılığı yoktur. Komutların farklı adresleme kipleri ve bu adresleme kiplerinde kullanılan yazmaçlar için değişik türevleri mevcuttur. Örneğin yazılımlar çevirici dili kullanılarak yazıldığında, MOV komutu yazmaçların verilerini bir yerden başka bir yere taşımada kullanılabileceği gibi belleğin bir adresindeki veriyi başka bir bellek adresine taşımada ya da bu işlemlerin hepsini karışık olarak yapmada kullanılabilir.

Aşağıdaki kod parçasında MOV komutunun yazmaç adreslemesi yapan bazı örnekleri verilmiştir.

```
1  MOV AL, CH
2  MOV BX, DX
3  MOV SI, DX
4  MOV SP, BP
```

Birinci satırdaki kod (MOV AL, CH) ile CH yazmacındaki veri AL yazmacına kaydedilmektedir.

İkinci satırdaki kod (MOV BX, DX) ile DX yazmacındaki veri BX yazmacına aktarılır.

Üçüncü satırdaki kod (MOV SI, DX) ile ise DX yazmacındaki veri SI yazmacına aktarılır.

Dördüncü satırdaki kod (MOV SP, BP) ile ise BP yazmacındaki veri SP yazmacına kaydedilir.

Yukarıda yapılan tüm işlemler yazmaç adreslemesine örnektir. Görüldüğü gibi MOV komutunun buradaki kullanım biçimiyle yalnızca yazmaçlar arasında işlem yapılmıştır.

3.2.6.2. Bellek Adresleme Kipleri

8086 mikroişlemcisinin değişik bellek adresleme kipleri bulunmaktadır.

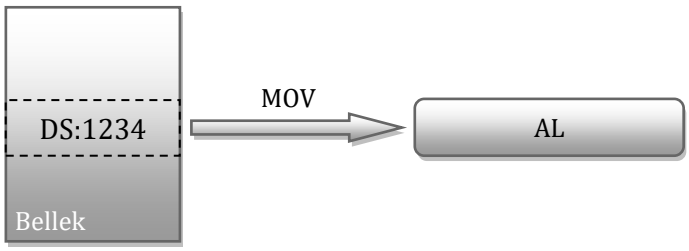
3.2.6.2.1. Doğrudan Adresleme Kipi

Doğrudan adresleme kipinde 16 bitlik sabit bir değer kullanılır. Bu sabit değer adres olarak kullanılarak belleğe erişilir.

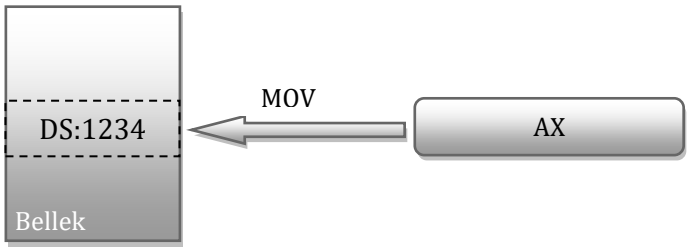
Örnek: Aşağıdaki kod parçası doğrudan adreslemeye örnektir:

```
1  MOV AL, DS:[1234H]
2  MOV DS:[1234H], AX
```

Kod parçasının birinci satırıyla (MOV AL, DS:[1234H] satırında) DS:1234 adresinden 1 baytlık veri AL yazmacına kaydedilmektedir.



İkinci satırda AX yazmacındaki iki baytlık veri DS:1234 adresine yazılmaktadır.



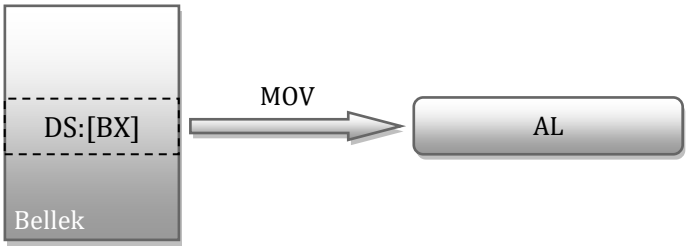
Yukarıdaki şekillerde de görüldüğü gibi 16 bitlik değişmez 1234H değeri adres olarak kullanılarak bellek adreslemesi yapılmıştır.

3.2.6.2.2. Yazmaç Değeriyle Dolaylı Adresleme Kipi

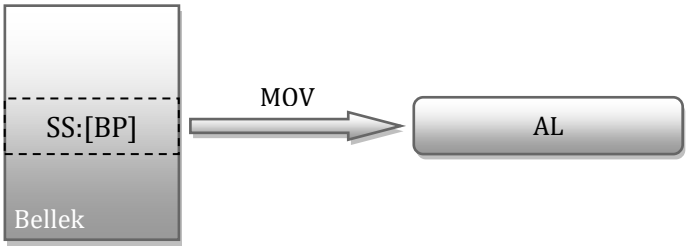
Yazmaç değeriyle dolaylı adresleme kipi kullanıldığında bir yazmacın değeri adres olarak kullanılarak herhangi bir bellek adresindeki veriler değiştirilebilir. Bu adresleme kipinin 4 farklı çeşidi vardır. Aşağıdaki kod parçasında bu dört farklı çeşit adresleme kipine örnekler sıralanmıştır:

```
1  MOV AL, [BX]
2  MOV AL, [BP]
3  MOV AL, [SI]
4  MOV AL, [DI]
```

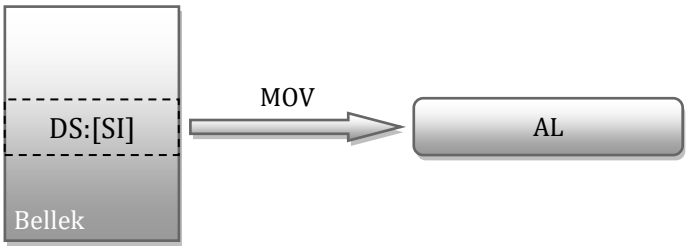
Yukarıdaki kod parçasında ilk satırda gösterilen adresleme yönteminde (MOV AL, [BX] satırında), bellek erişimi için kullanılacak adresin oluşturulması için bölüt adresi değeri DS yazmacının o andaki değeri, göreceli konum adresinin değeri ise BX yazmacının değeri olarak alınır. DS yazmacının değeri ile BX yazmacının değerinin birlikte oluşturduğu 32 bitlik adres erişilecek bellek adresidir. Başka bir deyişle, MOV AL, [BX]satırındaki adresleme yöntemiyle aşağıdaki şekilde de görüldüğü gibi AL yazmacına DS:[BX] adresindeki veri kaydedilmiştir.



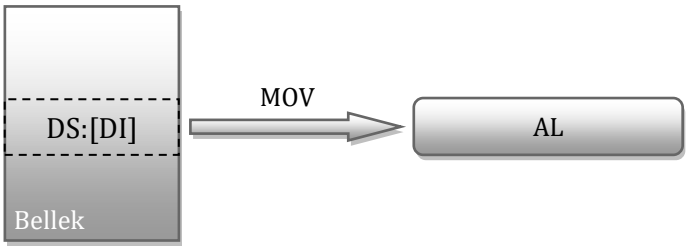
İkinci satırdaki adresleme yönteminde (MOV AL, [BP] satırında), göreceli konum adresini BP yazmacının değeri belirlerken, bölüt adresi olarak SS yazmacının değeri kullanılır. Böylece AL yazmacına SS:[BP] adresindeki veri kaydedilmiştir.



Üçüncü satırda gösterilen adresleme yönteminde (MOV AL, [SI] satırında), göreceli konum adresi olarak SI yazmacının değeri, bölüt adresi olarak da [BX] adreslemesinde olduğu gibi DS bölüt yazmacının değeri kullanılır. Böylece AL yazmacına DS:[SI] adresindeki veri yazılır.



Dördüncü satırda gösterilen adreslemede (MOV AL, [DI] satırında) ise göreceli konum adresi olarak DI yazmacının değeri ve bölüt adresi olarak DS kesim yazmacının değeri kullanılır. Böylece AL yazmacına DS:[DI] adresindeki veri yazılır.



Eğer varsayılan bölüt adresi yazmacı ([BP] için SS yazmacı, [BX], [SI] ve [DI] için DS yazmacı) yerine başka bir bölüt adresi yazmacının kullanılması istenirse satırın başında istenilen bölüt yazmacı açıkça belirtilmelidir. Örneğin [BX] için CS bölüt yazmacı, [SI] için DS bölüt yazmacı, [DI] için SS bölüt yazmacı ve [BP] için ES bölüt yazmacının kullanılması için kod parçası aşağıdaki biçimde yazılmalıdır:

```
1  CS MOV AL, [BX]
2  ES MOV AL, [BP]
3  DS MOV AL, [SI]
4  SS MOV AL, [DI]
```

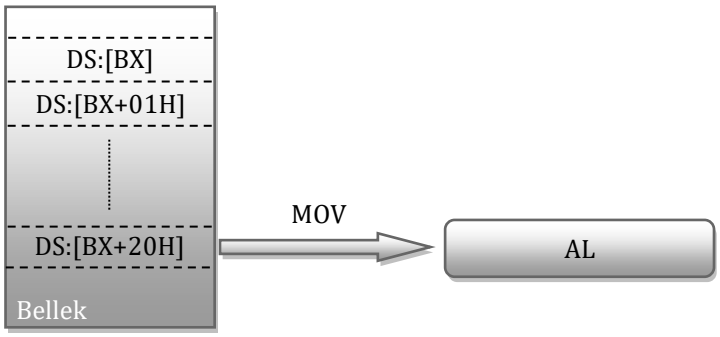
Varsayılan bölüt yazmaçlarının yerine bölüt yazmacı atanmasının uygulaması çeviriciden çeviriciye farklılık gösterir. Örneğin yukarıdaki kod bu kitaptaki tüm örnekler için kullanılan FASM çeviricisine uygundur.

3.2.6.2.3. Eklemeli Dizinle Adresleme Kipi

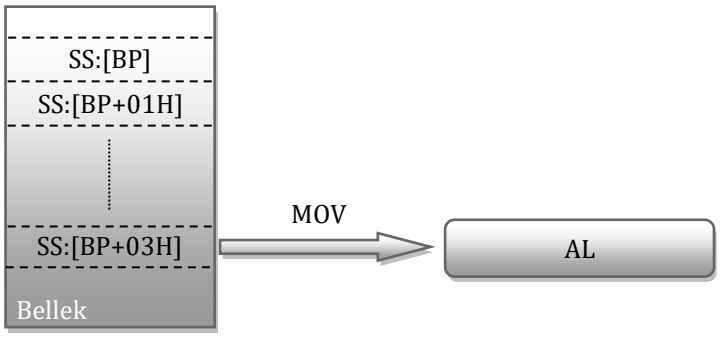
Eklemeli dizinle adresleme yönteminde yazmaçla dolaylı adresleme kipine benzer bir yol izlenir. Yazmacın yalnız başına adres olarak kullanılması yerine yazmaçtan okunan değer bir sayıyla toplanır.

```
1  MOV AL, [BX+20H]
2  MOV AL, [BP+03H]
3  MOV AL, [SI+04H]
4  MOV AL, [DI+07H]
```

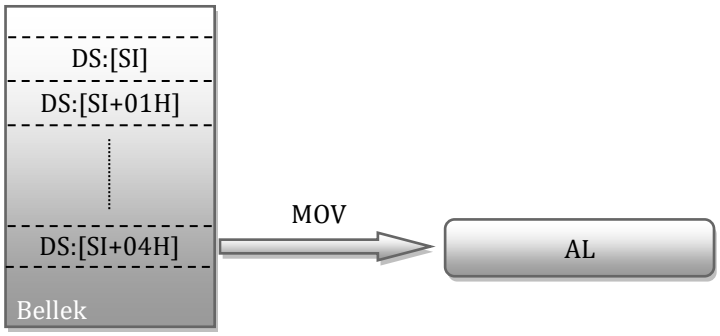
Yukarıdaki kod parçasında ilk satırdaki MOV AL, [BX+20H] ifadesinde göreceli konum adresi BX yazmacının değerinin 20H fazlasıdır ve bölüt adresi DS yazmacının o andaki değeridir. Bu durum aşağıdaki şekilde de gösterilmiştir.



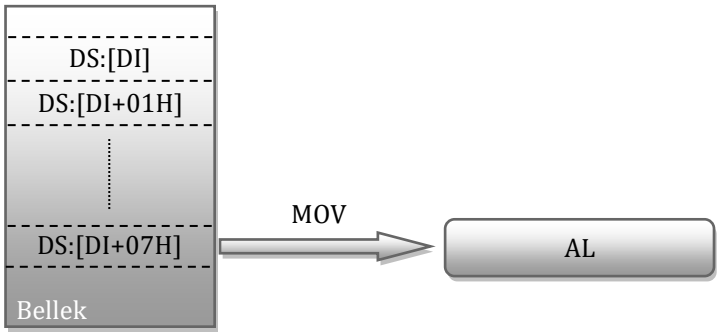
İkinci satırdaki MOV AL,[BP+03H] ifadesinde göreceli konum adresi BP yazmacının değerinin 03H fazlasıdır ve bölüt adresi değeri olarak SS yazmacının değeri kullanılır. Aşağıdaki şekilde bu durum gösterilmektedir.



Üçüncü satırdaki adresleme yönteminde göreceli konum adresi olarak SI yazmacının değerinin 04H fazlası, bölüt adresi değeri olarak da [BX] adreslemesinde olduğu gibi DS bölüt yazmacının değeri kullanılır.



Dördüncü satırdaki adreslemede ise göreceli konum adresi olarak DI yazmacının değerinin 07H fazlası ve bölüt adresi değeri olarak DS kesim yazmacının değeri kullanılır.



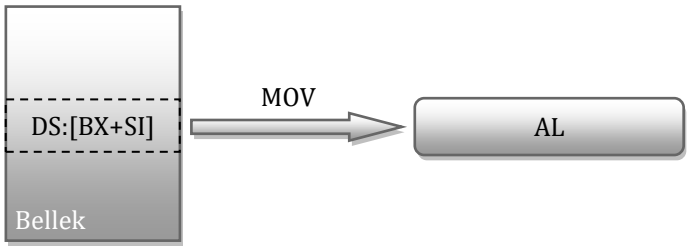
3.2.6.2.4. Tabanlı Dizinle Adresleme Kipi

Tabanlı dizinle adresleme kipinde taban yazmaçları olarak adlandırılan BX ya da BP yazmaçlarının değerleri ilk taban değerini oluşturup, diğer bir yazmacın değerinin bu taban değerine eklenmesi ile göreceli konum adresi belirlenir.

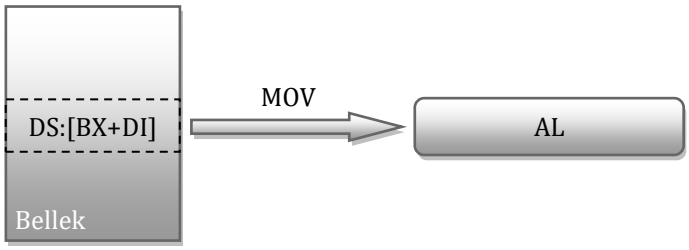
Aşağıdaki kod paçasında tabanlı dizinle adresleme kipinin bazı örnekleri gösterilmiştir.

```
1  MOV AL, [BX+SI]
2  MOV AL, [BX+DI]
3  MOV AL, [BP+SI]
4  MOV AL, [BP+DI]
```

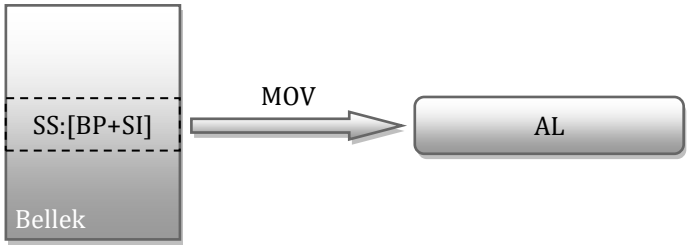
Yukarıdaki programın ilk satırında (MOV AL, [BX+SI] satırında) göreceli konum adresi olarak BX ve SI yazmaçlarının değerlerinin toplamı kullanılır. BX yazmacı taban olarak kullanıldığından işlemci veri bölütünün adreslendiğini varsayacak ve bölüt yazmacı olarak DS kullanılacaktır.



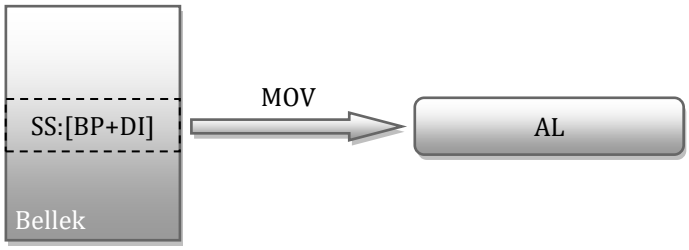
Kod parçasının ikinci satırında (MOV AL, [BX+DI] satırında) göreceli konum adresi olarak BX ve DI yazmaçlarının değerlerinin toplamı kullanılmaktadır. BX yazmacının kullanılıyor olması nedeniyle bölüt yazmacı olarak DS kullanılır.



Üçüncü satırda (MOV AL, [BP+SI] satırında) göreceli konum adresi olarak BP ve SI yazmaçlarının değerlerinin toplamı kullanılır. Taban değeri olarak BP yazmacının değeri kullanıldığından işlemci yığıt bölütünün adreslendiğini varsayacağından bölüt yazmacı olarak SS yazmacı kullanılır.



Dördüncü satırda ise (MOV AL, [BP+DI]satırında) göreceli konum adresi olarak BP ve DI yazmaçlarının değerlerinin toplamı kullanılır. BP yazmacı kullanıldığı için bölüt yazmacı olarak SS yazmacı kullanılır.

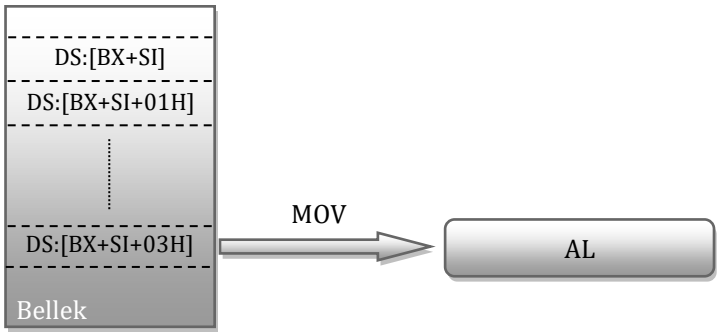


3.2.6.2.5. Tabanlı ve Eklemeli Dizinle Adresleme Kipi

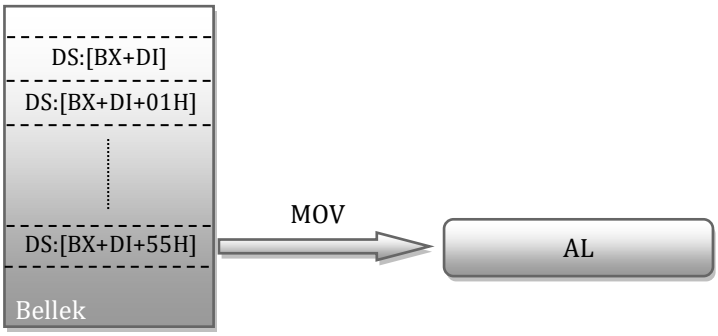
Tabanlı ve eklemeli dizinle adresleme kipinde hem taban yazmaçlarının değerleri hem de eklemeler kullanılır. Aşağıda tabanlı eklemeli dizinle adresleme kipine örnek olarak bir kod paçası verilmiştir. Programın ilk satırında (MOV AL, [BX+SI+03H]satırında) göreceli konum adresi olarak BX yazmacının değeri, SI yazmacının değeri ve 03H değerinin toplamı kullanılır.

```
1  MOV AL, [BX+SI+03H]
2  MOV AL, [BX+DI+55H]
3  MOV AL, [BP+SI+25H]
4  MOV AL, [BP+DI+0FH]
```

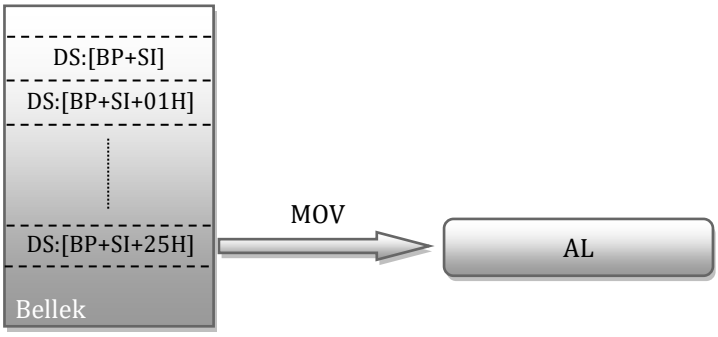
İlk satırda göreceli konum adresinin hesaplanması için BX kullanıldığından bölüt yazmacı olarak DS kullanılır. Aşağıdaki şekilde bu adreslemenin nasıl yapıldığı gösterilmiştir.



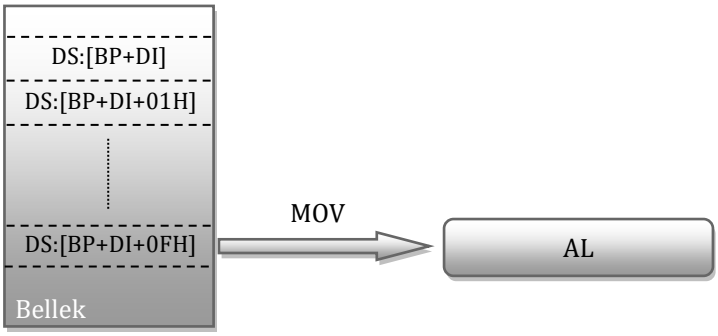
İkinci satırda (MOV AL, [BX+DI+55H]satırında) göreceli konum adresi olarak BX yazmacının değeri, DI yazmacının değeri ve 55H değerinin toplamı kullanılır. Bölüt yazmacı olarak DS yazmacı kullanılır. Aşağıdaki şekil işlemi özetlemektedir.



Üçüncü satırda göreceli konum adresi olarak BP yazmacının değeri, SI yazmacının değeri ve 25H değerinin toplamı kullanılır. Bölüt yazmacı olarak SS yazmacı kullanılır.



Dördüncü satırda ise göreceli konum adresi olarak BP yazmacının değeri, DI yazmacının değeri ve 25H değerinin toplamı kullanılır. Bölüt yazmacı olarak SS yazmacı kullanılır. Aşağıdaki şekil işlemi özetlemektedir.



4. ÇEVİRİCİ DİLİNE GİRİŞ

Bu bölümde assembly diline kısa bir giriş yapılacaktır. Öncelikle veri türleri ve bölütler hakkında bilgiler verilecektir. Ardından çevirici, bağlayıcı, konumlandırıcı ve hata ayıklayıcıları tanıtılıp kullanımlarına örnekler verilecektir. Çevirici dil kodlarının makine diline nasıl çevirildiği hakkında bilgiler verildikten sonra 8086 Komut Kümesi'ni tanıtmak amacıyla kısa açıklamalar ve kısa örnekler verilecektir. Komutların ve örneklerin hepsinin anlaşılmaması bu aşamada sorun değildir. Gelecek bölümlerde komutlar kullanımları üzerine daha geniş açıklamalar ve örnekler verilecektir.

Günümüzde birçok yüksek düzeyli dil varken çevirici dilinin öğrenilmesinin ve kullanılmasının 3 nedeni vardır:

- Çevirici dili ile oluşturulmuş bir program daha az yer kaplar.
- Çevirici dili ile oluşturulmuş bir program yalnızca yapması gereken işlemleri yaptığından daha hızlıdır.
- Çevirici dili kullanılarak yüksek düzeyli dillerin yapamadığı bazı işlemler yapılabilir.

Günümüzde bilgisayarlar hız ve sığım (kapasite) yönünden ne kadar güçlü olsalar da daha hızlı ve daha geniş sığımlı bilgisayarlar piyasaya sürüldüğünde sistemler yenilenmektedir. Çevirici dili ile yazılan programların hız ve sığım açısından tasarruf sağladığı düşünülürse günümüzün yüksek hızlı ve yüksek belleğe gereksinim duyan bilgisayarlarında bile çevirici dilinin ne kadar önemli olduğu anlaşılacaktır.

Çevirici dili ilk bakışta çok karmaşık ve anlamsız gelebilir; ancak çevirici dilinin ana mantığı kavrandığı zaman yüksek düzeyli dillerden daha "mantıklı" olduğu anlaşılacaktır. Ayrıca iyi bir çevirici dili bilgisi yüksek düzeyli diller ile program yazarken yazılımcıya yardımcı olacaktır. İşlemcilerin çalışma mantıkları çok basittir. Çağdaş mikroişlemciler yapısal olarak çok karmaşık olsalar bile çevirici dilin ana mantığından ayrılmamaktadırlar. Tüm işlemciler bir sonraki komutu alır, işler ve diğer komuta geçerler. Çevirici dili ile program yazarken bu işlemler açıkça görülebilmektedir. Her komut işlenenleri ile birlikte işlenir ve sıraki komuta geçilir.

4.1. Veri Türleri

İşlemcilerin değişik türlerde verileri kullanarak işlem yapabilir. Bu veri türleri daha çok, işlemcinin donanımının özelliklerine uygun olarak tanımlanır. Çevirici düzeyinde kodlama yapan programcının işlemci donanımın özelliklerini ve bu donanımla birlikte kullanılabilecek veri türlerini anlaması önemlidir.

En küçük veri türü bittir. Bir bit ya 0 ya da 1 değerine sahip olabilir. 4 tane bitin birlikte oluşturduğu veriye yarım bayt (nibble) denir. Bir yarım baytın göstereceği verinin değeri 0'dan 15'e kadardır. Bir bayt, 2 yarım bayttan ya da 8 tane bitten oluşur. Bir baytlık bir verinin alabileceği değerler 0 ile 255 arasında olabilir. Bir sözcük (word) iki bayttan oluşurve alabileceği değerler 0 ile 65535 arasındadır. Ayrıca literatürde 32 bitten oluşan ikili sözcükler (double word) ve 64 bitten oluşan dörtlü sözcükler (quad word) de bulunmaktadır. Aşağıdaki tabloda veri türleri ve bu veri türlerinin alabileceği değerler özetlenmiştir.

Adı	Bit Sayısı	Değer Aralığı
Bit	1	0 – 1
Yarım Bayt (nibble)	4	0 – 15
Bayt (byte)	8	0 – 255
Sözcük (word)	16	0 – 65.535
İkili Sözcük (double word)	32	0 – 4.294.967.295
Dörtlü Sözcük (quad word)	64	0 – 18.446.744.073.709.551.615

8086 işlemcisi 16 bitlik olduğundan kullanılan veri türleri bit, yarım bayt, bayt, sözcük ve bazen de ikili sözcüklerdir.

Çevirici dilinde bir değişken (bir bellek bölgesi) tanımlayıp ona değer atamak için **DB** (**d**efine **b**yte – bayt tanımla) ve **DW** (**d**efine **w**ord – sözcük tanımla) kodları kullanılır. Bu kodlar aşağıdaki biçimde kullanılır:

Değisken_adı* DB *değer
Değişken_adı* DW *değer

Değişken_adı bir harf ile başlamalıdır. Herhangi bir harf-sayı birleşimi değişken adı olarak kullanılabilir. Çevirici dili ile program yazılırken kullanılan bu değişkene ait *değer*'e bu *Değişken_adı* etiketi ile

ulaşılabilir. Eğer *Değişken_adı* yazılmazsa herhangi bir etiket belirtilmemiş ve yalnızca belirtilen *değer*'i taşıyan bir bellek bölgesi ayrılmış olur. *değer* olarak ikilik, onluk ya da onaltılık tabanda bir sayı yazılabilir. İkilik tabanda bir sayı yazıldığında sonuna "B" harfi (**b**inary – ikilik) ve onaltılık tabanda bir sayı yazıldığında sonuuna bir "H" harfi (**h**exadecimal – onaltılık) yazılarak kullanılan sayının tabanı belirtilmelidir. Onluk tabanda bir sayı kullanılırken sayının kendisinin yazılması yeterlidir.

Yazılan sayının değerinin, DB tanımlaması kullanıldığında bir bayta sığması, DW tanımlaması kullanıldığında ise bir sözcüğe sığması gerekir. Tanımlamanın ardından herhangi bir değer yazılması yerine "?" işareti kullanılırsa değişkenin ilk değeri olarak herhangi bir değer belirlenmez ancak bellekteki bir adrese *Değişken_adı* etiketi verilmiş olur.

Örnek: Aşağıda bazı değişken tanımlamaları gösterilmiştir:

```
degiskenim  DB 01110011B
degiskenim2 DW 0102H
degiskenim3 DB ?
degiskenim4 DW 12345
```

Eğer bir değişken aşağıda gösterildiği gibi tanımlanırsa 05H verisi 6 kez art arda yazılarak değişkenin 6 baytlık verileri tutacağı tanımlanmış olur.

```
degiskenim5 DB 6 DUP(05H)
```

4.2. Bölütler

Çevirici dili ile yazılan programlarda en az 3 tane bölüt kullanılır: kod bölütü, veri bölütü ve yığıt bölütü. Bu bölütlerin dışında ayrıca bir de ek bölüt kullanılabilir. Kod bölütünde, işlenmek üzere yazılmış olan çevirici dili komutları bulunur. Veri bölütünde, kod bölütündeki komutlar tarafından işlenecek olan veriler yer alır. Yığıt bölütü geçici saklama işlemlerinde kullanılırken ek bölüt ise yedek bir veri bölütü gibi kullanılabilmektedir.

Bölütler bellekte 16 ile tam olarak bölünen bir adresten başlar ve bellekte 64KB boyutunda bir bellek alanını simgeleyebilirler. 64KB ile sınırlandırılmış olmalarının nedeni 8085 mikroişlemcisinde yalnızca 16 adres iğnesinin bulunmasıdır. 16 adres iğnesi ile 2^{16} tane bayt (64KB) adreslenebilmektedir. 8086'nın 20 adres iğnesi bulunmasına ve bu adresleme bitleriyle 1MBlık (2^{10} bayt = 1MB) bellek alanını gösterebilmesine karşın uyumluluk nedenlerinden dolayı 64KB 8086'da da sınır olarak kabul edilmiştir. Bu nedenle kod, veri ve yığıt bölütlerinin her biri en fazla 64KB olabilir. Bazı yöntemler kullanılarak bu 64KB'ın da dışına çıkılabilmektedir. Bölütlerin nasıl kullanıldığının anlaşılabilmesi için gerçek adres, sanal adres ve göreceli konum adresi kavramlarının bilinmesi gerekir.

Gerçek Adres, Sanal Adres ve Göreceli Konum Adresi

8086 işlemcisinde gerçek adres, 20 iğnelik adres kapısının iğnelerine atanan 20 bitlik adrestir. Bu 20 bitlik adres bellek arayüz devrelerindeki kod çözücüler tarafından çözülür. Böylece 8086'da 00000H'dan başlayarak FFFFFH'ye kadar olan ROM ya da RAM'daki 1MB'lık bellek bölgesi adreslenebilmektedir. Göreceli konum adresi bir bölüt içinde kalan ve 0000H'dan başlayarak FFFFH'ye kadar olan bölgeyi adreslemektedir. Sanal adres 2 kısımdan oluşur: birinci kısım bölüt adresini, ikinci kısım ise göreceli konum adresini belirtir. 0200H:0000H sanal adresinde 0200H bölüt adresi ve 0000H göreceli konum adresidir. Sanal adres verildiğinde gerçek adres bulunabilir. Bunun için bölüt adresi bir basamak sola kaydırılır (ya da 10H ile çarpılır) ve göreceli konum adresiyle toplanır. Örneğin 0200H:0000H sanal adresine denk gelen gerçek adres:

```
0200H x 10H          = 02000H
02000H + 0000H       = 02000H
```

şeklinde hesaplanır ve gerçek adresin 02000H olduğu bulunur. Burada dikkat edilmesi gereken nokta iki ayrı sanal adresin aynı gerçek adresle eşleşebileceğidir. Örneğin 0000H:2000H sanal adresine denk gelen gerçek adres:

```
0000H x 10H          = 00000H
00000H + 2000H       = 02000H
```

şeklinde hesaplanır ve bulunan gerçek adresin, 0200H:0000H sanal adresine denk gelen gerçek adres ile aynı gerçek adres (02000H) olduğu görülür. Burada herhangi bir hata yoktur. Değişik bölüt adresi - göreceli konum adresi katışımları aynı gerçek adrese işaret edebilir.

4.2.1. Kod Bölütü

Yazılan bir programın çalıştırılabilmesi için programı oluşturan komutların bir yerden çekilmesi ve çalıştırılması gerekir. 8086 işlemcisi program kodunu **CS** yazmacı ile gösterilen kod bölütünden (code segment) çeker ve çalıştırır. Kod bölütünden bir komutun okunması için komutun bellekte bulunduğu gerçek adresin bilinmesi gerekir. Gerçek adres, bölüt adresi ve göreceli konum adreslerinden hesaplanabildiğinden, bir komut, belleğin bir bölütünden çekilirken bu bölütün neresinden çekildiğini belirtmek amacıyla göreceli konum adresinin de bölüt adresi ile birlikte belirtilmesi gerekir. Bu yüzden bir komutun bellekten getirilmesi için CS'nin yanında **IP** (instruction pointer – komut imleci) adresi de verilmektedir. IP göreceli konum adresinin değerini tutmaya yarayan bir imleçtir. Bir komut çekildikten sonra, çekilmiş olan komut aksini belirtmediği sürece, IP 1 artırılır ve sıradaki göreceli konum adresine geçilir. Örneğin 0200:0000 adresindeki komut çekilip çalıştırıldıktan sonra 0200:0001 adresine geçilir. 0200:0001 sanal adresine denk gelen gerçek adres:

0200H x 10H = 02000H
02000H + 0001H = 02001H

şeklinde hesaplanarak 02001H olarak bulunur. CS:IP 1234:0200 iken ise gerçek adres:

1234H x 10H = 12340H
12340H + 0200H = 12540H

şeklinde hesaplanarak 12540H olarak bulunur. Programın içerdiği komutlar bellekten getirilip çalıştırılırken bazı komutlar aracılığıyla IP'nin değeri değiştirilebilir. JMP (**ju**mp – atla), JE (**j**ump if **e**qual – eşit ise atla), JNE (**j**ump if **n**ot **e**qual – eşit değilse atla) gibi komutlar IP adresinin değerini değiştirebilir. CS'nin değeri ise bölütler arası bir CALL (**call** – çağır) komutu uygulandığında IP adresiyle birlikte değişebilir. Ancak CALL komutuyla atlanan alt yordamda RET (**ret**urn – geri dön) komutu çalıştırıldığında CS ve IP eski değerine dönecektir. CS'nin değerini olağan yollarla kalıcı olarak değiştirebilen bir komut yoktur. Çevirici dilinin komutlarına geçildiğinde bu kavramlara yeniden değinilecektir.

Aşağıdaki tablo bir dizi komutun bellekte bulunduğu biçimi göstermektedir:

CS:IP	İşlem Komutu ve İşlenen	Çevirici Komutları
1234:0000	B3 53	MOV BL, 'S'
1234:0002	B4 43	MOV AH, 'C'
1234:0004	B9 12 00	MOV CX, 12H
1234:0007	BA 13 00	MOV DX, 13H
1234:000A	03 CA	ADD CX, DX

İşlemci, komutları işlemeye 1234:0000 adresinden başladığında 1234:0000 adresindeki B3H ile 1234:0001 adresindeki 53H verisini yakalar ve bu verilerin MOV BL, 'S' çevirici dil koduna denk geldiğini algılar. Bir başka deyişle, MOV BL, 'S' çevirici kodu bir çevirici ile çevrilirse onaltılık tabandaki B3 ve 53 baytlarına dönüşecektir. Çevrilmiş bu çevirici dil kodu 1234:0000 ve 1234:0001 adreslerinde bulunmaktadır. 1234:0000'da MOV komutunun değişmez/sabit bir verinin değerini (S harfinin ASCII değerini) BL yazmacına aktaran bir çeşidi olan B3H ve 1234:0001'de bu değişmez/sabit verinin değeri (S harfinin onaltılık tabandaki ASCII değeri) bulunmaktadır. B3H 53H işlendikten sonra 1234:0002 adresine geçilir.

1234:0002 adresinde AH yazmacına C harfinin onaltılık tabandaki ASCII değerini aktaran B4 işlem komutu ve 1234:0003 adresinde C harfinin onaltılık tabandaki değeri olan 43H bulunmaktadır. MOV komutu BL yazmacına değişmez bir veri aktarırken B3H işlem komutuna çevrilirken, AH yazmacına değişmez bir veri aktarılırken B4H işlem komutuna çevrilmektedir. Bunun nedeni MOV komutunun yalnızca bir tane işlem komutu bulunması durumunda bu işlem komutu ile yalnızca tek bir yazmaca veri aktarabilmesidir. B3H veya B4H gibi işlem komutlarının nasıl hesaplandığı çevirici dilinin komutları detaylı olarak açıklanırken belirtilecektir. Bu aşamada tek bilinmesi gereken bir komutun farklı işlem biçimleri olduğunda bu komutun çevrilmesinden sonra ortaya çıkan işlem komutlarının değişiklik göstereceğidir.

1234:0004 adresindeki veri okunduğunda MOV CX, 12H görülecektir. Bu kod bir işlem komutu ve iki tane işlenen olmak üzere 3 baytlık alan kaplamaktadır. MOV CX, 12H kodu 12H baytını CX yazmacına kaydeden bir koddur. MOV BL, 'S' 2 baytlık yer kaplarken, MOV CX,12H 3 baytlık yer kaplamasının nedeni BL yazmacının 8 bitlik olmasıdır. Üçüncü bölümde belirtildiği gibi 16 bitlik genel amaçlı yazmaçlar olan AX, BX, CX ve DX yazmaçlarının üst ve alt kısımları 8 bitlik yazmaç olarak ayrı ayrı kullanılabilmektedir. BL yazmacı 16 bit olan BX yazmacının 8 bitlik alt yazmacıdır. MOV CX, 12H kodu CX yazmacına bir baytlık 12H verisini, CX yazmacının alt yarısına 12H ve üst yarısına 00H olarak yazar. Böylece MOV komutu CX yazmacının üzerine uygulanarak hem CL hem de CH yazmaçları değiştirilmiş olur. CX yazmacının boyutunun toplam iki bayt

olması nedeniyle B9 ile kodlanan MOV komutu iki baytlık işlenen kullanmakta ve işlem komutuyla birlikte MOV komutu toplam üç bayt yer tutmaktadır. Burada dikkat edilmesi gereken nokta CX yazmacına aktarılan 0012H değerinin işlenen olarak 12 00 sırasıyla MOV komutuna verilmesidir. Bunun nedeni Intel mikroişlemcilerinin çoğunda (ve bu kitapta örnek olarak kullanılan 8086 işlemcisinde) "Küçüğü Başta (Little Endian)" Kuralı'nın kullanılmasıdır.

Bayt Dizilişi Kuralları:

Küçüğü Başta (Little Endian), Büyüğü Başta (Big Endian) ve Ortası Başta (Middle Endian)

Günümüzde işlemcilerin bir bayttan büyük sayıları işlerken ve saklarken kullandığı 3 farklı kural vardır: Küçüğü Başta (Little Endian), Büyüğü Başta (Big Endian) ve Ortası Başta (Middle Endian). Küçüğü Başta kuralı kullanan işlemcilerde veriler saklanırken alt baytlar önce, üst baytlar ise sonra yazılır. Örneğin CX yazmacına bir değer saklanması amaçlandığında, yazılacak veri sıralanırken CX yazmacının alt sekiz bitini gösteren CL yazmacına yazılacak veri önce, üst 8 bitini gösteren CH üst yazmacının alacağı değer ise sonra yazılır. 16lık tabandaki ABCDEF12H değeri küçüğü başta kuralına göre (örneğin 2578:0000 adresinden başlayarak) bellekte aşağıdaki biçimde saklanır:

KÜÇÜĞÜ BAŞTA

2578:0000	2578:0001	2578:0002	2578:0003
12	EF	CD	AB

Görüldüğü gibi en anlamlı bayt olan 4B en son yazılırken en az anlamlı olan bayt en başa kaydedilmiştir. Küçüğü başta kuralı uygulandığında saklanan sayının en anlamsız baytı en küçük bellek adresine yazılır ve diğer sayının diğer baytlar bu adres bir artırılarak saklanır.

Büyüğü Başta kuralı ise Küçüğü Başta kuralının tam tersidir. Büyüğü başta kuralında üst baytlar en küçük bellek adresine yazıldıktan sonra alt baytlar sonraki bellek konumlarına sıralanırlar. Büyüğü Başta kuralını kullanan bir sistemde ABCDEF12H değeri aşağıdaki biçimde saklanır:

BÜYÜĞÜ BAŞTA

2578:0000	2578:0001	2578:0002	2578:0003
AB	CD	EF	12

Yukarıdaki örnekte görüldüğü gibi, büyüğü başta kuralı kullanıldığında en anlamlı bayt en küçük bellek adresine yazılır ve diğer baytlar ters sıra ile artan bellek konumlarına yerleştirilir.

Ortası Başta kuralını kullanan işlemciler Küçüğü Başta ve Büyüğü başta kurallarının bir karışımını kullanır. Ortası başta kuralında saklanan sayının 16 bitlik yarılarının arasında büyüğü başta kuralının, bu yarıların içindeki baytların arasında ise küçüğü başta kuralının uygulandığı düşünülebilir. ABCDEF12H verisinin ortası başta kuralı kullanan bir işlemcide nasıl saklandığı aşağıdaki örnekte gösterilmiştir:

ORTASI BAŞTA

2578:0000	2578:0001	2578:0002	2578:0003
CD	AB	12	EF

Günümüzde masaüstü ve dizüstü bilgisayar piyasasının neredeyse tamamında kullanılan Intel x86 mimarisinde küçüğü başta kuralı kullanılmaktadır. Motorola tarafından üretilen işlemcilerde büyüğü başta kuralı kullanılırken, ARM işlemcileri hem küçüğü başta hem de büyüğü başta kuralını kullanabilmektedir.

Örnekte 1234:0007, 1234:0008 ve 1234:0009 adreslerindeki BA 13 00 baytları MOV DX,13H komutunu oluşturmaktadır. Önceki MOV CX,12H komutunda olduğu gibi ilk bayt olan BA, DX yazmacına veri aktarılacağını ardından gelen iki bayt ise aktarılacak olan değerin 13H olduğunu göstermektedir. 1234:000A ve 1234:000B adreslerinde ise ADD CX,DX kodu bulunmaktadır. Bu kod CX ve DX yazmaçlarının değerlerini toplayıp sonucu CX yazmacına yazar ve bellekte saklanan 03 CA baytları bu komutun çevrilmiş biçimidir.

4.2.2. Veri Bölütü

DS (Data Segment - Veri bölütü) yazmacının yerini gösterdiği veri bölütünün kullanımının sağladığı yararlar aşağıdaki örnek incelenerek anlaşılabilir.

```
   ...
1  MOV BL, 0FFH
2  SUB BL, 10H
```

```
3   ADD BL, 11H
    ...
```

Yukarıdaki kod parçasında BL yazmacına yazılan FFH sayısından ilk önce 10H değeri çıkarılarak sonuç BL yazmacına kaydedilmekte, ardından BL yazmacının değeri 11H ile toplanıp sonuç yeniden BL yazmacına yazılmaktadır. İleriki bölümlerde daha ayrıntılı açıklanacak SUB komutu birinci işleneninden ikinci işleneni çıkarıp sonucu birinci işlenenin gösterdiği yazmaca aktaran bir çevirici komutudur. Örnekteki kod parçacığının çok uzun bir programın küçük bir bölümü olduğu düşünülürse, BL yazmacına yüklenen FFH değeri AAH ile değiştirilmek istendiğinde bu program satırının çok sayıda satır arasından bulunup değiştirilmesi gerekir. Aynı biçimde BL yazmacından 10H yerine 01H değerinin çıkarılması istenirse yine bu kod satırının bulunarak ilgili değişikliğin yapılması gerekir. Böyle bir zorlukla karşılaşmamak için, çevirici dilini kullanan programcı, programın içinde kullandığı değerleri veri bölütüne kaydettikten sonra, verinin kendisini kullanmak yerine verinin bulunduğu bellek adresini kullanabilir. Örneğin, eğer veri bölütü 0400H göreceli konumundan başlıyor ise bellekte DS:0400 adresine FFH, DS:0401 adresine 10H ve DS:0402 adresine 11H verileri önceden kaydedilirse örnekte verilen kod parçası aşağıdaki biçimde yazılabilir:

```
    ...
1   MOV BL, [0400]
2   SUB BL, [0401]
3   ADD BL, [0402]
    ...
```

Yukarıdaki program parçasının ilk satırında BL yazmacına MOV komutu ile doğrudan bir sayı yollamak yerine sayının bellekte bulunduğu konumun adresi olan [0400] yazılmıştır. Köşeli parantezler çeviriciye parantezin içindeki sayının kendisinin değil sayının gösterdiği bellek adresindeki verinin BL yazmacına yazılması gerektiğini anlatır. Bu tür bir adreslemede bölüt adresi olarak DS kullanılır. Göreceli konum adresi ise köşeli parantezlerin içine yazılmış olan 0400H değeridir. Böylece program parçacığının ilk satırındaki MOV komutuyla DS:0400 bellek adresindeki bayt BL yazmacına yazdırılmaktadır. İkinci satırdaki SUB komutu BL yazmacının değerinden DS:0401 adresinde bulunan baytı çıkarıp sonucu BL yazmacına yazmaktadır. Program parçacığının üçüncü satırında ise BL yazmacının içerdiği değer ile DS:0402 adresindeki bayt toplanıp sonuç BL yazmacına yazdırılmaktadır.

Örnekte verilen kod parçacığının uzun bir program içinde aranıp verilerin değerlerinin tek tek değiştirilmesi yerine yalnızca DS:0400, DS:0401 ve DS:0402 adreslerindeki veriler değiştirilebilir. Aşağıdaki örnek yukarıda verilen kod parçacığının, üzerinde bazı değişiklikler yapılmış halini göstermektedir.

```
    ...
1   MOV SI, 0400H
2   MOV BL, [SI]
3   INC SI
4   SUB BL, [SI]
5   INC SI
6   ADD BL, [SI]
    ...
```

İlk bakışta bu kod parçacığı daha önceki kod parçacığından farklı bir işlem yapıyor gibi görünse de gerçekte işlevleri aynıdır. Birinci satırdaki MOV SI,0400H komutuyla SI (source index – kaynak dizini) yazmacına 0400H sayısı yazılır. İkinci satırdaki MOV BL,[SI] komutu SI yazmacının değerini göreceli konum adresi olarak kullanarak belleğe erişir ve ilgili bellek adresindeki veriyi BL yazmacına aktarır. Böylece DS:0400 bellek adresindeki veri BL yazmacına yazılmış olur. Üçüncü satırdaki INC SI komutu ise SI yazmacının değerini bir artırarak 0401H yapar. Artırılan SI değeri dördüncü satırdaki SUB komutunda BL yazmacından çıkarılacak verinin bellekteki yerini bulmak için kullanılır. Beşinci satırda SI yazmacının değeri bir artırılarak bir sonraki satırdaki ADD komutunun BL yazmacının değerini DS:0402 adresindeki değer kadar artırması sağlanır.

Görüldüğü gibi [ ] ayraçları arasına göreceli konum adresini yazmak yerine, bu ayraçların için bir yazmaç adı yazılarak kullanılan yazmacın değerinin göreceli konum adresi olduğu belirtilebilir. Burada dikkat edilmesi gereken konu veri bölütünün yalnızca BX, SI ve DI yazmaçlarının değerlerini göreceli konum adresi olarak kabul etmesidir. DS:BX, DS:SI ve DS:DI kullanımları veri bölütünü adreslemek için uygun olmasına karşın DS:AX (ya da BX, SI ve DI dışında herhangi bir yazmaç) kullanımı kurallara ayıkırıdır. Eğer köşeli ayraç içine kural dışı bir yazmaç adı yazılırsa, çevirici hata verir ve yazılan kod makine diline çevrilmez.

Veri bölütüne erişim, 8086 işlemcisinin bellek erişim kurallarına uygun olarak yapılır. Örneğin veri bölütünün adresinin 0200H ve veri bölütünün içinde erişilmek istenen verinin göreceli konum adresinin 0201H olduğu varsayılırsa bellekte erişilmesi gereken gerçek adres aşağıdaki biçimde hesaplanır:

0200H x 10H = 02000H
02000H + 0201H = 02201H

Yukarıdaki adres hesabı sonucu bellekte erişilecek bellek konumunun gerçek adresi 02201H olarak bulunur.

4.2.3. Yığıt Bölütü

SS (stack segment – yığıt bölütü) yazmacının bellekteki konumunu gösterdiği yığıt bölütü hem okunabilen hem de yazılabilen bir bellek parçasıdır ve genellikle ana belleğin içinde yer alır. Yığıt belleğinde veriler geçici olarak saklanır. İşlemcide bulunan yazmaçların sayısının bazı teknik ve mali nedenlerden dolayı az olması programların yazılmasını zorlaştırdığından yığıt gibi geçici bir saklama alanın varlığı programlar yazılırken büyük kolaylık sağlamaktadır. Yığıt belleği sağladığı kolaylığa karşın işlemciden uzak olan ana bellekte yer alması ve yazmaçların işlemcinin içinde bulunması nedeniyle yazmaçlara göre daha yavaş erişim sağlar.

Yığıtın başındaki verinin adresi SS ve **SP** (stack pointer – yığın imleci) yazmaçlarının birlikte kullanılmasıyla bulunur. Erişilen verinin gerçek adresi SS yazmacındaki veri bir basamak sola kaydırılıp SP yazmacının değeri ile toplanmasıyla bulunur. Örneğin SS yazmacı yığıtın bellekteki konumunu 0400H olarak, SP yazmacı ise yığıt bölütünün içinde yığıtın başının göreceli konumunu 0000H olarak gösteriyorsa 0400:0000 adresine erişmek için gereken gerçek bellek adresi aşağıdaki gibi bulunur:

0400H x 10H = 04000H
04000H + 0000H = 04000H

Yığıt üzerinde herhangi bir işlem yapmadan önce MOV komutu ile SS ve SP yazmaçlarına ilk değerlerinin verilmesinde yarar vardır. Yığıta bir veri kaydetmek için PUSH komutu, yığıtta kayıtlı olan verinin yığıttan çekilmesi içinse POP komutu kullanılır. SP yazmacının değeri her PUSH komutundan sonra 1 azalır ve her POP komutundan sonra 1 artar. SP yazmacının değerindeki değişim biçimi, işlemcinin işleyeceği bir sonraki komutun yerini gösteren ve işlenen her komuttan sonra 1 artırılan IP yazmacının değerinin değişme yönteminin tam tersidir.

Örnek: Aşağıdaki örnek yığıt komutlarının nasıl çalıştığını göstermektedir:

```
1  MOV AX, 0200H
2  MOV SS, AX
3  MOV SP, 1238H
4  MOV AX, 0AFFH
5  MOV BH, 11H
6  MOV BL, 12H
7  MOV CX, 01H
8  PUSH AX
9  PUSH BX
10 POP AX
11 PUSH CX
12 POP BX
13 POP CX
```

Yukarıdaki örnek programın birinci satırında AX yazmacına 0200H verisi yazılmakta, ikinci satırda ise bu saklanan veri SS yazmacına yüklenmektedir. SS yazmacına doğrudan veri aktaran bir MOV işlem kodu bulunmaması nedeniyle SS yazmacına yapılan yüklemeler her zaman örnekteki gibi başka bir yazmaç üzerinden dolaylı olarak yapılır. SP yazmacına ise doğrudan veri aktarılabilir. Programın üçüncü satırında SP'ye 1238H verisi yüklenmektedir. Bu satırın ardından gelen 4, 5, 6, ve 7inci satırlarda yazmaçlara AX, BX ve CX yazmaçlarının değerleri MOV komutları aracılığıyla belirlenmektedir.

8inci satırdaki PUSH AX komutu ile AX yazmacının değeri 0200:1238 ve 0200:1236 adresleri arasına yazılır. AX yazmacının boyutunun 2 bayt olması nedeniyle bu saklama işleminin ardından SP yazmacının değeri 2 azalır. Ardından 9uncu satırdaki PUSH BX komutu ile BX yazmacının iki bayt uzunluğundaki değeri 0200:1236 ve 0200:1234 adreslerinin arasına yazılır. 10. satırdaki POP AX komutu SP'nin gösterdiği adresten başlayarak iki bayt okur ve AX yazmacına yazar. Böylece AX yazmacının değeri 1112H olarak belirlenmiş olur. Sonraki satırdaki PUSH CX komutu ile 0200:1236 ve 0200:1234 adreslerinin arasına CX'in değeri olan 0001H yazılır. 12inci satırdaki POP BX komutu ise SP yazmacı ile gösterilen yığıtın tepesinden iki baytlık bir veri çeker ve bu veriyi BX yazmacına yazar. Bu işlemin ardından 0200:1236 adresi ile 0200:1234 adresi arasında kalan verileri BX'e aktarılırken SP imleci de iki artırılarak yığıtın yeni tepe noktası olan 0200:1236 adresini göstermesi sağlanır. 13ücnü satırdaki POP CX komutu CX yazmacına 0200:1236 ve 0200:1238 adreslerinin

arasındaki adreslerde bulunan verileri CX yazmacına atar. Örnek programdaki PUSH ve POP işlemlerinin bellek üzerinde yaptığı değişiklikler aşağıdaki şekilde gösterilmektedir.

0200:1232	
0200:1233	
0200:1234	
0200:1235	
0200:1236	
0200:1237	
0200:1238	
0200:1239	
0200:1249	

1. Başlangıç durumu

0200:1232	
0200:1233	
0200:1234	
0200:1235	
0200:1236	**FF**
0200:1237	**0A**
0200:1238	
0200:1239	
0200:1249	

2. PUSH AX komutundan sonra

0200:1232	
0200:1233	
0200:1234	**12**
0200:1235	**11**
0200:1236	**FF**
0200:1237	**0A**
0200:1238	
0200:1239	
0200:1249	

3. PUSH BX komutundan sonra

0200:1232	
0200:1233	
0200:1234	
0200:1235	
0200:1236	**FF**
0200:1237	**0A**
0200:1238	
0200:1239	
0200:1249	

4. POP AX komutundan sonra

0200:1232	
0200:1233	
0200:1234	**01**
0200:1235	**00**
0200:1236	**FF**
0200:1237	**0A**
0200:1238	
0200:1239	
0200:1249	

5. PUSH CX komutundan sonra

0200:1232	
0200:1233	
0200:1234	
0200:1235	
0200:1236	**FF**
0200:1237	**0A**
0200:1238	
0200:1239	
0200:1249	

6. POP BX komutundan sonra

0200:1232	
0200:1233	
0200:1234	
0200:1235	
0200:1236	
0200:1237	
0200:1238	
0200:1239	
0200:1249	

7. POP CX komutundan sonra

Aşağıda sırasıyla AX, BX ve CX yazmaçlarının programdan önceki ve sonraki durumları gösterilmiştir.

	Önce	**Sonra**
AX	0AFF	1112
BX	1112	0001
CX	0001	0AFF

Örnekte de görüldüğü gibi, PUSH komutu ile yığıta veri yazılmadan hemen önce SP'nin değeri azaltılır. POP komutunda ise veri SP'nin o anda gösterdiği adresten okunur, veri okuma işleminin ardından SP'nin değeri bir artar. Yığıttan veri okunurken ya da yığıta veri yazılırken SP imlecinin değerinde oluşan bu değişiklik sırasını programcı aklında bulundurmalıdır.

Tamamı 16 bit (2 bayt) uzunluğunda olan yazmaçlar PUSH ve POP komutlarıyla birlikte kullanılırken bu yazmaçların parçaları ayrı ayrı 8 bitlik yazmaçlar olarak yığıttan okunamaz ya da yığıta geri yazılamaz. Bu yazmaçlarla işlem yapılırken 16 bitlik yazmacın tamamı kullanılmalıdır. Örneğin POP AL ya da PUSH BH gibi komutlar geçersizdir. Bu komutların yerine POP AX ya da PUSH BX kullanılmalıdır.

4.2.4. Ek Bölüt

Programlarda çok sık kullanılmayan ek bölütün bellekteki yeri ES (extra segment – ek bölüt) yazmacı ile tutulur. Çoğunlukla dizgi işlemleri için kullanılmaktadır. İleriki bölümlerde dizgi işlemleri ayrıntılı olarak açıklanırken bu konuya geri dönülecek ve ek bölüt daha ayrıntılı açıklanacaktır.

4.3. Bölütlerin Örtüşmesi

Bazı nedenlerden ötürü bölütlerin bellekte kapsadığı adres aralıkları örtüşebilir. Örneğin COM uzantılı dosyaların boyutları 64 KB'ı geçmeyecek biçimde tasarlanmak zorunda olduğu için bu dosyalarda bölütler birbirleriyle örtüşür. Programcı yazdığı programda bölütlere ayrı ayrı adresler vermek yerine tek bir bölüt adresi kullanarak hem kodunu hem de verilerini aynı bölüte yazabilir. Böylece bölüt içinde program kodundan önce, program kodunun arasında ya da program kodundan sonra programın kullandığı veriler konumlandırılabilir. Bu tür bir kullanım bölütlerin örtüşmesine örnektir.

4.4. Çevirici, Bağlayıcı, Konumlandırıcı ve Hata Ayıklayıcı

Çevirici diliyle yazılmış bir programın işlemcinin anlayabileceği biçimi alması için bir takım işlemlerden geçirilmesi gerekir. Program önce bir çeviriciyle (assembler) makine diline çevrilir, ardından bir bağlayıcı (linker) ile oluşturulan nesneler birbirine bağlanır ardından da bir konumlandırıcı (locator) kullanılarak bellek konumları ayarlanır. Bu kitap boyunca verilen örneklerde FASM çeviricisinin kodlama tarzı kullanılmıştır. Yazılan örnekler başlangıç aşamasında Windows ile birlikte gelen DEBUG.EXE programı

kullanılarak denenebilir. Programların çevrilmesi için Intel ASM86 çeviricisi, bağlanması için Intel LINK86 bağlayıcısı ve konumlandırılması için Intel LOC86 konumlandırıcısı kullanılabilir. Bu programların, kitaptaki örneklerin denenmesi için gereken komutları ve bu komutlarla birlikte kullanılan terimler kısaca açıklanacaktır. Örnek programların doğrudan 8086 uyumlu bir işlemci üzerinde çalıştırılabilmesi için ise üretilen kodların Intel'in kullandığı 16lık tabandaki sayı düzenine çevrilmesi gerekir. Bu amaçla Intel Hex biçim dönüştürücüsü olarak OH86 programı kullanılabilir. Bütün bu programlar hakkında daha ayrıntılı bilgi Ek B bölümünde sunulmuştur. Ek B bölümünde ayrıca piyasada günümüzde çok kullanılan NASM, MASM gibi çeviriciler hakkında bilgiler bulunmaktadır. Yazılan programların Linux işletim sistemi kullanan bilgisayarlarda nasıl çevrileceği, bağlanacağı, konumlandırılacağı ve deneneceği Ek C bölümünde anlatılmaktadır.

Yazılan çevirici dil kodlarının herhangi bir 8086 tabanlı eğitim düzeneği ya da 8086 deneme kartı üzerinde denenebilmesi için oluşturulan .hex uzantılı dosyaların bir aracı yazılım aracılığıyla donanıma yüklenmesi gerekir. Üzerinde x86 işlemcisi bulunan bir eğitim düzeneğinin yayın olarak bulunmadığı düşünülerek bu kitapta verilen örnekler DOS işletim sisteminde COM uzantılı çalıştırılabilir dosyalara dönüştürülecektir. Windows işletim sisteminin sunduğu DEBUG programı bu COM dosyalarını işleyerek kullanıcıya yazmaç değerleri gibi bilgileri vermektedir. Çeviri dilinde yazılan kodlar FASM programı kullanılarak .com uzantılı dosyalara dönüştürülebilir. Bunun yanında 10un bölümdeki örneklerde .bin (ikilik düzende saf çıktı) uzantılı çıktı almak için FASM programı kullanılacaktır.

4.4.1. Çevirici

Intel firmasının sağladığı ASM86 çeviricisi asm uzantılı dosyaları .obj türünde dosyalara çevirmek için kullanılır. OBJ dosyası yeniden konumlandırılabilen nesne kodunu (relocatable object code) içerir ve bağlayıcı programı tarafından kullanılır. Çevirici dil kullanılarak yazılmış deneme1.asm isimli bir program aşağıdaki satır kullanılarak OBJ dosyasına dönüştürülür:

```
Asm86 deneme1.asm m1 db ep
```

Bu satırdaki m1 terimi çevrilen kodun 80186 komut kümesini kullanacağını belirtmekte, db ve ep terimleri ise çevrilen kodun içinde herhangi bir hata bulunursa bu hatanın ekranda gösterilmesini sağlamaktadır.

4.4.2. Bağlayıcı

Intel firmasının sağladığı LINK86 bağlayıcısı çevirici tarafından oluşturulmuş olan bir ya da birden fazla OBJ dosyasını alır ve bu dosyaların birbirleriyle olan bağlantılarını sağlayarak .lnk uzantılı bir dosya yaratır. Yukarıdaki örnekte ASM86 çeviricisi ile çevrilen dosya aşağıdaki gibi bağlanır:

```
Link86 deneme1.obj to deneme1.lnk
```

Bu komut satırı çalıştırıldıktan sonra deneme1.obj dosyası deneme1.lnk dosyasına dönüştürülmüş olur.

4.4.3. Konumlandırıcı

Intel firmasının sağladığı LOC86 konumlandırıcısı bağlayıcı tarafından oluşturulmuş olan yeniden konumlandırabilen nesne kodu dosyasını alıp konumlandırarak yalın nesne koduna dönüştürür. Daha önce bağlayıcı tarafından bağlanan deneme1.lnk dosyası aşağıdaki gibi konumlandırılır:

```
Loc86 deneme1.lnk noic
```

Bu komut satırı çalıştırıldıktan sonra çıktı olarak deneme1 adlı uzantısız bir dosya oluşacaktır. Noic (No initial conditions) terimi konumlandırıcıya olağan şartlarda programın başına eklenen başlangıç koşullarının yaratılmayacağını söyler.

4.4.4. Intel Hex Format Çeviricisi

Konumlandırıcının oluşturduğu 0 ve 1lerden oluşan ikilik düzendeki dosya (binary file) işlemcinin üzerinde doğrudan çalıştırılamaz. İşlemciye anlayabileceği 16lık taban kodu olan Intel hex biçiminde bir dosya yüklenmelidir. Yaratılan ikilik düzendeki dosyanın IHEX biçimine dönüştürülmesi için Intel firmasının sağladığı OH86 yazılımı kullanılabilir. Bu dönüşümü sağlamak içi yalnızca dosya adının girilmesi yeterlidir:

```
Oh86 deneme1
```

Yukarıdaki komut satırı çalıştırıldıktan sonra çıktı olarak deneme1.hex adlı IHEX biçiminde bir dosya elde edilir. deneme1.hex dosyası bu doğrudan işlemciye yüklenebilir.

Intel Hex Biçimi

Intel Hex Kayıt Biçimi'nin oluşturulma nedeni programların kolay taşınabilmesinin sağlanması, dosya aktarımının kolaylaştırılması ve programların insanlar tarafından ikilik düzendeki dosyalara göre daha rahat okunabilen ASCII dosyası biçimine dönüştürülmesidir. Her hex kaydı 6 tane bölümden oluşur.

Bölüm	Karakter Sayısı	Tanım
1. Başlangıç Kodu	1	":" karakteri
2. Bayt Sayısı	2	Veri bölümündeki bayt sayısı
3. Adres	4	Kaydın belleğin neresine yükleneceğini belirten 2 baytlık adres
4. Tip	2	Kayıt tipi
5. Veri	0 – 2*n*	0 ile n bayt arasında kod ya da veri. *n*'nin değeri genellikle 32 ya da daha azdır.
6. Sağlama	2	Sağlama baytı

Kayıt tipi olarak 6 tane seçenek vardır:

00 - Veri kaydı
01 - Kayıt dosyası sonu
02 - Ek bölüt adres kaydı
03 - Başlangıç bölütü adres kaydı
04 - Ek doğrusal adres kaydı
05 - Doğrusal başlangıç adres kaydı

Sağlama baytı hesaplanırken 2. 3. 4. ve 5. bölümdeki baytlar toplanır. Toplamdaki en az anlamlı bayt alınır, terslenir ve 1 eklenir.

Aşağıdaki sadeleştirilmiş assembly dosyası .hex dosyasına çevrilecektir.

```
MOV AX, 12FFH
INT 03H
```

Birinci satırın makine dilindeki değeri B8 FF 12. İkinci satırın makine dilindeki değeri CC ye karşılık gelmektedir. Assembly kodu 0000:2000 adresine yüklenecek biçimde ayarlanıp .hex dosyasına çevrilince aşağıdaki hex kayıt dosyası oluşur.

Hex kayıt dosyasının görünümü:

```
:020000020200FA
:04000000B8FF12CC67
:00000001FF
```

: 02 0000 02 0200 FA

Burada birinci karakter başlangıç karakteridir. 02 bayt sayısıdır ve 2 baytlık verinin taşındığını belirtir. 0000 bellek adresini (göreceli konum adresi) göstermektedir. Takip eden 02 bunun ek bölüt adres kaydı olduğunu gösterir. Takip eden 2 baytlık 0200 veri bölüt adresidir. FA sağlama baytıdır. Bu verilerden anlaşıldığı gibi kullanılan bölüt adresi 0200 ve göreceli konum adresi 0000'dır. Bu adres 0000:2000 ile aynı gerçek adresi göstermektedir.

: 04 0000 00 B8FF12CC 67

Birinci karakter başlangıç karakteridir. 04 bayt sayısıdır ve 4 baytlık verinin taşındığını belirtir. 0000 bellek adresinin göstermektedir. Takip eden 00 bunun bir veri kaydı olduğunu gösterir. Takip eden 4 baytlık veri B8FF12CC kullanılacak makine kodudur. 67 sağlama baytıdır. Dikkat edilirse dönüştürülen ana çevirici dil kodunun makine dilindeki karşılığı olan B8 FF 12 CC kodunun burada olduğu görülür.

: 00 0000 01 FF

Birinci karakter başlangıç karakteridir. 00 bayt sayısını gösterir. Takip eden 0000 bellekteki adresi belirtir. 01 kayıt dosyası sonu kaydı olduğunu gösterir. FF sağlama verisidir.

4.4.5. Hata Ayıklayıcı

Hata ayıklayıcı olarak Windows ile birlikte gelen debug.exe programını DOS işletim sisteminin ilk yıllarında 8086 işlemcisi için yazılmış ve Intel x86 işlemci ailesi geriye dönük uyumlu olduğundan günümüze kadar fazla değişiklik geçirmeden gelmiştir. Bu hata ayıklayıcının DOS için yazılmıştır ve DOS için yazılan çalıştırılabilir programların COM ya da EXE dosyası olması gerekir. COM dosyalarında EXE dosyalarında olduğu gibi bir başlık kısmı olmadığından oluşturması daha bu dosyaların oluşturulması daha kolaydır. .com uzantılı bir dosya aşağıda gösterilen biçimde DEBUG programına terim olarak verilebilir:

```
debug C:\deneme1.com
```

Program yukarıda gösterilen biçimde yüklendikten sonra debug programının içinden aşağıdaki komut ile çağırılabilir:

```
g =0100
```

g (Go), debug programına 0100 göreceli konum adresine gitmesini söyler ve program belirtilen adresten başlatılır.

```
d 0100
```

d komutu ise debug programına 0100 göreceli konum adresinden başlayarak bellekteki verilerin gösterilmesini söyler.

4.4.6. FASM

FASM programı hem bir çevirici hem de bir bağlayıcıdır. Eğer programla birlikte parametreler kullanılmazsa çıktı olarak .com dosyası ya da .bin dosyası verir (parametre kullanılması durumunda başka türlerde çıktı verebilmektedir; ancak bu kitap boyunca bu iki çıktı biçimi kullanılacaktır). DOS işletim sisteminde COM dosyaları 0100H'den başladığı için başında ORG 0100H komutu bulunan bütün programlar çevrildiğinde .com dosyasına dönüşür. Aksi halde çevrilme işlemi sonunda bir BIN dosyası ortaya çıkar. .com uzantılı dosyalar DEBUG programı ile denenebilir. Çevirici dil ile programlamanın temelleri tam olarak anlatılıp bütün komut kümesi açıklanana kadar verilecek örnekler .com çıktısı alacak biçimde yazılacak ve DEBUG programı ile denenebilecektir.

Unutulmaması gereken konu çeviricilerin kullandıkları asm kodunun söz diziminin farklı olabileceğidir. Örneğin ASM86 ile FASM arasında kod yazılırken bazı küçük farklar vardır. ASM86 Standard Intel sözdizimi (syntax) kullanırken FASM daha yalın bir sözdizimi kullanmaktadır. Aynı durum diğer çeviriciler için de geçerlidir.

4.5. 8086 Komut Kümesi

Bu bölümde 8086 komut kümesi tanıtılacak ve her komut için kısa örnekler verilecektir. İlerleyen bölümlerde burada tanıtılan çevirici dil komutlarının kullanımları hakkında daha ayrıntılı bilgiler verilecektir. Ancak komutların ayrıntılı açıklamasın geçmeden önce bir kaç değişik örnek ile çevirici kodu bir çevirici ile çevrilirken gerçekte neler olduğu gösterilecektir.

4.5.1. Makine Diline Çevrim

Makine dili komutları 4 farklı bölümden oluşabilir. Bu bölümler aşağıdaki tabloda sıralanmıştır.

Bölümün Adı	Uzunluğu (Bayt)
İşlem Kodu Baytı	1 – 2
Adresleme Baytı	0 – 1
Uzaklık Baytı	0 – 1
Anlık Veri Baytı	0 – 2

Bir komut en az bir işlem kodu baytından oluşur. Örneğin NOP komutunda yalnızca işlem kodu baytı bulunur. Bazı komutlar ise bu dört bölümün dördünü de bulundurur. Örneğin MOV WORD PTR [BX + 0200H], 2000H assembly kodu 4 bölümden oluşur ve toplamda 6 baytlık yer kaplar.

Anlık veri baytları ve uzaklık baytlarında küçüğü başta kuralının kullanıldığı unutulmamalıdır.

1	0	0	0	1	0	1	1
İşlem Kodu						**D**	**W**

Yukarıdaki işlem kodu baytında işlem kodu 100010'dır. 100010 işlem kodu MOV komutuna karşılık gelmektedir. D yön bitidir ve 1 olduğu durumunda yazmaç ya da bellekten (R/M'den) yazmaca (REG'e) bir

veri aktarımı olur. W verinin bayt mı yoksa sözcük mü olduğunu belirtir. W örnekteki gibi 1 olduğunda verinin sözcük olduğu anlaşılır. Böylece yukarıdaki işlem koduyla bellek ya da yazmaçtan bir yazmaca bir sözcüklük bir veri aktarılacağı anlaşılır.

1	1	1	0	1	1	0	0
MOD		**REG**			**R/M**		

Yukarıda gösterilen adres baytı işlem kodundan sonra gelir. Adres baytının 3 ayrı bölümü vardır: MOD, REG ve R/M.

Aşağıda makine diline çevrimin farklı özellikleri kısaca açıklanmıştır. Intel 8086 komut kümesine geçildiğinde her komut ayrı ayrı incelenecektir.

4.5.1.1. Yazmaç ve R/M Bellek Adreslemeleri

Yazmaç ve R/M bellek adreslemeleri aşağıdaki tablo göz önünde bulundurularak yapılır.

MOD = 00

Kod	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Kod	Adresleme Kipi
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]
111	DS:[BX]

MOD = 01

Kod	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Kod	Adresleme Kipi
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]
111	DS:[BX]

MOD = 10

Kod	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Kod	Adresleme Kipi
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]
111	DS:[BX]

MOD = 11

Kod	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Kod	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

MOD'un 00 olması uzaklık baytının kullanılmayacağını belirtir. MOD'un 01 olması 1 baytlık bir uzaklık verisinin olacağını bildirir. MOD'un 10 olması 2 baytlık (ilk bayt alt uzaklık baytı, ikinci bayt üst uzaklık baytı olmak üzere) bir uzaklık verisinin olacağını belirtir. MOD'un 11 olması ise R/M'nin bir yazmaca eşit olduğunu belirtir.

4.5.1.2. Özel Adresleme

R/M'nin SS:[BP] adresleme kipine geçmesi özel bir durumdur ve makine koduna çevrim yapılırken normal işlemlerden farklı bir işlem yapılmaktadır. Bu özel durumda sabit bir bellek bölgesine erişim gerekir. MOV [1000H], AL assembly kodu bu özel duruma bir örnektir. Bu komut makine koduna aşağıdaki gibi çevrilir:

1	0	0	0	1	0	0	0
İşlem Kodu						**D**	**W**

0	0	0	0	0	1	1	0
MOD		**REG**			**R/M**		

0	0	0	0	0	0	0	0

0	0	0	1	0	0	0	0

100010 işlem kodu komutun bir MOV komutu olduğunu gösterir. D'nin 0 olması yazmaçtan (REG) yazmaca ya da belleğe (R/M) veri aktarılacağını belirtir. W'nun 0 olması 1 baytın aktarılacağı anlamına gelir. REG 000 ve W 0 olduğundan AL yazmacından R/M ye veri aktarılacaktır. R/M değeri 110 olduğundan tablodan SS:BP adreslemesi kullanıldığı görülür. [BP] adresleme kipi uzaklık baytı olmadan kullanılamaz. Ancak MOD'un 00 olması uzaklık baytı alınmayacağının bir göstergesidir. Buna karşın [BP] adreslemelerinde uzaklık verisi kullanılır ve yukarıdaki kodun uzaklık verisinde 1000H bulunmaktadır. Bütün bu parçalar bir araya getirilerek MOV [1000H], AL komutu elde edilir.

MOD değeri 00 olsa bile [BP] adreslemesi kullanılırken [BP] [BP+00] olarak değiştirilir ve 1 tane fazladan uzaklık baytı kullanılır. Eğer yukarıdaki makine kodunun bu özelliği göz ardı ederek çeviri dil koduna geri çevrilirse MOD'un değerinin 00 olması uzaklık baytının bulunmadığı anlamına geleceğinden ilk iki bayt MOV [BP], AL koduna denk olur. Son iki bayt ise ADD DL, AL koduna denk olacağından tersine çevrim yanlış bir sonuç çıkaracaktır.

Eğer özel durum kullanılmak istenmez ise yukarıdaki işlem 2 ayrı komutla yerine getirebilir.

```
MOV SI, 1000H
MOV [SI], AL
```

Böylece [BP+SI] adreslemesi kullanılarak özel durumun getirdiği karmaşıklık ortadan kaldırılabilir.

4.5.1.3. Anlık Veri Adreslemeleri

Anlık veri adreslemesine örnek olarak MOV WORD PTR [BX + 2000H],4432H çevirici dil komutu verilebilir. 4432 verisi doğrudan programdan gelmektedir ve komut bu veriyi DS x 10H + 2000H + BX bellek adresine (gerçek bellek adresine) yazacaktır. Bu komuttan türetilen makine kodu aşağıdaki gibidir:

1	1	0	0	0	1	1	1
İşlem Kodu							**W**

1	0	0	0	0	1	1	1
MOD					**R/M**		

0	0	0	0	0	0	0	0
Uzaklık Baytı (Alt)							

0	0	1	0	0	0	0	0
Uzaklık Baytı (Üst)							

0	0	1	1	0	0	1	0
Anlık Veri Baytı (Alt)							

0	1	0	0	0	1	0	0
Anlık Veri Baytı (Üst)							

Dikkat edilirse işlem kodu baytında yön belirten D'nin bulunmadığı görünür. İşlem kodu 7 bitliktir (MOV'un anlık veri kullanan özel bir şeklidir, REG kullanılmaz, en sonda anlık veri baytları bulunur) ve adresleme baytında REG de bulunmamaktadır. R/M değeri 111 olduğu için DS:[BX] adreslemesi kullanılacaktır. MOD değeri 10 olduğundan 2 baytlık bir uzaklık verisi bulunmaktadır. Ardından 2 baytlık anlık veri baytı gelmektedir.

WORD PTR, BYTE PTR Kullanımı

Eğer çevirici dil kodunda işlenenin boyutu belli ise BYTE PTR ya da WORD PTR gibi eklerin kullanılmasına gerek yoktur. Örneğin aşağıdaki kodlarda kullanılan işlenenlerin boyutları belli olduğundan BYTE PTR ya da WORD PTR'ın kullanımı gereksizdir.

```
MOV [BX], AL
MOV [BX], CH
```

Ancak aşağıdaki komutta işlenen boyutu tam olarak belli değildir.

```
MOV [BX], 02H
```

Yukarıdaki kodu çevirici ile çevirdiğinizde uygun komutun belirlenmesi için yeterince bilgi olmadığına dair bir hata iletisi alınır. Bu yüzden yukarıdaki komutun boyutunun belirlenmesi gerekir. Aktarılacak veri 2 bay uzunluğunda ise WORD PTR ile 0002H verisi, tek bayt uzunluğunda ise BYTE PTR ile 02H verisi [BX] adresine yüklenecektir.

```
MOV WORD PTR [BX], 02H
MOV BYTE PTR [BX], 02H
```

Not: FASM kullanırken WORD PTR yerine yalnızca WORD, BYTE PTR yerine ise de yalnızca BYTE yazmak yeterlidir.

4.5.1.4. Bölüt-Yazmaç Komutları

Bölüt yazmaçlarının değerleri PUSH, POP ve MOV gibi komutlarla değiştirildiği zaman REG'in değerleri farklı anlamlara gelir.

Kod	Bölüt Yazmacı
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

CS yazmacının değeri yalnızca özel durumlarda değişebileceği için bu yazmaç POP ve MOV komutu ile birlikte kullanılamaz. Örneğin MOV CS, AX kural dışı bir komuttur. CS yazmacının değeri CALL FAR komutu ile değiştirilebilir.

CS yazmacının üzerine veri yazılamamasına karşın CS yazmacının değeri başka yazmaçlara aktarılabilir. Örneğin MOV AX,CS komutu kurallara uygun bir komuttur ve makine dilinde bir karşılığı vardır. MOV komutunun bu durumda kullanılacak biçiminin işlem kodu 10001100'dır. R/M bir yazmaç olduğundan MOD değeri 11 olmalıdır. REG'in CS'yi simgelemesi gerektiğinden değeri 001 olmalıdır. R/M'nin AX için değeri 000'dır.

1	0	0	0	1	1	0	0

İşlem Kodu							
1	1	0	0	1	0	0	0
MOD			REG			R/M	

4.5.2. Veri Taşıma Komutları

4.5.2.1. MOV

MOV (**mov**e – taşı) komutu bir veriyi bir yerden alıp başka bir yere taşır. Aşağıda yapısı gösterilen MOV komutu *işlenen2* verisini *işlenen1*'e kopyalar. Örneğin bellekte kayıtlı olan bir baytın bir yazmaca aktarılması için MOV komutu kullanılır.

MOV *işlenen1*, *işlenen2*

Aşağıdaki tabloda MOV komutunun değişik türevleri gösterilmiştir.

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M* ↔ REG*	100010DW	MOD REG R/M	-	-
Anlık Veri → R/M	1100011W	MOD 000 R/M	Veri	W=1 ise Veri
Anlık Veri → REG	1011WREG	Veri	W=1 ise Veri	-
Bellek → Birikeç	1010000W	Adres (alt)	Adres (üst)	-
Birikeç → Bellek	1010001W	Adres (alt)	Adres (üst)	-
R/M → Bölüt Yazmacı	10001110	MOD 0 REG R/M	-	-
Bölüt Yazmacı → R/M	10001100	MOD 0 REG R/M	-	-

*REG = yazmaç R/M = yazmaç ya da bellek

Örnek: Aşağıdaki programda MOV komutunun yukarıdaki türevlerine örnekler verilmiştir.

```
1   ORG 0100H            ;COM dosyaları için göreceli adres
2   MOV AX, BX           ;Yazmaç → Yazmaç
3   MOV [SI], BX         ;Yazmaç → Bellek
4   MOV BX, [SI]         ;Bellek → Yazmaç
5   MOV BX, 1000H        ;Anlık Veri → REG
6   MOV WORD [BX], CX    ;Anlık Veri → Bellek
7   MOV AX, [1000H]      ;Bellek → Birikeç
8   MOV [SI], AX         ;Birikeç → Bellek
9   MOV DS, BX           ;Yazmaç → Bölüt Yazmacı
10  MOV DS, [SI]         ;Bellek → Bölüt Yazmacı
11  MOV BX, DS           ;Kesim Yazmacı → Yazmaç
12  MOV [SI], DS         ;Kesim Yazmacı → Bellek
13  INT 03H              ;Programın durdurulmasını sağlar
```

MOV komutu kullanılarak aşağıdaki 3 işlem yapılamaz:

- MOV komutu ile CS ve IP yazmaçlarının değerleri değiştirilemez.
- MOV komutu ile bir bölüt yazmacından diğer bir bölüt yazmacına veri aktarılamaz. Aracı bir genel amaçlı yazmaç kullanılmalıdır.
- MOV komutu ile bir bölüt yazmacına doğrudan anlık veri kaydedilemez. Aracı bir genel amaçlı yazmaç kullanılmalıdır.

4.5.2.2. PUSH

PUSH komutu yığıta bir verinin yazılması için kullanılır. PUSH komutunun ardından SP 2 azalır ve bellekteki SS:[SP] adresine *işlenen1* kaydedilir.

PUSH *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M	11111111	MOD 110 R/M	-	-
Yazmaç	01010REG	-	-	-
Bölüt Yazmacı	000REG110	-	-	-

80186 işlemcisi ve sonraki nesil x86 işlemciler PUSH ile bir anlık veriyi de yığıta yazabilir.

4.5.2.3. POP

POP komutu ile yığıtın tepesinden bir veri alınır. POP komutunun ardından SP 2 artar ve *işlenen1*'e SS:[SP] bellek adresindeki veri kaydedilir.

POP *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M	10001111	MOD 000 R/M	-	-
Yazmaç	01011REG	-	-	-
Bölüt Yazmacı	000REG111	-	-	-

Örnek: Aşağıda PUSH ve POP komutlarının yazmaçlarla birlikte nasıl kullanıldığını gösteren bir program gösterilmiştir.

```
1  MOV AX, 1000H
2  MOV BX, 2000H
3  MOV CX, 3000H
4  MOV DX, 4000H
5  PUSH AX
6  PUSH BX
7  POP AX
8  PUSH DX
9  PUSH CX
10 POP BX
11 POP CX
12 POP DX
13 INT 03H
```

Yukarıdaki kod FASM ile çevrilip DEBUG programı ile çalıştırıldığında yazmaç değerleri aşağıda gösterildiği biçimde değişir.

Yazmaç	Önce	Sonra
AX	1000H	2000H
BX	2000H	3000H
CX	3000H	1000H
DX	4000H	4000H

Örnek: Aşağıdaki program bellekteki 0100H ve 0102H göreceli konum adreslerindeki sözcükleri yığıta atan, ardından POP komutunu kullanarak bu verileri AX ve BX yazmaçlarına aktarmaktadır.

```
1  ORG 0100H
2  PUSH WORD [0100H]
3  PUSH WORD [0102H]
4  POP AX
5  POP BX
6  INT 03H
```

Yukarıdaki program çalıştıktan sonra 0102H göreceli konum adresinden aktarılan sözcük AX yazmacına, 0100H göreceli konum adresinden aktarılan veri ise BX yazmacına yazılacaktır.

Örnek: Aşağıdaki program PUSH ve POP komutlarını kullanarak DS bölüt yazmacının değerini BX yazmacına ve AX yazmacının değerini de DS bölüt yazmacına aktarır.

```
1  ORG 0100H
2  MOV AX, 1234H
3  PUSH DS
4  POP BX
5  PUSH AX
6  POP DS
7  INT 03H
```

4.5.2.4. XCHG

XCHG (e**xch**an**g**e – değiştirme) komutu işlenenlerin verilerini birbiriyle değiştirir. Komutun yürütülmesinin ardından *işlenen2*'nin değeri *işlenen1*'e, *işlenen1*'in değeri de *işlenen2*'ye aktarılmış olur.

XCHG *işlenen1,işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ↔ Yazmaç	1000011W	MOD REG R/M	-	-
Yazmaç ↔ Birikeç	10010REG	-	-	-

Örnek: Aşağıdaki program XCHG komutunun kullanımına bir örnektir.

```
1  ORG 0100H
2  MOV AX, 1234H
3  MOV BX, 4321H
4  MOV CX, 3214H
5  MOV DX, 1243H
6  XCHG BX, CX          ; Yazmaç ↔ Yazmaç
7  XCHG BX, [0120H]     ; Yazmaç ↔ Bellek
8  XCHG AX, DX          ; Yazmaç ↔ Birikeç
9  INT 03H
```

4.5.2.5. IN

IN komutu seçilen giriş kapısında o anda bulunan veriyi AL ya da AX yazmacına aktarır. Eğer kapı numarası 255'den daha büyük ise kapı numarasını belirtmek için DX yazmacı kullanılmalıdır.

IN *AL ya da AX, kapı numarası*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
255'den daha küçük kapı numarası	1110010W	Kapı numarası	-	-
255'den büyük kapı numarası	1110110W	-	-	-

Örnek: Aşağıdaki program FFE4H numaralı ve 04H numaralı kapılardan IN komutuyla nasıl veri okunacağını göstermektedir.

```
1  ORG 0100H
2  MOV DX,0FFE4H
3  IN AL, DX          ;FFE4H numaralı kapıdan verileri oku
4  IN AX, 04H         ;04H numaralı kapıdan verileri oku
5  INT 03H
```

4.5.2.6. OUT

OUT komutu AL ya da AX yazmacındaki veriyi seçilen kapıya iletir. Eğer kapı numarası 255'den daha büyük ise kapı numarasını belirtmek için DX yazmacı kullanılmalıdır.

OUT *kapı numarası*, *AL ya da AX*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
255'den daha küçük kapı numarası	1110011W	Kapı numarası	-	-
255'den büyük kapı numarası	1110111W	-	-	-

Örnek: Aşağıdaki program FFE4H numaralı ve 04H numaralı kapılara OUT komutuyla nasıl veri gönderileceğini göstermektedir.

```
1  ORG 0100H
2  MOV DX, 0FFE4H
3  MOV AX, 1234H
4  OUT DX, AX         ;FFE4H numaralı kapıya AX'deki değeri yaz
5  OUT 04H, AL        ;04H numaralı kapıya AL'deki değeri yaz
6  INT 03H
```

4.5.2.7. XLAT

XLAT komutu AL yazmacına DS:[BX+AL] bellek adresindeki baytı aktarır. MOV AL, [BX+AL] komutu ile aynı işlemi yapar.

XLAT [BX]

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
AL yazmacına bayt aktar	11010111	-	-	-

Örnek: Aşağıdaki program XLAT komutunun nasıl kullanıldığını göstermektedir.

```
1  ORG 0100H
2  MOV BX, 0105H
3  MOV AL, 04H
4  XLAT [BX]          ;DS:[BX+AL] (DS:0108) adresindeki baytı AL'ye kaydet
5  INT 03H
```

4.5.2.8. XLATB

XLATB komutu AL yazmacına DS:[BX+AL] adresindeki baytı aktarır. XLAT komutunun işlenen almayan kısa şeklidir.

XLATB

Örnek: Aşağıdaki program XLATB komutunun nasıl kullanıldığını göstermektedir.

```
1  ORG 0100H
2  MOV BX, 0104H
3  MOV AL, 03H
4  XLATB              ;AL yazmacına DS:0107H adresindeki baytı aktar
5  INT 03H
```

4.5.2.9. LEA

LEA (**l**oad **e**ffective **a**ddress – etkin adresi yükle) komutu etkin adresi belirtilen yazmaca kaydeder. Aşağıda gösterilen biçimde iki işlenenle kullanılır ve *işlenen1*'e *işlenen2*'nin adresi kaydedilir.

LEA *işlenen1*,*işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Yazmaca etkin adresi kaydet	10001101	MOD REG R/M	-	-

Örnek: Aşağıdaki örnek program LEA komutunun nasıl kullanıldığını göstermektedir.

```
1  ORG 0100H
2  MOV BX, 0100H
3  MOV DI, 0002H
4  LEA SI, [BX+DI]    ;SI yazmacına 0102H adresini kaydet
5  INT 03H
```

LEA komutu kullanılırken İkinci işlenenin bellekteki bir veriyi göstermesi gerekir. LEA bu verinin adresini hedef yazmacına kendisi atayacaktır.

4.5.2.10. LDS

LDS (**l**oad **DS** – DS'e yükle) komutu bellekteki çift sözcüğü bir genel amaçlı yazmaca ve DS bölüt yazmacına aktarır. *işlenen1*'e çift sözcüğün ilk parçası olan sözcük, DS bölüt yazmacına da çift sözcüğün ikinci parçası olan sözcük yazılır. *işlenen2* bir bellek imleci olmalıdır.

LDS *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Çift sözcüğü yazmaç ve DS'ye kaydet	11000101	MOD REG R/M	-	-

Örnek: Aşağıdaki örnek program LDS komutunun nasıl kullanıldığını göstermektedir.

```
1  ORG 0100H
```

```
2  MOV SI, [DEG1]     ;SI yazmacına DEG1 in göreceli adresini kaydet
3  LDS BX, [SI]       ;BX yazmacına 0100 adresindeki sözcüğü, DS'ye 0102
4  INT 03H            ;adresindeki sözcüğü kaydet
5  DEG1  dw  0100H    ;DEG1 değişkenine 0100H sözcüğünü kaydet
```

4.5.2.11. LES

LES (**l**oad **ES** – ES'ye yükle) komutu LDS komutunun yaptığı işlemin yaklaşık olarak aynısını yapar. Tek farkı DS yazmacı yerine ES yazmacının değerini değiştirmesidir.

LES *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Çift sözcüğü yazmaç ve ES'ye kaydet	11000100	MOD REG R/M	-	-

Örnek: Aşağıdaki örnek program LES komutunun nasıl kullanıldığını göstermektedir.

```
1  ORG 0100H
2  MOV SI, [DEG1]     ;SI yazmacına DEG1 in göreceli adresini kaydet
3  LES BX, [SI]       ;BX yazmacına 0100 adresindeki sözcüğü, ES'ye 0102
4  INT 03H            ;adresindeki sözcüğü kaydet
5  DEG1  dw  0100H    ;DEG1 değişkenine 0100H sözcüğünü kaydet
```

4.5.2.12. LAHF

LAHF (**l**oad **AH** with **f**lags – AH yazmacına bayrak bitlerini yükle) komutu SF, ZF, AF, PF ve CF bayrak bitlerini sırasıyla AH yazmacının 7, 6, 4, 2 ve 0 numaralı bitlerine kaydeder. Bayrak bitleri bu işlemden etkilenmez.

LAHF

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
AH yazmacına bayrakları kaydet	10011111	-	-	-

Örnek: Aşağıdaki küçük program LAHF komutunun program içinde nasıl kullanıldığını göstermektedir.

```
1  ORG 0100H
2  LAHF
3  INT 03H
```

4.5.2.13. SAHF

SAHF (**s**tore **AH** into **f**lags – AH yazmacını bayrak bitlerine kaydet) komutu AH yazmacının 7, 6, 4, 2 ve 0 bitlerini sırasıyla SF, ZF, AF, PF ve CF bayrak bitlerine kaydeder.

SAHF

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Bayraklara AH yazmacını kaydet	10011110	-	-	-

Örnek: Aşağıdaki SAHF komutunu kullanan küçük program çalıştırıldığında SF, ZF, AF, PF ve CF bayrak bitlerinin tamamı 1 olarak belirlenir.

```
1  ORG 0100H
2  MOV AH, 0FFH
3  SAHF
4  INT 03H
```

4.5.2.14. PUSHF

PUSHF komutu PUSH komutu gibi çalışır. Tek farkı işlenen kullanmaması ve bayrak yazmacının tamamını yığıta yazmasıdır.

PUSHF

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Bayrak yazmacını yığıta kaydet	10011100	-	-	-

Örnek: Aşağıdaki program çalıştırıldığında PUSHF komutu önce bayrak yazmacını yığıta yazar. Ardından bayrak yazmacı yığıttan POP komutu ile BX yazmacına aktarılır.

```
1  ORG 0100H
2  PUSHF          ;Bayrak yazmacını yığıta kaydet
3  POP BX         ;yığıttaki veriyi BX yazmacına kaydet
4  INT 03H
```

4.5.2.15. POPF

POPF komutu POP komutu gibi çalışır. Tek farkı yığıttan aldığı veriyi doğrudan bayrak yazmacına kaydetmesidir. Hedef olarak bayrak yazmacını kullandığından işleneni yoktur.

POPF

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Yığıttaki veriyi bayrak yazmacına kaydet	10011101	-	-	-

Örnek: Aşağıdaki program POPF komutunun nasıl kullanıldığını göstermektedir. Programın sonucunda bayrak yazmacına 1234H değeri yazılır.

```
1  ORG 0100H
2  MOV BX,1234H
3  PUSH BX        ;Yığıta 1234H değerini yaz
4  POPF           ;Yığıttaki veriyi bayrak yazmacına kaydet
5  INT 03H
```

4.5.3. Aritmetik Komutları

4.5.3.1. ADD

ADD (**add** – topla) komutu ile *işlenen1* ve *işlenen2*'nin değerleri toplanır ve sonuç *işlenen1*'e kaydedilir. *işlenen2*'nin değeri değişmez.

ADD *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	000000DW	MOD REG R/M	-	-
Anlık Veri ve R/M	100000SW	MOD 000 R/M	Veri	W=1 ise Veri
Anlık Veri ve Birikeç	0000010W	Veri	W=1 ise Veri	-

Örnek: Aşağıda ADD komutunun değişik kullanım biçimlerini gösteren bir örnek verilmiştir.

```
1  ORG 0100H
2  MOV AL, 03H
3  MOV AH, 04H
4  ADD AL, AH                ;Yazmaç ve Yazmaç
5  ADD [0100], AH            ;Bellek ve Yazmaç
6  ADD AH, [0100]            ;Yazmaç ve Bellek
7  ADD AX, 1234H             ;Birikeç ve Anlık Veri
8  ADD WORD [0102H], 1234H   ;Bellek ve Anlık Veri
9  INT 03H
```

8. satırda [0102H] göreceli adresindeki verinin boyutu belli olmadığı için önüne bu verinin bir sözcük boyutunda olduğunu belirten WORD belirleyicisi yerleştirilmiştir.

4.5.3.2. ADC

ADC (**ad**d with **c**arry – artan ile topla) komutu ile *işlenen1* ve *işlenen2* toplanır ve sonuç *işlenen1*'e kaydedilir. *işlenen2*'nin değeri ADD komutunda olduğu gibi değişmez. ADC'nin ADD komutundan tek farkı yapılan işlemin sonucunda CF bayrak biti 1 olursa *işlenen1*'e 1 eklenmesidir.

ADC *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	000100DW	MOD REG R/M	-	-

Anlık Veri ve R/M	100000SW	MOD 010 R/M	Veri	W=1 ise Veri
Anlık Veri ve Birikeç	0001010W	Veri	W=1 ise Veri	-

Örnek: Aşağıda ADC komutunun nasıl kullanıldığını gösteren bir örnek verilmiştir.

```
1  ORG 0100H
2  MOV BX, 1234H
3  MOV CX, 0FFFFH
4  ADC BX, CX
5  MOV DX, 0FFFFH
6  ADC DX, 0FFFFH
7  ADC AX, 1234H
8  INT 03H
```

6. satırda DX yazmacı ve FFFFH sayısının toplanmasının ardından CF bayrak biti 1 olacaktır ve bu nedenle sonuca 1 eklenecek ve DX yazmacının değeri FFFF olarak belirlenecektir. Eğer 6ıncı satırdaki ADC komutunun yerine ADD komutu kullanılırsa komutun ardından DX yazmacının alacağı değer FFFE'dir.

4.5.3.3. INC

INC (**inc**rease – artır) komutu *işlenen1*'in değerini 1 artırır. İşlem sonucunda elde çıksa da INC komutu CF bayrak bitinde değişikliğe neden olmaz.

INC *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M	1111111W	MOD 000 R/M	-	-
Yazmaç	01000REG	-	-	-

Örnek: Aşağıda INC komutunun nasıl kullanıldığını gösteren bir örnek verilmiştir.

```
1  ORG 0100H
2  MOV AX,0FFFH
3  INC AX
4  INC BYTE [0120H]
5  INT 03H
```

3. satırda AX yazmacının değeri bir artırılarak değeri 1000H yapılmıştır. 4. satırda ise bellekte 0120H göreceli konum adresindeki baytın değeri bir artırılmaktadır.

4.5.3.4. AAA

AAA (**A**SCII **A**djust for **A**DD – ADD komutu için ASCII düzenlemesi) komutu AL yazmacında ikilik düzende saklanan değeri paketlenmemiş bir ondalık değere (BCD - Binary Coded Decimal düzenine) çevirir. Komut çalıştırıldığı sırada, eğer AL yazmacının değeri 9'dan büyükse ya da AF bayrak biti 1 ise, AH yazmacının değeri 1, AL yazmacının değeri 6 artırılır ve AF ile CF bayrak bitleri 1 yapılır. AAA komutu çalıştırıldığı sırada AL yazmacının değeri 9'dan büyük değilse AF ve CF bayrak bitleri sıfırlanır. Her iki durumda da AL yazmacının üst yarım baytı mutlaka sıfır değerini alır. AAA komutu herhangi bir işlenen almaz.

AAA

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
AAA	00110111	-	-	-

Örnek: Aşağıda AAA komutunun nasıl kullanıldığını gösteren bir örnek verilmiştir.

```
1  ORG 0100H
2  MOV AH, 00H
3  MOV AL, 08H
4  AAA
5  MOV BX, AX
6  MOV AH, 00H
7  MOV AL, 0EH
8  AAA
9  MOV CX, AX
```

10	INT 03H

Yukarıdaki örnek programın ikinci ve üçüncü satırlarında AX yazmacının üst yarısına 00H ve alt yarısına 08H değerleri atanmıştır. Bu atama işleminin ardından AAA komutu çağrıldığında AL yazmacının tuttuğu veri olan 08H değeri 9'dan küçük olduğundan AL yazmacında herhangi bir değişiklik gerçekleşmez. Bu nedenle beşinci satırda BX yazmacına aktarılan değer AX yazmacının o anki değeri olan 0008H'dir. Programın altıncı ve yedinci satırlarında ise AX yazmacının değeri 000EH olarak belirlenmiştir. Bu durumda AL yazmacının sahip olduğu değer 0EH (onluk tabanda 14) olduğundan sekizinci satırdaki AAA komutu çalıştırıldığında bu değeri onluk tabana dönüştürür. Bu dönüşümün yapılması için 0EH sayısına 06H eklenir, AL yazmacının üst yarım baytı sıfırlanır ve AH yazmacına 1 eklenir. AAA komutu bu davranışıyla aslında üç ayrı 8086 komutunun yaptığı işin tek başına gerçekleştirir. Aşağıda AAA komutunun AX yazmacı üzerinde yaptığı işlemler, işlemi gerçekleştirebilecek 8086 komutlarıyla birlikte, adım adım gösterilmiştir.

AAA komutundan önce →	**AX**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**
1. ADD AL, 6 →	**AX**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
2. AND AL, 00001111B →	**AX**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
3. INC AH →	**AX**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**

Böylece örnek programın dokuzuncu satırından sonra CX yazmacı AX yazmacının o anki değeri olan 0104H değerini tutar. Dikkat edilirse, AH ve AL yazmaçlarının birlikte onluk tabandaki 14 sayısını gösterdiği görülecektir.

4.5.3.5. DAA

DAA (**D**ecimal **A**djust for **A**DD – ADD komutu için ondalık düzenleme) komutu AL yazmacında saklanan ikilik tabandaki değeri, BCD düzeninde paketlenmiş bir ondalık değere çevirir. Eğer AL yazmacının alt yarım baytının değeri 9'dan büyükse ya da AF bayrak biti 1 ise AL yazmacına 06H eklenir ve AF bayrak bitinin değeri 1 yapılır. Eğer AL yazmacının değeri 9F'den büyükse ya da CF bayrak biti 1 ise AL yazmacına 60H değeri eklenir ve CF bayrak bitinin değeri 1 olarak belirlenir.

DAA

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
DAA	00100111	-	-	-

Örnek: Aşağıda DAA komutunun nasıl kullanıldığını gösteren bir örnek verilmiştir.

```
1  ORG 0100H
2  MOV AL, 3EH
3  DAA
4  INT 03H
```

Yukarıdaki programda AL yazmacının değeri DAA komutundan önce 3EH olarak belirlenmiştir. AL yazmacının alt yarım baytının değeri 9'dan büyük olan E (onluk tabanda 14) sayısı olduğu için AL yazmacına 06H eklenecek ve yazmacın değeri 44H olacaktır.

4.5.3.6. SUB

SUB (**sub**tract – çıkarma) komutu ile *işlenen1*'den *işlenen2*'nin değeri çıkarılır ve sonuç *işlenen1*'e yazılır. Bu komut sonrasında *işlenen2*'nin değeri değişmez.

SUB *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç ya da tersi	001010DW	MOD REG R/M	-	-
R/M'den Anlık Veri	100000SW	MOD 101 R/M	Veri	SW=01 ise Veri
Birikeçten Anlık Veri	0010110W	Veri	W=1 ise Veri	-

Örnek: Aşağıda SUB komutunun değişik kullanım biçimlerini gösteren bir örnek program verilmiştir.

```
1  ORG 0100H
2  MOV AX, 1234H
3  MOV BX, 4321H
4  MOV CX, 1243H
5  SUB BX, CX
6  SUB CX, WORD [0110H]
```

```
7  SUB AX, WORD [0101H]
8  INT 03H
```

Yukarıdaki programın beşinci satırında içeriği 4321H olan BX yazmacından içeriği 1234H olan CX yazmacı çıkarılmaktadır. Çıkarma işlemi sonucunda bulunan fark olan 30EDH değeri BX yazmacının yeni değeri olacaktır. Yedinci ve sekizinci satırlarda ise CX ve AX yazmaçlarından bellekte belirtilen adreslerde bulunan veriler çıkarılmaktadır.

4.5.3.7. SBB

SBB (**su**b**t**ract with **b**orrow – elde ile çıkarma) komutu *işlenen1*'den *işlenen2*'yi çıkarır sonucu *işlenen1*'e kaydeder. SBB komutunun SUB komutundan tek farkı elde ile işlem yapmasıdır. SBB komutu CF bayrak biti 1 ise, çıkarma işlemine ek olarak *işlenen1*'den 1 çıkarır, CF bayrak biti 0 olduğunda ise SUB komutu ile aynı işlemi yapar. CF bayrak bitinin değeri ne olursa olsun *işlenen2*'nin değeri değişmez.

SBB *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç ya da tersi	000110DW	MOD REG R/M	-	-
R/M'den Anlık Veri	100000SW	MOD 011 R/M	Veri	SW=01 ise Veri
Birikeçten Anlık Veri	0001110W	Veri	W=1 ise Veri	-

Örnek: Aşağıda SBB komutunun nasıl kullanıldığını gösteren bir örnek program verilmiştir.

```
1  ORG 0100H
2  MOV AL, 03H
3  MOV BL, 02H
4  SBB AL, BL
5  MOV CL, 0FFH
6  ADD CL, 0FFH
7  MOV CL, 09H
8  MOV DL, 07H
9  SBB CL, DL
10 INT 03H
```

Dördüncü satırdaki SBB komutu CF bayrak biti 0 iken, dokuzuncu satırdaki SBB komutu ise CF bayrak biti 1 olması durumunda çıkarma işlemine örnektir. 4üncü satırdaki SBB komutu CF bayrak biti 0 olduğu durumda kullanıldığından bu komutun yerine doğrudan SUB komutu da kullanılabilir. Öte yandan CF bayrak biti 5inci ve 6ıncı satırlarda FFH ve FFH sayıları toplanarak bire eşit hale getirildiği için dokuzuncu satırdaki SBB komutu çıkarma sonucunda bulunan sonuçtan 1 çıkaracaktır. CF bayrak bitinin değerini birlemek için böyle bir işlem yapmak yerine doğrudan STC komutu da kullanılabilir.

4.5.3.8. DEC

DEC (**dec**rease – azalt) komutu INC komutunun tersidir. *işlenen1*'in değerini 1 azaltır.

DEC işlenen1

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M	1111111W	MOD 001 R/M	-	-
Yazmaç	01001REG	-	-	-

Örnek: Aşağıda DEC komutunun nasıl kullanıldığını gösteren bir örnek program verilmiştir. Bu program sonlandığında BL yazmacının değeri 02H olur.

```
1  ORG 0100H
2  MOV BL,03H
3  DEC BL
4  INT 03H
```

4.5.3.9. NEG

NEG (**neg**ate – ters çevir) komutu *işlenen1*'in 2'ye tümleyenini alır ve sonucu yeniden *işlenen1*'e kaydeder.

NEG *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
2'ye tümleyenini al	1111011W	MOD 011 R/M	-	-

Örnek: Aşağıda NEG komutunun nasıl kullanıldığını gösteren bir örnek program verilmiştir. Bu program sonlandığında BL yazmacının değeri FDH (11111101B) olur.

```
1  ORG 0100H
2  MOV BL,03H
3  NEG BL
4  INT 03H
```

4.5.3.10. CMP

CMP (**co**m**p**are – karşılaştır) komutu *işlenen1* ile *işlenen2*'nin değerlerini karşılaştırır. Karşılaştırmanın sonucuna göre değiştirilmesi gereken bayrak bitlerinin (OF, SF, ZF, AF, PF, CF) değerleri CMP komutu tarafından değiştirilir. Bu komut *işlenen1* ya da *işlenen2*'nin değerini değiştirmez.

CMP *islenen1*, *islenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	001110DW	MOD REG R/M	-	-
R/M ve Anlık Veri	100000SW	MOD 111 R/M	Veri	SW=01 ise Veri
Birikeç ve Anlık Veri	0011110W	Veri	W=1 ise Veri	-

Örnek: Aşağıdaki tabloda örnek olarak *işlenen1* ve *işlenen2* değerleri CMP komutu sonrasında değişmesi gereken bayrak bitlerinin durumları verilmiştir.

Durum	İşlenen1	işlenen2	CMP işlemi sonrasında Bayrak Bitleri
1	5	5	ZF = 1; PF = 1
2	5	6	CF = 1; SF = 1; PF = 1; AF = 1
3	5	4	-
4	5	-4	CF = 1; PF = 1; AF = 1
5	-5	5	SF = 1; PF = 1
6	-5	-5	ZF = 1; PF = 1
7	-5	-6	-

CMP komutunun ardından genellikle JE (**j**ump if **e**qual – eşit ise atla) benzeri bir atlama komutu kullanıldığından hangi bayrak bitinin 1 hangi bayrak bitinin 0 olduğunun bilinmesine genelde gerek yoktur.

Örnek: Aşağıda verilen örnek programda CMP komutu kullanılmıştır.

```
1  ORG 0100H
2  MOV AL, 13H
3  MOV BL, 13H
4  CMP AL, BL
5  INT 03H
```

AL ve BL yazmaçlarının değerleri 3üncü satırın ardından birbirine eşit olduğu için programın dördüncü satırındaki CMP komutu ZF bayrak bitini bire eşitler.

4.5.3.11. AAS

AAS (**A**SCII **A**djust after **S**ubtraction – Çıkarma işlemi sonrası ASCII düzenlemesi) komutu AL yazmacındaki değeri BCD biçiminde paketlenmiş bir ondalık değere çevirir. Eğer AL yazmacının değeri 9'dan büyükse ya da AF bayrak biti 1 ise AL yazmacının değeri 6, AH yazmacının değeri ise 1 azaltılarak AF ve CF bayrak bitleri bire eşitlenir. AL yazmacının değeri 9'dan büyük değilse AF ve CF bayraklarının değeri sıfırlanır. AF bayrağının değerinden bağımsız olarak AL yazmacının üst yarım biti AAS komutundan sonra her zaman sıfırlanır.

AAS

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
AAS	00111111	-	-	-

Örnek: Aşağıda verilen örnek programda AAS komutunun nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AX, 020AH
3  AAS
4  INT 03H
```

Programın ikinci satırındaki MOV komutu AH yazmacının değerini 02H ve AL yazmacının değerini 0A olarak belirlemektedir. Üçüncü satırdaki AAS komutu çalıştırıldığında AL yazmacının değeri 9'dan büyük olan A (onluk tabanda 10) sayısına eşit olduğu için AL yazmacının değeri 6 azaltılarak 04H AH yazmacının değeri ise 1 azaltılarak 01H yapılır. Böylece programın sonunda AX yazmacının değeri 0104H olarak belirlenir.

4.5.3.12. DAS

DAS (**D**ecimal **A**djust after **S**UB – SUB komutu sonrası ondalık düzenleme) komutu AL yazmacındaki değeri, paketlenmiş bir ondalık değere çevirir. Eğer AL yazmacının alt yarım baytının değeri 9'dan büyükse ya da AF bayrak biti 1 ise AL yazmacından 06H çıkarılır ve AF bayrak biti birlenir. Eğer AL yazmacının değeri 9F'den de büyükse ya da CF bayrak biti 1 ise AL yazmacından ayrıca 60H değeri çıkarılır ve CF bayrak biti birlenir.

DAS

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
DAS	00101111	-	-	-

Örnek: Aşağıda verilen örnek programda DAS komutunun nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AL, 0FFH
3  DAS
4  INT 03H
```

Programın ikinci satırında AL yazmacının değeri FF olarak belirlenmiştir. Bu değerin alt yarım baytı olan F, 9'dan büyük, değerin tamamı da 9F'den büyük olduğu için üçüncü satırdaki DAS komutuyla AL yazmacından 66H (önce 06H sonra 60H) çıkarılarak değeri 99'a eşitlenir ve CF bayrağı birlenir.

4.5.3.13. MUL

MUL (**mul**tiply – çarp) komutu *işlenen1*'in boyutu 1 bayt ise *işlenen1*'i AL yazmacı ile çarpıp sonucu AX yazmacına kaydeder. Eğer *işlenen1* iki baytlık bir sözcük ise *işlenen1* AX yazmacı ile çarpılır ve sonuç DX ve AX yazmaçlarına birlikte yazılır. CF ve OF bayrak bitleri sonucun üst baytları 0 olduğu sürece sıfıra eşittir.

MUL *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
MUL	1111011W	MOD 100 R/M	-	-

MUL komutu kullanılırken çarpma kuralları hatırlanmalı ve işlenen boyutlarına dikkat edilmelidir. Örneğin 4 bit uzunluğunda iki değerin çarpımı 8 bit uzunluğunda olabileceğinden sonuç 8 bitlik bir alanda saklanmalıdır. Bu nedenle MUL komutuna 8 bitlik bir işlenen verildiğinde komut 8 bitlik AL yazmacıyla işlem yapacak ve en fazla 16 bit uzunluğunda bir sonuç üretecektir. Bu 16 bitlik sonuç AL yazmacına sığmadığından AX yazmacının tamamı kullanılır. Benzer biçimde MUL komutu 16 bitlik bir işlenenle birlikte kullanıldığında, işlenen 16 bitlik AX yazmacıyla çarpılacak ve çıkan 32 bitlik sonuç, sonucun üst 16 biti DX yazmacında, alt 16 biti ise AX yazmacında olmak üzere saklanacaktır.

Örnek: Aşağıda verilen örnek programda MUL komutunun nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AL,02H
3  MOV BL,03H
4  MUL BL
5  MOV CX, AX
6  MOV BX, 1234H
7  MOV AL, 0FH
8  MUL BX
9  INT 03H
```

Programın ikinci satırında AL yazmacının değeri 2 olarak, üçüncü satırında ise BL yazmacının değeri 3 olarak belirlenmiştir. Dördüncü satırdaki MUL komutu çalıştığında AL yazmacının içerdiği 2 değeri ile BL yazmacının içerdiği 3 değeri çarpılacak ve ortaya çıkan çarpım değeri olan 6 AX yazmacına yazılacaktır. AX yazmacında saklanan 0006H değeri beşinci satırda CX yazmacına aktarılmıştır.

Örnek programın sekizinci satırında ise 16 bitlik BX yazmacı MUL komutuna işlenen olarak verildiği için MUL komutu AX × BX işlemini yapacaktır. 1234H × 000FH işleminin sonucunda ortaya çıkan 1110CH çarpımının üst 16 bitlik bölümü olan 0001H değeri DX yazmacına alt 16 bitlik bölümü olan 110CH değeri ise AX yazmacına yazılacaktır.

Bu örnekte de görüldüğü gibi, programcı MUL komutuyla birlikte kullandığı işlenenin boyutuna dikkat etmek zorundadır.

4.5.3.14. IMUL

IMUL komutu MUL komutunun işaretli işlem yapan türevidir. IMUL komutunun sonucunda bulunan çarpım *işlenen1*'e sığabiliyorsa CF ve OF bayrak bitleri sıfıra eşitlenir.

IMUL *islenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
IMUL	1111011W	MOD 101 R/M	-	-

Örnek: Aşağıda verilen örnek programda IMUL komutunun nasıl kullanıldığı gösterilmiştir. Bu programda MUL komutu için verilen örnektekinden farklı olarak AL yazmacının değeri sıfırdan küçük olan -02H değerine eşitlenmiştir. Program sonuçlandığında AX yazmacının değeri -2 × 3 işleminin sonucu onluk tabandaki karşılığı -6 olan FFFA sayısına eşitlenecektir (AL yazmacının değeri olan FA da -6 sayısının değerini göstermektedir. AH yazmacındaki FF değeri AL yazmacındaki değerinin işaretinin genişletilmiş halidir). OF ve CF bayrakları da işlemin sonucunda sıfırlanacaktır.

```
1  ORG 0100H
2  MOV AL, -02H
3  MOV BL, 03H
4  IMUL BL
5  INT 03H
```

4.5.3.15. AAM

AAM (**A**SCII **A**djust after **M**ultiplication – Çarpma sonrası ASCII düzenlemesi) komutu AL yazmacının değerini alıp 10'a böler ve sonucu AH yazmacına kaydeder. Bölme işlemi sonunda kalan olursa, AL yazmacına kaydedilir.

AAM

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
AAM	11010100	00001010	-	-

Örnek: Aşağıda verilen örnek programda AAM komutunun nasıl kullanıldığı gösterilmiştir. Programın ikinci satırında AL yazmacının değeri 0AH'ye (onluk tabanda 10'a) eşitlenir ve üçüncü satırdaki AAM komutunun ardından AH yazmacının değeri 01H olarak belirlenir. Bölme işlemi sonucunda kalan oluşmadığı için AAM komutu çalıştıktan sonra AX yazmacının aldığı değer 0100H olur.

```
1  ORG 0100H
2  MOV AL, 0AH
3  AAM
4  INT 03H
```

4.5.3.16. DIV

DIV (**div**ide – böl) komutu, *işlenen1* bir bayt uzunluğunda ise, AX yazmacını *işlenen1*'e böler ve bölümü AL yazmacına kalanı AH yazmacına kaydeder. *işlenen1* 16 bitlik bir sözcük ise DX ve AX yazmaçlarının birlikte oluşturduğu değeri *işlenen1*'e böler ve bölümü AX yazmacına kalanı DX yazmacına kaydeder.

DIV *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
DIV	1111011W	MOD 110 R/M	-	-

Örnek: Aşağıda verilen örnek programda DIV komutu kullanılarak nasıl bölme yapıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AX, 0034H
3  MOV BL, 03H
4  DIV BL
5  INT 03H
```

Programın başında AX yazmacının değeri 0034H olarak, BL yazmacının değeri ise 03H olarak belirlenmiş ve dördüncü satırdaki DIV komutu ile 34 / 3 işlemi yapılmıştır. BL yazmacı bir bayt uzunluğunda olduğundan, bu işlem sonucunda ortaya çıkan bölüm olan 11H AL yazmacına, kalan olan 01H ise AH yazmacına yazılır. Böylece AX yazmacının programın ardından aldığı değer 0111H olur.

DIV komutu kullanılırken işlenenin 0 olmamasına dikkat edilmesi gerekir. Sıfıra bölme işleminin sonucu sonsuz olduğundan böyle bir durumda işlemci sıfıra bölme hatası verecektir.

4.5.3.17. IDIV

DIV komutunun işaretli sayılarla işlem yapan türevidir. DIV komutunun kullanıldığı biçimde kullanılır.

IDIV *islenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
IDIV	1111011W	MOD 111 R/M	-	-

Örnek: Aşağıda verilen örnek programda IDIV komutu kullanılarak işaretli sayılarla nasıl bölme yapıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AX, -23H
3  MOV BL, 03H
4  IDIV BL
5  INT 03H
```

Yukarıdaki örnek program -23 / 3 işlemini AL yazmacındaki değeri BL yazmacındaki değere bölerek yapar ve ortaya çıkan bölümü AL yazmacına, kalanı ise AH yazmacına yazar. Bu işlem sonucunda bulunan bölüm -7, kalan ise 2'dir ve AX yazmacının iki yarısında tutulan bu değerler 2'ye tümleyen yöntemi kullanılarak saklanır. Program çalıştıktan sonra AX yazmacının değeri 02F9H olacaktır. -7 sayısı ikiye tümleyen yönteminde F9 olarak gösterilmektedir.

4.5.3.18. AAD

AAD (**A**SCII **A**djust before **D**ivision – Bölme öncesi ASCII düzenleme) komutu bölme işleminde kullanılmak üzere AX yazmacında BCD düzeninde saklanan onluk tabandaki sayıyı ikilik tabana dönüştürür.

AAD

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
AAD	11010101	00001010	-	-

Örnek: Aşağıda verilen örnek programda AAD komutunun nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AH, 01H
3  MOV AL, 02H
4  AAD
5  INT 03H
```

Programda AH yazmacına 01H değeri ve AL yazmacında 02H değeri verilmiştir. Bu iki yazmacın değerleri BCD düzeninde onluk tabandaki 12 sayısını göstermektedir. AAD komutu çalıştırıldığında, onluk tabandaki 12 sayısının AX yazmacının iki yarısından okunan basamakları kullanılarak, ikilik tabandaki 0CH değeri hesaplanır ve sonuç AL yazmacına kaydedilir.

4.5.3.19. CBW

CBW komutu (convert byte to word – baytı sözcüğe çevir) AL yazmacındaki baytı, iki baytlık bir sözcüğe dönüştürür. AH yazmacına, AL yazmacının en anlamlı biti bire eşit ise FFH değeri, değilse 00H değeri yazılır. CBW komutu böylece bir baytlık verilerin 16 bitlik sözcüklere işaret genişlemesi yoluyla çevrilmesini sağlar. CBW her zaman AL yazmacıyla çalışır ve başka işlenen almaz.

CBW

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CBW	10011000	-	-	-

Örnek: Aşağıda verilen örnek programda önce AX yazmacının içeriği sıfırlanmakta, ardından bu yazmacın alt yarısına FE değeri yüklenmektedir.

```
1  ORG 0100H
2  MOV AX, 0000H
3  MOV AL, 0FEH
4  CBW
5  INT 03H
```

CBW komutu çalıştırıldığında AL yazmacının en anlamlı biti 1 olduğu için AX yazmacının programdan sonraki değeri FFFEH olacaktır.

4.5.3.20. CWD

CWD (convert word to double word – sözcüğü çift sözcüğe çevir) komutu AX yazmacındaki 16 bitlik sözcüğü 32 bitlik bir ikili sözcüğe çevirir. Dönüşüm sonrasında ortaya çıkan verinin üst yarısını gösteren sözcük DX yazmacına, alt yarısını gösteren sözcük ise AX yazmacına kaydedilir. AX yazmacının en anlamlı biti 1 ise DX yazmacına FFFFH değeri, sıfır ise DX yazmacına 0000H değeri yazılır.

CWD

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CWD	10011001	-	-	-

Örnek: Aşağıda verilen örnek programda önce DX yazmacının içeriği sıfırlanmakta ve AX yazmacına FFFEH değeri yüklenmektedir. CWD komutunun ardından DX yazmacı AX yazmacının en anlamlı bitinin değeri ile doldurulacağından, DX yazmacı FFFFH değerini alır.

```
1  ORG 0100H
2  MOV DX, 0000H
3  MOV AX, 0FFFEH
4  CWD
5  INT 03H
```

AX yazmacına atanan başlangıç değerinin programda onaltılık tabanda 5 basamaklı bir sayı olarak gösterilmesine karşın işaret bitinin iki baytlık değerin en soldaki biti olduğuna dikkat edilmelidir.

4.5.4. Mantık Komutları

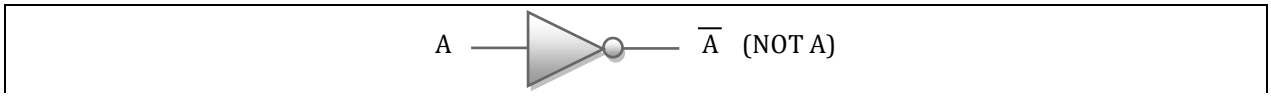
4.5.4.1. NOT

NOT komutu *işlenen1*'in her bitini tersine çevirir. NOT komutunun ardından *işlenen1*'in içinde yer alan her bir 0 biti 1'e, 1 biti ise sıfıra dönüşür.

NOT *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
NOT	1111011W	MOD 010 R/M	-	-

NOT komutunun işlevi mantıksal devre tasarımında kullanılan ve aşağıda simgesi gösterilen DEĞİL kapısının işleviyle aynıdır:



Şekil 5 – NOT komutu ile aynı işleve sahip olan DEĞİL kapısı

Örnek: Aşağıda NOT komutunun program içinde nasıl kullanıldığı gösterilmiştir. 3üncü satırdaki NOT komutunun ardından BL yazmacının değeri 11110000B olacaktır.

```
1  ORG 0100H
2  MOV BL, 00001111B
3  NOT BL
4  INT 03H
```

4.5.4.2. SHL

SHL (**sh**ift logical **l**eft – mantık sola ötele) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sola öteler. En soldaki bit öteleme sonucunda CF bayrak bitine yazılır, en sağdaki bit ise sıfırlanır. Eğer *işlenen1*'in işareti değişmez ise OF bayrağının değeri 0'a, değişirse 1'e eşitlenir.

SHL *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
SHL	110100VW	MOD 100 R/M	-	-

Örnek: Aşağıda SHL komutunun program içinde nasıl kullanıldığı gösterilmiştir. 3üncü satırdaki SHL komutunun ardından BL yazmacının değeri iki basamak sola ötelenerek 0011100B olacak, CF ve OF bayrakları sıfırlanacaktır.

```
1  ORG 0100H
2  MOV BL, 00001111B
3  SHL BL, 02H
4  INT 03H
```

4.5.4.3. SAL

SAL (**s**hift **a**rithmetic **l**eft – aritmetik sola ötele) komutu SHL komutu ile aynı işleve, işlem koduna ve adresleme koduna sahiptir. SHL komutunun kullanıldığı her programda SHL komutunun yerine SAL komutu yazılabilir.

SAL *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
SAL	110100VW	MOD 100 R/M	-	-

Örnek: SHL komutu için verilen örnekte SHL yerine aşağıda gösterildiği gibi SAL yazılırsa programın işlevi değişmez.

```
1  ORG 0100H
2  MOV BL, 00001111B
3  SAL BL, 02H
4  INT 03H
```

4.5.4.4. SHR

SHR (**sh**ift logic **r**ight – mantık sağa ötele) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sağa öteler. En sağdaki bit öteleme sonucunda CF bayrak bitine kaydırılır, en soldaki bit ise sıfırlanır. Eğer *işlenen1*'in işareti değişmez ise OF bayrağının değeri sıfırlanır, değişirse birlenir.

SHR *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
SHR	110100VW	MOD 101 R/M	-	-

Örnek: Aşağıdaki örnekte SHR komutu kullanılarak BL yazmacındaki verinin iki basmak sağa nasıl ötelendiği gösterilmektedir. Programın çalışmasının ardından BL yazmacının içeriği 00000011B olacak ve OF bayrağı sıfırlanacaktır.

```
1  ORG 0100H
2  MOV BL, 00001111B
3  SHR BL, 02H
```

```
4  INT 03H
```

4.5.4.5. SAR

SAR (**s**hift **a**rithmetic **r**ight – aritmetik sağa ötele) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sağa öteler. En sağdaki bit öteleme sonucunda CF bayrak bitine kaydedilir. En soldaki bit işaret biti kabul edildiğinden değeri değiştirilmez. SAR komutu aritmetik işlemlerde işaret biti yitirilmeden ötelenme işlemi yapılması için daha uygundur.

SAR *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
SAR	110100VW	MOD 111 R/M	-	-

Örnek: Aşağıdaki örnekte BL yazmacının değeri başlangıçta ikilik tabandaki karşılığı 10000100B olan 84H olarak belirlenmiştir. BL yazmacında saklanan bu değerin en soldaki biti 1 olduğu için saklanan sayı sıfırdan küçüktür.

```
1  ORG 0100H
2  MOV BL, 84H
3  SAR BL, 02H
4  INT 03H
```

SAR komutu çalıştırıldığında BL yazmacının değeri iki basamak sola ötelenecek, ancak soldaki işaret biti değiştirilmeyerek aynen kalacaktır. Ötele işlemi sırasında her adımda en soldaki bitten sağa ötelenecek bit işaret biti olan 1'dir. Böylece programın ardından BL yazmacı ikilik tabandaki karşılığı 11100001B olan E1H değerini tutacaktır.

4.5.4.6. ROL

ROL (**ro**tate **l**eft – sola döndür) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sola doğru döndürerek *işlenen1*'deki değeri öteler. Öteleme öncesinde en solda olan bit, öteleme sonrasında en sağa geçer ve aynı zamanda CF bayrak bitine kaydedilir. ROL komutunun ardından eğer *işlenen1*'in işareti değişmez ise OF bayrak bitinin değeri 0 olur. *işlenen1*'in işaretinin değerinin değişmesi durumunda OF bayrak biti birlenir.

ROL *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
ROL	110100VW	MOD 000 R/M	-	-

Örnek: Aşağıdaki örnekte BL yazmacının değeri başlangıçta ikilik tabandaki karşılığı 01000100B olan 44H olarak belirlenmiştir.

```
1  ORG 0100H
2  MOV BL, 44H
3  ROL BL, 02H
4  INT 03H
```

Üçüncü satırdaki ROL komutu BL yazmacının içeriğini sola doğru 2 basamak döndürecektir. İşlem sonucunda BL yazmacının alacağı değer 00010001B (11H) olacaktır. BL yazmacının işareti değişmediği için OF bayrak bitinin değeri sıfırdır.

4.5.4.7. ROR

ROR (**ro**tate **r**ight – sağa döndür) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sağa döndürür. Yani bitler sağa ötelenir. Öteleme öncesinde en sağda olan bit öteleme sonrasında en sola geçer. Bu bit aynı zamanda CF bayrak bitine kaydedilir. OF bayrak biti, eğer *işlenen1*'in işareti değişmez ise 0, aksi takdirde 1 değerini alır.

ROR *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
ROR	110100VW	MOD 001 R/M	-	-

Örnek: Aşağıdaki örnekte BL yazmacının değeri başlangıçta ikilik tabandaki karşılığı 10000100B olan 84H olarak belirlenmiştir. BL yazmacında saklanan bu değerin en soldaki biti 1 olduğu için saklanan sayı sıfırdan

küçüktür. ROR komutunun ardından BL yazmacının değeri 2 basamak sağa dönerek 00100001B (21H) olur. İşlemin sonunda BL yazmacı işaret değiştirdiği için OF bayrak biti birlenir.

```
1  ORG 0100H
2  MOV BL, 84H
3  ROR BL, 02H
4  INT 03H
```

4.5.4.8. RCL

RCL (**r**otate with **C**F **l**eft – CF bayrak biti ile birlikte sola döndür) komutu *işlenen1* ile CF'nin birlikte oluşturduğu değeri *işlenen2*'nin değeri kadar sola döndürerek öteler. Öteleme öncesinde en solda olan bit öteleme sonrasında en sağdaki bit olarak kullanılan CF bayrak bitine geçer. Bayrak bitindeki değer ise *işlenen1*'in en sağındaki bitine aktarılır. OF bayrak biti, eğer *işlenen1*'in işareti değişmez ise 0, aksi takdirde 1 değerini alır.

RCL *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
RCL	110100VW	MOD 010 R/M	-	-

Örnek: Aşağıdaki örnekte BL yazmacının değeri başlangıçta ikilik tabandaki karşılığı 01000100B olan 44H olarak belirlenmiş ve STC komutuyla CF bayrak biti birlenmiştir. RCL komutu ile BL yazmacı ile CF bayrak bitinin oluşturduğu değer sola doğru iki basamak döndürülerek, BL yazmacının değeri 00010010B (12H) değerine CF bayrak bitinin değeri ise 1'e getirilecektir.

```
1  ORG 0100H
2  MOV BL, 44H
3  STC              ;CF bayrak bitini 1 yapar
4  RCL BL, 02H
5  INT 03H
```

4.5.4.9. RCR

RCR (**r**otate with **C**F **r**ight – CF bayrak biti ile birlikte sağa döndür) komutu *işlenen1* CF'nin birlikte oluşturduğu değeri *işlenen2*'nin değeri kadar sağa döndürerek öteler. Öteleme öncesinde en sağda olan bit, öteleme sonrasında en soldaki bit olarak kullanılan CF bayrak bitine geçer. Bayrak bitindeki değer ise *işlenen1*'in en solundaki bite aktarılır. OF bayrak biti eğer *işlenen1*'in işareti değişmez is 0, aksi takdirde 1 değerini alır.

RCR *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
RCR	110100VW	MOD 011 R/M	-	-

Örnek: Aşağıdaki örnekte BL yazmacının değeri başlangıçta ikilik tabandaki karşılığı 01000100B olan 44H olarak belirlenmiş ve STC komutuyla CF bayrak biti birlenmiştir. RCR komutunun ardından CF bayrak biti ile BL yazmacının oluşturduğu değer sağa doğru iki basamak döndürülerek, BL yazmacının değeri 01010001B (11H) değerine CF bayrak bitinin değeri ise 0'a getirilecektir.

```
1  ORG 0100H
2  MOV BL, 44H
3  STC              ;CF bayrak bitini 1 yapar
4  RCR BL, 02H
5  INT 03H
```

4.5.4.10. AND

AND komutu mantıksal devre tasarımında kullanılan VE kapısının görevini görür. *işlenen1* ve *işlenen2*'nin her bitinin değerine göre aşağıdaki tabloya uygun olarak bir sonuç üretilir ve üretilen sonuç *işlenen1*'e aktarılır.

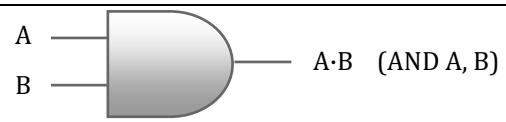
işlenen1	*işlenen2*	Sonuç
0	0	0
0	1	0

1	0	0
1	1	1

AND *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	001000DW	MOD REG R/M	-	-
R/M ve Anlık Veri	1000000W	MOD 100 R/M	Veri	W=1 ise Veri
Birikeç ve Anlık Veri	0010010W	Veri	W=1 ise Veri	-

AND komutunun işlevi, mantıksal devre tasarımında kullanılan ve aşağıda simgesi gösterilen VE kapısıyla aynıdır:



Şekil 6 – AND komutu ile aynı işleve sahip olan VE kapısı

Örnek: Aşağıda AND komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AL, 01010101B
3  MOV BL, 00001111B
4  AND AL, BL              ;Yazmaç VE Yazmaç
5  AND BL, 11110010B       ;Yazmaç VE Anlık Veri
6  AND BL, [0100H]         ;Yazmaç VE Bellek
7  AND AL, [0101H]         ;Birikeç VE Bellek
8  INT 03H
```

Programın başında AL yazmacının değeri 01010101B, BL yazmacının değeri ise 00001111B olarak belirlenmiştir. 4üncü satırdaki AND komutu iki yazmacın tuttuğu değerleri kullanarak VE işlemi yapar ve sonuç değeri olan 00000101B'yi AL yazmacına yazar. 5inci satırdaki AND komutu BL yazmacının değerini doğrudan 11110010B değeri ile VE işlemine sokmakta ve sonucu BL yazmacına yazmaktadır. 5inci satırın ardından BL yazmacının değeri 00000010B olur. Programın 6ıncı ve 7inci satırlarında ise AL ve BL yazmaçlarının değerleri bellekte bulunan değerlerle VE işlemine sokulmaktadır.

4.5.4.11. TEST

TEST komutu AND komutuna benzer biçimde işlenenler arasında VE işlemini gerçekleştiren bir komuttur. AND komutundan farklı olarak TEST komutunun ardından gerçekleştirilen VE işleminin sonucu herhangi bir yerde saklanmaz. Komutun ardından VE işleminin sonucuna bağlı olarak, yalnızca SF, ZF ve PF bayrak bitlerinin değerleri 1 ya da 0 olarak belirlenir. Test komutu daha çok döngülerin koşullarını denetlemek amacıyla kullanılır.

TEST *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	1000010W	MOD REG R/M	-	-
R/M ve Anlık Veri	1111011W	MOD 000 R/M	Veri	W=1 ise Veri
Birikeç ve Anlık Veri	1010100W	Veri	W=1 ise Veri	-

Örnek: Aşağıda AND komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV BL, 00001111B
3  TEST BL, 11110010B
4  INT 03H
```

3üncü satırdaki test komutunun ardından BL yazmacının değeri değişmez. Ancak VE işleminin sonucu 00000010B olduğu için işaret bitini gösteren SF bayrak bitinin değeri 0, sonucun sıfır olup olmadığını gösteren ZF bayrak biti 0 ve üretilen eşlik (parite) değerini gösteren PF bayrak biti de 0 olarak belirlenir. Eğer üçüncü satırdaki 11110010B değeri yerine 11110000B değeri kullanılırsa SF, ZF ve PF bayrak bitleri sırasıyla 0, 1 ve 1 değerlerini alır.

4.5.4.12. OR

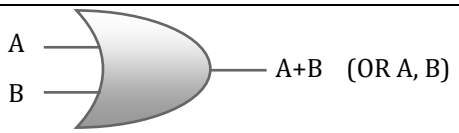
OR komutu mantıksal devre tasarımında kullanılan VEYA kapısının görevini görür. *işlenen1* ve *işlenen2*'nin her bitinin değerine göre aşağıdaki tabloya uygun olarak bir sonuç üretilir ve üretilen sonuç *işlenen1*'e aktarılır.

işlenen1	*işlenen2*	Sonuç
0	0	0
0	1	1
1	0	1
1	1	1

OR *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	000010DW	MOD REG R/M	-	-
R/M ve Anlık Veri	1000000W	MOD 001 R/M	Veri	W=1 ise Veri
Birikeç ve Anlık Veri	0000110W	Veri	W=1 ise Veri	-

OR komutunun işlevi, mantıksal devre tasarımında kullanılan ve aşağıda simgesi gösterilen VEYA kapısıyla aynıdır:



Şekil 7 – OR komutu ile aynı işleve sahip olan VEYA kapısı

Örnek: Aşağıda OR komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AL, 01010101B
3  MOV BL, 00001111B
4  OR AL, BL              ;Yazmaç VEYA Yazmaç
5  OR BL, 11110010B       ;Yazmaç VEYA Anlık Veri
6  OR BL, [0100H]         ;Yazmaç VEYA Bellek
7  OR AL, [0101H]         ;Birikeç VEYA Bellek
8  INT 03H
```

Programın başında AL yazmacının değeri 01010101B, BL yazmacının değeri ise 00001111B olarak belirlenmiştir. 4üncü satırdaki OR komutu iki yazmacın tuttuğu değerleri kullanarak VEYA işlemi yapar ve sonuç değeri olan 01011111B'yi AL yazmacına yazar. 5inci satırdaki OR komutu BL yazmacının değerini doğrudan 11110010B değeri ile VEYA işlemine sokmakta ve sonucu BL yazmacına yazmaktadır. 5inci satırın ardından BL yazmacının değeri 11111111B olur. Programın 6ıncı ve 7inci satırlarında ise AL ve BL yazmaçlarının değerleri bellekte bulunan değerlerle VEYA işlemine sokulmaktadır.

4.5.4.13. XOR

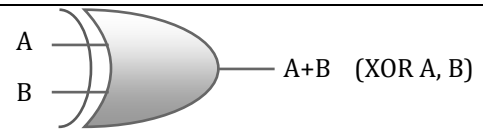
XOR komutu mantıksal devre tasarımında kullanılan ÖZEL VEYA kapısının görevini görür. *işlenen1* ve *işlenen2*'nin her bitinin değerine göre aşağıdaki tabloya uygun olarak bir sonuç üretilir ve üretilen sonuç *işlenen1*'e aktarılır.

işlenen1	*işlenen2*	Sonuç
0	0	0
0	1	1
1	0	1
1	1	0

XOR *işlenen1*, *işlenen2*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
R/M ve Yazmaç	001100DW	MOD REG R/M	-	-
R/M ve Anlık Veri	1000000W	MOD 110 R/M	Veri	W=1 ise Veri
Birikeç ve Anlık Veri	0011010W	Veri	W=1 ise Veri	-

XOR komutunun işlevi, mantıksal devre tasarımında kullanılan ve aşağıda simgesi gösterilen VEYA kapısıyla aynıdır:



Şekil 8 – XOR komutu ile aynı işleve sahip olan ÖZEL VEYA kapısı

Örnek: Aşağıda XOR komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  MOV AL, 01010101B
3  MOV BL, 00001111B
4  XOR AL, BL            ;Yazmaç ÖZEL VEYA Yazmaç
5  XOR BL, 11110010B     ;Yazmaç ÖZEL VEYA Anlık Veri
6  XOR BL, [0100H]       ;Yazmaç ÖZEL VEYA Bellek
7  XOR AL, [0101H]       ;Birikeç ÖZEL VEYA Bellek
8  INT 03H
```

Programın başında AL yazmacının değeri 01010101B, BL yazmacının değeri ise 00001111B olarak belirlenmiştir. 4üncü satırdaki XOR komutu iki yazmacın tuttuğu değerleri kullanarak ÖZEL VEYA işlemi yapar ve sonuç değeri olan 01011010B'yi AL yazmacına yazar. 5inci satırdaki XOR komutu BL yazmacının değerini doğrudan 11110010B değeri ile ÖZEL VEYA işlemine sokmakta ve sonucu BL yazmacına yazmaktadır. 5inci satırın ardından BL yazmacının değeri 11111101B olur. Programın 6ıncı ve 7inci satırlarında ise AL ve BL yazmaçlarının değerleri bellekte bulunan değerlerle ÖZEL VEYA işlemine sokulmaktadır.

4.5.5. Dizgi İşleme Komutları

4.5.5.1. REP/REPE/REPNE/REPNZ/REPZ

REP (**rep**eat – yinele) komutu işlenen olarak verilen MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW komutlarından birinin CX yazmacındaki değer kadar tekrar edilmesini sağlar. Her yinelemede CX yazmacının değeri 01H kadar azaltılır. CX yazmacı 0 olduğunda sıradaki komuta atlar.

REPE (**rep**eat if **e**qual – eşit ise yinele) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birini ZF bayrak biti 1 olduğu ve CX yazmacındaki değer sıfırdan büyük olduğu sürece yineler. ZF bayrak biti 0 olduğu anda yineleme durdurulur ve bir sonraki komuta atlanır. Her yinelemede CX yazmacının değeri 01H kadar azaltılır.

REPZ (**rep**eat if **z**ero – sıfır ise yinele) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birini CX yazmacındaki değer 0 olana kadar, ZF bayrak biti 1 olduğu sürece yineler. ZF bayrak biti 0 olduğu anda yineleme durdurulur ve bir sonraki komuta atlanır. Her yinelemede CX yazmacının değeri 01H kadar azaltılır.

REPNE (**rep**eat if **n**ot **e**qual – eşit değilse yinele) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birini CX yazmacındaki değer 0 olana kadar, ZF bayrak biti 0 olduğu sürece yineler. ZF bayrak biti 1 olduğu anda yineleme durdurulur ve bir sonraki komuta atlanır. Her yinelemede CX yazmacının değeri 01H kadar azaltılır.

REPNZ (**rep**eat if **n**ot **z**ero – sıfır değilse yinele) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birini CX yazmacındaki değer 0 olana kadar, ZF bayrak biti 0 olduğu sürece yineler. ZF bayrak biti 1 olduğu anda yineleme durdurulur ve bir sonraki komuta atlanır. Her yinelemede CX yazmacının değeri 01H kadar azaltılır.

REP işlenen1
REPE işlenen1
REPZ işlenen1
REPNE işlenen1
REPNZ işlenen1

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
REP	1111001Z	-	-	-

4.5.5.2. MOVSB/MOVSW

MOVSB komutu veri bölütünde bulunan DS:[SI] adresindeki bir baytı ek bölütte bulunan ES:[DI] adresine kaydeder. MOVSW komutu aynı işlemi yapar ancak bir bayt yerine iki baytlık bir sözcük aktarır. Okunan dizginin yönü DF bayrak bitinde 0 olarak gösterilmiş ise SI ve DI işlem sonunda 1 artırılır. DF bayrak bitinin 1 olması durumunda SI ve DI 1 azaltılır.

MOVSB **MOVSW**

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
MOVS	1010010W	-	-	-

Örnek: Aşağıda MOVSB komutunun program içinde REP komutuyla birlikte nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  CLD                    ;DF = 0
3  MOV SI, DEG1           ;SI'ya DEG1'in adresini kaydet
4  MOV DI, DEG2           ;DI'ya DEG2'nin adresini kaydet
5  MOV CX, 04H            ;CX = 4
6  REP MOVSB              ;MOVSB işlemini 4 kez tekrarla
7  INT 03H
8  DEG1 DB 1, 2, 3, 4
9  DEG2 DB 5, 6, 7, 8
```

Programın başında DF bayrak biti sıfırlanmış, SI ve DI yazmaçlarına sırasıyla DEG1 ve DEG2'nin başlangıç adresleri atanmıştır. Bu işlemlerin ardından REP komutunun yardımıyla DEG1'in gösterdiği bellek konumundan DEG2'nin gösterdiği bellek konumuna 4 bayt art arda aktarılır. İşlemin sonucunda DEG2'nin bellekte bulunduğu bölgedeki bayt değerleri sırasıyla 1, 2, 3, 4 olur.

4.5.5.3. CMPSB/CMPSW

CMPSB komutu ES:[DI] ve DS:[SI] adreslerinde bulunan iki baytı karşılaştırır. CMPSW komutu ise aynı bellek konumlarındaki iki baytlık iki sözcüğü birbiriyle karşılaştırır. OF, SF, ZF, AF, PF ve CF bayrak bitleri bu iki karşılaştırma işleminin sonucunda değişir. DF bayrak bitinin değeri 0 ise SI ve DI 1 artırılır, 1 ise SI ve DI 1 azaltılır.

CMPSB **CMPSW**

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CMPS	1010011W	-	-	-

Örnek: Aşağıda CMPSB komutunun program içinde REPE komutuyla birlikte nasıl kullanıldığı gösterilmiştir.

```
1        ORG 0100H
2        CLD                ;DF = 0
3        MOV SI, DEG1       ;SI'ya DEG1'in adresini kaydet
4        MOV DI, DEG2       ;DI'ya DEG2'nin adresini kaydet
5        MOV CX, 04H        ;CX = 4
6        REPE CMPSB         ;Baytları karşılaştır
7        JNZ ESIT_DEGIL     ;ZF = 0 ise ESIT_DEGIL'e atla
8        MOV AX, 1234H
9        INT 03H
10 ESIT_DEGIL:
11       MOV AX, 0000H
12       INT 03H
13       DEG1 DB 1, 2, 3, 4
14       DEG2 DB 1, 2, 3, 4
```

Programın başında DF bayrak biti sıfırlanmış, SI ve DI yazmaçlarına sırasıyla DEG1 ve DEG2'nin bellekteki başlangıç adresleri atanmıştır. Bu işlemlerin ardından REPE komutunun yardımıyla DEG1'in gösterdiği bellek konumundaki 4 bayt DEG2'nin gösterdiği bellek konumundaki 4 bayt ile art arda karşılaştırılır. REPE komutu birbirinden farklı iki baytı karşılaştırdığında döngü durur ve yedinci satırdaki JNZ komutu çalıştırılır. İki farklı bayt birbiriyle karşılaştırıldığında ZF'nin değeri 0 olacağından program

10uncu satırdaki ESIT_DEGIL satırına atlar ve AX yazmacının içeriğini sıfırlar. Eğer REPE komutu birbirinden farklı iki bayt bulamazsa dört kere döner ve ZF bitinin değeri 1 olur. Bu durumda AX yazmacının değeri 1234H olarak belirlenir.

4.5.5.4. SCASB/SCASW

SCASB komutu AL yazmacındaki değer ile ES:[DI] adresindeki baytı karşılaştırır. Karşılaştırma sonucunda OF, SF, ZF, AF, PF ve CF bayrak bitlerinin değerleri değişir. Komut çalıştırıldığı sırada DF bayrak biti 0 ise komutun işlenmesinin ardından DI yazmacının değeri bir artırılır; DF bayrak bitinin değerinin 1 olması durumunda ise DI yazmacının değeri bir azaltılır. Benzer biçimde SCASW komutu AX yazmacındaki 16 bitlik sözcük ile ES:[DI] adresinde bulunan sözcüğü karşılaştırır. SCASW komutu çalıştırıldığında DF bayrak biti 0 ise DI yazmacının değeri 2 artırılır; DF bayrak bitinin değerinin 1 olması durumunda ise DI yazmacının değeri 2 azaltılır.

SCASB
SCASW

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
SCAS	1010111W	-	-	-

Örnek: Aşağıda SCASB komutunun program içinde REPE komutuyla birlikte nasıl kullanıldığı gösterilmiştir.

```
1         ORG 0100H
2         CLD                 ;DF = 0
3         MOV DI, DEG1        ;DI'ya DEG1'in adresini kaydet
4         MOV CX, 04H         ;CX = 4
5         MOV AL, 02H
6         REPE SCASB          ;Baytları karşılaştır
7         JNZ ESIT_DEGIL      ;ZF = 1 ise ESIT_DEGIL e atla
8         MOV AX, 1234H
9         INT 03H
10  ESIT_DEGIL:
11        MOV AX, 0000H
12        INT 03H
13        DEG1 DB 02H, 02H, 02H, 02H
```

Yukarıdaki programın başında DF bayrak biti CLD komutu kullanılarak sıfırlanmış, DI yazmacına DEG1'in bellekteki başlangıç adresi atanmıştır. Beşinci satırda AL yazmacının değeri 02H olarak belirlenmiştir. Bu işlemlerin ardından REPE komutunun yardımıyla AL yazmacının değeri DEG1'in gösterdiği bellek konumundaki 4 bayt ile art arda karşılaştırılır. REPE komutu birbirinden farklı iki baytı karşılaştırdığında döngü durur ve yedinci satırdaki JNZ komutu çalıştırılır. İki farklı bayt birbiriyle karşılaştırıldığında ZF'nin değeri 0 olacağından program 10uncu satırdaki ESIT_DEGIL satırına atlar ve AX yazmacının içeriğini sıfırlar. Eğer REPE komutu, örnekteki program olduğu gibi, birbirinden farklı iki bayt bulamazsa dört kere döner ve ZF bitinin değeri 1 olur. Bu durumda AX yazmacının değeri 1234H olarak belirlenir. Bu örnek programda AL yazmacının değeri olan 02H ile DEG1 değişkeninin gösterdiği bellek konumunda art arda gelen 4 baytın değeri aynı olduğundan programın sonucunda AX yazmacının değeri 1234H olarak belirlenecektir.

4.5.5.5. LODSB/LODSW

LODSB komutu DS:[SI] adresindeki baytı AL yazmacına aktarır. Komut çalıştırıldığında DF bayrak bitinin değeri 0 ise SI yazmacının değeri 1 artırılır; DF bayrak bitinin değerinin 1 olması durumunda ise SI yazmacının değeri 1 azaltılır. LODSW komutu, LODSB komutuna benzer biçimde, DS:[SI] adresindeki 16 bitlik bir sözcüğü AX yazmacına aktarır. LODSW komutu çalıştırıldığında DF bayrak bitinin değeri 0 ise SI yazmacının değeri 2 artırılır; DF bayrak bitinin değerinin 1 olması durumunda ise SI yazmacının değeri 2 azaltılır.

LODSB
LODSW

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
LODS	1010110W	-	-	-

Örnek: Aşağıda LODSB komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  CLD                          ;DF = 0
3  MOV SI, DEG1                 ;SI'ya DEG1'in adresini kaydet
4  MOV CX, 3                    ;CX = 3
5  REP LODSB                    ;AL'ye 3 kere bayt yükle
6  INT 03H
7  DEG1 DB 01H, 02H, 03H, 04H, 05H
```

Programın başında SI yazmacına DEG1'in bellekteki başlangıç adresi atanmıştır. Bu bellek adresinde sırasıyla 01H, 02H, 03H, 04H ve 05H değerleri bulunmaktadır. CX yazmacının değeri 3 olarak belirlendiği için beşinci satırdaki REP komutu LODSB komutunu 3 kere çalıştıracaktır. DF bayrak biti ikinci satırdaki CLD komutu ile sıfırlandığı için LODSB komutu her çalıştığında SI yazmacının değeri bir artacaktır. Böylece REP komutunun 3 kere çalışması sırasında AL yazmacına sırasıyla 01H, 02H ve 03H değerleri aktarılacaktır.

4.5.5.6. STOSB/STOSW

STOSB komutu AL yazmacındaki baytı ES:[DI] adresine yazar. Komut çalıştırıldığı sırada DF bayrak bitinin değeri 0 ise DI yazmacının değeri 1 artar; DF bayrak bitinin değerinin 1 olması durumunda ise DI yazmacının değeri 1 azaltılır. Benzer biçimde STOSW komutu AX yazmacındaki 16 bitlik sözcüğü ES:[DI] adresine yazar. STOSW komutu çalıştığı sırada DF bayrak bitinin değeri 0 ise DI yazmacının değeri 2 artırılır; DF bayrak bitinin değerinin 1 olması durumunda ise DI yazmacının değeri 2 azaltılır.

STOSB
STOSW

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
STOS	1010101W	-	-	-

Örnek: Aşağıda STOSB komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1  ORG 0100H
2  CLD                          ;DF = 0
3  MOV DI, DEG1                 ;DI'ya DEG1'in adresini kaydet
4  MOV AL, 0FFH                 ;AL = FF
5  MOV CX, 4                    ;CX = 4
6  REP STOSB
7  INT 03H
8  DEG1 DB 01H, 02H, 03H, 11H
```

Programın başında DI yazmacına DEG1'in bellekteki başlangıç adresi atanmıştır. Program çalışmadan önce bu bellek adresinde sırasıyla 01H, 02H, 03H ve 11H değerleri bulunmaktadır. CX yazmacının değeri 4 olarak belirlendiği için altıncı satırdaki REP komutu STOSB komutunu 4 kere çalıştıracaktır. DF bayrak biti ikinci satırdaki CLD komutu ile sıfırlandığı için STOSB komutu her çalıştığında SI yazmacının değeri bir artacaktır. Böylece REP komutunun 4 kere çalışması sırasında AL yazmacındaki FF verisi DI yazmacının bellekte gösterdiği art arda sıralanmış 4 baytın üzerine yazılacaktır. Böylece programın çalışmasından önce bellekte 01H, 02H, 03H ve 11H değerlerinin bulunduğu baytların değerlerinin tamamı FFH olarak değişecektir.

4.5.6. Denetim Aktarma Komutları

Programcı yazdığı programda bütün komutların art arda işlenmesini istemeyebilir. Programın belirli noktalarında kullanıcıdan alınan girdilerin ya da başka koşulların denetlenerek farklı kod parçalarının çalıştırılması ve program denetimin programın değişik parçalarına aktarılması gerekebilir. Bu amaçla, işlendiği zaman işlemcinin bir sonraki adımda çalıştıracağı komutun yerini belirleyen "Denetim Aktarma Komutları" kullanılır. Bu komutlar bir koşula bağlı olabileceği gibi hiçbir koşula bağlı olmadan da çalışabilirler.

4.5.6.1. CALL

CALL komutu denetimin bir yordama aktarılmasını sağlar ve birkaç değişik adresleme çeşidi kullanır. CALL komutunun bölüt içi doğrudan adresleme, bölüt içi dolaylı adresleme, bölütler arası doğrudan adresleme ve bölütler arası dolaylı adresleme yapan türevleri bulunmaktadır. Bölüt içi adreslemenin kullanıldığı durumlarda, komut çalıştırıldığı sırada IP yazmacının değeri yığıta kaydedildikten sonra altyordama atlanır. İşlemci altyordamda bir RET komutu ile karşılaştığında, saklanan değerler yığıttan geri

alınır ve program alt yordamın çağrıldığı noktaya geri döner. CALL komutunun bölütler arasında kullanılması durumunda IP yazmacına ek olarak CS yazmacı da yığıta kaydedilir.

CALL *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4	Kod 5
Bölüt içi doğrudan	11101000	DISP* (alt)	DISP* (üst)	-	-
Bölüt içi dolaylı	11111111	MOD 010 R/M	-	-	-
Bölütler arası doğrudan	10011010	OFF*(alt)	OFF*(üst)	SEG*(alt)	SEG*(üst)
Bölütler arası dolaylı	11111111	MOD 011 R/M	-	-	-

*OFF= Göreceli Adres SEG= Bölüt Adresi DISP= Uzaklık Verisi

Örnek: Aşağıda CALL komutunun bölüt içinde doğrudan adreslemesine bir örnek verilmiştir.

```
1         ORG 0100H
2         CALL CAGRI
3         INT 03H
4   CAGRI:
5         MOV AX, 1234H
6         RET
```

Yukarıdaki program DEBUG yazılımı kullanılarak denendiğinde, program çalıştırıldıktan sonra AX yazmacının değerinin 1234H'ye eşit olduğu görülür. Programın ikinci satırı işlendikten sonra program dördüncü satırda başlayan CAGRI altyordamına atlar. Bu alt yordamda AX yazmacının değeri 1234H'ye eşitlendikten sonra altıncı satırdaki RET komutu programı altyordamın çağrıldığı noktaya geri döndürür. Program son olarak 3üncü satırın işlenmesiyle sonlanır.

DEBUG içinde "D" komutu kullanılarak programın bellekte saklandığı konumdaki verilere bakılırsa CALL komutunun karşılığının E8H olduğu görülür. Bellekte görülen bu kod yukarıdaki tablo yardımıyla çözüldüğünde kullanılan CALL komutunun "Bölüt içi doğrudan" adresleme türünü kullandığı anlaşılır.

Örnek: Aşağıda CALL komutunun bölüt içinde dolaylı adresleme türünü kullanmasına bir örnek verilmiştir.

```
1   ORG 0100H
2   MOV AX, 1234H
3   CALL AX
4   INT 03H
```

Bölüt içinde dolaylı adresleme yapılması istendiğinde yukarıdaki programda olduğu gibi göreceli adres bir yazmaca yazılır ve bu yazmaç işlenen olarak CALL komutuna verilir. Örnekte üçüncü satırdaki CALL komutu denetimi bellekte 1234H göreceli konumunda bulunan altyordama aktaracaktır.

4.5.6.2. RET/RETF

RET komutu, CALL komutu ile çağırılmış bir altyordamdan geri dönülmesini sağlar. RET komutu çalıştırıldığı anda IP yazmacını yığıttan geri yükler ve programın daha önce CALL komutu tarafından yığıta saklanan konumuna geri dönmesini sağlar. Bölütler arası bir CALL komutunun kullanılması durumunda RET yerine RETF komutu komutu kullanılmalıdır. RETF komutu çağrıldığında RET komutundan farklı olarak hem CS hem IP yazmaçlarının değerleri yığıttan geri yüklenir.

Eğer RET ve RETF komutlarıyle birlikte işlenen olarak anlık bir değer kullanılmış ise bu anlık değer SP yazmacına eklenir. RET ve RETF komutlarıyla birlikte işlenen kullanılması gerekli değildir; bu komutlar yalnız başlarına da kullanılabilirler.

RET *işlenen1* **RETF** *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Bölüt içi	11000011	-	-	-
Bölü içi ve anlık veri	11000010	Veri (alt)	Veri (üst)	-
Bölütler arası	11001011	-	-	-
Bölütler arası ve anlık veri	11001010	Veri (alt)	Veri (üst)	-

4.5.6.3. JMP

JMP komutu, programın *işlenen1*'de belirtilen adrese doğrudan, koşulsuz olarak, atlamasını sağlar.

JMP *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4	Kod 5
Bölüt içi doğrudan	11101001	DISP* (alt)	DISP* (üst)	-	-
Bölüt içi dolaylı	11111111	MOD 100 R/M	-	-	-
Bölütler arası doğrudan	11101010	OFF*(alt)	OFF*(üst)	SEG*(alt)	SEG*(üst)
Bölütler arası dolaylı	11111111	MOD 101 R/M	-	-	-
Bölüt içi doğrudan (kısa)	11101011	DISP*	-	-	-

*OFF= Göreceli Adres SEG= Bölüt Adresi DISP= Uzaklık Verisi

Örnek: Aşağıda JMP komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1        ORG 0100H
2        MOV AX, 0000H
3        MOV BX, 0000H
4        JMP AX_DOLDUR
5        MOV BX, 0FFFFH
6  AX_DOLDUR:
7        MOV AX, 0FFFFH
8        INT 03H
```

Yukarıdaki kod FASM ile çevrilip DEBUG ile denendiğinde BX yazmacının değerinin FFFFH olmadığı görülür. Bunun nedeni JMP komutu ile MOV BX, 0FFFFH komutunun üzerinden atlanmasıdır.

JMP komutuna işlenen olarak etiket girilmesi zorunlu değildir. Örneğin aşağıdaki kod doğrudan 1234H göreli adresine atlar.

```
1  ORG 0100H
2  JMP 1234H
3  INT 03H
```

4.5.6.4. JE/JZ

JE (**j**ump if **e**qual – eşit ise atla) ve JZ (**j**ump if **z**ero – sıfır ise atla) komutları ZF bayrak bitinin değerinin 1 olması durumunda *işlenen1*'de belirtilen adrese atlar. ZF bitinin değerinin 0 olması durumunda program bir sonraki komutun işlenmesiyle devam eder. JE ya JZ komutları işlenen olarak bir etiket alabileceği gibi doğrudan adres değeriyle kullanılabilir. JE ve JZ komutlarının, adları dışında, işlev açısından hiçbir farkları yoktur; program içinde biri diğerinin yerine kullanılabilir.

JE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JE/JZ	01110100	Uzaklık Verisi	-	-

Örnek: Aşağıda JE komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1        ORG 0100H
2        MOV AL, 03H
3        MOV BL, 03H
4        CMP AL, BL
5        JE ESIT
6        JMP ESIT_DEGIL
7        INT 03H
8  ESIT:
9        MOV CX, 0FFFFH
10       INT 03H
11 ESIT_DEGIL:
12       MOV CX, 0000H
13       INT 03H
```

Yukarıdaki örnek programın başında AL ve BL yazmaçlarının değerleri deneme amacıyla 03H değerine eşitlenmiştir. 4üncü satırdaki CMP komutu AL ve BL yazmaçlarının içerdiği değerleri karşılaştırır. CMP komutunun sonucunun hesaplanması için işlemci çıkarma işlemi yapar ve sonucun sıfır olup olmadığını ZF bayrak bitinde saklar. AL ve BL yazmaçlarının değerinin aynı olması durumunda ZF bayrak biti 0 değerini alır ve 5inci satırdaki JE komutu programın 8inci satırdaki ESİT etiketine atlamasını ve CX yazmacının FFFFH

değerini almasını sağlar. Yazmaçların değerinin aynı olmaması durumunda program 6ıncı satırı işler ve JMP komutu koşulsuz olarak ESIT_DEGIL etiketine atlayarak CX yazmacının 0000H değerini almasını sağlar. Örnek programın başına AL ve BL yazmaçlarının değerleri eşitlendiği için programın çalışması bittiğinde CX yazmacının değeri FFFFH olacaktır.

4.5.6.5. JL/JNGE

JL (**j**ump if **l**ess – az ise atla) ve JNGE (jump if not greater and equal – eşit ve büyük değilse atla) komutları SF ve OF bayrak bitlerinin değerlerinin aynı olmaması durumunda *işlenen1*'de belirtilen adrese atlar. CMP komutu kullanılırken, birinci işlenen ikinci işlenenden küçükse SF ve OF bayrak bitlerinin değeri eşitlenir. Böylece programın CMP komutunun ardından gelecek bir JL komutunun belirttiği adrese atlaması sağlanabilir. İşlenen olarak bir etiket ya da adres alan JL ve JNGE komutlarının işlev açısından hiçbir farkları yoktur.

JL *işlenen1*
JNGE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JL/JNGE	01111100	Uzaklık Verisi	-	-

Örnek: Aşağıda JL komutunun program içinde nasıl kullanıldığı gösterilmiştir.

```
1          ORG 0100H
2          MOV AL, 03H
3          MOV BL, 04H
4          CMP AL, BL
5          JL KUCUK
6          MOV CX, 0000H
7          INT 03H
8  KUCUK:
9          MOV CX,0FFFFH
10         INT 03H
```

Yukarıdaki örnek programda 4üncü satırdaki CMP komutu AL ve BL yazmaçlarının değerlerini karşılaştırarak SF ve OF bayrak bitlerinin değerlerini değiştirir. 5inci satırdaki JL komutu AL yazmacının değerinin BL yazmacının değerinden daha az olması durumunda programın KUCUK etiketine atlamasını sağlar.

4.5.6.6. JLE/JNG

JLE (**j**ump if **l**ess or **e**qual – az veya eşit ise atla) ve JNG (**j**ump if **n**ot **g**reater – büyük değilse atla) komutları SF ve OF bayrak bitleri birbirlerine eşit değilse veya ZF bayrak biti 1 ise atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçük veya eşitse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JLE *işlenen1*
JNG *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JLE/JNG	01111110	Uzaklık Verisi	-	-

```
1          ORG 0100H
2          MOV AL, 03H
3          MOV BL, 04H
4          CMP AL, BL
5          JLE KUCUK_ESIT
6          MOV CX, 0000H
7          INT 03H
8  KUCUK_ESIT:
9          MOV CX, 0FFFFH
10         INT 03H
```

4.5.6.7. JB/JNAE

JB (**j**ump if **b**elow – az ise atla) ve JNAE (**j**ump if **n**ot **a**bove and not **e**qual – büyük ve eşit değilse atla) komutları CF bayrak biti 1 olduğu durumda atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçükse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JB *işlenen1*
JNAE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JB/JNAE	01110010	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL,03H
3         MOV BL,04H
4         CMP AL,BL
5         JB KUCUK
6         MOV CX,0000H
7         INT 03H
8   KUCUK:
9         MOV CX,0FFFFH
10        INT 03H
```

4.5.6.8. JBE/JNA

JBE (**j**ump if **b**elow or **e**qual – az veya eşit ise atla) ve JNA (**j**ump if **n**ot **a**bove – büyük değilse atla) komutları CF bayrak biti 1 veya ZF bayrak biti 1 olduğu durumda atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçük veya eşitse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JBE *işlenen1*
JNA *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JBE/JNA	01110110	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, 03H
3         MOV BL, 04H
4         CMP AL, BL
5         JBE KUCUK_ESIT
6         MOV CX, 0000H
7         INT 03H
8   KUCUK_ESIT:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.9. JP/JPE

JP (**j**ump if **p**arity – eşlik 1 ise atla) ve JPE (**j**ump if **p**arity bit **e**ven – eşlik çift ise atla) komutları PF bayrak biti 1 olduğu durumda atlar. İşlenen olarak bir etiket kullanılabileceği gibi bir adres de kullanılabilir.

JP *işlenen1*
JPE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JP/JPE	01111010	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, 00010100B
3         MOV BL, 00000000B
4         CMP AL, BL
5         JP ESLIK_CIFT
```

```
6         MOV CX, 0000H
7         INT 03H
8   ESLIK_CIFT:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.10. JO

JO (**j**ump if **o**verflow – taşma var ise atla) komutu OF bayrak biti 1 ise atlar. İşlenen olarak bir etiket kullanılabileceği gibi bir adres de kullanılabilir.

JO *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JO	01110000	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, -125
3         MOV BL, 4
4         SUB AL, BL
5         JO TASMA_VAR
6         MOV CX, 0000H
7         INT 03H
8   TASMA_VAR:
9         MOV CX, 0FFFFH
10        INT 03H
```

Yukarıdaki örnekte -125 – 4 = -129 işlemi taşma bitini 1 yapmıştır çünkü sonucun -128 ile 127 arasında olması gerekmektedir.

4.5.6.11. JS

JS (**j**ump if **s**igned – işaretli ise atla) komutu SF bayrak biti 1 olduğu zaman atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JS *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JS	01111000	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, -125
3         MOV BL, 0H
4         CMP AL, BL
5         JS NEGATIF
6         MOV CX, 0000H
7         INT 03H
8   NEGATIF:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.12. JNE/JNZ

JNE (**j**ump if **n**ot **e**qual – eşit değilse atla) ve JNZ (**j**ump if **n**ot **z**ero – sıfır değilse atla) komutları ZF bayrak biti 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JNE *işlenen1*
JNZ *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNE/JNZ	01110101	Uzaklık Verisi	-	-

```
1         ORG 0100H
```

```
2         MOV AL, 03H
3         MOV BL, 04H
4         CMP AL, BL
5         JNE ESIT_DEGIL
6         MOV CX, 0000H
7         INT 03H
8   ESIT_DEGIL:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.13. JNL/JGE

JNL (**j**ump if **n**ot **l**ower – az değilse atla) ve JGE (**j**ump if **g**reater or **e**qual – eşit veya büyük ise atla) komutları SF ve OF bayrak bitleri birbirine eşit ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir. Yani CMP komutunun sonucu olarak birinci işlenen ikinci işlenenden büyük veya eşit ise JNL komutu atlar.

JNL *işlenen1*
JGE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNL/JGE	01111101	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, 02H
3         MOV BL, 01H
4         CMP AL, BL
5         JNL BUYUK_ESIT
6         MOV CX, 0000H
7         INT 03H
8   BUYUK_ESIT:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.14. JNLE/JG

JNLE (**j**ump if **n**ot **l**ower and not **e**qual – eşit ve küçük değilse atla) ve JG (**j**ump if **g**reater –büyük ise atla) komutları SF ve OF bayrak bitleri birbirine eşit ve ZF bayrak biti 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir. Yani CMP komutunun sonucu olarak birinci işlenen ikinci işlenenden büyük ise JNL komutu atlar.

JNLE *işlenen1*
JG *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNLE/JG	01111111	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, 02H
3         MOV BL, 01H
4         CMP AL, BL
5         JNL BUYUK
6         MOV CX, 0000H
7         INT 03H
8   BUYUK:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.15. JNB/JAE

JNB (**j**ump if **n**ot **b**elow – az değilse atla) ve JAE (**j**ump if **a**bove or **e**qual – büyük veya eşit ise atla) komutları CF bayrak biti 0 ise atlar. Yani CMP komutunun sonucunda birinci işlenen ikinci işlenenden küçük değilse bu durum oluşur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JNB *işlenen1*
JAE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNB/JAE	01110011	Uzaklık Verisi	-	-

```
1          ORG 0100H
2          MOV AL, 02H
3          MOV BL, 01H
4          CMP AL, BL
5          JNL BUYUK_ESIT
6          MOV CX, 0000H
7          INT 03H
8  BUYUK_ESIT:
9          MOV CX, 0FFFFH
10         INT 03H
```

4.5.6.16. JNBE/JA

JNBE (**j**ump if **n**ot **b**elow and not **e**qual – eşit ve az değilse atla) ve JA (**j**ump if **a**bove - büyük ise atla) komutları ZF ve CF bayrak bitleri 0 ise atlar. Yani CMP komutunun sonucu olarak birinci işlenen ikinci işlenenden büyükse atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JNBE *işlenen1*
JA *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNBE/JA	01110111	Uzaklık Verisi	-	-

```
1          ORG 0100H
2          MOV AL, 02H
3          MOV BL, 01H
4          CMP AL, BL
5          JNL BUYUK
6          MOV CX, 0000H
7          INT 03H
8  BUYUK:
9          MOV CX, 0FFFFH
10         INT 03H
```

4.5.6.17. JNP/JPO

JNP (**j**ump if **n**ot **p**arity – eşlik 0 ise atla) ve JPO (**j**ump if **p**arity **o**dd – eşlik tek ise atla) komutları PO bayrak biti 0 olduğunda atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JNP *işlenen1*
JPO *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNP/JPO	01111011	Uzaklık Verisi	-	-

```
1          ORG 0100H
2          MOV AL, 00011100B
3          MOV BL, 00000000B
4          CMP AL, BL
5          JNP ESLIK_TEK
6          MOV CX, 0000H
```

```
7         INT 03H
8   ESLIK_TEK:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.18. JNO

JNO (**j**ump if **n**ot **o**verflow – taşma yok ise atla) komutu OF 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JNO *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNO	01110001	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL,-125
3         MOV BL, 4
4         SUB AL, BL
5         JNO TASMA_YOK
6         MOV CX, 0000H
7         INT 03H
8   TASMA_YOK:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.19. JNS

JNS (**j**ump if **n**ot **s**igned – işaretsiz ise atla) komutu SF bayrak biti 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bi adres de girilebilir.

JNS *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JNS	01111001	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV AL, -125
3         MOV BL, 0H
4         CMP AL, BL
5         JS POZITIF
6         MOV CX, 0000H
7         INT 03H
8   POZITIF:
9         MOV CX, 0FFFFH
10        INT 03H
```

4.5.6.20. LOOP

LOOP (**loop** – döngü) komutu CX yazmacının değeri kadar islenen1'in adresine atlar. CX her atlamada bir azaltılır. CX yazmacının değeri 0 olduğunda atlanmaz ve işlemlere bir sonraki komuttan devam edilir. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

LOOP *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
LOOP (CX kere)	11100010	Uzaklık Verisi	-	-

```
1         ORG 0100H
2         MOV CX, 0005H
3         MOV BX, 0000H
4   YINELE:
```

```
5       INC BX
6       LOOP YINELE
7       INT 03H
```

4.5.6.21. LOOPZ/LOOPE

LOOPZ (**l**oop if **z**ero – sıfır ise atla) ve LOOPE (**l**oop if **e**qual – eşit ise atla) komutları CX yazmacının değeri 0'a eşit olmadığı ve ZF bayrak biti 1 olduğu sürece döngü devam eder. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

LOOPZ *işlenen1*
LOOPE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
LOOPZ/LOOPE	11100001	Uzaklık Verisi	-	-

```
1       ORG 0100H
2       MOV CX, 0005H
3       MOV BX, 0000H
4   YINELE:
5       INC BX
6       LOOPZ YINELE
7       INT 03H
```

4.5.6.22. LOOPNZ/LOOPNE

LOOPNZ (**loop** if **n**ot **z**ero – sıfır değilse atla) ve LOOPNE (**loop** if **n**ot **e**qual – eşit değilse atla) komutları CX yazmacının değeri 0'a eşit olmadığı ve ZF bayrak biti 0 olduğu sürece döngü devam eder. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

LOOPNZ *işlenen1*
LOOPNE *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
LOOPNZ/LOOPNE	11100000	Uzaklık Verisi	-	-

```
1       ORG 0100H
2       MOV CX, 0005H
3       MOV BX, 0000H
4   YINELE:
5       INC BX
6       LOOPNZ YINELE
7       INT 03H
```

4.5.6.23. JCXZ

JCXZ komutu CX yazmacı 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.

JCXZ *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
JCXZ	11100011	Uzaklık Verisi	-	-

```
1       MOV BX, 0000H
2       JCXZ ATLA1
3       SUB CX, 05H
4       JCXZ ATLA2
5   ATLA1:
6       MOV BX, 0FFFFH
7       INT 03H
8   ATLA2:
9       MOV BX, 0000H
```

10	INT 03H

4.5.6.24. INT

INT (**int**errupt – kesme) komutu işlenen olarak bir bayt alır. Bu bayt INT komutunun tipidir. INT komutu işlendiği zaman bayrak yazmaçları, CS kesim yazmacı ve IP yazmacı yığına kaydedilir. IF bayrak yazmacı sıfırlanır. Kesme yordamına kontrol devredilir.

INT *işlenen1*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
Tip belirtilen	11001101	Tip	-	-
Tip 3	11001100	-	-	-

1	ORG 0100H
2	MOV BX, 1234H
3	INT 03H

4.5.6.25. INTO

INTO komutu OF bayrak biti 1 ise INT 04H görevi görür.

INT0

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
INTO	11001110	-	-	-

1	ORG 0100H
2	MOV AL, -10
3	SUB AL, 127
4	INTO
5	INT 03H

4.5.6.26. IRET

IRET komutu kesme yordamından geri dönmeyi sağlar. IP, CS ve bayrak bitlerini yığından geri alır.

IRET

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
IRET	11001111	-	-	-

4.5.7. İşlemci Denetimi Komutları

4.5.7.1. CLC

CLC (**cl**ear **C**F, CF bayrak bitini sıfırla) komutu CF bayrak bitini 0 yapar.

CLC

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CLC	11111000	-	-	-

4.5.7.2. CMC

CMC (**co**m**m**plement **C**F- CF bayrak bitini ters çevir) komutu CF bayrak bitini ters çevirir (CF'nin değeri 1 ise değerini 0, değeri 0 ise değerini 1 yapar).

CMC

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CMC	11110101	-	-	-

4.5.7.3. STC

STC (**se**t **C**F – CF bayrak bitini 1 yap) komutu CF bayrak bitini 1 yapar.

STC

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
STC	11111001	-	-	-

4.5.7.4. CLD

CLD (**cl**ear **D**F – DF bayrak bitini 0 yap) komutu DF bayrak bitini 0 yapar.

CLD

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CLD	11111100	-	-	-

4.5.7.5. STD

STD (**se**t **D**F – DF bayrak bitini 1 yap) komutu DF bayrak bitini 1 yapar.

STD

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
STD	11111101	-	-	-

4.5.7.6. CLI

CLI (**cl**ear **I**F – IF bayrak bitini 0 yap) komutu IF bayrak bitini 0 yapar. Bu bitin sıfırlanmasıyla donanım kesmeleri devre dışı bırakılmış olur.

CLI

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
CLI	11111010	-	-	-

4.5.7.7. STI

STI (**se**t **I**F – IF bayrak bitini 1 yap) komutu IF bayrak bitini 1 yapar. Donanım kesmeleri devrede olur.

STI

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
STI	11111011	-	-	-

4.5.7.8. HLT

HLT (**h**a**lt** – dur) komutu sistemin durdurulmasını sağlar.

HLT

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
HLT	11110100	-	-	-

4.5.7.9. WAIT

WAIT (**wait** – bekle) komutu yardımcı işlemci işlemini bitirene kadar işlemciyi beklemeye alır. Bu şekilde yardımcı işlemci tarafından erişilmekte olan belleğe işlemcinin erişmeye çalışması engellenmiş olur.

WAIT

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
WAIT	10011011	-	-	-

4.5.7.10. ESC

ESC (**escape** – çık) komutu diğer işlemcilere veriyoluna erişme hakkı tanır. İşlemci tarafından bir NOP komutu gibi algılanır ama bellek işlenenini bellek yoluna verir.

ESC *anlık veri*, *yazmac* **ESC** *anlık veri, bellek*

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
ESC	11011XXX*	MOD XXX R/M	-	-

* X = Keyfi değer alabilir

4.5.7.11. LOCK

LOCK (**lock** – kitle) komutu sayesinde iki farklı işlemcinin aynı anda bir bellek bölgesini değiştirmesini engeller. XCHG, MOV, IN ve OUT komutları kullanılmadan önce eğer birden fazla işlemci kullanılıyorsa kullanılması gerekir.

LOCK

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
LOCK	11110000	-	-	-

4.5.7.12. NOP

NOP (**n**o **op**eration – işlem yok) komutu XCHG AL, AL komutu ile aynı işlemi yapar. NOP komutu sonucunda işlemcinin içinde, yazmaçların durumunda ya da bellekte herhangi bir değişiklik olmaz ancak işlemci 3 saat vuruşu boyunca başka işlem yapamaz. NOP komutu işlemcinin hiçbir işlem yapmadan gecikmesinin gerektiği durumlarda kullanılır.

NOP

Tanım	Kod 1	Kod 2	Kod 3	Kod 4
NOP	10000110	11000000	-	-

Yukarıdaki tablodaki değerler incelendiğinde işlem ve adres kodunun XCHG AL, AL komutunun işlem ve adres koduyla aynı olduğu görülür.

5. ÇEVİRİCİ DİL İLE MATEMATİK VE MANTIK İŞLEMLERİ

Bu bölümde assembly dilinin matematiksel ve mantıksal işlemlerde kullanımını birkaç örnek ile göstermeye çalışacağız. İlk olarak işaretsiz sayılar ile matematiksel işlemlere değineceğiz. Ardından işaretli sayılar ile matematiksel işlemleri inceleyeceğiz ve son olarak da mantıksal işlemleri inceleyeceğiz.

5.1. İşaretsiz Sayılar İle Matematiksel İşlemler

5.1.1. Toplama

Aşağıdaki kodu FASM çevirici ile çevirip DEBUG programı ile deneyelim.

```
1  ORG 0100H
2  MOV AX,01H
3  MOV BX,00000010B
4  ADD AX,BX
5  MOV CX,3
6  ADD BX,CX
7  INT 03H
```

Programı çalıştırdıktan sonra yazmaçlar aşağıdaki değerlere sahip olacaktır:

Yazmaç	Değer
AX	0003H
BX	0005H
CX	0003H

Yukarıdaki kodu inceleyelim:

Birinci satırdaki komut ile kodumuzdan bir .COM dosyası oluşturmak istediğimizi belirttik. 0100H yazmamızın nedeni .COM dosyalarının 0100H göreli adresinden başlamasıdır. Bu değer DOS tarafından belirlenmiştir.

İkinci satırdaki kod ile AX yazmacına onaltılık tabandaki 01 değeri kaydedilmiştir.

Üçüncü satırda BX yazmacına ikili 00000010 değeri yazılmıştır.

Dördüncü satırdaki *ADD AX,BX* komutu ile AX ve BX yazmaçlarının değerlerinin toplanıp AX yazmacına kaydedilmesi gerektiği belirtilmiştir. Kodu incelersek bu komut çağırılmadan önce AX yazmacında 1, BX yazmacında da 2 değerinin olduğunu göreceğiz. *ADD AX,BX* komutu işlendikten sonra toplam olan 3 değeri AX yazmacına kaydedilmiş olacaktır.

Beşinci satırdaki komut ile CX yazmacına ondalık 3 değeri kaydedilmiştir.

Altıncı satırda ise BX ve CX yazmaçlarının değerleri toplanıp sonuç BX yazmacına yazılmıştır. Bu komut işlendikten sonra BX yazmacının değeri ondalık 5 değerine eşit olacaktır.

Yedinci satırda ise INT 03H kesmesi çağırılmış ve programımız durdurulmuştur.

Yukarıdaki örnekte bayt boyutunda verileri topladık. Sıradaki örnekte kelime boyutundaki verileri toplayalım.

```
1  ORG 0100H
2  MOV AX,0101H
3  MOV BX,0000111100000010B
4  ADD AX,BX
5  MOV CX,300
```

```
6  ADD BX,CX
7  INT 03H
```

Program çalıştırıldıktan sonra yazmaçların yeni değerleri şunlar olacaktır:

Yazmaç	Değer
AX	1003H
BX	102EH
CX	012CH

Birinci satırdaki komut ile kodumuzdan bir .COM dosyası oluşturmak istediğimizi belirttik.

İkinci satırdaki kod ile AX yazmacına onaltılık tabandaki 0101 değeri kaydedilmiştir.

Üçüncü satırda BX yazmacına ikili 0000111100000010 değeri yazılmıştır.

Dördüncü satırdaki *ADD AX,BX* komutu ile AX ve BX yazmaçlarının değerlerinin toplanıp AX yazmacına kaydedilmesi gerektiği belirtilmiştir.

Beşinci satırdaki komut ile CX yazmacına ondalık 300 değeri kaydedilmiştir.

Altıncı satırda ise BX ve CX yazmaçlarının değerleri toplanıp sonuç BX yazmacına yazılmıştır.

Yedinci satırda ise INT 03H kesmesi çağırılmış ve programımız durdurulmuştur.

ADD komutu ile toplama işlemi yapma hakkında yeterli bilgi edindik. Yukarıdaki örneklerde baytları ve kelimeleri birbirleriyle topladık. Şimdi daha karmaşık bir toplama işlemi yapalım. İki tane ikili kelimeyi toplayalım. ADD komutu işlenen olarak en fazla bir kelime alabilir. İkili kelimeleri (veya daha uzun sayıları) toplayabilmek için ne yapabileceğimizi görelim.

2 tane ikili kelimemiz olsun:

01234567H
1234F679H

Şimdi bu sayıları kağıt üzerinde toplayalım.

0	1	2	3	4	5	6	7
							+
1	2	3	4	F	6	7	9
							=
1	3	5	8	3	B	E	0

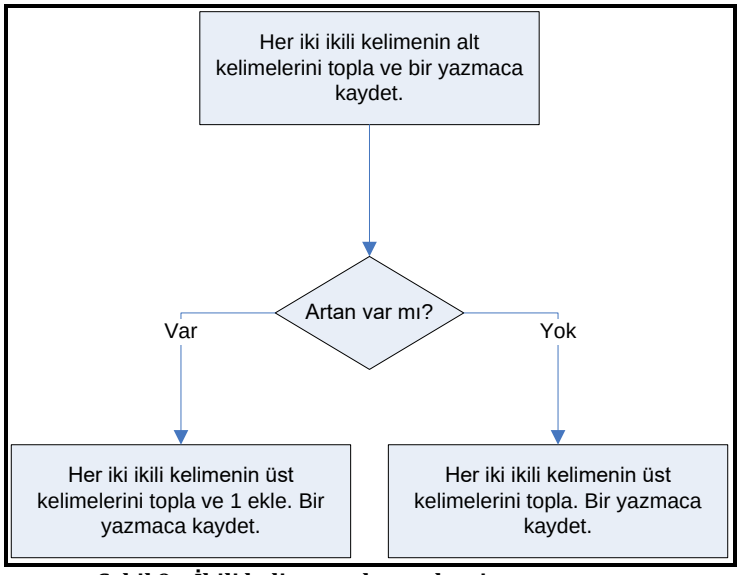
Dikkat edilirse kağıt üzerinde toplama işlemi yapılırken her basamak ayrı ayrı toplandı ve artan var ise bir sonraki basamağa aktarıldı. Aynı mantığı kodumuzda da uygulamamız mümkün. Sayıları basamak basamak ayırmak yerine ADD komutunun işlem yapabildiği en büyük veri boyutu olan kelimelere ayıralım. Sayıları aşağıdaki şekilde kelimelere ayıralım.

0	1	2	3
1	2	3	4

4	5	6	7
F	6	7	9

İlk önce her ikili kelimenin 2. kelimesini toplayacağız ve bir yazmaca kaydedeceğiz. Ardından eğer ilk toplama işleminden artan var ise 1. kelimeleri toplarken bu artanı 1. kelimelere ekleyip farklı bir yazmaca kaydedeceğiz. Bu işlemler için ADD komutunun yerine ADC komutunu kullanacağız. ADC komutu ADD ile aynı işlevleri yerine getirmekle beraber CF bayrak biti 1 olduğunda sonuca 1 ekler. CF bayrak bitinin 1 olması toplama işleminden artan olduğunu gösterir.

Programımızın algoritması ana hatlarıyla aşağıdaki şekildedir.



Şekil 9 – İkili kelime toplama algoritması

Aşağıda yukarıda anlatılan işlemleri yapan program verilmiştir.

```
1   ORG 0100H
2   JMP BASLA
3   DB 23H,01H,67H,45H
4   DB 34H,12H,79H,0F6H
5   BASLA: MOV SI,0104H
6   MOV AX,WORD [SI]
7   MOV SI,0108H
8   MOV BX,WORD [SI]
9   ADD AX,BX
10  DEC SI
11  DEC SI
12  MOV BX,WORD [SI]
13  MOV SI,0102H
14  MOV CX,WORD [SI]
15  ADC CX,BX
16  XCHG AX,BX
17  XCHG AX,CX
18  INT 03H
```

İkinci satırdaki *JMP BASLA* kodu *BASLA* etiketine atlanmasına neden olmuştur.

Üçünü ve dördüncü satırlarda toplayacağımız ikili kelimeler tanımlanmıştır. Her kelime Little Endian Kuralı'na uygun olacak şekilde kaydedilmiştir. Bu şekilde MOV AX,WORD [SI] kodu çalıştırıldığında istediğimiz kelime AX yazmacına kaydedilecektir.

Beşinci satırda SI yazmacına 0104H yani ilk ikili kelimenin ikinci kelimesinin adresi kaydedilmiştir. Altıncı satırda ise bu adresteki veriler AX yazmacına aktarılmaktadır.

Dokuzuncu satırda AX ile BX yazmaçları toplanmış ve sonuç AX yazmacına yazılmıştır. Neden ADC komutu değil de ADD komutu kullandık? ADD komutu kullanmamızın nedeni AX ile BX toplama işlemi öncesi bizim işlemimizi ilgilendiren herhangi bir artan olamamasıdır. Ek A'daki ADC ve ADD komutlarının işlenme sürelerine bakılacak olursa performans açısından ADC yerine ADD kullanılmasının herhangi bir yararının olmadığı, ikisinin de 3 işlem döngüsünde işlendiği, görülecektir. Yukarıdaki kodda ADC kullanılması herhangi bir değişikliğe yol açmayacaktır ama başka bir programa bu kodu ilave ederken ADD yerine ADC kullanılırsa

CF bayrak bitinin toplama işlemi öncesinde 1 olup olmadığına bakılıp sıfırlanması gerekmektedir aksi takdirde fazladan bir değer çıkacaktır.

Onuncu ve on birinci satırlarda SI yazmacının değeri 2 azaltılmıştır. Bunun yerine MOV komutu ile SI yazmacına bir değer verilebilirdi ama performans açısından burada DEC SI kullanmak daha doğrudur.

On beşinci satırda ADC komutu kullanılarak bir önceki ADD'den artan olduysa bu artan toplama işlemine eklenmektedir. Bu şekilde sonuç ikili kelimesinin ilk kelimesine ikinci kelimeden gelen artan eklenmektedir.

On altıncı ve on yedinci satırlardaki XCHG komutu ile AX ve CX yazmacında bulunan sonuçlar ilk kelime AX'de ikinci kelime BX'de olacak şekilde taşınmıştır.

Yani AX yazmacında sonuçta 1358H ve BX yazmacında 3BE0H değeri bulunacaktır. Elde etmemiz gereken sonucu bu şekilde elde etmiş olduk.

5.1.2. Çıkarma

Aşağıdaki kodu FASM ile çevirip DEBUG programı ile deneyelim.

```
1  ORG 0100H
2  MOV AX,05H
3  MOV BX,02H
4  SUB AX,BX
5  MOV CX,05H
6  SUB BX,CX
7  INT 03H
```

Sonuçta AX,BX ve CX yazmaçlarının değerleri aşağıdaki tablodaki gibi olacaktır.

Yazmaç	Değer
AX	0003H
BX	FFFDH
CX	0005H

Dördüncü satırda AX yazmacının değerinden BX yazmacının değeri çıkartılmış ve sonuç 03H bulunmuştur. Ardından 6. satırda BX yazmacının değerinden CX yazmacının değeri çıkartılmıştır ve sonuç -3 yani FFFDH çıkmıştır.

Bu örneğin ardından ikili kelimelerin birbirinden çıkarılması konusunu inceleyelim.

2 tane ikili kelimemiz olsun:

12345679H
012FFF67H

Şimdi bu sayıları kağıt üzerinde çıkartalım.

1	2	3	4	5	6	7	9
							-
0	1	2	F	F	F	6	7
							=
1	1	0	4	5	7	1	2

Toplama işlemi için oluşturduğumuz mantığı çıkarma için uygulayalım. İkili kelimeleri kelimelere ayıralım ve ayrı ayrı birbirlerinden çıkartalım. Çıkartma işlemini yaparken bir üst basamaktan gerektiğinde yararlanabilmemiz için SBB komutunu kullanabiliriz.

```
1  ORG 0100H
2  JMP BASLA
3  DB 34H,12H,79H,56H
4  DB 2FH,01H,67H,0FFH
```

```
5   BASLA: MOV SI,0104H
6   MOV AX,WORD [SI]
7   MOV SI,0108H
8   MOV BX,WORD [SI]
9   SUB AX,BX
10  DEC SI
11  DEC SI
12  MOV BX,WORD [SI]
13  MOV SI,0102H
14  MOV CX,WORD [SI]
15  SBB CX,BX
16  XCHG AX,BX
17  XCHG AX,CX
18  INT 03H
```

Çıkarma bir bakıma bir toplama işlemi olduğundan sadece ADD komutunun yerine SUB ve ADC komutunun yerine SBB komutunu koyarak ikili kelime çıkartabilen bir program yazmış olduk.

5.1.3. Çarpma

Çarpma işlemleri için MUL komutunu kullanabiliriz. MUL komutuna bakılacak olursa işlenen olarak bir bayt girildiğinde bu baytı AL yazmacı ile çarpıp sonucu AX yazmacına yazdığı görülecektir. Eğer işlenen bir kelime ise AX yazmacı ile bu kelime çarpılacak ve sonuç DX AX e kaydedilecektir.

Öncelikle bir bayt ile bir baytı çarpalım:

```
1   ORG 0100H
2   MOV AL,03H
3   MOV BL,04H
4   MUL BL
5   INT 03H
```

Yukarıdaki program 03H (3) sayısı ile 04H (4) sayısını çarpıp sonucu AX yazmacına kaydetmektedir. Programı DEBUG programı ile denersek AX yazmacında 0C (12) sayısının kayıtlı olduğunu göreceğiz.

Şimdide 2 tane kelimeyi birbirleriyle çarpalım:

```
1   ORG 0100H
2   MOV AX,0F234H
3   MOV BX,0FEFEH
4   MUL BX
5   INT 03H
```

Yukarıdaki programı DEBUG ile denersek F234H ve FEFEH sayılarının çarpımının F13FE798H değerine eşit olduğu görülecektir. Sonucun ilk 4 hanesi DX yazmacına, son 4 hanesi ise AX yazmacına kaydedilmiştir.

12345679H ve 012FFF67H ikili kelimelerimiz olsun. Bu iki ikili kelimeleri kağıt üzerinde çarpalım.

1	2	3	4	5	6	7	9
							x
0	1	2	F	F	F	6	7
							=

						7	F	6	E	5	D	4	F		
					6	D	3	A	0	6	D	6			
				1	1	1	1	1	1	1	1	7			
			1	1	1	1	1	1	1	1	7				
		1	1	1	1	1	1	1	1	7					
		2	4	6	8	A	C	F	2						
	1	2	3	4	5	6	7	9							
0	0	0	0	0	0	0	0								
													+		
0	1	5	9	E	1	B	C	E	6	8	5	1	A	F	

Dikkat edilirse her basamak ayrı ayrı çarpıldı ve sonuç her basamak için bir ikili kelime veya üçlü kelime boyutunda (9hane) oldu. FFFFFFFF ikili kelimesi ile F'in çarpımı 9 haneli olacaktır bu yüzden bize 3 kelimelik bir bellek alanı yeterlidir.

Yazacağımız program birinci ikili kelimenin ikinci kelimesini ikinci ikili kelimenin ilk kelimesi ile çarpıp sonucu geçici olarak 0202H göreli adresine kaydedecektir. Ardında birici ikili kelimenin ilk kelimesini ikinci ikili kelimenin ilk kelimesi ile çarpacak ve hane sırasını da gözeterek 0202H deki sayı ile toplayıp 0200H'a yazacak. Aynı işlemler ikinci kelimenin ilk kelimesi için de tekrarlanacaktır (veriler bu sefer de 0210H'a kaydedilecektir). Ardından 0200H ve 0210H deki sayılar haneler göz önünde bulundurularak toplanacak ve sonuç 0230H adresine yazılacaktır. Aşağıda yapılan işlemler gösterilmiştir.

1- 5679H x FF67H -> 564551AFH (0202H'a kaydet)

0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	020A	020B
00	00	45	56	AF	51	00	00	00	00	00	00

2- 1234H x FF67H -> 12291EECH
(DX yazmacında 1229H, AX yazmacında 1EECH değeri bulunacaktır.)

3- 1EECH ile 0202H adresindeki kelimeyi topla sonucu 0202H'a yaz.

4- 1291H ile bir önceki toplama işleminden artan var ise topla ve 0200H adresine yaz.

0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	020A	020B
29	12	31	75	AF	51	00	00	00	00	00	00

5- 5679H x 012FH -> 00665937H (0212H'a kaydet)

0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	021A	021B
00	00	66	00	37	59	00	00	00	00	00	00

6- 1234H x 012FH -> 00158B8CH
(DX yazmacında 0015H, AX yazmacında 8B8CH değeri bulunacaktır.)

7- 8B8CH ile 0212H adresindeki kelimeyi topla sonucu 0212H'a yaz.

8- 0015H ile bir önceki toplama işleminden artan var ise topla ve 0210H adresine yaz.

0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	021A	021B
15	00	F2	8B	37	59	00	00	00	00	00	00

9- 0200H ve 0210H adreslerine kaydedilmiş olan sayıları 0210H adresindekileri 2 bayt sola kaydırarak topla ve sonucu 0230H adresine yaz.

0	0	0	0	0	1	2	2	9	7	5	3	1	5	1	A	F
																+
0	0	0	1	5	8	B	F	2	5	9	3	7	0	0	0	0
																=
0	0	0	1	5	9	E	1	B	C	E	6	8	5	1	A	F

Yukarıda anlattıklarımızı yapan program aşağıdadır:

```
1   ORG 0100H
2   JMP BASLA
3   DW 1234H,5679H
4   DW 012FH,0FF67H
5   DW 0000H
6   DW 0000H
7   DW 0000H
8   DW 0000H
9   DW 0000H
10  DW 0000H
11  ;ADIM 1 2 3 4
12  BASLA: MOV SI,0104H
13  MOV AX,[SI]
14  MOV SI,0108H
15  MOV BX,[SI]
16  MUL BX
17  MOV SI,0110H
18  MOV [SI],DX
19  MOV SI,0204H
20  MOV [SI],AX
21  MOV SI,0108H
22  MOV BX,[SI]
23  MOV SI,0102H
24  MOV AX,[SI]
25  MUL BX
26  MOV SI,0114H
27  MOV [SI],DX
28  MOV SI,0110H
29  MOV BX,[SI]
30  ADD AX,BX
31  MOV SI,0202H
32  MOV [SI],AX
33  ADC DX,0000H
34  MOV SI,0200H
35  MOV [SI],DX
36  ;ADIM 5 6 7 8
37  MOV SI,0104H
38  MOV AX,[SI]
39  MOV SI,0106H
40  MOV BX,[SI]
41  MUL BX
42  MOV SI,0110H
43  MOV [SI],DX
44  MOV SI,0214H
45  MOV [SI],AX
46  MOV SI,0106H
47  MOV BX,[SI]
48  MOV SI,0102H
49  MOV AX,[SI]
50  MUL BX
```

```
51 MOV SI,0114H
52 MOV [SI],DX
53 MOV SI,0110H
54 MOV BX,[SI]
55 ADD AX,BX
56 MOV SI,0212H
57 MOV [SI],AX
58 ADC DX,0000H
59 MOV SI,0210H
60 MOV [SI],DX
61 ;ADIM 9
62 MOV SI,0204H
63 MOV AX,[SI]
64 MOV SI,0236H
65 MOV [SI],AX
66 MOV SI,0202H
67 MOV AX,[SI]
68 MOV SI,0214H
69 MOV BX,[SI]
70 ADD AX,BX
71 MOV SI,0234H
72 MOV [SI],AX
73 MOV SI,0200H
74 MOV AX,[SI]
75 MOV SI,0212H
76 MOV BX,[SI]
77 ADC AX,BX
78 MOV SI,0232H
79 MOV [SI],AX
80 MOV SI,0210H
81 MOV AX,[SI]
82 ADC AX,0000H
83 MOV SI,0230H
84 MOV [SI],AX
85 INT 03H
```

Programı incelediğimizde sonuçların 0230H göreceli adresine yazıldığını göreceğiz. Programı denediğimizde 0230H adresinde 15 00 1B 9E 68 CE AF 51 değerlerinin bulunduğunu göreceğiz. Değerler 0230 adresine Little Endian Kuralı'na uygun olacak şekilde yazılmıştır. Yani gerçekte sonuç 00159E1BCE6851AFH dır. Bu sonuç 12345679H ve 012FFF67H değerlerinin çarpımının sonucudur.

5.1.4. Bölme

Bölme işlemleri için DIV komutunu kullanabiliriz. Toplama, çıkarma ve çarpmada yaptığımız gibi öncelikle bir baytı bir bayta bölelim. Ardından bir kelimeyi bir kelimeye bölelim.

56H (86) sayısını 03H (03) sayısına bölelim.

```
1 ORG 0100H
2 MOV AL,86H
3 MOV BL,03H
4 DIV BL
5 INT 03H
```

Yukarıdaki kodu FASM ile çevirip DEBUG ile denediğimizde AX yazmacında 021CH değerinin bulunduğunu göreceğiz. AH da bulunan 02H (2) kalan ve AL de bulunan 1CH (28) bölümdür.

Şimdide 5656H (22102) sayısını 0303H (771) sayısına bölelim.

```
1  ORG 0100H
2  MOV DX,0000H
3  MOV AX,5656H
4  MOV BX,0303H
5  DIV BX
6  INT 03H
```

Programı denediğinizde AX yazmacında 001CH (28) ve DX yazmacında 0202H (514) değerini göreceksiniz. AX yazmacındaki değer bölüm ve DX yazmacındaki değer kalandır.

Şimdi de aşağıdaki kodu deneyelim:

```
1  ORG 0100H
2  MOV AX,5656H
3  MOV BX,0000H
4  DIV BX
5  INT 03H
```

Kodu denediğinizde "Bölme Taşması" hatası ile karşılaşacaksınız. Bunun nedeni 0'a bölme yapılmasıdır. Programlarımızı yazarken 0 bölme konusunda dikkatli olmamız gerekiyor. Bu tür kural dışı bir durum bölüm yazmaca sığmazsa da oluşabilir. Aşağıdaki durumu inceleyelim:

```
1  ORG 0100H
2  MOV DX,0FFFFH
3  MOV AX,0FFFFH
4  MOV BX,0001H
5  DIV BX
6  INT 0
```

Yukarıdaki programı denediğinizde yine "Bölme Taşması" kural dışı hata mesajını alacaksınız. Bunun nedeni DX ve AX den bulunan ikili kelimeyi yani FFFFFFFF sayısını BX yazmacının değeri olan 1'e bölü FFFFFFFF bulmamız. Eğer bir önceki bölümde DIV komutunun tanımına bakarsanız bu durumda bölümün AX yazmacına kaydedileceğini görürsünüz ama FFFFFFFF sayısı AX yazmacına kaydedilemeyecek kadar büyük bir sayıdır. Bu yüzden "Bölme Taşması" hatası oluşur. "Bölme Hatası" oluştuğunda tip 0 kesmesi oluşur. Kesmelere ileriki bölümlerde detaylı olarak değineceğiz.

5.2. İşaretli Sayılar İle Matematik İşlemleri

İşaretli sayıları önceki bölümlerde teorik olarak anlatmıştık. Şimdi de işaretli sayılar üzerinde yapılabilecek işlemlerden birkaçını örnekleyelim.

-5 sayısının aşağıdaki şekilde kaydedildiğini görmüştük.

1	1	1	1	1	0	1	1
İşaret	**Büyüklük**						

İlk bit işaret biti. Ardından gelen 7 bit ise büyüklük bitleridir. Büyüklüğün buradaki değeri $1 + 2^1 * 1 + 2^3 * 1 + 2^4 * 1 + 2^5 * 1 + 2^6 * 1 = 1 + 2 + 8 + 16 + 32 + 64 = 123$ yani 7BH'dır.

İşaretli sayılarda pozitif sayıların büyüklüğü 0 ile 127 arasında, negatif sayılarda ise -1 ile -128 arasındadır.

Sayıları işaretli sayılara çevirmek için aşağıdaki adımları takip edebiliriz. -12 sayısını elde etmeye çalışalım.

1- 12 sayısı yazılır:

00001100

2- Her bit ters çevrilir:

11110011

3- 1 eklenir:

11110100

Şimdide yukarıda -5'i ifade eden sayıyı yani 11111011'İ çevirelim.

1- 1 çıkartılır:

11111010

2- Her bit ters çevrilir:

00000101

Elde ettiğimiz sayı 5'e eşittir.

Bu kısa hatırlatmayı yaptıktan sonra işaretli sayılar üzerinde yapılan aritmetik işlemleri örnekleyelim.

5.2.1. Toplama

İlk önce +92 ve +15 sayılarını toplayalım:

```
1  ORG 0100H
2  MOV AX,+92
3  MOV BX,+15
4  ADD AX,BX
5  INT 03H
```

+92 sayısını yazalım:

0	1	0	1	1	1	0	0

+15 sayısını yazalım:

0	0	0	0	1	1	1	1

Ve bu iki sayıyı toplayıp aşağıdaki sonucu elde edelim:

0	1	1	0	1	0	1	1

Elde ettiğimiz sayı +107 yani 6BH'dır. Yukarıdaki programı denediğinizde AL yazmacında 6BH değerinin olacağını göreceksiniz.

Şimdide -92 ve +15 sayılarını toplayalım:

```
```

```
1  ORG 0100H
2  MOV AX,+92
3  MOV BX,+15
4  ADD AX,BX
5  INT 03H
```

-92 sayısını yazalım:

1	0	1	0	0	1	0	0

+15 sayısını yazalım:

0	0	0	0	1	1	1	1

Ve bu iki sayıyı toplayıp aşağıdaki sonucu elde edelim:

1	0	1	1	0	0	1	1

Elde ettiğimiz sayı -77 yani B3H sayısını elde etmiş olduk.

Şimdi birde -15 ve -92 sayılarını toplayalım:

-92 sayısını yazalım:

1	0	1	0	0	1	0	0

-15 sayısını yazalım:

1	1	1	1	0	0	0	1

Ve bu iki sayıyı toplayıp aşağıdaki sonucu elde edelim:

1	0	0	1	0	1	0	1

Elde ettiğimiz sayı -107 yani 95H sayısıdır.

Negatif sayıların -1 ile -128 arasında olduğunu belirtmiştik. Peki -92 ile -92 sayılarını toplarsak ne olur? Bir örnek ile görelim:

```
1  ORG 0100H
2  MOV AL,-92
3  MOV BL,-92
4  ADD AL,BL
5  INT 03H
```

-92 sayısını yazalım:

1	0	1	0	0	1	0	0

Ve bunu -92 ile toplayalım:

0	1	0	0	1	0	0	0

Elde ettiğimiz sayı +72 yani 48H. Programı denediğinizde AL yazmacında 48H değerinin olduğunu göreceksiniz ve yukarıda yaptığımız işlemi sağlamış olacağız. Peki neden -184 yerine +72 buluyoruz? Doğru olan sonucu yani -184 sayısını nasıl bulacağız?

-184 yerine +72 bulmamızın nedeni büyüklük alanının 7 bitlik olması ve maksimum 127 değerindeki bir sayıyı kabul edebilmesi. Yani burada bir taşma durumu oluşmaktadır ve aslında OF bayrak biti 1 olmuştur. Kodumuzu OF bayrak biti için bir kontrol ekleyerek bir daha yazalım ve deneyelim:

```
1   ORG 0100H
2   MOV AL,-92
3   MOV BL,-92
4   ADD AL,BL
5   JO TASMA_VAR
6   JMP TASMA_YOK
7   TASMA_VAR: MOV DX,1111H
8   INT 03H
9   TASMA_YOK: MOV DX,0000H
10  INT 03H
```

Bu kodu denediğinizde DX yazmacına 1111H sayısının yazıldığını göreceksiniz yani TASMA_VAR etiketimize atlanmış. OF bayrak biti 1 olmuştur.

Burada düzgün bir hesap yapabilmek için 8 bitlik yazmaçlar yerine 16 bitlik yazmaçlar kullanmamız gerekmektedir. Kodu 16 bitlik yazmaçlar için yeniden yazıp deneyelim:

```
1   ORG 0100H
2   MOV AX,-92
3   MOV BX,-92
4   ADD AX,BX
5   JO TASMA_VAR
6   JMP TASMA_YOK
7   TASMA_VAR: MOV DX,1111H
8   INT 03H
9   TASMA_YOK: MOV DX,0000H
10  INT 03H
```

Kodu denediğimizde AX yazmacında FF48H değerinin olduğunu göreceğiz. Yani:

1	1	1	1	1	1	1	1	0	1	0	0	1	0	0	0

Sayının işaret biti yani soldan 0 ile başlayarak saydığımızda 15. bit 1'dir. Yani bu sayı negatif bir sayıdır. Sayının değerini aşağıdaki adımları takip ederek bulalım:

1- Sayıdan 1 çıkartalım:

1	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1

2- Her biti ters çevirelim:

0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0

Yukarıdaki 16 bitlik sayının değeri $2^3 * 1 + 2^4 * 1 + 2^5 * 1 + 2^7 * 1 = 8 + 16 + 32 + 128 = 184$. Yani 16 bitlik yazmaçlar kullanarak doğru olan sonucu -184'ü elde etmiş olduk.

Taşma hatasından kurtulmanın başka bir yolu ise CBW komutunu kullanarak sayıyı genişletmektedir. CBW komutu AL yazmacının 7. biti alıp AH yazmacının tüm bitlerine kopyalar ve AL yazmacındaki 8 bitinde büyüklük olarak kullanılabilmesini sağlar.

-92 ve -92 sayılarını bu yöntemi kullanarak toplayalım:

```
1  ORG 0100H
2  MOV AX,-92
3  CBW
4  ADD AX,AX
5  INT 03H
```

AX yazmacının değerine bakılacak olursa FF48H olduğu yani -184'e eşit olduğu görülecektir.

5.2.2. Çıkarma

92 sayısından 184 sayısını çıkaran programı yazıp sonucu irdeleyelim.

```
1  ORG 0100H
2  MOV AX,92
3  MOV BX,184
4  SUB AX,BX
5  INT 03H
```

Programı denediğinizde AX yazmacındaki sonucun FFA4H olduğunu göreceksiniz. -92 sayısı FFA4H sayısına eşittir. Yani beklenen sonucu elde ettik.

Şimdide -92 sayısından 92 sayısını çıkarıp sonucu inceleyelim:

```
1  ORG 0100H
2  MOV AL,-92
3  MOV BL,92
4  SUB AL,BL
5  INT 03H
```

AX yazmacındaki değerin 0048H olduğunu göreceksiniz. Bu değer ondalık 72 sayısına eşittir. Sonucumuzun -184 olması gerekiyordu ama 72 çıktı. Bu hatanın çıkmasının nedeni negatif sayıların -1 ile -128 arasında olmasıdır ve bizim -184 değeri ile bu sınırı aşmış olmamızdır.

Kodumuzu aşağıdaki şekilde düzeltirsek sorundan kurtulmuş olacağız:

```
1  ORG 0100H
2  MOV AL,-92
3  CBW
4  MOV BX,AX
5  MOV AL,92
6  CBW
7  SUB BX,AX
8  INT 03H
```

BX yazmacındaki sonucun FF48H yani -184 olduğu görülecektir.

5.2.3. Çarpma

İşaretli çarpma işlemleri için IMUL adında özel bir komut vardır. Bu komutu örnekleyelim:

```
1  ORG 0100H
2  MOV AL,-4
3  MOV BL,-2
4  IMUL BL
5  MOV BX,AX
6  MOV AL,-4
7  MOV CL,2
8  IMUL CL
9  MOV CX,AX
10 INT 03H
```

Programı denediğinizde yazmaç değerlerinin aşağıdaki tablodaki değerlere sahip olduklarını göreceksiniz:

Yazmaç	Değer
BX	0008H
CX	FFF8H

CX yazmacında -4 ile 2 sayısının çarpımının sonucu, BX yazmacında ise -4 ile -2 sayılarının çarpımının sonucu bulunmaktadır.

5.2.4. Bölme

İşaretli sayılar ile bölme işlemlerinde IDIV komutunu kullanabiliriz. IDIV komutunu işlenen olarak bir bayt alırsa farklı, bir kelime alırsa farklı çalışır. 4. Bölümde tanımlanan IDIV komutuna bakılacak olursa bölümün AL yazmacına, kalanın ise AH yazmacına kaydedildiği görülecektir. İşlenen olarak kelime kullanıldığında ise AX yazmacına bölüm, AX yazmacına ise kalan yazılmaktadır.

İlk olarak işlenen olarak bir bayt girelim ve sonuçları inceleyelim.

```
1  ORG 0100H
2  MOV AX,-18
3  MOV BL,-2
4  IDIV BL
5  INT 03
```

Bu örnekte -18 sayısını -2 sayısına böldük. Programı DEBUG ile denediğinizde AL yazmacında 09H (9) ve AH yazmacında 00H (0) değerlerinin olduğunu göreceksiniz. Yani kalan 0 ve bölüm 9'dur.

Şimdide işlenen olarak bir kelime girelim.

```
1  ORG 0100H
2  MOV DX,0000H
3  MOV AX,65535
4  MOV BX,-4660
5  IDIV BX
6  INT 03
```

Bu örnekte 65535 sayısını -4660 sayısına böldük. Program denendiğinde DX yazmacında 0127H ve AX yazmacında FFF2H değerlerinin bulunduğu görülecektir. DX yazmacındaki 0127H yani ondalık 295 sayısı kalan ve AX yazmacındaki FFF2H yani ondalık -14 sayısı da bölümdür. 0'a bölme hatası ve bölme taşmaları gibi durumlara DIV komutunda olduğu gibi IDIV komutunda da dikkat edilmelidir.

5.3. Mantık İşlemleri

Mantıksal işlemlerde kullanmak üzere elimizde bir çok komut vardır. Aşağıda elimizde bulunan komutlar listelenmiştir.

- NOT
- SHL
- SAL
- SHR
- SAR
- ROL
- ROR
- RCL
- RCR
- AND
- TEST
- OR
- XOR

Bu komutları dördüncü bölümde tanıtırken tek tek örneklemiştik. Şimdi bu komutları kullanarak bir kaç örnek yapalım.

İlk olarak 8 bitlik 2 sayıyı toplayabilen bir devre tasarlayalım ve bu devreyi tasarladıktan sonra bilgisayarımızda yazacağımız program ile sınayalım. Aşağıda 2 tane 1 bitlik sayının toplamının doğruluk tablosu verilmiştir.

X_0	Y_0	A_0	T_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

X_0: Birinci sayı 1. bit
Y_0: İkinci sayı 1. bit
A_0: İki sayının 1. bitlerinin toplamından artan
T_0: İki sayının 1. bitlerinin toplamı

Tabloyu C_0 için karnaugh haritasına aktarırsak:

X_0 / B_0	**0**	**1**
0	0	0
1	0	1

Yukarıdaki karnaugh haritasını kullanarak A_0 için aşağıdaki denklemi elde edebiliriz:

$A_0 = X_0$ **AND** Y_0

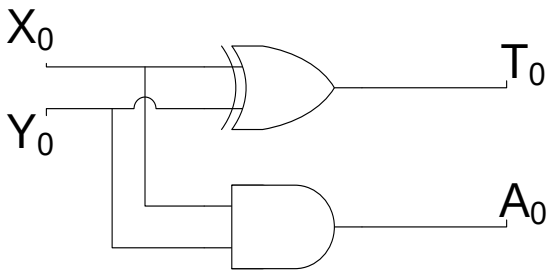
Ayını şekilde tabloyu T_0 için karnaugh haritasına aktarırsak:

X_0 / Y_0	**0**	**1**
0	0	1
1	1	0

Çizdiğimiz karnaugh haritasını kullanarak T_0 için aşağıdaki denklemi elde edebiliriz:

$T_0 = (X_0' \cdot Y_0) + (Y_0' \cdot X_0) = X_0$ **XOR** Y_0

Bu denklemleri kullanarak 1 bitlik 2 sayının toplamı için gerekli olan devreyi çizebiliriz:



Tasarladığımız bu devreyi artanları da hesaba katarak 16 bitlik hale getirmemiz gerekiyor. Bunun için X ve Y'nin yanında bir de bir önceki bit toplamından gelen artanın da toplamaya katılacağı bir mantıksal devrenin oluşturulması gerekiyor. Bu devre için gerekli doğruluk tablosunu oluşturalım:

A_n	X_n	Y_n	T_n	A_{n+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A_n: (n-2) inci bitten gelen artan
X_n: 1. sayı (n-1) inci bit
Y_n: 2. sayı (n-1) inci bit
T_n: 1. ve 2. sayının (n-1) inci bitleri ve (n-2) inci bitten gelen artanın toplamı
A_{n+1}: 1. ve 2. sayının (n-1) inci bitleri ve (n-2) inci bitten gelen artanın toplamından artan

T_n için karnaugh haritasını çizelim:

A_n / X_nY_n	**00**	**01**	**11**	**10**
0	0	1	0	1
1	1	0	1	0

Yukarıdaki karnaugh haritasını kullanarak T_n için aşağıdaki denklemi yazabiliriz:

$T_n = (X_n' \cdot Y_n' \cdot A_n) + (X_n' \cdot Y_n \cdot A_n') + (X_n \cdot Y_n \cdot A_n) + (X_n \cdot Y_n' \cdot A_n')$

$= A_n \cdot [(X_n' \cdot Y_n') + (X_n \cdot Y_n)] + A_n' \cdot [(X_n' \cdot Y_n) + (X_n \cdot Y_n')]$

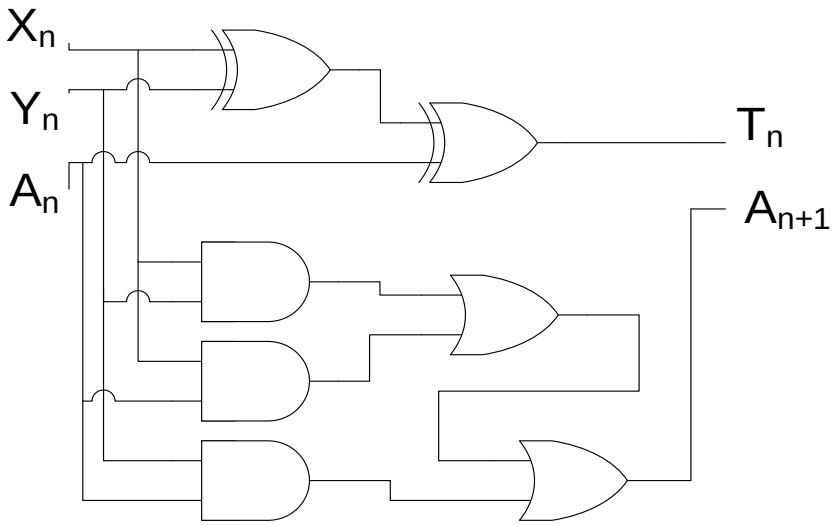
$= A_n$ **XOR** $(X_n$ **XOR** $Y_n)$

A_{n+1} için karnaught haritasını çizelim:

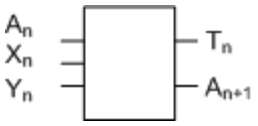
A_n / X_nY_n	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	1	1

$A_{n+1} = (X_n$ **AND** $Y_n)$ **OR** $(A_n$ **AND** $Y_n)$ **OR** $(A_n$ **AND** $X_n)$

T_n ve A_{n+1} için elde ettiğimiz denklemleri kulla
narak devremizi çizelim:

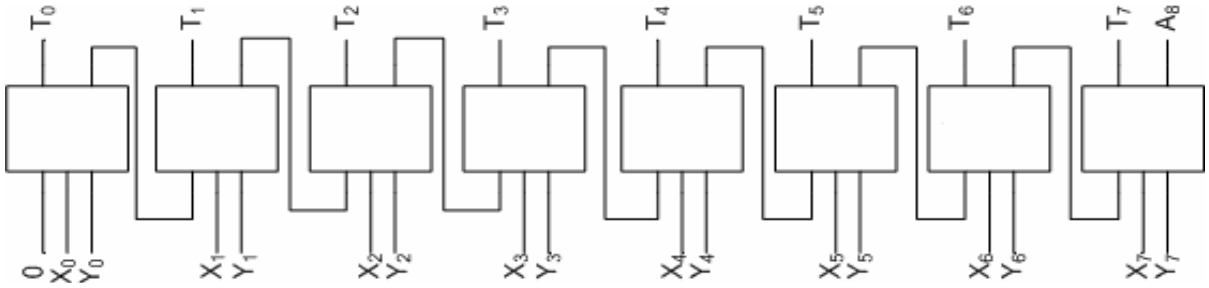


Yukarıda tasarladığımız devre 8 bitlik 2 sayının toplanması işlemi için uygun bir bir bitlik tam toplayıcıdır. Bu bir bitlik tam toplayıcıyı aşağıdaki şekilde gösterelim:



Bu bir bitlik tam toplayıcıyı kullanarak 8 bitlik toplayıcımızı tasarlayalım.

İlk tam bit toplayıcısına dikkat edilirse artan olarak 0 verildi. Bunun nedeni ilk bit toplamından önce herhangi bir bit toplamının olmamasıdır. Son tam bit toplayıcısında ise A_8 açıkta kalmıştır. A_8 1 olduğunda toplama işlemi artma hatası vermiş olacaktır. Sonuç $T_7T_6T_5T_4T_3T_2T_1T_0$ sayısıdır.



Yukarıda tasarladığımız devrenin assembly ile modelini oluşturup devremizin düzgün çalışıp çalışmadığını kontrol edelim. 01010101B ve 01000001B değerlerini toplayalım.

```
1  ORG 0100H
2  MOV AH,00000000B
3  MOV CH,00000000B
4  MOV CL,00000000B
5  MOV AL,00000000B
6  MOV BH,10000000B
7  MOV BL,10000000B
8  DONGU: ROL BH,1
9  ROL BL,1
10 PUSH BX
11 AND BH,01010101B
12 AND BL,01000001B
```

```
13  PUSH BX
14  XOR BH,BL
15  ROL AH,1
16  XOR BH,AH
17  OR CH,BH
18  POP BX
19  MOV DH,BH
20  MOV DL,DL
21  AND DL,AH
22  AND DH,AH
23  AND BH,BL
24  OR BH,DL
25  OR BH,DH
26  MOV AH,BH
27  POP BX
28  CMP BX,8080H
29  JE SON
30  JMP DONGU
31  SON: INT 03H
```

Yukarıdaki kodu açıklamadan önce denklemlerimizi hatırlayalım:

T_n = A_n **XOR** (X_n **XOR** Y_n)

A_{n+1} = (X_n **AND** Y_n) **OR** (A_n **AND** Y_n) **OR** (A_n **AND** X_n)

İkinci, üçüncü ve dördüncü satırlarda değişkenlere ilk değerler veriliyor.

Altıncı ve yedinci satırlarda ise BH ve BL yazmaçlarına 10000000B değeri veriliyor. Bu iki yazmaç sayıların sadece bir bitini yüklemekte kullanılacaktır. 1 olan hane yüklenecek olan hanedir. Yüklemek için on birinci ve on ikinci satırlarda olduğu gibi toplanacak olan sayılar ile AND lenmekte ve sadece toplanacak olan sayıların o biti BH ve BL yazmacına geçmektedir. Diğer bitler BH ve BL yazmacında 0 olduğundan AND işleminde yutulmaktadır. Sekizinci ve dokuzuncu satırlarda BH ve BL deki işlenme biti 1 basamak sola kaydırılmaktadır. Altıncı ve yedinci satırda ilk biti değil de en son biti 1 yapmamızın nedeni budur çünkü bu satıra gelindiğinde zaten otomatik olarak ilk bit 1 olacaktır. Eğer böyle yapmasaydık kodu biraz uzatmış olacaktık çünkü ilk durum için özel bir kod yazmamız gerekecekti.

Onuncu satırda BX yazmacının değerini yığına kaydediyoruz. Bu değer hangi bitin işleme tabi tutulacağının belirten değerdir.

On üçüncü satırda BX yazmacındaki yani BH ve BL yazmacındaki değerleri yığına atıyoruz. Bu şekilde ilk değerleri kullanmamız gerektiğinde, ikinci denklemde bir daha kullanmamız gerekecek, yeniden hesaplamak yerine hesaplı olan veriyi saklamış oluyoruz.

On dördüncü satırda T_n denklemindeki (X_n **XOR** Y_n) bölümünü hesaplayıp sonucu BH yazmacına kaydediyoruz.

On beşinci satırda ise AH yazmacının değerini bir sola kaydırıyoruz. Bunu yapmamızın nedeni AH yazmacında A_n değerinin olması ve artanı doğru BH biti ile işleme tabi tutmak için 1 bit kaydırmamız gerekmesidir.

On altıncı satırda artan ile BH yazmacındaki değeri XOR işlemine tabi tutuyoruz ve sonuç BH yazmacına yazılıyor.

On yedinci satırda CH yazmacında tutulacak olan sonuç ile BH yazmacını OR işlemine tabi tutuyoruz. Bu şekilde 1 bitte olan değişikliği CH yazmacına aktarabiliyoruz ve önceki CH yazmacındaki diğer bitlerin değerlerini kaybetmemiş oluyoruz.

On sekizinci satırda ikinci denklem için BH ve BL yazmaçlarını geri yüklüyoruz.

On dokuzuncu ve yirminci satırlarda BH ve BL yazmaçlarındaki değerleri DH ve DL yazmaçlarına kopyalıyoruz.

Yirmi birinci ve yirmi beşinci satırlar arasında ikinci denklemde belirtilen işlemleri yazmaçlarla uyguluyoruz.

Yirmi altıncı satırda BH yazmacındaki A_{n+1} değerini AH yazmacına kaydediyoruz. Yukarıda da belirttiğimiz gibi AH yazmacındaki değer bir sonraki denklemin A_n değeri olacaktır.

Yirmi yedinci satırda onuncu satırda yığına kaydettiğimiz BX yazmacının değerini BX yazmacına geri yüklüyoruz.

Yirmi sekizinci satırda en son bitleri kontrol edip etmediğimizi kontrol etmek amacıyla BX yazmacını 1000000010000000B değeri ile CMP komutuna tabi tutuyoruz. Eğer BX yazmacının değeri 8080H değerine eşitse yirmi dokuzuncu satırdaki atlama komutu ile *SON* etiketine atlıyoruz. Aksi taktirde *DONGU* etiketine geri atlanıyor.

Programı denediğimizde AH yazmacında 00H değeri ve CH yazmacında 96H değerinin olduğunu göreceğiz. AH yazmacında 00H değerinin bulunması toplama işlemimizden herhangi bir artan olmadığını göstermektedir.

Programımız ile birde 11110000B ve 11001100B değerlerini toplayalım. Programı DEBUG ile denediğimizde AH yazmacında 80H değerinin bulunduğunu ve CH yazmacında BCH değerinin bulunduğunu göreceksiniz. Yani son bitlerin toplamından bir artma olmuştur. Yani toplama işlemimizin sonucu 1BCH dır.

6. ASSEMBLY İLE DİZGİ VE BCD-ASCII İŞLEMLERİ

6.1. Dizgi İşlemleri

Dizgi işlemleri ya bir bayt yada bir kelime üzerinde yapılır. Dizgi işlemlerinde dizgiler DI ve SI yazmaçları ile adreslenirler. DF bayrak biti 0 ise DI ve SI yazmaçları bayt için 1 ve kelime için 2 artırılır. DF bayrak biti 1 ise DI ve SI yazmaçları bayt için 1, kelime için 2 ve ikili kelime için 4 azaltılır.

Dizgi komutlarına bakılacak olursa her dizgi komutunun birkaç şeklinin olduğu görülecektir. Mesela kısa şekillerinde işlenen belirtilmez ve otomatik olarak SI ve DI adresleri işlenen olarak kullanılır. Eğer bir kelime işlenen olarak kullanılacaksa SI yazmacı ile birlikte DS kesim yazmacı ve DI ile birlikte ES kesim yazmacı kullanılır.

Bu kısa girişi yaptıktan sonra 4. bölümde gördüğümüz dizgi komutlarını hatırlayalım ve ardından bu komutların kullanımlarını birkaç örnek ile örneklendirelim.

- REP/REPE/REPNE/REPNZ/REPZ
- MOVSB/MOVSW
- CMPSB/CMPSW
- SCASB/SCASW
- LODSB/LODSW
- STOSB/STOSW

Aşağıdaki tabloda yukarıdaki komutları özetleyelim:

Komut	Kaynak	Hedef	Ön Ek
MOVSB	DS:SI	ES:DI	REP
MOVSW	DS:SI	ES:DI	REP
CMPSB	DS:SI	ES:DI	REPE/REPNE
CMPSW	DS:SI	ES:DI	REPE/REPNE
SCASB	AL	ES:DI	REPE/REPNE
SCASW	AX	ES:DI	REPE/REPNE
LODSB	DS:SI	AL	-
LODSW	DS:SI	AX	-
STOSB	AL	ES:DI	REP
STOSW	AX	ES:DI	REP

6.1.1. DF Bayrak Biti

Yukarıda DF bayrak bitinin DI ve SI yazmaçlarının artırılışını ve azaltılışını kontrol ettiğinden bahsetmiştik. DF bayrak bitini değiştirmek için 2 tane assembly komutu bulunmaktadır.

CLD komutu DF bayrak bitini 0 a eşitler ve dizgi komutlarının SI ve DI yazmaçlarının değerlerini artırmasına neden olur.

STD komutu DF bayrak bitini 1 e eşitler ve dizgi komutlarının SI ve DI yazmaçlarının değerlerini azaltmasına neden olur.

6.1.2. REP MOVSB

MOVSB ile REP ön ekini kullanarak bir yere kaydettiğimiz “MERHABA DUNYA!” dizgisini her seferde bir bayt taşıyarak 0170H göreli adresine taşıyalım.

```
1  ORG 0100H
2  JMP BASLANGIC
3  VERI DB 'MERHABA DUNYA!'
4  BASLANGIC: MOV CX,14
5  MOV AX, CS
6  MOV DS, AX
7  MOV ES, AX
```

```
 8  CLD
 9  MOV SI, VERI
10  MOV DI, 0170H
11  REP MOVSB
12  INT 03H
```

3. satırda "MERHABA DUNYA!" dizgimizi tanımladık ve erişimi kolaylaştırmak için değişken adı olarak *VERI* seçti.

4. satırda CX yazmacına "MERHABA DUNYA!" dizgisindeki harf sayısını girdik.

5., 6. ve 7. satırlarda DS ve ES kesim yazmaçlarının değerlerini CS yazmacına eşit hale getirdik.

8. satırda CLD komutunu kullanarak DF bayrak bitini 0 a eşitledik. Bu sayede her MOVSB komutunda DI ve SI yazmaçlarının değerleri 1 artırılacak.

9. satırda SI yazmacına "MERHABA DUNYA!" dizgisinin adresini yükledik. İşlenen olarak değişken adının yazılması "MERHABA DUNYA!" dizgisinin göreli konum adresinin yüklenmesine neden oldu.

10. satırda DI yazmacına 0170H değerini yükledik. DI yazmacı MOVSB komutunda hedef göreli adresi olarak kullanıldığından hedef olarak 0170H adresini seçtik.

11. satırda ise "MERHABA DUNYA!" dizgisinin taşınmasını sağladık. 4. satırda CX yazmacına yüklediğimiz 14 değeri sayesinde MOVSB komutu 14 kez tekrarlandı ve doğru bir şekilde 0170H adresine "MERHABA DUNYA!" dizgisi taşındı.

Yukarıdaki kodu DEBUG ile deneyip 0170H göreli konum adresindeki verilere bakarsak "MERHABA DUNYA!" dizgisinin tamamen taşınmış olduğunu göreceğiz.

6.1.3. REP MOVSW

MOVSW ile REP ön ekini kullanarak bir yere kaydettiğimiz "MERHABA DUNYA!" dizgisini her seferde bir kelime taşıyarak 0170H göreli adresine taşıyalım.

```
 1  ORG 0100H
 2  JMP BASLANGIC
 3  VERI DB 'MERHABA DUNYA!'
 4  BASLANGIC: MOV CX,14
 5  MOV AX,CS
 6  MOV DS,AX
 7  MOV ES,AX
 8  CLD
 9  MOV SI,VERI
10  MOV DI,0170H
11  REP MOVSW
12  INT 03H
```

Yukarıdaki kodu DEBUG ile deneyip 0170H göreli konum adresindeki verilere bakarsak "MERHABA DUNYA!" dizgisinin tamamen taşınmış olduğunu göreceğiz.

6.1.4. REPE CMPSB

CMPSB komutunu ve REPE ön ekini kullanarak "MERHABA DUNYA!" ve "MERABA DUNYA!" dizgilerini karşılaştıralım. Birbirlerine eşit iseler BX yazmacına 1234H değeri, değilseler 4321H değeri yazdıralım.

```
 1  ORG 0100H
 2  JMP BASLANGIC
 3  VERI1 DB 'MERHABA DUNYA!'
 4  VERI2 DB 'MERABA DUNYA!'
 5  BASLANGIC: MOV CX,14
 6  MOV AX,CS
 7  MOV DS,AX
 8  MOV ES,AX
 9  CLD
```

```
10  MOV SI,VERI1
11  MOV DI,VERI2
12  REPE CMPSB
13  JE AYNI
14  JMP FARKLI
15  AYNI: MOV BX,1234H
16  INT 03H
17  FARKLI: MOV BX,4321H
18  INT 03H
```

3. satırda *VERI1* değişkenini tanımadık ve 4. satırda *VERI2* değişkenini tanımladık.

10. satırda SI yazmacına *VERI1* değişkeninin adresini yükledik ve 11. satırda DI yazmacına *VERI2* değişkeninin adresini yükledik.

13. satırda REPE CMPSB işleminin sonucuna göre *AYNI* etiketine atlanıp atlanılmayacağı kararlaştırılmaktadır.

14. satırda ise JMP komutu *AYNI* etiketine 13. satırda atlanmamışsa *FARKLI* etiketine atlanmasını sağlamaktadır.

Yukarıdaki kodu denediğimizde BX yazmacında 4321H değerinin olduğunu göreceğiz. Eğer *VERI2* yi "MERHABA DUNYA!" şekline getirip bir daha denersek BX yazmacında 1234H değerini göreceğiz.

REPNE CMPSB, REPE CMPSW ve REPNE CMPSW için benzer programlar yazılabilir.

6.1.5. REPNE SCASB

SCASB komutu ile REPNE ön ekini kullanarak "MERHABA DUNYA!" dizgisindeki "D" harfini "S" harfi ile değiştirelim.

```
1   ORG 0100H
2   JMP BASLANGIC
3   VERI DB 'MERHABA DUNYA!'
4   BASLANGIC: MOV CX,14
5   MOV AX,CS
6   MOV ES,AX
7   CLD
8   MOV DI,VERI
9   MOV AL,'D'
10  REPNE SCASB
11  JNE SON
12  DEC DI
13  MOV BYTE [DI],'S'
14  SON: INT 03H
```

10. satırda AL yazmacındaki değer ES:DI tarafından işaret edilen bellek bölgesindeki değer ile karşılaştırılıyor ve eşit olmadığı sürece DI yazmacının değeri 1 artırılarak karşılaştırmalara devam ediliyor.

11. satırda eğer eşit olan bir değer bulunamaz ise SON etiketine atlanıyor.

12. satırda eğer eşit bir harf bulunmuş ise DI yazmacının değeri 1 azaltılıyor çünkü her SCASB komutu işlemini yaptıktan sonra DI yazmacını 1 artırıyor. Doğru harfe işaret etmek için DI yazmacının değerini 1 azaltmamız gerekiyor.

13. satırda DI göreli konum adresindeki bayt değeri "S" harfinin değeri ile değiştiriliyor.

REPE SCASB, REPNE SCASW ve REPE SCASW için benzer programlar yazılabilir.

6.1.6. LODSB

Şimdi yazacağımız örnekte LODSB komutunu kullanarak bir adreste bulunan bir dizgiyi başka bir adrese taşıyacağız.

```
1  ORG 0100H
2  JMP BASLANGIC
3  YAZI DB 'MERHABA DUNYA!'
4  BASLANGIC: MOV AX,CS
5  MOV DS,AX
6  MOV ES,AX
7  CLD
8  MOV CX,14
9  MOV SI,YAZI
10 MOV DI,0200H
11 OKU: LODSB
12 MOV [DI],AL
13 INC DI
14 LOOP OKU
15 INT 03H
```

4. satırda CS kesim yazmacının değerini AX yazmacına aktardık. 5. ve 6. satırlarda ise bu değeri DS ve ES kesim yazmaçlarına yazdık.

7. satırda DF bayrak bitini 0 a eşitledik.

8. satırda "MERHABA DUNYA!" dizgimizin uzunluğu olan 14 sayısını CX yazmacına yazdırdık. Bu değeri yazmamızın nedeni LOOP OKU nun 14 kez okuma yapmasını sağlamaktır.

9. satırda "MERHABA DUNYA!" dizgimizin adresini SI yazmacına yazdırıyoruz.

10. satırda DI yazmacına 0200H değerini yazdırıyoruz. "MERHABA DUNYA!" dizgimizi 0200H adresine kopyalamak istediğimizden bu değeri yükledik.

11. satırda LODSB komutu ile SI yazmacının adresindeki veri AL yazmacına aktarılıyor.

12. satırda DI yazmacının adresine AL yazmacının değeri aktarılıyor ve 13. satırda DI yazmacının değeri bir sonraki bayta geçmek için 1 artırılıyor.
14. satırdaki LOOP komutu ise 14 kez 11. ,12. ve 13. satırdaki işlemlerin tekrarlanmasını sağlıyor.

6.1.7. REP STOSB

Şimdi de STOSB komutu ile birlikte REP ön ekini kullanarak bir adreste bulunan "MERHABA DUNYA!" dizgisini başka bir adrese taşıyacağız.

```
1  ORG 0100H
2  JMP BASLANGIC
3  YAZI DB 'MERHABA DUNYA!'
4  BASLANGIC: MOV AX,CS
5  MOV DS,AX
6  MOV ES,AX
7  CLD
8  MOV SI,YAZI
9  MOV DI,0200H
10 MOV CX,14
11 OKU: PUSH CX
12 MOV AL,[SI]
13 MOV CX,1
14 REP STOSB
15 INC SI
16 POP CX
17 LOOP OKU
```

18	INT 03H

10. satırda CX yazmacına 14 değerini aktardık. Bu değeri LOOP OKU döngümüzde kullanacağız.

11. satırda bu yüklediğimiz değeri yığına kaydettik.

12. satırda SI yazmacının değerinin adresindeki veriyi AL yazmacına kaydettik.

13. satırda REP STOSB komutunun ne kadar kez tekrarlanacağını belirtmek amacıyla CX yazmacına 1 değerini yükledik.

14. satırda REP STOSB komutunu kullandık. Bu sayede DI yazmacının değerinin adresine AL yazmacındaki veri kaydedildi. DI yazmacının değeri 7. satırda DF bayrak biti 0 a eşitlendiğinden 1 artırıldı.

15. satırda "MERHABA DUNYA!" dizgisinin bir sonraki harfini işaret etmek için SI yazmacının değerini 1 artırdık.

16. satırda LOOP OKU döngümüzde kullandığımız CX yazmacının değerini yığından geri CX yazmacına yükledik.

6.2. BCD-ASCII İşlemleri

x86 işlemcileri assembly komutlarının çoğu hem BCD hem de ASCII sayılarını işlenen olarak kabul edebilir. ASCII sistemi Amerika Birleşik Devletleri ve birçok ülkede standart olarak kabul edilmesine rağmen BCD sayıları tüm dünyada genel geçerdir. Ayrıca klavyeler, yazıcılar ve monitörler ASCII ile çalışmaktadır ve BCD sayıların gösteriminin yapılması gerekmektedir. Bunun için x86 işlemcileri bu iki sayı sistemi arasında geçiş yapabilmektedir. Komutların BCD ve ASCII sayıları ile kullanımlarını örneklendirmeden önce BCD sayı sistemi ve ASCII sayı sistemini inceleyelim.

6.2.1. BCD Sayı Sistemi

BCD (**b**inary **c**oded **d**ecimal – ikili kodlanmış onluk sayı) sayı sisteminin oluşturulma nedeni insanların onluk tabanda bir sayı gösterirken bir haneye 0 ile 9 arasında sayıları yazmasıdır. Tek veya çift haneli bir onluk sayının gösterimi BCD sayı sistemi ile diğer sistemlere göre çok daha kolaydır. BCD sayı sistemi 2 çeşittir:

- Paketlenmemiş BCD (Unpacked BCD) Sayı Sistemi
- Paketlenmiş BCD (Packed BCD) Sayı Sistemi

6.2.1.1. 6.2.1.1 Paketlenmemiş BCD Sayı Sistemi

Paketlenmemiş BCD sayı sisteminde 0 ile 9 arası sayılar bir baytın alt 4 bitine kodlanır. Üst 4 bit 0 a eşitlenir. Yani her BCD sayısı 1 baytlık yer kaplar. Aşağıda paketlenmemiş BCD sayı sisteminde 0 dan 9 a kadar olan sayılar verilmiştir:

Ondalık	Paketlenmemiş BCD
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
4	0000 0100
5	0000 0101
6	0000 0110
7	0000 0111
8	0000 1000
9	0000 1001

6.2.1.2. Paketlenmiş BCD Sayı Sistemi

Paketlenmiş BCD sayı sisteminde üst 4 bitin 0 a eşit olduğunu söylemiştik. Paketlenmiş BCD sayı sisteminde bu üst 4 bayt bir üst basamağın BCD değerini kaydetmek için kullanılır. Yani 8 bitin alt 4 biti

sayının ilk hanesi için, üst 4 bit ise sayının ikinci hanesi için kullanılmaktadır. Yani 2 basamak sadece 1 baytlık yer kaplamaktadır. Bu yüzden paketlenmiş BCD sayıları paketlenmemiş BCD sayılarına tercih edilmektedir. Aşağıda bazı ondalık sayılar için paketlenmiş BCD sayı değerleri verilmiştir.

Ondalık	Paketlenmiş BCD
10	0001 0000
21	0010 0001
32	0011 0010
43	0100 0011
54	0101 0100
65	0110 0101
76	0111 0110
87	1000 0111
98	1001 1000
19	0001 1001

6.2.2. ASCII Sayılar

ASCII sayılarını örneklendirmek için 0 dan 9 a kadar olan sayıların ASCII değerlerini gösterelim.

Ondalık	ASCII (İkili)	ASCII (Hex)
0	011 0000	30
1	011 0001	31
2	011 0010	32
3	011 0011	33
4	011 0100	34
5	011 0101	35
6	011 0110	36
7	011 0111	37
8	011 1000	38
9	011 1001	39

Dikkat edilirse yukarıdaki tabloda ASCII sayıları gösterirken ikili gösterimde 7 bit kullandık. ASCII kodlama sisteminde anlamlı olan bitler alt 7 bittir. 8. bit ise 0 veya 1 olabilir. Bazı uygulamalarda 8. bit eşlik biti olarak kullanılmaktadır.

6.2.3. Paketlenmemiş BCD – ASCII Çevirimi

Örnek olarak 0000 0101 paketlenmemiş BCD sayısını yani ondalık 5 sayısını ASCII sayı sistemine çevirelim. 5 sayısının ASCII gösterimi 011 0101 dir.

Yani yapmamız gereken paketlenmemiş BCD sayısının alt 4 bitini alıp üst 4 biti de 0011 olarak tamamlamak. Aşağıda yukarıdaki işlemi yapan program verilmiştir:

```
1   ORG 0100H
2   JMP BASLA
3   BCD_SAYI DB '5'
4   ASCII_SAYI DB 00H
5   BASLA: MOV SI,BCD_SAYI
6   MOV AL,[SI]
7   MOV SI,ASCII_SAYI
8   AND AL,00001111B
9   XOR AL,00110000B
10  MOV [SI],AL
11  INT 03H
```

5. satırda SI yazmacına BCD_SAYI değişkeninin adresi kaydediliyor.

6. satırda SI yazmacının değerinin işaretlediği adresteki veriler AL yazmacına aktarılıyor.

7. satırda SI yazmacına ASCII_SAYI değişkeninin adresi kaydediliyor.

8. satırda AL yazmacının alt 4 biti korunurken üst 4 biti 0 a eşitleniyor.

9. satırda üst 4 bite 0011 yazılıyor.

10. satıda ise AL yazmacının değeri ASCII_SAYI konumuna geri yükleniyor.

6.2.4. Paketlenmiş BCD – ASCII Çevirimi

Örnek olarak 0111 0101 paketlenmiş BCD sayısını yani ondalık 75 sayısını ASCII sayı sistemine çevirelim. 75 sayısının ASCII gösterimi 0011 0111 0011 0101 dir.

Yani yapmamız gereken paketlenmiş BCD sayısının alt 4 bitini ve üst 4 bitini ayrı ayrı alıp ASCII sayı sistemine uygun hale getirmek. Aşağıda yukarıdaki işlemi yapan program verilmiştir:

```
1   ORG 0100H
2   JMP BASLA
3   BCD_SAYI DB 01110101B
4   ASCII_SAYI DW 0000H
5   BASLA: MOV SI,BCD_SAYI
6   MOV AL,[SI]
7   MOV AH,[SI]
8   AND AL,00001111B
9   AND AH,11110000B
10  XOR AL,00110000B
11  SHR AH,4
12  XOR AH,00110000B
13  MOV SI,ASCII_SAYI
14  MOV [SI],AX
15  INT 03H
```

5. satırda SI yazmacına BCD_SAYI değişkeninin adresi kaydediliyor.

6. ve 7. satırlarda AL ve AH yazmaçlarına SI yazmacının değerinin işaretlediği veriler aktarılıyor.

8. satırda AL yazmacının alt 4 biti korunuyor, üst 4 bit 0 a eşitleniyor.

9. satırda AH yazmacının üst 4 biti korunuyor, alt 4 biti ise 0 a eşitleniyor.

10. satırda AL yazmacının üst 4 bitine 0011 yazdırılıyor.

11. satırda AH yazmacının değeri 4 bit sağa kaydırılıyor. (Paketlenmiş BCD sayısının üst 4 bitinde bir sonraki basamağın değeri bulunduğundan bu değeri alt 4 bite çekmemiz gerekiyor)

12. satırda AH yazmacının üst 4 bitine 0011 yazdırılıyor.

13. satırda SI yazmacına ASCII_SAYI değişkeninin adresi yazılıyor.

14. satırda AX yazmacının değeri SI yazmacının değerinin işaretlediği bellek konumuna aktarılıyor.

6.2.5. ASCII – Paketlenmemiş BCD Çevirimi

Örnek olarak 0011 0010 ASCII sayısını yani 2 sayısını paketlenmemiş BCD sayısına çevirelim. 2 sayısının paketlenmemiş BCD gösterimi 0000 0010 dır.

Yani 0011 0010 sayısının üst 4 bitini 0 a eşitlememiz gerekmektedir. Aşağıda bu işlemi yapan program verilmiştir:

```
1   ORG 0100H
2   JMP BASLA
3   BCD_SAYI DB 00H
4   ASCII_SAYI DB 32H
5   BASLA: MOV SI,ASCII_SAYI
6   MOV AL,[SI]
7   AND AL,00001111B
8   MOV SI,BCD_SAYI
9   MOV [SI],AL
10  INT 03H
```

5. satırda SI yazmacına ASCII_SAYI değişkeninin adresi kaydediliyor.

6. satırda AL yazmacına SI yazmacının değerinin işaretlediği veriler aktarılıyor.

7. satırda AL yazmacının alt 4 biti korunup üst 4 biti 0 a eşitleniyor.

8. satırda SI yazmacına BCD_SAYI değişkeninin adresi kaydediliyor.

9. satırda SI yazmacının değerinin işaretlediği bellek bölgesine AL yazmacının değeri aktarılıyor.

6.2.6. ASCII – Paketlenmiş BCD Çevirimi

Örnek olarak 0011 0010 0011 0001 ASCII sayısını yani 21 sayısını paketlenmiş BCD sayısına çevirelim. 21 sayısının paketlenmiş BCD gösterimi 0010 0001 dır.

Yani 0011 0010 0011 0001 sayısının ilk baytını paketlenmiş BCD sayısının üst 4 bitine, ikinci baytını ise paketlenmiş BCD sayısının alt 4 bitine yazmalıyız. Aşağıda bu işlemi yapan program verilmiştir:

```
1   ORG 0100H
2   JMP BASLA
3   BCD_SAYI DB 00H
4   ASCII_SAYI DW 3231H
5   BASLA: MOV SI,ASCII_SAYI
6   MOV AL,[SI]
7   MOV AH,[SI]
8   AND AL,00001111B
9   AND AH,00001111B
10  SHL AH,4
11  MOV SI,BCD_SAYI
12  MOV [SI],AX
13  INT 03H
```

5. satırda SI yazmacına ASCII_SAYI değişkeninin adresini kaydettik.

6. ve 7. satırlarda ise AL ve AH yazmaçlarına SI yazmacının değerinin işaretlediği bellek bölgesindeki değeri aktardık.

8. ve 9. satırlarda ise AL ve AH yazmaçlarının üst 4 bitlerini 0 a eşitledik. Bu işlemi yaparken alt 4 bitlerde herhangi bir değişikliğe neden olmamak için 1 ile AND ledik.

10. satırda AH yazmacını 4 bit sola kaydırdık. Bu şekilde paketlenmiş BCD sayısının üst 4 biti için verilmesi gereken bit değerlerini doğru bitlere kaydırmış olduk.

11. satırda SI yazmacına BCD_SAYI değişkeninin adresini kaydettik.

12. satırda SI yazmacının değerinin işaretlediği bellek bölgesine AX yazmacının değerini kaydettik.

7. KESMELER VE ÇEVRE BİRİMLERİNİN KULLANIMI

7.1. Kesmeler

x86 işlemcisinde bir kesme oluştuğu anda o an çalışan program durdurulur (kesilir) ve işlemci ISR (Interrupt Service Routine – Kesme Hizmeti Yordamı) adlı özel bir programa atlar. Her ISR aslında bellekte bulunan normal bir programdır. ISR işini bitirdiğinde işlemci kesildiği noktaya genellikle (bazı kesmelerde dönmez) geri döner.

ISR yukarıdan da anlaşılacağı gibi belli bir bellek bölgesine atlamaya dayalı bir sistemdir. Bunun dışında PIC (Programmable Interrupt Controller – Programlanabilir Kesme Denetleyicisi) veya APIC (Advanced Programmable Interrupt Controller – Gelişmiş Programlanabilir Kesme Denetleyicisi) kullanılabilir. Bu iki sistem donanımsaldır. İşlemcinin kesme iğnesine ve kesme yapan donanıma aynı anda bağlıdırlar ve kesmeleri bu şekilde denetlemektedirler.

Kesmeler ana 3 bölüme ayrılabilir:

- Donanım Kesmeleri
- Yazılım Kesmeleri
- Kural Dışı kesmeler

7.1.1. Donanım Kesmeleri

Donanımsal kesmeler adından da anlaşılacağı gibi donanımlar tarafından oluşturulmaktadırlar. Klavyenizde bir tuşa bastığınızda klavyeniz bir donanımsal kesme oluşturur. İşlemci bu kesmeyi işlemek için klavye veri giriş kodunu çalıştırır ve bastığınız tuşun kodu bir bellek bölgesine kaydedilir. Donanımsal kesmeler asenkrondur. Yani kesme oluştuğu anda çalışmakta olan programdan bağımsızdır.

7.1.2. Yazılım Kesmeleri

Yazılımsal kesmeler yazılımların içinden oluşturulabilen kesmelerdir. Yazılımsal kesmelere örnek olarak şu ana kadar çokça kullandığımız *INT 03H* kesmesini verebiliriz. İleride göreceğimiz INT 10H kesmesi kullanılarak ekrana yazı yazdırabiliriz.

7.1.3. Kural Dışı Kesmeler

Kural dışı kesmeleri çalışmakta olan programda kural dışı bir sonuç oluştuğunda ortaya çıkar. Mesela 0 a bölme veya izin olmayan bir bellek bölgesine erişmeye çalışmak kural dışı kesmelere bir örnektir.

Birçok işletim sisteminde kesmeler kullanılır. Mesela DOS, Linux gibi işletim sistemlerinde kesmeler kullanılabilir. Windows işletim sisteminde ise kesmeler yerine API (Application Programming Interface – Yazılım Programlama Ara Yüzü) kullanılır. API bir bakıma kesmelerin ara yüz yapılarak kullanımlarını kolaylaştıran bir halidir.

Şimdi işletim sisteminden bağımsız BIOS kesmelerini inceleyelim. DOS ve Linux kesmeleri için Ek C'ye bakınız.

7.1.4. BIOS Kesmeleri

BIOS kesmeleri bir bilgisayarda alt seviye programlar yazarken çok önemlidir. Aşağıda BIOS kesmelerinin bir listesi ve kısaca tanımları verilmiştir. Daha detaylı bir liste için Ek D ye bakınız.

Kesme	Tanım
INT 00H	İşlemci: Sıfıra Bölme
INT 01H	İşlemci: Hata ayıklama için adım adım yürütme
INT 02H	İşlemci: **NMI** (**N**on-**M**askable **I**nterrupt – Maskelenemeyen Kesme)
INT 03H	İşlemci: Hata ayıklama için duraksama noktası
INT 04H	İşlemci: Taşma hatası
INT 05H	Ekranı Yazdır (Print Screen)

INT 08H	IRQ0: Sistem zamanlayıcısı tarafından çağrılır
INT 09H	IRQ1: Klavye tarafından çağrılır
INT 0BH	IRQ3: COM2 (2. seri kapı) tarafından çağrılır
INT 0CH	IRQ4: COM1 (1. seri kapı) tarafından çağrılır
INT 0DH	IRQ5: Sabit disket denetleyicisi (PC/XT sistemlerde) veya 2. paralel kapı tarafından çağrılır (AT sistemlerinde)
INT 0EH	IRQ6: 1.44" disket sürücü denetleyicisi tarafından çağrılır
INT 0FH	IRQ7: 1. paralel kapı (yazıcı) tarafından çağrılır
INT 10H	Video Hizmetleri
INT 11H	Çevre birimi yüklendi
INT 12H	Konvansiyonel bellek boyutunu al
INT 13H	Alt Seviye Disk Hizmetleri
INT 14H	Seri I/O (Input/Output – Giriş/Çıkış)
INT 15H	Çeşitli Kesmeler
INT 16H	Klavye
INT 17H	Yazıcı Hizmetleri
INT 18H	INT 19H başarısızlıkla sonuçlanırsa kullanılır
INT 19H	**POST** (**P**ower-**O**n **S**elf **T**esting – Açılışta Otomatik Sınama) dan sonra işletim sistemini yüklemek için kullanılır
INT 1AH	**RTC** (**R**eal **T**ime **C**lock – Gerçek Zamanlı Saat) Hizmetleri
INT 1BH	"Break" tuşuna basıldığında çağrılır
INT 1CH	Yazılımsal zamanlayıcı kesmesi. INT 08H tarafından çağrılır.
INT 1DH	Adres imleci: **VPT** (**V**ideo **P**arameter **T**able – Vidyo Parametre Tablosu)
INT 1EH	Adres imleci: **DPT** (**D**iskette **P**arameter **T**able – Disket Parametre Tablosu)
INT 1FH	Adres imleci: **VGCT** (**V**ideo **G**raphics **C**haracter **T**able – Vidyo Grafikleri Karakter Tablosu)
INT 41H	Adres imleci: **FDPT 1** (**F**ixed **D**isk **P**arameter **T**able **1 –** 1. Sabit Disket Parametre Tablosu)
INT 46H	Adres imleci: **FDPT 2** (**F**ixed **D**isk **P**arameter **T**able **2 –** 2. Sabit Disket Parametre tablosu)
INT 4AH	RTC tarafından alarm için çağrılır
INT 70H	IRQ8: RTC tarafından çağrılır
INT 74H	IRQ12: Fare tarafından çağrılır
INT 75H	IRQ13: Matematiksel ek işlemci tarafından çağrılır
INT 76H	IRQ14: Birinci IDE denetleyicisi tarafından çağrılır
INT 77H	IRQ15: İkinci IDE denetleyicisi tarafından çağrılır

NMI

Yukarıdaki tabloda listelemiş olduğumuz tüm kesmeler ve bazı diğer kesmeler numaraları ile çağırabileceğimiz kesmeler maskelenebilir yani geçici olarak ihmal edilebilen kesmelerdir. Yalnız **NMI** (**N**on-**M**askable **I**nterrupt – Maskelenemeyen Kesme) adlarından da anlaşıldığı gibi maskelenemezler çünkü aşırı önemli durumlarda ve genellikle büyük sistem hatalarının sonucunda ortaya çıkarlar. NMI lerin ihmal edilebilmesi mümkün değildir, oluştukları zaman işlenirler. NMI oluşturacak bir sistem hatasına örnek olarak bellekten gelen verilerde bir eşlik hatasının oluşmasıdır. Bu eşlik hatası düzeltilmeden sistemin devam etmesi istenmeyen ve büyük veri kayıplarına yol açabilecek sonuçlar

doğurabileceğinden NMI gibi bir kavram ortaya atılmıştır.

Şimdi de yukarıdaki kesmelerden bazılarını örnekleyelim.

7.1.4.1. INT 00H

INT 00H 0'a bölme hatası oluştuğu zaman veya DIV veya IDIV komutlarıyla yapılan bölme işleminin bölümünün yazmaca sığmadığı zaman çağırılır.

8086, 8088, V20 ve V30 işlemcileri hata veren DIV veya IDIV komutunun yığına kaydettikten sonra CS:IP adresini yığına kaydederken 80286 ve 80386 gibi işlemciler sadece hata veren DIV veya IDIV komutunun CS:IP adresini yığına kaydeder.

7.1.4.2. INT 10H ve Monitör

INT 10H kesmesini kullanarak ekrana yazı yazdırabiliriz. Bu yazımız için farklı bir arka plan rengi seçebilir ve bir çok özelliğini kontrol edebiliriz. INT 10H kesmesinin alt yordamlarına ulaşabilmek için AH yazmacına INT 10H ı çağırmadan önce farklı değerler vermeliyiz. Aşağıda bu değerler ve işlevleri listelenmiştir.

AH	İşlev
00H	Video tipini ayarla AL = tip
01H	İmleç şeklini ayarla CH = başlama satırı CL = bitiş satırı Satır değerleri 0–15 arasında olduğundan imleci yok etmek için CH yazmacına 20H yüklenebilir.
02H	İmleç pozisyonunu ayarla BH = sayfa numarası DH = y koordinatı DL = x koordinatı **Not:** BIOS her sayfa için imleç pozisyonlarını ayrı ayrı tuttuğundan BH yazmacına 00H yazılması uygundur.
03H	İmleç pozisyonunu ve şeklini al BH = imlecin bulunduğu sayfa numarası Çıktı olarak: CH = başlama satırı CL = bitiş satırı DL = x koordinatı DH = y koordinatı
05H	Görüntü sayfasını ayarla AL = görüntü sayfası
06H	Temizle veya ekranı yukarı kaydır AL = ekranın kaç satır yukarı kaydırılacağı BH = temizlenen alan için ekran görüntü özelliği CL = temizlenecek alan için sol-üst köşe x koordinatı CH = temizlenecek alan için sol-üst köşe y koordinatı DL = temizlenecek alan için sağ-alt köşe x koordinatı DH = temizlenecek alan için sağ-alt köşe y koordinatı

	AL 00H ise temizleme görevi görür. Değilse ekran AL nin değeri kadar yukarı kaydırılır.
07H	Temizle veya ekranı aşağı kaydır AL = ekranın kaç satır aşağı kaydırılacağı BH = temizlenen alan için ekran görüntü özelliği CL = temizlenecek alan için sol-üst köşe x koordinatı CH = temizlenecek alan için sol-üst köşe y koordinatı DL = temizlenecek alan için sağ-alt köşe x koordinatı DH = temizlenecek alan için sağ-alt köşe y koordinatı AL 00H ise temizleme görevi görür. Değilse ekran AL nin değeri kadar aşağı kaydırılır.
08H	İmleç pozisyonundaki karakteri ve özelliklerini oku BH = imlecin bulunduğu sayfa numarası Çıktı olarak: AL = okunan karakter AH = karakterin özellikleri
09H	İmleç pozisyonuna karakter yaz AL = yazılacak karakter BH = sayfa numarası BL = karakter özelliği CX = karakterin tekrarlanma sayısı
0AH	İmleç pozisyonuna karakteri ve özellikleri yaz AL = yazılacak karakter BH = sayfa numarası
0BH	Kenarlık rengini ayarla BH = 00H BL = renk
0EH	Karakter yazdır (özellikleri kullanarak ve imleci yazdıktan sonra yeniden konumlandırarak) AL = yazılacak karakter BH = sayfa numarası
0FH	Video tipi ayarlarını al Çıktı olarak: AH = kolon sayısı AL = video tipi BH = sayfa numarası
13H	Dizgi yazdır

	AL = yazma kipi BH = sayfa numarası BL = özellik CX = dizgi uzunluğu DH = başlama satırı DL = başlama sütunu ES:BP = yazılacak dizginin adresi

Şimdi INT 10H kesmesini kullanarak ekrana "M" harfini yazdıralım:

```
1  ORG 0100H
2  MOV   AH, 0Eh
3  MOV   AL, 'M'
4  INT   10H
5  RET
```

2. satırda AH yazmacına 0EH değerini yükledik. Yukarıdaki tablodan bakılacak olursa AL yazmacına yazılacak karakteri ve BH yazmacına sayfa numarasını girmemiz gerektiğini göreceğiz. BH yazmacını olduğu değerde bırakmamız herhangi bir sorun teşkil etmeyecektir çünkü zaten başlangıçta o sayfanın değerine sahip olacaktır.

3. satırda AL yazmacına M harfinin ASCII değerini yazdık.

4. satırda ise INT 10H komutunu çağırdık ve karakterin ekrana yazılmasını sağladık.

5. satırdaki RET komutu ile de programımızı sonlandırdık.

Şimdi de bu M harfini kırmızı yazı rengi ile yazdıralım. Bunun için yukarıdaki tablodan da görülebileceği gibi AH = 09H durumunu kullanmamız gerekiyor:

```
1  ORG 0100H
2  MOV   AH, 09h
3  MOV   AL, 'M'
4  MOV   BL, 4
5  MOV   CX,1
6  INT   10H
7  RET
```

2. satırda AH yazmacına 09H değerini yükledik.

3. satırda AL yazmacına yazılacak olan karakterin ASCII kodunu yükledik. Yani M harfinin ASCII karşılığı olan 4DH değerini AL yazmacına yükledik.

4. satırda BL yazmacına yazının özellik değerini yazdık. Özellik değerleri için aşağıdaki tabloya bakınız.

5. satırda ise CX yazmacına karakterin ne kadar kez tekrarlanacağını belirttik. Sadece 1 tane M harfi istediğimizden 1 yazdık.

Hex	İkili	Renk
0	0000	Siyah
1	0001	Mavi
2	0010	Yeşil
3	0011	Camgöbeği

4	0100	Kırmızı
5	0101	Eflatun
6	0110	Kahverengi
7	0111	Açık gri
8	1000	Koyu gri
9	1001	Açık mavi
A	1010	Açık yeşil
B	1011	Açık camgöbeği
C	1100	Açık kırmızı
D	1101	Açık eflatun
E	1110	Sarı
F	1111	Beyaz

Şimdide ekrana “MERHABA DUNYA!” yazısını yazdıralım:

```
1   ORG 0100H
2   ; imleç konumunu al
3   MOV AH,03H
4   INT 10H
5   ; yazıyı yaz
6   MOV AL,01H
7   MOV BH,00H
8   MOV BL,01001111B
9   MOV CX,MESAJ_SON - MESAJ_BAS
10  PUSH CS
11  POP ES
12  MOV BP, MESAJ_BAS
13  MOV AH,13H
14  INT 10H
15  JMP SON
16  MESAJ_BAS DB 'MERHABA DUNYA!'
17  MESAJ_SON DB 00H
18  SON: RET
```

Yukarıdaki kodu çevirip DEBUG ile denediğinizde ekrana yazdırılan “MERHABA DUNYA!” yazısının kırmızı arka plan ve beyaz yazı rengine sahip olduğunu göreceksiniz. Kodu incelersek:

3. satırda AH yazmacına 03H değerini yüklüyoruz. Yani INT 10H’ın 03H alt yordamını kullanmak istiyoruz. INT 10H tablosuna bakarsak bu alt yordamının imleç pozisyonunu ve şeklini almaya yaradığını göreceğiz. CH yazmacına imlecin başlama satırı, CL yazmacına imlecin bitiş satırı, DL yazmacına imlecin x koordinatı ve DH yazmacına imlecin y koordinatı yazılacaktır.

4. satırda INT 10H kesmemizi çağırıyoruz.

6. satırda AL yazmacına 01H değerini kaydediyoruz. Bu değer ileride çağıracağımız INT 10H kesmesinin 13H numaralı alt yordamında yazma kipi olarak kullanılacaktır.

7. satırda sayfa numarası olarak kullanılacak olan BH yazmacının değeri atanıyor.

8. satırda yazılacak olan yazının özellikleri ayarlanıyor. Kırmızı arka plan ve beyaz yazı rengi.

9. satırda CX yazmacına “MERHABA DUNYA!” dizgisinin uzunluğu hesaplanıp kaydediliyor.
10. satırda yığına CS yazmacının değeri kaydediliyor.

11. satırda 10. satırda kaydedilmiş olan CS yazmacı değeri yığından çağrılıyor ve ES yazmacına yazdırılıyor. PUSH-POP ikilisini kullanmak yerine 2 tane MOV komutu kullanabilirdik ama performans açısından PUSH-POP kullanmak daha verimlidir. PUSH, POP ve MOV komutlarının çalıştırılma sürelerine bakacak olursa arada büyük bir farkın olduğunu göreceğiz.

12. satırda BP yazmacına "MERHABA DUNYA!" dizgisinin ilk harfinin göreli konum adresini yazdırıyoruz. INT 10H kesmesinin 13H numaralı alt yordamına bakacak olursak yazılacak olan dizginin ES:BP adresinden başlanarak yazılacağını görürüz. 11. ve 12. satırlarda ES:BP adresinin bizim "MERHABA DUNYA!" dizgimize işaret etmesini sağladık.

13. satırda AH yazmacına 13H değerini yazıyoruz ve hemen ardından 14. satırda INT 10H kesmesini çağırıyoruz. Bu şekilde INT 10H kesmesinin 13H numaralı alt yordamına erişmiş olduk.

7.1.4.3. INT 15H

INT 15H kesmesi ilk senelerde bilgisayarlara bağlı kasetleri yönetmek için kullanılmaktaydı. Kasetler günümüzde kullanımda olmadığından INT 15H kesmesi birçok farklı işlevi yerine getirmek için kullanılmaktadır. Mesela COH alt yordamını sayesinde bilgisayarın alt-model numarasına erişilebilir. Detaylı bilgi için Ek D'ye bakınız.

7.1.4.4. INT 09H ve Klavye

Klavyenin her tuşuna basışta ve bırakılışında INT 09H kesmesi çağırılır. Bu kesme donanımsal bir kesmedir (IRQ 1). ROM'da bulunan INT 09H yordamı aşağıdaki şekilde klavyeden gelen bilgileri işler:

- Herhangi bir ASCII karakter tuşuna basıldığında basılan tuşun ASCII kodu ve tarama kodu 32 baytlık klavye tampon belleğine kaydedilir. Tampon belleğe 0000:041E adresinden ulaşılabilir. Gelen ASCII ve tarama kodları 0000:041C adresindeki klavye imleç tamponunun işaretlediği bellek bölgesine kaydedilir. Her kayıttan sonra klavye imleç tamponunun değeri 2 artırılır. Eğer klavye imleç tamponunun değeri klavye tamponunun sonundan daha ileriye işaret ediyorsa geri başa döner.
- Pause/Duraklama" tuşuna basıldığında sonsuz bir döngüye girilir ve döngü geçerli bir ASCII tuşuna basılana kadar dönmeye devam eder.
- "Print Screen" tuşuna basıldığında INT 05H kesmesi çağırılır.
- "Break" tuşuna basıldığında INT 1BH kesmesi çağırılır.
- "Ctrl", "Alt" veya "Shift" tuşlarına basılırsa Shift Durumu (0000:0417 adresindeki) ve Genişletilmiş Shift Durumu (0000:0418 adresindeki) baytlar güncellenir.
- "Ctrl+Alt+Del" tuşlarının hepsine birden basılırsa 0000:0472 adresindeki baştan başlatma bayrağının değeri 1234H a eşitlenir ve kontrol POST (**P**ower **O**n **S**elf **T**est) ye devredilir.

7.1.4.4.1. INT 16H ve Klavye Servisleri

INT 16H kesmesini kullanarak klavyeden karakter alma gibi işlemleri çok basit bir şekilde yapabiliriz. INT 16H kesmesinde INT 10H kesmesinde olduğu gibi alt yordamlar vardır. Aşağıda bu alt yordamlara ulaşmak için gerekli olan AH yazmacı değerleri ve işlemleri listelenmiştir.

AH	İşlev
00H	Klavyeden basılan karakteri okur. Ekrana herhangi bir çıktı vermez. Çağırıldığı zaman bir tuşa basılana kadar bilgisayarı dondurur. Girdi olarak herhangi bir yazmaç almaz. Çıktı olarak AL ve AH yazmaçları verilir. -AL yazmacında basılan tuşun ASCII değeri bulunur. -AH yazmacında ise klavye tarama değeri bulunur. **Not 1:** Klavye ASCII tuş değerleri ve tarama değerleri için Ek E ye bakınız. **Not 2:** Klavyedeki bazı tuşların (Home,Page Up, vb.) ASCII karşılıkları yoktur. Bu tuşlarla işlemler yapabilmek için AH yazmacındaki klavye tarama değerleri kullanılmalıdır.
01H	Girdi durumunu okur. Basılan tuşun ASCII değerini herhangi bir yazmaca kaydetmez. Sadece bir tuşa basılıp basılmadığını kontrol etmek için kullanılır. Çağırıldığı zaman 00H alt yordamı gibi bilgisayarı dondurmaz. Basılan karakterin 00H alt yordamı kullanılarak klavye tampon belleğinden okunması gerekir.

<table>
<tr><td></td><td>Girdi olarak herhangi bir yazmaç almaz.
Çıktı olarak herhangi bir yazmaç verilmez.</td></tr>
<tr><td>02H</td><td>Kontrol tuşlarının durumunu okur.

Girdi olarak herhangi bir yazmaç almaz.
Çıktı olarak AL yazmacı verilir.

AL yazmacının bitlerinin değerlerine göre hangi kontrol tuşlarına basıldığı tespit edilir:
<table>
<tr><th>Bit</th><th>İşlev</th></tr>
<tr><td>0</td><td>“Right Shift” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>1</td><td>“Left Shift” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>2</td><td>“Ctrl” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>3</td><td>“Alt” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>4</td><td>“Scroll Lock” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>5</td><td>“Num Lock” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>6</td><td>“Caps Lock” tuşuna 1-basılmış 0-basılmamış</td></tr>
<tr><td>7</td><td>“Insert” tuşuna 1-basılmış 0-basılmamış</td></tr>
</table></td></tr>
<tr><td>10H</td><td>Genişletilmiş Karakter Oku</td></tr>
<tr><td>11H</td><td>Genişletilmiş girdi durumunu oku</td></tr>
<tr><td>12H</td><td>Genişletilmiş kontrol karakter tuşlarının durumunu oku</td></tr>
</table>

Yukarıdaki alt yordamlardan sadece ilk üçünü inceleyeceğiz.

7.1.4.4.2. 00H Alt Yordamı

Öncelikle “t” harfine basılana kadar kapanmayacak bir program yazalım. Klavyeden bastığımız her tuş ekrana yazdırılsın.

```
1  ORG 0100H
2  KARAKTER_OKU: MOV AH,00H
3  INT 16H
4  CMP AL,74H
5  JE SON
6  MOV AH,0EH
7  INT 10H
8  JMP KARAKTER_OKU
9  SON: RET
```

Yukarıdaki koda baktığımızda karakterin önce okunduğunu, ardından karşılaştırıldığını ve “t” harfine eşit olup olmama durumuna göre programın sonlandırdığını veya tekrar bir harf okuduğunu göreceğiz. Programı denediğimizde ise “t” dışındaki bir harfe bastığımız anda ekrana yazdırıldığını göreceğiz.

7.1.4.4.3. 01H Alt Yordamı

AH yazmacının değeri 00H olduğu durumda klavyeden karakter okunurken karakter basılana kadar kesme devam etmektedir. Yani yazdığımız bir program bir tuşa basılana kadar kilitlenmiştir. Bunun yerine AH yazmacının değerini 01H yaparak INT 16H kesmesini çağırırsak sadece anlık bir tuşa basılıp basılmadığını kontrol edebiliriz. Bir örnek ile 01 alt yordamını inceleyelim.

```
1  ORG 0100H
2  XOR CX,CX
3  ;;;;;;;;;;;Ekrana mesajı yazdır
4  MOV AH,03H
```

```
INT 10H
MOV AL,01H
MOV BH,00H
MOV BL,01001111B
MOV CX,MESAJ1_SON - MESAJ1_BAS
PUSH CS
POP ES
MOV BP, MESAJ1_BAS
MOV AH,13H
INT 10H
;;;;;;;;;;;;Klavyeyi oku
OKU: MOV AH,1
INT 16H
JZ ARTIR
MOV AH,00H
INT 16H
CMP AL,61H
JE SON
CMP AL,64H
JNE ARTIR
PUSH CX
CMP BYTE [SI],0FFH
JE YAZ
;;;;;;;;;;;;CX yazmacının değerini yaz
YAZ_ILK: MOV SI,MESAJ2_SON
MOV BYTE [SI],0FFH
INC DH
MOV DL,00H
MOV AL,01H
MOV BH,00H
MOV BL,01001111B
MOV CX,MESAJ2_SON - MESAJ2_BAS
PUSH CS
POP ES
MOV BP, MESAJ2_BAS
MOV AH,13H
INT 10H
POP CX
MOV AL,CH
MOV AH,0EH
INT 10H
MOV AL,CL
MOV AH,0EH
INT 10H
JMP ARTIR
YAZ: POP CX
MOV AH,02H
INT 10H
MOV AL,CH
MOV AH,0EH
INT 10H
MOV AL,CL
MOV AH,0EH
INT 10H
;;;;;;;;;;;;CX yazmacının değerini artır
ARTIR: INC CX
JMP OKU
JMP SON
MESAJ1_BAS DB 'CX YAZMACININ ASCII DEGERINI GOREMEK ICIN D, CIKMAK ICIN A HARFINE
BASINIZ!'
```

```
   MESAJ1_SON DB 00H
65 MESAJ2_BAS DB 'CX DEGERI:'
66 MESAJ2_SON DB 00H
67 SON: RET
```

3. ve 14. satır arasındaki kodlar ile ekrana "CX YAZMACININ ASCII DEGERINI GOREMEK ICIN D, CIKMAK ICIN A HARFINE BASINIZ!" dizgisi yazdırılıyor.

16. satırda AH yazmacına 01H değeri yükleniyor. Bu şekilde 17. satırda çağıracağımız INT 16H kesmesinin 01 alt yordamına erişiyoruz. INT 16H kesmesinin 01 numaralı alt yordamı çağırıldığı anda bir tuşa basılmış ise ZF bayrak bitini 1, basılmamış ise ZF bayrak bitini 0 değerinde tutar.

18. satırda eğer ZF bayrak biti 1 ise yani bir tuşa basılmamış ise *ARTIR* etiketine atlanır. Aksi takdirde yani bir tuşa basılmış ise devam edilir.

19. ve 20. satırdaki kodlar ile INT 16H kesmesinin 00H alt yordamına erişiyoruz. Bu işlemi yapmamızın nedeni klavyenin belleğinden karakterin ASCII kodunun AL yazmacına aktarılmasını sağlamak çünkü INT 16H kesmesinin 01H alt yordamı AL yazmacına klavyenin belleğindeki veriyi yazmaz.

21. satırda AL yazmacının değerini ASCII 'a' harfi ile karşılaştırıyoruz. Eğer eşit iseler 22. satırda belirtildiği gibi *SON* etiketine atlıyoruz.

23. satırda ise AL yazmacının değerini ASCC 'd' harfi ile karşılaştırıyoruz. Eğer eşit değilseler ARTIR etiketine 24. satırda belirtildiği gibi atlıyoruz. Eğer AL yazmacının değeri 'd' harfinin ASCII değerine eşit ise 25. satırdan devam ediliyor.

25. satırda CX yazmacının değerini yığına geçici olarak kaydediyoruz.

26. satırda ise SI yazmacının değerinin işaretlediği adresteki baytın FFH değerine eşit olup olmadığını kontrol ediyoruz. 27. satırda da eğer 26. satırda karşılaştırılan iki değer birbirine eşit ise *YAZ* etiketine atlanıyor.

29. satırda dikkat edilecek olursa *YAZ_ILK* etiketinin olduğu görülecektir. Böyle bir alt döngü oluşturmamızın nedeni ilk CX yazmacı değeri yazılırken ekrana "CX DEGERI:" dizgisinin yazdırılmasıdır. *MOV SI,MESAJ2_SON* kodu ile SI yazmacına *MESAJ2_SON* değişkeninin adresini yükledik. Bu değişkeni değerini yukarıda da belirttiğimiz gibi 26. satırdaki karşılaştırmada kullanacağız ve bu sayede ilk CX mesajının yazılıp yazılmadığını kontrol edebileceğiz.

30. satırda ilk CX değerini ilerleyen satırlarda yazacağımız için SI yazmacının değerinin işaret ettiği bellek bölgesine FFH değerini yüklüyoruz.

31. satırda DH yazmacının değerini 1 artırıyoruz. Bunu yapmamızın nedeni INT 10H kesmesinin 13H alt yordamının DH yazmacının değerini yazılacak dizginin y koordinatı değeri olarak kullanması ve bizim "CX DEGERI:" dizgisini "CX YAZMACININ ASCII DEGERINI GOREMEK ICIN D, CIKMAK ICIN A HARFINE BASINIZ!" mesajının altına yazmak isteyişimizdir.

32., 33., 34. ve 35. satırlarda DL, AL, BL ve BH yazmaçlarının değerlerini 13H alt yordamı için düzenledik.

36. satırda CX yazmacına yazdıracağımız dizginin uzunluğunu hesaplayıp kaydettik.

37. satırda CS yazmacının değerini yığına kaydettik ve hemen ardından 38. satırda bu değeri yığından alıp ES yazmacına kaydettik. 39. satırda ise BP yazmacına dizginin ilk baytının adresini kaydettik. 37.,38. ve 39. satırlardaki işlemleri yapmamızın nedeni 13H alt yordamının ES:BP adresindeki dizgiyi yazdırmasıdır. Yazdırılacak dizginin uzunluğu ise CX yazmacında tutulmaktadır.

40. ve 41. satırlar ile INT 10H kesmesinin 13H alt yordamını çağırıyoruz. Böylece "CX DEGERI:" dizgimizi yazdırmış olduk.

Şimdi sırada yapılması gereken şey 25. satırda *PUSH CX* ile yığına kaydettiğimiz CX yazmacı değerini geri yüklemek ve ekrana yazdırmak. Bunun için:

42. satırda *POP CX* ile CX yazmacının 25. satırda *PUSH CX* ile yığına kaydedilen değerini geri çağırıyoruz.

43. satırda AL yazmacına CH yazmacının değerini kaydediyoruz. Ardından 44. ve 45. satırlar ile INT 10H kesmesinin 0EH alt yordamını çağırıp ekrana AL yazmacının değerini yani CX yazmacının üst baytının değerini yazdırıyoruz.

46. satırda ise AL yazmacına CL yazmacının değerini kaydediyoruz ve hemen ardından 47. ve 48. satırlarda INT 10H kesmesinin 0EH alt yordamını çağırarak ekrana AL yazmacının değerini yazdırıyoruz.

49. satırda değer yazma işlemlerimiz tamamlandığından *ARTIR* etiketine atlıyoruz.

50. satırda YAZ etiketinin olduğunu göreceğiz. Bu sadece değer yazdırmak için yani değerden önce herhangi bir dizgi yazdırmayacağımız zaman kullanacağımız alt döngünün etiketidir. Bu satırda CX yazmacını yığındaki veriyle yani 25. satırdaki *PUSH CX* ile yığına kaydedilen veriyle yüklüyoruz.

51. ve 52. satırlar ile INT 10H kesmesinin 02H alt yordamını çağırıyoruz. Hatırlanacak olursa bu alt yordam imleç konumunu almaya yarayan bir alt yordam idi. BH yazmacına sayfa numarası, DH yazmacına y koordinatı ve DL yazmacına x koordinatı bu alt yordam aracılığıyla kaydedildi. Her yazma işleminde bu alt yordamı çağırmamızın gereksiz olabileceği düşünebilir ama program diğer işlemleri yaparken DL ve DH yazmaçlarının değerleri değişmiş ise istenmeyen yerlere değerlerin yazılması gibi durumlar oluşabilir, bu yüzden bu tür durumları önlemek amacıyla bu alt yordamı her yazma işleminde çağırıyoruz.

53., 54. ve 55. satırlardaki komutlar ile CH yazmacının değerini INT 10H kesmesinin E0H alt yordamını kullanarak yazdırıyoruz.

Aynı şekilde 56., 57. ve 58. satırlardaki komutlarla da CL yazmacının değerini INT 10H kesmesinin E0H alt yordamını kullanarak yazdırıyoruz.

60. satırda ise CX yazmacının değerini 1 artırıyoruz ve ardından herhangi bir tuşa basılıp basılmadığını kontrol etmek amacıyla *OKU* etiketine atlıyoruz.

7.1.4.4.4. 02H alt Yordamı

INT 16H kesmesinin 02H alt yordamı çağırıldığında AL yazmacı ile hatırlayacağınız gibi basılan kontrol tuşları hakkında bilgi vermektedir.

Bu alt yordamın nasıl çalıştığını görmek için aşağıdaki kodu kullanabiliriz:

```
1  ORG 0100H
2  MOV AH,02H
3  INT 16H
4  INT 03H
```

Kodu denemeden önce “Caps Lock” veya diğer kontrol tuşlarına basılacak olursa AL yazmacı üzerindeki etkisi bu şekilde gözlemlenebilir.

7.1.4.5. INT 14H ve Seri Kapı

INT 14H kesmesi kullanılarak seri kapı üzerinde işlemler yapılabilir. Veri alınıp gönderilebilir. Alt yordamları inceleyelim:

<table>
<tr><th>AH</th><th>İşlev</th></tr>
<tr><td>00H</td><td>Seri kapı başlatılması.

Girdi olarak AL yazmacı verilir. AL yazmacının değerine göre seri kapı ayarlanır ve başlatılır.
Çıktı olarak herhangi bir yazmaç verilmez.
<table>
<tr><th>Bit</th><th>İşlev</th></tr>
<tr><td>7
6
5</td><td>
<table>
<tr><th>Bitler (567)</th><th>Bilgi Akış Hızı (Baud)</th></tr>
<tr><td>000</td><td>110</td></tr>
<tr><td>001</td><td>150</td></tr>
<tr><td>010</td><td>300</td></tr>
<tr><td>011</td><td>600</td></tr>
<tr><td>100</td><td>1200</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>
<table>
<tr><td>101</td><td>2400</td></tr>
<tr><td>110</td><td>4800</td></tr>
<tr><td>111</td><td>9600</td></tr>
</table>
</td></tr>
<tr><td>4
3</td><td>
<table>
<tr><th>Bitler (34)</th><th>Eşlik Çeşidi</th></tr>
<tr><td>00</td><td>Eşlik yok</td></tr>
<tr><td>01</td><td>Tek eşlik</td></tr>
<tr><td>10</td><td>Eşlik yok</td></tr>
<tr><td>11</td><td>Çift eşlik</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>Sonlandırma biti sayısı.

0- Bir tane sonlandırma biti.
1- İki tane sonlandırma biti</td></tr>
<tr><td>1
0</td><td>
<table>
<tr><th>Bitler (01)</th><th>Karakter Boyutu</th></tr>
<tr><td>10</td><td>7 bit</td></tr>
<tr><td>11</td><td>8 bit</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>01H</td><td>Seri kapıdan karakter gönder.

Girdi olarak AL ve DX yazmaçları kullanılır. DX yazmacında kullanılmak istenen seri kapının numarası bulunmalıdır. AL yazmacındaki veri gönderilecek olan veridir.
Çıktı olarak AH yazmacı kullanılır. AH yazmacında 00H değeri var ise veri başarıyla gönderilmiş demektir. Eğer AH yazmacının 7. bitinde 1 var ise veri transferinde bir hata oluştuğu belirtilmiş olur.</td></tr>
<tr><td>02H</td><td>Seri kapıdan karakter al.

Girdi olarak DX yazmacı kullanılır. DX yazmacında kullanılmak istenen seri kapının numarası bulunmalıdır.
Çıktı olarak AL yazmacı kullanılır. AL yazmacında alınan veri bulunmaktadır.</td></tr>
<tr><td>03H</td><td>Seri kapı durumu.

Girdi olarak DX yazmacını alır. DX yazmacında durumu istenen seri kapının değeri bulunmalıdır.
Çıktı olarak AX yazmacını verir.
<table>
<tr><th>Bitler</th><th>İşlev</th></tr>
<tr><td>15</td><td>Zaman aşımı hatası</td></tr>
<tr><td>14</td><td>Verici öteleme yazmacı boş</td></tr>
<tr><td>13</td><td>Verici tutucu yazmacı boş</td></tr>
<tr><td>12</td><td>Duraklama tespit hatası</td></tr>
<tr><td>11</td><td>Kalıp hatası</td></tr>
<tr><td>10</td><td>Eşlik hatası</td></tr>
<tr><td>9</td><td>Veri taşma hatası</td></tr>
<tr><td>8</td><td>Veri mevcut</td></tr>
<tr><td>7</td><td>Alıcı hat sinyali</td></tr>
<tr><td>6</td><td>RI (modemlere özgü)</td></tr>
</table>
</td></tr>
</table>

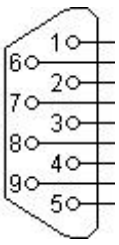
		5	DSR (**D**ata **S**et **R**eady)	
		4	CTS (**C**lear **T**o **S**end)	
		3	DRSD (modemlere özgü)	
		2	TERD (modemlere özgü)	
		1	DDSR (modemlere özgü)	
		0	DCTS (modemlere özgü)	

Not: Hangi seri kapının kullanıldığı yukarıdaki alt yordamlar çağırılırkenki DX yazmacı değerine bağılıdır. DX yazmacı 0 ile 3 arasında değerler alabilir. Yani COM1, COM2, COM3 veya COM4 olabilir.

RS232 standardı hakkında kısa bir bilgi verelim. RS232'yi özellikle incelememizin nedeni günümüzde bilgisayarlarda genellikle seri kapı olarak RS232 bulunmasıdır.

1969 yılında EIA (**E**lectronic **I**ndustries **A**lliance) tarafından seri iletişim için RS232-C standardı oluşturulmuştur. Bu standartta voltaj değerleri, sinyaller, kapasitans, kablo uzunluğu, kapıların tipleri ve iğne bilgileri belirlenmiştir. Standartta hiçbir şekilde veri iletim için herhangi bir karakter kodlama sınırı konmamıştır. Hata belirme hakkında herhangi bir kural yoktur. Ayrıca veri sıkıştırma üzerine de herhangi bir şey belirtilmemiştir. Aynı şekilde veri iletişim hızı hakkında da herhangi bir kısıtlama yapılmamıştır.

Aşağıda DB9 tipinde dişi RS232 konnektörü ve iğne bilgileri verilmiştir:

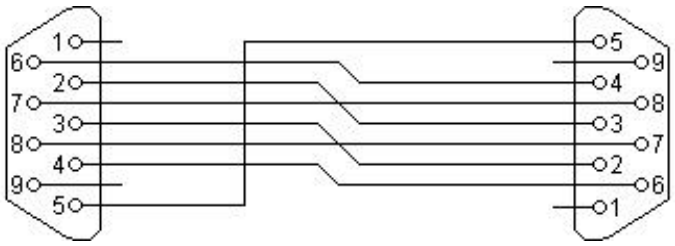


1. data carrier detect
2. receive data
3. transmit data
4. data terminal ready
5. signal ground
6. data set ready
7. request to send
8. clear to send
9. ring indicator

Veri alışverişinde karakter kodlaması, hız gibi bilgiler seri kapı donanımı tarafından kontrol edilmektedir. Genellikle bu donanım tek bir UART (**U**niversal **A**synchronous **R**eceiver/**T**ransmitter) entegre devresinden oluşmaktadır. Bu devre paralel olarak gelen veriyi seriye çevirmektedir.

Bu kesmeyi denemek için 2 farklı program yazalım. Bu programlardan biri bir bilgisayara yüklensin, diğeri de başka bir bilgisayara yüklensin. Bu iki bilgisayarı bir seri kablo ile birbirlerine bağlayalım.

Not: Seri kablonun "Null Modem" tipinde olması gerekmektedir. Eğer elinizde "Null Modem" tipinde seri kablo yok ise kablonun bir ucundaki seri kapının iğnelerinde aşağıdaki değişikliği yapmanız gerekmektedir. Aşağıdaki RS232 kapıları DB9 tipinde dişi kapılardır.



İlk olarak biri dinleyici biri yollayıcı olmak üzere 2 tane program yazalım. Dinleyici program seri kapıdan gelen verileri ekrana yazacak, yollayıcı ise sürekli olarak "S" harfini seri kapıdan yollayacaktır. Dinleyici COM1'e, yollayıcı ise COM2'ye bağlanmalıdır.

İlk olarak veri yollayıcıyı inceleyelim:

```
1  ;YOLLAYICI PROGRAM
2  ORG 0100H
3  MOV DX,0001H          ;COM2
4  MOV AH,00H            ;SERI KAPI AYARLAMA
5  MOV AL,00000010B      ;110 BAUD - 7 BİT VERİ - PARİTE BİTİ YOK
```

```
6                   ;1 TANE DURAKLAMA BİTİ
7   INT 14H
8   SERI_YOLLA: MOV AL, 'S' ;AL yazmacına S harfini yaz
9   MOV AH,01H          ;veri yollama için gerekli AH değerini yaz
10  INT 14H
11  TEST AH,00H         ;hata oluşup oluşmadığını kontrol et
12  JZ SERI_YOLLA       ;oluşmadıysa SERI_YOLLA ya geri don
13  SON: RET
```

Üçüncü ve yedinci satır arasındaki işlemler ile seri iletişimin kapısı, hızı, veri biti sayısı, eşlik biti sayısı ve duraklama biti sayısı ayarlandı. Yukarıdaki tablodan 00H alt yordamının parametrelerine bakacak olursak 110 baud veri iletişim hızı, 7 bit veri, 0 eşlik biti ve 1 tane duraklama biti ayarlarını yaptığımızı göreceksiniz. Seri iletişim kapısı olarak COM2 ayarlanmıştır.

Sekizinci satırda AL yazmacına 'S' harfinin ASCII değerini kaydediyoruz.

Dokuzuncu satırda AH yazmacına 01H değerini yükleyerek onuncu satırda atlayacağımız INT 14H kesmesinin AL yazmacındaki veriyi ayarlara göre iletmesini sağlıyoruz.

On birinci satırda AH yazmacının değerinin 00H değerine eşit olup olmadığına bakıyoruz. Eğer AH yazmacının değeri bu değere eşit ise veri iletişiminde herhangi bir hata oluşmamış demektir. Aksi halde bir hata oluşmuştur. Hata oluşmadıysa *SERI_YOLLA* etiketine atlanır. Hata oluştuysa program sonlandırılır.

Şimdi bir de dinleyiciyi inceleyelim:

```
1   ;DINLEYICI PROGRAM
2   ORG 0100H
3   MOV DX,0000H        ;COM1
4   MOV AH,00H          ;SERI KAPI AYARLAMA
5   MOV AL,00000010B    ;110 BAUD - 7 BİT VERİ - PARİTE BİTİ YOK
6                   ;1 TANE DURAKLAMA BİTİ
7   INT 14H
8   XOR AL,AL           ;AL yazmacının değerini 00H yap
9   SERI_DINLE: MOV AH,02H  ;AH yazmacına 02H değerini kaydet
10  INT 14H             ;INT 14H kesmesine atla
11  TEST AH,00H         ;AH yazmacının değerini 00H ile karşılaştır
12  JZ MESAJ_YAZ        ;MESAJ_YAZ etiketine atla
13  SON: RET
14
15  MESAJ_YAZ:
16  PUSH DX             ;DX yazmacını yığına at
17  MOV AH,0EH          ;AH yazmacına 0EH değerini yaz
18  INT 10H             ;INT 10H kesmesine atla
19  POP DX              ;DX yazmacına yığındaki veriyi kaydet
20  JMP SERI_DINLE      ;SERI_DINLE etiketine atla
```

Karakter yollayıcı programımızda olduğu gibi üçüncü ve yedinci satır arasındaki işlemler ile seri iletişimin ayarlarını yaptık. Seri iletişim kapısı olarak COM1 ayarlanmıştır. Veri iletişim hızı 110 baud, duraklama biti sayısı 1, eşlik biti sayısı 0 ve veri biti sayısı 7 olarak ayarlanmıştır.

Sekizinci satırdaki işlemle AL yazmacının değeri 00H değerine eşitlenmiştir. Bunun yerine *MOV AL,00H* komutu da kullanılabilirdi. Ama *XOR AL,AL MOV AL,00H* a göre daha hızlıdır.

Dokuzuncu ve onuncu satırdaki işlemlerle yani AH yazmacına 02H değerini atayıp INT 14H kesmesine atlayarak seri kapıdan veri almak istediğimizi belirttik.

On birinci satırdaki *TEST AH,00H* komutu ile AH yazmacının değerinin 00H değerine eşit olup olmadığını kontrol ettik. Eşit ise hata oluşmamış, değilse hata oluşmuştur. Eğer hata oluşmamış ise on ikinci satırdaki *JZ MESAJ_YAZ* komutu on beşinci satırdaki *MESAJ_YAZ* etiketine atlanılır.

On altıncı satırdaki *PUSH DX* komutu ile DX yazmacının değerini yığına kaydettik. Bunu yapmamızın nedeni on sekizinci satırdaki *INT 10H* komutunun DX yazmacının değerini değiştirmesidir. Her mesaj yazımı ardından kullanılacak olan seri kapıyı belirleyen DX yazmacına seri kapı değerini atamak yerine PUSH ve POP kullanmamız performans açısından bir kazanç sağlamaktadır.

On yedinci satırda AH yazmacına 0EH değerini kaydettik ve on sekizinci satırdaki INT 10H komutu ile ekrana AL yazmacındaki veriyi yazdırdık.

On dokuzuncu satırdaki POP ile DX yazmacına eski değeri geri kaydettik ve yirminci satırdaki kod sayesinde *SERI_DINLE* etiketine geri atladık.

İleriki bölümlerde girdi/çıktı komutları ile seri iletişimi işledikten sonra daha detaylı bir veri iletişim örneği yapacağız. Burada seri iletişime fazla değinilmemesinin nedeni bu bölümün amacının kesme mantığının ve kesmelerin kullanımlarının açıklanmasıdır.

7.1.4.6. INT 17H ve Yazıcı

INT 17H BIOS kesmesi ile yazıcı kontrol edilebilir. Aşağıda bu kesmenin alt yordamları ve işlevleri listelenmiştir.

<table>
<tr><th>AH</th><th>İşlev</th></tr>
<tr><td>00H</td><td>Bir karakter yazdırır.

Girdi olarak AL yazmacı kullanılır. AL yazmacındaki karakter yazıcıdan yazdırılır.
Çıktı olarak AH yazmacı verilir. Yazıcının durum bilgisi AH yazmacına kaydedilir.</td></tr>
<tr><td>01H</td><td>Yazıcıyı hazırlar.

Girdi olarak herhangi bir yazmaç almaz.
Çıktı olarak herhangi bir yazmaç verilmez.</td></tr>
<tr><td>02H</td><td>Yazıcının durum bilgilerini okur.

Girdi olarak herhangi bir yazmaç almaz.
Çıktı olarak AH yazmacı verilir. AH yazmacında yazıcının durum bilgisi kayıtlıdır.
<table>
<tr><th>Bit</th><th>İşlev</th></tr>
<tr><td>7</td><td>1- yazıcı meşgul 0- yazıcı meşgul değil</td></tr>
<tr><td>6</td><td>1- yazıcıdan tanınma bilgisi geldi</td></tr>
<tr><td>5</td><td>1- yazıcıda kağıt kalmadı</td></tr>
<tr><td>4</td><td>1- yazıcı seçildi</td></tr>
<tr><td>3</td><td>1- girdi/çıktı hatası</td></tr>
<tr><td>2</td><td>Kullanımda değil.</td></tr>
<tr><td>1</td><td>Kullanımda değil.</td></tr>
<tr><td>0</td><td>Zaman aşımı hatası</td></tr>
</table></td></tr>
</table>

7.1.4.7. INT 19H

INT 19H kesmesini kullanarak sistem baştan başlatılabilir. INT 19H'ı detaylı anlatmadan önce başlatılma şekillerini inceleyelim. 2 tip başlatılma şekli vardır: Sıcak başlangıç ve soğuk başlangıç.

Bilgisayar kapalı iken bilgisayarı açarsak "soğuk başlangıç" yapmış oluruz.

Ctrl+Alt+Del tuşlarına basarak bilgisayar yeniden başlatılırsa "sıcak başlangıç" yapılmış olur. Bu başlangıç şeklinde bellek sınamaları dışında bütün sistem sınamaları (Power On, POST, vb.) yapılmaktadır. Kesme vektörleri ve sistem zamanlayıcıları yeniden yüklenir. Bu şekilde sistem işletim sistemi başlatmaya uygun hale getirilir.

Bilgisayarın hangi şekilde başlatılacağı BIOS ROM'undaki bir yeniden başlatılma bayrağının değerine bağlıdır. Bayrağın değeri 1234H olursa sıcak başlangıç yapılır. Diğer tüm değerler soğuk başlangıç yapılmasına neden olur.

INT 19H'ın tek başına kullanılması ile sadece işletim sistemi yeniden yüklenir ve bazı durumlarda (ilerideki bölümlerde kendi işletim sistemimizi yazarken bu tür sorunlarla karşılaşacağız) işletim sistemi yüklenmeye hazır olmayabilir. INT 19H çağırıldığında kesme vektörleri yeniden yüklenmez. Eğer bazı kesmeler üzerinde işlemler yapılmış ise INT 19H ile işletim sistemi yüklendikten sonra bu değiştirilmiş kesmeler çağırıldığında istenmeyen sonuçlar elde edilebilir. Ayrıca zamanlayıcıların ve bazı donanımlar baştan yüklenmediğinden sorunlar oluşabilir. Bu yüzden INT 19H tek başına kullanılmamalıdır. Biz sistemi yeniden başlatmak için başka bir yöntem kullanacağız.

7.1.4.8. INT 1AH ve RTC

INT 1AH ile RTC (**R**eal **T**ime **C**lock – Gerçek Zamanlı Zamanlayıcı) ve sistem zamanlayıcısı yönetilebilir. Sistem zamanlayıcısı saniyenin 1/18 inde bir 1 artar (saniyede yaklaşık 18.2 kez). Sistem zamanlayıcısı bilgisayar açıldığında veya yeniden başlatıldığında sıfırlanır. RTC sistem zamanlayıcısı gibi saniyede yaklaşık 18.2 kez artar. RTC bir CMOS pili ile beslendiğinden bilgisayar kapatılsa bile zaman değerlerini korur. Bilgisayar tarafından sistem zamanlayıcısı kullanılmasına rağmen sistem zamanlayıcısı ile birlikte RTC de sürekli güncellenir. Birinin değerini değiştirmek diğerini etkilemez. Aşağıda INT 1AH'ın alt yordamları listelenmiştir. AH yazmacının değerine göre alt yordam seçilir ve işlenir.

AH	İşlev
00H	Sistem zamanlayıcı zamanını oku Girdi olarak herhangi bir yazmaç almaz. Çıktı olarak CX, DX ve AL yazmaçlarını verir. CX yazmacına zaman bilgisinin üst bitleri yazılır. DX yazmacına zaman bilgisinin alt bitleri yazılır. AL yazmacının değeri 24 saat geçmediyse 0'dır. 24 saat geçtiyse 1 e eşittir. **Not 1**: Sistem zamanlayıcısı bilgisayar açıldığı anda çalışmaya başlar. Yeniden başlatıldığında sıfırlanır. **Not 2:** Bu alt yordam çağırıldıktan sonra 0000:046C adresinde bulunan 4 baytlık veri CX:DX yazmacındaki veri ile aynı veri yani zamanlayıcı verisidir. **Not 3:** Bir günün geçtiğini belirten 0000:0470 adresindeki bayrak bu alt yordam tarafından otomatik olarak sıfırlanır.
01H	Sistem zamanlayıcı zamanını ayarla Girdi olarak CX ve DX yazmaçlarını alır. CX yazmacına zaman bilgisinin üst bitleri bulunmalıdır. DX yazmacına zaman bilgisinin alt bitleri bulunmalıdır. Çıktı olarak herhangi bir yazmaç kullanılmaz. **Not 1:** Bu alt yordamı çağırmadan önce 0000:046C adresine 4 baytlık zaman verisi kaydedilirse, alt yordam çağırıldığında bu veri zamanlayıcı verisi olarak kaydedilir. **Not 2:** Bir günün geçtiğini belirten 0000:0470 adresindeki bayrak bu alt yordam tarafından otomatik olarak sıfırlanır.
02H	RTC zamanını oku

	Girdi olarak herhangi bir yazmaç kullanılmaz. Çıktı olarak CH,CL,DH,DL yazmaçları ile CF bayrak biti kullanılır. CF bayrak biti zamanlayıcı çalışmıyorsa 1 eşittir. Aksi halde sıfırlanır. CH yazmacına BCD kodlamasında saat değeri yazılır. CL yazmacına BCD kodlamasında dakika değeri yazılır. DH yazmacına BCD kodlamasında saniye değeri yazılır. DL yazmacının değeri güneş ışığından yararlanma sistemi açık ise birdir. Aksi halde sıfırlanır.
03H	RTC zamanını ayarla Girdi olarak CH,CL,DH ve DL yazmaçlarını alır. CH yazmacına BCD kodlamasında saat değeri yazılmalıdır. CL yazmacına BCD kodlamasında dakika değeri yazılmalıdır. DH yazmacına BCD kodlamasında saniye değeri yazılmalıdır. DL yazmacına ise güneş ışığından yararlanma sistemi kullanılmak isteniyorsa 1 değeri aksi halde 0 değeri yazılmalıdır. Çıktı olarak herhangi bir yazmaç kullanılmaz.
04H	RTC tarihini oku Girdi olarak herhangi bir yazmaç kullanılmaz. Çıktı olarak CH,CL,DH,CL yazmaçları ile CF bayrak biti kullanılır. CF bayrak bitine RTC çalışmıyorsa 1 değeri verilir. Aksi takdirde sıfırlanır. CH yazmacına BCD kodlamasında yüzyıl değeri yazılır. (20. yüzyıl için 20) CL yazmacına BCD kodlamasında yıl değeri yazılır. DH yazmacına BCD kodlamasında ay değeri yazılır. DL yazmacına BCD kodlamasında gün değeri yazılır.
05H	RTC tarihini ayarla Girdi olarak CH,CL,DH ve DL yazmaçları kullanılır. CH yazmacında BCD kodlamasında yüzyıl değeri yazılır. CL yazmacına BCD kodlamasında yıl değeri yazılır. DH yazmacına BCD kodlamasında ay değeri yazılır. DL yazmacına BCD kodlamasında gün değeri yazılır. Çıktı olarak herhangi bir yazmaç kullanılmaz.
06H	RTC alarmını ayarla Girdi olarak CH, CL ve DH yazmaçlarını kullanılır. CH yazmacına BCD kodlamasında saat değeri yazılır. CL yazmacına BCD kodlamasında dakika değeri yazılır. DH yazmacına BCD kodlamasında saniye değeri yazılır. Çıktı olarak CF bayrak biti verilir. CF bayrak biti RTC çalışmıyorsa veya alarm önceden ayarlanmışsa 1 değerine sahiptir. Diğer durumlarda sıfırdır. **Not 1:** Alarm INT 4AH kesmesinin çağırılmasıyla çalmaya başlar. **Not 2:** Bilgisayar kapalı iken alarm ayarlı olsa bile çalmaz.
07H	RTC alarmını kapat Girdi olarak herhangi bir yazmaç kullanılmaz. Çıktı olarak herhangi bir yazmaç kullanılmaz.

<table>
<tr><td></td><td></td></tr>
<tr><td>08H</td><td>Bilgisayarın otomatik açılma zamanı.

Girdi olarak CH,CL ve DH yazmaçları kullanılır.
CH yazmacına BCD kodlamasında saat değeri yazılır.
CL yazmacına BCD kodlamasında dakika değeri yazılır.
DH yazmacına BCD kodlamasında saniye değeri yazılır.
Çıktı olarak CF bayrak biti kullanılır.
CF bayrak biti RTC çalışmıyorsa veya otomatik açılma zamanı önceden ayarlandıysa 1 değerini alır. Diğer durumlarda sıfırlanır.</td></tr>
<tr><td>09H</td><td>Alarm zamanını ve durumunu oku

Girdi olarak herhangi bir yazmaç kullanılmaz.
Çıktı olarak CH,CL,DH,DL yazmaçları ve CF bayrak biti kullanılır.
CF bayrak biti RTC çalışıyorsa 1 değerine sahip olur. Diğer durumlarda sıfırlanır.
CH yazmacına BCD kodlamasında saat değeri yazılır.
CL yazmacına BCD kodlamasında dakika değeri yazılır.
DH yazmacına BCD kodlamasında saniye değeri yazılır.
CL yazmacına ise alarm durumu yazılır.

<table><tr><th>CL Değeri</th><th>Anlam</th></tr><tr><td>00H</td><td>Alarm açık değil</td></tr><tr><td>01H</td><td>Alarm açık ama sistemi kapalı ise açmaz</td></tr><tr><td>02H</td><td>Alarm açık ve sistemi kapalı ise açar</td></tr></table></td></tr>
<tr><td>0AH</td><td>Sistem zamanlayıcı tarihini oku

Girdi olarak herhangi bir yazmaç almaz.
Çıktı olarak CX yazmacı ve CF bayrak biti kullanılır.
CF bayrak bitine bir hata oluşmuş ise 1 yazılır. Aksi halde 0 yazılır.
CX yazmacına 1 Ocak 1980 tarihinden beri geçen gün sayısı yazılır.</td></tr>
<tr><td>0BH</td><td>Sistem zamanlayıcı tarihini ayarla

Girdi olarak CX yazmacı kullanılır.
CX yazmacına 1 Ocak 1980 tarihinden beri geçen gün sayısı yazılır.</td></tr>
</table>

Not: Yukarıdaki alt yordamlardan bazıları sistemin özelliklerine bağlı olarak bulunmayabilir. İlk 8 alt yordam çoğu sistemde bulunmaktadır.

RTC zamanı ve tarih bilgilerini aşağıdaki şekilde okuyabiliriz:

7.1.4.9. INT 11H

INT 11H yüklü sistem birimlerini listeler. Girdi olarak herhangi bir yazmaç almazken çıktı olarak AX yazmacını verir.

AX Bitleri	Anlam

<table>
<tr><td>15
14</td><td>Bağlı yazıcı sayısı.</td></tr>
<tr><td>13</td><td>1- Seri yazıcı bağlı
2- Seri yazıcı bağlı değil</td></tr>
<tr><td>12</td><td>1- Oyun adaptörü bağlı
2- Oyun adaptörü bağlı değil</td></tr>
<tr><td>11
10
9</td><td>Yüklü RS232 kartı sayısı</td></tr>
<tr><td>8</td><td>1- DMA var
2- DMA yok</td></tr>
<tr><td>7
6</td><td>Bağlı disket sürücü sayısı</td></tr>
<tr><td>5
4</td><td>Video kipi
<table>
<tr><th>Bitler (45)</th><th>Anlam</th></tr>
<tr><td>00</td><td>Yok</td></tr>
<tr><td>01</td><td>40x25 Renkli</td></tr>
<tr><td>10</td><td>80x25 Renkli</td></tr>
<tr><td>11</td><td>80x25 Siyah/Beyaz</td></tr>
</table></td></tr>
<tr><td>3
2</td><td>Kullanımda değil.</td></tr>
<tr><td>1</td><td>1- Matematiksel yardımcı işlemci mevcut
2- Matematiksel yardımcı işlemci mevcut değil</td></tr>
<tr><td>0</td><td>1- Disket sürücüsünde disket var
2- Disket sürücüsünde disket yok</td></tr>
</table>

Not: AX yazmacına yazılan yukarıdaki bilgilere aynı zamanda kesme çağırıldıktan sonra 0000:0410 adresindeki kelime okunarak da ulaşılabilir.

7.1.4.10. INT 13H – Alt Seviye Disk Servisleri

INT 13H kesmesi diskler üzerinde işlemler yapabiliriz. INT 13H kesmesi ilk yazıldığında 8GB'lık disk boyutu limiti vardı. Böyle bir limit bulunmasının nedeni pist başına 1024 silindir, 256 başlık ve 63 sektör bulunmasıdır. Her sektörde 512 bayt olduğundan:

1024 x 256 x 63 x 512 = 8455716864 bayt = 7,875GB

(1024bayt = 1kB 1024kB = 1MB 1024MB = 1GB)

Günümüzde piyasada 8GB'lık sabit diskler bulunamamaktadır. Hatta 120GB altında sabit disket bile bulunamamaktadır. Peki 8GB üzerine erişim nasıl sağlanmaktadır? 1990'lı yıllarda bu INT 13H kesmesine ek bazı yordamlar (INT 13H EXT – **Ext**ended - Genişletilmiş) eklendi. Bu yordamlar sayesinde yüksek boyutlarda sabit diskler üzerinde işlemler yapılabilmektedir. Daha detaylı bilgi için Ek D'ye bakınız.

Şimdi INT 13H alt yordamlarını kullanarak bir disket sürücü biçimlendiricisi yazalım. Tabii ki sürücü biçimlendiriciyi yazmadan önce disket sürücüyü, kullanacağımız alt yordamları ve biçimlendirme işlemlerini inceleyelim.

Disket Sürücü

Disket sürücü manyetik bir depolama birimidir yani kutuplamalara göre 0 veya 1 (kuzey-güney) değerleri depolanmaktadır. Disket sürücüden bir okuma/yazma başlığı ile manyetik endüksiyon kullanılarak kutuplamalar okunabilir veya manyetikleştirme işlemleri ile veri yazılabilir.

Disketlerin içi pistlere ve sektörlere ayrılmışlardır. Pistler çevresel bölmelerdir. Sektörler ise disket üzerinde tanımlanan açısal bölmelerdir. Disketler bir veya iki yüzlü olabilirler. İki yüzlü olduklarında doğal olarak kapasiteleri ikiye katlanmaktadır. Bir pist ve sektörün kesişmesiyle oluşan bölmelere göreli sektörler denir.

Pistler dıştan başlayarak numaralandırılırlar. Yani toplam pist sayısı n ise en dıştaki pistin numarası 0, en içteki pistin numarası n-1'dir. m sayısı biçimlendirilmiş sektör sayısı ise sektörler 1'den başlayarak m e kadar numaralandırılabilir.

Bu bilgiler sayesinde bir disketin kapasitesini rahatça hesaplayabiliriz:

Disketin kapasitesi = C x n x m x A

Disketteki yüz sayısı: C
Bir yüzde bulunan pist sayısı: n
Pist başına sektör sayısı: m
Göreli sektör boyutu: A

Göreli sektör sayısını ise aşağıdaki şekilde hesaplayabiliriz:

Göreli sektör sayısı: C x n x m

Disketteki yüz sayısı: C
Bir yüzde bulunan pist sayısı: n
Pist başına sektör sayısı: m
Göreli sektör boyutu: A

Günümüzde piyasada bulunan disket sürücüler genellikle:
3"1/2 HD (**H**igh **D**ensity – Yüksek Yoğunluk) tipinde,
2 yüzlü,
Pist başına 18 sektörlü,
Yüz başına 80 pistli,
Disket başına 2880 sektörlü
1440kB kapasitesinde gibi özelliklere sahip olan disket sürücülerdir. Bu tip disketlerin tanımlama kodları F0H'dır.

Disket sürücüleri yönetmek için kullanabileceğimiz INT 13H alt yordamları:

AH	İşlev
00	Yeniden başlat. Disket sürücüyü ilk açılışta başlatmak için veya herhangi bir hata oluştuğunda yeniden başlatmak için kullanılır. Girdi olarak DL yazmacı kullanılır. DL yazmacına kullanılacak olan sürücünün numarası yazılmalıdır. Çıktı olarak AH ve AL yazmaçları ile CF bayrak biti kullanılır. Hata var ise CF bayrak biti 1'e eşitlenir. Hata yoksa CF bayrak biti 0'a eşitlenir. AH yazmacında durum belirtilir. (INT 13H için AH yazmacı durum anlam tablosuna ek D'den bakınız)
01	

	Sürücü durumunu oku. Disket sürücünün durumunu okuyabilmek için kullanılır. Girdi olarak DL yazmacı alınır. DL yazmacına sürücü numarası kaydedilmelidir. Çıktı olarak AH ve AL yazmaçları ile CF bayrak biti kullanılır. Hata var ise CF bayrak biti 1'e eşitlenir. Hata yoksa CF bayrak biti 0'a eşitlenir. AH yazmacında durum belirtilir. (INT 13H için AH yazmacı durum anlam tablosuna ek D'den bakınız)
02	Disket sürücüyü oku. Disket sürücüyü okur ve bir veya daha fazla sektörü belleğe kaydeder. Girdi olarak AL,CH,CL,DH,DL yazmaçlarını ve ES:BX adresini alır. AL yazmacına okunacak sektör sayısı yazılmalıdır. CH yazmacında silindir numarasının alt 8 biti bulunmalıdır. CL yazmacında sektör numarası bulunmalıdır. Üst 2 bitinde sadece sabit disketlere mahsus olmak üzere silindir numarasının üst 2 biti kaydedilmelidir. DH yazmacına başlık numarası kaydedilmelidir. DL yazmacına sürücü numarası kaydedilmelidir. ES:BX adresi veri tamponu olarak kullanılır. Çıktı olarak AH ve AL yazmaçları ile CF bayrak biti kullanılır. Hata var ise CF bayrak biti 1'e eşitlenir. Hata yoksa CF bayrak biti 0'a eşitlenir. AH yazmacında durum belirtilir. (INT 13H için AH yazmacı durum anlam tablosuna ek D'den bakınız)
03	Disket sürücüye yaz. Disket sürücüye bellekten bir veya daha fazla sektörlük veri aktarır. Girdi olarak AL,CH,CL,DH,DL yazmaçlarını ve ES:BX adresini alır. AL yazmacında yazılacak sektör sayısı bulunmalıdır. CH yazmacında silindir numarasının alt 8 biti bulunmalıdır. CL yazmacında sektör numarası bulunmalıdır. Üst 2 bitinde sadece sabit disketlere mahsus olmak üzere silindir numarasının üst 2 biti kaydedilmelidir. DH yazmacına başlık numarası kaydedilmelidir. DL yazmacına sürücü numarası kaydedilmelidir. ES:BX adresi veri tamponu olarak kullanılır. Çıktı olarak AH yazmacı ve CF bayrak biti verilir. CF bayrak biti hata varsa 1 değerine, hata yoksa 0 değerine sahip olur. AH yazmacında durum belirtilir. (INT 13H için AH yazmacı durum anlam tablosuna ek D'den bakınız)
04	Disket sürücü sektör sınama. Disket sürücüde bulunan bir veya daha fazla sektörü sınar. Girdi olarak AL,CH,CL,DH,DL yazmaçları ile ES:BX adresi kullanılır. AL yazmacına sınanacak sektör sayısı yazılmalıdır. CH yazmacında silindir numarasının alt 8 biti bulunmalıdır. CL yazmacında sektör numarası bulunmalıdır. Üst 2 bitinde sadece sabit disketlere mahsus olmak üzere silindir numarasının üst 2 biti kaydedilmelidir. DH yazmacına başlık numarası kaydedilmelidir. DL yazmacına sürücü numarası kaydedilmelidir. ES:BX adresi veri tamponu olarak kullanılır.

<table>
<tr><td></td><td>Çıktı olarak AH yazmacı ile CF bayrak biti verilir.
CF bayrak biti hata varsa 1 değerine, hata yoksa 0 değerine sahip olur.
AH yazmacında durum belirtilir. (INT 13H için AH yazmacı durum anlam tablosuna ek D'den bakınız)</td></tr>
<tr><td>05</td><td>Formatla.
Pistleri formatla.

Girdi olarak AL,CH,DH,DL ve ES:BX alınır.
AL yazmacına formatlanacak sektör sayısı girilir.
CH yazmacına pist numarası girilir.
DH yazmacına başlık numarası girilir.
DL yazmacına sürücü numarası girilir.
EX:BX adres tamponu olarak kullanılır.

Adres tamponu aşağıdaki şekilde olmalıdır:

<table>
<tr><th>Göreli Konum</th><th>Anlam</th></tr>
<tr><td>00</td><td>Pist numarası</td></tr>
<tr><td>01</td><td>Başlık numarası</td></tr>
<tr><td>02</td><td>Sektör numarası</td></tr>
<tr><td>03</td><td>Sektör boyutu:

00H – 128 bayt
01H – 256 bayt
02H – 512 bayt
03H – 1024 bayt</td></tr>
</table>

Çıktı olarak AH yazmacı ve CF bayrak biti verilir.
CF bayrak biti hata varsa 1 değerine, hata yoksa 0 değerine sahip olur.
AH yazmacında durum belirtilir. (INT 13H için AH yazmacı durum anlam tablosuna ek D'den bakınız)</td></tr>
</table>

İleriki bölümlerde INT 13H kesmesinin kullanımını detaylı olarak örnek yazılımlar ile göreceğiz. Aşağıda günümüzde yaygın olarak kullanılan FAT dosyalama sistemi hakkında bilgilendirici bir yazı bulunmaktadır. Disketlerde ve sabit disklerde verilerin nasıl düzenlendiği hakkında bilgi edinmek amacıyla okunabilir.

FAT

FAT Çeşitleri

Günümüzde kullanımda olan FAT dosyalama sistemleri (**F**ile **A**llocation **T**able – Dosya Ayırma Tablosu) FAT12, FAT16 ve FAT32 dir. Bu dosyalama sistemlerinin isimlerindeki numaralar tablolarında kullandıkları bit sayılarını göstermektedir. Tablo kullanılmasının nedeni kullanımda olan ve kullanımda olmayan disket bölgelerini ayırabilmek ve bu sayede verilerin üzerine yanlışlıkla başka verilerin yazılmasını engellemek, disketi düzenlemek. Ayrıca birde VFAT (**V**irtual **FAT** – Sanal FAT) adıyla geçen bir dosyalama sitemi daha vardır. Bu dosyalama sisteminde uzun dosya adları da kullanılabilmektedir.

Şimdi FAT dosyalama sisteminin bölümlerini inceleyelim:

Küme (Cluster)

FAT biriminde bir gurup sektör bir küme oluşturur. Sadece veri bölgelerinde bulunan sektörler kümelere katılabilirler. Dosyaların ve klasörlerin verileri bu kümelerin içinde tutulur. Her kümenin boyutu (içinde bulundurduğu sektör sayısı) önyükleme kaydında tutulur. Bu değer 1 ile 128 sektör

arasında olabilir.

Önyükleme Kaydı (Boot Record)

FAT12 ve FAT16 dosyalama sistemlerinde önyükleme kaydı için 1 sektör ayrılırken FAT32'de 32 sektör ayrılır. Önyükleme kaydının göreli konumlarının anlamları aşağıdaki tabloda verilmiştir.

Göreli Konum	Uzunluk (Bayt)	Anlam
03	8	8 baytlık ASCII karakter formatında dosya sistemini oluşturan sistemin adı.
0B	2	Sektör başına düşen bayt sayısı bu alana girilmelidir. Bu değer genellikle 512'ye (0200H) eşittir.
0D	1	Küme başına düşüne sektör sayısı bu alana girilmelidir. Bu değerin 0'dan farklı olması gerekmektedir.
0E	2	Ayrılmış sektör sayısı. Yukarıda belirttiğimiz gibi FAT12 ve FAT16 için buraya 01H değeri, FAT32 için 20H değeri girilmelidir.
10	1	Diskette bulunan FAT sayısıdır. Genellikle 2 tan tablo bulunduğundan 2 değeri verilir.
11	2	Kök dizinde kaç giriş olduğunu belirtir. FAT32 de bu değer değişken olabileceğinden 0 girilmesi gereklidir. FAT12 ve FAT16 da her dizin 32 baytlık yer tutuğundan buraya girilen değerin 32 ile çarpımının sektör başına düşen bayt sayısına tam bölünebilir olması gerekir aksi bazı alanlar boşa gitmiş olur.
13	2	Disketteki toplam sektör sayısı. Toplam sektör sayısı 65535 den fazla ise buraya 0 yazılıp 20H göreli adresine toplam sektör sayısı yazılır.
15	1	Ortam tanımlayıcısı değeri. 1.44MB disket sürücüleri için bu değer F0'dır.
16	2	FAT başına düşen sektör sayısı yazılır. FAT32 için bu değer 0 a eşit olmalıdır. FAT12 ve FAT16 da ise bu değer 0 dan farklı olmalıdır.
18	2	Başlık başına düşen sektör sayısı.
1A	2	Başlık başına düşen silindir sayısı.
1C	4	Önyükleme kaydı ile ilk bölüm arasındaki sektör sayısı.
20	4	Büyük toplam sektör sayısı.
24	4	Büyük FAT başına düşen toplam sektör sayısı.
28	2	Genişletilmiş bayraklar. (FAT32)
2A	2	Dosya sisteminin sürüm bilgileri. (FAT32)
2C	4	Kök dizininin küme numarası girilmelidir. Genellikle değeri 2 dir. (FAT32)
30	2	Dosya sistemi bilgi sektörünün numarası burada tutulur. **Göreli Konum (sektör içi) / Boyut (Bayt) / Anlam** 00 / 4 / 41615252H değerine sahip olmalıdır. 1E4 / 4 / 61417272H değerine sahip olmalıdır. 1E8 / 4 / Sürücü üstündeki boş küme sayısı. Değeri bilinmiyorsa -1 değeri girilir. 1EC / 4 / En son ayrılan kümenin numarası. 1F0 / 12D / Ayrılmış. 1FE / 2 / AA55H değerine sahip olmalıdır.
32	2	Önyükleme kaydının yedeğinin tutulduğu sektörün numarası.
34	12D	Ayrılmış.

Ayrıca aşağıdaki tabloda FAT32 ve FAT12/FAT16 da farklı göreli konumlarda bulunan alanlar verilmiştir:

FAT12/FAT16	FAT32	Uzunluk (Bayt)	Anlam

24	40	1	BIOS sürücü numarası
25	41	1	Ayrılmış
26	42	1	29H
27	43	4	Seri numarası
2B	47	11	Birim etiketi
36	52	8	Dosya sistemi etiketi (FAT12,FAT16 veya FAT32)

Önyükleme kaydının boyutu 1 sektördür ve 0000H sektöründe bulunur.

Dosya Ayırma Tablosu (File Allocation Table)

FAT12 için 12 bit, FAT16 için 16 bit ve FAT32 için 32 bit uzunluğunda veri içeren bir tablodur. Dosya sisteminde hangi bölümlerin hangi dosyalara ait olduğunu belirtir. Genellikle bir dosya ayırma tablosunun kopyası bazı hatalı durumlarda kurtarma yapabilmek için disket içinde tutulur.

Dosya ayırma tablosunun ilki 0001H adresinden başlar ve boyutu 9 sektördür. Dosya ayırma tablosunun ikincisi yani birincinin kopyası 000AH adresinden başlar ve boyutu 9 sektördür.

Kök Dizini

Kök dizininin diğer dizinlerden farkı "." Ve ".." girişlerini bulundurmamasıdır (yan kökte olmasıdır). FAT12 ve FAT16'da kök dizini dosya ayırma tablosunun hemen ardındadır.

Kök dizini 0013H adresinden başlar ve boyutu 14 sektördür.

Veri Bölgesi

Dosyaların ve klasörlerin verileri veri bölgesinde tutulur. Veri bölgesi yukarıda da belirtildiği gibi kümelere ayrılmıştır ve her kümenin boyu birbirine eşit olmalıdır.

Veri bölgesi 0033H adresinden başlar ve boyutu 2847 sektördür.

Klasörler ve Dosyalar

Kök klasör dışındaki tüm klasörler FAT12 dosyalama sisteminde dosya gibi tutulurlar. Her dosyanın boyutu klasörde tutulurken klasörlerin boyutları herhangi bir yerde tutulmaz. Klasörlerin boyutları sahip olduğu küme sayısına bağlıdır, yani küme boyutlarının tam katıdır. Bir klasöre yeni bir dosya eklendiyse ve klasörün kümesi dolmuş ise klasörün boyutu 1 küme genişletilir. Kök klasörün boyutu sabit olduğundan kök klasörde böyle bir genişletme durumu yoktur.

Klasörler klasör girişlerine ayrılır. Her klasör girişinde klasörün içinde bulunan dosya veya klasörün adı, özellikleri, boyutu, tarihi, zamanı ve küme numarası bulunur ve klasör girişleri 32 bayt uzunluğundadır.

Uzunluk (bayt)	Anlam
8	Ad Dosya veya klasör adı burada tutulur. Dosya veya klasör adı yazıldıktan sonra boşluk karakteri yazılmalıdır. İlk baytın değeri bazı ek bilgiler vermektedir. - 00H: Giriş boş ve bu girişten sonraki girişler kullanımda değil. - 05H: İlk karakterin değeri E5H - 2EH: İlk karakter "." Veya ".." dır. "." Girişi klasörün kendisini işaretlemekte ekin ".." girişi üst klasörü işaretlemektedir. Kök klasör için ".." ve "." Girişleri geçersizdir. - E5H: Giriş silindi ve kullanıma müsait.
3	

	Uzantı Dosya uzantısı burada tutulur.
1	Özellik Bu baytta dosyanın veya klasörün özellikleri tutulur. 0-4 arasındaki 5 bit kullanılmaktadır. 00001: Salt okunur 00010: Sistem 00100: Gizli 01000: Birim 10000: Klasör **Not:** 0FH değeri **LFN** (**L**ong **F**ile **N**ame – Uzun Dosya Adı) sisteminin adlandırmada kullanıldığını belirtir. Biz burada sadece kısa dosya adları ile ilgileniyoruz. Uzun dosya adları hakkında detaylı bilgi için ek F'ye bakınız.
1	Ayrılmış.
1	10ms katı olarak yaratılma zamanı tutulur (VFAT'a özgü bir özelliktir)
2	Yaratılma saati tutulur. (VFAT'a özgü bir özelliktir)
2	Yaratılma tarihi tutulur. (VFAT'a özgü bir özelliktir)
2	Erişim tarihi tutulur. (VFAT'a özgü bir özelliktir)
2	Küme sayısının üst 16 biti. (FAT32'ye özgü bir özelliktir.)
2	Güncellenme saati. (VFAT'a özgü bir özelliktir)
2	Güncellenme tarihi. (VFAT'a özgü bir özelliktir)
2	16 bitlik küme sayısı.
4	Bayt olarak dosya boyutu. Klasörler için dosya boyutu 0'dır.

8. Girdi/Çıktı İşlemleri

Üçüncü bölümde mikroişlemcideki girdi/çıktı iğnelerinden bahsetmiştik. Dördüncü bölümde ise girdi/çıktı işlemlerinin hangi x86 komutlarıyla yapılabileceğini göstermiştik. Bu bölümde ise girdi/çıktı işlemlerinin nasıl yapılması gerektiği hakkında bilgi edineceğiz. Girdi/çıktı işlemlerini incelerken seri iletişim, paralel iletişim ve ekran kartına girdi/çıktı yapmayı göreceğiz.

8.1.1. Seri İletişim

Bir önceki bölümde seri iletişim için INT 13H kesmesini kullanmıştık. Şimdi ise seri iletişimi kesme kullanmadan, direk olarak girdi/çıktı komutları ile gerçekleyeceğiz. Bunun için bilgisayarda bulunan 8250 Seri İletişim Çipi'ne erişmemiz gerekmektedir.

8.1.2. 8250

8250 çipinde mikroişlemcilerde olduğu gibi bazı yazmaçlar bulunmaktadır ve bizim yapmamız gereken bu yazmaçları kontrol etmektir. 8250 üzerinde 9 tane girdi/çıktı yazmacı bulunmaktadır. COM1 seri kapısının adresi 03F8H ve COM2 seri kapısının adresi 02F8H'dır.

Aşağıdaki tabloda COM1 ve COM2 seri kapıları için 8250 yazmaç adresleri ve yazmaçların işlevleri verilmiştir.

<table>
<tr><th>Adres (COM1)</th><th>Adres (COM2)</th><th>İşlev</th></tr>
<tr><td>03F8</td><td>02F8</td><td>Veri alış-veriş yazmacı. (Baud Rate Divisor Latch yazmacı alt baytı olarak da kullanılır.)

Bu yazmaç gerçekte biri veri girişi, diğeri de veri çıkışı için kullanılan 2 tane alt yazmaçtan oluşmaktadır. Veri girişi için bu adresteki yazmacın okunması, veri çıkışı için ise bu adresteki yazmaca veri yazılması gerekmektedir.</td></tr>
<tr><td>03F9</td><td>02F9</td><td>Kesme yönetim yazmacı. (Baud Rate Divisor Latch yazmacı üst baytı olarak da kullanılır.)

Bu yazmacı kullanarak 8250 çipinin kesme özelliklerini değiştirebilir veya özelliklerini okuyabilirsiniz. 8250 çipi 4 tane kesme oluşturabilir. Bunlar:

1- Karakter alındı kesmesi (0. bit ile kontrol edilir.)
2- Karakter verildi kesmesi (1. bit ile kontrol edilir.)
3- İletişim hatası kesmesi (2. bit ile kontrol edilir.)
4- Durum değişikliği kesmesi (3. bit ile kontrol edilir.)

Bu kesmeleri açmak için bu adresteki baytın bitlerinden uygun olana 1 değerini, kapatmak için 0 değerini yazmak yeterlidir. 4-7 arasındaki bitler kullanımda değildir.

Eğer ayarlamalara göre Baud Rate Divisor Latch yazmacı seçildiyse aşağıdaki tabloya göre veri iletişiminin bps hız değeri girilmelidir. Burada önemli olan her iki tarafın da aynı hız değerinde çalışmasıdır.
<table>
<tr><th>Bps Değeri</th><th>3F9 Değeri</th><th>2F9 Değeri</th><th>3F8 Değeri</th><th>2F8 Değeri</th></tr>
<tr><td>110</td><td>04</td><td>04</td><td>17</td><td>17</td></tr>
<tr><td>300</td><td>01</td><td>01</td><td>80</td><td>80</td></tr>
<tr><td>600</td><td>00</td><td>00</td><td>C0</td><td>C0</td></tr>
<tr><td>1200</td><td>00</td><td>00</td><td>60</td><td>60</td></tr>
<tr><td>1800</td><td>00</td><td>00</td><td>40</td><td>40</td></tr>
</table>
</td></tr>
</table>

		2400 00 00 30 30 3600 00 00 20 20 4800 00 00 18 18 9600 00 00 0C 0C 19200 00 00 06 06 38400 00 00 03 03 56000 00 00 01 01
03FA	02FA	Kesme tanımlama yazmacı. Salt okunur bir yazmaç olup oluşan kesme hakkında bilgi almak için kullanılır. 0. bit 0 değerine sahip ise bir kesme oluştuğu, 1 değerine sahip ise herhangi bir kesmenin oluşmadığı anlaşılır. 1. ve 2. bitler ne tip kesme oluştuğu hakkında bilgi verir. **Bit Değeri (21)** / **Anlamı** 00 – Durum değişikliği kesmesi 01 – Veri gönderildi kesmesi 10 – Veri alındı kesmesi 11 – İletişim hatası kesmesi 3-7 arasındaki bitler kullanımda değildir ve 0 değerinde tutulmaları gerekmektedir. İletişim hatası kesmesi veri taşması, eşlik, kalıp veya duraklama hatalarından biri oluştuğundan oluşur. Kesme tanımlama yazmacından bu kesme silinip bir sonraki kesme (var ise) alınmak istenirse hat durum yazmacının okunması gerekir. Veri alındı kesmesi veri alış yazmacına veri geldiği zaman oluşur. Bu kesmeyi yazmaçtan silmek için veri alış yazmacından verinin okunması gerekir. Veri gönderildi kesmesi veri veriş yazmacı boşaltıldığında oluşur ve bu kesmeyi yazmaçtan silmek için veri veriş yazmacına yeni bir veri yazılır veya kesme tanımlama yazmacı bu kesme için okunur. Durum değişikliği kesmesi CTS,DST,RI,RLS durumları değiştiğinde oluşur ve bu kesmeyi yazmaçtan silmek için durum değişikliği yazmacı okunmalıdır. Aynı anda birden fazla kesme oluşabileceğinden 8250 çipi oluşan kesmeleri seviyelendirmiştir. En az öneme durum değişikliği kesmesi sahipken, en önemli kesme tahmin edilebileceği gibi iletişim hatası kesmesidir.
03FB	02FB	Hat kontrol yazmacı. Bu yazmaç sayesinde yapılacak olan seri iletişimin özellikleri belirlenebilir. **Bit** / **İşlev** 7 – Baud Rate Divisor Latch. Bu bitin değeri 1 ise 03F8h/2F8h and 3F9h/2F9h ve Baud Rate Divisor Latch olarak kullanılır. Bu bitin değerinin 8250 çipinin veri iletim hızını ayarlarken 1, diğer durumlarda 0 değerine sahip olması gerekir.

<table>
<tr><td></td><td></td><td>
<table>
<tr><td></td><td></td></tr>
<tr><td>6</td><td>İletimi bitin değeri 1 ise duraklat.</td></tr>
<tr><td>5
4</td><td>Eşlik kontrol.
00: Tek
01: Çift
10: Eşlik her zaman 1 değerinde
11: Eşlik her zaman 0 değerinde</td></tr>
<tr><td>3</td><td>Eşlik açık (1) / kapalı (0).</td></tr>
<tr><td>2</td><td>Durdurma bitleri.
0 ise 1, 1 ise 2 bit durdurma biti olarak kullanılır.</td></tr>
<tr><td>1
0</td><td>Veri uzunluğu.
00: 5 bit
01: 6 bit
10: 7 bit
11: 8 bit</td></tr>
</table>
</td></tr>
<tr><td>03FC</td><td>02FC</td><td>Durum kontrol yazmacı.
<table>
<tr><th>Bit</th><th>İşlev</th></tr>
<tr><td>7
6
5</td><td>0</td></tr>
<tr><td>4</td><td>Loopback modu.
1 olduğu zaman loopback modu açıktır.
0 olduğu zaman loopback modu kapalıdır.

Bu mod açık olduğu zaman veri alış yazmacı ile veri veriş yazmacı birbiri ile iletişir. Yazılımlar test edilirken kullanılabilir. Bu mod açıkken herhangi bir kesme oluşmaz.</td></tr>
<tr><td>3</td><td>Kesme aktifleştirme biti.
1 olduğu zaman kesmeler aktiftir.
0 olduğu zaman kesmeler kapalıdır.</td></tr>
<tr><td>2</td><td>Kullanımda değil.</td></tr>
<tr><td>1</td><td>RTS
1 olduğu zaman iletişim yapılabilir.
0 olduğu zaman iletişim yapılmamalıdır.
CTS ile beraber kullanılmalıdır.</td></tr>
<tr><td>0</td><td>DTR
1 olduğu zaman iletişim yapılabilir.
0 olduğu zaman iletişim yapılmamalıdır.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><td></td><td>DSR ile beraber kullanılmalıdır.</td></tr></table></td></tr>
<tr><td>03FD</td><td>02FD</td><td>Hat durum yazmacı.

Bu yazmaç iletim durumu hakkında bilgi verir ve salt okunur bir yazmaçtır.

<table><tr><th>Bit</th><th>İşlev</th></tr><tr><td>7</td><td>Kullanımda değil</td></tr><tr><td>6</td><td>Verici öteleme yazmacı boş.
Bu bitin değeri 1 ise öteleme yazmacı boştur.</td></tr><tr><td>5</td><td>Verici tutucu yazmacı boş.
Verici tutucu ve öteleme yazmaçları boş ise 1 değerini alır. Aksi takdirde 0 değerini alır.</td></tr><tr><td>4</td><td>Duraklama hatası
Duraklama sinyali geldiğinde oluşur ve duraklama hatası biti 1 değerini alır. Hat durum yazmacı okunduğunda duraklama hatası biti 0 değerini alır.</td></tr><tr><td>3</td><td>Kalıp hatası
Durdurma biti olmayan bir veri geldiğinde bu hata olurşur ve kalıp hatası biti 1 değerini alır. Hat durum yazmacı okunduğunda kalıp hatası biti 0 değerini alır.</td></tr><tr><td>2</td><td>Eşlik hatası
Veri iletiminde bir eşlik hatası saptandığında bu bit 1 değerini alır. Hat durum yazmacı okunduğunda eşlik hatası biti 0 değerini alır.</td></tr><tr><td>1</td><td>Taşma hatası
Veri alış yazmacında bulunan bir veri okunmadan bir başka veri gelir ise veri taşma hatası oluşur ve veri taşma hatası biti 1 değerini alır. Hat durum yazmacı okunduğunda taşma hatası biti 0 değerini alır.</td></tr><tr><td>0</td><td>Veri hazır
Veri alış yazmacına veri geldiğinde veri hazır biti 1 değerini alır.
Veri alış yazmacından vernin okunması veri hazır bitini 0 a eşitler.</td></tr></table></td></tr>
<tr><td>03FE</td><td>02FE</td><td>Modem durum yazmacı.
Salt okunur bir yazmaçtır. Bazı modem sinyalleri ve el sıkışma işlemi hakkında bilgi verir.

<table><tr><th>Bit</th><th>İşlev</th></tr><tr><td>7</td><td>DCD biti.
Modemler tarafından kullanılır.</td></tr></table></td></tr>
</table>

		6	RI biti. Modemler tarafından kullanılır.
		5	DTS biti. Karşı donanımın DTR sine bağlıdır. Karşı donanımın DTR biti aktifleştirildiğinde DTS biti aktifleşir ve karşı donanımın meşgul olmadığı anlaşılır.
		4	CTS biti. Karşı donanımın RTS sine bağlıdır. Karşı donanımın RTS biti aktifleştirildiğinde CTS biti aktif olur ve veri gönderiminin yapılabileceği anlaşılır.
		3	DCD biti değiştiğinde bu bit 1 değerini alır. Modem durum yazmacı okunduğunda bitin değeri 0 a eşitlenir.
		2	RI biti değiştiğinde bu bit 1 değerini alır. Modem durum yazmacı okunduğunda bitin değeri 0 a eşitlenir.
		1	DSR biti değiştiğinde bu bit 1 değerini alır. Modem durum yazmacı okunduğunda bitin değeri 0 a eşitlenir.
		0	CTS biti değiştiğinde bu bit 1 değerini alır. Modem durum yazmacı okunduğunda bitin değeri 0 a eşitlenir.
03FF	02FF	Shadow Receive Register (read only, not available on original PCs).	

8.1.3. Örnek

İlk örneğimizde COM1 kapısını ve loopback kullanarak seri iletişimi gerçekleyeceğiz. Seri iletişim kesmelerini dolayısıyla kullanmayacağız. Aşağıdaki kodu DEBUG ile deneyelim.

```
0  ORG 0100H
1  PUSH CS
2  PUSH CS
3  POP DS
4  POP ES
5  CALL ILETISIM_AYARLA
6  MOV AL,'A'
7  CALL VERI_YOLLA
8  DINLE: CALL DURUM_OKU
9  CMP AL,61H
10 JE VERI_ALINABILIR
11 CMP AL,60H
12 JE VERI_ALINAMAZ
13 CMP AL,40H
14 JE VERI_ALINAMAZ
15 CMP AL,60H
16 JE VERI_ALINAMAZ
17 SON: INT 03H
18
19 VERI_ALINABILIR: CALL VERI_AL
```

```
20  JMP SON
21
22  VERI_ALINAMAZ: JMP DINLE
23
24  ;AL yazmacındaki veriyi yollar
25  VERI_YOLLA: MOV DX,03F8H
26  OUT DX,AL
27  RET
28
29  ;Veriyi alır ve AL yazmacına kaydeder
30  VERI_AL: MOV DX,03F8H
31  IN AX,DX
32  RET
33
34  ;Hat durumunu okur ve AX yazmacına kaydeder
35  DURUM_OKU: MOV DX,03FDH
36  IN AX,DX
37  RET
38
39  ;Seri iletişimi ayarlar
40  ILETISIM_AYARLA: MOV DX,03FBH
41  MOV AX,10000010B
42  OUT DX,AX
43  MOV DX,03F9H
44  MOV AX,04H
45  OUT DX,AX
46  DEC DX
47  MOV AX,17H
48  OUT DX,AX
49  MOV DX,03FBH
50  MOV AX,00000010B
51  OUT DX,AX
52  MOV DX,03FCH
53  MOV AX,00010011B
54  OUT DX,AX
55  RET
```

1., 2., 3. ve 4. satırdaki PUSH-POP komurları ile ES ve DS yazmaçlarının değerlerini CS yazmacının değerine eşitledik. Bu işlem MOV komutu kullanılarak da yapılabilirdi ama aynı işlemin PUSH-POP ile yapılması MOV komutu ile yapılmasına göre çok daha hızlıdır. Bu yüzden bu yöntemi tercih ettik.

5. satırda *ILETISIM_AYARLA* yordamımızı çağırdık. 44. satırdan başlayan *ILETISIM_AYARLA* yordamımız bir seri bağlantının başlatılabilmesi için gerekli tüm işlemleri yapmaktadır. Öncelikle 44. satırda DX yazmacına 03FBH değeri kaydedilmiş ve ardından 45. satırda AX yazmacına 10000010B değeri yazıldıktan sonra OUT DX,AX komutu ile 03FBH adresine AX verisi gönderilmiştir. 03FBH adresinde hat kontrol yazmacının bulunduğunu biliyoruz. AX yazmacı ile bu yazmaca yollanan 10000010B verisinin anlamı:

7. bit: Veri iletişim hızını ayarlamak istiyorum. Yani 03F9H ve 03F8H adreslerini veri hızı parametrelerini girmek için kullanacağım.

6. bit: Veri iletişimini duraklatmak istemiyorum.

5. ve 4. bitler: Veri iletişiminde eşlik kullanmak istersem tekli eşlik sistemini kullanmak isterim.

3. bit: Veri iletişiminde eşlik biti kullanmak istemiyorum.

2. bit: Veri iletişiminde sadece 1 tane durdurma biti kullanmak istiyorum.

1. ve 0. bit: Veri iletişiminde 7 bitlik veriler kullanacağım.

Veri iletişiminde 7 bitlik veri kullanmak istememizin nedeni örneğimizde ASCII karakterler göndermemiz ve ASCII karakterlerin 7 bit olmasıdır. Bu şekilde belirli bir oranda performans kazancı sağlayacağız. 47. satırda DX yazmacına 03F9H değerini ve, 48. satırda AX yazmacına 04H değerini giriyoruz ve 49. satırda AX yazmacındaki veriyi 03F9H adresine aktarıyoruz. 50., 51., 52. ve 53. satırlardaki işlemler ile

de 03F8H adresine 17H verisini aktarıyoruz. Veri hızı tablosuna bakacak olursak 03F9H adresine 04H ve 03F8H adresine 17H değerlerinin girilmesi 110 bps hızında iletişim yapacağımızı göstermektedir. 53., 54. ve 55. satırdaki işlemler ile 03FBH adresine 00000010B verisini yolluyoruz yani sadece 7. biti değiştiriyoruz. Bunu yapmamızın nedeni veri iletişim hızını ayarlamış olmamız ve artık 03F8H ve 03F9H adresindeki yazmaçları veri iletişiminde kullanmak istememizdir. 56., 57. ve 58. satırlardaki işlemler ile 03FCH adresine 00010011B verisini aktarıyoruz. Bu adreste durum kontrol yazmacı bulunmaktadır ve 00010011B verisinin durum kontrol yazmacına aktarılmasının anlamı:

7., 6. ve 5. bitler: Her zaman 0 değerini alması gerektiği için 0 yapıldı.

4. bit: Loopback modunu kullanmak isityorum. Yani seri kapı kendi içinde iletişime geçecek gönderilen veri geri alınacaktır.

3. bit: Kesmeleri kullanmak istemiyorum.

2. bit: Kullanımda olmayan bir bit olduğundan 0 değeri verildi.

0. ve 1. bit: RTS ve DTR bitlerini 1 yaparak iletişim yapılabileceği, meşgul olunmadığı belirtildi.

59. satırdaki RET komuru ile *ILETISIM_AYARLA* yordamımız sonlandırılıyor ve program akışı 6. satırdan devam ediyor.

6. satırda AL yazmacına ASCII 'A' karakterini yazdık ve 7. satırdaki CALL komutu ile *VERI_YOLLA* yordamını çağırdık. 29. ve 31. satırlar arasında bulunan *VERI_YOLLA* yordamı 03F9H adresine AL yazmacındaki veriyi aktardı yani veri veriş yazmacına AL yazmacındaki veri yazıldı. 42. satırdaki RET komutu ile de program bu yordam çağırılıldığında kaldığı yerden devam ediyor.

8. satırdaki CALL komutu ile *DURUM_OKU* yordamına atlanıldı. 39. ve 41. satırlar arasında bulunan *DURUM_OKU* yordamı ise 03FDH yazmacındaki veriyi alıp AX yazmacına kaydetti (AX yazmacının alt baytı yani AL yazmacına kaydetti). Yani AL yazmacı bit değerlerine göre aşağıdaki anlamları alır:

7. bit: Kullanımda olmadığından her zaman 0'dır.

6. bit: Bitin değeri 1 ise verici öteleme yazmacı boştu. Aksi takdirde iletimde ötelemeye devam edildiği anlaşılır.

5. bit: Bitin değeri 1 ise verici tutucu yazmacı ve öteleme yazmacı boştur. Yani iletim tamamlanmıştır. Bitin değeri 0 ise iletimin devam ettiği anlaşılır.

4. bit: Bitin değeri 1 ise duraklama hatası oluşmuştur.

3. bit: Bitin değeri 1 ise eşlik hatası oluşmuştur.

2. bit: Bitin değeri 1 ise kalıp hatası oluşmuştur.

1. bit: Bitin değeri 1 ise taşma hatası oluşmuştur.

0. bit: Bitin değeri 1 ise verinin hazır olduğu anlaşılır.

9. ve 15. satırlar arasındaki işlemler ile de *DURUM_OKU* yordamının çıktısı olan AL yazmacının değeri karşılaştırılıyor ve uygun bir şekilde gerekli atlamalar yapılıyor. AL yazmacında 61H değerinin olması veri veriş yazmaçlarının boş olduğu ve verinin gelip alınmaya hazır olduğu anlamına gelir. AL yazmacının 60H değerine sahip olması ise veri veriş yazmaçlarının boş olduğunu fakat verinin henüz gelmediğini belirtir. AL yazmacının 00H değerine sahip olması ise hem verici tutucu hemde verici öteleme yazmaçlarının dolu olduğunu belirtir. AL yazmacının değerinin 40H olması ise sadece veri öteleme yazmacının dolu olduğunu göstermektedir.

25. satırdaki *VERI_ALINAMAZ* etiketine atlanıldığında *DINLE* etiketine geri atlanır. Bu sayede *DURUM_OKU* yordamı yeniden çağırılarak durum verisi yeniden okunur.

21. satırdaki *VERI_ALINABILIR* etiketine atlanıldığında ise 34. ve 36. satırlar arasında bulunan *VERI_AL* yordamı çağırılır. Bu yordam AX yazmacına 03F8H adresindeki veriyi aktarır. Bu sayede programı DEBUG ile denediğinizde program sonlandırıldığında AX yazmacında 'A' ASCII karakterinin değerini göreceksiniz.

Dokuzuncu bölümde seri iletişim kullanarak bir sohbet yazılımı yazacağız.

8.1.4. Paralel İletişim

Paralel iletişim tek yönlü veya çift yönlü olabilir. Günümüzde yazıcıların çoğunda USB kullanılmasına rağmen tek yönlü iletişime en güzel örnek olarak bilgisayar ve paralel kapılı yazıcı arasındaki bağlantıyı verebiliriz. Çift yönlü iletişime örnek olarak 2 bilgisayar arasında veri alış-verişi için yapılan bağlantı verilebilir veya paralel kapıyı kullanan bir EEPROM programlayıcı buna örnek olabilir.

Paralel kapı iğnelerini inceleyelim:

İğne	İşlev
1	Veri bulunup bulunmadığını belirtir. Veri bulunduğunda 0, bulunmadığında 1 değerini alır. Çıkış olarak kullanılır.
2 3 4 5 6 7 8 9	8 bitlik veri yolu. 2. pin 0. veriyolu, 9. pin 7. veriyoludur. Çıkış olarak kullanılır.
10	Yollanan veri yerine ulaştığı zaman veriyi alan donanım tarafından gönderilen kabul sinyalini bulundurur. Kabul edildiği zaman 0, edilmediği zaman 1 değerindedir. Giriş olarak kullanılır.
11	Veri yollanacak olan donanım meşgul ise bu donanım tarafından 0 değeri verilir. Meşgul değilse 1 değerini alır. Giriş olarak kullanılır.
12	Yazıcı tarafından kağıt bittiğinde 1 değeri verilir. Aksi halde 0 değerindedir. Giriş olarak kullanılır.
13	1 değerine sahip iken yazıcı seçilidir, aksi halde seçili değildir. Giriş olarak kullanılır.
14	0 olduğunda yazıcı otomati olarak her CR'den sonra yeni satıra geçer. Çıkış olarak kullanılır.
15	0 olması yazıcıda bir hata oluştuğunu gösterir. Giriş olarak kullanılır.
16	50ms'den daha uzun bir süre 0 olduğunda yazıcı kendini hazırlar. Çıkış olarak kullanılır.
17	1 olduğunda yazıcı kendini kapar. Çıkış olarak kullanılır.
18 19 20 21 22 23 24 25	Toprak.

Paralel kapıyı kullanmak istediğimizde paralel kapı donanımı tarafından bize ayrılan 3 adet yazmaç yani 3 tane giriş/çıkış adresi vardır. Bunlar veri yazmacı (LPT1: 03BCH, LPT2: 0378H), durum yazmacı (LPT1: 03BDH, LPT2: 0379H) ve kontrol yazmacıdır (LPT1: 03BEH, LPT2: 037AH).

<table>
<tr><th>Adres (LPT1)</th><th>İşlev</th></tr>
<tr><td>03BC</td><td>Veri bulunur. Okunabilir veya yazılabilir.</td></tr>
<tr><td>03BD</td><td>
<table>
<tr><th>Bit</th><th>İşlev</th></tr>
<tr><td>7</td><td>0 ise çevre birimi meşgul demektir.</td></tr>
<tr><td>6</td><td>0 ise çevre biriminden tanımlama cevabı gelmiş demektir.</td></tr>
<tr><td>5</td><td>1 ise çevre biriminin (yazıcı) kağıdı bitmiş demektir.</td></tr>
<tr><td>4</td><td>1 ise çevre birimi seçilmiş demektir.</td></tr>
<tr><td>3</td><td>0 ise çevre birimi hatası oluşmuş demektir.</td></tr>
<tr><td>2</td><td>0 ise çevre birimi açık demektir.</td></tr>
<tr><td>1
0</td><td>Kullanımda değil.</td></tr>
</table>
</td></tr>
</table>

03BE	**Bit**	**İşlev**
	7 6	Kullanımda değil.
	5	0 ise veri çıkışı, 1 ise veri girişi yapılır. (PS/2 sistemlerinde)
	4	1 ise paralel kapı açıktır.
	3	1 ise yazıcı açılır.
	2	0 ise yazıcı kendini başlatır.
	1	1 ise her CR den sonra yazıcı yeni satıra geçer.
	0	1 ise hatta veri var demektir.

8.1.5. Tek Yönlü İletişim

Bir önceki bölümde kesmeleri incelerken INT 17H kesmesini görmüştük. Bu kesme BIOS'un yazıcı hizmetlerini veren kesmedir ve tek yönlü olarak yazıcı ile iletişimde kullanılır. BIOS'un bize sağladığı bu kesmeyi kullanmak istemezsek kendi yazıcı yordamlarımızı girdi/çıktı komutlarını kullanarak kendimiz yazabiliriz.

Tek yönlü iletişimde yazıcı ile sağlıklı bir bağlantı kurabilmek için bazı karşılıklı elsıkışmaların yapılması gerekmektedir. Aşağıda alıcı ve vericinin yapması gerekenler listelenmiştir:

Alıcı
1) Alıcı taraf meşgul hattını kapatmalıdır. 2) Alıcı taraf tetikleme hattı açılana kadar beklemelidir. 3) Alıcı taraf veriyi veri hattından okumalıdır. 4) Alıcı taraf kabul hattını açmalıdır. 5) Alıcı taraf tetikleme hattı kapanana kadar beklemelidir. 6) Alıcı taraf kabul hattını kapatmalıdır. 7) Alıcı taraf yukarıdaki adımları her bayt için tekrarlamalıdır.
Verici
1) Verici taraf meşgul hattını kontrol etmelidir. Eğer meşgul hattı açıksa verici taraf meşgul hattı kapanana kadar beklemelidir. 2) Verici taraf veri hattından veriyi yollamalıdır. 3) Verici taraf tetikleme hattını açmalıdır. 4) Verici taraf kabul hattı açılana kadar beklemelidir. 5) Verici taraf tetikleme hattını kapatmalıdır.. 6) Verici taraf kabul hattı kapanana kadar beklemelidir. 7) Verici taraf yukarıdaki adımları her bayt için tekrarlamalıdır.

8.1.6. Vidyo Belleği Girdi/Çıktı İşlemleri

Ekranda gördüğümüz her piksel (ekranda oluşturulabilecek en küçük nokta – ekrana biraz yaklaşıp bakılacak olursa pikseller ayırt edilebilir-) vidyo belleğinden alınıp ekrana aktarılmaktadır. Bu vidyo belleğinde yapılacak olan işlemler direk olarak ekranda da değişikliklere neden olacaktır. Bu durum günümüzdeki 3 boyutlu oyunlardan basit bir Linux konsol ekranına kadar her durum için geçerlidir. Ekranın yüksek çözünürlükte olması vidyo belleğinin bu yüksek çözünürlüğü destekleyecek boyutta olmasını gerektirir çünkü her piksel bu vidyo belleğinden alınıp ekrana çizilmektedir.

20x5 piksel çözünürlük sağlamak için kullanabileceğimiz bir vidyo belleğini inceleyelim:

Y\X	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
2	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79

4	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99

20 x 5 = 100 tane pikselimiz var. Başka bir değişle 20 sütun ve 5 satır dan oluşan 100 hücreli bir tablomuz var. Her hücre bir bayt olduğundan 20x5 piksel çözünürlüğündeki bir vidyo belleği tam olarak 100 baytlık alan kaplamaktadır.

Vidyo belleğindeki 49. hücreyi değiştirmemiz ekrandaki 3. satır ile 10. sütunun kesiştiği yerdeki pikseli değiştirecektir.

Aynen diğer girdi/çıktı işlemlerinde olduğu gibi vidyo belleğinin de bir adresi vardır ve biz bu adresi kullanarak belli bir pikseli değiştirebiliriz. 000A0000 adresinden 000BFFFF adresine kadar (dikkat edin bunlar fiziki adres değerleri, lojik adres olarak A000:0000 dan B000:FFFF e kadar olan adresler kullanılabilir) olan adresler piksellere işaret etmektedir ve piksel değerlerini değiştirmek için kullanılabilir.

Peki pikselleri direk olarak değiştirmek ne işimize yarayacak? 2 durum düşünelim:

1- Bir program yazdınız ve bir konsol ortamında veya tamamen kendi işletim sisteminizin ortamında ekrana yazı yazdırmak istiyorsunuz...

Bu işlemi BIOS'un size sunduğu INT 10H kesmesinin alt yordamlarını kullanarak yapabilirsiniz. Ama unutmayınki BIOS kesmeleri yavaştır. Yani bir INT 10H kesmesini kullanarak ekrana bir ASCII 'A' karakterini yazdırmak aynı işlemi vidyo belleğine direk erişip yapmaktan daha yavaştır. Ayrıca işletim sistemi yazarken BIOS kesmesi kullanamayacağınız durumlarda oluşabilmektedir.

Başka bir olasılık ise BIOS'u olmayan, dolayısıyla BIOS'un sunduğu kullanımı kolay kesmelerin olmadığı bir ortamda çalışan bir makinada bu yazı yazdırma işlemini yapmanız gerekebilir. O zaman o makinanın vidyo belleğine erişilip, bellekte direk değişiklikler yapılarak ekrana yazı yazdırılabilir.

2- Programlarınızda sadece yazı kullanmak canınızı sıkıyor ve artık grafiksel bazı özellikler eklemek istiyorsunuz... Oyun yapmak istiyorsunuz...

Tabi günümüzdeki sistemlerde bu işler genellikle belirli bir işletim sistemi ortamında belirli yazılım kütüphaneleri kullanılarak yapılmaktadır. 3 boyutlu grafikler çizmek için tek tek geometrik hesapları yapmak yerine OpenGL gibi kütüphaneleri kullanarak kolayca ekrana odanızın 3 boyutlu modelini çizebilirsiniz veya bir **GUI** (**G**raphical **U**ser **I**nterface – Grafik Kullanıcı Arayüzü) ye sahip bir program yazmak için direk olarak GUI kütüphanelerini kullanabilirsiniz. Bütun bunları yapmak istiyorsanız zaten assembly dili üzerinde çalışmak yerine C,C++,C#,Java,D,... gibi dilleri üzerinde çalışmanız daha mantıklı gelebilir. Daha derinden baktığınızda ise aslında bunun duruma göre değiştiğini göreceksiniz. Mesela kendi işletim sisteminizi yazıyorsanız bu tür işlemleri işletim sisteminizi belirli bir seviyeye getirene kadar kendiniz yapmalısınız ve belirli bir noktadan sonra piyasada bulunan üst seviyeli grafik kütüphanelerini ve dilleri kendi sisteminize entegre edebilirsiniz.

Yukarıda bahsettiğim konular daha çok kişisel bilgisayarlar için geçerli. Öyle ortamlarda çalışmanız gerekebilirki sizden önce kimse o ortam için herhangi bir kütüphane oluşturmamış olabilir. Bu yüzden de her şeyi sıfırdan sizin yazmanız gerekebilir. Bütünleşik sistemler bu tür durumlara örnek olarak verilebilir.

Kendi grafik kütüphanenizi yazmak isteyebilirsiniz. Bunun için performans başta olmak üzere birçok nedeniniz olabilir.

Günümüzdeki sistemlerin işleyişi hakkında derin bilgiye sahip olarak o sistemlerin geliştirilmesinde çalışmak isteyebilirsiniz. Performansı artırma, açıkları kapatma gibi konular üzerinde çalışmak isteyebilirsiniz.

Tüm bunları söyledikten sonra asıl konumuza girelim.

8.1.7. BIOS Vidyo Kipleri

Yazı ve grafik olmak üzere iki tane ana vidyo kipi vardır. Yazı kipinde ekrana piksel değil ASCII karakterler yazıdırılır. Grafik kipte ise tek tek pikseller ile uğraşılır.

BIOS'da bulunan INT 10H kesmesinin 00H alt yordamı vidyo kipini ayarlamakta kullanılır. Aşağıda bu alt yordamı çağırırken AL yazmacının aldığı değerlere göre seçilen vidyo kiplerinden bazılarını listelenmiştir:

AL Değeri	Vidyo Kipi	Çözünürlük	Renk Sayısı
01H	Yazı	40 x 25	16
03H	Yazı	80 x 25	16
04H	CGA	320 x 200	4

06H	CGA	640 x 200	2
0DH	EGA	320 x 200	16
0EH	EGA	640 x 200	16
10H	EGA	640 x 350	16
11H	VGA	640 x 480	16
12H	VGA	640 x 480	16
13H	VGA	320 x 200	256

VGA = **V**ideo **G**raphics **A**rray - Vidyo Grafik Dizisi
CGA = **C**olor **G**raphics **A**dapter – Renkli Grafik Bağdaştırıcısı
EGA = **E**nhanced **G**raphics **A**dapter – Geliştirilmiş Grafik Bağdaştırıcısı

İlk örneğimizde 40 x 25 çözünürlüğündeki 16 renkli yazı vidyo kipini kullanacağız. Ardından da 320 x 200 çözünürlüğündeki 256 renkli grafiksel VGA vidyo kipini kullanacağız. Örneklerimize geçmeden önce birkaç konuyu açıklığa kavuşturalım.

16 renkli yazı vidyo kipinde her bayt bir piksele karşılık gelmektedir. Bu baytın bitlerinin işlevlerini aşağıdaki tabloda bulabilirsiniz:

<table>
<tr><th>Bit</th><th>Anlam</th></tr>
<tr><td>7</td><td>Değeri 1 ise yazı yanıp söner. 0 ise yazı sabittir.</td></tr>
<tr><td>6
5
4</td><td>Arkaplan rengidir.
<table><tr><th>İkili</th><th>Renk</th></tr><tr><td>000</td><td>Siyah</td></tr><tr><td>001</td><td>Mavi</td></tr><tr><td>010</td><td>Yeşil</td></tr><tr><td>011</td><td>Camgöbeği</td></tr><tr><td>100</td><td>Kırmızı</td></tr><tr><td>101</td><td>Eflatun</td></tr><tr><td>110</td><td>Kahverengi</td></tr><tr><td>111</td><td>Açık gri</td></tr></table></td></tr>
<tr><td>3
2
1
0</td><td>Yazı rengidir.
<table><tr><th>İkili</th><th>Renk</th></tr><tr><td>0000</td><td>Siyah</td></tr><tr><td>0001</td><td>Mavi</td></tr><tr><td>0010</td><td>Yeşil</td></tr><tr><td>0011</td><td>Camgöbeği</td></tr><tr><td>0100</td><td>Kırmızı</td></tr><tr><td>0101</td><td>Eflatun</td></tr><tr><td>0110</td><td>Kahverengi</td></tr><tr><td>0111</td><td>Açık gri</td></tr><tr><td>1000</td><td>Koyu gri</td></tr><tr><td>1001</td><td>Açık mavi</td></tr><tr><td>1010</td><td>Açık yeşil</td></tr><tr><td>1011</td><td>Açık camgöbeği</td></tr><tr><td>1100</td><td>Açık kırmızı</td></tr><tr><td>1101</td><td>Açık eflatun</td></tr><tr><td>1110</td><td>Sarı</td></tr><tr><td>1111</td><td>Beyaz</td></tr></table></td></tr>
</table>

Ayrıca yazı modunda dikkat edilmesi gereken başka bir husus ise karakterlerin B8000H ve BFFFFH fiziki bellek bölgelerine yazılmalarıdır. Bellekte karakterin ASCII kodu ve ardından da renk özelliği tutulmaktadır. Aşağıdaki kodu ele alalım:

```
1  MOV AX,0B800H
2  MOV ES,AX
3  MOV AH,0FH
4  MOV AL,61H
5  MOV SI,0000H
6  ES MOV [SI],AX
```

Yukarıdaki kod parçacığını inceleyelim. İlk iki satırdaki işlemlerle ES kesim yazmacına B800H değerini yazdık. Ardından 3. ve 4. satırdaki işlemlerle AX yazmacına 0F61H değerini yazdık. 5. satırda SI yazmacına 0000H değerini girdik ve 6. satırdaki işlem ile B800:0000 adresine AX yazmacını kaydettik.

Şimdi aklınıza şöyle bir soru gelebilir: Yukarıda önce ASCII karakter kodunun sonra renk kodunun kaydedilmesi gerektiği söylendi ama burada neden üst bayt olan AH yazmacına renk kodu ve alt bayt olan AL yazmacına ASCII karakter kodu yazıldı?

İlk bölümlerde bilgisayar sistemlerinin bu özelliğinden bahsetmiştik. Sistemimiz Little Endian Kuralı'na göre çalıştığından belleğe AX yazmacı yazılırken tersten yazılır. Yani önce AL yazmacı, AL yazmacından sonra AH yazmacı yazılır. Yani tersten yazılıyormuş gibi görünsede aslında iki ters bir düze neden olmaktadır.

Yukarıdaki tablodan da anlaşılabileceği gibi CGA sisteminden sonra EGA sistemi ardından da VGA sistemi ortaya çıkmıştır. 320 x 200 çözünürlüğündeki 256 renk VGA sisteminde piksellerin renk değerleri 8 bitliktir (8 bit ile 256 tane farklı durum oluşturulabilir) yani 1 bayttır. Bu yüzden 320 x 200 çözünürlükteki bir VGA vidyo kipi yaklaşık 62.5KB alan kaplamaktadır.

Peki günümüzdeki 1024 x 768 çözünürlüklü 32 bitlik vidyo kipleri nerede? Bu 32 bit tam olarak ne?

32 bitin anlamı burada 32 bitlik vidyo kipinin kullanıldığıdır. Bu 32 bitin tamamı renkler için kullanılmamaktadır. Renkler için genellikle 24 bit kullanılmaktadır. Yani her piksel için 3 baytlık bir değer kullanılmaktadır ve bu baytlar **RGB** (**R**ed **G**reen **B**lue – Kırmızı Yeşil Mavi) verilerini içermektedirler. Bu sayede kırmızı renk derinliği 256, yeşil renk derinliği 256 ve mavi renk derinliği 256 dır. Bu renk derinliklerinin kombinasyonları ile çokça duyduğumuz 16.7 milyon renk sayısına ulaşılmıştır. Günümüzde cep telefonlarında bile bu renk seviyesine ulaşılmıştır. Buradan 1024 x 768 çözünürlükte 24 bitlik renge sahip 32 bit vidyo kipinin ihtiyacı olan belleği hesaplayabiliriz:

1024 x 768 x (32/8) = 3145728 bayt = 3MB

Dikkat ettiyseniz her piksele 24 bitlik renk yetmesine rağmen hesaplamada 32 bit kullanıldı. Bunun nedeni grafik kartlarının işlemcileri 8, 16, 32 ve 64 bitlik veriler ile çalışmaktadır. Örnek olarak 8 bitlik AL yazmacının üst 4 bitinin değeri ile işlem yapmak için bu 8 bitlik AL yazmacının tamamını okumalıyız. Özel olarak 24 bitin kullanılması çoğu hızlandırma özelliğini etkileyeceğinden her piksele 32 bit ayrılmaktadır. Ama bu 32 bitin sadece 24 biti anlamlıdır bu yüzden 1 bayt boşa gitmektedir. Bazı sistemlerde ise saydamlık veya maske değeri olarak kullanılmaktadır.

Yukarıdaki INT 10H kesmesi 00H alt yordamı listesinde 1024 x 768 çözünürlükte 32 bitlik vidyo kiplerini göremedik. Peki 1024 x 768 çözünürlüğe nasıl geçebiliriz? Tam olarak yanıt vermeden önce birkaç noktaya daha deyinmemiz gerekiyor.

VGA kartlardan sonra **SVGA** (**S**uper **VGA**) kartları çıktı ve bu kartlar daha yüksek çözünürlükleri desteklemeye başladılar. Tabii bir sorun vardı. Her kart farklı özellikte farklı vidyo kipleri sunuyordu ve tam olarak ortada bir standart yoktu. Hatta bir firmanın ürettiği farklı SVGA modelleri bile birbirleriyle uyulu değildi. Bunun üzerine yaklaşık 150 tane büyük grafik kartı üreticisi yazılım yazımını kolaylaştırmak BIOS'a bir eklenti oluşturdular ve bu eklenti **VESA** (**V**ideo **E**lectronics **S**tandards **A**ssociation – Vidyo Elektroniği Standardizasyon Birliği) SVGA adını aldı. Bunun üzerine INT 10H kesmesine bazı eklemeler yapıldı çünkü yeni vidyo tiplerinin bir şekilde desteklenmesi gerekiyordu. VESA SVGA eklentisini kullanabilmek için AX yazmacına 4F02H değeri, BX yazmacına istenen vidyo kipinin numarası yazılmalı ve ardından INT 10H

kesmesi çağırılmalıdır. Aşağıda BX yazmacının değerleri ve bu değerlerin hangi vidyo kiplerine karşılık geldiğini gösteren bir tablo göreceksiniz:

BX Değeri	Vidyo Kipi	Çözünürlük	Renk
0100H	Grafik	640 x 400	256
0101H	Grafik	640 x 480	256
0102H	Grafik	800 x 600	16
0103H	Grafik	800 x 600	256
0104H	Grafik	1024 x 768	16
0105H	Grafik	1024 x 768	256
0106H	Grafik	1280 x 1024	16
0107H	Grafik	1280 x 1024	256
0108H	Yazı	80 x 60	16
0109H	Yazı	132 x 25	16
010AH	Yazı	132 x 43	16
010BH	Yazı	132 x 50	16
010CH	Yazı	132 x 60	16
010DH	Grafik	320 x 200	32768
010EH	Grafik	320 x 200	65536
010FH	Grafik	320 x 200	16.7 milyon
0110H	Grafik	640 x 480	32768
0111H	Grafik	640 x 480	65536
0112H	Grafik	640 x 480	16.7 milyon
0113H	Grafik	800 x 600	32768
0114H	Grafik	800 x 600	65536
0115H	Grafik	800 x 600	16.7 milyon
0116H	Grafik	1024 x 768	32768
0117H	Grafik	1024 x 768	65536
0118H	Grafik	1024 x 768	16.7 milyon
0119H	Grafik	1280 x 1024	32768
01AH	Grafik	1280 x 1024	65536
01BH	Grafik	1280 x 1024	16.7 milyon

Ayrıca eğer BIOS'unuz seçtiğiniz vidyo kipini destekliyorsa AH yazmacı 00H değerini geri döndürür. Aksi takdirde 01H değerini geri döndürür. Bu şekilde yazılımlarınıza otomatik olarak birkaç vidyo kipi denetip uygun olan birini kullandırabilirsiniz.

Yeri gelmişken VESA SVGA eklentilerinin alt yordamlarına bir bakalım:

AX Yazmacı Değeri	İşlev
4F00H	SVGA bilgilerini çıktı olarak verir.
4F01H	SVGA vidyo kipi bilgilerini çıktı olarak verir.
4F02H	SVGA veya VGA vidyo kiplerini ayarlamak için kullanılır. BX yazmacına istenen vidyo kipinin kodu girilmelidir.
4F03H	Hangi SVGA veya VGA vidyo kipinin kullanımda olduğunu çıktı olarak BX yazmacında verir.
4F04H	SVGA vidyo durumunu kaydetme veya geri yüklemede kullanılır. Çoklu işlem yapan işletim sistemleri sıklıkla kullanır.

4F05H	Bilgisayarlar tasarlandığında, özellikle de IBM PC tasarlandığında 64KB bellek o zamanın kullanımına göre muazzam bir rakamdı. Bu yüzden bilgisayarlar tasarlanırken "yeterince" büyük olan 64KB boyutunda bir bellek alanı vidyo belleğine ayrıldı. Tabi işler zamanla değişti ve hızlı bir şekilde gereksinimler bunun üzerine çıktı. Mesela 1280 x 1024 çözünürlük ve 24 bit renkte çalışabilmek için 1280 x 1024 x 3 = 3.75MB bellek gerekmektedir. Bu değer ayrılan 64KB değerinin 60 katıdır. Sistemlerin 1280 x 1024 x 24 gibi yüksek çözünürlüklerde çalışabilmeleri için bir yöntem geliştirilmiştir. Bu yöntemde gereksinim olan bellek 64KB'lık parçalara ayrılmıştır. Yani büyük bir ekran küçük 64KB lık parçalara ayrılıp belleğin çeşitli yerlerinde tutulmaktadır. Bu parçalara "bellek pencereleri" denmiştir. Yukarıda da bahsettiğimiz gibi her SVGA kartında bu işlem farklı şekilde yapılmaktaydı ve VESA'nın piyasaya sürülmesi ile bu iş büyük ölçüde kolaylaştı. VESA sayesinde hangi "bellek penceresinin" kullanıma alınması gerektiği ayarlanabilmekte ve bu sayede bu yüksek çözünürlüklere ulaşılabilmektedir. Mesela 1024 x 768 çözünürlükte ve 256 renkte grafik kipinde çalışalım. Kordinatları (600,300) olan pikseli ele alalım. (768 x 300) + 600 = 231000 ıncı bayta erişmemiz gerekmektedir. Her "bellek penceresi" 64KB olduğundan bu piksel üçüncü bellek penceresinin 34392 inci pikselidir.
4F06H	Lojik ekran genişliğini ayarlamaya yarar. 1024 x 768 çöznürlüğünde çalışırken 2048 lojik ekran genişliği seçilirse görüntünün sadece yarısı görünür. Diğer yarı başka bir ekran tarafından gösterilebilir (ekran kartının çift ekran desteğinin olması gerekir).
4F07H	Lojik ekranın neresinin gerçek ekranda gösterileceğini ayarlamak için kullanılır.
4F08H	Kullanımda olan paletin bit genişliğini almak veya değiştirmek için kullanılır.

Yani VESA SVGA eklentilerini kullanarak günümüzde kullanılan 1024 x 768 çözünürlüğü ve 24 bit rengi elde edebiliriz. Ama ne yazık ki DOS ortamında yüksek çözünürlükler kullanılamadığından 1024 x 768 çözünürlük ve 24 bit renk kipini bu bölümde örnekleyemeyeceğiz.

Peki neden bu standart çıktığından beri renk sayısı artırılmadı? Neden hala 16.7 milyon renk çoğu bilgisayarda en yüksek seçenek olarak kalmakta? Bilimsel araştırmaların sonucunda insan gözünün kırmızı, yeşil veya mavinin 100 derinlik katmanını algılayabildiğini ortaya koydu. Yani RGB değeri olarak (50,100,150) ve (50,100,151) renklerini insanlar ayırt edememekteler. Yalnız (50,100,150) ve (50,100,152) renklerindeki derinlik farkını fark edebilmekteler. Bu kırmızı, yeşil ve mavi renk başına 256 tane derinlik insanlara çokca yetmektedir. Yani 16.7 milyon renk biz insanlara yönelik fiziki bir sınırlamadır. Tabi gelecek yıllarda yapay zeka ve robotik teknolojilerinin gelişmesiyle robotlara yönelik olarak bu renk sayısı artırılabilir.

Son olarak artık yeni VESA standartlarının piasaya sürülmeyeceğini söyleyelim. VESA'nın son sürümü ile birlikte ekran kartını sınama özellikleri yazılımcıya sunulmaktadır ve istenen bir çözünürlük verildiğinde o çözünürlük veya ona en yakın çözünürlük ayarlanmaktadır.

8.1.8. Vidyo Kartı Yazmaçları

Ekran kartlarında palet üzerinde işlem yapabilmek için ayrılmış 4 adet yazmaç bulunmaktadır. Bu yazmaçları tanıtmadan önce paletin ne işe yaradığını ve paleti neden değiştirmek isteyeceğinizi merak edebilirsiniz.

256 renkli bir vidyo kipini ele alalım. Ön tanımlı olan 256 rengin içinde belki sizin ihtiyacınız olan renk yok. O zaman ya daha yüksek bir renk çözünürlüğünü kullanmanız gerekecek yada paletinizdeki bazı kullanmadığınız renkleri istediğiniz renk değerleriyle değiştirebileceksiniz.

Palet üzerinde işlem yapmamızı sağlayan dört adet yazmaç vardır.

İlk yazmacımıza 03C6H adresinden erişebiliriz. Bu yazmaç ile palet maskesi ayarlanabilir. Girdi olarak 8 bitlik maske değerini alır veya çıktı olarak 8 bitlik kullanımda olan maske değerini verir. Bu yazmaç otomatik olarak FFH değerini almaktadır ve büyük bir ihtimalle bu yazmaç üzerinde işlem yapma ihtiyacı hissetmeyeceksiniz.

İkinci yazmacımız 03C7H adresindeki okuma yazmacıdır. Bu yazmaç ile paletteki belirli bir girişi okumak için seçebiliriz.

Üçüncü yazmacımız 03C8H adresindeki yazma yazmacıdır. Bu yazmaç ile paletteki belirli bir girişi yazmak için seçebiliriz.

Dördüncü ve sonuncu yazmacımız 03C9H adresinde bulunmaktadır. Bu yazmaç ile 03C7H adresindeki yazmaç kullanılarak seçilen girişin değeri okunabilir veya 03C8H adresindeki yazmaç kullanarak seçilen girişin değeri değiştirilebilir. Bu yazmaçtan veri okunurken veya yazılırken dikkat edilmesi gereken şey 3 kez okuma veya yazma yapılmasıdır. Okunan veya yazılan ilk değer o girişin kırmızı renk değeri ikinci değer yeşil renk değeri ve üçüncü değer ise mavi renk değeridir.

Şimdi paletin nasıl değiştirilebileceğini biliyoruz. Diyelimki ekrana sürekli bir şeyler çizdiriyoruz ve çizim devam ederken palette çizilirken kullanılan bir rengin değerini değiştirdik. Ne olur? Ekranda renkler bozulacaktır. Bunu önlemek için bir şekilde ekranın dikey tekrarlanma anını bekleyip o anda paleti değiştirmemiz gerekmektedir. Bunun için ekran kartının 03DAH adresindeki yazmacını kullanabiliriz. Dikey tekrarlanma anını beklemek için aşağıdaki yordamı kullanabiliriz:

```
1  DIKEY_TEKRAR_BEKLE: MOV DX,03DAH
2  DONGU1: IN AL,DX
3  TEST AL,08H
4  JNZ DONGU1
5  DONGU2: IN AL,DX
6  TEST AL,08H
7  JZ DONGU2
8  RET
```

Yordamımızın ilk döngüsünde AL yazmacının değeri 08H olana kadar bekliyoruz. Ardından ikinci döngüye geçiyoruz ve AL yazmacının değeri 08H'den farklı bir değer olana kadar bekliyoruz. Bu yazdığımız yordamı çağırdıktan hemen sonra paleti değiştirmenizin herhangi bir sakıncası kalmayacaktır.

Aslında yukarıda açıklanan yazmaç sayısından çok daha fazla yazmaç ekran kartınızda mevcuttur ve isterseniz bunların hepsini kullanabilirsiniz ama genelde kullanılan yazmaçlar yukarıdakilerdir.

8.1.8.1. Örnek

Bazı temel konuları işlediğimize göre ilk örneğimize geçebiliriz. İlk örneğimizde 40 x 25 çözünürlükte 16 renkli yazı vidyo kipini kullanacağımızı söylemiştik. O zaman ekrana "MERHABA DUNYA!" yazısını yazdırmayı deneyelim.

Öncelikle 40 x 25 çözünürlük ve 16 renkli yazı moduna geçmek için gerekli INT 10H kesmesi 00H alt ordamı değerini bilmemiz gerekiyor. 00H alt yordamının girdi olarak AL yazmacını aldığını biliyoruz. Tablolarımıza bakarak AL yazmacına girmemiz gereken değerin 01H olduğunu görebiliriz. Ayrıca ekrana siyah arkaplana sahip beyaz harfler yazmak istiyoruz. Bunun için yazı kipinde yazı tipi için 00001111B değerini kullanmamız gerekmektedir. Kodumuza geçelim:

```
1  ORG 0100H
2  PUSH CS
3  POP DS
4  JMP BASLA
5
6  VIDYOKIPI DB 01H
7  M1_B DB 'MERHABA DUNYA!'
8  M1_S DB '00'
9
10 BASLA: MOV AX,0B800H
11 MOV ES,AX
```

```
12  CALL VIDYO_KIPI_SEC
13  MOV CX,M1_S - M1_B
14  MOV SI,0FFFEH
15  MOV AH,0FH
16  MOV DI,M1_B -1
17
18  DONGU: INC SI
19  INC SI
20  INC DI
21  MOV AL,[DI]
22  ES MOV [SI],AX
23  LOOP DONGU
24
25  BEKLE: MOV AH,1
26  INT 16H
27  JZ BEKLE
28  MOV AH,00H
29  INT 16H
30  CMP AL,61H
31  JE SON
32  JMP BEKLE
33  SON: CALL VIDYO_KIPI_SEC
34  INT 03H
35
36  ;vidyo kipini seçer ve eski kipi kaydeder
37  VIDYO_KIPI_SEC: MOV AH,0FH
38  INT 10H
39  MOV BL,AL
40  MOV AH,00H
41  MOV AL,[VIDYOKIPI]
42  MOV [VIDYOKIPI],BL
43  INT 10H
44  RET
```

2. ve 3. satırdaki işlemler ile DS kesim yazmacına CS kesim yazmacının değerini kaydettik.

6. satırda bir değişken tanımladık. Bu değişken seçilecek olan vidyo kipinin kodunu tutmaktadır. 7. ve 8. satırlarda ise "MERHABA DUNYA!" dizgisini tanımladık.

10. satırda AX yazmacına B800H değerini kaydettik ve 11. satırda bu AX değerini ES kesim yazmacına kaydettik.

12. satırda 37. satırdan başlayan *VIDYO_KIPI_SEC* yordamımızı çağırdık. 37. ve 38. satırdaki işlemler ile INT 10H kesmesinin 0FH yordamını kullanarak şu an kullanımda olan vidyo kipini AL yazmacına kaydettik. 39. satırda bu AL yazmacı değerini BL yazmacına kopyaladık. Bunu yapmamızın nedeni kullanımda olan vidyo kipini yeni vidyo kipine geçince VIDYOKIPI değişkenimize kaydetmek istememiz. 40. satırda AH yazmacına 00H değerini yükledik. Bu değer 43. satırdaki INT 10H kesmesinin 00H alt yordamına geçmekte kullanılacak. 41. satırda AL yazmacına VIDYOKIPI değişkenimizin içeriğini kaydettik ve 42. satırda kullanımda olan vidyo kipinin kodunu VIDYOKIPI değişkenine kaydettik. 43. satırdaki INT 10H kesmesi ve 00H yordamı ile vidyo kipini yeni vidyo kipine yani başlangıçta VIDYOKIPI değişkeninde bulunan vidyo kipi değerine ayarladık. 44. satırdaki RET komutu ile 13 satıra atladık.

13. satırdaki işlem ile CX yazmacına "MERHABA DUNYA!" dizgisinin uzunluğunu kaydettik. 14., 15. ve 16. satırlarda SI, AH ve DI yazmaçlarına bir ilk değer verdik. AH yazmacına 0FH değerini vermemizin nedeni siyah arkaplan üzerine beyaz yazı yazmak için gerekli renk kodunun 0FH olmasıdır. SI yazmacına FFFEH değerini girmemizin nedeni ise 18. satırdaki döngümüzün her seferinde SI yazmacının değerini 2 artırması. Yani döngünün ilk dünüşünde SI yazmacının değeri 0000H değerini alacaktır. Benzer bir durum DI yazmacının ilk değeri için de geçerlidir. DI yazmacının ilk değeri "MERHABA DUNYA!" dizgisinin adresinden bir bayt daha düşük bir değer almaktadır. Döngüyü farklı bir şekilde tasarlarsak ilk değer olarak SI yazmacına direk 0000H da yazılabilir. Mesela döngüyü aşağıdaki şekilde tasarlasaydık ilk SI değeri olarak 0000H

vermemiz ve DI yazmacına da ilk değer olarak "MERHABA DUNYA!" dizgisinin ilk karakterinin adresini vermemiz uygun olacaktı.

```
18  DONGU: MOV AL,[DI]
19  ES MOV [SI],AX
20  INC SI
21  INC SI
22  INC DI
23  LOOP DONGU
```

Döngüyü yukardaki şekilde yazmak performans açısından pek bir şey değiştirmemektedir. SI yazmacının ilk değerinin 0000H olması da aynı şekilde pek bir etki yaratmamaktadır ama DI yazmacının ilk değerindeki fazladan hesaplama ortadan kaldırıldığından az da olsa bir performans kazancı sağlamış olduk.

18. satırdaki karakter yazdırma döngümüze geçelim. Ilk olarak SI yazmacının değeri 2 artırılıyor ve bunun ardından 21. satırda DI yazmacının değeri 1 artırılıyor. 20. satırda DI yazmacının değerinin işaretlediği adresteki veri –ki bu veri bizim "MERHABA DUNYA!" dizgimizin bir karakteridir – AL yazmacına kaydediliyor. 22. satırdaki işlemle ise ES:[SI] adresine AX yazmacı kaydediliyor. Bu sayede vidyo belleğine karakterimizi ve renk kodumuzu kaydetmiş oluyoruz.

25. ve 34. satır arasındaki kodlar ile program 'a' harfine basılana kadar açık tutuluyor. Böyle bir şey yapmasaydık program hemen kapanacak ve herhangi bir şey görmemiz mümkün olmayacaktır.

8.1.8.2. Örnek

Yazı modundaki örneğimizin ardından 320 x 200 çözünürlüğündeki 256 renkli grafiksel VGA vidyo kipini kullanan bir örnek yapalım. Bu örnek iki farklı program içerecektir. İlk programımız ile ekrana biz çizgi çizdireceğiz. İkinci programımızda ise bu çizginin rengini palet ile oynayarak değiştireceğiz.

İlk programımızın kodlarına geçmeden önce 320 x 200 çözünürlük ve 256 renk için kullanmamız gereken parametrelere bakalım. INT 10H kesmesi 00H yordamı için AL değeri olarak 13H vermemiz gerektiğini tablodan bakarak öğrenebiliriz. Yazı modundan farklı olarak grafiksel modda B800:0000 adresinden başlama gibi bir zorunluluğumuz yok. Ayrıca dikkat etmemiz gereken başka bir şey ise 320 x 200 = 64000 tane pikselimizin olduğu. Yani 0000H dan başlayak F9FFH tane göreli piksel adresimiz var. Yani A000:0000 dan başlayarak A000:F9FF adresine kadar olan pikselleri kullanabiliriz. Ekrana düz bir çizgi çizdirmemiz çok kolay olacağından biz sol üst köşeden her seferede bir sütun sağa kayan ve bir satır sağa kayan bir çizgi ve benzer şekilde sağ üst köşeden her seferde bir satır aşağı ve bir sütun sola kayan bir çizgi çizelim. Yani renklendirmemiz gereken piksellerin göreli konumlarını aşağıdaki formülle bulabiliriz:

1. çizgi: 0000H + (N x 0141H)
2. çizgi: 013F + (N x 13FH)

Burada *N* 00H dan başlayarak C7H a kadar her kademede 01H artan bir sayı.

Bütün bunlardan bahsettikten sonra kodumuza geçebiliriz:

```
1   ORG 0100H
2   PUSH CS
3   POP DS
4   JMP BASLA
5
6   VIDYOKIPI DB 13H
7
8   BASLA: MOV AX,0A000H
9   MOV ES,AX
10  CALL VIDYO_KIPI_SEC
11  MOV CX,0C8H
12  MOV AL,66H
13  MOV SI,0000H
14  MOV DI,013FH
15
16  DONGU: ES MOV BYTE [SI],AL
```

```
17  ADD SI,0141H
18  ES MOV BYTE [DI],AL
19  ADD DI,013FH
20  LOOP DONGU
21
22  BEKLE: MOV AH,1
23  INT 16H
24  JZ BEKLE
25  MOV AH,00H
26  INT 16H
27  CMP AL,61H
28  JE SON
29  JMP BEKLE
30  SON: CALL VIDYO_KIPI_SEC
31  INT 03H
32
33  VIDYO_KIPI_SEC: MOV AH,0FH
34  INT 10H
35  MOV BL,AL
36  MOV AH,00H
37  MOV AL,[VIDYOKIPI]
38  MOV [VIDYOKIPI],BL
39  INT 10H
40  RET
```

Görüleceği gibi yukarıdaki kod yazı kipi için yazdığımız koda çok benziyor. Sadece farklılıklardan bahsedelim.

ES kesim yazmacına bu sefer B800H yerine A000H yazdık. Bunu yapmamızın nedeni yukarıda da bahsettiğimiz gibi grafik kipin A000H adresinden başlamasıdır. Ayrıca yazı modunun aksine her piksel 2 değil sadece 1 bayt yer tutmaktadır ve sadece pikselin renk kodu yazılmalıdır. Örneğimizde AL yazmacındaki 66H değerini renk kodu olarak kullandık. Yani 256 renk değerinden 67. yi kullandık. Her göreli konum artışı bir sonraki piksele denk gelmektedir.

Çizim döngümüz ise gayet basit. İlk çizginin göreli konum adresi SI ve ikinci çizginin göreli konum adresi DI adresinde tutulmakta. SI yazmacının değeri her döngü dönüşünde 0141H artırılarak 1 satır ve 1 sütün sağa kaydırılıyor. Benzer şekilde her döngü dönüşünde DI yazmacının değeri 013FH artırılmakta ve bu sayede bir satır alta ve bir satır sola kaymaktadır. Peki bu 0141H ve 013FH değerleri nereden geldi? Nasıl hesaplandı?

320 ondalık değerinin onaltılık tabandaki karşılığı 0140H dır. Bu çözünürlükte ekranın ilk satırının başı 0000H göreli konumunda ve ilk satırın sonu 0000H + 013FH = 013FH göreli konumundadır. 0000H'ın 0140H artırılması bir alt satırın 0. sütununa geçilmesine neden olur. 0141H artırılması ise bir alt satırın 1. sütununa geçilmesine neden olur. Bu yüzden her seferde 0141H lık bir artırım yapmaktayız. Benzer bir durum 2. çizgi için de geçerli.

11. satırda değeri verilen CX yazmacı döngünün ne kadar döneceğini belirlemektedir. En üst satırdan en alt satıra 0 dan başlayarak gidileceği düşünülürde en son yani en alttaki satır 199. satır olacaktır. 199 ondalık değerinin onaltılık tabandaki değeri C7H dır. Dongunun yapısı gereği bu değerin 1 fazlası gerektiğinden C8H değerini kullandık.

Şimdi örneğimizin 2. kısmına geçelim. Yukarıda paletin 67. girişini kullandık. Şimdi paletin 67. girişine paleti değiştirerek farklı bir değer verelim. Paleti değiştirmek için kullanabileceğimiz yazmaçların 03C7H, 03C8H ve 03C9H adreslerinde bulunduklarını görmüştük.

```
1  ORG 0100H
2  PUSH CS
3  POP DS
4  JMP BASLA
5  VIDYOKIPI DB 13H
6  BASLA: MOV AX,0A000H
7  MOV ES,AX
8  CALL VIDYO_KIPI_SEC
```

```
 9  MOV CX,0C8H
10  MOV AL,66H
11  MOV SI,0000H
12  MOV DI,013FH
13  DONGU: ES MOV BYTE [SI],AL
14  ADD SI,0141H
15  ES MOV BYTE [DI],AL
16  ADD DI,013FH
17  LOOP DONGU
18
19  MOV CX,1111H
20  LP1: PUSH CX
21  MOV CX,0FFFFH
22  LP2: LOOP LP2
23  POP CX
24  LOOP LP1
25
26  CALL DIKEY_TEKRAR_BEKLE
27  MOV DX,03C8H
28  MOV AL,66H
29  OUT DX,AL
30  INC DX
31  MOV AL,0FFH
32  OUT DX,AL
33  MOV AL,00H
34  OUT DX,AL
35  MOV AL,00H
36  OUT DX,AL
37
38  BEKLE: MOV AH,1
39  INT 16H
40  JZ BEKLE
41  MOV AH,00H
42  INT 16H
43  CMP AL,61H
44  JE SON
45  JMP BEKLE
46  SON: CALL VIDYO_KIPI_SEC
47  INT 03H
48
49  DIKEY_TEKRAR_BEKLE: MOV DX,03DAH
50  DONGU1: IN AL,DX
51  TEST AL,08H
52  JNZ DONGU1
53  DONGU2: IN AL,DX
54  TEST AL,08H
55  JZ DONGU2
56  RET
57
58  VIDYO_KIPI_SEC: MOV AH,0FH
59  INT 10H
60  MOV BL,AL
61  MOV AH,00H
62  MOV AL,[VIDYOKIPI]
63  MOV [VIDYOKIPI],BL
64  INT 10H
65  RET
```

Yukarıdaki örneğin bir önceki örnekten tek farklı 19. ve 36. satırlar arasındaki kodlardır.

19. ve 24. satırlar arasındaki işlemler sayesinde program bir süre beklemektedir. Beklemesinin nedeni her dönüşte belirli ama çok kısa bir süre harcayan döngülerin iç içe girdiklerinde yaklaşık olarak bu sürenin 300000000 katına çıkması.

26. ve 36. satırlar arasındaki işlemler sayesinde de paletin 67. girişinin rengi değiştirilmiştir. Bunun için öncelikle 27. ve 29. satırlar arasındaki kodlar ile 03C8H adresindeki yazmaca değiştirilecek palet konumunun numarası verilmiştir. Ardından 30. ve 26. satırlardaki işlemler aracılığıyla önce kırmızı renk değeri, ardından yeşil renk değeri ve son olarak da mavi renk değeri 03C9H yazmacına yazılmıştır. Bu 3 bayt yazıldıktan sonra otomatik olarak bir sonraki palet konumuna geçilmektedir.

49. satırdaki *DIKEY_TEKRAR_BEKLE* yordamımızı daha önceden tanıtmıştık.

8.1.8.3. Örnek

Bu örneğimiz mantık olarak yukarıdaki örneklerle aynı olsa da daha "renkli" olacak. Ekranın ortasına 10 piksel kalınlığında bir çizgi çizip bu çizgiyi kademeli olarak renklendireceğiz.

```
 1  ORG 0100H
 2  PUSH CS
 3  POP DS
 4  JMP BASLA
 5  VIDYOKIPI DB 13H
 6  BASLA: MOV AX,0A000H
 7  MOV ES,AX
 8  CALL VIDYO_KIPI_SEC
 9  MOV CX,0C8H
10  MOV AL,00H
11  MOV SI,009BH
12
13  DONGU_DIKEY: PUSH CX
14  MOV CX,0AH
15  DONGU_YATAY: ES MOV BYTE [SI],AL
16  INC SI
17  LOOP DONGU_YATAY
18  POP CX
19  INC AL
20  ADD SI,0136H
21  CALL GECIKME
22  LOOP DONGU_DIKEY
23
24  BEKLE: MOV AH,1
25  INT 16H
26  JZ BEKLE
27  MOV AH,00H
28  INT 16H
29  CMP AL,61H
30  JE SON
31  JMP BEKLE
32  SON: CALL VIDYO_KIPI_SEC
33  INT 03H
34
35  GECIKME: PUSH CX
36  MOV CX,0111H
37  LP1: PUSH CX
38  MOV CX,0FFFFH
39  LP2: LOOP LP2
40  POP CX
41  LOOP LP1
42  POP CX
43  RET
44
45  VIDYO_KIPI_SEC: MOV AH,0FH
```

```
46  INT 10H
47  MOV BL,AL
48  MOV AH,00H
49  MOV AL,[VIDYOKIPI]
50  MOV [VIDYOKIPI],BL
51  INT 10H
52  RET
```

Bir önceki örnekteki gecikme kodlarımızı bir yordam (35. satırda) olarak bu örnekte de kullandık. Tabii ki gecikme bittiğinde gecikme çağırılmadan önceki CX yazmacı değerlerini kaybetmemek için yordamın başına ve sonuna PUSH-POP komutları ekledik ve gecikmeyi bir önceki örneğe göre biraz kısalttık.

13. ve 22. satırlar arasındaki işlemler sayesinde renkli çizgimizi çizdik. Çizdirilirken gördüğümüz çizilme efektini gecikme yordamımız ile sağladık.

9. Basit Bir İşletim Sistemi

Bu bölümde basit bir işletim sistemi yazacağız. Tabi ki yazmaya başlamadan önce işletim sistemleri hakkında biraz bilgi edinmemiz gerekmektedir.

9.1. İşletim Sistemi Nedir?

İşletim sistemi bilgisayarın açıldıktan sonra hafızaya yüklenen ve bilgisayar donanımının yönetimi, yazılımların çalışmasını sağlayacak ortamın hazırlanmasında görevli özel bir yazılımdır.

Ayrıca işletim sistemi yazılımların belleğe ve girdi/çıktı donanımına erişimini sağlamakla sorumludur. Bu görevini her zaman yapmakla yükümlü değildir, yani girdi/çıktı işlemlerini serbest de bırakabilir.

9.2. Basit İşletim Sistemimiz

İşletim sistemimiz aşağıdaki parçalardan oluşacak:

- Önyükleme yazılımı
- Genel menü yazılımı
- Hesap makinası yazılımı
- Basit bir metin düzenleyici yazılım
- Seri iletişim yazılımları
- 1024 x 768 piksel çözünürlük ve 24 bit renk vidyo kipinde ekrana resim çizdirme yazılımı

İşletim sistemimiz ve işletim sistemimize yazdığımız programlar diskete kaydedilecektir. İlk olarak önyükleme programı belleğe yüklenecek ve ardından da önyükleme kodumuz sayesinde genel menü yazılımımız belleğe yüklenip çalıştırılacaktır.

Önyükleme yazılımımız ilk yüzün, ilk pistinin, ilk sektöründe bulunacaktır. Zaten BIOS ayarları gereği önyükleme programımızın burada bulunması zorunludur. BIOS disketin ilk sektörünü okuyup 07C0H:0000H bellek adresine kaydeder ve bu adresten başlayarak işlemci kodu çalıştırır. Önyükleme programımız bazı ayarlamaları yaptıktan sonra disketten genel menü programını okutup belleğin 1000H:0000H adresine kaydedecek ve bu adrese atlanacaktır.

Genel menü yazılımımız disketin ikinci sektöründe bulunacaktır. Genel menü yazılımımız sayesinde diğer yazılımlarımımızı disketten okutup belleğe yükletip çalıştırabileceğiz. Genel menü yazılımı disketten okuduğu yazılımları 1000H:0420H adresine kaydedecek ve buraya atlayacaktır. Her yazılım sonlandığında geri genel menü yazılımına yani 1000H:0000H adresine atlayacaktır. Ayrıca bilgisayarın yeniden başlatılması ve kapatılması gibi işlemler genel menü yazılımı sayesinde yapılabilecektir.

Hesap makinası yazılımımız disketin dördüncü sektöründe bulunacaktır. Ekrandan sayılar "w","a","s","d" ve "enter" tuşları ile seçilebilecek ve seçilen sayılar toplanabilecektir.

Metin düzenleyici yazılımımız disketin beşinci sektöründe bulunacaktır. Yazılım açıldığı zaman boş bir sayfa açılacak ve bu sayfaya klavyeden bazı karakterler girilebilecektir.

Ekrana resim çizdirme yazılımımız disketin altıncı sektöründe bulunacaktır. Çizdireceği resim disketin ikinci yüzünün ilk pistinin ilk sektöründen başlayarak diskette kayıtlı tutulacaktır. Çizdirme işlemi yapılırken resim disketten okunacaktır.

Yukarıda yazacağımız yazılımların hepsini bir şekilde diskete yazdırmamız gerekmektedir. Disket üzerinde yukarıdaki açıklamalardan da anlayacağınız gibi basit bir dosya sistemi oluşturduk ve bu dosya sistemine uyacak şekilde yazılımların diskete kaydedilmesini sağlayacak ayrı bir yazılıma ihtiyacımız var. Bu işlemi DOS ortamında yapmak için bir .com yazılımı yazmamız gerekmektedir. Yazacağımız bu yazılım .bin dosyalarımızı kullanarak disketimizi dolduracaktır.

9.2.1. Disket Yükleyici

Disket yükleyici yazılımımızın kodu aşağıda bulunmaktadır.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;           BASİT İŞLETİM SİSTEMİ                                    ;
;                                                                    ;
;           ISLETIM SISTEMI DISKETI HAZIRLAMA PROGRAMI               ;
```

```
;                                                                                   ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  p0.bin          Sektör 1 Yüz 1     ;
;  p1.bin          Sektör 2 Yüz 1     ;
;  p2.bin          Sektör 4 Yüz 1     ;
;  p3.bin          Sektör 5 Yüz 1     ;
;  p4.bin          Sektör 6 Yüz 1     ;
;  p5.bin          Sektör 7 Yüz 1     ;
;  p6.bin          Sektör 8 Yüz 1     ;
;  resim.bin       Sektör 1 Yüz 2     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ORG 0100H
JMP BASLANGIC
DOSYA1 DB "C:\resim.bin", 00H          ; Diskete yüklenecek resim dosyasının adresi
DOSYA2 DB "C:\p0.bin", 00H             ; On yükleme kodunun adresi
DOSYA3 DB "C:\p1.bin", 00H             ; Genel menü yazılımının adresi
DOSYA4 DB "C:\p2.bin", 00H             ; Hesap makinesi yazılımının adresi
DOSYA5 DB "C:\p3.bin", 00H             ; Metin düzenleyici yazılımının adresi
DOSYA6 DB "C:\p4.bin", 00H             ; Resim gösterici yazılımının adresi
DOSYA7 DB "C:\p5.bin", 00H             ; Seri iletişim verici
DOSYA8 DB "C:\p6.bin", 00H             ; Seri iletişim alıcı
DOSYAIMLECI DW 0000H                   ; Dosya imlecinin tutulduğu değişken
GECICI DW 0000H                        ; Yazmaç değeri kaydetmek için kullanılan geçici değişken
MESAJ1 DB "* Diskette resim dosyasi yazildi.        *",0AH,0DH,00H
MESAJ2 DB "* On yukleme kodu diskete yazildi.        *",0AH,0DH,00H
MESAJ3 DB "* Genel menu kodu diskete yazildi.        *",0AH,0DH,00H
MESAJ4 DB "* Hesap makinasi kodu diskete yazildi.    *",0AH,0DH,00H
MESAJ5 DB "* Metin duzenleyici kodu diskete yazildi.  *",0AH,0DH,00H
MESAJ6 DB "* Resim gosterme kodu diskete yazildi.    *",0AH,0DH,00H
MESAJ7 DB "* Seri iletisim alici kodu diskete yazildi.*",0AH,0DH,00H
MESAJ8 DB "*Seri iletisim verici kodu diskete yazildi.*",0AH,0DH,00H
TAMPON DB 512 DUP(00H)                 ; Tampon değişkeni
BASLANGIC: PUSH CS                     ; CS yazmacını yığına at
POP DS                                 ; DS yazmacına yığından veri yükle
PUSH CS                                ; CS yazmacını yığına at
POP ES                                 ; ES yazmacına yığından veri yükle
; Resim dosyasını diskete yaz
MOV CL,01H                             ; CL yazmacına 01H kaydet (Disket 1. sektör)
MOV CH,00H                             ; CH yazmacına 00H kaydet (Disket 1. Pist)
PUSH CX                                ; CX yazmacının değerini yığına kaydet
MOV SI,DOSYA1                          ; DOSYA1 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC                          ; Dosyayı aç
DONGU1: MOV AH,3FH                     ; AH yazmacına 3FH yükleyerek 3FH alt yordamını ayarla
MOV DX, TAMPON                         ; Tampon adresini ayarla
MOV CX,0200H                           ; 512 bayt oku
MOV SI, DOSYAIMLECI                    ; SI yazmacına DOSYAIMLECI'nin adresini kaydet
MOV BX,[SI]                            ; Dosya imlecini ayarla
INT 21H                                ; INT 21H çağır
JC HATA3                               ; Hata var. HATA3 etiketine atla
CMP AX, CX                             ; AX ile CX yazmaçlarını karşılaştır
JNE DOSYA_SONU                         ; Dosya sonuna gelindiyse DOSYA_SONU etiketine atla
; Okunan veriyi diskete yaz
POP CX                                 ; CX yazmacının değerini yığından al
MOV AH,03H                             ; AH yazmacına 03H yazarak 03H alt yordamını belirt
MOV AL,01H                             ; Yazılacak sektör sayısı (512bayt = 1 sektör)
MOV DH,01H                             ; Yazılacak yüz (01H = 2. yüz)
MOV DL,00H                             ; Sürücü numarası ( 00H = diskete yaz)
MOV BX, TAMPON                         ; BX yazmacına tampon adresini yaz
INT 13H                                ; INT 13H çağır
JC HATA4                               ; Hata var. HATA4 etiketine atla.
INC CL                                 ; CL yazmacının değerini artır (sektör numarasını)
CMP CL,13H                             ; 18 sektör tamamlanmış mı?
JNE DEVAM                              ; Hayır tamamlanmamış artırmaya devam et.
INC CH                                 ; 18 sektör tamamlandı. 1 pist artır.
MOV CL,01H                             ; Sektör numarasını ayarla.
DEVAM: PUSH CX                         ; CX yazmacını yığına kaydet
JMP DONGU1                             ; Sonraki baytı oku
DOSYA_SONU: CALL DOSYA_KAPAT           ; Açık dosyayı kapat
MOV SI,MESAJ1                          ; Resim dosyasının yazıldığını belirt
CALL SATIR_YAZ                         ; Satırı yaz
                                       ; Önyükleme kodunu diskete yaz
MOV CL,01H                             ; CL yazmacına 01H kaydet (Disket 1. sektör)
MOV CH,00H                             ; CH yazmacına 00H kaydet (Disket 1. Pist)
```

```
MOV SI, GECICI              ; SI yazmacına GECICI değişkeninin adresini kaydet
MOV [SI],CX                 ; GECICI değişkenine CX yazmacının değerini yaz
MOV SI,DOSYA2               ; DOSYA2 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ2               ; Genel menü dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz
; Genel Menu yazılımını diskete yaz
MOV CL,02H                  ; CL yazmacına 02H kaydet (Disket 2. sektör)
MOV CH,00H                  ; CH yazmacına 00H kaydet (Disket 1. Pist)
MOV SI, GECICI              ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                 ; GECICI degiskenine CX yazmacının degerini yaz
MOV SI,DOSYA3               ; DOSYA3 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ3               ; Genel menu dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz
; Hesap makinası yazılımını diskete yaz
MOV CL,04H                  ; CL yazmacına 04H kaydet (Disket 4. sektör)
MOV CH,00H                  ; CH yazmacına 00H kaydet (Disket 1. Pist)
MOV SI, GECICI              ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                 ; GECICI degiskenine CX yazmacının degerini yaz
MOV SI,DOSYA4               ; DOSYA4 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ4               ; Genel menu dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz
                            ; Metin duzenleyici yazilimini diskete yaz
MOV CL,05H                  ; CL yazmacına 05H kaydet (Disket 5. sektör)
MOV CH,00H                  ; CH yazmacına 00H kaydet (Disket 1. Pist)
MOV SI, GECICI              ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                 ; GECICI degiskenine CX yazmacının degerini yaz
MOV SI,DOSYA5               ; DOSYA5 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ5               ; Genel menu dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz
                            ; Resim gösterici yazılımını diskete yaz
MOV CL,06H                  ; CL yazmacına 06H kaydet (Disket 6. sektör)
MOV CH,00H                  ; CH yazmacına 00H kaydet (Disket 1. Pist)
MOV SI, GECICI              ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                 ; GECICI degiskenine CX yazmacının degerini yaz
MOV SI,DOSYA6               ; DOSYA6 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ6               ; Genel menu dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz
                            ; Seri iletisim alıcı yazılımını diskete yaz
MOV CL,07H                  ; CL yazmacına 06H kaydet (Disket 6. sektör)
MOV CH,00H                  ; CH yazmacına 00H kaydet (Disket 1. Pist)
MOV SI,GECICI               ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                 ; GECICI degiskenine CX yazmacının degerini yaz
MOV SI,DOSYA7               ; DOSYA6 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ7               ; Genel menu dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz
                            ; Seri iletişim verici yazılımını diskete yaz
MOV CL,08H                  ; CL yazmacına 06H kaydet (Disket 6. sektör)
MOV CH,00H                  ; CH yazmacına 00H kaydet (Disket 1. Pist)
MOV SI,GECICI               ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                 ; GECICI degiskenine CX yazmacının degerini yaz
MOV SI,DOSYA8               ; DOSYA6 değişkeninin adresini SI yazmacına kaydet
CALL DOSYA_AC               ; Dosyayı aç
CALL DISKETE_YAZ            ; Dosyayı diskete yaz
MOV SI,MESAJ8               ; Genel menu dosyasının yazıldığını belirt
CALL SATIR_YAZ              ; Satırı yaz

JMP SON                     ; SON etiketine atla

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; DISKETE_YAZ                ;
; Dosyayı diskete yaz        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DISKETE_YAZ:
DONGU2: MOV AH,3FH          ; AH yazmacına 3FH yükleyerek 3FH alt yordamını ayarla
```

```
MOV DX,TAMPON                       ; Tampon adresini ayarla
MOV CX,0200H                        ; 512 bayt oku
MOV SI,DOSYAIMLECI                  ; SI yazmacına DOSYAIMLECI'nin adresini kaydet
MOV BX,[SI]                         ; Dosya imlecini ayarla
INT 21H                             ; INT 21H çağır
JC HATA3                            ; Hata var. HATA3 etiketine atla
CMP AX,CX                           ; AX ile CX yazmaçlarını karşılaştır
JNE DOSYA_SONU2                     ; Dosya sonuna gelindiyse DOSYA_SONU etiketine atla
                            ; Okunan veriyi diskete yaz
MOV SI,GECICI                       ; SI yazmacına GECICI degiskeninin adresini yaz
MOV CX,[SI]                         ; GECICI degiskeninin icerigini CX yazmacına geri yukle
MOV AH,03H                          ; AH yazmacına 03H yazarak 03H alt yordamını belirt
MOV AL,01H                          ; Yazılacak sektör sayısı (512bayt = 1 sektör)
MOV DH,00H                          ; Yazılacak yüz (00H = 1. yüz)
MOV DL,00H                          ; Sürücü numarası ( 00H = diskete yaz)
MOV BX,TAMPON                       ; BX yazmacına tampon adresini yaz
INT 13H                             ; INT 13H çağır
JC HATA4                            ; Hata var. HATA4 etiketine atla.
INC CL                              ; CL yazmacının değerini artır (sektör numarasını)
CMP CL,13H                          ; 18 sektör tamamlanmış mı?
JNE DEVAM2                          ; Hayır tamamlanmamış artırmaya devam et.
INC CH                              ; 18 sektör tamamlandı. 1 pist artır.
MOV CL,01H                          ; Sektör numarasını ayarla.
DEVAM2 :MOV SI,GECICI               ; SI yazmacına GECICI degiskeninin adresini kaydet
MOV [SI],CX                         ; GECICI degiskenine CX yazmacının degerini kaydet
JMP DONGU2                          ; Sonraki baytı oku
DOSYA_SONU2: CALL DOSYA_KAPAT       ; Açık dosyayı kapat
RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; DOSYA_KAPAT                   ;
; Açık dosyayı kapatmaya yarar  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DOSYA_KAPAT: MOV AH,3EH     ; AH yazmacına 3EH yükleyerek 3EH alt yordamını ayarla
MOV SI,DOSYAIMLECI                  ; DOSYAIMLECI'nin adresini SI yazmacına kaydet
MOV BX,[SI]                         ; BX yazmacına DOSYAIMLECI'ni kaydet
INT 21H                     ; INT 21H çağır
JC HATA2                    ; Hata var. HATA2 etiketine atla
RET                                 ; Geri dön.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; DOSYA_AC                      ;
; Dosya açar                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DOSYA_AC: MOV AH,3DH                ; AH yazmacına 3DH yükleyerek 3DH alt yordamını ayarla
MOV AL,00H                          ; Dosya açma argümanı olarak 00H kullan
MOV DX,SI                           ; DX yazmacına DOSYA'nın adresini kaydet
INT 21H                             ; INT 21H çağır
JC HATA1                            ; Hata var. HATA1 etiketine atla
MOV SI,DOSYAIMLECI                  ; SI yazmacına DOSYAIMLECI'ni kaydet
MOV [SI],AX                         ; AX yazmacındaki veriyi DOSYAIMLECI'ne kaydet
RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; HATA                          ;
; Hata oluştuğunda atlanır      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
HATA1: MOV AL,31H                   ; Açma hatası
CALL EKRANA_YAZDIR                  ; EKRANA_YAZDIR yordamını çağır
JMP SON                     ; Yazılımı sonlandır

HATA2: MOV AL,32H                   ; Kapatma hatası
CALL EKRANA_YAZDIR                  ; EKRANA_YAZDIR yordamını çağır
JMP SON                     ; Yazılımı sonlandır

HATA3: MOV AL,33H                   ; Okuma hatası
CALL EKRANA_YAZDIR                  ; EKRANA_YAZDIR yordamını çağır
JMP SON                     ; Yazılımı sonlandır

HATA4: MOV AL,34H                   ; Disket yazma hatası
CALL EKRANA_YAZDIR                  ; EKRANA_YAZDIR yordamını çağır
JMP SON                     ; Yazılımı sonlandır
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
233
234 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
235 ; EKRANA_YAZDIR                ;
236 ; Ekrana bir bayt yazdırır       ;
237 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
238 EKRANA_YAZDIR:                          ; Ekrana AL yazmacındaki veriyi yazar
239 PUSH AX                         ; AX yazmacının içeriğini yığıta at
240 MOV AH,0EH                              ; 0EH alt yordamı çağırılacağından AH yazmacına 0EH yaz
241 INT 10H                         ; INT 10H kesmesini çağır
242 POP AX                                  ; AX yazmacının içeriğini yığıttan geri al
243 RET                                     ; Yordamdan çık
244
245 SON: MOV AX,4C00H                       ; Yazılımı sonlandırmak için AX yazmacına 4C00H yükle
246 INT 21H                         ; INT 21H çağır
247
248 INCLUDE 'yordamlar.inc'
```

Koda dikkat edilecek olunursa (21. ve 28. satırlar) yazılımlarımızın dosya adlarının aşağıdaki tablo ile uyumlu olması gerekmektedir:

Dosya Adı	Dosya Tanım
p0.bin	Önyükleme yazılımı
p1.bin	Genel menü yazılımı
p2.bin	Hesap makinası yazılımı
p3.bin	Metin düzenleyici yazılımı
p4.bin	Resim gösterici yazılımı
p5.bin	Seri iletişim verici yazılımı
p6.bin	Seri iletişim alıcı yazılımı
resim.bin	Resim verisi

Ayrıca dosyaların C:'de bulunması gerekmektedir. Eğer farklı dosya adları kullanılmak istenirse ve dosyalar farklı bir adresten yüklenmek istenilirse o zaman 21. ve 28. satırlar arasındaki kodlarda değişiklik yapılması gerekmektedir.

Gerekli açıklamalar yorum şeklinde kodun içinde yapıldığından burada kod hakkında fazladan bir açıklama yapmaya gerek yoktur. .com dosyamız çalıştırılmadan önce disket sürücüye bir disket yerleştirilmelidir.

9.2.2. Önyükleme Yazılımı

Önyükleme yazılımımızın disketin ilk sektöründe bulunması gerekir. Bilgisayar açıldığında disket eğer disket sürücüye takılı ise ve disketten önyükleme yapmak için gerekli olan BIOS ayarları yapılı ise disketten ön yükleme yapılır. Disketten ön yükleme yapılırken disketin ilk sektörü okunur ve disketin ilk sektörü 07C0H:0000H adresine kaydedilir.

Önyükleme programımız sırasıyla aşağıdaki işlemleri yapacaktır:

1. Vidyo kipini ayarla.
2. Bir açılış mesajı göster.
3. Bir süre bekle.
4. Disketin birinci sektörünü oku ve 1000H:0000H adresine kaydet.
5. 1000H:0000H adresine atla

Kodumuzu yazmaya başlayalım.

Öncelikle programımız belleğe yüklendiği zaman 0000H göreli adresine yazılacağından emin olmamız gerekmektedir:

```
ORG 0000H
```

Ardından ekrana yazdıracağımız açılış mesajlarını tanımlayalım. Tabii ki bu açılış mesajlarının birer komut gibi işlenmemesi için mesajların üzerinden atlamamız gerekiyor. Bunun için mesajlarımızdan sonra bir *BASLANGIC* etiketi tanımlayalım ve oraya atlayalım (Atlama işlemi için sadece JMP yerine JMP FAR kullanmamızın nedeni atlanacak olan mesafenin +/- 127 bayttan daha fazla olmasıdır) ve *BASLANGIC* etiketinden sonra ilk iş olarak CS kesim yazmacının değerini DS kesim yazmacına aktararak muhtemel adres karışıklıklarını ortadan kaldıralım:

```
 5  JMP FAR 07C0H:BASLANGIC
 6
 7  ;DEGISKEN TANIMLAMALARI
 8  M_ACILIS0 DB "************************************",0AH,0DH,00H
 9  M_ACILIS1 DB "* BASIT ISLETIM SISTEMI - 1. ASAMA *",0AH,0DH,00H
10  M_ACILIS2 DB "*                                  *",0AH,0DH,00H
11  M_ACILIS3 DB "* HOSGELDINIZ!                     *",0AH,0DH,00H
12
13  BASLANGIC: PUSH CS
14  POP DS
```

Şimdi vidyo kipini ayarlamamız gerekiyor. BIOS'un açılışta ayarladığı vidyo kipi bize aslında bu aşamada yeterli ama ileriki aşamalarda bu vidyo kipini değiştireceğimizden şimdiden vidyo kipini değiştiren bir kod yazalım:

```
15  ;VIDYO KIPINI DEGISTIR
16  MOV AH,00H       ;AH = 00H => VIDYO KIPI SEC
17  MOV AL,03H       ;AL = 03H => 16 RENK 80 x 25 YAZI VIDYO KIPI
18  INT 10H          ;INT 10H KESMESİNE ATLA
```

Görüldüğü gibi yukarıdaki kod ile INT 10H kesmesinin 00H yordamı çağırıldı ve parametre olarak AL yazmacında 03H değeri verildi. Yani 16 renk 80 x 25 yazı vidyo kipine geçmek için gerekli işlemler yapıldı.

Vidyo kipini de istediğimiz vidyo kipine ayarladıktan sonra açılış mesajlarını ekrana yazdırabiliriz.

```
20
21  ;AÇILIŞ MESAJLARINI YAZ
22  MOV SI,M_ACILIS0
23  CALL SATIR_YAZ
24  MOV SI,M_ACILIS1
25  CALL SATIR_YAZ
26  MOV SI,M_ACILIS2
27  CALL SATIR_YAZ
28  MOV SI,M_ACILIS2
29  CALL SATIR_YAZ
30  MOV SI,M_ACILIS3
31  CALL SATIR_YAZ
32  MOV SI,M_ACILIS0
33  CALL SATIR_YAZ
```

Yukarıdaki kodda *SATIR_YAZ* yordamı çağırılmadan önce SI yazmacına açılış mesajlarının adreslerinin kaydedildiğini görüyoruz. Peki neden?

Bu soruyu cevaplamak için SATIR_YAZ yordamının ne olduğunu bilmemiz gerekiyor. Kodumuzun 58. satırında INCLUDE 'yordamlar.inc' ibaresini görebilirsiniz. Bunun anlamı koda 'yordamlar.inc' adlı bir dosyanın gerektiğinde eklenmesidir. Peki bu yordamlar.inc dosyasında neler var? Şimdilik sadece şu iki yordam var:

```
1  ;**************************************
```

```
 2  ;SATIR_YAZ
 3  ;[SI] adresindeki dizgiyi 00H
 4  ;değerine ulaşana kadar ekrana yazdırır
 5  ;**************************************
 6  SATIR_YAZ:
 7  MESAJ_YAZ_HARF_AL: MOV AL,[SI]
 8  INC SI
 9  OR AL,AL
10  JZ MESAJ_YAZ_SON
11  MOV BX,00H
12  MOV AH,0EH
13  INT 10H
14  JMP MESAJ_YAZ_HARF_AL
15  MESAJ_YAZ_SON: RET
16  ;**************************************
17
18  ;**************************************
19  ;GECIKME
20  ;Belirli bir gecikme faktörünü CX
21  ;yazmacının degeri kadar tekrarlar
22  ;**************************************
23  GECIKME:
24  GECIKME_LP1: PUSH CX
25  MOV CX,0FFFFH
26  GECIKME_LP2: LOOP GECIKME_LP2
27  POP CX
28  LOOP GECIKME_LP1
29  GECIKME_SON: RET
```

'yordamlar.inc' dosyasında tanımladığımız ilk yordam *SATIR_YAZ* ve ikinci yordam *GECIKME*'dir.

SATIR_YAZ yordamına bakacak olursak:

7. satırda AL yazmacına SI yazmacının değerinin işaretlediği adresteki verinin kaydedildiği görülür. SATIR_YAZ yordamını çağırmadan önce SI yazmacına yazılacak olan mesajın adresinin kaydedildiğini hatırlayın. Yani AL yazmacına bu 7. satırdaki işlem ile yazdırılacak olan satırın ilk harfi kaydedildi.

8. satırda SI yazmacının değeri 1 artırıldı çünkü döngünün bir sonraki dönüşünde doğal olarak bir sonraki karakteri okumak istiyoruz.

9. satırda AL yazmacını AL yazmacı le VEYA işlemine tabi tuttuk. Aslında bunun yerine CMP AL,00H da koyabilirdik. Tabii ki baktığınızda CMP AL,00H ile OR AL,AL kodlarının tamamen farklı işlemler yaptığını söyleyeceksiniz. Evet aslında çok farklı işlemler yapıyorlar ama bizim burada yapmaya çalıştığımız şey AL yazmacının değerinin 00H'a eşit olup olmadığını bulmak. Bu amaç için her iki komut da kullanılabilir. Peki neden 00H'a eşit olup olmadığını bulmak istiyoruz? Eğer değişkenlerin tanımlamalarına bakarsanız dizgiden sonra 0AH 0DH ve 00H değerlerini göreceksiniz. 0AH ASCII karakter sisteminde yeni satır anlamına gelmektedir. Aynı şkeilde 0DH da ASCII karakter sisteminde "Enter" tuşunun karşılığıdır. Yani biz ekrana bu iki karakteri yazdırdığımızda (ki aslında karakterden çok kontrol değerleridir) bir sonraki satıra atlanacak. 00H ise tamamen bizim seçtiğimiz bir satır sonlandırıcı değer. Kullanılmayan herhangi bir değer de seçilebilirdi. Yani kısaca 00H değerine gelindiğinde yazdırma işlemi durdurulacaktır.

10. satıra bakacak olursak zaten yukarıda da dediğimiz gibi AL yazmacı 00H a eşit olduğu tespit edilince 15. satırdaki *MESAJ_YAZ_SON* etiketine atlandığını ve bu etiketin bulunduğu yerde ise RET komutunun olduğunu göreceğiz. Eğer AL yazmacı 00H a eşit değilse program 11. satıra geçer.

11. satırda BX yazmacına 00H ve 12. satırda AH yazmacına 0EH değerinin atandığını göreceğiz. Ardından 13. satırda ise INT 10H kesmesine atlanıyor. Yani INT 10H kesmesini 0EH yordamına geçiliyor. Bu yordam parametre olarak AL yazmacını ve BX yazmacını alır. AL yazmacında yazılacak olan karakter bulunmalıdır. BX de ise daha önce 7. bölümde de değindiğimiz gibi sayfa numarasının bulunması gerekir.

14. satırdaki JMP komutu ile 7. satıra geri atlanır ve bir sonraki karakter AL yazmacına kaydedilir ve döngü bu şekilde devam eder.

Şimdi kaldığımız yerden önyükleme programımızın koduna geri dönelim:

```
34  ;BİR SÜRE BEKLE
35  MOV CX,0FFFFH
36  CALL GECIKME
37  MOV CX,0FFFFH
38  CALL GECIKME
39  MOV CX,0FFFFH
40  CALL GECIKME
```

Yukarıda görüldüğü gibi CX yazmacına 0FFFFH değeri yüklendikten sonra *GECIKME* yordamının çağırıldığını görüyoruz.

'yordamlar.inc' dosyamıza bir daha geri dönersek 23. satırda *GECIKME* yordamımızı bulacağız. Yordamımızı incelersek *GECIKME_LP2* döngüsünün 0FFFFH kes kendi içinde tekrar edildiğini göreceğiz. CX yazmacına yordamı çağırmadan önce girdiğimiz değer ise bu *GECIKME_LP2* döngüsüne kaç kez girileceğini gösteren bir değerdir. Yani *GECIKME_LP2* döngüsüne 0FFFFH kes girilir ve *GECIKME_LP2* döngüsü her seferinde 0FFFFH kes döner. Her döngüde komutlar işlendiğinden belirli bir zaman kaybı olmaktadır. Bu zaman kaybını yeteri kadar tekrarlayınca ise bir tür gecikme oluşturmaktayız.

Kaldığımız yerden önyükleme programımıza bir daha geri dönersek:

```
42  ;DISKETTE SEKTÖR 1'İ 1000H:0000H ADRESİNE YÜKLE
43  MOV AH,02H          ;INT 13H KESMESI 02H YORDAMI
44  MOV AL,02H          ;OKUNACAK SEKTÖR SAYISI
45  MOV CH,00H          ;PIST NUMARASI 00H = 0. PİST
46  MOV CL,02H          ;SEKTÖR NUMARASI 02H = 1. SEKTÖR
47  MOV DH,00H          ;OKUNACAK YUZ
48  MOV DL,00H          ;00H = DISKETTEN OKU (08H = 1. SABIT DISKTEN OKU)
49  MOV BX,1000H        ;BX YAZMACINA 1000H DEĞERİNİ YAZ
50  MOV ES,BX           ;ES YAZMACINA BX YAZMACININ DEĞERİNİ YAZ
51  MOV BX,0000H        ;BX YAZMACINA 0000H YAZ
52  INT 13H             ;INT 13H KESMESİNE ATLA
```

Yukarıdaki kod sayesinde disketten belirli sektörleri okuyabiliriz. INT 13H için verilen parametreleri incelersek:

AH = 02H ile 02H yordamına geçileceği,
AL = 02H iki sektörün okunacağı,
CH = 00H ile okunacak sektörün sıfırıncı pistte bulunduğunu,
CL = 02H ile okunacak sektörün birinci sektör olduğu,
DH = 00 ile ilk yüzün okunacağı ve
ES:BX = 1000H:0000H ile okunacak olan verinin bu adrese kaydedileceği anlaşılır.

Bir sektör okumak yerine bir seferde iki sektör okumamızın nedeni yazacağımız genel menü yazılımının göreceli olarak uzun olması ve iki sektöre anca sığmasındandır.

Yukarıdaki kod ile birinci sektörü okuyup 1000H:0000H adresine kaydettik. Şimdi tek yapmamız gereken bu adrese atlamak:

```
55  ;GENEL MENÜ PROGRAMINA ATLA
56  JMP 1000H:0000H
57
58  INCLUDE 'yordamlar.inc'
```

56. satırdaki JMP 1000H:0000H komutu sayesinde genel menü programımıza atlıyoruz.

58. satırdaki INCLUDE ‘yordamlar.inc’ ile çeviriciye bu dosyanın çeviri yapılırken gerektiğinde kullanılacağını belirttik.

Aşağıda önyükleme programımızın tamamını toplu halde bulabilirsiniz:

```
 1  ORG 0000H
 2  JMP FAR 07C0H:BASLANGIC       ; 07C0:BASLANGIC'a atla
 3
 4  ;DEGISKEN TANIMLAMALARI
 5  M_ACILIS0 DB "************************************",0AH,0DH,00H
 6  M_ACILIS1 DB "* BASIT ISLETIM SISTEMI          *",0AH,0DH,00H
 7  M_ACILIS2 DB "*                               *",0AH,0DH,00H
 8  M_ACILIS3 DB "* HOSGELDINIZ!                  *",0AH,0DH,00H
 9
10  BASLANGIC: PUSH CS            ; CS yazmacını yığına at
11  POP DS                        ; DS yazmacını yığından doldur
12
13  ; Vidyo kipini degistir
14  MOV AH,00H                    ; AH = 00H => Vidyo kipi sec
15  MOV AL,03H                    ; AL = 03H => 16 renk 80 x 25 yazı vidyo kipi
16  INT 10H                       ; INT 10H kesmesine atla
17
18  ; Açılış mesajlarını yaz
19  MOV SI,M_ACILIS0              ; SI yazmacına mesajın adresini yaz
20  CALL SATIR_YAZ                        ; SATIR_YAZ yordamını çağır
21  MOV SI,M_ACILIS1              ; SI yazmacına mesajın adresini yaz
22  CALL SATIR_YAZ                        ; SATIR_YAZ yordamını çağır
23  MOV SI,M_ACILIS2              ; SI yazmacına mesajın adresini yaz
24  CALL SATIR_YAZ                        ; SATIR_YAZ yordamını çağır
25  MOV SI,M_ACILIS2              ; SI yazmacına mesajın adresini yaz
26  CALL SATIR_YAZ                        ; SATIR_YAZ yordamını çağır
27  MOV SI,M_ACILIS3              ; SI yazmacına mesajın adresini yaz
28  CALL SATIR_YAZ                        ; SATIR_YAZ yordamını çağır
29  MOV SI,M_ACILIS0              ; SI yazmacına mesajın adresini yaz
30  CALL SATIR_YAZ                        ; SATIR_YAZ yordamını çağır
31
32  ; Bir süre bekle
33  MOV CX,0FFFFH                 ; CX yazmacına 0FFFFH yazdır
34  CALL GECIKME                  ; GECIKME yordamını çağır
35  MOV CX,0FFFFH                 ; CX yazmacına 0FFFFH yazdır
36  CALL GECIKME                  ; GECIKME yordamını çağır
37  MOV CX,0FFFFH                 ; CX yazmacına 0FFFFH yazdır
38  CALL GECIKME                  ; GECIKME yordamını çağır
39
40  ; Disket sektör 1'i 1000H:0000H adresine yükle
41  MOV AH,02H                    ; INT 13H kesmesi 02H alt yordamı
42  MOV AL,02H                    ; Okunacak sektör sayısı
43  MOV CH,00H                    ; Pist numarası 00H = 0. pist
44  MOV CL,02H                    ; Sektör numarası 02H = 1. sektör
45  MOV DH,00H                    ; Okunacak yüz
46  MOV DL,00H                    ; 00H = disketten oku
47  MOV BX,1000H                  ; BX yazmacına 1000H değerini yaz
48  MOV ES,BX                     ; ES yazmacına BX yazmacının değerini yaz
49  MOV BX,0000H                  ; BX yazmacına 0000H yaz
50  INT 13H                       ; INT 13H çağır
51
52  ; Genel menü programına atla
53  JMP 1000H:0000H               ; 1000:0000'a atla
```

```
54  DB 247 DUP (00H)
55
56  INCLUDE 'yordamlar.inc'
```

Tabii ki önyükleme programımız sadece bu şekilde ilk sektöre yüklenirse bazı sistemlerde önyükleme programı olarak algılanamayabilir çünkü bazı BIOS sürümleri önyükleme sektörünün son 2 baytının AA55H değerine sahip olmasını şart koşmaktadır. Günümüzdeki sistemlerin çoğu AA55H verisine gereksinim duymamaktadır. Eğer eski bir sistem ile çalışıyorsanız AA55H verisini disketin ilk yüzünün, ilk sektörünün son iki baytına yazdırmayı unutmayınız.

54. satırda görülen "DB 247 DUP (00H)" kodu dosyanın boyutunu tam olarak 512 bayta tamamlamakta kullanılmaktadır. Aksi halde disket yükleyici yazılımımız yazılımımızı düzgün bir şekilde diskete yazamayacaktır.

9.2.3. Genel Menü Yazılımı

Genel menü yazılımımız disketin ikinci ve üçüncü sektöründe bulunmaktadır. Önyükleme yazılımı tarafından 1000H:0000H bellek göreli adresine kaydedilmekte ve kontrol bu yazılıma geçirilmektedir.

Genel menü yazılımımızın görevi kullanıcıya bazı seçenekler sunmaktır. Bu seçenekler yazacağımız çeşitli yazılımları disketten yüklemek ve çalıştırmak, menüyü yeniden yazdırmak, sistemi yeniden başlatmak ve sistemi sonlandırmaktır.

Eğer bir yazılım açılmak istenirse yazılım disketin ilgili sektöründen okunacak ve belleğin 1000:0420 adresine kaydedilecektir. Yazılım belleğe yüklendikten sonra 1000:0420 adresine atlanılacaktır.

Hemen kodumuza geçelim:

```
1   ORG 0000H
2   PUSH CS                     ; CS yazmacını yığına at
3   POP DS                      ; DS yazmacını yığından doldur
4   JMP BASLANGIC               ; BASLANGIC etiketine atla
5   M_MENU0 DB "****************************************",0AH,0DH,00H
6   M_MENU1 DB "*- MENU---------------------------------*",0AH,0DH,00H
7   M_MENU2 DB "*1 - MENU'YU YENIDEN YAZ              *",0AH,0DH,00H
8   M_MENU3 DB "*2 - HESAP MAKINASI YAZILIMI          *",0AH,0DH,00H
9   M_MENU4 DB "*3 - METIN DUZENLEYICI YAZILIMI       *",0AH,0DH,00H
10  M_MENU5 DB "*4 - EKRANA RESIM CIZDIRME YAZILIMI   *",0AH,0DH,00H
11  M_MENU6 DB "*5 - SERI ILETISIM - ALICI YAZILIMI   *",0AH,0DH,00H
12  M_MENU7 DB "*6 - SERI ILETISIM - VERICI YAZILIMI  *",0AH,0DH,00H
13  M_MENU8 DB "*7 - BILGISAYARI YENIDEN BASLAT       *",0AH,0DH,00H
14  M_MENU9 DB "*8 - BILGISAYARI KAPAT                *",0AH,0DH,00H
15  M_MENU10 DB "*---------------------------------------*",0AH,0DH,00H
16  M_MESAJ1 DB "*Bilgisayar Kapatiliyor...*",0AH,0DH,00H
17  M_MESAJ2 DB "*Bilgisayar Yeniden Baslatiliyor...*",0AH,0DH,00H
18  BASLANGIC:
19
20  ; Vidyo kipini sec
21  MOV AH,00H                  ; AH = 00H => Vidyo kipi sec
22  MOV AL,03H                  ; AL = 03H => 16 renk 80 x 25 yazı vidyo kipi
23  INT 10H                     ; INT 10H kesmesine atla
24
25  ; Menüyü ekrana yazdır
26  MOV SI,M_MENU0
27  CALL SATIR_YAZ
28  MOV SI,M_MENU1
29  CALL SATIR_YAZ
30  MOV SI,M_MENU2
31  CALL SATIR_YAZ
32  MOV SI,M_MENU3
33  CALL SATIR_YAZ
34  MOV SI,M_MENU4
```

```
35  CALL SATIR_YAZ
36  MOV SI,M_MENU5
37  CALL SATIR_YAZ
38  MOV SI,M_MENU6
39  CALL SATIR_YAZ
40  MOV SI,M_MENU7
41  CALL SATIR_YAZ
42  MOV SI,M_MENU8
43  CALL SATIR_YAZ
44  MOV SI,M_MENU9
45  CALL SATIR_YAZ
46  MOV SI,M_MENU10
47  CALL SATIR_YAZ
48  MOV SI,M_MENU0
49  CALL SATIR_YAZ
50
51  ; Klavyeden bir sayı girilene kadar bekle
52  KLAVYE_OKU: MOV AH,01H
53  INT 16H
54  JZ KLAVYE_OKU
55  MOV AH,00H
56  INT 16H
57  CMP AL,31H
58  JE SECENEK_1
59  CMP AL,32H
60  JE SECENEK_2
61  CMP AL,33H
62  JE SECENEK_3
63  CMP AL,34H
64  JE SECENEK_4
65  CMP AL,35H
66  JE SECENEK_5
67  CMP AL,36H
68  JE SECENEK_6
69  CMP AL,37H
70  JE SECENEK_7
71  CMP AL,38H
72  JE SECENEK_8
73  JMP KLAVYE_OKU
74
75  SECENEK_1: JMP BASLANGIC
76  SECENEK_2: MOV CL,04H
77  JMP YUKLE
78  SECENEK_3: MOV CL,05H
79  JMP YUKLE
80  SECENEK_4: MOV CL,06H
81  JMP YUKLE
82  SECENEK_5: MOV CL,07H
83  JMP YUKLE
84  SECENEK_6: MOV CL,08H
85  JMP YUKLE
86  SECENEK_7: JMP YENIDEN_BASLAT
87  SECENEK_8: JMP KAPAT
88
89  YENIDEN_BASLAT:
90  MOV SI,M_MESAJ2          ; SI yazmacına M_MESAJ2 mesajının adresini yaz
91  CALL SATIR_YAZ    ; SATIR_YAZ yordamını çağır
92
93  ; Soğuk başlatma ayarlarını yap
94  MOV AX,0040H          ; AX yazmacına 0040H yaz
```

```
95  MOV ES,AX                   ; AX yazmacının içeriğini ES yazmacına aktar
96  MOV BX,0072H            ; ES:BX adresi => 0040:0072
97  MOV WORD [ES:BX],1234H      ; gerekli değeri ES:BX adresine yaz
98  JMP 0FFFFH:0FFF0H           ; bilgisayarı yeniden başlat
99
100 ; INT 15H kesmesini kullanarak sistemi sonlandır
101 KAPAT: MOV SI,M_MESAJ1      ; SI yazmacına M_MESAJ1 mesajının adresini yaz
102 CALL SATIR_YAZ              ; SATIR_YAZ yordamını çağır
103 MOV AX,5300H
104 MOV BX,0000H
105 INT 15H
106 PUSH AX
107
108 MOV AX,5308H
109 MOV BX,1
110 MOV CX,1
111 INT 15H
112
113 MOV AX,5308H
114 MOV BX,0FFFFH
115 MOV CX,1
116 INT 15H
117
118 MOV AX,5301H
119 MOV BX,0000H
120 INT 15H
121 MOV AX,530EH
122 MOV BX,0000H
123 POP CX
124
125 INT 15H
126 MOV AX,530DH
127 MOV BX,1
128 MOV CX,1
129 INT 15H
130
131 MOV AX,530FH
132 MOV BX,1
133 MOV CX,1
134 INT 15H
135
136 MOV AX,5307H
137 MOV BX,1
138 MOV CX,3
139 INT 15H
140 JMP KAPAT
141
142 ; Disketten program yükle
143 YUKLE: MOV AH,02H
144 MOV AL,01H              ; okunacak sektör sayısı
145 MOV CH,00H              ; okunacak pist numarası
146 MOV DH,00H              ; okunacak yüz
147 MOV DL,00H              ; disketten oku
148 MOV BX,1000H     ; bx yazmacına 1000H yaz
149 MOV ES,BX               ; bx yazmacının içeriğini es yazmacına aktar
150 MOV BX,0420H   ; bx yazmacına 0420H yükle
151 INT 13H                 ; int 13H cağır
152
153 SON: DB 00EAH  ; JMP FAR
154 DW 0420H, 1000H         ; 1000:0420 -> Program
```

```
155  DB 151 DUP (00H)
156
157  include 'yordamlar.inc'
```

Görüldüğü gibi genel menü yazılımımız oldukça basit. Ama 93. ve 140. satırlar arasındaki kodları incelemekte fayda var.

```
93   ; Soğuk başlatma ayarlarını yap
94   MOV AX,0040H          ; AX yazmacına 0040H yaz
95   MOV ES,AX             ; AX yazmacının içeriğini ES yazmacına aktar
96   MOV BX,0072H          ; ES:BX adresi => 0040:0072
97   MOV WORD [ES:BX],1234H    ; gerekli değeri ES:BX adresine yaz
98   JMP 0FFFFH:0FFF0H     ; bilgisayarı yeniden başlat
```

98. satırda FFFF:FFF0 adresine atlanıyor. Bu adrese atlamadan önce ise 0040:0072 adresine 1234H kelimesi yükleniyor. 1234H kelimesinin bu adrese yüklenmesi ve ardından da FFFF:FFF0 adresine atlanılması sıcak başlangıç yapmak için gereklidir. 1234H verisi yerine başka herhangi bir veri girilirse soğuk başlangıç yapılacaktır.

100. ve 140. satırlar arasındaki kodlar sayesinde ise bilgisayarımızı kapatabiliriz. Bu yöntem her bilgisayar için geçerli olmayacaktır ve beklide işletim sisteminizi deneyeceğiniz bilgisayarda çalışmayacaktır. Bunun nedeni bir bilgisayarı kapatmak için yapılması gereken işlemlerin bilgisayardan bilgisayara farklılık göstermesidir.

```
100  ; INT 15H kesmesini kullanarak sistemi sonlandır
101  KAPAT: MOV SI,M_MESAJ1     ; SI yazmacına M_MESAJ1 mesajının adresini yaz
102  CALL SATIR_YAZ             ; SATIR_YAZ yordamını çağır
103  MOV AX,5300H
104  MOV BX,0000H
105  INT 15H
106  PUSH AX
107
108  MOV AX,5308H
109  MOV BX,1
110  MOV CX,1
111  INT 15H
112
113  MOV AX,5308H
114  MOV BX,0FFFFH
115  MOV CX,1
116  INT 15H
117
118  MOV AX,5301H
119  MOV BX,0000H
120  INT 15H
121  MOV AX,530EH
122  MOV BX,0000H
123  POP CX
124
125  INT 15H
126  MOV AX,530DH
127  MOV BX,1
128  MOV CX,1
129  INT 15H
130
131  MOV AX,530FH
132  MOV BX,1
```

```
133 MOV CX,1
134 INT 15H
135
136 MOV AX,5307H
137 MOV BX,1
138 MOV CX,3
139 INT 15H
```

103. ve 106. satırlar arasındaki koda bakacak olursak AX yazmacına 5300H ve BX yazmacına 0000H verisi yazıldıktan sonra INT 15H kesmesinin çağrıldığını görürüz. INT 15H kesmesinin 5300H alt yordamı bilgisayarda yüklü çeşitli donanımları kontrol eder ve APM sürücüsü sürümünü AX yazmacına kaydeder. 106. satırda ise AX yazmacının verisi yığına atılır.

108. ve 111. satırlar arasındaki kodda ise AX yazmacına 5308H, BX yazmacına 1 ve CX yazmacına verileri yazıldıktan sonra INT 15H kesmesinin çağrıldığını görüyoruz. BX yazmacına 1 verisinin girilmesi **APM** (**A**dvanced **P**ower **M**anagement – Gelişmiş Güç Yönetimi) 1.1 veya daha yeni bir sürümünün kullanıldığını belirtir. Eski bilgisayarlarda APM sürümüne bağlı olarak BX yazmacına FFFFH verisinin yazılması gerekebilir. Bu yüzden 113. ve 116. satırlar arasındaki koda FFFFH verisi yazılmıştır. CX yazmacına 1 verisinin girilmesi ise güç yönetim özelliklerinin kullanılacağını belirtir. CX yazmacına 0 yazılırsa güç yönetimi kullanılamaz.

118. ve 120. satırlar arasındaki kodlar sayesinde gerçek kip arayüzüne erişilir.

121. ve 125. satırlar arasındaki kodlar sayesinde ise APM bağlantısı elde edilir. Dikkat edilecek olursa 106. satırda yığına atılan APM sürücü sürümü 123. satırdaki POP CX kodu ile CX yazmacına yazılmıştır.

126. ve 129. satırlar arasındaki kodlara bakacak olursak BX yazmacına 1 ve CX yazmacına 1 yazıldıktan sonra INT 15H kesmesinin 530DH alt yordamının çağırıldığını görürüz. Bu sayede birim güç yönetimi etkinleştirilmiş olur.

131. ve 134. satırlar arasındaki kodlar ile güç yönetimi devreye sokulur.

136. ve 139. satırlar arasındaki kodlar ile bilgisayar kapatılır. Yukarıdaki işlemlerin yapılması bu son işlemin yapılması için gereklidir. Aksi takdirde sistem sonlandırılamaz. Bazı eski güç kaynakları otomatik olarak kapatılmayı desteklemediğinden bu yöntem eski bilgisayarları kapatmak için kullanılamayabilir.

9.2.4. Hesap Makinası Yazılımı

Hesap makinası yazılımımız belleğin 0420H göreceli adresine yerleştirilecektir. W", "A", "S", "D" ve "ENTER" tuşları kullanılarak ekrandan sayılar seçilebilecek ve toplanabilecektir. Diğer aritmetik işlemlerin de eklenmesi gayet basittir. Burada sadece bir örnek teşkil etmesi amacıyla toplama işlemi gösterilmiştir.

```
ORG 0420H
PUSH CS
POP DS
JMP BASLANGIC
M_SATIR0 DB "     ",0AH,0DH,00H
M_SATIR1 DB "1 2 3",0AH,0DH,00H
M_SATIR2 DB "4 5 6",0AH,0DH,00H
M_SATIR3 DB "7 8 9",0AH,0DH,00H
M_SATIR4 DB "0 + =",0AH,0DH,00H
DEGER1 DW 0000H
DEGER2 DW 0000H
BASLANGIC:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;VIDYO_KIPI_AYARLA --- Vidyo Kipini Ayarlar       ;
;                                                 ;
;Açıklama: Vidyo kipini ve çeşitli özelliklerini  ;
;ayarlar.                                         ;
```

```
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
VIDYO_KIPI_AYARLA: MOV AH,00H
MOV AL,03H
INT 10H
MOV SI,M_SATIR0
CALL SATIR_YAZ
MOV SI,M_SATIR1
CALL SATIR_YAZ
MOV SI,M_SATIR2
CALL SATIR_YAZ
MOV SI,M_SATIR3
CALL SATIR_YAZ
MOV SI,M_SATIR4
CALL SATIR_YAZ
MOV AH,01H
MOV CH,20H
INT 10H
MOV AX,0B800H
MOV ES,AX
MOV SI,00A0H
ES MOV AL, BYTE [SI+01H]
NOT AL
ES MOV BYTE [SI+01H],AL
MOV DI,DEGER1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;KLAVYE OKU --- Klavyeden Girilen ASCII Kodu Oku   ;
;                                        ;
;Açıklama: Klavyeden girilen kodu okur ve koda göre;
;bir yere atlama yapar.                   ;
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
KLAVYE_OKU: MOV AH,01H
INT 16H
JZ KLAVYE_OKU
MOV AH,00H
INT 16H
CMP AL,77H
JE YUKARI
CMP AL,61H
JE SOL
CMP AL,73H
JE ASAGI
CMP AL,64H
JE SAG
CMP AL,1BH
JE SON
CMP AL,0DH
JE ISLE
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;YUKARI --- İmleci Yukarı Kaydır          ;
;                                        ;
;Açıklama: İmleci yukarı kaydırır.        ;
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
YUKARI:
CALL KONUM_GUNCELLE
```

```
SUB SI,0A0H
CALL KONUM_GUNCELLE
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ASAGI --- İmleci Aşağı Kaydır                    ;
;                                                 ;
;Açıklama: İmleci aşağı kaydırır                  ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ASAGI:
CALL KONUM_GUNCELLE
ADD SI,0A0H
CALL KONUM_GUNCELLE
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SAG --- İmleci Sağa Kaydır                       ;
;                                                 ;
;Açıklama: İmleci Sağa Kaydırır                   ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SAG:
CALL KONUM_GUNCELLE
ADD SI,02H
CALL KONUM_GUNCELLE
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SOL --- İmleci Sola Kaydır                       ;
;                                                 ;
;Açıklama: İmleci Sola Kaydırır                   ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SOL:
CALL KONUM_GUNCELLE
SUB SI,02H
CALL KONUM_GUNCELLE
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;KONUM_GUNCELLE --- İmlec Konumu Güncelle         ;
;                                                 ;
;Açıklama: İmlecin konumunu günceller.            ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
KONUM_GUNCELLE:
ES MOV AL, BYTE [SI+01H]
NOT AL
ES MOV BYTE [SI+01H],AL
RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ISLE --- Degiskenleri Ayarla                     ;
;                                                 ;
;Açıklama: Kullanımda olan değişkeni ayarlar ve   ;
;seçilen sayının ekrana yazdırılmasını sağlar     ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ISLE: ES MOV AL, BYTE [SI]
```

```
CMP AL,2BH
JE TOPLA
CMP AL,3DH
JE ESIT
MOV AH,0FH
ES MOV WORD [0006H], 0F00H
ES MOV WORD [0008H], AX
AND AL,0FH
MOV [DI],AL
JMP KLAVYE_OKU
TOPLA: MOV DI,DEGER2
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ESIT --- Sayıları Topla                    ;
;                                          ;
;Açıklama: Sayıları toplar ve ekrana yazdırır    ;
;                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ESIT: MOV DI,DEGER1
MOV AL,[DI]
MOV DI,DEGER2
MOV AH,[DI]
ADD AL,AH
CMP AL,0AH
JB D1
ES MOV WORD [0006H],0F31H
SUB AL,0AH
D1: ADD AL,30H
MOV AH,0FH
ES MOV WORD [0008H],AX
MOV WORD [DI],0000H
MOV DI,DEGER1
MOV WORD [DI],0000H
JMP KLAVYE_OKU

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SON --- Programı Sonlandır                 ;
;                                          ;
;Açıklama: Ana menüye geri dönülür           ;
;                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SON: DB 00EAH          ; JMP FAR
DW 0000H, 1000H        ; 1000:0000 -> ANA MENU
DB 191 DUP (00H)
include 'yordamlar.inc'
```

9.2.5. Metin Düzenleyici Yazılımı

Metin düzenleyici yazılımımız ekrana bir kutucuk çizecektir ve bu kutucuk içine klavyeden karakterler girilebilecektir. Metin düzenleyici yazılımımız da hesap makinası yazılımımız gibi belleğin 0420H göreli adresine kaydedilecek ve oradan çalıştırılacaktır.

```
ORG 0420H
PUSH CS
POP DS
JMP BASLANGIC

BASLANGIC:

```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;VIDYO_KIPI_AYARLA --- Vidyo Kipini Ayarlar        ;
;                                                 ;
;Açıklama: Vidyo kipini ve çişitli özelliklerini   ;
;ayarlar.                                         ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
VIDYO_KIPI_AYARLA: MOV AH,00H
MOV AL,03H
INT 10H
MOV AX,0B800H
MOV ES,AX
MOV SI,0000H
MOV DX,0000H
MOV AH,01H
MOV CH,20H
INT 10H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;KUTUCUK_OLUSTUR --- Bir Kutucuk Oluşturur         ;
;                                                 ;
;Açıklama: İçine yazı yazılacak olan kutucuğu oluş-;
;turur.                                           ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
KUTUCUK_OLUSTUR: MOV AL,0F0H
MOV CX,33H
MOV BX,65H
CIZIM1: ES MOV BYTE [SI+BX],AL
DEC BX
DEC BX
LOOP CIZIM1
MOV CX,11H
MOV SI,0000H
CIZIM2: ADD SI,0A0H
ES MOV BYTE [SI+01H],AL
ES MOV BYTE [SI+65H],AL
LOOP CIZIM2
MOV CX,33H
MOV BX,65H
CIZIM3: ES MOV BYTE [SI+BX],AL
DEC BX
DEC BX
LOOP CIZIM3
MOV SI,00A2H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;KLAVYE_OKU1 --- Klavye ASCII Kodunu Oku           ;
;                                                 ;
;Açıklama: Klavyeden girilen tuşların ASCII kodla- ;
;rını AL yazmacına kaydeder.                       ;
;                                                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
KO1: CALL IMLEC_GUNCELLE
MOV AH,01H
INT 16H
JZ KO1
MOV AH,00H
INT 16H
KLAVYE_OKU1: MOV AH,01H
```

```
INT 16H
JZ KLAVYE_OKU1
MOV AH,00H
INT 16H
CMP AL,1BH
JE KO2
CMP AL,0DH
JE ALT_SATIR
CMP AL,08H
JE GERI_DON
CMP AL,00H
JE KLAVYE_OKU1
JMP YAZI_YAZ

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;KLAVYE_OKU2 --- Klavye Tarama Kodunu Oku          ;
;                                                  ;
;Açıklama: Klavyeden girilen tuşların tarama kod-  ;
;larını AL yazmacına kaydeder.Bu sayede yön tuş-   ;
;ları kullanılır                                   ;
;                                                  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
KO2: CALL IMLEC_GUNCELLE
MOV AL,80H
OUT 60H,AL
KLAVYE_OKU2: IN AL,60H
MOV BL,AL
TEST BL,80H
JNZ SHORT KLAVYE_OKU2
CMP AL,50H
JE ASAGI
CMP AL,48H
JE YUKARI
CMP AL,4DH
JE SAG
CMP AL,4BH
JE SOL
CMP AL,3BH
JE KO1
CMP AL,01H
JE CIKIS
JMP KLAVYE_OKU2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;YAZI_YAZ --- Kutucuk içine karakteri yaz          ;
;                                                  ;
;Açıklama: Kutucuğa karakter yazmada kullanılır    ;
;                                                  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
YAZI_YAZ: ES MOV BYTE [SI],AL
INC DX
MOV AX,DX
PUSH DX
XOR DX,DX
MOV BX,0031H
DIV BX
CMP DX,0000H
JE SATIR_ATLA
POP DX
INC SI
```

```
INC SI
JMP KLAVYE_OKU1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ALT_SATIR  --- Alt satıra in                    ;
;                                             ;
;Açıklama: "Enter" tuşuna basıldığında kullanılır, ;
;DX yazmacının değerini güncellemede kullanılır    ;
;                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ALT_SATIR:MOV AX,DX
MOV BX,0031H
XOR DX,DX
DIV BX
INC AX
CMP AX,0010H
JE BASA_ATLA2
INC AX
XOR DX,DX
MUL BX
MOV DX,AX
PUSH DX
JMP SATIR_ATLA

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SATIR_ATLA  --- Alt satıra in                   ;
;                                             ;
;Açıklama: "Enter" tuşuna basıldığında veya genis- ;
;lik yetmediğinde bir alt satıra inmek için kulla- ;
;nılır                                          ;
;                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SATIR_ATLA: MOV AX,SI
MOV BX,00A0H
XOR DX,DX
DIV BX
CMP AX,0010H
JE BASA_ATLA1
INC AX
MUL BX
MOV SI,AX
ADD SI,02H
POP DX
JMP KLAVYE_OKU1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;BASA_ATLA  --- Kutunun başına geri döner         ;
;                                             ;
;Açıklama: Kutunun ilk hanesine geri döner        ;
;                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
BASA_ATLA1: POP DX
BASA_ATLA2: MOV SI,00A2H
MOV DX,0000H
JMP KLAVYE_OKU1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ASAGI  --- İmleci Asağı Kaydır                  ;
;                                             ;
;Açıklama: İmleci aşağı kaydırır                 ;
```

```
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ASAGI: CALL IMLEC_GUNCELLE
ADD DX,0031H
ADD SI,00A0H
CALL POZISYON_KONTROL
CALL IMLEC_GUNCELLE
CALL TAMPON_SIFIRLA
JMP KLAVYE_OKU2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;YUKARI  --- İmleci Yukarı Kaydır                 ;
;                                        ;
;Açıklama: İmleci yukarı kaydırır                 ;
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
YUKARI: CALL IMLEC_GUNCELLE
SUB DX,0031H
SUB SI,00A0H
CALL POZISYON_KONTROL
CALL IMLEC_GUNCELLE
CALL TAMPON_SIFIRLA
JMP KLAVYE_OKU2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SAG  --- İmleci Saga Kaydır                      ;
;                                        ;
;Açıklama: İmleci saga kaydırır                   ;
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SAG: CALL IMLEC_GUNCELLE
INC DX
INC SI
INC SI
CALL POZISYON_KONTROL
CALL IMLEC_GUNCELLE
CALL TAMPON_SIFIRLA
JMP KLAVYE_OKU2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SOL  --- İmleci Sola Kaydır                      ;
;                                        ;
;Açıklama: İmleci sola kaydırır                   ;
;                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SOL: CALL IMLEC_GUNCELLE
DEC DX
DEC SI
DEC SI
CALL POZISYON_KONTROL
CALL IMLEC_GUNCELLE
CALL TAMPON_SIFIRLA
JMP KLAVYE_OKU2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;POZISYON KONTROL  --- İmlecin Konumunu Kontrol Et ;
;                                        ;
;Açıklama: İmlecin konumunu kontrol edip yeniden   ;
;konumlandırır.                            ;
;                                        ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
POZISYON_KONTROL: PUSH DX
PUSH DX
POP AX
MOV BX,0031H
CMP DX,0000H
JL PK2
XOR DX,DX
DIV BX
CMP AX,0010H
JE PK1
INC AX
MOV BX,00A0H
PUSH DX
MUL BX
POP DX
MOV SI,AX
ADD SI,DX
ADD SI,DX
INC SI
INC SI
POP DX
JMP PK3
PK1: POP DX
MOV SI,00A2H
MOV DX,0000H
JMP PK3
PK2: POP DX
MOV SI,0A62H
MOV DX,030FH
PK3: RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GERI_DON  --- Yazılan karakteri sil            ;
;                                               ;
;Açıklama: "Backspace" tuşunun yaptığı işlemi yapar;
;                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
GERI_DON: DEC SI
DEC SI
ES MOV BYTE [SI],' '
DEC DX
JMP KLAVYE_OKU1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;IMLEC_GUNCELLE  --- İmleci Güncelle            ;
;                                               ;
;Açıklama: İmleci güncelle                      ;
;                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
IMLEC_GUNCELLE: ES MOV AL, BYTE [SI+01H]
NOT AL
ES MOV BYTE [SI+01H],AL
RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TAMPON_SIFIRLA  --- Klavye Tamponunu Sıfırla     ;
;                                               ;
;Açıklama: Klavye tamponuna bir karakter yazar    ;
;                                               ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
TAMPON_SIFIRLA: MOV AL,80H
OUT 60H,AL
RET

CIKIS: DB 00EAH            ; JMP FAR
DW 0000H, 1000H            ; 1000:0000 -> ANA MENU
DB 60 DUP (00H)
include 'yordamlar.inc'
```

9.2.6. Resim Çizdirme Yazılımı

Çizdirmek istediğimiz resim 400 x 400 çözünürlükte bir resim olsun. Bellekte 1024x768 çözünürlük ve 24bit renkte her piksel için dört bayt kullanıldığından bize 400 x 400 x 4 = 640000 bayt gerekmektedir. Yani yaklaşık 0.6 megabayt alana ihtiyacımız var. Ama burada dikkat etmemiz gereken şey resmimizin doğrusal olmaması, yani 1024 piksellik genişliğin tamamını kullanmadan bir alt satıra geçilmesinin gerekmesi. Yani resmin boyutlarını bozmadan ekrana yazdırabilmek için bellekte 1024 x 399 x 4 + 400 x 4 = 1635904 bayt = 1.56 megabayt yol katetmemiz gerekmektedir.

Üçüncü bölümde 8086 mikroişlemcisinin adresleme için 20 iğne kullandığından yani tam olarak 1 megabayt adresleme yapabildiğinden bahsettik. Ama resmi ekrana çizdirebilmek için 1.56 megabaytlık bir adresleme yapmamız gerekiyor. Bunun üstesinden gelebilmek için vidyo belleğinin bir özelliğini kullanacağız.

Vidyo belleğinin kullanacağımız özelliğine vidyo öbeği denir. Yani vidyo belleği önceden belirlenmiş belirli parçalara bölünmüştür ve biz ayrı ayrı bu öbeklere resmimizi yazdırabiliriz.

```
ORG 0420H
PUSH CS
POP DS
JMP BASLANGIC

VIDYOKIPI DB 03H
M_HATA1 DB "*Disketten veri okunamadi!          *",0AH,0DH,00H
M_HATA2 DB "*1024x768x24 Vidyo kipine geçilemedi!*",0AH,0DH,00H
CRTC_INFO_BLOCK DW 0400H    ; yatay piksel sayısı
CRTC_INFO_BLOCK2 DW 0018H  ; yatay eşzamanlama başı (piksel cinsinden)
CRTC_INFO_BLOCK3 DW 00A0H   ; yatay eşzamanlama sonu (piksel cinsinden)
CRTC_INFO_BLOCK4 DW 0300H   ; tarama çizgisi sayısı
CRTC_INFO_BLOCK5 DW 0003H   ; dikey eşzamanlama başı (tarama çizgisi cinsinden)
CRTC_INFO_BLOCK6 DW 001DH   ; dikey eşzamanlama sonu (tarama çizgisi cinsinden)
CRTC_INFO_BLOCK7 DB 03H       ; bayraklar
CRTC_INFO_BLOCK8 DW 03DFH
CRTC_INFO_BLOCK9 DW 0D240H            ; piksel saati (Hz)
CRTC_INFO_BLOCK10 DW 1770H            ; ekran tazelenme sıklığı 0.01Hz cinsinden
CRTC_INFO_BLOCK11 DB 28H DUP(00H)
GECICI1 DW 0000H                              ; gecici verilerin tutulduğu degisken
GECICI2 DW 0000H                              ; gecici verilerin tutulduğu değişken
GECICI3 DW 0000H                              ; gecici verilerin tutulduğu değişken
BASLANGIC:
; Vidyo belleği için gerekli ayarlamaları yap
CALL VIDYO_KIPI_SEC     ; Vidyo kipini sec
CMP AH,01H                        ; 01H ile sonucu karşılaştır
JE HATA_VIDYO_KIPI      ; Sonuç 01H ise HATA_VIDYO_KIPI etiketine atla
MOV AX,0A000H                   ; AX yazmacına A000H degerini yaz
PUSH AX                           ; AX yazmacını yığına at
POP ES                    ; Yığından ES yazmacına değer ata
PUSH CS                           ; CS yazmacının değerini yığına at
POP DS                    ; Yığından DS yazmacına değer ata
MOV SI,0000H              ; SI yazmacına 0000H değerini yaz
MOV CX,0001H              ; CX yazmacını sıfırla
PUSH CX                           ; CX yazmacını yığına kaydet
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; DISKETTEN_OKU               ;
```

```
; Dosyayı diskete yaz            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DISKETTEN_OKU: MOV DI,1000H
POP CX
MOV AH,02H              ; AH yazmacına 03H yazarak 03H alt yordamını belirt
MOV AL,12H              ; Yazılacak sektör sayısı (18x512bayt =18 sektör)
MOV DH,01H              ; Yazılacak yüz (01H = 2. yüz)
MOV DL,00H              ; Sürücü numarası ( 00H = diskete yaz)
MOV BX,DI               ; BX yazmacına tampon adresini yaz
PUSH CS                 ; CS yazmacını yığına kaydet
POP ES          ; ES yazmacına yığından veri kaydet
INT 13H         ; INT 13H çağır
JC HATA_DISKET_OKUMA            ; Hata var. HATA4 etiketine atla.
INC CH                  ; 18 sektör tamamlandı. 1 pist artır.
MOV CL,01H                      ; Sektör numarasını ayarla.
PUSH CX                         ; CX yazmacını yığına yaz
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV AX,0A000H                   ; AX yazmacına A000H degerini yaz
PUSH AX                         ; AX yazmacını yığına at
POP ES                  ; Yığından ES yazmacına değer ata
PUSH CS                         ; CS yazmacının değerini yığına at
POP DS                  ; Yığından DS yazmacına değer ata

;****************************************
; RESIM_YAZ                             *
; tampondaki veriyi ekrana yazdır       *
;****************************************

RESIM_YAZ: MOV CH,00H           ; CH yazmacına 00H yukle. CH yatay sayac gorevi gorecektir.
MOV DX, DI                      ; DX yazmacına DI yazmacının değerini yaz
MOV DI, GECICI3                 ; DI yazmacına GECICI3'un adresini yaz
MOV CX, [DI]            ; CX yazmacına GECICI3'un degerini yaz
MOV DI, DX              ; DI yazmacına DX yazmacının değerini yaz
RY:
DS MOV BYTE DH,[DI]                     ; Resim Kırmızı renk degeri
INC DI                                  ; DI yazmacının degerini artır
DS MOV BYTE DL,[DI]                     ; Resim Yeşil renk degeri
INC DI                                  ; DI yazmacının degerini artır
DS MOV BYTE BH,[DI]                     ; Resim Mavi renk degeri
INC DI                                  ; DI yazmacının degerini artır
ES MOV BYTE [SI],DH                     ; Ekran Mavi renk degeri
INC SI                                  ; SI yazmacının degerini artır
ES MOV BYTE [SI],DL                     ; Ekran Yeşil renk degeri
INC SI                                  ; SI yazmacının degerini artır
ES MOV BYTE [SI],BH                     ; Ekran Kırmızı renk degeri
INC SI                                  ; SI yazmacının degerini artır
ES MOV BYTE [SI],00H                    ; Ekran Alpha (00H olmalı)
INC SI                                  ; SI yazmacının degerini artır
INC CH                                  ; CH yazmacının degerini artır
CMP CH,00H                              ; Bir alt satıra geçilmeli mi?
JE ALT_SATIRA_GEC                       ; Evet. ALT_SATIRA_GEC etiketine atla
JMP RY                          ; Hayır. RY etiketine geri atla

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ALT_SATIRA_GEC         ;
; Alt satıra geç         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ALT_SATIRA_GEC: INC CL          ; CL yazmacının degerini artır
MOV CH,00H                      ; CH yazmacının degerini sıfırla
```

```
MOV DX,DI                         ; DX yazmacına DI yazmacını yaz
MOV DI,GECICI3            ; DI yazmacına GECICI3'ün adresini yaz
MOV [DI],CX                       ; GECICI3'e CX yazmacının değerini yaz
MOV DI,DX                         ; DI yazmacına DX yazmacının değerini yaz
CMP CL,10H                        ; CL yazmacının degerini 10H (16) ile karşılaştır
JE ALT_OBEGE_GEC                  ; 16 ise ALT_OBEGE_GEC etiketine atla
ADD SI,0C00H              ; Değilse SI yazmacına 0C00H değerini ekle
CMP DI,3400H              ; DI yazmacı ile 3400H değerini karşılaştır
JE DISKETTEN_OKU                  ; Eşit. O zaman DISKETTEN_OKU etiketine atla
JMP RESIM_YAZ             ; Eşit değil. RY etiketine atla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ALT_OBEGE_GEC           ;
; Alt öbeğe geç           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ALT_OBEGE_GEC: MOV CL,00H    ; CL yazmacına 00H degerini yaz
MOV DX,DI                                 ; DX yazmacına DI yazmacını yaz
MOV DI,GECICI3            ; DI yazmacına GECICI3'ün adresini yaz
MOV [DI],CX                       ; GECICI3'e CX yazmacını yaz
MOV DI,DX                         ; DI yazmacına DX yazmacını yaz
MOV AX,4F05H              ; AX:4F05 INT 10H - Seçili bank numarasını al
MOV BH,01H                        ; BH yazmacına 01H yaz
MOV BL,00H                        ; BL yazmacına 00H yaz
MOV SI,0000H              ; SI yazmacını sıfırla
INT 10H                           ; INT 10H çağır
MOV SI,GECICI2            ; SI yazmacına GECICI2'nin adresini yaz
MOV WORD DX,[SI]                  ; GECICI2'nin değerini DX yazmacına yaz
INC DX                    ; DX yazmacının degerini artır
MOV AX,4F05H              ; AX:4F05 INT 10H - Bir sonraki öbeği seç
MOV BH,00H                        ; BH yazmacına 00H yaz
MOV BL,00H                        ; BL yazmacına 00H yaz
INT 10H                           ; INT 10H çağır
MOV AX,4F05H              ; AX:4F05
MOV BH,00H                        ; BH yazmacına 00H yaz
MOV BL,01H                        ; BL yazmacına 01H yaz
INT 10H                           ; INT 10H çağır
MOV SI,0000H              ; SI yazmacını sıfırla
CMP DL,10H                        ; Resim bitti mi?
JE SON                    ; Evet.
MOV SI,GECICI2            ; SI yazmacına GECICI2'nin adresini yaz
MOV WORD [SI],DX                  ; GECICI2'ye DX yazmacının değerini yazdır
MOV SI,0000H              ; SI yazmacını sıfırla
CMP DI,3400H              ; DI yazmacının değeri 3400H mı?
JE DISKETTEN_OKU                  ; Evet. DISKETTEN_OKU etiketine atla
JMP RESIM_YAZ             ; Hayır. RESIM_YAZ etiketine atla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; HATA_DISKET_OKUMA           ;
; Disket okuma hatasını belirtir  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
HATA_DISKET_OKUMA: MOV AH,00H        ; Hata oluşmuş.
MOV AL,03H                           ; INT 10H 03H alt rutini
INT 10H                              ; INT 10H çağır.
MOV SI,M_HATA1            ; SI yazmacına hata yazısının adresini yaz
CALL SATIR_YAZ            ; SI yazmacı ile belirtilen yazı ekrana yazdırılsın
JMP SON                              ; SON etiketine atla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; HATA_VIDYO_KIPI              ;
; Vidyo kipi hatasını belirtir    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
HATA_VIDYO_KIPI: MOV AH,00H ; Hata oluşmuş.
MOV AL,01H                                  ; INT 10H 01H alt rutini
INT 10H                                     ; INT 10H çağır.
MOV SI,M_HATA2                              ; SI yazmacına hata yazısının adresini yaz
CALL SATIR_YAZ              ; SI yazmacı ile belirtilen yazı ekrana yazdırılsın
JMP SON                                     ; SON etiketine atla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SON: POP CX                                  ; CX yazmacını yığından doldur
MOV CX,0FFFFH                                ; CX yazmacına 0FFFFH yazdır
CALL GECIKME                                 ; GECIKME yordamını çağır
MOV CX,0FFFFH                                ; CX yazmacına 0FFFFH yazdır
CALL GECIKME                                 ; GECIKME yordamını çağır

DB 00EAH                                     ; JMP FAR
DW 0000H, 1000H                              ; 1000:0000 -> Menü

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; VIDYO_KIPI_SEC               ;
; Vidyo kipi seçilir            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
VIDYO_KIPI_SEC: PUSH CS ; CS yazmacını yığına at
POP ES                      ; ES yazmacını yığından doldur
MOV DI,CRTC_INFO_BLOCK           ; DI yazmacına CRTC_INFO_BLOCK
MOV AX,4F02H                ; AX yazmacına 4F02H yaz
MOV BX,0118H                ; BX yazmacına 0118H - Banklı vidyo belleği kullan
INT 10H                        ; INT 10H çağır
RET                            ; Geri dön
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DB 61 DUP (00H)
include 'yordamlar.inc'
```

9.2.7. Seri İletişim Alıcı Yazılımı

```
ORG 0420H
PUSH CS
POP DS
JMP BASLANGIC
M_MESAJ0 DB "ALICI",0AH,0DH,0AH,0DH,00H
M_MESAJ1 DB "MESAJ:",00H
BASLANGIC:
CALL VIDYO_KIPI_AYARLA                  ; Vidyo kipini ayarla
MOV SI,M_MESAJ0                     ; Sayıları ve işaretleri yazdır
CALL SATIR_YAZ                          ; Sayıları ve işaretleri yazdır
MOV SI,M_MESAJ1                     ; Sayıları ve işaretleri yazdır
CALL SATIR_YAZ                          ; Sayıları ve işaretleri yazdır
MOV AH,01H                          ; AH yazmacına 01H yaz
MOV CH,20H                          ; CH yazmacına 20H yaz
INT 10H                             ; INT 10H çağır. İmleci yok et
MOV AX,0B800H                       ; AX yazmacına B800 yaz
MOV ES,AX                           ; AX yazmacını ES yazmacına aktar
PUSH CS                             ; CS yazmacını yığına at
POP DS                           ; Yığından DS yazmacını doldur
MOV SI,014CH                     ; Imlec 015EH dan başlasın
CALL RS232_AYARLA                   ; RS232'yi ayarla
```

```
DONGU: MOV AL,00H                 ; AL yazmacının değerini 00H yap
MOV AH,02H                        ; AH yazmacına 02H değerini kaydet
INT 14H                           ; INT 14H kesmesine atla
CMP AL,00H                        ; AL yazmacı ile 00H verisini karşılaştır
JE DONGU                          ; Eşitse DONGU'ye atla
CMP AL,80H                        ; AL yazmacı ile 80H verisini karşılaştır
JE DONGU                          ; Eşitse DONGU'ye atla
CMP AL,60H                        ; AL yazmacı ile 60H verisini karşılaştır
JE DONGU                          ; Eşitse DONGU'ye atla
CMP AL,1BH                        ; AL yazmacı ile 1BH verisini karşılaştır
JE SON                          ; Eşitse SON etiketine atla
ES MOV [SI],AL                  ; Ekrana SI konumuna AL yazmacını yaz
INC SI                      ; SI yazmacının değerini artır
INC SI                      ; SI yazmacının degerini bir daha artır.
JMP DONGU                         ; DONGU etiketine atla

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; RS232 Ayarla                         ;
; - RS232'nin ayarlanmasında kullanılır     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
RS232_AYARLA: MOV DX,0000H ; COM1
MOV AH,00H                        ; Seri kapı ayarlama
MOV AL,00000010B                  ; 110 baud - 7 bit veri - eşlik biti yok - 1 tane duraklama biti
INT 14H                           ; INT 14H çağır
RET                               ; Geri dön.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; VIDYO_KIPI_AYARLA                    ;
; - 80x25 Vidyo kipini ayarlar          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
VIDYO_KIPI_AYARLA: MOV AH,00H          ; AH = 00H
MOV AL,03H                      ; AL = 03H => 16 renk 80 x 25 yazı vidyo kipi
INT 10H                           ; INT 10H çağır
RET                               ; Geri dön.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SON: DB 00EAH                   ; JMP FAR
DW 0000H, 1000H                   ; 1000:0000 -> ANA MENU

DB 379 DUP (00H)
include 'yordamlar.inc'
```

9.2.8. Seri İletişim Verici Yazılımı

```
ORG 0420H
PUSH CS                           ; CS yazmacını yığına at
POP DS                          ; Yığından DS yazmacını doldur
JMP BASLANGIC                   ; BASLANGIC etiketine atla
M_MESAJ0 DB "VERICI",0AH,0DH,0AH,0DH,00H
M_MESAJ1 DB "MESAJ:"
TAMPON DB 00H
BASLANGIC:

CALL VIDYO_KIPI_AYARLA              ; Vidyo kipini ayarla
MOV SI,M_MESAJ0                   ; Sayıları ve işaretleri yazdır
CALL SATIR_YAZ                      ; Sayıları ve işaretleri yazdır
```

```
MOV SI,M_MESAJ1                    ; Sayıları ve işaretleri yazdır
CALL SATIR_YAZ                          ; Sayıları ve işaretleri yazdır
MOV AH,01H                         ; AH = 01H
MOV CH,20H                         ; CH = 20H
INT 10H                            ; INT 10H çağır. İmleci kaldır.
MOV AX,0B800H                           ; AX yazmacına vidyo bellek adresini yaz
MOV ES,AX                          ; ES yazmacına AX yazmacını aktar
MOV SI,014CH                    ; İmleç 014CH dan başlasın
CALL RS232_AYARLA                  ; RS232_AYARLA yordamını çağır

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; KLAVYE_OKU                    ;
; - Klavyeden girilen veriyi okur        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
KLAVYE_OKU: MOV AH,01H                  ; AH yazmacına 01H yaz
INT 16H                            ; INT 16H çağır
JZ KLAVYE_OKU                   ; Sıfır bayrağı 1 atla
MOV AH,00H                         ; AH yazmacına 00H yaz
INT 16H                            ; INT 16H çağır
CMP AL,1BH                         ; AL yazmacı ile 1BH karşılaştır (ESC)
JE SON                          ; Eşit. O zaman yazılımı bitir.
MOV DI,TAMPON                           ; DI yazmacına TAMPON'un adresini yaz
MOV [DI],AL                        ; TAMPON'a AL yazmacını yaz
ES MOV [SI],AL                  ; Ekrana AL yazmacının içeriğini yaz
JMP SERI_YOLLA                          ; SERI_YOLLA etiketine atla

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; RS232_AYARLA                   ;
; - RS232'nin ayarlanmasında kullanılır    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
RS232_AYARLA: MOV DX,0000H ; COM1
MOV AH,00H                         ; Seri kapı ayarlama
MOV AL,00000010B                   ; 110 baud - 7 bit veri - eşlik biti yok - 1 tane duraklama biti
INT 14H                            ; INT 14H çağır
RET                                ; Geri dön
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; SERI_YOLLA                    ;
; - RS232 kapısından AL yazmacındaki       ;
;   veriyi yollar                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SERI_YOLLA: MOV AH,01H                  ; Veri yollama için AH değerini yaz
MOV DI,TAMPON                           ; DI yazmacına tamponun adresini yaz
MOV AL,[DI]                        ; Tampondaki veriyi AL yazmacına yaz
CMP AL,00H                         ; AL yazmacı ile 00H verisini karşılaştır
JE KLAVYE_OKU                   ; Eşitseler KLAVYE_OKU etiketine atla.
INC SI                          ; SI yazmacının degerini artır
INC SI                          ; SI yazmacının degerini artır
INT 14H                            ; INT 14H çağır
MOV BYTE [DI],00H                  ; TAMPON'a 00H yaz
JMP KLAVYE_OKU                          ; Oluşmamış. Geri dön.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; VIDYO_KIPI_AYARLA                ;
; - 80x25 Vidyo kipini ayarlar           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
VIDYO_KIPI_AYARLA: MOV AH,00H           ; AH = 00H
MOV AL,03H                        ; AL = 03H => 16 renk 80 x 25 yazı vidyo kipi
```

```
INT 10H                                    ; INT 10H çağır
RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SON: DB 00EAH                           ; JMP FAR
DW 0000H, 1000H                            ; 1000:0000 -> ANA MENU

DB 365 DUP (00H)
include 'yordamlar.inc'
```

10. EK A – x86 Komutlarının Listesi

Burada 8086 mikroişlemcisinin komut kümesini ve ardından da diğer x86 mikroişlemcilerinin getirdiği yeni komutları bulabilirsiniz.

10.1. 8086 Komutları

Komutlar	İşlenen	Tanım
MOV	İŞLENEN1 İŞLENEN2	MOV (**mov**e – taşı) komutu bir veriyi bir yerden alıp başka bir yere taşır. *işlenen2*'yi *işlenen1*'e kopyalar. Mesela bellekte kayıtlı olan bir baytı bir yazmaca kaydeder. MOV komutu ile aşağıdaki 3 işlemi yapmanız mümkün değildir: - MOV komutu ile CS ve IP yazmaçlarının değerleri değiştirilemez. - MOV komutu ile bir kesim yazmacından diğer bir kesim yazmacına veri aktarılamaz. Aracı bir genel amaçlı yazmaç kullanılmalıdır. - MOV komutu ile bir kesim yazmacına anlık veri kaydedilemez. Aracı bir genel amaçlı yazmaç kullanılmalıdır.
PUSH	İŞLENEN1	PUSH komutu ile yığına bir veri kaydedilebilir. SP 2 azaltılır ve SS:[SP] ye *işlenen1* kaydedilir. 80186 işlemcisi ve üzeri x86 işlemcileri PUSH ile bir anlık veriyi de yığına kaydedebilir.
POP	İŞLENEN1	POP komutu ile yığından bir veri alınır. SP 2 artırılır ve *işlenen1*'e SS:[SP] kaydedilir.
XCHG	İŞLENEN1 İŞLENEN2	XCHG (e**xch**an**g**e – değiştirme) komutu işlenenlerin verilerini birbiriyle değiştirir. Yani *işlenen2*'nin değerini *işlenen1*'e, *işlenen1*'in değerini *işlenen2*'ye kaydeder.
IN	AL veya AX KAPI	IN komutu AL veya AX yazmaçlarına seçilen kapıdaki veriyi aktarır. Eğer kapı numarası 255'den daha büyük ise kapı numarasını belirtmek için DX yazmacı kullanılmalıdır.
OUT	KAPI AL veya AX	OUT komutu AL veya AX yazmaçlarındaki veriyi seçilen kapıya iletir. Eğer kapı numarası 255'den daha büyük ise kapı numarasını belirtmek için DX yazmacı kullanılmalıdır.
XLAT	[BX]	XLAT komutu AL yazmacına DS:[BX+AL] adresindeki baytı aktarır. MOV AL,[BX+AL] komutu ile aynı işlemi yapar.
XLATB	-	XLATB komutu AL yazmacına DS:[BX+AL] adresindeki baytı aktarır. XLAT komutunun işlenen almayan kısa şeklidir.
LEA	İŞLENEN1 İŞLENEN2	LEA (**l**oad **e**ffective **a**ddress – etkin adresi yükle) komutu etkin adresi belirtilen yazmaca kaydeder. Aşağıda belirtilen biçimde yazılır. *işlenen1*'e *işlenen2*'nin adresi kaydedilir.
LDS	İŞLENEN1 İŞLENEN2	LDS (**l**oad **DS** – DS'e yükle) komutu bellekteki ikili kelimeyi bir genel amaçlı yazmaç ve DS kesim yazmacına kaydeder. *işlenen1*'e ikili kelimenin ilk kelimesi kaydedilir. DS kesim yazmacına ikili kelimenin ikinci kelimesi kaydedilir. *işlenen2* bir bellek imleci olmalıdır.
LES	İŞLENEN1 İŞLENEN2	LES (**l**oad **ES** – ES'ye yükle) komutu LDS komutunun yaptığı işlemin yaklaşık olarak aynısını yapar. Tek farkı DS kesim segmenti yerine ES kesim segmentinin değerini değiştirmesidir.
LAHF	-	LAHF (**l**oad **AH** with **f**lags – AH yazmacına bayrak bitlerini yükle) komutu SF,ZF,AF,PF ve CF bayrak bitlerini sırasıyla AH yazmacının 7,6,4,2 ve 0 bitlerine kaydeder. Bayrak bitleri bu işlemden etkilenmez.
SAHF	-	SAHF (**s**tore **AH** into **f**lags – AH yazmacını bayrak bitlerine kaydet) komutu AH yazmacının 7,6,4,2 ve 0 bitlerini sırasıyla SF,ZF,AF,PF ve CF bayrak bitlerine kaydeder.
PUSHF	-	PUSHF komutu PUSH komutu gibi çalışır. Tek farkı sadece bayrak yazmacının yığına kaydedilmesidir.
POPF	-	POPF komutu POP komutu gibi çalışır. Tek farkı yığından aldığı veriyi direk bayrak yazmacına kaydetmesidir.

<table>
<tr><td>ADD</td><td>İŞLENEN1
İŞLENEN2</td><td>ADD (add – topla) komutu ile işlenen1 ve işlenen2'nin değerleri toplanır ve sonuç işlenen1'e kaydedilir. işlenen2'nin değeri değişmez.</td></tr>
<tr><td>ADC</td><td>İŞLENEN1
İŞLENEN2</td><td>ADC (add with carry – artan ile topla) komutu ile işlenen1 ve işlenen2 toplanır ve sonuç işlenen1'e kaydedilir. işlenen2'nin değeri ADD komutunda da olduğu gibi değişmez. Eğer CF bayrak biti 1 ise işlenen1'e 1 eklenir.</td></tr>
<tr><td>INC</td><td>İŞLENEN1</td><td>INC (increase – artır) komutu işlenen1'in değerini 1 artırır. CF bayrak bitinde değişikliğe neden olmaz.</td></tr>
<tr><td>AAA</td><td>-</td><td>AAA (ASCII Adjust for ADD – ADD komutu için ASCII düzenleme) komutu AL yazmacındaki değeri paketlenmemiş bir ondalık değere çevirir. Eğer AL yazmacının değeri 9'dan büyükse veya AF bayrak biti 1 ise AH yazmacının değeri 1, AL yazmacının değeri 6 artırılır. AF ve CF bayrak bitleri 1 yapılır. AAA komutu herhangi bir işlenen almaz. AL yazmacının değeri 9'dan büyük değilse AF ve CF bayrak bitleri sıfırlanır. Her iki durumda da AL yazmacının üst yarım baytı sıfırlanır.</td></tr>
<tr><td>DAA</td><td>-</td><td>DAA (Decimal Adjust for ADD – ADD komutu için ondalık düzenleme) komutu AL yazmacındaki değeri paketlenmiş bir ondalık değere çevirir. Eğer AL yazmacının alt yarım baytının değeri 9'dan büyük veya AF bayrak biti 1 ise AL yazmacına 06H eklenir ve AF bayrak biti birlenir. Eğer AL yazmacının değeri 9F'den büyük veya CF bayrak biti 1 ise AL yazmacına 60H değeri eklenir ve CF bayrak biti birlenir.</td></tr>
<tr><td>SUB</td><td>İŞLENEN1
İŞLENEN2</td><td>SUB (subtract – çıkarma) komutu ile işlenen1'den işlenen2'nin değeri çıkarılır ve sonuç işlenen1'e kaydedilir. Bu komut sonrasında işlenen2'nin değeri değişmez.</td></tr>
<tr><td>SBB</td><td>İŞLENEN1
İŞLENEN2</td><td>SBB (subtract with borrow – elde ile çıkarma) komutu CF bayrak biti 1 ise işlenen1'den işlenen2'yi çıkartır, ardından işlenen1'den 1 çıkartır ve sonucu işlenen1'e kaydeder. CF bayrak biti 0 olduğunda SUB komutu ile aynı işlemi yapar. Her iki CF bayrak biti durumunda da işlenen2'nin değeri değişmez.</td></tr>
<tr><td>DEC</td><td>İŞLENEN1</td><td>DEC (decrease – azalt) komutu INC komutunun tersidir. işlenen1'in değerini 1 azaltır.</td></tr>
<tr><td>NEG</td><td>İŞLENEN1</td><td>NEG (negate – ters çevir) komutu işlenen1'in 2'nin tümleyenini alır ve işlenen1'e geri kaydeder.</td></tr>
<tr><td>CMP</td><td>İŞLENEN1
İŞLENEN2</td><td>CMP (compare – karşılaştır) komutu işlenen1 ile işlenen2'nin değerlerini karşılaştırır. Karşılaştırmanın sonucuna göre değiştirilmesi gereken bayrak bitlerinin (OF, SF, ZF, AF, PF, CF) değerini değiştir. Bu komut işlenen1 ve işlenen2'nin değerini değiştirmez.
Aşağıdaki tabloda örnek olarak işlenen1 ve işlenen2 değerleri CMP komutu sonrasında değişmesi gereken bayrak bitlerinin durumları verilmiştir.
<table>
<tr><th>Durum</th><th>İşlenen1</th><th>işlenen2</th><th>Bayrak Bitleri</th></tr>
<tr><td>1</td><td>5</td><td>5</td><td>ZF = 1, PF = 1</td></tr>
<tr><td>2</td><td>5</td><td>6</td><td>CF = 1, SF = 1, PF = 1, AF = 1</td></tr>
<tr><td>3</td><td>5</td><td>4</td><td>-</td></tr>
<tr><td>4</td><td>5</td><td>-4</td><td>CF = 1, PF = 1, AF = 1</td></tr>
<tr><td>5</td><td>-5</td><td>5</td><td>SF = 1, PF = 1</td></tr>
<tr><td>6</td><td>-5</td><td>-5</td><td>ZF = 1, PF = 1</td></tr>
<tr><td>7</td><td>-5</td><td>-6</td><td>-</td></tr>
</table></td></tr>
<tr><td>AAS</td><td>-</td><td>AAS (ASCII Adjust after Substraction – Çıkarma işlemi sonrası ASCII düzenleme) komutu AL yazmacındaki değeri paketlenmiş bir ondalık değere çevirir. Eğer AL yazmacının değeri 9'dan büyük veya AF bayrak biti 1 ise AL yazmacının değeri 6, AH yazmacının değeri 1 azaltırılır. AF ve CF bayrak bitleri bire eşitlenir. AL yazmacının değeri 9'dan büyük değilse AF ve CF bayraklarının değeri sıfırlanır. Her iki durumda da AL yazmacının üst yarım biti sıfırlanır.</td></tr>
<tr><td>DAS</td><td>-</td><td>DAS (Decimal Adjust after SUB – SUB komutu sonrası ondalık düzenleme) komutu AL yazmacındaki değeri paketlenmiş bir ondalık değere çevirir. Eğer AL yazmacının alt yarım baytının değeri 9'dan büyük</td></tr>
</table>

		veya AF bayrak biti 1 ise AL yazmacından 06H çıkarılır ve AF bayrak biti birlenir. Eğer AL yazmacının değeri 9F'den büyük veya CF bayrak biti 1 ise AL yazmacından 60H değeri çıkarılır ve CF bayrak biti birlenir.
MUL	İŞLENEN1	MUL (**mul**tiply – çarp) komutu *işlenen1* bir bayt ise *işlenen1*'i AL yazmacı ile çarpıp sonucu AX yazmacına kaydeder. *işlenen1* kelime ise *işlenen1*'i AX yazmacı ile çarpıp sonucu DX AX yazmaçlarına kaydeder. CF ve OF bayrak bitleri sonucun üst baytları 0 olduğu sürece sıfıra eşittir.
IMUL	İŞLENEN1	IMUL komutu MUL komutunun işaretli olanıdır. CF ve OF bayrak bitleri sonuç *işlenen1*'e sığabiliyorsa sıfıra eşittir.
AAM	-	AAM (**A**SCII **A**djust after **M**ultiplication – Çarpma sonrası ASCII düzenleme) komutu AL yazmacının değerini alıp 10'a böler ve sonucu AH yazmacına kaydeder. Artan olursa AL yazmacına kaydedilir.
DIV	İŞLENEN1	DIV (**div**ide – böl) komutu *işlenen1* bir bayt ise *işlenen1*'i AX yazmacına böler ve bölümü AL yazmacına kalanı AH yazmacına kaydeder. *işlenen1* kelime ise DX AX i *işlenen1*'e böler ve bölümü AX yazmacına kalanı DX yazmacına kaydeder.
IDIV	İŞLENEN1	DIV komutunun işaretli olanıdır.
AAD	-	AAD (**A**SCII **A**djust before **D**ivision – Bölme öncesi ASCII düzenleme) komutu bölme işlemi için 2 tane BCD sayısını hazırlar.
CBW	-	CBW komutu (convert byte to word – baytı kelimeye çevir) AL yazmacındaki baytı kelimeye çevirir. AL yazmacının en anlamlı biti bire eşit ise AH yazmacına FFH değeri yazılır. Değilse Ah yazmacına 00H değeri yazılır.
CWD	-	CWD (convert word to double word – kelimeyi ikili kelimeye çevir) komutu AX yazmacındaki kelimeyi ikili kelimeye çevirir. Üst kelime DX yazmacına, alt kelime AX yazmacına kaydedilir. AX yazmacının en anlamlı biti 1 ise DX yazmacına FFFFH değeri yazılır. Değilse DX yazmacına 0000H değeri yazılır.
NOT	İŞLENEN1	NOT komutu *işlenen1*'in her bitini tersine çevirir.
SHL	İŞLENEN1 İŞLENEN2	SHL (**sh**ift logical **l**eft – lojik sola ötele) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sola öteler. En soldaki bit öteleme sonucunda CF bayrak bitine kaydedilir. En sağdaki bit sıfırlanır. Eğer *işlenen1*'in işareti değişmez ise OF 0, aksi takdirde 1'e eşittir.
SAL	İŞLENEN1 İŞLENEN2	SAL (**s**hift **a**rithmetic **l**eft – aritmatik sola ötele) komutu SHL komutu ile aynı işleve sahiptir. İşlem ve adresleme kodu aynıdır.
SHR	İŞLENEN1 İŞLENEN2	SHR (**sh**ift logic **r**ight – lojik sağa ötele) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sağa öteler. En sağdaki bit öteleme sonucunda CF bayrak bitine kaydedilir. En soldaki bit sıfırlanır. Eğer islenen1'in işareti değişmez ise OF 0, aksi takdirde 1'e eşittir.
SAR	İŞLENEN1 İŞLENEN2	SAR (**s**hift **a**rithmetic **r**ight – aritmatik sağa ötele) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sağa öteler. En sağdaki bit öteleme sonucunda CF bayrak bitine kaydedilir. En soldaki bit işaret biti kabul edildiğinden değeri değiştirilmez. Eğer *işlenen1*'in işareti değişmez ise OF 0, aksi takdirde 1'e eşittir. SAR komutu aritmetik işlemlerde işaret biti ötelenmediğinden daha uygundur.
ROL	İŞLENEN1 İŞLENEN2	ROL (**ro**tate **l**eft – sola döndür) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sola döndürür. Yani bitler sola ötelenir. Öteleme öncesinde en solda olan bit öteleme sonrasında en sağa geçer. Bu bit aynı zamanda CF bayrak bitine kaydedilir. Eğer *işlenen1*'in işareti değişmez ise OF 0, aksi takdirde 1 değerini alır.
ROR	İŞLENEN1 İŞLENEN2	ROR (**ro**tate **r**ight – sağa döndür) komutu *işlenen1*'i *işlenen2*'nin değeri kadar sağa döndürür. Yani bitler sağa ötelenir. Öteleme öncesinde en sağda olan bit öteleme sonrasında en sola geçer. Bu bit aynı zamanda CF bayrak bitine kaydedilir. Eğer *işlenen1*'in işareti değişmez ise OF 0, aksi takdirde 1 değerini alır.
RCL	İŞLENEN1 İŞLENEN2	RCL (**r**otate with **C**F **l**eft – CF bayrak biti ile beraber sola döndür) komutu *işlenen1* + CF yi *işlenen2*'nin değeri kadar sola döndürür. Yani bitler sola ötelenir. Öteleme öncesinde en solda olan bit öteleme sonrasında en sağa yani CF bayrak bitine geçer. Bayrak bitindeki değer ise en sağa geçer. Eğer *işlenen1*'in işareti değişmez ise OF 0, aksi takdirde 1 değerini alır.

RCR	İŞLENEN1 İŞLENEN2	RCR (**r**otate with **C**F **r**ight – CF bayrak biti ile beraber sağa döndür) komutu *işlenen1* + CF yi *işlenen2*'nin değeri kadar sağa döndürür. Yani bitler sağa ötelenir. Öteleme öncesinde en sağda olan bit öteleme sonrasında en sola yani CF bayrak bitine geçer. Bayrak bitindeki değer ise en sola geçer. Eğer *işlenen1*'in işareti değişmez ise OF 0, aksi takdirde 1 değerini alır.
AND	İŞLENEN1 İŞLENEN2	AND komutu mantıksal VE kapısının görevini görür. Sonuç *işlenen1*'e aktarılır.
TEST	İŞLENEN1 İŞLENEN2	TEST komutu AND komutuna benzer bir komuttur. Sonuç herhangi bir yerde kaydedilmez. Sadece SF, ZF ve PF bayrak bitleri sonuca göre 1 veya 0 değerini alır.
OR	İŞLENEN1 İŞLENEN2	OR komutu mantıksal VEYA kapısının görevini görür. Sonuç islenen1'e aktarılır.
XOR	İŞLENEN1 İŞLENEN2	XOR komutu mantıksal ÖZEL VEYA kapısının görevini görür. Sonuç *işlenen1*'e aktarılır.
REP	İŞLENEN1	REP (**rep**eat – tekrarla) komutu işlenen olarak verilen MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW komutlarından birinin CX yazmacındaki değer kadar tekrar edilmesini sağlar. Her tekrarda CX yazmacının değeri 01H azaltılır. CX yazmacı 0 olduğunda sıradaki komuta atlar.
REPE	İŞLENEN1	REPE (**rep**eat if **e**qual – eşit ise tekrarla) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birinin CX yazmacındaki değer 0 olana kadar ve ZF bayrak biti 1 olduğu sürece tekrarlar. ZF bayrak biti 0 olduğu anda tekrarlama durdurulur ve bir sonraki komuta atlanır. Her tekrarda CX yazmacının değeri 01H azaltılır.
REPZ	İŞLENEN1	REPZ (**rep**eat if **z**ero – sıfır ise tekrarla) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birinin CX yazmacındaki değer 0 olana kadar ve ZF bayrak biti 1 olduğu sürece tekrarlar. ZF bayrak biti 0 olduğu anda tekrarlama durdurulur ve bir sonraki komuta atlanır. Her tekrarda CX yazmacının değeri 01H azaltılır.
REPNE	İŞLENEN1	REPNE (**rep**eat if **n**ot **e**qual – eşit değilse tekrarla) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birinin CX yazmacındaki değer 0 olana kadar ve ZF bayrak biti 0 olduğu sürece tekrarlar. ZF bayrak biti 1 olduğu anda tekrarlama durdurulur ve bir sonraki komuta atlanır. Her tekrarda CX yazmacının değeri 01H azaltılır.
REPNZ	İŞLENEN1	REPNZ (**rep**eat if **n**ot **z**ero – sıfır değilse tekrarla) komutu işlenen olarak verilen CMPSB, CMPSW, SCASB, SCASW komutlarından birinin CX yazmacındaki değer 0 olana kadar ve ZF bayrak biti 0 olduğu sürece tekrarlar. ZF bayrak biti 1 olduğu anda tekrarlama durdurulur ve bir sonraki komuta atlanır. Her tekrarda CX yazmacının değeri 01H azaltılır.
MOVSB / MOVSW	-	MOVSB komutu DS:[SI] adresindeki bir baytı ES:[DI] adresine kaydeder. MOVSW komutu bayt yerine bir kelime kaydeder. DF bayrak biti 0 ise SI ve DI işlem sonunda 1 artırılır, 1 ise SI ve DI 1 azaltılır.
CMPSB / CMPSW	-	CMPSB komutu ES:[DI] ve DS:[SI] adresindeki baytları karşılaştırır. CMPSW komutu bayt yerine kelimeleri karşılaştırır. OF,SF,ZF,AF,PF ve CF bayrak bitleri karşılaştırma sonucuna göre değer alır. DF bayrak biti 0 ise SI ve DI 1 artırılır, 1 ise SI ve DI 1 azaltılır.
SCASB / SCASW	-	SCASB komutu AL yazmacındaki bayt ile ES:[DI] adresindeki baytı karşılaştırır. OF,SF,ZF,AF,PF ve CF bayrak bitleri karşılaştırma sonucuna göre değer alır. DF bayrak biti 0 ise DI 1 artırılır, 1 ise DI 1 azaltılır. SCASW AX yazmacındaki kelime ile ES:[DI] adresindeki kelimeyi karşılaştırır. DF bayrak biti 0 ise DI 2 artırılır, 1 ise DI 2 azaltılır.
LODSB / LODSW	-	LODSB komutu DS:[SI] adresindeki baytı AL yazmacına kaydeder. DF 0 ise SI 1 artırılır, 1 ise 1 azaltılır. LODSW komutu DS:[SI] adresindeki kelimeyi AX yazmacına kaydeder. DF bayrak biti 0 ise SI yazmacı 2 artırılır, DF bayrak biti 1 ise SI yazmacı 2 azaltılır.
STOSB / STOSW	-	STOSB komutu ES:[DI] adresine AL yazmacındaki baytı kaydeder. DF bayrak biti 0 ise DI 1 artırılır, 1 ise 1 azaltılır.

		STOSW komutu ES:[DI] adresine AX yazmacındaki kelimeyi kaydeder. bayrak biti 0 ise DI 2 artırılır, 1 ise 2 azaltılır.
CALL	İŞLENEN1	CALL komutu kontrolun bir yordama aktarılmasını sağlar. CALL komutu birkaç adresleme çeşidi kullanır. Kesim içi direk adresleme, kesim içi dolaylı adresleme, kesimler arası direk adresleme ve kesimler arası dolaylı adresleme yapan CALL türevleri vardır. Kesim içi adresleme kullanıldığı takdirde CALL komutu çalıştırıldığında IP yazmacının değeri yığına kaydedilir ve prosedüre atlanır. Bir RET komutu ile karşılaşılınca değerler yığından geri alınır ve kalınan noktadan işlemler devam eder. Kesimler arası CALL komutu kullanıldığında ise IP ve CS yazmaçlarının değerleri yığına kaydedilir.
JMP	İŞLENEN1	JMP komutu sayesinde *işlenen1'*deki adrese atlanır.
RET / RETF	İŞLENEN1	RET komutu CALL komutu ile çağırılmış bir prosedürden geri dönülmesini sağlar. IP yazmacını yığından geri yükler. RETF komutu kesimler arası bir CALL komutu kullanılıyorsa kullanılmalıdır. RETF CS ve IP yazmaçları yığından geri yüklenir. Eğer işlenen olarak anlık bir veri yazılmış ise bu anlık veri SP yazmacına eklenir. İşlenen kullanılmayabilir.
JE / JZ	İŞLENEN1	JE (**j**ump if **e**qual – eşit ise atla) ve JZ (**j**ump if **z**ero – sıfır ise atla) komutları ZF bayrak biti 1 ise atlar. Aksi takdirde bir sonraki komuttan işlemler devam eder. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JL / JNGE	İŞLENEN1	JL (**j**ump if **l**ess – az ise atla) ve JNGE (jump if not greater and not equal – eşit ve büyük değilse atla) komutları SF ve OF bayrak bitleri birbirlerine eşit değilse atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçükse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JLE / JNG	İŞLENEN1	JLE (**j**ump if **l**ess or **e**qual – az veya eşit ise atla) ve JNG (**j**ump if **n**ot **g**reater – büyük değilse atla) komutları SF ve OF bayrak bitleri birbirlerine eşit değilse veya ZF bayrak biti 1 ise atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçük veya eşitse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JB / JNAE	İŞLENEN1	JB (**j**ump if **b**elow – az ise atla) ve JNAE (**j**ump if **n**ot **a**bove and not **e**qual – büyük ve eşit değilse atla) komutları CF bayrak biti 1 olduğu durumda atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçükse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JBE / JNA	İŞLENEN1	JBE (**j**ump if **b**elow or **e**qual – az veya eşit ise atla) ve JNA (**j**ump if **n**ot **a**bove – büyük değilse atla) komutları CF bayrak biti 1 veya ZF bayrak biti 1 olduğu durumda atlar. CMP komutu kullanılırken eğer birinci işlenen ikinci işlenenden küçük veya eşitse bu durum oluşmuş olur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JP / JPE	İŞLENEN1	JP (**j**ump if **p**arity – eşlik 1 ise atla) ve JPE (**j**ump if **p**arity bit **e**ven – eşlik çift ise atla) komutları PF bayrak biti 1 olduğu durumda atlar. İşlenen olarak bir etiket kullanılabileceği gibi bir adres de kullanılabilir.
JO	İŞLENEN1	JO (**j**ump if **o**verflow – taşma var ise atla) komutu OF bayrak biti 1 ise atlar. İşlenen olarak bir etiket kullanılabileceği gibi bir adres de kullanılabilir.
JS	İŞLENEN1	JS (**j**ump if **s**igned – işaretli ise atla) komutu SF bayrak biti 1 olduğu zaman atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JNE / JNZ	İŞLENEN1	JNE (**j**ump if **n**ot **e**qual – eşit değilse atla) ve JNZ (**j**ump if **n**ot **z**ero – sıfır değilse atla) komutları ZF bayrak biti 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JNL / JGE	İŞLENEN1	JNL (**j**ump if **n**ot **l**ower – az değilse atla) ve JGE (**j**ump if **g**reater or **e**qual – eşit veya büyük ise atla) komutları SF ve OF bayrak bitleri birbirine eşit ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir. Yani CMP komutunun sonucu olarak birinci işlenen ikinci işlenenden büyük veya eşit ise JNL komutu atlar.

JNLE / JG	İŞLENEN1	JNLE (**j**ump if **n**ot **l**ower and not **e**qual – eşit ve küçük değilse atla) ve JG (**j**ump if **g**reater –büyük ise atla) komutları SF ve OF bayrak bitleri birbirine eşit ve ZF bayrak biti 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir. Yani CMP komutunun sonucu olarak birinci işlenen ikinci işlenenden büyük ise JNL komutu atlar.
JNB / JAE	İŞLENEN1	JNB (**j**ump if **n**ot **b**elow – az değilse atla) ve JAE (**j**ump if **a**bove or **e**qual – büyük veya eşit ise atla) komutları CF bayrak biti 0 ise atlar. Yani CMP komutunun sonucunda birinci işlenen ikinci işlenenden küçük değilse bu durum oluşur. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JNBE / JA	İŞLENEN1	JNBE (**j**ump if **n**ot **b**elow and not **e**qual – eşit ve az değilse atla) ve JA (**j**ump if **a**bove - büyük ise atla) komutları ZF ve CF bayrak bitleri 0 ise atlar. Yani CMP komutunun sonucu olarak birinci işlenen ikinci işlenenden büyükse atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JNP / JPO	İŞLENEN1	JNP (**j**ump if **n**ot **p**arity – eşlik 0 ise atla) ve JPO (**j**ump if **p**arity **o**dd – eşlik tek ise atla) komutları PO bayrak biti 0 olduğunda atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JNO	İŞLENEN1	JNO (**j**ump if **n**ot **o**verflow – taşma yok ise atla) komutu OF 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JNS	İŞLENEN1	JNS (**j**ump if **n**ot **s**igned – işaretsiz ise atla) komutu SF bayrak biti 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bi adres de girilebilir.
LOOP	İŞLENEN1	LOOP (**loop** – döngü) komutu CX yazmacının değeri kadar islenen1'in adresine atlar. CX her atlamada bir azaltılır. CX yazmacının değeri 0 olduğunda atlanmaz ve işlemlere bir sonraki komuttan devam edilir. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
LOOPZ / LOOPE	İŞLENEN1	LOOPZ (**l**oop if **z**ero – sıfır ise atla) ve LOOPE (**l**oop if **e**qual – eşit ise atla) komutları CX yazmacının değeri 0'a eşit olmadığı ve ZF bayrak biti 1 olduğu sürece döngü devam eder. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
LOOPNZ / LOOPNE	İŞLENEN1	LOOPNZ (**loop** if **n**ot **z**ero – sıfır değilse atla) ve LOOPNE (**loop** if **n**ot **e**qual – eşit değilse atla) komutları CX yazmacının değeri 0'a eşit olmadığı ve ZF bayrak biti 0 olduğu sürece döngü devam eder. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
JCXZ	İŞLENEN1	JCXZ komutu CX yazmacı 0 ise atlar. İşlenen olarak bir etiket girilebileceği gibi bir adres de girilebilir.
INT	İŞLENEN1	INT (**int**errupt – kesme) komutu işlenen olarak bir bayt alır. Bu bayt INT komutunun tipidir. INT komutu işlendiği zaman bayrak yazmaçları, CS kesim yazmacı ve IP yazmacı yığına kaydedilir. IF bayrak yazmacı sıfırlanır. Kesme yordamına kontrol devredilir.
INTO	-	INTO komutu OF bayrak biti 1 ise INT 04H görevi görür.
IRET	-	IRET komutu kesme yordamından geri dönmeyi sağlar. IP,CS ve bayrak bitlerini yığından geri alır.
CLC	-	CLC (**cl**ear **C**F, CF bayrak bitini sıfırla) komutu CF bayrak bitini 0 yapar.
CMC	-	CMC (**co**m**p**lement **C**F- CF bayrak bitini ters çevir) komutu CF bayrak bitini ters çevirir.
STC	-	STC (**se**t **C**F – CF bayrak bitini 1 yap) komutu CF bayrak bitini 1 yapar.
CLD	-	CLD (**cl**ear **D**F – DF bayrak bitini 0 yap) komutu DF bayrak bitini 0 yapar.
STD	-	STD (**se**t **D**F – DF bayrak bitini 1 yap) komutu DF bayrak bitini 1 yapar.
CLI	-	CLI (**cl**ear **I**F – IF bayrak bitini 0 yap) komutu IF bayrak bitini 0 yapar. Donanım kesmeleri devre dışı bırakılmış olur.
STI	-	STI (**se**t **I**F – IF bayrak bitini 1 yap) komutu IF bayrak bitini 1 yapar. Donanım kesmeleri devrede olur.
HLT	-	HLT (**h**a**lt** – dur) komutu sistemin durdurulmasını sağlar.
WAIT	-	WAIT (**wait** – bekle) komutu yardımcı işlemci işlemini bitirene kadar işlemciyi beklemeye alır. Bu şekilde yardımcı işlemci tarafından erişilmekte olan belleğe işlemcinin erişmeye çalışması engellenmiş olur.
ESC	ANLIK VERI YAZMAC/BELLEK	ESC (**escape** – çık) komutu diğer işlemcilere veriyoluna erişme hakkı tanır. İşlemci tarafından bir NOP komutu gibi algılanır ama bellek

		işlenenini bellek yoluna verir.
LOCK	-	LOCK (**lock** – kitle) komutu sayesinde iki farklı işlemcinin aynı anda bir bellek bölgesini değiştirmesini engeller. XCHG,MOV,IN ve OUT komutları kullanılmadan önce eğer birden azla işlemci kullanılıyorsa kullanılması gerekir.
NOP	-	NOP (**n**o **op**eration – işlem yok) komutu XCHG AL,AL komutu ile aynı işlemi yapar. Yani herhangi bir şey yapmaz sadece 3 saat darbesi boyunca işlemciyi meşgul tutmuş olur.

10.2. 80186/80188 ile eklenen komutlar

Komut	İşlenenler	Tanım
BOUND	KAYNAK SINIR	KAYNAK yazmacında belirtilen dizilim dizini belirtilen alt ve üst sınırlar ile karşılaştırılır. SINIR adresinde karşılaşılan ilk kelime alt sınırı, bir sonraki kelime ise üst sınırı belirler. Sınırlardan taşılmışsa 5 numaralı kesmeye çağırılır.
ENTER	AYIR SEVİYE	Yığıtı yüksek seviyeli bir dilin yordamı için ayarlar. AYIR yazmacı yığıttan ayrılması gereken alanın boyutunu belirtir. SEVİYE ise yordamın dallanma seviyesini belirtir. Genellikle LEAVE komutu ile beraber kullanılır.
LEAVE	-	ENTER komutu ile ayrılan yığıt bölgesini genel kullanıma açar. SP ve BP yazmaçlarının değerlerini ENTER komutu öncesi değerlere döndürür.
INS	HEDEF KAPI	KAPI yazmacında belirtilen kapıdan veri alınıp ES:DI adresine yazılır. HEDEF işleneninin herhangi bir etkisi yoktur. DF bayrağı sıfırlanıp bu komut çağırıldığında DI değeri arttırılır, DF bayrağı birlendiğinde ise DI değeri azaltılır.
OUTS	KAPI KAYNAK	KAYNAK'tan bayt, kelime veya ikili kelimeyi alıp DX yazmacında belirtilen kapıya iletir. İşlenenler kullanılmadan çağırıldığında KAYNAK olarak DS:SI adresi kullanılır. DF bayrağı sıfırlanıp bu komut çağırıldığında SI değeri arttırılır, DF bayrağı birlendiğinde ise SI değeri azaltılır. Kapı numarası 0-255 arasında ise dolaysız veri olarak giriş yapılabilir. Aksi takdirde DX yazmacında belirtilmelidir. Genellikle sadece DX yazmacının 10 biti kullanıldığından sadece 1024 tane kapı belirtilebilir. Daha fazla kapı için özel donanımsal bileşenler kullanılmaktadır.
POPA	-	(E)DI,(E)SI,(E)BP,(E)SP,(E)DX,(E)CX,(E)BX,(E)AX yazmaçlarına yığıttaki veriler yukarıdaki sırayla POP'lanır. (E)SP değeri gerçekte yazmaca yazılmaz bu yüzden atılmış gibi düşünülmelidir.
PUSHA	-	(E)AX,(E)BX,(E)CX,(E)DX,(E)SP,(E)BP,(E)SI,(E)DI yazmaçları sırasıyla yığıta atılır.SP yazmacının değeri PUSH öncesi SP yazmacı değerine eşittir.

10.3. 80286 ile eklenen komutlar

Komut	İşlenenler	Tanım
ARPL	HEDEF KAYNAK	HEDEF'in RPL bitlerini KAYNAK'ın RPL bitleri ile karşılaştırır. HEDEF'in RPL bitlerinin değeri KAYNAK'ın RPL bitlerinin değerinden az ise HEDEF'in RPL bitleri KAYNAK'ın RPL bitlerine eşitlenir ve ZF bayrağı birlenir. Aksi takdirde ZF bayrağı sıfırlanır.
CLTS	-	MSR (**M**achine **S**tatus **R**egister – Makine Durum Yazmacı) deki TS bayrağını sıfırlar. Bu komut özel bir komut olmakla beraber genellikle sadece işletim sistemleri tarafından kullanılmaktadır.
LAR	HEDEF KAYNAK	LAR (**L**oad **A**ccess **R**ights – Erişim Yetkilerini Yükle) komutu ile erişim yetkileri yüklenir. HEDEF yazmacının üst baytına erişim yetkileri baytı yazılır ve alt HEDEF

		baytı KAYNAK seçimine göre sıfırlanır. Yükleme işlemi başarılı olursa Z bayrak biti birlenir.
STR	HEDEF	Kullanımda olan görev yazmacını HEDEF yazmacına yazar.
SLDT	HEDEF	LDT yazmacını HEDEF yazmacına kaydeder.
SGDT	HEDEF	GDT yazmacını HEDEF yazmacına kaydeder.
LTR	KAYNAK	Kullanımda olan görev yazmacına KAYNAK yazmacındaki veriyi yazar.
LSL	HEDEF SEÇİCİ	SEÇİCİ bölüt sınırını HEDEF yazmacına seçici geçerli ve yetki seviyesinde ise yazar. Yükleme işlemi başarı ile tamamlanırsa ZF bayrak biti birlenir, aksi takdirde sıfırlanır.
LMSW	KAYNAK	KAYNAK yazmacındaki veriyi MSW (**M**achine **S**tatus **W**ord – Makine Durum Kelimesi) ne atar.
SMSW	HEDEF	MSW'deki veriyi HEDEF yazmacına yazar.
SIDT	HEDEF	IDT (**I**nterrupt **D**escriptor **T**able – Kesme Tanım Tablosu) yazmacının değerini HEDEF yazmacına kaydeder.
VERR	SEÇİCİ	SEÇİCİ'nin geçerli ve yetki seviyesinde olup olmadığını kontrol eder. Bölüt okunabilir ise ZF bayrak biti birlenir. Aksi takdirde sıfırlanır.
LIDT	KAYNAK	KAYNAK yazmacındaki veriyi IDT yazmacına yazar.
LLDT	KAYNAK	LDT (**L**ocal **D**escriptor **T**able – Yerel Betimleyici Tablosu) yazmacına KAYNAK yazmacındaki veriyi yazar.
VERW	SEÇİCİ	SEÇİCİ'nin geçerli ve yazma yetki seviyesinde olup olmadığını kontrol eder. Bölüt yazılabilir ise ZF bayrak biti birlenir aksi takdir de sıfırlanır.
LGDT	KAYNAK	GDT (**G**lobal **D**escriptor **T**able – Genel Betimleyici Tablosu) yazmacına KAYNAK yazmacındaki veriyi yazar.

10.4. 80386 ile eklenen komutlar

Komut	İşlenenler	Tanım
BSF/BSR	HEDEF KAYNAK	KAYNAK'ın ilk 1 değerine sahip bitini arar. ZF bayrak biti eğer bit bulunur ise birlenir ve HEDEF'e dizin yüklenir. Bit bulunamazsa ZF bayrak biti sıfırlanır. BSF ileri yönde tarama yapmaktadır. BSR geri yönde tarama yapar.
BT	HEDEF KAYNAK	KAYNAK ile belirtilen dizindeki HEDEF biti CF bayrak bitine aktarılır.
BTC	HEDEF KAYNAK	KAYNAK ile belirtilen dizindeki HEDEF biti CF bayrak bitine terslenip aktarılır.
BTR	HEDEF KAYNAK	KAYNAK ile belirtilen dizindeki HEDEF biti CF bayrak bitine aktarılır. Ardından HEDEF'deki bu bit sıfırlanır.
BTS	HEDEF KAYNAK	KAYNAK ile belirtilen dizindeki HEDEF biti CF bayrak bitine aktarılır. Ardından HEDEF'deki bu bit birlenir.
CDQ	-	EAX yazmacındaki işaretli ikili kelimeyi dörtlü kelimeye çevirip EDX ve EAX yazmaçlarına kaydeder. EAX yazmacının üst bitleri EDX yazmacına kopyalanır. olarak çevirir.
CWDE	-	AX yazmacındaki işaretli bir kelimeyi EAX yazmacına işaretli bir ikili kelime olarak kopyalar. AX yazmacındaki işaret biti EAX yazmacına doğru genişletilir.
IRETD / IRETDF	-	Bu komut kesme yapılan yere geri dönülmesini sağlar. IP, CS yazmaçlarının değerleri ve bayrak bitlerinin değerleri yığıttan geri alınır. İşlemci hata kesmelerinin sonucunda hatanın oluştuğu komuta dönülür çünkü hata kesmesi sırasında CS:IP yığıta yazılmaktadır.
JECXZ	ETİKET	ECX yazmacı sıfır ise ETİKET'e atlanır. ECX'in sıfır olup olmadığı sınanırken işaretsiz karşılaştırma yapılır.
LFS	HEDEF KAYNAK	KAYNAK'tan 32 bitlik bellek işaretçisi okunur. HEDEF ve FS'ye yazılır. Bağıl konum HEDEF yazmacına yazılır, bölüt ise FS'ye yazılır.

LGS	HEDEF KAYNAK	KAYNAK'tan 32 bitlik bellek işaretçisi okunur. HEDEF ve GS'ye yazılır. Bağıl konum HEDEF yazmacına yazılır, bölüt ise GS'ye yazılır.
MOVSX	HEDEF KAYNAK	KAYNAK'daki veriyi HEDEF'e işaret genişlemeli olarak iletir.
MOVZX	HEDEF KAYNAK	KAYNAK'daki veriyi HEDEF'e sıfırları genişleterek iletir.
POPF/POPFD	-	Kelime (POPF) veya ikili kelimeleri (POPFD) yığıttan alıp bayrak yazmacına yazar ve SP yazmacının değerini arttırır. POPF komutu için SP 2, POPFD için SP 4 arttırılmaktadır.
PUSHF / PUSHFD	-	Bayrak yazmacını yığıta atar. PUSHF ile bir kelime yığıta atılırken PUSHFD ile bir ikili kelime atılır.
LSS	HEDEF KAYNAK	KAYNAK'tan 32 bitlik bellek işaretçisi okunur. HEDEF ve SS'e yazılır. Bağıl konum HEDEF yazmacına yazılır, bölüt ise SS'e yazılır.
LODSD	-	DS:SI adresindeki veriyi (yazı) akümülatör yazmacına aktarır. Yön bayrak biti birlenmiş ise SI 4 azaltılır, sıfır ise 4 arttırılır. Genellikle REP ön ekleriyle beraber kullanılmaktadır.
SETAE / SETNB	HEDEF	HEDEF baytını CF bayrak biti sıfır ise bire eşitler, aksi halde sıfıra eşitler.
SETB / SETNAE	HEDEF	HEDEF baytını CF bayrak biti bir ise bire eşitler, aksi halde sıfıra eşitler.
SETBE / SETNA	HEDEF	HEDEF baytını CF bayrak biti sıfır veya ZF bayrak biti bir ise bire eşitler, aksi takdirde sıfıra eşitler.
SETC	HEDEF	HEDEF baytını CF bayrak biti bir ise bire eşitler aksi halde sıfıra eşitler.
SETE / SETZ	HEDEF	HEDEF baytını ZF bayrak biti bir ise bire eşitler, aksi takdirde sıfıra eşitler.
SETG / SETNLE	HEDEF	HEDEF baytını ZF sıfır veya SF bayrak biti OF bayrak bitine eşitse birler, aksi halde sıfıra eşitler.
SETGE / SETNL	HEDEF	HEDEF baytını SF bayrak biti OF bayrak bitine eşitse birler, aksi halde sıfırlar.
SETLE / SETNG	HEDEF	HEDEF baytını ZF sıfır veya SF bayrak biti OF bayrak bitine eşit değilse birler, aksi halde sıfıra eşitler.
SETL / SETNGE	HEDEF	HEDEF baytını SF bayrak biti OF bayrak bitine eşit değilse birler, aksi halde sıfıra eşitler.
SETNE / SETNZ	HEDEF	HEDEF baytını ZF sıfır ise birler, aksi halde sıfırlar.
SETS	HEDEF	HEDEF baytını SF bayrak biti bir ise birler, aksi halde sıfırlar.
SETNS	HEDEF	HEDEF baytını SF bayrak biti sıfır ise birler, aksi halde sıfırlar.
SETNC	HEDEF	HEDEF baytını CF bayrak biti sıfır ise birler, aksi halde sıfırlar.
SETO	HEDEF	HEDEF baytını OF bayrak biti bir ise birler, aksi halde sıfırlar.
SETNO	HEDEF	HEDEF baytını OF bayrak biti sıfır ise birler, aksi halde sıfırlar.
SETP / SETPE	HEDEF	HEDEF baytını PF bayrak biti bir ise birler, aksi halde sıfırlar.
SETNP / SETPO	HEDEF	HEDEF baytını PF bayrak biti sıfır ise birler, aksi halde sıfırlar.
SHLD / SHRD	HEDEF KAYNAK SAYAÇ	SHLD komutu ile HEDEF SAYAÇtaki değer kadar sola kaydırılır ve HEDEF'in sağında açılan bir pozisyonları KAYNAK'ın en anlamlı bitlerinden başlayarak doldurulur. SHRD komutu ile HEDEF SAYAÇtaki değer kadar sola kaydırılır ve HEDEF'in sağında açılan bir pozisyonları KAYNAK'ın en anlamlı bitlerinden başlayarak doldurulur. Her iki komutta da SAYAÇ'ın sadece 5 biti kullanılır.
STOS	-	Akümülatördeki veriyi ES:(E)DI adresine yükler. DF bayrak bitinin durumuna bağlı olarak (E)DI yazmacının değeri arttırılır veya azaltılır. Genellikle REP ön ekiyle beraber kullanılır.

10.5. 80486 ile eklenen komutlar

Komut	İşlenen	Tanım
BSWAP	YAZMAÇ	32 bitlik YAZMAÇ'ı büyüğü başta kodlamasından küçüğü başta kodlamasına çevirir. YAZMAÇ 16 bitlik ise bu komut düzgün çalışmaz.
CMPXCHG	HEDEF KAYNAK	Akümülatördeki veriyi HEDEF deki veri ile karşılaştırır. Eşit iseler HEDEF'e KAYNAK'ın değeri atanır. Diğer durumlarda HEDEF'e akümülatörün değeri yüklenir.
INVD	-	İşlemcinin iç önbelleğini boşaltır. Dış önbelleklerin boşaltılması için emir verilir.
INVLPG	-	Translation Look-Aside tamponunun bir sayfasını geçersizleştirir.
WBINVD	-	İşlemcinin iç önbelleğini boşaltır. Ayrıca dış önbelleğe mevcut verinin geri yazılmasını emreder. Bu işlemden sonra ise dış önbellek boşaltılır.

10.6. Pentium ile eklenen komutlar

Komut	İşlenen	Tanım
CMPXCHG8B	HEDEF	EDX:EAX adresindeki 64 bitlik veriyi HEDEF adresindeki 64 bitlik veri ile karşılaştırır. Veriler eşit ise ECX:EBX adresindeki 64 bitlik veri HEDEF'e yazılır. Aksi takdirde HEDEF'deki 64 bitlik veri EDX:EAX adresine kaydedilir. HEDEF burada 64 bitlik (8 bayt) bir bellek yeridir. EDX:EAX yazmaç ikilisinde üst 32 bitlik veri EDX, alt 32 bitlik veri EAX yazmacında tutulur. ECX:EBX yazmaç ikilisinde de üst 32 bitlik veri ECX, alt 32 bitlik veri EBX yazmacında tutulur. Bu komut LOCK komutu ile beraber kullanılabilir.
RDMSR	-	ECX yazmacı tarafından belirtilen 64 bitlik bir MSR (**M**odel **S**pecific **R**egister – Özel Modellenmiş Yazmaç) yazmacının içeriğini EDX:EAX yazmaç ikilisine kaydeder. Bu komut çalıştırılırken 0. İzin seviyesinde veya gerçek adresleme kipinde çalışılması gerekmektedir. Aksi takdirde koruma hatası verilecektir. ECX yazmacında belirtilen MSR ihtiyatlı veya kullanım dışı ise gene bir koruma hatası verilecektir. Bu komut kullanılmadan önce CPUID komutu kullanılarak MSR yazmaçlarının desteklenip desteklenmediği denetlenmelidir.
RDPMC	-	Bu komut ile perfomans sayaçları okunabilmektedir. Pentium işlemcilerde 40 bitlik 2 adet performans sayacı bulunmaktadır ve MSR yazmaç bölgesinde bulunmaktadır. Bu sayaçlar çeşitli olayları izlerken kullanılabilmektedir. Bu komutun kullanılabilir olup olmadığı ne yazık ki CPUID komutu ile tespit edilememektedir.
RDTSC	-	EDX:EAX yazmaç ikilisine işlemcinin zaman damga sayacının değerini yazar. Zaman damga sayacı 64 bitlik bir MSR'de bulunmaktadır. İşlemci her saat darbesinde bu sayacın değerini arttırır. İşlemci ilklendirildiğinde ise sıfırlar. TSD (Time Stamp Disable – Zaman Damgasını Kapat) bayrağının kullanımı RDTSC komutunu kısıtlar. Sadece sıfırıncı izin seviyesinde kullanılabilir hale gelir.

		Zaman damga sayacının değeri RDMSR komutu ile sıfırıncı izin seviyesinde iken de okunabilir. RDTSC komutu seri bir komut değildir. Çalışmaya başlamak için diğer komutların bitmesini beklemesine gerek yoktur. Aynı şekilde RDTSC komutu çalışırken diğer komutlar da çalıştırılmaya başlanabilir.
WRMSR	-	ECX yazmacı tarafından belirtilen MSR'ye EDX:EAX yazmaç ikilisindeki veriyi yazar. Bu komut çalıştırılırken 0. İzin seviyesinde veya gerçek adresleme kipinde çalışılması gerekmektedir. Aksi takdirde koruma hatası verilecektir. ECX yazmacında belirtilen MSR ihtiyatlı veya kullanım dışı ise gene bir koruma hatası verilecektir. Bu komut kullanılmadan önce CPUID komutu kullanılarak MSR yazmaçlarının desteklenip desteklenmediği denetlenmelidir.
CPUID	GİRDİ	Bu komut ile işlemci tanımlaması yapılabilmektedir. Çıktı olarak EAX,EBX,ECX,EDX yazmaçları kullanılmaktadır. Bu komut sayesinde satıcı olarak Intel firması, işlemci ailesi, modeli ve benzer bilgiler öğrenilebilmektedir. GİRDİ olarak EAX yazmacı kullanılır. EAX yazmacına girilen veri sayesinde çıktı olarak ne çıkması gerektiği ayarlanabilmektedir.

10.7. Pentium Pro (P6) ile eklenen komutlar

Komut	İşlenen	Tanım
CMOVA	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyükse (CF=0 ZF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVAE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük veya eşit ise (CF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVB	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan küçükse (CF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF ile KAYNAK eşitse (ZF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVG	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyükse (ZF=0 SF=OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVGE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük veya eşit ise (SF=OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVL	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan küçükse (SF<>OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVLE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük veya eşit ise (ZF=1 SF<>OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNA	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük değilse (CF=1 ZF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNAE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük veya eşit değilse (CF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.

CMOVNB	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan küçük değilse (CF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNBE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan küçük veya eşit değilse (CF=0 ZF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNC	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Artan yoksa (CF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAK'a eşit değilse (ZF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNG	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük değilse (ZF=1 SF<>OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNGE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan büyük veya eşit değilse (SF<>OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNL	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan küçük değilse (SF=OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNLE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. HEDEF KAYNAKtan küçük veya eşit değilse (SF=OF) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNO	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Taşma yoksa (OF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNP	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Parite yoksa (PF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNS	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. İşaret yoksa (SF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVNZ	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Sıfır değilse (ZF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVO	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Taşma varsa (OF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVP	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Parite varsa (PF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVPE	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Parite çift ise (PF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVPO	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Parite tek ise (PF=0) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVS	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. İşaret varsa (SF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
CMOVZ	HEDEF KAYNAK	HEDEF ile KAYNAK karşılaştırılır. Sıfır varsa (ZF=1) HEDEF'e KAYNAK yazılır. Diğer durumlarda değerler korunur.
SYSENTER	-	Sistem çağrısı giriş noktasına hızlı geçiş yapmakta kullanılır.
SYSEXIT	-	Sistem çağrısı giriş noktasından hızlı çıkış yapmakta kullanılır.

10.8. Pentium III (SSE) ile eklenen komutlar

Komut	İşlenen	Tanım
MASKMOVQ	KAYNAK HEDEF	KAYNAK'daki veriyi HEDEF'deki maskeye bakarak (hangi bitlerin yazılacağını seçerek) 64 bitlik bir bellek bölgesine yazar. KAYNAK ve HEDEF işlenenleri MMX yazmaçlarıdır. Bellek bölgesi EDI ve DS yazmaçları tarafından belirlenir.
MOVNTPS	HEDEF KAYNAK	Kayan nokta veriyi XMM yazmacından m128'e aktarmak için kullanılır. Bu şekilde önbellek sıradüzeninde oluşan kirlilikler azaltılır. KAYNAK işlenenindeki ikili dötrlü kelimeyi HEDEF işlenenine

		önbellek sıradüzenininde en az kirlilik oluşturacak şekilde yazar. KAYNAK işleneni bir XMM yazmacıdır ve dört adet paketlenmiş kayan nokta verisi tutmaktadır. HEDEF işleneni ise 128 bitlik bir bellek bölgesidir.
MOVNTQ	HEDEF KAYNAK	Dörtlü ikili kelimeyi XMM yazmacından m64'e aktarmak için kullanılır. Bu şekilde önbellek sıradüzeninde oluşan kirlilikler azaltılır. KAYNAK işlenindeki dörtlü kelimeyi HEDEF işlenenine önbellek sıradüzenininde en az kirlilik oluşturacak şekilde yazar. KAYNAK işleneni bir MMX yazmacıdır ve içinde paketlenmiş bir tamsayı değeri bulunmaktadır. HEDEF işleneni ise 64 bitlik bir bellek bölgesidir.
PREFETCH0	KAYNAK	KAYNAK bellek bölgesindeki veriyi geçici olarak önbelleğe önceden getirir. Tüm önbellek seviyelerine önceden getirme yapılır.
PREFETCH1	KAYNAK	KAYNAK bellek bölgesindeki veriyi önbelleğe önceden getirir. L1 dışındaki tüm önbellek seviyelerine önceden getirme yapılır.
PREFETCH2	KAYNAK	KAYNAK bellek bölgesindeki veriyi önbelleğe önceden getirir. L2 dışındaki tüm önbellek seviyelerine önceden getirme yapılır.
PREFETCHNTA	KAYNAK	KAYNAK bellek bölgesindeki veriyi geçici olmayarak önbelleğe önceden getirir. Tüm geçici olmayan önbellek seviyelerine önceden getirme yapılır.
SFENCE	-	Bu komut çalıştırıldığında komut çalıştırılmadan önceki tüm bellek yazma işlemlerinin sonlanması beklenir. Bu şekilde SFENCE komutundan sonra herhangi bir bellek yazma işleminin yürürlülükte olmadığı garanti edilebilir.
ADDPS	/	Dört adet tek duyarlı kayan nokta değeri (32 bitlik), dört adet tek duyarlı kayan nokta değere ekler.
ADDSS	/	Dört adet tek duyarlı kayan nokta değerin ilkini başka bir tek duyarlı kayan nokta değere ekler. Diğer üç değer sabit kalır.
SUBPS	/	Dört adet tek duyarlı kayan nokta değeri (32 bitlik), dört adet tek duyarlı kayan nokta değerden çıkarır.
SUBSS	/	Dört adet tek duyarlı kayan nokta değerin ilkini başka bir tek duyarlı kayan nokta değerden çıkarır. Diğer üç değer sabit kalır.
MULPS	/	Dört adet tek duyarlı kayan nokta değeri (32 bitlik), dört adet tek duyarlı kayan nokta değer ile çarpar.
MULSS	/	Dört adet tek duyarlı kayan nokta değerin ilkini başka bir tek duyarlı kayan nokta değer ile çarpar. Diğer üç değer sabit kalır.
DIVPS	/	Dört adet tek duyarlı kayan nokta değeri (32 bitlik), dört adet tek duyarlı kayan nokta değer ile böler.
DIVSS	/	Dört adet tek duyarlı kayan nokta değerin ilkini başka bir tek duyarlı kayan nokta değer ile böler. Diğer üç değer sabit kalır.
RCPPS	/	Dört adet tek duyarlı kayan nokta değerin tersini alır. (1 / x)
RCPSS	/	Dört adet tek duyarlı kayan nokta değerin ilkinin tersini alır. Diğer üç değer sabit kalır.
SQRTPS	/	Dört adet tek duyarlı kayan nokta değerin kare kökünü alır.
SQRTSS	/	Dört adet tek duyarlı kayan nokta değerin ilkinin kare kökünü alır. Diğer üç değer sabit kalır.
RSQRTPS	/	Dört adet tek duyarlı kayan nokta değerinin kare kökünün tersini alır.
RSQRTSS	/	Dört adet tek duyarlı kayan nokta değerinin ilkinin kare kökünün tersini alır. Diğer üç değer sabit kalır.
MAXPS	/	Dört adet tek duyarlı kayan nokta değerin ikili olarak en büyüklerini bulup çıktı olarak verir.
MAXSS	/	Dört adet tek duyarlı kayan nokta değerin ikili olarak en büyüklerini bulup çıktı olarak verir. Sadece ilk ikili ele alınır. Diğer değerler sabit kalır.
MINPS	/	Dört adet tek duyarlı kayan nokta değerin ikili olarak en

		küçüklerini bulup çıktı olarak verir.
MINSS	/	Dört adet tek duyarlı kayan nokta değerin ikili olarak en küçüklerini bulup çıktı olarak verir. Sadece ilk ikili ele alınır. Diğer değerler sabit kalır.
PAVGB	/	Sekiz baytın içinden iki değerin ortalamasını çıktı olarak verir.
PAVGW	/	Dört kelimenin içinden iki değerin ortalamasını çıktı olarak verir.
PSADBW	/	Sekiz adet sekiz bitlik değerin mutlak değerinin toplamını çıktı olarak verir.
PEXTRW	/	Dört kelimenin birinden bir değer çıkarır.
PINSRW	/	Dört kelimenin birine bir değer ekler.
PMAXSW	/	Dört adet işaretli kelime içinden ikili olarak en büyük değerleri verir.
PMAXUB	/	Sekiz adet işaretsiz bayt içinden ikili olarak en büyük değeri çıktı olarak verir.
PMINSW	/	Dört adet işaretli kelime içinden ikili olarak en büyük değeri çıktı olarak verir.
PMINUB	/	Sekiz adet işaretsiz bayt içinden ikili olarak en küçük değeri çıktı olarak verir.
PMOVMSKB	/	Sekiz adet baytın en anlamlı bitlerini kullanarak bir adet maske baytı oluşturur.
PMULHUW	/	Dört adet işaretsiz kelimeti çarpar ve sonucun en anlamlı on altı bitini kaydeder.
PSHUFW	/	Dört adet kelimeyi karar.
ANDNPS	/	Dört adet tek duyarlı değere başka dört adet tek duyarlı değerin mantıksal tersine mantıksal AND operatörünü uygular.
ANDPS	/	Dört adet tek duyarlı değer ile başka dört adet tek duyarlı değere mantıksal AND operatörünü uygular.
ORPS	/	Dört adet tek duyarlı değer ile başka dört adet tek duyarlı değere mantıksal OR operatörünü uygular.
XORPS	/	Dört adet tek duyarlı değer ile başka dört adet tek duyarlı değere mantıksal XOR operatörünü uygular.
CMPXXPS	/	Dört adet tek duyarlı değeri karşılaştırır. Komut ifadesindeki XX yerine aşağıdaki harfler konarak neye göre karşılaştırma yapılması gerektiği seçilebilir. EQ: eşit LT: küçük LE: küçük eşit NE: eşit değil NLT: küçük değil NLE: küçük eşit değil ORD: sıralı UNORD: sırasız
CMPXXSS	/	İki adet tek duyarlı değeri karşılaştırır. Komut ifadesindeki XX yerine aşağıdaki harfler konarak neye göre karşılaştırma yapılması gerektiği seçilebilir. EQ: eşit LT: küçük LE: küçük eşit NE: eşit değil NLT: küçük değil NLE: küçük eşit değil ORD: sıralı UNORD: sırasız
COMISS	/	İki adet tek duyarlı değeri karşılaştırır ve sonuçları EFLAGS'de tutar.
UCOMISS	/	İki adet tek duyarlı deperi karşılaştırır ve sonuçları EFLAGS'de tutar. COMISS'den farkı QNaN hataları için ayrıklık verilmemesidir.
CVTPI2PS	/	İki adet 32 bitlik tamsayı değerini iki adet 32 bitlik kayan nokta

		değere çevirir.
CVTPS2PI	/	İki adet 32 bitlik kayan nokta değeri iki adet 32 bitlik tamsayı değere çevirir.
CVTSI2SS	/	Bir adet 32 bitlik tamsayı değeri 32 bitlik bir kayan nokta değere çevirir.
CVTSS2SI	/	Bir adet 32 bitlik kayan nokta değeri 32 bitlik bir tamsayı değere çevirir.
CVTTPS2PI	/	İki adet 32 bitlik tamsayı değerini iki adet 32 bitlik kayan nokta değere çevirir. Bu işlem yapılırken CVTPI2PS komutundan farklı olarak kırpmalar uygulanır.
CVTTSS2SI	/	Bir adet 32 bitlik kayan nokta değeri 32 bitlik tamsayı değere çevirir. Bu işlem yapılırken kırpmalar uygulanır.
FXRSTOR	/	FP ve SSE durumlarını geri yükler.
FXSAVE	/	FP ve SSE durumunu kaydeder.
LDMXCSR	/	MXCSR yazmacını yükler.
STMXCSR	/	MXCSR yazmacını kaydeder.
MOVAPS	/	128 bitlik bir değeri taşır.
MOVHLPS	/	Üst bitleri (yarım) alt bitlere taşır.
MOVLHPS	/	Alt bitleri (yarım) üst bitlere taşır.
MOVHPS	/	64 bitlik bir değeri bir XMM yazmacının üst yarısına yükler.
MOVLPS	/	64 bitlik bir değeri bir XMM yazmacının alt yarısına yükler.
MOVMSKPS	/	Tek duyarlı bir değerin üst bitlerini 32 bitlik bir yazmacın alt dört bitine taşır.
MOVSS	/	Dört tek duyarlı veriden ilkini taşır. Diğer üç veri sabit kalır.
MOVUPS	/	128 bitlik bir veriyi taşır. Adres hizalanmamış olabilir.
MASKMOVQ	/	64 bitlik bir veriyi verilen maskeye göre taşır.
MOVNTPS	/	128 bitlik bir veriyi belleğe doğrudan taşır. Önbellek kullanılmaz.
MOVNTQ	/	64 bitlik bir veriyi belleğe doğrudan taşır. Önbellek kullanılmaz.
SHUFPS	/	Dört adet tek duyarlı veriyi karar.
UNPCKHPS	/	Üst yarımdan tek duyarlı veri açar.
UNPCKLPS	/	Alt yarımdan tek duyarlı veri açar.

11. EK B

Burada FASM ve NASM derleyicilerinin, ilki Windows ikincisi Linux ortamında olmak üzere, bu kitaptaki örnekleri uygulamaya yetecek kadar temelleri gösterilecektir. Derleyicilerin bu temel özellikler dışında birçok özelliği bulunmaktadır. Yüksek seviyeli dillerde bulunan IF-ELSE dallanmalarına benzer dallanmaları ASM koduna uygulamak mümkün olmaktadır. Daha detaylı bilgi için lütfen derleyicilerin kullanma kılavuzlarını inceleyiniz.

11.1. 1. FASM Kullanımı

FASM (**F**lat **As**se**m**bler) bedava dağıtılan bir x86 çevirici-bağlayıcısıdır. Kullanımı çoğu çeviricilere göre kolaydır. Özellikle girilmesi gereken çeviriciye özel kodlar yoktur. Bağlayıcısını kendi içinde bulunur bu yüzden fazladan bir bağlayıcı kullanımını gerektirmez. Intel söz dizimi kullanılır.

Küçük bir kod yazalım ve bunu FASM yardımıyla çevirip bağlayalım. Kodumuz DOS ekranına "Merhaba Dünya!" yazısını yazdırsın.

```
1   ORG 0100H
2   JMP BASLANGIC
3   MESAJ DB 'MERHABA DUNYA!'
4   MESAJ_UZUNLUK DW $-MESAJ
5   BASLANGIC: MOV AH,03H       ;İMLEÇ KONUMUNU AL
6   INT 10H                 ;INT 10H ÇAĞIR
7   MOV AH,13H                  ;DIZGI YAZDIR
8   MOV AL,01H                  ;YAZMA KİPİ 01
9   MOV BH,00H                  ;SAYFA 0
10  MOV BL,0FH                  ;BEYAZ YAZI - SİYAH ARKAPLAN
11  MOV SI,MESAJ_UZUNLUK
12  MOV CX,[SI]                 ;MESAJ UZUNLUĞU
13  PUSH CS
14  POP ES
15  MOV BP,MESAJ                ;TAMPON ADRESI ES:BP
16  INT 10H                 ;INT 10H ÇAĞIR
17  SON: JMP SON
```

Bu kodu FASM programının editörüne yazalım ve yukarıdaki menüden "File->Save" seçerek ekb1.asm olarak kaydedelim. Ardından yine yukarıdaki menülerden "Run->Run" seçeneğini seçerek programımızı çevirelim ve çalıştıralım. Gördüğünüz gibi DOS ekranına:

```
Merhaba Dünya!
```

yazdırdık. Eğer ekb1.asm dosyasının bulunduğu klasöre göz gezdirirseniz ekb1.com adlı çalıştırılabilir bir dosya bulacaksınız.

11.2. 2. Linux'ta NASM Kullanımı

NASM (**N**etwide **As**se**m**bler) bedava dağıtılan bir x86 derleyicisidir ve Intel söz dizimini kullanır. Bu kitapta Intel söz dizimi kullanılmıştır ama AT&T söz dizimi kullanılmak istenirse Linux ile dağıtılan "as" çeviricisi kullanılabilir.

Linux için kod yazmaya geçmeden önce birkaç konudan bahsetmemiz gerekiyor. Öncelikle Linux korumalı kipte çalışan 32 bitlik bir işletim sistemidir. DOS ise gerçek kipte çalışan 16 bitlik bir işletim sistemidir.

"16 bit veya 32 bit olması yazılımcı açısından neyi değiştirir?" gibi bir soru öncelikle aklınıza gelecektir.

Öncelikle bu kitapta x86 serisinin ilk mikroişlemcisine (yani Intel 8086 mikroişlemcisine) uygun kod yazılmıştır. Ek A'da 8086'dan sonra gelen mikroişlemciler ve bu mikroişlemcilerin x86 komut dizisine yaptıkları eklemelere bakabilirsiniz. Ek A'daki listenin sonlarına gidilecek olursa bazı komutların sadece 32 bitlik mikroişlemcilerde ve hatta hatta 64 bitlik mikroişlemcilerde bulunduğunu göreceksiniz. Yani öncelikle 16 bit veya 32 bit olması kullanılabilecek komut sayısını değiştirmektedir. Ayrıca 16 bitlik 8086 mikroişlemcisinde yazmaçlar 16 bitliktir ama yeni Intel mikroişlemcileri ve benzerlerinde 32 bitliktir. AX, BX, CX ve DX yazmaçları 8086 için en geniş yazmaçlar iken 32 bitlik mikroişlemcilerde EAX, EBX, ECX ve EDX en geniş yazmaçlardır. Bu kitapta 8086'ya odaklanıldığından yeni mikroişlemcilerde bulunan yeni yazmaçlardan detaylı olarak bahsetmeyeceğiz ama burada Linux üzerinde çalışan bir örnek verebilmek için bilmemiz gereken EAX, AX yazmacının 32 bite genişletilmiş şeklidir. Yani EAX yazmacına bir veri yazıldığında üst 16 bite ve alt 16 bite yazılır. Alt 16 bit AX'dir. Üst 16 bite ayrıca erişebilmek mümkün değildir. 32 bitin tamamı okunmalı veya veri kaydırma komutları ile üst bitler alt bitlere kaydırılmalıdır. EBX, ECX ve EDX için de aynı durum geçerlidir. EBP ve ESP de görüldüğü gibi BP ve SP yazmaçlarının 32 bitlik şekilleridir.

"Peki gerçek ve korumalı kipler?" gibi bir soru da aklınıza gelecektir.

Gerçek kip 8086 işlemcisinin çalıştığı ve diğer yeni işlemciler açıldığında ilk olarak yüklenen kiptir. 8086 mikroişlemcisinde 20 bitlik adres iğneleri olduğundan 1MB lık bir bellek alanı adreslenebilmektedir ve gerçek kip bu yüzden sadece 1MB lık bir bellek alanını adresleyebilmektedir. Teknolojinin gelişmesiyle gereksinimler arttığından ve daha geniş bellekler üretildiğinden daha fazla adresleme yapılmasına ihtiyaç duyulmuştur. Bunu sağlayabilmek için ve geriye dönük uyumluluğu kaybetmemek için "korumalı kip" 80386 işlemcisine eklenmiştir ve bu sayede daha geniş bellekler adreslenebilmektedir. Geriye dönük uyumluluğun kaybedilmesinin istenmemesinin nedeni 8086 ve 8088 gibi mikroişlemcilerin birçok IBM bilgisayar ile piyasaya sürülmüş olması ve hali hazırda bulunan yazılımların bu mikroişlemcilerle uyumlu olmasıdır. Korumalı kipin getirdiği bazı kısıtlamalar vardır. Burada bizi ilgilendiren en büyük kısıtlaması BIOS kesmelerinin korumalı kipte direk olarak kullanılamamasıdır. Bu yüzden Linux'ta BIOS kesmelerine benzer "sistem çağrıları" bulunmaktadır. Linux sistem çağrılarının küçük bir kısmını Ek C'de bulabilirsiniz.

Tabii ki x86 komut dizisi geriye dönük uyumlu olduğundan 32 bitlik bir mikroişlemcide 8086 için yazılan kod sorunsuzca çalışabilmektedir. Yani burada 32 bitlik bir mikroişlemci için yazacağımız kod 8086 da çalışmayacak ama 8086 da çalışan bir kod bu sistemde çalışabilecektir (zaten eğer elinizde bir 8086 mikroişlemci deney seti veya benzeri yok ise şu ana kadar bunu yaptınız).

Öncelikle Linux sistem çağrılarını kullanan bir kod yazalım. Ardından da bu kodumuzu NASM ile çevirelim.

Kodumuz Linux terminal ekranına "Merhaba Dünya!" yazısını yazsın ve kapansın. Aşağıdaki kodu yazıp deneme1.asm olarak kaydediniz.

```
1   SECTION .DATA
2       mesaj:  DB 'Merhaba Dünya!',0AH
3       mesaj_uzunluk: EQU $-mesaj
4   SECTION .TEXT
5       GLOBAL _start
6
7   _start:
8       MOV EAX,04H          ;"sys_write"
9       MOV EBX,01H          ;dosya tanımlayıcısı, 01 = ekran
10      MOV ECX,mesaj        ;yazılacak mesajın adresi
11      MOV EDX,mesaj_uzunluk  ;yazılacak mesajın uzunluğu
12      INT 80H              ;sistem çağrısı
13
14      MOV EAX,01H          ;"sys_exit"
15      MOV EBX,00H          ;00 hata kodu
16      INT 80H              ;sistem çağrısı
```


Kodu NASM ile çevirmek için terminal ekranına

```
nasm ekb2.asm
```

yazınız. Kodunuz çevrilmiş olacaktır ve ekb2.o adlı bir dosya oluşturulacaktır. Tabi bu kodu çalıştırmadan önce bir de bağlayıcıdan geçirmeniz gerekmektedir. Bağlayıcı olarak "LD" bağlayıcısını kullanabilirsiniz. Kodumuzu bağlamak için terminal ekranına:

```
ld -s -o ekb2 ekb2.o
```

yazalım. Kodumuzu bağladığımız için artık çalıştırabiliriz. Terminal ekranına:

```
./ekb2
```

yazarak kodumuzu çalıştıralım. Görüldüğü gibi terminal ekranına:

```
Merhaba Dünya!
```

yazdırdık.

Kodu inceleyecek olursak *_start* etiketinden sonra şu ana kadar yazdığımız kodlardan çok da farklı olmayan bir kod yazdığımızı göreceğiz. *_start* etiketinden önce gelen kodlar ise çeviriciden çeviriciye değişebilen kodlardır. Mesela FASM derleyicisinde buna benzer bazı ön kodlar kullanmaya gerek yok iken NASM veya ASM86 da gerek vardır. Detaylı bilgi için lütfen çeviricilerin kullanım kılavuzlarına başvurunuz. Burada kodumuzun başlangıç noktasını _start etiketi ile açıkça belirtmemiz gerekmektedir, "LD" bağlayıcısının istediği bir parametredir.

12. EK C

Burada çok kullanılan DOS kesmeleri ve Linux kesmeleri hakkında bilgi vereceğiz.

12.1. DOS Kesmeleri

12.1.1. INT 21H – Girdi/Çıktı Kesmesi

12.1.1.1. AH = 01H

Standart giriş biriminden (genellikle klavye) bir karakter okur. Okunan karakterin ASCII kodu AL yazmacına kaydedilir. Karakter konsol ekranından görülebilir.

12.1.1.2. AH = 02H

Standart çıkış birimine (genellikle ekran) bir karakter yazar. DL yazmacına yazılacak olan karakter. Çıktı olarak AL yazmacı verilir. AL yazmacında son yazılan karakter bulunur. Genellikle AL yazmacının değeri DL yazmacına eşit olmaktadır. DL yazmacına 09H yazılırsa (yani tab karakteri) AL yazmacının değeri 20H olacaktır çünkü tab birkaç boşluğun art arda sıralanmasıyla oluşturulmaktadır.

12.1.1.3. AH = 05H

Yazıcıya karakter yazar. Yazıcıdan çıktı olarak alınmak istenen karakter DL yazmacına kaydedilmelidir.

12.1.1.4. AH = 06H

DL yazmacındaki karakteri konsola yazdırır. Çıktı olarak AL yazmacında son yazılan karakter bulunur. DL yazmacının değeri FFH a eşit olmamalıdır.

12.1.1.5. AH = 06H

DL yazmacına girdi olarak FFH değeri verilirse konsoldan girilen karakter okunabilir. ZF bayrak biti karakter yok ise bire eşitlenir ve AL yazmacına 00H değeri verilir. Aksi halde ZF bayrak biti sıfırlanır ve AL yazmacına karakterin ASCII değeri yazılır.

AL yazmacının değerinin 00H gelmesi özel kontrol tuşlarından birine basıldığı anlamına gelir. Bu kesme yeniden çağırılarak özel kontrol tuşu hakkındaki bilgi AL yazmacından alınabilir.

12.1.1.6. AH = 07H

Standart giriş biriminden (genellikle klavye) bir karakter okur. Okunan karakterin ASCII kodu AL yazmacına kaydedilir. Karakter konsol ekranından görünemez.

12.1.1.7. AH = 08H

07H ile aynı işleve sahiptir. Girilen karakter konsol ekranına basılmaz.

12.1.1.8. AH = 09H

Standart çıkış birimine dizgi yazdırır. Yazılacak olan dizgi DS:DX adresinde bulunmalı ve sonlandırıcı olarak '$' karakteri kullanılmalıdır. Çıktı olarak AL yazmacından 24H değeri alınır.

12.1.1.9. AH = 0AH

Klavyeden girilen karakterleri tampona kaydeder. Tampon DS:DX adresinde bulunur. Tamponun ilk baytında tamponun tutabileceği maksimum karakter sayısı bulunur. İkinci baytında son girdi yapıldığında girilen karakter sayısı veya okunmuş olan karakter sayısı tutulur (satırbaşı kodu hesaba katılmaz). Geri kalan baytlarında ise okunan karakterler son satırbaşı ASCII kodu ile beraber tutulur.

12.1.1.10. AH = 0BH

STDIN durumunu öğrenmek için kullanılır. Çıktı olarak AL yazmacı kullanılır. AL yazmacının değeri 00H ise her hangi bir karaktere basılmadığı, FFH ise bir karaktere basıldığı anlaşılır.

12.1.1.11. AH = 0CH

Tamponu temizler ve STDIN okuma fonksiyonunu çalıştırır. Girdi olarak AL yazmacını alır. AL yazmacında eğer 01H, 06H, 07H, 08H veya 0AH bulunuyorsa önce tampon temizlenir ardından yukarıda belirtilen bu kesmelerden biri kullanılır. AL yazmacında yukarıda belirtilen INT 21H alt kesmelerinden birinin kodu bulunmuyorsa sadece tampon temizlenir.

12.1.1.12. AH = 0DH

Diski yeniden başlatır. Değiştirilmiş olan disk tamponlarını diske yazar ama klasör bilgilerinde değişiklik yapmaz.

12.1.1.13. AH = 0EH

Varsayılan sürücüyü seçer. Girdi olarak DL yazmacında sürücü kodunu alır. A için 00H, B için 01H girilmelidir. Çıktı olarak AL yazmacına yüklü disk sürücülerinin sayısı yazılır.

12.1.1.14. AH = 19H

Varsayılan sürücüyü alır. Çıktı olarak AL yazmacında varsayılan sürücünün kodu verilir.

12.1.1.15. AH = 25H

Kesme vektörü ayarlanır. AL yazmacına kesme numarası girilir. DS:DX adresinde ise kesme kodu bulundurulur.

12.1.1.16. AH = 2AH

Sistem tarihi alınır. Çıktı olarak CX yazmacında sene, DH yazmacında ay, CL yazmacında gün ve AL yazmacında ise haftanın günü verilir.

12.1.1.17. AH = 2BH

Sistem tarihi ayarlanır. Girdi olarak CX ve DX yazmaçları kullanılır. CX yazmacına sene, DH yazmacına ay ve DL yazmacına gün yazılmalıdır. Çıktı olarak AH yazmacı kullanılır. Eğer AH yazmacının değeri 00H ise sistem tarihi başarıyla değiştirilmiştir. Aksi halde tarihin geçersiz olduğu anlamında FFH değeri verilir ve sistem tarihi değiştirilmez.

12.1.1.18. AH = 2CH

Sistem zamanı alınır. Çıktı olarak CH yazmacında saat, CL yazmacında dakika, DH yazmacında saniye ve DL yazmacında saniyenin yüzde birini verilir.

12.1.1.19. AH = 2DH

Sistem zamanı ayarlanır. Girdi olarak CX ve DX yazmaçları kullanılır. CH yazmacına saat, CL yazmacına dakika, DH yazmacına saniye ve DL yazmacına saniyenin yüzde biri girilmelidir. Eğer AH yazmacının değeri 00H ise sistem zamanı başarıyla değiştirilmiştir. Aksi halde tarihin geçersiz olduğu anlamında FFH değeri verilir ve sistem zamanı değiştirilmez.

12.1.1.20. AH = 2EH

Denetleme bayrağı ayarlanır. Girdi olarak AL yazmacı kullanılır. AL yazmacının değeri 00H ise denetleme bayrağı kapalı, 01H ise açıktır. Bayrağın açık olması tüm disklerin yazma işlevlerinin denetlendiği anlamına gelir.

12.1.1.21. AH = 30H

DOS sürümünü alır. Girdi olarak AL yazmacı kullanılır. AL yazmacının değeri 00H ise OEM numarası, 01H ise sürüm bayrağı verilir. Çıktı olarak AX,BX ve CX yazmaçları kullanılır. AL yazmacında ana sürüm numarası, AL yazmacında alt sürüm numarası, BL:CX adresinde 24 bitlik seri numarası, BH yazmacında OEM numarası veya sürüm bayrağı bulunur.

12.1.1.22. AH = 35H

Kesme vektörü alınır. Girdi olarak AL yazmacına kesme numarası girilmelidir. Çıktı olarak ES:BX adresine kesme kodu yazılır.

12.1.1.23. AH = 36H

Boş alan alınır. Girdi olarak DL yazmacına sürücü numarası girilmelidir. Varsayılan sürücü numarası 00H dır. Çıktı olarak AX,BX,CX ve DX yazmacı kullanılır. Sürücü geçersiz ise AX yazmacına FFFFH değeri yazılır. Geçerli ise AX yazmacına küme başına sektör sayısı, BX yazmacına boş küme sayısı, CX yazmacına sektör başına düşün bayt sayısı ve DX yazmacına toplam küme sayısı yazılır.

12.1.1.24. AH = 39H

"MKDIR" komutunun işlevini görür yani bir alt klasör yaratır. Girdi olarak DS:DX adresine klasörün ASCII karakterleriyle adresi girilmelidir. Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Eğer işlem başarılıysa CF bayrak biti ve AX yazmacı sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına hata kodu yazılır. Hata kodu 03H veya 05H olabilir.

12.1.1.25. AH = 3AH

"RMDIR" komutunun işlevini görür yani bir alt klasörü siler. Silinecek olan klasörün boş olması gerekmektedir. Girdi olarak DS:DX adresine klasörün ASCII karakterleriyle adresi girilmelidir. Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Eğer işlem başarılıysa CF bayrak biti ve AX yazmacı sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına hata kodu yazılır. Hata kodu 03H, 05H, 06H veya 10H olabilir.

12.1.1.26. AH = 3BH

"CHDIR" komutunun işlevini görür yani klasör değiştirir. Girdi olarak DS:DX adresine klasörün ASCII karakterleriyle adresi girilmelidir. Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Eğer işlem başarılıysa CF bayrak biti ve AX yazmacı sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına hata kodu yazılır. Hata kodu olarak 03H verilir.

12.1.1.27. AH = 3CH

Yeni bir dosya oluşturulur veya dosyanın boyutu sıfırlanır. Girdi olarak CX yazmacına dosya özellikleri ve DS:DX adresine dosyanın ASCII karakterleriyle adresi girilmelidir. İşlem başarılıysa CF bayrak biti sıfırlanır ve AX yazmacına dosyanın etiketi yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 03H, 04H veya 05H hata kodlarından biri yazılır.

12.1.1.28. AH = 3DH

Dosya açmak için kullanılır. Girdi olarak AL yazmacına erişim kuralları yazılmalı ve DS:DX adresine dosyanın ASCII karakterleriyle adresi girilmelidir. Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. İşlem başarılıysa CF bayrak biti sıfırlanır ve AX yazmacına dosyanın etiketi yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 01H, 02H, 03H, 04H, 05H, 0CH veya 56H hata kodlarından biri yazılır.

12.1.1.29. AH = 3EH

Açılmış dosyayı kapatmak için kullanılır. Girdi olarak BX yazmacında dosyanın etiketi girilmelidir. İşlem başarılıysa CF bayrak biti sıfırlanır ve AX yazmacına dosyanın etiketi yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 06H hata kodu yazılır.

12.1.1.30. AH = 3FH

Açılmış dosyayı okumak için kullanılır. Girdi olarak BX yazmacına dosyanın etiketi, CX yazmacına okunacak bayt sayısı yazılmalıdır. İşlem başarılı ise CF bayrak biti sıfırlanır, AX yazmacına okunan baytların sayısı yazılır ve okunan baytlar DS:DX adresine kaydedilir. Aksi halde CF bayrak biti birlenir ve AX yazmacına 05H veya 06H hata kodundan biri yazılır.

12.1.1.31. AH = 40H

Açılmış dosyaya veri yazmak için kullanılır. Girdi olarak BX yazmacına dosyanın etiketi, CX yazmacına yazılacak baytların sayısı yazılmalıdır. DS:DX adresine ise yazılacak baytlar konulmalıdır. İşlem başarılı ise CF bayrak biti sıfırlanır ve AX yazmacına yazılan bayt sayısı yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 05H veya 06H hata kodundan biri yazılır.

12.1.1.32. AH = 41H

Dosya silmeye yarar. Girdi olarak DS:DX adresine dosyanın adresi ASCII karakterleriyle yazılmalıdır ve CL bayrak bitine silme işleminin özellikleri yazılmalıdır. İşlem başarılı ise CF bayrak biti ve AX sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 02H, 03H veya 05H hata kodlarından biri yazılır.

Bu kesme aslında dosyayı silmemektedir. Sadece FAT tablosundan girişini silmektedir. Bazı silinmiş dosya kurtarma yazılımları bu özellik sayesinde silinen dosyaları kurtarabilmektedirler.

12.1.1.33. AH = 42H

Dosya konumunu ayarlar. Girdi olarak AL yazmacına başlangıç konumu girilmelidir. 00H bulunulan konum, 01H dosya başı ve 02H dosya sonunu gösterir. BX yazmacına dosyanın etiketi girilmelidir. CX:DX adresine ise başlangıç konumundan uzaklık girilmelidir. İşlem başarılı olursa CF bayrak biti sıfırlanır ve DX:AX adresine yeni dosya konumu yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 01H veya 06H hata kodlarından biri yazılır.

12.1.1.34. AH = 43H

Dosya özelliklerini okumak veya ayarlamak için kullanılır.

Dosya özelliklerini okumak için girdi olarak AL yazmacına 00H değeri ve DS:DX adresine okunacak dosyanın adresi ASCII karakterler ile yazılmalıdır. İşlem başarılı olursa CF bayrak biti sıfırlanır ve CX yazmacına dosya özellikleri yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 01H, 02H, 03H veya 05H hata kodlarından biri yazılır.

Dosya özelliklerini yazmak için AL yazmacına 01H değeri ve CX yazmacına yazılacak dosya özellikleri yazılmalıdır. DS:DX adresine ise okunacak dosyanın adresi ASCII karakterler ile yazılmalıdır. İşlem başarılı olursa CF bayrak biti ve AX yazmacı sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 01H, 02H, 03H veya 05H hata kodlarından biri yazılır.

Aşağıda dosya özellikleri baytının tablosu bulunmaktadır:

7	6	5	4	3	2	1	0
paylaşılabilirlik	kullanımda değil	arşiv	klasör	veri birimi etiketi	sistem	gizli	salt okunur

12.1.1.35. AH = 47H

Kullanımda olan klasörü öğrenmek için kullanılır. Girdi olarak DL yazmacına sürücü numarası yazılmalı ve DS:SI adresi 64 baytlık klasör adresini içerebilecek şekilde ayarlanmalıdır. İşlem başarılı olursa CF bayrak biti sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 0FH hata kodu yazılır.

12.1.1.36. AH = 4CH

Yazılımı geri dönüş koduyla beraber sonlandırır. Girdi olarak AL yazmacına geri dönüş kodu yazılmalıdır. Çıktı olarak herhangi bir yazmaç veya bayrak kullanılmaz.

Yazılımın kullandığı bütün dosyalar kapatılır ve bellekten silinir.

12.1.1.37. AH = 4DH

Geri dönüş kodunu almaya yarar. Çıktı olarak AH yazmacına sonlandırılma tipi yazılır. Sonlandırılma tipi 00H için normal sonlandırılma, 01H için Ctrl+C ile sonlandırma, 02H için kritik hata ile sonlandırma ve 03H için sonlandırma ve tutulma anlamlarına gelir. AL yazmacına ise geri dönüş kodu yazılır.

12.1.1.38. AH = 54H

Denetleme bayrağını alır. Çıktı olarak AL yazmacına denetleme bayrağı yazılır. AL yazmacının değeri 00H ise denetleme kapalı, 01H ise denetleme açıktır.

12.1.1.39. AH = 56H

Dosyayı yeniden adlandırmaya yarar. Girdi olarak DS:DX adresine dosyanın ASCII karakterler ile adresi girilmeli ve ES:DI adresine dosyanın yeni adı girilmelidir. CL yazmacına ise özellikler girilmelidir. İşlem başarılı ise CF bayrak biti sıfırlanır. Aksi takdirde CF bayrak biti birlenir ve AX yazmacına 02H, 03H, 05H veya 11H hata kodlarından biri yazılır.

12.1.1.40. AH = 57H

Dosyanın son yazım tarihini ve saatini öğrenmede veya değiştirmede kullanılır.

Son yazım tarihi ve saatini almak için girdi olarak AL yazmacına 00H değeri ve BX yazmacına dosyanın etiketi yazılmalıdır. İşlem başarılı ise CF bayrak biti sıfırlanır, CX yazmacına son değiştirilme zamanı yazılır ve DX yazmacına son değiştirilme tarihi yazılır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 01H veya 06H hata kodlarından biri girilir.

Son yazım tarihi ve saatini değiştirmek için ise girdi olarak BX yazmacına dosyanın etiketi, CX yazmacına yeni zaman ve DX yazmacına yeni tarih girilmelidir. İşlem başarılı ise CF bayrak biti sıfırlanır. Aksi halde CF bayrak biti birlenir ve AX yazmacına 01H veya 06H hata kodundan biri yazılır.

Aşağıda önce tarih ve ardından zaman kelimelerinin yapısı verilmiştir:

15 14 13 12 11	10 9 8 7 6 5	4 3 2 1 0
Saat	Dakika	Saniye

15 14 13 12 11 10 9	8 7 6 5	4 3 2 1 0
Yıl	Ay	Gün

12.1.2. INT 33H – Fare Kesmesi

12.1.2.1. AH = 00H

Farenin yüklü olup olmadığını anlamada kullanılır. Girdi olarak herhangi bir yazmaç almaz. Çıktı olarak AX ve BX yazmaçları kullanılır. AX yazmacının değeri 0000H ise fare sürücüsü yüklü değildir, FFFFH ise fare sürücüsü yüklüdür. BX yazmacına faredeki düğme sayısı yazılır.

12.1.2.2. AH = 01H

Fare imlecinin gösterilmesini sağlar. Girdi veya çıktı olarak herhangi bir yazmaç kullanılmaz.

12.1.2.3. AH = 02H

Fare imlecinin gizlenmesini sağlar. Girdi veya çıktı olarak herhangi bir yazmaç kullanılmaz.

12.1.2.4. AH = 03H

Fare konumunu ve düğme durumlarını öğrenmede kullanılır. Girdi olarak herhangi bir yazmaç almaz. Çıktı olarak BX, CX ve DX yazmaçları kullanılır. CX yazmacına yatay pozisyon (x) ve DX yazmacına dikey pozisyon (y) değeri yazılır. BX yazmacına ise düğme durumu yazılır. Düğme durum tablosu:

15 14 13 12 11 10 9 8 7 6 5 4 3 2	1	0
Kullanımda değil	Sağ Düğme	Sol Düğme

12.1.2.5. AH = 04H

Fare imleç konumunu ayarlamada kullanılır. Girdi olarak CX yazmacına yatay konum, DX yazmacına dikey konum girilmelidir. Çıktı olarak herhangi bir yazmaç kullanılmaz.

12.1.2.6. AH = 05H

Fare düğme basılma durumunun öğrenilmesinde kullanılır. Girdi olarak BX yazmacı kullanılır. BX yazmacına 0 değerinin verilmesi sol düğmenin bilgilerinin istendiğini, 1 değerinin verilmesi ise sağ düğmenin bilgilerinin istendiğini gösterir. Çıktı olarak AX, BX, CX ve DX yazmaçları kullanılır. AX yazmacına düğme durumu yazılır. BX yazmacına düğmeye basılma sayısı yazılır. Kesme çağırıldıktan sonra bu sayı sıfırlanır. CX yazmacına son düğme basılma anındaki yatay pozisyon ve DX yazmacına son düğme basılma anındaki dikey pozisyon yazılır. Düğme durum tablosu:

15 14 13 12 11 10 9 8 7 6 5 4 3 2	1	0
Kullanımda değil	Sağ Düğme	Sol Düğme

12.1.2.7. AH = 06H

Fare düğme bırakılma durumunun öğrenilmesinde kullanılır. Girdi olarak BX yazmacı kullanılır. BX yazmacına 0 değerinin verilmesi sol düğmenin bilgilerinin istendiğini, 1 değerinin verilmesi ise sağ düğmenin bilgilerinin istendiğini gösterir. Çıktı olarak AX, BX, CX ve DX yazmaçları kullanılır. AX yazmacına düğme durumu yazılır. BX yazmacına düğmenin bırakılma sayısı yazılır. Kesme çağırıldıktan sonra bu sayı sıfırlanır. CX yazmacına son düğme bırakılma anındaki yatay pozisyon ve DX yazmacına son düğme bırakılma anındaki dikey pozisyon yazılır. Düğme durum tablosu:

15 14 13 12 11 10 9 8 7 6 5 4 3 2	1	0
Kullanımda değil	Sağ Düğme	Sol Düğme

12.1.2.8. AH = 07H

Fare imlecinin yatay sınırlarının ayarlanmasında kullanılır. Girdi olarak CX yazmacına izin verilen en küçük yatay konum değeri yazılmalıdır. DX yazmacına ise izin verilen en büyük yatay konum değeri girilmelidir. Çıktı olarak herhangi bir yazmaç kullanılmaz.

12.1.2.9. AH = 08H

Fare imlecinin dikey sınırlarının ayarlanmasında kullanılır. Girdi olarak CX yazmacına izin verilen en küçük dikey konum değeri yazılmalıdır. DX yazmacına ise izin verilen en büyük dikey konum değeri girilmelidir. Çıktı olarak herhangi bir yazmaç kullanılmaz.

12.2. Linux Kesmeleri (Sistem Çağrıları)

Linux, DOS'un aksine, 32 bitlik korumalı kipte çalışan bir işletim sistemidir. Bu yüzden de BIOS kesmelerine direk olarak erişilememektedir. Bunun yerine Linux işletim sistemlerinde INT 80H kesmesine bazı parametreler verilir ve BIOS kesmelerinin işlevlerinin benzerleri yerine getirilir. Her INT 80H alt kesmesinin bir etiketi vardır, çevirici dili ile programlarımızı yazarken bu etiketlerin herhangi bir önemi yoktur. C ve benzeri diller ile programlar yazılırken bu etiketler kullanılır. Aşağıda bazı Linux sistem çağrılarını inceleyeceğiz. Aşağıda detayları verilen sistem çağrıları genellikle DOS ve BIOS da kesme olarak eşleniği olan sistem çağrılarıdır. Aslında Linux çağrıları BIOS ve/veya DOS kesmelerine göre daha iyi organize edilmiştir ve kullanımları daha açıktır.

Ek B'de korumalı kip ve gerçek kip arasındaki farkları belirtmiştik. Linux 32 bitlik korumalı kipte çalıştığından (yani 32 bitlik yazmaçları olaran x86 işlemcilerinde çalışabildiğinden) 32 bitlik yazmaçları vardır. Ek B'de bu yazmaçlara değinmiştik. Linux sistem çağrıları AX, BX, CX veya DX gibi 16 bitlik yazmaçlar yerine EAX, EBX, ECX ve EDX (ESI, EDI ve EBP de kullanılmaktadır) gibi 32 bitlik yazmaçları parametre olarak almaktadır.

12.2.1. INT 80H

12.2.1.1. EAX = 01H "sys_exit"

Girdiler	Çıktılar	İşlev
EBX = durum belirteci	-	Programı sonlandırır.

12.2.1.2. EAX = 03H "sys_read"

Girdiler	Çıktılar	İşlev
EBX = dosya tanımlayıcısı ECX = okuma tamponu işaretçisi EDX = okunacak bayt sayısı	Herhangi bir hata oluşmaz ise EAX yazmacından okunan bayt sayısı alınır. Aksi takdirde bir hata kodu alınır.	Bir dosya tanımlayıcısından veri okunur.

12.2.1.3. EAX = 04H "sys_write"

Girdiler	Çıktılar	İşlev
EBX = dosya tanımlayıcısı ECX = yazma tamponu işaretçisi EDX = yazılacak bayt sayısı	Herhangi bir hata oluşmaz ise EAX yazmacından yazılan bayt sayısı alınır. Aksi takdirde bir hata kodu alınır.	Bir dosya tanımlayıcısına veri yazılır.

12.2.1.4. EAX = 05H "sys_open"

Girdiler	Çıktılar	İşlev
EBX = ASCII dosya adresi işaretcisi ECX = dosya erişim bitleri EDX = dosya yetki bayrakları	Herhangi bir hata oluşmaz ise EAX yazmacından dosya tanımlayıcısı alınır. Aksi takdirde bir hata kodu alınır.	Dosya/sürücü açma ve/veya oluşturmada kullanılır.

Dosya erişim bitlerinden en çok kullanılanları:

Yazmaç Değeri	Anlam
00000000	Salt okunur. (O_RDONLY)
00000001	Salt yazılır. (O_WRONLY)
00000002	Yazılabilir ve okunabilir. (O_RDWR)
00000100	Oluştur. (O_CREAT)
00002000	Dosya sonuna ekle. (O_APPEND)
00100000	Büyük dosya olduğunu belirtir. (O_LARGEFILE)
00200000	Klasör olduğunu belirtir. (O_DIRECTORY)

Dosya yetki bayraklarından en çok kullanılanları:

Yazmaç Değeri	Anlam
00004000	Çalıştırıldığında kullanıcı kodunu ayarla. (S_ISUID)
00002000	Çalıştırıldığında gurup kodunu ayarla. (S_ISGID)
00001000	Dosyanın başkaları tarafından silinmesini, yeniden adlandırılmasını ve yerinin değiştirilmesini önlemede kullanılır. Sadece dosyanın bulunduğu klasörün sahibi ve ayrıcalıklı kullanıcılar yukarıdaki işlemleri yapabilir. (S_ISVTX)
00000400	Sahip tarafından okunabilir. (S_IRUSER)
00000200	Sahip tarafından yazılabilir. (S_IWUSER)
00000100	Sahip tarafından çalıştırılabilir ve aranabilir. (S_IXUSR)
00000040	Gurup tarafından okunabilir. (S_IRGRP)
00000020	Gurup tarafından yazılabilir. (S_IWGRP)
00000010	Gurup tarafından çalıştırılabilir ve aranabilir. (S_IXGRP)
00000004	Herkes tarafından okunabilir. (S_IROTH)
00000002	Herkes tarafından yazılabilir. (S_IWOTH)
00000001	Herkes tarafından çalıştırılabilir ve aranabilir. (S_IXOTH)

12.2.1.5. EAX = 06H "sys_close"

Girdiler	Çıktılar	İşlev
EBX = dosya tanımlayıcısı	Herhangi bir hata oluşmaz ise EAX yazmacı sıfırlanır. Aksi takdirde bir hata kodu alınır.	Dosya/sürücü kapatmak için kullanılır.

12.2.1.6. EAX = 08H "sys_creat"

Girdiler	Çıktılar	İşlev
EBX = ASCII dosya adresi işaretcisi ECX = dosya yetki bayrakları	Herhangi bir hata oluşmaz ise EAX yazmacına dosya tanımlamayıcısı yazılır. Aksi takdirde bir hata kodu alınır.	Dosya yaratmak için kullanılır.

12.2.1.7. EAX = 0CH "sys_chdir"

Girdiler	Çıktılar	İşlev
EBX = ASCII klasör adresi işaretcisi	Herhangi bir hata oluşmaz ise EAX yazmacı sıfırlanır. Aksi takdirde bir hata kodu alınır.	Kullanımda olan klasörü değiştirmek için kullanılır.

12.2.1.8. EAX = 0DH "sys_time"

Girdiler	Çıktılar	İşlev
EBX = tampon işaretçisi	Herhangi bir hata oluşmaz ise EAX yazmacına 1 Ocak 1970'den beri geçen saniye sayısı yazılır. Ayrıca EBX yazmacına bir tampon adresi girildiyse o adrese de aynı sonuç yazılır. Aksi takdirde bir hata kodu alınır.	1 Ocak 1970'den beri geçen saniye sayısını verir.

12.2.1.9. EAX = 13H "sys_lseek"

Girdiler	Çıktılar	İşlev
EBX = dosya tanımlayıcısı ECX = uzaklık bilgisi EDX = başlangıç değeri bayrağı	Herhangi bir hata oluşmaz ise EAX yazmacına işaretçinin konumu yazılır. Aksi takdirde bir hata kodu alınır.	Dosya işaretçisinin konumunu değiştirmede kullanılır.

Başlangıç değeri bayrakları:

Yazmaç Değeri	Anlam
00000000	Dosya başı. (SEEK_SET)
00000001	İşaretçinin bulunduğu konum. (SEEK_CUR)
00000002	Dosya sonu. (SEEK_END)

12.2.1.10. EAX = 19H "sys_stime"

Girdiler	Çıktılar	İşlev
EBX = saniye cinsinden süre	Herhangi bir hata oluşmaz ise EAX yazmacı sıfırlanır. Aksi takdirde bir hata kodu alınır.	1 Ocak 1970'den beri geçen saniye sayısını ayarlamada kullanılır.

12.2.1.11. EAX = 26H "sys_rename"

Girdiler	Çıktılar	İşlev
EBX = eski adres işaretçisi ECX = yeni adres işaretçisi	Herhangi bir hata oluşmaz ise EAX yazmacı sıfırlanır. Aksi takdirde bir hata kodu alınır.	Dosya adının değiştirilmesinde kullanılır.

12.2.1.12. EAX = 27H "sys_mkdir"

Girdiler	Çıktılar	İşlev
EBX = klasör adres işaretçisi ECX = dosya yetki bayrakları	Herhangi bir hata oluşmaz ise EAX yazmacı sıfırlanır. Aksi takdirde bir hata kodu alınır.	Klasör oluşturmak için kullanılır.

12.2.1.13. EAX = 28H "sys_rmdir"

Girdiler	Çıktılar	İşlev

EBX = klasör adres işaretçisi	Herhangi bir hata oluşmaz ise EAX yazmacı sıfırlanır. Aksi takdirde bir hata kodu alınır.	Boş bir klasörü silmede kullanılır.

13. EK D

13.1. BIOS Kesmeleri

Aşağıda çokça kullanılan BIOS kesmeleri hakkında bilgi bulabilirsiniz.

Kesme	Tanım
INT 00H	DIV ve IDIV komutları kullanıldığında sıfıra bölme veya bölme taşması oluştuğunda otomatik olarak işlemci tarafından çağrılır. Bazı sistemlerde ekranda bir hata mesajı gösterir. DOS ortamında bölme hatası olduğunu belirten bir mesaj ekrana yazdırılır.
INT 01H	Hata ayıklama için adım adım yürütme işlemini gerçekleştirmek için kullanılır. TF bayrak biti bire eşit ise her komuttan sonra çağrılır.
INT 02H	**NMI** (**N**on-**M**askable **I**nterrupt – Maskelenemeyen Kesme) oluştuğunda ortaya çıkar. Eşlik hatası, klavye kesmesi, yardımcı işlemci kesmesi, girdi/çıktı kanalı kontrolü, disket denetleyici açılışı, sistemi beklemeye alma, gerçek zamanlı sistem saati, sistem **WDT** (**W**atch**d**og **T**imer – Gözcü Saati**)** kesmesi ve **DMA** (**D**irect **M**emory **A**ccess – Doğrudan Belleğe Erişim) zamanlaması gibi alanlarda kullanılır.
INT 03H	Hata ayıklama için duraksama noktası belirtmede kullanılır. Makine dilinde karşılığı CCH'dır.
INT 04H	Taşma hatası oluştuğunda ortaya çıkar. OF bayrak biti birlenmiş ise INTO komutu tarafından çağırılır. Aksi halde herhangi bir işlem yapmaz, NOP komutuna eşdeğerdir.
INT 05H	Ekranı yazdırmada kullanılır. Klavyeden "Print Screen" tuşuna basıldığında otomatik olarak çağrılır.
INT 08H	IRQ0: Sistem zamanlayıcısı tarafından çağrılır. 18,2 saniyede bir sistem zamanlayıcısını günceller.
INT 09H	IRQ1: Klavye tarafından çağrılır. Klavyeden bir veri alındığında oluşur.
INT 0BH	IRQ3: COM2 (2. seri kapı) tarafından çağrılır.
INT 0CH	IRQ4: COM1 (1. seri kapı) tarafından çağrılır.
INT 0DH	IRQ5: Sabit disket denetleyicisi (PC/XT sistemlerde) veya 2. paralel kapı tarafından çağrılır (AT sistemlerinde).

<table>
<tr><td></td><td></td></tr>
<tr><td>INT 0EH</td><td>IRQ6: 1.44” disket sürücü denetleyicisi tarafından çağrılır.
Yapılması istenen işlem yapıldığında disket sürücü tarafından çağrılır.</td></tr>
<tr><td>INT 0FH</td><td>IRQ7: 1. paralel kapı (yazıcı) tarafından çağrılır.
Yazıcı hazır olduğunda LP1 tarafından çağrılır.</td></tr>
<tr><td>INT 10H</td><td>Video Hizmetleri
<table>
<tr><th>AH</th><th>İşlev</th></tr>
<tr><td>00H</td><td>Vidyo tipini ayarlar.

AL = Vidyo tipi
<table>
<tr><th>AL Değeri</th><th>Vidyo Kipi</th><th>Çözünürlük</th><th>Renk Sayısı</th></tr>
<tr><td>01H</td><td>Yazı</td><td>40 x 25</td><td>16</td></tr>
<tr><td>03H</td><td>Yazı</td><td>80 x 25</td><td>16</td></tr>
<tr><td>04H</td><td>CGA</td><td>320 x 200</td><td>4</td></tr>
<tr><td>06H</td><td>CGA</td><td>640 x 200</td><td>2</td></tr>
<tr><td>0DH</td><td>EGA</td><td>320 x 200</td><td>16</td></tr>
<tr><td>0EH</td><td>EGA</td><td>640 x 200</td><td>16</td></tr>
<tr><td>10H</td><td>EGA</td><td>640 x 350</td><td>16</td></tr>
<tr><td>11H</td><td>VGA</td><td>640 x 480</td><td>16</td></tr>
<tr><td>12H</td><td>VGA</td><td>640 x 480</td><td>16</td></tr>
<tr><td>13H</td><td>VGA</td><td>320 x 200</td><td>256</td></tr>
</table>
VGA = Video Graphics Array - Vidyo Grafik Dizisi
CGA = Color Graphics Adapter – Renkli Grafik Bağdaştırıcısı
EGA = Enhanced Graphics Adapter – Geliştirilmiş Grafik Bağdaştırıcısı</td></tr>
<tr><td>01H</td><td>İmleç şeklini ayarlar.

CH = başlama satırı
CL = bitiş satırı

Satır değerleri 0-15 arasında olduğundan imleci yok etmek için CH yazmacına 20H yüklenebilir.</td></tr>
<tr><td>02H</td><td>İmleç pozisyonunu ayarlar.

BH = sayfa numarası
DH = y koordinatı
DL = x koordinatı

Not: BIOS her sayfa için imleç pozisyonlarını ayrı ayrı tuttuğundan BH yazmacına 00H yazılması uygundur.</td></tr>
<tr><td>03H</td><td>İmleç pozisyonunu ve şeklini al

BH = imlecin bulunduğu sayfa numarası</td></tr>
</table></td></tr>
</table>

			Çıktı olarak: CH = başlama satırı CL = bitiş satırı DL = x koordinatı DH = y koordinatı
		05H	Görüntü sayfasını ayarlar. AL = görüntü sayfası
		06H	Temizle veya ekranı yukarı kaydırır. AL = ekranın kaç satır yukarı kaydırılacağı BH = temizlenen alan için ekran görüntü özelliği CL = temizlenecek alan için sol-üst köşe x koordinatı CH = temizlenecek alan için sol-üst köşe y koordinatı DL = temizlenecek alan için sağ-alt köşe x koordinatı DH = temizlenecek alan için sağ-alt köşe y koordinatı AL 00H ise temizleme görevi görür. Değilse ekran AL nin değeri kadar yukarı kaydırılır.
		07H	Temizle veya ekranı aşağı kaydırır. AL = ekranın kaç satır aşağı kaydırılacağı BH = temizlenen alan için ekran görüntü özelliği CL = temizlenecek alan için sol-üst köşe x koordinatı CH = temizlenecek alan için sol-üst köşe y koordinatı DL = temizlenecek alan için sağ-alt köşe x koordinatı DH = temizlenecek alan için sağ-alt köşe y koordinatı AL 00H ise temizleme görevi görür. Değilse ekran AL nin değeri kadar aşağı kaydırılır.
		08H	İmleç pozisyonundaki karakteri ve özelliklerini okur. BH = imlecin bulunduğu sayfa numarası Çıktı olarak: AL = okunan karakter AH = karakterin özellikleri
		09H	İmleç pozisyonuna karakter yazar. AL = yazılacak karakter BH = sayfa numarası BL = karakter özelliği CX = karakterin tekrarlanma sayısı
		0AH	İmleç pozisyonuna karakteri ve özellikleri yazar. AL = yazılacak karakter BH = sayfa numarası

0BH	Kenarlık rengini ayarlar. BH = 00H BL = renk
0EH	Özellikleri kullanarak ve imleci yazdıktan sonra yeniden konumlandırarak bir karakter yazar. AL = yazılacak karakter BH = sayfa numarası
0FH	Vidyo tipi ayarlarını alır. Çıktı olarak: AH = kolon sayısı AL = video tipi BH = sayfa numarası
10H	Vidyo palet yazmaçlarını yönetmede kullanılır.

AL	İşlev
00H	Palet yazmacını ayarlar. BL = ayarlanacak palet yazmacı BH = kaydedilecek renk değeri
01H	Kenarlık rengini ayarlar. BH = kaydedilecek renk değeri
02H	Kenarlık yazmacını ve tüm palet yazmaçlarının değerlerini yaz. ES:DX = 17 baytlık palet listesi **Not:** 17 baytlık palet listesinde 0 – 15 baytları palet yazmaçları değerlerine, 16. bayt ise kenarlık rengi için ayrılmıştır.
07H	Palet yazmacını okur. BL = palet yazmacı numarası Çıktı olarak BH yazmacında palet yazmaç renk değerini verir.
08H	Kenarlık rengi yazmacını okur. Çıktı olarak BH yazmacında kenarlık rengi değerini verir.
09H	Kenarlık yazmacını ve tüm palet yazmaçlarının değerlerini oku. Çıktı olarak ES:DX yazmacına 17 baytlık palet listesi yazılır.

	13H	Dizgi yazdır AL = yazma kipi BH = sayfa numarası BL = özellik CX = dizgi uzunluğu DH = başlama satırı DL = başlama sütunu ES:BP = yazılacak dizginin adresi

INT 11H

Çevre birimi tespiti yapılır.

Çıktı olarak AX yazmacı verilir.

AX Bitleri	**Anlam**
0	0 – Disket sürücü yüklü değil. 1 – Disket sürücü yüklü.
1	0 – 8087 yardımcı işlemci yüklü değil. 1 – 8087 yardımcı işlemci yüklü.
2 **3**	Her zaman 11 değerindedir.
4 **5**	Çalışmakta olan vidyo modu. 01 – 40 x 25 renkli 10 – 80 x 25 renkli 11 – 80 x 25 siyah/beyaz
6 **7**	Yüklü disket sürücü sayısı.
8	0 – DMA sistemi var. 1 – DMA sistemi yok.
9 **10** **11**	Yüklü RS232 kartı sayısı.
12	0 – Oyun kumandası girdi/çıktı kartı yüklü değil. 1 – Oyun kumandası girdi/çıktı kartı yüklü.
13	0 – Harici modem yok. 1 – Harici modem var.

	14 **15**	Yüklü yazıcı sayısı.

INT 12H

Bellek boyutunu al.

Çıktı olarak AX yazmacında her biri 1kB boyutunda bellek bölgesi sayısını verir.

INT 13H

Alt Seviye Disk Hizmetleri.

AH	**İşlev**
00H	Disk sistemini baştan başlat. DL = sürücü numarası **Not:** Sürücü numarasının yedinci biti bire eşit ise sabit disketler ve disket sürücü yeniden başlatılır.
01H	Disk sistemi durumunu oku. DL = sürücü numarası **Not:** Sürücü numarasının yedinci biti bire eşit ise sabit disket durumu okunur. Çıktı olarak AL yazmacı verilir.

AL	**Anlam**
00H	Hata yok.
01H	Hatalı komut.
02H	Adres belirteci bulunamadı.
03H	Disk yazmaya karşı korumalı.
04H	Sektör bulunamadı.
05H	Yeniden başlatılma işlemi başarısız. (sabit disket)
06H	Disket değiştirildi.
07H	Parametre tanımlamada hata oluştu. (sabit disket)
08H	DMA hatası. (disket)
09H	DMA 64kB sınırını geçti.
0AH	Hatalı sektör bulundu. (sabit disket)
0BH	Hatalı pist bulundu. (sabit disket)
0CH	Tanınmayan pist bulundu.
0DH	Hatalı sektör sayısı. (sabit disket)
0EH	Veri kontrol adres belirteci bulundu. (sabit disket)
0FH	DMA hatası. (sabit disket)
10H	CRC hatası.
11H	ECC düzeltildi. (sabit disket)
20H	Denetleyici hatası.
40H	Arama hatası.
80H	Zaman aşımı hatası.
AAH	Sabit disket hazır değil.
BBH	Tanımlanamayan sabit disket hatası.
CCH	Sabit disket yazma hatası.
E0H	Sabit disket durum yazmacı hatası.

		FFH Sabit disket tanıma işlemi hatası.
	02H	Sektör oku. AL = okunacak sektör sayısı CH = okunacak pist CL = ilk okunacak sektör (Üst 2 bitinde sadece sabit disketlere mahsus olmak üzere silindir numarasının üst 2 biti kaydedilmelidir.) DH = okuma başlığı (arka / ön) DL = sürücü numarası ES:BX = okunan verilerin yazılacağı tampon Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Hata var ise CF bayrak biti birlenir, aksi halde sıfırlanır. AH yazmacına durum bilgisi, AL yazmacına ise okunan sektör sayısı yazılır.
	03H	Sektör yaz. AL = yazılacak sektör sayısı CH = yazılacak pist CL = ilk yazılacak sektör (Üst 2 bitinde sadece sabit disketlere mahsus olmak üzere silindir numarasının üst 2 biti kaydedilmelidir.) DH = yazma başlığı (arka / ön) DL = sürücü numarası ES:BX = okunan verilerin yazılacağı tampon Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Hata var ise CF bayrak biti birlenir, aksi halde sıfırlanır. AH yazmacına durum bilgisi, AL yazmacına ise yazılan sektör sayısı yazılır.
	04H	Sektörleri kontrol et. AL = kontrol edilecek sektör sayısı CH = pist numarası CL = ilk kontrol edilecek sektör (Üst 2 bitinde sadece sabit disketlere mahsus olmak üzere silindir numarasının üst 2 biti kaydedilmelidir.) DH = başlık (arka / ön) DL = sürücü numarası Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Hata var ise CF bayrak biti birlenir, aksi halde sıfırlanır. AH yazmacına durum bilgisi, AL yazmacına ise kontrol edilen sektör sayısı yazılır.
	05H	Pisti biçimlendirir. Disket için: AL = pist üzerinde oluşturulmak istenen sektör sayısı CH = pist numarası CL = ilk sektör numarası DH = başlık (arka / ön) DL = sürücü numarası ES:BX = 4 baytlık adres verisi 1. bayt: pist 2. bayt: başlık

		3. bayt: sektör 4. bayt: sektör başuna düşen bayt sayısı (0 = 128 bayt, 1 = 256 bayt, 2 = 512 bayt, 3 = 1024 bayt) Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum kodu yazılır. Sabit disket için: AL = serpiştirme değeri CH = silindir numarası (8. ve 9. silindir numarası bitleri CL yazmacının üst iki bitidir) CL = sektör numarası DH = başlık (ön /arka) DL = sürücü numarası ES:BX = 512 baytlık biçim tamponu. Pist başına düşen her bayt için iki baytlık bir değer yazılır. İlk değer 00H olursa düzgün sektör, 80H olursa hatalı sektör anlamına gelir. İkinci değer ise sektör numarası olmalıdır. Çıktı olarak AH yazmacına durum değeri yazılır.
	08H	Sürücü parametlerini al. DL = sürücü numarası Çıktı olarak aşağıdaki yazmaçlar kullanılır. AH = durum değeri BL = sürücü tipi DL = ardışık sürücü sayısı DH = başlık sayısı CL = sektör sayısı CH = silindir sayısı ES:DI = sürücü parametre tablosu
	0AH	Sabit disketten oku. DL = sürücü numarası DH = başlık numarası CH = silindir numarası (silindir numarasının 8. ve 9. bitleri CL yazmacının üst iki bitidir) CL = sektör numarası ES:BX = okunan verilerin yazılacağı tamponun adresi Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri ve AL yazmacına okunan sektör sayısı yazılır.
	0BH	Sabit diskete yaz. DL = sürücü numarası DH = başlık numarası CH = silindir numarası (silindir numarasının 8. ve 9. bitleri CL yazmacının üst iki bitidir) CL = sektör numarası ES:BX = yazılacak verinin tutulduğu tampon

		Çıktı olarak CF bayrak biti ve AX yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri ve AL yazmacına yazılan sektör sayısı yazılır.	
	0CH	Sabit diskette silindire git. DL = sürücü numarası DH = başlık numarası CH = silindir numarası (silindir numarasının 8. ve 9. bitleri CL yazmacının üst iki bitidir) Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	0DH	Sabit disketi yeniden başlat. DL = sürücü numarası Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	0EH	Sabit disket denetleyicisinin tamponundan veri oku. ES:BX = okunan verilerin yazılacağı tampon Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	0FH	Sabit disket denetleyicisinin tamponuna veri yaz. ES:BX = denetleyici tamponuna yazılacak verinin tutulduğu tampon Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	10H	Sabit disketin hazır olup olmadığını sına. DL = sürücü numarası Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	11H		

		Sabit disketi kalibre et. DL = sürücü numarası Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	12H	Sabit disket denetleyici RAM sınaması. Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	13H	Sabit disket tanılaması. Çıktı olarak CF bayrak biti ve AH yazmacı kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına durum değeri yazılır.	
	15H	Sürücü tipini al. DL = sürücü numarası Çıktı olarak CF bayrak biti, AH yazmacı ve CX:DX adresindeki tampon kullanılır. Hata oluşmuş ise CF bayrak biti birlenir aksi takdirde sıfırlanır. AH yazmacına sürücü tipi yazılır. 00H = sürücü yok 01H = disket sürücü var ama değişiklik geçirmemiş 02H = disket sürücü var ve değişiklik geçirmiş 03H = sabit disket CX:DX = 512 baytlık sektör sayısı	
	16H	Disket sürücü değişiklik tespiti. DL = kontrol edilecek sürücü numarası Çıktı olarak AH yazmacı kullanılır. AH yazmacına disket değişiklik değeri yazılır. 00H = disket sürücüde değişiklik yok 06H = disket sürücüde değişiklik var	
	17H	Sürücü tipini ayarla. AL = sürücü tipi 00H = disk yok 01H = normal tipte disk, normal tipte sürücüde bulunuyor 02H = normal tipte disk, yüksek kapasiteli sürücüde bulunuyor 03H = yüksek kapasiteli disk, yüksek kapasiteli sürücüde bulunuyor	

		04H = 720kB boyutunda disket, 720kB lık sürücüde bulunuyor DL = sürücü numarası

INT 14H

Seri **I/O** (**I**nput/**O**utput – Girdi/Çıktı)

AH = 00H — İşlev

USART ilklendirmesi.

DX = kapı numarası
AL = ilklendirme parametreleri

AL Yazmacı Bit	İşlev
7 6 5	**Bitler (567) / Bilgi Akış Hızı (Baud)** 000 — 110 001 — 150 010 — 300 011 — 600 100 — 1200 101 — 2400 110 — 4800 111 — 9600
4 3	**Bitler (34) / Eşlik Çeşidi** 00 — Eşlik yok 01 — Tek eşlik 10 — Eşlik yok 11 — Çift eşlik
2	Sonlandırma biti sayısı. 0- Bir tane sonlandırma biti. 1- İki tane sonlandırma biti
1 0	**Bitler (01) / Karakter Boyutu** 10 — 7 bit 11 — 8 bit

Çıktı olarak AX yazmacı kullanılır.

AH yazmacına RS232 durum verisi yazılır.

AH	Anlam

		Yazmacı Bit / **Anlam** **7** Zaman aşımı. **6** Veri alış-veriş öteleme yazmacı boş. **5** Veri alış-veriş tampon yazmacı boş. **4** Durdurma verisi ile karşılaşıldı. **3** Kare hatası. **2** Eşlik hatası. **1** Aşma hatası. **0** Veri hazır. AL yazmacına modem durum verisi yazılır. **AL Yazmacı Bit** / **Anlam** **7** Alış hattı sinyali tespit edildi. **6** Zil komutu tespit edildi. **5** DSR **4** CTS **3** Alış hattı sinyalinde bir değişiklik tespit edildi. **2** Zil komutu azalan kenar tespit edildi. **1** Delta DSR **0** Delta CTS
	01H	Karakter gönder. AL = karakterin ASCII kodu DX = kapı numarası Çıktı olarak AH yazmacı kullanılır. AH yazmacına kapı durumu yazılır.
	02H	Karakter al. DX = kapı numarası Çıktı olarak AL ve AH yazmaçları kullanılır. AL yazmacına alınan karakterin ASCII kodu girilir. AH yazmacına RS232 durum verisi yazılır.
	03H	USART durumunu al. DX = kapı numarası Çıktı olarak AX yazmacı kullanılır. AX yazmacına kapı durumu yazılır.
INT 16H	Klavye	
INT 17H	Yazıcı Hizmetleri	

<table>
<tr><th>AH</th><th>İşlev</th></tr>
<tr><td>00H</td><td>
Yazıcıdan bir karakter çıktısı al.

Girdi olarak AL ve DX yazmaçları kullanılır.

AL = karakterin ASCII kodu.
DX = yazıcı kapısı (0 – 3)

Çıktı olarak AH yazmacı kullanılır.

<table>
<tr><th>AH Bitleri</th><th>Anlam</th></tr>
<tr><td>7</td><td>Meşgul değil.</td></tr>
<tr><td>6</td><td>Alındılandı.</td></tr>
<tr><td>5</td><td>Yazıcıda kağıt bitti.</td></tr>
<tr><td>4</td><td>Yazıcı seçildi.</td></tr>
<tr><td>3</td><td>Girdi/Çıktı hatası.</td></tr>
<tr><td>2</td><td>-</td></tr>
<tr><td>1</td><td>-</td></tr>
<tr><td>0</td><td>Zaman aşımı.</td></tr>
</table>
</td></tr>
<tr><td>01H</td><td>
Yazıcı ile bağlantı ilklendirme.

Girdi olarak DX yazmacı kullanılır.

DX = yazıcı kapısı (0 – 3)

Çıktı olarak AH yazmacı kullanılır.

<table>
<tr><th>AH Bitleri</th><th>Anlam</th></tr>
<tr><td>7</td><td>Meşgul değil.</td></tr>
<tr><td>6</td><td>Alındılandı.</td></tr>
<tr><td>5</td><td>Yazıcıda kağıt bitti.</td></tr>
<tr><td>4</td><td>Yazıcı seçildi.</td></tr>
<tr><td>3</td><td>Girdi/Çıktı hatası.</td></tr>
<tr><td>2</td><td>-</td></tr>
<tr><td>1</td><td>-</td></tr>
<tr><td>0</td><td>Zaman aşımı.</td></tr>
</table>
</td></tr>
<tr><td>02H</td><td>
Durum bilgisi al.

Girdi olarak DX yazmacı kullanılır.

DX = yazıcı kapısı (0 – 3)

Çıktı olarak AH yazmacı kullanılır.

<table>
<tr><th>AH Bitleri</th><th>Anlam</th></tr>
<tr><td>7</td><td>Meşgul değil.</td></tr>
<tr><td>6</td><td>Alındılandı.</td></tr>
</table>
</td></tr>
</table>

	<table><tr><td></td><td><table><tr><td>**5**</td><td>Yazıcıda kağıt bitti.</td></tr><tr><td>**4**</td><td>Yazıcı seçildi.</td></tr><tr><td>**3**</td><td>Girdi/Çıktı hatası.</td></tr><tr><td>**2**</td><td>-</td></tr><tr><td>**1**</td><td>-</td></tr><tr><td>**0**</td><td>Zaman aşımı.</td></tr></table></td></tr></table>
INT 18H	INT 19H başarısızlıkla sonuçlanırsa kullanılır.
INT 19H	**POST** (**P**ower-**O**n **S**elf **T**esting – Açılışta Otomatik Sınama) dan sonra işletim sistemini yüklemek için kullanılır.
INT 1AH	**RTC** (**R**eal **T**ime **C**lock – Gerçek Zamanlı Saat) Hizmetleri
INT 1BH	"Break" tuşuna basıldığında çağrılır.
INT 1CH	Yazılımsal zamanlayıcı kesmesi. INT 08H tarafından çağrılır.
INT 1DH	Adres imleci: **VPT** (**V**ideo **P**arameter **T**able – Vidyo Parametre Tablosu)
INT 1EH	Adres imleci: **DPT** (**D**iskette **P**arameter **T**able – Disket Parametre Tablosu)
INT 1FH	Adres imleci: **VGCT** (**V**ideo **G**raphics **C**haracter **T**able – Vidyo Grafikleri Karakter Tablosu)
INT 41H	Adres imleci: **FDPT 1** (**F**ixed **D**isk **P**arameter **T**able **1 –** 1. Sabit Disket Parametre Tablosu)
INT 46H	Adres imleci: **FDPT 2** (**F**ixed **D**isk **P**arameter **T**able **2 –** 2. Sabit Disket Parametre tablosu)
INT 4AH	RTC tarafından alarm için çağrılır.
INT 70H	IRQ8: RTC tarafından çağrılır.
INT 74H	IRQ12: Fare tarafından çağrılır.
INT 75H	IRQ13: Matematiksel ek işlemci tarafından çağrılır.
INT 76H	IRQ14: Birinci IDE denetleyicisi tarafından çağrılır.
INT 77H	IRQ15: İkinci IDE denetleyicisi tarafından çağrılır.

14. EK E

14.1. Klavye ASCII Kodları

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	O
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

14.2. Kontrol Karakterleri ve Anlamları

ASCII karakter tablosunda bulunan kontrol karakterleri eski terminal aletlerine komut olarak giriliyorlardı, günümüzde çoğu anlamlarını yitirmiştirler. Kitap boyunca 0AH ve 0DH değerlerine karşılık gelen LF ve CR komutlarını çokça kullandık. LF (line feed – satır besle) kontrol karakteri yeni bir satıra geçilmesini (yani bir bakıma ekranın veya yazıcının yeni bir satır ile beslenmesi) sağlarken CR (carriage return – satır başı) satırın başına gidilmesini sağlar. Diğer kontrol karakterlerini inceleyelim:

NUL : (**nul**l – sıfır)	Herhangi bir karaktere basılmadığında bu kontrol karakteri iletilir.
SOH : (**s**tart **o**f **h**eading – başlığın başı)	Veri iletiminde başlığın başını simgeler.
STX : (**s**tart of te**x**t – metin başı)	Veri iletiminde başlığın sonu ve metinin başını simgeler.
ETX : (**e**nd of **t**e**x**t – metin sonu)	Veri iletiminde metinin sonunu simgeler.
EOT : (**e**nd **o**f **t**ransmission – iletim sonu)	Veri iletiminde bir iletim bloğunun sonunu simgeler.
ENQ : (**enq**uiry – sorgu)	Veri iletiminde uç birimlerin durumlarını sorgulamada kullanılır.
ACK : (**ack**nowledge – alındılama)	Veri iletiminde bir verinin alındığını veya bir komutun kabul edildiğini belirtmek için kullanılır.
BEL : (**bel**l – zil)	Uç birimini zil ile uyarmada kullanılır.
BS : (**b**ack**s**pace – geri adım)	İmleci veya yazıcı ucunu bir sola kaydırmada kullanılır.
TAB : (horizontal **tab** – yatay sekme)	İmleci veya yazıcı ucunu yatay eksende önceden belirlenmiş bir konuma göndermede kullanılır.
LF : (**l**ine **f**eed – satır besle)	Ekranda veya sayfada yeni bir satıra geçilmesini sağlar.
VT : (**v**ertical **t**ab – dikey sekme)	İmleci veya yazıcı ucunu dikey eksende önceden belirlenmiş bir konuma göndermede kullanılır.
FF : (**f**orm **f**eed – sayfa besle)	Ekranın veya yazıcının yeni bir sayfa ile beslenmesini sağlar.
CR : (**c**arriage **r**eturn – satır başı)	İmlecin veya yazıcı ucunun satır başına gitmesini sağlar.
SO : (**s**hift **o**ut – özel koda geçiş)	ASCII karakter tablosundaki karakterler yerine başka özel karakterlerin

	konulmasını sağlar.
SI : (**s**hift **i**n – özel koddan çıkış)	Özel karakterlerin yerine ASCII karakter tablosundaki karakterlerin konmasını sağlar.
DLE : (**d**ata **l**ink **e**scape – veri bağı çıkış)	İki birim arasındaki veri iletişiminin sonlandırılmasının istendiğini belirtir.
DC1 : (**d**evice **c**ontrol **1** – birinci aygıt kontrol karakteri)	Bir sistemin uzaktan kontrolünü sağlar.
DC2 : (**d**evice **c**ontrol **2** – ikinci aygıt kontrol karakteri)	
DC3 : (**d**evice **c**ontrol **3** – üçüncü aygıt kontrol karakteri)	
DC4 : (**d**evice **c**ontrol **4** – dördüncü aygıt kontrol karakteri)	
NAK : (**n**egative **ac**k**n**owledge – negatif alındılama)	Veri iletiminde bir verinin alınamadığını veya bir komutun kabul edilmediğini belirtmek için kullanılır.
SYN : (**syn**chronous idle – eşzamanlama karakteri)	Sistemlerin eşzamanlanmasına yardımcı olan bir kontrol karakteri.
ETB : (**e**nd of **t**ransmission **b**lock – iletim bloğunun sonu)	Veri iletiminde bir iletim bloğunun sonuna ulaşıldığını belirtir.
CAN : (**can**cel – iptal)	İşlenmekte olan bir komutu sonlandırır.
EM : (**e**nd of **m**edium – ortam sonu)	Veri ortamının yada kullanılan kısmının sonuna gelindiğini belirtir.
SUB : (**sub**stitute – yedek)	Hatalı bir veri alındığında veya bulunduğunda, hatalı veri bu kontrol karakteri ile değiştirilir.
ESC : (**esc**ape – çıkış/değiştirme)	Bazı sistemlerde sistemden çıkış yapabilmede kullanılır. Bazı sistemlerde ise kendisinden sonra gelen karakterlerin anlamlarını değiştirmede kullanılır.
FS : (**f**ile **s**eparator – dosya ayırıcı)	Dosya olarak tanımlanan verilerin diğer verilerle karışmaması için kullanılan bir ayırıcı karakter.
GS : (**g**roup **s**eparator – grup ayırıcı)	Grup olarak tanımlanan verilerin diğer verilerle karışmaması için kullanılan bir ayırıcı karakter.
RS : (**r**ecord **s**eparator – tutanak ayırıcı)	Tutanak olarak tanımlanan verilerin diğer verilerle karışmaması için kullanılan bir ayırıcı karakter.
US : (**u**nit **s**eparator – birim ayırıcı)	Birim olarak tanımlanan verilerin diğer verilerle karışmaması için kullanılan bir ayırıcı karakter.
SP : (**sp**ace – boşluk)	Herhangi bir karakter yazılmadan imlecin veya yazıcı ucunun bir sağa kaydırılmasını sağlar.
DEL : (**del**ete – silmek)	Yazılmış bir karakterin silinmesinde kullanılır.

14.3. Klavye Tarama Kodları

Klavyeden tuşlanan karakterin veya tuşun kodunu alırken INT 16H kullanmak yerine 80 numaralı kapıyı direk olarak okursak klavyeden gönderilen tarama kodları ile karşılaşırız. Bu tarama kodları sistemden sisteme değişiklik gösterebilir, bu yüzden biz güncel bir IBM tipi klavyenin tarama, yani günümüzde kullanılan kişisel bilgisayarların çoğu için geçerli olan, kodlarını vereceğiz.

Tuş	Normal	Shift	Ctrl	Alt	Tuş	Normal	Shift	Ctrl	Alt
A	1E	1E	1E	1E	]	1B	1B	1B	--
B	30	30	30	30	;	27	27	-	--

C	2E	2E	2E	2E	'	28	28	-	-
D	20	20	20	20	`	29	29	-	-
E	12	12	12	12	\	2B	2B	2B	--
F	21	21	21	21	,	33	33	-	-
G	22	22	22	22	.	34	34	-	-
H	23	23	23	23	/	35	35	-	-
I	17	17	17	17	F1	3B	54	5E	68
J	24	24	24	24	F2	3C	55	5F	69
K	25	25	25	25	F3	3D	56	60	6A
L	26	26	26	26	F4	3E	57	61	6B
M	32	32	32	32	F5	3F	58	62	6C
N	31	31	31	31	F6	40	59	63	6D
O	18	18	18	18	F7	41	5A	64	6E
P	19	19	19	19	F8	42	5B	65	6F
Q	10	10	10	10	F9	43	5C	66	70
R	13	13	13	13	F10	44	5D	67	71
S	1F	1F	1F	1F	F11	85	87	89	8B
T	14	14	14	14	F12	86	88	8A	8C
U	16	16	16	16	SAO	4D	4D	74	9D
V	2F	2F	2F	2F	SOO	4B	4B	73	9B
W	11	11	11	11	YUO	48	48	8D	98
X	2D	2D	2D	2D	AŞO	50	50	91	A0
Y	15	15	15	15	BS	0E	0E	0E	0E
Z	2C	2C	2C	2C	DEL	53	53	93	A3
1	02	02	-	78	END	4F	4F	75	9F
2	03	03	03	79	ESC	01	01	01	01
3	04	04	-	7A	HOME	47	47	77	97
4	05	05	-	7B	INS	52	52	92	A2
5	06	06	-	7C	NK 5	-	4C	8F	-
6	07	07	07	7D	NK *	37	-	96	37
7	08	08	-	7E	NK -	4A	4A	8E	4A
8	09	09	-	7F	NK +	4E	4E	-	4E
9	0A	0A	-	80	NK /	35	35	95	A4
0	0B	0B	-	81	PgDn	51	51	76	A1
-	0C	0C	0C	82	PgUp	49	49	84	99
=	0D	0D	-	83	PrtSc	-	-	72	-
[	1A	1A	1A	--	SPACE	39	--	--	--
TAB	0F	0F	94	A5					

- : Değer döndürmez.

-- : INT 16H kesmesinin 02H alt kesmesi ile algılanabilir.

SAO : Sağ Ok Tuşu

SOO : Sol Ok Tuşu

YUO : Yukarı Ok Tuşu

AŞO : Aşağı Ok Tuşu

NK : Sayısal Klavye